Collaborative Context-based Reasoning

2007

Gilbert Barrett
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Engineering Commons

STARS Citation

https://stars.library.ucf.edu/etd/3073

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
COLLABORATIVE CONTEXT-BASED REASONING

by

GILBERT C. BARRETT
B.S. University of Central Florida, 2001
M.S. University of Central Florida, 2003

A dissertation submitted in partial fulfillment of the requirements for
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2007

Major Professor: Avelino Gonzalez
ABSTRACT

This dissertation explores modeling collaborative behavior, based on Joint Intentions Theory (JIT), in Context-Based Reasoning (CxBR). Context-Based Reasoning is one of several contextual reasoning paradigms. And, Joint Intentions Theory is the definitive semantic framework for collaborative behaviors. In order to formalize collaborative behaviors in CxBR based on JIT, CxBR is first described in terms of the more popular Belief, Desire, and Intention (BDI) model. Once this description is established JIT is used as a basis for the formalism for collaborative behavior in CxBR. The hypothesis of this dissertation is that this formalism allows for effective collaborative behaviors in CxBR. Additionally, it is also hypothesized that CxBR agents inferring intention from explicitly communicating Contexts allows for more efficient modeling of collaborative behaviors than inferring intention from situational awareness. Four prototypes are built and evaluated to test the hypothesis and the evaluations are favorable. Effective collaboration is demonstrated through cognitive task analysis and through metrics based on JIT definitions. Efficiency is shown through software metric evaluations for volume and complexity of code.
This dissertation is dedicated to my family, friends, and educators who have inspired and encouraged, me over the years. The list is long but each of you know who you are. Thanks. I couldn’t have done it without your support.
ACKNOWLEDGMENTS

Thank you to Greg Holifield, Brian Stensrud, and Kevin Creese for your many hours of worthwhile discussion related to Context-Based Reasoning, Artificial Intelligence, and Human Behavior Modeling. Thank you to my teaching assistant Brit Crawford and the students in my Software Design classes for your many hours of work on RoboCup. This work was funded in part by U.S. Army Simulation Training and Instrumentation Command (STRICOM), the U.S. Air Force Research Lab (AFRL) and The Link Foundation.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................................ x
LIST OF TABLES .......................................................................................................................... xiii

1 INTRODUCTION .................................................................................................................. 1
  1.1 Intelligent Agents .............................................................................................................. 2
  1.2 Collaborative Behaviors .................................................................................................. 3
    1.2.2 Joint Intention Theory ............................................................................................. 5
    1.2.3 SharedPlans ............................................................................................................. 7
  1.3 Context-Based Reasoning ............................................................................................... 9
    1.3.1 CxBR Agent’s Intrinsic Knowledge ..................................................................... 11
    1.3.2 CxBR Agent’s Extrinsic Knowledge .................................................................... 12
    1.3.3 Legacy CxBR Framework .................................................................................... 13
    1.3.4 CxBR and BDI ...................................................................................................... 14
  1.4 Summary ...................................................................................................................... 15

2 PRIOR RESEARCH ............................................................................................................. 17
  2.1 Collaborative Behavior Framework Implementations .................................................. 17
    2.1.1 STEAM ................................................................................................................. 17
    2.1.2 GRATE ................................................................................................................. 19
  2.2 Contextual Modeling and Reasoning Paradigms .......................................................... 22
    2.2.1 Context Mediated Behavior .................................................................................. 22
    2.2.2 Contextual Reasoning as Applied in SART ......................................................... 24
    2.2.3 Context-based Reasoning ..................................................................................... 28

3 PROBLEM DEFINITION .................................................................................................... 31
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2</td>
<td>Context Structure</td>
<td>62</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Context Topology</td>
<td>71</td>
</tr>
<tr>
<td>5.3</td>
<td>Prototypes</td>
<td>79</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Prototype I: CxBR Team</td>
<td>81</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Prototype II: CxBRwJIT Team Model</td>
<td>82</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Prototype III: CCxBR Team Model: Explicit Context Communication</td>
<td>82</td>
</tr>
<tr>
<td>6</td>
<td>EVALUATION OF PROTOTYPES</td>
<td>84</td>
</tr>
<tr>
<td>6.1</td>
<td>Test Method</td>
<td>84</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Cognitive Task Analysis</td>
<td>85</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Joint Intention Theory</td>
<td>86</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Volume and Complexity</td>
<td>87</td>
</tr>
<tr>
<td>6.2</td>
<td>The Test Plan</td>
<td>87</td>
</tr>
<tr>
<td>6.3</td>
<td>Evaluation Data</td>
<td>90</td>
</tr>
<tr>
<td>6.4</td>
<td>Results</td>
<td>90</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Prototype Evaluation: Average Score Comparisons</td>
<td>90</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Prototype Evaluations: Average Ball Distance Comparisons</td>
<td>95</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Prototype Evaluations: Average Ball Possession Time Comparisons</td>
<td>97</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Prototype Evaluations: Average Shared ComboContext Analysis</td>
<td>100</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation Discussion</td>
<td>105</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Unexpected discoveries</td>
<td>106</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Prototype Evaluations: A Qualitative Review</td>
<td>106</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Volume and Complexity Metric Results</td>
<td>109</td>
</tr>
<tr>
<td>7</td>
<td>SUMMARY/CONCLUSION &amp; FUTURE WORK</td>
<td>111</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1: Object Relationship for a Possible Implementation of an Agent (a.k.a. AIP) (Stensrud et al. 2004) Reprinted with permission .............................................................. 10

Figure 1.2: Contextual Knowledge Schema for CxBR Agent ............................................... 12

Figure 1.3: Class Structure of Legacy CxBR Framework illustrates inherited classes used for modeling i.e. agent[i], context[j], and mission[k] ........................................................................ 14

Figure 4.1 Simplified CxBR Model (UML class diagram - diamond denotes composition) .... 45

Figure 4.2 Class Diagram of Team Hierarchy Model (UML class diagram - triangle denotes inheritance, diamond denotes composition) ................................................................. 46

Figure 4.3 Sample Model of Tank Platoon (UML class diagram - triangle denotes inheritance, diamond denotes composition) .......................................................................................... 47

Figure 5.1 State Graph for Simple Soccer Team ................................................................. 58

Figure 5.2 Diagonal Combination Play .............................................................................. 60

Figure 5.3 Box Combination Play ..................................................................................... 61

Figure 5.4 Two Player Bounding Combination Play ....................................................... 62

Figure 5.5 Three Types of ComboContexts and Their Sub-Contexts ................................. 63

Figure 5.6 Regions Defined for Selecting the ComboContexts ......................................... 68

Figure 5.7 Region for Selecting ComboBoundingControl Context .................................... 69

Figure 5.8 Region for Selecting ComboBoxControl Context ............................................ 70

Figure 5.9 Region for Selecting ComboDiagonalControl Context ..................................... 71

Figure 5.10 State Graph for Simple Soccer Team (revisited) ............................................... 72
Figure 5.11 State Graph Including Combination Contexts with Inbound and Outbound Transitions Highlighted ................................................................................................................ 73

Figure 5.12 State Graph Including All Combination Contexts Inbound and Outbound Transitions Simplified for Clarity .................................................................................................................... 74

Figure 5.13 Simplified UML Class Representation .................................................................................................................. 81

Figure 6.1 Team Comparisons ............................................................................................................................................ 89

Figure 6.2 Average Score Against Base Team ...................................................................................................................... 91

Figure 6.3 Average Score Against CCXBR Team ..................................................................................................................... 92

Figure 6.4 Average Score Against CxBRwJIT Team .................................................................................................................. 93

Figure 6.5 Average Score Against CxBR Team .......................................................................................................................... 94

Figure 6.6 Average Ball Distance Against Base Team ........................................................................................................... 95

Figure 6.7 Average Ball Distance Against CCXBR Team ....................................................................................................... 96

Figure 6.8 Average Ball Distance Against CxBRwJIT Team .................................................................................................. 96

Figure 6.9 Average Ball Distance Against CxBR Team ........................................................................................................... 97

Figure 6.10 Average Ball Possession Time Against Base Team ............................................................................................. 98

Figure 6.11 Average Ball Possession Time Against CCXBR Team .......................................................................................... 99

Figure 6.12 Average Ball Possession Time Against CxBRwJIT Team .................................................................................. 99

Figure 6.13 Average Ball Possession Time Against CxBR Team ............................................................................................ 100

Figure 6.14 Average Shared Combo Context Time Against Base Team .................................................................................. 101

Figure 6.15 Average Shared Combo Context Time Against CCXBR Team .................................................................................. 102

Figure 6.16 Average Shared Combo Context Time Against CxBRwJIT Team ........................................................................ 103

Figure 6.17 Average Shared Combo Context Time Against CxBR Team .................................................................................. 105
Figure A.1 Class Structure of Legacy CxBR Framework illustrates inherited classes used for modeling i.e. agent[i], context[j], and mission[k] Reprinted with permission. (Barrett et. al. 2003) (repeat of Figure 1.3) .......................................................................................................................... 124

Figure A.2 Object Relationship for a Possible Implementation of an Agent Reprinted with permission. (Barrett et. al. 2003) .......................................................................................................................... 125

Figure A.3 Class Structure of Affect Enhanced CxBR Framework illustrates inherited classes used for modeling i.e. mood[i], emotion[j], agent[k], etc........................................................... 126

Figure A.4 Object Relationship for a Possible Implementation of an Affective Agent Reprinted with permission. (Barrett et. al. 2003) .......................................................................................................................... 127

Figure A.5 Object Model of Appliance Company .......................................................................................................................... 129

Figure A.6 Sequence diagram illustrating embedded nature of Context execution .......... 130

Figure A.7 UML Object Representation of a Teamwork Model................................................ 134

Figure A.8 Co-Context and Context Set Example Illustration ................................................... 136

Figure A.9 Combination Play 1: Dashed lines indicate player movement. Solid lines indicate ball movement .................................................................................................................................................. 139

Figure A.10 Combination Play 2: Dashed lines indicate player movement. Solid lines indicate ball movement .................................................................................................................................................. 139

Figure A.11 Context/Play topology ............................................................................................140
LIST OF TABLES

Table 5.1 Diagonal Combination Play Context ................................................................. 65
Table 5.2 Box Combination Play Context ...................................................................... 66
Table 5.3 Two Person Bounding Combination Play Context ........................................ 67
Table 5.4 Collaborative Transition Logic for All Teams .................................................... 80
Table 6.1 Cognitive Task Analysis Comparisons .............................................................. 107
Table 6.2 Shared Collaborative Context Comparisons ..................................................... 108
Table 6.3 Volume and Complexity Metrics ..................................................................... 109
Table B.1 Evaluations Against Prototype I: CxBR .......................................................... 155
Table B.2 Evaluations Against Prototype II: CxBRwJIT .................................................... 156
Table B.3 Evaluations Against Prototype III: CCxBR ...................................................... 158
Table B.4 Evaluations Against Base Team (Simple Soccer Team) ..................................... 159
Table B.5 Evaluations Against Prototype I: CxBR .......................................................... 160
Table B.6 Evaluations Against Prototype II: CxBRwJIT .................................................... 162
Table B.7 Evaluations Against Prototype III: CCxBR ...................................................... 163
Table B.8 Evaluations Against Base Team (Simple Soccer Team) ..................................... 164


1 INTRODUCTION

The research presented in this dissertation focuses on extending Context-based Reasoning to facilitate modeling collaborative behaviors. Collaborative behaviors are a fundamental necessity of teamwork. In order to model teamwork in a simulated environment, it is necessary to model the underlying collaborative behaviors of team-members. Context-based Reasoning is an Artificial Intelligence (AI) paradigm developed primarily for use in modeling human tactical behavior. The established theories on collaborative behaviors approach modeling from a Belief-Desire-Intention (BDI) perspective. There is a necessity to expand these formalizations beyond this limited modeling perspective. Moreover, there is a necessity to extend the definition of Context-based Reasoning (CxBR) to include a formalism for collaborative behaviors. Specifically, this research formalizes the connection between CxBR and BDI and uses this as a bridge to define collaborative behaviors in CxBR.

Context-based Reasoning is founded on the premise that humans make decisions based upon their current circumstances, or context. Human context is largely determined by the situational awareness of the decision maker. Traditionally, in CxBR an autonomous agent’s current context is determined by a set of sentinel rules. The agent’s current context determines the agent’s behavior through functions defined specifically for that context. The CxBR paradigm is described in detail at the end of this chapter.
Included in this chapter is a discussion of background concepts pertinent to the research described in the remainder of this dissertation. These background topics are: intelligent agents, collaborative behaviors, and Context-based Reasoning.

1.1 Intelligent Agents

Modeling teamwork entails modeling multiple agents to work collaboratively towards some mutual goal. As such, single autonomous agents will first be defined before discussing collaborative behaviors of multiple autonomous agents. For the purpose of this research, agents will primarily refer to software entities designed to simulate human behavior. In general, Luger (2002) states that the following four criteria must be met for intelligent software agents: situated, autonomous, flexible, and social.

To be situated means an agent must be able to interact with its environment. Thus, the agent receives input from the environment and is capable of affecting change in the environment. An autonomous agent is self-governing and able to interact with its environment without direct intervention by other agents. A flexible agent is responsive and proactive: it makes decisions in reaction to the environment, and it is opportunistic. Finally, a social agent is capable of interacting with other software or human agents. (Luger, 2002)

Perhaps the most common paradigm for modeling intelligent agents is the belief-desire-intention (BDI) model. (Georgeff, et al. 1999) Georgeff loosely defines BDI in AI terms as follows:

- Beliefs - represent knowledge the agent possesses of the world.
• Desires - correspond to goals of the agent.
• Intentions - are commitments toward a plan.

It is necessary to introduce the BDI model here because the more popular collaborative behavior theories are described in terms of BDI. These theories are introduced in the following section and elaborated upon in the next chapter.

1.2 Collaborative Behaviors

There are two dominant theories on modeling collaborative behaviors: Joint Intention Theory (Cohen and Levesque, 1991) and SharedPlans (Grosz and Kraus, 1999). There are many definitions given for collaborative behaviors. Hara et al. (2001) put the issue in perspective by stating that when considering these definitions of collaborative behaviors “two common elements emerge: working together for a common goal and sharing.” (Hara et. al., 2001)

1.2.1 Collaborative Behavior: Defined

Given here are some of the more prominent definitions related to collaboration, collaborative behaviors, teamwork and the modeling of such:

• “Collaborative behavior – coordinated activity in which the participants work jointly with each other to satisfy a shared goal – is more than the sum of individual acts (Searle, 1990; Grosz and Sidner, 1990) and may be distinguished from both interaction and simple coordination in terms of the commitments agents make to each other (Bratman, 1992; Grosz and Kraus, 1996; Grosz 1996).” (Grosz and Kraus, 1999)
• “It appears that a group acts more as a single agent with beliefs, goals, and intentions of its own, over and above the individual ones … joint activity is one that is performed by individuals sharing certain specific mental properties.” (Cohen and Levesque, 1991)

• “Teamwork … is more than a simple union of simultaneous coordinated activity … while teamwork does involve coordination, in addition, it at least involves a common team goal and cooperation among team members.” (Tambe, 1997)

• “A comprehensive model of collaboration should specify: (i) under what circumstances a social action should be initiated; (ii) what conditions need to be established before cooperation can proceed; (iii) how the individual agents should behave when carrying out their local activities related to the joint action; (iv) how (when) agents should interact with their fellow team members; and (v) when the joint action should terminate.” (Jennings, 1993)

It should be noted that collaboration is more than cooperative or coordinated efforts. The main difference is that collaboration is concerned with mutually shared goals. A common example used to illustrate the difference between coordination and collaboration is driving (in the presence of other drivers) versus driving in a convoy. Driving (assuming a road populated with numerous drivers) requires coordination between all drivers, in order to safely maneuver. However, each driver has its own goal (destination) and plan on how to carry out that goal. In contrast, the drivers in a convoy share a common goal and plan.
1.2.2 Joint Intention Theory

Joint Intention Theory (Cohen and Levesque, 1991) is widely accepted and considered (Jennings, 1995; Tambe, 1997) as a foundation of modeling collaborative interactions of intelligent agents. The theory is ideally suited for agents following a BDI model, as much of the theory is concerned with agents’ goals, intentions, and beliefs both individually and jointly. The following definitions and theorems from Cohen and Levesque (1991) capture the essence of Joint Intention Theory:

Definition 1
An agent has a **persistent goal** relative to q to achieve p iff:
1. The agent believes that p is currently false
2. The agent wants p to be true eventually
3. It is true (and the agent knows it) that (2) will continue to hold until the agent comes to believe either that p is true, or that it will never be true, or that q is false.

Definition 2
An **agent intends** relative to some condition to do an action just in case the agent has a persistent goal (relative to that condition) of having done the action and, moreover, having done it, believing throughout that the agent is doing it.

Definition 3
An agent has a **weak achievement goal** relative to q and with respect to a team to bring about p if either of these conditions holds:
1. The agent has a normal achievement goal to bring about p, that is, the agent does not yet believe that p is true and has p eventually being true as a goal.
2. The agent believes that p is true, will never be true, or is irrelevant (thus q is false), but has a goal that the status of p be mutually believed by all the team members.

Definition 4
A team of agents have a **joint persistent goal** relative to q to achieve p just in case
1. They mutually believe that p is currently false
2. They mutually know they all want p to eventually be true
3. It is true (and mutually known) that until they come to mutually believe either that p is true, that p will never be true, that q is false, they will continue to mutually believe that they each have p as a weak achievement goal relative to q and with respect to the team.
Theorem 1
If a team consists of a single member, then the team has a joint persistent goal iff that agent has an individual persistent goal.

Theorem 2
If a team has a joint persistent goal to achieve p, then each member has p as an individual persistent goal.

Theorem 3
If a team is jointly committed to some goal, then under certain conditions, until the team as a whole is finished, if one of the members comes to believe that the goal is finished but that this is not yet mutually known, she will be left with a persistent goal to make the status of the goal mutually known.

Definition 5
A team of agents jointly intends, relative to some escape condition, to do an action iff the members have a joint persistent goal relative to that condition of their having done the action and, moreover, having done it, mutually believing throughout that they were doing it.

Theorem 4
If a team jointly intends to do an action, and one member believes that she is the only agent of that action, then she privately intends to do the action.

Theorem 5
An individual who intends to perform actions a and b concurrently intends to perform a (respectively, b) relative to the broader intention.

Theorem 6
If a team jointly intends to do a complex action consisting of the team members concurrently doing individual actions, then the individuals will privately intend to do their share relative to the joint intention.

Theorem 7
An agent intends to do a sequential action in a stepwise fashion, the agent also intends to do each of the steps, relative to the larger intention.

Theorem 8
If a team jointly intends to do a sequential action, then the agent of any part will intend to do that part relative to the larger intention, provided that she will always know when the antecedent part is over, when she is doing her share, and when she has done it.

Theorem 9
If a team intends to do a sequence of actions in a joint stepwise fashion, the agents of any of the steps will jointly intend to do the step relative to the larger intention.

Examination of Joint Intention Theory’s definitions and theorems shows that the theory is primarily concerned with the intentions of agents to accomplish a mutually shared goal. Loosely speaking, if the agents believe they are working towards and committed to the same goal then they are in fact collaborating. JIT is widely accepted throughout the AI and Modeling and Simulation (M&S) community and forms the foundation upon which most frameworks and theories on teamwork are built. Likewise, JIT is the foundation upon which collaboration between CxBR agents is defined here.

The second major paradigm regarding collaborative behaviors of intelligent agents is SharedPlans (Grosz and Kraus, 1999), which is briefly introduced in the following section. SharedPlans lacks the simplicity of JIT. As such, it does not lend itself as well to the task of defining collaboration within CxBR models, yet it is introduced here for the sake of thoroughness.

1.2.3 SharedPlans

The premise upon which SharedPlans is built states “A theory of collaboration must treat not only the intentions, abilities, and knowledge about action of individual agents, but also their coordination in group planning and acting. It also must account for the ways in which plans are incrementally formed and executed by the participants.” (Grosz and Kraus, 1999) To this end, SharedPlans is concerned with many meta-predicates. Five or the major meta-predicates, which are used to distinguish different types of plans are: full individual plans (FIP), partial individual
plans (PIP), full shared plans (FSP), partial shared plans (PSP), and shared plans of indefinite completeness (SP). Two meta-predicates used to represent beliefs regarding an agent’s abilities are 1) can bring about (CBA) and 2) can bring about group (CBAG). The meta-predicates are used in a seemingly complex algorithm to decide if a shared plan exists. Agents are required to build a complicated hierarchy of knowledge about the formation of individuals’ plans and then attempt to reason towards a collaborative plan. Individual and group means-ends reasoning are performed by Elaborate _Individual and Elaborate _Group respectively.

Intention is dissected within the SharedPlans formalism and the concepts of intending to do an action and intending that a proposition hold are introduced.

The intentional attitude, intending-that, was introduced into the SharedPlan formalization to account for the commitment participants in a group activity make to one another’s actions and to their joint activity. Intentions-that, like intentions-to, serve both to constrain the intentions an agent adopts and to affect its plan-based reasoning. (Grosz and Kraus, 1999)

The high complexity of SharedPlans is contrary to the simplistic approach of CxBR towards modeling knowledge. For this reason, SharedPlans is not considered here to be the best initial approach for formalizing collaborative behaviors within CxBR. However, the basic premise of SharedPlans regarding the importance of planning to collaborative behaviors is accepted. The research presented here, which is justified by JIT, does account for the formation of individual and group plans.
Most pertinent to the research discussed throughout this dissertation is, of course, the CxBR paradigm itself. For those not familiar with CxBR, a brier introduction to the topic follows.

1.3 Context-Based Reasoning

Context-based Reasoning is founded on the premise that humans make decisions based on a subset of their knowledge, which is referenced according to one’s current situation, environment, or context. In other words, for any given situation, there are but a limited number of expected occurrences; therefore, one’s actions and reactions are chosen from past and expected experiences. This prevents one from having to search through the complete store of knowledge in order to make decisions. Instead, knowledge is referenced and decisions are made based on situational contexts.

Rules and functions are categorized and organized hierarchically within Missions, Major Contexts and Sub Contexts. The highest level or categorization is the Mission Context. The next lower levels of knowledge categorization are contained in Major Contexts followed by Sub-Contexts. Missions contain any universal rules, those rules that are checked regardless of the current Major Context, and list all possible Major Contexts that could conceivably be involved in the mission. A Major Context contains the control elements specific to that given situation, a list of all possible Sub Contexts, and a list of all Major Contexts that can be transitioned to, from the current Major Context. Sub Contexts are analogous to Major Contexts but represent a lower level of abstraction. Missions, Major Contexts, and Sub Contexts are object-oriented classes and
as such can also contain additional methods and data structures relevant to their purpose (Barrett et al., 2002).

Agents (sometimes, traditionally referred to as Autonomous Intelligent Platforms (AIP) by CxBR modelers) are assigned a mission, which defines the high level goal the agent is expected to pursue for a given scenario. The Mission also specifies the Contexts that will be applicable to the agent in that Mission. Contexts provide the agent’s planning and reaction abilities and specify any other possible Contexts that can be reached from the currently active context. The relationship between Contexts, Missions, and agents is illustrated in Figure 1.1.

![Figure 1.1: Object Relationship for a Possible Implementation of an Agent (a.k.a. AIP) (Stensrud et. al. 2004) Reprinted with permission](image)

Figure 1.1: Object Relationship for a Possible Implementation of an Agent (a.k.a. AIP) (Stensrud et. al. 2004) Reprinted with permission
1.3.1 CxBR Agent’s Intrinsic Knowledge

Putting aside for the moment an agent’s missions, goals, personality, affect, and other such high-level concerns, consider the fact that each agent must have a more fundamental set of behavioral characteristics. These include low-level behavior, such as motor skills; sensory data, what the agent perceives about its world; and environmental knowledge, what the agent remembers with regard to its historical perception of the world. Much of this is concerned with the physical representation of the agent.

Low-level behaviors are closely related to dynamic physical characteristic of the agent. These low-level behaviors are fundamental in defining the agent. This is true in the sense that the agent is defined by the low-level behaviors of which it is capable. It is also true in the sense that the constraints of the behaviors themselves further define the agent. Consider a behavior such as movement and a corresponding function called move() to represent this behavior. Different agent types should be characterized in distinctly different ways by how move() defines them. For example, move() to a helicopter allows for three dimensional movement through space but there are certain constraints that must be adhered to regarding maximum velocity, maximum altitude, attitude of the aircraft, etc. A fish would also have a low-level behavior defined by move(). However, the maximum velocity or maximum altitude of a fish will obviously differ from that of a helicopter.

In addition to low-level behaviors, in CxBR each agent has some perception of and knowledge about its surrounding world. What is of particular importance here, as in the other areas of knowledge representation employed by CxBR agents, is the flexibility the modeler is permitted
in choosing knowledge representation paradigms. The method through which memory is implemented for a model is not constrained by the CxBR paradigm. A set of data structures stored in memory could be used to allow fast retrieval of information. Alternatively, a database could be interfaced with the model to allow storage and retrieval of large quantities of data.

Figure 1.2: Contextual Knowledge Schema for CxBR Agent

1.3.2 CxBR Agent’s Extrinsic Knowledge

Each agent is aware of its current Mission at any given time. Missions, Contexts, and Moderators are objects in CxBR that support the autonomous behavior of an agent. Their
interrelated nature and interdependencies are depicted in Figure 1.1. As a brief oversimplification, a Mission is composed of a set of Contexts, which themselves can be modified by one or more sets of Context Moderators.

As shown in Figure 1.2, a Mission may contain the following knowledge: the agent’s high-level goal, mission constraints, and the Context topology, among other things. Contexts contain high-level behavior representation, sub-goals, Context transition topology, Context transition criteria, and Context Moderator affects. Not represented by the figure, Major contexts can include a set of Sub Contexts and low-level behaviors. Context Moderators are relatively new to CxBR and provide an optional way of expanding the richness of agent behavior through influencing the Context behavior. Moderators must contain Moderator transition criteria and Moderator transition topology. Moderators have been useful in representing mood and emotion (Barrett, *et al.* 2003) and are given further attention in Chapter 7. This section is meant only an introduction to CxBR knowledge representation. Many of the constructs mentioned here will be discussed in greater detail in Chapter 2 and Chapter 7.

1.3.3 Legacy CxBR Framework

The traditional CxBR Framework, created by Lars Norlander (1999) and illustrated in Figure 1.3, includes four abstract classes, which create an application program interface (API) to facilitate modeling human behaviors. The AIP (Autonomous Intelligent Platform) class provides an interface for modeling agents. The Mission and Context classes provide the means for an agent to react and plan within a given scenario. The Inference Engine is used for pattern matching rules with facts, from both a local and a global fact base, and executing these rules.
1.3.4 CxBR and BDI

The relationship between Context-based Reasoning and BDI models may not be readily apparent, at first. However, as outlined in this section, all aspects of BDI models are captured in CxBR models. The most direct relationship is between Missions and Desires. A Mission in CxBR is the highest level goal the agent is attempting to accomplish and, as such, represents the agent’s long time desire.

CxBR agents’ planning is a result of the Contexts encountered while attempting to carry out a Mission. Contexts are commonly activated in reaction to the environment. Activation of any particular Context is also determined by the agent’s beliefs, which are in a sense the agent’s perception and is captured in the CxBR transition rules. A CxBR Context defines the possible
actions an agent may take when facing a particular context or situation. Thus, the agent’s current intentions are determined by the agent’s current Context. As a result, the actual plan that an agent follows can be explained in terms of the sequence of Contexts activated during an attempted Mission. Beliefs and intentions are closely related to the active Context of an agent.

To provide a foundation for developing a formalism for modeling collaborative behaviors in CxBR based upon the tenets of JIT, first, the relationship between CxBR and BDI will be formalized in Chapter 4: Conceptual Approach.

1.4 Summary

The major collaborative behavior paradigms do not specifically address particular reasoning paradigms but do rely heavily on the BDI modeling paradigm in their formalisms. Joint Intentions Theory defines teamwork based on agent’s intentions and commitments. SharedPlans extends this, in an more complex implementation, to include ways in which plans are formed and executed.

Joint Intention Theory provides a set of definitions for agent’s individual and team goals and intentions. These definitions provide a foundation from which numerous theorems further speculate about agents’ beliefs, desires and intentions. Joint Intentions Theory is widely accepted and implemented throughout the AI community and directly relates to the more popular AI paradigms such as Belief-Desire-Intention models. However, it does not directly relate to less traditional models, such as contextual reasoning models.
SharedPlans redefines collaborative behaviors and includes “a mental state view of plans.” (Grosz and Kraus, 1999) SharedPlans defines meta-predicates to represent agents’ beliefs. Among other definitions developed are those for Full Individual Plans, Partial Individual Plans, Full SharedPlans, Partial Shared Plans, SharedPlans, intentions-to, intentions-that. These meta-predicates are used to develop a complex and seemingly complicated system for specifying collaborative behaviors. However, as with Joint Intentions Theory, the specification does not relate well to alternate AI paradigms, such as contextual reasoning paradigms.

No contextual reasoning paradigm has yet included a formalism of collaborative behaviors. Additionally, the Context-Based Reasoning specification and description does not specifically relate CxBR to other reasoning paradigms, such as BDI, nor does it fully describe an agent’s reasoning and behavior in generally accepted AI terms such as belief, desire, and intentions. This dissertation aims to remedy these shortcomings by first formalizing the relation between CxBR and BDI and next deriving, from JIT, a formalism for collaborative behaviors in CxBR. This problem and solution will be expanded upon in Chapter 3.
2 PRIOR RESEARCH

In this chapter, two major areas are considered. First, implementations of collaborative behavior frameworks are reported in Section 2.1. This is followed by a discussion of various context modeling and context reasoning paradigms in Section 2.2.

2.1 Collaborative Behavior Framework Implementations

There have been many implementations of the more popular collaborative behavior theories JIT and SharedPlans. This section highlights two such implementations. STEAM, based on both JIT and SharedPlans, is discussed in Section 2.1.1, and GRATE, which utilizes a modeling paradigm – Responsibility Model. GRATE is based on JIT and SharedPlans, and is reviewed in Section 2.1.2. These implementations show that the collaborative behavior theories are at least mature enough to permit actual implementation.

2.1.1 STEAM

STEAM (Shell for TEAMwork) [Tambe 1997, Tambe et. al. 1999] is based primarily on Joint Intention Theory but uses some ideas from SharedPlans. Agents build a complex hierarchical structure of joint intentions, individual intentions and beliefs about others’ intentions. The shell, STEAM, is based on enhancements to the SOAR architecture including about 300 domain-
independent SOAR rules. Additional C-language rules are added to the shell beneath the higher layer of SOAR rules.

Numerous teamwork models are reported to be implemented using STEAM (Tambe et al. 1999), including a RoboCup simulated soccer team and a helicopter pilot agent used to simulate real world combat. The RoboCup team placed third against twenty-nine teams in the 1997 RoboCup simulated soccer league competition and fourth against thirty-seven teams in 1998. However, other than score and overall placement, there doesn’t seem to be any reported data on specific aspects of teamwork for this competition.

In addition to a single joint intention to guarantee teamwork, STEAM addresses four additional issues: coherence in teamwork, tradeoffs in the amount of information teammates maintain about one another, the analog of the ‘unreconciled’ case in SharedPlans, and a generalization of STEAM’s communication capabilities. STEAM’s reasoning is primarily a hierarchy of reactive states. Some of the modifications to the SOAR architecture include generalization of operators to represent team operators, representation of team states, and restrictions on team state modifications. (Tambe 1997)

Considered in terms of teamwork, STEAM provides a good hybridization of JIT and SharedPlans. Even more pertinent to the research covered in this dissertation, however, STEAM is a rule-based system and differs greatly from contextual reasoning paradigms such as CxBR. This fact is quite relevant to the need, explained in Chapter 3, to extend collaborative framework and definitions to include contextual reasoning paradigms.
2.1.2 GRATE

This is a multi-agent Distributed AI (DAI) system of cooperative agents. The agents’ collaborative efforts are based on Joint Intention Theory. GRATE (Generic Rules and Agent model Testbed Environment) [Jennings, 1995] is a prototype system for a larger project, ARCHON [Wittig, 1992].

The Responsibility Model is defined based on Joint Intention Theory. It provides a high-level architecture utilized for coordinating actions between cooperating agents. The model considers two separate constructs for modeling cooperative behavior. Individual and group behaviors are handled as distinct reasoning constructs. For individual behavior in a social context, intention is the controlling concept. Extending this to deal with groups of collaborating agents, joint intention becomes the controlling concept. Examining models concerned with joint intentions, the following points were made about cooperative problem solving:

- Agents must agree they wish to cooperate to achieve their joint goal [Cohen and Levesque 1991, Levesque et. al. 1990, Searle et. al. 1990, Tuomela and Miller 1988]
- Agents must agree on a common recipe for attaining their joint goal [Lochbaum et. a l. 1990]
Actions performed by different agents, in the context of the joint action, are interdependent [Lochbaum et al. 1990, Rao et al. 1992]

Agents must have conventions for monitoring the viability of their commitments [Levesque et al. 1990]” Jennings [1995]

Essentially, the Responsibility Model encapsulates all of JIT and includes parts of SharedPlans. In addition to the requirement of joint intentions, collaborating agents must agree upon a common recipe for accomplishing the goal.

The GRATE system is one of many possible implementations of the Responsibility Model as “the model does not specify how intentions are represented, how commitment is described, what mechanisms are used to obtain agreements, or how to develop the common recipe.” [Jennings, 1995] The following simplifying assumptions were made in implementing GRATE: communication is foolproof and message delay time is known to all agents, beliefs have one level of nesting, agents share a global clock reference, and agents are able to predict the time taken to execute their activities.

GRATE is employed in a distributed artificial intelligence (DAI) domain. The agents themselves comprise an expert system to diagnose complex system failures. These expert diagnostic agents are able to collaborate in maintaining and troubleshooting complex systems, yet this limited application is not entirely analogous to human behavior representation. However, intentions and plans are still critical in ascertaining collaboration of these expert DAI agents.
As with the other popular paradigms and their implementations, the Responsibility Model is founded on a BDI view of agent behavior and GRATE is an implementation based on this view. Alternate AI modeling paradigms, such as contextual reasoning, are not particularly considered in the formalization or the implementation of the Responsibility Model.

There are numerous other examples of teamwork implementations, including COLLAGEN (Rich and Sidner 1997) and ALLIANCE (Parker 1998), which utilize the SharedPlans and JIT. Other modern teamwork implementations include RAP teamwork models [Scerri et. al. 2003], and RMTDP [Nair et. al. 2003]. Most noteworthy, what these implementations all have in common is that none are based on a contextual reasoning paradigm.

Some of the current work related to collaborative behaviors tends to focus on particular aspects of collaboration such as communication and commitment [Dignum and Eijk 2007][Chaib-draa et. al. 2006] [Mallya and Signh 2007][ Flores et. al. 2007]. These approaches are not yet relevant at this stage of formalizing collaborative behaviors in Context Based Reasoning but should be considered in future efforts aimed at improving this research.

The following section examines contextual reasoning paradigms, including attempts at modeling collaborative behaviors.
2.2 Contextual Modeling and Reasoning Paradigms

Three major perspectives on contextual reasoning and modeling are discussed in this section. All three of these paradigms share many commonalities. However, there are many differences also, some subtle while others not so subtle. Because the research done here is primarily concerned with CxBR, the other paradigms are compared and contrasted to CxBR where appropriate. Context Mediated Behavior [Turner, 1998] is covered in Section 2.2.1. Next, contextual reasoning as applied in SART, and Context Graphs, [Brézillon et. al., 2000 and Brézillon 2003, 2004, and 2005] is discussed in Section 2.2.2. Finally, Context-based Reasoning [Gonzalez 1998, Stensrud et. al. 2004] is revisited in Section 2.2.3.

2.2.1 Context Mediated Behavior

In Context Mediated Behavior (CMB), contexts are captured and represented as context schemas, called c-schemas. These c-schemas contain information regarding: important features to cause the c-schema to be active, standing orders, events, goals, and actions. In addition to c-schemas, procedural schemas, p-schemas, are used to define actions for agents. A context manager (CM) is used to reason about which c-schemas should be active for a given context.

Contexts are distinguished from environment situations in Turner’s [1998] definition, “We use the term context to mean any identifiable configuration of environmental, mission-related, and agent-related features that has predictive power for an agent’s behavior. The term situation is used to refer to the entire set of circumstances surrounding an agent, including the agent’s own internal state. Context is thus the elements of the situation that should impact behavior.” Thus,
features in the environment that are inconsequential to the agents reasoning and behavior are not of concerning in defining a context from that environment. An example is given where vehicles are attempting to refuel. The relative position of the vehicles is an important feature of the situation while the color of the vehicles is not. Thus, a context for refueling could occur in many different situations, which contain varied, unimportant features.

Much of the concept involved with CMB is parallel to that of CxBR. One notable exception is the way that multiple c-schemas can be active or merged to create a new c-schema. Also different is that c-schemas are used in much the same manner as cases in case-based reasoning. This differs from CxBR, as Contexts in the implementation of CxBR provide a more active functionality. The reason for the divergence in implementation of context between the two paradigms is likely because of the context manager in CMB. In CxBR the context managing is done within the Context objects themselves.

The multiple selection of c-schemas in CMB is important relative to the research proposed here. This is very closely related to the concept of Co-Contexts proposed for CxBR, which would allow more than one concurrently active Context. However, because of the differing design and implementation of CMB and CxBR, namely the way contexts are managed, Co-contexts still pose interesting research possibilities within CxBR. This is discussed further in Chapter 7.

Context Mediated Behavior [Turner, 1998] is based on early work done in Context-sensitive Reasoning [Turner, 1993]. Some work was done in modeling collaborative behaviors between autonomous underwater vehicles using Context-sensitive Reasoning [Turner, 1994]. However,
nothing particularly novel was presented. The focus was on handling unexpected events, and most of the work reported was in the early stages of development. No evidence was found regarding the progression of this work in using contexts for reasoning about collaborative behavior. [Turner, 1996]

2.2.2 Contextual Reasoning as Applied in SART

SART (French acronym for “support system for traffic control”) [Brézillon, 2000] is an intelligent system used to help operators control a subway line. The intelligence is based on contextual reasoning. There are many parallels between the design of SART’s contextual reasoning and that of CMB and CxBR, with some notable differences. However, first, an explanation of some terms is necessary.

There are three major areas of knowledge representation: external knowledge, contextual knowledge, and procedural knowledge. Again, this is not different than the knowledge represented by CMB or CxBR except as a matter of semantics. External knowledge is implied to be the knowledge an agent knows outside of contextual knowledge and procedural knowledge, such as particular elements of events or characteristics regarding the environment. Contextual knowledge includes what is analogous to transition criteria in CxBR and any knowledge necessary to reason about what procedural context should be used. Procedural context knowledge is where an agent’s actions are defined. Brézillon [2002] explains, “Proceduralized context defines what the focus of attention is, and contextual knowledge defines the context of the focus of attention.”
Of particular interest in the way modeling of contexts is considered is the onion metaphor. The metaphor describes the distance of contextual knowledge with respect to a proceduralized context. Some bits of contextual knowledge are assumed to be more important to the action taken than others that become more relevant as details and time unfold. This is considered a layering of knowledge but seems somewhat analogous to weighting the knowledge. Brézillon does not clearly explain how this layer is accomplished in practice. However, it is somewhat analogous to the hierarchy of contexts established in CxBR with Sub-contexts and Sub-sub-Contexts, except that in CxBR these levels are traversed and settled upon before taking action on a higher level of Context. This is also similar to the implementation used in CMB.

Of most notable importance to the research of incorporating Co-contexts into CxBR is the way that these layers described above can be mutually active. It is not difficult to see that situations could occur in which the current implementation of CxBR does not account for all the subtleties of context layers, because they may be defined in separate Contexts or Sub-contexts, which cannot, at this time, be activated concurrently. An example taken from Brézillon [2002] will help illustrate this point.

Consider an ill traveler, who requires medical attention. The first contextual knowledge is that a sick traveler is on the train (subway). This is used to infer a proceduralized context of stop at the next station. Further contextual knowledge of do not stop in a tunnel and do not touch an injured passenger is also activated. This contextual knowledge is explained to be at a further distance from the proceduralized context of stop at the next station. Regarding CxBR, the layers
of contextual knowledge represented would be hard to capture in the current specification of the paradigm. The reason is that currently, the paradigm specifies that only one context can be active at any given point in time. The contextual knowledge of *do not stop in a tunnel* and *do not touch an injured passenger* could both be captured as Sub-Contexts of *sick traveler on the train*, but the Sub-Contexts could not be concurrently active. Of course, the knowledge of both of these contexts could be captured in one Sub-Context but then they are bound and not available for individual activation. Regardless, the modeling complexity seems to be increased beyond that required to simply allow both contexts to be concurrently active. This is further justification for research of Co-Contexts within CxBR.

The concept of distance between contextual knowledge and proceduralized knowledge is introduced as the onion metaphor by Brézillon [2002] but not expanded upon. Is this distance quantitative or qualitative? Is there such thing as a context distance unit or is it measured in relative terms such as near and far? There is no clear indication by Brézillon of how these distances are computed or used. Also, there is no clear indication of how the layers, which are partly representative of these distances, are built.

2.2.2.1 *Contextual Graphs Paradigm Applied to Group Activities*

The Contextual Graphs (CxG) paradigm underlying the SART implementation has not yet been applied to collaborative behaviors for autonomous agents. However, research has been recently performed in applying CxG in a groupware domain to facilitate collaboration of human agents in a small development team environment. [Borge et. al. 2004]
The research is applied to the CO2DE group editor to facilitate individuals collaborating to build/edit a diagram as a group. Contextual knowledge of an individual user is processed and stored for other users to access when attempting to understand the motivation of the original user. “Capture, storage, awareness, and visualization, are all processing steps done by the system on the basis of user’s specification and pre-established rules.” [Borges et. al. 2004]

Although the purpose of this work is not exactly to model collaborative behaviors but to facilitate collaboration of human agents, it is nevertheless relevant to the topic, especially since it deals with collaboration and context.

Borges et. al. [2004] state, “Working in a group supposes to manage context explicitly. Not only individual contexts need to be proceduralized, but also the group context. Group context involves all knowledge relating to the group, including group composition, rules, roles, goals, strategies, coordination procedures, etc. Therefore, group context is not simply the union or intersection of individual contexts (the whole is not the sum of the parts).” This is certainly a true reflection of modeling group context. Some preliminary work in this area was already performed by Barrett and Gonzalez [2002] but not referenced by Borges. In fact, the concept of Embedded Context is in part what is described by Borges et. al. above. Embedded Contexts were first implemented in CxBR by Barrett and Gonzalez [2002] and are described in this dissertation in the chapter on relevant work. It is also described briefly in this chapter under section 2.2.3.3. Although the concept, which was introduced by Barrett and Gonzalez [2002] has been implemented, it has not been rigorously tested, and, therefore, is not presented directly with
The main research described in this dissertation. However, a discussion of the topic is provided in Chapter 7: Relevant Ideas.

The work done by Borge et. al. [2004] does address context in a group setting. However, it does not attempt to use these contexts for autonomous reasoning. Furthermore, no connection is made by the authors between the research and the commonly accepted descriptions of collaboration. The main focus of the work is in relating context to awareness and attempting to parse this and display it in a way that is useful to team members.

2.2.3 Context-based Reasoning

An introduction to Context-based Reasoning (CxBR) is provided in Section 1.3. The focus here will be on previous work done towards modeling collaborative behaviors in CxBR.

2.2.3.1 Cooperating AIPs in the Context-Based Reasoning Paradigm: Johansson’s Research

The work performed by Johansson [1999] did much in terms of modeling collaborative behaviors in CxBR. He modified the CxBR framework built by Norlander [1999] to provide a mechanism for facilitating collaboration between agents. A Teamworking AIP Class was introduced to the framework. This newly-introduced class focuses primarily on communication between collaborating agents.

Johansson’s incorporation of collaborative behavior capabilities in CxBR includes assignment of a team mission, team-mission-context, to the group. The team mission includes a specification of
the sub-goals for individual team member AIPs. [Devero and Gonzalez 2001] (note: Lars Johansson changed his name to Lars Devero in 2000) Sub-goals are defined by Contexts, which is consistent with the CxBR specification. [Stensrud et. al. 2004]

Devero and Gonzalez [2001] state, “With the knowledge gathered from the Joint Intentions Theory and its derivatives, the items needed for the concept of cooperating AIPs were identified and incorporated in the CxBR paradigm.” It is also explained that:

An AIP that works as part of a team needs to have information about the following items:
1. Must know status of itself and possibly, other AIPs.
2. What is the team mission?
3. When is the team objective reached?
4. What is one individual AIP’s part of the mission?
5. How does an AIP communicate with the other AIPs?
6. When does it desist in its attempt to reach its sub-goal?
7. How and when does it change its sub-goal?

Devero and Gonzalez [2001] imply that these requirements are self-evident. While the implementation of Johansson’s work does essentially follow JIT theorems, there is no formal proof or description justifying this. The inclusion of team goals and sub-goals within team-mission-context essentially provides for consistency with the JIT specification. However, there is no description in Johansson [1999] nor Devero and Gonzalez [2001] that links CxBR constructs, such as Missions and Contexts, to JIT language, such as belief, desire, intention, and commitment. Additionally, no extension to JIT or the other collaborative behavior theories is presented, although it is promised and deemed necessary. The implementation of this work performed by Devero and Gonzalez [2001] depends upon an explicit teamwork model. While this is certainly a valued approach, it does place constraints on the teamwork models
implemented. An alternative method could allow models to be implemented without explicitly specifying the teamwork model. One such approach could rely upon an implicit team-mission-context, which could be an extension to the concept of implicit individual agent mission-context, which is discussed in Chapter 7.

There is a need to formalize the connection between contextual reasoning paradigms and commonly accepted collaborative behavior models. This problem is explained in greater detail in the following chapter.
3 PROBLEM DEFINITION

This chapter provides a concise statement of the problem that is the motivation for the hypothesis and research performed. First, the larger problem concerning the lack of formalized collaborative theories relation to alternate modeling paradigms is described. Next, this problem is analyzed in terms of how it relates specifically to Context-Based Reasoning. The final two sections briefly discuss a hypothetical remedy and its contributions.

3.1 General Problem

The major collaborative behavior paradigms do not specifically address particular modeling paradigms, However, they do rely heavily on the BDI modeling paradigm in their formalisms. Joint Intentions Theory defines teamwork based on agents’ intentions and commitments. SharedPlans extends this, in a complex implementation, to include ways in which plans are formed and executed.

Joint Intentions Theory provides a set of definitions for agents’ individual and team goals and intentions. These definitions provide a foundation from which numerous theorems further speculate about agents’ beliefs, desires and intentions. Joint Intentions Theory is widely accepted and implemented throughout the AI community and directly relates to the more popular AI paradigms such as Belief-Desire-Intention models. However, it does not directly relate to less traditional models, such as contextual reasoning models.
SharedPlans redefines collaborative behaviors and includes “a mental state view of plans.” (Grosz and Kraus, 1999) SharedPlans defines meta-predicates to represent agent’s beliefs. Among other definitions developed are those for Full Individual Plans, Partial Individual Plans, Full SharedPlans, Partial Shared Plans, SharedPlans, intentions-to, intentions-that. These meta-predicates are used to develop a complex and complicated system for specifying collaborative behaviors. However, as with Joint Intentions Theory, the specification does not explicitly relate to alternate AI paradigms, such as contextual reasoning. Again, as with JIT, SharedPlans is defined in terms of BDI.

3.2 Specific Problem

No contextual reasoning paradigm has yet included a formalism of collaborative behaviors. Additionally, the Context Based Reasoning specification and description does not specifically relate CxBR to other reasoning paradigms, such as BDI, nor does it fully describe an agent’s reasoning and behavior in terms such as belief, desire, and intentions.

3.3 Motivation

Modeling collaborative behaviors in a contextual reasoning paradigm offers potential benefits over other modeling paradigms, such as: simplicity of representation, ability to predict teammates’ intentions and actions, and greater ease of communication by conveying the current context or the next context. Effectively modeling collaboration in contextual reasoning
paradigms requires a common grounding in perspective. This common grounding could be accomplished by adopting the standard of other modeling paradigms, in particular JIT.

3.4 Hypothesis

1) A formalization of collaborative behaviors – derived from Joint Intention Theory – in Context-Based Reasoning will provide an **effective** means of modeling teamwork.

2) Modeling CxBR agents that infer intention from explicitly communicating Contexts allows for more **efficient** modeling of collaborative behaviors than modeling agents that infer intention from situational awareness.

3.5 Contributions of Dissertation to Community

The research presented in this dissertation provides the following contributions to the AI and the modeling and simulation (M&S) community:

1. Provide a formalism linking CxBR to BDI, which allows for a formalization of collaborative behaviors in CxBR founded on the definitions set forth by JIT.

2. Provide a formalization of collaborative behaviors for agents utilizing a form of contextual reasoning, specifically CxBR.

3. Provide test data collected during the use of a well-known, freely-available infrastructure, the soccer simulator [Buckland, 2005].
4. Provide source code for the Collaborative Context-Based Reasoning (CCxBR) team prototype used for testing.

The following chapter specifies the conceptual approach to solving the problem explained in this chapter.
4 CONCEPTUAL APPROACH

The concept of teamwork within a contextual-reasoning-perspective is approached starting from CxBR and justifying collaboration based on JIT. To this end, BDI and CxBR are related in Section 4.1, while Section 4.2 provides a formal method of specifying teamwork in CxBR justified by JIT. This is followed by a description of building teams in *Context-Based Reasoning*, Section 4.3. Section 4.4 discusses intention recognition in CxBR and Sub-section 4.4.2 explains *Collaborative Context-Based Reasoning* (CCxBR) where Contexts are explicitly communicated for the purpose of establishing joint intentions. Finally, Section 4.5 approaches the concept of teamwork with CxBR from an entirely different, yet very consistent perspective, as the formalism provided in Section 4.2.

The relationship between the concepts of this chapter and the hypothesis of Chapter 3 is as follows:

1. In Section 4.2 a formalism for collaboration in CxBR is provided. As the hypothesis states the formalism will provide an effective means to model teamwork in CxBR.

2. In Section 4.4 intention recognition through explicitly communicating Contexts is discussed along with intention recognition through inference. As the second part of the hypothesis states, explicitly communicating Contexts will allow for more efficient modeling of teamwork.
In order to effectively model collaborative agents utilizing CxBR, it is first necessary to formalize modeling collaborative behaviors for the paradigm. The approach taken here first relates CxBR to the more popular Belief-Desire-Intention (BDI) [Georgeff et al. 1999] models. This relation serves two purposes. First, developers unfamiliar with CxBR are provided a familiar base of reference. Moreover, the more significant reason is that the current theories on collaboration: JIT [Cohen and Levesque 1991] and SharedPlans [Grosz and Kraus 1999] are primarily concerned with reasoning from a BDI perspective. The formalization of collaborative behaviors within CxBR is founded on the most widely accepted of these collaborative theories, Joint Intentions Theory (JIT) [Cohen and Levesque 1991], [Tambe 1997], [Jennings 1993], and [Jennings 1996].

Much of this chapter directly relates to the background discussion provided for both BDI and JIT. It may be necessary for the reader to refer back to those sections often, especially section 1.1 Intelligent Agents and section 1.2.2 Joint Intentions, to understand the formalization presented in this chapter.

4.1 BDI and CxBR Formalisms

In this section, a preliminary attempt at formalizing aspects of CxBR is presented in the form of corollaries. A complete formalization for CxBR does not yet exist and is not provided here. Rather, the aspects focused on are, of course, those most pertinent to building a foundation for modeling collaborative behaviors in CxBR. For the purpose of discussing collaborative
behaviors and teamwork the most noteworthy aspect relating CxBR to BDI is that CxBR fully encompasses the BDI structure. It does so in the following way.

Beliefs: Represent an agent’s knowledge about its environment. CxBR is a paradigm designed to represent an agent’s knowledge through Missions, Major Contexts, and Sub-Contexts. However, an agent’s environmental knowledge is largely independent of the CxBR structure.

Environmental knowledge could be stored in any suitable data structure. It is also important to note that although CxBR provides an organization for representing tactical knowledge in terms of Missions, Major Contexts, and Sub-Contexts – the exact AI representation of this knowledge is not restricted by CxBR. For example, the knowledge required for transitioning to any given Context could be captured in rules or could be contained in a neural network. Both rules [Gonzalez 1998, Norlander 1999, Barrett et. al. 2003, Stensrud et. al. 2004] and neural networks [Stensrud 2005] have been successfully implemented as means to determine Context transitions. CxBR is only concerned with the organization of the knowledge and not the specific AI implementation used to reason about the knowledge.

Desires: Correspond to goals of the agent. A CxBR Mission includes the highest level goal for an agent. Sub-goals, primarily concerned with accomplishing the
Mission’s goal, are either implicitly or explicitly contained within Contexts and Sub-Contexts.

Provided an oversimplified example of a driving scenario: *return home from work*, an obvious CxBR Mission *returnHomeFromWorkMission* would include the overall goal of *reach home*, which implicitly includes sub-goals of *avoid accidents with vehicles* and *avoid accidents with pedestrians*. A few of the likely Contexts contained in this Mission might be: *highwayDrivingContext*, *cityDrivingContext*, and *vehicleFailureContext*. Regardless of which Context is Active, the highest-level goal remains *reach home*. However, each Context is likely concerned with a set of sub-goals, which are likely necessary to accomplish the highest-level Mission goal. For instance, during *cityDrivingContext* the sub-goal of *obey traffic control devices* is certainly pertinent. This sub-goal *obey traffic control devices* allows for the fulfillment of other goals which may not be explicitly stated such as: *avoid citations* and *avoid collisions*.

The example above shows that an agent’s desires are present, both explicitly and implicitly, throughout the context hierarchy.

**Intentions:** Are commitments toward a plan. A CxBR plan consists of a sequence of Major Contexts. During a scenario the plan is composed of the series of sequential Active Contexts. An agent’s current Active Context determines the agent’s
current plan. Therefore, an agent’s intentions are determined by the agent’s Active Context.

**Corollary 1:** High level desires and goals are captured in CxBR Missions. Lower sub-goals and desires are captured in the Major Contexts and Sub-Contexts associated with the Mission.

To allow for more specific definitions and theorems of collaboration in CxBR, the formalism for transition logic must be expanded. “In order for a Context to be Active certain prerequisites must be met (with the exception of choosing a default Active Context). The prerequisites for choosing an Active Context are specified as the Transition Criteria and captured as part of the Transition Logic.” [Stensrud et. al. 2004] Transition Logic should be considered as a set of criteria. It is possible that a criterion member from this set is sufficient for causing a Context transition, yet this criterion member may not be strictly necessary for the transition. This criterion belongs to a disjunctive set of transition criteria. In contrast, there may be certain criteria that are necessary for Context transition. In other words, in order for the Context transition to occur, certain criteria are absolutely required. These elements belong to a conjunctive set, which will be termed Transition Requirements. Hence, Context $c$ with Transition Requirement $q$ is Active iff $q$ is believed true.

**Corollary 2:** Given that fact $q$ is a transition requirement of Context $c$, if $c$ is the Active Context, then $q$ is believed true.
Corollary 3: Given that Context $c$ is part of the context set associated with Mission $m$, and $m$ contains goal $g$, if $c$ is the Active Context for an agent with Mission $m$, then the agent has Mission goal $g$ and any other goals (Mission sub-goals) of $c$.

Corollary 4: The sequence of Active Contexts for a CxBR agent is the agent’s plan.

Corollary 5: The currently Active Context indicates an agent’s commitment to achieve the Active Context’s goal (the Mission’s current sub-goal) and the Mission’s goal.

These corollaries will be referred to later in this chapter during the explanation for implementing JIT in CxBR. Before that, however, an examination of Joint Intention Theory from a contextual reasoning perspective is presented in the following section.

4.2 Collaboration in CxBR

A major obstacle for agents seeking collaboration is in verifying the intentions and beliefs of prospective collaborators. It is expected that the complexity of this verification can be reduced through communicating in Contexts. In order for a Context to be Active, for a correctly specified CxBR model, there are certain conditions that are known to be true and other conditions that are implied. Knowledge of these conditions provides insight into an agent’s beliefs, desires, and intentions. For simulated agents, each team member could be provided with knowledge of the other team members’ Context and Mission specification. This knowledge includes transition requirements and goals of the Contexts and Mission. Therefore, once a
prospective collaborating agent’s Active Context is recognized or communicated, the recipient can infer the prospective agent’s beliefs, desires and intentions. This inference can then be used to help determine whether or not collaboration exists between the agents. This understanding of belief, desire, and intention based on knowledge of a teammate’s currently Active Context is the essence of CCxBR.

**Theorem 1:** A CxBR agent has a **persistent goal** (individual commitment) relative to $q$ to achieve $p$ iff: The Active Context $c$ has transition requirement $q$ and goal $p$ or if the Active Context $c$ has transition requirement $q$ and the Mission $m$ has goal $p$.

This is justified from the JIT Definition 1 of a persistent goal and CxBR corollaries 2 and 3.

**Theorem 2:** A CxBR agent **intends** relative to some condition, consisting at a minimum of some transition criterion, to do an action determined by the agent’s Active Context for the sake of a persistent goal.

This theorem is adapted from the JIT Definition 2 for intentions and the fact that actions in CxBR are determined by the agent’s Active Context.

**Theorem 3:** A CxBR agent has a **weak achievement goal** relative to $q$ and with respect to a team to bring about $p$ if: The agent’s Active Context is part of Mission $m$’s related Contexts intended to accomplish $p$, or the agent has a goal that the status of $p$ be mutually believed by all other teammates, regardless of the current state of $p$ (true, false, or irrelevant).
This is justified by the JIT definition for weak achievement goal and the fact that the sequence of Contexts forms a plan that allows the accomplishment of the Mission goal.

**Theorem 4:** A team of CxBR agents has a **joint persistent goal** relative to \( q \) to achieve \( p \) if each agent shares the same team Mission with goal \( p \).

This is justified by the JIT Definition 4 for joint persistent goal and Corollary 1.

**Theorem 5:** A team of CxBR agents **jointly intends** to do some action iff they share the same team Mission and believe they are accomplishing their Mission by following a plan of Mission-related Contexts.

**Theorem 6:** Theorem 4 and theorem 5 could both be extended to include Context rather than Mission as a matter of scale.

This is justified since Mission is essentially a special form of context. “The general idea of contexts is sub-divided hierarchically into three types: These are 1) the *Mission Context*, 2) the *Major-Contexts* and 3) the *Sub-contexts*.” [Gonzalez and Ahlers, 1998] When considering a Context hierarchy including multiple Missions, the Missions themselves become the Context-set of some higher Mission. “No more than one Mission will be active at any one time, and Missions are mutually exclusive. So, a new Mission would have to bump an existing Mission from active status. While it is conceivable that in real life a submarine may be charged with simultaneous
multiple missions, in practice, however, there would be little need to change Mission Contexts during the course of a training session.” [Gonzalez and Ahler, 1998] It could be argued that the representation of multiple Missions is simply a matter of scale on which the simulated agent is expected to operate. An agent capable of multiple Missions is certainly more versatile than an agent capable of a single Mission. In real world applications, where agents represent humans or autonomous robots, agents only capable of a single Mission are quite handicapped in their usefulness. For the purpose of this research, the Context hierarchy is assumed to be scalable beyond one Mission.

Based on JIT definitions and CxBR corollaries, the theorems detailed here provide a foundation for modeling teams in CxBR. The next section will outline the building of such models.

4.3 Modeling Teams with CxBR

The assignment of Missions and their associated Contexts through the CxBR paradigm provides a feasible method of modeling abstract team entities as well as the team members themselves. Consider a real world analogy. A business might have an associated mission statement, which is distinct from any missions that the business employees might themselves have. Thus, a CxBR model of such a small business could represent the business itself as an entity that is composed of some small number of employees. A larger business, on the other hand, might be represented as a number of departments, which are are composed of a number of employees. In this case, the business, each department, and the employees could each be represented as a distinct entity, including a specific Mission (and associated Contexts) for each entity.
4.3.1 Simple CxBR Model

To better facilitate understanding hierarchical team modeling in CxBR, first, consider a simple CxBR model. Such a model is diagramed in Figure 4.1. As shown, each agent is assigned a Mission, which includes a specified mission goal. The Mission itself specifies a set of associated Contexts that the agent will likely encounter while attempting to accomplish the mission goal. Each Context includes a set of Transition Criteria and Transition Requirements. These Criteria and Requirements specify the conditions that will cause the associated Context to become Active. Contexts may have a set of associated Sub-Contexts. Sub-Contexts are essentially Contexts at a lower level of resolution. For example, a driver agent may have a Mission of drive-home-from-work. The Mission may include Contexts such as highway-driving, city-driving, emergency-road-service. The highway-driving Context could have Sub-Contexts of passing-other-driver, being-passed-by-other-driver, and exit-highway. A Transition Requirement for the exit-highway Sub-Context might be approaching-highway-exit, while a Transition Criterion for the same Sub-Context might be need-to-refuel. The fact base is used to store facts that the agent is aware of, including perceptions, beliefs, and inferences about the environment and other agents.
Teams not requiring a structured chain-of-command hierarchy can be modeled following the structure outlined in this subsection. However, teams requiring a structured chain-of-command are better modeled by implementing a Team Construct Class, as outlined in the next subsection. The justification for choosing a Team Construct Class or not is thought to be intuitively obvious, once the following example is understand. The benefit is that the chain-of-command or hierarchical command structure can be explicitly modeled using the Team Construct Class.

Figure 4.1 Simplified CxBR Model (UML class diagram - diamond denotes composition)
4.3.2 Modeling Team Hierarchies

The complexity of representing complex team structures is simplified by adding a layer of abstraction to the simplified model discussed in the previous subsection. Teams and team members alike can be represented as intelligent entities. This allows for clear representation of teams’ missions, goals, and objectives in addition to individual team members’ missions, goals, and objectives.

The addition of a Team Construct Class, as illustrated in Figure 4.2, allows for nesting of team hierarchies. An example realization of such a nesting, a platoon of tanks, is illustrated in Figure 4.3. As shown, a tank platoon is composed of two sections, each of which is composed of two
tanks. The platoon itself represents a team of sections, yet each section is a team of two tanks. At such a small scale, such distinctions might seem trivial. However, the hierarchy provides a means of scalability for larger formations, such as companies, divisions, battalions, brigades, etc. Although there are undoubtedly limits to scalability, these limits are not because of representation.

Figure 4.3 Sample Model of Tank Platoon (UML class diagram - triangle denotes inheritance, diamond denotes composition)

Modeling abstract entities such as a team, platoon, business, etc., which represent group formations, present certain challenges compared to modeling concrete entities such as a team member, a tank, or a sales manager. For example, the concept of applying a mission to an abstract entity such as a business or team differs in some respects from applying a mission to an
agent such as a salesman or a ball player. These team formations present emergent properties making them greater than the sum of their individual members, which is, of course, a result of collaboration. “Collaborative behavior – coordinated activity in which the participants work jointly with each other to satisfy a shared goal – is more than the sum of individual acts [Searle 1990] and [Grosz and Sidner 1990] and may be distinguished from both interaction and simple coordination in terms of the commitments agents make to each other [Bratman 1992], [Grosz and Kraus 1996], and [Grosz 1996].” [Tambe 1997] Thus, the mission of a team is also more than the combination of individual team member missions. Worthy of consideration, however, is the method for modeling both a team and its individual team members, to allow for simultaneous representation of each within the same model. At the lowest level of representation for entities, the bottom layer of the team hierarchy, the agents’ physical behaviors should be modeled, while the highest layer provides for representation of abstract team formations – as intelligent entities.

Consider the tank platoon illustrated in Figure 4.3. As shown the lowest level of this team hierarchy represents the tanks. The second level of the hierarchy represents the tanks sections. And, the top level of the hierarchy is the tank platoon. Characteristics of a tank section should emerge from the behavior of the tanks within the section. Likewise, the characteristics of the platoon emerge from the sections behaviors, which emerge from the tanks behavior. In this regard, the team construct is an abstraction that acts as a container whose behavior represents the emergent behavior of the individual team members.

In contrast to the above example, consider a model that does not require as fine a resolution. The specification of the model may be concerned with the behavior of a tank platoon but may not
require/desire modeling to the level of individual tanks. In this case, the tank platoon is modeled as a single entity versus a team of sections composed of tanks.

4.4 Intention Recognition

As established in Section 4.1 CxBR agents’ current intentions are indicated by their respective Active Contexts. Theorems 5 and Theorems 6 from Section 4.2 explain that joint intentions between two CxBR team members, require the sharing of a common Mission (Theorem 5) or the sharing of a common Context (Theorem 6). In addition, each agent must be aware that it shares the same Active Context with the other, as Context awareness implicitly allows for intention awareness. In order to recognize each others’ intentions then, CxBR agents need only be aware of each others’ currently Active Context. This Active Context awareness could come from inference, as discussed in section 4.4.1, or it could come from explicit communication, as discussed in Section 4.4.2.

4.4.1 Collaborative Algorithm

Given the formalism of Section 4.2, the algorithm for collaborating in Context-Based Reasoning is simple and straightforward enough that it is perhaps subtly elusive. Therefore, an elaboration is provided here.

To paraphrase Theorem 5 and Theorem 6, two agents collaborate towards a common goal iff they share an Active Mission, Context, or Sub-Context with that goal and are both mutually aware of sharing this goal. Given this the collaborative algorithm for CxBR agents A and B, both of whom share the same Context Topology, is:
• Agent A currently has an Active Context $Cx$ with Goal $G_x$ or Agent A currently has for an Active Context a Sub-Context of $Cx$, $CxS$.

• Agent B infers Agent A’s Active Context to be $Cx$ or $CxS$

• From knowing Agent A’s Active Context, Agent B knows of Agent A’s intention to achieve Goal $G_x$

• Being aware of Agent A’s intention to achieve $G_x$ and, likewise, desiring the achievement of Goal $G_x$, Agent B transitions to an Active Context of either $Cx$ or a Sub Context of $Cx$

• Agent A becomes aware of Agent B’s Active Context and therefore becomes aware of Agent B’s commitment to $G_x$

• Given this awareness, Agent A continues with its commitment to $G_x$ with an Active Context of $Cx$ or a Sub-Context of $Cx$.

• The agents are now collaborating – they are both mutually committed to the achievement of $G_x$ (indicated by their respective Active Contexts) and they are both mutually aware of the other’s commitment.

A more detailed perspective of how this algorithm is implemented is provided in Section 5.2.3. The following sub-sections discuss a couple of possible mechanisms for Context recognition (which provides a means for intention recognition by CxBR agents).
4.4.2 Context Inference

As discussed in Section 4.1 every Context in CxBR has an associated set of Transition Conditions. When a Context’s Transition Conditions is satisfied then the Context is Active. It is possible then for an agent to infer the currently Active Context of another agent if the first agent:

- Knows the other agents Context topology and
- Knows the other agents perception of the environment.

Both pieces of this knowledge puzzle could be challenging for an agent to obtain, depending on the team model, simulation, or agents involved. It is not unreasonable, however, to expect that a team of agents be provided with a priori knowledge of each others context topology and the ability to communicate perceived knowledge of the environment. Moreover, homogeneous agents in a team, situated in the same environment, are likely to have the same Context Topology and could easily have the same perception of the environment.

To the point of Context inference, for the purpose of intention recognition, a CxBR agent that is aware of another CxBR agent’s Context Topology can infer that agent’s currently Active Context by inferring which Transition Conditions have fired. Knowing the agent’s currently Active Context then allows for the agent to know if it shares a joint intention.

4.4.3 Explicitly Communicating Context - CCxBR

As the section heading here makes almost self-explanatory, currently Active Contexts can simply be communicated. This eliminates the concerns of the previous section, 4.4.1, regarding
perception. This method of knowing the intentions of another agent through explicit communication of the agent’s Active Context is the essential technique of Collaborative Context-Based Reasoning (CCxBR).

4.5 Chapter Summary

There are two concepts developed in this chapter that are of notable importance and relevance to the hypothesis of this dissertation, stated in Chapter 3. First, collaboration in CxBR is formalized by explaining the relationship between CxBR and BDI and then extending JIT for CxBR. A set of theorems are developed for CxBR which should allow for modeling effective teamwork. Second, explicit communication of Context is contrasted with inferring Context for the sake of intention recognition. The explicit communication of Context is the essence of CCxBR and should allow for efficient modeling of teamwork. The effective and efficient aspects of modeling teamwork in CxBR are key to testing the hypothesis, which will be explained in Chapter 6 but, before that, the next chapter will discuss the prototypes built for testing the hypothesis.
5 PROTOTYPES’ DESCRIPTION

The corollaries and theorems specified in Chapter 4 regarding collaboration amongst CxBR agents deal primarily with understanding intentions through the beliefs that an agent has regarding another agent’s Active Context. In other words, agent1 can infer certain facts about agent2 based on knowing agent2’s Active Context, assuming agent1 possesses some understanding of agent2’s Contexts. The theorems propose a method of ascertaining collaboration through this knowledge of prospective teammates Contexts and Mission(s).

As a way of testing this, three models were built. The testing itself, as well as the test results, is the subject of Chapter 6. However, it is introduced briefly here in order to show the motivation for building these three prototypes. All three models represent a soccer team capable of playing on Mat Buckland’s Simple Soccer simulator [Buckland 2005]. The first of these teams neither include the ability to recognize and reason about teammates’ intentions nor their Active Contexts. The second model provides a means for agents to infer intentions based on teammates observed actions. The third model incorporates the theories set forth in Chapter 4 resulting in a CCxBR prototype. Agents in this model are able to know the Active Contexts of their teammates and infer their teammates’ intentions based on their Active Context. Each of these models is discussed further in their respective sections, of this chapter. Again, the exact nature of the tests and the motivation for this type of testing is discussed in the following chapter.
5.1 Simple Soccer

This section provides an overview of the Simple Soccer simulator and outlines the characteristics common to all the soccer team prototype models which will be built for testing the collaborative theory for CxBR presented in the previous chapter. First however, it is worth noting the motivation for choosing the Simple Soccer Simulator.

5.1.1 Prototyping Environment Motivation

The Simple Soccer Simulator was not the first platform chosen for implementing soccer teams and testing the theories of this research. The first prototyping approach was attempted using the RoboCup soccer simulator. The RoboCup simulator provides a rich environment for developing and testing simulated soccer teams. However, it is the richness – and the correspondingly imposed constraints – of the RoboCup simulator environment that ultimately made it a less than desirable choice as a prototyping and testing platform for this research.

As an example of this richness and its constraint, simulated players in the RoboCup environment cannot recognize with certainty all players on the field. This is a great feature and constraint that forces players to deal with perception. Players could account for recognizing another player at a great distance by storing last known locations and reasoning about likely or possible player positions based on time, player position, player’s max speed, etc. Remember, this is one example of the richness and constraints of the RoboCup environment.

Building players that can deal with the perception and recognition problem offered as an example above detracts from the focus of building players for testing the theories of this
research. Additionally, the issues associated with perception (again, one example of the richness and constraints) actually cause noise in the test data by making it harder to isolate the desired behaviors for testing.

By contrast, the Simple Soccer Simulator lacks the richness of the RoboCup simulator and also lacks the corresponding constraints. The Simple Soccer Simulator provides an environment that is much more conducive to quickly building a select set of behaviors and easily isolating those behaviors during testing.

5.1.2 Simple Soccer Simulator

The Simple Soccer Simulator (SSS) provides a rectangular playing field, enclosed by walls. (This is similar to indoor soccer.) Two goals are located one at each end of the field. Two opposing teams of five players, four field players and one goalie, attempt to score points by kicking the ball through the opposing team’s goal, which is guarded by that team’s goalie. Following kick-off, play continues until a goal is scored, at which time the ball is placed back in the center of the field to resume play at another kick-off. The simulator’s implementation is not particularly relevant to the research of this project, so a technical description is not provided here.

For the curious reader, Buckland [2005] provides a detailed technical description in Chapter 4 of his book “Programming Game AI by Example.” In addition the well commented source code and pre-compiled executables can be downloaded from www.wordware.com/files/ai. This source code combined with the source code contained in Appendix B would allow a reproduction
of the prototypes described in this chapter and used for testing of Chapter 3’s hypothesis, explained in the next chapter.

5.1.3 Simple Soccer Team

The Simple Soccer Team included with the Simple Soccer Simulator is important to this research because the Simple Soccer Team is used as a base for all of the prototype teams to be tested. The characteristics described below are thus shared by all teams. In the following sections of this chapter, characteristics specific to each of the prototype teams will be described, including Table 5.3.1 which highlights the most important similarities and differences.

Buckland [2005] includes a soccer team with the Simple Soccer Simulator. The team consists of five players: a goalie, two defenders, and two attackers. Basic heuristics are implemented using steering behaviors and a Finite State Machine (FSM).

Players are provided the following steering behaviors:

- **Seek** causes the player to move towards a target location. Speed is not adjusted as the player approaches the target so overshoot is likely.

- **Arrive** is similar seek except deceleration is applied as the player approaches the target location. The deceleration allows the player to arrive at the target location without overshoot.

- **Separation** steers players away from other players.

- **Pursuit** anticipates the destination of a moving object and steers the player in the direction of the objects destination.
• **Interpose** computes the midpoint between two objects of interest (the ball and an opposing player) and steers the player to this midpoint.

Field players utilize eight states:

• **GlobalPlayerState** primarily provides message routing. Messages include `Msg_SupportAttacker`, `Msg_GoHome`, `Msg_ReceiveBall`, `Msg_PassToMe`, `Msg_Wait`. As can be inferred by the similarity between the message names and the state names, the messages often determine a player’s next active state.

• **Wait** causes the player to stay positioned at steering behavior’s target location.

• **ReceiveBall** causes the player to either *pursuit* the ball or *arrive* at the ball’s target. The choice between *pursuit* and *arrive* is determined by three factors: randomness, an opposing player being within a threatening radius, or if the receive is in the third of the soccer pitch closest to the opponent’s goal.

• **KickBall** provides logic for shooting at the goal or attempting a pass to another player.

• **Dribble** allows the player to do as the name implies.

• **ChaseBall** is used when the player’s team is not in possession of the ball and the player is the closest team member to the ball.

• **ReturnToHomeRegion** cause the player to return to its home region, defined based on player type (right defender, left defender, right attacker, left attacker, goalie)
• *SupportAttacker* allows the player to provide support to a teammate that is in control of the ball. Support spots are determined heuristically based on how well the players might be able to complete a pass and proximity to the opponent’s goal.

![State Graph for Simple Soccer Team](image)

**Figure 5.1 State Graph for Simple Soccer Team**

### 5.2 CxBR Team Models

In addition to the basic functionality common to all team models, all CxBR prototype teams will also possess similar tactical abilities. Tactical abilities allow the teams to play at something more than a rudimentary level of play. In other words, the teams should be capable of more intelligent play than simply chasing the ball around the field in what is described as a ‘bumble bee’ fashion for teams of young and inexperienced human players. As mentioned in the previous section, the Simple Soccer Team provides some rudimentary tactical heuristics which will be
retained by the CxBR prototypes. Possessing the same basic tactics for all models allows a more fair comparison of the collaborative characteristics of each team during testing. This section highlights the models’ tactics, common to all CxBR teams, beginning with a basic explanation of plays, followed by Context Topology and structure.

5.2.1 Play Descriptions

This section provides a basic explanation of the tactical team plays implemented in the CxBR models. There are three two-player combination plays. Each play is illustrated and described below. The reader is assumed to have some basic familiarity with the game of soccer. Therefore, simple skills such as passing and dribbling are not reviewed in this document. In contrast, the tactical elements of the combination plays discussed in this section are critical to the implementation of their associated Contexts and relevant to the matter of collaboration. It should be noted that the play nomenclature given here is the authors description and not necessarily common to the game.
Like all the plays illustrated in this section, the diagonal combination play, Figure 5.2, is primarily meant to provide a controlled and somewhat evasive means of moving the ball downfield.
The box combination play, depicted in Figure 5.3, provides an alternate method to quickly move the ball downfield. It is important when playing against skilled and reasoning entities to possess an arsenal of various tactics to accomplish like tasks. Any thinking and learning team would quickly learn to foil a single strategy used exclusively to accomplish a task. Thus, numerous plays are implemented to provide various means of accomplishing the task of ball movement. For the CxBR prototype teams three such combination plays are implemented. The two player bounding combination play is illustrated in Figure 5.4.
Each play illustrated in this section corresponds to a Context of similar name. Captured within the Context is the functionality for players to move the ball according to design. All CxBR prototypes are provided these tactical Contexts, including the least collaborative model. This is meant to provide as fair an attempt at modeling team dynamics within the collaborative paradigm chosen for each prototype. The differences in the way the prototypes handle collaboration is discussed in the following section.

5.2.2 Context Structure

Each of the three combination plays from the previous section are modeled as a context and named: ComboDiagonalContext, ComboBoundingContext, and ComboBoxContexts for the
diagonal, bounding, and box plays respectively. ComboContext is only naming convention, which is used in the prototypes. ComboContext does not have any special meaning in CxBR or the collaborative concepts discussed in this dissertation.

Each ComboContext contains three Sub-Contexts. The Sub-Contexts of ComboDiagonalContext are ComboDiagonalControl, ComboDiagonalSupport, and ComboDiagonalReceive, as illustrated in Figure 5.5. Also shown in Figure 5.5, the Sub-Contexts of ComboBoxContext are ComboBoxControl, ComboBoxSupport, and ComboBoxReceive, and the Sub-Contexts of ComboBoundingContext are ComboBoundingControl, ComboBoundingSupport, and ComboBoundingReceive. The dynamic relationship of these Sub-Contexts, the Transition Conditions and how they fit into the Context Topology and FSM, is discussed in the following sub-section. But, first, the static structure of the ComboContexts is discussed here.

![Figure 5.5 Three Types of ComboContexts and Their Sub-Contexts](image-url)
The term Context Element is defined here as the components which comprise a Context, i.e. Transition Requirements, Context Topology, High-level Behaviors, associated Sub-Contexts, and Context Goal (Mission Sub-Goal). Tables 5.1 through 5.3 highlight the Context Elements for ComboDiagonalContext, ComboBoxContext, and ComboBoundingContext respectively.

The ComboDiagonalContext, highlighted in Table 5.1, models the team diagonal combination play shown in Figure 5.2. As the illustration shows, two players collaborate by running downfield parallel to one another and passing the ball back and forth in what becomes a diagonal pattern. The goal is to advance the team’s ball position downfield, maintain possession, and shoot the ball at the opponent’s goal if it is possible to score.
Table 5.1 Diagonal Combination Play Context

<table>
<thead>
<tr>
<th>Context:</th>
<th>ComboDiagonalContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans Req:</td>
<td>Teammate in appropriate region for ComboDiagonalContext as illustrated in figure 5.7</td>
</tr>
<tr>
<td>High-Level Beh:</td>
<td>Team play coordinated as illustrated in Figure 5.2</td>
</tr>
<tr>
<td>Sub-Contexts:</td>
<td>ComboDiagonalControl, ComboDiagonalSupport, and ComboDiagonalReceive</td>
</tr>
</tbody>
</table>

The ComboBoxContext, highlighted in Table 5.2, models the team box combination play shown in Figure 5.3. As the illustration shows, two players collaborate by running a criss-cross pattern while passing the ball to each other in a pattern that alternates ball travel from parallel and perpendicular to the length of the field. The goal is to advance the team’s ball position downfield, maintain possession, and shoot the ball at the opponent’s goal if it is possible to score.
<table>
<thead>
<tr>
<th>Context:</th>
<th>ComboBoxContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans Req:</td>
<td>Teammate in appropriate region for ComboBoxContext as illustrated in figure 5.8</td>
</tr>
<tr>
<td>High-Level Beh:</td>
<td>Team play coordinated as illustrated in Figure 5.3</td>
</tr>
<tr>
<td>Sub-Contexts:</td>
<td>ComboBoxControl, ComboBoxSupport, and ComboBoxReceive.</td>
</tr>
</tbody>
</table>

The ComboBoundingContext, highlighted in Table 5.3, models the team bounding combination play shown in Figure 5.4. As the illustration shows, the ball is passed downfield in a near straight line, while the two players alternate bounding past each other in a leap-frog fashion. The goal is to advance the team’s ball position downfield, maintain possession, and shoot the ball at the opponent’s goal if it is possible to score.
Table 5.3 Two Person Bounding Combination Play Context

<table>
<thead>
<tr>
<th>Context:</th>
<th>ComboBoundingContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans Req:</td>
<td>Teammate in appropriate region for ComboBoundingContext as illustrated in figure 5.9</td>
</tr>
<tr>
<td>High-Level Beh:</td>
<td>Team play coordinated as illustrated in Figure 5.4</td>
</tr>
<tr>
<td>Sub-Contexts:</td>
<td>ComboBoundingControl, ComboBoundingSupport, and ComboBoundingReceive</td>
</tr>
</tbody>
</table>

The Transition Conditions for each of the three ComboContexts are dependent on the regional position of a potential receiving player to the player currently in possession of the ball. Figure 5.6 illustrates the various regions. In the figure, the agent at the center is in possession of the ball. When considering if a ComboContext would be appropriate, the agent first looks for a teammate in the corridor, denoted in the figure as ComboBounding and shaded medium gray, forward and aft. If no teammate is in this region, the agent looks next in the regions denoted in the figure as ComboBox and shaded dark gray. If no teammate exists in this region, the agent will finally look in the region denoted ComboDiagonal and shaded light gray.
If a teammate is in the corridor forward or aft of the agent in possession of the ball, as shown in Figure 5.7, then the agent in possession will enter the ComboBoundingContext and transition to the Sub-Context ComboBoundingControl. This player will then attempt a pass to the identified teammate and send a message to the teammate that the pass is coming. The teammates reaction to this information differs based on the various prototypes. The details of this behavior will be explained in the following section. For now, the focus is on how each of the ComboContexts become Active.
If a teammate is in the ComboBox region relative to the agent in possession of the ball, as illustrated in Figure 5.8, then the agent in possession will enter the ComboBoundingContext and transition to the Sub-Context ComboBoxControl. This player will then attempt a pass to the identified teammate and send a message to the teammate that the pass is coming. As explained for the ComboBoundingContext, the teammates reaction to this information differs based on the various prototypes. Again, the details of this behavior will be explained in the following section. For now, the focus is on how each of the ComboContexts become Active.
If a teammate is in the ComboDiagonal region relative to the agent in possession of the ball, as illustrated in Figure 5.9, then the agent in possession will enter the ComboDiagonalContext and transition to the Sub-Context ComboDiagonalControl. This player will then attempt a pass to the identified teammate and send a message to the teammate that the pass is coming. As explained for the previous ComboContexts and at the risk of being redundant, the teammates reaction to this information differs based on the various prototypes. Again, the details of this behavior will be explained in the following section. For now, the focus is on how each of the ComboContexts become Active.
5.2.3 Context Topology

As explained in the background section for CxBR, there are often times when only certain logical transitions between the currently Active Context and the next Active Context can take place. This is quite analogous to the states of a Finite State Machine (FSM). The Context Topology for a CxBR model maps the allowable transitions between Active Contexts. Transition Requirements, depending on complexity, may or may not be included as part of the Context Topology. For simplicity in integrating the ComboContexts with the Simple Soccer Team, the Sub-Contexts of each ComboContext are modeled as states within the FSM. The exact implementation is included in Appendix B.
Before examining the Context Topology it is worth revisiting the FSM graph of the simple soccer team, first shown in Figure 5.1. This is redisplayed in Figure 5.10. A direct comparison of Figure 5.10 with Figure 5.11 clearly shows the integration of the ComboContexts to the Simple Soccer Team’s FSM.

![State Graph for Simple Soccer Team (revisited)](image)

**Figure 5.10 State Graph for Simple Soccer Team (revisited)**
The box marked ComboContexts in Figure 5.11 is a simplified representation of all three ComboContexts. This box could be replicated and included in the graph, with all the same links, three distinct times. The individual replications would then be labeled: ComboDiagonalContext, ComboBoxContext, and ComboBoundingContext. This expanded representation is illustrated in Figure 5.12. Note that the inbound and outbound transition links have been simplified to reduce diagram clutter.

The actual links of Figure 5.12 can be inferred by referring back to Figure 5.11. There is one input link for each ComboContext. This transition is from KickBall to the ComboControl Sub-Context of each ComboContext. This is explained in greater detail for each ComboContext in the following paragraphs.
When an agent is in possession of the ball and prepared to kick, in other words the agent is in the KickBall state, it will evaluate the ComboContext Transition Conditions and determine if a combo play is appropriate. If a ComboContext is not appropriate, the agent will pass or shoot the ball as determined by the KickBall state, without attempting one of the three collaborative combo plays modeled, combo diagonal, combo box, or combo bounding.

However, if an agent is in the KickBall state and the Transition Conditions for ComboBoundingContext is met, as defined in Section 5.2.2 and illustrated in Figure 5.7, then the agent will enter the ComboBoundingContext by transitioning to the ComboBoundingControl Sub-Context.
If the Transition Condition for ComboBoundingContext is not met then the agent will test the Transition Condition for ComboBoxContext, as defined in Section 5.2.2 and illustrated in Figure 5.8. If this Transition Condition is satisfied, the agent will enter the ComboBoxContext and transition to the ComboBoxControl Sub-Context.

Finally, if the Transition Conditions of the previously mentioned ComboContexts are not met and the Transition Conditions for the ComboDiagonalContext is met, then the agent will transition to the ComboDiagonalControl Sub-Context.

The Context flow is different for the various prototypes. So, although the specific prototypes and their differences are not explained until the following section, they will be introduced here in order to discuss the Context Topology. For now, it is enough to be aware that there are three specific CxBR implementations. They are Prototype I: CxBR; Prototype II: CxBRwJIT; and Prototype III: CCxBR.

The following Sub-Sections 5.2.3.1 – 5.2.3.3 provide pseudo-code descriptions of the behavior and Context/State transitions of the ComboContext Sub-Contexts for each prototype model. The Sub-Contexts are generalized and discussed collectively as the logic is consistent throughout with the exception of how the plays are executed in terms of movement and position. In other words, the pseudo-code for ComboControlContext is descriptive of ComboDiagonalControl, ComboBoxControl, and ComboBoundingControl; the pseudo-code for ComboSupportContext is descriptive of ComboDiagonalSupport, ComboBoxSupport, and ComboBoundingSupport; and
the pseudo-code for ComboReceive is descriptive of ComboDiagonalReceive, ComboBoxReceive, and ComboBoundingReceive.

5.2.3.1 Simplified behavior flow for Prototype III: CCxBR

**ComboControl (Sub-Context)**

If shot at goal possible
   Kick ball at goal (play broken)
If player open for pass that allows the tactical play to be continued
   
   \{  
      Calculate ballTarget  
      Calculate comboSupportSpot  
      Kick ball at ball target spot  
      Send Msg_ReceiveBallComboContext to teammate, including ballTarget  
      Transition to ComboSupportContext  
   \}
If no shot on goal AND cannot continue ComboContext
   Transition to KickBallState ball (play broken)

**ComboSupport (Sub-Context)**

If team loses possession of ball
   Transition to ChaseBallState (play broken)
If not at comboSupportSpot
   Move towards comboSupportSpot
If at comboSupportSpot
   If teammate not in same ComboContext
      Transition to SupportAttackerState (play broken)
   Else wait at comboSupportSpot while teammate is in ComboContext
If Msg_ReceiveBallComboContext is received from teammate
   Transition to ComboReceiveContext

**ComboReceive (Sub-Context)**

If team loses possession of ball
   Transition to ChaseBallState (play broken)
If ball within receiving range
   Pursuit the ball (changes steering behavior not State or Context)
Else if ball not within receiving range
   Move to ballTarget
If `ballTarget` reached
   Turn to face the ball and wait for it to come within receiving range
If ball received
   Transition to `ComboControlContext`

5.2.3.2 *Simplified behavior flow for Prototype II: CxBRwJIT*

**ComboControl (Sub-Context)**

If shot at goal possible
   Kick ball at goal (play broken)
If player open for pass that allows the tactical play to be continued
   {
      Continue `ComboContext`
      Calculate `ballTarget`
      Calculate `comboSupportSpot`
      Kick ball at ball target spot
      Send `Msg_ReceiveBall` to teammate, including `ballTarget`
      Transition to `ComboSupportContext`
   }
If no shot on goal AND cannot continue `ComboContext`
   Transition to `KickBallState` ball (play broken)

**ComboSupport (Sub-Context)**

If team loses possession of ball
   Transition to `ChaseBallState` (play broken)
If not at `comboSupportSpot`
   Move towards `comboSupportSpot`
If at `comboSupportSpot`
   If teammate not in same `ComboContext`
      Transition to `SupportAttackerState` (play broken)
   Else
      Wait at `comboSupportSpot` while teammate is in `ComboContext`
If `Msg_ReceiveBall` is received from teammate
   {
      Infer whether or not a `ComboContext` is suitable
      If `ComboX__XContext` seems appropriate
         Transition to `ComboX__XReceiveContext`
      Else if `ComboContext` does not seem appropriate
         Transition to `ReceiveBallState`
   }
**ComboReceive (Sub-Context)**

If team loses possession of ball
   Transition to *ChaseBallState* (play broken)
If ball within receiving range
   Pursuit the ball (changes steering behavior not State or Context)
Else if ball not within receiving range
   Move to *ballTarget*
   If *ballTarget* reached
      Turn to face the ball and wait for it to come within receiving range
   If ball received
      Transition to *ComboControlContext*  

5.2.3.3 *Simplified behavior flow for Prototype I: CxBR*

**ComboControl (Sub-Context)**

If shot at goal possible
   Kick ball at goal (play broken)
If player open for pass that allows the tactical play to be continued
   
   \{  
   Continue *ComboContext*  
   Calculate *ballTarget*  
   Calculate *comboSupportSpot*  
   Kick ball at ball target spot  
   Send *Msg_ReceiveBall* to teammate, including *ballTarget*  
   Transition to *ComboSupportContext*  
   \}
If no shot on goal AND cannot continue *ComboContext*
   Transition to *KickBallState* ball (play broken)  

**ComboSupport (Sub-Context)**

If team loses possession of ball
   Transition to *ChaseBallState* (play broken)
If not at *comboSupportSpot*
   Move towards *comboSupportSpot*
If at *comboSupportSpot*
   If teammate not in same *ComboContext*
      Transition to *SupportAttackerState* (play broken)
Else
    Wait at comboSupportSpot while teammate is in ComboContext
If Msg ReceiveBall is received from teammate
    Transition to ReceiveBallState

ComboReceive (Sub-Context)

If team loses possession of ball
    Transition to ChaseBallState (play broken)
If ball within receiving range
    Pursuit the ball (changes steering behavior not State or Context)
Else if ball not within receiving range
    Move to ballTarget
If ballTarget reached
    Turn to face the ball and wait for it to come within receiving range
If ball received
    Transition to ComboControlContext

5.3 Prototypes

All prototypes described below share the Contexts, Sub-Contexts, game states, and behaviors described thus far in this chapter. Regarding the prototypes’ distinctions, each will be introduced briefly and then explained in greater detail in their appropriate subsection below.

The simplest of these prototypes, Prototype I: CxBR, consists of five CxBR agents grouped together as a team without any special knowledge regarding teamwork. Next in sophistication is the prototype that attempts to infer intentions of teammates based on their perceivable actions, Prototype II: CxBRwJIT. The final prototype, Prototype III: CCxBR, utilizes Context communication to infer intentions of teammates. Figure 5.12 provides a simplified UML class representation highlighting the similarities and differences of the three prototype teams. All three prototypes’ players have the same Context Set and Mission. However, their difference lies
in how these players reason about teamwork. For the CxBR team there is no reasoning about teammates’ intentions. The CxBRwJIT utilizes the `InferIntentions()` to recognize teammate’s Active Context to attempt to predict collaboration with teammates. Finally the CCxBR team utilizes the `CommunicateContext()` to recognize teammate’s Active Context and infer intentions from those Contexts.

### Table 5.4 Collaborative Transition Logic for All Teams

<table>
<thead>
<tr>
<th>Team Type</th>
<th>Combination Contexts</th>
<th>Collaborative Transition Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Team (Simple Soccer Team)</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>Prototype I: CxBR</td>
<td>ComboDiagonal ComboBox ComboBounding</td>
<td>None. Does not consider teammates intentions.</td>
</tr>
<tr>
<td>Prototype II: CxBRwJIT</td>
<td>ComboDiagonal ComboBox ComboBounding</td>
<td>Context inferencing. Infers teammates intentions based on relative position when the ball is passed.</td>
</tr>
<tr>
<td>Prototype III: CCxBR</td>
<td>ComboDiagonal ComboBox ComboBounding</td>
<td>Context communication. Current ComboContext is explicitly communicated when the ball is passed.</td>
</tr>
</tbody>
</table>
5.3.1 Prototype I: CxBR Team

This particular attempt at modeling a team is little more than a collection of individual autonomous agents grouped together in hope of achieving a common purpose. The soccer tactics and skills provided to this team is the same as the other two prototypes, discussed below, with the following major exceptions: this team has no structured planned method for collaboration; specifically, the agents do not attempt to infer the intentions of teammates through observation of action or Context recognition.

This does not mean the agents lack reasoning or autonomy. They have the same plays and Contexts, described in the previous section, available to them as the other two prototypes. However, they are not provided the ability to reason about teammates intentions, which is provided to agent’s of the other two prototypes. Again, unique to this team’s agents, Contexts are selected without consideration of teammates’ intentions.
5.3.2 Prototype II: CxBRwJIT Team Model

The second prototype CxBR soccer team, Context Based Reasoning with Joint Intention Theory (CxBRwJIT), is capable of collaboration. In fact, the methods used are built on the concepts specified by JIT. Players attempt to infer the intention of teammates throughout the game. This is done without the aid of direct communication between the players. A player wishing to initiate a combination play will infer that a teammate in the proper formation is also performing this play. The teammate will have to infer from its actions, the intent of the initiating player. These regions used in inference are illustrated in Figures 5.7 through 5.9. Given the limited number of plays known to the players, play inference is expected to be fairly successful.

5.3.3 Prototype III: CCxBR Team Model: Explicit Context Communication

The third prototype built implements the CCxBR theory specified in the previous chapter. Based on the premise that knowing a teammate’s currently Active Context allows for an understanding of that player’s intentions and plans, collaboration is handled through Context recognition. For the purpose of this research, when deemed appropriate, players directly communicate their currently Active Contexts, using the say command. Knowing a teammate’s currently Active Context, an agent can be sure whether they are collaborating or not. The difference between the CCxBR prototype discussed here and the other the CxBRwJIT prototype discussed above is the matter of inferring intent based on Context recognition rather than inferring intent based on player and ball actions. Again, for this work, Context recognition is assumed to be flawless. It might be expected that groups which train together to perform as a team exhibit a high likelihood of recognizing other members current context. Often in real world models, this is communicated
directly, such as a quarterback calling plays in a huddle or radio communication between units in a military organization.
6 EVALUATION OF PROTOTYPES

The hypothesis being investigated in this research is:

1) A formalization of collaborative behaviors – derived from Joint Intention Theory – in Context-Based Reasoning will provide an **effective** means of modeling teamwork.

2) Modeling agents that infer intention from explicitly communicating Contexts allows for more **efficient** modeling of collaborative behaviors than modeling agents that infer intention from situational awareness.

Thus far, an approach for modeling teamwork in CxBR has been formalized. In order to provide a testing platform, three prototypes have been designed and implemented. This chapter describes the method for testing the **effectiveness** of the specified formalism against the hypothesis statement and for measuring the **efficiency** of modeling teamwork in CxBR. The testing methodology and the test plan are explained first, followed by a report of the test results.

6.1 Test Method

Researchers have taken numerous approaches to measuring teamwork. Some have attempted to look at high-level patterns of tactical knowledge [Devaney 1998, Intille 1999, & Kaminka 2001]. More innovative techniques include a fractal decomposition of behaviors to predict teamwork [Adibi 2002]. Meanwhile, others have attempted to quantify the component parts of teamwork, such as communication [Tambe 1997]. These are only a few examples of the many varied approaches toward testing teamwork. To date, there is not an agreed upon best method.
Important to the selection of the evaluation method is the scientific foundation for doing so. There are two components to the evaluation method used for determining collaboration, Cognitive Task Analysis and JIT definitions. Additionally, metrics are used for comparing efficiency and effectiveness of the collaborative approaches. Much of the testing method is built on Cognitive Task Analysis. The remaining tests are concerned with the definitions provided by Joint Intention Theory. Volume and complexity metrics are used to determine efficiency.

6.1.1 Cognitive Task Analysis

Cognitive Task Analysis (CTA) is used by psychologists as a means of analyzing knowledge and competency in teams. CTA, which is referred to as shared mental models by Canon-Bowers, et. al. [1993], “must identify, define and describe the cognitive processes and knowledge associated with teamwork process (e.g. communication, coordination, adaptability)… Several researchers have attempted to assess the degree of sharedness in knowledge and have found evidence that suggests a positive relationship between the degree of team knowledge and team performance.” [Blickensderfer et. al. 2000] In other words, team performance is dependent upon team knowledge, including the requisite components of communication, coordination, and adaptability.

Of importance to testing the hypothesis of the research described in this dissertation is to show that the implementation does in fact create a working team. In keeping with the foundation of CTA, the first part of the approach taken towards testing is to look at team performance on a macro scale. Relatively speaking, better team performance is indicative of better team
knowledge. Therefore, with respect to CTA, testing is approached by measuring very observable and quantifiable aspects of team performance. In our application of soccer teamwork, these include:

- Distance of ball movement – measured in Cartesian coordinate units, where the soccer pitch dimensions are 694 units from goal to goal and 344 units from side to side (694x344).
- Time of ball possession – time is measured in cycles, where the simulator is set to run 60 cycles/second.
- Number of goal successes – a measure of each team’s goals scored against the opposing team.

6.1.2 Joint Intention Theory

The CCxBR theorems are based soundly on the Joint Intention Theory definitions to justify collaboration. Theorem 6 of CCxBR defines joint intentions of two CxBR agents as two agents that share the same Context. Collaboration, jointly intending to accomplish a mutual goal, can thus be measured directly within homogeneous CxBR agents by comparing when the two agents share the same Context. Two agents sharing dissimilar or similar Sub-Contexts of the same Major-Context possess the same goal of the shared Major-Context. For example, if Agent A is in the ComboDiagonalControl Context (a Sub-Context of ComboDiagonalContext) and Agent B is in the ComboDiagonalSupport Context (also a Sub-Context of ComboDiagonalContext), then these Agent A and Agent B jointly intend to accomplish the mutual goal of the
ComboDiagonalContext (From Table 5.1, the ComboDiagonalContext Goals: Move ball downfield in coordinated diagonal pattern. Score Goal. Maintain Possession.) Direct measures of collaboration include:

- Number of cycles when two agents share the same tactical context – in the prototype this is indicated by the two agents having Active a Sub-Context of the mutually shared Major-Context, as discussed in the example above.

6.1.3 Volume and Complexity

To compare efficiency of implementing the various collaborative paradigms, volume and complexity metrics are considered. Volume metrics include lines of code (LoC) and token count (the number of words or symbols used in implementation). Complexity will consider the number of unique programming constructs necessary for implementation. To summarize, volume and complexity metrics will compare:

- Token count
- Lines of code
- Object count
- Attribute count
- Method count

6.2 The Test Plan

For testing purposes, three prototypes were built. These are described in Chapter 5, Section 5.3. The three prototypes are:
1. Prototype I: CxBR Team
2. Prototype II: CxBRwJIT Team
3. Prototype III: CCxBR Team

All of these teams’ members possess the same level of skill and are afforded the same tactical ability, with one major exception. The way that each team reasons about teamwork differs greatly. The CxBR Team contains no reasoning regarding teamwork, intentions, or group level planning. However, as previously stated, each team member does possess the same skill set (low-level behaviors), and Context set as the team members of the other two prototypes. The CxBRwJIT team is built on valid Joint Intention Theory principles. The agents attempt to infer the intentions of other team members based primarily on teammate position, as discussed in Section 5.3.2. The CCxBR team is also built on valid Joint Intention Theory principles. However, rather than attempting to infer intentions based on player and ball movement, the CCxBR team members infer intentions based on knowing a player’s currently Active Context.

The fact that all teams share the same Context set and low-level behavior skills and abilities is significant. This allows for a more legitimate comparison of teamwork than would be possible given teams of vastly different skill levels and abilities. To better understand this point, consider an example of the inverse. Create two identical teams, Team A and Team B. Each possesses the same skill set, skill-level and teamwork knowledge. Then, while retaining the same level of teamwork knowledge, impair the skill-level of each member of Team B. In other words, Team B is modified to allow for less ability in ball handling, passing, accuracy, etc. It seems intuitively obvious that Team A should outperform Team B. It should be equally obvious that if the skill-
level and skill sets are equal the team with the greater team knowledge should outperform a team with less team knowledge.

A control team (Simple Soccer Team) will be used for testing purposes and was introduced in Section 5.1.3. This team is provided as part of the Simple Soccer Simulator. The differences between the control team and the prototypes are explained in great detail in Section 5.2.

Testing was conducted by running 30 minute games. Game data was collected. Each of the four teams competed, in a set of ten matches, against one another, as illustrated in the Figure 6.1.

![Figure 6.1 Team Comparisons](image-url)

The data collected for each set of team play is averaged and compared in the following section.

Of the four teams, the CxBRwJIT team and the CCxBR team are the only two capable of jointly intending to accomplishing a mutual goal while being aware of their mutual intentions, which is the JIT definition of collaboration, at the level of tactical play, namely the set of ComboContexts. Therefore, with respect to the volume and complexity metrics, the mechanism for inferring
intentions is only compared for the CxBRwJIT and CCxBR teams. No such mechanism exists, explicitly or implicitly, in the other teams. The other teams arguably do not collaborate at the tactical level. Even in the case of the CxBR team, where agent’s occasionally share the same ComboContext, the sharing of intentions to achieve a common goal is not mutually known.

6.3 Evaluation Data

The raw data collected is included in Appendix A along with an explanation of what each data element and each data set represents. The data also includes the statistically mean for each relevant set of evaluations. This data is analyzed and summarized in the remainder of this chapter.

6.4 Results

The results for CTA and JIT metrics are provided in this section. As stated in Section 6.1, these metrics are:

- Number of goal successes
- Distance of ball movement
- Time of ball possession
- Number of cycles when two agents share the same tactical context

6.4.1 Prototype Evaluation: Average Score Comparisons

Quantitative comparisons of averaged data collected from the simulation for all metrics are reported in the graphs of Figures 6.2 through 6.17. Again, the raw data and the calculated
averages (means) are provided in Appendix A. An exhaustive set of team comparisons is provided in the figures below. As such, there will be four sets of graphs for each comparison. Each graph of the comparison set will represent all teams compared against a specified team. The first set of graphs, discussed next, will illustrate this pattern.

Figures 6.2 through 6.5 are all indicators of average score comparisons. Each figure provides a graphical comparison of all four team’s performance against a specified team. Figure 6.2 shows how each team scored on average against the Base team. Figure 6.3 through 6.5 shows how each team scored on average against CCxBR, CxBRwJIT, and CxBR, respectively.

![Figure 6.2 Average Score Against Base Team](image_url)
As illustrated in Figure 6.2, the CCxBR team on average out scored all other teams when playing against the Base team. With an average score of 36.3 to 28.6, CCxBR vs Base, the CCxBR team scored higher on average than any other team for total points and for margin of victory (36.3-28.6= 7.7). The second-best average scoring team against the Base team is the CxBRwJIT team, with average scores of 32.2 to 28.8, CxBRwJIT vs Base. The CxBR team was outscored on average by the Base team. In all the figures below, the table below each graph provides the actual values indicated by the height of the individual bars in the graphs.

![Figure 6.3 Average Score Against CCXBR Team](image)

The CCxBR team on average out scored each of the other three teams. The best competitor, CxBRwJIT, lost by an average score of 44.2 to 38.7 (CCxBR to CxBRwJIT), as shown in Figure 6.3. Against CCxBR, the CxBR team scored on average slightly above the Base team.
However, as the graph clearly shows, the margin of victory for CCxBR over CxBR is far greater than the margin of victory of CCxBR over the Base. The large difference in this margin of victory can be explained by looking ahead to Figure 6.7 and Figure 6.11. The Base team maintained possession of the ball far longer than the CxBR team. Therefore, the CCxBR team had greater ball possession time available for scoring against the CxBR team than for scoring against the Base team.

A graphical comparison of average scoring against the CxBRwJIT team again shows that CCxBR out scores the other teams on average, as illustrated in Figure 6.4. The slight margin of victory against the Base team is interesting enough to note here. Although not of special interest to the research, it can be concluded from the graphs that the Base team on average provides
better defensive play than the other teams. This is shown in the average scoring graphs through the relatively lower margin of victory.

![Figure 6.5 Average Score Against CxBR Team](image)

<table>
<thead>
<tr>
<th>Team Type</th>
<th>Opponents</th>
<th>CCxBR</th>
<th>CxBRwJIT</th>
<th>CxBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>29.7</td>
<td>49.3</td>
<td>44.3</td>
<td>33.6</td>
</tr>
<tr>
<td>CxBR</td>
<td>28.4</td>
<td>30.3</td>
<td>30.4</td>
<td>33.6</td>
</tr>
</tbody>
</table>

**Figure 6.5 Average Score Against CxBR Team**

It can be seen in each of the graphs in Figure 6.2 through Figure 6.5 that the CCxBR team is the highest scoring team on average and is followed closely by the CxBRwJIT team. **This relatively higher scoring performance by both of these teams is supporting evidence for the hypothesis claim with respect to the effectiveness of teamwork in Context-Based Reasoning using Joint Intention Theory.**
6.4.2 Prototype Evaluations: Average Ball Distance Comparisons

It is obvious from Figures 6.6 through 6.9 that on average the Base team moves the ball a much greater distance than either of the other three prototype teams. This was initially surprising especially given the scoring performance of the CCxBR and CxBRwJIT teams. However, after some consideration, the longer possession time of the Base team makes sense. The tactical plays, ComboContexts, provided to the other three prototypes and not given to the Base team all have a behavioral-goal of moving the ball downfield and shooting at the opponent’s goal. Whereas, the Base team is not as behavioral-goal oriented to move the ball downfield. The Base team’s heuristics are more concerned with finding a safe spot to pass the ball and shooting at the opponent’s goal when the opportunity presents itself.

![Bar chart showing average ball distance against Base Team between different teams](image)

**Figure 6.6 Average Ball Distance Against Base Team**
Figure 6.7 Average Ball Distance Against CCxBR Team

Figure 6.8 Average Ball Distance Against CxBRwJIT Team
6.4.3 Prototype Evaluations: Average Ball Possession Time Comparisons

The discussion above regarding average distance of ball movement by the prototype teams, as illustrated in Figures 6.6 through 6.9, extends to averaged ball possession time by the prototype teams, as illustrated in Figures 6.10 through 6.13. The higher ball possession time by the Base team over the other three prototypes is due to the Base team lacking a behavior-goal to drive the ball towards the opponent’s goal. The Base team’s heuristic for passing the ball does consider relative closeness to the opponent’s goal but the heuristic is more concerned with maintaining ball possession. On the other hand, the very tactics of the ComboContexts is to move the ball downfield. These tactics sacrifice some safety of possession for progress downfield, which
could also account for the Base teams higher ball possession time. More aggressive and higher risk plays by the other three prototypes are sometimes going to result in possession turnover.

Figure 6.10 Average Ball Possession Time Against Base Team
Figure 6.11 Average Ball Possession Time Against CCxBR Team

Figure 6.12 Average Ball Possession Time Against CxBRwJIT Team
6.4.4 Prototype Evaluations: Average Shared ComboContext Analysis

To informally paraphrase JIT, for two agents to be collaborating they must both knowingly be attempting to achieve the same goal. Carried over to the evaluative prototypes, this translates to two agents must knowingly share Active Contexts with the same goal. For the ComboContexts modeled in these prototypes that means that two agents must concurrently have any one of the three ComboContext Sub-Contexts as their Active Context. For example, if Agent A has the ComboDiagonalControl Sub-Context as its Active Context and Agent B has the ComboDiagonalSupport Sub-Context as its Active Context – and both agents are aware of the other’s currently Active Context, then the agents are collaborating (in the ComboDiagonalContext) and mutually attempting to achieve the goal of this Context (move the
ball downfield in a diagonal pattern as illustrated in Chapter 5 and opportunistically attempt to score against the opponent’s goal).

![Graph showing Cycles Collaborating](image)

**Figure 6.14 Average Shared Combo Context Time Against Base Team**

This very definition of collaboration is measured and displayed graphically in Figures 6.14 through 6.17. As with the previous graphs, each of these graphs shows all four prototypes results versus a single team type. Figure 6.14 displays the results of all teams versus the Base team. Figure 6.15 shows the shared ComboContexts of all teams against the CCxBR team. Lastly, Figures 6.16 and 6.17 illustrated the shared ComboContexts of all teams versus CxBRwJIT and CxBR respectively.
There are a couple of noteworthy considerations related to the data displayed in these graphs and the meaning of that data. First, the Base team does include any of the ComboContexts and, therefore, the Base team will sow zero for results in all of these graphs. Second, although the CxBR team does contain these ComboContexts and the teammates can and do concurrently select the ComboContext Sub-Contexts, the CxBR teams’ agents do not have a mechanism for Context or intention recognition. Therefore, even though a CxBR team’s agents may concurrently have the ComboContext Sub-Contexts as their Active Contexts, the CxBR team’s agents are never collaborating towards these goals because they are not mutually aware of each other’s Active Context (and therefore not aware of each other’s intention).
The graphs in Figures 6.14 through 6.17 contain much concise data. Each of the four opposing team types are shown with each of the three ComboContexts depicted in the bar graphs. In Figure 6.14 it is shown that the Base team never enters into any of the ComboContexts, intuitively obvious since the Base team does not include a model of the ComboContexts. It can also be seen that the CCxBR team agents have concurrently Active ComboDiagonalContexts 9206 cycles ComboBoxContexts 182 cycles, and ComboBoundingContexts 40 cycles. The CxBRwJIT team’s agents have concurrently Active ComboDiagonalContexts 8306 cycles, ComboBoxContexts 99 cycles and ComboBoundingContexts 24 cycles. Finally, the CxBR team’s agents have concurrently Active ComboDiagonalContexts 1087 cycles, ComboBoxContexts 3 cycles and ComboBoundingContexts 0 cycles.
As mentioned previously, since the CCxBR team’s agents and the CxBRwJIT team’s agents are mutually aware of each other’s Active Context (and therefore mutually aware of goals and intentions) the numbers given represent the cycles during which these agents were collaborating in these tactical plays. However, the CxBR team’s agents are unaware of each other’s Active Context, goals, and/or intentions. The number for concurrently Active ComboContexts by the CxBR team by definition does not represent cycles of collaboration for this team. The fact that the CxBR agents concurrently decided to attempt those tactical plays were based on situational awareness of player and ball position and not on teammate intention.

Consider the data from Figure 6.10 in conjunction with the data from Figure 6.14. From Figure 6.14 the CCxBR team on average collaborated on the tactical combination plays for a combined total of (9206 + 182 + 40) = 9428 cycles. And, from Figure 6.10, on average the CCxBR team maintained possession of the ball against the Base Team for 35986 cycles. This indicates that the CCxBR team collaborated on the tactical combination plays for just over one fourth (35986 / 9428) the time that they had possession of the ball. This total ratio of time collaborating on the combination tactical plays to total possession time is fairly consistent for both CCxBR and CxBRwJIT across all graphs.
The other three fourths of the time that these two prototype team types are not collaborating towards the combination tactical plays includes time spent on: kick-off, defense, other offensive ball movement, and goalie plays.

6.5 Evaluation Discussion

Before examining the evaluation results as they relate to the hypothesis of this dissertation, it is first worth noting some unexpected discoveries during the prototype evaluations and data analysis. This is followed by a qualitative summary of the evaluation data analysis; a reporting and summary of volume and complexity metrics; and, a revisit of the hypothesis statement for consideration against the data analysis.
6.5.1 Unexpected discoveries

There were several surprising results in the evaluation data for the four prototypes. Unexpectedly, the CxBRwJIT team performed very near as well as the CCxBR team with respect to collaboration and with respect to cognitive task analysis. Additionally, although the CCxBR team is clearly more efficient to model, the CxBRwJIT implementation is closer than expected. Tactical performance by the CCxBR team and the CxBRwJIT team with respect to the combination box and combination bounding plays is surprisingly poor. Given the obvious scoring advantage of both the CCxBR team and the CxBRwJIT team over the Base team it was initially surprising to see that the Base team maintains possession longer and moves the ball further during a game. Each of these discoveries is addressed further in the summary conclusion section of Chapter 8.

6.5.2 Prototype Evaluations: A Qualitative Review

The results for CTA and JIT metrics are qualitatively summarized in Tables 6.1 and 6.2, respectively. Because CCxBR is the implementation of the hypothesis to be tested from Chapter 3, its placement in the tables is highlighted in bold type face. This qualitative summary is abstracted from the data provided and explained in Appendix A.
<table>
<thead>
<tr>
<th></th>
<th>Ranked Highest</th>
<th>Ranked 2(^{nd}) Highest</th>
<th>Ranked 3(^{rd}) Highest</th>
<th>Ranked 4(^{th}) Highest</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average Score Against</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td>Base</td>
</tr>
<tr>
<td>CxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td>Base</td>
</tr>
<tr>
<td>CxBRwJIT</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td>Base</td>
</tr>
<tr>
<td>CCxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td>Base</td>
</tr>
<tr>
<td><strong>Average Distance Against</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
</tr>
<tr>
<td>CxBR</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
</tr>
<tr>
<td>CxBRwJIT</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
</tr>
<tr>
<td>CCxBR</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
</tr>
<tr>
<td><strong>Average Ball Control Time Against</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBR</td>
<td>CxBRwJIT</td>
</tr>
<tr>
<td>CxBR</td>
<td>Base</td>
<td>CxBR</td>
<td>CxBRwJIT</td>
<td>CCxBR</td>
</tr>
<tr>
<td>CxBRwJIT</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBR</td>
<td>CxBRwJIT</td>
</tr>
<tr>
<td>CCxBR</td>
<td>Base</td>
<td>CCxBR</td>
<td>CxBR</td>
<td>CxBRwJIT</td>
</tr>
<tr>
<td></td>
<td>Ranked Highest</td>
<td>Ranked 2\textsuperscript{nd} Highest</td>
<td>Ranked 3\textsuperscript{rd} Highest</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Average Shared Combo</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal Context Against</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CxBRwJIT</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CCxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td><strong>Average Shared Combo Box</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context Against</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CxBRwJIT</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CCxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td><strong>Average Shared Combo Box</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context Against</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CxBR</td>
<td>CxBRwJIT</td>
<td>CCxBR</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CxBRwJIT</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
<tr>
<td>CCxBR</td>
<td>CCxBR</td>
<td>CxBRwJIT</td>
<td>CxBR</td>
<td></td>
</tr>
</tbody>
</table>
6.5.3 Volume and Complexity Metric Results

Quantitative comparisons of volume and complexity metrics are given in Table 6.3. The source code responsible for the intention recognition (through Context Communication versus Context Recognition) is hand counted for both CCxBR and CxBRwJIT. The data is included in the table and summarized in the far right column. The data clearly shows that modeling collaborative behaviors using CCxBR is more efficient than modeling the same behaviors using CxBRwJIT.

<table>
<thead>
<tr>
<th></th>
<th>CCxBR</th>
<th>CxBRwJIT</th>
<th>CxBRwJIT/ CCxBR Ratio</th>
<th>CxBRwJIT Increase above CCxBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token count</td>
<td>41</td>
<td>161</td>
<td>161/41</td>
<td>393%</td>
</tr>
<tr>
<td>Lines of code</td>
<td>25</td>
<td>41</td>
<td>41/25</td>
<td>164%</td>
</tr>
<tr>
<td>Object count</td>
<td>5</td>
<td>9</td>
<td>9/5</td>
<td>180%</td>
</tr>
<tr>
<td>Attribute count</td>
<td>9</td>
<td>17</td>
<td>17/9</td>
<td>189%</td>
</tr>
<tr>
<td>Method count</td>
<td>4</td>
<td>12</td>
<td>12/4</td>
<td>300%</td>
</tr>
</tbody>
</table>

To further investigate the efficiency, three additional testing Contexts were added. The purpose of adding these Contexts is to investigate the change in volume and complexity. The prediction is that adding Contexts to the CCxBR model results in a fairly constant change or linear increase in complexity adding one Context only requires that Context and its communication mechanism to change. In contrast, it is predicted that adding one Context to the CCxBRwJIT model is a
linear change and exponential increase in complexity, as adding one Context requires that
Context to be modified and all existing Contexts to be modified (in order to deal with intention
inference given the lack of intention communication).

The results turns out to be less drastic than the prediction, however. As the prototype models’
ComboContext Transition Requirements are based on spatial regions of the soccer pitch, it is not
very difficult for the CCxBR modeler to distinguish the likely Active Context for based on these
Transition Requirements. The actual results closely follow the results provided in Table 6.3 and
increase linearly for both the CCxBR model and the CxBRwJIT model.
7 SUMMARY/CONCLUSION & FUTURE WORK

7.1 Summary and Conclusion

Joint Intention Theory (JIT) [Cohen and Levesque, 1991], the commonly accepted definition for teamwork, is specified from a Belief, Desire, and Intention (BDI) [Georgeff et. al. 1999] perspective. No contextual reasoning theories [Brézillon 2005, Gonzalez and Ahlers 1998, Stensrud et. al. 2004, Turner 1993] formally specify definitions for teamwork. Additionally, these contextual reasoning paradigms are not explained in terms of the more popular BDI paradigm.

This research addresses these issues using Context-Based Reasoning (CxBR) [Stensrud et. al. 2004] and JIT to:

1. formalize the relationship between CxBR and BDI
2. formalize collaborative behaviors in CxBR
3. propose an efficient and effective method for implementing collaborative behaviors in CxBR

A formalization of the relationship between CxBR and BDI is provided in Chapter 4. This chapter also contains the formalization of collaborative behaviors, based on JIT definitions, within CxBR. An alternate justification for the formalism of collaborative behaviors in CxBR based on JIT is provided in Chapter 7.
Two approaches to modeling teamwork in CxBR are prototyped, as described in Chapter 5. The first approach infers the intentions of a teammate based on that teammate’s actions. The second and more novel approach infers intentions based on the teammates currently Active Context, which is communicated directly.

7.1.1 Hypothesis Evaluation

The two-part hypothesis originally stated in Chapter 3 is:

1) A formalization of collaborative behaviors—derived from Joint Intention Theory—in Context-Based Reasoning will provide an effective means of modeling teamwork.

2) Modeling CxBR agents that infer intention from explicitly communicating Contexts allows for more efficient modeling of collaborative behaviors than modeling agents that infer intention from situational awareness.

Evaluation of effectiveness and efficiency of the models and their respective implementation efforts are each considered in the following two subsections.

7.1.1.1 Effectiveness of Collaborative Models

In this dissertation, cognitive task analysis and Joint Intention Theory definitions provide the metrics for evaluating effectiveness of collaborative models. CTA is employed by comparing team performance characteristics such as average scoring, average ball movement, average ball possession time. JIT definitions provide a more concrete indication of collaboration in comparing directly when agents share the same intentions and goals by examining when they share the same Active Contexts (or set of Sub-Contexts).
The data analysis provided in Section 6.4 shows: CCxBR teams on average outscore all other prototype teams; teams utilizing ComboContexts to implement tactical combination plays actually possess the ball less due to more efficient offensive capabilities; and CCxBR agents and CxBRwJIT agents share the same ComboContexts approximately 25% of the time they possess the ball. Each of these is an indicator of effective collaboration using CCxBR and CxBRwJIT, both of which rely on the JIT adaptation for Context-Based Reasoning.

7.1.1.2 Efficiency in Modeling

Software Engineers often use Volume and Complexity metrics for comparing relative difficulty of implementation between applications. Volume metrics show the amount of code generated. Complexity metrics show the number of interrelated things a developer must be concerned with while during implementation.

The results of the volume and complexity metrics reported and summarized in Table 6.3 clearly show that using explicit Context communication for intention recognition in the CCxBR prototype is a more efficient implementation method for modeling collaborative behaviors in Context-Based Reasoning than using intention recognition through inference in the CxBRwJIT prototype.

7.1.1.3 Evaluation Conclusion

The test results reported in Chapter 6 clearly show:

1. The formalization of collaborative behaviors within CxBR results in effective teamwork.
2. The context recognition method, CCxBR, of inferring intentions based on an agent’s currently Active Context is an efficient method of modeling teamwork in CxBR.
7.2 Discussion

In Chapter 6 several notable surprises were mentioned including:

1. The CxBRwJIT team performed very near as well as the CCxBR team with respect to collaboration and with respect to cognitive task analysis.

2. Although the CCxBR team is clearly more efficient to model, the CxBRwJIT implementation is closer than expected.

3. Tactical performance by the CCxBR team and the CxBRwJIT team with respect to the combination box and combination bounding plays is surprisingly poor.

4. Given the obvious scoring advantage of both the CCxBR team and the CxBRwJIT team over the Base team it was initially surprising to see that the Base team maintains possession longer and moves the ball further during a game.

5. On average the Base team out scores the CxBR team by a narrow margin, 29.7 to 28.4.

The close performance characteristics of CxBRwJIT relative to CCxBR are justifiable, as both of the prototypes are built using the formalism derived in Chapter 4. The difference between the two prototypes is that the CCxBR prototype agents infer intentions through explicit communication of Contexts between agents while the CxBRwJIT prototype agents infer intentions based on situational awareness. It was expected that intention inference based on situational awareness would lead to less effective collaboration because of occasional miscalculations regarding intention. The fact that there are a limited number of tactical plays modeled as ComboContexts probably accounts in part for the closeness in performance between the two prototypes. Having only three tactical plays in which to reason about and having very
distinct characteristics for selecting those plays makes inference about which play a teammate is committed to less arduous.

The limited number of tactical plays that are modeled also offers an explanation to the second point in the list at the start of this sub-section. If the model were to scale up and the number of tactical plays being modeled increased, then the difficulty of addressing this for the CCxBR model would be constant and relatively simple. Each additional play modeled as a ComboContext in the CCxBR prototype would require adding the play name to the message system and would require adding only a few of lines of code for sending and receiving the communications of this particular play. However, for each additional play modeled as a ComboContext in the CxBRwJIT prototype, reasoning about other players’ commitment to execute that play would have to be explicitly implemented and reasoning about all other plays already modeled would have to be reviewed to ensure that those plays are still distinguishable from the newly modeled play. This difficulty in implementation is linear in complexity: every time a new play is modeled, reasoning about all existing plays must be modified and reasoning about the new play must be implemented. This does not scale well.

An inspection of the evaluation data in Chapter 6 shows that the ComboDiagonalContext is very effective and heavily used by the prototypes. However, the ComboBoxContext, and specially so the ComboBoundingContext, are much less effective and much less used. Observation of game play during development and testing indicates that this is likely a result of how the Simple Soccer Team (used as the Base team for evaluation and used as the foundation for which all other team prototypes were built) plays. The heuristics of the Simple Soccer Team tend to cause the players to be in positions towards the center of the field or between two attackers which foils
the attackers’ ability to execute the combination box or combination bounding play. This results in the attackers: initially choosing another play (combination diagonal) over either of these two, aborting this play once chosen, and losing possession of the ball after attempting this play.

The fourth point mentioned in the list is with respect to ball possession time and ball movement. The evaluation data shows that the Base team controls the ball about 20% longer than the other prototypes and moves the ball about 30% farther than the other prototypes, despite the fact that the other teams all outscore the Base team. This initially seems strikingly odd. However, considering the how the Base team is extended to create the other prototypes this is explainable. The implementation of each of the other prototypes includes adding three tactical combination plays to the Base team. (How these prototypes agents select these combination plays based on teammate activity is what distinguishes each of these three for one another.) Each of the three combination plays are primarily offensive and are concerned with efficiently moving the ball downfield and attempting a shot at the opponent’s goal. This is in contrast to the Base team which also seeks to move the ball downfield but in a much less tactically coordinate way. The Base team heuristically looks for open positions to where players can pass the ball. Downfield is favored but, lacking coordinated tactics, finding an open positions is largely reactive to opponent position. This method of moving the ball often results in three or four players passing the ball around (from side to center-downfield to side to center-upfield to side) without making much real progress in advancing towards the opponents goal. On the other hand, the prototypes that included the combination tactical plays quickly advance the ball towards the opponent’s goal and shoot, thereby scoring or losing possession.
Perhaps each of these four points and especially the last two points are all affected by tuning the model. For example, the prototype developer could tune the behavior of the ComboBoxContext to make it more effective against the heuristics employed by the Base team or the criteria (Transition Requirements) for selecting the ComboBoxContent could be modified to cause the play to be selected more or less often. This may make for a better simulated soccer team and may make for better play against the Base team. However, it likely would not provide any better data for this research. It could skew the results some by making ComboBoxContext more utilized and probably reduce the utilization of ComboDiagonalContext. This does not really change the evaluation of effectiveness in collaboration. It simply changes the frequency of which each ComboContext is selected for collaborating.

The fifth point listed, the Base team’s average higher score over the CxBR team was initially surprising but is now justifiable. In addition to the heuristics inherited from the Base team, the CxBR team has the additional ComboContexts. However, unlike the CCxBR and the CxBRwJIT teams, the CxBR team is not given an effective mechanism for collaboratively utilizing these plays (ComboContexts). As such, when a CxBR agent enters a ComboContext, it is much more likely to result in a broken play (than resulting in a combination play as intended by the agent entering the ComboContext) because the agent’s teammate is unlikely to chose this ComboContext also. The teammate is more likely (as shown by results regarding shared ComboContexts) to enter a different ComboContext or run one of the inherited (from the Base team) heuristic tactics. Thus the intentionally missing mechanism for collaborating with respect to ComboContext causes slightly worse performance by the CxBR agents than if they were not
provided the additional tactical plays (since in reality they were only really provided an awareness of these plays without being provided a mechanism for running them effectively).

There is an anomaly in the qualitative data reported in the Prototype Evaluation section. A closer look at the data however shows that this anomaly exists more in the representation than in the performance. The table is concerned with relative ranking of all teams versus all other teams taken individually. The CxBR team appears to out perform (in relative ranking) the CxBRwJIT team in both ball distance moved and ball possession time. Consider, however, that both of these measures (distance and time) for two opposing team represent each teams portion of the whole. To understand this, consider the time comparison.

As Figure 6.13 shows, CxBRwJIT possesses the ball for 44,624 cycles versus the CxBR team where as CxBR possesses the ball for 45,168 versus the CxBR team (playing itself). This indicates, as the qualitative ranking in Section 6.5.2 shows, that CxBR possesses the ball a greater time than CxBRwJIT versus CxBR. However, if these times are viewed as a ratio of possession versus CxBR then, CxBRwJIT possesses the ball 44,624cycles/43,610 cycles or 1.023 times as long as the CxBR team and CxBR possesses the ball 45,168 cycles/ 45,168 cycles or 1.000 times as long as the CxBR team. Thus the CxBRwJIT team actually controls the ball a greater portion of the total time than the CxBR team when they are both played against the CxBR team. This same analysis holds for the ball distance movement and the anomaly in the ranking data reported.
The following is an outline of suggested future work related to this research and CxBR.

7.3 Future Work

- Compare CCxBR to SharedPlans. Show that the premise upon which SharedPlans is built, that individual and group plans must be considered in a formalization of collaborative behaviors, is met through using Active Contexts for plan recognition. Compare modeling with CCxBR to modeling with SharedPlans: the simplicity of building models with CCxBR should far exceed that of SharedPlans.

- Investigate similarities between Context topologies and the hierarchy of reactive plans used by STEAM. It may be possible to utilize STEAM to implement teamwork models in CxBR, if the Context topology can be converted to a hierarchy of reactive plans.

- Develop a diagramming language and tool for modeling behavior in CxBR. There is a particular need for tools that clearly represent multi-agent interactions.

- Run-time Reconfigurable Contexts

- CxBR-Model-Specification Parser/ Interpretive Framework

7.4 Run-time Reconfigurable Contexts

The first generation CxBR Framework provides a means of implementing Contexts as OO Classes that can be instantiated for use as objects. All of the applicable knowledge for the Context is included in the Class, during development time. There are shortcomings to this approach. One such shortcoming is relevant to learning new Contexts during run-time. As it is
now, in order to change the knowledge contained in a Context, the Class must be modified and recompiled off-line. This has the additional disadvantage of requiring a developer in the loop for the system to learn additional Contexts.

An alternate approach that would overcome the necessity for recompiling is to store the Context knowledge in a persistent data structure, such as a text file, for run-time retrieval. To prevent inefficiencies due to reading from a file on the hard drive versus reading from system memory, once read from the file, the Context knowledge could be stored in an object.

The contextual data should include, at a minimum, information necessary for transition logic and information for high-level agent behavior resulting from the Context. The transitional logic could be in the form of antecedents necessary for the selection of the associated Context as an Active Context. For some other form of transition logic besides simple conditional rules, such as a neural network employed for the sake of recognizing the Active Context, the data included in the contextual data file would have to be configured in a manner demanded by the reasoning mechanism. Using a neural network as an example, the transition logic contained in the contextual data file could consist of a vector containing node topology and weights for each node.

Likewise, the high-level agent behavior determined by a Context must take an appropriate form relative to the reasoning mechanism employed for this purpose. For a rule-based mechanism the contextual data should include antecedents and associated consequents. Thus, once the Context
is active and the appropriate antecedents are met the rule will fire and the agent should behave as dictated by the consequent.

### 7.5 CxBR-Model-Specification Parser/Interpretive Framework

Extending the above idea of dynamically reading contextual data to build Contexts during runtime, it is possible that the framework could create an entire model in this same manner. In other words, CxBR models could be built dynamically from a specification. The framework should include a parser that reads a text file specifying the model. From this specification an entire CxBR model can be built by creating the appropriate objects for Mission, Contexts, Sub-Contexts, Context Moderators; and configuring the appropriate reasoning mechanisms for transition logic and high-level behaviors.
APPENDIX A. COLLATERAL WORK
There is a necessity to extend Context-based Reasoning (CxBR) to incorporate new concepts pertinent to modeling collaborative behaviors. To this end, future research should further investigate Co-contexts and Embedded Contexts to the CxBR paradigm. Co-contexts and Embedded Contexts are introduced in this chapter. In order to discuss these new concepts a review of the legacy CxBR Framework is offered first followed by a brief discussion of modeling affect in CxBR. While the work to date on modeling affect in CxBR is not yet concerned with its effect on teamwork, the representation and implementation methods employed is directly pertinent to the discussion of Co-contexts. In fact, Co-contexts themselves are considered a form of Context Moderator as emotion is in the affect model. Other topics presented in this chapter include the Social Construct Class and Implied Missions. The description of the legacy framework is also pertinent to the discussion of the Social Construct Class.

A.1 Background: The Approach Taken to Model Affect in CxBR

The traditional CxBR Framework includes four abstract classes, as illustrated in Figure A.1. These create an application program interface to facilitate modeling human behaviors. The AIP class provides an interface for modeling agents. The Mission and Context classes provide the means for an agent to act, react, and plan within a given scenario. The Inference Engine is used for pattern matching facts from both a local and a global fact base.
Figure A.1 Class Structure of Legacy CxBR Framework illustrates inherited classes used for modeling i.e. agent[i], context[j], and mission[k] Reprinted with permission. (Barrett et. al. 2003) (repeat of Figure 1.3)

Agents are assigned a mission, which defines the high level goal the agent is expected to pursue for a given scenario. The mission also specifies the contexts that will be applicable to the agent. Contexts provide the agent’s planning and reaction abilities and specify any other possible contexts that can be reached from the currently active context. The relationship between contexts, missions, and agents is illustrated in Figure A.2.
The purpose of enhancing the legacy CxBR Framework is to provide a feasible means of implementing agents capable of affective reasoning. A comparison of Figure A.1 and Figure A.3 illustrates the additional abstract classes added to the legacy framework. The Enhanced CxBR Framework includes Mood and Emotion. The relationship between Mood and Emotion within CxBR parallels that of Mission and Context. In the same way that a mission specifies all possible contexts the agent can experience, a mood specifies all possible emotions that an agent may experience. This design follows from the fact that humans’ emotions are quite dependent upon their mood, i.e. an individual in a depressed mood is not likely to experience the emotion of joy. [Lisetti and Bianchi, 2002]
Figure A.3 Class Structure of Affect Enhanced CxBR Framework illustrates inherited classes used for modeling i.e. mood[i], emotion[j], agent[k], etc.

Adding affect to agents modeled in CxBR means the agent must be assigned a mood (or set of possible moods depending on the intended scenario) and assigned a set of emotions possible for that mood. This is analogous to the assignment of a Mission and its associated Context set. Figure A.4 diagrams the relationship between an agent and its associated objects of mission, context, mood, and emotion. Although not included in the diagram, personality traits are defined in the agent object itself. The personality traits are specified in the Mood. Personality effects how an agent reacts given a specific Active Mood and Active Emotion. In other words, an agent modeled to represent emotional maturity will not react in the same manner as an agent modeled to represent psychosis. The exact manner in which these agents react based on their mood, emotion, and personality is a matter better left for the psychologists. The fidelity of this representation in a CxBR model is an area for future research. The important issue here is that the mechanism is designed into the CxBR framework to provide a means of modeling these characteristics. Context transition is influenced by the corresponding mood and emotion of an agent.
Figure A.4 Object Relationship for a Possible Implementation of an Affective Agent
Reprinted with permission. (Barrett et. al. 2003)

The incorporation of modeling affect in CxBR is mentioned in this section primarily because of its analogous nature to Co-Contexts. Co-Contexts, which will be discussed following the next section on Embedded Context, represent another form of Context Moderator. Emotion is the first successful attempt at incorporating Context Moderators into CxBR. [Barrett et. al 2003, and Stensrud et. al. 2004]

A.2 Embedded Contexts

Embedded Contexts result from the contextual knowledge representation of entities within a hierarchical chain of command. This embedding of contexts best serves the purpose of entity
control during simulation run-time. The hierarchical nature of Embedded Contexts is discussed in the following subsection, A.2.1. The sequencing of entity control within hierarchical teams is discussed in the subsequent subsection, A.2.2.

A.2.1 Knowledge Representation

Embedded Contexts represent the knowledge of entities within a hierarchical superior/subordinate structure. The Embedded Contexts are the Contexts of these entities throughout this hierarchy. The concept may best be explained through example. The business example introduced in section 4.3 is expanded here. The business is composed of a set of collaborating entities. These entities may themselves be teams of other entities, such as regions, divisions, departments, etc., or these entities may be atomic entities such as managers, salespersons, laborers, etc. Obviously, the business could be composed of a combination of team and atomic entities. The formation of team entities could be recursive. In other words, a team composed of teams of teams. For example, a business may have regions, which are comprised of divisions, which are comprised of individual outlets, which are comprised of individual employees. The importance of the hierarchical structure to the current topic lies in the representation of tactical knowledge throughout the hierarchy.
Consider an appliance-company with two divisions, sales and service. The two divisions are overseen by a general manager/owner. The sales division has five employees, a sales manager, two salespersons, and two delivery persons. The service department also has five employees, a service manager, a receptionist/dispatcher/secretary, and three technicians. This is represented as an object diagram in Figure A.5. Note that each object, autonomous entity, modeled contains its own Mission and Context set. Abstract entities such as the business itself or the two divisions each contain their respective Mission and Context set. An example of such a Mission for the business is the mission statement for the company. This could be something along the lines of, “provide quality and affordable appliances and reliable appliance repair to the community, while earning a profit.” Likewise, the service division may contain a Mission such as, “provide efficient appliance repair to sales customers as well as non-sales customers, by factory certified
technicians, for all major brands of appliances, while maintaining a profit.” For the sake of conciseness, accept that each of these respective Missions will have a set of associated Contexts, which, along with other factors, might be based on factors such as economy, customer base size, and increases or decreases in these factors.

![Sequence diagram illustrating embedded nature of Context execution](image)

**Figure A.6 Sequence diagram illustrating embedded nature of Context execution**

A.2.2 Implementation

The hierarchical structure of the team itself and the particular schema used to represent knowledge within CxBR allow for easily understandable, scalable models of teams. The execution of tactical behaviors by a CxBR agent is based largely on the currently Active Context. These details of CxBR agents’ behavior provide a means for scalability and verifiability by embedding the sequencing of team Contexts. Again, this may best be illustrated by example.
Continuing with the business example, consider the sequence diagram in Figure 7.6. When the Appliance Company is actively simulating some tactical behavior, it is necessary to consider the tactical behavior of the company’s component members, as the company itself is simply an abstraction composed of other abstract or atomic entities. Thus, during run time execution over some given time period, the currently Active Context of the Appliance Company sequences through the currently Active Contexts of each of its component members: the Service Division, the Sales Division, and the General Manager. This sequencing through Active Contexts of component members continues recursively for all abstract entities. In this example, as illustrated in Figure 7.6, the tactical behavior of the Service Division sequences through each of the divisions component members’ Active Contexts. Since the Service Division is composed of all atomic entities, the division’s behavior emerges as a result of each of its component member’s behaviors. As all atomic entities comprising the abstract entities continue to behave according to their Active Contexts, the behavior of the abstract entities continue to emerge. The behavior of the Sales Division, thus, emerges from the behavior of its component entities, the Sales Manager, the Sales Persons, and the Delivery Persons. Thus, the Appliance Company behaves according to the emergent behavior of its Service Division, Sales Division, and General Manager.

Again, the Contexts within CxBR primarily determine the tactical behavior of an agent. Given this, it seems intuitive to allow the Context of the abstract entity to consider the behavior of its component entities in determining tactical behavior.

Two unrelated points are worth mentioning here. First, the atomic level entities could also be abstract entities. This could occur in cases where the simulation does not require modeling to the
smallest possible real-world component. For this purpose, the lowest level entity representation would attempt to provide suitable behavioral responses, without relying on the emergence of lower level entities behaviors. The second noteworthy point is in regard to the philosophical nature of embedding Contexts. The sequencing through and possible influencing of component entities’ Active Contexts by a higher-level abstract entity’s Context could have sociological justifications and parallels. This could be analogous to the Social Contract discussed by philosophers. [Cortina, 2003] The first of these two points is considered self-evident, while the latter is presented as a topic for future research.

A.2.3 A General Hierarchical Teamwork Model for CxBR

Correct design and implementation of CxBR Missions, Major Contexts, and Sub Contexts are critical in properly representing collaborative behaviors between agents. Teamwork between entities is then accomplished by appropriately assigning these Missions, and their associated Contexts to these entities. A superior to subordinate relationship is established by allowing the superior to perform these assignments dynamically for any entities under the superior’s command. Alternatively, in the absence of a modeled chain of command, Missions and Contexts are assigned in the traditional CxBR manner of hard coding the Mission and allowing an agent to choose the active Major Context based on their sentinel rules. Regardless of which method is used in choosing the active Major Context, by order or by autonomy, the functionality for collaborative behaviors must be included in these Contexts. Contexts in CxBR control entity behaviors by executing the rules and functions contained in an Active Context related to a specific situation. Collaborative behaviors, therefore, are facilitated by designing and implementing rules that provide a teamwork aspect between entities. In other words, an entity’s actions are influenced by other members of its team through the rules and functions specified by
the entity’s active Context. The following section details an example consisting of a platoon of tanks. Through appropriately designed rules and functions, each tank is necessarily concerned with the current action and status of its platoon mates. Thus, action planning may include constraints based on the platoon mates’ circumstances.

A general design for implementing collaborative behaviors in CxBR is illustrated in Figure A.7. As shown, agents are assigned at least one Mission Context and the necessary Major Contexts and Sub Contexts to carry out those Missions. The association between supervisor and subordinate is controlled through the Contexts assigned to these agents. Specifically, the high-level Major Contexts contain the necessary logic for managing the mid-level Supervisors and possibly the workers. The details of how high-level Supervisors manage mid-level Supervisors is, of course, model dependant. Additionally, the type of logic employed by the high-level Major Contexts is implementation dependant. For example, this logic could be captured in a neural network, or it could be captured in the more traditional CxBR approach of conditional rules utilizing the ExecuteScripts method of the framework. Regardless of the implementation chosen, the effect is to provide high-level Supervisors authority over mid-level Supervisors and workers.
Figure A.7 UML Object Representation of a Teamwork Model

The high-levelSupervisor directs the mid-levelSupervisor by communicating which mid-levelMission and/or which mid-levelMajorContext to make active. The effect of this assignment is that the high-levelSupervisor has thus directed the mid-levelSupervisor as to which actions should be performed since the mid-levelSupervisors’ (as with all agents) immediate actions are determined by the rules and functions specified in the Active Context. When orders are issued, a flag is set indicating such. The subordinates will monitor this flag when considering which Context to make active at each time step. The flag is thus used to assign a precedence of orders over what the subordinate may choose to do alternatively. Of course, the subordinate could be provided with the ability to disregard orders under certain conditions. The supervisor-to-subordinate interaction is recursive throughout the chain of command. Typically, as in most
chain of commands, supervisors are concerned with communicating with the subordinates directly under them. However, this could be modeled differently if applicable.

A.3 Co-Contexts

Co-Contexts are a specific type of Context Moderator. Context Moderators are specified for CxBR in Stensrud et al. [2004] and first implemented in the form of affect, mood, and emotion as discussed in Barrett et al. [2003]. The purpose of a Context Moderator is to provide a means of influencing the tactical behavior of an agent by influencing the agents Active Context. Thus, the Active Context still ultimately determines the agent’s tactical behavior, but the Active Context can be severely affected by an Active Moderator. Context Moderators within CxBR are characteristic of Contexts in the sense that, like Contexts, an Active Moderator is selected, according to some specified transition criteria, from a set of applicable Moderators. A Co-Context could, to a limited extent, be considered a second, concurrently active Active-Context. This is not completely accurate, but the simplicity should allow for an easier understanding of the purpose and function of Co-Contexts.

Specifically, Co-Contexts exist to provide better modularity and reusability of knowledge which would otherwise be captured strictly through Contexts. Consider the Sales Manager from the Appliance Company example above. Much of the Sales Managers tasks and knowledge are likely similar to the Sales Persons of this model. However, in addition to the situations (contexts) that a Sales Person may encounter, the Sales Manager has supervisory and managerial
situations to be concerned with as well. The parallel structure between Co-Contexts and Contexts is illustrated in Figure A.8.

It is possible that the knowledge and its associated behavior captured in Co-Contexts could be handled through Contexts alone. However, consider even a small model that provides six Contexts for a Sales Person, and assume that managerial knowledge could, also, be captured in six Contexts. In order to model the above Sales Manager who performs both roles, using only Contexts, there would be a minimum of 6 x 6 or 36 Contexts necessary to capture all possible combinations of sales Contexts and managerial Contexts. Thus, Co-Contexts not only allow for better modularity and reuse of Contexts but Co-Contexts also reduce the model complexity, which is of course true for Context Moderators in general.
A.4 Implied Missions

The current specification for CxBR requires an explicitly specified Mission. The typical approach to modeling knowledge for a single scenario in CxBR is through the use of a Mission, a required set of Major Contexts, and a possible set of Sub Contexts. The Mission’s primary functions are: to explicitly specify a high-level goal (high-level relative to the Mission’s Contexts), to explicitly specify the Contexts to be expected/included as part of the tactical mission, and to explicitly specify the Context topology, which defines transitional Contexts from any given Active Context. It is possible that this functionality could be captured, particularly for small scale missions, without being explicitly specified. An alternate approach, hereby termed Implied Missions, is to capture the above functionality within the Contexts themselves.

Perhaps this is best illustrated through the use of an example. Consider modeling a soccer player. Specifically, our example will model an offensive team member capable of executing numerous plays. These plays are designed to quickly move the ball down field past defenders and to allow an attack on the defender’s goal at opportune times. The number of plays considered here will be limited to facilitate our explanation. However, it should be obvious that this could scale to a larger number of plays.

A set of basic plays that could be used to model offensive players would be:

- Kick-off
- Attack goal
- Combination play
- Lost ball possession
Recover ball possession

Each of these plays could be modeled as a Major Context.

Two plays are illustrated in Figure A.9 and Figure A.10. These two combination plays are each designed to allow two players to quickly move the ball upfield, past defending players, into better scoring position. Combination play 1 is executed by two players running along parallel lines. The ball is passed diagonally between the players as their forward motion leap frogs one another’s. As illustrated in Figure A.10, combination play 2 moves the ball upfield by passing forward and laterally as the players run a diagonal pattern. Both plays serve the same purpose of moving the ball upfield to provide a better position for scoring possibilities.
Figure A.9 Combination Play 1: Dashed lines indicate player movement. Solid lines indicate ball movement.

Figure A.10 Combination Play 2: Dashed lines indicate player movement. Solid lines indicate ball movement.
Each of the basic plays listed above could be considered a separate Context. Considering a limited number of each type of play, a Context topology is illustrated in Figure A.11.

Consider the basic logic or reasoning involved in either of the two combination plays:

IF the players are in a field position that requires moving the ball a long distance AND IF player two is moving in the correct direction to execute the play
AND IF there are no defenders present to intercept the ball pass
THEN player one should pass the ball
AND player one should run to next relative field position to receive the anticipated return ball pass from player two.

The overall team goal of winning the game by scoring more points than the other team is not explicitly stated. The mission-goal of scoring more points than the other team is implicit. The
Contexts and their associated topology allow an offensive player to score points without the Mission-Context specifying this as the high-level goal. Contexts can be assigned to an agent directly rather than assignment through a Mission Context, and the Context topology is easily captured within the Contexts themselves, by specifying possible Contexts transitions for each Context. From an implementation point of view, the Mission-Context is not necessarily required in order to build a functioning model. In fact, it is not evident that explicitly implementing a Mission could or would enhance the implementation of this model.

A.5 Extending Joint Intention Theory

A formalism for collaborative behaviors in CxBR based on JIT is provided in Chapter 4 of this dissertation. This section presents an alternate justification for collaborative behaviors in CxBR based on JIT. It is placed here in the related work chapter to allow the principles of Chapter 5, which stand on their own merit, to be explained more concisely and more clearly.

This second approach, considers collaborative behaviors starting from JIT and moving towards CxBR. This additional approach could be considered redundant, much the same as providing a top-down design of a system after the system has already been designed from a bottom-up perspective. Nonetheless, the approach is still described here for the sake of thoroughness and to provide another perspective towards the concepts presented. The reader is cautioned not to get confused in the overlap. The first three sections of this chapter provide all of the necessary details to justify and specify collaboration in CxBR. The fifth section is, again, only added for the sake of redundancy. (While considering the inclusion of this additional perspective, the many proofs of Pythagorean’s Theorem come to mind. There exist at least 46 distinct proofs, yet
this does not diminish the possibility of more clearly proving the theorem with a 47th proof. Nor does it diminish the validity of the 47th proof.)

In this section, the definitions of JIT are specified and represented as Contexts. These definitions will be considered in the highest form of abstraction. As such, much of what would be model-dependent characteristics of the Contexts is left unaccounted for, while exclusive focus is given to JIT constructs.

The ideas presented in this section require a basic understanding of knowledge representation in CxBR and a basic understanding of the JIT definitions. Both will be presented briefly here before proposing the new ideas. First, consider the knowledge contained in a CxBR Context. Transition criteria represent the sufficient conditions for the Context to become the current Active Context, (i.e. the Context that currently controls the agent’s behavior. Transition criteria is a set of state data representing factors both internal and external to the agent. Once the Context is active, the agent’s high level behavior is controlled by reasoning towards accomplishment of the Mission’s sub-goals. Finally, any Context that could be the next Active Context, is included as part of the Context Topology.

Consider the definitions of JIT quoted in Chapter 1. Analysis of these shows they all contain a common set of elements. In particular, each specifies a state (or context) for an agent or team of agents; each specifies a set of conditions associated with that state; and each specifies a goal associated with that state.
Speaking of contexts in general, a context describes a state. A particular state or context is differentiated from any other state or context by some set of elements. As a simplified example, water falling from the sky usually indicates that it is raining. The definition of rain is: “Water condensed from atmospheric vapor and falling in drops.” [American Heritage Dictionary 2000] Relying on this definition and applying what is known about the perspective of contexts, rain can be described as a context that exists when the condition of “Water condensed from atmospheric vapor and falling in drops” is satisfied. Extending this perspective towards contextual states based on the conditions of a definition, the JIT definitions can be described as contextual states as follows:

Consider the definition for a persistent goal.

An agent has a **persistent goal** relative to q to achieve p iff:
1. The agent believes that p is currently false
2. The agent wants p to be true eventually
3. It is true (and the agent knows it) that (2) will continue to hold until the agent comes to believe either that p is true, or that it will never be true, or that q is false.

The state described is an agent having a persistent goal; the associated conditions are *q is true, p is believed false, p is desired true, AND p continues to be desired true until (the agent believes p is true OR the agent believes p will never be true OR q becomes false)*, the associated goal is to achieve p or, to make p true.

The previous reasoning regarding contextual states can now be viewed from the specific paradigm of CxBR, in which case the JIT constructs can be modeled as follows:
Context: $\text{PG}_x$ (some persistent goal)

Transition Requirements: $q \text{ AND } \neg p \text{ AND want } p \text{ AND } [(\text{want } p) \text{ AND } 
\neg (p \text{ OR } p \text{ unattainable OR } \neg q)]$

Goal (mission sub-goal): $p$

High Level Behavior: (application-dependent procedure) $\rightarrow p$

Context Topology: Model dependent topology based on the set of Contexts allowable to be Active following the currently Active Context

As shown above, a CxBR agent has a persistent goal relative to $q$ to achieve $p$ iff the agent’s Active Context has the transition requirements listed above and defined by JIT as necessary conditions for a persistent goal. The goal $p$ may be explicit within the Context or simply implicit since it is a Transition Requirement. The agent’s behaviors are model dependent and based on what is deemed reasonable to achieve $p$. The Context Topology is also model-dependent.

A couple of notes may be necessary before continuing. First, it could be argued that a persistent goal is more accurately represented as a Mission in CxBR than a Major Context. This may be true, but it is really a matter of scale. Relying on the idea that a Mission (also, referred to as a Mission-Context) is a special form of a Context [Gonzalez and Ahler, 1998], all of what is described above regarding the necessary conditions for a persistent goal, and the associated components that fit the CxBR perspective for a Context can be captured as a Mission-Context or a Major-Context. Thus, for simplicity’s sake, the general term Context is used. This does not
necessarily imply that a Mission is exactly equivalent to a Major Context. Implementations usually allow for a higher level of abstraction with regard to Missions than Major Contexts. Major Contexts typically require a greater level of behavior-oriented details.

Second, it should be obvious that in CxBR Contexts allow for more information than simply specifying the necessary or sufficient conditions to activate the Context. This is because CxBR Contexts are concerned with controlling agent behavior based on the agent’s understanding of its environment. Consider the earlier example of rain. The condition of “Water condensed from atmospheric vapor and falling in drops” is sufficient to indicate a state of rain. However, from a human (or agent) perspective, the context of rain will likely influence behavior. Let’s extend the example to a human pedestrian. Consider the pedestrian at the outset of a rainContext. One immediate goal is likely stayDry. This goal will influence the pedestrian to take appropriate actions, such as: donRainGear, seekShelter, or deployUmbrella. This extension of the rain context to human perspective and behavior provides an example of the other components contained in a CxBR Context. Now that this has been covered the remaining definitions of JIT will be discussed in a general and abstract way without relying on specific examples.

Next, consider the JIT definition for weak achievement goal, definition 2:

An agent has a weak achievement goal relative to q and with respect to a team to bring about p if either of these conditions holds:

1. The agent has a normal achievement goal to bring about p, that is, the agent does not yet believe that p is true and has p eventually being true as a goal.
2. The agent believes that p is true, will never be true, or is irrelevant (thus q is false), but has a goal that the status of p be mutually believed by all the team members.
The state describe is an agent’s *weak achievement goal*, the associated conditions are \[ q \text{ is true AND } p \text{ is believed false AND } p \text{ is desired true} \] OR \[ (p \text{ is true OR } p \text{ will never be true OR } q \text{ is false}) \text{ AND the agent has the goal of making the status of } p \text{ known to all team members} \] the associated goal is *to achieve } p \text{ or make the status of } p \text{ known to entire team}.

These constructs can be modeled in CxBR as follows:

**Context:** \[ WAG_x \text{ (some weak achievement goal)} \]

**Transition Requirements:** \[ q AND \neg p \text{ AND want } p \] OR \[ (p \text{ OR } p \text{ unattainable OR } \neg q) \text{ AND want } p \text{ known by entire team} \]

**Goal (mission sub-goal):** \[ q AND \neg p \text{ AND want } p \rightarrow p \] OR \[ (p \text{ OR } p \text{ unattainable OR } \neg q) \rightarrow \text{ make status of } p \text{ known to team} \]

**High Level Behavior:** (application-dependent procedure) \( \rightarrow p \)

\( \rightarrow \text{ make } p \text{ known to team} \)

**Context Topology:** Model dependent topology based on the set of Contexts allowable to be Active following the currently Active Context
First of all, note that the weak achievement goal of JIT is concerned with a condition relative to a team. This is unlike the persistent goal, which is not concerned with a team condition. That noted, consider the representation of weak achievement goal as a Context. In order for the Context to become Active, the Transition Requirements \[q \text{ AND } \neg p \text{ AND want p} \] OR \[(p \text{ OR } p \text{ unattainable OR } \neg q ) \text{ AND want } p \text{ known by entire team}\] must be met. The goal will depend on whether \( p \) is known false and desired true and \( q \) is true. In this case, the goal is \( p \). However, if \( p \) is known true, or \( p \) is believed unattainable, or \( q \) becomes false (thus, \( p \) is irrelevant), then the goal is to make the status of \( p \) known to the entire team. The high level behaviors are application-dependent but should allow for a procedure to achieve the Context goal, achieve \( p \) or make status of \( p \) known to the entire team. As a note, it is possible that the WAG\(_x\) could be modeled as two separate contexts: one to provide for achieving \( p \), and the other to provide for informing all team members about the status of \( p \). However, since the description of WAG\(_x\) is meant to be abstract and taken directly from the JIT definition based on its necessary or sufficient conditions, the explanation here will adhere to a single Context representation. Implementations of specific Contexts within a given model could be separated into two Contexts. However, model complexity and the possibility of creating a combinatorial explosion in the number of Contexts modeled should be carefully considered, when making such a decision.

Again, consider the JIT definition for a joint persistent goal, Definition 3:

A team of agents have a joint persistent goal relative to \( q \) to achieve \( p \) just in case
1. They mutually believe that \( p \) is currently false
2. They mutually know they all want \( p \) to eventually be true
3. It is true (and mutually known) that until they come to mutually believe either that \( p \) is true, that \( p \) will never be true, that \( q \) is false, they will continue to mutually
believe that they each have \( p \) as a weak achievement goal relative to \( q \) and with respect to the team.

The state described is an agent’s \textit{joint persistent goal}, the associated conditions are \([q \text{ is true} \text{ AND } p \text{ is mutually believed false} \text{ AND } p \text{ is mutually desired true}] \text{ AND } [\sim (\text{mutually known that } p \text{ is true OR mutually known that } p \text{ will never be true OR mutually known that } q \text{ is false}) \text{ AND all team members mutually believe that each member has } p \text{ as a weak achievement goal relative to } q \text{ and with respect to the team.}]\) the associated goal is \textit{to achieve }\( p \).

These constructs can be modeled in CxBR as follows:

\begin{itemize}
  \item \textbf{Context:} \( \text{JPG}_x (\text{some joint persistent goal}) \)
  \item \textbf{Transition Requirements:} \([q \text{ AND } \sim p \text{ AND } \text{want } p] \text{ AND } \sim (\text{mutually known that } p \text{ is true OR mutually known that } p \text{ will never be true OR mutually known that } q \text{ is false}) \text{ AND all team members mutually believe each member has } p \text{ as a weak achievement goal relative to } q \text{ and with respect to the team.}]\)
  \item \textbf{Goal (mission sub-goal):} \( p \)
  \item \textbf{High Level Behavior:} (application-dependent procedure) \( \rightarrow p \)
  \item \textbf{Context Topology:} Model dependent topology based on the set of Contexts allowable to be Active following the currently Active Context
\end{itemize}
Important to the definition for joint persistent goal, and, therefore, important to the Context representation as well, are the facts that condition $q$ is relative to the team, as with the weak achievement goal, and that the goal $p$ is a mutual goal of all team members. The Transition Requirements for this Context to become Active are $[q \land \neg p \land \text{want } p] \land [\neg (\text{mutually known that } p \text{ is true OR mutually known that } p \text{ will never be true OR mutually known that } q \text{ is false}) \land \text{all team members mutually believe each member has } p \text{ as a weak achievement goal relative to } q \text{ and with respect to the team.}]$ When the Context is Active, the team’s high level behaviors are towards achievement of $p$.

It should be noted that the team of collaborating agents has a JPGx Context while the team members themselves have a WAGx Context. This nesting of Contexts is referred to here as Embedded Contexts.

Embedded Contexts are described in detail in Chapter 7 of this dissertation. Much of Embedded Contexts is application dependent, having to do with the control of entities and their associated team. The particular implementation is independent of CCxBR and will vary analogous to the variation in implementation of behavior control for individual agents within CxBR. In other words, the implementation of low-level behaviors of agents can be done using neural networks, a rule-based implementation, or any other appropriate knowledge representation and is independent of the CxBR specification. In the same way, the implementation of individual reasoning (including Context selection) related to the team Context is independent of the CCxBR specification. Furthermore, the actual behaviors modeled for individual agents in CxBR are obviously dependent upon the particular model being represented. This, again, carries over to
CCxBR in much the same way. The actual model for how team Contexts influence or control individual team members is application dependent.

Equally important to take note of is the difference between the JPG Context, the WAG Context, and the PG Context in terms of their relationship to individual versus team agents. At the individual level and without regard to conditions or goals relative to a team is the PG Context. In contrast, the WAG Context is again at the individual level but is concerned with conditions relative to the team. Finally, the JPG Context is a team Context. The members of a team with an Active JPG Context each have a WAG Context as their individual Active Context.

Now that the JIT definitions regarding agent and team goals are represented in CxBR terms, it is possible to consider individual and team intentions of these agents. Consider the JIT definition for intention of an individual:

Definition 4:
An agent intends relative to some condition to do an action just in case the agent has a persistent goal (relative to that condition) of having done the action and, moreover, having done it, believing throughout that the agent is doing it.

Adopting this to the contextual representation for PG, it should be obvious that an agent with an Active PG Context intends relative to q to accomplish an action, determined by the recipe to bring about p.
Likewise, considering the JIT definition, a team of agents with a joint intention, for a team of agents with an Active JPG Context \textit{jointly intends} relative to \( q \) to accomplish an action, determined by the recipe to bring about \( p \).

\textbf{Definition 5:}
A team of agents \textbf{jointly intends}, relative to some escape condition, to do an action iff the members have a joint persistent goal relative to that condition of their having done the action and, moreover, having done it, mutually believing throughout that they were doing it.

In this last section, JIT is extended to include a CxBR contextual representation of its tenets. The first three sections of this chapter approaches formalizing collaborative behaviors within CxBR by starting from CxBR and moving towards JIT. The bi-directional approach to this problem provides two compatible justifications to modeling collaborative behaviors in CxBR based on JIT. However, the examples of modeling in Section 4.3 are built primarily with the CxBR to JIT approach in mind.

\textbf{A.6 Summary}
This chapter contains a set of work related to modeling in CxBR. Embedded Contexts are specifically to modeling teams in CxBR. On the other hand, Context Moderators, including Co-Contexts and Affect Moderators, are general techniques for modeling CxBR agents regardless of concerns for collaboration. All of these techniques are compatible with CCxBR.
APPENDIX B. TEST RESULT DATA
The raw test data collected during evaluation is provided in this appendix along with a statistical mean average for each set of data. Each team type is evaluated against all other team types for ten 30-minute (108,000 simulation cycles) games. The Tables below are organized as follows.

Each table represents a set of test runs against one particular team type. The tables are split vertically with a gray divider. On the left side of the divider is the team against which all four teams are evaluated. The teams are codified as follows:

- 0 = Base Team (Simple Soccer Team)
- 1 = CCxBR (Prototype III)
- 2 = CxBRwJIT (Prototype II)
- 3 = CxBR (Prototype I)

The columns for both halves of the tables (each team) are organized as follows:

<table>
<thead>
<tr>
<th>Team Type</th>
<th>Goals Scored</th>
<th>Ball Movement</th>
<th>ComboDiagonal</th>
<th>ComboBox</th>
<th>ComboBounding</th>
<th>Possession Time</th>
</tr>
</thead>
</table>

Where:

- Team Type is one of the prototypes or control team as indicated above.
- Goals Scored is the number of goals the team scored against the opposing team.
- Ball Movement is the total distance the team moved the ball during the game.
- ComboDiagonal is the number of cycles that two players were simultaneously in a ComboDiagonal Sub-Context (ComboDiagonalControl, ComboDiagonalSupport, or ComboDiagonalReceive).
- **ComboBox** is the number of cycles that two players were simultaneously in a ComboBox Sub-Context (ComboBoxControl, ComboBoxSupport, or ComboBoxReceive).

- **ComboBounding** is the number of cycles that two players were simultaneously in a ComboBounding Sub-Context (ComboBoundingControl, ComboBoundingSupport, or ComboBoundingReceive).

- **Possession Time** is the total number of cycles that the team was in control of the ball. (passing the ball from kicker to receive counts as ball possession unless the ball is intercepted).

Horizontally, each line represents a game where the team on the left played against the team on the right. The sections of the table where there is no team on the right represent matches between teams of the same type. For these matches, the right side data is transposed to the bottom of the left hand side of the table and statistics are computed for the aggregate. So, whereas ten games for dissimilar teams are listed side by side, for two teams A and B of the same team type the team A’s stats are listed in the first ten rows of the left side of the table and team B’s stats are listed in the next ten rows on the left side of the table. This is done to compute the statistics collectively for this team type and is explained in detail here should the reader be inclined to review the data separately.

Below every data set collected for two opposing team types is a line with the mean average of the data collected for those ten games. This line is highlighted in bold, italics and underlined.
Columns are:

<table>
<thead>
<tr>
<th>Team Type</th>
<th>Goals Scored</th>
<th>Ball Movement</th>
<th>ComboDiagonal</th>
<th>ComboBox</th>
<th>ComboBounding</th>
<th>Possession Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>S</td>
<td>Move</td>
<td>CD</td>
<td>Cx</td>
<td>CB</td>
<td>Time</td>
</tr>
</tbody>
</table>

The first four tables of data provide the average mean for all data sets. The following four tables provide the standard deviation for all data sets.

### Table B.1 Evaluations Against Prototype I: CxBR

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>23</td>
<td>6196.7</td>
<td>972</td>
<td>1</td>
<td>0</td>
<td>34296</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>67301.6</td>
<td>1155</td>
<td>0</td>
<td>0</td>
<td>37878</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>63507.5</td>
<td>1037</td>
<td>2</td>
<td>0</td>
<td>35130</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>65285.8</td>
<td>1077</td>
<td>0</td>
<td>0</td>
<td>36616</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>66441.4</td>
<td>1049</td>
<td>1</td>
<td>0</td>
<td>37148</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>66438.7</td>
<td>1289</td>
<td>1</td>
<td>0</td>
<td>36951</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>62219</td>
<td>1185</td>
<td>7</td>
<td>0</td>
<td>34799</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>62174.5</td>
<td>1048</td>
<td>7</td>
<td>0</td>
<td>34912</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>64553.4</td>
<td>1090</td>
<td>6</td>
<td>0</td>
<td>36032</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>55610.5</td>
<td>974</td>
<td>1</td>
<td>0</td>
<td>30870</td>
</tr>
<tr>
<td>3</td>
<td>28.4</td>
<td>63522.91</td>
<td>1086.6</td>
<td>2.8</td>
<td>0</td>
<td>35463.2</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>62219</td>
<td>1185</td>
<td>7</td>
<td>0</td>
<td>34799</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>62174.5</td>
<td>1048</td>
<td>7</td>
<td>0</td>
<td>34912</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>64553.4</td>
<td>1090</td>
<td>6</td>
<td>0</td>
<td>36032</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>55610.5</td>
<td>974</td>
<td>1</td>
<td>0</td>
<td>30870</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>72327.2</td>
<td>1615</td>
<td>3</td>
<td>0</td>
<td>42265</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>73707.1</td>
<td>1228</td>
<td>11</td>
<td>0</td>
<td>43118</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>74394.2</td>
<td>1384</td>
<td>3</td>
<td>0</td>
<td>42940</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>72466.3</td>
<td>1296</td>
<td>6</td>
<td>0</td>
<td>41871</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>72517.9</td>
<td>1521</td>
<td>8</td>
<td>0</td>
<td>41339</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>72364.2</td>
<td>1493</td>
<td>10</td>
<td>0</td>
<td>42620</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>66674.4</td>
<td>1211</td>
<td>7</td>
<td>0</td>
<td>38473</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>75191.3</td>
<td>1385</td>
<td>1</td>
<td>0</td>
<td>43967</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>74694.6</td>
<td>1356</td>
<td>12</td>
<td>0</td>
<td>43015</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>76147.5</td>
<td>1634</td>
<td>4</td>
<td>0</td>
<td>43795</td>
</tr>
<tr>
<td>T</td>
<td>S</td>
<td>Move</td>
<td>CD</td>
<td>Cx</td>
<td>CB</td>
<td>Time</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>--------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>3</td>
<td>30.3</td>
<td>73048.47</td>
<td>1412.3</td>
<td>6.5</td>
<td>0</td>
<td>42340.3</td>
</tr>
</tbody>
</table>

Table B.2 Evaluations Against Prototype II: CxBRwJIT
<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>26</td>
<td>64421.6</td>
<td>7331</td>
<td>0</td>
<td>31</td>
<td>35340</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>62231.9</td>
<td>8361</td>
<td>133</td>
<td>28</td>
<td>33697</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>60655.6</td>
<td>7063</td>
<td>157</td>
<td>0</td>
<td>33755</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>65669.4</td>
<td>8286</td>
<td>226</td>
<td>76</td>
<td>34688</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>64419.59</td>
<td>8306.8</td>
<td>98.6</td>
<td>23.9</td>
<td>35249.4</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>74546.2</td>
<td>10794</td>
<td>299</td>
<td>0</td>
<td>42766</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>78461.3</td>
<td>11863</td>
<td>507</td>
<td>37</td>
<td>43249</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>68936.2</td>
<td>9246</td>
<td>235</td>
<td>34</td>
<td>39423</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>81988.5</td>
<td>13548</td>
<td>323</td>
<td>0</td>
<td>44881</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>73415.6</td>
<td>10242</td>
<td>648</td>
<td>0</td>
<td>42417</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>79505.0</td>
<td>10320</td>
<td>797</td>
<td>63</td>
<td>44391</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>72012.6</td>
<td>9326</td>
<td>370</td>
<td>32</td>
<td>42005</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>71833.1</td>
<td>9219</td>
<td>269</td>
<td>133</td>
<td>40561</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>74242.1</td>
<td>10350</td>
<td>588</td>
<td>0</td>
<td>41439</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>71841.8</td>
<td>9939</td>
<td>242</td>
<td>107</td>
<td>41504</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>76278</td>
<td>9200</td>
<td>1019</td>
<td>0</td>
<td>44047</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>78943.5</td>
<td>10314</td>
<td>273</td>
<td>33</td>
<td>44754</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>78814.8</td>
<td>10786</td>
<td>259</td>
<td>39</td>
<td>44286</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>83849.8</td>
<td>10381</td>
<td>279</td>
<td>34</td>
<td>46218</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>76195.4</td>
<td>10677</td>
<td>364</td>
<td>0</td>
<td>43889</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>81738.5</td>
<td>10852</td>
<td>537</td>
<td>0</td>
<td>45211</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>76674.9</td>
<td>11086</td>
<td>133</td>
<td>0</td>
<td>43890</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>71604</td>
<td>11269</td>
<td>446</td>
<td>0</td>
<td>40685</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>75256.5</td>
<td>10020</td>
<td>262</td>
<td>33</td>
<td>42314</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>80924.4</td>
<td>11994</td>
<td>448</td>
<td>0</td>
<td>45681</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>76070.6</td>
<td>9610</td>
<td>435</td>
<td>72</td>
<td>44267</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>76045.3</td>
<td>11172</td>
<td>304</td>
<td>0</td>
<td>43183</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>74086.6</td>
<td>8568</td>
<td>519</td>
<td>128</td>
<td>42393</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>72638.9</td>
<td>10779</td>
<td>134</td>
<td>0</td>
<td>41017</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>72850.4</td>
<td>10375</td>
<td>514</td>
<td>37</td>
<td>42621</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>74378.8</td>
<td>10642</td>
<td>147</td>
<td>36</td>
<td>42396</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>74527.2</td>
<td>8869</td>
<td>483</td>
<td>67</td>
<td>43817</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
<td>74429.7</td>
<td>10570</td>
<td>167</td>
<td>33</td>
<td>41722</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>79593.8</td>
<td>10478</td>
<td>450</td>
<td>0</td>
<td>44916</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>75978.8</td>
<td>9780</td>
<td>473</td>
<td>107</td>
<td>43696</td>
</tr>
<tr>
<td>2</td>
<td>39.35</td>
<td>76544.1</td>
<td>10371.1</td>
<td>382.3</td>
<td>30.95</td>
<td>43550.15</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
<td>76023.2</td>
<td>9634</td>
<td>1065</td>
<td>31</td>
<td>42640</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>80290.3</td>
<td>9729</td>
<td>626</td>
<td>34</td>
<td>45722</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>77184.4</td>
<td>9563</td>
<td>487</td>
<td>68</td>
<td>43829</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>79326.9</td>
<td>9701</td>
<td>657</td>
<td>0</td>
<td>45268</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>78637.6</td>
<td>10082</td>
<td>673</td>
<td>0</td>
<td>45106</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>80433.8</td>
<td>10250</td>
<td>953</td>
<td>113</td>
<td>45655</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>74899.8</td>
<td>10286</td>
<td>713</td>
<td>0</td>
<td>43293</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>82856.7</td>
<td>11757</td>
<td>717</td>
<td>75</td>
<td>46216</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>79672.7</td>
<td>9256</td>
<td>762</td>
<td>32</td>
<td>45767</td>
</tr>
<tr>
<td>T</td>
<td>S</td>
<td>Move</td>
<td>CD</td>
<td>Cx</td>
<td>CB</td>
<td>Time</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>74492.1</td>
<td>10149</td>
<td>1041</td>
<td>36</td>
<td>42740</td>
</tr>
<tr>
<td>2</td>
<td>44.3</td>
<td>78381.75</td>
<td>10040.7</td>
<td>769.4</td>
<td>38.9</td>
<td>44623.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>29</td>
<td>74538.2</td>
<td>1275</td>
<td>19</td>
<td>0</td>
<td>43792</td>
</tr>
<tr>
<td>3</td>
<td>30.4</td>
<td>75397.25</td>
<td>1502.6</td>
<td>7.5</td>
<td>0.5</td>
<td>43609.9</td>
</tr>
</tbody>
</table>

**Table B.3 Evaluations Against Prototype III: CCxBR**

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>26</td>
<td>60564</td>
<td>9300</td>
<td>0</td>
<td>0</td>
<td>32306</td>
</tr>
<tr>
<td>4</td>
<td>67808.7</td>
<td>9949</td>
<td>133</td>
<td>0</td>
<td>37151</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>69325.1</td>
<td>9684</td>
<td>0</td>
<td>72</td>
<td>37029</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>70830.6</td>
<td>8355</td>
<td>140</td>
<td>41</td>
<td>38882</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>67980.5</td>
<td>8607</td>
<td>206</td>
<td>0</td>
<td>37342</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>65438.2</td>
<td>9733</td>
<td>284</td>
<td>59</td>
<td>35557</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>66748.6</td>
<td>9564</td>
<td>512</td>
<td>0</td>
<td>36408</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>64628.6</td>
<td>9404</td>
<td>173</td>
<td>229</td>
<td>34943</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>62944.8</td>
<td>8860</td>
<td>259</td>
<td>0</td>
<td>35283</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>65193.2</td>
<td>8613</td>
<td>119</td>
<td>0</td>
<td>34962</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.5</td>
<td>36.3</td>
<td>66116.23</td>
<td>9206.9</td>
<td>182.6</td>
<td>40.1</td>
</tr>
<tr>
<td>2</td>
<td>28.6</td>
<td>605320.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54744</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48</td>
<td>78565.5</td>
<td>10223</td>
<td>792</td>
<td>34</td>
<td>43301</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>77920.9</td>
<td>11112</td>
<td>397</td>
<td>0</td>
<td>43340</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>70135.9</td>
<td>10998</td>
<td>622</td>
<td>105</td>
<td>40530</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>74233.4</td>
<td>10844</td>
<td>690</td>
<td>80</td>
<td>46121</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>78436.4</td>
<td>10821</td>
<td>513</td>
<td>73</td>
<td>44829</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>74665.1</td>
<td>11288</td>
<td>732</td>
<td>115</td>
<td>42559</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>75761.8</td>
<td>10991</td>
<td>765</td>
<td>77</td>
<td>42761</td>
</tr>
<tr>
<td>1</td>
<td>37</td>
<td>73897.1</td>
<td>11499</td>
<td>711</td>
<td>59</td>
<td>42427</td>
</tr>
<tr>
<td>1</td>
<td>52</td>
<td>80066.8</td>
<td>11272</td>
<td>568</td>
<td>189</td>
<td>44509</td>
</tr>
<tr>
<td>1</td>
<td>37</td>
<td>70052.8</td>
<td>10933</td>
<td>598</td>
<td>31</td>
<td>39504</td>
</tr>
<tr>
<td>1</td>
<td>41</td>
<td>73484.9</td>
<td>9554</td>
<td>646</td>
<td>55</td>
<td>41090</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>75024.7</td>
<td>11761</td>
<td>241</td>
<td>165</td>
<td>41592</td>
</tr>
<tr>
<td>1</td>
<td>48</td>
<td>80260.9</td>
<td>12008</td>
<td>617</td>
<td>99</td>
<td>43826</td>
</tr>
<tr>
<td>1</td>
<td>38</td>
<td>73449.2</td>
<td>11916</td>
<td>177</td>
<td>59</td>
<td>40866</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>73070.4</td>
<td>10510</td>
<td>301</td>
<td>42</td>
<td>42847</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>73424.4</td>
<td>10326</td>
<td>903</td>
<td>62</td>
<td>41435</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>77977.1</td>
<td>11063</td>
<td>454</td>
<td>38</td>
<td>42523</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>75355.6</td>
<td>11088</td>
<td>495</td>
<td>107</td>
<td>42928</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>70842.2</td>
<td>10770</td>
<td>604</td>
<td>0</td>
<td>40856</td>
</tr>
<tr>
<td>1</td>
<td>48</td>
<td>82350.4</td>
<td>11977</td>
<td>900</td>
<td>0</td>
<td>45504</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>42.25</td>
<td>75448.78</td>
<td>11047.7</td>
<td>586.3</td>
<td>69.5</td>
<td>42667.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>39</td>
<td>77616.4</td>
<td>10209</td>
<td>770</td>
<td>0</td>
<td>44676</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>74546.2</td>
<td>10794</td>
<td>299</td>
<td>0</td>
<td>42766</td>
</tr>
<tr>
<td>1</td>
<td>46</td>
<td>76178.4</td>
<td>9622</td>
<td>437</td>
<td>0</td>
<td>42149</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>78461.3</td>
<td>11863</td>
<td>507</td>
<td>37</td>
<td>43249</td>
</tr>
<tr>
<td>1</td>
<td>55</td>
<td>83153.6</td>
<td>10862</td>
<td>887</td>
<td>0</td>
<td>45485</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>68936.2</td>
<td>9246</td>
<td>235</td>
<td>34</td>
<td>39423</td>
</tr>
<tr>
<td>1</td>
<td>39</td>
<td>72743.4</td>
<td>9389</td>
<td>302</td>
<td>147</td>
<td>41422</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>81988.5</td>
<td>13548</td>
<td>323</td>
<td>0</td>
<td>44881</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>78058.1</td>
<td>11547</td>
<td>430</td>
<td>89</td>
<td>44220</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>73415.6</td>
<td>10242</td>
<td>648</td>
<td>0</td>
<td>42417</td>
</tr>
<tr>
<td>1</td>
<td>37</td>
<td>75490</td>
<td>11473</td>
<td>374</td>
<td>33</td>
<td>43571</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>76950.5</td>
<td>10320</td>
<td>797</td>
<td>63</td>
<td>44391</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>79082.1</td>
<td>12063</td>
<td>530</td>
<td>0</td>
<td>44038</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>72012.6</td>
<td>9326</td>
<td>370</td>
<td>32</td>
<td>42005</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>Move</td>
<td>CD</td>
<td>Cx</td>
<td>CB</td>
<td>Time</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>77656.7</td>
<td>9780</td>
<td>514</td>
<td>91</td>
<td>44432</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>74889.2</td>
<td>11378</td>
<td>531</td>
<td>34</td>
<td>41401</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>79516.4</td>
<td>11675</td>
<td>779</td>
<td>226</td>
<td>44762</td>
</tr>
<tr>
<td>1</td>
<td>44.2</td>
<td>77438.43</td>
<td>10799.8</td>
<td>555.4</td>
<td>62</td>
<td>43615.6</td>
</tr>
<tr>
<td>1</td>
<td>54</td>
<td>78746.8</td>
<td>10507</td>
<td>564</td>
<td>36</td>
<td>43883</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>82722.7</td>
<td>12671</td>
<td>653</td>
<td>0</td>
<td>45782</td>
</tr>
<tr>
<td>1</td>
<td>55</td>
<td>78185.1</td>
<td>9677</td>
<td>909</td>
<td>0</td>
<td>43259</td>
</tr>
<tr>
<td>1</td>
<td>52</td>
<td>76968.6</td>
<td>11802</td>
<td>1284</td>
<td>0</td>
<td>44262</td>
</tr>
<tr>
<td>1</td>
<td>54</td>
<td>78341.3</td>
<td>10856</td>
<td>676</td>
<td>0</td>
<td>43703</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>79004.2</td>
<td>10816</td>
<td>430</td>
<td>0</td>
<td>44801</td>
</tr>
<tr>
<td>1</td>
<td>66</td>
<td>84065.5</td>
<td>11445</td>
<td>1135</td>
<td>30</td>
<td>45332</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>81667.8</td>
<td>11290</td>
<td>509</td>
<td>0</td>
<td>46561</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>77873.8</td>
<td>10297</td>
<td>1046</td>
<td>78</td>
<td>44349</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>77520.8</td>
<td>10872</td>
<td>1122</td>
<td>32</td>
<td>43167</td>
</tr>
<tr>
<td>1</td>
<td>49.6</td>
<td>79509.66</td>
<td>11023.3</td>
<td>832.8</td>
<td>17.6</td>
<td>44509.9</td>
</tr>
</tbody>
</table>

Table B.4 Evaluations Against Base Team (Simple Soccer Team)
Table B.5 Evaluations Against Prototype I: CxBR

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25</td>
<td>104189</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54585</td>
<td>1</td>
<td>36</td>
<td>67980.5</td>
<td>8607</td>
<td>206</td>
<td>0</td>
<td>37342</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>104351</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54150</td>
<td>1</td>
<td>42</td>
<td>65438.2</td>
<td>9733</td>
<td>284</td>
<td>59</td>
<td>35557</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>104899</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54715</td>
<td>1</td>
<td>34</td>
<td>66748.6</td>
<td>9564</td>
<td>512</td>
<td>0</td>
<td>36408</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>107120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>55077</td>
<td>1</td>
<td>37</td>
<td>64628.6</td>
<td>9404</td>
<td>173</td>
<td>229</td>
<td>34943</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>105153</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54493</td>
<td>1</td>
<td>35</td>
<td>62944.8</td>
<td>8860</td>
<td>259</td>
<td>0</td>
<td>35283</td>
</tr>
<tr>
<td>0</td>
<td>24</td>
<td>110626</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56413</td>
<td>1</td>
<td>38</td>
<td>65193.2</td>
<td>8613</td>
<td>119</td>
<td>0</td>
<td>34962</td>
</tr>
<tr>
<td>0</td>
<td>28.6</td>
<td>105320.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54744</td>
<td>1</td>
<td>36.3</td>
<td>66116.23</td>
<td>9206.9</td>
<td>182.6</td>
<td>40.1</td>
<td>35986.3</td>
</tr>
</tbody>
</table>

Table B.5 Evaluations Against Prototype I: CxBR
<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>31</td>
<td>64535.4</td>
<td>1080</td>
<td>6</td>
<td>0</td>
<td>36032</td>
<td>0</td>
<td>27</td>
<td>110416</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56773</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>55610.5</td>
<td>974</td>
<td>1</td>
<td>0</td>
<td>30870</td>
<td>0</td>
<td>33</td>
<td>120873</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>61134</td>
</tr>
<tr>
<td>3</td>
<td>4.22</td>
<td>34144.7</td>
<td>98.11</td>
<td>2.88</td>
<td>0.00</td>
<td>1995.88</td>
<td>0</td>
<td>4.69</td>
<td>4791.56</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2025.31</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>72327.2</td>
<td>1615</td>
<td>3</td>
<td>0</td>
<td>42265</td>
<td>1</td>
<td>54</td>
<td>78746.8</td>
<td>10507</td>
<td>564</td>
<td>36</td>
<td>43883</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>73707.1</td>
<td>1228</td>
<td>11</td>
<td>0</td>
<td>43118</td>
<td>1</td>
<td>44</td>
<td>82722.7</td>
<td>12671</td>
<td>653</td>
<td>0</td>
<td>45782</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>74394.2</td>
<td>1384</td>
<td>3</td>
<td>0</td>
<td>42940</td>
<td>1</td>
<td>55</td>
<td>78185.1</td>
<td>9677</td>
<td>909</td>
<td>0</td>
<td>43259</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>72466.3</td>
<td>1296</td>
<td>6</td>
<td>0</td>
<td>41871</td>
<td>1</td>
<td>52</td>
<td>76986.8</td>
<td>11802</td>
<td>1284</td>
<td>0</td>
<td>44262</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>72517.9</td>
<td>1521</td>
<td>8</td>
<td>0</td>
<td>41339</td>
<td>1</td>
<td>54</td>
<td>78341.3</td>
<td>10856</td>
<td>676</td>
<td>0</td>
<td>43703</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>72364.2</td>
<td>1493</td>
<td>10</td>
<td>0</td>
<td>42620</td>
<td>1</td>
<td>45</td>
<td>79004.2</td>
<td>10816</td>
<td>430</td>
<td>0</td>
<td>44801</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>66674.4</td>
<td>1211</td>
<td>7</td>
<td>0</td>
<td>38473</td>
<td>1</td>
<td>66</td>
<td>84065.5</td>
<td>11445</td>
<td>1135</td>
<td>30</td>
<td>45332</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>75191.3</td>
<td>1385</td>
<td>1</td>
<td>0</td>
<td>43967</td>
<td>1</td>
<td>40</td>
<td>81667.8</td>
<td>11290</td>
<td>509</td>
<td>0</td>
<td>46561</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>74694.6</td>
<td>1356</td>
<td>12</td>
<td>0</td>
<td>43015</td>
<td>1</td>
<td>43</td>
<td>77873.8</td>
<td>10297</td>
<td>1046</td>
<td>78</td>
<td>44339</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>76147.5</td>
<td>1634</td>
<td>4</td>
<td>0</td>
<td>43795</td>
<td>1</td>
<td>43</td>
<td>77520.8</td>
<td>10872</td>
<td>1122</td>
<td>32</td>
<td>43167</td>
</tr>
<tr>
<td>3</td>
<td>4.22</td>
<td>2611.68</td>
<td>149.55</td>
<td>3.75</td>
<td>0.00</td>
<td>1579.20</td>
<td>1</td>
<td>7.99</td>
<td>2421.72</td>
<td>894.71</td>
<td>302.96</td>
<td>26.26</td>
<td>1109.62</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>76664.6</td>
<td>1571</td>
<td>2</td>
<td>5</td>
<td>44025</td>
<td>2</td>
<td>49</td>
<td>76023.2</td>
<td>9634</td>
<td>1085</td>
<td>31</td>
<td>42640</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>75610.8</td>
<td>1368</td>
<td>4</td>
<td>0</td>
<td>43755</td>
<td>2</td>
<td>42</td>
<td>80290.3</td>
<td>9729</td>
<td>626</td>
<td>34</td>
<td>45722</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>79951.4</td>
<td>1723</td>
<td>1</td>
<td>0</td>
<td>46067</td>
<td>2</td>
<td>36</td>
<td>77184.4</td>
<td>9563</td>
<td>487</td>
<td>68</td>
<td>43829</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>73617.8</td>
<td>1407</td>
<td>11</td>
<td>0</td>
<td>42461</td>
<td>2</td>
<td>44</td>
<td>79326.9</td>
<td>9701</td>
<td>657</td>
<td>0</td>
<td>45268</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>76017.3</td>
<td>1460</td>
<td>11</td>
<td>0</td>
<td>44651</td>
<td>2</td>
<td>40</td>
<td>78637.6</td>
<td>10082</td>
<td>673</td>
<td>0</td>
<td>45106</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>71724.8</td>
<td>1517</td>
<td>14</td>
<td>0</td>
<td>41849</td>
<td>2</td>
<td>47</td>
<td>80433.8</td>
<td>10250</td>
<td>953</td>
<td>113</td>
<td>45655</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>77308</td>
<td>1807</td>
<td>2</td>
<td>0</td>
<td>44060</td>
<td>2</td>
<td>41</td>
<td>74899.8</td>
<td>10286</td>
<td>713</td>
<td>0</td>
<td>43293</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>75562.5</td>
<td>1392</td>
<td>4</td>
<td>0</td>
<td>43069</td>
<td>2</td>
<td>42</td>
<td>82856.7</td>
<td>11757</td>
<td>717</td>
<td>75</td>
<td>46216</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>72977.1</td>
<td>1506</td>
<td>7</td>
<td>0</td>
<td>42370</td>
<td>2</td>
<td>50</td>
<td>79762.7</td>
<td>9256</td>
<td>762</td>
<td>32</td>
<td>45767</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>74538.2</td>
<td>1275</td>
<td>19</td>
<td>0</td>
<td>43792</td>
<td>2</td>
<td>52</td>
<td>74492.1</td>
<td>10149</td>
<td>1014</td>
<td>36</td>
<td>42740</td>
</tr>
<tr>
<td>3</td>
<td>2.91</td>
<td>23500.02</td>
<td>163.04</td>
<td>6.02</td>
<td>1.58</td>
<td>1238.32</td>
<td>2</td>
<td>5.06</td>
<td>2662.32</td>
<td>690.01</td>
<td>189.47</td>
<td>37.00</td>
<td>1359.98</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>83675.8</td>
<td>1469</td>
<td>3</td>
<td>0</td>
<td>46658</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>82251.2</td>
<td>1332</td>
<td>3</td>
<td>0</td>
<td>46904</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>79973.8</td>
<td>1411</td>
<td>18</td>
<td>0</td>
<td>45904</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>75364.4</td>
<td>1609</td>
<td>19</td>
<td>0</td>
<td>43564</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>79064</td>
<td>1324</td>
<td>16</td>
<td>0</td>
<td>45094</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>38</td>
<td>76368.9</td>
<td>1526</td>
<td>5</td>
<td>0</td>
<td>43457</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>41</td>
<td>80303.8</td>
<td>1451</td>
<td>13</td>
<td>0</td>
<td>45023</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>77009.5</td>
<td>1122</td>
<td>27</td>
<td>0</td>
<td>43724</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>80450.9</td>
<td>1400</td>
<td>43</td>
<td>0</td>
<td>46362</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>77676.7</td>
<td>1344</td>
<td>59</td>
<td>0</td>
<td>44768</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>79827.4</td>
<td>1571</td>
<td>56</td>
<td>0</td>
<td>46061</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>81426.3</td>
<td>1530</td>
<td>50</td>
<td>0</td>
<td>47285</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>39</td>
<td>77276.9</td>
<td>1237</td>
<td>41</td>
<td>0</td>
<td>44245</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>82344</td>
<td>1584</td>
<td>53</td>
<td>0</td>
<td>46744</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>75310.9</td>
<td>1351</td>
<td>49</td>
<td>0</td>
<td>43594</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>41</td>
<td>75519.2</td>
<td>1545</td>
<td>37</td>
<td>0</td>
<td>44238</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table B.6 Evaluations Against Prototype II: CxBRwJIT

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>29</td>
<td>66289.7</td>
<td>8014</td>
<td>176</td>
<td>0</td>
<td>37021</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>66448.7</td>
<td>8584</td>
<td>21</td>
<td>38</td>
<td>36123</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>58269.5</td>
<td>7250</td>
<td>36</td>
<td>34</td>
<td>32779</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>68153.3</td>
<td>10346</td>
<td>130</td>
<td>0</td>
<td>37020</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>64850.2</td>
<td>8632</td>
<td>62</td>
<td>32</td>
<td>35033</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>67206</td>
<td>9237</td>
<td>45</td>
<td>0</td>
<td>37038</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>64421.6</td>
<td>7331</td>
<td>0</td>
<td>31</td>
<td>35340</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>62231.9</td>
<td>8361</td>
<td>133</td>
<td>28</td>
<td>33697</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>60655.6</td>
<td>7063</td>
<td>157</td>
<td>0</td>
<td>33755</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>65669.4</td>
<td>8286</td>
<td>226</td>
<td>76</td>
<td>34688</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>29</td>
<td>66289.7</td>
<td>8014</td>
<td>176</td>
<td>0</td>
<td>37021</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>66448.7</td>
<td>8584</td>
<td>21</td>
<td>38</td>
<td>36123</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>58269.5</td>
<td>7250</td>
<td>36</td>
<td>34</td>
<td>32779</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>68153.3</td>
<td>10346</td>
<td>130</td>
<td>0</td>
<td>37020</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>64850.2</td>
<td>8632</td>
<td>62</td>
<td>32</td>
<td>35033</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>67206</td>
<td>9237</td>
<td>45</td>
<td>0</td>
<td>37038</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>64421.6</td>
<td>7331</td>
<td>0</td>
<td>31</td>
<td>35340</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>62231.9</td>
<td>8361</td>
<td>133</td>
<td>28</td>
<td>33697</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>60655.6</td>
<td>7063</td>
<td>157</td>
<td>0</td>
<td>33755</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>65669.4</td>
<td>8286</td>
<td>226</td>
<td>76</td>
<td>34688</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>38</td>
<td>74546.2</td>
<td>10794</td>
<td>299</td>
<td>0</td>
<td>42766</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>78461.3</td>
<td>11863</td>
<td>507</td>
<td>37</td>
<td>43249</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>68936.2</td>
<td>9246</td>
<td>235</td>
<td>34</td>
<td>39423</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>81988.5</td>
<td>13548</td>
<td>323</td>
<td>0</td>
<td>44881</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>73415.6</td>
<td>10242</td>
<td>648</td>
<td>0</td>
<td>42417</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>76950.5</td>
<td>10320</td>
<td>797</td>
<td>63</td>
<td>44391</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>72012.6</td>
<td>9326</td>
<td>370</td>
<td>32</td>
<td>42005</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>71833.1</td>
<td>9219</td>
<td>269</td>
<td>133</td>
<td>40561</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>74242.1</td>
<td>10350</td>
<td>588</td>
<td>0</td>
<td>41439</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>71841.8</td>
<td>9393</td>
<td>242</td>
<td>107</td>
<td>41504</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>38</td>
<td>76278</td>
<td>9200</td>
<td>1019</td>
<td>0</td>
<td>44047</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>78943.5</td>
<td>10314</td>
<td>273</td>
<td>33</td>
<td>44754</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>78814.8</td>
<td>10786</td>
<td>259</td>
<td>39</td>
<td>44286</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>83849.8</td>
<td>10381</td>
<td>279</td>
<td>34</td>
<td>46218</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>76195.4</td>
<td>10677</td>
<td>364</td>
<td>0</td>
<td>43889</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>76674.9</td>
<td>11086</td>
<td>133</td>
<td>0</td>
<td>43890</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>71604</td>
<td>11269</td>
<td>446</td>
<td>0</td>
<td>40685</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>75256.5</td>
<td>10020</td>
<td>262</td>
<td>33</td>
<td>42314</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>80924.4</td>
<td>11994</td>
<td>448</td>
<td>0</td>
<td>45681</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>76070.6</td>
<td>9610</td>
<td>435</td>
<td>72</td>
<td>44267</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>76545.3</td>
<td>11172</td>
<td>304</td>
<td>0</td>
<td>43183</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>74088.6</td>
<td>8568</td>
<td>519</td>
<td>128</td>
<td>42393</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>38</td>
<td>76278</td>
<td>9200</td>
<td>1019</td>
<td>0</td>
<td>44047</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>78943.5</td>
<td>10314</td>
<td>273</td>
<td>33</td>
<td>44754</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>78814.8</td>
<td>10786</td>
<td>259</td>
<td>39</td>
<td>44286</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>83849.8</td>
<td>10381</td>
<td>279</td>
<td>34</td>
<td>46218</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>76195.4</td>
<td>10677</td>
<td>364</td>
<td>0</td>
<td>43889</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>76674.9</td>
<td>11086</td>
<td>133</td>
<td>0</td>
<td>43890</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>71604</td>
<td>11269</td>
<td>446</td>
<td>0</td>
<td>40685</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>75256.5</td>
<td>10020</td>
<td>262</td>
<td>33</td>
<td>42314</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>80924.4</td>
<td>11994</td>
<td>448</td>
<td>0</td>
<td>45681</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>76070.6</td>
<td>9610</td>
<td>435</td>
<td>72</td>
<td>44267</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>76545.3</td>
<td>11172</td>
<td>304</td>
<td>0</td>
<td>43183</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>74088.6</td>
<td>8568</td>
<td>519</td>
<td>128</td>
<td>42393</td>
</tr>
<tr>
<td>T</td>
<td>S</td>
<td>Move</td>
<td>CD</td>
<td>Cx</td>
<td>CB</td>
<td>Time</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-------</td>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>-------</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>72638.9</td>
<td>10779</td>
<td>134</td>
<td>0</td>
<td>41017</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>72850.4</td>
<td>10375</td>
<td>514</td>
<td>37</td>
<td>42621</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>74378.8</td>
<td>10642</td>
<td>147</td>
<td>36</td>
<td>42396</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>74527.2</td>
<td>8869</td>
<td>483</td>
<td>67</td>
<td>43817</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
<td>74429.7</td>
<td>10570</td>
<td>167</td>
<td>33</td>
<td>41722</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>79593.8</td>
<td>10478</td>
<td>450</td>
<td>0</td>
<td>44916</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>75978.8</td>
<td>9780</td>
<td>473</td>
<td>107</td>
<td>43696</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>3203.38</td>
<td>837.12</td>
<td>203.10</td>
<td>37.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>49</td>
<td>76023.2</td>
<td>9634</td>
<td>1065</td>
<td>31</td>
<td>42640</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>80290.3</td>
<td>9729</td>
<td>626</td>
<td>34</td>
<td>45722</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>77184.4</td>
<td>9563</td>
<td>487</td>
<td>68</td>
<td>43829</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>79326.9</td>
<td>9701</td>
<td>657</td>
<td>0</td>
<td>45268</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>78637.6</td>
<td>10082</td>
<td>673</td>
<td>0</td>
<td>45106</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>80433.8</td>
<td>10250</td>
<td>953</td>
<td>113</td>
<td>45655</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>74899.8</td>
<td>10286</td>
<td>713</td>
<td>0</td>
<td>43293</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>82856.7</td>
<td>11757</td>
<td>717</td>
<td>75</td>
<td>46216</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>79672.7</td>
<td>9256</td>
<td>762</td>
<td>32</td>
<td>45767</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>74492.1</td>
<td>10149</td>
<td>1041</td>
<td>36</td>
<td>42740</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2682.32</td>
<td>690.01</td>
<td>189.47</td>
<td>37.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48</td>
<td>78565.5</td>
<td>10223</td>
<td>792</td>
<td>34</td>
<td>43301</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>77920.9</td>
<td>11112</td>
<td>397</td>
<td>0</td>
<td>43340</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>70135.9</td>
<td>10998</td>
<td>622</td>
<td>105</td>
<td>40530</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>74233.4</td>
<td>10844</td>
<td>690</td>
<td>80</td>
<td>46121</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>78436.4</td>
<td>10821</td>
<td>513</td>
<td>73</td>
<td>44829</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>74665.1</td>
<td>11288</td>
<td>732</td>
<td>115</td>
<td>42559</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>75761.8</td>
<td>10991</td>
<td>765</td>
<td>77</td>
<td>42761</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>73897.1</td>
<td>11499</td>
<td>711</td>
<td>59</td>
<td>42427</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>80066.8</td>
<td>11272</td>
<td>568</td>
<td>189</td>
<td>44509</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>70052.8</td>
<td>10933</td>
<td>598</td>
<td>31</td>
<td>39504</td>
</tr>
</tbody>
</table>

Table B.7 Evaluations Against Prototype III: CCxCB
<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41</td>
<td>73484.9</td>
<td>9554</td>
<td>646</td>
<td>55</td>
<td>41090</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>75024.7</td>
<td>11761</td>
<td>241</td>
<td>165</td>
<td>41592</td>
</tr>
<tr>
<td>1</td>
<td>48</td>
<td>80260.9</td>
<td>12008</td>
<td>617</td>
<td>99</td>
<td>43826</td>
</tr>
<tr>
<td>1</td>
<td>38</td>
<td>73449.2</td>
<td>11916</td>
<td>177</td>
<td>59</td>
<td>40866</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>73070.4</td>
<td>10510</td>
<td>301</td>
<td>42</td>
<td>42847</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>73424.4</td>
<td>10326</td>
<td>903</td>
<td>62</td>
<td>41435</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>77977.1</td>
<td>11063</td>
<td>454</td>
<td>38</td>
<td>42523</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>75355.6</td>
<td>11088</td>
<td>495</td>
<td>107</td>
<td>42928</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>70842.2</td>
<td>10770</td>
<td>604</td>
<td>0</td>
<td>40856</td>
</tr>
<tr>
<td>1</td>
<td>48</td>
<td>82350.4</td>
<td>11977</td>
<td>900</td>
<td>0</td>
<td>45504</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.53</td>
<td></td>
<td>3435.25</td>
<td>618.84</td>
<td>200.30</td>
<td>50.60</td>
<td>1727.46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39</td>
<td>77616.4</td>
<td>10209</td>
<td>770</td>
<td>0</td>
<td>44676</td>
</tr>
<tr>
<td>1</td>
<td>46</td>
<td>76178.4</td>
<td>9622</td>
<td>437</td>
<td>0</td>
<td>42149</td>
</tr>
<tr>
<td>1</td>
<td>55</td>
<td>83153.6</td>
<td>10862</td>
<td>887</td>
<td>0</td>
<td>45485</td>
</tr>
<tr>
<td>1</td>
<td>39</td>
<td>72743.4</td>
<td>9389</td>
<td>302</td>
<td>147</td>
<td>41422</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>78058.1</td>
<td>11547</td>
<td>430</td>
<td>89</td>
<td>44220</td>
</tr>
<tr>
<td>1</td>
<td>37</td>
<td>75490</td>
<td>11473</td>
<td>374</td>
<td>33</td>
<td>43571</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>79082.1</td>
<td>12063</td>
<td>530</td>
<td>0</td>
<td>44038</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>77656.7</td>
<td>9780</td>
<td>514</td>
<td>91</td>
<td>44432</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>74889.2</td>
<td>11378</td>
<td>531</td>
<td>34</td>
<td>41401</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>79516.4</td>
<td>11675</td>
<td>779</td>
<td>226</td>
<td>44762</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.20</td>
<td></td>
<td>2868.84</td>
<td>970.56</td>
<td>193.17</td>
<td>76.52</td>
<td>1453.46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>54</td>
<td>78746.8</td>
<td>10507</td>
<td>564</td>
<td>36</td>
<td>43883</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>82722.7</td>
<td>12671</td>
<td>653</td>
<td>0</td>
<td>45782</td>
</tr>
<tr>
<td>1</td>
<td>55</td>
<td>78185.1</td>
<td>9677</td>
<td>909</td>
<td>0</td>
<td>43259</td>
</tr>
<tr>
<td>1</td>
<td>52</td>
<td>76968.6</td>
<td>11802</td>
<td>1284</td>
<td>0</td>
<td>44262</td>
</tr>
<tr>
<td>1</td>
<td>54</td>
<td>78341.3</td>
<td>10856</td>
<td>676</td>
<td>0</td>
<td>43703</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>79004.2</td>
<td>10816</td>
<td>430</td>
<td>0</td>
<td>44801</td>
</tr>
<tr>
<td>1</td>
<td>66</td>
<td>84065.5</td>
<td>11445</td>
<td>1135</td>
<td>30</td>
<td>45332</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>81667.8</td>
<td>11290</td>
<td>509</td>
<td>0</td>
<td>46561</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>77873.8</td>
<td>10297</td>
<td>1046</td>
<td>78</td>
<td>44349</td>
</tr>
<tr>
<td>1</td>
<td>43</td>
<td>77520.8</td>
<td>10872</td>
<td>1122</td>
<td>32</td>
<td>43167</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.98</td>
<td></td>
<td>2421.72</td>
<td>334.71</td>
<td>302.56</td>
<td>26.26</td>
<td>1109.62</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>98494.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>50343</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>91077.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47806</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>94095.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49454</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>94017.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>48654</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>94466</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47746</td>
</tr>
<tr>
<td>0</td>
<td>28</td>
<td>96582.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49984</td>
</tr>
</tbody>
</table>

Table B.8 Evaluations Against Base Team (Simple Soccer Team)
<table>
<thead>
<tr>
<th>T</th>
<th>S</th>
<th>Move</th>
<th>CD</th>
<th>Cx</th>
<th>CB</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33</td>
<td>96756.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49068</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>88098.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46284</td>
</tr>
<tr>
<td>0</td>
<td>24</td>
<td>93495.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>48572</td>
</tr>
<tr>
<td>0</td>
<td>29</td>
<td>94096.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47966</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>85562</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>44798</td>
</tr>
<tr>
<td>0</td>
<td>17</td>
<td>89804.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47751</td>
</tr>
<tr>
<td>0</td>
<td>19</td>
<td>88335.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46694</td>
</tr>
<tr>
<td>0</td>
<td>26</td>
<td>89882.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47072</td>
</tr>
<tr>
<td>0</td>
<td>28</td>
<td>90704.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>45822</td>
</tr>
<tr>
<td>0</td>
<td>23</td>
<td>85186.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>44367</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>81537.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42638</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>95171.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>48659</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>88694.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46530</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>88356.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46626</td>
</tr>
<tr>
<td>4.24</td>
<td>4394.43</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1944.90</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>40</td>
<td>113288</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>58086</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>99163.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>52330</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>103015</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53952</td>
</tr>
<tr>
<td>0</td>
<td>21</td>
<td>101400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53639</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>104189</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54585</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>104351</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54150</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>104899</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54715</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>107120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>55077</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>105153</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54493</td>
</tr>
<tr>
<td>0</td>
<td>24</td>
<td>110626</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56413</td>
</tr>
<tr>
<td>5.40</td>
<td>4161.87</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1569.96</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>107559</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56032</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>112170</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>58093</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>116355</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59731</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>106201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>55081</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>106136</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54868</td>
</tr>
<tr>
<td>0</td>
<td>29</td>
<td>102221</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54343</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>111910</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>57759</td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>112357</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>57185</td>
</tr>
<tr>
<td>0</td>
<td>29</td>
<td>114715</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>58431</td>
</tr>
<tr>
<td>0</td>
<td>29</td>
<td>106692</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54898</td>
</tr>
<tr>
<td>3.01</td>
<td>4496.29</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1846.11</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>35</td>
<td>112952</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>58156</td>
</tr>
<tr>
<td>0</td>
<td>23</td>
<td>106028</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>55472</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>111770</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56935</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>109648</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56368</td>
</tr>
<tr>
<td>0</td>
<td>29</td>
<td>103971</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53788</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>107802</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56859</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>38</td>
<td>113120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>57712</td>
</tr>
<tr>
<td>0</td>
<td>28</td>
<td>114729</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>59346</td>
</tr>
<tr>
<td>0</td>
<td>27</td>
<td>110416</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56773</td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>120873</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>61134</td>
</tr>
<tr>
<td>0</td>
<td>4.69</td>
<td>4791.56</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2025.31</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>62219</td>
<td>1185</td>
<td>7</td>
<td>0</td>
<td>34799</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>62174.5</td>
<td>1048</td>
<td>7</td>
<td>0</td>
<td>34912</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>64553.4</td>
<td>1080</td>
<td>6</td>
<td>0</td>
<td>36032</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>55610.5</td>
<td>974</td>
<td>1</td>
<td>0</td>
<td>30870</td>
</tr>
<tr>
<td></td>
<td>4.22</td>
<td>3414.47</td>
<td>98.11</td>
<td>2.88</td>
<td>0.00</td>
<td>1995.88</td>
</tr>
</tbody>
</table>

166
APPENDIX C. PROTOTYPE CODE
The simulation environment (Simple Soccer Simulator) and the base soccer team (Simple Soccer Team) are both provided as a courtesy of Buckland [2005] via www.wordware.com/files/ai. The prototypes built for this research are an extension of Buckland’s source code. For the complete source code, visit Buckland’s site referenced here. This appendix includes the exact source code modifications to build all three prototypes evaluated in this dissertation. To reproduce this work, substitute the code below for the respective files from Buckland.

FieldPlayer.cpp

#include "FieldPlayer.h"
#include "PlayerBase.h"
#include "SteeringBehaviors.h"
#include "2D/Transformations.h"
#include "2D/Geometry.h"
#include "misc/Cgdi.h"
#include "2D/C2DMatrix.h"
#include "Goal.h"
#include "Game/Region.h"
#include "game/EntityFunctionTemplates.h"
#include "ParamLoader.h"
#include "SoccerTeam.h"
#include "time/Regulator.h"
#include "Debug/DebugConsole.h"

#include <limits>

using std::vector;

//------------------------------- dtor --------------------------------------

//------------------------------------------------------------------------
FieldPlayer::~FieldPlayer()
{
    delete m_pKickLimiter;
    delete m_pStateMachine;
}

//------------------------------- ctor -------------------------------------

FieldPlayer::FieldPlayer(SoccerTeam* home_team,
    int home_region,
    int team_region,
    int team_region

State<FieldPlayer>* start_state,
Vector2D heading,
Vector2D velocity,
double mass,
double max_force,
double max_speed,
double max_turn_rate,
double scale,
player_role role): PlayerBase(home_team,
home_region,
heading,
velocity,
mass,
max_force,
max_speed,
max_turn_rate,
scale,
role)
{
    //set up the state machine
    m_pStateMachine = new StateMachine<FieldPlayer>(this);

    if (start_state)
    {
        m_pStateMachine->SetCurrentState(start_state);
        m_pStateMachine->SetPreviousState(start_state);
        m_pStateMachine->SetGlobalState(GlobalPlayerState::Instance());

        m_pStateMachine->CurrentState()->Enter(this);
    }

    m_pSteering->SeparationOn();

    //set up the kick regulator
    m_pKickLimiter = new Regulator(Prm.PlayerKickFrequency);
}

//------------------------------- Update -------------------------------
//
//------------------------------- Update -------------------------------

void FieldPlayer::Update()
{
    //run the logic for the current state
    m_pStateMachine->Update();

    //calculate the combined steering force
    m_pSteering->Calculate();

    //if no steering force is produced decelerate the player by applying a
    //braking force
    if (m_pSteering->Force().isZero())
    {
        const double BrakingRate = 0.8;
    }
m_vVelocity = m_vVelocity * BrakingRate;
}

//the steering force's side component is a force that rotates the
//player about its axis. We must limit the rotation so that a player
//can only turn by PlayerMaxTurnRate rads per update.
double TurningForce = m_pSteering->SideComponent();

Clamp(TurningForce, -Prm.PlayerMaxTurnRate, Prm.PlayerMaxTurnRate);

//rotate the heading vector
Vec2D RotateAroundOrigin(m_vHeading, TurningForce);

//make sure the velocity vector points in the same direction as
//the heading vector
m_vVelocity = m_vHeading * m_vVelocity.Length();

//and recreate m_vSide
m_vSide = m_vHeading.Perp();

//now to calculate the acceleration due to the force exerted by
//the forward component of the steering force in the direction
//of the player's heading
Vector2D accel = m_vHeading * m_pSteering->ForwardComponent() / m_dMass;

m_vVelocity += accel;

//make sure player does not exceed maximum velocity
m_vVelocity.Truncate(m_dMaxSpeed);

//update the position
m_vPosition += m_vVelocity;

//enforce a non-penetration constraint if desired
if(Prm.bNonPenetrationConstraint)
{
    EnforceNonPenetrationConstraint(this,
    AutoList<PlayerBase>::GetAllMembers());
}

//------------ HandleMessage --------------------------------------
//  routes any messages appropriately
//------------
bool FieldPlayer::HandleMessage(const Telegram& msg)
{
    return m_pStateMachine->HandleMessage(msg);
}

//------------------ Render ----------------------------------------
//  //------------------
void FieldPlayer::Render()
{
    gdi->TransparentText();
    gdi->TextColor(Cgdi::grey);

    // set appropriate team color
    if (Team()->Color() == SoccerTeam::blue){gdi->BluePen();}
    else{gdi->RedPen();}

    // render the player's body
    m_vecPlayerVBTrans = WorldTransform(m_vecPlayerVB,
     Pos(),
     Heading(),
     Side(),
     Scale());

    gdi->ClosedShape(m_vecPlayerVBTrans);

    // and 'is 'ead
    gdi->BrownBrush();
    if (Prm.bHighlightIfThreatened && (Team()->ControllingPlayer() == this) &&
    isThreatened()) gdi->YellowBrush();
    gdi->Circle(Pos(), 6);

    // render the state
    if (Prm.bStates)
    {
        gdi->TextColor(0, 170, 0);
        gdi->TextAtPos(m_vPosition.x, m_vPosition.y -20,
        std::string(m_pStateMachine->GetNameOfCurrentState()));
    }

    // show IDs
    if (Prm.bIDs)
    {
        gdi->TextColor(0, 170, 0);
        gdi->TextAtPos(Pos().x-20, Pos().y-20, ttos(ID()));
    }

    if (Prm.bViewTargets)
    {
        gdi->RedBrush();
        gdi->Circle(Steering()->Target(), 3);
        gdi->TextAtPos(Steering()->Target(), ttos(ID()));
    }
}

FieldPlayer.h
#pragma warning (disable:4786)
#ifndef FIELDPLAYER_H
#define FIELDPLAYER_H
//------------------------------------------------------------------------------
//
// Name:   FieldPlayer.h
//
// Desc:   Derived from a PlayerBase, this class encapsulates a player
//         capable of moving around a soccer pitch, kicking, dribbling,
//         shooting etc
//
// Author: Mat Buckland 2003 (fup@ai-junkie.com)
//------------------------------------------------------------------------------
#include <vector>
#include <string>
#include <algorithm>
#include <cassert>
#include "FieldPlayerStates.h"
#include "2D/Vector2D.h"
#include "FSM/StateMachine.h"
#include "PlayerBase.h"
#include "FSM/StateMachine.h"
#include "time/Regulator.h"

class CSteeringBehavior;
class SoccerTeam;
class SoccerPitch;
class Goal;
struct Telegram;

class FieldPlayer : public PlayerBase
{
private:

    // an instance of the state machine class
   StateMachine<FieldPlayer>*  m_pStateMachine;

    // limits the number of kicks a player may take per second
    Regulator*                  m_pKickLimiter;

public:

    FieldPlayer(SoccerTeam* home_team,
                int home_region,
                State<FieldPlayer>* start_state,
                Vector2D heading,
                Vector2D velocity,
                double mass,
double max_force,
double max_speed,
double max_turn_rate,
double scale,
player_role role);

~FieldPlayer();

//call this to update the player's position and orientation
void Update();

void Render();

bool HandleMessage(const Telegram& msg);

StateMachine<FieldPlayer>* GetFSM() const {return m_pStateMachine;}

bool isReadyForNextKick() const {return m_pKickLimiter->isReady();}

};

#endif

FieldPlayerStates.cpp

#include "FieldPlayerStates.h"
#include "Debug/DebugConsole.h"
#include "SoccerPitch.h"
#include "FieldPlayer.h"
#include "SteeringBehaviors.h"
#include "SoccerTeam.h"
#include "Goal.h"
#include "2D/geometry.h"
#include "SoccerBall.h"
#include "ParamLoader.h"
#include "Messaging/Telegram.h"
#include "Messaging/MessageDispatcher.h"
#include "SoccerMessages.h"
#include "MetricData.h"
#include "time/Regulator.h"

//uncomment below to send state info to the debug window
#define PLAYER_STATE_INFO_ON

#define DEBUG

173
Global state

GlobalPlayerState* GlobalPlayerState::Instance()
{
    static GlobalPlayerState instance;

    return &instance;
}

void GlobalPlayerState::Execute(FieldPlayer* player)
{
    //if a player is in possession and close to the ball reduce his max speed
    if((player->BallWithinReceivingRange()) && (player->isControllingPlayer()))
    {
        player->SetMaxSpeed(Prm.PlayerMaxSpeedWithBall);
    }
    else
    {
        player->SetMaxSpeed(Prm.PlayerMaxSpeedWithoutBall);
    }
}

bool GlobalPlayerState::OnMessage(FieldPlayer* player, const Telegram& telegram)
{
    switch(telegram.Msg)
    {
    case Msg_ReceiveBall:
    {
        //set the target
        player->Steering()->
        >SetTarget(*(static_cast<Vector2D*>(telegram.ExtraInfo)));

        if(player->Team()->Type() == SoccerTeam::CxBRwJIT)
        {
            PlayerBase::combo_context comboContext = player-
            >SelectComboReceiveContext();

            if(comboContext == PlayerBase::COMBO_BOUNDING)
            {
                //change state
                player->GetFSM()->
                >ChangeState(ComboBoundingReceive::Instance());
            }
            else if(comboContext == PlayerBase::COMBO_BOX)
            {
                //change state
            }
        }
    }
    }
break;

case Msg_ReceiveBallComboDiagonal:
{
  if(player->Team()->Type() != SoccerTeam::CxBRwJIT)
  {
    //set the target
    player->Steering()->SetTarget(*(static_cast<Vector2D*>(telegram.ExtraInfo)));

    //change state
    player->GetFSM()->ChangeState(ComboDiagonalReceive::Instance());
  }
  else if(player->Team()->Type() == SoccerTeam::CxBRwJIT)
  {
    //TODO -- add code for transitioning to comboContexts based
    on inference rather than communication (which is used by CCxBR)
    assert(0);
  }

  return true;
}
break;

case Msg_ReceiveBallComboBox:
{
    if(player->Team()->Type() != SoccerTeam::CxBRwJIT)
    {
        // set the target
        player->Steering()->SetTarget(*(static_cast<Vector2D*>(telegram.ExtraInfo)));
        // change state
        player->GetFSM()->ChangeState(ComboBoxReceive::Instance());
    }
    else if(player->Team()->Type() == SoccerTeam::CxBRwJIT)
    {
        // TODO -- add code for transitioning to comboContexts based on inference rather than communication (which is used by CCxBR)
        assert(0);
    }
    return true;
}
break;

case Msg_ReceiveBallComboDribblePass:
{
    if(player->Team()->Type() != SoccerTeam::CxBRwJIT)
    {
        // set the target
        player->Steering()->SetTarget(*(static_cast<Vector2D*>(telegram.ExtraInfo)));
        // change state
        player->GetFSM()->ChangeState(ComboDribblePassReceive::Instance());
    }
    else if(player->Team()->Type() == SoccerTeam::CxBRwJIT)
    {
        // TODO -- add code for transitioning to comboContexts based on inference rather than communication (which is used by CCxBR)
        assert(0);
    }
    return true;
}
break;

case Msg_ReceiveBallComboBounding:
{
    if(player->Team()->Type() != SoccerTeam::CxBRwJIT)
    {
        // set the target
        player->Steering()->SetTarget(*(static_cast<Vector2D*>(telegram.ExtraInfo)));
        // change state
        player->GetFSM()->ChangeState(ComboBoundingReceive::Instance());
    }
    else if(player->Team()->Type() == SoccerTeam::CxBRwJIT)
    {
        // TODO -- add code for transitioning to comboContexts based on inference rather than communication (which is used by CCxBR)
        assert(0);
    }
    return true;
}
break;

player->Steering()->SetTarget(*(static_cast<Vector2D*>(telegram.ExtraInfo)));  

    //change state
    player->GetFSM()->ChangeState(ComboBoundingReceive::Instance());
}
else if(player->Team()->Type() == SoccerTeam::CxBRwJIT)
{
    //TODO -- add code for transitioning to comboContexts based
    //on inference rather than communication (which is used by CCxBR)
    assert(0);
}

    return true;
}

break;

case Msg_SupportAttacker:
{
    //if already supporting just return
    if (player->GetFSM()->isInState(*SupportAttacker::Instance()))
    {
        return true;
    }

    //set the target to be the best supporting position
    player->Steering()->SetTarget(player->Team()->GetSupportSpot());

    //change the state
    player->GetFSM()->ChangeState(SupportAttacker::Instance());
    return true;
}

break;

case Msg_Wait:
{
    //change the state
    player->GetFSM()->ChangeState(Wait::Instance());
    return true;
}

break;

case Msg_GoHome:
{
    player->SetDefaultHomeRegion();
    player->GetFSM()->ChangeState(ReturnToHomeRegion::Instance());
    return true;
}
break;

case Msg_PassToMe:
{
    //get the position of the player requesting the pass
    FieldPlayer* receiver = static_cast<FieldPlayer*>(telegram.ExtraInfo);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " received request from " << receiver->ID() << " to make pass " << "";
    #endif

    //if the ball is not within kicking range or their is already a //receiving player, this player cannot pass the ball to the player //making the request.
    if (player->Team()->Receiver() != NULL ||
        !player->BallWithinKickingRange())
    {
        #ifdef PLAYER_STATE_INFO_ON
        debug_con << "Player " << player->ID() << " cannot make requested pass <cannot kick ball>" << "";
        #endif

        return true;
    }

    //make the pass
    player->Ball()->Kick(receiver->Pos() - player->Ball()->Pos(),
                         Prm.MaxPassingForce);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " Passed ball to requesting player" << "";
    #endif

    //let the receiver know a pass is coming
    Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
                            player->ID(),
                            receiver->ID(),
                            Msg_ReceiveBall,
                            &receiver->Pos());

    //change state
    player->GetFSM()->ChangeState(Wait::Instance());

    player->FindSupport();

    return true;
}
break;

}//end switch

return false;
}

//****************************************************************************
** CHASEBALL
ChaseBall* ChaseBall::Instance()
{
    static ChaseBall instance;

    return &instance;
}

void ChaseBall::Enter(FieldPlayer* player)
{
    player->Steering()->SeekOn();

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters chase state" << "at pos "
    " << player->Pos().x << ", " << player->Pos().y << "";
    #endif
}

void ChaseBall::Execute(FieldPlayer* player)
{
    //if the ball is within kicking range the player changes state to KickBall.
    if (player->BallWithinKickingRange())
    {
        player->GetFSM()->ChangeState(KickBall::Instance());

        return;
    }

    //if the player is the closest player to the ball then he should keep
    //chasing it
    if (player->isClosestTeamMemberToBall())
    {
        player->Steering()->SetTarget(player->Ball()->Pos());

        return;
    }

    //if the player is not closest to the ball anymore, he should return back
    //to his home region and wait for another opportunity
    player->GetFSM()->ChangeState(ReturnToHomeRegion::Instance());
void ChaseBall::Exit(FieldPlayer* player)
{
    player->Steering()->SeekOff();
}

//***************************************************************************
**SUPPORT ATTACKING PLAYER

SupportAttacker* SupportAttacker::Instance()
{
    static SupportAttacker instance;

    return &instance;
}

void SupportAttacker::Enter(FieldPlayer* player)
{
    player->Steering()->ArriveOn();

    // needs to be context sensitive when finding support spot
    if(player->Team()->Type() == SoccerTeam::BASE)
    {
        player->Steering()->SetTarget(player->Team()->GetSupportSpot());
    }
    else
    {
        // TO BE ADDED TO BE ADDED TO BE ADDED TO BE ADDED TO BE ADDED TO BE ADDED
        // TRANSITION TO COMBO CONTEXT SUPPORT AS APPROPRIATE
    }

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters support state" << "" << 
               player->ComboContext() << ";
    #endif
}

void SupportAttacker::Execute(FieldPlayer* player)
{
    //if his team loses control go back home
    if(!player->Team()->InControl())
    {
        player->GetFSM()->ChangeState(ReturnToHomeRegion::Instance()); return;
    }

    //if the best supporting spot changes, change the steering target
if (player->Team()->GetSupportSpot() != player->Steering()->Target())
{
    player->Steering()->SetTarget(player->Team()->GetSupportSpot());
    player->Steering()->ArriveOn();
}

//if this player has a shot at the goal AND the attacker can pass
//the ball to him the attacker should pass the ball to this player
if (player->Team()->CanShoot(player->Pos(),
    Prm.MaxShootingForce))
{
    player->Team()->RequestPass(player);
}

//if this player is located at the support spot and his team still have
//possession, he should remain still and turn to face the ball
if (player->AtTarget())
{
    player->Steering()->ArriveOff();

    //the player should keep his eyes on the ball!
    player->TrackBall();
    player->SetVelocity(Vector2D(0,0));

    //if not threatened by another player request a pass
    if (!player->isThreatened())
    {
        player->Team()->RequestPass(player);
    }
}
}

void SupportAttacker::Exit(FieldPlayer* player)
{
    //set supporting player to null so that the team knows it has to
    //determine a new one.
    player->Team()->SetSupportingPlayer(NULL);

    player->Steering()->ArriveOff();
}

//************************************************************************
RETURN TO HOME REGION

ReturnToHomeRegion* ReturnToHomeRegion::Instance()
{
    static ReturnToHomeRegion instance;
    return &instance;
}
void ReturnToHomeRegion::Enter(FieldPlayer* player)
{
    player->Steering()->ArriveOn();

    if (!player->HomeRegion()->Inside(player->Steering()->Target(), Region::halfsize))
    {
        player->Steering()->SetTarget(player->HomeRegion()->Center());
    }

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ReturnToHome state" << "",
    #endif
}

void ReturnToHomeRegion::Execute(FieldPlayer* player)
{
    if (player->Pitch()->GameOn())
    {
        //if the ball is nearer this player than any other team member &&
        //there is not an assigned receiver && the goalkeeper does not gave
        //the ball, go chase it
        if (player->isClosestTeamMemberToBall() &&
            (player->Team()->Receiver() == NULL) &&
            !player->Pitch()->GoalKeeperHasBall())
        {
            player->GetFSM()->ChangeState(ChaseBall::Instance());
            return;
        }
    }

    //if game is on and close enough to home, change state to wait and set the
    //player target to his current position.(so that if he gets jostled out of
    //position he can move back to it)
    if (player->Pitch()->GameOn() &&
        player->HomeRegion()->Inside(player->Pos(), Region::halfsize))
    {
        player->Steering()->SetTarget(player->Pos());
        player->GetFSM()->ChangeState(Wait::Instance());
    }

    //if game is not on the player must return much closer to the center of his
    //home region
    else if (!player->Pitch()->GameOn() && player->AtTarget())
    {
        player->GetFSM()->ChangeState(Wait::Instance());
    }
}
void ReturnToHomeRegion::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
}

//***************************************************************************
** WAIT

Wait* Wait::Instance()
{
    static Wait instance;

    return &instance;
}

void Wait::Enter(FieldPlayer* player)
{
    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters wait state" << " at pos " << player->Pos().x << ", " << player->Pos().y << ";";
    #endif

    //if the game is not on make sure the target is the center of the player's
    //home region. This is ensure all the players are in the correct positions
    //ready for kick off
    if (!player->Pitch()->GameOn())
    {
        player->Steering()->SetTarget(player->HomeRegion()->Center());
    }
}

void Wait::Execute(FieldPlayer* player)
{
    //if the player has been jostled out of position, get back in position
    if (!player->AtTarget())
    {
        player->Steering()->ArriveOn();

        return;
    }

    else
    {
        player->Steering()->ArriveOff();

        player->SetVelocity(Vector2D(0,0));

        //the player should keep his eyes on the ball!
        player->TrackBall();
    }
if ( player->Team()->InControl()    &&
    (!player->isControllingPlayer()) &&
    player->isAheadOfAttacker() )
{
    player->Team()->RequestPass(player);
    return;
}

if (player->Pitch()->GameOn())
{
    //if the ball is nearer this player than any other team member AND
    //there is not an assigned receiver AND neither goalkeeper has
    //the ball, go chase it
    if (player->isClosestTeamMemberToBall() &&
        player->Team()->Receiver() == NULL  &&
        !player->Pitch()->GoalKeeperHasBall())
    {
        player->GetFSM()->ChangeState(KickBall::Instance());
        return;
    }
}

void Wait::Exit(FieldPlayer* player)
{
}

//************************************************************************
KICK BALL
************************************************************************
KickBall* KickBall::Instance()
{
    static KickBall instance;
    return &instance;
}

void KickBall::Enter(FieldPlayer* player)
{
    //let the team know this player is controlling
    player->Team()->SetControllingPlayer(player);

    if (player->Role() == PlayerBase::attacker)
    {
        PlayerBase::combo_context cc = player->SelectComboControlContext();
        player->GetFSM()->ChangeState(KickBall::Instance());
    }
}
switch(cc)
{
case PlayerBase::COMBO_DIAGONAL:
    player->GetFSM()->ChangeState(ComboDiagonalControl::Instance());
    break;

case PlayerBase::COMBO_BOX:
    player->GetFSM()->ChangeState(ComboBoxControl::Instance());
    break;

case PlayerBase::COMBO_BOUNDING:
    player->GetFSM()->ChangeState(ComboBoundingControl::Instance());
    break;

case PlayerBase::COMBO_DRIBBLE_PASS:
    player->GetFSM()->ChangeState(ComboDribblePassControl::Instance());
    break;
}

return;
}

//the player can only make so many kick attempts per second.
if (!player->isReadyForNextKick())
{
    player->GetFSM()->ChangeState(ChaseBall::Instance());
}

#ifdef PLAYER_STATE_INFO_ON
debug_con << "Player " << player->ID() << " enters kick state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
#endif

void KickBall::Execute(FieldPlayer* player)
{
    //calculate the dot product of the vector pointing to the ball
    //and the player's heading
    Vector2D ToBall = player->Ball()->Pos() - player->Pos();
    double dot = player->Heading().Dot(Vec2DNormalize(ToBall));

    //cannot kick the ball if the goalkeeper is in possession or if it is
    //behind the player or if there is already an assigned receiver. So just
    //continue chasing the ball
    if (player->Team()->Receiver() != NULL   ||
player->Pitch()->GoalKeeperHasBall() ||
(dot < 0) )
{
#ifdef PLAYER_STATE_INFO_ON
  debug_con << "Goalie has ball / ball behind player" << "":
#endif

  player->GetFSM()->ChangeState(ChaseBall::Instance());

  return;
}

/* Attempt a shot at the goal */

//if a shot is possible, this vector will hold the position along the
//opponent's goal line the player should aim for.
Vector2D BallTarget;

//the dot product is used to adjust the shooting force. The more
//directly the ball is ahead, the more forceful the kick
double power = Prm.MaxShootingForce * dot;

//if it is determined that the player could score a goal from this position
//OR if he should just kick the ball anyway, the player will attempt
//to make the shot
if (player->Team()->CanShoot(player->Ball()->Pos(),
  power,
  BallTarget)                   
  ||
  (RandFloat() < Prm.ChancePlayerAttemptsPotShot))
{
#ifdef PLAYER_STATE_INFO_ON
  debug_con << "Player " << player->ID() << " attempts a shot at " <<
  BallTarget << "":
#endif

  //add some noise to the kick. We don't want players who are
  //too accurate! The amount of noise can be adjusted by altering
  //Prm.PlayerKickingAccuracy

  BallTarget = AddNoiseToKick(player->Ball()->Pos(), BallTarget);

  //this is the direction the ball will be kicked in
  Vector2D KickDirection = BallTarget - player->Ball()->Pos();

  player->Ball()->Kick(KickDirection, power);

  if(player->Team()->Color() == SoccerTeam::blue)
  {
    MetricData::IncrementBlueTeamGoalAttempts();
  }
  else
  {
    MetricData::IncrementRedTeamGoalAttempts();
  }

  //change state
player->GetFSM()->ChangeState(Wait::Instance());
player->FindSupport();
return;
}

/* Attempt a pass to a player */
//if a receiver is found this will point to it
PlayerBase* receiver = NULL;

power = Prm.MaxPassingForce * dot;

//test if there are any potential candidates available to receive a pass
if (player->isThreatened() &&
    player->Team()->FindPass(player,
    receiver,
    BallTarget,
    power,
    Prm.MinPassDist))
{
    //add some noise to the kick
    BallTarget = AddNoiseToKick(player->Ball()->Pos(), BallTarget);

    Vector2D KickDirection = BallTarget - player->Ball()->Pos();
    player->Ball()->Kick(KickDirection, power);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " passes the ball with force " << power << " to player "
    << receiver->ID() << " Target is " << BallTarget << ""
    #endif

    //let the receiver know a pass is coming
    Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
    player->ID(),
    receiver->ID(),
    Msg_ReceiveBall,
    &BallTarget);

    //the player should wait at his current position unless instructed
    //otherwise
    player->GetFSM()->ChangeState(Wait::Instance());
    player->FindSupport();
    return;
}

//cannot shoot or pass, so dribble the ball upfield
else
{
  player->FindSupport();

  player->GetFSM()->ChangeState(Dribble::Instance());
}

//***************************************************************************
DRIBBLE
Dribble* Dribble::Instance()
{
  static Dribble instance;

  return &instance;
}

void Dribble::Enter(FieldPlayer* player)
{
  //let the team know this player is controlling
  player->Team()->SetControllingPlayer(player);

#if defined PLAYER_STATE_INFO_ON
  debug_con << "Player " << player->ID() << " enters dribble state" << " at
  pos " << player->Pos().x << ", " << player->Pos().y << "";
#endif
}

void Dribble::Execute(FieldPlayer* player)
{
  double dot = player->Team()->HomeGoal()->Facing().Dot(player->Heading());

  //if the ball is between the player and the home goal, it needs to swivel
  //the ball around by doing multiple small kicks and turns until the player
  //is facing in the correct direction
  if (dot < 0)
  {
    //the player's heading is going to be rotated by a small amount (Pi/4)
    //and then the ball will be kicked in that direction
    Vector2D direction = player->Heading();

    //calculate the sign (+/-) of the angle between the player heading and
    //the facing direction of the goal so that the player rotates around in the
    //correct direction
    double angle = QuarterPi * -1 *
                     player->Team()->HomeGoal()->Facing().Sign(player->Heading());

    Vec2DRotateAroundOrigin(direction, angle);
// this value works well when the player is attempting to control the
// ball and turn at the same time
const double KickingForce = 0.8;

player->Ball()->Kick(direction, KickingForce);
}

// kick the ball down the field
else
{
    player->Ball()->Kick(player->Team()->HomeGoal()->Facing(),
    Prm.MaxDribbleForce);
}

// the player has kicked the ball so he must now change state to follow it
player->GetFSM()->ChangeState(ChaseBall::Instance());
return;
}

//************************************************************************
RECEIVEBALL

ReceiveBall* ReceiveBall::Instance()
{
    static ReceiveBall instance;

    return &instance;
}

void ReceiveBall::Enter(FieldPlayer* player)
{
// let the team know this player is receiving the ball
player->Team()->SetReceiver(player);

// this player is also now the controlling player
player->Team()->SetControllingPlayer(player);

// there are two types of receive behavior. One uses arrive to direct
// the receiver to the position sent by the passer in its telegram. The
// other uses the pursuit behavior to pursue the ball.
// This statement selects between them dependent on the probability
// ChanceOfUsingArriveTypeReceiveBehavior, whether or not an opposing
// player is close to the receiving player, and whether or not the receiving
// player is in the opponent's 'hot region' (the third of the pitch closest
// to the opponent's goal
const double PassThreatRadius = 70.0;

if ((player->InHotRegion() ||
    RandFloat() < Prm.ChanceOfUsingArriveTypeReceiveBehavior) &&
!player->Team()->isOpponentWithinRadius(player->Pos(),
PassThreatRadius))
{ player->Steering()->ArriveOn();

#ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters receive state (Using
    Arrive)" << "";
#endif
}
else
{
    player->Steering()->PursuitOn();

#ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters receive state (Using
    Pursuit)" << "";
#endif
}
}

void ReceiveBall::Execute(FieldPlayer* player)
{
//if the ball comes close enough to the player or if his team lose control
//he should change state to chase the ball
if (player->BallWithinReceivingRange() || !player->Team()->InControl())
{
    player->GetFSM()->ChangeState(ChaseBall::Instance());

    return;
}

if (player->Steering()->PursuitIsOn())
{
    player->Steering()->SetTarget(player->Ball()->Pos());
}

//if the player has 'arrived' at the steering target he should wait and
//turn to face the ball
if (player->AtTarget())
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->TrackBall();
    player->SetVelocity(Vector2D(0,0));
}
}

void ReceiveBall::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetReceiver(NULL);
}
ComboDiagonalControl
ComboDiagonalControl* ComboDiagonalControl::Instance()
{
    static ComboDiagonalControl instance;

    return &instance;
}

void ComboDiagonalControl::Enter(FieldPlayer* player)
{
    // Let the team know this player is controlling
    player->Team()->SetControllingPlayer(player);

    // Keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_DIAGONAL);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboDiagonalControl state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
    #endif
}

void ComboDiagonalControl::Execute(FieldPlayer* player)
{
    // Calculate the dot product of the vector pointing to the ball
    // and the player's heading
    Vector2D ToBall = player->Ball()->Pos() - player->Pos();
    double dot = player->Heading().Dot(Vec2DNormalize(ToBall));

    // Cannot kick the ball if the goalkeeper is in possession or if it is
    // behind the player or if there is already an assigned receiver. So just
    // continue chasing the ball
    if (player->Team()->Receiver() != NULL   ||
    player->Pitch()->GoalKeeperHasBall()   ||
    (dot < 0)     )
    {
        #ifdef PLAYER_STATE_INFO_ON
        debug_con << "Goaly has ball / ball behind player" << "";
        #endif
        player->GetFSM()->ChangeState(ChaseBall::Instance());
        return;
    }

    /*------------------------------------------------------------------- Attempt a shot at
    the goal ----------------------------------*/

    // If a shot is possible, this vector will hold the position along the
    // opponent's goal line the player should aim for.
    Vector2D BallTarget;

    // The dot product is used to adjust the shooting force. The more
// directly the ball is ahead, the more forceful the kick
double power = Prm.MaxShootingForce * dot;

// if it is determined that the player could score a goal from this position
if (player->Team() -> CanShoot(player->Ball()->Pos(),
    power,
    BallTarget))
{
    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " attempts a shot at " << BallTarget << ""
    #endif

    // add some noise to the kick. We don't want players who are too accurate! The amount of noise can be adjusted by altering
    // Prm.PlayerKickingAccuracy
    BallTarget = AddNoiseToKick(player->Ball()->Pos(), BallTarget);

    // this is the direction the ball will be kicked in
    Vector2D KickDirection = BallTarget - player->Ball()->Pos();

    player->Ball()->Kick(KickDirection, power);

    if(player->Team()->Color() == SoccerTeam::blue)
    {
        MetricData::IncrementBlueTeamGoalAttempts();
    }
    else
    {
        MetricData::IncrementRedTeamGoalAttempts();
    }

    // change state
    player->GetFSM()->ChangeState(Wait::Instance());
    player->FindSupport();
    return;
}

/* ---------------------------------------- Attempt a pass to a player ---
---------------------------------------- */

// if a receiver is found this will point to it
PlayerBase* receiver = NULL;

power = Prm.MaxPassingForce * dot;

// test if the other attacker can receive a pass
if (player->isThreatened() &&
    player->Team()->FindPassComboDiagonal(player,
        receiver,
        BallTarget,
        power,
        5,
        100))
Vector2D suppSpot = player->Team()->CalculateSupportSpotComboDiagonal(player->Pos(), BallTarget);
player->Team()->SetSupportSpotComboDiagonal(suppSpot);

Vector2D KickDirection = BallTarget - player->Ball()->Pos();
player->Ball()->Kick(KickDirection, power);

#ifdef PLAYER_STATE_INFO_ON
  debug_con << "Player " << player->ID() << " passes the ball with force " << power << " to player " << receiver->ID() << " Target is " << BallTarget << "";
#endif

// let the receiver know a pass is coming
// and for CCxBR team the play is ComboDiagonal
if(player->Team()->Type() == SoccerTeam::CCxBR)
{
  Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
                          player->ID(),
                          receiver->ID(),
                          Msg_ReceiveBallComboDiagonal,
                          &BallTarget);
}
else
{
  Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
                          player->ID(),
                          receiver->ID(),
                          Msg_ReceiveBall,
                          &BallTarget);
}

// the player should attempt to continue the ComboDiagonal collaborative play
player->GetFSM()->ChangeState(ComboDiagonalSupport::Instance());

return;

// cannot shoot or pass, so dribble the ball upfield
else
{
  player->FindSupport();

  player->GetFSM()->ChangeState(Dribble::Instance());
}
void ComboDiagonalControl::Exit(FieldPlayer* player)
{
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

// ------------------------------------
ComboDiagonalSupport* ComboDiagonalSupport::Instance()
{
    static ComboDiagonalSupport instance;

    return &instance;
}

void ComboDiagonalSupport::Enter(FieldPlayer* player)
{
    // let the team know this player is the supporting player
    player->Team()->SetSupportingPlayer(player);

    player->Steering()->ArriveOn();

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_DIAGONAL);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboDiagonalSupport
state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
    #endif
}

void ComboDiagonalSupport::Execute(FieldPlayer* player)
{
    // if his team loses control
    // he should change state to chase the ball
    if (!player->Team()->InControl())
    {
        player->GetFSM()->ChangeState(ChaseBall::Instance());

        return;
    }

    Vector2D debugVector1 = player->Team()->GetSupportSpotComboDiagonal();

    player->Steering()->SetTarget(debugVector1);

    // if the player has 'arrived' at the steering target he should wait and
    // turn to face the ball - unless the play is broken (the other attacker no
    // longer in same combo)
    if (player->AtTarget())
    {
        player->Steering()->ArriveOff();
        player->Steering()->PursuitOff();
        player->TrackBall();
    }
player->SetVelocity(Vector2D(0, 0));

    // find teammate and determine current context
    std::vector<PlayerBase*>::const_iterator curPlyr = player->Team()->
    >Members().begin();

    // iterate through all this player's team members and find the other
    attacker,
    // calculate if it is in a position to be passed the ball
    for (curPlyr; curPlyr != player->Team()->Members().end(); ++curPlyr)
    {
        // make sure the potential receiver being examined is not this player
        // and that it is the other attacker
        if ((*curPlyr != player) &&  (*curPlyr)->Role() ==
            PlayerBase::player_role::attacker)
        {
            if(player->Team()->Type() == SoccerTeam::CCxBR)  // for
                CCxBR direct context recognition
            {
                if(((*curPlyr)->ComboContext() !=
                    PlayerBase::COMBO_DIAGONAL)
                {
                    player->GetFSM()->
                    >ChangeState(SupportAttacker::Instance());
                }
            }
        }
    }

void ComboDiagonalSupport::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetSupportingPlayer(NULL);

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

void ComboDiagonalSupport::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetSupportingPlayer(NULL);

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

ComboDiagonalReceive* ComboDiagonalReceive::Instance()
{
    static ComboDiagonalReceive instance;

    return &instance;
}

void ComboDiagonalReceive::Enter(FieldPlayer* player)
{
    // let the team know this player is receiving the ball
    player->Team()->SetReceiver(player);
//this player is also now the controlling player
player->Team()->SetControllingPlayer(player);

// use arrive to direct the receiver to the position sent by the passer in
its telegram.
player->Steering()->ArriveOn();

// keep track of combo states for use in context determination
player->SetComboContext(PlayerBase::COMBO_DIAGONAL);

#ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboDiagonalReceive
    state" " at pos " << player->Pos().x << 
    ", " << player->Pos().y << "";
#endif
}

void ComboDiagonalReceive::Execute(FieldPlayer* player)
{
    // ------------------------  First, check for state transitions
    // ------------------------

    //if the ball is within kicking range the player changes state to KickBall.
    if (player->BallWithinKickingRange())
    {
        player->GetFSM()->ChangeState(ComboDiagonalControl::Instance());
        return;
    }

    //if player's team loses control he should change state to chase the ball
    if (!player->Team()->InControl())
    {
        player->GetFSM()->ChangeState(ChaseBall::Instance());
        return;
    }

    // ------------------------  Next, carry out current state
    // behavior  ------------------------

    //if the ball comes close enough to the player he should pursuit the ball
    if (player->BallWithinReceivingRange() )
    {
        player->Steering()->PursuitOn();
        return;
    }

    if (player->Steering()->PursuitIsOn())
    {
        player->Steering()->SetTarget(player->Ball()->Pos());
    }

    //if the player has 'arrived' at the steering target he should wait and
    //turn to face the ball
if (player->AtTarget()) {
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->TrackBall();
    player->SetVelocity(Vector2D(0,0));
}

void ComboDiagonalReceive::Exit(FieldPlayer* player) {
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetReceiver(NULL);
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

//************************************************************************
COMBOBOX
ComboBoxControl* ComboBoxControl::Instance() {
    static ComboBoxControl instance;
    return &instance;
}

void ComboBoxControl::Enter(FieldPlayer* player) {
    // let the team know this player is controlling
    player->Team()->SetControllingPlayer(player);
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_BOX);
    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboBoxControl state"
    << " at pos " << player->Pos().x << ", " << player->Pos().y << "\n";
    #endif
}

void ComboBoxControl::Execute(FieldPlayer* player) {
    // calculate the dot product of the vector pointing to the ball
    // and the player's heading
    Vector2D ToBall = player->Ball()->Pos() - player->Pos();
    double dot = player->Heading().Dot(Vec2DNormalize(ToBall));
    // cannot kick the ball if the goalkeeper is in possession or if it is
    // behind the player or if there is already an assigned receiver. So just
// continue chasing the ball
if (player->Team()->Receiver() != NULL ||
    player->Pitch()->GoalKeeperHasBall() ||
    (dot < 0))
{
    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Goaly has ball / ball behind player" << ";
    #endif

    player->GetFSM()->ChangeState(ChaseBall::Instance());

    return;
}

/*---------------------------------------------------- Attempt a shot at
the goal ---------------------------------*/

// if a shot is possible, this vector will hold the position along the
// opponent's goal line the player should aim for.
Vector2D BallTarget;

// the dot product is used to adjust the shooting force. The more
// directly the ball is ahead, the more forceful the kick
double power = Prm.MaxShootingForce * dot;

// if it is determined that the player could score a goal from this position
if (player->Team()->CanShoot(player->Ball()->Pos(),
    power,
    BallTarget))
{
    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " attempts a shot at " << 
    BallTarget << ";
    #endif

    // add some noise to the kick. We don't want players who are
    // too accurate! The amount of noise can be adjusted by altering
    // Prm.PlayerKickingAccuracy
    BallTarget = AddNoiseToKick(player->Ball()->Pos(), BallTarget);

    // this is the direction the ball will be kicked in
    Vector2D KickDirection = BallTarget - player->Ball()->Pos();

    player->Ball()->Kick(KickDirection, power);

    if(player->Team()->Color() == SoccerTeam::blue)
    {
        MetricData::IncrementBlueTeamGoalAttempts();
    }
    else
    {
        MetricData::IncrementRedTeamGoalAttempts();
    }

    // change state
player->GetFSM()->ChangeState(Wait::Instance());
player->FindSupport();
    return;
}

/*---------------------------------------- Attempt a pass to a player ---
----------------------------------------*/

// if a receiver is found this will point to it
PlayerBase* receiver = NULL;

power = Prm.MaxPassingForce * dot;
power *= .8;

// test if the other attacker can receive a pass
bool debugIsThreatened = player->isThreatened();
bool debugFindPass = player->Team()->FindPassComboBox(player,
receiver,
BallTarget,
power,
5,
70);

if (debugFindPass)
{
    Vector2D suppSpot = player->Team()->CalculateSupportSpotComboBox(player->Pos(), receiver->Pos(), BallTarget);
    player->Team()->SetSupportSpotComboBox(suppSpot);

    Vector2D KickDirection = BallTarget - player->Ball()->Pos();

    player->Ball()->Kick(KickDirection, power);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " passes the ball with force " << power << " to player "
    << receiver->ID() << " Target is " << BallTarget << "";
    #endif

    // let the receiver know a pass is coming
    // and for CCxBR team the play is ComboBox
    if(player->Team()->Type() == SoccerTeam::CCxBR)
    {
        Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
        player->ID(),
        receiver->ID(),
        Msg_ReceiveBallComboBox,
        &BallTarget);
    }
    else
    {
        Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
        player->ID(),
        receiver->ID(),
        Msg_ReceiveBallComboBox,
        &BallTarget);
    }
}
receiver->ID(),
Msg_ReceiveBall,
&BallTarget);

  //the player should attempt to continue the ComboBox collaborative play
  player->GetFSM()->ChangeState(ComboBoxSupport::Instance());

  return;
}

  //---------------------------------------------  cannot shoot or pass, so
  dribble the ball upfield
  else
  {
    player->FindSupport();

    player->GetFSM()->ChangeState(Dribble::Instance());
  }
}

void ComboBoxControl::Exit(FieldPlayer* player)
{
// keep track of combo states for use in context determination
  player->SetComboContext(PlayerBase::NO_COMBO);
}

  // ------------------------------------
ComboBoxSupport* ComboBoxSupport::Instance()
{
  static ComboBoxSupport instance;

  return &instance;
}

void ComboBoxSupport::Enter(FieldPlayer* player)
{
  //let the team know this player is the supporting player
  player->Team()->SetSupportingPlayer(player);

  // use arrive to reach combo support spot
  player->Steering()->ArriveOn();

  // keep track of combo states for use in context determination
  player->SetComboContext(PlayerBase::COMBO_BOX);

#if defined PLAYER_STATE_INFO_ON
  debug_con << "Player " << player->ID() << " enters ComboBoxSupport state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
#endif
void ComboBoxSupport::Execute(FieldPlayer* player)
{
    // if his team loses control
    // he should change state to chase the ball
    if (!player->Team()->InControl())
    {
        player->GetFSM()->ChangeState(ChaseBall::Instance());
        return;
    }

    Vector2D debugVector1 = player->Team()->GetSupportSpotComboBox();
    player->Steering()->SetTarget(debugVector1);

    // if the player has 'arrived' at the steering target he should wait and
    // turn to face the ball - unless the play is broken (the other attacker no
    // longer in same combo)
    if (player->AtTarget())
    {
        player->Steering()->ArriveOff();
        player->Steering()->PursuitOff();
        player->TrackBall();
        player->SetVelocity(Vector2D(0,0));

        // find teammate and determine current context
        std::vector<PlayerBase*>::const_iterator curPlyr = player->Team()->Members().begin();

        // iterate through all this player's team members and find the other
        // attacker,
        // calculate if it is in a position to be passed the ball
        for (curPlyr; curPlyr != player->Team()->Members().end(); ++curPlyr)
        {
            // make sure the potential receiver being examined is not this player
            // and that it is the other attacker
            if ( (*curPlyr != player) && (*curPlyr)->Role() == PlayerBase::player_role::attacker)
            {
                if (player->Team()->Type() == SoccerTeam::CCxBR) // for
                // CCxBR direct context recognition
                if ((*curPlyr)->ComboContext() != PlayerBase::COMBO_BOUNDING)
                {
                    player->GetFSM()->ChangeState(SupportAttacker::Instance());
                }
            }
        }
    }
}
void ComboBoxSupport::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();

    player->Team() -> SetSupportingPlayer(NULL);

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

ComboBoxReceive* ComboBoxReceive::Instance()
{
    static ComboBoxReceive instance;

    return &instance;
}

void ComboBoxReceive::Enter(FieldPlayer* player)
{
    // let the team know this player is receiving the ball
    player->Team()->SetReceiver(player);

    // this player is also now the controlling player
    player->Team()->SetControllingPlayer(player);

    // pursue the ball
    player->Steering()->PursuitOn();

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_BOX);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboBoxReceive state"
               << " at pos " << player->Pos().x << ", " << player->Pos().y << ";"
    #endif
}

void ComboBoxReceive::Execute(FieldPlayer* player)
{

    // ------------------------  First, check for state transitions  ---------------------------

    // if the ball is within kicking range the player changes state to KickBall.
    if (player->BallWithinKickingRange())
    {
        player->GetFSM()->ChangeState(ComboBoxControl::Instance());
        return;
    }

    // if player's team loses control he should change state to chase the ball
    if (!player->Team()->InControl())
    {
    
    
    }
//if the ball comes close enough to the player he should pursue the ball
if (player->BallWithinReceivingRange() )
{
    player->Steering()->PursuitOn();
    return;
}

if (player->Steering()->PursuitIsOn())
{
    player->Steering()->SetTarget(player->Ball()->Pos());
}

//if the player has 'arrived' at the steering target he should wait and
//turn to face the ball
if (player->AtTarget())
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->TrackBall();
    player->SetVelocity(Vector2D(0,0));
}

void ComboBoxReceive::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetReceiver(NULL);
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

//************************************************************************
COMBODRIBBLEPASS
ComboDribblePassControl* ComboDribblePassControl::Instance()
{
    static ComboDribblePassControl instance;
    return &instance;
}

void ComboDribblePassControl::Enter(FieldPlayer* player)
{
// TODO: Implement
assert(0);

// keep track of combo states for use in context determination
player->setComboContext(PlayerBase::COMBO_DRIBBLE_PASS);

#ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboDribblePassControl state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
#endif
}

void ComboDribblePassControl::Execute(FieldPlayer* player)
{
    // ------------------------  First, check for state transitions
    // ------------------------  Next, carry out current state behavior

}

void ComboDribblePassControl::Exit(FieldPlayer* player)
{
    // keep track of combo states for use in context determination
    player->setComboContext(PlayerBase::NO_COMBO);

    // ------------------------  First, check for state transitions
    // ------------------------  Next, carry out current state behavior

}

ComboDribblePassSupport* ComboDribblePassSupport::Instance()
{
    static ComboDribblePassSupport instance;
    return &instance;
}

void ComboDribblePassSupport::Enter(FieldPlayer* player)
{
    // TODO: Implement
    assert(0);

    // keep track of combo states for use in context determination
    player->setComboContext(PlayerBase::COMBO_DRIBBLE_PASS);

    #ifdef PLAYER_STATE_INFO_ON
        debug_con << "Player " << player->ID() << " enters ComboDribblePassSupport state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
    #endif
}

void ComboDribblePassSupport::Execute(FieldPlayer* player)
{
    // ------------------------  First, check for state transitions
    // ------------------------  Next, carry out current state behavior

}
Next, carry out current state behavior

void ComboDribblePassSupport::Exit(FieldPlayer* player)
{
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

ComboDribblePassReceive* ComboDribblePassReceive::Instance()
{
    static ComboDribblePassReceive instance;

    return &instance;
}

void ComboDribblePassReceive::Enter(FieldPlayer* player)
{
    // TODO: Implement
    assert(0);

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_DRIBBLE_PASS);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboDribblePassReceive state" << " at pos " << player->Pos().x << "",
        " player->Pos().y << "";
    #endif
}

void ComboDribblePassReceive::Execute(FieldPlayer* player)
{
    // ------------------------  First, check for state transitions
    ---------------------------
    // ------------------------  Next, carry out current state
    behavior  ---------------------------

}

void ComboDribblePassReceive::Exit(FieldPlayer* player)
{
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

COMBOBOUNDING
ComboBoundingControl* ComboBoundingControl::Instance()
{
    static ComboBoundingControl instance;

    // ------------------------  First, check for state transitions
    ---------------------------
    // ------------------------  Next, carry out current state
    behavior  ---------------------------

}
return &instance;
}

void ComboBoundingControl::Enter(FieldPlayer* player)
{
    // let the team know this player is controlling
    player->Team()->SetControllingPlayer(player);

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_BOUNDING);

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboBoundingControl
state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "";
    #endif
}

void ComboBoundingControl::Execute(FieldPlayer* player)
{
    // calculate the dot product of the vector pointing to the ball
    // and the player's heading
    Vector2D ToBall = player->Ball()->Pos() - player->Pos();
    double dot = player->Heading().Dot(Vec2DNormalize(ToBall));

    // cannot kick the ball if the goalkeeper is in possession or if it is
    // behind the player or if there is already an assigned receiver. So just
    // continue chasing the ball
    if (player->Team()->Receiver() != NULL ||
        player->Pitch()->GoalKeeperHasBall() ||
        (dot < 0))
    {
        #ifdef PLAYER_STATE_INFO_ON
        debug_con << "Goaly has ball / ball behind player" << "";
        #endif

        player->GetFSM()->ChangeState(ChaseBall::Instance());
        return;
    }

    /*---------------------------------------- Attempt a shot at
    the goal ---------------------------------*/

    // if a shot is possible, this vector will hold the position along the
    // opponent's goal line the player should aim for.
    Vector2D BallTarget;

    // the dot product is used to adjust the shooting force. The more
    // directly the ball is ahead, the more forceful the kick
    double power = Prm.MaxShootingForce * dot;

    // if it is determined that the player could score a goal from this position
    if (player->Team() -> CanShoot(player->Ball() -> Pos(),
        /*-------------*/
power, BallTarget))
{
#ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " attempts a shot at " << BallTarget << "\n";
#endif

    // add some noise to the kick. We don't want players who are
    // too accurate! The amount of noise can be adjusted by altering
    // Prm.PlayerKickingAccuracy
    BallTarget = AddNoiseToKick(player->Ball()->Pos(), BallTarget);

    // this is the direction the ball will be kicked in
    Vector2D KickDirection = BallTarget - player->Ball()->Pos();
    player->Ball()->Kick(KickDirection, power);

    if(player->Team()->Color() == SoccerTeam::blue)
    {
        MetricData::IncrementBlueTeamGoalAttempts();
    }
    else
    {
        MetricData::IncrementRedTeamGoalAttempts();
    }

    // change state
    player->GetFSM()->ChangeState(Wait::Instance());
    player->FindSupport();
    return;
}

/*================================================================----------
 Attempt a pass to a player -----------------------------*/

// if a receiver is found this will point to it
PlayerBase* receiver = NULL;

    power = Prm.MaxPassingForce * dot;

    // test if the other attacker can receive a pass
    if (player->Team()->FindPassComboBounding(player,
        receiver, BallTarget, power, 5, 100))
    {
        Vector2D suppSpot = player->Team()->CalculateSupportSpotComboBounding(player->Pos(), receiver->Pos(), BallTarget);
        player->Team()->SetSupportSpotComboBounding(suppSpot);
Vector2D KickDirection = BallTarget - player->Ball()->Pos();

double const powerScale = .6;
player->Ball()->Kick(KickDirection, power * powerScale);

#ifdef PLAYER_STATE_INFO_ON
double power;
player->Ball()->Kick(KickDirection, power * powerScale);

debug_con << "Player " << player->ID() << " passes the ball with force " << power << " to player " << receiver->ID() << " Target is " << BallTarget; 

#endif

// let the receiver know a pass is coming
// and for CCxBR team the play is ComboBounding
if(player->Team()->Type() == SoccerTeam::CCxBR)
{
    Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY, 
        player->ID(), 
        receiver->ID(), 
        Msg_ReceiveBallComboBounding, 
        &BallTarget);
}
else
{
    Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY, 
        player->ID(), 
        receiver->ID(), 
        Msg_ReceiveBall, 
        &BallTarget);
}

//the player should attempt to continue the ComboBounding collaborative
player->GetFSM()->ChangeState(ComboBoundingSupport::Instance());

return;
}

void ComboBoundingControl::Exit(FieldPlayer* player)
{
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}
ComboBoundingSupport* ComboBoundingSupport::Instance()
{
    static ComboBoundingSupport instance;

    return &instance;
}

void ComboBoundingSupport::Enter(FieldPlayer* player)
{
    // let the team know this player is the supporting player
    player->Team()->SetSupportingPlayer(player);

    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::COMBO_BOUNDING);

    // use arrive to reach the combo support spot
    player->Steering()->ArriveOn();

    #ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboBoundingSupport state" << " at pos " << player->Pos().x << ", " << player->Pos().y << "\n;"
    #endif
}

void ComboBoundingSupport::Execute(FieldPlayer* player)
{
    // if his team loses control
    // he should change state to chase the ball
    if (!player->Team()->InControl())
    {
        player->GetFSM()->ChangeState(ChaseBall::Instance());
        player->GetFSM()->GetNameOfCurrentState();
        return;
    }

    Vector2D debugVector1 = player->Team()->GetSupportSpotComboBounding();
    player->Steering()->SetTarget(debugVector1);

    // if the player has 'arrived' at the steering target he should wait and
    // turn to face the ball - unless the play is broken (the other attacker no
    // longer in same combo)
    if (player->AtTarget())
    {
        player->Steering()->ArriveOff();
        player->Steering()->PursuitOff();
        player->TrackBall();
        player->SetVelocity(Vector2D(0,0));
    }
}

209
// find teammate and determine current context
std::vector<PlayerBase*>::const_iterator curPlyr = player->Team()->Members().begin();

// iterate through all this player's team members and find the other attacker,
// calculate if it is in a position to be passed the ball
for (curPlyr; curPlyr != player->Team()->Members().end(); ++curPlyr)
{
    // make sure the potential receiver being examined is not this player
    // and that it is the other attacker
    if ( (*curPlyr != player) && ( (*curPlyr)->Role() == PlayerBase::player_role::attacker))
    {
        if(player->Team()->Type() == SoccerTeam::CCxBR)  // for CCxBR direct context recognition
        {
            if((*curPlyr)->ComboContext() != PlayerBase::COMBO_BOUNDING)
            {
                player->GetFSM()->ChangeState(SupportAttacker::Instance());
            }
        }
    }
}

void ComboBoundingSupport::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetSupportingPlayer(NULL);
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

// ------------------------------------
ComboBoundingReceive* ComboBoundingReceive::Instance()
{
    static ComboBoundingReceive instance;
    return &instance;
}

void ComboBoundingReceive::Enter(FieldPlayer* player)
{
    // let the team know this player is receiving the ball
    player->Team()->SetReceiver(player);
    // this player is also now the controlling player
}

void ComboBoundingSupport::Exit(FieldPlayer* player)
player->Team()->SetControllingPlayer(player);

    // keep track of combo states for use in context determination
player->SetComboContext(PlayerBase::COMBO_BOUNDING);

    // pursue the ball
player->Steering()->PursuitOn();

#ifdef PLAYER_STATE_INFO_ON
    debug_con << "Player " << player->ID() << " enters ComboBoundingReceive
state" << " at pos " << player->Pos().x << ", " << player->Pos().y << ""
#endif
}

void ComboBoundingReceive::Execute(FieldPlayer* player)
{
    // ------------------------  First, check for state transitions
    // ------------------------

    //if the ball is within kicking range the player changes state to KickBall.
    if (player->BallWithinKickingRange())
    {
        player->GetFSM()->ChangeState(ComboBoundingControl::Instance());
        return;
    }

    //if player's team loses control he should change state to chase the ball
    if (!player->Team()->InControl())
    {
        player->GetFSM()->ChangeState(ChaseBall::Instance());
        return;
    }

    // ------------------------  Next, carry out current state behavior
    // ------------------------

    //if the ball comes close enough to the player he should pursuit the ball
    if (player->BallWithinReceivingRange())
    {
        player->Steering()->PursuitOn();
        return;
    }

    if (player->Steering()->PursuitIsOn())
    {
        player->Steering()->SetTarget(player->Ball()->Pos());
    }

    //if the player has 'arrived' at the steering target he should wait and
    //turn to face the ball
    if (player->AtTarget())
    {
        player->Steering()->ArriveOff();
    }
}
void ComboBoundingReceive::Exit(FieldPlayer* player)
{
    player->Steering()->ArriveOff();
    player->Steering()->PursuitOff();
    player->Team()->SetReceiver(NULL);
    // keep track of combo states for use in context determination
    player->SetComboContext(PlayerBase::NO_COMBO);
}

FieldPlayerStates.h

#ifndef FIELDPLAYERSTATES_H
#define FIELDPLAYERSTATES_H
//------------------------------------------------------------------------
// Name: FieldPlayerStates.h
// Desc: States for the field players of Simple Soccer. See my book
//       for detailed descriptions
// Author: Mat Buckland 2003 (fup@ai-junkie.com)
//------------------------------------------------------------------------
#include <string>
#include "FSM/State.h"
#include "Messaging/Telegram.h"
#include "constants.h"

class FieldPlayer;
class SoccerPitch;

//------------------------------------------------------------------------
class GlobalPlayerState : public State<FieldPlayer>
{
private:
    GlobalPlayerState();

public:
//this is a singleton
static GlobalPlayerState* Instance();
    
void Enter(FieldPlayer* player){}

void Execute(FieldPlayer* player);

void Exit(FieldPlayer* player){}

bool OnMessage(FieldPlayer*, const Telegram&);
};

//------------------------------------------------------------------------
class ChaseBall : public State<FieldPlayer>
{
    private:
        ChaseBall(){}

    public:
        //this is a singleton
        static ChaseBall* Instance();

    void Enter(FieldPlayer* player);

    void Execute(FieldPlayer* player);

    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//------------------------------------------------------------------------
class Dribble : public State<FieldPlayer>
{
    private:
        Dribble(){}

    public:
        //this is a singleton
        static Dribble* Instance();

    void Enter(FieldPlayer* player);

    void Execute(FieldPlayer* player);

    void Exit(FieldPlayer* player){}

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};
class ReturnToHomeRegion: public State<FieldPlayer>
{
private:

ReturnToHomeRegion();

public:

//this is a singleton
static ReturnToHomeRegion* Instance();

void Enter(FieldPlayer* player);

void Execute(FieldPlayer* player);

void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&){return false;}};

//---------------------------------------------

class Wait: public State<FieldPlayer>
{
private:

Wait();

public:

//this is a singleton
static Wait* Instance();

void Enter(FieldPlayer* player);

void Execute(FieldPlayer* player);

void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&){return false;}};

//---------------------------------------------

class KickBall: public State<FieldPlayer>
{
private:

KickBall();

public:

//this is a singleton
static KickBall* Instance();

void Enter(FieldPlayer* player);
void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player){}
bool OnMessage(FieldPlayer*, const Telegram&){return false;}

//------------------------------------------------------------------------
class ReceiveBall: public State<FieldPlayer>
{
private:
    ReceiveBall();

public:
    //this is a singleton
    static ReceiveBall* Instance();
    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
    void Exit(FieldPlayer* player);
    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//----------------------------------------------------------------------------
class SupportAttacker: public State<FieldPlayer>
{
private:
    SupportAttacker();

public:
    //this is a singleton
    static SupportAttacker* Instance();
    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
    void Exit(FieldPlayer* player);
    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//----------------------------------------------------------------------------
class ComboDiagonalControl : public State<FieldPlayer>
{ private:
const static int COMBO_DIA_SUPPORT_SPOT_DISTANCE = 50;

ComboDiagonalControl(){

public:

    //this is a singleton
    static ComboDiagonalControl* Instance();

    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//-------------------------------------------------------------
class ComboDiagonalSupport : public State<FieldPlayer>
{
private:

    ComboDiagonalSupport(){

public:

    //this is a singleton
    static ComboDiagonalSupport* Instance();

    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//-------------------------------------------------------------
class ComboDiagonalReceive : public State<FieldPlayer>
{
private:

    ComboDiagonalReceive(){

public:

    //this is a singleton
    static ComboDiagonalReceive* Instance();

    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&) { return false; }

// this is a singleton
static ComboBoxControl* Instance();

void Enter(FieldPlayer* player);
void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&) { return false; }

// this is a singleton
static ComboBoxSupport* Instance();

void Enter(FieldPlayer* player);
void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&) { return false; }

// this is a singleton
static ComboBoxReceive* Instance();

void Enter(FieldPlayer* player);
void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&) { return false; }
// this is a singleton
static ComboBoxReceive* Instance();

void Enter(FieldPlayer* player);

void Execute(FieldPlayer* player);

void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//----------------------------------------------------------------------------
class ComboDribblePassControl : public State<FieldPlayer>
{
private:

    ComboDribblePassControl(){

public:

    // this is a singleton
    static ComboDribblePassControl* Instance();

    void Enter(FieldPlayer* player);

    void Execute(FieldPlayer* player);

    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

//----------------------------------------------------------------------------
class ComboDribblePassSupport : public State<FieldPlayer>
{
private:

    ComboDribblePassSupport(){

public:

    // this is a singleton
    static ComboDribblePassSupport* Instance();

    void Enter(FieldPlayer* player);

    void Execute(FieldPlayer* player);

    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};
class ComboDribblePassReceive : public State<FieldPlayer>
{
    private:

    ComboDribblePassReceive();

    public:

    //this is a singleton
    static ComboDribblePassReceive* Instance();

    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

class ComboBoundingControl : public State<FieldPlayer>
{
    private:

    ComboBoundingControl();

    public:

    //this is a singleton
    static ComboBoundingControl* Instance();

    void Enter(FieldPlayer* player);
    void Execute(FieldPlayer* player);
    void Exit(FieldPlayer* player);

    bool OnMessage(FieldPlayer*, const Telegram&){return false;}
};

class ComboBoundingSupport : public State<FieldPlayer>
{
    private:

    ComboBoundingSupport();

    public:

    //this is a singleton
    static ComboBoundingSupport* Instance();

    void Enter(FieldPlayer* player);
void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&){return false;};

// this is a singleton
static ComboBoundingReceive* Instance();

void Enter(FieldPlayer* player);
void Execute(FieldPlayer* player);
void Exit(FieldPlayer* player);

bool OnMessage(FieldPlayer*, const Telegram&){return false;};
```cpp
int MetricData::redTeamComboDiagonalCollaboration = 0;
int MetricData::redTeamComboBoundingCollaboration = 0;
int MetricData::redTeamComboBoxCollaboration = 0;
int MetricData::blueTeamComboDiagonalCollaboration = 0;
int MetricData::blueTeamComboBoundingCollaboration = 0;
int MetricData::blueTeamComboBoxCollaboration = 0;

bool MetricData::redTeamLastControl = false;
bool MetricData::blueTeamLastControl = false;

Vector2D MetricData::lastBallPosition = Vector2D(0,0);
SoccerTeam::team_type MetricData::redTeamType = SoccerTeam::BASE;
SoccerTeam::team_type MetricData::blueTeamType = SoccerTeam::BASE;

MetricData::MetricData()
{
}

void MetricData::OutputResultsToFile()
{
    ofstream out;
    out.open("comboTest.data", ios::out | ios::app);
    assert(out.is_open());
    out << redTeamType << "\t" << redTeamScore << "\t" <<
        redTeamGoalAttempts << "\t" << redTeamBallDistance << "\t" <<
        redTeamComboDiagonalCollaboration << "\t" <<
        redTeamComboBoundingCollaboration << "\t" <<
        redTeamComboBoxCollaboration << "\t" <<
        redTeamPossessionCycles << "\n";
    out.close();
    // assert(0);
}

MetricData.h

#ifndef METRICDATA_H
#define METRICDATA_H
#pragma warning (disable:4786)

//------------------------------------------------------------------------
//
//  Name:   MetricData.h
//
#endif // METRICDATA_H
```
/ Desc: Used for tracking data and metrics for measuring collaboration between players
/ Author: Gil Barrett
/------------------------------------------------------------------------
#include "SoccerTeam.h"
#include "2D/Vector2D.h"

class MetricData
{
public:
  MetricData();

  void SetRedTeamType(SoccerTeam::team_type type){redTeamType = type;}
  void SetBlueTeamType(SoccerTeam::team_type type){blueTeamType = type;}
  SoccerTeam::team_type GetRedTeamType(){return redTeamType;}
  SoccerTeam::team_type GetBlueTeamType(){ return blueTeamType;}
  void static IncrementRedTeamGoalAttempts(){redTeamGoalAttempts++;}
  void static IncrementBlueTeamGoalAttempts(){blueTeamGoalAttempts++;}
  void IncrementRedTeamPossessionCycles(){redTeamPossessionCycles++;}
  void IncrementBlueTeamPossessionCycles(){blueTeamPossessionCycles++;}
  int GetRedTeamScore(){ return redTeamScore;}
  int GetBlueTeamScore(){ return blueTeamScore;}
  int GetRedTeamGoalAttempts(){ return redTeamGoalAttempts;}
  int GetBlueTeamGoalAttempts(){ return blueTeamGoalAttempts;}
  int GetRedTeamPossessionCycles();
  int GetBlueTeamPossessionCycles();
  void SetRedTeamScore(int score){redTeamScore = score;}
  void SetBlueTeamScore(int score){blueTeamScore = score;}
  void AddToRedTeamBallDistance(double dist){redTeamBallDistance += dist;}
  void AddToBlueTeamBallDistance(double dist){blueTeamBallDistance += dist;}
  double GetRedTeamBallDistance(){return redTeamBallDistance;}
  double GetBlueTeamBallDistance(){return blueTeamBallDistance;}
  void IncrementRedTeamComboDiagonalCollaboration(){redTeamComboDiagonalCollaboration++;}
void IncrementBlueTeamComboDiagonalCollaboration(){blueTeamComboDiagonalCollaboration++;}
void IncrementRedTeamComboBoundingCollaboration(){redTeamComboBoundingCollaboration++;}
void IncrementBlueTeamComboBoundingCollaboration(){blueTeamComboBoundingCollaboration++;}
void IncrementRedTeamComboBoxCollaboration(){redTeamComboBoxCollaboration++;}
void IncrementBlueTeamComboBoxCollaboration(){blueTeamComboBoxCollaboration++;}

int GetRedTeamComboDiagonalCollaboration(){ return redTeamComboDiagonalCollaboration;}
int GetBlueTeamComboDiagonalCollaboration(){ return blueTeamComboDiagonalCollaboration;}
int GetRedTeamComboBoundingCollaboration(){ return redTeamComboBoundingCollaboration;}
int GetBlueTeamComboBoundingCollaboration(){ return blueTeamComboBoundingCollaboration;}
int GetRedTeamComboBoxCollaboration(){ return redTeamComboBoxCollaboration;}
int GetBlueTeamComboBoxCollaboration(){ return blueTeamComboBoxCollaboration;}

void SetRedTeamLastControl(bool flag){redTeamLastControl = flag;}
void SetBlueTeamLastControl(bool flag){blueTeamLastControl = flag;}
bool GetRedTeamLastControl(){ return redTeamLastControl;}
bool GetBlueTeamLastControl(){ return blueTeamLastControl;}
Vector2D GetLastBallPosition(){ return lastBallPosition;}
void SetLastBallPosition(const Vector2D& pos){lastBallPosition = pos;}

void OutputResultsToFile();

private:
SoccerTeam::team_type static redTeamType;
SoccerTeam::team_type static blueTeamType;

int static redTeamScore;
int static blueTeamScore;
int static redTeamGoalAttempts;
int static blueTeamGoalAttempts;
int static redTeamPossessionCycles;
int static blueTeamPossessionCycles;

double static redTeamBallDistance;
double static blueTeamBallDistance;
int static redTeamComboDiagonalCollaboration;
int static redTeamComboBoundingCollaboration;
int static redTeamComboBoxCollaboration;
int static blueTeamComboDiagonalCollaboration;
int static blueTeamComboBoundingCollaboration;
int static blueTeamComboBoxCollaboration;
bool static redTeamLastControl;
bool static blueTeamLastControl;
Vector2D static lastBallPosition;

};
#endif

PlayerBase.cpp
#include "PlayerBase.h"
#include "SteeringBehaviors.h"
#include "2D/Transformations.h"
#include "2D/Geometry.h"
#include "misc/Cgdi.h"
#include "2D/C2DMatrix.h"
#include "Game/Region.h"
#include "ParamLoader.h"
#include "Messaging/MessageDispatcher.h"
#include "SoccerMessages.h"
#include "SoccerTeam.h"
#include "ParamLoader.h"
#include "Goal.h"
#include "SoccerBall.h"
#include "SoccerPitch.h"
#include "Debug.DebugConsole.h"

using std::vector;

//============================================================================
// dtor  
//============================================================================
PlayerBase::~PlayerBase()
{
    delete m_pSteering;
}

//============================================================================
// ctor 
//============================================================================
PlayerBase::PlayerBase(SoccerTeam* home_team,
    int home_region,
    Vector2D heading,
    Vector2D velocity,
    double mass,
    double max_force,
double max_speed,
double max_turn_rate,
double scale,
player_role role):

MovingEntity(home_team->Pitch()->GetRegionFromIndex(home_region)-
>Center(),
    scale*10.0,
    velocity,
    max_speed,
    heading,
    mass,
    Vector2D(scale,scale),
    max_turn_rate,
    max_force),
    m_pTeam(home_team),
    m_dDistSqToBall(MaxFloat),
    m_iHomeRegion(home_region),
    m_iDefaultRegion(home_region),
    m_PlayerRole(role),
    m_ComboContext(PlayerBase::NO_COMBO)
{

    //setup the vertex buffers and calculate the bounding radius
    const int NumPlayerVerts = 4;
    const Vector2D player[NumPlayerVerts] = {Vector2D(-3, 8),
        Vector2D(3,10),
        Vector2D(3,-10),
        Vector2D(-3,-8)};

    for (int vtx=0; vtx<NumPlayerVerts; ++vtx)
    {
        m_vecPlayerVB.push_back(player[vtx]);

        //set the bounding push_back to the length of the
        //greatest extent
        if (abs(player[vtx].x) > m_dBoundingRadius)
        {
            m_dBoundingRadius = abs(player[vtx].x);
        }

        if (abs(player[vtx].y) > m_dBoundingRadius)
        {
            m_dBoundingRadius = abs(player[vtx].y);
        }
    }

    //set up the steering behavior class
    m_pSteering = new SteeringBehaviors(this,
        m_pTeam->Pitch(),
        Ball());

    //a player's start target is its start position (because it's just waiting)
    m_pSteering->SetTarget(home_team->Pitch()->GetRegionFromIndex(home_region)-
        >Center());
void PlayerBase::TrackBall()
{
    RotateHeadingToFacePosition(Ball()->Pos());
}

void PlayerBase::TrackTarget()
{
    SetHeading(Vec2DNormalize(Steering()->Target() - Pos()));
}

bool SortByDistanceToOpponentsGoal(const PlayerBase* const p1,
    const PlayerBase* const p2)
{
    return (p1->DistToOppGoal() < p2->DistToOppGoal());
}

bool SortByReversedDistanceToOpponentsGoal(const PlayerBase* const p1,
    const PlayerBase* const p2)
{
    return (p1->DistToOppGoal() > p2->DistToOppGoal());
}

bool PlayerBase::PositionInFrontOfPlayer(Vector2D position) const
{
    Vector2D ToSubject = position - Pos();

    if (ToSubject.Dot(Heading()) > 0)
    {
        return true;
    }
    else
return false;

//------------------------- IsThreatened ---------------------------------
//
//  returns true if there is an opponent within this player's
//  comfort zone
// **************************************************************************
bool PlayerBase::isThreatened() const
{
    //check against all opponents to make sure none are within this
    //player's comfort zone
    std::vector<PlayerBase*>::const_iterator curOpp;
    curOpp = Team()->Opponents()->Members().begin();
    for (curOpp; curOpp != Team()->Opponents()->Members().end(); ++curOpp)
    {
        //calculate distance to the player. If dist is less than our
        //comfort zone, and the opponent is in front of the player, return true
        if (PositionInFrontOfPlayer((*curOpp)->Pos()) &&
            (Vec2DDistanceSq(Pos(), (*curOpp)->Pos()) < Prm.PlayerComfortZoneSq))
        {
            return true;
        }
    } // next opp

    return false;
}

//----------------------------- FindSupport ---------------------------------
//
//  determines the player who is closest to the SupportSpot and messages him
//  to tell him to change state to SupportAttacker
// **************************************************************************

void PlayerBase::FindSupport() const
{
    if (Team()->SupportingPlayer() == NULL)
    {
        PlayerBase* BestSupportPly = Team()->DetermineBestSupportingAttacker();
        Team()->SetSupportingPlayer(BestSupportPly);
        Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
                                  ID(),
                                  Team()->SupportingPlayer()->ID(),
                                  Msg_SupportAttacker,
                                  NULL);
    }
    PlayerBase* BestSupportPly = Team()->DetermineBestSupportingAttacker();
if the best player available to support the attacker changes, update
the pointers and send messages to the relevant players to update their
states

if (BestSupportPly && (BestSupportPly != Team()->SupportingPlayer()))
{
    if (Team()->SupportingPlayer())
    {
        Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
            ID(),
            Team()->SupportingPlayer()->ID(),
            Msg_GoHome,
            NULL);
    }
}

    Team()->SetSupportingPlayer(BestSupportPly);
    Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
        ID(),
        Team()->SupportingPlayer()->ID(),
        Msg_SupportAttacker,
        NULL);
}

//calculate distance to opponent's goal. Used frequently by the
passing//methods
double PlayerBase::DistToOppGoal() const
{
    return fabs(Pos().x - Team()->OpponentsGoal()->Center().x);
}
double PlayerBase::DistToHomeGoal() const
{
    return fabs(Pos().x - Team()->HomeGoal()->Center().x);
}

bool PlayerBase::isControllingPlayer() const
{ return Team()->ControllingPlayer()==this; }

bool PlayerBase::BallWithinKeeperRange() const
{ return (Vec2DDistanceSq(Pos(), Ball()->Pos()) < Prm.KeeperInBallRangeSq); }

bool PlayerBase::BallWithinReceivingRange() const
{ return (Vec2DDistanceSq(Pos(), Ball()->Pos()) < Prm.BallWithinReceivingRangeSq); }

bool PlayerBase::BallWithinKickingRange() const
bool PlayerBase::InHomeRegion() const
{
    if (m_PlayerRole == goal_keeper)
    {
        return Pitch()->GetRegionFromIndex(m_iHomeRegion)->Inside(Pos(),
                   Region::normal);
    }
    else
    {
        return Pitch()->GetRegionFromIndex(m_iHomeRegion)->Inside(Pos(),
                   Region::halfsize);
    }
}

bool PlayerBase::AtTarget() const
{
    return (Vec2DDistanceSq(Pos(), Steering()->Target()) <
            Prm.PlayerInTargetRangeSq);
}

bool PlayerBase::isClosestTeamMemberToBall() const
{
    return Team()->PlayerClosestToBall() == this;
}

bool PlayerBase::isClosestPlayerOnPitchToBall() const
{
    return isClosestTeamMemberToBall() &&
           (DistSqToBall() < Team()->Opponents()->ClosestDistToBallSq());
}

bool PlayerBase::InHotRegion() const
{
    return fabs(Pos().y - Team()->OpponentsGoal()->Center().y ) <
            Pitch()->PlayingArea()->Length()/3.0;
}

bool PlayerBase::isAheadOfAttacker() const
{
    return fabs(Pos().x - Team()->OpponentsGoal()->Center().x ) <
            fabs(Team()->ControllingPlayer()->Pos().x - Team()->OpponentsGoal()->
                   Center().x);
}

SoccerBall* const PlayerBase::Ball() const
{
    return Team()->Pitch()->Ball();
}
SoccerPitch* const PlayerBase::Pitch() const
{
    return Team()->Pitch();
}

const Region* const PlayerBase::HomeRegion() const
{
    return Pitch()->GetRegionFromIndex(m_iHomeRegion);
}

//----------------------------- SelectContext ------------------------
//
//  selects and returns a combo context
//  used to select the ComboXxxxxxReceive context, which attempts to continue
the collaboration process
//
//------------------------------------------------------------------------
--
PlayerBase::combo_context PlayerBase::SelectComboReceiveContext()
{
    combo_context comboContext;

    // get the other attacker
    std::vector<PlayerBase*>::const_iterator otherPlyr = Team()->Members().begin();

    // iterate through all this player's team members and find the other attacker,
    // calculate if it is in a position to be passed the ball
    for (otherPlyr; otherPlyr != Team()->Members().end(); ++otherPlyr)
    {
        // make sure the potential receiver being examined is not this player
        // and that it is the other attacker
        if ( (*otherPlyr != this) && (*otherPlyr)->Role() ==
            PlayerBase::player_role::attacker)
        {
            break;
        }
    }
    assert( (*otherPlyr)->Role() == PlayerBase::player_role::attacker );

double dist = Pos().Distance( (*otherPlyr)->Pos() );

    if( (dist < 40) &&
        (RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 18) ==
         PlayerBase::FRONT  ||
        RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 18) ==
         PlayerBase::REAR))
    {
        comboContext = PlayerBase::COMBO_BOUNDING;
    }

    else if( (dist < 80) &&
        (RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 50) ==
         PlayerBase::LEFT_FRONT ||
        RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 70) ==
         PlayerBase::LEFT_REAR))
    {
        comboContext = PlayerBase::COMBO_REAR;
    }

    return comboContext;
}
RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 50) == PlayerBase::RIGHT_FRONT))
{
    comboContext = PlayerBase::COMBO_BOX;
}
else if(RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 200) == PlayerBase::LEFT
    || RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 200) == PlayerBase::RIGHT)
{
    comboContext = PlayerBase::COMBO_DIAGONAL;
} else
{
    comboContext = PlayerBase::NO_COMBO;
}
}
return comboContext;

PlayerBase::combo_context PlayerBase::SelectComboControlContext()
{
    combo_context comboContext;

    std::vector<PlayerBase*>::const_iterator otherPlyr = Team()->Members().begin();

    for (otherPlyr; otherPlyr != Team()->Members().end(); ++otherPlyr)
    {
        //make sure the potential receiver being examined is not this player
        if ( (*otherPlyr != this) && ( (*otherPlyr)->Role() == PlayerBase::player_role::attacker))
        {
            break;
        }
    }
    assert( (*otherPlyr)->Role() == PlayerBase::player_role::attacker );

    relative_region relRegion = RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 20);
    double dist = Pos().Distance( (*otherPlyr)->Pos() );

    // get the other attacker
    std::vector<PlayerBase*>::const_iterator otherPlyr = Team()->Members().begin();

    //iterate through all this player's team members and find the other attacker,
    //calculate if it is in a position to be passed the ball
    for (otherPlyr; otherPlyr != Team()->Members().end(); ++otherPlyr)
    {
        //make sure the potential receiver being examined is not this player
        if ( (*otherPlyr != this) && ( (*otherPlyr)->Role() == PlayerBase::player_role::attacker))
        {
            break;
        }
    }
    assert( (*otherPlyr)->Role() == PlayerBase::player_role::attacker );

    relative_region relRegion = RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 20);
    double dist = Pos().Distance( (*otherPlyr)->Pos() );
if(Team()->Type() == SoccerTeam::BASE)
{
    comboContext = PlayerBase::NO_COMBO;
}
else // both CCxBR and CxCBwJIT select context the same way initially
{
    if( (dist < 80) &&
        (RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 50) ==
         PlayerBase::LEFT_REAR ||
        RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 50) ==
         PlayerBase::RIGHT_REAR))
    {
        comboContext = PlayerBase::COMBO_BOX;
    }
    else if( (dist < 40) &&
             (RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 15) ==
              PlayerBase::FRONT ||
             RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 15) ==
              PlayerBase::REAR))
    {
        comboContext = PlayerBase::COMBO_BOUNDING;
    }
    else if(RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 200) ==
              PlayerBase::LEFT ||
                RegionRelativeToFirst(Pos(), (*otherPlyr)->Pos(), 200) ==
                PlayerBase::RIGHT)
    {
        comboContext = PlayerBase::COMBO_DIAGONAL;
    }
    else
    {
        comboContext = PlayerBase::NO_COMBO;
    }
}

return comboContext;

PlayerBase::relative_region PlayerBase::RegionRelativeToFirst(Vector2D const &firstPos, Vector2D const &secondPos, double const corridorWidth)const
{
    // ray width is used to create a corridor for determining
    // FRONT,REAR,LEFT,RIGHT
    int rayWidth = corridorWidth;
    // establish direction to opponents goal
    // forward is relative to opponents goal and irrelevant to player
    // heading
    bool forwardIsPlusX = true;
    if(Team()->OpponentsGoal()->Center().x < 100)
    {
        forwardIsPlusX = false;
    }
// favor Left/Right over Front/Rear (in case of both corridors, side corridor returned)
// check Left and Right corridor
if(fabs(firstPos.x - secondPos.x) < rayWidth)
{
    if(forwardIsPlusX)
    {
        if(secondPos.y > firstPos.y)
        {
            return relative_region::RIGHT;
        }
        else
        {
            return relative_region::LEFT;
        }
    }
    else
    {
        if(secondPos.y > firstPos.y)
        {
            return relative_region::LEFT;
        }
        else
        {
            return relative_region::RIGHT;
        }
    }
}

// check Forward and Rear corridor
if(fabs(firstPos.y - secondPos.y) < rayWidth)
{
    if(forwardIsPlusX)
    {
        if(firstPos.x > secondPos.x)
        {
            return relative_region::REAR;
        }
        else
        {
            return relative_region::FRONT;
        }
    }
    else
    {
        if(firstPos.x > secondPos.x)
        {
            return relative_region::FRONT;
        }
        else
        {
            return relative_region::REAR;
        }
    }
}
// deal with all four remaining quadrants
if(secondPos.x > firstPos.x && secondPos.y > firstPos.y)
{
    if(forwardIsPlusX)
    {
        return relative_region::RIGHT_FRONT;
    }
    else
    {
        return relative_region::LEFT_REAR;
    }
}
else if(secondPos.x > firstPos.x && secondPos.y < firstPos.y)
{
    if(forwardIsPlusX)
    {
        return relative_region::LEFT_FRONT;
    }
    else
    {
        return relative_region::RIGHT_REAR;
    }
}
else if(secondPos.x < firstPos.x && secondPos.y > firstPos.y)
{
    if(forwardIsPlusX)
    {
        return relative_region::RIGHT_REAR;
    }
    else
    {
        return relative_region::LEFT_FRONT;
    }
}
else if(secondPos.x < firstPos.x && secondPos.y < firstPos.y)
{
    if(forwardIsPlusX)
    {
        return relative_region::LEFT_REAR;
    }
    else
    {
        return relative_region::RIGHT_FRONT;
    }
}

PlayerBase.h

#pragma warning (disable:4786)
#ifndef PLAYERBASE_H
#define PLAYERBASE_H
//------------------------------------------------------------------------
//  Name: PlayerBase.h

PlayerBase.h

#pragma warning (disable:4786)
#ifndef PLAYERBASE_H
#define PLAYERBASE_H
//------------------------------------------------------------------------
//  Name: PlayerBase.h

234
// // Desc: Definition of a soccer player base class. The player inherits // from the autolist class so that any player created will be // automatically added to a list that is easily accessible by any // other game objects. (mainly used by the steering behaviors and // player state classes) // // Author: Mat Buckland 2003 (fup@ai-junkie.com) // // ------------------------------------------------------------------------------------------------ 

#include <vector>
#include <string>
#include <cassert>
#include "misc/autolist.h"
#include "2D/Vector2D.h"
#include "Game/MovingEntity.h"

class SoccerTeam;
class SoccerPitch;
class SoccerBall;
class SteeringBehaviors;
class Region;

class PlayerBase : public MovingEntity,
                public AutoList<PlayerBase>
{

public:

    enum player_role{goal_keeper, attacker, defender};

    enum combo_context
    {
        COMBO_BOUNDING,
        COMBO_BOX,
        COMBO_DIAGONAL,
        COMBO_DRIBBLE_PASS,
        NO_COMBO,
        NULL_COMBO
    };

    enum relative_region
    {
        LEFT_FRONT,
        FRONT,
        RIGHT_FRONT,
        LEFT,
        RIGHT,
        LEFT_REAR,
        REAR,
        RIGHT_REAR
    };
protected:

    // this player's role in the team
    player_role          m_PlayerRole;

    // a pointer to this player's team
    SoccerTeam*          m_pTeam;

    // the steering behaviors
    SteeringBehaviors*   m_pSteering;

    // the region that this player is assigned to.
    int                  m_iHomeRegion;

    // the region this player moves to before kickoff
    int                  m_iDefaultRegion;

    // the distance to the ball (in squared-space). This value is queried
    // a lot so it's calculated once each time-step and stored here.
    double               m_dDistSqToBall;

    // the vertex buffer
    std::vector<Vector2D> m_vecPlayerVB;
    // the buffer for the transformed vertices
    std::vector<Vector2D> m_vecPlayerVBTrans;

    combo_context        m_ComboContext;

public:

    PlayerBase(SoccerTeam*       home_team,
                int             home_region,
                Vector2D        heading,
                Vector2D        velocity,
                double          mass,
                double          max_force,
                double          max_speed,
                double          max_turn_rate,
                double          scale,
                player_role     role);

    virtual ~PlayerBase();

    // returns true if there is an opponent within this player's
    // comfort zone
    bool isThreatened() const;

    // rotates the player to face the ball or the player's current target
    void TrackBall();
    void TrackTarget();
// this messages the player that is closest to the supporting spot to
// change state to support the attacking player
void        FindSupport()const;

// returns true if the ball can be grabbed by the goalkeeper
bool        BallWithinKeeperRange()const;

// returns true if the ball is within kicking range
bool        BallWithinKickingRange()const;

// returns true if a ball comes within range of a receiver
bool        BallWithinReceivingRange()const;

// returns true if the player is located within the boundaries
// of his home region
bool        InHomeRegion()const;

// returns true if this player is ahead of the attacker
bool        isAheadOfAttacker()const;

// returns true if a player is located at the designated support spot
bool        AtSupportSpot()const;

// returns true if the player is located at his steering target
bool        AtTarget()const;

// returns true if the player is the closest player in his team to
// the ball
bool        isClosestTeamMemberToBall()const;

// returns true if the point specified by 'position' is located in
// front of the player
bool        PositionInFrontOfPlayer(Vector2D position)const;

// returns true if the player is the closest player on the pitch to the ball
bool        isClosestPlayerOnPitchToBall()const;

// returns true if this player is the controlling player
bool        isControllingPlayer()const;

// returns true if the player is located in the designated 'hot region' --
// the area close to the opponent's goal
bool        InHotRegion()const;

player_role Role()const{return m_PlayerRole;}

double        DistSqToBall()const{return m_dDistSqToBall;}
void        SetDistSqToBall(double val){m_dDistSqToBall = val;}

// calculate distance to opponent's/home goal. Used frequently by the
// passing
// methods
double        DistToOppGoal()const;
double        DistToHomeGoal()const;
void SetDefaultHomeRegion() {m_iHomeRegion = m_iDefaultRegion;}
SoccerBall* const Ball() const;
SoccerPitch* const Pitch() const;
SteeringBehaviors* const Steering() const {return m_pSteering;}
const Region* const HomeRegion() const;
void SetHomeRegion(int NewRegion) {m_iHomeRegion = NewRegion;}
SoccerTeam* const Team() const {return m_pTeam;}
void SetComboContext(const combo_context context) {m_ComboContext = context;}
combo_context ComboContext() const {return m_ComboContext;}
combo_context SelectComboControlContext();
combo_context SelectComboReceiveContext();
relative_region RegionRelativeToFirst(Vector2D const &firstPos, Vector2D const &secondPos, double const corridorWidth) const;

#endif

SoccerMessages.cpp
#include "SoccerMessages.h"

inline std::string MessageToString(int msg) {
    switch (msg) {
    case Msg_ReceiveBall:
        return "Msg_ReceiveBall";
    case Msg_ReceiveBallComboDiagonal:
        return "Msg_ReceiveBallComboDiagonal";
    case Msg_ReceiveBallComboBox:
        return "Msg_ReceiveBallComboBox";
    case Msg_ReceiveBallComboDribblePass:
        return "Msg_ReceiveBallComboDribblePass";
    case Msg_ReceiveBallComboBounding:
        return "Msg_ReceiveBallComboBounding";
    }
}
SoccerMessages.h

#ifndef SOCCER_MESSAGES_H
#define SOCCER_MESSAGES_H

#include <string>

enum MessageType
{
    Msg_ReceiveBall,
    Msg_ReceiveBallComboDiagonal,
    Msg_ReceiveBallComboBox,
    Msg_ReceiveBallComboDribblePass,
    Msg_ReceiveBallComboBounding,
    Msg_PassToMe,
    Msg_SupportAttacker,
    Msg_GoHome,
    Msg_Wait
};

//converts an enumerated value to a string
inline std::string MessageToString(int msg);

#endif

SoccerPitch.cpp
const int NumRegionsHorizontal = 6;
const int NumRegionsVertical   = 3;

//------------------------------- ctor -----------------------------------
//------------------------------------------------------------------------
SoccerPitch::SoccerPitch(int cx, int cy):m_cxClient(cx),
    m_cyClient(cy),
    m_bPaused(false),
    m_bGoalKeeperHasBall(false),

    m_Regions(NumRegionsHorizontal*NumRegionsVertical),
    m_bGameOn(true)
{
    //define the playing area
    m_pPlayingArea = new Region(20, 20, cx-20, cy-20);

    //create the regions
    CreateRegions(PlayingArea()->Width() / (double)NumRegionsHorizontal,
        PlayingArea()->Height() / (double)NumRegionsVertical);

    //create the goals
    m_pRedGoal  = new Goal(Vector2D( m_pPlayingArea->Left(), (cy-
        Prm.GoalWidth)/2),
        Vector2D(1,0));

    m_pBlueGoal = new Goal( Vector2D( m_pPlayingArea->Right(), (cy-
        Prm.GoalWidth)/2),
        Vector2D(-1,0));

    //create the soccer ball
m_pBall = new SoccerBall(Vector2D((double)m_cxClient/2.0, (double)m_cyClient/2.0),
                        Prm.BallSize,
                        Prm.BallMass,
                        m_vecWalls);

//create the teams
m_pRedTeam  = new SoccerTeam(m_pRedGoal, m_pBlueGoal, this,
                               SoccerTeam::red, SoccerTeam::BASE);
//create the walls
m_pBlueTeam = new SoccerTeam(m_pBlueGoal, m_pRedGoal, this,
                               SoccerTeam::blue, SoccerTeam::BASE);

//make sure each team knows who their opponents are
m_pRedTeam->SetOpponents(m_pBlueTeam);
//create the walls
m_pBlueTeam->SetOpponents(m_pRedTeam);

//create the walls
Vector2D TopLeft(m_pPlayingArea->Left(), m_pPlayingArea->Top());
Vector2D TopRight(m_pPlayingArea->Right(), m_pPlayingArea->Top());
Vector2D BottomRight(m_pPlayingArea->Right(), m_pPlayingArea->Bottom());
Vector2D BottomLeft(m_pPlayingArea->Left(), m_pPlayingArea->Bottom());

m_vecWalls.push_back(Wall2D(BottomLeft, m_pRedGoal->RightPost()));
m_vecWalls.push_back(Wall2D(m_pRedGoal->LeftPost(), TopLeft));
m_vecWalls.push_back(Wall2D(TopLeft, TopRight));
m_vecWalls.push_back(Wall2D(TopRight, m_pBlueGoal->LeftPost()));
m_vecWalls.push_back(Wall2D(m_pBlueGoal->RightPost(), BottomRight));
m_vecWalls.push_back(Wall2D(BottomRight, BottomLeft));

ParamLoader* p = ParamLoader::Instance();

SoccerPitch::~SoccerPitch()
{
    delete m_pBall;
    delete m_pRedTeam;
    delete m_pBlueTeam;
    delete m_pRedGoal;
    delete m_pBlueGoal;
    delete m_pPlayingArea;

    for (unsigned int i=0; i<m_Regions.size(); ++i)
    {
        delete m_Regions[i];
    }
}

//----------------------------- Update -----------------------------------
void SoccerPitch::Update()
{
    if (m_bPaused) return;

    static int tick = 0;

    // update the balls
    m_pBall->Update();

    // update the teams
    m_pRedTeam->Update();
    m_pBlueTeam->Update();

    // if a goal has been detected reset the pitch ready for kickoff
    if (m_pBlueGoal->Scored(m_pBall) || m_pRedGoal->Scored(m_pBall))
    {
        m_bGameOn = false;

        // reset the ball
        m_pBall->PlaceAtPosition(Vector2D((double)m_cxClient/2.0,
                                         (double)m_cyClient/2.0));

        // get the teams ready for kickoff
        m_pRedTeam->GetFSM()->ChangeState(PrepareForKickOff::Instance());
        m_pBlueTeam->GetFSM()->ChangeState(PrepareForKickOff::Instance());
    }

    PrintTestData();
}

// ------------------------ Testing Output -------------------------------
// output test data indicating which state attackers are in, for teams using
// ComboXxxxxxContexts
// ------------------------ Testing Output -------------------------------
void SoccerPitch::PrintTestData()
{
    static int updateCounter = 0;
    PlayerBase::combo_context cntxtAttacker1 = PlayerBase::NULL_COMBO;
    PlayerBase::combo_context cntxtAttacker2 = PlayerBase::NULL_COMBO;

    if(m_pRedTeam->Type() != SoccerTeam::BASE)
    {
        std::vector<PlayerBase*>::const_iterator plyr = m_pRedTeam->Members().begin();
        cntxtAttacker1 = PlayerBase::NULL_COMBO;
        cntxtAttacker2 = PlayerBase::NULL_COMBO;
        for (plyr; plyr != m_pRedTeam->Members().end(); ++plyr)
        {
            // find the attackers
            if( (*plyr)->Role() == PlayerBase::player_role::attacker )
            {
                if(cntxtAttacker1 == PlayerBase::NULL_COMBO)

```cpp
{  
cntxtAttacker1 = (*plyr)->ComboContext();  
} else  
{  
cntxtAttacker2 = (*plyr)->ComboContext();  }
if(cntxtAttacker1 == cntxtAttacker2)
{
switch(cntxtAttacker2)
{
case PlayerBase::COMBO_BOUNDING:
{  
metricData->IncrementRedTeamComboBoundingCollaboration();
break;
}
case PlayerBase::COMBO_BOX:
{  
metricData->IncrementRedTeamComboBoxCollaboration();
break;
}
case PlayerBase::COMBO_DIAGONAL:
{  
metricData->IncrementRedTeamComboDiagonalCollaboration();
break;
}
case PlayerBase::COMBO_DRIBBLE_PASS:
{  
// out << PlayerBase::COMBO_DRIBBLE_PASS << "\t";
break;
}
case PlayerBase::NO_COMBO:
{  
// "\t";
break;
}
}
}
}
if(m_pBlueTeam->Type() != SoccerTeam::BASE)
{
std::vector<PlayerBase*>::const_iterator plyr = m_pBlueTeam->Members().begin();
cntxtAttacker1 = PlayerBase::NULL_COMBO;
cntxtAttacker2 = PlayerBase::NULL_COMBO;
for (plyr; plyr != m_pBlueTeam->Members().end(); ++plyr)
{

```
if( (*plyr)->Role() == PlayerBase::player_role::attacker )
{
    if(cntxtAttacker1 == PlayerBase::NULL_COMBO)
    {
        cntxtAttacker1 = (*plyr)->ComboContext();
    }
    else
    {
        cntxtAttacker2 = (*plyr)->ComboContext();
    }
    if(cntxtAttacker1 == cntxtAttacker2)
    {
        switch(cntxtAttacker2)
        {
            case PlayerBase::COMBO_BOUNDING:
            {
                metricData->IncrementBlueTeamComboBoundingCollaboration();
                break;
            }
            case PlayerBase::COMBO_BOX:
            {
                metricData->IncrementBlueTeamComboBoxCollaboration();
                break;
            }
            case PlayerBase::COMBO_DIAGONAL:
            {
                metricData->IncrementBlueTeamComboDiagonalCollaboration();
                break;
            }
            case PlayerBase::COMBO_DRIBBLE_PASS:
            {
                out << PlayerBase::COMBO_DRIBBLE_PASS << " \t";
                break;
            }
            case PlayerBase::NO_COMBO:
            {
                "\t";
                break;
            }
        }
    }
}
if(m_pRedTeam->InControl())
{
    metricData->IncrementRedTeamPossessionCycles();
    if(metricData->GetRedTeamLastControl())
    {
metricData->AddToRedTeamBallDistance(Ball()->Pos().Distance(metricData->GetLastBallPosition()));
}
metricData->SetRedTeamLastControl(true);
metricData->SetBlueTeamLastControl(false);
}
else if(m_pBlueTeam->InControl())
{
    metricData->IncrementBlueTeamPossessionCycles();
    if(metricData->GetBlueTeamLastControl())
    {
        metricData->AddToBlueTeamBallDistance(Ball()->Pos().Distance(metricData->GetLastBallPosition()));
    }
    metricData->SetRedTeamLastControl(false);
    metricData->SetBlueTeamLastControl(true);
}
else
{
    metricData->SetRedTeamLastControl(false);
    metricData->SetBlueTeamLastControl(false);
}
metricData->SetLastBallPosition(Ball()->Pos());

updateCounter++;
// 108000 at 60 frames/sec = 30 mins
if(updateCounter > 108000)
{
    metricData->SetRedTeamType(m_pRedTeam->Type());
    metricData->SetBlueTeamType(m_pBlueTeam->Type());
    metricData->SetRedTeamScore(m_pRedTeam->OpponentsGoal()->NumGoalsScored());
    metricData->SetBlueTeamScore(m_pBlueTeam->OpponentsGoal()->NumGoalsScored());

    metricData->OutputResultsToFile();
    assert(0);
}

void SoccerPitch::CreateRegions(double width, double height)
{
    //index into the vector
    int idx = m_Regions.size()-1;

    for (int col=0; col<NumRegionsHorizontal; ++col)
    {
        for (int row=0; row<NumRegionsVertical; ++row)
        {
            m_Regions[idx--] = new Region(PlayingArea()->Left()+col*width,
                                          PlayingArea()->Top()+row*height,
                                          width,
                                          height);
        }
    }
}
PlayingArea()->Left()+(col+1)*width,
PlayingArea()->Top()+(row+1)*height,
idx);
}
}

//------------------------------- Render -------------------------------
//------------------------------------------------------------------------
bool SoccerPitch::Render()
{
    //draw the grass
    gdi->DarkGreenPen();
gdi->DarkGreenBrush();
gdi->Rect(0,0,m_cxClient, m_cyClient);

    //render regions
    if (Prm.bRegions)
    {
        for (unsigned int r=0; r<m_Regions.size(); ++r)
        {
            m_Regions[r]->Render(true);
        }
    }

    //render the goals
    gdi->HollowBrush();
gdi->RedPen();
gdi->Rect(m_pPlayingArea->Left(), (m_cyClient-Prm.GoalWidth)/2,
             m_pPlayingArea->Left()+40, m_cyClient - (m_cyClient-Prm.GoalWidth)/2);
gdi->BluePen();
gdi->Rect(m_pPlayingArea->Right(), (m_cyClient-Prm.GoalWidth)/2,
            m_pPlayingArea->Right()-40, m_cyClient - (m_cyClient-Prm.GoalWidth)/2);

    //render the pitch markings
    gdi->WhitePen();
gdi->Circle(m_pPlayingArea->Center(), m_pPlayingArea->Width() * 0.125);
gdi->Line(m_pPlayingArea->Center().x, m_pPlayingArea->Top(),
          m_pPlayingArea->Center().x, m_pPlayingArea->Bottom());
gdi->WhiteBrush();
gdi->Circle(m_pPlayingArea->Center(), 2.0);

    //the ball
    gdi->WhitePen();
gdi->WhiteBrush();
m_pBall->Render();

    //Render the teams
    m_pRedTeam->Render();
m_pBlueTeam->Render();

    //render the walls
gdi->WhitePen();
for (unsigned int w=0; w<m_vecWalls.size(); ++w)
{
    m_vecWalls[w].Render();
}

// show the score
    gdi->TextColor(Cgdi::red);
    gdi->TextAtPos((m_cxClient/2)-50, m_cyClient-18, "Red: " + ttos(m_pBlueGoal->NumGoalsScored()));

    gdi->TextColor(Cgdi::blue);
    gdi->TextAtPos((m_cxClient/2)+10, m_cyClient-18, "Blue: " + ttos(m_pRedGoal->NumGoalsScored()));
    return true;
}

SoccerPitch.h

#ifndef SOCCERPITCH_H
#define SOCCERPITCH_H
#pragma warning (disable:4786)
//------------------------------------------------------------------------
//
//  Name:   SoccerPitch.h
//
//  Desc:   A SoccerPitch is the main game object. It owns instances of
//          two soccer teams, two goals, the playing area, the ball
//          etc. This is the root class for all the game updates and
//          renders etc
//
//  Author: Mat Buckland 2003 (fup@ai-junkie.com)
//
//------------------------------------------------------------------------
#include <windows.h>
#include <vector>
#include <cassert>

#include "2D/Wall2D.h"
#include "2D/Vector2D.h"
#include "constants.h"

class Region;
class Goal;
class SoccerTeam;
class SoccerBall;
class SoccerTeam;
class PlayerBase;
class MetricData;

class SoccerPitch
{
public:

MetricData* metricData;
SoccerBall* m_pBall;
SoccerTeam* m_pRedTeam;
SoccerTeam* m_pBlueTeam;
Goal* m_pRedGoal;
Goal* m_pBlueGoal;

//container for the boundary walls
std::vector<Wall2D> m_vecWalls;

//defines the dimensions of the playing area
Region* m_pPlayingArea;

//the playing field is broken up into regions that the team
//can make use of to implement strategies.
std::vector<Region*> m_Regions;

//true if a goal keeper has possession
bool m_bGoalKeeperHasBall;

//true if the game is in play. Set to false whenever the players
//are getting ready for kickoff
bool m_bGameOn;

//set true to pause the motion
bool m_bPaused;

//local copy of client window dimensions
int m_cxClient,
m_cyClient;

//this instantiates the regions the players utilize to position
//themselves
void CreateRegions(double width, double height);

public:

SoccerPitch(int cxClient, int cyClient);
~SoccerPitch();
void Update();
bool Render();
void TogglePause(){m_bPaused = !m_bPaused;}
bool Paused() const{return m_bPaused;}
int cxClient() const{return m_cxClient;}

248
int    cyClient() const { return m_cyClient; }

bool   GoalKeeperHasBall() const { return m_bGoalKeeperHasBall; }
void   SetGoalKeeperHasBall(bool b) { m_bGoalKeeperHasBall = b; }

const Region* const     PlayingArea() const { return m_pPlayingArea; }
const std::vector<Wall2D>& Walls() { return m_vecWalls; }
SoccerBall* const        Ball() const { return m_pBall; }

const Region* const GetRegionFromIndex(int idx)
{
    assert ((idx > 0) && (idx < m_Regions.size()));

    return m_Regions[idx];
}

bool   GameOn() const { return m_bGameOn; }
void   SetGameOn() { m_bGameOn = true; }
void   SetGameOff() { m_bGameOn = false; }

private:
    void PrintTestData();
};

#endif

SoccerTeam.cpp
#include "SoccerTeam.h"
#include "SoccerPitch.h"
#include "Goal.h"
#include "PlayerBase.h"
#include "GoalKeeper.h"
#include "FieldPlayer.h"
#include "misc/utils.h"
#include "SteeringBehaviors.h"
#include "GoalKeeperStates.h"
#include "ParamLoader.h"
#include "2D/geometry.h"
#include "Game/EntityManager.h"
#include "Messaging/MessageDispatcher.h"
#include "SoccerMessages.h"
#include "TeamStates.h"
#include "Debug/DebugConsole.h"
#include <windows.h>

using std::vector;

//===--------------------------------- ctor ----------------------------------===
//===---------------------------------  ===
Goal*        opponents_goal,
SoccerPitch* pitch,
team_color   color,
team_type
m_pOpponentsGoal(opponents_goal),
    m_pHomeGoal(home_goal),
m_pOpponents(NULL),
m_pPitch(pitch),
m_Color(color),
    m_Type(type),
m_dDistSqToBallOfClosestPlayer(0.0),
m_pSupportingPlayer(NULL),
m_pReceivingPlayer(NULL),
m_pControllingPlayer(NULL),
m_pPlayerClosestToBall(NULL)
{
    //setup the state machine
    m_pStateMachine = new StateMachine<SoccerTeam>(this);
    m_pStateMachine->SetCurrentState(Defending::Instance());
    m_pStateMachine->SetPreviousState(Defending::Instance());
    m_pStateMachine->SetGlobalState(NULL);

    //create the players and goalkeeper
    CreatePlayers();

    //set default steering behaviors
    std::vector<PlayerBase*>::iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    { (*it)->Steering()->SeparationOn();
    }

    //create the sweet spot calculator
    m_pSupportSpotCalc = new SupportSpotCalculator(Prm.NumSupportSpotsX,
                                                                    Prm.NumSupportSpotsY,
                                                                    this);
}

//---------------------- dtor ------------------------------
//------------------------------------------------------------------------
SoccerTeam::~SoccerTeam()
{
    delete m_pStateMachine;
    std::vector<PlayerBase*>::iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    { delete *it;
    }
    delete m_pSupportSpotCalc;
```cpp
void SoccerTeam::Update()
{
    //this information is used frequently so it's more efficient to
    //calculate it just once each frame
    CalculateClosestPlayerToBall();

    //the team state machine switches between attack/defense behavior. It
    //also handles the 'kick off' state where a team must return to their
    //kick off positions before the whistle is blown
    m_pStateMachine->Update();

    //now update each player
    std::vector<PlayerBase*>::iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        (*it)->Update();
    }
}

void SoccerTeam::CalculateClosestPlayerToBall()
{
    double ClosestSoFar = MaxFloat;
    std::vector<PlayerBase*>::iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        //calculate the dist. Use the squared value to avoid sqrt
        double dist = Vec2DDistanceSq((*it)->Pos(), Pitch()->Ball()->Pos());

        //keep a record of this value for each player
        (*it)->SetDistSqToBall(dist);
        if (dist < ClosestSoFar)
        {
            ClosestSoFar = dist;
            m_pPlayerClosestToBall = *it;
        }
    }
}
```
m_dDistSqToBallOfClosestPlayer = ClosestSoFar;
}

//------------------- DetermineBestSupportingAttacker ------------------------
//
// calculate the closest player to the SupportSpot
//-------------------
PlayerBase* SoccerTeam::DetermineBestSupportingAttacker()
{
    double ClosestSoFar = MaxFloat;
    PlayerBase* BestPlayer = NULL;

    std::vector<PlayerBase*>::iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        //only attackers utilize the BestSupportingSpot
        if ( ((*it)->Role() == PlayerBase::attacker) && ((*it) != m_pControllingPlayer) )
        {
            //calculate the dist. Use the squared value to avoid sqrt
            double dist = Vec2DDistanceSq((*it)->Pos(), m_pSupportSpotCalc->GetBestSupportingSpot());

            //if the distance is the closest so far and the player is not a
            //goalkeeper and the player is not the one currently controlling
            //the ball, keep a record of this player
            if ((dist < ClosestSoFar) )
            {
                ClosestSoFar = dist;
                BestPlayer = (*it);
            }
        }
    }

    return BestPlayer;
}

//-------------------------- FindPass ------------------------------
//
//  The best pass is considered to be the pass that cannot be intercepted
//  by an opponent and that is as far forward of the receiver as possible
//--------------------------
bool SoccerTeam::FindPass(const PlayerBase*const passer,
                          PlayerBase*&           receiver,
                          Vector2D&              PassTarget,
                          double                  power,
                          double                  MinPassingDistance)const
{

    std::vector<PlayerBase*>::const_iterator curPlyr = Members().begin();

    return true;
double ClosestToGoalSoFar = MaxFloat;
Vector2D Target;

//iterate through all this player's team members and calculate which
//one is in a position to be passed the ball
for (curPlyr; curPlyr != Members().end(); ++curPlyr)
{
    //make sure the potential receiver being examined is not this player
    //and that it is further away than the minimum pass distance
    if ( (*curPlyr != passer) &&
        (Vec2DDistanceSq(passer->Pos(), (*curPlyr)->Pos()) >
         MinPassingDistance*MinPassingDistance))
    {
        if (GetBestPassToReceiver(passer, *curPlyr, Target, power))
        {
            //if the pass target is the closest to the opponent's goal line found
            // so far, keep a record of it
            double Dist2Goal = fabs(Target.x - OpponentsGoal()->Center().x);

            if (Dist2Goal < ClosestToGoalSoFar)
            {
                ClosestToGoalSoFar = Dist2Goal;

                //keep a record of this player
                receiver = *curPlyr;

                //and the target
                PassTarget = Target;
            }
        }
    }

    //next team member
}

if (receiver) return true;
else return false;

//-- FindPassComboDiagonal --

bool SoccerTeam::FindPassComboDiagonal(const PlayerBase*const passer,
                                    PlayerBase*& receiver,
                                    Vector2D& PassTarget,
                                    double power,
                                    double MinYDistance,
                                    double MaxDistBehindPasser) const
{
    std::vector<PlayerBase*>::const_iterator curPlyr = Members().begin();
double ClosestToGoalSoFar = MaxFloat;
Vector2D Target;

// iterate through all this player's team members and find the other attacker,
// calculate if it is in a position to be passed the ball
for (curPlyr; curPlyr != Members().end(); ++curPlyr)
{
    // make sure the potential receiver being examined is not this player
    // and that it is the other attacker
    if ( (*curPlyr != passer) &&  (*curPlyr)->Role() ==
        PlayerBase::player_role::attacker)
    {
        if (GetBestPassToReceiverComboDiagonal(passer, *curPlyr, Target,
                                                    power))
        {
            // if the pass target is the closest to the opponent's goal line found
            // so far, keep a record of it
            double Dist2Goal = fabs(Target.x - OpponentsGoal()->Center().x);

            if (Dist2Goal < ClosestToGoalSoFar)
            {
                ClosestToGoalSoFar = Dist2Goal;

                // keep a record of this player
                receiver = *curPlyr;

                // and the target
                PassTarget = Target;
            }
        }
    }
}
}

bool SoccerTeam::FindPassComboBox(const PlayerBase*const passer,
                                   PlayerBase*& receiver,
                                   Vector2D& PassTarget,
                                   double power,
                                   double MinYDistance,
                                   double MaxDistBehindPasser) const
{

    std::vector<PlayerBase*>::const_iterator curPlyr = Members().begin();

    Vector2D Target;

    // iterate through all this player's team members and find the other attacker,
    // calculate if it is in a position to be passed the ball
    for (curPlyr; curPlyr != Members().end(); ++curPlyr)
{  
    // make sure the potential receiver being examined is not this player  
    // and that it is the other attacker  
    if ( (*curPlyr != passer) &&  ( (*curPlyr)->Role() ==  
        PlayerBase::player_role::attacker) )  
    {  
        if (GetBestPassToReceiverComboBox(passer, *curPlyr, Target, power))  
        {  
            // keep a record of this player  
            receiver = *curPlyr;  
            // and the target  
            PassTarget = Target;  
        }  
    }  
}  
} // next team member

if (receiver) return true;
else return false;

bool SoccerTeam::FindPassComboDribblePass(const PlayerBase* const passer, 
    PlayerBase*& receiver, 
    Vector2D& PassTarget, 
    double power, 
    double MinYDistance, 
    double MaxDistBehindPasser) const  
{
    // Todo: implement
    assert(0);
    return true;
}

bool SoccerTeam::FindPassComboBounding(const PlayerBase* const passer, 
    PlayerBase*& receiver, 
    Vector2D& PassTarget, 
    double power, 
    double MinYDistance, 
    double MaxDistBehindPasser) const  
{
    std::vector<PlayerBase>::const_iterator curPlyr = Members().begin();

    double ClosestToGoalSoFar = MaxFloat;
    Vector2D Target;

    // iterate through all this player's team members and find the other attacker,
    // calculate if it is in a position to be passed the ball
    for (curPlyr; curPlyr != Members().end(); ++curPlyr)  
    {  
        // make sure the potential receiver being examined is not this player
        // and that it is the other attacker
        if ( (*curPlyr != passer) &&  ( (*curPlyr)->Role() ==  
            PlayerBase::player_role::attacker))
if (GetBestPassToReceiverComboBounding(passer, *curPlyr, Target, power))
{
    //if the pass target is the closest to the opponent's goal line found
    // so far, keep a record of it
    double Dist2Goal = fabs(Target.x - OpponentsGoal()->Center().x);
    if (Dist2Goal < ClosestToGoalSoFar)
    {
        ClosestToGoalSoFar = Dist2Goal;
        //keep a record of this player
        receiver = *curPlyr;
        //and the target
        PassTarget = Target;
    }
}
}  //next team member

if (receiver) return true;
else return false;

//---------------------- GetBestPassToReceiver ---------------------------
//
// Three potential passes are calculated. One directly toward the receiver's
// current position and two that are the tangents from the ball position
// to the circle of radius 'range' from the receiver.
// These passes are then tested to see if they can be intercepted by an
// opponent and to make sure they terminate within the playing area. If
// all the passes are invalidated the function returns false. Otherwise
// the function returns the pass that takes the ball closest to the
// opponent's goal area.
//------------------------------------------------------------------------

bool SoccerTeam::GetBestPassToReceiver(const PlayerBase* const passer,
    const PlayerBase* const receiver,
    Vector2D&               PassTarget,
    double                   power)const
{
    //first, calculate how much time it will take for the ball to reach
    //this receiver, if the receiver was to remain motionless
    double time = Pitch()->Ball()->TimeToCoverDistance(Pitch()->Ball()->Pos(),
        receiver->Pos(),
        power);

    //return false if ball cannot reach the receiver after having been
    //kicked with the given power
    if (time < 0) return false;

    //the maximum distance the receiver can cover in this time
double InterceptRange = time * receiver->MaxSpeed();

// Scale the intercept range
const double ScalingFactor = 0.3;
InterceptRange *= ScalingFactor;

// now calculate the pass targets which are positioned at the intercepts
// of the tangents from the ball to the receiver's range circle.
Vector2D ip1, ip2;
GetTangentPoints(receiver->Pos(),
    InterceptRange,
    Pitch() -> Ball()->Pos(),
    ip1,
ip2);

const int NumPassesToTry = 3;
Vector2D Passes[NumPassesToTry] = {ip1, receiver->Pos(), ip2};

// this pass is the best found so far if it is:
// 1. Further upfield than the closest valid pass for this receiver
//    found so far
// 2. Within the playing area
// 3. Cannot be intercepted by any opponents

double ClosestSoFar = MaxFloat;
bool  bResult      = false;

for (int pass=0; pass<NumPassesToTry; ++pass)
{
    double dist = fabs(Passes[pass].x - OpponentsGoal()->Center().x);

    if ((dist < ClosestSoFar) &&
        Pitch()->PlayingArea()->Inside(Passes[pass]) &&
        isPassSafeFromAllOpponents(Pitch()->Ball()->Pos(),
                                    Passes[pass],
                                    receiver,
                                    power))
    {
        ClosestSoFar = dist;
        PassTarget   = Passes[pass];
        bResult      = true;
    }
}

return bResult;
}

//---------------------------------------------------------------------
GetBestPassToReceiverComboDiagonal

// Five potential passes are calculated. One directly toward the receiver's
// current position and two that are between the rcvr and the opponents
goal, along the rcvrs y-line
// and within the circle of radius 'range' from the receiver.
// These passes are then tested to see if they can be intercepted by an
// opponent and to make sure they terminate within the playing area. If
// all the passes are invalidated the function returns false. Otherwise
// the function returns the pass that takes the ball closest to the
// opponent's goal area.
//------------------------------------------------------------------------
bool SoccerTeam::GetBestPassToReceiverComboDiagonal(const PlayerBase* const
passer,
const PlayerBase* const receiver,
Vector2D& PassTarget,
double power)const
{
    double time = Pitch()->Ball()->TimeToCoverDistance(Pitch()->Ball()->Pos(),
receiver->Pos(),
power);
    if (time < 0) return false;
    double InterceptRange = time * receiver->MaxSpeed();
    const double ScalingFactor = 0.8;
    InterceptRange *= ScalingFactor;
    const int NumPassesToTry = 5;
    Vector2D Passes[NumPassesToTry];
    double deltaIP = InterceptRange/(NumPassesToTry - 1);
    for(int i=0; i<NumPassesToTry; ++i)
    { 
        Passes[i] = receiver->Pos();
        if(OpponentsGoal()->Center().x < 100)
        { 
            Passes[i].x -= deltaIP * i;
        }
        else
        { 
            Passes[i].x += deltaIP * i;
        }
    }
}
// this pass is the best found so far if it is:
// 1. Further upfield than the closest valid pass for this receiver
//    found so far
// 2. Within the playing area
// 3. Cannot be intercepted by any opponents

double ClosestSoFar = MaxFloat;
bool bResult = false;

for (int pass = 0; pass < NumPassesToTry; ++pass)
{
    double dist = fabs(Passes[pass].x - OpponentsGoal()->Center().x);

    if ((dist < ClosestSoFar) &&
        Pitch()->PlayingArea()->Inside(Passes[pass]) &&
        isPassSafeFromAllOpponents(Pitch()->Ball()->Pos(),
                                   Passes[pass],
                                   receiver,
                                   power))
    {
        ClosestSoFar = dist;
        PassTarget = Passes[pass];
        bResult = true;
    }
}

return bResult;

bool SoccerTeam::GetBestPassToReceiverComboBox(const PlayerBase* const passer,
                                                const PlayerBase* const receiver,
                                                Vector2D& PassTarget,
                                                double power) const
{
    // first, calculate how much time it will take for the ball to reach
    // this receiver, if the receiver was to remain motionless
    double time = Pitch()->Ball()->TimeToCoverDistance(Pitch()->Ball()->Pos(),
                                                         receiver->Pos(),
                                                         power);

    // return false if ball cannot reach the receiver after having been
    // kicked with the given power
    if (time < 0) return false;

    // the maximum distance the receiver can cover in this time
    double InterceptRange = time * receiver->MaxSpeed();

    // Scale the intercept range
    const double ScalingFactor = 0.5;
    InterceptRange *= ScalingFactor;

    // now calculate the pass targets which are positioned at the intercepts
    // of the from the ball to the receiver's x-line.
const int NumPassesToTry = 3;
Vector2D Passes[NumPassesToTry];

// need a relative postion of receiver wrt passer
Vector2D tempPos = receiver->Pos();
// relative_region relRegion = RegionRelativeToFirst(passer->Pos(),
receiver->Pos());

// set the intercept points
double deltaIP = InterceptRange/(NumPassesToTry - 1);
// scale based on max of 45degrees
deltaIP *= .707;

Vector2D pPos = passer->Pos();
Vector2D rPos = receiver->Pos();

// Vector2D supportSpot;

// create set of intercept points based on teammates relative position

double const offset = 10.0;

if(OpponentsGoal()->Center().x > 100)
{
    if(rPos.x >= pPos.x + offset && rPos.y >= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].y = passer->Pos().y;
            Passes[i].x = receiver->Pos().x - deltaIP;

            Passes[i].x += deltaIP * i;

            // supportSpot.y = receiver->Pos().y;
            // supportSpot.x = passer->Pos().x + 1.5 * (receiver->Pos().x -
            passer->Pos().x);
        }
    }
    else if(rPos.x >= pPos.x + offset && rPos.y <= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].y = passer->Pos().y;
            Passes[i].x = receiver->Pos().x - deltaIP;

            Passes[i].x += deltaIP * i;
        }
    }
    else if(rPos.x <= pPos.x + offset && rPos.y >= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].x = passer->Pos().x;
        }
    }
}
Passes[i].y = receiver->Pos().y - deltaIP;
Passes[i].y += deltaIP * i;
}
else if(rPos.x <= pPos.x + offset && rPos.y <= pPos.y)
{
    for(int i=0; i<NumPassesToTry; ++i)
    {
        Passes[i].x = passer->Pos().x;
        Passes[i].y = receiver->Pos().y - deltaIP;
        Passes[i].y -= deltaIP * i;
    }
}
else
{
    if(rPos.x >= pPos.x - offset && rPos.y >= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].x = passer->Pos().x;
            Passes[i].y = receiver->Pos().y - deltaIP;
            Passes[i].y += deltaIP * i;
        }
    }
    else if(rPos.x >= pPos.x - offset && rPos.y <= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].x = passer->Pos().x;
            Passes[i].y = receiver->Pos().y - deltaIP;
            Passes[i].y -= deltaIP * i;
        }
    }
    else if(rPos.x <= pPos.x - offset && rPos.y >= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].y = passer->Pos().y;
            Passes[i].x = receiver->Pos().x - deltaIP;
            Passes[i].x -= deltaIP * i;
        }
    }
    else if(rPos.x <= pPos.x - offset && rPos.y <= pPos.y)
    {
        for(int i=0; i<NumPassesToTry; ++i)
        {
            Passes[i].y = passer->Pos().y;
            Passes[i].x = receiver->Pos().x - deltaIP;
        }
    }
}

Passes[i].x -= deltaIP * i;
}
}

// the pass is the best found so far if it is:
// 1. closest to receiver
// 2. Within the playing area
// 3. Cannot be intercepted by any opponents

double FarthestSoFar = 0.0;
bool bResult = false;

for (int pass=0; pass<NumPassesToTry; ++pass)
{
    double dist = Passes[pass].DistanceSq(receiver->Pos());
    if ((dist > FarthestSoFar) &&
        Pitch()->PlayingArea()->Inside(Passes[pass]) &&
        isPassSafeFromAllOpponents(Pitch()->Ball()->Pos(),
                                   Passes[pass],
                                   receiver,
                                   power))
    {
        FarthestSoFar = dist;
        PassTarget = Passes[pass];
        bResult = true;
    }
}

return bResult;
}

bool SoccerTeam::GetBestPassToReceiverComboDribblePass(const PlayerBase*
const passer,

    const PlayerBase* const receiver,
    Vector2D& PassTarget,
    double power)const
{
    //Todo: implement
    assert(0);
    return true;
}

bool SoccerTeam::GetBestPassToReceiverComboBounding(const PlayerBase* const
passer,

    const PlayerBase* const receiver,
    Vector2D& PassTarget,
    double power)const
{
    //first, calculate how much time it will take for the ball to reach
    //this receiver, if the receiver was to remain motionless
    double time = Pitch()->Ball()->TimeToCoverDistance(Pitch()->Ball()->Pos(),
                                                        receiver->Pos(),

//return false if ball cannot reach the receiver after having been
//kicked with the given power
if (time < 0) return false;

//the maximum distance the receiver can cover in this time
double InterceptRange = time * receiver->MaxSpeed();

//Scale the intercept range
const double ScalingFactor = 0.75;
InterceptRange *= ScalingFactor;

//now calculate the pass targets which are positioned at the intercepts
//from the ball to the receiver's x-line.
const int NumPassesToTry = 3;
Vector2D Passes[NumPassesToTry];

PlayerBase::relative_region relRegion;
Vector2D debugPos1 = passer->Pos();
Vector2D debugPos2 = receiver->Pos();
double const offset = 10;

relRegion = passer->RegionRelativeToFirst(debugPos1, debugPos2, 100);
if(passer->Team()->OpponentsGoal()->Center().x > 100)
{
    switch(relRegion)
    {
    case PlayerBase::LEFT:
    {
        Passes[0].x = passer->Pos().x + InterceptRange;
Passes[0].y = passer->Pos().y;
Passes[1].x = passer->Pos().x + .8 * InterceptRange;
        Passes[1].y = passer->Pos().y;
Passes[2].x = passer->Pos().x + InterceptRange;
Passes[2].y = passer->Pos().y - offset;
        break;
    }
    case PlayerBase::RIGHT:
    {
        Passes[0].x = passer->Pos().x + InterceptRange;
Passes[0].y = passer->Pos().y;
Passes[1].x = passer->Pos().x + .8 * InterceptRange;
        Passes[1].y = passer->Pos().y;
Passes[2].x = passer->Pos().x + InterceptRange;
Passes[2].y = passer->Pos().y + offset;
        break;
    }
    }
}
else
{
    switch(relRegion)
    {
    case PlayerBase::LEFT:
    {
        Passes[0].x = passer->Pos().x - InterceptRange;
        Passes[0].y = passer->Pos().y;
        Passes[1].x = passer->Pos().x - .8 * InterceptRange;
        Passes[1].y = passer->Pos().y;
        Passes[2].x = passer->Pos().x - InterceptRange;
        Passes[2].y = passer->Pos().y + offset;
        break;
    }
    case PlayerBase::RIGHT:
    {
        Passes[0].x = passer->Pos().x - InterceptRange;
        Passes[0].y = passer->Pos().y;
        Passes[1].x = passer->Pos().x - .8 * InterceptRange;
        Passes[1].y = passer->Pos().y;
        Passes[2].x = passer->Pos().x - InterceptRange;
        Passes[2].y = passer->Pos().y - offset;
        break;
    }
    }
}

// this pass is the best found so far if it is:
// 1. Further upfield than the closest valid pass for this receiver found so far
// 2. Within the playing area
// 3. Cannot be intercepted by any opponents

double ClosestSoFar = MaxFloat;
bool  bResult      = false;

for (int pass=0; pass<NumPassesToTry; ++pass)
{
    double dist = fabs(Passes[pass].x - OpponentsGoal()->Center().x);

    if (( dist < ClosestSoFar) &&
        Pitch()->PlayingArea()->Inside(Passes[pass]) &&
        isPassSafeFromAllOpponents(Pitch()->Ball()->Pos(),
                                    Passes[pass],
                                    receiver,
                                    power))
    {
        ClosestSoFar = dist;
        PassTarget   = Passes[pass];
        bResult      = true;
    }
return bResult;
}

//-- ------------------------ isPassSafeFromOpponent ------------------------
// test if a pass from 'from' to 'to' can be intercepted by an opposing
// player
//-- ------------------------
bool SoccerTeam::isPassSafeFromOpponent(Vector2D from,
   Vector2D target,
   const PlayerBase* const receiver,
   const PlayerBase* const opp,
   double PassingForce) const
{
    //move the opponent into local space.
    Vector2D ToTarget = target - from;
    Vector2D ToTargetNormalized = Vec2DNormalize(ToTarget);
    Vector2D LocalPosOpp = PointToLocalSpace(opp->Pos(),
        ToTargetNormalized,
        ToTargetNormalized.Perp(),
        from);

    //if opponent is behind the kicker then pass is considered okay(this is
    //based on the assumption that the ball is going to be kicked with a
    //velocity greater than the opponent's max velocity)
    if ( LocalPosOpp.x < 0 )
    {
        return true;
    }

    //if the opponent is further away than the target we need to consider if
    //the opponent can reach the position before the receiver.
    if (Vec2DDistanceSq(from, target) < Vec2DDistanceSq(opp->Pos(), from))
    {
        if (receiver)
        {
            if ( Vec2DDistanceSq(target, opp->Pos()) >
                Vec2DDistanceSq(target, receiver->Pos()) )
            {
                return true;
            }
        }
        else
        {
            return false;
        }
    }
    else
    {
        return true;
    }
}
// calculate how long it takes the ball to cover the distance to the position orthogonal to the opponent's position
double TimeForBall = Pitch()->Ball()->TimeToCoverDistance(Vector2D(0,0), Vector2D(LocalPosOpp.x, 0), PassingForce);

// now calculate how far the opponent can run in this time
double reach = opp->MaxSpeed() * TimeForBall + Pitch()->Ball()->BRadius() + opp->BRadius();

// if the distance to the opponent's y position is less than his running range plus the radius of the ball and the opponent's radius then the ball can be intercepted
if (fabs(LocalPosOpp.y) < reach) {
    return false;
}

return true;

bool SoccerTeam::isPassSafeFromAllOpponents(Vector2D from, Vector2D target, const PlayerBase* const receiver, double PassingForce) const
{
    std::vector<PlayerBase*>::const_iterator opp = Opponents()->Members().begin();

    for (opp; opp != Opponents()->Members().end(); ++opp)
    {
        if (!isPassSafeFromOpponent(from, target, receiver, *opp, PassingForce))
        {
            debug_on
            return false;
        }
    }

    return true;
}

// ------------------------ CanShoot ------------------------
/\ // Given a ball position, a kicking power and a reference to a vector2D
// this function will sample random positions along the opponent's goal-
// mouth and check to see if a goal can be scored if the ball was to be
// kicked in that direction with the given power. If a possible shot is
// found, the function will immediately return true, with the target
// position stored in the vector ShotTarget.
//----------------------------------------------------------------------------------------
bool SoccerTeam::CanShoot(Vector2D  BallPos,
    double     power,
    Vector2D& ShotTarget)const
{
    //the number of randomly created shot targets this method will test
    int NumAttempts = Prm.NumAttemptsToFindValidStrike;

    while (NumAttempts--)
    {
        //choose a random position along the opponent's goal mouth. (making
        //sure the ball's radius is taken into account)
        ShotTarget = OpponentsGoal()->Center();

        //the y value of the shot position should lay somewhere between two
        //goalposts (taking into consideration the ball diameter)
        int MinYVal = OpponentsGoal()->LeftPost().y + Pitch()->Ball()->BRadius();
        int MaxYVal = OpponentsGoal()->RightPost().y - Pitch()->Ball()->
            BRadius();

        ShotTarget.y = (double)RandInt(MinYVal, MaxYVal);

        //make sure striking the ball with the given power is enough to drive
        //the ball over the goal line.
        double time = Pitch()->Ball()->TimeToCoverDistance(BallPos,
            ShotTarget,
            power);

        //if it is, this shot is then tested to see if any of the opponents
        //can intercept it.
        if (time >= 0)
        {
            if (isPassSafeFromAllOpponents(BallPos, ShotTarget, NULL, power))
            {
                return true;
            }
        }
    }

    return false;
}

//---------------------- ReturnAllFieldPlayersToHome ------------------------
--
// sends a message to all players to return to their home areas forthwith
//------------------------
void SoccerTeam::ReturnAllFieldPlayersToHome() const
{
    std::vector<PlayerBase*>::const_iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        if ((*it)->Role() != PlayerBase::goal_keeper)
        {
            Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY,
                (*it)->ID(),
                Msg_GoHome,
                NULL);
        }
    }
}

void SoccerTeam::Render() const
{
    std::vector<PlayerBase*>::const_iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        (*it)->Render();
    }

    // show the controlling team and player at the top of the display
    if (Prm.bShowControllingTeam)
    {
        gdi->TextColor(Cgdi::white);
        if ((Color() == blue) && InControl())
        {
            gdi->TextAtPos(Pitch()->cxClient()-150, 3, "Controlling Player: " + ttos(m_pControllingPlayer->ID()));
        } else if ((Color() == red) && InControl())
        {
            gdi->TextAtPos(Pitch()->cxClient()-150, 3, "Controlling Player: " + ttos(m_pControllingPlayer->ID()));
        } else if (m_pSupportSpotCalc != NULL)
        {
            m_pSupportSpotCalc->Render();
        }
    }
}

// render the sweet spots
if (Prm.bSupportSpots && InControl())
{ m_pSupportSpotCalc->Render();

268
//
//  creates the players
//----------------------------------------------------------------------------
void SoccerTeam::CreatePlayers()
{
    if (Color() == blue)
    {
        //goalkeeper
        m_Players.push_back(new GoalKeeper(this, 1,
                                           TendGoal::Instance(),
                                           Vector2D(0,1),
                                           Vector2D(0.0, 0.0),
                                           Prm.PlayerMass,
                                           Prm.PlayerMaxForce,
                                           Prm.PlayerMaxSpeedWithoutBall,
                                           Prm.PlayerMaxTurnRate,
                                           Prm.PlayerScale));

        //create the players
        m_Players.push_back(new FieldPlayer(this, 6,
                                             Wait::Instance(),
                                             Vector2D(0,1),
                                             Vector2D(0.0, 0.0),
                                             Prm.PlayerMass,
                                             Prm.PlayerMaxForce,
                                             Prm.PlayerMaxSpeedWithoutBall,
                                             Prm.PlayerMaxTurnRate,
                                             Prm.PlayerScale,
                                             PlayerBase::attacker));

        m_Players.push_back(new FieldPlayer(this, 8,
                                             Wait::Instance(),
                                             Vector2D(0,1),
                                             Vector2D(0.0, 0.0),
                                             Prm.PlayerMass,
                                             Prm.PlayerMaxForce,
                                             Prm.PlayerMaxSpeedWithoutBall,
                                             Prm.PlayerMaxTurnRate,
                                             Prm.PlayerScale,
                                             PlayerBase::attacker));

        m_Players.push_back(new FieldPlayer(this, 3,
                                             Wait::Instance(),
                                             Vector2D(0,1),
                                             Vector2D(0.0, 0.0),
                                             Prm.PlayerMass,
                                             Prm.PlayerMaxForce,
                                             Prm.PlayerMaxSpeedWithoutBall,
m_Players.push_back(new FieldPlayer(this,
5,
Wait::Instance(),
Vector2D(0,1),
Vector2D(0.0, 0.0),
Prm.PlayerMass,
Prm.PlayerMaxForce,
Prm.PlayerMaxSpeedWithoutBall,
Prm.PlayerMaxTurnRate,
Prm.PlayerScale,
PlayerBase::defender));

} else {

    //goalkeeper
    m_Players.push_back(new GoalKeeper(this,
16,
TendGoal::Instance(),
Vector2D(0,-1),
Vector2D(0.0, 0.0),
Prm.PlayerMass,
Prm.PlayerMaxForce,
Prm.PlayerMaxSpeedWithoutBall,
Prm.PlayerMaxTurnRate,
Prm.PlayerScale));

    //create the players
    m_Players.push_back(new FieldPlayer(this,
9,
Wait::Instance(),
Vector2D(0,-1),
Vector2D(0.0, 0.0),
Prm.PlayerMass,
Prm.PlayerMaxForce,
Prm.PlayerMaxSpeedWithoutBall,
Prm.PlayerMaxTurnRate,
Prm.PlayerScale,
PlayerBase::attacker));

    m_Players.push_back(new FieldPlayer(this,
11,
Wait::Instance(),
Vector2D(0,-1),
Vector2D(0.0, 0.0),
Prm.PlayerMass,
Prm.PlayerMaxForce,
Prm.PlayerMaxSpeedWithoutBall,
Prm.PlayerMaxTurnRate,
Prm.PlayerScale,
PlayerBase::attacker));

m_Players.push_back(new FieldPlayer(this,
  12,
  Wait::Instance(),
  Vector2D(0,-1),
  Vector2D(0.0, 0.0),
  Prm.PlayerMass,
  Prm.PlayerMaxForce,
  Prm.PlayerMaxSpeedWithoutBall,
  Prm.PlayerMaxTurnRate,
  Prm.PlayerScale,
  PlayerBase::defender));

m_Players.push_back(new FieldPlayer(this,
  14,
  Wait::Instance(),
  Vector2D(0,-1),
  Vector2D(0.0, 0.0),
  Prm.PlayerMass,
  Prm.PlayerMaxForce,
  Prm.PlayerMaxSpeedWithoutBall,
  Prm.PlayerMaxTurnRate,
  Prm.PlayerScale,
  PlayerBase::defender));

}

//register the players with the entity manager
std::vector<PlayerBase*>::iterator it = m_Players.begin();

for (it; it != m_Players.end(); ++it)
{
  EntityMgr->RegisterEntity(*it);
}

PlayerBase* SoccerTeam::GetPlayerFromID(int id) const
{
  std::vector<PlayerBase*>::const_iterator it = m_Players.begin();

  for (it; it != m_Players.end(); ++it)
  {
    if ((*it)->ID() == id) return *it;
  }

  return NULL;
}
void SoccerTeam::SetPlayerHomeRegion(int plyr, int region) const
{
    assert ((plyr>=0) && (plyr<m_Players.size()));

    m_Players[plyr]->SetHomeRegion(region);
}

//---------------------- UpdateTargetsOfWaitingPlayers ----------------------

void SoccerTeam::UpdateTargetsOfWaitingPlayers() const
{
    std::vector<PlayerBase*>::const_iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        if ( (*it)->Role() != PlayerBase::goal_keeper )
        {
            //cast to a field player
            FieldPlayer* plyr = static_cast<FieldPlayer*>(*it);

            if ( plyr->GetFSM()->isInState(*Wait::Instance()) || plyr->GetFSM()->isInState(*ReturnToHomeRegion::Instance()) )
            {
                plyr->Steering()->SetTarget(plyr->HomeRegion()->Center());
            }
        }
    }
}

//--------------------------- AllPlayersAtHome ------------------------------

bool SoccerTeam::AllPlayersAtHome() const
{
    std::vector<PlayerBase*>::const_iterator it = m_Players.begin();
    for (it; it != m_Players.end(); ++it)
    {
        if (!(*it)->InHomeRegion())
        {
            return false;
        }
    }

    return true;
}
void SoccerTeam::RequestPass(FieldPlayer* requester) const
{
    // maybe put a restriction here
    if (RandFloat() > 0.1) return;

    if (isPassSafeFromAllOpponents(ControllingPlayer()->Pos(), requester->Pos(), requester, Prm.MaxPassingForce))
    {
        // tell the player to make the pass
        // let the receiver know a pass is coming
        Dispatcher->DispatchMsg(SEND_MSG_IMMEDIATELY, requester->ID(), ControllingPlayer()->ID(), Msg_PassToMe, requester);
    }
}

bool SoccerTeam::isOpponentWithinRadius(Vector2D pos, double rad)
{
    std::vector<PlayerBase*>::const_iterator end = Opponents()->Members().end();
    std::vector<PlayerBase*>::const_iterator it;

    for (it=Opponents()->Members().begin(); it !=end; ++it)
    {
        if (Vec2DDistanceSq(pos, (*it)->Pos()) < rad*rad)
            return true;
    }

    return false;
}
Vector2D SoccerTeam::CalculateSupportSpotComboDiagonal(Vector2D const &playerPos, Vector2D const &ballTarget) {
    Vector2D supportSpot;

    // base this distance on twice the passing distance up field
    // if the receiving player (based on ball target) is behind the passer
    // the passer still moves upfield
double xSupportSpotDistance = 2 * fabs(playerPos.x - ballTarget.x);

    supportSpot.y = playerPos.y;
    if( OpponentsGoal()->Center().x > playerPos.x) {
        supportSpot.x = playerPos.x + xSupportSpotDistance;
    } else {
        supportSpot.x = playerPos.x - xSupportSpotDistance;
    }

    return supportSpot;
}

Vector2D SoccerTeam::CalculateSupportSpotComboBox(Vector2D const &playerPos, Vector2D const &rcvrPos, Vector2D const &ballTarget) {
    Vector2D supportSpot;

    if(OpponentsGoal()->Center().x > 100) {
        // rcvr in front
        if(rcvrPos.x > playerPos.x) {
            supportSpot.x = ballTarget.x;
            supportSpot.y = playerPos.y + 1.2 * (rcvrPos.y - playerPos.y);
        } else // rcvr in rear
        {
            supportSpot.y = ballTarget.y;
            supportSpot.x = playerPos.x + 1.2 * (playerPos.x - rcvrPos.x);
        }
    } else {
        // rcvr in front
        if(rcvrPos.x < playerPos.x) {
            supportSpot.x = ballTarget.x;
            supportSpot.y = playerPos.y + 1.2 * (rcvrPos.y - playerPos.y);
        } else // rcvr in rear
        {
Vector2D SoccerTeam::CalculateSupportSpotComboDribblePass(Vector2D const &playerPos, Vector2D const &ballTarget) {
    Vector2D supportSpot;
    //Todo: implement
    assert(0);
    return supportSpot;
}

Vector2D SoccerTeam::CalculateSupportSpotComboBounding(Vector2D const &playerPos, Vector2D const &rcvrPos, Vector2D const &ballTarget) {
    Vector2D supportSpot;
    double const scale = 2.5;
    double const offset = 10;

    // we care about left versus right
    if(OpponentsGoal()->Center().x > 100) {
        supportSpot.x = rcvrPos.x + fabs(rcvrPos.x - ballTarget.x) * scale;

        if(rcvrPos.y > ballTarget.y) {
            supportSpot.y = ballTarget.y - offset;
        } else {
            supportSpot.y = ballTarget.y + offset;
        }
    } else {
        supportSpot.x = rcvrPos.x - fabs(rcvrPos.x - ballTarget.x) * scale;

        if(rcvrPos.y > ballTarget.y) {
            supportSpot.y = ballTarget.y - offset;
        } else {
            supportSpot.y = ballTarget.y + offset;
        }
    }
}
return supportSpot;
}

SoccerTeam.h

#ifndef SOCCERTEAM_H
#define SOCCERTEAM_H
#pragma warning (disable:4786)

//------------------------------------------------------------------------
// Name: SoccerTeam.h
// Desc: class to define a team of soccer playing agents. A SoccerTeam
// contains several field players and one goalkeeper. A SoccerTeam
// is implemented as a finite state machine and has states for
// attacking, defending, and KickOff.
// Author: Mat Buckland 2003 (fup@ai-junkie.com)
//------------------------------------------------------------------------

#include <vector>
#include "Game/Region.h"
#include "SupportSpotCalculator.h"
#include "FSM/StateMachine.h"

class Goal;
class PlayerBase;
class FieldPlayer;
class SoccerPitch;
class GoalKeeper;
class SupportSpotCalculator;

class SoccerTeam
{
public:

   enum team_color {blue, red};
   enum team_type
   {
      BASE,        // no mods to the base team
      CCxBR,       // mods for context recognition (explicit)
      CxBRwJIT,    // mods to infer context
      CxBR         // mods for context tactics as with above but no
                    // recognition
   };

277
private:

    //an instance of the state machine class
   StateMachine<SoccerTeam>*  m_pStateMachine;
    
    //the team must know its own color!
    team_color                m_Color;
    
    //team type -- prototypes for testing CCxBR
    team_type     m_Type;
    
    //pointers to the team members
    std::vector<PlayerBase*>  m_Players;
    
    //a pointer to the soccer pitch
    SoccerPitch*              m_pPitch;
    
    //pointers to the goals
    Goal*                    m_pOpponentsGoal;
    Goal*                    m_pHomeGoal;
    
    //a pointer to the opposing team
    SoccerTeam*               m_pOpponents;
    
    //pointers to 'key' players
    PlayerBase*              m_pControllingPlayer;
    PlayerBase*              m_pSupportingPlayer;
    PlayerBase*              m_pReceivingPlayer;
    PlayerBase*              m_pPlayerClosestToBall;
    
    //the squared distance the closest player is from the ball
    double                     m_dDistSqToBallOfClosestPlayer;
    
    //players use this to determine strategic positions on the playing field
    SupportSpotCalculator*    m_pSupportSpotCalc;
    
    // players use this to track supporting position for ComboDiagonal
    Vector2D                 m_vSupportSpotComboDiagonal;
    Vector2D                 m_vSupportSpotComboBox;
    Vector2D                 m_vSupportSpotComboDribblePass;
    Vector2D                 m_vSupportSpotComboBounding;
    
    //creates all the players for this team
    void CreatePlayers();

    //called each frame. Sets m_pClosestPlayerToBall to point to the player
    //closest to the ball.
    void CalculateClosestPlayerToBall();

public:

    SoccerTeam(Goal*        home_goal,
Goal* opponents_goal,
SoccerPitch* pitch,
team_color color,
    team_type type);

~SoccerTeam();

// the usual suspects
void Render() const;
void Update();

// calling this changes the state of all field players to that of
// ReturnToHomeRegion. Mainly used when a goal keeper has
// possession
void ReturnAllFieldPlayersToHome() const;

// returns true if player has a clean shot at the goal and sets ShotTarget
// to a normalized vector pointing in the direction the shot should be
// made. Else returns false and sets heading to a zero vector
bool CanShoot(Vector2D BallPos,
    double power,
    Vector2D& ShotTarget = Vector2D()) const;

// The best pass is considered to be the pass that cannot be intercepted
// by an opponent and that is as far forward of the receiver as possible
// If a pass is found, the receiver's address is returned in the
// reference, 'receiver' and the position the pass will be made to is
// returned in the reference 'PassTarget'
bool FindPass(const PlayerBase* const passer,
    PlayerBase*& receiver,
    Vector2D& PassTarget,
    double power,
    double MinPassingDistance) const;

// Used by the ComboDiagonal Context to find pass
// The best pass is considered to be the pass that cannot be intercepted
// by an opponent and that is as far forward of the receiver as possible
// If a pass is found, the receiver's address is returned in the
// reference 'receiver' and the position the pass will be made to is
// returned in the reference 'PassTarget'
bool FindPassComboDiagonal(const PlayerBase* const passer,
    PlayerBase*& receiver,
    Vector2D& PassTarget,
    double power,
    double MinXDistance,
    double MaxDistBehindPasser) const;

bool FindPassComboBox(const PlayerBase* const passer,
    PlayerBase*& receiver,
    Vector2D& PassTarget,
    double power,
    double MinXDistance,
    double MaxDistBehindPasser) const;
bool FindPassComboDribblePass(const PlayerBase* const passer,
    PlayerBase*& receiver,
    Vector2D& PassTarget,
    double power,
    double MinXDistance,
    double MaxDistBehindPasser) const;

bool FindPassComboBounding(const PlayerBase* const passer,
    PlayerBase*& receiver,
    Vector2D& PassTarget,
    double power,
    double MinXDistance,
    double MaxDistBehindPasser) const;

//Three potential passes are calculated. One directly toward the receiver's
//current position and two that are the tangents from the ball position
//to the circle of radius 'range' from the receiver.
//These passes are then tested to see if they can be intercepted by an
//opponent and to make sure they terminate within the playing area. If
//all the passes are invalidated the function returns false. Otherwise
//the function returns the pass that takes the ball closest to the
//opponent's goal area.
bool GetBestPassToReceiver(const PlayerBase* const passer,
    const PlayerBase* const receiver,
    Vector2D& PassTarget,
    const double power)const;

//Five potential passes are calculated. One directly toward the receiver's
//current position and four that are...
bool GetBestPassToReceiverComboDiagonal(const PlayerBase* const
    passer,
    const PlayerBase* const receiver,
    const Vector2D& PassTarget,
    const double power)const;

//
//bool GetBestPassToReceiverComboBox(const PlayerBase* const
//    passer,
//    const PlayerBase* const receiver,
//    const Vector2D& PassTarget,
//    const double power)const;

//
//bool GetBestPassToReceiverComboDribblePass(const PlayerBase* const
//    passer,
//    const PlayerBase* const receiver,
Vector2D& PassTarget,
    const
double power)const;

    //
    bool GetBestPassToReceiverComboBounding(const PlayerBase* const
    passer,
    PlayerBase* const receiver,
    Vector2D& PassTarget,
    const
double power)const;

    //test if a pass from positions 'from' to 'target' kicked with force
    //'PassingForce'can be intercepted by an opposing player
    bool isPassSafeFromOpponent(Vector2D    from,
    Vector2D    target,
    const PlayerBase* const receiver,
    const PlayerBase* const opp,
    double       PassingForce)const;

    //tests a pass from position 'from' to position 'target' against each
    //member
    //of the opposing team. Returns true if the pass can be made without
    //getting intercepted
    bool isPassSafeFromAllOpponents(Vector2D from,
    Vector2D target,
    const PlayerBase* const receiver,
    double     PassingForce)const;

    //returns true if there is an opponent within radius of position
    bool isOpponentWithinRadius(Vector2D pos, double rad);

    //this tests to see if a pass is possible between the requester and
    //the controlling player. If it is possible a message is sent to the
    //controlling player to pass the ball asap.
    void RequestPass(FieldPlayer* requester)const;

    //calculates the best supporting position and finds the most appropriate
    //attacker to travel to the spot
    PlayerBase* DetermineBestSupportingAttacker();

    const std::vector<PlayerBase*> & Members() const{ return m_Players; }

    StateMachine<SoccerTeam>* GetFSM() const{ return m_pStateMachine; }

    Goal* const HomeGoal() const{ return m_pHomeGoal; }
    Goal* const OpponentsGoal() const{ return m_pOpponentsGoal; }

    SoccerPitch* const Pitch() const{ return m_pPitch; }
    SoccerTeam* const Opponents() const{ return m_pOpponents; }
    void SetOpponents(SoccerTeam* opps){ m_pOpponents = opps; }
team_color
Color() const { return m_Color; }

team_type
Type() const { return m_Type; }

void SetPlayerClosestToBall(PlayerBase* plyr)
{ m_pPlayerClosestToBall = plyr; }
PlayerBase* PlayerClosestToBall() const { return m_pPlayerClosestToBall; }

double ClosestDistToBallSq() const { return m_dDistSqToBallOfClosestPlayer; }

Vector2D GetSupportSpot() const { return m_pSupportSpotCalc->GetBestSupportingSpot(); }

Vector2D GetSupportSpotComboDiagonal() const { return m_vSupportSpotComboDiagonal; }
void SetSupportSpotComboDiagonal(Vector2D supportSpot)
{ m_vSupportSpotComboDiagonal = supportSpot; }
Vector2D GetSupportSpotComboBox() const { return m_vSupportSpotComboBox; }
void SetSupportSpotComboBox(Vector2D supportSpot)
{ m_vSupportSpotComboBox = supportSpot; }
Vector2D GetSupportSpotComboDribblePass() const { return m_vSupportSpotComboDribblePass; }
void SetSupportSpotComboDribblePass(Vector2D supportSpot)
{ m_vSupportSpotComboDribblePass = supportSpot; }
Vector2D GetSupportSpotComboBounding() const { return m_vSupportSpotComboBounding; }
void SetSupportSpotComboBounding(Vector2D supportSpot)
{ m_vSupportSpotComboBounding = supportSpot; }

Vector2D CalculateSupportSpotComboDiagonal(Vector2D const &playerPos, Vector2D const &ballTarget);
Vector2D CalculateSupportSpotComboBox(Vector2D const &playerPos, Vector2D const &rcvrPos, Vector2D const &ballTarget);
Vector2D CalculateSupportSpotComboDribblePass(Vector2D const &playerPos, Vector2D const &ballTarget);
Vector2D CalculateSupportSpotComboBounding(Vector2D const &playerPos, Vector2D const &rcvrPos, Vector2D const &ballTarget);

PlayerBase* SupportingPlayer() const { return m_pSupportingPlayer; }
void SetSupportingPlayer(PlayerBase* plyr)
{ m_pSupportingPlayer = plyr; }

PlayerBase* Receiver() const { return m_pReceivingPlayer; }
void SetReceiver(PlayerBase* plyr)
{ m_pReceivingPlayer = plyr; }

PlayerBase* ControllingPlayer() const { return m_pControllingPlayer; }
void SetControllingPlayer(PlayerBase* plyr)
{ m_pControllingPlayer = plyr; }
//rub it in the opponents faces!
Opponents()->LostControl();
}

bool InControl() const {if (m_pControllingPlayer) return true; else return false;}
void LostControl() {m_pControllingPlayer = NULL;}

PlayerBase* GetPlayerFromID(int id) const;

void SetPlayerHomeRegion(int plyr, int region) const;

void DetermineBestSupportingPosition() const {m_pSupportSpotCalc->DetermineBestSupportingPosition();}

void UpdateTargetsOfWaitingPlayers() const;

//returns false if any of the team are not located within their home region
bool AllPlayersAtHome() const;

std::string Name() const {if (m_Color == blue) return "Blue"; return "Red";}
};

#endif
REFERENCES


