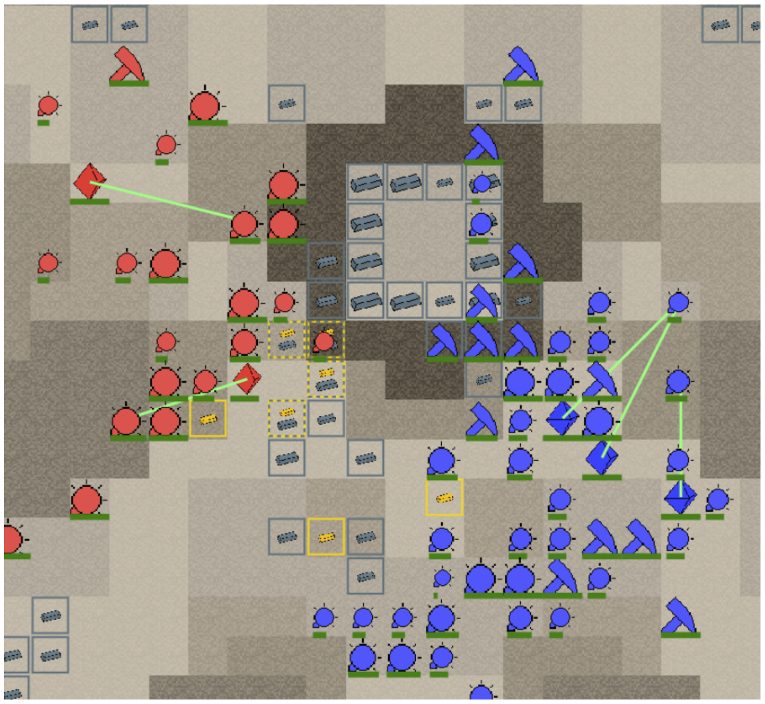
5 Musketeers - Battlecode 2022 Strategy Guide

David Lyons

February 2022

1 Introduction



1.1 Battlecode Introduction

Battlecode is an AI competition run every year throughout the month of January. At the beginning of the month, MIT releases a new 2 player game, and teams code up a bot that plays said game. There are two regular tournaments, a qualifier where the top 16 are chosen, and a final where the top 16 battle it out for cash prizes.

1.2 Team Introduction

Winston Cheung, Maxwell Jones, Bharath Sreenivas, and I are computer science students in our Junior year at Carnegie Mellon University. Last year, we were the 3 Musketeers even though there were four of us, so this year we made sure to get the number right. This was our second year doing Battlecode, so we were able to get up to speed more quickly, and things that took us a while last year were more obvious to us this year. We also got to reuse some old code, such as pathfinding.

This year added the additional challenge of school starting partway through the competition, and with rule changes intended to balance the game added the week we were in school and a snowstorm postponing finals, it was a much more intense ending than last year. We ended up doing as well or better in every tournament this year, finishing in semis in the second sprint tournament and 7th in the final tournament as the highlights. This strategy guide offers our perspective on how we adapted to this game, which brought many interesting challenges that weren't present last year. If you want to take a look at our code, it is linked here.

2 Game Overview

Battlecode 2022 was about a post-apocalyptic landscape where robots battle it out for control of the terrain. But to make the battle more complicated, there were mutations. Buildings could move or be upgraded. Random anomalies could change the map or kill some units. The resources in the game were gold and lead, which teams used to build robots that could fight the opponent. There were two ways to win:

- Destroy all of the opposing team's home bases
- After 2000 rounds, a tiebreaker goes to the team with more bases, then more gold, and finally more lead

The rectangular map could be as small as 20x20 or as large as 60x60. Each square had a predetermined amount of rubble, which controlled how long it took robots to move through it or perform actions while on it. There were seven different units this year:

- **Soldiers:** These are exactly what they sound like. They are your main fighters. They go into battle and can attack any other unit. They have relatively small health and damage amounts, but they are half the cost of a watchtower.
- **Miners:** Throughout the map, there are gold and lead deposits. When units die, their lead gets added to or creates new deposits. Miners allow you to reap the rewards of these graveyards and add to your team's wealth. They can't attack, but they are cheaper than soldiers.
- **Builders:** The cheapest of the robots, builders have two main purposes. First, they can heal buildings, such as your home base. Second, they can build new buildings, besides home bases. This leads to a variety of different strategies.
- **Sages:** Sages are, by far, the most powerful robots. They have longer vision and attack ranges, and although slower, they deal much more damage. They also have a variety of different attacks. Their normal attack is fine, but they also have a charge that knocks 22% off all nearby enemies and a fury that knocks 10% off all nearby buildings. However, these sages are not easy to acquire. They cost gold instead of lead, which doesn't exist naturally on the map.

- **Archons:** These are the home bases. They produce robots, namely builders, soldiers, and miners. They can be upgraded to have more health, and they can be healed, but if their health reaches 0, they are destroyed. When you lose all of these, the game ends. They are also the biggest tiebreaker. If more of yours survive, you win the game.
- **Laboratories:** In labs, scientists "transmute" lead into gold. This gold can be used for sages or upgrading buildings (the last three listed here). Labs are very expensive to build, but the gold they create can turn the tide on games very quickly.
- **Watchtowers:** Watchtowers have more health and damage than soldiers. A more expensive choice for combat, they are buff and tough, ready to defend from large swarms of enemies. Their ability to move means you can surround and close in on an enemy as well.

3 Reflection on Changes

Battlecode 2022 offered new challenges relative to the previous year's game. The biggest two changes that really forced us to think were resources and communication. The resource pool last year was simpler. There was only influence, and we had to decide how to divide it amongst troops. This year, there was both lead and gold. We had to decide how much lead to use for lead robots and how much lead to turn into gold. Moreover, whereas last year every Enlightenment Center had its own influence count, here there is only a global resource count. Not only do we have to decide how to split lead among units for an Archon, but we have to decide how to split up lead amongst Archons. This turned out to be a surprisingly daunting task. Last year, communication was simple. Every robot had a flag, and an Enlightenment Center could read all of the flags of its subordinates. All robots could read the flag of their Enlightenment Center. This year, however, communication was limited to a single global array: 64 spots with 16 bits each. At first, this seems like an advantage, because now it doesn't matter which Archon a unit came from - everyone can see everything. As you'll see, though, this meant that robots synthesizing information was very, very difficult.

4 Strategy Development

4.1 Sprint 1

4.1.1 Early Attack Strategies

As always with Battlecode, it pays off to start small. The most valuable unit for most of the month was soldiers. Cheap but effective, the first strategy was probably the most sensible: soldier rush. 6 bits was enough to store each of the coordinates of a map location, so it took a little less than one spot per Archon, with some room for extra information to be used later. Thus, the first 9 spots of our global array were used to keep track of Archon locations, since each team had no more than 4. The first was a setup flag inside which we kept track of counts. Why would we do that, though? The game had a function called rc.getArchonCount(), which could tell you exactly how many you had, so why would we count them ourselves? Well, just because we know how many there are total doesn't mean we know how many there are so far. When Archon #2 goes, how does it know it's #2? How does it know which space in the global array to put its map location in? Counting them manually was useful for things like this (and much more, as we'll see later). Similarly, we counted the enemies in the setup flag. For now, we assumed Archons don't move. No teams were really using moving Archons this early. Nonetheless, we avoided bugs by using another array spot to store the enemy ids. They were all single digits, so it didn't take much space. We also used four more spots so that our Archons could store arbitrary information. We called these data flags. When a unit saw an enemy base, they'd check to see if that map location was already in our array. If not, they'd increment the count and store the map location. Using the remaining 4 bits, we also kept track of how much health the Archon had left. Now of course 4 bits is not enough for over 1000 health, so we divided it into buckets. Soldiers would "rush" towards the Archon with lowest health once enemy Archons were found. So the basic strategy was, make miners to mine lead, then when we have enough lead make soldiers. The soldiers explore around until someone sees an enemy base, and then we rush it and show it no mercy.

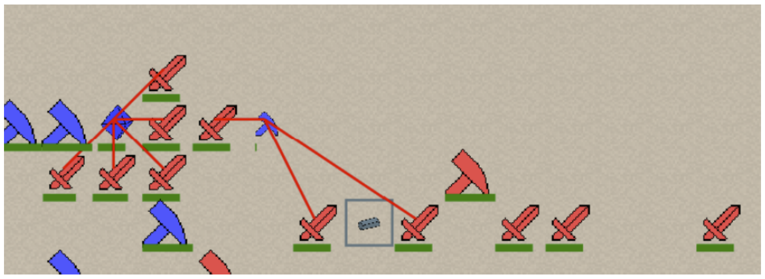


Figure 1 - Soldiers rush the enemy base.

4.1.2 Archon Organization

Last year, much of the magic was happening in the Enlightenment Center code because they had to loop through all flags. With that communication gone, though, most of the magic was actually happening in the units themselves. Nonetheless, Archon organization was still important, and there's one valuable lesson we learned from last year. For most of last year's competition, we had a state stack. When we entered some Enlightenment Center state, we pushed the current state onto the stack, and when we were done, we popped off the last state. While sensible, this led to tons of bugs and stacks a mile high. I figured out that we could instead have a general control sequence at the beginning of a turn that takes our current state and various information and does a "toggle", bringing us to our state for the upcoming round. We used this organization very early on this year, and it was clean and robust. We had an under attack state where we defend ourselves with soldiers and a chilling state where we just make normal units. Like last year, we tried to have an obesity state, where we have more lead than we know what to do with. Last year, there was no such thing as too much influence. You could make arbitrarily large politicians and slanderers. This year, though, the most lead you could spend per turn was 300, one soldier per Archon. Too much more than that, and you were certainly obese. When obese, we wanted to be making more builders. This is because the builders can use that excess lead to build watchtowers.

On small, high lead maps, watchtowers were vicious. Maxwell made the watchtowers into a checkerboard formation, and then later the watchtowers would rush the enemy as if they were soldiers.

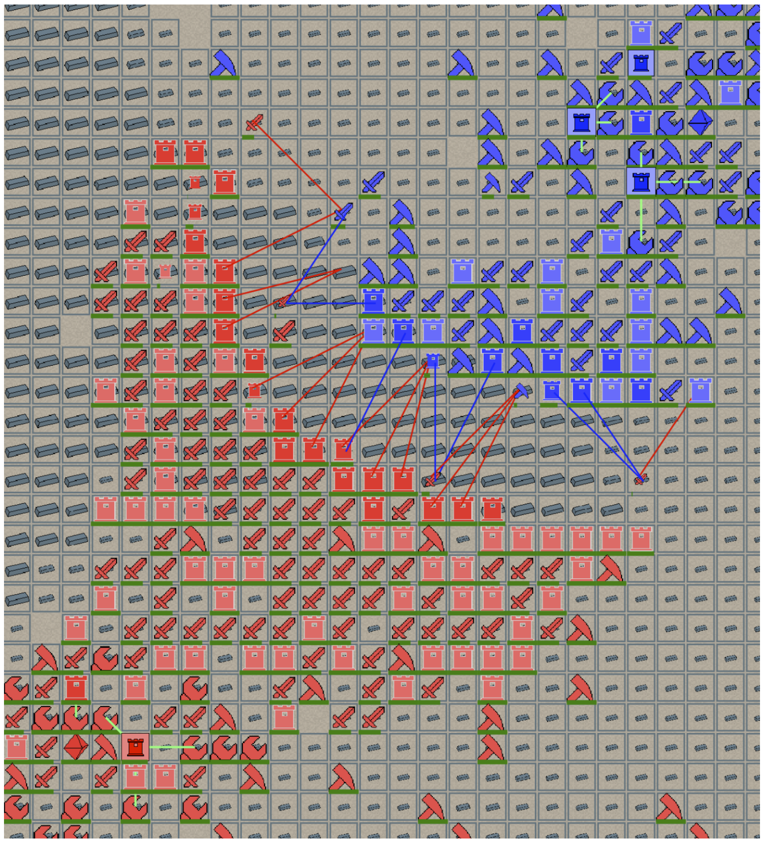


Figure 2 - Watchtowers reinforce soldiers on high lead maps.

4.1.3 Resource Allocation I: All or Nothing

Remember how we said there's a global resource pool? Well that makes things a little difficult when it comes to deciding who is going to build. Let's say that I'm an Archon, and there's 75 lead. Should I build a soldier? Well if I'm the last Archon, probably, because no one else can spend it, but what if I'm the first Archon? If I spend, no one else can. This was a complicated task, and our first draft was the simplest. We figured we should only spend lead if there's enough for everybody. In other words, if we need x lead for all of our builders, and we have n Archons, we should only build if our total lead is $75n + x$ or more. That's all and good for the first Archon, but what about the second? For this, we had to do some math. Get our Archon number divided by the number of total Archons (remember how we said the count would be useful?), and then assume our current lead is that percentage. We can then calculate the percentage of lead to reserve for the next Archon to be able to take a fair share. This was reasonable, but it wasn't very robust. How do you know that everyone wants to build a soldier? What if you make barely enough lead that you don't have enough for everybody for a lot of rounds in a row? These questions highlighted the limitations of our first model, but it was at least functional for the time being.

4.1.4 Early Defense Strategies

One problem with our initial strategy was that it was entirely offensive. If someone flanks us and attacks faster than our rush (or defends our rush), we lose. We somehow needed a way to be able to balance our soldiers between attack and defense. Our first attempt was defense soldiers. When an Archon sees enemies nearby, we spawn soldiers specifically designed for defensive purposes. This was another strategy we took from last year. When a bot spawns, it can look in its adjacent squares to figure out which base it came from. Then, it can read that array element to figure out what the data flag was set to. This gives it information about what type of unit it is. If we're under attack, we can spawn defense soldiers. They can defend our base before returning to the rush. This was okay, but then after defending an attack, they sat around and did nothing. So our second attempt was to stop using defense soldiers and instead have a portion of soldiers return home if there was a distress beacon being sent. This worked but led to an unfortunate oscillation problem. If an enemy goes inside our Archon's sensor radius for a single turn and then leaves, a distress call will send to our soldiers, who will move towards home. But then the next turn, the soldiers will go back to rushing because there's no one else in sight, so we think we've neutralized the threat. Then the next turn, if the enemy pops in again, they'll go back home. This can repeat, making our soldiers effectively useless. This unfortunately was an issue for a while and was only resolved later with clustering. Stay tuned.

4.1.5 Lead Depletion

One interesting feature of the game is that lead regenerates. If there's some lead on a square, there will be some more lead on the square in a few turns. However, lead does not generate. So if there's no lead on a square, there will never be lead on a square. As such, we wanted to be strategic about how we mined lead. The way we did it is that if we're a certain distance from our home base, we don't mine all the lead and let it regenerate, but if we're far enough away, we "offensively" mine and completely dry out lead deposits. That makes our enemies poorer. Our miners generally explored, sometimes going into high rubble to do so. This ended up being beneficial since many maps had high amounts of lead hidden behind high rubble.

4.1.6 Clustering

Okay, so you didn't have to stay tuned that long. Our solution to the soldier problem was that, rather than targeting bases, we target enemies. Every turn, every one of our units sees some number of enemies. We add up the x locations and y locations we see and average them to get the average enemy location on the map. Better than that, though, if an enemy we see has coordinates that are very far from the average we've seen so far, we consider them the start of a "new" average. We can have up to three of these averages, giving us the general clusters enemies happen to be in. This is useful because now, rather than rushing bases, Bharath made it so that we send our soldiers towards the closest enemy clusters. If they happen to be at home, great, we'll defend. If a single soldier pops into our radius, that isn't enough to shift the average. We'll just keep doing what we're doing. The Archon can spawn a few soldiers to take out the intruder, and then those soldiers will join the clusters. This offered a stronger offense/defense combo.

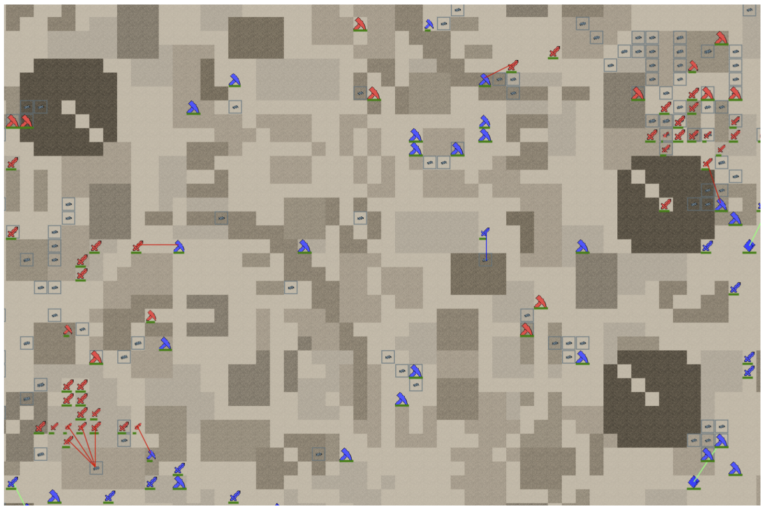


Figure 3 - Soldiers move in two clusters towards enemy averages.

4.1.7 Resource Allocation II: Cluster Prioritization

This new clustering method suggested a new way to deal with the resource problems we mentioned earlier. If all clusters are near a single Archon, there's really no reason for the others to make anything unless there's a lot of excess lead. We should prioritize Archons based on their proximity to enemy clusters. If there's one Archon closest to all clusters, then whenever there isn't enough lead for everybody, our #1 priority should be making sure the close Archon gets to build. Then, the others can try, in no particular order. If there are two Archons near the clusters, they should alternate who gets to build if there are limited options. Vice versa with 3. To accomplish this, I used another array spot to store the last three Archons to be "prioritized" so that we could pick the one chosen longest ago when forced to pick someone. Moreover, we could store lead spent so far in another array spot so that we could figure out how much lead we had this turn, regardless of which Archon we are, first or last or someone in between. Finally, we can have each Archon "guess" what unit they're going to build the next turn so that we can more accurately guess what's needed, rather than assume everyone wants a soldier. Then, we could prioritize as follows: Begin with an array [1,2,3,4] (or however many Archons we have). First, sort this array into two partitions, based on who is close to a cluster and who is not. Second, sort the close to cluster partition based on who has been chosen most recently so that we make sure every Archon near a cluster gets their chance at the limelight. Third, map these to build guesses to get how much lead each one has declared it intends to spend, in order. Fourth, calculate our total lead by adding the lead stored in the array to how much lead we have currently. Fifth, iterate through the array, subtracting the lead needed by each Archon. If the Archon sees itself, then it builds, for all the Archons prioritized before us must also have gotten to build. It was an elegant solution that allowed the streams of soldiers to be much more controlled. This gave us an advantage over the enemy, letting us flank them better and waste less time traveling. Later I added a small fix, having each Archon store in an index whether they can build, based on the rubble they're on and how much cooldown is left. We only want to say they "took" a turn if they actually built, otherwise we could have starvation where turns are skipped. We only update the most recently chosen Archons if they actually built something. This was the last change we made before the first tournament, which got us into the top 8.

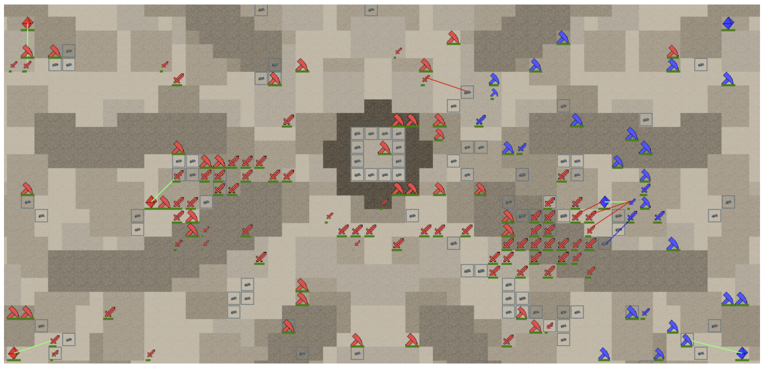


Figure 4 - The Archon closest to the enemy is making all of the soldiers.

4.1.8 Resource Allocation III: Handling Death

Unfortunately, while the cluster-based resource allocation was very effective, the code we submitted for sprint 1 (the first tournament) had a major bug. As such, I spent the entire day of the tournament fixing this bug, and then we submitted the fix when submissions reopened that night. The bug was that our prioritization code assumed Archons didn't die. Specifically, we had inconsistent behavior if the first or last Archons die. The middle ones, not so bad. The last Archon does very important things such as mark which Archons were chosen this round, and the first Archon does very important things such as resetting the build guesses back to 0. When one of those died, we had buggy code with improper prioritization. In fact, calling this a bug is an understatement, because it's not something you can just "fix" like a Null check. No, this turned out to be an issue that required a full redesign of our code. Specifically, I implemented a "turn order." Similar to a token ring in distributed systems, we had a number. When an Archon ends its turn, it increments this number. If the turn is equal to our current Archon count, we set it back to 0. I had to implement numerous edge cases, but eventually, I got something that successfully kept track of which turn number you were. Then, we could base things like clearing out flags and setting things on which turn number you were rather than which Archon number you were. This would shift the duties in the case of a crisis. We also kept track in an index of which Archons were alive so that we don't try to prioritize someone who's dead. However, even though we can only spend lead on the alive Archons, we still have to keep track of the dead ones. If Archons 1, 2, and 4 are alive, it's not like we can make an array of size 3, because then looping through that array is nontrivial. You can't do your standard (int i = 0; i < n; i++). You'll consider the wrong ones. This is another place where our count came in handy. Even if Archons are dead, we still have a count of how many we had at our peak. We can make an array of the right size and just ignore the elements that we've marked as not alive. With this new mechanism in place, we could safely allocate resources to whoever was closest to the enemies, without ever worrying about errors when someone dies. This turn order ended up being essential to most of the things we did with our code in the future, specifically when we needed something to be done on the first or last turn. For instance, anytime our units counted something, like the average enemy location, we needed to reset those values to 0 on the next turn. This fix allowed us to rise about four places on the rankings.

4.2 Sprint 2

4.2.1 Balance Change I: Lower Gold Costs

Notice that we haven't really talked about gold yet. The initial game design was unbalanced, for it costs so much to build a lab and transmute gold that it's not worth it. As a result, no teams built any labs or sages, and very few teams built watchtowers. The games were won and lost with soldiers. Attempting to fix this, the developers (devs) decreased the building costs of these units. Somehow, though, it was still unbalanced. It still wasn't worth it to build anything expensive. So, for sprint 2, soldiers remained the primary force of power. We did, however, have a slick strategy. If rounds were getting close to a tiebreaker, we would build a lab and transmute a little bit of lead to get a little bit of gold. Thus, in a game that was dead even, we could win solely off of having like, 1 gold. It was sneaky, but it worked because for sprint 2, the devs gave us maps with extremely high rubble. No one could make progress towards the enemy base, and any soldier who tried died. Lots of games ended in a tie, and we got to semis (only losing on a 3-2 thanks to our tiebreaker).

4.2.2 Healing

One nice feature in this game is that damage is not permanent. A soldier can be healed by Archons, and Archons can be healed by builders. You can keep yourself alive for longer periods of time. And if you heal a soldier rather than make a new one, that saves a lot of lead. This can turn the tables quickly. Shortly after sprint 1, we adjusted our code to send soldiers below a certain health threshold to the nearest Archon to heal before returning to their cluster to fight. This provided very good results initially. However, there was a big problem with it at first. Reviewing games, I noticed that several soldiers just kind of went out of commission. Our healing strategy involved an Archon picking the soldier with the least health, healing them until they were full, and then moving onto the next one. This seems sensible enough, but the problem is that soldiers were getting injured rather quickly since so much of the game involved soldier clashes. So, by the time you heal a soldier, another one is ready to be healed. This is also fine. The problem is starvation. If you have two soldiers, one has 20 health, and the other has 10, the 10 will be healed. But what if by the time the 10 is full, another 10 comes? If this keeps happening, the 20 never gets healed. I saw replays where this would go on for a while, with soldiers just sitting at home waiting to be healed for over 1000 rounds. This was bad because since we had often nearly a dozen soldiers out of commission like this, it allowed our enemies to have more soldiers than us in the clash in the middle. They got the upper hand and slowly advanced on us. Even though the soldier needed healing, it was more advantageous for them to just go into the fight and do their best, even if it meant death. To resolve these issues, I added a heal cap and timeout to prevent too many soldiers from staying for too long. This strategy, coupled with good soldier micro changes, made us ranked 1st for several days.

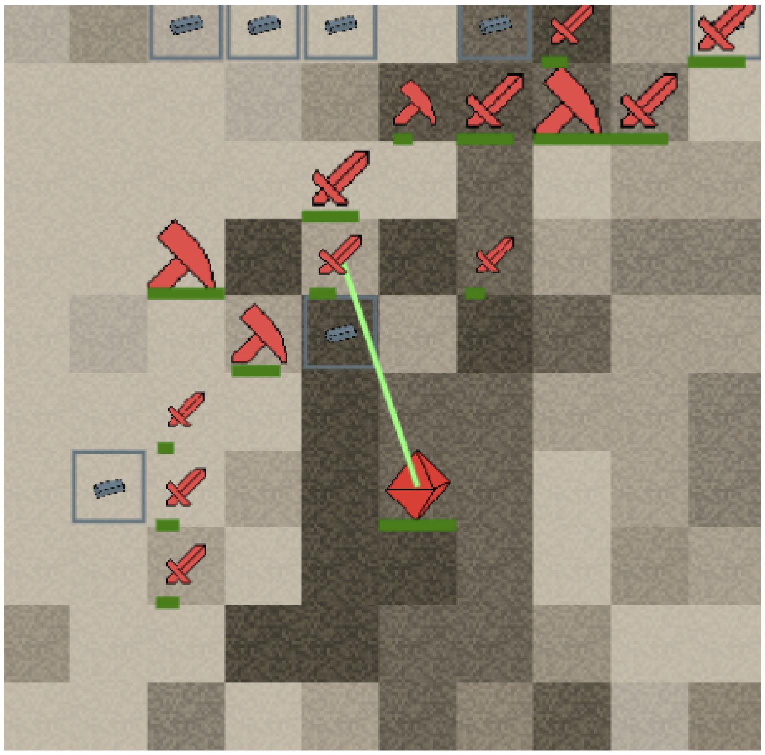


Figure 5 - Low health soldiers are huddling around the base to be healed.

4.2.3 Moving Archons

Isn't it a little unfortunate that soldiers have to go all the way back home to get healed? Over the course of a week, Winston added a feature so that Archons move towards clusters, which was probably the most intense change we made. Usually we'd only move one at a time and not move if there were any enemies in sight, but assuming it was reasonable to do so, we moved our Archons towards the fight, settling them down on the lowest rubble square we could find so that they could build quickly. We used my prioritization code for that, since if an Archon is not being prioritized for making units, it can move without compromising our forces. This allowed quick healing, which meant that our soldiers could fight without stopping yet still stay alive. This became a very common strategy, with the majority of top teams having their Archons move. Later, we also had Archons move if they were spawned on high rubble or were on high rubble because of a vortex anomaly.

Figure 6 - Archons move towards the fight.

4.3 The Final Stretch

4.3.1 Balance Change II: Even Lower Gold Costs

Didn't we just do this? Right after sprint 2, the devs made another balance change, which was basically just the same one as the last one, just done right this time so that gold actually mattered. The result was that, within hours after submissions reopened, the rankings completely shifted. Suddenly, every team was shifting from soldier strategies to a sage spam. They'd build labs as early as possible and throw so many sages at the enemy that it would overwhelm soldiers. This required us to completely change our strategy...right after school had just started. It was not a great position for us, and the road to finals was much more intense than last year.

4.3.2 Labs and Sages

In order to make sages, we needed to make labs. Making them at round 1800 would no longer be good enough. Now, we made them at about round 100. As the week went on, we advanced the quality of our lab strategies. We made more labs so that we could make more gold at once and have less turns between each sage. We mutated our labs to give them a better transmutation rate so that we'd spend less lead on the gold and still have enough to spawn soldiers. I worked on sages a little bit before sprint 2, but now it was really time to make them powerful. For starters, since gold is exclusive from lead, we didn't need to worry about prioritization between soldiers and sages. It wouldn't interfere with our lead calculations. As such, I had the prioritized Archons build sages so that they could join the fight quickly, which was good since sages had slow movement. The key to the sage strategy was that since sages have such a big vision and attack radius, you could attack people without them ever seeing you. Since sages move so slowly, they can die quickly. If they get surrounded by soldiers, they can't escape in time. By dancing outside of the soldier vision radius, though, they could assassinate the soldier from the shadows. I had our sages calculate the average enemy location and avoid it whenever possible, moving away from the closest enemy as well. We didn't want a sage to get backed into a corner, though, so it only ran away if it could run away into a spot of less or equal rubble than the sage's original location. Otherwise, we rely on our large health value to keep us alive for a bit. I also had the sages do math, deciding which would do more damage between charging to damage all the robots, furying to damage all the watchtowers, labs, and Archons, or just attacking a single enemy. I did this with damage scores, which tallied up the damage and added bonus points if we killed enemies, with more bonus points for killing more powerful enemies like sages or soldiers. By this combination of moving in and out and blasting enemies by the bunch, sages became our most powerful unit very quickly.

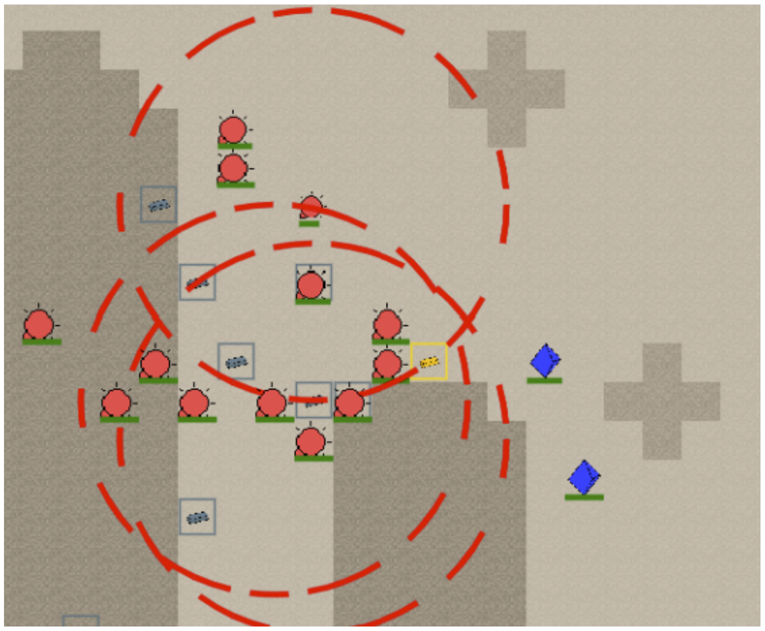


Figure 7 - Large hordes of sages become the new army.

4.3.3 Qualifiers and the Final Tournament

Qualifiers was daunting. As stated, we fell hard in the rankings when the changes were released because school commitments made it harder to respond to game changes as quickly. We went from being a top 5 team to struggling to remain in the top 20. We were resilient though, and we kept improving our code. About half an hour before the deadline, the code we pushed was finally able to do well. It beat most teams on the leaderboard with relative consistency. In unranked scrimmages, we were again a top contender. Within hours, we rose from 16th to about 8th. Unfortunately, though, most of that improvement was after they decided seeds for qualifiers. We were stuck with the twelfth seed, which is the bottom seed a US team needs to qualify. To win, we'd either need to beat the fifth seed or beat the thirteenth seed in the loser bracket. Moreover, the thirteenth seed had a strategy that, while overall not consistent, was very consistent against our strategy. We lost to them more than we lost to top eight teams. This was a very bad position. In the tournament though, our improved sage code maneuvered better than the teams we were facing, and we were able to beat the fifth seed, securing a position in finals. Making only slight improvements between qualifiers and finals like bug fixes and some of the above sage strategies like damage scores, we were ranked top 8 going into the final tournament. Thanks to qualifying through the winner's bracket, we were the seventh seed. Then finals was delayed a week because of a Boston snowstorm. Unlike last year, the maps for the final tournament were just as good if not better than the maps used throughout the season. It was an intense tournament, with the 1 seed vs. the 16 seed being a 3-2 split in the very first round. We did better than last year. Like last year, we lost the first round in the winners bracket, to a team that ended up making top 8. Then we won the first round in the loser bracket, again like last year. In the second round of the loser bracket, we faced Babyducks, last year's champion. It was a daunting opponent, but we were able to prevail. We lost the next round to a team ranked very highly in scrims. Overall, we were very happy with our performance.

4.3.4 Honorable Mentions

There are some strategies that were very interesting but that we didn't end up using. First was a sage sacrifice. Remember how, prior to the second balance update, it was unprofitable to create the gold to make sages? Well what if you already had the gold? If you have an Archon commit suicide when a miner is nearby, you can salvage the gold and instantly build a sage. We experimented with this. On one scrim, we faced Kryptonite, who was a very highly ranked team. We sacrificed and got a sage, which proceeded to immediately kill a nearby Archon. With the early lead, we took the game in only a few hundred rounds. Later, Babyducks ended up adopting this strategy, making it their highlight of sprint 2. However, we found this to generally be inconsistent, and once the balance update made sages affordable, this was no longer necessary. Another good strategy was farming. If you spawn a builder and immediately have it commit suicide on a square without lead, you create a lead deposit. It regenerates, and miners can get lead without going very far. Doing this a lot creates a lead farm around the base. Many teams used this strategy, and some beat us with it. We experimented with farming but found little success. It could've been explored further.

5 Thoughts about the Game

On the one hand, I really liked that this year's game made more sense. Soldiers fight, miners mine, and builders build. Watchtowers and labs are exactly what they sound like. It was much easier to introduce my mom to the game so that she could watch streams with me, whereas last year there were politicians, slanderers, and muckrakers that were somehow killing each other? This year's game felt like a true robot battle. On the other hand, the fixed unit costs made for monotonous strategies. Everyone was doing soldier rushes, and then everyone was doing sage spams. Fixed costs meant that there was a mathematically ideal strategy, whereas last year, you could make certain types of bots more powerful by giving them more influence. One mechanic, anomalies, was completely ignored. There was a time where we had sages avoid charges, but after a while, there were so many sages that it didn't matter. Fury and abyss there was nothing you could do about, and vortex just meant scooting an Archon over a little bit so it's on lower rubble. Very few strategies cared about the global anomalies in the game. Mutations were also somewhat irrelevant. Many teams in finals mutated labs to make them transmute lead into gold more efficiently, but other than that, there wasn't a huge focus on mutation in most cases. Last year, there were three robots and one building. This year, there were four robots and three buildings. Creating a cohesive strategy that involved seven different units was very difficult, which is why I think strategies ended up only using some. Less units with more flexibility could vary the strategies.

6 Insight from a Second Year

Last year, we made several recommendations to new players based on our rookie experience. This year, we applied the things we've learned, which allowed us to do even better. I think the biggest learning experience was the first year, for sure, and I don't really have any new recommendations. However, we did approach our workflow differently, and that turned out to pay off huge. First of all, we had more versions. Last year, we had a bot for each sprint tournament, a bot for qualifiers, and a bot for finals. This year, we had 20 bots, one for each time we made a major change. Whenever we made a small change, we tested it against some of the most recent bots to make sure it worked against several versions, not just one. Last year before finals, we made a spreadsheet of how our bot did against our qualifiers submission on every map. This year, we did that from the very beginning and had a massive spreadsheet tracking how each change did against all our different bots. See it here.

7 Final Thoughts

CMU had 4/12 of the US finalists. I mean come on...