

Battlecode 2021 Postmortem

Isaac Liao

February 4, 2021

Overview

This document provides an overview of my experience participating in the Battlecode 2021 competition on a one-person team called **wololo**, which placed seventh in the final tournament. I first outline the rules of the competition game, then I explain the theoretical and strategical side of the game that arises from these rules, and finally I detail my implementation of the strategies found, as well as the algorithms used to do so. I finish with a timeline of my code’s development with respect to major events throughout the duration of the competition. At the time of this competition, I was an MIT sophomore student who had already competed in the previous Battlecode 2020 competition on a one-person team called **asdf**, which championed the Newbie Tournament and placed 13th in the US Qualifying Tournament.



Figure 1: The popular game Age of Empires II was the first game I ever coded an AI for. Players often recollected the monk from Age of Empires I, who made a peculiar sound resembling “wololo” while converting other players’ units to his own team.

1 Competition Game Rules

The Battlecode 2021 game featured a rectangular map of width and height between 32 and 64 square tiles of unit length, each with a “passability” value between 0.1 and 1. “Robots”, classified as “units” and “buildings”, each with their own non-negative amount of “conviction”, belonging to either of two teams or which are “neutral”, were situated on each of these tiles. The game consisted of 3000 rounds (later changed to 1500 by **Teh Devs**), and in each round, every robot would run one iteration of its team’s code, whose purpose was to decide whether the robot should take an action for that round, and what action to take. Each robot began a “cooldown” period after every action during which no other actions could be taken, and whose duration was inversely proportional to the passability of the tile on which the robot resided. Additionally, a “vote” was provided on each round, which would be awarded to the team of the building which chose to bid and pay the greatest amount of its conviction for that round. If at any time before the last round, one team owned no units and buildings, the opponent team won and the game ended. If the last round was reached, the team which had accumulated the greater number of votes won. The competitors on each team were responsible for providing the code for their team which their robots ran, in order to try to produce the winning outcome when the game was played between two teams.

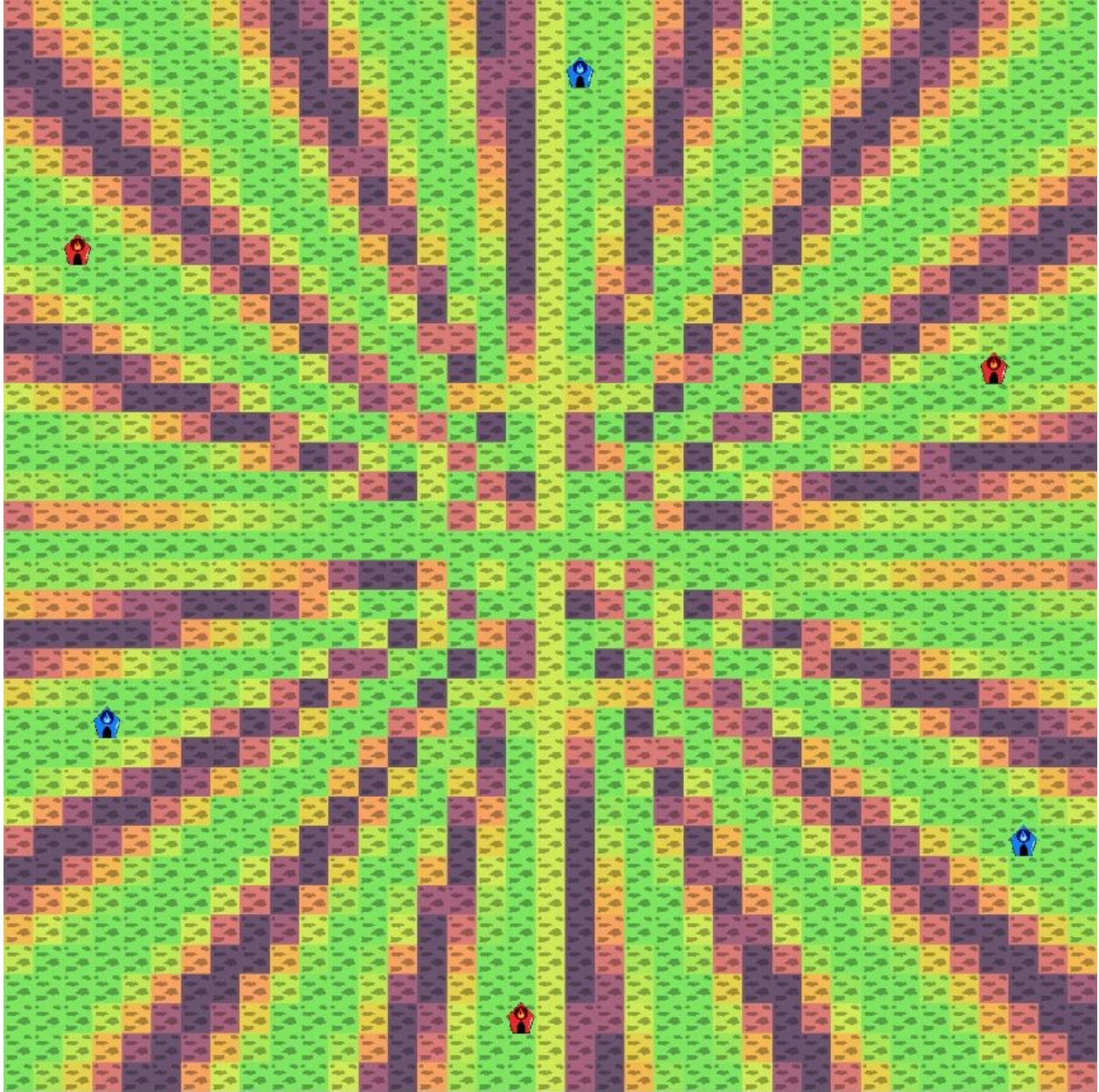


Figure 2: The map Radial used in the Battlecode Sprint 2 Tournament. Greener tiles were more passable while purple tiles were less passable.

1.1 Robots

There were four types of robots—three units and one building:





- An **Enlightenment Center** (EC) was a building which could not move. Its only allowed action was to build any kind of unit on an adjacent tile¹ using any amount of its own conviction, which it transferred from itself to the unit. This amount of conviction was then called the unit’s “influence”, which acted as a cap for the amount of conviction that the unit could have at any given time. If a muckraker was built, then only 0.7 times the amount of conviction spent was transferred to the muckraker, and the other 30% was lost. All non-neutral ECs passively gained some influence per turn, proportional to the

¹Adjacency was defined as the eight tiles surrounding a single tile.

square root of the round number, to allow them to continue building so that the game did not stagnate.

- A **Slanderer** was a unit whose only allowed action was to slowly move to an adjacent tile. For the first 50 turns of a slanderer’s life, it “embezzled” giving the EC which built the slanderer some conviction every turn, so long as the slanderer existed and the EC did not switch teams. The amount of conviction produced per turn was linear multiplied by a decaying exponential, in the influence of the slanderer. If a slanderer existed for 300 turns, it then “camouflaged” and became a politician.
- A **Politician** was a unit whose possible actions were to quickly move to an adjacent tile or to “empower” with a specified radius of at most 3, removing itself from the map, and dividing and transferring its own conviction (after an initial tax of 10 conviction) equally among other robots within the radius, increasing the conviction of affected friendly robots and decreasing the conviction of all other affected robots. Any affected robot that ended up with a negative conviction was removed from the map if it was a slanderer or muckraker, and otherwise its conviction was negated and it changed to the team of the empowering robot. Additionally, all convictions for all affected units were then capped at their influence.
- A **Muckraker** was a unit whose possible actions were to move to an adjacent tile or to “expose” a slanderer at most $\sqrt{12}$ tiles away. Upon exposing a slanderer, all the politicians on the team of the exposing muckraker obtained a “buff factor” for 50 rounds, which was multiplied to the empowering politician’s conviction before it was divided between units within the empower radius. The buff applied had originally been $1.01^{\text{influence of exposed slanderer}}$, but was quickly changed to $1.001^{\text{influence of exposed slanderer}}$ in an effort by **Teh Devs** to balance the game. Multiple buffs at the same time resulted in them combining multiplicatively.

Table 1: Robot types

Symbol	Name	Action Cooldown
	Enlightenment Center	2/passability
	Slanderer	2/passability
	Muckraker	1.5/passability
	Politician	1/passability

Each robot was only able to sense map tiles and robots within a certain sensor radius as determined by the type of robot, and alone could not discern where the map boundaries were relative to itself, unless it had travelled to the boundary to sense it. Lastly, each robot had a unique ID number which could be sensed by other nearby robots, and carried a “flag” set to a 24-bit integer of choice every round, which other nearby robots could sense. ECs had the additional bonus of being able to see the flag of any robot for which it had the ID number, and any robot could see the flag of an EC for which it had the ID number.

All robots were constrained to use a limited number of “bytecodes” per round, which measured the computation costs of each block of Java code, in order to keep computation costs low. If any robot exceeded its bytecode limit as determined by its robot type, it would halt its code for the round and take no action, and proceed the next round where its code left off, losing one round’s worth of action.

1.2 General Principles

Every year, the Battlecode game features a map with many robots which run their own code. **Teh Devs** generally try to design the rules of the game such that at least several strategic paradigms are available, most commonly the “rush” and the “turtle”, which commonly appear in many strategy games far beyond Battlecode. The concept of a “rush” is to quickly overwhelm the opponent as early as possible to obtain a winning position and end the game early before the main parts of the opponent’s plan come to fruition, and the concept of a “turtle” is to build a strong defense to defend many weaker “economic” units which reside within, to gain an advantage of resources over the opponent which is used to overwhelm them later in the game. From the rules of the Battlecode 2021 game, it could be quickly determined that conviction was a

major resource, the slanderer was therefore intended to be an “economic” unit, the “muckraker” was intended to be an “attacking” unit (as it could expose the “economic” slanderers), and the politician was intended to serve doubly as a “defending” unit (as it could empower to remove invading muckrakers) and an “attacking” unit (as it could convert ECs). I therefore believe that **Teh Devs** expected a typical “balanced” strategy to involve a group of slanderers and an EC to produce conviction income, defended by politicians to empower against and remove approaching muckrakers, combined with some muckrakers and politicians on the offense, trying to expose opponent slanderers and convert opponent ECs. As each robot runs its own independent copy of the code, the Battlecode game typically features some method of inter-robot communication to allow for coordination between robots as required by more complex strategies, which in this year, was allowed by the flags carried by the robots.

2 Theoretical Strategy

In this section, I explain the theoretical side of the game, to develop an understanding of how the rules of the game gave rise to the strategies used, and how and why they worked, without too much regard for their implementation, which is discussed later.

2.1 Economy

Anything which I would consider to be an important economic/strategic “resource” satisfies the following:

- If your team had none of it, you’ve probably lost,
- You could remove it from the opponent,
- It could be difficult to obtain if the opponent was trying to prevent you from doing so.

At first glance, one would likely assume that an important resource in Battlecode 2021 was total conviction, since **Teh Devs** clearly designed it to be a resource, as evident by their explanation of the rules of the game. Indeed, if your team had no conviction you probably didn’t have any units, you could remove conviction from the opponent by empowering politicians nearby, and conviction could be difficult to obtain if the opponent tried to convert your EC and expose your slanderers. However, this was not the only resource: another was the total number of units belonging to your team; you could prevent the opponent from building units by using your own to stand on tiles adjacent to opponent ECs, preventing them from building on adjacent tiles, and you could also use one politician to empower near multiple opponent units to convert or remove them—both tactics could be difficult for the opponent to stop. Assuming that a game lasted until the final round, the number of votes could also be an important resource; you could win more votes than the opponent by bidding with extremely large amounts of conviction, and in some situations this could also be difficult for the opponent to stop. In the interest of simplicity, I only consider the **difference** in total conviction, unit number, and number of votes between the two teams as the relevant currency when discussing trading. My robots often faced the choice of whether to sacrifice some amount of one currency to gain some amount of another, i.e. to perform a trade, and here it was crucial that the correct decision be made using an estimated exchange rate.

2.2 Slanderer-Free Trading

Certain actions could be taken to force a strict gain of resource advantage or an exchange of one currency for another, and others could force a trade of currencies. In this section, it is assumed that by default for simplicity, that all non-neutral ECs never built any slanderers, and always attempted to build a 1 conviction muckraker every time they were able to take an action, because this allowed a gain of 1 unit count at no cost, which could not be bad for the team. These cheap muckrakers could then travel to the opponent’s EC to stand on the adjacent tiles and prevent it from building units there, essentially “burying” it in muckrakers. Every turn that my own EC was able to build a unit while my opponent’s was buried resulted in me gaining an advantage of 1 unit count for free, and thus the opponent was forced to respond. The opponent might choose to also bury my own EC, resulting in a net exchange of no resources, and/or they might choose to

prevent me from burying their EC. This was possible by forming a membrane of units around their EC across which no muckrakers could pass. Although this seemed like a good option, it was doomed to fail because I could simply bury the opponent's entire membrane preventing units from escaping, such that the number of units which could be built by the opponent's EC before it ran out of building locations was upper bounded by the area within the membrane, resulting in the same gain of 1 unit count for free per action for my team. The only other way for my opponent to prevent me from burying their EC was for them to build politicians which empowered to remove my burying muckrakers from the map. If one of their politicians empowered to remove exactly 1 muckraker, my opponent gained no net unit count but incurred a conviction loss of at least the empower tax, thus giving me a guaranteed advantage, unless they also built at least one slanderer to reproduce the lost conviction, which is discussed in Section 2.4. The only alternative response by the opponent which did not guarantee a net loss of resources was for an opponent politician to empower to remove at least two muckrakers, resulting in a net unit count gain and net conviction loss for my opponent, i.e. a trade of about 1 unit count for about 12 conviction. With good micro as discussed in Section 5, I was able to move my muckrakers into positions which didn't often allow for such favorable exchange rates for my opponent, forcing them to empower to remove less than 2 muckrakers from the map in the average case, to trade off their ~ 12 conviction spent on their politician for $\ll 1$ unit count in the average case. Since the game allowed an EC to produce up to $1/2$ a muckraker per round (each muckraker may force a trade of $\gg 12$ conviction) and no more than ~ 7 influence per round to buy defensive politicians, a constant stream of burying muckrakers from my EC could force such expensive trades which drained the opponent EC of conviction faster than it passively gained in most cases, slowly bringing it to 0 conviction, when it could no longer produce defensive politicians and consequently got buried by my muckrakers.



Figure 3: My units in blue, completely surrounding the opponent's EC and cluster of muckrakers in red.

The conclusion of this analysis is that in a scenario which does not involve slanderers, it was always favorable to produce a constant stream of muckrakers which attempt to micro while burying an opponent EC. That said, my opponent should therefore choose to use the same strategy or one which would force at least as favorable an advantage (I have not found any such alternative strategy). Thus, the ideal game without consideration for slanderers or neutral ECs was one where both teams rushed with muckrakers to bury the opponent's EC using good micro, and both teams defended with politicians attempting to empower near as many muckrakers as possible, slowly draining both ECs to 0 conviction at which point both ECs would be buried at the same time² and the team with the better bidding algorithm would win by vote. If the opponent had worse micro, did not attempt to bury you, or attempted to modify the strategy by buying a significant number of votes at any point, they might not have been able to force as favorable trades as you could, and would have less conviction in their EC than you would have in yours. They would then run out before you do and you would successfully bury them first, preventing them from building any muckrakers required to bury you, at which point you could freely clear the map of opponent muckrakers by building and empowering politicians, and proceed to safely build slanderers for a gigantic income without risk of exposure to buy most of the remaining votes and win by vote. Therefore I believe that this strategy is close to the Nash Equilibrium strategy if barring the use of any slanderers.

Oftentimes, the game arrived at a point where all the neutral ECs will have been captured, and muckrakers of both teams were abundant throughout the entire map to expose any slanderers made. In such situations, the game should be played as I have described. If not all the neutral ECs have been captured, sending a politician to capture them resulted in a gain of $1/2$ a unit count per round in most cases, so it was extremely important to capture the neutral ECs as early as possible. Delaying by even 20 rounds resulted in a loss of

²Teams rarely ever played perfectly until this outcome, so further analysis tends to be futile.

a 10 unit count advantage, which could have been used to force a trade of $\gg 120$ conviction, which was very significant in the early game.

2.3 Slanderers

The graph relating a slanderer's conviction to the amount of conviction it produced for its parent EC is shown in Figure 4. Ignoring the conviction which was retained by the slanderer, one can observe that the conviction gain was locally maximized around a specific set of slanderer convictions, due to the way in which the rules for slanderer income produced many downward-sloping diagonal lines in the graph, and thus it only made sense to build slanderers with these specific convictions. Additionally, it can be observed that a slanderer with 949 conviction produced the maximum possible amount of 551 net conviction for its parent EC. Consequently, if an EC had a small amount of conviction, it could spend it on slanderers to multiply and exponentially grow its conviction, but if an EC had a large amount of conviction, spending 949 conviction on a slanderer for every action produced the maximum possible rate of conviction gain, to linearly grow its conviction, again ignoring any conviction which was retained by the slanderer. One may wish to ignore the conviction retained by slanderers if it is expected that they would all be exposed before they camouflaged into politicians. Otherwise upon camouflage, the retained conviction then became useful, as the resulting politician might use it to empower near other robots, most notably its parent EC, so that the EC could re-spend the conviction in the politician *as well as* the embezzle income to build even more large slanderers. In this case, one might desire to build slanderers past 949 conviction, as the total conviction of the politician upon camouflage and the embezzle income would be approximately double the initial conviction, allowing for the EC's conviction to undergo unlimited slow exponential growth.³ The building of slanderers often required a temporary sacrifice of unit count, as the slanderers took EC actions to build, and yet became sitting ducks without much use, hiding from opponent muckrakers until they camouflaged into politicians. Moreover, if a slanderer was exposed, the sacrifice of unit count became permanent, and this was often a very large risk which could easily lose the game.

2.4 Trading with Slanderers

Assume that the opponent attempted to rush with burying muckrakers, and I was forced to defend. In a slanderer-free world, I would be forced to constantly trade by building politicians in a manner guaranteed to drain my EC to 0 conviction. Now consider instead what would have happened if I also built slanderers which I protected from exposure by opponent muckrakers using encircling politicians which empowered near any approaching muckrakers, a defensive "turtle" strategy first spearheaded by team **babyducks**. The politician-muckraker trading would work in the same way as in the slanderer-free case described before, draining my EC of conviction faster than it could replenish. But now if the slanderers built produced an income which exceeded the conviction lost through defense minus what I would have gained by building bury-rushing muckrakers instead, then I would have a guaranteed conviction advantage, else the opposite is true and my opponent would have a guaranteed advantage, thus for this strategy to work, I must have spent a significant amount on slanderers, as defense is very conviction-expensive, especially in the early game. Additionally, since defenses typically traded net zero unit count if the opponent had good micro, a constant stream of rushing units could use up almost all the unit count production of my EC, stopping me from building slanderers, further constricting my conviction income. Considering that my opponent must have explored the map using their muckrakers in order to find my EC(s) and bury it(them), I do have a small amount of time to freely build a small slanderer economy, with the hope that it would have grown large enough to be able to sustain the defense costs once my opponent's muckrakers reached me. On maps with generally low passability and where the ECs were very close to each other, this was not possible as my opponent's muckrakers would find me before I developed the economy required to sustain a defense and the optimal strategy was to rush and bury like in the slanderer-free case. On higher passability maps with ECs far apart, it was very easy to build a stronger slanderer economy in the early game and sustain

³For some unknown reason, no discussion of this strategy ever took place of my knowledge among the many competitors despite their diligent study of the game. As far as I know, the only place this was ever used was in my code submission for the Battlecode Sprint 1 tournament. I did not have enough time to include and debug this strategy for the other tournaments, though it would have probably made drastic late game economic improvements.

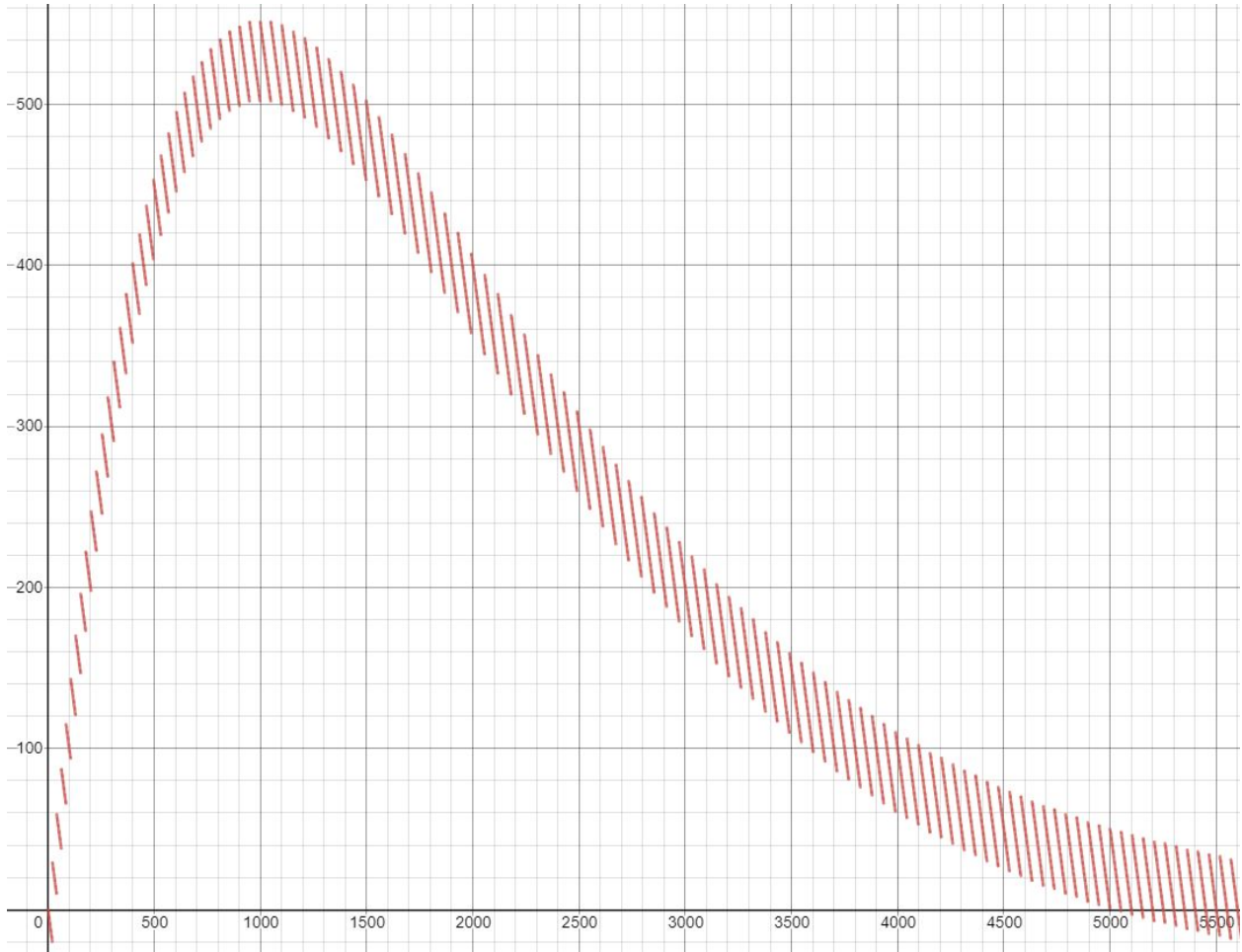


Figure 4: On the horizontal axis is the amount of conviction spent to build a slanderer, and on the vertical axis is the amount of conviction generated by the slanderer minus the amount spent to build it. Credits go to [EZ Money] ambysco for producing and sharing this graph.

a defense, as my opponent's muckrakers could not reach me in time to prevent me from doing so, giving me a conviction advantage and net zero unit count, and the optimal early game strategy was to build and protect a slanderer economy, i.e. to turtle. Consequently, my opponent should also try to turtle in this situation, and whichever player could spend more on slanderers while protecting them from exposure would be at a conviction advantage. In this case, early attempts to convert expensive neutral ECs might hinder the slanderer economy, as conviction would be spent on the politician used to convert it, which could have instead been spent on a slanderer. On the other hand, having an extra EC allowed me to constantly build burying muckrakers, each of which traded at my opponent's defenses for a value of > 12 conviction and forced them to replenish their defenses, costing both players roughly equal unit count, which I would have more of. If neutral ECs were cheap, it might be more favorable to convert them early to promptly drain 12 conviction from my opponent, but if they were expensive it might be worth converting them later to spend more on early slanderer economy since spending large percentages of my total EC conviction drastically delayed the exponential growth of my slanderer economy, possibly giving my opponent as much as $3\times$ more conviction than me. Thus it was extremely important to judge the distance and passability between my EC and my opponent's to determine whether to rush and bury or to turtle with a defended slanderer economy, and to judge the conviction of the neutral ECs to determine the best possible time to spend on converting them.

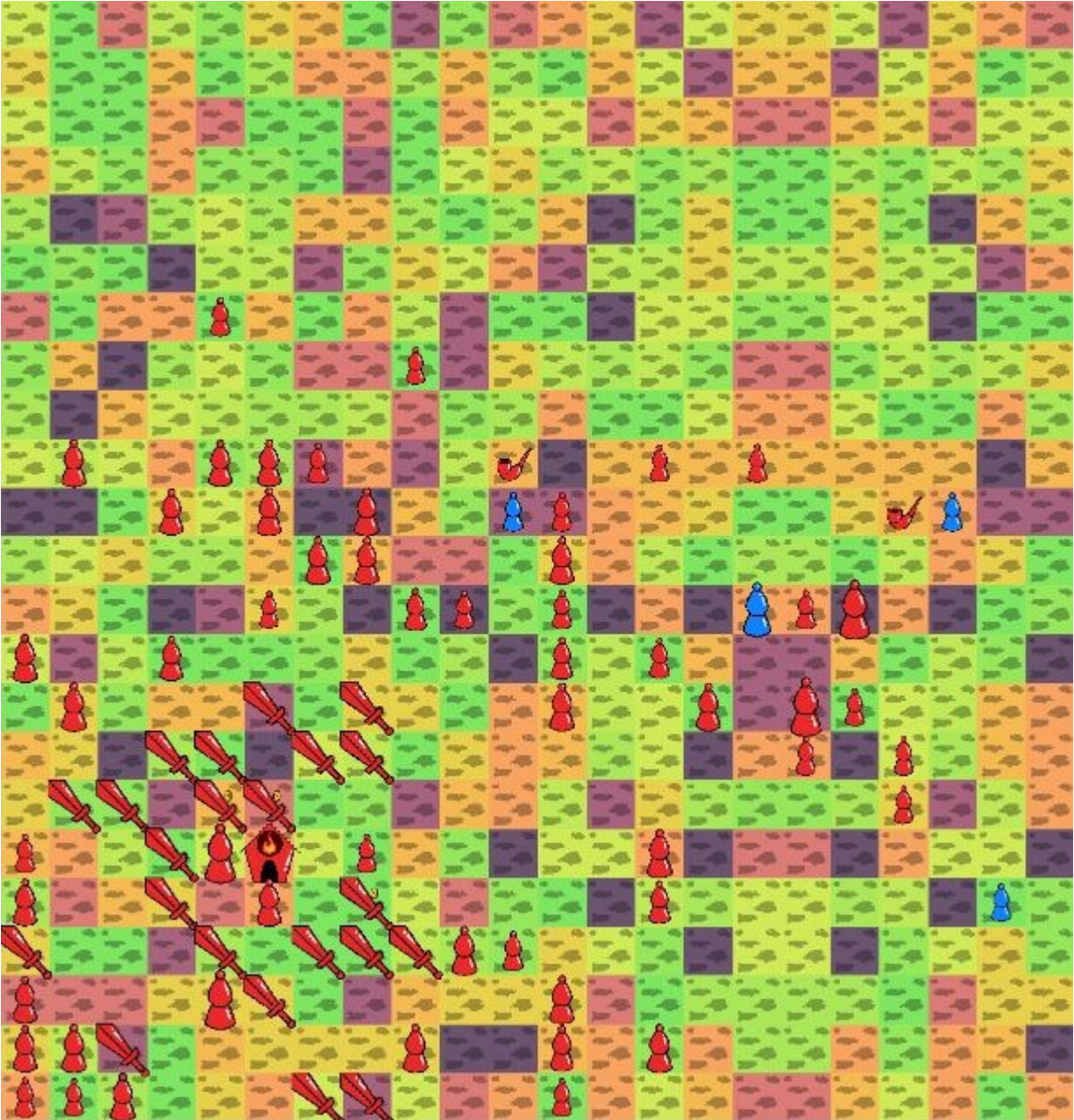


Figure 5: An early variant of the turtle strategy pioneered by team **babyducks** in red.

3 Algorithms

In this section I explain several algorithmic techniques which I used in my code, some of them fairly unique. For more details, the code I used for the competition is available at <https://github.com/iliao2345/Battlecode2021>, though I must warn that it is slightly messy due to the time-restrictive nature of Battlecode.

3.1 Reversed For Loop

It was known among many Battlecode competitors that the method of iteration with the lowest bytecode cost per iteration was the “reversed for loop”. This technique was only useful for Battlecode, and does not

likely have any application elsewhere.

```
1 for (int i=stop; --i>=0;) {  
2     // loop contents  
3 }
```

3.2 Distributed Bellman-Ford Algorithm

Oftentimes, it was useful for a large number of units to know how far they each were from a set of features on the map. Here, it is assumed that units were present throughout the areas of the map between the units and the features, in a dense enough quantity such that the visibility graph among the units was very well-connected. Given that each unit was capable of limited computational power and was able to communicate through its flag to nearby units, I used the following method to compute the desired shortest distances from each unit to the set of features. Each unit showed on its flag an approximate distance to the nearest feature. Each round, the approximate distance was incremented, and then for each nearby unit, the approximate distance was capped at each of the distances shown by the flags of the nearby unit plus the distance to that unit. If a unit sensed a feature at any time, it capped its approximate distance at the distance from the unit to the feature. Many different distance functions were appropriate depending on the usage case of this protocol. For pathfinding, it was often useful to use the euclidean distance divided by the passability of the tile which the unit was on, as this took into account that units travelled slower on low passability tiles, while still maintaining a relatively low bytecode usage. For many other applications though, I often used a distance of one for every sensible unit, as travel duration was not of significant concern.

3.3 Integer Cycler

Common Java utility functions were often assigned extremely high bytecode costs by Battlecode, and thus competitors often constructed their own custom data structures for replacement. In my case, I required a set of integers which could be iterated through, to which elements may be added and from which elements could be removed. I therefore created the “integer cycler”, which consisted of a set of objects arranged in a cycle, each with a reference to the next and previous object, and each storing one integer, like in a linked list, but circular. I could easily add an element or remove an element by manipulating the references in a local area of the data structure, and also track the size of the cycle by incrementing and decrementing a size variable every time an integer was added or removed.

3.4 Optimal Empower Radius

When a politician empowered, it had to choose the correct radius with which to empower to capture the set of units which allowed it to perform the best trade possible. In this section, I assume that an exchange rate between unit count and conviction was given, such that any net unit gain could be converted to net conviction gain, and only one currency had to be considered while weighing the favorability of various choices of empower radius. The first step was to count the number of robots within each possible empower radius under consideration. Only the radii of $1, \sqrt{2}, 2, \sqrt{5}, \sqrt{8}, 3$ needed to be considered, as all robots lay on lattice points and the maximum allowed empower radius was 3. I iterated through each sensed robot within 3 tiles of the empowering politician and counted the number which were exactly at each radius, then took a cumulative sum over the radii to cheaply count the robots within each radius. I then proceeded to compute the conviction quota given to an affected robot for every radius, remembering to take into account the empower tax, any buffs, and floor division effects. Then again, I iterated through each sensed robot within 3 tiles of the empowering politician, this time counting the amount of conviction and unit count each team would gain and lose for all of the radii. By default, the empowering politician lost one unit count and all of its conviction because it would be removed from the map. ECs were treated as though they consisted of a very large unit count (I used 50 for neutral ECs and 100 for other ECs) since they could be used to produce unit count for a long duration of the game. I treated currency gained/lost for the empowering politician’s opponent team as currency lost/gained for the empowering politician’s own team respectively, and finally converted from unit count to conviction using the given exchange rate to produce the final net effective conviction gain/loss for an empowerment at every possible radius. This information could then be used in

a myriad of ways. For example, if any of the empowerment radii would produce a net positive effective conviction gain, then it would be favorable for the politician to empower with that radius, as the outcome would be favorable according to the exchange rate. Otherwise, it would be favorable for the politician to not empower at all, because all resulting outcomes would be unfavorable according to the exchange rate. Using this algorithm to judge whether and how my politicians should empower, I implicitly ensured that:

- Large politicians did not waste conviction by empowering to affect much smaller units which only held small amounts of conviction before it was capped at their influence.
- Small politicians never attempted to convert a lone large politician, wasting on empower tax without converting, unless they had a significant buff from exposed slanderers.
- Politicians would often attempt to convert and/or remove multiple opponent units at once.
- Politicians occasionally deliberately “healed” friendly units whose conviction was lower than their influence.
- Politicians would almost always choose to convert an EC to their team whenever possible.
- Politicians tended to prefer to convert opponent politicians over removing opponent muckrakers.
- Upon my team receiving a sufficiently large buff from exposed slanderers, politicians would often convert two or more opponent politicians at the same time, bringing them to their maximum conviction, such that they would proceed to do the same. This began a chain reaction, which was often able to clear out the majority of the opponent’s local politicians in an extremely short period of time, while also producing many leftover politicians for my own team in the process.

3.5 Minimax Move Direction

Thinking one step ahead of an opponent politician who wanted to empower, I wanted my own units to move in such a manner to ensure that the opponent could never make a good trade by empowering. It turned out that it was within bytecode limits to compute all the net effective conviction gains for every possible empower radii of one opponent politician, given that one of my units in question moved in every possible direction (or stood still) first. This enabled my unit to move in a direction which allowed for the worst possible best (a min of a max, also known as “minimax”) empower radius for the opponent politician. In most cases, the opponent politician would have no good (positive net effective conviction) empower radii, and so the opponent politician could also choose not to empower, leaving me with many possible move directions which all resulted in zero minimaxxed net effective conviction gain. The computation involved running an optimal empower radius calculation for the opponent politician merely twice. The first time, I performed the calculation excluding my unit which I could move, such that the results were as though my unit was outside all the empower radii, and the second time, I performed the calculation including my unit as if it were at a distance of one from the empowering politician, such that the results were as though my unit was inside all the empower radii, all in all giving me the net effective conviction gain if my robot was inside or outside every possible radius. The next step was to compute the best possible net effective conviction gain for the empowering politician if my robot resided specifically at each radius. I took a cumulative max of the “inside” gains and a cumulative max of the “outside” gains in the opposite direction, and then took the max of consecutive pairs of “inside” cumulative max gains and “outside” cumulative max gains, because if my robot was at a certain radius, then the best net effective conviction gain was given by the max of the “inside” gains for all the radii that my robot was inside and the “outside” gains for all the radii that my robot was outside. My opponent could also choose to not empower, so I lower bounded the calculated values by zero. Finally, I computed the distance to the empowering robot after every possible move direction for my own robot, and assigned that move direction to the opponent’s best possible net effective conviction gain associated with that radius, concluding the “max” portion of the minimax calculation. Since Battlecode was a zero-sum game, I negated the calculated gains to produce the worst possible loss that the opponent was able to force on my team by empowering for every direction my unit could move. I could then choose to move in a direction for which the opponent politician could force at worst the minimum possible loss for

my team, i.e. to perform the “min” portion of the minimax calculation. By doing this, I implicitly ensured that:

- Units tried to avoid opponent politicians if approaching them would allow them to empower to remove and/or convert multiple units at once. Multiple removals and/or conversions is a tactic which most top teams like **California Roll (Chop Suey)** relied on very heavily to win.
- Small politicians tried to avoid larger opponent politicians which were less than about twice their size, to prevent the opponent from performing efficient conversions.
- Small units would stand next to large opponent politicians which attempted to empower to convert an EC, so that the conviction they empowered with was diluted and wasted up until the point that the opponent was no longer able to convert the EC.
- Units, especially politicians, tended to avoid opponent politicians if the opponent team had a large buff, hindering chain reactions in favor of the opponent.
- Large units attempted to dilute an opponent politician’s empower conviction if doing so would prevent a smaller friendly unit from being converted and/or removed.
- Bury-rushing units fled from defensive politicians as soon as they were built by the opponent EC to avoid efficient multiple conversions/removals, and returned to bury the EC immediately once these defensive politicians were gone.
- Small units did not waste time diluting an opponent politician’s empower conviction if the opponent’s own units congested and diluted themselves sufficiently in the first place, which was a common occurrence near an EC being buried.

A very similar calculation would be done when considering a friendly politician who wanted to empower; the intermediate step of negating the gains was simply removed as the empowering politician was on my own team. By doing this, I also implicitly ensured that:

- Units tried to get out of the way of friendly politicians which could empower soon, so to not unnecessarily dilute their empower conviction.
- Units with less conviction than influence sometimes tried to move towards friendly politicians which could potentially “heal” them.

3.6 Bury Sensing

When I buried an opponent EC, the EC was almost always a part of a cluster of opponent units on the same team as the EC, and the whole cluster was surrounded by my burying units. If my opponent’s units escaped to leave a void, the opponent EC could gain enough space to build more units, making my burying effort unsuccessful, so I could not allow units to escape from the cluster. Yet if too many units were used to bury the cluster, I was unable to perform trading using the unit count which was present in the burying units, so I would incur an effective unit count disadvantage. Thus, I always attempted to bury the opponent EC and the surrounding cluster using a one tile thick layer of burying units, freeing the remaining units to perform other functions. To do this, my burying units needed to determine whether the opponent EC had already been buried and could be ignored, and whether they were part of the one tile thick layer which had to stay in place while other units could be free. Ordinarily, I would perform a graph search beginning from my opponent’s EC to determine whether the cluster was completely enveloped in my burying units or was adjacent to at least one empty tile, but bytecode limitations severely limited the usage of such an algorithm. Therefore, my code performed a highly truncated version of a graph search in the form of a hardcoded triple nested loop, and assumed that the opponent EC was not buried if the third loop called for the graph search to continue further beyond. A unit determined whether it was part of the layer by determining whether the truncated graph search ever reached itself. Despite the fact that the graph search was hardcoded and truncated to ignore many possible use cases, it tended to function perfectly in the vast majority of cases.

3.7 Communication

Since communication through robot flags was limited to 24 bits per round for every robot, it was essential that information be fairly compressed and shared very sparingly between robots. To share information, I divided up the 24 bits of the flag into multi-bit segments, each representing an integer of data which ranged from zero to the maximum integer which the segments could allow. Often to selectively transmit more important information, I used the first few bits of the flag to indicate how the rest of the bits should have been interpreted.

The map always had coordinates which were offset from the origin by large integers, such that communicating the full coordinates of important features was not reasonably possible within the amount of bits given. Furthermore, no robot could easily tell the positions of the edges of the map without visiting them, and could not use them as a reference point to communicate offsets from. Luckily, **Teh Devs** explained in their lecture how transmitting the coordinates mod 128 was sufficient for any receiving robot to pinpoint a unique location, because the map was at most 64 tiles in either dimension. I happily used this idea to communicate locations between my robots, sometimes decreasing the resolution of the location data to save bits for other uses, if only approximate locations were required.

Certain communicated quantities such as amounts of conviction could become extremely large over the course of the game, yet would require precise resolution when they were small. For these quantities, I communicated the logarithm of the quantity (with some rescaling to make the relevant range of quantities match with the size of the segment of bits used) and took the exponent upon reading the flag to restore the value.

4 Strategy Implementation

To execute the strategies described in Section 2, my code assigned each unit a “role”, which described the functional portion of the strategy which the unit helped to execute, and the manner in which it did so. Each unit’s role was also indicated through the flag it showed, so that other units were aware of its intentions.

4.1 Explorers

Arguably the most important reason that units were useful was because they were able to explore the map and find features such as map edges, neutral ECs, and most importantly of course, my opponent. Units in bulk had to be able to provide rapidly expanding vision and coverage over the map, relaying information about map features back home to Enlightenment Centers through flags, so that time-sensitive decisions could be made as early as possible. As such, all of my units were explorers by default, unless they were specifically assigned to a different role. To perform exploration, I relied on a distributed Bellman-Ford Algorithm as described in Section 3.2, where the relevant targetted features were areas of the map where my units were largely absent, to allow each robot to determine how far it was from the nearest unexplored location. An explorer judged its area of the map to be unexplored iff a full 180 degree view of its circular sensor area was completely void of other explorers. If an explorer saw a map edge, it would pretend that the map edge was a mirror, and that it could sense another explorer (its reflection) on the other side, so that it would not judge areas off of the map to be unexplored and attempt to explore them. Each explorer also tracked its own “momentum vector” capped at a maximum length, which was its preferred direction of movement that it tried to move towards for every action. This momentum vector for any given explorer would be updated every round by adding a “net force vector”, which was produced by treating every other sensed explorer (including its own map edge reflections) as a point charge which it would repel from iff the sensed explorer was least as far from the nearest unexplored area as itself, pushing it towards the nearest unexplored location. The reason for moving according to a momentum vector rather than directly through the force vector was to make explorers preferably move in straight lines away from home base when alone, helping to prevent re-exploration of previously explored areas. Since all interactions between explorers were repulsive and none were attractive, I guaranteed that explorers would try to spread themselves as thinly as possible to cover the greatest possible area of the map.

One might wonder why I chose to compute the distance to the nearest unexplored location and repel selectively, rather than simply allowing all of the explorers to repel uniformly. Consider if my opponent

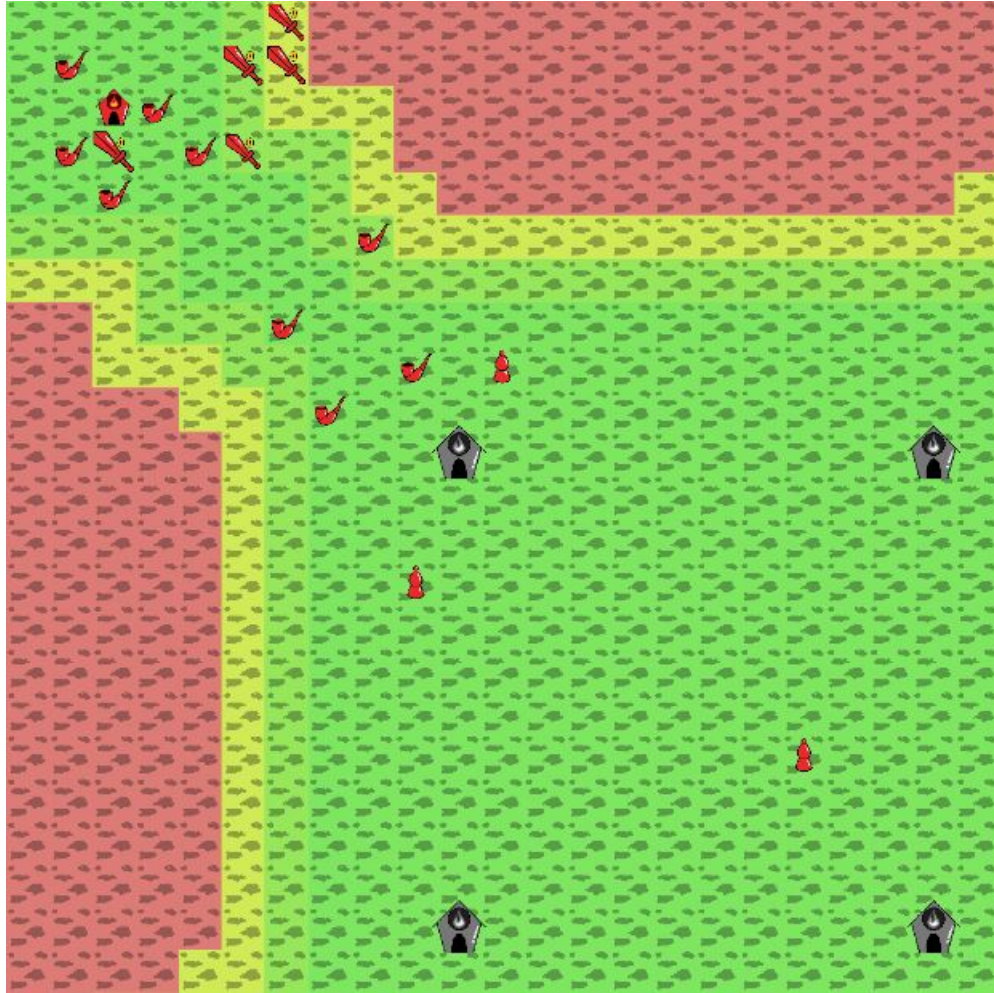


Figure 6: Exploration was absolutely essential for the collection of information regarding neutral ECs.

actively attempted to prevent me from exploring some area of the map. The only feasible and effective way to do this was to empower many politicians to remove or convert my many explorers which approached this area. As discussed in Section 2.4, this was highly a costly trade for my opponent, so it made sense that I would want to make these trades as rapidly as possible, as they were generally to my advantage, and to do so, all of my explorers should best disregard explorer density gradients and travel directly towards the nearest unexplored area which the opponent attempted to restrict, where my explorers were subsequently traded away by my opponent's empowerment. This produced much more rapid trading than if my explorers were to direct themselves towards the restricted area through explorer density gradients, as large numbers of units were required to produce significant explorer density gradients, holding up extra unit count which was not available for trading at a time when such trading was much to my advantage. An important scenario which arose often was when my opponent attempted to defend a group of slanderers by preventing me from exploring near them, and in this situation it was evidently crucial that my forceful exploration of their territory strained their defense effort as much as possible, as I required my muckrakers to find and expose their slanderers to halt or slow their exponentially growing economy as quickly as possible.

4.2 Slanderer Lattice

Since slanderers had to be kept safe at all times from exposure to muckrakers, it was beneficial for them not to wander and to remain in one place at all times, so that they would be less likely to encounter opponent

muckraker and be exposed. On the other hand, exploration was incredibly important, and slanderers could serve as extremely quick explorers in the opening of the game since they did not spawn with a cooldown like the other units. Ultimately after experimenting with both options, I decided that it would be best for slanderers to remain in their place, as the conviction which they produced often outweighed their exploratory capabilities. Although I could simply instruct the slanderers to never move, this would immediately cause congestion as the area nearby my ECs would quickly become filled with slanderers, preventing my ECs from spawning any more units. I therefore needed a way to “store” large numbers of slanderers in place, while still allowing units to travel where they needed to go, and thus I chose to create a “slanderer lattice”. This featured a checkerboard pattern of “storage tiles” alternating with “outgoing tiles”; slanderers would stand still and wait on storage tiles if possible, and otherwise they would travel away from the EC somewhat randomly amongst the outgoing tiles until they reached a storage tile on which they could stand and wait. Since the outgoing tiles were sufficiently well-connected (because units could move diagonally), congestion near the EC was generally avoided as units could move out of the way of each other sufficiently rapidly. Other versions of slanderer lattice which I developed also featured “ingoing tiles” to facilitate unit movement directed towards and away from the central EC at the same time, while still allowing for storage locations, but this lattice structure required more space and therefore more map control in order to not congest, especially on maps with low passability or map edges near my EC, so I did not use those versions of lattice for my final code submission.

4.3 Buriers

As described in Section 2.2, a crucial strategy in the slanderer-free game was to completely envelop or “bury” the opponent’s EC in units, such that it could not spawn units on any tiles. To do this, I implemented a role called a “burier”, whose job was to travel to and stand on any tile adjacent to the cluster of opponent units surrounding their EC. Using a bury sensing algorithm as described in Section 3.6, I ensured that in the successfully buried state, each burier knew that if it was part of the stationary layer of units surrounding the opponent cluster, it would stay still, and otherwise it would change roles to become an explorer. Furthermore, if any explorer saw a non-friendly EC with no buriers burying it or one of those buriers signalled on its flag that the EC has not yet been completely buried, it too would change roles to become a burier, to help contribute to the burying effort which the opponent may be actively attempting to resist. Lastly, buriers always judged neutral ECs as completely buried, so that they would not take up unit count by signalling for other units to come help to bury the neutral EC. In effect, this always left one burier near the neutral EC who would not leave to explore because it saw no other buriers burying the EC, thinking the EC had gone unnoticed. Although unintended, this turned out to be greatly beneficial because this burier would often be able to micro to greatly dilute the empower conviction of any opponent politician who attempted to convert the neutral EC, causing the conversion to fail, after which another single burier would come to replace the first one, and could do the same, massively delaying the opponent by causing the opponent to repeatedly fail to convert the neutral EC.

4.4 Targetters

It was often the case that one of my ECs had vastly greater conviction than an opponent EC, and in this case the best possible play was for my EC to send one or more politicians to opponent EC to empower and convert it, spending enough conviction on my politician(s) to outstrip all of the conviction stored in the opponent EC. Thus, I implemented a role called the “targetter” to represent the politicians sent for conversion, which would specifically target and travel towards the coordinates of the opponent EC in question, rather than (relatively) aimlessly exploring in order to find it. Buriers would signal on their flags the approximate coordinates of the targetted EC as well as the conviction required to capture it, and my ECs would read the flags to decide whether it had enough conviction to send a targetter to capture each non-friendly EC that it was aware of. My ECs would communicate the role and target location during the initial cooldown period of any targetter(s) which it built. Targetters also showed on their flags a “get out of the way” bit, which if set to 1 when the targetter was experiencing congestion, would instruct friendly units and especially buriers to move away from the targetter to give it some space to reach its target. The amount of conviction required to capture an opponent EC was estimated by the buriers to be thrice the conviction of the EC added to

any conviction it would gain by the time the targetter reached the EC, as estimated by the distance to the friendly EC. This was under the assumption that the targetter’s empower conviction would be diluted by no more than two other units when it attempted to convert the EC, due to congestion by friendly buriers as well as micro by the opponent. Each of my ECs would send at most 1 targetter at a time for each target EC, and would send another if the first one was unsuccessful.

4.5 Guards

To defend slanderers, I needed to be able to remove any opponent muckrakers which came too close to my slanderer lattices and threatened to expose them, but at the same time, I had to spend the least amount of conviction possible on defense, since defense was incredibly conviction-expensive as I was constantly offering favorable trades for the opponent. Taking inspiration from the turtle strategy of **babyducks**, I opted to produce sparse layers of “guard” politicians to surround my EC and the slanderers which it produced, which would empower to remove any nearby muckrakers. These guards would use a distributed Bellman-Ford algorithm as described in Section 3.2 to determine the number of layers of guards separating themselves from the protected EC and slanderers. They would compute their distance to the previous layer, repelling from it if they got too close and attracting if they got too far, and would always repel from other guards in the same layer, ignoring guards of the next layer and beyond. This allowed the guards to spread themselves into sparse yet uniformly thick defensive layers surrounding the protected EC and slanderers, covering as much of the perimeter as possible with the least possible number of guards, each of them taking up valuable conviction to build. Later, I noticed that this guard strategy could be even further improved. Instead of maintaining a layer of guards which held up a large amount of conviction that could have been spent on slanderers, I used my explorers’ flags to communicate to the EC the position and size of every approaching muckraker, and built a specific guard specialized to travel to and remove each and every one of them, much like how I used targetters to convert ECs. Any guard which “missed” its target for one reason or another would revert to the sparse layer defense strategy, and another guard could be built and sent to account for the missed muckraker if necessary. This way, every opponent muckraker would be accounted for, and yet almost no more conviction than absolutely necessary would be spent to defend my slanderers, saving more conviction to be spent elsewhere.

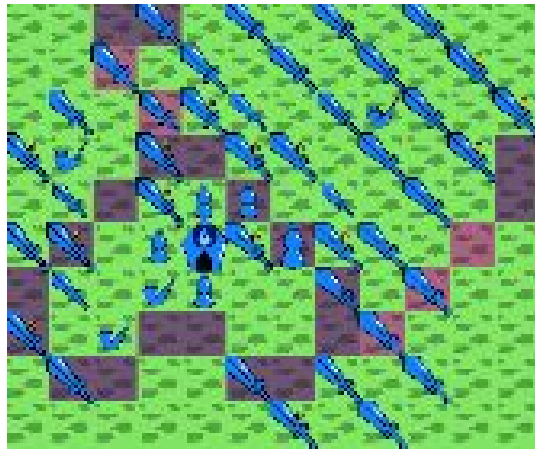


Figure 7: A slanderer lattice, with newly built guards in the center ready to traverse through the lattice to reach their targets.

5 Micro/Macro Pipeline

5.1 Units

My units typically made decisions in two steps: the “micro” step and the “macro” step. The micro step was dedicated to choices which would incur an immediate effect in one round, while the macro step handled all other strategies. In the micro step, I found the largest nearby politician regardless of team, and then used the minimaxer algorithm described in Section 3.5 to compute the minimaxxed effective net conviction gains for my team with respect to the politician’s choice of empowerment for every direction that my unit could move in. If my unit was a politician, I also used the optimal empower radius algorithm described in Section 3.4 to compute the gains for if my politician chose to empower. I then chose to perform the action which maximized the calculated gains.⁴ If there was a tie between multiple move directions (which occurred in most cases), then all of the best move directions were fed to the macro step, which chose one of these move directions. This ensured that my units always prioritized actions which urgently, clearly, and immediately produced gains or prevent opponent gains, and only afterwards did they consider taking actions to encourage more unpredictable longer-term gains.

During the macro step, only the set of best possible move directions which tied in the micro step was considered. The role which the unit played determined where the unit wanted like to travel and for what reason. This information was then passed to a short-distance greedy pathfinding algorithm which decided which of the available moves led the unit to its destination in the fastest manner.

5.2 Enlightenment Centers

My enlightenment centers operated by choosing what to build on a list of options sorted by decreasing priority: first guards, then targetters, then “unburying politicians”, then slanderers, then explorers. For the first three options, the EC built as close as possible to the intended target, so that less distance was traversed to reach the target, which on high-passability maps could make as much as a 30 round difference in traversal time. The “unburying politician” was simply a small politician intended to empower to remove multiple of the opponent’s burying units. This was only done if enough units were present to slow my EC’s build rate below its calculated default rate, which varied by map. Cheap explorers were always built as close as possible to the largest nearby opponent politician, to increase the chances that the newly built explorer would dilute the politician’s empower conviction in case it attempted to convert my EC. The reason that targetters were deemed to be more important than unburying politicians was because oftentimes while an opponent attempted to bury my EC, it was possible for my targetter to sneak out and convert the opponent’s EC, which replaced the advantage lost through my own EC being buried, completely defeating the purpose of my opponent’s burying effort. This idea was taken with inspiration from the team **Bytecode Mafia**, which sometimes thwarted my burying rushes by sneaking large politicians out like such. All units except for explorers were also barred from being built if building them would result in nearby opponent politicians having enough total conviction to convert my EC.

Since ECs were the only robot that could build units for all the various roles, I let my ECs manage the large scale, long-term economic and strategical decisions of the game. By using explorer flags to measure the approximate distance to the opponent, my EC could judge whether an early burying rush would be feasible, in which case it needed to capture neutral ECs as early as possible. To save as much economic currency as possible, I only chose to build guards if there was a large enough income from slanderers worth defending, and only chose to build slanderers if opponent muckrakers were far enough away that defense would not likely be too taxing. This allowed my ECs to switch between the rush and turtle strategies multiple times in during a heated match, whenever it deemed that doing so would be approximately economically favorable. One caveat of this was that when I finished burying the ECs of a team like **Blue Dragon**, they would leave many muckrakers around preventing my ECs from switching to the turtle strategy, but would also leave a large amount of conviction in their ECs to prevent me from capturing them. As they had a better voting strategy than me and our incomes were proportional, they could buy the votes more efficiently than me and

⁴I had to include a special case here to disallow politicians to empower and waste their conviction purely to prevent being converted by an opponent politician. If I hadn’t done this, opponent politicians of the right size could have simply walked among my own politicians, repeatedly causing them all to empower and waste conviction and unit count, at no cost to the opponent.

win by vote despite my complete control over all the other currencies of the game. To handle this, I had to include a special case to override and force a turtle strategy iff the local explorers signalled the nearest unexplored location to be very far away, i.e. the whole map was not only explored, but somewhat densely filled with explorers too, which could only happen if I had full control as the explorers try to spread. This would allow me to buy most of the votes remaining using a growing slanderer economy and win by vote. The team **Nikola** was still able to overcome this special case, by ensuring that the muckrakers left behind after burying were large enough to be too expensive for me to remove, so that my ECs could never begin building slanderers whose income was needed to buy the votes.

On a sidenote, since all my units required an exchange rate estimate to properly operate their minimaxer algorithms, I used my ECs to calculate an estimated exchange rate before sending it through flags to all the units for their own use.⁵ While in hindsight a dynamically tuned exchange rate would lead to much more efficient and exploitative trading, I simply used a linear function of my ECs' total income per round, which was sufficient for fairly strong performance because most other teams did not make use of exchange rates anyways.

6 Timeline and the Meta

The “meta” is a common term when discussing popular games which refers to the typical strategy played. The meta tends to vary throughout classes of player strengths, and evolves as players learn over time. This is definitely the case for Battlecode 2021. In this section, I outline major events in chronological order such as developments in the meta, developments in my own code, and significant changes to the game rules.

6.1 Week 1

The first week of Battlecode is always the week in which teams learn the details of the game and how they enable certain strategies. During this time, the game typically undergoes many balancing changes and bug fixes as players spot loopholes in the game rules. This year, it was noticed that the exposure of slanderers often led to exponentially large empower buffs already capable of overflowing integers within the first tenth of the duration of the game. **Teh Devs** patched this within the first few days by decreasing the base of the exponential buff tenfold, but it soon became clear that this was not enough. Right before the first sprint tournament, they patched this issue again by capping all buffs and convictions to a maximum value, finally stopping competitors from being able to break the game.

Due to the game being somewhat unplayable for the duration of the first week, I restricted myself to the development of highly general algorithms and techniques which could be used later on in the competition, so that the first week's worth of time would not be wasted. Things I developed included the distributed Bellman-Ford algorithm (Section 3.2), basic uniform-repulsion exploration code (Section 4.1), various versions of slanderer lattice (Section 4.2), burier code (Sections 4.3 and 3.6), and communication code (Section 3.7). I also developed some techniques which were not included in my final submission:

- **Relay Chain:** Instead of repelling from explorers at least as far from the target as itself, an explorer would instead attract to explorers closer to the target than itself. This caused explorers to form long chains which led them all towards one global target through the fastest paths possible, easily overwhelming whatever was being targetted through sheer numbers of explorers. Explorers which did not see any other explorer to relay the target information to would travel back to their home ECs in search of other explorers to notify of the target, so that every explorer would know where to go. Additional features included an imitation of “tension” which would allow the chains to straighten over time, shortening distance travelled as much as possible. I later removed the relay chain behavior because it would cause my explorers to have trouble splitting their focus to target multiple locations at once.
- **Membrane:** This was an early variant of my guard role, which consisted of many low-cost units that formed an impassable barrier around guarded units by intentional congestion, rather than by

⁵The resulting mysterious flag behavior tended to garner the curious attention of other strong teams such as **3 Musketeers**. I was eager to explain what my flags represented and how they worked.



Figure 8: A spiral shaped relay chain leading away from my EC.

empowerment. The congestion was carefully controlled such that the membrane was permeable to only friendly units and could expand and contract whenever the membrane contents did the same.

During this first week, I developed a strategy involving a burying muckraker rush which switched to a turtle with a slanderer lattice surrounded by a membrane at the 1000th round, implicitly assuming that the opponent had been buried by then. My slanderer lattice featured ingoing and outgoing tiles as well as storage tiles, and used a slow doubling growth strategy as explained in Section 2.3, which enabled my EC and each slanderer to reach over 10^6 conviction by the end of the game, at which point I would build many gigantic politicians of a special “exterminator” role to convert and remove any remaining opponent robots to win. I had not yet learned the economics required for the optimal empower radius or minimaxer algorithms yet, so my micro was much worse than in my final submission, but still strong enough for me to successfully bury most players at the time. I submitted the code for this strategy a day or two before the Sprint 1 tournament to allow time for debugging by testing against other teams through Battlecode scrimmages.

6.2 Sprint 1

The Sprint 1 tournament occurred after the first week of Battlecode with the purpose of allowing competitors to see and test the early meta of the game, so that they could more easily make informed strategic decisions for the future. The tournament format was single elimination, meaning any team which lost once would be eliminated immediately. At the time, my team had risen to the 13th rank on the scrimmage server, whose purpose was to automatically rank teams through the Elo system by running games, and thus I received the 13th seed for the tournament. Although I did win my first matchup 3-0, I lost my second 1-2, because my ECs had built so many explorers that the whole map would become congested and my exterminators could not reach my opponent.

It turned out that many teams that did well, including the Sprint 1 winner **Super Cow Powers**, tended to build slanderers and large politicians in attempt to outstrip the opponent EC’s conviction and convert it, and this formed the early meta of the game.

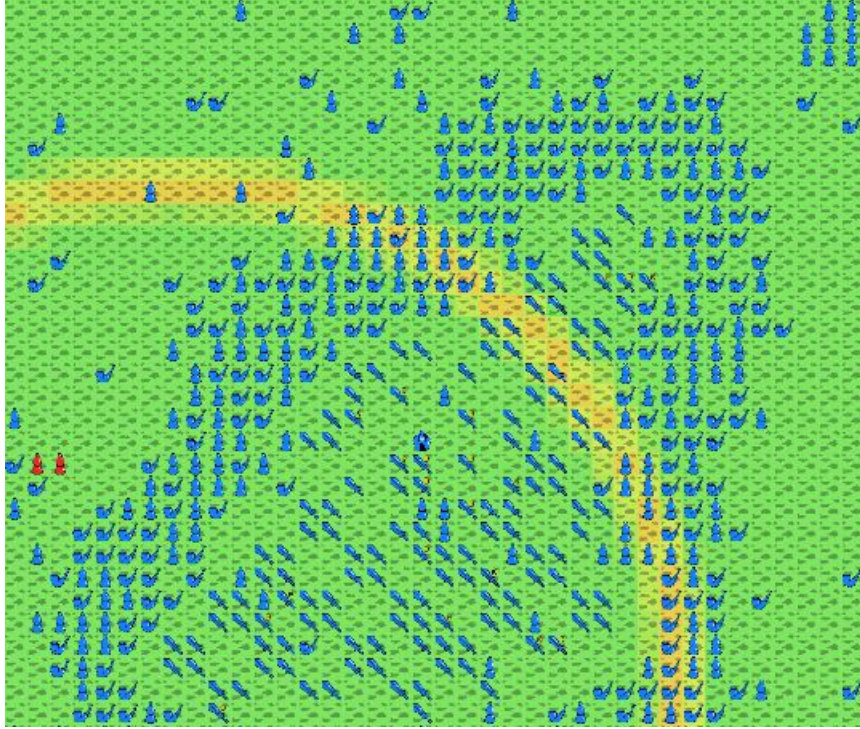


Figure 9: A large slanderer lattice protected by a membrane.

6.3 Week 2

Right after the submission deadline for the Sprint 1 tournament, I began developing a completely new set of code based on new principles, only ever referring back to my Sprint 1 code to use as an opponent while testing, because of an impending rule change by **Teh Devs** which halved the number of rounds in the game. I believed that this would make my strategy non-viable since the slow exponential growth technique which I required would not produce enough conviction in time for me to be able to end the game by extermination. My refusal to modify my Sprint 1 code during my development of my new code led to me losing every game as a result of the rule change, and my team plummeted far down the scrimmage rankings to nearly the 150th rank, though I knew I would make it back up once I finished and submitted my next strategy. Motivated by the abundance of unprotected slanderers that I saw many of my opponents building, I believed that my optimal strategy should be primarily to exploit such blunders, using the following technique:

- **Self-empower:** Upon receiving an empower buff as a result of exposing my opponent's slanderers, my ECs would spend all of their conviction on politicians, which would then immediately empower, multiplying my ECs' conviction by the empower buff. If done at maximum speed, this could be repeated five times before the empower buff expired. Oftentimes, I was able to expose more than enough slanderers over time to raise my ECs' conviction to the game's conviction cap, if not far above the opponent's own total conviction, which gave me a sure victory as I spent my plentiful conviction on exterminators.

At this point, controversy had been brewing amongst the competitors about whether this technique was an intended feature from **Teh Devs**, and whether it should be removed from the game and how so. In my eyes though, this was an issue to be dealt with later once **Teh Devs** had provided some clarification; the current rules of the game made the risk of opponent self-empowerment too great for the use of any slanderers, and I thought that no player should ever build slanderers as a result. Furthermore after having thought about some discussion I saw on the Battlecode Discord server mentioning that unit count was an essential resource, I fleshed out the economic theory of the game as described in Section 2.2, culminating with my

development of all the roles and the optimal empower radius and minimaxer algorithms in Sections 3.4 and 3.5, drastically improving my micro to the point where I would easily beat my old bot’s micro every time. These developments concluded with my choice to perform my signature muckraker rush **every single time**, and to punish any opponents who built slanderers through self-empowerment.

Meanwhile, the top team **babyducks** had introduced a radical new strategy to the game: the slanderer turtle, featuring many slanderers encircled by politicians which empowered to protect from exposure by opponent muckrakers. Its success quickly inspired many other teams to replicate this strategy, and by the end of week 2, the majority of top teams used some variant of it, and turtles became the meta. Some turtles would send gigantic muckrakers (known by competitors as “buff mucks” or “buffrakers”) to disorient the defenses of other turtles. The team **California Roll (Chop Suey)** even sent muckrakers around the edges of the map to “muck flank” other turtles which featured directionally biased defenses, an idea which won the surprise Adaptive Strategy Award. Seemingly, my muckraker rush would be completely at odds with the dominant popular strategies of the game, and one would likely prevail, though I wasn’t sure which it would be.

Again, several days before the Sprint 2 tournament, I uploaded my strategy for testing and debugging until the deadline. However since I was severely underranked this time, I had to test purely by requesting scrimmages against high-ranked teams with the autoaccept scrimmages option on,⁶ as my new strategy demolished the locally ranked competition. By the time all scrimmages were stopped for the tournament to run, I still had not been able to remotely find the approximate ranking of my new strategy through requested scrimmages.

6.4 Sprint 2

In a similar manner to the Sprint 1 tournament, the Sprint 2 tournament was single elimination, and occurred after the second week of Battlecode with the purpose of allowing competitors to see how the meta of the game had developed, so that they could more easily make informed strategic decisions for the future. Though I entered the tournament as the 132nd seed, I was flying blind as I had no clue what my true ranking was. While I won my first matchup 3-0 against a 100+ seeded team, I was shocked to find that my second opponent was the mighty team **Super Cow Powers**, winner of Sprint 1, which then went on to clear the entire bracket to become the winner of Sprint 2 as well. Though I had been immediately defeated as a result of my unpredictable ranking, I was also pleasantly surprised to find that I had won my first game due to self-empowerment and lost the next two, losing the best of three. It also turned out that my second game was lost largely by my EC’s failure to recognize the opportunity to self-empower, as it saw a few units which could potentially dilute the empowerment, but I hadn’t programmed it to recognize that these units could not dilute it enough. Clearly, I still had much work to do, but my close match against **Super Cow Powers** really solidified my confidence in my muckraker rush, which I continued to use for the rest of Battlecode 2021.

As expected from the meta developments during week 2, most of the top teams in Sprint 2 implemented a slanderer turtle, with the occasional buff muck and muck flank thrown in, with the notable exception of the team **java :ghosthug:** which also performed a muckraker burying rush. The meta remained this way throughout the rest of Battlecode.

6.5 Week 3

As my rating began its long climb from the depths, discourse of the self-empower technique became more heated, with many players pressuring **Teh Devs** to change the rules of the game, since self-empowerment caused games involving turtles to swing very easily, while other players (including myself) expressed aversion to such a drastic rule change this late into the game. After several days of careful deliberation, **Teh Devs** decided to massively reduce the effect of the exposure buff through multiple changes, most notably by disabling buffs applied to friendly ECs, rendering self-empowerment useless. At the time, I had finally risen to the 12th rank, when my primary weapon against all the teams near me had been obsolesced, and I fully expected my rank to plummet back down again, as this was the second time that a rule change would force me to completely rethink my strategy. Instead, I was astonished to see my ranking continue

⁶Such teams were sometimes difficult to find; many top teams turned the option off to avoid being flooded with scrimmages.

to climb, eventually peaking at 4th at one point. Seeking an explanation, I learned that most top turtles operated under the assumption that their turtle was worth protecting, and thus attempted to defend until they had enough defense built up to support slanderers, consequently using up all their unit count on defense against my burying rushes regardless of whether it would work for them or not. I then drained the opponent of conviction since defense was expensive, rather than by boosting my own conviction through self-empowerment, leading to the same winning outcome as before. This change in my understanding of the game compounded with the observation that I tended to do better on smaller open maps then grew into the economic theory described in Section 2.4. While I had learned that my pure muckraker rush still worked extremely well on small maps, some teams began to learn not to waste conviction on defenses while facing my muckraker rush, with team **Blue Dragon** paying the saved conviction on votes, forcing me to abandon my “exterminator” role, to turtle after burying to gain a conviction advantage, and to drive vote prices high enough to win by vote myself, and team **BattlePath** paying the saved conviction on large politicians to convert the remainder of my ECs, to which the economic theory suggested my rush could have no winning response at all. Other teams like **monky** began to fortify and improve their politician defenses, to the point that they could defeat my rush every time on medium to large size maps and any maps with lots of high-passability tiles. Strained, I even tried instructing my slanderers to perform exploration as well, trying to squeeze every bit of unit count out of early converted ECs that I could. Clearly I could not muckraker rush every single time if I was to continue as a top team for long, but with the big US Qualifying Tournament fast approaching, any major change to my strategy would be untested, and too risky to implement in time, and so I submitted my last pure muckraker rush strategy for the Qualifiers.

6.6 US Qualifiers

The purpose of the US Qualifying Tournament was to choose 12 US-based teams to compete in the Final Tournament. It used a best of 5 double elimination format, meaning any team which lost once would be moved from the winners bracket to the losers bracket, and any team which lost in the losers bracket would be eliminated. I was seeded 8th, and had to either win my first 3 matches or only lose once out of my first 5-6 matches (depending on when I lost my first match) in order to qualify for the Finals, and to my horror, the maps for this tournament leaned far towards the larger side. While I won my first two 5-0 and 4-1, my third opponent was the 9th seeded **3 Musketeers**, who I lost to 4-1. Luckily, I managed to win my next two matches 5-0 and 3-2 in the losers bracket, qualifying me for the Final Tournament. My final 3-2 victory against the formidable **Dis Team** was extremely close, with **Dis Team**’s turtle pulling ahead of my rush with a 24× influence lead over me on one of the large maps, but my strong micro slowly but successfully trading it all away at their defenses to produce one of my three wins; I was at their mercy and it could have very easily gone the other way.

6.7 Final Half-Week

During this final period, qualifying teams (US and international) finished developing their code for the final tournament. As I knew the other teams would be hardening their defenses during this time, I took a big risk and quickly coded up a brand new turtle strategy with a guard targetting system as described in Section 4.5, so that I’d at least have a chance on large low-passability maps. To retain all the work I had done on my rush, I let my ECs dynamically transition between rushing and turtling depending on context, betting that I’d be nimble enough to still overwhelm the opponent with my rush on small maps, and yet turtle harder than my opponent on larger maps. Luckily, this turned out to be successful as my new dynamic strategy stood firmly against several turtling teams which had dominated my pure rushing strategy in the past.

6.8 Final Tournament

The Final Tournament took place among the top 12 US teams and the top 4 international teams, and was double elimination format, featuring mostly large maps. Almost all of the maps featured ECs placed on low-passability tiles, decreasing their default build rate, and many teams were blindsided by this as every map that had been previously used only included ECs on maximum-passability tiles;⁷ teams did not know

⁷with the exception of maptestsmall

how their code would handle this, making the results of the tournament more unpredictable. I recieved the 11th seed for the tournament, as barely any ranked matches were set up to stablize the rankings of any strategies which had been changed. In chronological order, I played against:

- 6th seeded **babyducks**, where I won 3-2,
- 3rd seeded **Kryptonite**, where I lost 2-3,
- 9th seeded **confused**, where I won 4-1,
- 7th seeded **monkey**, where I lost 2-3.

finally putting me in seventh place this year, tied with team **Nikola**. After being placed in the losers bracket, **babyducks** continued on an incredible 8-match win streak, ultimately defeating team **Producing Perfection**, the champion of the winners bracket—twice in a row—to win the final tournament.

7 Special Acknowledgements

In addition to all the previously mentioned teams, I would like to thank teams **monkey** and **3 Musketeers** for the constant challenge of their strong defenses that they allowed me to freely play against (by keeping the auto-accept scrimmages on). I tested my strategies against them very often, and they were instrumental in the development of my code.

I would like to congratulate team **babyducks** for winning the final tournament, team **Super Cow Powers** for winning both sprint tournaments, and additionally teams **Producing Perfection** and **Malott Fat Cats**, since all four of them held the highest elo score on the Battlecode 2021 scrimmage rankings for long periods of time.

A special thanks to [**California Roll (Chop Suey)**] **stonet2000** for developing a more detailed Battlecode 2021 game visualizer, which in addition to the already shown, displayed the size of robots and an income graph to help competitors more easily view this essential game information.

Thanks to [**Soju**] **cyberwind** for creating and sharing a graph of squared euclidean distances of lattice points, which I referred to very often while developing my code.

And finally, a big thanks to **Teh Devs** for all the time and hard work they put into developing and maintaining such a great game and competition every year, and to **the sponsors** for supporting and making it all possible.