Reflection in Action:
Model-Based Self-Adaptation in Game Playing Agents

Patrick Ulam, Ashok Goel, Joshua Jones

College of Computing
Georgia Institute of Technology
Atlanta, USA 30332
{pulam, goel, jkj}@cc.gatech.edu

Abstract

Computer war strategy games offer a challenging domain for AI techniques for learning because they involve multiple players, the world in the games is only partially observable and the state space is extremely large. Model-based reflection and self-adaptation is one method for learning in such a complex domain. In this method, the game-playing agent contains a model of its own reasoning processes. When the agent fails to win a game, it uses its self-model and (possibly) traces of its execution to analyze the failure and modify its knowledge and reasoning accordingly. In this paper, we describe an experimental investigation of model-based reflection and self-adaptation for a specific task (defending a city) in a computer war strategy game called Civilization. Our results indicate that at least for limited tasks, model-based reflection enables effective learning, and further, when traces are used in conjunction with the model, the effectiveness of learning appears to increase with the size of the trace.

Game Playing

As AI techniques for learning become increasingly sophisticated, it becomes feasible to operate AI agents in increasingly complex environments. There are, however, many environments whose complexity outstrips the ability of current AI techniques to guide the agent to take proper actions. This problem is particularly evident in game playing.

Many game-playing agents use traditional search techniques such as minimaxing (VonNeumann and Morgenstern 1944). Some agents, such as TD-Gammon, use reinforcement learning to determine the proper moves to make for a given game state (Tesauro 1995). Both of these techniques enable optimal game playing under certain circumstances and hence may be a good choice for some games. Unfortunately, these techniques also have some drawbacks. The biggest drawback of these techniques is their computational complexity when dealing with large state spaces even in relatively straight-forward two-player, fully observable games such as chess. At present, it is unclear whether these techniques can be scaled up to more complex two-player fully-observable games such as GO, let alone multi-player partially-observable games.

Model-Based Reflection and Self-Adaptation

Recently AI has developed a set of techniques for model-based introspection/reflection. In this general technique, an agent is endowed with a model of its reasoning processes. The agent uses its self-model to analyze its experiences and to make modifications to its knowledge and reasoning accordingly. This general technique has been used in domains ranging from route planning (Fox and Leake 1995; Stroulia and Goel 1994, 1996) to assembly and disassembly planning (Murdock and Goel 2001, 2003) to playing chess (Krulwich et al 1992). It has proved useful for learning new concepts (Krulwich et al 1992), improving case indexing (Fox and Leake 1995), reorganizing domain knowledge and reconfiguring the task structure (Stroulia and Goel 1994, 1996), and adapting and transferring the domain knowledge and the task structure to new problems (Murdock and Goel 2001, 2003).

The goal of this work is to investigate the use of model-based reflection in learning to play multi-player games in partially observable worlds with extremely large state spaces. In order to conduct experiments, we needed to adopt a specific model-based reflection technique. Murdock (2001) describes in detail a knowledge-based shell called the Reflective Evolutionary Mind (REM) that uses a knowledge-representation called the task-method-knowledge language (TMKL). An agent, such as a game-playing agent, is encoded in TMKL, which specifies the task-method-knowledge structure of the agent. A task is specified by the types of information it takes as input and gives as output. A task may have multiple methods applicable to it. A method decomposes a task into subtasks and specifies the control of processing over the subtasks. The primitive tasks at the lowest level of this recursive task-method decomposition either correspond to an action in the world or an associated knowledge-based procedure for accomplishing them. Thus, a TMKL model of an agent specifies its reasoning architecture, including the tasks the agent accomplishes and the knowledge it uses to accomplish them.

REM reflects on an agent encoded in TMKL. In particular, it uses the TMKL model and the trace of the agent's processing to localize the modification needed to an agent to address a failure or to accomplish a new task. A trace is a record of the agent's processing in the execution of a particular task, and is expressed in terms of the sequence of subtasks that were executed and the knowledge states each subtask generated. Thus the trace specifies the actual internal processing of the agent. When the agent fails at a particular task, REM uses information about the failure, the trace of processing, and the TMKL model to localize the modifications needed to agent to address the failure.

Once REM has localized the modification to a portion of the task-method-knowledge structure of the agent, it identifies a specific modification and executes it. Its set of general adaptation methods include failure-driven adaptation, generative planning, and reinforcement learning. Thus, REM combines model-based reflection with both generative planning and reinforcement learning. However, in the experiments reported in this paper, we used only REM's method of failure-driven adaptation.

The Civilization Game

We chose a popular computer war strategy game called Civilization as the domain for this work. Many incarnations of this game exist ranging from several commercial variations designed for play on a computer, board game variations, as well as a free variant available in the public domain called FreeCiv that was used in this work. Civilization is a multiple-player game in which a player competes against several software agents that come with the game. Each player takes control of a civilization from its primitive beginnings at the start of the game and, as the game progresses, develops the early civilization into a modern civilization at the termination of the game. As the game continues, each player explores the world and learns more about it and also encounters other players. Each player can make alliances with other players, or attack the other players while defending its own assets from them. In the course of a game (which can take a few hours to play), each player makes a large number of decisions for his civilization ranging from when and where to build cities on the playing field, what sort of infrastructure to build within the cities and between the civilization's cities, to how to defend the civilization. Civilization belongs to the genre of war strategy games that are often called 4X games, where the four X's represent the four main tasks in the game: exploration, expansion, exploitation, and military extermination.

We chose FreeCiv as the domain for a number of reasons. As mentioned above FreeCiv is a multi-player game with a partially-observable world. In addition, FreeCiv has a huge state space making it intractable for general search based or reinforcement learning techniques; hence, an agent designed to play the game would need to use techniques to reduce the complexity of adapting the agent within the game so as to make the problem tractable. Since FreeCiv is an incredibly complex game, we believe that it will need a modular agent design, where each module may encapsulate smaller, simpler, more easily addressed aspects of game play.

The Defend City Task

Due to the extremely complex nature of the Civilization game, this work addresses only one component of the game to serve as a means of determining the effectiveness of model-based reflection and self-adaptation. The component addressed in this work deals with the defense of one of the agent's cities from enemy civilizations. This task was chosen for several reasons. The task is important to the creation of a general purpose civilization playing agent in that the player's cities are the cornerstone in the player's civilization. Most actions in the game consist of building, upgrading, and fighting for cities. This task is also common enough such that the agent must make numerous decisions concerning the proper defense of the city during the time span of a particular game.

Figure 1 illustrates a TMKL model of the city defense task. The overall *Defend City* task can be decomposed into two sub-tasks: the evaluation of the defense needs for a city and the building of a particular structure or unit at that city. The *Evaluate Defense Need* task can be further decomposed into two additional subtasks, a task to check internal factors in the city for defensive requirements and a task to check for factors external to the immediate vicinity of the city for defensive requirements. The *Defend City* task is executed at different times throughout the game. It is executed each turn that the agent is not building a defensive unit in a particular city in order to determine if production should be switched to a defensive unit. It is also executed whenever a city has finished production of a particular building or unit.

The internal evaluation task utilizes knowledge concerning the current number of troops that are positioned in a particular city to determine if the city has an adequate number of defenders barring any extraneous circumstances. The external evaluation of a city's defenses examines the area around a city for a specified radius for nearby enemy combat units. It utilizes the knowledge of the number of units, their distance from the city, and the number of units currently allocated to defend the city in order to provide an evaluation of the need for additional defense. These evaluation tasks are represented as linear value functions, the results of which are knowledge states which are then utilized by the agent to build the appropriate item at the city. The *Build Defense* task utilizes the knowledge states generated by the evaluation subtasks along with knowledge concerning the current status of the build queue, as well as the technology currently known by the agent to determine what should be built for a given iteration of the task. The *Build Defense* task then proceeds to build a defensive unit appropriate to

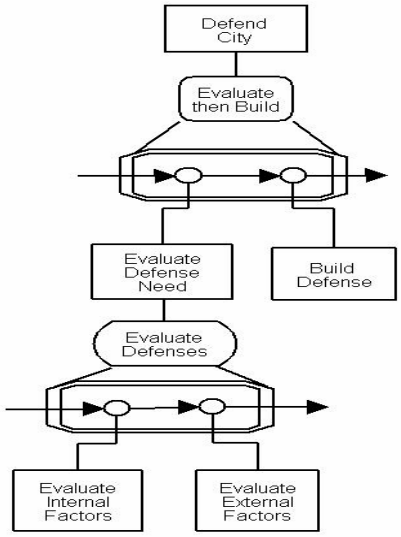

Figure 1. *Defend City* Task Model

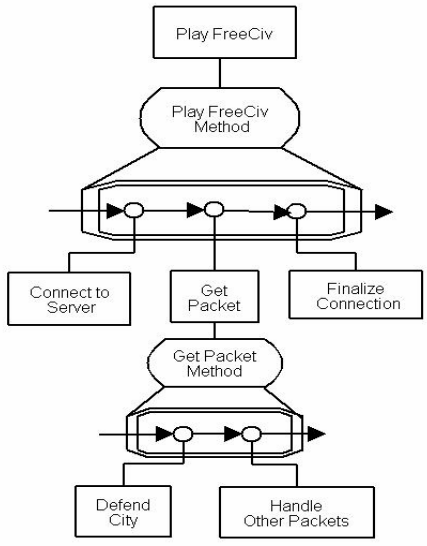

Figure 2. FreeCiv Agent Model

the civilization's technological level or wealth to keep the citizens of the city happy.

The goal of the *Defend City* task is to provide for the defense of a city for a certain number of years. The task is considered successful if the city has not been conquered by opponents by the end of this time span. If an enemy takes control of the city, the task is considered a failure. In addition, if the city enters civil unrest, a state in which the city revolts because of unhappiness, usually caused due the neglect of infrastructure in a particular city, then again the task is considered to have failed.

Implementation

We developed a simple software agent, called the FreeCiv agent, capable of playing the Civilization game in the REM/TMKL framework. Figure 2 illustrates the working of the FreeCiv agent. Note that the *Defend City* task illustrated in Figure 1 is a subtask of the *Get Packet* task illustrated in Figure 2 as it is executed as a response to certain packets received.

A number of scalability problems arose while integrating the FreeCiv agent with the REM system. In order to reduce both memory and time overhead, we directly endowed the FreeCiv agent with a model of the *Defend City* task. In addition, the execution trace for these tasks in these models was created and analyzed within the FreeCiv framework, bypassing REM's trace generation and analysis mechanism. This enabled rapid execution of the agent. The entire trace was kept within memory for the lifetime of the agent. When a failure is detected, the execution of the agent is stopped and a routine for the analysis of the trace is called. The analysis performed is a variant of REM's failure-driven model transfer. Figure 3 illustrates the high-level algorithm for failure-driven adaptation. The first step is the localization of the cause for the failure through the use of information about the failure and the model of the failed task. Information about the failure is simply the type of failure that occurred. Using this information, the model is analyzed to determine in which task the failure has occurred. Given the trace and the location of the model in which the cause for the failure is suspected to lie, the agent then analyzes the execution traces available to it to determine to the best of its ability where precisely in the given portion of the model the error may have occurred. This is done through the use of a failure library containing common failure conditions found within the *Defend City* task. Figure 4 depicts the failure library for the *Defend City* task. Failure types for specific levels in that task hierarchy are shown. Specific failures for a given task may be indexed through trace analysis. If a failure has been determined to occur, it is then used to index into a library of adaptation strategies that will modify the task such in the manner indicated by the library. Figure 5 illustrates the library of adaptation strategies used in the *Defend City*. If multiple errors are found within the trace analysis, a single error is chosen probabilistically so as to minimize the chance of over-adaptation of the agent.

Experimental Setup

To determine the effectiveness of using model based reflection within the FreeCiv game and more specifically the *Defend City* task, a series of experiments were conducted. In each individual experiment, twenty-five games of FreeCiv were played by the agent. Each game was conducted with a total of 8 players including the FreeCiv agent described above. The trials were set to run for one hundred turns at the hardest difficulty level. If the task was successful no adaptation occurred. If during the trial, the agent's city is conquered or the city's citizens revolt, the *Defend City* task is considered failed. When a

Algorithm: **failure-driven-model-analysis-adaptation** (model, trace, feedback)	
Inputs:	model: Model of task to be analyzed
	trace: Portion of the execution traces of model to be analyzed
	feedback: Feedback received concerning task failure
Outputs:	Task model with possible modifications
Effects:	Possible modification to given model to address diagnoses causes of failure in task execution

```
failure-driven-model-analysis-adaptation (model, trace, feedback)
  failure-location = localize-failure(model, feedback)
  for each task execution in trace
     execution-instance = next-trace-instance(trace)
     error-type = index-failure-library(execution-instance, failure-location)
     add-hypothesis(error-type)
  error-selection = select-hypotheses()
  adapted-model = adaptation-model(error-selection, model)
  return adapted-model
```

Figure 3. Algorithm for failure driven model analysis and adaptation

Model Location (*task*)	**Types of Failures (*indexed via traces)***
Defend-City	*Unit-Build-Error, Wealth-Build-Error, Citizen-Unrest-Misevaluation, Defense-Present-Misevaluation, Proximity-Misevaluation, Threat-Level-Misevaluation, None*
Evaluate- Defense-Need	*Citizen-Unrest-Misevaluation, Defense-Present-Misevaluation, Proximity-Misevaluation, Threat-Level-Misevaluation, None*
Build-Defense	*Unit-Build-Error, Wealth-Build-Error, None*
Evaluate-Internal-Factors	*Citizen-Unrest-Misevaluation, Defense-Present-Misevaluation, None*
Evaluate-External-Factors	*Proximity-Misevaluation, Threat-Level-Misevaluation, None*

Figure 4. Failure library used for the *Defend City* task by model location

Adaptation Strategies	**Effects**
Increase Search Radius	Increases the radius in which to search for enemies
Decrease Search Radius	Decreases the radius in which to search for enemies
Increase Base Defense	Increases the number of defenders to build regardless of outside factors
Decrease Base Defense	Decreases the number of defenders to build regardless of outside factors
Increase Enemy Activity Threshold	Increases the threshold for the number of enemies nearby required to generate response
Decrease Enemy Activity Threshold	Decreases the threshold for the number of enemies nearby required to generate response
Increase Ally Proximity Radius	Increases the radius in which to account for ally proximity in defense evaluation
Decrease Ally Proximity Radius	Decreases the radius in which to account for ally proximity in defense evaluation

Figure 5. Library of adaptation strategies available for *Defend City* task

failure occurs, execution of the task is halted and the reflection sequence is initiated. The trace of execution is examined for the failure point. In addition to the trace of the task execution generated at the time of failure, the agent could also utilize the trace of up to the last ten executions of the *Defend City* task. We also conducted a series of control trials lesioning the ability of the agent to adapt via reflection. In a third experimental condition, the agent reflected upon only the failure that occurred without trace information. The number of successful trials for each configuration was measured as well as the number of trials in which failures were present and adaptation occurred. The resulting weights in the linear value function utilized for the *Evaluate Internal Factors* and *Evaluate External Factors* tasks were recorded.

Result Summary

A summery of the results can be seen in Figure 6. All three agent configurations (no ability to adapt – the control condition, model-based adaptation only, and model and trace-based adaptation) complete several trials but the number of successful trials increases with the amount of trace utilized in the analysis of task failures. Interestingly, the trials in which the model was the only means of aiding reflection resulted in equal or better performance than all trials except for those trials that utilized significant amounts of trace. The increase in the number of task execution traces utilized corresponds

with a similar increase in the number of adaptation strategies used in model-based adaptation. In the trials in which only the model was utilized, the model enabled localization of the causes of the failure but did not necessarily identify the failure. In this condition, causes for failures were chosen randomly from the failure library based upon where the failure was localized in the model. This resulted in unusually high levels of adaptation, many of which proved beneficial. This seems to indicate that reflection with only the model and feedback may prove effective when the number of failures remains manageable.

This is because the agent is able to localize the error with sufficient accuracy to determine a point of modification. This localization alone illustrates one of the advantages of such a model-based approach. The reduction in applicable adaptation strategies due to the localization allows for significant advantages in selecting a specific modification (even if done at random). On the other hand, without execution traces to illustrate how the agent actually executed the task within the context of its current knowledge states, it is forced to assume the failure was due to the improper execution of the task and select an adaptation strategy regardless of whether the task was executed properly or not. It was indeed noted that although performance of the methods of model-based adaptation and model-based adaptation with use of traces appears to be equally good in some sense, the specific modifications made by the two methods are quite different. The lack of available trace information results in an overall increase in the number of modifications made, which may prove detrimental as the complexity of the tasks addressed increases.

Conclusions and Future Work

The results from the experiments that utilized reflection to guide the learning of an agent in a subset of the FreeCiv game indicate that model-based reflection is an effective means in which an agent can learn to act in highly complex, partially observable environments. The ability to localize the failure within the task model coupled with the utilization of task execution traces to further narrow the space of possible adaptations allow for significant performance improvements in the task described herein. The results also show that the larger the number of the task execution traces analyzed, the better the performance of that agent in the *Defend City* task. Future work will focus on using the localization capabilities of model-based reflection to allow the agent to determine where in the state space adaptation must occur and then utilizing numerical machine learning techniques upon this reduced state space thus eliminating the adaptation libraries.

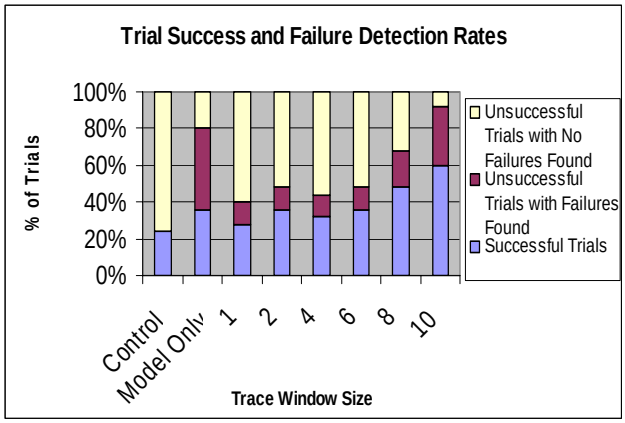

Figure 6. Success and failure rates for trials performed with varying trace window sizes. Failed trials further illustrated as trails in which model based reflection found an error and trails in which it did not.

References

Bates, M. A.; Bowden, B. V.; Strachey, C.; and Turing, A. M. 1953. "*Digital Computers Applied to Games*" in B.V. Bowden, ed., Faster Than Thought, Pitman, 286-310.

Fox, S. and Leake, D.B. 1995. Learning to Refine Indexing by Introspective Reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada: Morgan Kaufmann.

Krulwich, B., Birnbaum, L., and Collins, G. 1992. Learning Several Lessons from One Experience. In *Proceedings of the Forth Annual Conference of the Cognitive Science Society*, 242-247. Bloomington, In.

Murdock, J. W. and Goel, A. K. 2001. Meta-Case-Based Reasoning: Using Functional Models to Adapt Case-Based Agents. In *Proceedings of the 4th. International Conference on Case-Based Reasonin*g, 407-421. Vancouver, Canada.

Murdock, J. W. and Goel, A. K. 2003. Localizing Planning with Functional Process Models. In *Proceedings of the Thirteenth International Conference on Automated Planning & Scheduling*. Trento, Italy.

Murdock, J. W. 2001 Self-Improvement through Self-Understanding: Model-Based Reflection for Agent Adaptation, Ph.D. Thesis, College of Computing, Georgia Institute of Technology.

Stroulia, E. and Goel, A.K. 1994. Learning Problem-Solving Concepts by Reflecting on Problem Solving. In *Proceedings of the 1994 European Conference on Machine Learning*. p. 287-306.

Stroulia, E. and Goel A.K. 1996. A Model-Based Approach to Blame Assignment: Revising the Reasoning Steps of Problem Solvers. *Proceedings of AAAI'96*, 959-965. AAAI Press.

Von Neumann, J. and Morgensten, O. 1944. *The Theory of Games and Economic Behavior*. Princeton University Press, New Jersey, 1980 ed.