Cout for Clout - Battlecode 2024 Strategy Guide

Daniel Qiu, Johnny Liu, Max Wang, David Wei

February 2024

Table Of Contents

Table Of Contents 1
Team Introduction 2
Overall Timeline 3
Strategy Development 3
- Package name: MacroPath 3
- Package name: HotlineBling 3
- Package name: BobTheBuilder 4
- Package name: WaxtheBuilder 4
- Package name: WaxingMoon (Final Submission) 5
  - Attack Micro 5
  - Defense 5
- The Tournament 6

Strategy Thoughts 8
- Attacker/Healer/Chicken Micro - XSquare is Always Right 8
  - General Ideas 9
  - Chicken Micro 9
  - Healer Micro 9
  - Attack Micro 9
- Communication 9
  - Message Queue - 4 Musketeers is always shooting 10
  - One sitting duck - Chenyx is always watching 10
  - Flag Passing - Wololo is always Wolo-ing 11
  - Other 11
- Placing Traps 11
- Filling Water 12
- Flag Sniping 13

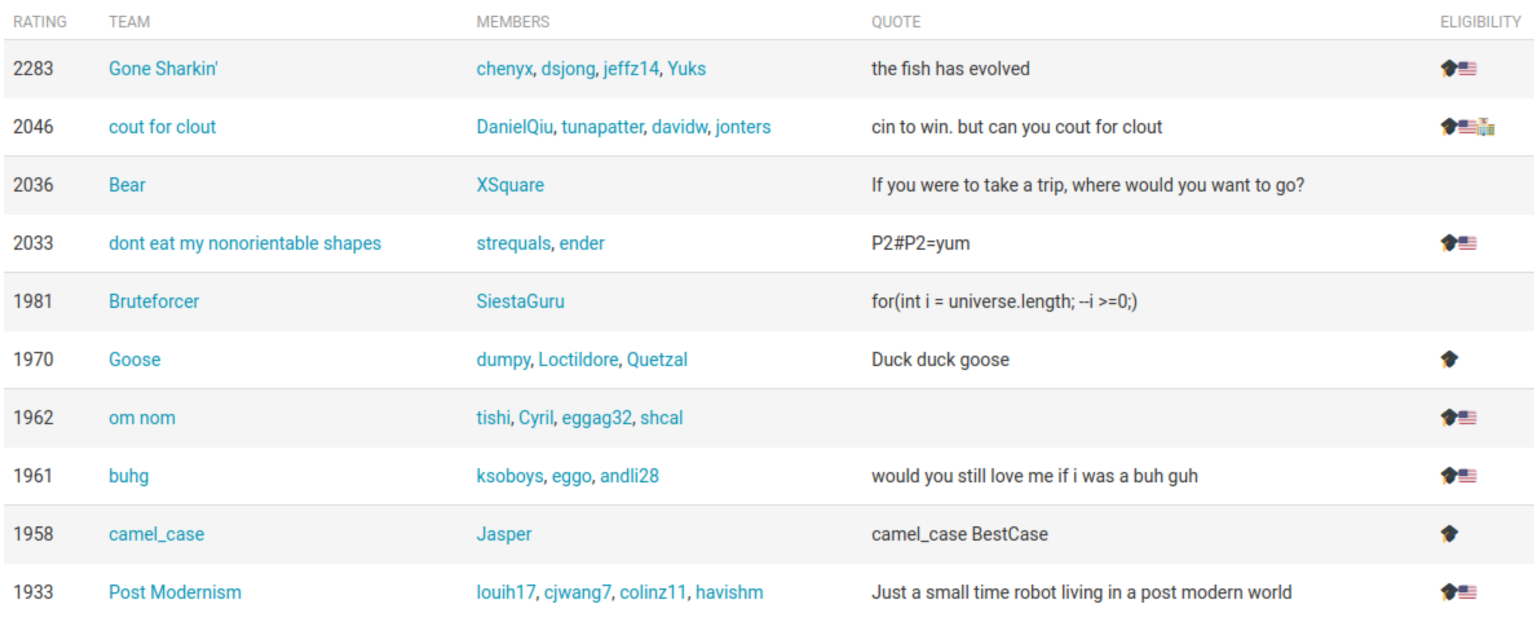
Overall Thoughts 13

Team Introduction

We're Cout for Clout, senior year high schoolers from Thomas Jefferson High School for Science and Technology. Since we were named Cin to Win last year, we decided to continue the C++ rhymes and name ourselves the opposite. This is Johnny and Daniel's third year doing battlecode, Max's second year doing battlecode, and David's first. We placed 1st in the high school division, and ended up ranking 2nd on the final combined leaderboard (including college students and post-graduates).

Our Code:
https://github.com/ChiMasterBing/battlecode24

Final Submission:
https://github.com/ChiMasterBing/battlecode24/tree/main/src/waxingmoon



RATING	TEAM	MEMBERS	QUOTE	ELIGIBILITY
2283	Gone Sharkin'	chenyx, dsjong, jeffz14, Yuks	the fish has evolved	🎓🇺🇸
2046	cout for clout	DanielQiu, tunapatter, davidw, jonters	cin to win. but can you cout for clout	🎓🇺🇸🏫
2036	Bear	XSquare	If you were to take a trip, where would you want to go?	
2033	dont eat my nonorientable shapes	strequals, ender	P2#P2=yum	🎓🇺🇸
1981	Bruteforcer	SiestaGuru	for(int i = universe.length; --i >=0;)	
1970	Goose	dumpy, Loctildore, Quetzal	Duck duck goose	🎓
1962	om nom	tishi, Cyril, eggag32, shcal		🎓🇺🇸
1961	buhg	ksoboys, eggo, andli28	would you still love me if i was a buh guh	🎓🇺🇸
1958	camel_case	Jasper	camel_case BestCase	🎓
1933	Post Modernism	louih17, cjwang7, colinz11, havishm	Just a small time robot living in a post modern world	🎓🇺🇸

Figure 1: rankings post-final submission deadline

Overall Timeline

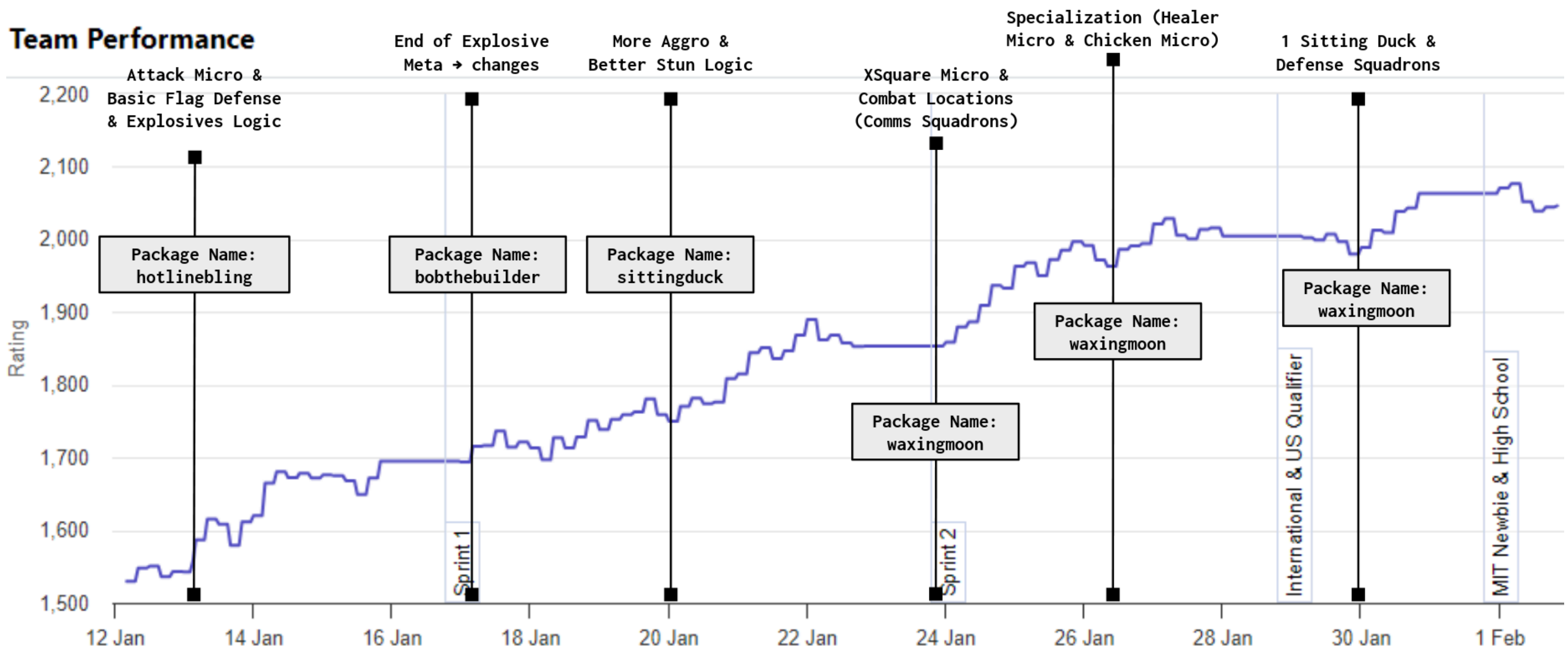


Figure 2: A timeline along with package names for our submissions. (Graph Credit: camel_case)

Strategy Development

Package name: MacroPath

Stand on the shoulders of giants. -Probably XSquare

Our first act was to rip off all of the basic functionality from previous code. We stole the symmetry and structure from our 2023 code and put in bug-find from 4 Musketeers, ignoring their global obstacle calculations. Finally, we wrote basic attack micro and established basic macro strategy.

Package name: HotlineBling

I know when that hotline bling, that can only mean one thing. -Drake

Here, we added the functionality for ducks to call each other on their cell phone. Specifically, we wanted to stop enemies from stealing our flag. A crude distress system was put in, which made ducks mark the closest spawn to where the flag was being stolen. Jailed ducks would then spawn there. We also put in general communication plumbing, such as communicating friendly spawn locations and tagging each one of our ducks with their exact move order number.

Package name: BobTheBuilder

BOB THE BUILDER, CAN HE BUILD IT??? BOB THE BUILDER, NO HE CAN'T!!! -Not Bob the Builder

We saw how a lot of top teams, including buhg and Super Cow Powers, managed to quickly level up to get 3 level 6 builders and build traps for 50% of the cost. Because we idolize every single top team, we decided to try it as well. Although this strategy allowed us to place more traps, these traps were much less efficient, and it only did better against a few teams, so we got rid of it.

However, even though we don't use level 6 builders, traps are still important. We made the first of many changes in trap logic. See Placing Traps for full details.

Package name: SittingDuck

There's no sitting duck. Offense is the best defense -Sun Tzu Art of War

Our defense sucks. SuperCowPowers and other teams have pioneered the "sitting duck" strategy, so we decided to try it. Turns out, using 3 ducks to sit on your spawn makes you lose just about every micro battle. By doing a simple test of 47 vs 50 ducks, we found that the version with 47 lost 90% of the games, and it wasn't even close. Thus, we gave up on sitting ducks. Instead we decided to be even less defensive and go full aggro – making our micro as aggressive as possible. This improved our rating a whopping 130 points (from 1720 to 1850).

Having aggro micro was not enough, we need to be more aggressive! So, we put in a simple system for ducks to spawn at the location of closest combat. Also, we put in better early game, spawning more ducks closer to your opponent, and calculating symmetry better by exploiting the properties of spawn locations. This change helped our ducks get to the fight way quicker and improved our rating a decent bit.

Package name: WaxtheBuilder

WAX THE BUILDER, CAN HE BUILD IT??? WAX THE BUILDER, NO HE CAN'T!!! -We Aren't Very Creative

We ended up making micro a bit more passive and trying the builder strategy again — we hoped that our more passive micro would help. It ended up still not working well, and we got rid of it again and never put it back after seeing that Gone Sharkin' did not use it.

We also developed a new Comms system. With these Comms systems, we were able to extend the idea of combat locations beyond just "spawning." For example, we would have code that summons friendly ducks to combat locations, locations that we determined would have a lot of combat happening there.

We also tried to make our bot use water bombs and healers to build up levels. This ended up not working well, but it helped us develop better healing micro which led to (somewhat) improved wall combat.

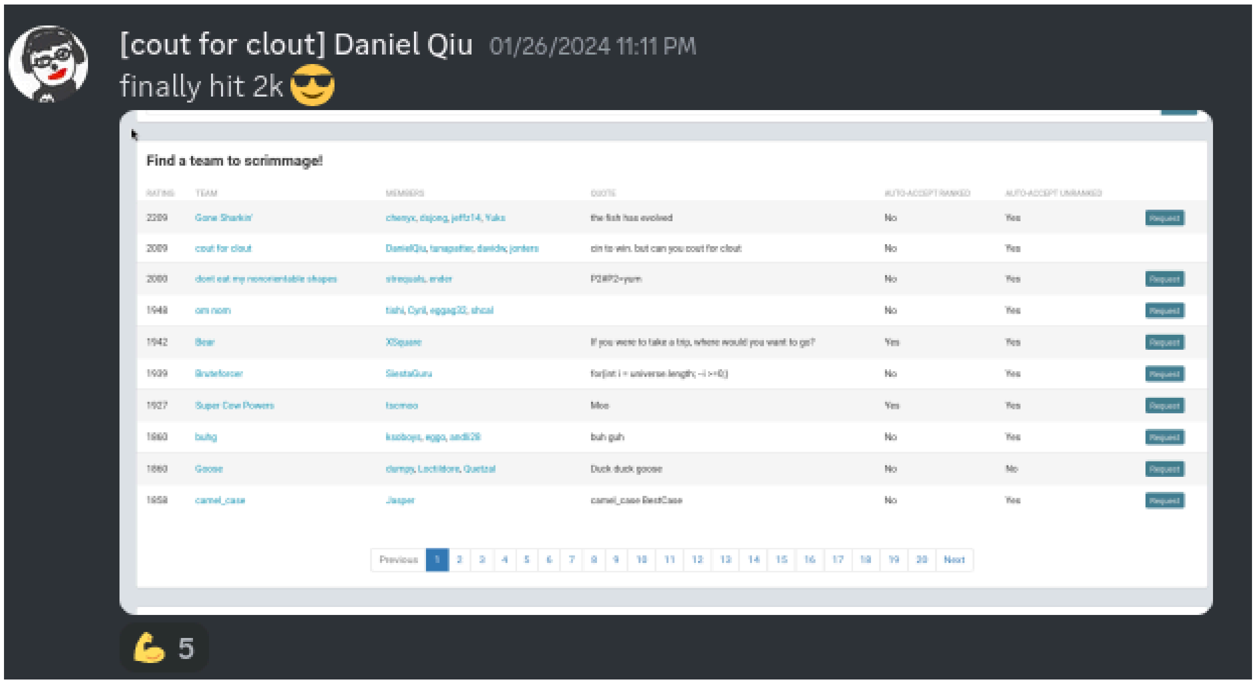
Package name: WaxingMoon (Final Submission)

Attack Micro

床前明月光 疑是地上霜 举头望明月 低头思故乡 *-Li Bai*

One night, as we gazed upon the waxing moon, we struck a brilliant idea. All the top teams seemed to have XSquare's attack micro, and our attack micro seemed to be holding us back. We seemed to do OK against all the teams except XSquare, whose secret micro formula demolished us over and over again. So we did the one logical thing that anyone would do: rob XSquare's micro and turn it against him. As the generous open-source man he is, it took approximately 20 minutes to steal his micro from his 2022 github.

After this improvement, we shot up from the 1850s to 1950s. Our high rating motivated us, and as we worked day and night on bug fixes and improving attack, we finally broke the barrier to 2000. See Attack Micro section for more information.



[cout for clout] Daniel Qiu 01/26/2024 11:11 PM

finally hit 2k 😎

Find a team to scrimmage!

RATING	TEAM	MEMBERS	QUOTE	AUTO-ACCEPT RANKED	AUTO-ACCEPT UNRANKED	
2209	Gone Sharkin'	chenyx, dsjong, jeffz14, Yuks	the fish has evolved	No	Yes	Request
2089	cout for clout	DanielQiu, tanapatter, davide, jonters	cin to win. but can you cout for clout	No	Yes	
2080	dont eat my nonorientable shapes	streqeals, ender	P2#P2=yum	No	Yes	Request
1948	om nom	tishi, Cyril, eggag33, shcal		No	Yes	Request
1942	Bear	XSquare	If you were to take a trip, where would you want to go?	Yes	Yes	Request
1939	Bruteforcer	SiestaGuru	for(int i = universe.length; --i >=0;)	No	Yes	Request
1927	Super Cow Powers	tscmoo	Moo	Yes	Yes	Request
1860	bufig	kuoboys, eggo, andi28	buh guh	No	Yes	Request
1860	Goose	dampy, Lochtore, Quetzal	Duck duck goose	No	No	Request
1858	camel_case	Jasper	camel_case BestCase	No	Yes	Request

Previous 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 Next

💪 5

Figure 3: 2k!!!!

Defense

Sometimes full moons turn bad dreams into horrible nightmares -Plants vs. Zombies: Heroes.

Over 2000 rating, second on the overall leaderboard, and over 150 rating points above the second highest high school team, we could just chill now…. Or could we?

One night, as we gazed upon a full moon we realized that a 1600 rated player (Immortal Jellyfish) had beaten us. Soon many other 1600s and 1700s would beat us, sniping our flags as we tried to battle them head-on. Thus began our quest of defense. Using a new Comms system that we developed, we worked on adding defense, and adding a few tricks like flag passing. See Communication.

The Tournament

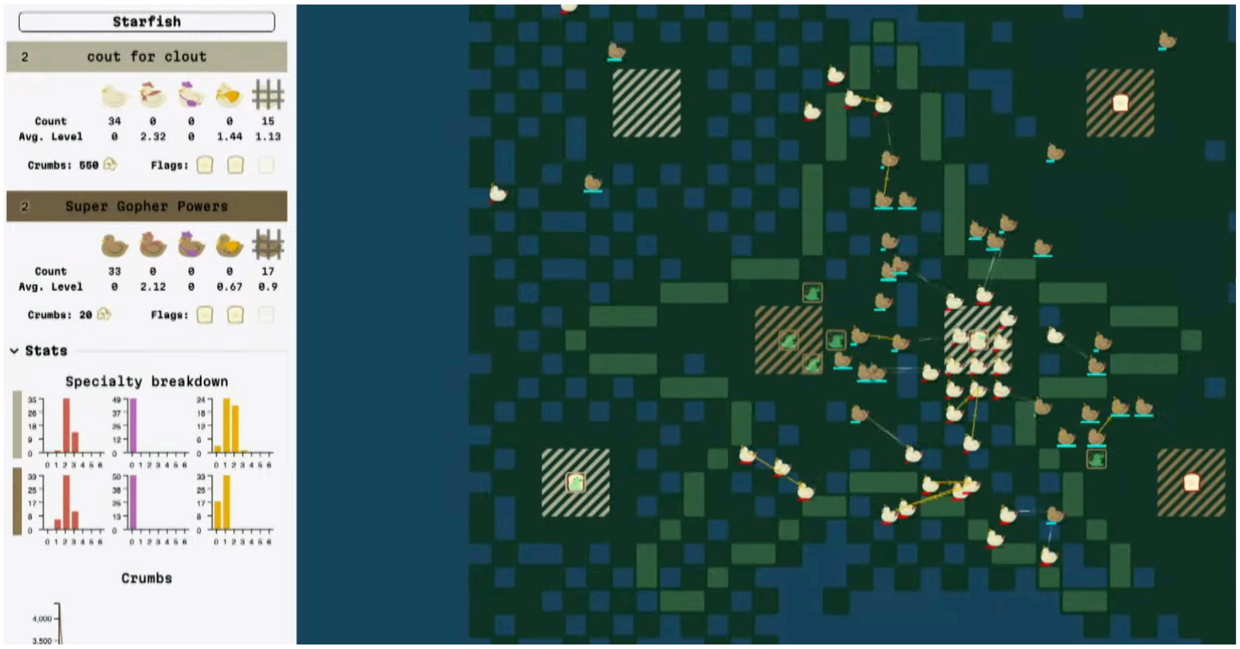
The qualifier round for high schoolers occurred and we were one of the top 2 high school teams invited. Knowing that we were almost 200 points above the other high school team, and having not lost a single game, we thought we were chilling. Little did we know that we were in for a roller coaster ride.

We arrived at the hotel Friday night, eager to meet Battlecoders. Maybe one of us was a bit too eager, proceeding to ask random people in the hotel if they were Chenyx, the legend from Gone Fishin'/Sharkin'. Luckily, Chenyx is a kind man, and offered to meet us down at the hotel lobby, saving us from too much embarrassment. We talked about Battlecode strategies for a bit, and then went back to our rooms for a long day tomorrow.

After exploring a bit of Cambridge the next day, we met up with other teams: om nom, dont eat my nonorientable shapes, pat, buhg, Gone Sharkin, hadoop, wololo, Post Modernism, DIABLOSIS:Naga. Thanks to everyone who talked to us, it was very interesting hearing about everyone's strategies and stories :D.

The final high school tournament begun, and the quick win we thought would happen turned into a stressful nightmare, as we traded games one by one. Sometimes we would win due to better micro, but sometimes our less aggressive micro would just be pushed so far back to spawn that we ended up blocking our own spawn, and suffocating.

The score was 2-2, with a nail biting final match. Gopher (brown)'s ducks were surrounding our spawn, thus forcing our ducks onto our own spawn, and blocking our own spawn. Since our ducks prioritize spawning from spawn zones with a lot of enemy ducks, our ducks would soon be overwhelmed quickly and we would have lost.



Starfish

2 cout for clout

Count	34	0	0	0	15
Avg. Level	0	2.32	0	1.44	1.13

Crumbs: 550 Flags:

2 Super Gopher Powers

Count	33	0	0	0	17
Avg. Level	0	2.12	0	0.67	0.9

Crumbs: 20 Flags:

Stats

Figure 4: Brown (Gopher) is surrounding us (white) and has a completely winning position

However, a few brown ducks somehow made their way to our bottom and top left spawn zones, which activated defense at the bottom right, and freeing our ducks from being sandwiched by the opponent.

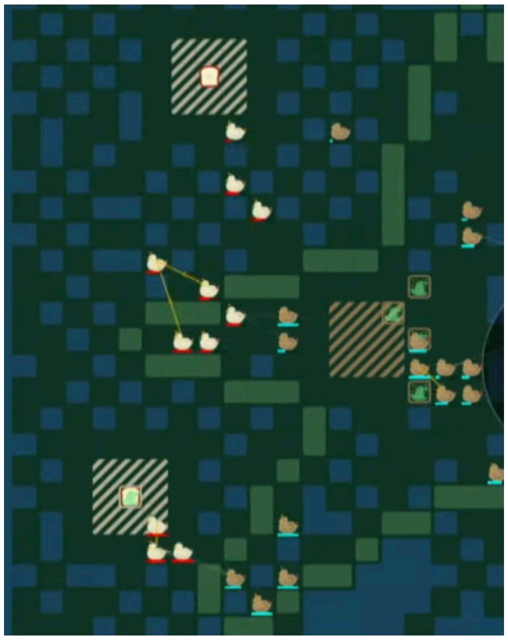


Figure 5: Our defenses are activated, and we achieve a better position

Then the game just simplified into a micro battle, which we won.

Strategy Thoughts

Attacker/Healer/Chicken Micro - XSquare is Always Right

Throughout every package we spent a lot of time improving and modifying the attacker micro. We started off with a bad attacker micro, using lots of if statements and a combat BFS to move. The if statements would decide whether to push or retreat and the BFS would move towards the closest friendly spawn or the enemy to achieve that. Outside of pathfinding, this BFS had two main features: try to avoid diagonals and try to avoid water. However, our attacker micro ended up getting too complicated to modify, and a pain in the neck due to the unpredictable nature of our BFS.

We ended up using XSquare's attack micro strategy, which was to look at any direction a robot could go, and for any direction, have a heuristic to weigh how good a square was. This simple idea and clean framework proved to be a crucial advantage, leading us to dominate the top leaderboards.

Along with XSquare's basic micro layout, here are a few ideas that we implemented:

General Ideas

- Use X-Square style micro!!!
- Bucket behavior based on cooldowns and ranges
- Move towards lower health allies
- Move towards lower health enemies
- Move towards allies that were stunned
- Move backwards if the ally behind you has more health than you
- Move towards a combat location (inspired by Lanchester's laws and Wololo's lecture)
- Lvl 4+ Healers mainly heal, Lvl 4+ attackers mainly attack
    - Consider attacker DPS increase: 128 (lvl 3) → 170 → 218 → 420 (lvl 6) [insane]
- Don't be too clumped up (or else micro becomes bad because you can't move anywhere)
- More healers, because they were easier to level up, and could protect attackers from dying.

Chicken Micro

- Happens if health<=450+(Max(attackLvl, healLvl)-3)*150
- Avoid enemy troops if too low on health
- Turn into healer, and try to mainly heal

Healer Micro

- Happens if rc.getID()%3<2 and HealLvl>4
- Try to avoid enemy troops and move towards ally troops

Attack Micro

- Attack!!!
- If cooldown<20, let opponent move-hit you because it is not necessarily a bad thing, and allows you to not waste any turns if your opponent is stunned

Communication

This year, communicating between ducks was much simpler, as ducks could access and modify the shared array at any time. Due to this ease-of-use, we were able to utilize communications for purposes not limited to the following:

- Summoning ducks for attack or defense
- Ally and enemy flag information
- Map info and pathfinding
- Flag passing
- Setup

Message Queue - 4 Musketeers is always shooting

We built off of 4 Musketeers' buffer pool framework, adding many of our own structures on top. We allocated sections of the shared array for different purposes. Many of these sections (e.g. ally status information) would be parsed and updated every turn.

However, there were some messages that would not be sent as often, and of which there were too many types to allocate array space for. We implemented a queue for these messages, as it was the most efficient way for bots to read new messages and write their own. This queue system does not require the clearing of slots (saving 75 bytecode per write operation), and both reading/writing happen on the same turn.

This message queue is used for our squadron system, which summons ducks to combat locations. The reason for this is Lanchester's square Law, adding one duck increases the power of a large army by a lot, but a small army by a little. Thus, we try to send ducks to combat locations so they are more close together, rather than just sending them straight to the target.

In addition to tracking combat fronts, squadrons doubles up to serve as our defense system. Specifically, we implemented special criteria to call a squadron message of top priority when our flag is being picked up. This causes nearby ducks to swarm the enemy stealing our flag.

One sitting duck - Chenyx is always watching

However, we still could get sniped, as we left no duck sitting on our bread. If any enemy duck approached while no friendly ducks were there, we would get robbed. We were complaining about this in VC when Chenyx suggested a genius idea, to keep track of whether your flag was under attack using the rc.canSpawn() command. This can directly be exploited to detect if there's a duck on your spawn pad. By keeping one duck permanently unspawned, and having other ducks communicate if they are stepping on the spawnpad, we can "magically" spawn reinforcements as soon as an enemy steps on our spawn pad. This is much more efficient than sitting duck as it only immobilizes one duck, as opposed to three. By Lanchester's Law, this 2 duck difference cannot be overlooked.

Flag Passing - Wololo is always Wolo-ing

After watching some replays against Wololo, we also added flag passing, which helped our troops get the flag to its destination much faster.

There are two main advantages to flag passing:

1. The movement cooldown for holding a flag is 20. However, a duck could drop a flag a distance sqrt(2) from its position, and a duck that is a distance sqrt(2) from the dropped flag position can pick it up. Thus, in two turns, flag passing allows for a maximum travel distance of 2sqrt(2) as opposed to the normal sqrt(2) distance.
2. A large weakness of bugNav is dealing with multiple robots, leading the flag carrier to be blocked by other friendly ducks. Flag passing significantly reduces this issue since the flag holder would not need to bugNav through all the ducks; it can just pass the flag instead.

We allocate a few slots in Comms for the flag holder to assign closer robots to pick up the flag. The new robot is picked based on Manhattan distance to the closest spawn. Euclidean distance poses some problems as it's harder to measure the actual benefit of passing to the chosen robot. Ducks in the process of bug finding do not pass flags to avoid getting stuck. Also, accounting for the flag carrier's movement cooldown would be beneficial, but was not included in time for our final submission.

Other

- We communicate all friendly duck's relative move order at the beginning of the game
- We communicate the status and location of all enemy flags.
    - To retarget ducks faster after picking up a flag
    - To remove the imprecision of broadcast locations

Placing Traps

We claim that traps that are triggered quickly are generally better. In the pre-sprint-1 explosive trap meta, this was true as it would allow us to more quickly push back opponents. This effect is even more pronounced in the stun meta as you risk your robots not being there if the trap is triggered later on. Thus, in each possible build location, we count the number of enemy robots in radius 8. More robots would signify a higher chance that one of them would trigger our trap. This also causes traps to be placed in higher density areas, affecting more enemy ducks.

Another heuristic used to maximize the speed our traps would be triggered is to see if the trap-building location is between the enemy and your flag. If this is true, it is much more likely that they will walk on your trap.

The one nuance is preventing too many stun traps triggering at once on a single duck, as that's inefficient. At first, we tried a grid system to play traps at modulo 3 locations. However, this would result in sub-par trap placements in terms of trigger speed. The final scheme combined both of these ideas, where we use the trigger-speed heuristics, while choosing not to place traps if there's another one in an +-pattern.

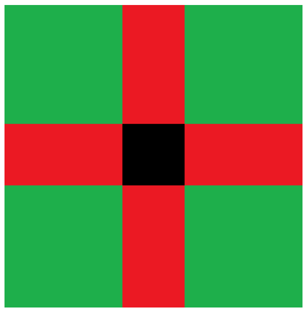
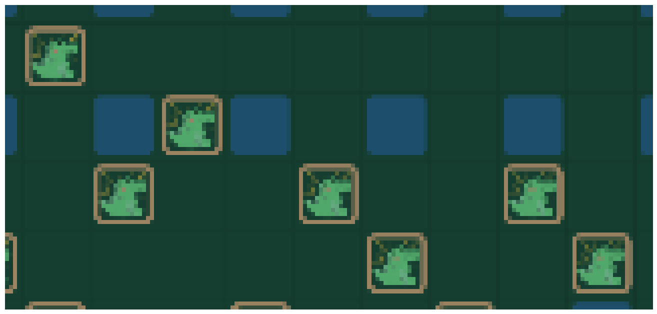



Figure 6 & 7: Trap placement

On the left, the red squares represent where a new trap cannot be placed. This generates the diagonal patterns seen on the right. These layered diagonals trap placements yield the following benefits:

- Only diagonal movements can cause multiple traps to detonate at once
    - This rarely happens as it requires enemy ducks to be perfectly placed, and other enemy ducks behind to not trigger the stun beforehand (essentially meaning that your opponent is sending in ducks alone)
- Once the first layer of traps is detonated, another step forward will detonate another layer of traps, creating consistent stun coverage

These heuristics are weighed against a threshold for building traps based on the amount of available crumbs.

Filling Water

We use a simple scheme that is on-par with most top teams. The most simple variant is filling in a checkerboard pattern when there's no enemies in action/vision range, which already suffices for top ladder scrimmages.

We modify this slightly to account for maps such as *racetrack*. Instead of filling in water at (x, y % 2 = 1), we look at how water is distributed. For each location that contains water and is within a distance of 2, we tally the number of impassable squares in North South East West, and we fill if that number is greater than 1. This usually generates a checkerboard as well, and is more robust against wacky water patterns.

Flag Sniping

We allocate 3 ducks to be snipers, whose sole purpose is to harass the opponent / sneak their flag out. They are sent to different broadcast locations in hopes of stumbling across an undefended flag. This scheme also yields the unintended benefit of more space being visited, and providing better defense calls.

Overall Thoughts

This year was very micro-heavy, so a lot of our time was spent on optimizing micro. Overall our Postmortem focused on the big picture, but the majority of our time was on the finer details of various micros. A lot of improvements are intuitive, not necessarily complicated, and not necessarily worthy of writing down here. What's important is to take the time to try those things and mess around. Most teams use the same ideas; it's the execution of those ideas that make most of the difference, only possible through careful analysis of scrimmages and your code. Finally, it's so helpful to be involved in the community. Whether it's Chenyx, or XSquare, or someone else, chances are talking to them will yield something useful.

Thanks to everyone who played for making battlecode so enjoyable, Super Gopher Powers for scaring the crumbs out of us, SPAARK for making us improve our defense, XSquare for making his micro public, 4 Musketeers for making their bugNav and Comms public, Wololo for his cool lecture, Gone Sharkin' for always being better than us, Jasper for always being there in VC, and all the people we met in person for being so cool to hang out with. And finally, a massive thank you to Teh Devs, especially Evan, for running Battlecode and making this all possible.

Battlecode 24, as always, was incredibly enjoyable. We'll definitely be competing next year, and we hope to interact with everyone there.

– Cout for Clout