USING FIRST ORDER INDUCTIVE LEARNING AS AN ALTERNATIVE TO A SIMULATOR IN A GAME ARTIFICIAL INTELLIGENCE

A Thesis
Presented to
The Academic Faculty

by

Kathryn Anna Long

In Partial Fulfillment
of the Requirements for the Degree
Bachelor's in Computer Science with Research Option in the
School of Computer Science

Georgia Institute of Technology
May 2009

USING FIRST ORDER INDUCTIVE LEARNING AS AN ALTERNATIVE TO A SIMULATOR IN A GAME ARTIFICIAL INTELLIGENCE

Approved by:

Dr. Ashwin Ram, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Santi Ontañón
School of Computer Science
Georgia Institute of Technology

Dr. Amy Bruckman
School of Computer Science
Georgia Institute of Technology

Date Approved: May 1, 2009

ACKNOWLEDGEMENTS

I would like to thank my mother and father, who have supported me during every step of my journey, as well as my fiancée for his understanding and support. I would like to thank the SAIC Scholars Program for introducing me to academic research, and Zsolt Kira for being an excellent mentor during my time with the program. Most importantly, I would like to thank Ashwin Ram and Santi Ontañón for supporting me and mentoring me on the research presented in this thesis.

TABLE OF CONTENTS

ABSTRACT

Currently many game artificial intelligences attempt to determine their next moves by using a simulator to predict the effect of actions in the world. However, writing such a simulator is time-consuming, and the simulator must be changed substantially whenever a detail in the game design is modified. As such, this research project set out to determine if a version of the first order inductive learning algorithm could be used to learn rules that could then be used in place of a simulator.

By eliminating the need to write a simulator for each game by hand, the entire Darmok 2 project could more easily adapt to additional real-time strategy games. Over time, Darmok 2 would also be able to provide better competition for human players by training the artificial intelligences to play against the style of a specific player. Most importantly, Darmok 2 might also be able to create a general solution for creating game artificial intelligences, which could save game development companies a substantial amount of money, time, and effort.

CHAPTER 1

INTRODUCTION

If you have ever played a computer game, changes are good that you have played against an artificial intelligence. Chances are also good that this artificial intelligence chose its actions based - at least in part - on the output of a simulator. In almost all games in which the human player is competing against a built-in artificial intelligence, a simulator is running in the background helping the artificial intelligence make the most appropriate moves based on the difficulty level desired by the player.

Most game companies design their games and the simulators for these games side-by-side. Unfortunately, companies often make multiple changes to their game design between the development of the simulator and the public release. Although these changes may be small in terms of game design, they may be substantial in terms of simulator design. In some cases, the company is forced to delay its release date, revert back to the previous game design, or release the game with a sub-par simulator.

However, if companies could find a method to learn the effect of actions on the world without using a hand-coded simulator, much time and effort could be saved. It is with this focus that this project began. Specifically, this research project set out to determine if a version of the first order inductive learning algorithm could be used to learn rules that could then be used in place of a simulator. As this project is a large undertaking involving many people and parts, the entire project has not been completed at this time. However, I have accomplished my low project goal of using the first order inductive learning algorithm to learn rules within Darmok 2.

The work described in this paper has been completed as part of the Darmok 2 project in the Georgia Tech Cognitive Computing Lab. Darmok 2 builds off of many of the ideas and lessons from the original Darmok project, also from Georgia Tech's Cognitive Computing Lab. The Darmok 2 project as a whole has the goal of creating real-time case-based reasoning algorithms that enable a game artificial intelligence to play strategically and learn from experience in real-time strategy games. More detail concerning the Darmok 2 project as whole will be given in the Darmok 2 Background chapter.

CHAPTER 2

LITERATURE REVIEW

An understanding of both the past work completed on the Darmok 2 project and the goals of the Darmok 2 project are needed in order to understand why replacing a hard-coded simulator with a learned simulator is desirable. Understanding the background of the Darmok 2 project also helps explain why our approach is novel and note-worthy. As such, it is important to understand the Darmok 2 architecture [9] and the case based planning approach currently used in Darmok 2 [12]. It is also important to be familiar with previous work, including implementation of a real-time case based planning and execution approach designed to deal with real-time strategy games [8,13], design of a domain independent off-line adaptation technique for finding and improving plans in real-time strategy games [14], and creation of a situation assessment algorithm which improves plan retrieval for case-based planning [6].

In this research, we used the first order inductive learning algorithm [10, 11] to learn a set of rules that we expect can be used in place of a simulator. Most readers will find other research [4] on planning, execution, and learning to be relevant though, as well as work on learning in a noisy environment [16]. Both papers display other learning algorithms and approaches that are relevant to the rules-based learning presented here.

Finnsson and Björnsson discuss why it is necessary to simulate the world and attempt to predict the effects of actions in the world [3]. They take a unique approach towards the computer Go game by using Monte Carlo/UCT simulation techniques for action selection. Other research [1] has also studied simulation and Monte-Carlo Tree

Search as a way to solve the computer Go game successfully, but it is doubtful this approach could be abstracted enough to be useful to the Darmok 2 problem.

One research group found that efficient learning can be achieved when either a human trainer or a training program is available to provide solution traces on demand [15]. However, this approach would be too numerically intensive to function accurately or quickly enough in Darmok 2's real-time strategy game environment. Another research group employs an interesting approach [5] that uses inductive logic programming to acquire rules necessary for prediction. Specifically, this approach adapts its own behavior by avoiding actions which are predicted to be failures. It is hard to determine if such an approach could be successful for our research problem, but this approach may be considered if we are unable to use the rules learned by the first order inductive learning approach to simulate the effect of actions on the game state effectively.

CHAPTER 3

DARMOK 2 BACKGROUND

The work of this thesis was completed as part of the Darmok 2 project in the Georgia Tech Cognitive Computing Lab. As such, it is impossible to discuss my thesis research without at least explaining the goals of the Darmok 2 project as a whole and familiarizing the reader with our system.

Darmok 2 is a real-time case-based planning system designed to play real-time strategy games. The main focus of Darmok 2 is to explore learning from unannotated human demonstrations. Although Darmok 2 could theoretically play any type of game, we are currently focusing on real-time strategy games because the planning nature of the Darmok 2 system handles these games better than those that focus more on reactive actions [7]. We will come back to this point later in this section when we discuss the four games currently being studied by the Darmok 2 team.

Make Me Play Me

The most impressive contribution of Darmok 2 to the reader who may not be familiar with artificial intelligence is that given demonstrations exhibiting a particular strategy, Darmok 2 can learn this strategy and create an artificial intelligence that employs this strategy. We have recently created a new social gaming website specifically for Darmok 2 called Make Me Play Me. The site is currently in private alpha testing, but we hope to open it up to the public very soon. The main idea behind the site is that users play against an artificial intelligence to create traces, where a trace is merely a log of the

user's actions and the state of the environment at important points in the game. The user can then 'Make Me' by choosing traces to use in training a Mind Engine. A Mind Engine is an artificial intelligence that you can train using traces from games you have played. The Mind Engine will then play using the strategies that you employed during these chosen games. Users can then 'Play Me' by playing their Mind Engine against other Mind Engines or humans.

Games

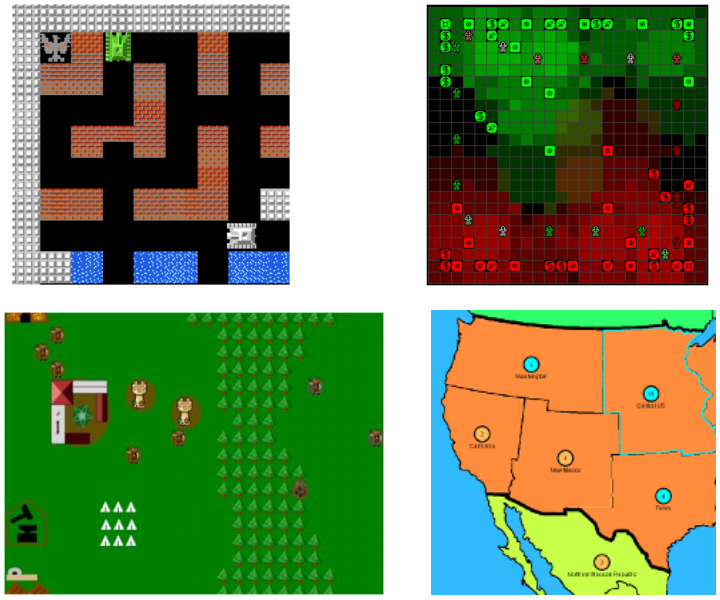


Figure 1: The four games implemented for Darmok 2 – Starting in the upper left corner and going clockwise: BattleCity, Towers, Vanquish, and S2.

We have implemented four games specifically for evaluating Darmok 2: BattleCity, Towers, S2, and Vanquish (all shown in Figure 1). BattleCity is an action game in which the player controls a tank with the goal of destroying all of the enemy tanks or destroying all of the enemy bases. Towers is a multiplayer towers defense game, where players build towers in order to stop enemy forces from attacking the player's base while the player's own forces attack the enemy base(s). S2 is a real-time strategy game

modeled after Warcraft II, with some simplifications. Finally, Vanquish is a turn-based game modeled after Risk, with the only simplification being the lack of Risk cards. For each game, a set of subgoals and sensors were defined to allow for hierarchical learning. For example, BattleCity has subgoals such as ‘get in line with enemy base goal’ and ‘destroy enemies goal’, and sensors such as ‘block ahead sensor’ and ‘next shot delay sensor’.

Each game requires different skills - BattleCity requires fast reflexes and reactive behavior, while Towers requires geometrical planning skills in order to optimally place the towers. S2 requires long term planning and strategic reasoning in order to optimally manage resources and units, and Vanquish requires intermediate planning and strategic reasoning to stage attacks optimally [7]. The wide variance in the types of games implemented for Darmok 2 was intentional, as we wanted to show the flexibility of our system.

CHAPTER 4

USING FIRST ORDER INDUCTIVE LEARNING TO LEARN RULES

The main focus of my research was to use the first order inductive learning algorithm to learn rules. These rules could then be used to predict how actions in the world might influence the game state. This work was broken down into three distinct research parts. Two parts needed to be and have been completed for this thesis, and one remains to be completed. The two parts which have been completed are discussed below, and the third part is discussed in the Future Work chapter.

First Order Inductive Learner

Many algorithms could be used to learn rules applicable to the Darmok 2 environment, but we decided to use the First Order Inductive Learner (FOIL) algorithm because of its ability to take in predicates and produce a rule list in which the individual rules are ranked by their Laplace accuracy. Our implementation of the First Order Inductive Learner is based on the learner written by Frans Coenen [2] and the algorithm originally designed by Ross Quinlan [10, 11].

FOIL works by constructing a set of clauses that classify all positive examples of a specified goal, while ruling out all negative examples. We start with a single empty clause on the left-hand side and the goal predicate on the right-hand side. The single empty clause classifies every example as positive, so we must add a single literal to the left-hand side to make the clause more specific. We try all possible literals, attempting to

pick the one that when added makes the right-hand side clause agrees with some subset of the positive examples and as few of the negative examples as possible. If the left-hand side clause still agrees with some of the negative examples, then repeat the process of adding another literal until the left-hand side clause agrees with none of the negative examples. If the left-hand side clause now agrees with none of the negative examples, we add this clause to the solution set of clauses and remove the subset of the positive examples that the clause agrees with from the training set. We start again with a single empty clause on the left-hand side, and continue this process until no positive examples remain in the training set. At this point, the clauses in the solution set of clauses are considered the rule list.

Applying the General Learner to the Darmok 2 Environment

In this step we apply the general first order inductive learner algorithm to the Darmok 2 environment with the goal of producing rules that can help explain particular actions in the environment. Note that the explanation below is largely supplemented by the examples given in the Experimental Design and Experimental Results chapters.

Before we can apply the general learner to the Darmok 2 environment, we must map each possible goal, sensor, and difference state to a unique integer, and gather at least a couple traces. Traces are relatively simple to create; one must merely play the desired game and the program will automatically record your actions into a trace file. Once the traces are uploaded to the Darmok 2 code base, our program looks through each trace, and for each entry in each trace, determines which goals, sensors, or difference states are positive in that entry. The program records positive goals, sensors, or

difference states according to their numerical mappings, and then appends whether the entry is a positive or negative instance of the attribute we are considering. Then the program inputs this data into the first order inductive learner algorithm, and the first order inductive learner outputs the rule list that classifies the particular attribute we are considering. The program repeats this process for each defined attribute, such that we end up with a separate rule list for each defined attribute.

CHAPTER 5

EXPERIMENTAL DESIGN

All of the data displayed in this chapter concerns the BattleCity game, but similar data could be obtained for Towers, S2, and Vanquish.

Mapping Used for BattleCity

The first step in the process is to assign a unique integer to each possible goal, sensor, and difference state. It is important for our implementation of the first order inductive learner that the integers start with 0, that no integers are skipped, and that the classifications (positive and negative in this case) are listed last. The following list depicts the mappings used for BattleCity:

```
newEntity: 0
bc.d2.sensors.EnemyInLineSensor: 1
bc.d2.sensors.NextShotDelaySensor: 2
bc.d2.conditions.GetInLineWithEnemyBaseGoal: 3
bc.d2.conditions.GetInLineWithEnemyGoal: 4
bc.d2.conditions.DestroyEnemiesGoal: 5
bc.d2.sensors.NextMoveDelaySensor: 6
bc.d2.sensors.BlockAheadSensor: 7
bc.d2.conditions.WinGameGoal: 8
disappearedEntity: 9
bc.d2.sensors.WallAheadSensor: 10
bc.d2.sensors.PlayerBaseInLineSensor: 11
bc.d2.sensors.EnemyBaseInLineSensor: 12
bc.d2.conditions.DestroyEnemyBaseGoal: 13
changedEntity: 14
positive: 15
negative: 16
```

Positive Goals, Sensors, and Difference States from a Trace Before Mapping

Following is a sample of what part of one trace looks like when we display only the goals, sensors, and difference states that were positive in that entry. Note that although the following representation only shows the first four entries from one trace, a common BattleCity trace can easily contain over one hundred entries.

```
[bc.d2.sensors.BlockAheadSensor,
bc.d2.sensors.EnemyBaseInLineSensor,
bc.d2.sensors.EnemyInLineSensor,
bc.d2.sensors.NextMoveDelaySensor,
bc.d2.sensors.NextShotDelaySensor,
bc.d2.sensors.PlayerBaseInLineSensor,
bc.d2.sensors.WallAheadSensor, changedEntity, newEntity,
positive]

[bc.d2.sensors.BlockAheadSensor,
bc.d2.sensors.EnemyBaseInLineSensor,
bc.d2.sensors.EnemyInLineSensor,
bc.d2.sensors.NextMoveDelaySensor,
bc.d2.sensors.NextShotDelaySensor,
bc.d2.sensors.PlayerBaseInLineSensor,
bc.d2.sensors.WallAheadSensor, changedEntity, negative]

[bc.d2.sensors.BlockAheadSensor,
bc.d2.sensors.EnemyBaseInLineSensor,
bc.d2.sensors.EnemyInLineSensor,
bc.d2.sensors.NextMoveDelaySensor,
bc.d2.sensors.NextShotDelaySensor,
bc.d2.sensors.PlayerBaseInLineSensor,
bc.d2.sensors.WallAheadSensor, changedEntity, negative]

[bc.d2.sensors.BlockAheadSensor,
bc.d2.sensors.EnemyBaseInLineSensor,
bc.d2.sensors.EnemyInLineSensor,
bc.d2.sensors.PlayerBaseInLineSensor,
bc.d2.sensors.WallAheadSensor, changedEntity, negative]
```

Positive Goals, Sensors, and Difference States from a Trace After Mapping

Following is a sample of what part of one trace looks like when we display only the mappings for the goals, sensors, and difference states that were positive in that entry.

Note that the following representation shows the same four entries from the above trace after the mapping has been completed.

```
7 12 1 6 2 11 10 14 0 15
7 12 1 6 2 11 10 14 16
7 12 1 6 2 11 10 14 16
7 12 1 11 10 14 16
```

The above sample represents the format in which the traces are input into the first order inductive learning algorithm.

Output from First Order Inductive Learner

Following is a sample of the output from the first order inductive learner. There are many results displayed in the following results chapter, but I have included a sample here for completeness. Below you can see some sample rules - and their associated Laplace accuracies - for newEntry and NextShotDelaySensor. What these rules mean will be discussed in detail in the results section. It is important to note that rules for both positive and negative classifications are given – this is because we run the algorithm once looking for positive classifications (as the algorithm was described earlier), and then once more looking for negative classifications.

```
Creating rules for newEntity...
(1)  {7}  ->  {16}  0.72%
(2)  {8 12}  ->  {16}  0.71%
(3)  {3}  ->  {16}  0.69%
(4)  {8 13}  ->  {15}  0.5%
(5)  {13}  ->  {16}  0.5%
(6)  {8}  ->  {15}  0.36%
(7)  {3}  ->  {15}  0.31%
(8)  {7}  ->  {15}  0.28%

Creating rules for bc.d2.sensors.NextShotDelaySensor...
(1)  {7}  ->  {16}  0.82%
(2)  {8}  ->  {15}  0.8%
(3)  {3}  ->  {15}  0.35%
```

CHAPTER 6

EXPERIMENTAL RESULTS AND DISCUSSION

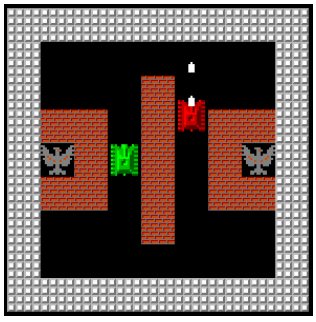
All of the results displayed here concern the BattleCity game, but we hope to run similar trials on Towers in the near future. We obtained the following output from the first order inductive learning algorithm after training it on four traces in which a player played BattleCity merely attempting to win. In other words, the player had no preference whether he won by shooting the enemy or shooting the enemy base – he merely did whichever was easier.

```
Creating rules for newEntity...
(1)  {3}  ->  {16}  0.8%
(2)  {8}  ->  {16}  0.8%
(3)  {7}  ->  {15}  0.29%

Creating rules for bc.d2.sensors.NextShotDelaySensor...
(1)  {7}  ->  {16}  0.82%
(2)  {8}  ->  {15}  0.8%
(3)  {3}  ->  {15}  0.35%

Creating rules for bc.d2.conditions.GetInLineWithEnemyBaseGoal...
(1)  {3}  ->  {15}  0.95%

Creating rules for bc.d2.conditions.DestroyEnemiesGoal...
(1)  {7}  ->  {16}  1.0%
(2)  {8}  ->  {15}  0.8%

Creating rules for bc.d2.sensors.NextMoveDelaySensor...
(1)  {7}  ->  {16}  0.82%
(2)  {8}  ->  {15}  0.8%
(3)  {3}  ->  {15}  0.35%

Creating rules for bc.d2.conditions.WinGameGoal...
(1)  {7}  ->  {16}  1.0%
(2)  {8}  ->  {15}  0.8%

Creating rules for disappearedEntity...
(1)  {8}  ->  {15}  0.8%
(2)  {3 7}  ->  {16}  0.8%
(3)  {7}  ->  {16}  0.73%

Creating rules for bc.d2.sensors.WallAheadSensor...
(1)  {3}  ->  {15}  0.95%
```

```
Creating rules for bc.d2.conditions.DestroyEnemyBaseGoal...
(1)  {12}  ->  {16}  1.0%
(2)  {13}  ->  {15}  0.67%
```

First of all, it is important to note that the first order inductive learning algorithm was unable to find any rules to classify EnemyInLineSensor, GetInLineWithEnemyGoal, BlockAheadSensor, PlayerBaseInLineSensor, EnemyBaseInLineSensor, and changedEntity. This can occur for at least two simple reasons: (1) the player did not encounter this situation during the traces that we trained upon, or (2) the algorithm was unable to find a conclusive classification.

Second, we should consider why we obtained some of the rules that we did. For example, WallAheadSensor is true when GetInLineWithEnemyBaseGoal is true. This makes logical sense because in the simple map we used (shown to the right), the enemy's base (illustrated by the eagle symbol on the right side of the map) is surrounded by a wall, so the player was forced to face the wall if he wanted to win by shooting the enemy's base. Since there is no other reason why the player would want to face a wall, it makes sense that if the player is facing the wall, he is also likely in line with the enemy's base.



Another rule that makes logical sense is the DestroyEnemiesGoal. DestroyEnemiesGoal is negative when BlockAheadSensor is true, which is logical because there is no possible way for a player to destroy an enemy and also have block ahead of his tank. Additionally DestroyEnemiesGoal is true when WinGameGoal is true, which makes sense because if the player destroys the enemy then he wins the game. The reader can examine additional rules by referencing the mappings for BattleCity on page eleven.

Now, consider the results obtained from the first order inductive learner after training it on ten traces in which a player played BattleCity merely attempting to win.

```
Creating rules for newEntity...
(1)  {7}  ->  {16}  0.72%
(2)  {8 12}  ->  {16}  0.71%
(3)  {3}  ->  {16}  0.69%
(4)  {8 13}  ->  {15}  0.5%
(5)  {13}  ->  {16}  0.5%
(6)  {8}  ->  {15}  0.36%
(7)  {3}  ->  {15}  0.31%
(8)  {7}  ->  {15}  0.28%

Creating rules for bc.d2.sensors.NextShotDelaySensor...
(1)  {7}  ->  {16}  0.82%
(2)  {8}  ->  {15}  0.8%
(3)  {3}  ->  {15}  0.35%

Creating rules for bc.d2.conditions.GetInLineWithEnemyBaseGoal...
(1)  {3}  ->  {15}  0.95%

Creating rules for bc.d2.conditions.DestroyEnemiesGoal...
(1)  {7}  ->  {16}  1.0%
(2)  {8}  ->  {15}  0.8%

Creating rules for bc.d2.sensors.NextMoveDelaySensor...
(1)  {7}  ->  {16}  0.82%
(2)  {8}  ->  {15}  0.8%
(3)  {3}  ->  {15}  0.35%

Creating rules for bc.d2.conditions.WinGameGoal...
(1)  {7}  ->  {16}  1.0%
(2)  {8}  ->  {15}  0.8%

Creating rules for disappearedEntity...
(1)  {8}  ->  {15}  0.8%
(2)  {3 7}  ->  {16}  0.8%
(3)  {7}  ->  {16}  0.73%

Creating rules for bc.d2.sensors.WallAheadSensor...
(1)  {3}  ->  {15}  0.95%

Creating rules for bc.d2.conditions.DestroyEnemyBaseGoal...
(1)  {12}  ->  {16}  1.0%
(2)  {13}  ->  {15}  0.67%
```

These results are exactly the same as those obtained when we only trained on four traces, except for the newEntity rules (see comparison chart in the Appendix). In BattleCity, a ‘new entity’ usually refers to a bullet being fired, which happens very haphazardly. Hence, we should not be concerned that the newEntity rules drastically

change as the number of traces trained upon varies. It is interesting that the results are otherwise the same between the run using four traces and the run using ten traces. This is most likely due to using traces that were created by a player exhibiting a consistent playing style. Despite the fact that the player exhibited different strategies in different traces – in some games he attacked the enemy's tank and in other he attacked the enemy's base – the player did not force himself to use either strategy and instead used the strategy that came most naturally during each game.

Alternatively, let us consider the results obtained from the first order inductive learner after training it on four traces in which a player played BattleCity and attempted to only win by attacking the enemy's tank.

```
Creating rules for newEntity...
(1)  {8}  ->  {15}  0.5%
(2)  {7}  ->  {15}  0.28%

Creating rules for bc.d2.sensors.NextShotDelaySensor...
(1)  {7}  ->  {16}  0.77%
(2)  {8}  ->  {15}  0.75%

Creating rules for bc.d2.conditions.DestroyEnemiesGoal...
(1)  {7}  ->  {16}  0.99%
(2)  {8}  ->  {15}  0.75%

Creating rules for bc.d2.sensors.NextMoveDelaySensor...
(1)  {7}  ->  {16}  0.77%
(2)  {8}  ->  {15}  0.75%

Creating rules for bc.d2.sensors.BlockAheadSensor...
(1)  {7}  ->  {15}  0.99%
(2)  {8}  ->  {15}  0.75%

Creating rules for bc.d2.conditions.WinGameGoal...
(1)  {7}  ->  {16}  0.99%
(2)  {8}  ->  {15}  0.75%

Creating rules for disappearedEntity...
(1)  {8}  ->  {15}  0.75%
(2)  {7}  ->  {15}  0.2%
```

Notice that the results are substantially different for this run than for the two earlier runs (see comparison chart in the Appendix). Additionally, note that the first

order inductive learner was unable to find any rules to classify EnemyInLineSensor, GetInLineWithEnemyBaseGoal, GetInLineWithEnemyGoal, PlayerBaseInLineSensor, WallAheadSensor, EnemyBaseInLineSensor, and DestroyEnemyBaseGoal in this run – two more than in each of the two runs discussed above. This is most likely because the above runs contained a mixture of two strategies (shoot at enemy tank and shoot at enemy base), but this run only contained one strategy (shoot at enemy tank). By only training on this one strategy, the learner is not exposed to situations such as destroying the enemy's base. As such, the learner cannot define rules to classify such unseen situations.

CHAPTER 7

CONCLUSION

Although this research project is not yet complete, it has shown that the first order inductive learning algorithm can successfully learn rules in BattleCity. Additionally, preliminary results have shown that the first order inductive learning algorithm will likely be able to also learn rules successfully in Towers.

The rules learned by the algorithm for BattleCity make logical sense when analyzed, and are generally similar to what we expected and hoped to obtain when we began this project. The original question of whether a version of the first order inductive learning algorithm can be used to learn rules that can then be used in place of a simulator has yet to be definitively answered. However, we have made a substantial step in positively answering the question of whether the first order inductive learning algorithm can learn rules in games such as BattleCity.

As will be discussed more in the Future Work chapter, we hope to more generally answer the question of whether the first order inductive learning algorithm can learn rules successfully with respect to all four games implemented for Darmok 2. We also hope to make more progress on the question of whether a version of the first order inductive learning algorithm can be used to learn rules that can then be used in place of a simulator.

CHAPTER 8

FUTURE WORK

I plan to continue working on this project during Summer 2009. During the summer I hope to determine whether the first order inductive learning algorithm can learn rules successfully with respect to all four games implemented for Darmok 2. If so, I hope to determine whether these rules can successfully be used in place of a manually-written simulator.

If the rules learned by the first order inductive learning algorithm can be used in place of a simulator, we must determine a fair metric for deciding whether the learned simulator is actually 'better' than the original manually-written simulator. Of course the learned simulator would be 'better' in certain ways - in that it does not require the time and effort of a manually-coded simulator and that it is able to be relearned almost instantly. However, the question still remains of whether the level of game play afforded by the learned simulator will be as strong as that of a manually-coded simulator.

It would also be productive to determine if other learning algorithms might be better suited for learning rules that can be used in place of a simulator, especially if the rules learned by the first order inductive learning algorithm are not suitable for replacing a manually-coded simulator.

Finally, there is the high arching goal of creating a general solution for building game artificial intelligences. Although this research still has a long way to go before reaching this, we certainly seem to be on the right track.

REFERENCES

[1] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In Fourth Artificial Intelligence and Interactive Digital Entertainment Conference.

[2] Frans Coenen. The LUCS-KSS Implementations of FOIL (First Order Inductive Learner). Webpage, 13 Febuary 2004. Accessed 7 March, 2009.

[3] Hilmar Finnsson and Yngvi Björnsson. Simulation-Based Approach to General Game Playing. In 23rd AAAI Conference on Artificial Intelligence, 2008.

[4] Sergio Jimenes, Fernando Fernandez, and Daniel Borrajo. The PELA Architecture: Integrating Planning and Learning to Improve Execution. In 23rd AAAI Conference on Artificial Intelligence, 2008.

[5] Tohgoroh Matsui, Nobuhiro Inuzuka, and Hirohisa Seki. A Proposal for Inductive Learning Agent Using First-Order Logic. In Inductive Logic Programming, 2000.

[6] Kinshuk Mishra, Santiago Ontañón, and Ashwin Ram. Situation Assessment for Plan Retrieval in Real-Time Strategy Games. In 9th European Conference on Case-Based Reasoning, 2009.

[7] Santi Ontañón, Kane Bonnette, Prafulla Mahindrakar, Marco A. Gómez-Martín, Katie Long, Jainarayan Radhakrishnan, Rushabh Shah and Ashwin Ram. Learning from Human Demonstrations for Real-Time Case-Based Planning. To appear in IJCAI STRUCK 2009.

[8] Santi Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Real-Time Case-Based Planning. Unpublished preprint, 2008.

[9] Santi Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-Based Planning and Execution for Real-Time Strategy Games. In ICCBR 2007, volume 4626 of Lecture Notes in Computer Science, pages 164-178. Springer Berlin / Heidelberg, 2007.

[10] J.R. Quinlan. Learning Logical Definitions from Relations. In Machine Learning 5, pages 239-266, 1990. Kluwer Academic Publishers, 1990.

[11] J.R. Quinlan and R.M. Cameron-Jones. FOIL: A Midterm Report. In Proceedings of the European Conference on Machine Learning, volume 667 of Lecture Notes in Computer Science, pages 3-20. Springer-Verlag, 1993.

[12] Ashwin Ram, Santiago Ontañón, and Manish Mehta. Artificial Intelligence for Adaptive Computer Games. In Twentieth International FLAIRS Conference on Artificial Intelligence, 2007.

[13] Neha Sugandh, Santiago Ontañón, and Ashwin Ram. On-Line Case-Based Plan Adaptation for Real-Time Strategy Games. In 23rd AAAI Conference on Artificial Intelligence, 2008. pages 702-707.

[14] Andrew Trusty, Santiago Ontañón, and Ashwin Ram. Stochastic Plan Optimization in Real-Time Strategy Games. In 4th Conference on Artificial Intelligence and Interactive Digital Entertainment, 2008.

[15] Thomas J. Walsh and Michael L. Littman. Efficient Learning of Action Schemas and Web-Service Descriptions. In 23rd AAAI Conference on Artificial Intelligence, 2008.

[16] James Westendorp. Noise-Resistant Incremental Relational Learning Using Possible Worlds. In ILP 2002, number 2583 in LMAI, pages 317-332. Springer-Varlag Berlin Heidelberg, 2003.

APPENDIX

The following is a comparison chart of the rules produced by the first order inductive learner after training on (1) four traces in which a player played BattleCity merely attempting to win (general strategy), (2) ten traces in which a player played BattleCity merely attempting to win (general strategy), and (3) four traces in which a player played BattleCity and attempted to only win by attacking the enemy's tank (attack-enemy strategy).

newEntity

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{3} -> {16} 0.8%	{7} -> {16} 0.72%	{8} -> {15} 0.5%
{8} -> {16} 0.8%	{8 12} -> {16} 0.71%	{7} -> {15} 0.28%
{7} -> {15} 0.29%	{3} -> {16} 0.69%	
	{8 13} -> {15} 0.5%	
	{13} -> {16} 0.5%	
	{8} -> {15} 0.36%	
	{3} -> {15} 0.31%	
	{7} -> {15} 0.28%	

bc.d2.sensors.NextShotDelaySensor

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{7} -> {16} 0.82%	{7} -> {16} 0.82%	{7} -> {16} 0.77%
{8} -> {15} 0.8%	{8} -> {15} 0.8%	{8} -> {15} 0.75%
{3} -> {15} 0.35%	{3} -> {15} 0.35%	

bc.d2.conditions.GetInLineWithEnemyBaseGoal

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{3} -> {15} 0.95%	{3} -> {15} 0.95%	n/a

bc.d2.conditions.DestroyEnemiesGoal

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{7} -> {16} 1.0%	{7} -> {16} 1.0%	{7} -> {16} 0.99%
{8} -> {15} 0.8%	{8} -> {15} 0.8%	{8} -> {15} 0.75%

bc.d2.sensors.NextMoveDelaySensor

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy

{7} -> {16} 0.82%	{7} -> {16} 0.82%	{7} -> {16} 0.77%
{8} -> {15} 0.8%	{8} -> {15} 0.8%	{8} -> {15} 0.75%
{3} -> {15} 0.35%	{3} -> {15} 0.35%	

bc.d2.sensors.BlockAheadSensor

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
n/a	n/a	{7} -> {15} 0.99%
		{8} -> {15} 0.75%

bc.d2.conditions.WinGameGoal

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{7} -> {16} 1.0%	{7} -> {16} 1.0%	{7} -> {16} 0.99%
{8} -> {15} 0.8%	{8} -> {15} 0.8%	{8} -> {15} 0.75%

disappearedEntity

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{8} -> {15} 0.8%	{8} -> {15} 0.8%	{8} -> {15} 0.75%
{3 7} -> {16} 0.8%	{3 7} -> {16} 0.8%	{7} -> {15} 0.2%
{7} -> {16} 0.73%	{7} -> {16} 0.73%	

bc.d2.sensors.WallAheadSensor

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{3} -> {15} 0.95%	{3} -> {15} 0.95%	n/a

bc.d2.conditions.DestroyEnemyBaseGoal

4-trace General strategy	10-trace General strategy	4-trace Attack-Enemy strategy
{12} -> {16} 1.0%	{12} -> {16} 1.0%	n/a
{13} -> {15} 0.67%	{13} -> {15} 0.67%	