Battlecode 2026 Postmortem

Nuaym Syed
Team: Lorem Ipsum
LinkedIn Repository
Last Updated: 2/13/2026

Contents

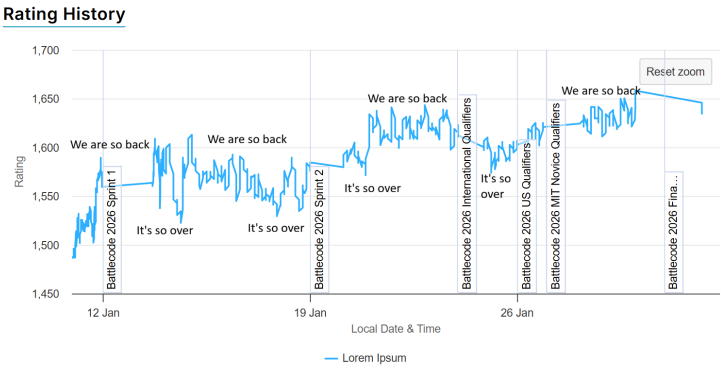

Figure 1: Most consistent rating graph

1 Introduction

Hi everyone! Thanks for coming to read this postmortem. This was my first year competing in Battlecode, but I think it might be valuable to document the journey over the 2026 Battlecode cycle.

A little about me:
I am currently a first year at the University of Washington in Seattle (major undeclared). Outside of programming, I love reading, chess, teaching, and my math homework (for the first and last 15 minutes I work on it). I heard of this competition because somebody had mentioned it in an off-topic chat from a nonprofit I work/volunteer at (Mustang Math) and was like "this looks cool, I'll probably do it this January."

I had a really fun time in this competition! My bot was "ok," qualifying for the final tournament at MIT via the U.S college qualifiers but unfortunately loosing it's second match at said tournament (by 3-2). My bot generally had hovered between 30-60th place overall (including International, High School, and Non-Competing teams), and I got lucky seeding at the U.S qualifiers. Throughout the entire of my code I do not believe I used any battlecode-specific optimizations (such as something called code-gen). I definitely recommend new competitors to read high ranking teams postmortems rather than mine, but I would like to offer my experience and thought process into the mix.

Also know that if I use the words "we" or "us," it's really just me ;_;

2 Game and Terminology

Most people reading this probably have some knowledge about this years game. If not, here are the main points:

- **Rat King:** This is a 3x3 game piece that can be controlled. These can spawn baby rats and do some rudimentary attacks. Each team starts with one, and looses if they no longer have any rat kings. A new rat king (to a maximum of 5) can be spawned if 7 baby rats assemble in a blob. Past round 1200 (out of 2000) you can only spawn if you have one rat king. Each rat king consumes 2 cheese per round. If not enough cheese in global stock, rat kings begin to loose health.
- **Baby Rats:** These are the main game piece. They can collect cheese from cheesemine and return to the king. They can also have rudimentary attack capability as well as a particular ability to pick up other baby rats and throw them.
- **Cats:** These are neutral game pieces that cannot be controlled. They can scratch or pounce. Pounce insta-kills any rat. They can here baby rat "squeaks" (see communication) and if they do they may "chase" your rat.
- **Cheese mines:** These spawn cheese, the main resource in the game.
- **Traps:** Traps cost cheese to place. Can be either for enemy rats or for cats. Cat traps can only be placed for 100 round after backstab mode is turned on (if you did not initiate the backstab)
- **Dirt:** Dirt is a removable wall that costs cheese to dig. Can be places as well.

- **Communication:** Global array of 64 integer values between 0-1023. We also had local ability for rats to "squeak" in a radius of 4, at most once per turn, to transfer a 32 bit integer to other rats.

The game has two modes. It starts in cooperation mode, where the win condition is points that heavily favors damage done to cats. If a team attacks another team, it triggers backstab mode, which has points heavily favoring the number of living rat kings. If all rat kings die it is an automatic lost. Generally very few games ended in cooperation mode. For more information there exists the game specs.
There are also something called online scrimmages (I will call these scrims) where we can face other competitors before the competition. There is a elo-system that determines your seed in competitions, determined by ranked matches. Ranked matches happened automatically once every 1-3 hours, or you can turn in auto-accept ranked so teams lower rated than you can automatically play ranked games (I had this off). There also are unranked scrims (which I had auto-accept on for).

3 Development Timeline

v0 Survivor (Pre-Sprint 1) → **v1** Fighter (Sprint 1) → **v2** Prospector (Sprint 2) → **v3** Maybe (US Qualifiers) → **v4** Unknown (Final Tournament)

4 Initial Strategy/Information

The game was released around 4-5 o'clock to where I live. Essentially that night I had designed my first iteration of a strategy. It was so cleverly documented in a word document, but I accidentally forgot to save it. So this information is essentially what I remember and what was programmed in V0.

4.1 Communication

We essentially had 640 bits in the global array. From this I had allotted it to be something along these lines:

- First 60 bits should be used to send location info of the (up to 5) rat kings
- Conceptually maybe rat kings can control "army points" where rats can "gather" to form either new rat kings or lead an attack against an enemy. This was never implemented.
- Subsequent bits should be used to transfer information about mines, rat kings, and cats

4.2 Baby rats

My original idea was to split the baby rats into "jobs," which are listed below:

- Soldier - These either circled near our existing enemy king to actively tried to attack enemy kings.
- Miner - The main goal for these was to focus on collecting cheese and going back to the rat king as quick as possible.

- Scout - These are intended to find new mines as soon as possible as well as find enemy rat kings.

This system was eventually removed in V2.

4.3 Misc

Some other things:

- Before the engine update where you cannot spawn another rat king if you already have two after round 1200, the main goal was just to spawn as many rat kings as possible in the last 200 rounds
- Because of how overpowered cats had seemed, rats should just "run away" from cats (this was changed eventually).
- I didn't think much about combat with other rats (which in hindsight I should have), but mostly the idea was to just attack whenever (which I later found out was a bad strategy).

5 V0 - Survivor (Pre-Sprint Bot)

This was my first bot, it's intention being to simply just go around and beat the doNothingBot. The time frame for this was somewhere around the night the game was announced to the morning of the day before scrims opened (Saturday morning for me).

5.1 Changes from initial strategy.

This bot was essentially my initial strategy, however there was a couple things changed and omitted:

- This bot had the ability to support more than one rat king. However, this bot does not have a spawning functionality.
- This bot did not make use of traps or anything similar.
- This bot had no combat ability (for cats and against other baby rats).
- This bot only had support for sharing data regarding mines (nothing about cats or enemy rat kings). However, the infrastructure for receiving such signals were implemented (but never used).

5.2 FSM (Finite State Machine)

All versions of my bot was a large Finite State Machine (FSM), which just means it switched between states. The Baby rats had 5 states for the bot at this point in time:

- Scout - Moves around to a random edge and tries to find something
- Mine - Goes into this state when it sees a cheese and goes to try to mine it
- Return - Enters this state when it has raw cheese and is heading back to the rat king
- Cat - Enters this state when it sees a cat (runs away)

- Report - Enters this state when the bot has a report (saw a mine and wants to go back to the king)

The reason we had separate states for return and report was that the return state could get "distracted" in more mines or whatever. The report state does not get distracted.
Essentially the only difference between scouts and miners (the jobs) was that miners would go to the nearest mine if it knows of one, while scouts continue to try to explore.

5.3 Misc. Strategy Choices

Some other miscellaneous strategy choices were:

- We spawn 20 rats in the initial phase. Afterwards we spawn up until rat costs exceeded 30.
- We originally chose to have 20% scouts and 80% miners.
- Because it was my first time in battlecode, I didn't know much about path finding. So I used a vibecoded version of bugnav.
- Each turn the bots would scan the area around them and update an internal map (this was then used to determine whether another bot is blocking their way or if it was a wall or dirt).
- Communication was limited to the shared array and bots squeeking to the rat king. We followed the procedure listed in the initial strategy.

5.4 Implementation Specifics

Some implementation specifics:

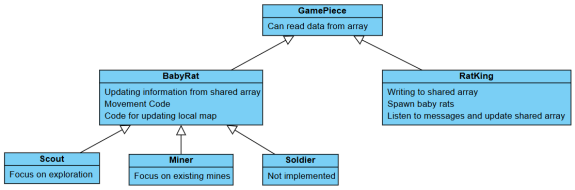

Figure 2: Enter Caption

- We developed a hierarchy class structure for our code. It is described in the diagram.
- Every time a new rat is spawned we "randomly" pick to be either a miner or a scout (random double).
- By our (very bad) code design, each of the three BabyRat classes had their own versions of each state (which later was really hard to manage).

6 V1 - Fighter (Sprint 1 Bot)

Like the name suggests, this bot actually had combat capability. It was also the Sprint 1 submission, as well as the first bot to hit the scrims.

6.1 Major Changes

Major changes/additions:

- A (vibecoded) attack code. This can attack, as well as ratnap enemy rats. Due to it being vibecoded, it had a lot of issues (most of which stemming from the fact that it only considers throwing at places it can actually see).
- Added an attack state (for obvious reasons)
- Soldiers!
  - Depending on a constant for aggressiveness (our final submission had it set to 0) the bot will either go towards where it thinks the enemy rat king is (opposite side) or just spin around the main rat king for defense.
- Added a coefficient to spawn new rats (so we need at least four times the current rat cost to spawn a new rat)
- Basic rat king spawning (but they essentially just spawn whenever it can and it's slightly later in the game, and despite that they still think they're baby rats, and issue not fixed until later).
- From what we noticed from scrims, strategies where we place dirt, rattraps, or cat traps seemed to be pretty effective.
- Rat kings can move away from danger!
- The bot now sets traps. Generally traps are only set by the rat kings, when dodging an enemy rat or a cat.

6.2 Scrims, Sprint 1 Performance, and takeaways.

So overall my first iteration of the bot did it's job to some extent. successfully beat most rush bots (bots that just rush to attack the king). Here are some issues I had noticed in scrims and Sprint 1:

- Despite me programming code that prevents bots from targeting cheesemines and cheese across walls, I forgot to implement that same feature for enemy rats
- All my bots magically "died" all the time
- Many of the time we are only capped at around 20 rats (because the spawning wasn't that aggressive) which had lead to a lot of wasted cheese
- We usually didn't get absolutely cooked because of cats, but not having cat damage was taken a toll in some cases in scrims.
- We often ran out of cheese despite having plenty rats to go and collect cheese.

All of these features eventually led to the significantly better Sprint 2 bot.

7 V2 - Prospector (Sprint 2 Bot)

When I explain the changes to this bot, the name would make a lot of sense.

7.1 Major Changes

This had essentially three MAJOR strategy fixes:

- Regarding spawning - inspired from our goat XSquare we made it so we adjust the maximum rat costs based off of the number of known mines, and excess cheese.
- Regarding GamePieces - Like I said I found out that much of the time my rats kept dying (which are expensive to replace). So instead I made the decision to remove scouts and soldiers, which significantly increased the survivability of my bots (since the scouts and soldiers often got in 1v3 situations). Obviously I shifted scouting capabilities to miners.
- Regarding lack of cheese - This may be surprising, but did you know the more cheese you collect the more cheese you get? Originally I had rats pick up a maximum of 40 cheese. After my scrimmages with Podemice, I realized that much of the time the movement cooldown from collecting too much cheese is not significant, so I was able to move it to 100 and significant gains were received.

As you can see, the name prospector comes from the significant improvement in cheese mining!

7.2 Misc. Improvements

Some other things:

- Switched the already vibecoded pathfinding to another vibecoded Pathfinding with a one step BFS.
- We added joint attacking, so bots starts squeaking the location, type, and the current round regarding the most important enemy near them (usually just the nearest, but rat king take priority). We tests implementations where they were just always squeaking, but usually then we get cooked by cats (who can hear and pounce on squeaks), so we ended up removing these.
- Apparently sometime in the middle of this the old anti-rush code was removed, so I put it back in (imagine my surprise when my bot all of a sudden was getting absolutely destroyed by some of my rush testing bots).
  - For those wondering, The anti-rush code was essentially most just spawning more rats when in danger of the king getting rushed. It may seem trivial, but I found this to be sufficient against all rush bots.

7.3 Sprint 2, Scrims, and Performance

Despite my bot becoming significantly stronger this time around, my bot went from seed 36 in Sprint 1 to seed 55 in Sprint 2. The bot did generally well up until it played against The Complex Merlin in it's third game (which later went on to get second place in the high school tournament). Here are some things noticed:

- Much of the time we just spam rats willy- given money which worked really well up till it didn't. I found many other teams only spawn rats when they find a mine or something similar

- Our bots seemed to only focus on a couple cheesemines and because they don't explore enough they loose potential on a lot of cheese.
- Our vibecoded attack micro since Sprint 1 (has not changed at all) got destroyed in almost all head to head combat scenarios (there was one edgecase where sometimes we did better in maps like Thunderdome where the rat kings are right next to each other).
- Our pathfinding seemed to suck (nothing much about this).

All of these were attempted to be addressed in my US qualifier bot.

8 V3 - Maybe (US Qualifier Bot)

It was nearing midterm season so I needed to come up with a ton of fixes fast. So, I had a lot of cool strategy implementations in this bot.

8.1 Major Changes

- Something we realized embarrassingly late was that when a new rat king was created, it always thought of itself as the first rat king, so we always got scenarios where the two rat kings are essentially fighting each other for the same data in the shared array.
  - The reason for this was because I was too lazy to transfer data, I just deleted the old data and created a new Rat King object when a rat becomes a rat king. The issue with this is that the new rat king doesn't have the old data regarding which rat kings were alive and dead. This lead to all of them taking the first spot. So we fixed it pretty simply by keeping the data.
- Improved Rat King information on the global array (see below)
- Revamped baby rat spawning! (see below)
- Better Rat King Navigation - Number one is that the rat kings has a very trivial bugnav implementation so they can go around walls!
- Revamped attack system (see below).

8.2 New Rat King Communications

As a refresher, we originally used 12 bits per rat king in the global array to send information regarding location (two 6 bit integers for x,y). The issue that arose from this is that originally I had no checkers to count the number of rat kings (only create a new one if that slot is empty), so we sometimes got a ton of dead kings polluting the array. (As something funny, because of the previous issue this actually didn't lead to much a problem since they all just casually used the same 12 bits so it never mattered).

- My solution to this was to revamp Rat Kings. Now, each rat king uses 14 bits in the array, First 12 tells us the location like usual, but now there are two more bits. One of these bits are used to convey a sense of "criticality". This just means if the Rat King is low on health (we set it to under 50). If we only have one rat king in criticality, then the rats are supposed to try to create a new one. The other one was simply a "counter." Essentially if a babyrat notices

this bit flipped from last turn, then the rat king is alive, else it is dead. This generally was very stable.

- Critical rat king creation essentially worked like this: If you sensed at least 7 friend near you (either directly or via squeaks) then we squeak once to summon them together to form a rat king. Notice this only works if the rats were constantly squeaking (which we were forced to do).
- There is an edge case where if a rat king exceeds bytecode one turn, and it doesn't get to flip it's bit, a rat can mark it as dead. Usually this is not an issue since next turn it'll magically "revive," but there is an edge case where if a rat king is created in the same turn, since it considers the existing rat king dead, it sets that as it's ID, and then that leads to two rat kings fighting for the same array. I never noticed this in scrims or tournaments, and only know about this by talking to TSPAARK.

8.3 New Spawning Logic

For the first 100 rounds of the game, we had a static spawning implementation where we spawn up to 20 rats. Something very funny is that because of poor if-else code writing... this virtually never ran so we never actually got the startup phase.

We also added an infinite build limit. This was added right before the qualifier (and was super aggressive), but essentially above 1200 cheese we just build infinitely. This lead to some beautifully flat graphs on open maps, along with up to 80 rats, but obviously might have taken a toll on closed maps where conserving cheese is the play. What my testing noticed is that games on closed maps generally were ones I was already going to loose, so I made the (risky) decision to implement this with hopes that it would give me a significant edge on open maps.

8.4 Revamped Attack System

I completely changed the (previously vibecoded) attack system. Now, rats have something called "memory." Each memory is implemented via a `Memory` class, which have fields `robotLoc`, `type`, `friend`, `round` respectively. It also had an `isStale` method to check if the memory is "stale" (old). The idea is each time a bot comes to know of an enemy (either via it's own vision cone or a squeak from another rat) if there is enough room in it's memory, it add that memory, storing it's location, the type of bot (cat, baby rat, rat king), whether it is a friend or foe (this was necessary to avoid throwing at friends), and the round number it head of. Cats, baby-rats, and rat kings all individually had their own timeout, where if a memory was older than that we consider it an empty memory (for example, Cats had 5 rounds, rats had 0 (has to be this round), kings have 10).

- There is a certain tolerance for baby rats, as in a babyrat will not chase an enemy if that enemy is significantly higher health than it, or is relatively far away from it. Baby rats automatically target enemy rat kings.
- Each tick, if we can one-shot a rat, we attack it. Else, if we notice we have a good throwing direction, we pick up the rat and throw it.
- To determine if there is a good throwing direction, we just loop through the memory and determine if we throw a rat in that direction, will it hit. We prioritize hitting enemy rat kings and enemy cats, else we just throw it at an enemy baby rat (or we were supposed to, I later realized we might've been throwing it at our own as well because I forgot the check).

- This implementation is very flawed, and was more fleshed out in our final bot, but just adding this increased our win rat to above 85% in local scrims, and while there was less of a benefit in online scrims, it still did show up.
- Note that as of now we do not have any toll at throwing at our own rat kings or rats, or any checks for throwing at walls (added later).
- Another thing is that adding this (especially without loop unrolling and codegen) this was pretty bytecode heavy. This forced me to remove the one-step BFS the rats had, but it didn't significantly decrease pathfinding ability.

8.5 US Qualifier Performance

I entered qualifiers at seed 14. Like expected, my bot seemed to do well on open maps with plenty of cheese with less influence from the opposing side, whereas they generally did not do well in situations where there was head to head combat (specifically with teams higher rated than me).

We got somewhat lucky seeding, where our qualifying match (for the final tournament) was against "Anyone Can Cook" (seed 12), which we had scrimmed quite a lot with. Overall, their bot often beat ours on more combat designed maps, where we would consistently beat them in larger open maps with a lot of cheese. We got lucky in said qualifying match (with 5 games per match), 3 of which were maps where there was less interaction, which came to our advantage giving us the win, and qualifying us for the final tournament at MIT. This team specifically was one of my go-to testing team in the scrims before quals, so I would like to give them a shoutout!

Something very particular I had noticed was that on some of the maze maps, my bots objectively just get stuck in circles and cannot appear to pathfind back to the king. The cause of this was very trivial and is discussed more in the next section.

9 V4 - Unknown (Final Tournament Bot)

The biggest regret I have is that because of all my midterms piling up, I was not able to spend enough time as I should have on this bot. So, I mostly had quality of life improvements and bug fixed into this bot.

9.1 Major Changes

- Fixed rat path finding
  - So hilariously, the reason why my rats sucked at many maze maps was how I implemented movement. So essentially, each time a target changed, the code was supposed to reset bugging so it could conceivably re-evaluate if it should bug. The thing is, let's say a rat was going to the king, and the king moved. I never wrote any code determining the difference for if a King moved or if a new target was given. So the bot thinks that it is a new target, and because of that they reset bugging, which caused the bots to essentially not pathfind to the king when returning (since my king was very mobile much of the time).
  - My solution to this was rather than keep track if the king moved, I just said each time the target changed, if it changed less than a certain distance it would not reset. The ben-

efit is that it covers the case of the rat king moving (since the rat king cannot teleport) as well as covers cases where let's say the rat king is right next to the mine it wants to go to, or nearby.

- Increased combat abilities (see below)
- Something we put off for a while - Rat kings can now dig to place rats, so if it is surrounded by dirt at the beginning of the game it can still place rats.
- We changed "startup phase" so that it stops when we discover our first mine.
- Rat kings prioritize moving to mines that are not near existing rat kings. This spread them out more evenly increasing cheese on open maps.
- We "added" a feature preventing rats from going into 1v3s. The issue is that the way it was implemented made it actually worse for the bot (will talk about this in a moment).
- We accidentally worsened the bots by stopping it from squeaking every turn (more on this later).

9.2 Further Combat Additions

Some quality of life additions:

- Switched our system from "throw at first good thing" to a weighted system, where we have weights for throwing at rat kings, cats, and other baby rats, as well as negative weights for throwing at our rat kings and ally rats.
- Added ability to throw at walls
- Switched throw-and-hit detection to a ray-system with a bit of "wiggle room." This is more accurate than the previous implementation which just used the dir given from battlecode's built in `directionTo` function.
- Instead of only picking up if we know we have a good throw, we just pick up regardless. This is since we can just carry the rat for a while up until we get a good throw direction. If the 10-round timeout occurs for carries, then we just throw at the best direction we have.

9.3 Final Tournament Performance

This bot was seeded 15/16 in the final tournament at MIT. It lost it's first game to second seed General Stoke's Theorem (second place), and the one after it faced 10th seed, Food. We had a close game against food, unfortunately loosing 3v2 at the end of the day (in a co-op map the cats jumped over to the other side so we lost to cat damage).

Despite it's lackluster performance, I did analyze the games and realize significant bugs:

- A lot of my bot dependent on being able to know where it's allies are (from rat king creation to other determining if there are enough allies nearby). So... by toning down the squeaking I broke half the bot.
- The better pathfinding was amazing since the bots actually transferred cheese now! The issue is that my old bot ironically relied on the clumping of rats to spawn rat kings, so when that issue was fixed... less rat kings spawned.

10 Some Misc. information

10.1 Testing & Tuning

For testing and tuning I really only had two things to go off of, the client runner and performance in online scrims. At some point I had tried implementing a (vibecoded) tuner to tune constants, but after realizing it's too slow on my system and doesn't do well I never ended up using it. The code for said tuner is in the repository if you would like to see it, though in my testing it did not work.

My process looked like this:

1. First, test it locally against older bots on a select variety of maps (usually one to three). These are maps where I would see the new feature/fix reflect the most, so generally the new bot should outperform the old one on these maps. If there are any noticeable discrepancies fix.
2. After it works for a select few maps, test it against the old bot(s) on the entire variety of maps (by the end of the month I had around 7 custom maps, ranging from lots of cats/cheese to very closed maps, as well as provided maps from the Sprints. If the bot clearly under performs (lets say it gets around 30% winrate against the old one) most likely something is wrong, so we investigate and fix. Even if the bot appears to perform roughly slightly above 50% we accept it (which is pretty low).
3. When we send to online scrims, we then start scrimming a lot of teams within our rating range. If the new update appears to be performing worse than usual, then we usually pull back and attempt to fix it.

Outside of the day I tried using the "tuner," I only really used the client runner and online scrims to test. I generally focused on analyzing macro and micro strategies rather than the overall score (hence the low threshold) since often minor changes do not significantly increase bot's performance.

10.2 Bytecode

This is a game where you have to write efficient algorithms given a certain computing limit (called bytecode). Many teams consistently build to keep their usage under the limit. Honestly I hardly had that issue, and when it did occur I just lowered the loop bounds in order to make everything work.

Many other competitors at my rating and above had a feature called code-gen. I recommend taking a look at their postmortems on the Battlecode website once they get uploaded. The idea is we "abuse" bytecode limitations with some of the following:

- Switch statements rather than for loops - Switch statements use less bytecode than a for loop.
- Strings and String builders rather than arrays - String manipulation generally is often less bytecode intensive than their array counterparts (despite being slower overall).
- Some other funny things that I actually do not know about.

I generally didn't use any of these "code-gen" optimizations. In doing so much of my algorithms were limited, but end of the day what matters more is the algorithm itself than how efficient it is,

since many teams used codegen who did not perform well.

10.3 AI

Battlecode permits AI to be used in it's development. So, essentially for large parts of my code for earlier bots, and decently sized chunks of code in my final bot was in fact written by AI. The only reason why is because I am a single person and I didn't have the time necessary to invest into building a lot of infrastructure and code from scratch.

I want to cement that all strategy decisions were made by me. All debugging was by me (include debugging AI code).
Here is what part of the code *was* written by AI:

- Pathfinding - 100% debugged by me but since I didn't have any infrastructure most of my pathfinding was written by AI. However I did make a lot of edits to it.
- Ratnapping up until V3 - We covered this earlier, the reason being I didn't have time to code any combat related code at that time.
- Very trivial non-battlecode related algorithms. Because of the lack of time some of my helper functions like `canGoStraight` (which checks to see if something is across a wall or not) were written by an AI (which uses the in built map). Generally these are more common and simpler to implement so AI was able to do these with ease.
- Communication - More specifically bit masking and reading. This was something common in many places outside of battlecode so it made sense. Choices regarding what, when, or how to communicate was made by me, AI simply wrote the encoding and decoding.

11 Conclusion & Reflection

So what's next? Once I get some free time back from all my schoolwork and other responsibilities I will probably release a V5 - Post bot, where I can further tune things and practice on making some algorithms more efficient for next year:

- Fix the squeaking breaking-changes.
- Most likely implement some version of Om-Nom's BFS + BugnNav implementation of pathfinding (which is known to be very powerful)
- Consider revamping mapping system to be more efficient and more useful.
- My bot was super messy and most likely not the most efficient, where we had all the states essentially reside in one file. Best to start splitting up modules.
- Finite state machines (like my bot) are cool and work! However there are a number of alternatives which also could be valuable to consider. I plan to test these alternatives out and see which ones are viable.

Thank you to **Teh Devs** for making this a great year! I hope to see all of you again next year!