Using behaviour trees to model and compare agents in a turn based strategy game

GROUP 3: 5

Marko Lazic
1995-01-17
mlazic@kth.se

Fredrik Omstedt
1997-03-16
omstedt@kth.se

Abstract

Artificial intelligence is an important part of games. Especially in single player games, AI is in most cases required to make the game fun. It is not enough to simply have AI in the game, it must also not be too easy or too hard as this results in the game being boring. Research in how to create good artificial intelligences is therefore important. In this report, three different AIs for the Battlecode 2018 competition have been compared. More specifically, this paper is part of a group of papers, each implementing its own AI and comparing it with the others. The AIs battled in the competition and were compared on their success rate in matches. It was concluded that harvesting resources efficiently and quickly is vital in Battlecode 2018 in order to be successful, whereas good combat strategies are only important when two AIs battle with an even amount of units. Furthermore, it seems that strategies are highly dependent on the structure of the map played. Due to the experiments conducted, this may have skewed the results, and as such further research must be conducted in which this bias is eliminated.

1 Introduction

When playing games, the use of artificial intelligence (AI) can be very important. In the cases of single player games, AI is in almost all cases necessary for the game to be challenging and fun. Not only must AI exist, but it must also be good. If an AI is too dumb, the game will not be challenging enough, and if it is too smart, the game will be impossible to finish. Both these cases result in boring games. It is therefore important to research and determine how to write good AI.

Battlecode is an annual AI competition hosted by students at MIT [1]. Although the competition differs somewhat from year to year, the main goal is to write an AI for controlling different units in a turn based strategy game. These units are then used to battle opposing AIs, and a tournament is held to determine which AI is the best.

In this report, implementations of AIs used in the 2018 Battlecode competition are described and compared. More specifically, this paper is part of a group of three papers, all of which describe one approach to creating an AI, followed by comparing this AI with the others.

In Battlecode 2018, a game is played on two grid maps containing passable and impassable terrain, one representing Earth and one representing Mars, and is played over 1000 turns or until one player has no units left. After turn 750, Earth is flooded, resulting in all units on that map being killed. The objective of the game is to survive as long as possible, either by killing the opponent's units before the flood, or by escaping to Mars through rockets and continuing the battle there. The surviving player is declared the winner, and if both players survive until the game ends, the value of the players' units are compared, and the one with the highest value is declared the winner.

There are several units in the game: Rockets, factories, workers, knights, rangers, mages and healers. Rockets are used to send units from Earth to Mars, or vice versa. Factories are stationary units used to produce all types of units except factories and rockets. Workers are used to harvest resources (hereonafter described as karbonite) required to build units, and to build factories and rockets.

The remaining units are all used for combat. Knights deal a lot of damage but can only attack adjacent units. Rangers can see and attack from long ranges, but are slower and cannot attack enemies that are too close. Mages attack in blasts, also dealing damage to all adjacent units when attacking a cell in the grid map. Finally, healers cannot attack enemy units, but instead are able to heal friendly units.

Another aspect of the game is the concept of research. It is possible to upgrade the units by spending turns doing research. For instance, workers can be upgraded to harvest more karbonite, and rangers can be upgraded to see farther.

In the experiments described in this report, the concept of travelling to Mars is omitted from the game. This results in no rockets being built, and the game ending after 750 turns (as all remaining units are killed by the flood). This was done due to time constraints and to put focus on the combat aspects of the game.

The AI presented in this report utilized behaviour trees in order to control the different units in a decentralized fashion. All types of units were implemented in this way. Moreover, a global strategy depending on the map was used. More information can be found in section 3.

The AIs battled on two separate occasions. In the first, this paper's AI won all its matches, finishing first of all groups. In the second, it won one third of its matches, finishing last of all groups. This result is largely based on how this paper's AI, whilst having good combat strategies, did not harvest karbonite fast enough compared to the other AIs. More details can be found in section 4.

1.1 Contribution

This paper is especially important in regards to the Battlecode competition. It showcases a comparison of AIs and discusses why some were better than others. This can help people participating in the competition know what to focus on when creating AIs. Furthermore, it might also showcase examples of how AIs might work in turn based strategy games in general, and as such might be useful to the gaming industry as well, as it describes and compares what might be too dumb, too good, and good enough.

1.2 Outline

In section 2, papers related to this one are presented. Solutions to similar problems and similar techniques used are described. In section 3, the implementation of this paper's AI is presented and described. Both the global strategy and the strategy for each unit is explained. In section 4, the experiments conducted are described, and the results from these experiments are presented and analyzed. Finally, in section 5, the paper is summarized and conclusions are drawn from the results.

2 Related work

In this section, various papers related to this one are presented. Their contents and how they relate to this paper are briefly described.

[2] presents behaviour trees as a more modular alternative to Finite State Machines. The paper also describes how behaviour trees work and provides pseudo-code for their implementation. It is concluded that robustness and safety are preserved in behaviour trees while providing higher level of modularity. This was useful knowledge for this paper's implementation, as we tested many different strategies with minor changes. Modularity helps a lot in these cases.

[3] proposes using ADAPTA (Allocation and Decomposition Architecture for Performing Tactical AI) for designing AI for turn based strategy games. This paper used influence maps generated by a neural network combined with an evolutionary algorithm for choosing optimal weights. The AI bot was able to learn how to outperform some simple hard coded tactics. However, this papers focus only on the combat part of AI, while completely skipping resource management and unit choices, as is necessary in the case of this paper.

[4] used reinforcement learning to find optimal placements for the cities in the turn-based strategy game Civilization IV. The Q-learning algorithm which was used was able to outperform the standard game AI in short matches. However, learning showed to be computationally unfeasible after crossing a certain threshold of game complexity. Nonetheless, it showcases that, unlike in this paper, learning approaches may also be used to some extent in turn based strategy games.

In [5], a genetic algorithm is used for optimization of units' movement in turn based strategy games. Though the results presented in [5] were promising, the report suggests using the genetic algorithm during each turn to optimize the units' movement. This might be computationally unfeasible.

[6] used the Minimax algorithm to update the weights in a reinforcement learning algorithm. The algorithm yielded a 71.28% winning percentage against a deterministic rush agent. However, the state space in the game they used is rather small compared to the state space of Battlecode 2018. The game described in [6] has a small 23x23 game board without any obstacles. Using this approach for Battlecode 2018 would be much harder and potentially computationally unfeasible.

3 Method

In this section, the implementation of this paper's AI is presented and described.

3.1 Behaviour trees

The behaviour trees used for all units are similar to those presented in [2]. The general structure of a behaviour tree is that each node can return a status of either success, failure or running. Leaf nodes in the tree correspond to conditions and actions. Conditions check if certain conditions hold, and return success or failure depending on if they do. Actions perform something that changes the state of the game, and return success or failure depending on if the action succeeded, and running if the action has not finished yet.

There are two types of intermediary nodes, fallbacks and sequences. Fallbacks check all children in order, and return success as soon as a child returns success, running as soon as a child returns running, and failure if all children return failure. Sequences on the other hand, return success if all children return success, and failure or running as soon as a child returns failure or running, respectively.

These nodes are used to build the trees containing all logic for the units in this paper's AI. A python framework was written by the authors of this paper in order to do this.

3.2 Strategies

This paper will describe two different Battlecode 2018 strategies. In the first strategy, called Angry Chicken, all units except workers move randomly until they see an enemy. If an enemy is in the unit's vision range, the unit will move towards it and fight it until either unit dies or the enemy escapes the unit's vision range. It will then go back to moving randomly. This strategy is highly defensive since random exploration will keep units reasonably close to the factories which created them, and because only those units who see enemies will engage in fighting them.

In the second strategy, called Kamikaze, a unit will move randomly if no enemies are visible by any friendly unit. If there exist enemies visible by any unit, all units move towards their closest enemy. This strategy is highly offensive since as soon as one unit spots an enemy, all available units will attack it.

Worker behaviour is independent of both these strategies and will always focus on building factories and harvesting karbonite. This is explained more in subsubsection 3.3.1.

3.2.1 Global Strategy

The strategy that will be used globally in a game is determined by three different factors: the number of choke points on the map, the size of the map and the A* distance to enemy starting positions. Choke points are in this paper defined as all points on the A* path between the two AIs' starting positions that transit trough passages that are less then three cells wide. Figure 1 describes how these three factors affect the choice of strategy.

3.2.2 Research Queue

The research order is the same regardless of strategy and was determined based on what was useful in combat situations. The following order was used:

1. Worker level 1: harvest an additional 1 karbonite at a time.
2. Ranger level 1: decreases the movement cooldown by 5.
3. Mage level 1: increases the standard attack damage by 15 health points (HP).
4. Knight level 1: decreases damage done on knights by 5HP.
5. Healer level 1: increases healing by 2HP.
6. Ranger level 2: increases the vision range by 30.
7. Mage level 2: increases the standard attack damage by an additional 15HP.
8. Knight level 2: decreases damage done on knights by an additional 5HP.
9. Mage level 3: increases the standard attack damage by an additional 15HP.
10. Knight level 3: unlocks the javelin ability, allowing knights to also attack units at a short range.
11. Healer level 2: increases healing by an additional 5HP.

3.3 Units

In this section, the action priorities of the units are presented. These determined how the units' behaviour trees were implemented.

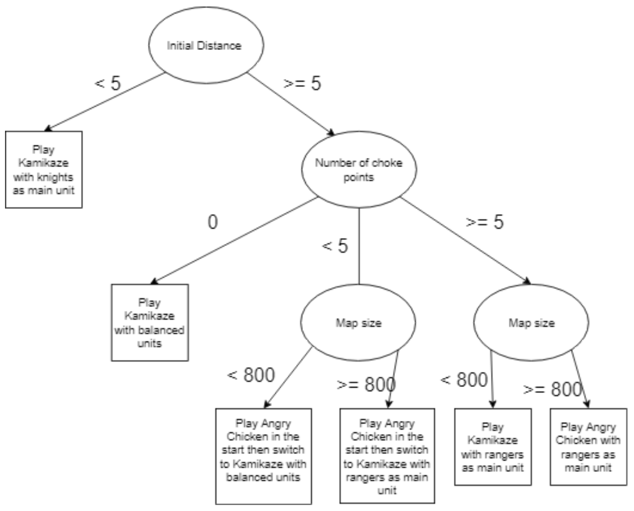
Initial Distance
< 5
>= 5
Play Kamikaze with knights as main unit
Number of choke points
0
< 5
>= 5
Play Kamikaze with balanced units
Map size
Map size
< 800
>= 800
< 800
>= 800
Play Angry Chicken in the start then switch to Kamikaze with balanced units
Play Angry Chicken in the start then switch to Kamikaze with rangers as main unit
Play Kamikaze with rangers as main unit
Play Angry Chicken with rangers as main unit

Figure 1: Tree describing the AI's global strategy choice at the start of a game.

3.3.1 Workers

The following list describes the worker unit's action priorities:

1. Build on an existing factory blueprint. Factories are added as blueprints to the map and require workers to build on them for a fixed amount of turns before being completed. If a worker is adjacent to a blueprint, it will build on it before doing anything else. This action is given the highest priority in order to finish building factories as soon as possible.

2. Avoid enemies. This is done since workers do not have any attack abilities. If a worker sees an enemy it will simply move in the opposite direction if possible.

3. Add a factory blueprint. This action is performed if the following four conditions are fulfilled:

   (a) The current number of factories is less than the needed number of factories (as determined by the global strategy).

   (b) There is enough karbonite to add a blueprint.

   (c) The proposed position is not adjacent to already existing blueprints or factories.

   (d) The proposed position will not block movement of units.

4. Mine Karbonite if an adjacent cell contains some.

5. Find Karbonite. If no karbonite exists adjacent to the worker, the worker uses A* (a heuristic path finding algorithm, described in [7]) to find the path to the closest available karbonite.

6. Random exploration. This is done in order to make sure workers do not do nothing if no karbonite exists.

3.3.2 Factories

The actions of factories are performed in the following order:

1. Unload units created by the factory. This is given the highest priority to make sure units are able to help on the battlefield as quickly as possible.

2. Build workers if needed. This is determined by the global strategy as described in subsubsection 3.2.1

3. Build healers if nearby units are damaged in order to heal them.

4. Build knights if enemies are nearby in order to defend the factory.

5. Build according to the global strategy's needs.

3.3.3 Knights

The highest prioritized action for knights is attacking enemies. If there is an enemy in an adjacent cell the knight will attack it. Furthermore, if the javelin ability is researched and there is an enemy in javelin range the knight will use it to attack. Otherwise if an enemy is in the knight's vision range but not in its attack range, the knight will move towards it using A*.

If no enemy is in vision range, the knight will focus on movement. This movement depends on the current global strategy. If Kamikaze is used, the knight will find the closest enemy and find an A* path towards it, which it then moves on. If Angry Chicken is used, or if no enemy was found in Kamikaze, the knight will move in a random direction.

3.3.4 Rangers

Rangers' attack pattern is slightly different from other units since they cannot attack enemies that are too close to them. If an enemy is in a ranger's vision range one or more of the following three actions will be performed:

1. Attack enemies if they are in the ranger's attack range.

2. Move away from enemies if they are too close.
3. Move towards enemies if they are too far away.

Since units can move and attack in the same turn, it is possible for the ranger to perform several of these actions.

If no enemy unit is in a ranger's vision range, the ranger moves in the same way as knights.

3.3.5 Mages

Just like knights and rangers, mages will try to attack enemies if they are within their vision range. Since mages deal damage to adjacent cells as well, they will find the best target cell to attack by maximizing the number of enemies damaged by the attack, and minimizing the number of friendly units damaged. The cell that best fits these criteria is the cell a mage will attack.

If no enemy unit is in a mage's vision range, the mage moves in the same way as knights and rangers.

3.3.6 Healers

Healers can only heal nearby damaged friendly units. If there is a damaged friendly unit in healing range, the healer will find the one with the lowest HP and heal it. If no friendly damaged units in are in range, the healer will instead find the closest one and move towards it using A*. If no injured friendly damaged units exist, the healer will explore randomly.

4 Experimental results

In this section, the setup of the experiments conducted is described. Moreover, the results from the experiments are presented and analyzed.

4.1 Experimental setup

The experiment consisted of three groups implementing AIs for Battlecode 18 during a period of 3 weeks. In the final week all groups ran their AIs against each other on two separate occasions (the semi finals and the finals), with two days in between these occasions to make room for improvements. Each group got to choose one home map to play on. All groups played 3 matches against all other groups: one on their home map, one on the opponent's home

Bot 1	**Bot 2**	**Map**	**Winner**
BAUBot	!losers	Desert	BAUBot
!losers	BAUBot	Socket	BAUBot
!losers	BAUBot	Chambers	BAUBot
BAUBot	DanMark	Desert	BAUBot
DanMark	BAUBot	Corners	BAUBot
BAUBot	DanMark	Goldrush	BAUBot
DanMark	!losers	Corners	DanMark
!losers	DanMark	Socket	DanMark
DanMark	!losers	Mirror	DanMark

Table 1: Results of the semi finals matches between all groups.

Match	**Result**
BAUbot vs DanMark	3-0
BAUbot vs !losers:	3-0
!losers vs DanMark:	0-3

Table 2: Summarized results of the semi finals.

map, and one on a random map. The AI implemented in this paper is called BAUBot.

4.2 Experiments

Table 1 and Table 2 show the results of the semi finals matches while Table 3 and Table 4 show the results of the finals matches.

These results are interesting since the losing group of the semi finals

Bot 1	**Bot 2**	**Map**	**Winner**
BAUBot	!losers	OldTown	BAUBot
!losers	BAUBot	Socket	!losers
!losers	BAUBot	Cross	!losers
BAUBot	DanMark	OldTown	DanMark
DanMark	BAUBot	Orgo	DanMark
BAUBot	DanMark	Joust	BAUBot
DanMark	!losers	Orgo	DanMark
!losers	DanMark	Socket	!losers
DanMark	!losers	Joust	!losers

Table 3: Results of the finals matches between all groups.

Match	**Result**
BAUbot vs DanMark	1-2
BAUbot vs !losers:	1-2
!losers vs DanMark:	2-1

Table 4: Summarized results of the finals.

became the winning group of the finals, and vice versa. The reason for this change is quite easily explained. In the semi finals, both DanMark and !losers had erroneous behaviour in their workers, resulting in them not harvesting karbonite as expected. Furthermore, all groups produced a small amount of workers for all maps, which meant that factory building and resource gathering was slow. Since most maps played were quite big, this gave BAUbot the time to generate units in order to be able to properly attack units. Since BAUbot did not have issues in harvesting karbonite, and since it utilized healers (which none of the other groups did), it outperformed the other AIs and could easily win the matches.

During the finals, the other groups had fixed these problems, but they had also realized that resources are very important, especially in the early game. Workers have an ability in which they can replicate for an amount of karbonite to create new workers, and this sped up the process of creating factories. Because of this, the other groups could mass produce units and rush towards the opponent. Since BAUbot did not consider this, its units were often quickly destroyed by hordes of enemies.

It was however clear that BAUBot's combat strategies were very good compared to the other groups. In Table 1, it is shown that BAUbot won against !losers on the map OldTown. This map is full of choke points and is quite big, which resulted in BAUbot having just the right amount of time to be able to defend against incoming hordes of enemies. !losers had the upper hand for the majority of the game due to their rush in producing units, but BAUbot could defend itself and this resulted in BAUbot winning.

It is therefore clear that being able to efficiently and quickly harvest karbonite is a key factor in Battlecode 2018. This was the definitive reason for BAUbot performing so poorly in the finals. However, one question remains: why did !losers win over DanMark?

It is assumed that the answer to the above question stems from the choice in maps. DanMark utilized a strategy which premiered moving to the middle of the map in order to take control of it. However, Socket is such a small map that fights between the AIs are bound to happen almost instantaneously. The moving of DanMark might have wasted time needed for building factories, giving !losers just a few rounds of an advantage.

Furthermore, Table 3 shows that DanMark faced both groups on the map Joust and lost both times. This map is also small, but it is quite difficult to move from one group's starting point to the others due to choke points. It is therefore quite easy for rangers and mages to pick off units as they come towards the middle. Therefore, DanMark's strategy of moving towards the middle failed here as well. The authors of DanMark also mentioned that the workers seemed to behave erroneously on this map, so this might also have affected the results.

From the above paragraphs, it is shown that it is not certain that the final standings would have occurred if other maps would have been played. It might have been better to play more random maps in order to determine the best group's AI, as this would have eliminated bias towards any maps played. Further research must be conducted in order to determine if this did affect the results.

5 Summary and Conclusions

In this report, three different AIs written for the Battlecode 2018 competition in regards to their success against each other in the competition. More specifically, three papers have been produced, each specifying the details of their own AI and how it compared to the others. In this paper, the AI (called BAUBot) utilized behaviour trees to decentralize decisions made by the units on the map. A global strategy was also used to determine the amount of different units needed, and whether to be more offensive or more defensive during matches.

From the results, it was shown that harvesting karbonite quickly and efficiently is one of the most important aspects in the game. This was determined by seeing how BAUBot went from the dominating AI in the semi finals, in which the other groups had erroneous harvesting strategies, to the worst AI in the finals, in which the other groups had focused a lot more on harvesting. Superior combat skills are important when facing a similar amount of enemies, something that was shown when BAUBot faced !losers on OldTown, but being able to produce as many units as possible as quickly as possible is very important.

Finally, it seems that the choice in map affects the results, and could be the cause of !losers winning the finals. Since only one random map was chosen in each match, this affects the results quite a lot. Further research must therefore be conducted in which more maps are battled upon between groups to determine which AI is the best overall.

References

[1] Battlecode. `https://www.battlecode.org/`. Accessed 2019-04-29.

[2] Michele Colledanchise and Petter Ogren. How behavior trees modularize robustness and safety in hybrid systems. pages 1482–1488, 2014.

[3] Maurice Bergsma and Pieter Spronck. Adaptive intelligence for turn-based strategy games. *Vigiliae Christianae*, 01 2008.

[4] S. Wender and I. Watson. Using reinforcement learning for city site selection in the turn-based strategy game civilization iv. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 372–377, Dec 2008.

[5] Kristo Radion Purba, Liliana Liliana, and Johan Pranata. Optimization of units movement in turn-based strategy game. *JIRAE (International Journal of Industrial Research and Applied Engineering)*, 1(1):33–37, 2016.

[6] Sulaeman Santoso and Iping Supriana. Minimax guided reinforcement learning for turn-based strategy games. In *2014 2nd International Conference on Information and Communication Technology (ICoICT)*, pages 217–220. IEEE, 2014.

[7] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.