Technical report

Battlecode 2021 — M.A.R.S.

Dik van Genuchten, Koen Ligthart,
Max de Louw, Gijs Pennings
mars2021@serpentineai.nl

2020

Abstract

We, a team of students from Eindhoven University of Technology and Serpentine AI, created the Ministry of Alien RattleSnakes bot (M.A.R.S.) to participate in the 2021 Battlecode competition by MIT. An adaptive voting system was combined with thorough scouting and semi-centrally coordinated attacks, with the goal of securing an early map-wide advantage. To this end, a communication protocol was devised, as well as an algorithm to scout the complete map and attempt to achieve a uniform presence. The bot's performance could have been further improved by fine-tuning parameters (for example, for managing the economy) and implementing more extensive communication between units (to coordinate a global strategy, for instance). In the end, we finished 88 out of 559 teams in total.

Contents

1 Introduction

Battlecode is a yearly competition hosted by the Massachusetts Institute of Technology (MIT) [4]. Like last year, we participated in this contest as a team from Serpentine [2, 3]. The competition is a game on a 2D grid on which no real players compete, but rather bots made by teams of four people from around the world. In less than a month's time between publication and final submission, teams must create the best bot possible. Teams can challenge each other to so-called scrimmages, which determine their ranking according to an Elo rating system. There are also official competitions with deadlines throughout January in which all teams participate, the last of which decides the final ranking.

Every year, a theme is picked for Battlecode; this year the theme was politics. For this reason, units in the game have been given roles fitting this theme. For example, the 'Politician' unit has the ability to convince other units. More details are provided in the next section.

In this report, we will first shortly describe the rules of this year's game. Next, we will discuss the strategy and implementation[1] of the M.A.R.S. bot, the results including our ranking, and then possible improvements to our final submission. Finally, the ideas covered are shortly reiterated.

2 Environment

Each game consists of at most 1500 rounds, which act as ticks for the code of both teams. After that, the winner is determined. In each of these rounds a vote is called, and both teams can vote 'influence' (the currency of the game). The team that bids the largest amount wins that round's vote. At the end of the game, the team that received the most votes wins the game. Alternatively, when all of a team's robots are destroyed, it loses the game, even if not all 1500 rounds have passed. This is only a rough overview; for specifics please refer to the official specification [5].

The map is a symmetric, rectangular, 2D grid on which headquarters are placed (details below). These are either neutral or owned by one of the two players. Furthermore, each tile has a certain 'passability', which indicates how fast units can move through it. Figure 3 shows an example of a map.

There are four types of robots, of which only the Enlightenment Center cannot move. All robots have an attribute called 'conviction', which represents their health points. The robots can be summarized as follows.

- **Enlightenment Centers** (ECs) produce units (at most one per turn) and are responsible for bidding influence to win votes. They act as headquarters, but note that a team can own multiple of them.
- **Politicians** have the ability to 'empower' in a certain radius, during which the Politician itself is destroyed. Its conviction is distributed over all robots in its range: friendly robots will gain conviction, while enemy robots will lose it. Afterwards, enemy ECs and Politicians with negative conviction are converted to your team, while Muckrakers and Slanderers are simply destroyed.
- **Slanderers** generate influence every round for the first 50 of their existence, and convert to Politicians after 300 rounds. They represent fake news reporters, and are indistinguishable from Politicians to enemy Slanderers and Politicians.
- **Muckrakers** have the ability to 'expose' Slanderers of the other team, killing them.

Communication between robots is very limited, which is one of the main challenges of the competition. Robots were only able to exchange information to nearby robots by changing their appearance in the form of a colored flag, represented by a 24-bit integer. Robots can change their flag color at most once per round, which severely limits the amount of data outbound messages can contain. ECs serve a special role because they can read the flags of all robots, and because all robots can read theirs.

[1] The source code can be found on GitHub: `https://github.com/TeamSerpentine`.

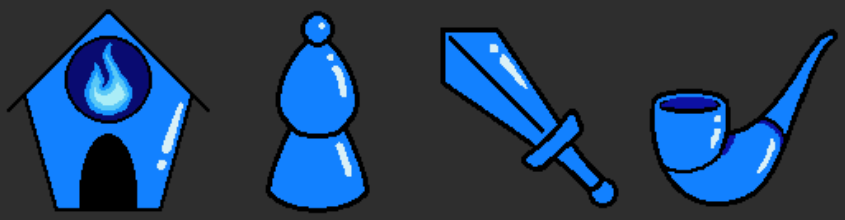

Figure 1: Images of each of the robot types in the game; from left to right: EC, Politician, Muckraker, and Slanderer.

3 Strategy

3.1 Build order

The main strategy of the M.A.R.S. bot can be summarized by investigating the order in which an EC builds its units. This is a prioritized list of possible units that an EC can build to do certain tasks. When an EC detects enemy Muckrakers within its range, its highest priority will be to build Politicians to defeat the opposing Muckrakers. When the visible area is clear of Muckrakers, the EC then proceeds to build Slanderers until its income per round exceeds $\sqrt{\text{round}/2} + 8$. Before Slanderers are built, it is ensured that there are enough defensive Politicians to protect these Slanderers. Specifically, more are built if less than 10 influence is invested into Politicians for each Slanderer. When an EC builds a Politician, it gives either an offensive or a defensive label, which (partly) determines the behavior of the Politician. In the early stage of the game (rounds 1-50), ECs make some Slanderers regardless of the fact that there will not be any defensive Politicians to defend them, because the enemy will likely not have that many Muckrakers. By building these Slanderers, a steady early supply of influence income is generated.

After considering these building possibilities, if the EC knows locations of other neutral or enemy ECs, it will try to make offensive Politicians and instruct them to head towards that location to gain control of the EC. If none of the previous conditions are met, the EC will build either Muckrakers or offensive Politicians. Offensive Politicians will only be built if the EC has an abundance of influence, specifically more than $20\sqrt{\text{round}} + 4 \cdot \text{income} + 200$, where 'income' is the influence income per round from Slanderers. This is to make sure that large amounts of influence do not remain unspent during the course of the game. In practice, Muckrakers are built more often and also mainly in the earlier rounds after the initial few Slanderers have been built. Table 1 provides a succinct overview.

Priority	Condition	Unit	Function
1	Nearby enemy Muckrakers	Defensive Politician	Defeat enemy Muckrakers
2	Not enough Slanderer protection	Defensive Politician	Create preventive defense
3	Not enough income	Slanderer	Increase influence income
4	There are known attack targets	Offensive Politician	Attack the known EC
5	EC has expandable influence	Offensive Politician	Invest influence to prepare for future attacks
6	(otherwise)	Muckrakers	Explore the map and expose enemy Slanderers

Table 1: Build order of an EC.

Using this build order, the bot starts off by creating 4-5 Slanderers to generate income. Then, it creates approximately 10 Muckrakers to look for map features such as ECs and map borders. Often, an early target is found, which is then attacked. It was decided to spend around 10 influence on defensive Politicians for each Slanderer, which turned out to be relatively little. Therefore, the bot is rather offensive compared to other bots.

3.2 Communication

All robots actively use their flags to communicate information. For a unit, almost all communication is between it and the EC that built it. A brief per-robot overview is provided below.

- Muckrakers encode the locations of ECs into their flags.
- Politicians instruct other Politicians that do not know an allied EC (e.g. when 'their' EC was conquered by the enemy or the Politician was converted) with the ID of their own EC.
- Slanderers give off signals when they are threatened by enemy Muckrakers.
- ECs read off the flags of Slanderers when they are being threatened and use this information to instruct defensive Politicians to go to that location. They also transmit the attack targets obtained through the flags of Muckraker scouts.

3.3 Voting system

The goal of our voting strategy was to spend the least amount of influence in total, such that the M.A.R.S. bot held the majority of the votes in the end. This involves carefully choosing which rounds to skip, and how much to spend on other rounds. Hence, it is necessary to determine the minimum amount of influence to outbid the opponent. In addition, enough influence needs to be kept so units can be built.

Therefore, it was chosen to record how much influence was spent every round, as well as to keep track of the amount of votes that were won in total. This way, it was possible to check whether the last vote was won, in which case the M.A.R.S. bot bids less. When the previous vote was lost, it bids more. Furthermore, a minimum of 3 influence was always voted, to ensure the vote was won when the opponent withheld.

3.4 Units

3.4.1 Muckraker

Muckrakers had two purposes: taking out enemy Slanderers and scouting the map. Since enemy Slanderers can be dealt with regardless of the Muckraker's or Slanderer's conviction, only Muckrakers with 1 influence are created. If multiple enemy Slanderers are in range, the strongest is exposed. To maximize the chances of finding and killing as many enemy Slanderers as possible, Muckrakers spread out as far out and as uniformly as possible, the details of which can be found in section 4.3. This behavior also made them prime subjects to scout the map, especially because of their high sensor and detection radius. Using the flag to communicate with their 'home' EC, Muckrakers send back the location of the border and ECs of any affiliation.

3.4.2 Politician

The M.A.R.S. bot made a distinction between offensive and defensive Politicians. The main difference between them was which 'orders' from the EC they followed. Offensive Politicians were given enemy or neutral ECs to target, while defensive Politicians received locations of approaching enemy Politicians or Muckrakers.

Before moving, a Politician first always scans the area around it, regardless of its type. For each distance away from the Politician, it would count the number of enemy units and determine the maximum conviction of all enemy units within that range. Then, it would try to move towards the target, or, if no target was known, randomly around its associated EC. If close to its target, it would determine the smallest radius such that (1) all enemy units within it would be killed or (2) its target would be affected. If such a radius was found, it would empower with that radius.

3.4.3 Slanderer

During the first 50 turns of their existence, Slanderers generate an income equal to

$$\lfloor (\tfrac{1}{50} + 0.03e^{-0.001x}) \cdot x \rfloor$$

every turn, with x being the initial investment made into the Slanderer. Due to the floor function some values yield a higher return on investment than others. When needed according to the build order, an EC builds the most expensive Slanderer from a pre-calculated list. This list consists of possible influence investments with minimal loss to the floor function. It contained no values greater than 1000, since at that point the Slanderer costed more influence then it generated.

Much income is lost when enemy Muckrakers kill Slanderers. To prevent this as much as possible, Slanderers try to flee from Muckrakers. Slanderers wander towards the edges of the map and then stick to those, running away from any enemy Muckraker within its radius.

4 Implementation

4.1 Class structure

Although each robot type has unique abilities, they share much common functionality as well. To minimize code duplication, inheritance was used to reuse common functionality between robots. Specifically, the functionality of each robot type was implemented in a subclass of the `Robot` or `Unit` base class, where `Unit` itself is also an (abstract) subclass of `Robot`. The `Robot` base class contains methods that are useful for all robot types (such as a utility method to return the closest of two locations to the robot's location), while the `Unit` base class extends this with methods that were useful for *mobile* robots (such as a utility method to try to move in a random direction). Next to utility methods, the main purpose of these base classes (and specifically `Robot`) was to provide a common framework for all robots. For example, it stores the `RobotController` object on which all actions (such as moving or using an ability) are executed.

The main entry to the M.A.R.S. bot is the `RobotPlayer` that, depending on the robot's type, created an object of the correct concrete class and invoked the `loop` method. This mechanism can be classified as the strategy design pattern [1]. The `loop` method is defined in the `Robot` base class, and calls the abstract `step` method every turn. Concrete subclasses only needed to override this `step` method, and can therefore abstract away from turns altogether.

4.2 Flag communication

The flags of all robots contain three dedicated bits that give information on what the type of message is that they transmit via their flag. The bit layout of the 24-bit flags can be found in Figure 2. The rest of the flag contains coordinates and possibly a Politician type.

The flag of the EC contains a special bit, called the 'scanbit', which is required for seamless communication from units to an EC. Because an EC might have built a lot of units, processing and reading all flags can consume a lot of bytecode, which makes it infeasible to read all flags in a single round. Therefore, this sometimes needs to be distributed over multiple rounds. By inverting the scanbit of the EC flag after it has read all flag of all its units, the units can have a guarantee of whether their flag has been read. This can be used by the units to determine the amount of rounds that they need to present their flag with a message for the EC. This system is not required for messages destined to be read by units, as those do not have to spend large amounts of bytecode on flag reading. These messages are just shown as the flag for exactly one round.

The map boundary coordinates are not known by default, and can have large values. Therefore, coordinates are encoded into 7 bits using modular arithmetic. Because it is known that the map is at

most 64 by 64 tiles large, units can deduce the absolute coordinates from coordinates modulo 128 by comparing them with their current location.

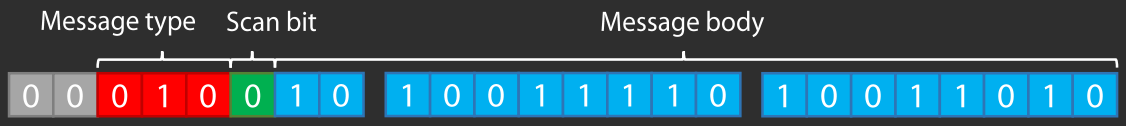

Figure 2: The bit layout used for flag communication between robots. The two leftmost bits are unused.

4.3 Muckraker dispersion

To make Muckrakers spread out as uniformly as possible, they were modeled as charged particles in an (electric) field. The specifics are discussed below. This results in emergent behavior, preventing any part of the map from being missed. Note that the border was also considered 'charged'. Using this technique, many Muckrakers grouped together (which is the case at, for example, the EC that spawns them) will automatically dissipate outwards. With enough Muckrakers and time, this will average out to a global network of Muckrakers on the map. An example of this in the early stages of the game can be seen in Figure 3.

At the start of every turn, every Muckraker scans all units around it. If a unit is a friendly Muckraker, it is added to a list of 'repellers'. Next, it is checked whether any map borders are inside the sensor radius, which are then also added to the list. This is done by checking the four cardinal directions in the following fashion. First, it is checked whether the tile furthest away, but still within sensor range, is on the map. If so, there is no border in that direction. If not, all tiles between that tile and the location of the Muckraker itself are checked linearly. The search is stopped as soon as the first map-tile is encountered. Note that if, for example, a border has been found north of the Muckraker, the south does not have to be checked. This is due to the minimum map size and maximum Muckraker sensor radius.

This procedure yields a list of repellers, which can be compared to charged particles. Now, for each possible direction to move in (including `CENTER`, not moving at all) the value of the 'field' is calculated at that point, by summing the reciprocals of the distance of all 'repellers' to the Muckraker. The inspiration for this was Coulomb's law, namely

$$F = k\frac{q_1 q_2}{r^2},$$

where r is the distance between two charged particles. Finally, the Muckraker tries to move in the direction in which this sum is the smallest.

5 Results

5.1 Strategy results

Because the Battlecode scrimmaging platform does not provide clear bot performance statistics, this section aims to give an insight into how specific strategies of the M.A.R.S. bot behaves in some scenarios.

After creating a few initial Slanderers, the bot often manages to gain a significant amount of map vision by scouting with Muckrakers which can be seen in Figure 3.

When the enemy bot does not create defensive units quickly and an enemy EC is known, the M.A.R.S. bot usually manages to take over nearby enemy ECs. An example of this can be seen in Figure 4.

The offensive strategy does not always work. In Figure 5 it can be seen that the M.A.R.S. bot controls 5/6 ECs at round 165, but that the enemy team has a much larger income because they invested in

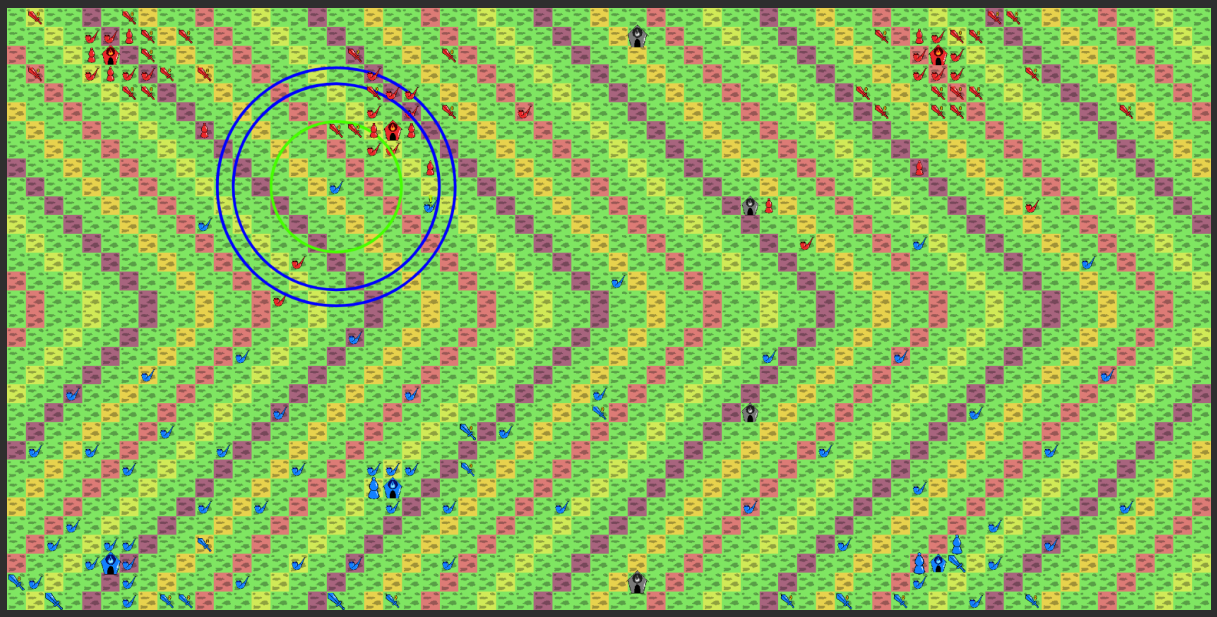

Figure 3: The M.A.R.S. bot (blue) scouting with Muckrakers at round 52. The highlighted Muckraker communicates the location of the enemy EC to their own EC.

Slanderers in the top left corner. Because of the large amount of income that the red team has, they were able to take over all other ECs with their Politicians at round 559.

5.2 Competition results

In the end the M.A.R.S. bot had an ELO-rating of 1313, ranking 88 out of 559 teams, and during the International Qualifying Tournament lost its first match, bringing it in the loser bracket in which it won 3 rounds (out of 7), eventually losing against 'Serpentine – Viper', the other team from Serpentine. The recordings of all matches during the Qualifying tournament are available online[2].

6 Discussion

6.1 Economy

While the build order as explained in Table 1 worked well generally, it lacked when resources were sparse. For example, on some maps, especially if the opponent exposed Slanderers early, too little Slanderers were produced, which significantly slowed down the economy and hampered production of other units in the process. In extreme cases, long stretches of time were encountered where no units were produced at all, which left the position extremely vulnerable.

6.2 EC improvements

A major limitation of the overall strategy was the fact that it was local to every EC individually. In other words, there was no communication between ECs. Using flags, ECs could have 'exchanged' plans, resulting in a more coordinated response to threats or vulnerabilities in the enemy's defense, in the ideal scenario.

Another limitation concerning ECs was the reliance on the 'home base' for every unit. Almost all communication between units went through the EC that spawned them. In practice this worked well, until this EC was taken by the enemy, which left its associated units uncoordinated. This could have been solved in two ways: (1) by exchanging strategy locally between units, without using the EC

[2] `https://challonge.com/bc21_quals_b9d3af0/`

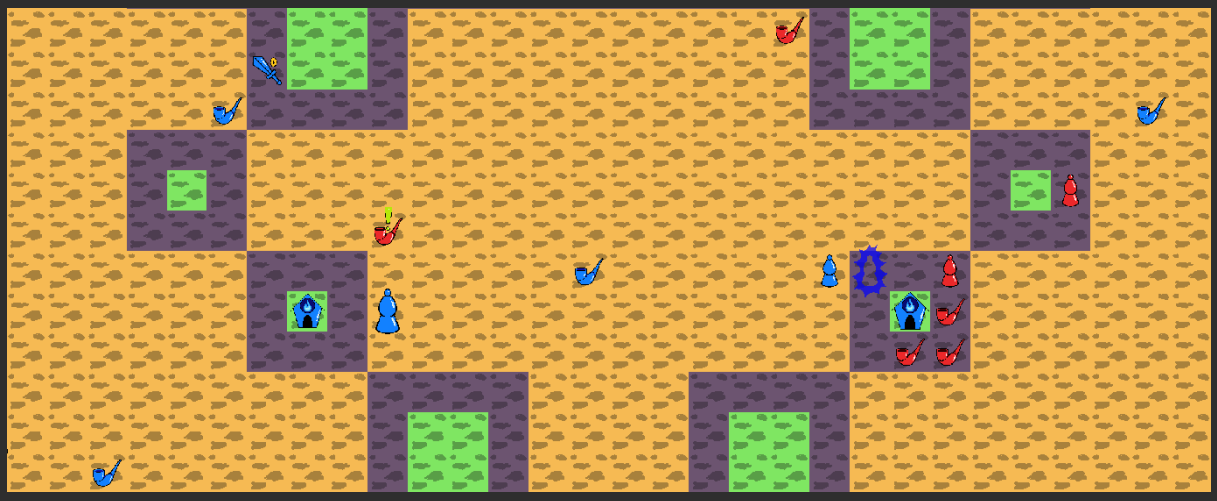

Figure 4: The M.A.R.S. bot (blue) attacking the right EC with politicians. The blue outline of a pawn indicates that a blue Politician is empowering.

as an intermediary, and (2) by having units designating another EC as their 'home base', after losing their previous to the enemy. Of course, (1) is limited by sensor range, but can still be useful in many scenarios.

6.3 Scouting issues

The scouting strategy that was employed (as described in detail in section 4.3) resulted in a global, uniformly spread out network of Muckrakers, meaning no places (ECs or borders) were missed. However, it was slow to start, which meant that opponents using a simpler but faster scouting method, such as 'running' with one Muckraker in every one of the eight directions, would sometimes get the upper hand by finding important landmarks earlier. For optimal results, both strategies could have been combined into a hybrid to provide both fast initial exploration, as well as late-game control and vision.

Furthermore, when the Muckrakers had map control they did not communicate local danger (e.g. enemy units) to the nearby friendly units. If the M.A.R.S. bot were to either neutralize the danger (Politicians) or flee from the danger (Slanderers), it would have been stronger.

6.4 Vulnerable Slanderers

Although Muckrakers did scan for and report map borders, the ECs never used this information. For instance, they could have been used for symmetry calculations, to determine locations of other ECs. Perhaps more importantly, this information was not given to Slanderers, which could have used it to determine better 'hiding' spots. In the latest M.A.R.S. bot, Slanderers would often 'clump up' near the ECs where they were created, or next to a border close to it. Once an enemy Muckraker had spotted this group, it became too easy of a target. Using border coordinates, however, would potentially allow Slanderers to hide near the boundary of the map, away from the action near the ECs. Another improvement concerning Slanderers would have been to accompany each Slanderer by a 'body guard' of one or more (defensive) Politicians.

6.5 Pathfinding

Unfortunately, we did not come around to implementing a pathfinding algorithm. The M.A.R.S. bot performed surprisingly well without it, partly due to the fact that this year's Battlecode featured no impenetrable obstacles (apart from other units). Furthermore, we implemented a simple utility function[3] that applied a basic strategy to maneuver around obstructing units. If, for example, a unit

[3] `tryMovePreferred` in the `Unit` class

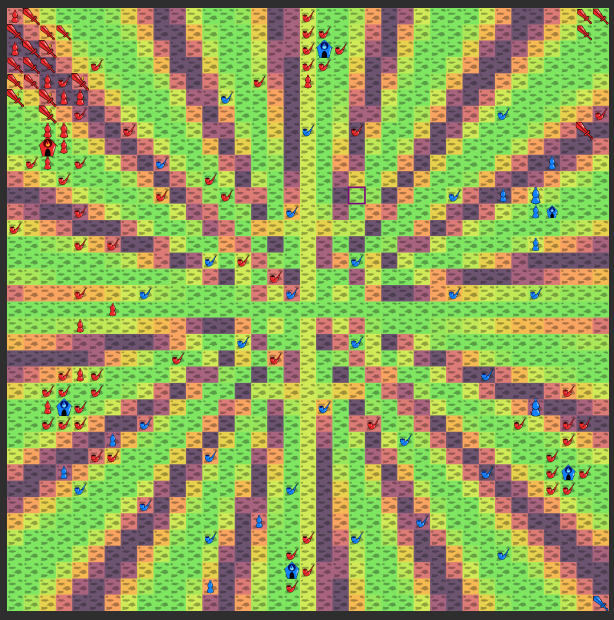

Figure 5: The M.A.R.S. bot (blue) controlling 5/6 ECs at round 165. The blue outline of a pawn indicates that a blue Politician is empowering.

needed to move north, it did this by not only trying to move north directly, but also north-east and north-west. This very simple function proved useful in preventing deadlocks.

7 Conclusion

In the end, the M.A.R.S. bot is capable of generating influence, scouting and controlling the map, as well as (semi-)organized attacks on the enemy ECs. The lack of decentralized communication was a major weak point, since each channel of communication was limited to the EC and the respective unit only. For example, when the Muckrakers had map control they did not communicate local danger to nearby friendly units. Furthermore, most parameters remained static throughout the game and were chosen arbitrarily, and thus could not be adapted to the environment. In summary, the individual systems within the bot were good, but due to the lack of interaction between them, they were not able to react optimally in a lot of cases. Given more time, these problems could have been solved, but note that this time constraint is also a challenge of the competition.

8 Acknowledgments

This project was partly done within Serpentine, the student team of the Eindhoven University of Technology (TU/e) that competes in AI programming contests. We would like to thank the TU/e, as well as our industry partners VBTI and Flowserve for supporting our work. Furthermore, we want to thank MIT for organizing this competition each year and opening it up for all students.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1994.

[2] D. Genuchten, B. Grooten, and M. Ronhaar. Noodles. Technical report, Serpentine, March 2020.

[3] C. Lepelaars, T. Tomilin, and P. Voors. SnakeEyes. Technical report, Serpentine, March 2020.

[4] MIT. Battlecode 2021. `https://2021.battlecode.org`.

[5] MIT. Battlecode 2021 — specification. `http://2021.battlecode.org/specs/specs.md.html`.