SPAARK Battlecode 2025 Postmortem

Maitian Sha, Suvanth Erranki, Erik Ji, Jie Gao

Table of Contents

Team Introduction

We are team SPAARK. We are four high school students from West Windsor-Plainsboro High School South and this is our third year doing battlecode. We got 3rd place in 2024 and 5th/6th place in 2023, so we were really determined to try and do better this year.

GitHub repository: https://github.com/erikji/battlecode25
High School Tournament submission package name: TSPAARKHS
Final bot package name: SPAARK

There are archived versions of old bots with different strategies, labeled with dates, as well as verbatim copies of tournament submissions available.

Game Intro

(basically condensed specs)

This year's game is a lot like Splatoon. Much unlike previous years, instead of direct combat, each robot has two bars, paint and HP. Robots lose paint for performing actions, standing on enemy or neutral paint, or being too close to allies. If a robot runs out of paint, it starts losing HP quickly, and dies at 0 HP. There are two currencies, paint and chips. Much like previous years, this is a game about territory control and economy, with different specializations and tradeoffs of bots.

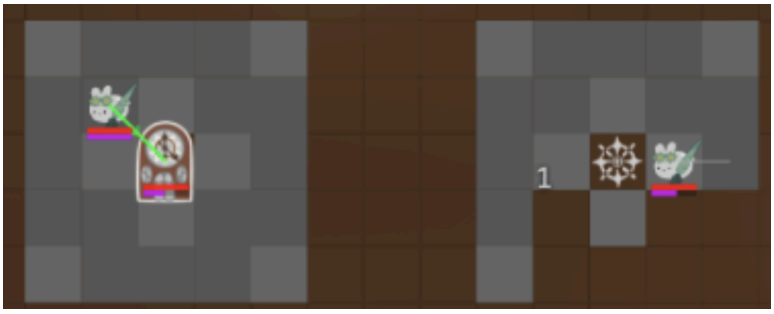
Three robots:

- **Mopper**: costs a lot of chips but little paint, good at removing paint from enemy robots and enemy paint on the map
- **Splasher**: costs a lot of paint but little chips, good at painting a large area
- **Soldier**: costs some paint and chips, good at painting patterns (described below). Importantly, it cannot paint over enemy paint, so it needs Mopper/Splasher support to take on enemy robots.

Three towers:

- **Paint tower**: produces paint
- **Chip tower**: produces chips
- **Defense tower**: has longer range and better attack damage, buffs allied towers' attack, gains chips for every attack that hits >=1 enemy

To build towers, you need to create a precise paint pattern with soldiers: (Left is a completed money tower pattern, right is an incomplete one)



There are two types of paint, primary and secondary, so you can create these patterns.

Every tower can do an AoE attack **and** a single target attack every turn, depleting HP from enemy robots. They can also spawn robots.

There is also the Special Resource Pattern, another 5x5 pattern, which once completed gives you an extra 3 resources/turn for all towers. (important for large maps) Each SRP costs 200 chips and takes 50 turns to activate.

Sprint 1 Strategy

The first few days were spent implementing every feature the game had to offer. Some things we did:

1. **Spawning bots**
    a. Our algorithm for spawning bots was very simple for sprint 1.
    b. We would always spawn a soldier first and then a splasher.
    c. Then towers will spawn bots in a repeating pattern of mopper-soldier-splasher -soldier-splasher-splasher-splasher.
    d. Towers would prioritize spawning bots close to the map center.
    e. We also tried heuristics based on map size, and sometimes spawning less soldiers made the bot win. These changes became obsolete with updates and also adding weights before sprint 2.
2. **Basic attack/move micro for soldiers**
    a. Soldiers would move in, attack tower, and then attack and move out in order to get twice as many hits on the tower.
3. **Basic exploration**
    a. We stored an array of 60 longs to keep track of explored tiles (each long can be used as 60 bits, so we effectively have a 2d array but more bytecode efficient)
    b. Bots would choose a random tile that has not been explored to try and explore it.
    c. We tried many different other strategies, such as sending bots to the center of map quadrants. The tradeoff though was that the bots would spread out too far, especially on larger maps with more walls, and wouldn't be able to return in time. Even on smaller maps, this gave mixed results. This strategy ended up being unused after revamp.
4. **Building towers**
    a. We used rc.getNumberTowers() to try and keep the number of paint and money towers balanced, so robots would build a money tower if we had an even number of towers and a paint tower if we had an odd number.
    b. Towers upgrade if we have a lot of money
5. **Tower attacking**
    a. Attack the bot with the lowest hp, unless it has less hp than our attack power, in which case attack the bot with the highest hp. If two bots have the same hp, attack soldiers, then moppers, then splashers. This definitely could be changed to be more optimal, but we never did.
6. **Build SRP, expand SRP (complicated but effective)**
    a. "Smart" SRP building greedily places SRPs and then attempts to tile them optimally, but will settle for a less-optimal tiling if necessary (obstructions by walls, ruins, or other SRPs)
    b. SRPs take second priority, if a soldier doesn't know about any ruins or opponent towers that it hasn't explored recently, it will build SRPs
7. **Add defaultMicro**
    a. Predict paint loss based on which square we move to
    b. Micro

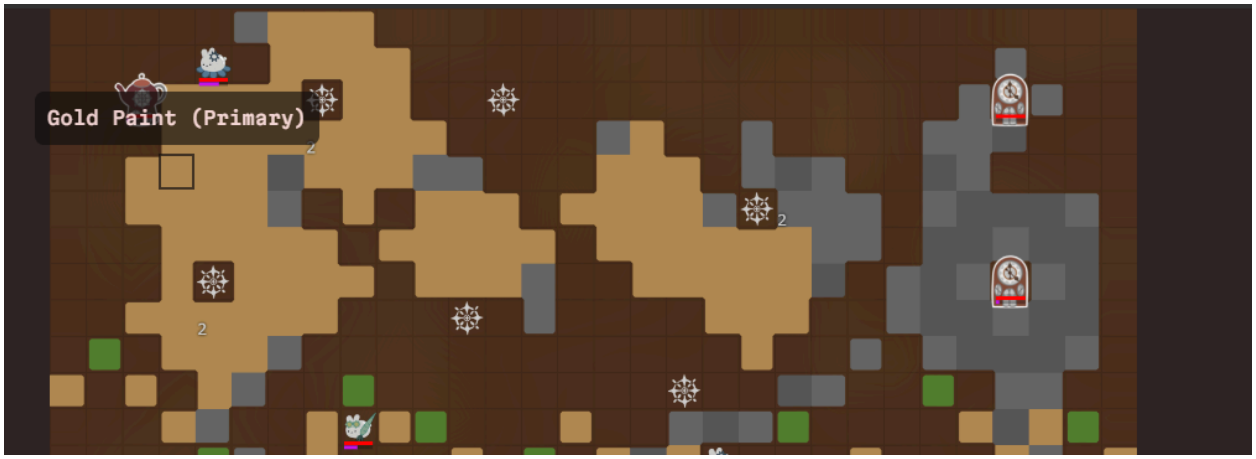
8. **Comms**
    a. We stored all towers/ruins, and for each tower/ruin we stored its location, its team, and its type (paint, money, defense, ruin).
    b. We also stored the map symmetry using 3 bits, with each bit representing if the symmetry is valid or not. Up until Sprint 2, symmetry detection was actually broken, but we didn't use symmetry info so it didn't matter.
    c. Comms
9. **Retreating to refill paint**
    a. We made robots retreat to the nearest paint tower if they had less than 1/3 of their paint capacity.
10. **Pathfinding (bugnav)**
    a. Because the maps have less obstructions, pathfinding is less important in this game, so we stuck with our bug2 pathfinding instead of last year's BFS (Bellman Ford Search) as it used too much bytecode.

The code was definitely suboptimal, and we ended up getting 0-5'ed by XSquare. It also didn't help that soldiers refused to build SRPs due to 2 lines getting inadvertently removed.

Sprint 2 Strategy

At this point, there were still many things we obviously needed to improve:

1. **Mopper/splasher micro**
    a. Micro
2. **Tons of defaultMicro/soldier micro improvements**
    a. Micro
3. **Tons of bytecode optimization**
4. **Splasher rush**
    a. Our main strategy for sprint 2 was splasher rush, which is spawning 1 soldier and 1 splasher from each tower at the start, and having the splashers guess symmetry and mess up opponent tower building.



5. **Change spawning strategies**
    a. We changed towers to instead have a weight for each robot type and this made it possible to vary how many of each bot we spawned.
6. **Better exploration**
    a. Bots would use known tower/ruin data from comms combined with the symmetry data to guess symmetry and head to opponent towers
    b. This was basically our entire macrostrategy for splashers and moppers.
7. **Better retreating**
    a. We changed the amount of paint for the bot to enter retreat mode to be dependent on how much paint it lost due to being next to ally robots or being on neutral/opponent paint.
    b. We noticed that some bots were dying while circling a paint tower to get paint due to the paint loss from being next to your own bots, so we made bots wait 2 squares away from the tower, until it detects the tower has enough paint.
8. **Better comms**
    a. We implemented rc.broadcastMessage for towers.
    b. We also changed to the 12 by 12 map to optimize bytecode.
    c. Comms
9. **Camel_Case bugnav**

    a. Last year's bug2 pathfinding was bad so we copied camel_case's bugnav and changed it a bit.
    b. https://github.com/jmerle/battlecode-2024/blob/master/src/camel_case_v21_final/BugNavigator.java

10. Change tower building

    a. Our Sprint 1 bot was very slow at building towers because every time a new tower got built all the bots would change the paint patterns from money to paint or vice versa. Instead of using rc.getNumberTowers() we used the comms system to get how many paint and money towers we currently have.
    b. We also made the bot place markers around ruins to communicate which tower it was going to build.
    c. We also experimented with the ratio of money/paint tower building, but ultimately kept a 1:1 ratio initially (before updating it later).

11. Loop unrolling

    a. We began some loop unrolling for moppers and splashers, though soldiers remained loopy other than some SRP code.

This got us to the second page, but even after extensive optimization (and a few overnight runs), it was still losing to the top teams…

High School Tournament Strategy

We didn't make top 16 in Sprint 2, and it was mainly because our "optimal" splasher rush strategy was easily countered by the two main strategies top teams were using:

Soldier build:

1. Spawn 4-6 soldiers at the beginning
2. Spam money towers to get 500 paint each, spawning more bots in exponential growth

By the time our splashers would arrive at the scene, the opponent likely would have already finished 1-2 money towers and begun a few more. Our strategy mainly focused on being able to stop the opponent from building their first few money towers, however the top teams were able to build towers faster than our splashers could arrive, which made our splashers not as useful.

Soldier rush:

1. Spawn 4-6 soldiers at the beginning
2. The soldiers would guess symmetry and head to opponent towers to destroy them

When our splashers arrived at the opponent towers, there would be no ruins that were being built, making the splashers useless. Meanwhile, the opponent soldiers would already be attacking our towers, and the two soldiers we spawned to build towers would not be able to build more towers fast enough before the two original towers were destroyed.

In Sprint 2 we ended up getting 0-5'ed by confused using the soldier rush strategy. However, soldier rush was nerfed after the sprint 2 balance changes that made your initial towers spawn at level 2.

On Jan 22, we started to work on **solidbuild** (now renamed SPAARK), copying the soldier build strategy. Initially, it had a meager 25% winrate against SPAARK, but with some tweaking, we were able to make it win consistently.

Some small last-minute optimizations we did:

1. **Money tower disintegrating, if we have a ton of chips**
    a. Betterdisintegrating
2. **Defense tower building (and disintegrating, if it hasn't seen an enemy)**
    a. Defense towers would be built if it is within sqrt(35) of the map center and enemy paint/bots are detected.
3. **Don't retreat unless you are close to the tower**
    a. Surprisingly, this won against old bots by a very large margin
    b. Bots refill if they pass by a tower that has paint in it, even if they are almost full
4. **Mopper paint transfer**
    a. Moppers will try to transfer some of their paint if they see an ally bot with 0 paint.
5. **Reduce congestion**
    a. After 100 rounds, towers will only spawn bots every 2 turns and only if there are <4 ally bots nearby.
6. **Change spawn weights (again)**

a. Spawn more splashers depending on how many paint towers we have

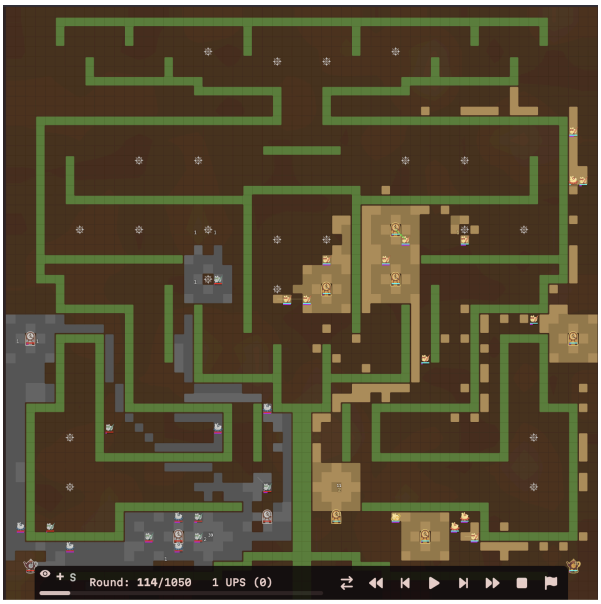
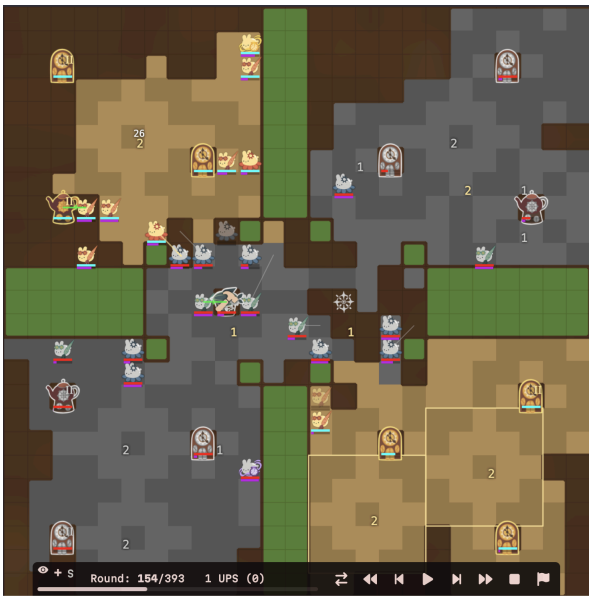
We never tried money tower flickering, though in our defense, only Pantheon and Gone Whalin had tried to use it up to that point.

High School Tournament Experience

During the HS qualifiers, we had a nail-biting 2-3 loss to The Realer Merlin. After obliterating a few teams in the losers bracket we faced The Realer Merlin (we were gold) in a rematch on five maps:

1. Pathfinding challenge, where we luckily found the center ruins, claiming them before TRM saw it and easily winning the rest of the game
2. We lost center control, and they were able to build a defense tower in the center and completely destroy us.
3. We were able to find the center ruins (which were the only ones not initially blocked by enemy paint) and quickly get a defense tower and a second money tower. TRM did not place paint near ruins, so their economy lagged, getting us a second win.
4. TRM seemed to be winning initially, quickly whipping up patterns for two annoying defense towers, but critically, we were able to get rid of their single money tower. With only the defense tower to produce money, our bots were able to avoid feeding it too much until we slowly but surely came back to win the game.
5. Wasn't played on stream, but we lost it, so match #4 was actually necessary for us to advance.

Us and TRM were neck-and-neck during the qualifiers and it was really a coin toss as to who would win the loser's bracket.

Game 1	Game 2
	
Game 3	Game 4

After the quals, we had a surprising jump to 3rd place on the leaderboards, and then dropped to 10th, and then went back up to 6th. Meanwhile, Pantheon, the other HS qualifier, was also randomly going up and down on the leaderboards, making for an interesting final match.

Rating	Team	Members	Quote	Eligibility	Ranked Scrimmages	Unranked Scrimmages
2102	Old But Gold	XSquare, dirbaio	Make Java Great Again		Auto-Accept ✓	Auto-Accept ✓
2090	confused	skipiano	?		Manual	Auto-Accept ✓
1976	SPAARK	maitian, SuvanthE, sungodtemple, jie__gao	waxed lightly weathered cut copper stairs	us	Auto-Accept ✓	Auto-Accept ✓

Our peak position on the leaderboard. We got to 2018 rating at one point yay

Since our high school is in New Jersey, we drove to MIT. Multitudes of thanks to Teh Devs for organizing the finalist dinner and career fair (we got a ton of merch)! We also had a fun time at the MIT Museum.

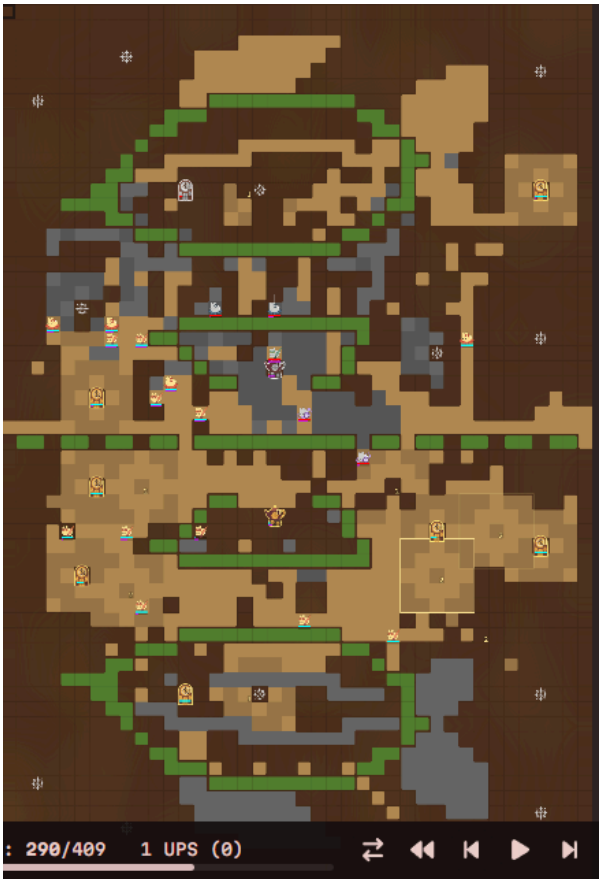
We were not expecting to win the HS final tournament, as we went through the losers bracket and had to win two matches. We were also roughly tied in scrimmages with Pantheon.

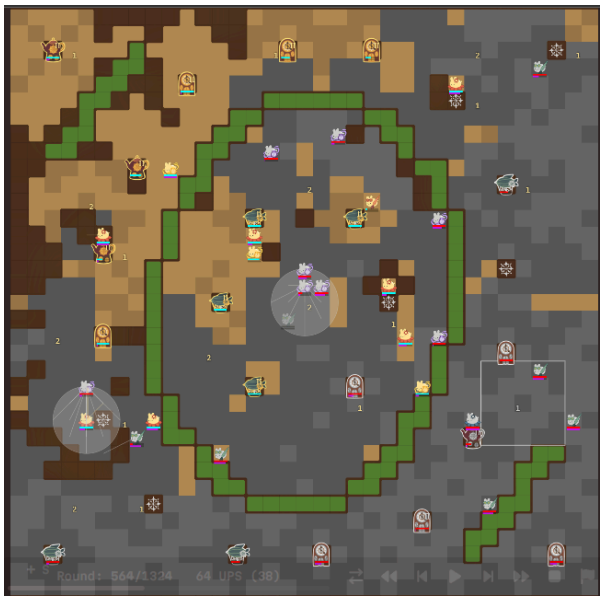
During the actual tournament, we found out that Pantheon had actually targeted their strategy against TRM instead of us, so we actually had a glimmer of hope in winning against them. Because the game was *balanced well this year (hint hint)*, their counter to TRM's strategy wouldn't be as effective on our strategy. Also it turned out we were winning against scrimmages before the tournament, so uhhh…

Final Tournament Matches

Match One: (We were gold)

1. We found the ruins on the side and were able to destroy Pantheon's paint tower.
2. We were able to get to the ruins in the center faster and build a defense tower there, and we also were able to build more SRPs.
3. Pantheon was able to quickly destroy our paint tower, and we lost
4. This game was super stressful, as we claimed the center really early on and built 4 defense towers. However, Pantheon was able to sneak around the edges and destroy many of our towers. Pantheon gained a lot of map control (68.9% to be exact), but our defense and paint towers clutched up and we came back.
5. Wasn't played on stream, but we won, so the result of game 4 didn't matter ._.

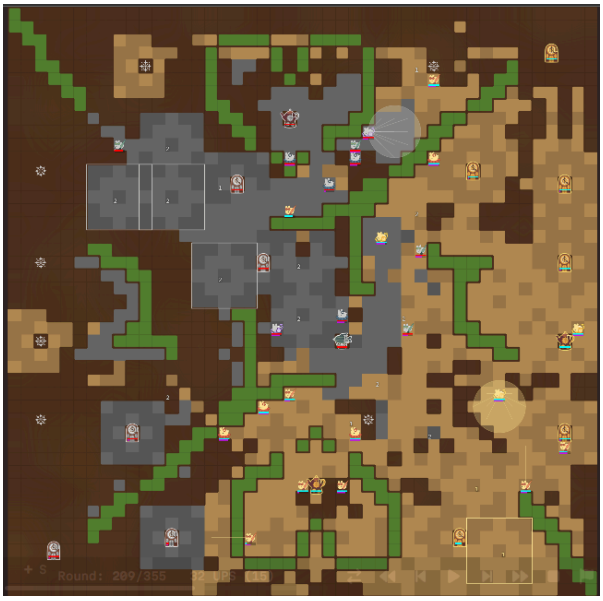
Game 1	Game 2
	
Game 3	Game 4

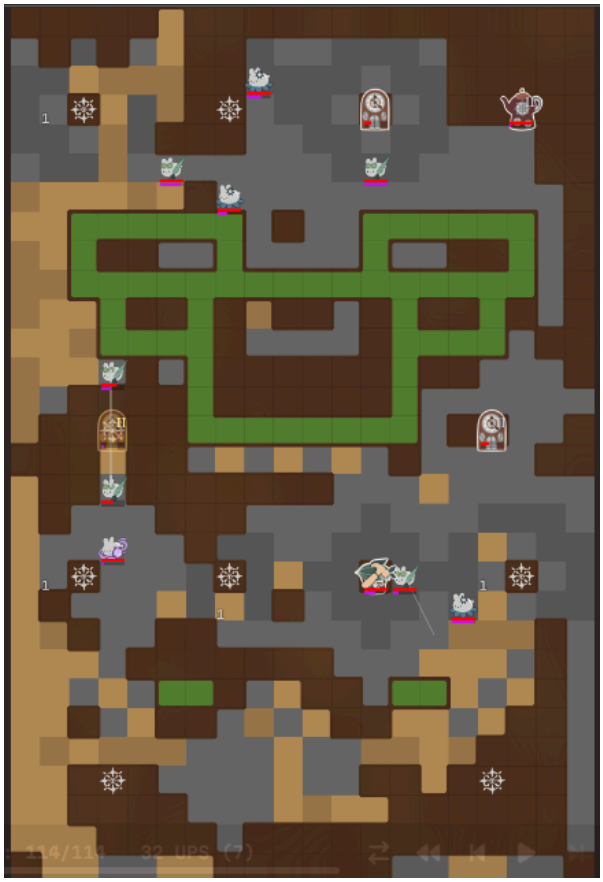
	

Because we won the first game, we were both at one loss, so we played a second match (as silver):

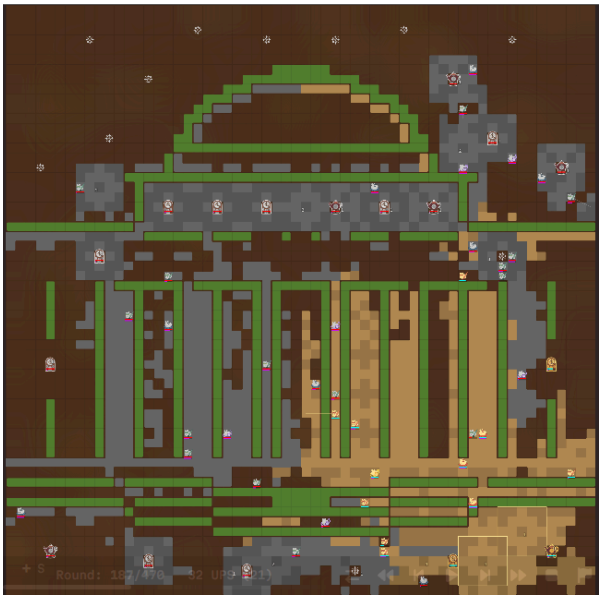
1. Pantheon got all the ruins on the right, thanks to their bias towards exploring edges, and we quickly lost as they came into the center.
2. Our soldiers quickly destroyed both of Pantheon's starting towers, cutting off their resource supply and winning us a game in just 114 turns.
3. Our soldiers somehow found all the ruins at the top of the map, giving us crazy econ and letting us crush Pantheon in the center.
4. On this map, both us and Pantheon had the same idea of spamming money towers. We were tied up, each with three money towers, and our soldiers were constantly moving to the opponent's side to destroy their money towers and vice versa. Luckily, we had taken out their paint tower in the early game, giving us a slight advantage that allowed us to win.
5. Wasn't played on stream, but we won, so the result of game 4 didn't matter ._.

Game 1	Game 2

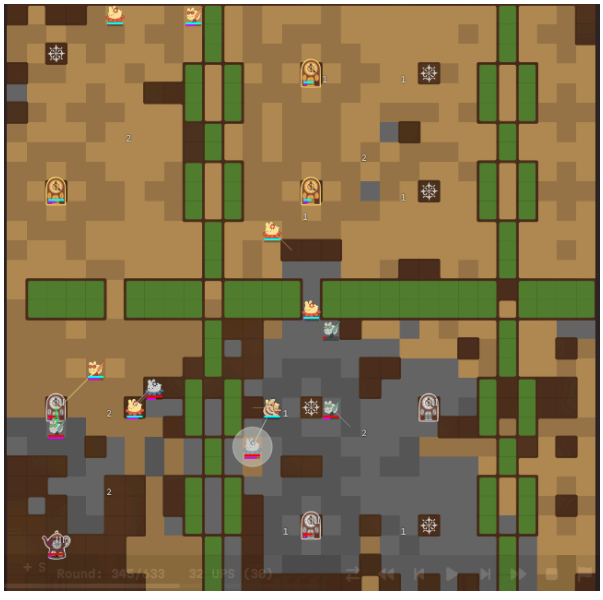




Game 3



Game 4



Strategies that we think are really cool

Micro (get it its in a 'micro' font size)

We were very trash at microstrategy ("micro") in 2024. For example, this match of 3 with cout for clout, where our ducks get absolutely destroyed in micro and only win due to aggressive flag passing: Battlecode 2024 Match - SPAARK vs. cout for clout

This year, we decided to make micro better. Micro is essentially weighting different attacks on how much paint/chips/hp/etc you gain. We based our micro on the most apparent resource which is paint: 5 score ←→ 1 paint

We defined microstrategy as a functional interface in Micro.java, and we placed our default micro function in Motion.java as defaultMicro. We can extend defaultMicro to things like moveWithPaintMicro, which for soldiers produces weights with the knowledge it can paint under itself.

In the defaultMicro function, we choose where to move based on the direction the macro is telling us to go as well as paint penalties.

Here's everything we put into defaultMicro:

- 20 score (4 paint) for moving in correct direction
- 15 score (3 paint) for moving in adjacent direction
- Lose score based on what paint the tile you are moving to has (5 score per paint loss)
    - Moppers x2
    - [cool strat] Keep track of movement cooldown and the function for adding cooldown, if you won't be able to move next turn, then multiply penalty
- Lose score based on how many allies are surrounding the tile you move to (5 score per paint loss)
- Lose a lot of score if the tile you move to is in range of enemy tower, even more if you can get 1 shotted
- Lose 20 score if you are in move+attack range of enemy mopper
- Lose 1 score if you visited the tile < 5 turns ago

Splashers and moppers also have micro for which tile to attack. For example, mopping enemy paint with an enemy robot on it gives 5 paint for removing enemy paint, removes 10 paint from the enemy robot, and gives 5 paint to the mopper, for a total of (5+10+5)*5=100 score. We then added a few extra bonuses, such as 150 score for mopping next to a ruin, or ~100 (depending on the bot's money cost) score for reducing an opponent robot to 0 paint.

Lastly, one **very important** part of micro that we overlooked in both 2023 and 2024 was considering both options of attacking then moving **and** moving then attacking. While this seems like a simple observation, a lot of code frameworks make it hard to consider both ways and thus

have crappy micro. We had to refactor the code for Mopper.java and Splasher.java to incorporate this.

```
public static void exploreAttackScores2() throws Exception {
    // dont delete this function its here in case the codegen for
    // exploreAttackScores gets too big
}
```

In general, while it seems like you should implement some complex algorithm for micro, just taking all the game constants at face value is enough to be competitive with top teams.

RNG

Random number generation is not very optimized in Java, taking ~50 bytecode per usage. As seen on this blog post, it seems that Java just has a bad rng library. Luckily, we can use our own algorithm (xorshift):

```
public class Random {
    public static int state;
    public static int rand() throws Exception {
        //xorshift32
        state ^= state << 13;
        state ^= state >> 17;
        state ^= state << 15;
        return state&2147483647;
    }
}
```

XORshift pseudorandom number generator

Obviously, this random function is pretty bad, but it's good enough for battlecode purposes. If we don't care whether the result is positive or negative, the bitwise AND can also be removed to save 2 more bytecode. Here's the final comparison:

```
java.util.Random rand = new java.util.Random(seed:1000);

int a = Clock.getBytecodeNum();
rand.nextInt(bound:2147483647);
System.out.println((Clock.getBytecodeNum() - a)+" java");

a = Clock.getBytecodeNum();
Random.rand();
System.out.println((Clock.getBytecodeNum() - a)+" custom");
```

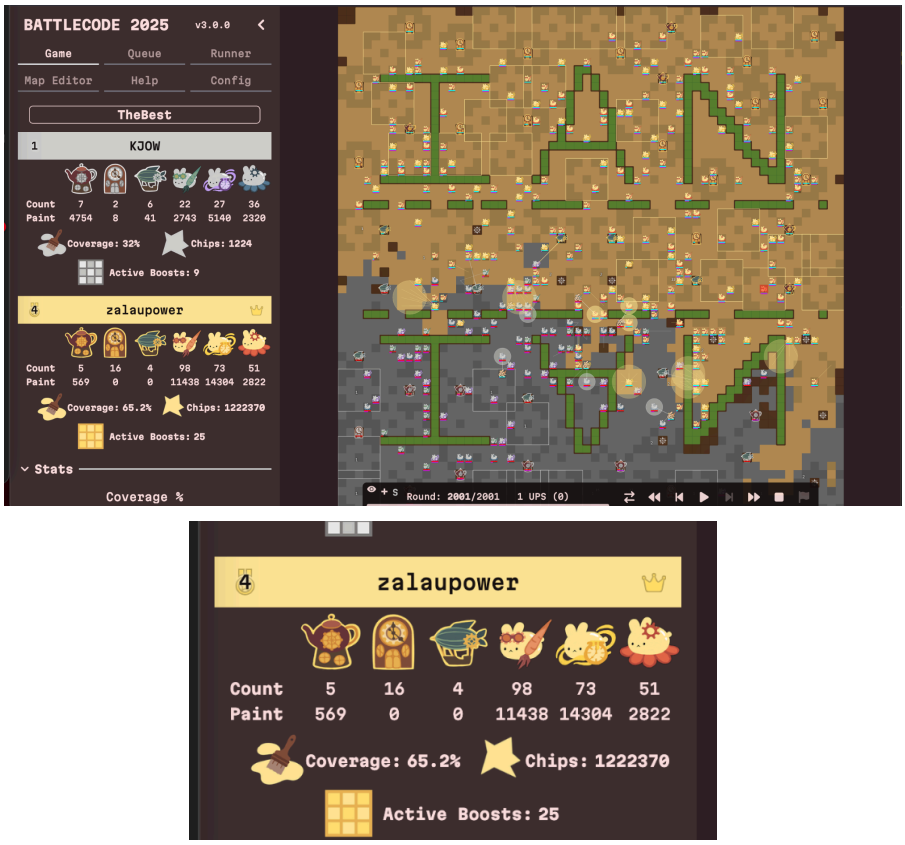
```
[Team A, ID #1, Round 1] 55 java
[Team A, ID #1, Round 1] 24 custom
```

Comparison of java.util.Random vs. custom XOR-shift random

Since bots typically use a crap-ton of rng, this is a very easy band-aid fix to reduce bytecoding.

Better disintegrating

In past years, rc.disintegrate() was a joke function that you only used for debugging, etc. but this year it becomes useful in certain situations, like this tournament game, where zalaupower almost threw by only having 5 paint towers. (Look at their chip count)



Top: zalaupower with a territory advantage; Bottom: zalaupower's unbalanced chip/paint towers

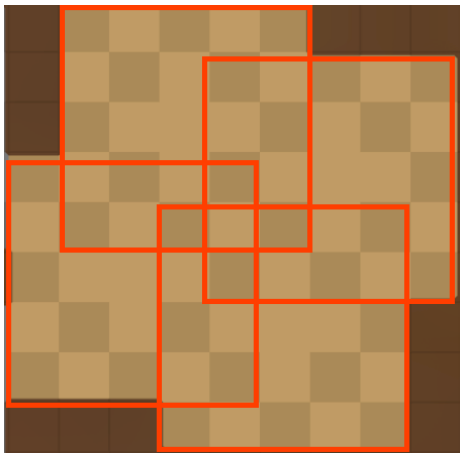
For money towers, if we have around 20k chips and we are still gaining chips, then we can probably disintegrate. The soldiers will then build a new paint tower (which costs 8500 chips) in its place.

For defense towers, if we haven't seen an enemy robot or paint in the last 30 turns, then we can probably disintegrate. The soldiers will then build a new paint/money tower.

Betterdisintegrating helps us dynamically change our towers based on the current resource situation, allowing us to convert money tower spam in early game to paint towers. It works best in a stalemate position, which is rarely ever achieved, but still cool…

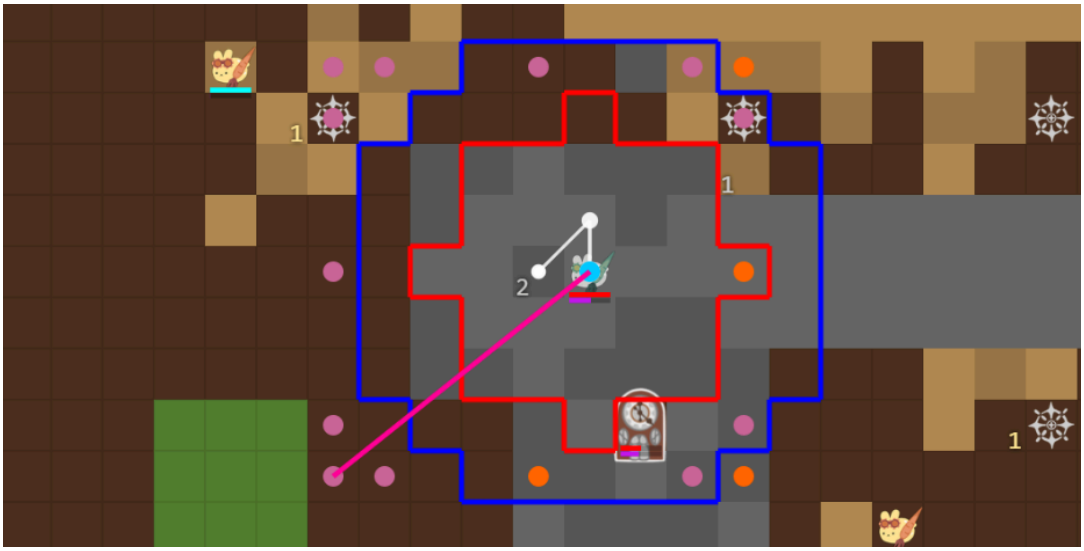
“Smart” SRP building

SRPs can be tiled to cram more SRPs into a smaller space. Prior to Sprint 1 balance changes, the optional SRP tiling pattern looked something like this:



Pre-Sprint 1 balance SRP tiling pattern

If tiled infinitely, we get 10 squares/srp, a significant improvement over the naive 25 squares/srp. This was unintended, and was too easy to code - most teams simply painted checkerboards by default and then placed a single primary paint wherever they could to complete the pattern.



Queued SRP locations - magenta: queued, orange: disqualified

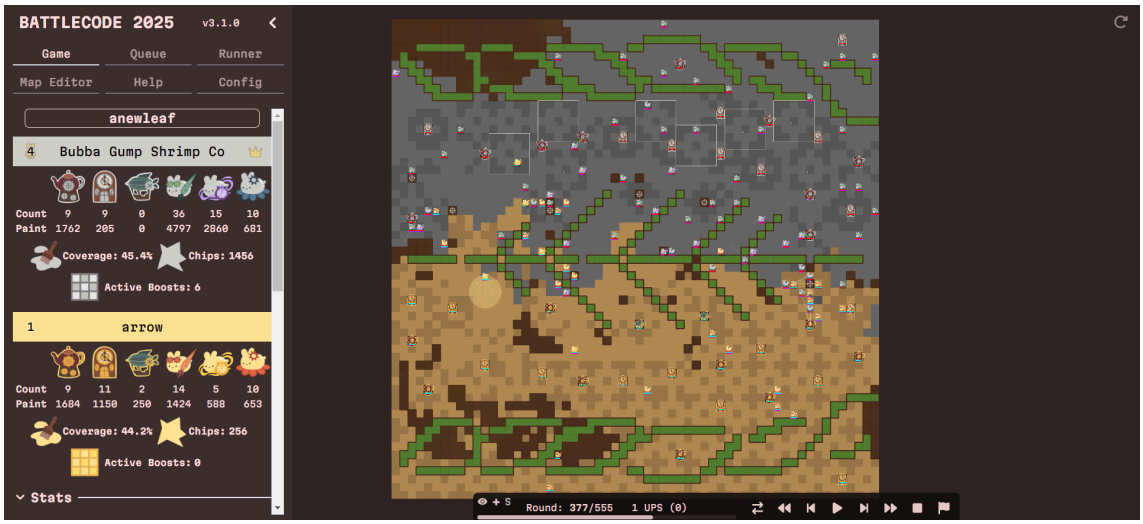
Soldiers would greedily try to place SRPs wherever they could:

1. Obviously must be a paintable 5x5 area
2. Must be clear of 5x5 area around unbuilt ruins to avoid self-obstruction
3. Must not interfere with other SRPs

Once finished building an SRP, it would queue up 4 “optimal” SRP tiling locations, and 12 less-optimal positions (but still more efficient than the naive tiling), shown by the magenta indicator dots. Soldiers would then visit these locations to build another SRP, but if at any point it

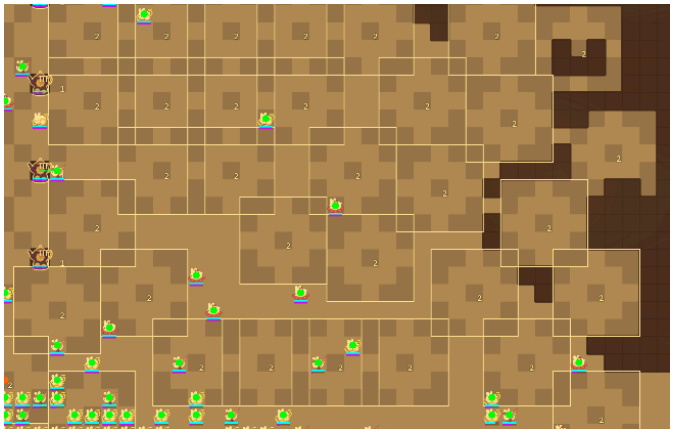
could see an obstruction, it would disqualify the location immediately and move on to the next, shown by the orange dots (because of the way ours is coded, some locations don't get disqualified because the bot is still checking locations before it in the queue).

This had an advantage over index-based SRP building, which many teams used.



arrow is unable to build any SRPs due to walls and ruins obstructing their index-based patterns

The smart method can fit SRPs in locations index-based SRPs would be obstructed in because of a single tile misalignment, giving it an advantage in more cluttered maps, but loses some efficiency on more open maps, where soldiers far apart would build SRPs that tiled on different grids. However, in tournaments, most of the maps have a good amount of walls and obstructions.



Inefficient SRP tiling on open maps

Comms (the code is in POI.java lol)

This year, each bot can send 1 message to a tower every turn as long as it is within sqrt(20) distance and is connected by ally paint. Towers can send up to 20 messages a turn. Each tower can also have the option to broadcast a message to all towers within sqrt(80) distance. As each message is a 32-bit integer, we split it up into two 16-bit integers so information can be sent faster.

Each 16 bit integer was either a tower/ruin message or a symmetry message. This would be determined by bits 13-15. If they were all 1, it would be a symmetry message, otherwise it was a tower/ruin message.

Tower/ruin messages:

Bits 1-12: Location, 6 bits for X and 6 bits for Y

Bits 13-15: Type and team. There are 7 options for this, ruin, ally paint, ally money, ally defense, opponent paint, opponent money, opponent defense.

Bit 16: Relay indicator. This bit was used for broadcasted messages. Broadcast messages have this bit set to 1, so towers know that this message should be relayed to other towers.

Symmetry messages:

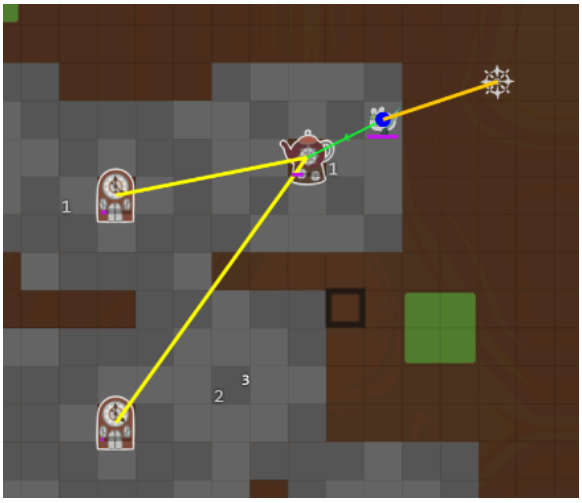
Bits 1-3: Ruled out symmetries (horizontal, vertical, rotational)

Bits 13-15: Symmetry message indicator, a message is a symmetry message if all these bits are 1.

Each bot had a 144 length array to store data for up to 144 towers/ruins (there can be a max of 144 ruins). Each bot would also have a 144 length array of StringBuilders, to store which bot IDs know about each tower. The purpose of the StringBuilders is to avoid towers spamming the same message to a bot 20 times.

The bots can then be informed of points of interest and is used for macrostrategy (e.g. “go to opponent tower” being most of the straitegy)

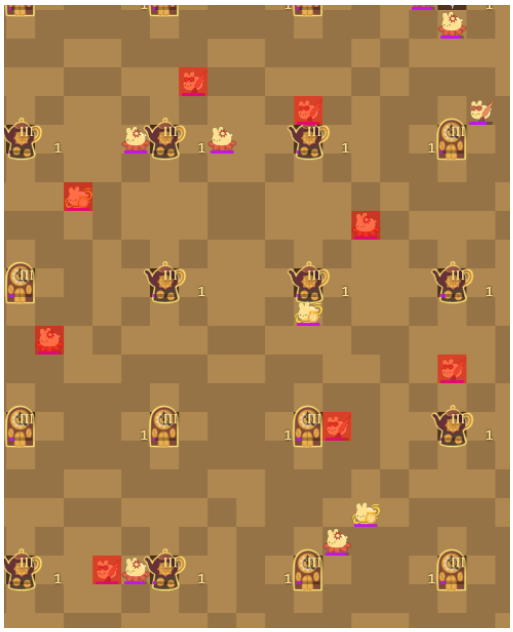
After sprint 1 added the ability for towers to broadcast messages, we also implemented this into our comms system. Towers will broadcast a message with data about themselves once they are created, and also every 100 turns after that. In a broadcast message, the relay indicator on bit 16 is set to 1, and if a tower receives a broadcasted message it will relay it to other towers.



The newly created tower uses broadcastMessage to send its info to nearby towers

The cool /5 comms trick

The sprint 1 comms code was very bytecode inefficient. Every time a bot received a message about a tower/ruin, it would have to search through all 144 indices of the array to find which index the tower is stored in. This is very bytecode costly and led to bots running out of bytecode after getting spammed with 20 messages from a tower.



Bots running out of bytecode every time they go near a new tower, due to reading messages

After sprint 1, we optimized the bytecode by creating a 12 by 12 array, which stores a map from tower/ruin locations to indices. Only a 12 by 12 array is needed because there can only be 1 ruin in each 5x5 square, so we can divide each coordinate of the location by 5. This map allows us to directly get the index of the tower in the array instead of having to loop through the entire 144 length array.

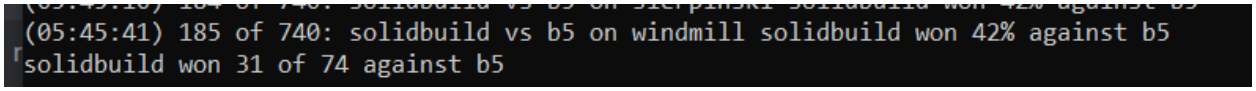
Lessons Learned

Copy other teams (but not always)

This is how we found out about the soldier building and splasher rush strategies, both of which carried us for some time. Because the game was well-balanced, there are drawbacks to adopting certain strategies, and soldier rush occasionally beat our bot in scrimmages, until after the sprint 2 balance changes which made it much worse.

Make a runmatches.py script and look at your scrims

Scrimmage analysis is still more important than raw win rate against old bots. Also note that running overnight scrims and hoping for a random improvement is actually very productive:



The best run Maitian got on January 23 (b5 is the new bot)

The best run Erik got on January 19 (I is the new bot)

Communicate well with your team

It is important to push any significant changes, even if you are still working on them, to ensure that your team knows what part of the code you are changing, and so you can stay updated with the current version.

One system of organization we used was tasks on Discord to let others know of the bugs or improvements we are making. We plan to keep and improve upon this for future years.

DON’T make last-minute changes

In both 2023 and 2024 we threw by submitting a last minute change that didn’t work. If you really want to make a last minute change, use a testing script and make sure it isn’t losing. Luckily, for the HS tournament we avoided a bug by using our testing script, giving us an error margin of 5 minutes.

Work on the right thing at the right time

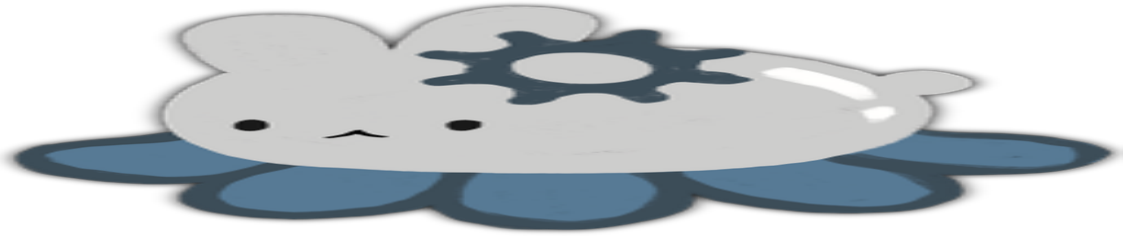
The macro is (probably) going to change many times over the course of the three weeks, so it’s best to start with things like navigation, micro, code organization, testing/auxiliary scripts, creating a bunch of testing maps, and other things that won’t change much. As the dust settles from Sprint 1 and the strategy becomes more well-defined (especially with balance changes), you can implement whatever macro the top teams are using and quickly rise to the top with a solider base.

Overall Thoughts

We thought that this year's game was very enjoyable, with many different strategies. This year felt a lot more macro focused, with new strategies coming out all the time. There wasn't a single end-all strategy that had no drawbacks, ensuring constant action as teams evolve.

Thanks to Pantheon for being an awesome team to hang around, The Realer Merlin for scaring the chips out of us, SPAARK for their runmatches.py script from 2024 (which was copied from 4 Musketeers' run_matches.py script from 2023 (which was copied from Producing Perfection's run_matches.py script from 2022)), camel_case for letting us copy their bugnav, wololo and cout for clout for teaching us the importance of micro, and everyone who submitted a bot. Thanks Locke for introducing us to the game, and also for being Locke. And finally, a huge thanks to Teh Devs for making a unique and genuinely fun game this year.

—SPAARK



Unused "mopper.png" found in the client source

Rickroll just for XSquare - never gonna give you up
Battlecode client with sound