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

Stefan Wender and Ian Watson

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

Abstract. This paper presents a navigation component based on a hybrid case-based reasoning (CBR) and reinforcement learning (RL) approach for an AI agent in a real-time strategy (RTS) game. Spatial environment information is abstracted into a number of influence maps. These influence maps are then combined into cases that are managed by the CBR component. RL is used to update the case solutions which are composed of unit actions with associated fitness values. We present a detailed account of the architecture and underlying model. Our model accounts for all relevant environment influences with a focus on two main subgoals: damage avoidance and target approximation. For each of these subgoals, we create scenarios in the StarCraft RTS game and look at the performance of our approach given different similarity thresholds for the CBR part. The results show, that our navigation component manages to learn how to fulfill both sub-goals given the choice of a suitable similarity threshold. Finally, we combine both subgoals for the overall navigation component and show a comparison between the integrated approach, a random action selection, and a target-selection-only agent. The results show that the CBR/RL approach manages to successfully learn how to navigate towards goal positions while at the same time avoiding enemy attacks.

1 Introduction

RTS games are becoming more and more popular as challenging test beds that offer a complex environment for AI research. Commercial RTS game such as StarCraft in particular are a polished environment that offer complex test settings. These games have been identified as including numerous properties that are interesting for AI research, such as incomplete information, spatial and temporal reasoning as well as learning and opponent modeling [5]. For these reasons we chose StarCraft as a test bed for a machine learning (ML) approach that tries to learn how to manage combat units on a tactical level ("micromanagement").

Micromanagement requires a large number of actions over a short amount of time. It requires very exact and prompt reactions to changes in the game environment. The micromanagement problem involves things like damage avoidance,

target selection and, on a higher, more tactical level, squad-level actions and unit formations [18]. Micromanagement is used to maximize the utility of both individual units and unit groups, as well as to increase the lifespan of a player's units. A core component of micromanagement is unit navigation. There are numerous influences a unit has to take into account when navigating the game world, including static surroundings, opposing units and other dynamic influences. Navigation and pathfinding is not a problem unique to video games, but a topic that is of great importance in other areas of research such as autonomous robotic navigation [9]. The appeal of researching navigation in video games lies in having a complex environment with various different influences that can be precisely controlled. Navigation is also integrated as a supplementary task in the much broader task of performing well at defeating other players and can therefore be used and evaluated in numerous ways.

The navigation component presented in this paper is also not a standalone conception. We are currently developing a hierarchical architecture for the entire reactive and tactical layers in an RTS game. The AI agent based on this architecture uses CBR for its memory management, RL for fitness adaption and influence maps(IMs)/potential fields [8] for the abstraction of spatial information. This approach is based on previous research [19] where we used RL in an agent in a simplified combat scenario. While StarCraft offers a very large and complex set of problems, our previous RL agent focuses on a small subset of the overall problem space and is limited to one simple scenario.

Our high-level aim is to create an adaptive AI agent for RTS games that minimizes the amount of knowledge that is stored in deterministic sub-functions, but instead learns how to play the game. We aim to do this by using an approach that manages large amounts of obtained knowledge in the shape of cases in a casebase on the one hand, and has the means to acquire and update this knowledge through playing the game itself on the other hand.

2 Related Work

Unit navigation in an RTS game is closely related to autonomous robotic navigation, especially when looking at robotic navigation in a simulator without the added difficulties of managing external sensor input [10]. Case-based reasoning, sometimes combined with RL, has been used in various approaches for autonomous robotic navigation. [13] describes the *self-improving robotic navigation system* (SINS) which operates in continuous environment. One aim of SINS is, similar to our approach, to avoid hand-coded, high-level domain knowledge. Instead, the system learns how to navigate in a continuous environment by using reinforcement learning to update the cases.

Kruusmaa [9] develops a CBR-based system to choose the least risky routes in a grid-based map, similar to our infleucen map abstraction, and lead faster to a goal position. However, the approach only works properly when there are few, large obstacles and the state space is not too big.

Both of these approaches are non-adversarial however, and thus do not account for a central feature of RTS games that raise the complexity of the problem space considerably. We address the presence of opposing units by using influence maps to abstract the battlefield which can contain dozens of units on either side at any one time.

The A* algorithm [7] and its many variations are the most common exponents of search-based pathfinding algorithms that are used in video games while also being successfully applied in other areas. However, the performance of A* depends heavily on a suitable heuristic and is also very computation-intensive as they frequently require complex pre-computation to enable real-time performance. A* variations, such as the real-time heuristic search algorithm kNN-LRTA* [4], which uses CBR to store and retrieve subgoals to avoid re-visting known states too many times, have been used extensively in commercial game AI and game AI research.

Different forms of both RL and CBR are popular techniques in game AI research, either separately or as a combination of both approaches. The reinforcement learning component in this paper is based on our previous research in [19] and a subsequent integration of single-pass CBR. While our previous simple RL approach shows good results for the one specific scenario that is tested, the work presented in this paper enables the generalization to a much larger and more complex area of problems through the integration of CBR.

Using only RL for learning diverse actions in a complex environment becomes quickly unfeasible due to the curse of dimensionality [2]. Therefore, the addition of CBR to manage the obtained knowledge and to retain information in the form of cases offers benefits both in terms of the amount of knowledge that is manageable and in terms of knowledge generalization. [11] describes the integration of CBR and RL for navigation in a continuous environment. Both state- and action-space are continuous and the agent has to learn effective movement strategies for units in a RTS game. As a trade-off for working with a non-discretized model, the authors only look at movement component of the game from a meta-level perspective where actions are given to groups of units instead of individuals. As a result, this approach is situated on a tactical level rather than the reactive level in terms of the overall game (see Section 3).

Influence maps or potential fields for the abstraction of spatial information have been used in a number of domains, including game AI, mostly for navigation purposes. Initially IMs were developed for robotics [8].

[16] use influence maps to produce *kiting* in units, also in StarCraft. *Kiting* is a hit-and-run movement that is similar to movement patterns that our agent learns for micromanagement. Their approach is similar to [19] in the behavior they are trying create and similar to our current approach in that they also use influence maps. However, their focus lies solely on the hit-and-run action and not on general maneuverability and the potential to combine pathfinding and tactical reasoning into a higher-level component as in our current approach.

[6] describes the creation of a multi-agent bot that is able to play entire games of StarCraft using a number of different artificial potential fields. However, some

of the parts of the bot use non-potential field techniques such as rule-based systems and scripted behavior, further emphasizing the need to combine different ML techniques to address larger parts of the problem space in an RTS game.

3 Pathfinding and Navigation as Part of a Hierarchical Agent Architecture

RTS games can be split into logical tasks that fall into distinct categories such as strategy, economy, tactics or reactive maneuvers. These tasks can in turn be grouped into layers [18]. Figure 1 shows the subdivision of an RTS into tasks and layers that are involved in playing the game.

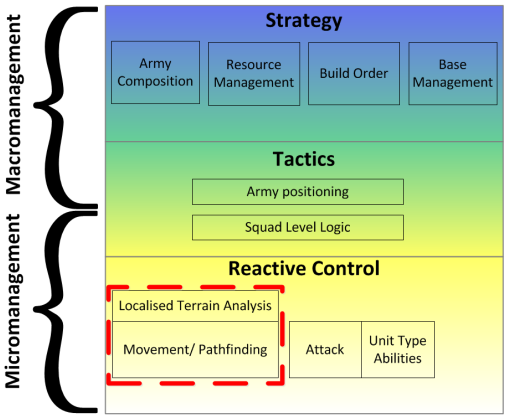

Fig. 1. RTS Game Layers and Tasks

For the creation of an AI agent that learns how to play a game, these tasks and layers can be used to create areas of responsibility for certain parts of an agent architecture. The layering leads to most RTS agents being inherently hierarchical and having a top-down approach when it comes to giving orders or executing actions and plans [12]. A high-level planer will decide on an action, which will subsequently be passed down to subordinate managers. These managers might then pass on parts of these action to managers that are even further down in the hierarchy.

Macromanagement is the umbrella term commonly used for tasks in the strategy layer and for high-level tactical decisions like army positioning in terms of the entire map. *Micromanagement* on the other hand describes the reactive layer as well as squad-level decisions in the tactical layer [12].

The ML agent we are currently working on is based on hierarchical CBR combined with a RL component. The agent learns how to play the entire micromanagement part of a game, from tactical decision making down to reactive control and pathfinding.

Hierarchical CBR is a sub-genre of CBR where several case-bases are organized in an interlinked order. [14] coins the term HCBR to describe a CBR system for software design. The authors use a blackboard based architecture that controls reasoning with a hierarchical case-base. The case-base is structured as a partonomic hierarchy where one case is split into several layers. Finding a solution to a software design problem requires several iterations where later iterations are used to fill lower levels of the case.

Our general architecture is similar, though slightly more complex. The case-bases are not strictly partonomic, since lower-level case-bases do not use solutions from higher level states as the only input for their case descriptions. Furthermore, our case-bases are more autonomous in that they run on different schedules which can result in differing numbers of CBR-cycle executions for different case-bases.

Each level in our architecture is working autonomously while using output from higher levels as part of its problem description. Each level is also represented by different case-bases. The pathfinding component presented here is provided with a target position that is decided on an intermediate level of the overall agent architecture. The goal given by this intermediate level is one of the influences the pathfinding component works with. At the same time, while a unit is trying to reach its goal location, the agent is trying to satisfy two other subordinate goals: minimizing any damage received and avoiding obstacles in the environment. The model that was created to reflect these requirements is described in detail in the next section.

4 CBR/RL Integration and Model

The core part of the pathfinding component is a case-base containing cases based on the environment surrounding a single unit. Navigation cases use both information from the game world and the target position given by a higher-level reasoner on the architecture layer above. Case descriptions are mostly abstracted excerpts of the overall state of the game environment. Case solutions are movement actions of a unit. Units acting on this level are completely autonomous, i.e. there is only one unit per case. As a result, the information taken from the game world is not concerned with the entire battlefield, but only with these immediate surroundings of the unit in question.

4.1 Case Description

The case description consists of an aggregate of the following information:

- Static information about the environment.
- Dynamic information about opposing units.

– Dynamic information about allied units.
– Dynamic information about the unit itself.
– Target specification from the level above.

Most of this information is encoded in the form of influence maps (IMs) [8]. We are using three distinct influence maps: One for the influence of allied units, one for the influence of enemy units and one for the influence of static map properties such as cliffs and other impassable terrain. This influence map for static obstacles also contains information on unit positions, as collision detection means that a unit can not walk through other units.

While higher levels of reasoning in our architecture use influence maps that can cover the entire map, the relevant information for a single unit navigating the game environment on a micromanagement level is contained in its immediate surroundings. Therefore, the perception of the unit is limited to a fixed 7x7 cutout of the overall IM directly surrounding the unit. An example of the different IMs for varying levels of granularity and other information contained in the environment can be seen in Figure 2.

The red numbers denote influence values. The influence values are based on the damage potential of units. This figure only shows enemy influence in order to not clutter the view. Any square in the two influence maps for enemy and allied units has assigned to it the damage potential for adjacent units. The IM containing information on passable terrain only contains a true/false value for each field. In figure 2, impassable fields are shaded.

The 'local' IM that represents the perception of a single unit it surrounds, is marked by the green lines that form a sub-selection of the overall yellow IM. Only values from this excerpt are used in the case descriptions.

In addition to the spatial information contained in the IMs, three other relevant values are part of a case description: *unit type*, *previous action* and *target location*. The *target location* indicates one of the 7x7 fields surrounding a unit and is the result of a decision on the higher levels. In Figure 2, the current target location is marked by the blue **X**. The *unit type* is indicative of a number of related values such as speed, maximum health and ability to fly. The *previous action* is relevant as units are able to carry momentum over from one action to the next.

The case solutions are concrete game actions. We currently use four *Move* actions for the different directions, one for every 90° (see Figure 3).

The case-base in the CBR part of the pathfinding component reflects a general CBR cycle [1], however does not use all of the cycle's phases. During the **Retrieval** step, the best-matching case is found using a simple kNN algorithm where $k = 1$, given the current state of the environment. If a similar state is found, its solution, made up of all possible movement actions and associated fitness values, is returned. These movement actions can then be **Reused** by executing them.

Currently, there is no **Revision** of case descriptions stored in any of the case-bases. Once a case has been stored, the case descriptions, consisting of the

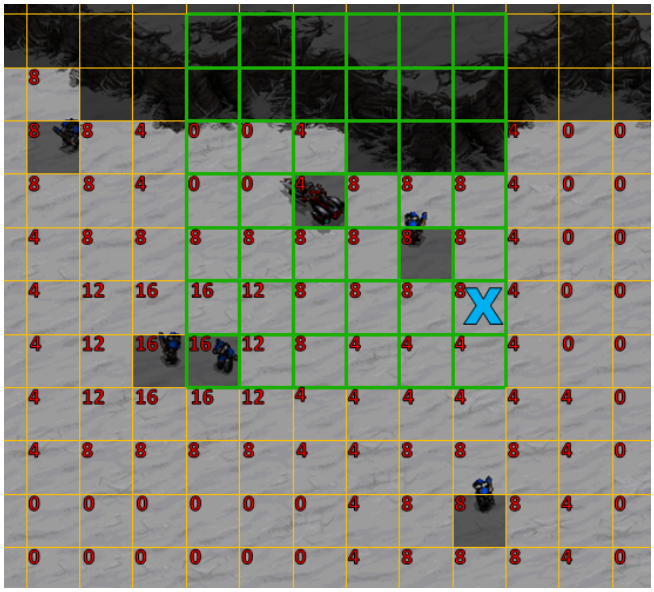

Fig. 2. Game Situation with Influence Map Overlay

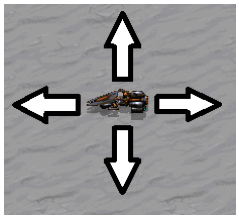

Fig. 3. Possible Movements for a Unit

abstracted game information and of the solutions from higher-up levels, are not changed anymore. However, the fitness values associated with case solutions are adjusted each time a case is selected. These updates are done using RL (see Section 4.2).

Retention is part of the retrieval process. Whenever a case retrieval does not return a sufficiently similar case, a new one is generated from the current game environment state.

The similarity measure that is used to retrieve the cases plays a central role, both in enabling the retrieval of a similar case, and in deciding when new cases have to be created if no sufficiently similar case exists in the case-base. Part of the novelty of our approach is the use of influence maps as central part of our case descriptions. These IMs are then used in the similarity metric when looking for matching cases.

In all three types of influence maps that are used, the similarity between a map in a problem case and a map stored in the case base is an aggregate of the similarities of the individual map fields. Comparing the map on accessibility of single map plots is comparably easy since each field is identified by a Boolean value, which means that similarity between plot of IMs in a problem case and in a case in the case base is either 0% or 100%. The similarity of a single field describing the damage potential of enemy or own units, is decided by the difference in damage to its counterpart in the stored case. I.e. if one field has the damage potential 20 and the counterpart in the stored case has the damage potential 30, the similarity is decided by the difference of 10. Currently, we are mapping all differences into one of four similarities between 0% and 100%. This also depends both on if unit influence exists in both cases and on how big the difference is. An example similarity computation is shown in Figure 4.

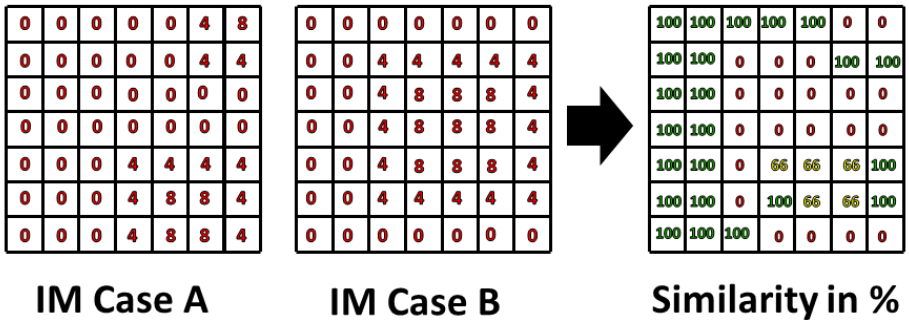

Fig. 4. Example of IM Field Similarity Computation

We decided not to abstract the information stored in the IM fields any further and do a direct field-to-field comparison. However, as a large case-base could slow the search down considerably, we are also investigating methods to reduce the current number of cases. This will start with methods to account for simple geometric transformations on the IMs such as rotation. In higher levels of our architecture (see Section 3), where larger areas of the map are compared and a brute-force nearest-neighbor search would be unfeasible, we are already using more abstract comparisons.Because of the requirement for very precise actions when navigating, using the same methods for the pathfinding component is impractical.

The similarity between *Last Unit Actions* is determined by how similar the movement directions for chosen are, i.e. same direction means a similarity of 1, orthogonal movement 0.5 and opposite direction means 0. *Unit Type* similarity is taken from a table that is based on expert knowledge as it takes a large number of absolute (health,energy, speed) and relative (can the unit fly, special abilities)

attributes into account. The similarity between *Target Positions* is computed as a the distance between target positions in cases in relation to the maximum possible distance (dimension of the local IM field).

Table 1 summarizes the case description and related similarity metrics.

Table 1. Case-Base Summary

Attribute	Description	Similarity Measure
Agent Unit IM	Map with 7x7 fields containing the damage potential of adjacent allied units.	Normalized aggregate of field similarity.
Enemy Unit IM	Map with 7x7 fields containing the damage potential of adjacent allied units.	Normalized aggregate of field similarity.
Accessibility IM	Map with 7x7 fields containing true/false values about the accessibility.	Normalized aggregate of field similarity.
Unit Type	Type of a unit.	Table of unit similarities between 0 and 1 based on expert knowledge.
Last Unit Action	The last movement action taken.	Value between 0 and 1 depending on the potential to keep momentum between actions.
Target Position	Target position within the local 7x7 map.	Normalized distance.

4.2 Reinforcement Learning for Solution Fitness Adaption

The reinforcement learning component is used to learn fitness values for case solutions. This part of the architecture is based on our previous research [19]. While the complexity of the model here is much higher, the problem domain remains the same, which is why we chose to use a one-step Q-learning algorithm for policy evaluation. One-step Q-learning showed, in this specific setting, the best performance when compared to other tested algorithms by a small margin. Furthermore, its implementation is comparably simple and evaluation using one-step algorithms are computationally inexpensive when compared to algorithms using eligibility traces.

Q-learning is an off-policy temporal-difference (TD) algorithm that does not assign a value to states, but to state-action pairs [17]. Since Q-learning works independent of the policy being followed, the learned action value Q function directly approximates the optimal action-value function Q^*.

In order to use RL, we express the task of updating the fitness of case solutions as a *Markov Decision Process* (*MDP*). A specific MDP is defined by a quadruple $(S, A, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$. In our case the state space S is defined through

the case descriptions. The action space A is the case solutions. $\mathcal{P}^a_{ss'}$ are the transition probabilities. While StarCraft has only few minor non-deterministic components, its overall complexity means that the transition rules between states are stochastic. As a result, different subsequent states s_{t+1} can be reached when taking the same action a_t in the same state s_t at different times in the game. $\mathcal{R}^a_{ss'}$ represents the expected reward, given a current state s, an action a and the next state s'.

Reward is computed on a per-unit and per-action basis, similar to our simple RL agent. The reward signal is crucial to enable the learning process of the agent and has to be carefully selected. As the agent tries to satisfy, depending on the given scenario, a number of different goals, we use a composite reward signal.For damage avoidance, the reward signal includes the difference in health h of the agent unit in question between time t and time $t+1$. The reward signal also includes a negative element that is based on the amount of time t_a it takes to finish an action a. This is to encourage fast movement towards the goal position. To encourage target approximation, the other central requirement besides damage avoidance, there is a feedback (positive or negative) for the change in distance d_target between the unit's current position and the target position. Finally, to account for inaccessible fields and obstacles, there is a penalty if a unit chooses an action that would access a non-accessible field. This penalty is not part of the regularly computed reward after finishing an action but is attributed whenever such an action is chosen and leads to immediate re-selection. The resulting compound reward that is computed after finishing an action is therefore

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

When looking at scenarios that evaluate the performance of the two subgoals target approximation and damage avoidance, only the parts of the reward signal that are relevant for the respective subgoal are used.

5 Empirical Evaluation and Results

A core parameter to determine in any CBR approach is, given a chosen case representation and similarity metric, a suitable similarity threshold that enables the best possible performance for the learning process. That means on the one hand creating enough cases to distinguish between inherently different situations, while on the other hand not inundating the case-base with unnecessary cases. Keeping the number of cases to a minimum is especially important when using RL. In order for RL to approximate an optimal policy p^*, there has to be a sufficient state- or state-action space coverage [15]. If the case-base contains too many cases, not only will retrieval speeds go down, but a lack of state-action space coverage will diminish the performance or even prevent meaningful learning altogether.

The model described in Section 4 is an aggregate of influences for several subgoals. For this reason, we decided to have two steps in the empirical evaluation. First, the two main parts of our approach, target approximation and damage

avoidance, are evaluated separately. This yields suitable similarity thresholds for both parts that can then be integrated for an evaluation of the overall navigation component.

For each of the steps we created suitable scenarios in the StarCraft RTS game. The first capability we evaluate is damage avoidance. The test scenario contains a large number of enemy units that are arbitrarily spread across a map. The scenario contains both mobile and stationary opponents, essentially creating a dynamic maze that the agent has to navigate through. The agent's aim is to stay alive as long as possible.

In order to test the agent's ability to find the target position, it controls a unit in a limited-time scenario with no enemies and randomly generated target positions. Therefore, the performance metric is the number of target positions an agent can reach within the given time. Accessibility penalties, *Unit Type* and *Previous Action* computation are part of both scenarios.

Based on initial test runs and experience from previous experiments, we used a learning rate $\alpha = 0.05$ and a discount rate $\alpha = 0.3$ for our Q-learning algorithm. The agent uses a declining ϵ-greedy exploration policy that starts with $\epsilon = 0.9$. This means, that the agent initially selects actions 90% random and 10% greedy, i.e. choosing the best known action. The exploration rate declines linearly towards 0 over the course of the 1000 games for the damage-avoidance scenario and 100 games for the target-approximation respectively. After each full run of 1000 or 100 games, there is another 30 games where the agent uses only greedy selection. The performance in this final phase therefore showcases the best policy the agent learned. The difference in the length of experiments (1000 vs 100 games) is due to the vastly different state-space sizes between the two scenarios. When looking at target approximation, the target can be any one of the fields in a unit's local IM, i.e. one of 49 fields. As a result, there can be a maximum of 49 different states even with 100% similarity threshold. On the other hand, there are $n^4 9$ possible states based on the damage potential of units, where n is the number of different damage values. This also explains why the generalization that CBR provides during case retrieval is essential to enable any learning effect at all. Without CBR, the state-action space coverage would remain negligibly small even for low n-values.

The results for both variants are shown in Figure 5 and Figure 6.

The results from the two sub-problem evaluation scenarios were used as a guideline to decide on a 80% similarity threshold to evaluate the overall algorithm (see Section 6 for more details). Both subgoals were equally weighted. The scenario for the evaluation of the integrated approach is similar to the one used for the damage avoidance sub-goal. The target metric is no longer the survival time however, but the number of randomly generated positions the agent can reach before it is destroyed. The run length for the experiment was increased to 1600 games to account for the heightened complexity. The resulting performance in the overall scenario can be seen in Figure 7. The figure also shows a random action selection baseline, as well as the outcome for using only target-approximation without damage avoidance.

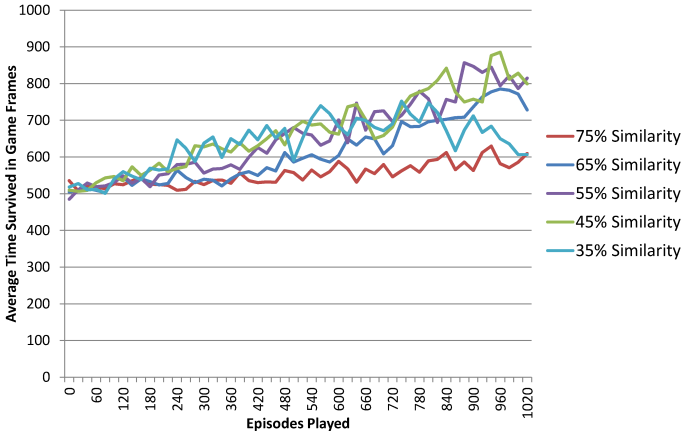

Fig. 5. Results for the Damage Avoidance Scenario

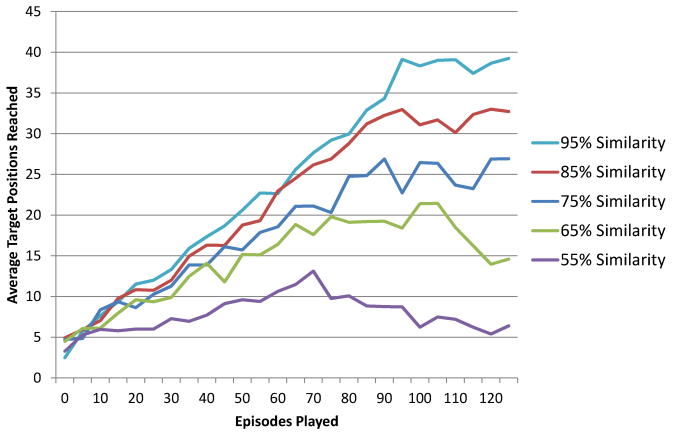

Fig. 6. Results for the Target Approximation Scenario

6 Discussion

The evaluation for the damage avoidance subgoal indicates, that there is only a narrow window in which learning an effective solution is actually achieved in the given 1000 episodes: With a similarity threshold of 35% there is not enough differentiation between inherently different cases/situations and the performance

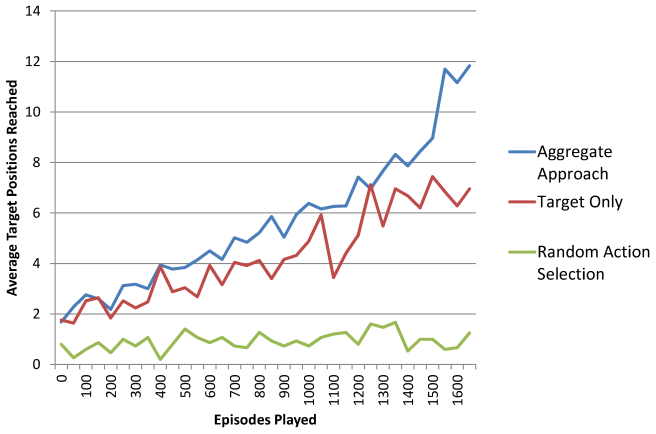

Fig. 7. Results for the Combined Scenario

drops of steeply towards the end. On the other end of the spectrum, the worst results with practically no improvement in the overall score is achieved for a 75%similarity threshold. This means that with such a high threshold, the state-action space coverage drops to a level where learning a 'good' strategy is no longer guaranteed: For a similarity threshold of 75% there are 7000 cases in the case-base after 1000 episodes played and less than 1/2 of all state-action pairs have been explored at least once.

Target approximation is slightly different in that an effective solution is achieved for any similarity threshold above 65%. This is due to the fact that even at 95% similarity there are still only $49 * 4 = 196$ state-action pairs to explore. The number of cases in the case-base varies however. At 95% similarity threshold, there is 49 different cases, i.e. the maximum possible number. At 85% similarity there exist about 20 different cases and at 75% only around 9 cases. This is important when we combine target approximation with damage avoidance for the overall approach since this also combines the state-space of the two subgoals. Therefore, twice as many cases for only a $\tilde{2}0\%$ higher reward as for the 75% compared to the 85% threshold or the 85% compared to the 95% threshold is a bad tradeoff. For this reason, we chose to use a 75% similarity threshold for the target approximation sub-goal.

Both subgoal approaches show a performance improvement in their chosen metrics over the run of their experiments. Since the target approximation algorithm manages to exhaustively explore the state-action space, the performance improvement is more visible in this scenario. This also means, that the best possible solution in this scenario has been found, as a 95% similarity threshold

with all possible states and a fully explored state-action space guarantee that the optimal policy p^* must have been found. For the damage avoidance scenario on the other hand, there is still a potential for improvement, since significant parts of the state-action space remain unexplored: Even for a 55% threshold only about 3/4 of all possible actions are executed at least once.

The results for the overall algorithm in Figure 7 show, that the agent manages to successfully learn how to navigate towards target positions. The performance of the agent far outperforms the random action selection. However, the results also show that the combined algorithm initially only has a small advantage over the target-approximation-only agent and only towards the end performs better. This is far less of an improvement than expected, especially given the fact that these experiments reached a sufficiently high state-action coverage of about 2/3. This indicates, that the chosen overall similarity threshold of 80% is too low and there is not enough differentiation between inherently different cases/situations to find the optimal policy.

7 Conclusion and Future Work

This paper presented an approach to unit navigation based on a combination of CBR, RL and influence maps for an RTS game AI agent. We illustrated the general tasks involved in a RTS game, how these tasks can be grouped into logical layers and how navigation is part of the micromanagement problem that we aim to address with an agent based on a hierarchical CBR architecture. The important information was identified and we designed a model that encompasses these different influences that are important for navigation and pathfinding in RTS games.

The empirical evaluation showed how our approach learned how to successfully learn to achieve partial goals of the navigation problem. Using the findings from the partial evaluation we ran an evaluation for the entire approach including all relevant influences and goals. This evaluation shows that our model and approach manage to successfully learns how to navigate a complex game scenario that tests all different sub-goals. Our approach significantly outperforms a random action selection and marginally outperforms the partial, target-approximation-only agent.

In terms of the ML algorithm, a more sophisticated case retrieval method and an evaluation of alternative RL methods are first on our list for future work. We are considering k-d trees to pre-index the cases [20] as a larger number of enemy or own units automatically leads to a more complex state-space, something that could slow down the retrieval speeds considerably.

We are also looking at using a different RL algorithm for our exploration strategy such as Softmax action selection [15] or the R-Max algorithm [3].

The next step in terms of the overall approach will be the evaluation of the integration of the navigation component presented in this paper into the overall HCBR agent architecture.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI communications 7(1), 39–59 (1994)
2. Bellman, R., Bellman, R.E., Bellman, R.E., Bellman, R.E.: Adaptive control processes: a guided tour, vol. 4. Princeton university press Princeton (1961)
3. Brafman, R.I., Tennenholtz, M.: R-max-a general polynomial time algorithm for near-optimal reinforcement learning. The Journal of Machine Learning Research 3, 213–231 (2003)
4. Bulitko, V., Bjornsson, Y., Lawrence, R.: Case-based subgoaling in real-time heuristic search for video game pathfinding. Journal of Artificial Intelligence Research 39, 269–300 (2010)
5. Buro, M., Furtak, T.: Rts games and real-time ai research. In: Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS). pp. 63–70. Citeseer (2004)
6. Hagelbäck, J.: Multi-Agent Potential Field Based Architectures for Real-Time Strategy Game Bots. Ph.D. thesis, Blekinge Institute of Technology (2012)
7. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on 4(2), 100–107 (1968)
8. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. The international journal of robotics research 5(1), 90–98 (1986)
9. Kruusmaa, M.: Global navigation in dynamic environments using case-based reasoning. Autonomous Robots 14(1), 71–91 (2003)
10. Laue, T., Spiess, K., Röfer, T.: Simrobot–a general physical robot simulator and its application in robocup. In: RoboCup 2005: Robot Soccer World Cup IX, pp. 173–183. Springer (2006)
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. Ontanón, 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 Transactions on Computational Intelligence and AI in Games (2013)
13. Ram, A., Santamaria, J.C.: Continuous case-based reasoning. Artificial Intelligence 90(1), 25–77 (1997)
14. Smyth, B., Cunningham, P.: Déjà vu: A hierarchical case-based reasoning system for software design. In: ECAI. vol. 92, pp. 587–589 (1992)
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
16. Uriarte, A., Ontañón, S.: Kiting in rts games using influence maps. In: Workshop Proceedings of the Eighth Artificial Intelligence and Interactive Digital Entertainment Conference (2012)
17. Watkins, C.: Learning from Delayed Rewards. Ph.D. thesis, University of Cambridge, England (1989)
18. Weber, B., Mateas, M., Jhala, A.: Building human-level ai for real-time strategy games. In: 2011 AAAI Fall Symposium Series (2011)
19. Wender, S., Watson, I.: Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar. In: Computational Intelligence and Games (CIG), 2012 IEEE Conference on (2012)
20. Wess, S., Althoff, K., Derwand, G.: Using k-d trees to improve the retrieval step in case-based reasoning. Topics in Case-Based Reasoning pp. 167–181 (1994)