Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game

David W. Aha[1], Matthew Molineaux[2], and Marc Ponsen[3]

[1] Navy Center for Applied Research in Artificial Intelligence,
Naval Research Laboratory (Code 5515), Washington, DC 20375
[2] ITT Industries, AES Division, Alexandria, VA 22303
[3] Department of Computer Science and Engineering,
Lehigh University, Bethlehem, PA 18015
[1,2] {first.last}@nrl.navy.mil [3] mjp304@lehigh.edu

Abstract. While several researchers have applied case-based reasoning techniques to games, only Ponsen and Spronck (2004) have addressed the challenging problem of learning to win real-time games. Focusing on WARGUS, they report good results for a genetic algorithm that searches in plan space, and for a weighting algorithm (*dynamic scripting*) that biases subplan retrieval. However, both approaches assume a static opponent, and were not designed to transfer their learned knowledge to opponents with substantially different strategies. We introduce a plan retrieval algorithm that, by using three key sources of domain knowledge, removes the assumption of a static opponent. Our experiments show that its implementation in the Case-based Tactician (CAT) significantly outperforms the best among a set of genetically evolved plans when tested against random WARGUS opponents. CAT communicates with WARGUS through TIELT, a testbed for integrating and evaluating decision systems with simulators. This is the first application of TIELT. We describe this application, our lessons learned, and our motivations for future work.

1 Introduction

Research on artificial intelligence (AI) and games has an extraordinary history that dates from 1950. Several luminaries have contributed to this field, and automated game-playing programs now exist that outperform world champions in classic games such as checkers, Othello, and Scrabble (Schaeffer, 2001). These efforts brought about significant advancements in search algorithms, machine learning techniques, and computer hardware. As a barometer of continued strong interest, several conferences (e.g., *International Game-On Conference on Computer Games: AI, Design and Education*, *AI and Interactive Digital Entertainment*) and journals (e.g., *Journal of Intelligent Games and Simulation*, *Journal of Game Development*) are devoted to AI and games.

In recent years, AI researchers (e.g., Laird & van Lent, 2001; Buro, 2003) have begun focusing on complex strategy simulation games that offer a variety of challenges, including partially observable environments that contain adversaries who

H. Muñoz-Avila and F. Ricci (Eds.): ICCBR 2005, LNAI 3620, pp. 5 – 20, 2005.

modify the game state asynchronously, and whose decision models are unknown. Among these, games that simulate the evolution of civilizations are particularly intriguing due to their enormous state spaces, large decision spaces with varying abstraction levels, multiple decision threads (e.g., economy, combat), and their need for resource management processes.

Although many studies exist on learning to win classical board games and other games with comparatively smaller search spaces, few studies exist on learning to win complex strategy games. Some argue that agents require sophisticated representations and reasoning capabilities to perform competently in these environments, and that these representations are challenging to construct (e.g., Forbus *et al.*, 2001). Fortunately, sufficiently good representations exist for a small number of gaming environments. In particular, Ponsen and Spronck (2004) developed a lattice for representing and relating abstract states in WARGUS, a moderately complex real-time strategy game. They also sharply reduced the decision space by employing a high-level language for game agent actions. Together, these constrain the search space of useful plans and state-specific subplans (i.e., *tactics*). This allowed them to focus on an ambitious performance task: winning real-time strategy games. They reported good results for a genetic algorithm that learns complete plans, and for a weight-learning algorithm (*dynamic scripting*) that learn policies for selecting tactics that combine into successful plans. However, both approaches assume a fixed adversary, and were not designed to transfer learned knowledge so as to defeat opponents that use dissimilar strategies.

In this paper, we relax the assumption of a fixed adversary, and develop a case-based approach that learns to select which tactic to use at each state. We implemented this approach in the Case-based Tactician (CAT), and report learning curves that demonstrate its performance quickly improves with training even though the adversary is randomly chosen for each WARGUS game. CAT is the first case-based system designed to win against random opponents in a real-time strategy game.

We briefly review case-based approaches in games research and introduce WARGUS in Section 2. We detail our approach and CAT in Section 3. We review our empirical methodology and CAT's results in Section 4, and close with a discussion in Section 5 that mentions several future research objectives.

2 Background

2.1 Case-Based Reasoning Research in Games

Many taxonomies exist for distinguishing computer games. For example, Laird and van Lent (2001) distinguish game genres into action (e.g., WOLFENSTEIN™), adventure, role-playing (e.g., BALDUR'S GATE™), strategy, god (e.g., SIMCITY™), team sports (e.g., MADDEN NFL FOOTBALL™) and individual sports games. Fairclough et al.'s (2001) taxonomy differs slightly, in that they classify god games as a sub-type of strategy games, and place THE SIMS™ in a different category. We instead adopt a taxonomy that is biased by our reading of case-based reasoning (CBR)

Table 1. A partial summary on applying CBR to games

Genre	Examples	Description	Approaches	Performance Task
Classic board	chess (Kerner, 1995), checkers (Powell *et al.*, 2004), Othello (De Jong & Schultz, 1988)	n^2 board, 2-person, no uncertainty	rote learning, min-max, case acquisition	learn evaluation function, move selection, win
Adventure	Bonji's Adventures in Calabria (Fairclough & Cunningham, 2004)	puzzle-solving	planning & dialogue generation	storyline/plot generation and management
Team Sports	RoboCup Soccer (Wendler & Lenz, 1998; Gabel & Veloso, 2001; Wendler *et al.*, 2001; Karol *et al.*, 2003)	real-time multi-agent coordination and planning	various	identify preference location, passing, action selection, select team members
Real-time individual	Bilestoad™ (Goodman, 1994), Space Invaders™ (Fagan & Cunningham, 2003)	real-time single character planning	projective visualization, plan recognition	inflict/avoid damage, plan recognition
Real-time God	SimCity™ (Fasciano, 1996)	real-time single city management	plan adaptation	planning
Discrete Strategy	Freeciv (Ulam *et al.*, 2004)	turn-based civilization management	reflection-guided plan failure recovery	defend a city
Real-time strategy	**Wargus** (Cheng & Thawonmas, 2004; this paper)	**real-time limited civ. management**	hierarchical cases, **plan selection**	manage sub-tasks, **win**

research in games. In particular, we add categories that reflect more traditional games, ignore some that have not attracted strong CBR interest (e.g., action games), and refine categories of real-time games.

Many researchers have published work on CBR in games. We distinguish a subset of this research according to task characteristics (i.e., game genre, state and decision space complexity, adversarial presence, timing constraints, performance task) and CBR approach. Table 1 summarizes some of this work.

Several researchers have addressed classic board games, beginning with Arthur Samuel's (1959) rote learning approach for playing checkers. De Jong and Schultz's (1988) GINA instead memorized a partial game tree for playing Othello. Chess has been a popular topic. For example, Kerner (1995) described a method for learning to evaluate abstract patterns. More recently, Powell et al.'s (2004) CHEBR learned to play checkers given only a paucity of domain knowledge. While these efforts focused on adversarial games, they are turn-based rather than real-time, and have comparatively small decision complexities (see Section 2.2).

Case-based approaches have rarely been used in adventure games. However, this genre may become a rich focus for automated story plot generation. For example, Fairclough and Cunningham (2004) described OPIATE, which uses a case-based planner and constraint satisfaction to provide *moves* for a story director agent so as to ensure that characters act according to a coherent plot. Also, Díaz-Agudo et al. (2004) described a knowledge-intensive approach that extracts constraints from a user's interactively-provided specification, uses them to guide case retrieval and adaptation, and then creates a readable plot using natural language generation techniques. These projects extend earlier CBR work on story generation (e.g., Meehan, 1981).

Fig. 1. A screen shot of a WARGUS game

ROBOCUP SOCCER is a popular CBR focus. Wendler and Lenz (1998) described an approach for identifying where simulated agents should move, while Wendler et al. (2001) reported strategies for learning to pass. Gabel and Veloso (2001) instead used a CBR approach to select (heterogeneous) members for a team. Karol et al. (2003) proposed a case-based action selection algorithm for the 4-legged league. While these real-time environments are challenging strategically, they do not involve complicating dimensions common to strategy games, such as economies, research, and warfare.

Some researchers have addressed real-time individual games. Goodman (1994) applied a projective visualization approach for BILESTOAD, a personal combat game, to predict actions that would inflict damage on the adversary and/or minimize damage to oneself. Fagan and Cunningham (2003) instead focused on a plan recognition task; they acquire cases (state-action planning sequences) for predicting the next action of a human playing SPACE INVADERS™. In contrast, CAT does not perform projective modeling, and does not learn to recognize adversarial plans. Instead, it acquires cases concerning the application of a subplan in a given state, learns to select subplans for a given state, and executes them in a more complex gaming environment.

Fasciano's (1996) MAYOR learns from planning failures in SIMCITY™, a real-time city management game with no traditional adversaries. MAYOR monitors planning expectations and employs a causal model to learn how to prevent failure repetitions, where the goal is to improve the ratio of successful plan executions. In contrast, CAT does not employ plan monitoring or causal goal models, and does not adapt retrieved plans. Rather, it simply selects, at each state, a good tactic (i.e., subplan) to retrieve. Also, our gaming environment includes explicit adversaries.

Ulam et al. (2004) described a meta-cognitive approach that performs failure-driven plan adaptation for FREECIV, a complex turn-based strategy game. While they employed substantial domain knowledge in the form of task models, it was only

enough to address a simple sub-task (defending a city). In contrast, CAT performs no adaptation during reuse, but does perform case acquisition. Also, CAT focuses on winning a game rather than on performing a subtask.

2.2 Reducing the Decision Complexity of WARGUS: A Real-Time Strategy Game

In this paper, we focus on WARGUS (Figure 1), a real-time strategy (RTS) game that is a clone of the popular commercial game WARCRAFT II™. WARGUS uses STRATAGUS, an open-source engine for building RTS games. WARGUS is an excellent environment for AI research because its fairly mature code can be modified for experimentation.

RTS games usually focus on military combat (versus one or more adversaries), although they also include decision dimensions concerning tasks such as exploration, economic development, research advancement, and limited diplomacy. For example, WARCRAFT™, AGE OF EMPIRES™, and EMPIRE EARTH™ require players to control armies (of multiple unit types) and defeat all opponents in real-time.

Humans and adversaries can use any available *action* to form their game *strategy*, which is a plan. Typical actions include selecting a building to construct, researching a specific new technology, setting a destination for a selected group, and assigning a task to a group (e.g., construct a building). Humans are limited to executing a single new action at any one moment, while existing actions continue to execute simultaneously. Typically, RTS games provide users with a varying set of opponent strategies, each encoded as a *script.* A subplan in these scripts is called a *tactic*.

In addition to having relatively a large state space (e.g., we experiment with a 128x128 map that can involve dozens of units and buildings), WARGUS' decision space is comparatively large. An analysis of this complexity requires some understanding of the game. Winning (i.e., by destroying all the enemy units and buildings) requires managing three key resources: buildings, the workforce, and an army. Spending too little time on the army can lead to a crushing defeat at the hands of a strong neighbor, while spending too much time will cause a lag in research accomplishments, which prevent you from creating army units that are as strong as your neighbors. A balance must be maintained among these three resources. To do this, successful WARGUS players execute orders in one location, hurry to another, and try to return attention to the first location before its orders have terminated.

The decision space is the set of possible actions that can be executed at a particular moment. We estimate this as $O(2^W(A*P) + 2^T(D+S) + B(R+C))$, where W is the current number of workers, A is the number of assignments workers can perform (e.g., create a building, gather gold), P is the average number of workplaces, T is the number of troops (fighters plus workers), D is the average number of directions that a unit can move, S is the choice of troop's stance (i.e., stand, patrol, attack), B is the number of buildings, R is the average choice of research objectives at a building, and C is the average choice of units to create at a building. For the simple early game scenario shown in Figure 1 (which includes some off-screen troops and an off-screen building), this estimate yields a decision complexity of 1.5×10^3, which is substantially higher than the average number of possible moves in many board games (e.g., for chess, this is approximately 30).

Standard domain knowledge (e.g., cannot attack now, need more wood for building) could reduce the number of sensible choices for a WARGUS player in Figure

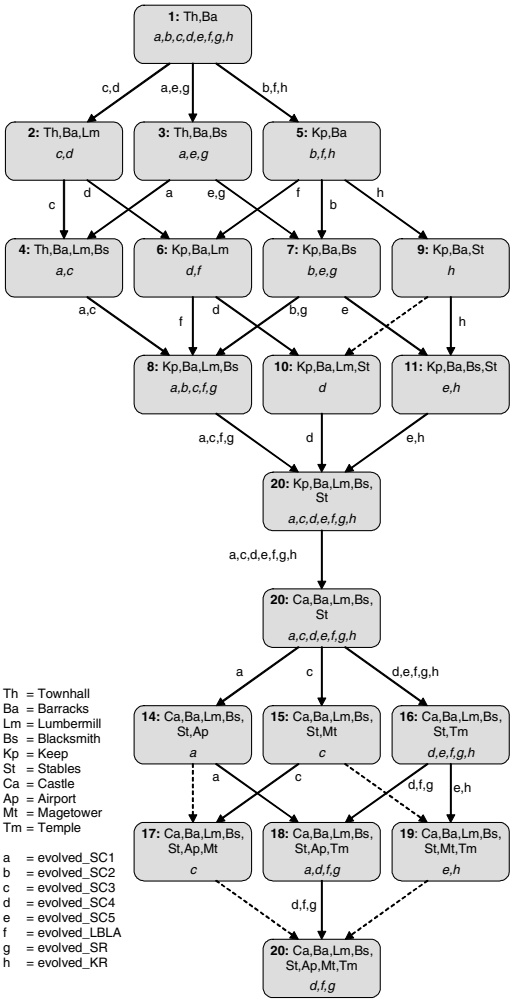

Fig. 2. A building-specific state lattice for WARGUS, where nodes represent states (defined by a set of completed buildings), and state transitions involve constructing a specific building. Also displayed are the evolved counter-strategies (a-h) that pass through each state

1's scenario to roughly ten. However, acquiring, encoding, and using this knowledge is challenging. Thus, existing research efforts on RTS games often focus on simpler tasks. For example, Guestrin et al. (2003) applied relational Markov decision process models for some limited WARGUS scenarios (e.g., 3x3 combat). They did not address more complex scenarios because their planner's complexity grows exponentially with the number of units. Similarly, Cheng and Thawonmas (2004) proposed a case-based plan recognition approach for assisting WARGUS players, but only for low-level management tasks. Their state representation is comprehensive and incorporates multiple abstraction levels. In contrast, CAT employs a simple case representation yet focuses on the complete task of winning the game.

To do this, CAT employs three significant sources of domain knowledge, the first two of which were developed by Ponsen and Spronck (2004), who used dynamic scripting to learn to win WARGUS games against a static opponent from a fixed initial state. The first source, *a state lattice*, is an abstraction of the state space, while the second source, *a set of tactics for each state*, is an abstraction of the decision space.

Figure 2 displays their building state lattice. Consisting of 20 states, it defines sequences of building constructions that can occur during a WARGUS game, where each state corresponds to the types of constructed buildings, which in turn determine the unit types that can be trained and technologies that can be researched. State changes occur when a tactic creates a new building. For example, starting with a Town Hall and a Barracks, the next building choices are a Lumber Mill, a Blacksmith, and a Keep, which replaces the Town Hall. Building these cause transitions from State 1 to States 2, 3, and 5, respectively.

Table 2. Features used in the case Descriptions

Feature	Description
$\Delta Kills_{i-1}$	Number of opponent combat & worker units killed minus the same for oneself, in the preceding state
$\Delta Razings_{i-1}$	Number of opponent buildings destroyed minus same for onself, in the preceding state
$Buildings_{o}$	Number of opponent buildings ever created
$CombatUnits_{o}$	Number of opponent combat units ever created
$Workers_{o}$	Number of opponent worker units ever created
$Buildings_{p,i}$	Number of own buildings currently existing
$CombatUnits_{p,i}$	Number of own combat units currently existing
$Workers_{p,i}$	Number of own worker units currently existing

Ponsen and Spronck (2004) manually designed their tactics and then improved them by searching the space of strategies using a genetic algorithm, where each chromosome is a complete plan (*counter-strategy*). Tactics were manually extracted from chromosomes. They used dynamic scripting to learn weights on tactics, and reported good learning performances versus four opponent strategies. In contrast, Ponsen et al. (2005) automatically acquired tactics by extracting them from the chromosomes based on the building states (i.e., all actions in a building state comprise one tactic). We used this same automatic approach for CAT.

In this paper, we also rely on this state lattice and a set of state-specific tactics. However, we add a third knowledge source: cases that map game situations to tactics and their performance. We will use all three to test how well CAT can play in games against a single, randomly selected WARGUS opponent.

3 Case-Based Strategy Selection

Our case-based approach for selecting which subplan (tactic) to use in each state employs the state lattice and state-specific tactics libraries described in Section 2. By doing this, the decision space (i.e., the number of tactics per state) becomes small, and an attribute-value representation of game situations suffices to select tactics. We define a case *C* as a tuple of four objects:

$$C = \langle \text{BuildingState, Description, Tactic, Performance} \rangle$$

where BuildingState is an integer node index in the state lattice, Description is a set of features of the current situation (see Section 3.1), Tactic is a counter-strategy's sequence of actions for that BuildingState, and Performance is a value in [0,1] that reflects the utility of choosing that tactic for that BuildingState, where higher values indicate higher performance (see Section 3.3). We next use Aamodt and Plaza's (1994) task decomposition model to detail our approach.

3.1 Retrieval

CAT retrieves cases when a new state in the lattice is entered (i.e., at the game's start, and when a transition building is finished). At those times, it records values for the eight features shown in Table 2, which we selected because they were available and

are intuitively informative. They also balance information on recent game changes (i.e., the first two features), the opponent's situation (e.g., $Workers_o$), and the player's situation (e.g., $Workers_{p,i}$). When games begin, the value of the first two features is 0 (i.e., because no units have yet been killed and no buildings have yet been razed), while the others have small values (e.g,. only a few workers exist at the game's start, as exemplified in Figure 1). About 50 units are created, per side, in a short game, and a player's limit is 200. In addition to the ten in the state lattice, buildings include farms, towers, and a few others that do not cause state transitions.

Cases are grouped by BuildingState, and, after each game ends, at most one case is recorded per BuildingState. Our experiments involve repeated trials of only 100 games. Therefore, CAT does not require a fast indexing strategy for our evaluation.

CAT's function for computing the similarity between a stored case C and the current game description S is defined as:

$$\mathrm{Sim}(C, S) = (C_{Performance}/\mathrm{dist}(C_{Description}, S)) - \mathrm{dist}(C_{Description}, S)$$

where dist() is the (unweighted, unnormalized) Euclidean distance among the eight features. This simple function emphasizes distance, and prefers higher-performing cases (i.e., those whose Tactic has performed well when selected in previous games) among those whose distance to S is similar (i.e., if two cases are at approximately the same distance from S, then the higher performer among them will have greater similarity). This function is particularly useful for BuildingState 1 (i.e., the game's start), where case Descriptions are all identical, and thus equally distant to the game's initial state. We will consider more elaborate similarity functions in future work.

CAT uses a modified k-nearest neighbor function to select case Tactics for retrieval. Among the k most similar cases, it retrieves one with the highest Performance. However, to gain experience with all tactics in a state, case retrieval is not performed until each available tactic at that state is selected e times, where e is CAT's *exploration* parameter. During exploration, CAT randomly retrieves one of the least frequently used tactics for reuse. Exploration also takes place whenever the highest Performance among the k-nearest neighbors is below 0.5.

3.2 Reuse

CAT's reuse process is given the retrieved case Tactic. While adaptation takes place, it is not controlled by CAT, but is instead performed at the level of the action primitives in the context of the WARGUS game engine (e.g., if an action requests the creation of a building, the game engine decides its location and which workers will construct it, which can differ in each game situation).

3.3 Revision

Revision involves executing the reused tactic in WARGUS, and evaluating the results. No repairs are made to these tactics; they are treated as black boxes.

Evaluation yields the Performance of a case's Tactic, which is measured at both a local and global level. That is, CAT records the WARGUS game score for both the player and opponent at the start of each BuildingState and at the game's end, which

occurs when one player eliminates all of the other's units and buildings, or when we terminate a game if no winner has emerged after ten minutes of clock time.

We define the Performance for a Tactic t of case C with BuildingState b as a function of its "global" ($\Delta Score_i$) and "local" ($\Delta Score_{i,b}$) impact on the game score, where the former focuses on relative changes between the time that t begins executing in b and when the game ends, while the latter focuses only on changes during state b:

$$C_{Performance} = \sum_{i=1,n} C_{Performance,i} / n$$

$$C_{Performance,i} = ½(\Delta Score_i + \Delta Score_{i,b})$$

$$\Delta Score_i = (Score_{i,p} - Score_{i,p,b}) / ((Score_{i,p} - Score_{i,p,b}) + (Score_{i,o} - Score_{i,o,b}))$$

$$\Delta Score_{i,b} = (Score_{i,p,b+1} - Score_{i,p,b}) / ((Score_{i,p,b+1} - Score_{i,p,b}) + (Score_{i,o,b+1} - Score_{i,o,b}))$$

where n is the number of games in which C was selected, $Score_{i,p}$ is the player's WARGUS score at the end of the ith game in which C is used, $Score_{i,p,b}$ is player p's score before C's Tactic is executed in game i, and $Score_{i,p,b+1}$ is p's score after C's Tactic executes (and the next state begins). Similarly, $Score_{i,o}$ is the opponent's score at the end of the ith game in which C is used, etc. Thus, C's performance is updated after each game in which it is used, and equal weight is given to how well the player performs during its state and throughout the rest of the game.

3.4 Retention

During a game, CAT records a Description when it enters each BuildingState, along with the score and Tactic selected. It also records the scores of each side when the game ends, along with who won (neither player wins a tie). For each BuildingState traversed, CAT checks to see whether a case C exists with the same <Description, Tactic> pair. If so, it updates C's Performance. Otherwise, CAT creates a new case C for that BuildingState, Description, Tactic, and Performance as computed in Section 3.3 (this counts as C's first application). Thus, while duplicate cases are not created, CAT liberally creates new ones, and does not employ any case deletion policy.

4 Evaluation and Analysis

Our evaluation focuses on examining the hypothesis that CAT's method for selecting tactics significantly outperforms (1) a uniform selection strategy and (2) simply using the best counter-strategy. We report evidence that supports this hypothesis.

4.1 Competitors: WARGUS Players

Eight opponent scripts (see Table 3) were available for our experiments; some were publicly available and others we manually developed. For each opponent, we used Ponsen and Spronck's genetic algorithm to evolve a set of counter-strategy scripts. We use the best-performing counter-strategies among these (i.e., one per opponent) as a source of tactics, which are sequences of actions within a single building state of a counter-strategy. The names of the counter-strategies are shown in the lower left of Figure 2, which indicates that, for example, the evolved_sc1 counter-strategy includes tactics for building states 1, 3, 4, and 8, among others.

Table 3. WARGUS opponents used in the experiments

Opponent	Description
LBLA	This balances offensive actions, defensive actions, and research.
SR	Soldier's Rush: This attempts to overwhelm the opponent with cheap offensive units in an early state of the game.
KR	Knight's Rush: This attempts to quickly advance technologically, launching large offences as soon as strong units are available.
SC1-SC5	The top 5 scripts created by students, based on a class tournament.

The first WARGUS competitor, *Uniform*, selects tactics at each BuildingState according to a uniform distribution. Uniform should perform poorly because its selection is not guided by performance feedback. Uniform performs identically to CAT during its early stages of an experiment when, due to exploration, it randomly selects a tactic to use at each building state. To compute its results, we ran Uniform 48 times, six times per opponent script, and report its percentage of wins in Figure 4.

The next eight competitors are the counter-strategies. At each building state, they must use the tactic that defines them. Because they were evolved from different manually-generated scripts, we expect their performance to vary. Also, they should outperform Uniform, at least against the opponent on which they were trained. We ran each counter-strategy 10 times per opponent script, recorded the percentage of games that each won among their 80 games, and report the best performer in Figure 4.

The final competitor is CAT, which, after an initial exploration period during which it selects tactics using a uniform distribution, learns to intelligently select tactics at each building state. It should outperform Uniform because it learns to map game situations to a state's tactics, where selected tactics have performed well under similar game situations. Again, the counter-strategies supply the tactics for CAT to select. Thus, it might always select tactics from the same (e.g., best-performing) counter-strategy. Ideally, it should instead select tactics from *different* counter-strategies throughout a game, and across different games, to outperform the best performing counter-strategy. This is feasible: as shown in Figure 2, each counter-strategy traverses a unique sequence of building states. Thus, CAT may learn to select different tactics, for different game situations, from the same building state.

4.2 TIELT Integration

Integrating AI systems with gaming simulators can be a difficult and arduous task. Also, the resulting interface may not be reusable for similar integrations that a researcher may want to develop. Therefore, we used TIELT (Testbed for Integrating and Evaluating Learning Techniques) to perform this integration. TIELT (Aha & Molineaux, 2004) is a freely available middleware tool (http://nrlsat.ittid.com) that facilitates the integration of decision systems and simulators. It provides specific support for machine learning systems, and for complex gaming simulators. We actively support its users, and will use it in a few 2005 workshops and competitions.

TIELT integrations require constructing or reusing five knowledge bases, as shown in Figure 3. The Game Model is a (usually partial) declarative representation of the

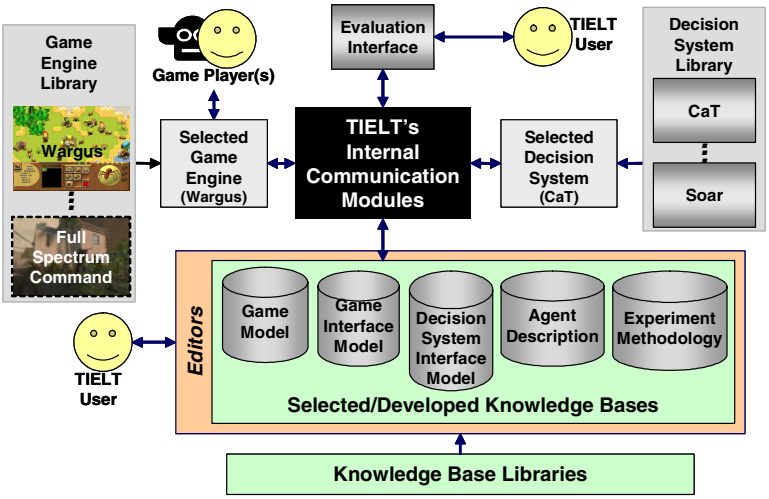

Fig. 3. TIELT's functional integration architecture

game. The Game Interface and Decision System Interface Models define the format and content of messages passed between TIELT, the selected game engine, and the decision system. The Agent Description includes an executable task structure that distinguishes the responsibilities of the decision system from those of game engine components. Finally, the Experiment Methodology encodes how the user wants to test the decision system on selected game engine tasks. Our knowledge bases will be available at TIELT'S www site for others to use (e.g., in future comparison studies).

We created a set of knowledge bases for integrating CAT with WARGUS using TIELT. This required several modifications of STRATAGUS so that it could provide access to subroutines that are germane to our investigation. Also, we increased the speed of WARGUS by a factor of 10 to run games more quickly. (A typical game still required an average of over 3-4 minutes to execute.)

This integration introduced a degree of non-determinism; while Ponsen and Spronck's (2004) earlier work involved updating WARGUS' code directly, our TIELT integration of CAT involves multiple threads and subsequent communication latencies. Thus, although two WARGUS games may have the same initial state and decision system, their results may differ greatly. This is the same that we might expect of a human playing the game, or any external process giving commands.

TIELT'S Game Model for this integration includes operators for key game tasks like building armies, researching technology advances, and attacking. Other operators obtain information about the game (e.g., the player's score). It also contains information about key events such as when a building or unit is completed. This information is maintained by the Game Interface Model, which contains information about the interface format. Future research using STRATAGUS can reuse these models.

The Agent Description links CAT to the abstractions of the Game Model. TIELT retrieves instructions from CAT and executes them using operators. When events recognized by the Game Model occur, it notifies CAT, using information from the

Decision System Interface Model to communicate. For example, when a building is finished, WARGUS sends a message to TIELT. The Game Interface Model interprets this, and fires a Game Model event. The Agent Description, listening for this event, notifies CAT and asks for further instructions. This Agent Description would need to be rewritten to work with another decision system, but the abstractions available from the Game Model simplify this task.

The Experiment Methodology defines the number of runs, when to stop the game, and resets CAT's memory when experiments begin. Also, it records game data into an EXCEL file for post-experiment analyses. It permits us to repeat experiments overnight, and record any data passing from STRATAGUS to CAT, or vice versa.

4.3 Empirical Methodology

We compared CAT versus its competitors for its ability to win WARGUS games. We used a fixed initial scenario, on a 128x128 tile map, involving a single opponent. The opponent was controlled by one of the eight scripts listed in Table 3. With the exception of the student scripts, these were all used in (Ponsen & Spronck, 2004).

Our only independent variable was the approach used to play against WARGUS opponents (i.e., Uniform, one of the eight counter-strategies, or CAT). We fixed CAT's two parameters during the experiment. The value of k (the number of nearest neighbors to consider when retrieving cases) was set to 3, as was the value of e (the exploration parameter, which determines the number of times that tactics in each state are used prior to permitting case reuse). We have not tuned either parameter setting.

For dependent variables, we collected average statistics on the percentage of games won, lost, and tied, along with the average final game scores for both players.

We ran Uniform on 48 games – six times for each of the eight opponents. We also tested each of the eight counter-strategies versus each opponent ten times. Averaging over multiple games helps to ameliorate the effects of non-deterministic game play.

However, the non-determinism introduced by TIELT's integration prevents testing the competitors on the same problems. Furthermore, it prevents us from periodically testing CAT on the same test set at different points during its training process. Therefore, we report CAT's average results across a sliding window, corresponding to the preceding n games, during training. We set n to 25 in the experiments (i.e., the measure after 100 games is the percentage of wins in games 76-100), and tested CAT in five trials of 100 games each. (Time limitations prevented us from running additional studies.) Cases are acquired throughout each trial, and the opponent for each game was randomly selected from a uniform distribution on the eight available.

4.4 Results

Figure 4 summarizes our results. As expected, Uniform performs poorly, winning an average of 22.9% of its games. CAT begins at this level, and after 100 games its average winning percentage is 82.4%, while saving an average of 302.4 cases. The best performer among the counter-strategies is evolved_SC5, which won 72.5% of its games. We compared the results (i.e., whether the player had won) for evolved_SC5 versus the final 25 games for CAT's five trials. A one-tail t-Test (2-sample assuming

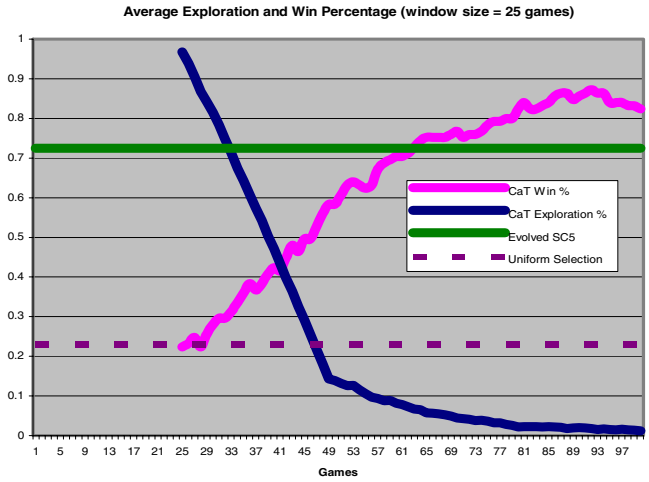

Fig. 4. Comparison of average winning percentages of CAT (across 5 runs) vs. the non-learning Uniform and best performing counter-strategy (Evolved SC5). Also shown is CAT's average exploration percentage. CAT's plots are across a window of the 25 preceding games

unequal variances) does not quite reveal a significant difference (p=0.053). However, there is a significant performance difference (p=0.00004) for the same test when examining the ratios of the final game scores (i.e., player_score/(player_score + opponent_score)), where CAT's average ratio was 0.65 while evolved_SC5's was 0.59. Also, CAT shows signs of overtraining, having peaked at 87.2%. Thus, CAT clearly outperforms the best individual counter-strategy, and correcting for overtraining should further increase its performance.

Also shown in Figure 4 is the average percentage of explorations performed by CAT. This starts at 100% through 24 games and quickly drops to 1%. With this setting for *e*, CAT relies almost exclusively on cases after 100 games.

5 Discussion

The results described in Section 4.4, although encouraging, are somewhat premature. We have not analyzed whether correlations among the eight features motivate using a more structured and informative case representation, such as the ones proposed in (Muñoz-Avila & Aha, 2004). Perhaps features should be differentially weighted, or normalized in distance computations. Also, CAT cheats; we will replace the three opponent-focused features (Table 2) with ones that a human player can observe (e.g., opponent score, observed opponent buildings) to determine whether CAT required them to obtain its good performance.

We have not yet tuned CAT's parameters, and it probably needs a management policy (e.g., for case filtering, deletion) to prevent overfitting, which may be the cause of CAT's final dip in performance in Figure 4. Moreover, CAT does not yet perform

plan adaptation, which, given an appropriate domain model, may substantially improve performance. Finally, opponent modeling or plan recognition techniques could also prove useful (e.g., they may increase CAT's learning rate).

Our empirical methodology could be improved by training on a subset of opponent scripts and testing on the remainder, thus better assessing for generalization and transfer. A larger set of opponent scripts (e.g., created using Ponsen and Spronck's (2004) genetic algorithm) would be handy for this type of evaluation.

Ponsen and Spronck's (2004) dynamic scripting algorithm also learns to select which tactic to use at each building state. Its results could be compared with CAT's, for a specific opponent, but this requires that they use the same set of tactics. In their experiments, dynamic scripting selected from an average of 40 tactics per building state level, while in our experiments CAT needed to select among eight per building state level (in the lattice). We will compare these approaches in our future work.

CAT builds on Ponsen and Spronck's (2004) work by relaxing the need to train separately for each opponent. However, we have not tested its ability to work with random initial states, or multiple simultaneous opponents. Also, CAT'S knowledge sources provide no opportunity for online learning, which requires a topology of abstract game states that can recur within a game play, and a means for recognizing them. Thus, we will consider such state topologies for WARGUS and other games.

Dynamic scripting learns weight settings for plans in a plan retrieval process. Like CAT, it exploits plan performance data. However, it does not identify game situations in which those plans should be selected. We expect that dynamic scripting could be enhanced by providing it with game situation information, which may help it to increase its learning rate, and allow it to train on multiple opponents simultaneously.

6 Conclusion

We introduced an approach for case acquisition and tactic (subplan) selection, and its implementation in CAT (Case-based Tactician). We described its application to winning games against WARGUS opponents. CAT is the first case-based reasoning system designed to win real-time strategy (RTS) games against randomly selected opponents. Using Ponsen and Spronck's (2004) state lattice, CAT selects a tactic for each state transition, where the tactics were automatically extracted from a set of genetically evolved counter-strategies. Our experiments showed that CAT learns to perform significantly better than the best performing counter-strategy against WARGUS opponents. In particular, after 100 games, it wins over 80% of its games.

This is the first significant application of TIELT, a tool that assists with integrating decision systems (e.g., CAT) with simulators (e.g., WARGUS). Our experience provided us with key lessons for its continued development (e.g., on how to integrate RTS games, and the evaluation methodologies it should support).

CAT's algorithm has not yet been tailored for this application; its performance can probably be further improved. Also, many interesting research issues require further attention, such as how to enhance dynamic scripting, CAT's applicability to online learning tasks, and learning to win in other gaming engines.

Acknowledgements

This research was supported by DARPA's Information Processing Technology Office and the Naval Research Laboratory.

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications,* **7**, 39-59.

Aha, D.W., & Molineaux, M. (2004). Integrating learning in interactive gaming simulators. In D. Fu & J. Orkin (Eds.) *Challenges in Game AI: Papers of the AAAI'04 Workshop* (Technical Report WS-04-04). San José, CA: AAAI Press.

Buro, M. (2003). Real-time strategy games: A new AI research challenge. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (pp. 1534-1535). Acapulco, Mexico: Morgan Kaufmann.

Cheng, D.C., & Thawonmas, R. (2004). Case-based plan recognition for real-time strategy games. *Proceedings of the Fifth Game-On International Conference* (pp. 36-40). Reading, UK: University of Wolverhampton Press.

De Jong, K., & Schultz, A.C. (1988). Using experience-based learning in game playing. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 284-290). Ann Arbor, MI: Morgan Kaufmann.

Díaz-Agudo, B., Gervás, P., & Peinado, F. (2004). A case based reasoning approach to story plot generation. *Proceedings of the Seventh European Conference on Case-Based Reasoning* (pp. 142-156). Madrid, Spain: Springer.

Fagan, M., & Cunningham, P. (2003). Case-based plan recognition in computer games. *Proceedings of the Fifth International Conference on Case-Based Reasoning* (pp. 161-170). Trondheim, Norway: Springer.

Fairclough, C.R., & Cunningham, P. (2004). AI structuralist storytelling in computer games. *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education.* Reading, UK: University of Wolverhampton Press.

Fairclough, C., Fagan, M., Mac Namee, B., Cunningham, P. (2001). Research directions for AI in computer games. *Proceedings of the Twelfth Irish Conference on Artificial Intelligence & Cognitive Science* (pp. 333-344). Maynooth, Ireland: Unknown publisher.

Fasciano, M.J. (1996). *Everyday-world plan use* (Technical Report TR-96-07). Chicago, Illinois: The University of Chicago, Computer Science Department.

Forbus, K., Mahoney, J., & Dill, K. (2001). How qualitative spatial reasoning can improve strategy game AIs. In J. Laird & M. van Lent (Eds.) *Artificial Intelligence and Interactive Entertainment: Papers from the AAAI Spring Symposium* (Technical Report SS-01-02). Stanford, CA: AAAI Press.

Gabel, T., & Veloso, M. (2001). *Selecting heterogeneous team players by case-based reasoning: A case study in robotic soccer simulation* (Technical Report CMU-CS-01-165). Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Goodman, M. (1994). Results on controlling action with projective visualization. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 1245-1250). Seattle, WA: AAAI Press.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. *Proceedings of the Eighteenth International Joint Conference on AI* (pp. 1003-1010). Acapulco, Mexico: Morgan Kaufmann.

Karol, A., Nebel, B., Stanton, C., & Williams, M.-A. (2003). Case based game play in the RoboCup four-legged league: Part I the theoretical model. In D. Polani, B. Browning, A. Bonarini, & K. Yoshida (Eds.) *RoboCup 2003: Robot Soccer World Cup VII*. Padua, Italy: Springer.

Kerner, Y. (1995). Learning strategies for explanation patterns: Basic game patterns with applications to chess. *Proceedings of the First International Conference on Case-Based Reasoning* (pp. 491-500). Sesimbra, Portugal: Springer.

Laird, J.E., & van Lent, M. (2001). Interactive computer games: Human-level AI's killer application. *AI Magazine,* **22**(2), 15-25.

Meehan, J. R. (1981). Tale-spin and micro tale-spin. In R.C. Schank & C.K. Riesbeck (Eds.) *Inside computer understanding*. Hillsdale, NJ: Erlbaum.

Muñoz-Avila, H., & Aha, D.W. (2004). On the role of explanation for hierarchical case-based planning in real-time strategy games. In P. Gervás & K.M. Gupta (Eds.) *Proceedings of the ECCBR 2004 Workshops* (Technical Report 142-04). Madrid, Spain: Universidad Complutense Madrid, Departamento di Sistemos Informáticos y Programación.

Ponsen, M.J.V., Muñoz-Avila, H., Spronck, P., & Aha, D.W. (2005). Automatically acquiring domain knowledge for adaptive game AI using evolutionary learning. To appear in *Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence*. Pittsburgh, PA: AAAI Press.

Ponsen, M., & Spronck, P. (2004). Improving adaptive game AI with evolutionary learning. *Computer Games: Artificial Intelligence, Design and Education* (pp. 389-396). Reading, UK: University of Wolverhampton.

Powell, J.H., Hauff, B.M., & Hastings, J.D. (2004). Utilizing case-based reasoning and automatic case elicitation to develop a self-taught knowledgeable agent. In D. Fu & J. Orkin (Eds.) *Challenges in Game Artificial Intelligence: Papers from the AAAI Workshop* (Technical Report WS-04-04). San Jose, CA: AAAI Press.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development,* **3**(3), 210-229.

Schaeffer, J. (2001). A gamut of games. *AI Magazine*, **22**(3), 29-46.

Ulam, P., Goel, A., & Jones, J. (2004). Reflection in action: Model-based self-adaptation in game playing agents. In D. Fu & J. Orkin (Eds.) *Challenges in Game Artificial Intelligence: Papers from the AAAI Workshop* (Technical Report WS-04-04). San Jose, CA: AAAI Press.

Wendler, J., Kaminka, G. A., & Veloso, M. (2001). Automatically improving team cooperation by applying coordination models. In B. Bell & E. Santos (Eds.) *Intent Inference for Collaborative Tasks: Papers from the AAAI Fall Symposium* (Technical Report FS-01-05). Falmouth, MA: AAAI Press.

Wendler, J., & Lenz, M. (1998). CBR for dynamic situation assessment in an agent-oriented setting. In D.W. Aha & J.J. Daniels (Eds.), *Case-Based Reasoning Integrations: Papers from the AAAI Workshop* (Technical Report WS-98-15). Madison, WI: AAAI Press.