The Kragle - Battlecode 25 Postmortem

Justin Ottesen, Andrew Bank, Matt Voynovich

Last Updated: January 28, 2025

1 Introduction

We believe our postmortem will be of unique help to future Battlecode participants. Often, teams who make postmortems are those who have consistently placed highly from the start. We have the perspective of a team that spent years with no consideration of qualifiers, who this year made the jump to be one of the higher level contenders, going toe-to-toe with the 2 seed in the US qualifiers.

Hopefully you find this useful!

1.1 Our Team - The Kragle

We are a team of three computer science students at Rensselaer Polytechnic Institute (RPI). In this year's competition, we made 'the leap' from being a team that couldn't submit a final bot to being a contender for the final tournament. We'll outline some tips and tricks for other aspiring teams to make 'the leap' themselves. We are historically terrible at coming up with team names, and ended up choosing "The Kragle" after Justin had a burst of inspiration working on a `MicroManager` class shortly after re-watching The Lego Movie. Just you wait for what we cook up next year. Some more information about us is below:

Andrew Bank has competed in Battlecode since 2021. He is a Computer Science and Computer Systems Engineer at RPI and will graduate in this spring 2025. Andrew has interned at Johns Hopkins Applied Physics Lab, he has created the Circuit Randomizer for Personalized Learning as an undergraduate research project under Professor Shayla Sawyer, and is currently working on another undergraduate research project: the MusicX project for Professor Sawyer's Mercer Xlab. In his free time, Andrew enjoys having more Strava follwers than Instagram followers, and he enjoys treating Stardew Valley like an operating systems optimization problem while playing with his siblings.

Justin Ottesen has competed in Battlecode since 2022. He began his undergraduate degree in Computer Science in Fall 2021, and graduated early in Spring 2024. He stayed at RPI for his Master's, where he does research with the BRAINS Lab under Oshani Seneviratne with a focus on incentive design for smart contract protocols, and is on track to graduate in Fall 2025. Outside of classwork, he has worked as an intern at Nasuni since Summer 2023, and is on the D3 NCAA Cross Country and Track teams at RPI. In his free time, he enjoys hiking, running, programming, and playing Mario Kart Wii.

Matthew Voynovich started competing in Battlecode this year. He began his undergraduate degree majoring in both Computer Science and Information Technology and Web Science at RPI in Fall of 2023. He is scheduled to graduate in Spring 2026 and plans to pursue a masters under the Co-Terminal program at RPI. Currently, Matthew works as an undergrad researcher under Thomas Morgan studying quantum phase investigations, and in his free time enjoys long distance running with his friends in the Rensselaer Running Club.

1.2 Past Performance

Andrew was the only member to compete in 2021. In 2022, Justin joined Andrew, working together again in 2023 and 2024 along with some other teammates. We did not perform particularly well in any of these years, typically holding a 1200-1500 rating, with no notable performances in any of the tournaments. Unfortunately, the RPI spring semester starts very early, so we were always balancing classwork along with the competition. This year we went all in, and entered the US qualifier tournament with the 10 seed, rated at 1730. I guess you'll have read this to see how we did...

Although we aren't sure our mess of a repository will be useful to anyone else, our GitHub repositories can be found below:

2021 **Fire Nation** - Lost to the sands of time...

2022 **Kernel Byters** - https://github.com/justinottesen/Kernel-Byters

2023 **PC Ghosts** - https://github.com/andrewkbank/bc2023 (This is still private)

2024 **Goat House** - https://github.com/justinottesen/battlecode24

2025 **The Kragle** - https://github.com/justinottesen/battlecode25 (This is still private)

1.3 Game Overview

This year's introduction message is shown below:

> *The bread and food of yore has begun to run out, forcing robot society to adapt. Gone are the jovial ducks, replaced by steampunk robot bunnies who have converted their need for nutrients into a reliance on paint. These bunnies have become territorial, forming clans and defense formations to protect the resource that keeps them running.*
>
> *For the past two centuries, these bunnies have stayed within their own territory, but clans have begun to degrade their environment and need to start branching out. Will these clans be able to expand their territory and generate enough paint to protect their families? Or will they stray too close to other clans and be wiped out in conflict?*

As hinted above, this year's game was a competition of territory control between paint-crazy bunnies. The first team to paint 70% of the map would win. Each team had two starting towers, which could produce resources and robots. These robots had different abilities, but their goal was to work together to build more towers and paint as much territory as possible.

We saved copies of both the initial and final specs to our repository in case they are taken down in the future. For a full description of the game, see our repository. If you are new to Battlecode, we highly recommend this Battlecode Guide, written by XSquare, a long time Battlecode competitor.

2 Strategy & Implementation

2.1 Sprint 1

2.1.1 Setup

The first thing we did was decide how to structure our code. We took a lot of inspiration from XSquare's 2024 code. We created an abstract `Robot` class, and created a subclass for each robot type (`Soldier`, `Splasher`, `Mopper`, `Tower`). This heavily simplified our `RobotPlayer` class, and helped us split functionality between differing robot types[1]. Along with these classes, we also had utility classes, like `MapData`, `Explore`, `Bugpath`, and others.

[1] In hindsight, it may have been even smarter to have another level to the hierarchy where we had Unit as the base class, Robot and Tower as subclasses, and other types inherit from these. Towers don't need to do pathfinding, and they can't really learn much about the map, so this was bad organization and wasted bytecode

Our first priority was getting a bot that could effectively explore and capture towers. Building towers would allow our team to increase their resource income and create more robots. We had each tower spawn a soldier and a mopper who would search for a ruin. We copied the capture logic from examplefuncsplayer. The soldier was to paint the pattern around the tower, and the mopper was to clear any enemy paint in the way.

2.1.2 Resources & Towers

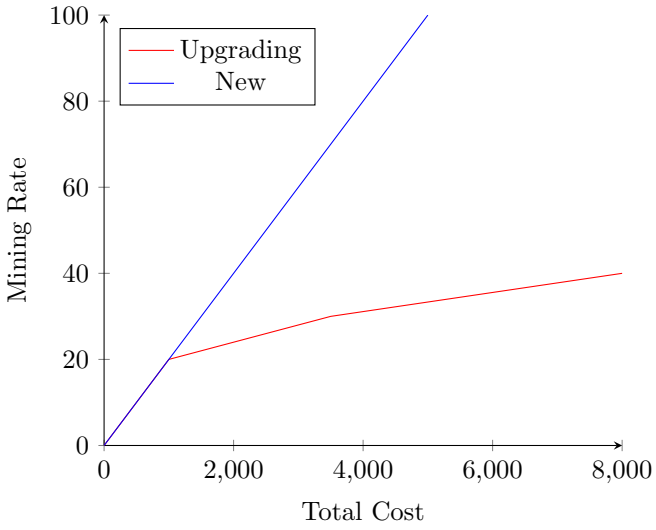
We made two observations very early on about resources:

1. It is not worth upgrading towers.
2. It Money towers are always better than Paint towers.

The first one is relatively obvious looking at the costs of new towers vs upgrades. For Money towers, the following table shows the cost associated with each level of the tower:

Level	Upgrade Cost (Chips)	Mining Rate (Chips / Turn)
1	1000	20
2	2500	30
3	5000	40

From this table, we can graph the cost and mining rates:

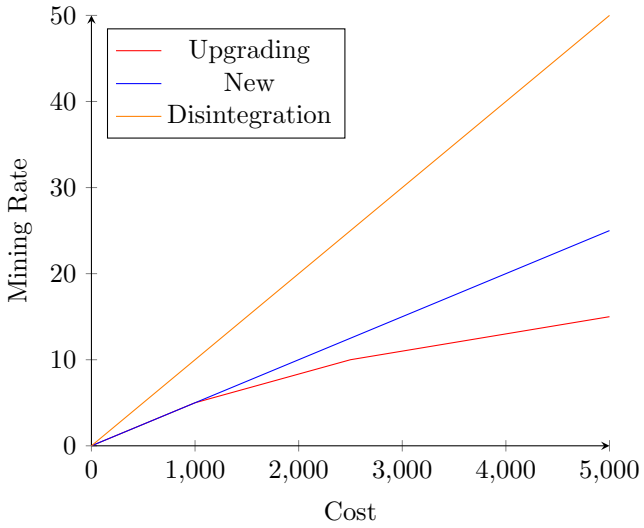


As is clearly shown here, when you have the choice between spending on a new tower, or upgrading an existing one, there is no reason to upgrade. The same can be said for Paint towers, but the next observation makes that redundant.

Even though paint is arguably the more important resource, Paint towers are useless[2] because money towers actually produce paint more effectively than paint towers. Each tower spawns with 500 paint. Since money towers produce 20 chips per turn and they cost 1,000 chips, they pay for themselves in 50 turns. So, if a Money tower calls `rc.disintegrate()`, we can trade 1,000 chips for 500 paint. If we do this every 50 turns, money towers have an effective paint mining rate of 10 paint per turn, which is double what paint towers are capable of.

Here is the corresponding graph to the above for paint production vs chip cost:

[2]This is true until you factor in SRPs, but we are getting there



There are a few other big bonuses of this strategy:

1. We now have the choice of converting chips to paint. With paint towers, if we have more paint than we need or can use, we are stuck with it.
2. We no longer have to coordinate what tower types we are building, since we are always building money towers (we ignore defense towers for now).
3. Since chips are global and paint is local, we can use money towers that are far away from the active parts of the map to subsidize paint production where it is needed most.

With this, it made a very useful optimization trivial. Marking a tower pattern costs 25 paint, which is significant considering the soldier's paint capacity is 200, and it already takes a minimum of 120 paint to capture a ruin. Since we knew we were only building money towers, we could avoid marking the pattern and save paint.

2.1.3 Special Resource Patterns

Once SRPs are introduced, the math on paint production changes. Each active SRP boosts paint production by 3 units per tower per turn. The following table shows how the number of SRPs affects the per turn production rates of different towers:

SRPs	Money Tower Chips	Money Tower Paint	Paint Tower Paint
0	20	10	5
1	23	11.5	8
2	26	13	11
3	29	14.5	14
4	32	16	17
5	35	17.5	20

As you can see, once there are 4 or more SRPs, paint towers are better suited for their intended purpose. We decided not to worry about this and stick with our money tower only strategy, to keep things simple. Plus, we (regrettably) did not prioritize SRPs for Sprint 1, our soldiers simply checked if the current location could have an SRP and if so, they marked one.

2.1.4 Pathfinding & Map Representation

Units stored their knowledge of the map in memory, however we ended up rewriting this system, so this will be described further later. Units traversed the map using a BugNav algorithm. We used the BugNav

algorithm from XSquare's 2024 code, and it worked with minor tweaking. If you have questions about how to implement this, there was a lecture on it this year here.

Optimal pathfinding wasn't a high priority this year since ruins/towers guaranteed a lot of open space, and the specs guaranteed walls to take up less than 20% of the map. In theory, Teh Devs could've made maps with few ruins and many walls, but that never happened. Even though something like unrolled BFS would've been optimal, BugNav worked well enough in practice, and because source code for it already existed, we saved a lot of time not worrying about pathfinding.

2.1.5 Sprint 1 Performance

The bracket for Sprint 1 can be found here.

We entered the tournament as the 55 seed out of 160 teams, with a rating of 1533, our first match was a 4-1 win against the 74 seed, "be right back" with a rating of 1483. Their strategy was to do an early rush with soldiers to try and take out our towers, then produce a single splasher to cover some paint. This worked against us on the first game, as our starting soldiers got stuck trying to capture an SRP with enemy paint, and our moppers didn't help them.

The left image below shows the result of our loss, and the right image shows one of our wins. On this particular map, the opponent's rush got trapped against a wall.

Our next matchup did not go as well for us. We faced against the 10 seed "immutable", who was had a rating of 1737 at the time, and got swept 5-0. We kept up in the early game, our initial soldiers seemed to perform about equally in finding ruins, but after some starting territory was established, they always pulled away with a commanding lead. There were three main reasons for this:

1. They prioritized SRPs heavily. In Sprint 1, this was definitely the meta, since the pattern allowed for an absurd amount of overlap, and there was no cost to making them. This gave them a much stronger economy than ours.

2. They used their economy better, setting on about 50% soldiers, and 25% each of moppers and splashers. We usually ended up with about even soldiers and moppers with no splashers, since we did not have time to create splasher logic by this point. Their balance allowed for much better expansion.

3. They explored the map better. Our soldiers would often get stuck on ruins they couldn't complete without help, and our moppers were terrible at finding soldiers to help. This meant we got stuck anytime we saw enemy paint.

We are sure there are many other things they did better than us, but these were the three big differences that made the others negligible.

2.2 Sprint 2

Add sprint 2 writeup here

2.3 US Qualifiers

Add quals writeup here

3 Conclusion

Add conclusion writeup here