Combining Case-Based Reasoning and Reinforcement Learning for Tactical Unit Selection in Real-Time Strategy Game AI

Stefan Wender(✉) and Ian Watson

The University of Auckland, Auckland, New Zealand
{s.wender,ian}@cs.auckland.ac.nz

Abstract. This paper presents a hierarchical approach to the problems inherent in parts of real-time strategy games. The overall game is decomposed into a hierarchy of sub-problems and an architecture is created that addresses a significant number of these through interconnected machine-learning (ML) techniques. Specifically, individual modules that use a combination of case-based reasoning (CBR) and reinforcement learning (RL) are organised into three distinct yet interconnected layers of reasoning. An agent is created for the RTS game StarCraft and individual modules are devised for the separate tasks that are described by the architecture. The modules are individually trained and subsequently integrated in a micromanagement agent that is evaluated in a range of test scenarios. The experimental evaluation shows that the agent is able to learn how to manage groups of units to successfully solve a number of different micromanagement scenarios.

Keywords: CBR · Reinforcement learning · Game AI · Layered learning

1 Introduction

An area that has always been at the forefront of interesting AI utilization is games. Games provide a fertile breeding ground for new approaches and an interesting and palpable test area for existing ones. And as games such as checkers and chess are devised as high-level abstractions of mechanisms and processes in the real world, creating AI that works in these games can eventually lead to AI that solves real-world problems.

One of the most popular genres of computer video games is real-time strategy (RTS). RTS is a genre of computer video games in which players perform simultaneous actions while competing against each other using combat units. Often, RTS games include elements of base building, resource gathering and technological developments and players have to carefully balance expenses and high-level strategies with lower-level tactical reasoning. RTS games incorporate many different elements and are related to areas such as robotics and military simulations. RTS games can be very complex and, especially given the real-time

A. Goel et al. (Eds.): ICCBR 2016, LNAI 9969, pp. 413–429, 2016.
DOI: 10.1007/978-3-319-47096-2_28

aspect, hard to master for human players. Since they bear such a close resemblance to many real-world problems, creating powerful AI in an RTS game can lead to significant benefits in addressing those related real-world tasks.

The creation of powerful AI agents that perform well in computer games is made considerably harder by the enormous complexity these games exhibit. The complexity of any board game or computer game is defined by the size of its state- and decision space. A state in chess is defined by the position of all pieces on the board while the possible actions at a certain point are all possible moves for these pieces. [14] estimated the number of possible states in chess as 10^{43}. The number of possible states in RTS games is vastly bigger. [2] estimated the decision-complexity of the *Wargus* RTS game (i.e. the number of possible actions in a given state) to be in the 1,000s even for simple scenarios that involve only a small number of units. StarCraft, a pioneering commercial RTS game from 1998, is even more complex than Wargus, with a larger number of different unit types and larger combat scenarios on bigger maps, leading to more possible actions. [20] estimated the number of possible states in StarCraft, defined through hundreds of possible units for each player on maps that can have maximum dimension of 256×256 tiles, to be in excess of 10^{11500}. In comparison, chess has a decision complexity of about 30.

The topic of this paper is the creation of an agent that focuses on the tactical and reactive tasks in RTS games, the so-called 'micromanagement'. Our agent architecture is split into several interconnected layers that represent different levels of the decision making process. The agent uses a set of individual CBR/RL modules on these different levels of reasoning in a fashion that is inspired by the layered learning model [16]. The combination of CBR and RL that is described in this paper is performed in order to enable the agent to address more complex problems by using CBR as an abstraction- and generalisation-technique.

2 Related Work

Creating the overall model as well as the individual sub-components of the architecture was influenced by previous research that evaluated the suitability of RL for the domain [21] and a combination of CBR and RL for small-to-medium-sized micromanagement problems [23].

Reinforcement Learning. The application of RL algorithms in computer game AI has seen a big increase in popularity within the past decade, as RL is very effective in computer games where perfect behavioural strategies are unknown to the agent, the environment is complex and knowledge about working solutions is usually hard to obtain. Recently, the *UCT* algorithm (*Upper Confidence Bounds applied to Trees*) [9], an algorithm based on Monte-Carlo Tree Search (MCTS), has lead to impressive results when applied to games. MCTS and UCT are closely related to RL which is partially based on Monte-Carlo methods.

[7] overcame this and described the use of heuristic search to simulate combat outcomes and control units accordingly. Because of the aforementioned lack in speed and precision of the StarCraft game environment, the authors first

created their own simulator, *SparCraft*, to evaluate their approach and later re-integrate the results into a game-playing agent. Apart from MCTS and UCT however, few of the new theoretical discoveries in RL have made it into game AI research. Most research in computer game AI, including this paper, works with the well-tested temporal difference (TD) RL algorithms such as Q-learning [19]. Q-learning integrates different branches of previous research such as dynamic programming and trial-and-error learning into RL. [3] extended an online Q-learning technique with CBR elements in order for the agent to adapt faster to a change in the strategy of its opponent. The resulting technique, *CBRetaliate*, tried to obtain a better matching case whenever the collected input reading showed that the opponent was outperforming it. As a result of the extension, the CBRetaliate agent was shown to significantly outperform the Q-learning agent when it came to sudden changes in the opponent's strategy.

Case-Based Reasoning and Hybrid Approaches. Using only RL for learning diverse actions in a complex environment quickly becomes infeasible and additional modifications such as ways of inserting domain knowledge or combining RL with other techniques to offset its shortcomings are necessary.

Combining CBR with RL has been identified as a rewarding hybrid approach [5] and has been done in different ways for various problems.

[8] extended the standard *GDA* algorithm presented in [12] into *Learning GDA*. LGDA was created by integrating CBR with RL, i.e. the agent tried to choose the best goal, based on the expected reward. While the integration of CBR and RL differs from the approach pursued in the CBR/RL modules in this paper, the online acquisition of knowledge using a CBR/RL approach is similar.

[11] described the integration of CBR and RL in a continuous environment to learn effective movement strategies for units in a RTS game. This approach was unique in that other approaches discretize these spaces to enable machine learning. As a trade-off for working with a non-discretized model, the authors only looked at the movement component of the game from a meta-level perspective where orders are given to groups of units instead of individuals and no orders concerning attacks are given.

An example of an approach which obtains knowledge directly from the environment is [4]. The authors used an iterative learning process that is similar to RL and employed that process and a set of pre-defined metrics to measure and grade the quality of newly-acquired knowledge while performing in the RTS game *DEFCON*. Similar to this approach, the aim in this paper and the CBR/RL modules created as part of it is to acquire knowledge directly through interaction with the game. The learning process is controlled by RL which works well in this type of unknown environment without previous examples of desired outcomes. CBR is then used for managing the acquired knowledge and generalising over the problem space.

Hierarchical Approaches and Layered Learning. Combining several ML techniques, such as CBR and RL, into hybrid approaches leads to more powerful techniques that can be used to address more complex problems. However, problems such as those simulated by commercial RTS games with many actors

in diverse environments still need significant abstraction in order for agents to solve the problems they are confronted with. A common representation of the problems that are part of RTS games is in a hierarchical architecture [13].

An early application of hierarchical reasoning in RTS games was described in [6], where planning tasks in RTS games are divided into a hierarchy of three different layers of abstraction. This is similar to the structure identified in the next section, with separate layers for unit micromanagement, tactical planning in combat situations and high-level strategic planning. The authors used *MCPlan*, a search/simulation based Monte Carlo planning algorithm, to address the problem of high-level strategic planning.

Layered learning (LL) was devised for computer robot soccer, an area of research that pursues similar goals as RTS games and can be regarded as a simplified version of these combat simulations [16]. The main differences between the two are the less complex domain and less diverse types of actors in computer soccer. Additionally, computer soccer agents often compute their actions autonomously while RTS game agents orchestrate actions between large numbers of objects [13]. Because of the many similarities, LL makes an excellent, though as of now mostly unexplored, paradigm for a machine learning approach to RTS game AI. [10] combine both original and a concurrent LL approach [24] to create *overlapping layered learning* for tasks in the simulated robotic soccer domain. The original paradigm froze components once they had acquired learning for their tasks. The concurrent paradigm purposely kept them open during learning subsequent layers, thus finding a middle ground between freezing each layer once learning is complete and always leaving previously learned layers open.

3 A Hybrid Hierarchical CBR/RL Architecture

The hierarchical architecture and its constituent separate modules that address the micromanagement problem in RTS games are based on previous approaches described in [21,23]. Subdividing the problem enables a more efficient solution than when addressing the problem on a single level of abstraction, something which would either result in case representations which are too complex to be used for learning in reasonable time, or that require such a high level of abstraction that it prevents any meaningful learning process.

The structure of the core problems inherent in RTS games such as StarCraft, shown in Fig. 1, leads to most RTS agents being hierarchical [13]. The architecture we devised covers the micromanagement component of the game, enclosed in the solid red square shown in Fig. 1. Reconnaissance is currently not part of the framework, as the CBR/RL agent only works with units which are already visible.

Based on this task decomposition, three distinct organisational layers are identified. The *Tactical Level* is the highest organisational level and represents the entire world the agent has to address, i.e. the entire battlefield and the entire solid red square in the figure. The *Squad Level* is indicated by the dotted green square. Sub-tasks represented here concern groups of units, potentially spread over the entire battlefield. Finally, the *Unit Level* is the bottommost layer. This

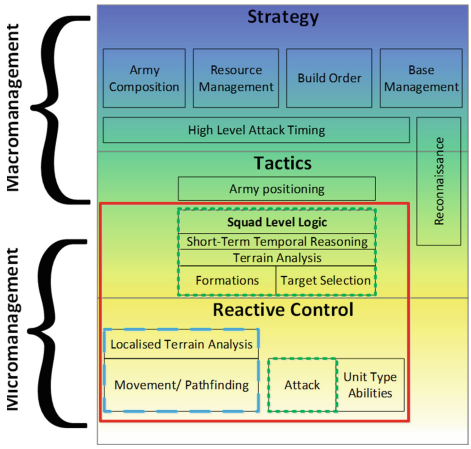

Fig. 1. RTS micromanagement tasks

layer covers pathfinding, works on a per-unit basis and is denoted by the dashed blue square in the diagram. Translating this layered problem representation into a CBR/RL architecture is done through a number of hierarchically interconnected case-bases. The approach to hierarchical CBR here is strongly inspired by that in [15], which describes a hierarchical CBR (HCBR) system for software design. One major difference between the approach described here and the one in [15] is that the use of RL for updating fitness values in the hierarchically interconnected case-bases means that each case-base has its own Adaptation-part of the CBR cycle [1]. Figure 2 shows the case-bases resulting from modeling the problem in this hierarchical fashion. Both the tactical level and the unit level are represented by a single case-base. The unit level is only responsible

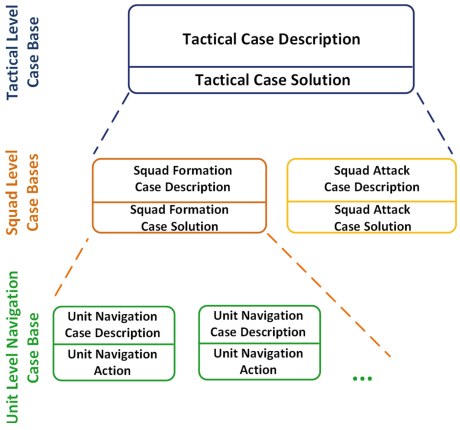

Fig. 2. Hierarchical structure of the case-bases

for *Navigation*. The intermediate squad level has one case-base for two possible actions on that level, *Attack* and *Formation*. Each case-base is part of a distinct CBR/RL module. Higher levels can then use the lower level components to interpret their solutions. As a result, higher levels base their learning process on the knowledge previously acquired on lower levels. RL relies in its learning process on the fact that similar actions lead to similar results. Otherwise the learning process continues until a stable policy is found with non-changing fitness values for state-action pairs. This would be difficult to achieve within a reasonable time if lower-level case-bases change fitness values at the same time as higher-level case-bases. Therefore, it was decided to evaluate and train lower level components first, retain the acquired knowledge for the respective tasks in the appropriate case-bases and subsequently evaluate the next-higher level using the lower-level cases as a foundation. In order to avoid diluting the learning- and evaluation process of higher levels, cases in lower-level case-bases are not changed once they are reused by a higher-level evaluation.

This evaluation and training procedure is not ideal since it partially negates the online learning characteristic of the CBR/RL agent. However, the alternative is a very noisy learning process that would seriously complicate the use of RL.

4 Lower-Level Modules

The individual modules that make up the overall architecture all follow a similar design and use a hybrid CBR/RL approach [23]. This section sums up the three lower-level modules (*Pathfinding*, *Attack* and *Formation*) and the MDP framework that is created for them [17]. All modules use a Q-learning algorithm to learn how to maximise the rewards for their respective tasks. Structure and implementation of the module for *Tactical Unit Selection* is described in detail in the next section. Underlying the decomposition into the modules described here is the analysis of tasks that are relevant to micromanagement in RTS games as displayed in Fig. 1.

4.1 Unit Pathfinding

Unit navigation and movement is a core component of any RTS game and also extends to other areas such as autonomous robotic navigation. This module is described in detail in [22] and is concerned with controlling a single agent unit (Table 1).

States

Table 1. Navigation case-base summary

Attribute	Description
Agent unit IM	Map with 7×7 fields containing the damage potential of adjacent allied units.
Enemy unit IM	Map with 7×7 fields containing the damage potential of adjacent enemy units.
Accessibility IM	Map with 7×7 fields containing true/false values about the accessibility.
Unit type	Type of a unit.
Last unit action	The last movement action taken.
Target position	Target position within the local 7×7 map

Actions. The case solutions are concrete game actions. There currently are four *Move* actions for the four different cardinal directions, i.e. one for every 90°.

Reward Signal. The compound reward $\mathcal{R}^a_{ss'}$ that is computed after finishing an action is based on damage taken during the action Δh_{unit}, the time the action took t_a and the change in distance to the chosen target location Δd_{target}.

$$\mathcal{R}^a_{ss'} = \Delta h_{unit} - t_a + \Delta d_{target}.$$

4.2 Squad-Level Coordination

Squad-level modules define and learn how to perform actions that coordinate groups of units while re-using the pathfinding component on the lowest level of the architecture.

Unit Formations. Tactical formations are an important component in RTS games, which often resemble a form of military simulator and are heavily inspired by real-life combat strategy and tactics. The *Formation* module creates formations that are a variant of dynamic formations [18] and learn through CBR/RL the best unit-slot associations, i.e. which slot in the formation a certain unit is assigned to (Table 2).

States

Table 2. Formation state case description

Category	Attribute	Type
Index	Unit # Agent	Integer
Unit	Type	Enum
	Health	Integer
	Position	Integer
Opponent	Attacking Damage towards the Formation Center from each of the 8 (Inter)Cardinal Directions	Integer

Actions. Actions are an assignment of the controlled units to certain slots in the formation. This means that the available actions are basically a permutation of all available units over all available formation slots.

Reward Signal. The two main criteria for an effective formation-forming action were decided to be the speed with which the action is executed t_{form} and, weighted slightly higher, the potential damage that units in the formation can deliver at any one point in time d_{avg}.

$$r_{form} = 1.5 * d_{avg} - t_{form}.$$

Unit Attack. The goal of using attacking units in the most efficient way is to focus on a specific opponent unit in order to eliminate it and, as a result, also eliminate the potential damage it can do to agent units. As part of this *Attack* component, it also was decided to simplify the module by giving all agent units assigned to a single *Attack* action the same target. More complex attacking behaviour can then be created by queuing several *Attack* action after another (Table 3).

States

Table 3. Attack state case description

Category	Attribute	Type
Index	Units Opponent	Integer
Target Unit	Type	Enum
	Health	Integer
	Average Distance to Attackers	Integer
Agent	Combined Attacking Unit Damage	Integer

Actions. The potential case solutions/actions for attack cases are the attack targets. This means that there is one solution for each attack target/enemy unit.

Reward Signal. The reward signal is composed of components for the time it takes to finish the attack action t_{att}, the damage done to the target *dam* as well as the damage removed if the target is eliminated dam_{elim}.

$$r_{att} = dam + dam_{elim} - t_{att}.$$

Unit Retreat. While also a selectable action like Attack and Formation, *Retreat* does not use CBR/RL and thus doesn't have its own module in Fig. 2. The *Retreat* action is designed to avoid potential sources of damage. The *Retreat* action takes into account a larger area of the immediate surroundings of a unit when compared to these other actions, a 15×15 plot, compared to 7×7 used for pathfinding. In a two-step process that also takes into account the influence of neighbouring plots, the action selects the area with the lowest amount of enemy influence/damage potential.

5 Tactical Unit Selection

The *Tactical Unit Selection* component is structured in a way similar to that of lower-level components, based on a hybrid CBR/RL integration. Given the decomposition of the problem as described in Fig. 1, the task of the *Tactical Unit Selection* component is to find an ideal distribution of units among the three different modules on the level below, i.e. *Formation*, *Attack* and *Retreat*.

One major simplification that was introduced in order to avoid increasing the number of possible solutions exponentially and making learning infeasible with the current model is that all units assigned to *Attack* or *Formation* actions will perform the same action. This means that any unit assigned to an attack will attack the same target. Any unit assigned to a formation, will be part of the same formation.

5.1 Tactical Decision Making Model

The model used for the *Tactical Unit Selection* module, similar to those for *Formation* and *Attack* components, describes the problem in terms of an MDP. As this problem integrates the three lower-level modules, the model also combines elements of these modules.

States. *Tactical Unit Selection* states (or cases) are basically a combination of *Attack* and *Formation* states. However, some of the attributes that those state models use are part of both *Attack* and *Formation*, while others contain the same information but in less detail.

Table 4. Tactical state case description

Category	Attribute	Type
Index	Units Agent	Integer
	Units Opponent	Integer
Unit Agent	Type	Enum
	Health	Integer
	Damage	Integer
	Quadrant	Integer
	Cooldown	Boolean
Unit Opponent	Type	Enum
	Health	Integer
	Damage	Integer
	Quadrant	Integer
	Average Distance	Integer

The resulting composition of the case description of a *Tactical Unit Selection* state can be seen in Table 4.

Opponent units have two attributes containing different information (direction versus distance, relative to agent units) that indicate their position: *Quadrant* and *AverageDistance*. Agent units also have the *Quadrant* attribute to indicate their position relative to each other. The Boolean *Cooldown* value indicates if a unit's weapon is currently in cooldown or if it can be used. *Type* only distinguishes among *Melee*, *Ranged* and *Air* instead of specific unit types.

Given this composition, the dimensionality of the case description is considerably higher than for previous modules. For example, in a scenario with $n_a = 4$ agent units and $n_o = 5$ opponent units, case descriptions have $2+4*5+5*5 = 47$ attributes.

Actions. *Tactical Unit Selection* case solutions are distributions of the available agent units among the three available actions, i.e. triples (n_a, n_f, n_r) that indicate how many units are assigned to each action type. The overall number of solutions for n units distributed among the three categories is thus $\binom{3+n-1}{n}$. Given five agent units, the possible distributions for (*Attack*,*Formation*,*Retreat*) can be (5,0,0), (4,1,0) ... (0,0,5). For $n = 5$ units the number of solutions is therefore $\binom{3+n-1}{n} = 21$. This definition leads to a requirement for limiting the number of controlled units, if the number of learning episodes is to remain reasonable. The maximum number of agent and opponent units used in the evaluation scenarios was set to *ten*. By allowing a maximum of ten agent units in a game state, a single case can have at most $\binom{12}{10} = 66$ possible solutions.

Reward. The reward signal contains a negative component t_{tac} for the time it takes for a *Tactical Unit Selection* action to complete, a negative

component dam_{opp} for the damage that agent units received while performing the last action and two positive components, dam_{ag} for the damage done by agent units as well as dam_{elim} for the summed-up damage potential of all opponent units eliminated during the last action. Additionally, a third negative component dam_{loss} is added: this represents the damage potential lost when an agent unit is eliminated.

$$r_{tac} = dam_{ag} + dam_{elim} - dam_{opp} - dam_{loss} - t_{tac}.$$

Overall, the agent should attempt to choose solutions which eliminate opposing units quickly, while sustaining no (or only very little) damage to its own units.

5.2 CBR/RL Algorithm

Figure 3 shows a graphical representation of the steps and components involved in assigning actions to the available units. The algorithm chooses, from top to bottom, a *Tactical Unit Selection* unit distribution and, based on this distribution, an attack target, a formation unit-to-slot assignment as well as retreat destinations. Using the unit destinations computed through the lower-level components, the *Navigation* component then manages the unit movement. There

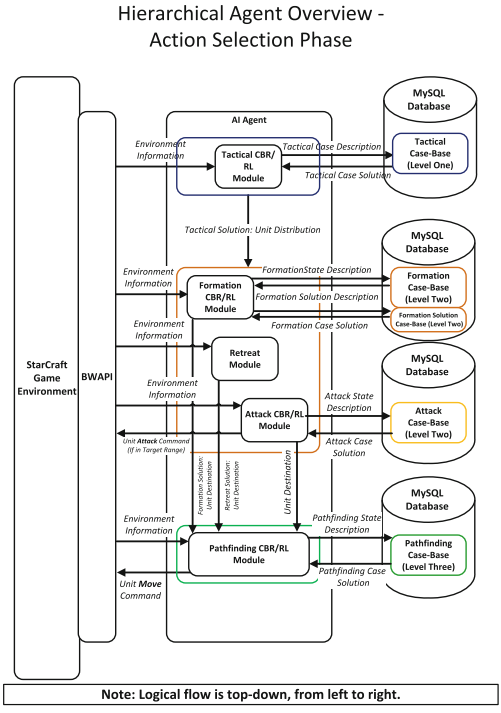

Fig. 3. Action selection using hierarchical CBR/RL for unit micromanagement

can be several *Navigation* actions until a unit reaches the destination assigned to it by one of the higher-level modules. There is always at most one action for *Attack*, *Formation* and *Retreat*, or zero, if no unit is assigned to a specific action category. The overall *Tactical Unit Selection* action is finished once all modules on lower levels indicate they are finished with their tasks.

6 Experimental Setup and Evaluation

Depending on the choice of parameters, large numbers of episodes can be required for finding optimal policies. Since this can easily become prohibitive if complex scenarios are used, a first step is an analysis of the case-base behaviour in a subset of the test scenarios, to find an appropriate threshold ψ that determines how similar a retrieved case in the CBR component has to be. Using a low ψ would mean that fewer cases are required to cover the entire case-space. However, this might lead to the retrieval of non-matching cases for a given situation and thus to sub-optimal performance due to a bad solution. Therefore, the selected ψ should lead to an optimal trade-off between performance and learning time. A number of representative micromanagement combat situations were created for the evaluation, each one with the aim to win the overall scenario against the built-in AI while retaining as much of the agent's own force as possible. Unit numbers and types vary between scenarios, as does the layout of the environment. Unit types are limited to standard non-flying units. The chosen algorithmic parameters for the CBR and RL components are listed in Table 5. The parameters are similar to those used successfully for evaluation and training of the *Navigation*, *Attack* and *Formation* modules. Starting positions are always a random spread opposite each other and the map-size is 2048×2048 pixels, the smallest possible StarCraft map size. Every experiment was run five times and the results were averaged.

Table 5. Tactical decision making evaluation parameters

Parameter	Values
Scenario	A(3vs5), B(6vs6), C(5vs5), D(4vs9), E(10vs10)
Number of games	100–100,000
Algorithm	One-Step Q-learning
Case-base similarity threshold ψ *A, B*	$30\,\% - 95\,\%$
Case-base similarity threshold ψ *C, D, E*	$80\,\%$
RL learning rate α	0.1
RL discount factor γ	0.8
RL exploration rate ϵ	0.8–0

6.1 Results

The first two scenarios were, as stated above, run with a number of different similarity thresholds ψ. Table 6 shows the results for *Scenario A*. Table 7 shows the results for *Scenario B*. The reward is normalized to a value between 0 % and 100 %. 0 % is achieved in a game in which agent units are eliminated without doing any damage. 100 % is a perfect game in which all opponents are eliminated without the agent units sustaining any damage. This allows to compare results of scenarios with different absolute values for maximum and minimum rewards.

As results in both the tables show, similarity thresholds between 80 % and 95 % lead to results that are roughly within a 10 % interval in terms of overall performance. However, the number of cases and, more importantly, the number of overall solutions increases significantly among the different thresholds. Therefore, it was decided to use a threshold of $\psi = 80\,\%$ for the subsequent evaluation scenarios. Given the results from the case-base analysis, the number of training episodes was set based on the number of agent units. The number of training

Table 6. Tactical decision making evaluation scenario A

ψ	# Episodes	# Cases	# Solutions	# Actions	Max. % Reward
95 %	100,000	2,376.4	18,853.0	47.32	92.48 %
90 %	60,000	1,265.2	9,976.4	45.27	87.33 %
85 %	20,000	366.8	2,570.2	41.15	82.93 %
80 %	8,000	192.0	1,299.6	41.06	81.82 %
70 %	1,500	52.6	293.4	35.43	70.16 %
60 %	800	39.2	224.6	30.96	60.73 %
50 %	500	29.2	156.2	23.7	52.79 %
40 %	300	17.6	95.8	11.49	33.35 %
30 %	100	13	69	10.36	25.47 %

Table 7. Tactical decision making evaluation scenario B

ψ	# Episodes	# Cases	# Solutions	# Actions	Max. % Reward
95 %	160,000	1570.4	31,201.6	7.10	78.15 %
90 %	75,000	699.8	12,339.0	6.92	75.13 %
85 %	30,000	324	5,324.20	6.96	73.01 %
80 %	15,000	259.8	3755.8	6.99	69.97 %
70 %	7,500	159.4	2092.6	7.45	63.09 %
60 %	5,000	95	1,309.2	8.72	57.99 %
50 %	3,000	63.2	801.4	9	55.19 %
40 %	2,000	47.2	631.4	9.5	45.53 %
30 %	1,500	38	517	10.18	43.44 %

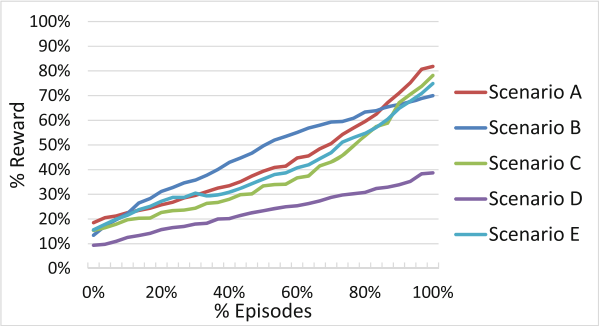

Fig. 4. Performance results for all scenarios

episodes is set to 15,000 for *Scenario C*, 10,000 for *Scenario D* and 50,000 for *Scenario E*. These comparably high amount of training episodes was chosen to ensure an optimal or near-optimal policy.

The results in Fig. 4 show that the hierarchical RL/CBR agent achieves a notable increase in average reward obtained for all five scenarios over the duration of their respective training runs. In terms of reward development, there is a difference between *Scenarios B* and *D* which use melee units only, and the other three scenarios. *Scenarios B* and *D* show an almost linear reward development over the time their respective experiments run. *Scenarios A*,*C* and *E*, which all use both melee and ranged units, show reward development curves that are more similar to those encountered in previous evaluations.

7 Discussion

Scenarios A and *B* have about ten *Tactical Unit Selection* actions (i.e. *Attack*, *Formation* or *Retreat*) in an average episode for the lowest, worst-performing setting of $\psi = 30\,\%$ where there is only a single case for each agent-opponent unit number combination. For higher thresholds, which allow for a more optimized performance, the number of actions diverges significantly. For *Scenario A*, the number of *Tactical Unit Selection* actions exceeds 40 for $\psi >= 80\,\%$. The reason for this is the learned hit-and-run strategy that performs best for the units in this particular scenario and which requires extensive use of *Retreat* actions. Lower similarity thresholds mean there is not enough distinction between inherently different cases, which in turn does not allow the agent to learn and effectively execute this hit-and-run strategy. The melee-unit-focused *Scenario B* teaches the agent a fundamentally different strategy, indicated by the average number of *Tactical Unit Selection* actions. For $\psi >= 70\,\%$, the average number of actions per game is below nine. This is due to the main strategy in this scenario, which is based on focusing attacks (covered by the *Attack* action) combined with minimal regrouping or retreating through *Formation* or *Retreat* actions. There is no use

for extensive *Retreat* patterns since opponent- and agent unit types are identical, which means hit-and-run style attacks are useless.

The fact that agent and opponent use identical melee units in *Scenario B* also explains the difference in overall maximum rewards achieved. While the hit-and-run strategy allows the agent to achieve perfect or near-perfect rewards of more than 90 % for *Scenario A*, the average reward in *Scenario B* reaches a maximum value of just below 80 %. This is because attacking melee units with other melee units will always lead to suffering a certain amount of damage. The low number of actions required for optimal performance in *Scenario B* also means that it is easier to achieve good results in terms of average reward by using random untried solutions.

In all scenarios, the AI agent manages to obtain a significant improvement in the average reward. For all army compositions in the different scenarios, the agent finds optimal or near-optimal policies. Due to the unit types involved, *Scenario A* is the only scenario where the army composition theoretically allows a 'perfect game', i.e. eliminating all enemy units without sustaining damage. The agent manages to obtain more than 80 % average reward in this scenario. In *Scenarios C* and *E*, which both contain melee units that are harder to manage and are basically guaranteed to sustain damage when they attack, the agent manages to obtain above 75 % of the maximum possible reward. Even in *Scenario D*, which only uses melee units, the agent reaches nearly 70 % of the possible reward, pointing to effective use of focus-fire and manoeuvring.

When comparing the reward development of the different scenarios as depicted in Fig. 4, there is a difference between *Scenarios B* and *D* which use only melee units and the other three scenarios. This directly reflects the ideal behaviours in those scenarios and how these behaviours are reflected in action-selection policies. Optimal behaviour in a given scenario depends both on the layout of the scenario and on the agent and opponent army compositions.

8 Conclusion and Future Work

Overall, the results show that the hierarchical CBR/RL agent successfully learns the micromanagement tasks it was built to solve. The agent learns near-optimal policies in all evaluated scenarios which cover a range of in-game situations. The agent successfully re-uses the lower-level modules created for the squad-level tasks and the knowledge stored while training these modules.

One major restricting condition which was introduced to avoid a combinatorial explosion of possible solutions is limiting *Attack* and *Formation* to a single action for all units assigned to the appropriate category on the highest level. The evaluation of the hierarchical architecture showed that for the tested scenarios, the implementation achieved good to very good results on all occasions. However, it could already be observed that the performance suffered slightly for bigger scenarios when compared to the excellent results in scenarios with fewer units. One way to overcome this limitation would be to introduce another level above the currently highest level. The additional level would then simply perform a pre-allocation of all available units among several lower-level modules.

An important aspect which could be part of future work is the comparison of the approach presented here to other bot architectures. While this comparison will require additional logic to also address the strategic layer such a test could provide valuable insights into the power of adaptive online ML in relation to other ML, static and search-based approaches.

Currently there is a separate training phase for each of the lower-level modules. Creating modules which can be trained concurrently would be one way to accelerate the learning process. Other possible ways of improving performance would be through speeding up the individual CBR/RL components by employing better algorithmic techniques such as improved case-retrieval.

In summary, the key contribution of this paper is an integrated hierarchical CBR/RL agent which learns how to solve both reactive and tactical RTS game tasks. The creation of the individual hybrid CBR/RL modules for tasks in RTS game micromanagement is based on thorough analyses of TD RL algorithms, CBR behaviour and the relevant problem domain tasks. The resulting agent architecture acquires the required knowledge through online learning in the game environment and is able to re-use the knowledge to successfully solve tactical RTS game scenarios.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun. **7**(1), 39–59 (1994)
2. Aha, D.W., Molineaux, M., Ponsen, M.: Learning to win: case-based plan selection in a real-time strategy game. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS (LNAI), vol. 3620, pp. 5–20. Springer, Heidelberg (2005). doi:10.1007/11536406_4
3. Auslander, B., Lee-Urban, S., Hogg, C., Muñoz-Avila, H.: Recognizing the enemy: combining reinforcement learning with strategy selection using case-based reasoning. In: Althoff, K.-D., Bergmann, R., Minor, M., Hanft, A. (eds.) ECCBR 2008. LNCS (LNAI), vol. 5239, pp. 59–73. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85502-6_4
4. Baumgarten, R., Colton, S., Morris, M.: Combining AI methods for learning bots in a real-time strategy game. Int. J. Comput. Games Technol. **2009**, 1–10 (2008)
5. Bridge, D.: The virtue of reward: performance, reinforcement and discovery in case-based reasoning. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS (LNAI), vol. 3620, pp. 1–1. Springer, Heidelberg (2005). doi:10.1007/11536406_1
6. Chung, M., Buro, M., Schaeffer, J.: Monte carlo planning in RTS games. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (2005)
7. Churchill, D., Saffidine, A., Buro, M.: Fast heuristic search for RTS game combat scenarios. In: Proceedings of the Eight Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2012) (2012)
8. Jaidee, U., Muñoz-Avila, H., Aha, D.: Integrated learning for goal-driven autonomy. In: Proceedings of the Twenty-Second International Conference on Artificial Intelligence (IJCAI 2011) (2011)
9. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Machine Learning ECML 2006, pp. 282–293 (2006)

10. MacAlpine, P., Depinet, M., Stone, P.: UT austin villa 2014: Robocup 3D simulation league champion via overlapping layered learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI) (2015)
11. Molineaux, M., Aha, D., Moore, P.: Learning continuous action models in a real-time strategy environment. In: Proceedings of the Twenty-First Annual Conference of the Florida Artificial Intelligence Research Society, pp. 257–262 (2008)
12. Muñoz-Avila, H., Aha, D., Jaidee, U., Klenk, M., Molineaux, M.: Applying goal driven autonomy to a team shooter game. In: Proceedings of the Florida Artificial Intelligence Research Society Conference, pp. 465–470 (2010)
13. Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game AI research and competition in starcraft. IEEE Trans. Comput. Intell. AI Games **3**(4), 293–311 (2013)
14. Shannon, C.E.: Programming a computer for playing chess. In: Levy, D. (ed.) Computer Chess Compendium, pp. 2–13. Springer, New York (1950)
15. Smyth, B., Cunningham, P.: Déjà vu: A hierarchical case-based reasoning system for software design. In: ECAI, vol. 92, pp. 587–589 (1992)
16. Stone, P.: Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. MIT Press, Cambridge (1998)
17. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
18. Van Der Heijden, M., Bakkes, S., Spronck, P.: Dynamic formations in real-time strategy games. In: 2008 IEEE Symposium on Computational Intelligence and Games, pp. 47–54. IEEE (2008)
19. Watkins, C.: Learning from Delayed Rewards. Ph.d. thesis, University of Cambridge, England (1989)
20. Weber, B.: Integrating Learning in a Multi-Scale Agent. Ph.d. thesis, University of California, Santa Cruz (2012)
21. Wender, S., Watson, I.: Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: broodwar. In: IEEE Symposium on Computational Intelligence and Games (CIG) (2012)
22. Wender, S., Watson, I.: Combining case-based reasoning and reinforcement learning for unit navigation in real-time strategy game AI. In: Lamontagne, L., Plaza, E. (eds.) ICCBR 2014. LNCS (LNAI), vol. 8765, pp. 511–525. Springer, Heidelberg (2014). doi:10.1007/978-3-319-11209-1_36
23. Wender, S., Watson, I.: Integrating case-based reasoning with reinforcement learning for real-time strategy game micromanagement. In: Pham, D.-N., Park, S.-B. (eds.) PRICAI 2014. LNCS (LNAI), vol. 8862, pp. 64–76. Springer, Heidelberg (2014). doi:10.1007/978-3-319-13560-1_6
24. Whiteson, S., Stone, P.: Concurrent layered learning. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 193–200. ACM (2003)