Case-based reasoning for improved micromanagement in Real-time strategy games.

Tomasz Szczepański and Agnar Aamodt

Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
NO-7491, Trondheim, Norway
szczepan@stud.ntnu.no, agnar@idi.ntnu.no

Abstract. Real-time strategy (RTS) gameplay can be divided into macromanagement and micromanagement. Other researches have employed Case-based reasoning (CBR) and case-based planning in real-time strategy games that have beaten static scripted computer opponents. Unlike much of the previous work where CBR and case-based planning is used to improve the macromanagement in RTS games, we present a CBR system that can be used to improve the micromanagement quality in RTS games. We explore various ways of case representations as well as case adaptation mechanisms suited for a micromanagement environment. By managing to beat a hard-coded computer opponent we conclude that our approach can be used to aid human players against computer opponents and increase the quality of the micromanagement of a computer player. Our experiments have been conducted within the Warcraft 3 gaming environment.

Keywords: case-based reasoning, real-time strategy games, micromanagement

1 Introduction

The computer player performance in the popular real-time strategy (RTS) game genre has always been poor. Although AI techniques have successfully been applied in other related game genres like classic board games, the computer players in RTS games (often commonly referred to as game AI) are still lagging behind and can easily be defeated by amateurs [4]. Unlike games like chess where each player waits until the opponent makes his move, game flow in RTS games is simultaneous and continuous. Players usually compete within areas of resource gathering, structure building and army management, watching hundreds or even thousands of interacting objects from a top-down perspective. Due to the genre's nature, the game AI has to make decisions in real-time in an inaccessible, non-deterministic, dynamic and continuous environment with vast search spaces. Here traditional search methods no longer apply [4, 6, 8].

RTS gameplay can further be divided into macromanagement and micromanagement. Micromanagement is the way a player manages his units during combat or resource gathering. In the RTS game context a unit is a single character, often a worker or a solider, that can have several associated attributes like attack type, armor type or movement speed. In contrast to macromanagement, which focuses on overall game strategies involving a player's army (where to move, when to attack or whether to flee), micromanagement describes all the small details that involve the

individual units themselves (movement, enemy unit targeting, spell casting etc.). The macromanagement has been addressed by many studies of CBR in relation to RTS games [5, 6, 7]. However, very little has been done in relation to micromanagement which is usually just handled by the game itself.

In the study presented here a case-based reasoning system has been implemented in the game Warcraft 3. The system focuses on the micromanagement of units during battles. Specifically, we are interested in how case-based reasoning can improve the quality of micromanagement in a real-time strategy game. We study how to beat the already implemented game AI in Warcraft 3 as well as how a CBR system can be used to aid human players. To avoid confusion, the already implemented game AI in Warcraft 3 will be referred to as "computer opponent" while our implemented CBR system will be referred to as "CBR player".

The rest of this paper is structured as follows: Chapter 2 gives a brief introduction to micromanagement in Warcraft 3. In Chapter 3, 4, 5 and 6 we present our implemented CBR system, with the results presented in Chapter 7. Chapter 8 continues with a discussion of the results. Finally Chapter 9 summarizes and provides conclusions about the project.

2 Background

Warcraft 3: Reign of Chaos is an RTS computer game released by Blizzard Entertainment in July 2002. Despite its age, Warcraft 3 is still a popular game. This makes it easy to find suitable human testers for systems implemented in its game environment.

During typical "melee" gameplay each player starts with a main building and five workers. By gathering resources new buildings can be constructed and the player gains access to new units, technologies and structures (Figure 1). By specific build strategies and unit control in battle each player tries to get the upper hand to win the game by eventually destroying all opponents [2].

Before choosing Warcraft 3 as our game environment we also considered using the Wargus/Stratagus environment [10], an open source clone of the game Warcraft 2 (older version of Warcraft 3), and ORTS [11], an open source programming environment for RTS games. Wargus has been successfully used in other studies that show how a CBR approach can beat scripted computer opponents [6]. However, those environments are focusing mostly on other aspects of RTS games than micromanagement in battle [9]. Wargus is mostly designed for macromanagement while ORTS is focusing more on low level tasks like pathfinding, formations and resource gathering.

Environments like Wargus are not suitable micromanagement environments for human players. Compared to Warcraft 3, units in Wargus die quickly and the interface does not give a good overview of their status. Under such conditions, a human player has problems to react in time and can not micromanage efficiently. The other problem is the lack of unit healing in Wargus. The number of units a player can build is limited in RTS games. This makes wounded units a problem in Wargus because they take up spots for healthy units that can be trained. A lot of micromanagement is about keeping units alive. If you have a player with very bad

Fig. 1. A screenshot from Warcraft 3.

micromanagement battling a player with a very good micromanagement, the player with good micromanagement might win the whole battle without loosing a single unit. In Warcraft 3, keeping units alive is important because players can heal wounded units at much cheaper cost than by replacing dead units with newly trained ones [3]. This makes good micromanagement in Warcraft 3 very beneficial. Warcraft 3 suffers however from limitations caused by not being open source. Everything must be created by tools offered by Warcraft 3 meaning that the whole CBR system must be created from scratch. Luckily, Warcraft 3 has an internal scripting language [13] that makes this possible.

Since open source, both Wargus and ORTS can be converted to a suitable micromanagement environment for our needs. The problem is the amount of work needed to do so. Because our study was limited in time, we could not afford such an approach. The other reason we chose Warcraft 3, despite its limitations, is that it is very well suited as a micromanagement environment. Micromanagement in Warcraft 3 is of key importance and thus a very important aspect of the game. Moreover, the already implemented micromanagement in Warcraft 3 has room for improvement. Winning against the computer opponent in Warcraft 3 is fairly easy especially if its micromanagement behaviors are exploited. One example of this is that the computer opponent focuses on certain units with high priority. By knowing this a human player can keep an eye on those units ordering them to move away if necessarily. During battle the computer opponent will use a lot of time and effort by running after those units trying to kill them. The human player's force can concentrate on the computer's units and take them down one at a time. Such exploits does not contribute to the fun of playing a seemingly dumb opponent. To create a realistic and smart computer opponent in games like Warcraft 3, improving both high level strategies and micromanagement is necessary.

3 System Architecture

When designing and implementing our system we had to keep in mind that it would run on an end-user's computer inside the Warcraft 3 game (by using the tools offered by Warcraft 3 we created a map containing the CBR system as well as the cases). Therefore we wanted our CBR system to be simple with relatively few and simple cases.

Since our focus is on micromanagement, our system does not run on an ordinary melee map but on a custom map made as an arena. The arena is surrounded by limiting barriers (limits both air and ground passage). The system uses the three first steps in the 4R CBR model [1] and can be summarized by the following algorithm:

1. Store the current game state as a new unsolved case in memory.
2. Compare the new case to the existing cases in memory and retrieve the most similar one.
3. Adapt and execute the retrieved case.
4. If in training mode and a problem is observed, add a new case to the system.
5. Wait 1 second then go to 1.

Figure 2 shows a more detailed diagram of the CBR system. Every second, the current game state is abstracted into a case and compared to previously stored and solved cases. The most similar case is retrieved and the solution provided by that case is executed. At any time during execution new cases can be added. This is typically done when the system is executing a wrong case or a new case is needed (this is done by an expert observing the system). Adding cases is simply done by entering the "correct" strategy that is supposed to be followed. Since the new case will be the most similar case to the current game state, it will be executed next time (in one second). For a screenshot of the CBR system see Figure 3.

4 Case Structure and Indexing

The game state relevant to our system consists of units battling each other in a battlefield area. We are not interested in a player's resources, structures or units outside the battlefield. Our choice of relevant attributes that were added to the case descriptions describing the game state were "unit type", unit remaining "hit points" (determines the amount of damage a unit can withstand), unit remaining "mana points" (mana in Warcraft 3 is "magical energy" that some units can use to cast spells) and "unit position" relative to the battlefield (x and y coordinate). The battlefield in our system is a square shaped region where the battle takes place. Everything outside that region is not relevant and thus not included in the case description of the game state.

Our case structure is a simplified version of the proposed structure by Cheng et al. [5] and consists of three parts:

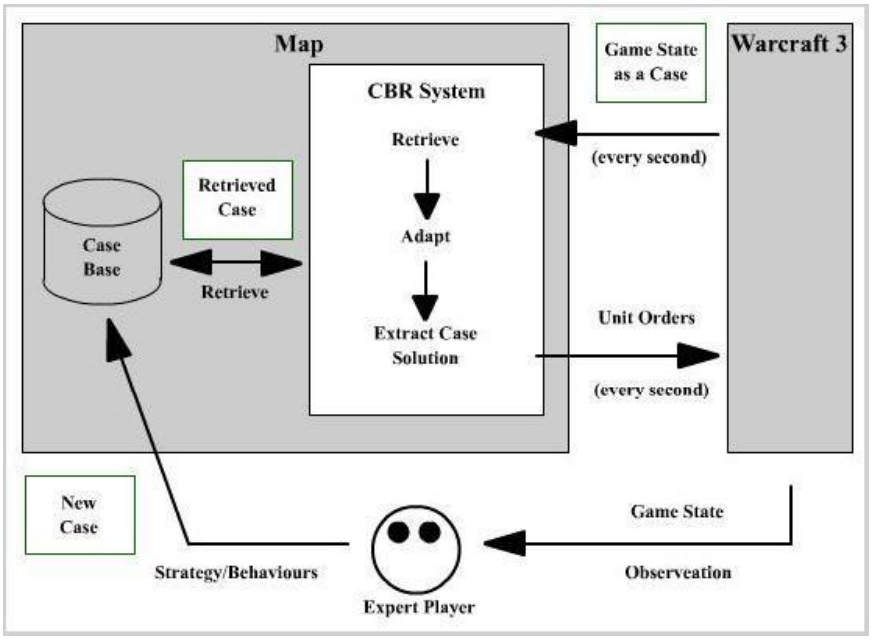

Fig. 2. Overview of the CBR system.

1. Condition Part (primary indexing)
2. Description Part (attribute values when the case was recorded)
3. Solution Part (strategy and unit behaviors)

The condition part of a case consists of a set of conditions that have to be fulfilled to be able to execute the solution part of the case. This is also the primary indexing part of our cases, i.e. only cases where the condition parts can be fulfilled are compared to the current game state. If the solution part of a case has an action that orders a unit to cast a certain spell, the condition part needs to check whether such a unit exists on the battlefield and if it exists, whether it has enough mana points to cast the spell. The description part of a case simply describes the situation when the case was learned. It consists of two arrays, one for each player, which stores our chosen attributes (unit type, remaining hit points, remaining mana points and position). Even though Warcraft 3 allows twelve players to battle each other on one map, we limited our system to two players.

When a case that is most similar to the current game state is retrieved and its conditional part can be fulfilled, the solution part can be executed. The solution part consists of a strategy that contains actions and behaviors. Actions are simply individual unit orders like attacking an enemy unit, moving to a specific point, healing a friendly unit or using an item from inventory. Behaviors are used to both decrease system reaction time (currently 1 second) and the number of cases. A behavior is a set of actions that trigger when some condition or conditions are met.

Fig. 3. A screenshot from the CBR system during training. Left: similarity trace generated by comparing the a new unsolved case to existing solved cases. Upper right: helping tool used by the expert during training. Lower right: overview over learned cases.

Something important might happen that requires a fast response between the 1-second intervals our CBR system currently uses. By using behaviors it is possible to react quickly by setting the important happening as a condition. An example of a behavior of a unit might be to retreat when getting low on health or cast a spell when having enough mana points. Using behaviors can also potentially decrease the number of cases needed. For example if a player has ten different units in his or her army and wants to retreat with a unit once it gets low on hit points, at least ten cases must be learned (one for each unit when its hit points are at a certain low value). By setting an escape behavior on the units involved, only one case would be needed. To make things convenient, mutually exclusive behaviors can be set simultaneously. For example, a unit can have a behavior that triggers differently depending on what type of attribute has changed.

5 Similarity Metric

Our case retrieval process uses a weighted nearest neighbor algorithm. It is a similar algorithm that Ontañón et al. use in their project [7], except for two differences. Our approach does not use goals (therefore the part of the equation matching goals is not included), and adds a weight to each attribute. The similarity metric is as follows:

$$d(c1, c2) = dGS(c1.U, c2.U) \quad (1)$$

Here c1 and c2 are the compared cases while c1.U and c2.U are the sets of units contained in those cases. dGS( ) is a Euclidean distance between game states (between units). To measure distance between two units the following distance is used:

$$dGS(u1, u2) = \sqrt{\Sigma ((pi - qi)/Pi)^2 * Wi} \quad \text{if same unit type} \quad (2.1)$$

$$dGS(u1, u2) = \sqrt{\Sigma 1 * Wi} \quad \text{otherwise} \quad (2.2)$$

u1 and u2 are the units compared. Pi is the maximum value of the attribute i (pi and qi are the values of attribute i belonging to the compared units) while Wi is the weight of attribute i. When units are different, the distances between their corresponding attributes is set to 1.

This metric favors cases with equal amount of units. The similarity metric can have a value between 0 and $2\sqrt{n}$ assuming the weights are set to 1. Here n is the maximum number of units contained in either the compared case or the current game state. n usually has values between 4 and 40, which is the amount of units involved in a typical Warcraft 3 battle. Similarity values of $2\sqrt{n}$ can be obtained by comparing two totally different cases, where n is the number of units in the largest case (the case describing most units). During case retrieval all executable cases are compared, by use of the described metric, with the current game state, and the most similar case (the case with the smallest d(c1, c2) value) is chosen. If no case is found, the units just follow their old orders from the last case executed.

6 Case Matching

The case matching process starts by retrieving the subset of cases from the case base which fulfill the condition part of the new case. This requires a 100% match and increases the performance of the system as well as prevents selection of cases that can not be executed. The similarity assessment is then executed among these cases based on the similarity of the description part of the cases (hit points, mana points, unit type and position).

We implemented four different case matching methods, as detailed by Szczepański [12]:

1. Unit sorting. Units are constantly sorted by remaining hit points. The existing cases are also sorted in the same way. During matching the first unit in the new case is matched with the first unit in an old case etc.
2. Unit similarity. Most similar units matched together, same metric as described in Section 3.3. The existing cases are not sorted.
3. Commitment. Units are numerated before battle, numeration does not change. During matching equally numerated units are compared.
4. Unit sorting with enemy in reverse order. Similar approach as 1. Differs by sorting of enemy units (sorted inversely).

Approach 1 and 4 are motivated by one of the most important principles of micromanagement: killing one unit at the time reduces the DPS (damage per second) of the enemy army faster that killing several units at the time (it is assumed that killing N units at the time takes N times longer than killing one unit). This is because in Warcraft 3 a unit deals the same amount of damage regardless of its remaining hit points unless it is dead. Approach 2 tried to find the most similar units in the current new case (current game state) and the retrieved case when applying strategies and behaviors. The 3rd approach was the simplest possible. Units were enumerated before battle and the enumeration did not change.

The presorting of units in cases can be viewed as a type of case adaptation because this sorting could also have been done after case matching in order to fit the solution of a past case to the current problem. By doing it beforehand efficiency effects are gained. In that sense our system can be viewed as performing case adaptation by enumerating the units on the battlefield in such a way that when the solution part of the retrieved case is executed, the outcome will be similar to the expected/intended outcome observed when the retrieved case was learned.

Each matching approach was tested in a very simple setting of two teams of 5 units battling each other. On that basis approach 4 was selected because it was the approach that needed the least amount of cases to beat a computer opponent. Currently, the system sorts units by their remaining hit points both when considering the current game state, and when considering/comparing to the case description parts. Enemy units are sorted such that the first unit has the least remaining hit points while the CBR system's units are sorted in the opposite way (the first unit has the most remaining hit points). This makes the cases reusable (attacking units with lowest remaining hit points is most often the correct/optimal play).

7 Testing

Having our limited case representation in mind (no information about terrain etc.) and that our CBR system focuses purely on micromanagement, we isolated the testing battlefield to a perfectly flat rectangular area. The battlefield, shown in Figure 3, contains two armies (controlled by the CBR player and the computer opponent) fighting each other in a mirror battle (both battling armies consist of the same units with the same attribute values).

The main purpose of the tests was to compare the performance of our CBR system against human players to the performance of the computer opponent in Warcraft 3 in the same setting. The testing was divided into four parts. We started by training the CBR system against a computer player. We then tested the trained system against the computer player without training. Next, both the CBR system and the computer player were tested against human opponents. Finally, to test the applicability of the CBR system as an aiding tool, both human players and the CBR system were tested in a cooperative mode against the computer player. A test proceeded as long as there were units on the battlefield belonging to different players.

The tests involving human players were divided into three categories: novice, casual and expert players. A player would be considered novice if he had some minor RTS experience while expert would be a person able to easily beat an "insane"

computer opponent (the hardest difficulty of a computer opponent in Warcraft 3). We had 10 persons for the tests. Those were students (the novice players) and people from the Warcraft 3 gaming community (the casual and expert players).

The CBR player was trained by playing repeatedly against a computer opponent. Whenever the system did execute a wrong case (due to lack of a better alternative or a similar case) an expert paused the game and added a new case. After learning 25 cases, our system was able to beat the game AI in Warcraft 3 (this process is summarized in Table 1). After some tweaking and testing of the weight variable during the training, we did not observe any increase in quality of the retrieval process of the CBR system. Thus, we decided to keep the weights constant end equal to 1. A screenshot from the training of the CBR system is shown in Figure 3.

Training step	Cases added	Units lost during test by CBR system	Computer units killed during test
1	7	All	7
2	6	All	11
3	5	All	6
4	3	All	7
5	3	All	7
6	1	5	All

Table 1. Testing resuslts during training.

When finished with the training, we tested the trained CBR player against the insane computer opponent. The CBR player won all of the 10 games by an average loss of 2.5 units out of 14 per game. Next, we performed tests where human players played against a computer opponent. The feedback received from novice players was that the micromanagement provided by the insane computer opponent was simply too hard to beat. Similar feedback was received from the casual players. The expert players on the other hand complained that the insane computer opponent was too easy to beat (the expert players managed many times to beat the computer opponent without loosing a single unit). The same response was received when human testers played against the CBR player. The major difference was that the expert players needed some games to figure out how to beat the CBR system. They did not manage to beat the CBR player without loosing only a few units. Finally players played with the CBR player on their side such that the CBR player was given the control of all unselected units. The purpose of this test was to see whether or not the CBR player could aid human players in battle. Because stored cases can contain solutions with a lot of dependencies, it was interesting to see if the interaction by human players would sabotage those solutions. Interestingly, all groups of players managed to beat the insane computer opponent. The dependency observed was that the novice and casual players did better in this setting than the experienced players. The experienced players found this setting very disturbing because the CBR player was destroying their executed strategies. One example of this was when an expert ordered a near-death unit to retreat while a couple of seconds later the CBR player brought it back into battle. The novice and the casual players liked this setting because they could relax and focus only on their heroes instead of the whole army.

8 Discussion

Our approach managed to create a working system that was capable of beating both computer and novice/casual human players. The system also added more challenge to the micromanagement for experienced players. Having the CBR system on their side, novice and casual players were able to easily beat the insane computer opponent. By using a setup that gives a novice or casual player aid from a CBR player, while putting the CBR player against expert players, one should be able to increase the entertainment value of a played RTS game. However it is also important to note that since our number of human testers was low, the results are a mere indication of what to expect from our CBR system in the future.

Though working, the CBR system also has some weaknesses that were encountered during the training in Chapter 4. During the development phase the unit setup was rather simple and the five attributes used (unit type, hit points left, mana points left, x position and y position) sufficed. Those attributes do not give any information about the past history of what the various units have been doing for the last few seconds. This means that our CBR system does not distinguish between a very active, moving unit and a passive, stationary unit. The system will continue to attack a unit independent on whether or not the attacked unit has been moving around in circles, avoiding most of the attacks. We also observed that our similarity metric was inefficient in many situations. This was because we compared the position x and position y attributes directly instead of looking at the configuration and patterns of units on the battlefield. Using unit positions directly without abstracting it into more complex structures like unit formations, causes bade case reuse.

We also encountered a problem with unnecessary and unintended unit movement. Unit behaviors and actions are defined by the position in the unit list sorted by the attribute “remaining hit points”. When units swap places in this list, they also get new corresponding actions assigned. Repositioning is needed when such units are far from each other. If such units swap a lot, most of the game time will be used for moving units back and forth resulting in huge damage loss. To avoid this problem to appear, an expert needs to foresee this and adjust the strategy such that swaps in the unit list sorted by the attribute “remaining hit points” occur as seldom as possible.

9 Conclusion

The work presented in this paper shows how a CBR system focusing on micromanagement for the RTS game Warcraft 3 can outperform the original game AI in addition to novice and casual human players. Even though we encountered some problems during testing against human players, our approach looks promising.

There are several ways our system can be improved. To prevent unit chasing and inefficient case choosing, our case representation can be extended to include information about both opponent playing style and unit activity attributes. To avoid classifying units that run small distances back and forth as very active units, it is important to not only consider the total distance traveled but also the effective change in position during a time interval. An increased reusability of cases might be obtained

by comparing unit position attributes as patterns and not directly. Better/new case adaptation approach is also needed to reduce unnecessary unit movement. One solution could be to have a dynamic case adaptation that sorts by remaining hit points when the unit count is low, but changes to some more suited case adaptation approach when the number of units increases. Complex actions/behaviors that need longer time to complete are not supported by our approach. Combining macromanagement with micromanagement might be one way to solve this. Another opportunity is to convert our approach into a case-based planning system.

Acknowledgments

The authors wish to thank Helge Langseth, Vidar Holen and Audun Marøy, who through their earlier work [9] inspired the startup of this research. Also thanks to our multinational testing team Aaron Holmes, Asgeir Aakre, Gaute Oftedal, Henrik Skov Jakobsen, Kenneth Wannebo, Marcin Szczepański, Marcus Persson, Rasmus Anker-Nilsen, Sigurd Dahl for their contributions to the evaluation of the system.

References

[1] Aamodt, Agnar, and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39-59, 1994

[2] Blizzard Entertainment, ed. (2002). Warcraft 3 Instruction Manual. pp. 19-30

[3] Blizzard Entertainment. Warcraft 3 Strategy Guide. http://www.battle.net/WAR3/

[4] Buro, M. Call for AI Research in RTS Games, AAAI-04 AI in Games Workshop, San Jose 2004

[5] 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, 2004

[6] David Aha, Mattew Molineaux, and Marc Ponsen. Learning to win: Case-based plan selection in a real time strategy game. In *ICCBR'2005*, number 3620 in LNS, pages 5-20. Springer-Werlag, 2005.

[7] Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, Ashwin Ram: Case-Based Planning and Execution for Real-Time Strategy Games. ICCBR 2007: 164-178

[8] Schwab, B. (2004). AI game engine programming. Charles River Media.

[9] Vidar Holen, Audun Marøy. Reinforcement learning in Wargus. Project Report. NTNU-IDI, December 2007.

[10] Marc J.V. Ponsen, Stephen Lee-Urban, Héctor Muñoz-Avila, David W. Aha, Matthew Molineaux: Stratagus: An Open-Source Game Engine for Research in Real-Time Strategy Games. IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games. Edinburgh, July 2005

[11] M. Buro. ORTS project. http://external.nj.nec.com/homepages/mic, March 2002

[12] Tomasz Szczepański. Case-based reasoning for improved micromanagement in Real-time strategy games. Project Report NTNU-IDI, December 2008.

[13] Pang, Jeff. JASS Manual. http://jass.sourceforge.net/doc/, 2003.