Big Red Battlecode:
What we've learned from two years of competing

MIT Battlecode, an AI/programming tournament open for students from high school to grad school, is finally wrapping up its 2019 competition, and as Big Red Battlecode (Top 32: 2018, 8th: 2019), we've decided to unpack what we've learned, and how the competition has gone, for both 2019 and a bit of 2018 as well.

The competition itself already has an interesting premise: build an AI to compete in a real-time strategy (RTS) game against other AIs. Every iteration of Battlecode has differing specs, but many similarities within specs involve requiring AIs to **manage resources, constructing buildings and units,** and **fighting against enemy units**. At a fairly basic level, people who compete in Battlecode, which takes place over the month of January (during MIT's IAP) often have to constantly keep up with strategies, approaches, and changes in the specs as the devs get ahold of and adapt to the ever-changing metagame.

In 2018, after a long sprint, we were able to make it to the Top 32 teams within the competition: just one game away from being funded to watch the finals in person and meet both the devs and other finalists. In 2019, we were able to actually crack Top 16 and compete in the finalist tournament, securing 8th place overall. In the process, we've learned a lot about ourselves, about software engineering, and about writing a good bot fast.

But first, just to lay out some context, here's a tl;dr of the game specs for 2019

The Game (and apparently you've just lost it)

The game this year, Battlecode Crusade, was a fairly standard RTS game without **too** many new gimmicks. The two sides, Blue and Red, each start out with castles, and the main goal is to either destroy the enemy's castles, have a larger combined unit health, and have more resources (in that order).

Castles are able to use karbonite and fuel to produce four types of units: "pilgrims" (which can mine karbonite and fuel, and construct churches: structures that can also build units but are less important and have lower health than castles), "crusaders" (which have lots of HP and a short-range "laser" attack, similar to previous years), "prophets" (which are fragile but have a long-range attack, but are unable to attack at short ranges), and "preachers" (which have a short-range attack that deals damage in 3x3 square, rather than to a single enemy). Units can signal to each other or directly talk to the castle, but signaling, moving, mining, and attacking cost fuel to complete, and each team only had a small passive income of fuel to draw on. To force teams to manage computational complexity, every unit was given 100ms + 20ms per turn to execute their code: if units went over this capacity, they would "stall" and would have to wait until they had more time to execute

There were a few big changes from last year however:

- Karbonite and fuel were concentrated in depots around the map, which differed from 2018's competition (where there was a limited amount of karbonite on every tile of the first planet), but each resource depot had an infinite amount of the corresponding resource, and units who mined that resource would have to deposit it at a castle or church if it was to contribute to the total resource pool. Luckily, units didn't have to carry fuel for their own movements: those were directly deducted from the total pool.
- The code we wrote would be run on individual units under their own process, rather than being a controller for a large group of units a la 2018's Earth and Mars controllers. This was a reversion to the competition's tooling in 2017 and before, and since we started to compete in 2018 (where individual controllers, and thus entire armies, were able to share a large amount of information at no cost and centrally plan extremely well), this definitely threw us a bit off guard. Bots couldn't share vision and thus would have to (mostly) act based on local information, though every unit was given a copy of the initial map (including karbonite and fuel locations) and units could signal to each other (with costs based on the range that the signal would go through).
- Every unit worked by conducting some computation and then returning an "action:" whether to move, attack, give, mine, etc. In doing so, this prevented units from moving and attacking on the same turn, or from moving and giving on the same turn, and greatly changed the end-game viability of certain strategy types.
- Rather than being primarily Java-based (as prior competitions were), this game was run through the Node Package Manager, running every single bot in JavaScript. Teams who didn't want to use JavaScript to write their bot would have to **transpile** their code using the Battlecode infrastructure, and while it (mostly) worked alright, issues with the Java transpiler (as we stuck to using Java this year) before qualifiers caused a few hiccups for us in terms of submitting our updated bot to the scrimmage servers. Debugging was also somewhat of a pain: because code was run in Javascript rather than being run in Java, it was fairly likely that exceptions based on equating two *clearly not equal/differently typed* objects would be found within robot logs.

A brief history of the meta, and our strategy relative to it:

Looking at the above specs, we expected that the game would primarily be focused on tense battles between individual resource points before one side would out-macro the other. Unfortunately, at the very start of the competition, this wasn't really the case:

Pre-Sprint: The rise of Volatile, and the rock-paper-scissors meta:
Early on, preachers were so amazingly powerful that simply building a few preachers and having them rush directly to the enemy castle (which was easy to find, as maps reflected everything including the castles) would overwhelm any attempt to stop them.

Crusaders were useless, as preachers could move to the enemy castle before enemy castles could produce enough crusaders to overwhelm them. Prophets were useless: preachers were tanky enough that, by running fast enough, they could take a few shots from prophets before moving close enough to prevent prophets from shooting and brutally destroy them.
Unit clumping gave preacher rushes easy targets to destroy en masse, and so even early game defensive preachers were not as optimal as rushing preachers: one-by-one streams of preachers could deal enough damage to devastate the one tightly-clumped group of preachers that a defensive player could dish out.

We also saw the rise of a more interesting equilibrium between three separate strategies. As players started to wise up to the problems of AoE preachers, there were three distinct strategies that (mostly) ignored economy and focused on a short early-game: either teams could rush with preachers, rush with prophets (as Delucia's Delinquents had done), or defend with preachers, and each strategy had distinct advantages and drawbacks. Rushing with preachers meant that the first-attack advantage laid with defending with preachers, but rushing with prophets ensured that defending preachers would have no chance to even sight the prophets before getting destroyed, and rushing with preachers against a prophet rush mostly worked because precisely-controlled preachers could close the gap against prophets and destroy them before getting destroyed themselves.

Volatile, in our eyes, rose to the top of the heap. Though ranked scrimmages were unfortunately not available, we found that Volatile was able to make the most optimized preacher rush, and in doing so, we were able to develop a lot of our own "turtle" strategy (build out a ton of units near the base, and win on unit health) by comparing it as a counter to Volatile's rush. That being said, we didn't perform as well in the Sprint tournament.

Since our bot was fairly anti-rush, being built to spawn preachers for defense, and since the dominant meta at the time was a preacher rush (which was especially helpful in the early-game with splash damage and higher HP), it was hard for us to get far. The sprint tournament had 4 winners on account of having multiple byes, and so primarily rush bots reached the top of the tournament bracket.

Pre-Seeding: Spec changes and the "waiting game" meta:

After the spec changes (where church and castle health were buffed and castles were given an attack), the game moved from being rush-prone to becoming a lot more defensive. We altered our earlier defensive strategy to produce a large mass of prophets while also quickly producing pilgrims to gather fuel and karbonite depots around the map, and creating a "prophet ball" became the norm within the metagame. The clear superiority of attacking over moving as an action, the sheer range of prophets over preachers, and the increased amount of resources and time that defensive bots had due to structure buffs made viable macro a long-term plan. While earlier rushes still worked if individual bots were too greedy (i.e. too focused on pilgrims over defense), the era of the competitive rush was over.

This caused some interesting implications. Many teams that had been working on this strategy for a while had already built prototypes of macro bots early on, making the new meta shift fairly drastic: the two “tiers” of teams ended up being those who were writing macro bots from the beginning (or could slightly alter their code to become a macr obot) and those who had to pick up the macro strategy on the fly.

We eventually moved to creating a “prophet tentacle,” a large stream (3-4 units wide) of prophets which could essentially fire over each other in order to slowly overwhelm the enemy, and as the maps were all reflections, we found that it was easy to quickly communicate and recompute the locations of each individual castle. However, our strategy itself focused more on applying continuous pressure while maintaining a strong defense, and this had mixed results against the current meta: a “lattice” structure of prophets with spaces between each of them to enable other units to move throw the lattice easily. In some cases, our tentacle could apply enough power to destroy the lattice, but in others (depending on how big the lattice was before we started attacking), we were unable to overcome the attack-over-move advantage that disincentivized offense and encouraged more static gameplay just because the enemy lattice was so big.

In our case, though, the seeding tournament went fairly well for us, but in the process, the seeding tournament took the place of ranked scrimmages. Because of our success and the success of other lattice and eco-prophet rush teams within the meta, more and more teams started to adopt this strategy and develop their own variants, creating a metagame that was a lot more stable and a lot less prone to fast shifts.

Pre-Qualifiers: Settling into a comfortable metagame:
Now, with the metagame having focused mostly on a macro-based setup, the main difference between bots was whether to focus more on economy or more on building units (either for attack or defense). There was a distinct Nash equilibrium here: while most heavy econ-bots were evenly matched against each other and most unit-based bots were fairly matched, the dichotomy between bots that built units and bots that more overwhelmingly focused on economy was crucial to securing victory. Many fairly high-ranked teams had switched over to a very econ-heavy strategy, ironically making their bots more vulnerable to the rushes that seemed all-but-dead after the spec change.

A lot of good teams, including Oak’s Last Disciple, fell due to this issue: either your bot was unlucky enough to focus too much on economy against an aggressive bot on a small, resource-poor map, or your bot was unlucky enough to focus too much on attacking while the enemy could comfortably build up a defense and hold out. If teams didn’t anticipate the rise of now-rogue strategies such as rushes, they were removed fairly fast, and the teams that qualified for finals mostly focused on trying to balance unit production and resource gathering as long as possible.
Pre-Finals & Finals: “This is the endgame”

The good part about the pre-finals stretch of the game was that, for the most part, the meta quickly shrunk to have a single overarching strategy rather than two or three (i.e. focusing on just econ with defense rather than too many pure econ or too many rush bots being built), and so a lot of teams had leeway to experiment with small boosts that helped them edge out other teams in the finals. Containment strategies, endgame crusader spamming (to win on unit health), and a focus on pressuring and bullying enemies out of resource depots, became more of a priority (we implemented crusader defenses and preacher raids in order to quickly clear out nearby depots of enemies), but a lot of the improvements were mostly castle-directed. I'll briefly detail a few of them below:

- *The Prophet Wall:* **Team Barcode** used this to his advantage in the qualifiers and early in the finals, focusing on building prophets to create a (nearly impenetrable) 2-3 unit deep wall of prophets separating his side of the map from the enemies. It was quite hard to stop unless the enemy bot started early enough.
- *The Prophet Waves:* **Big Red Battlecode** changed its "tentacle" strategy to one which synchronizes the prophets to move in a perpendicular line towards the enemy in (fairly discrete) waves of prophets. In doing so, we could bring more prophets to bear on a given enemy before individual prophets were killed, and if the entire wave moved at once, not enough prophets would be killed (if they were moving into enemy defenses) before they could fire back.
- *Church Lightning:* **Knights of Cowmelot** used this high-cost gamble to devastating effect in the finals. The main idea is to break a lattice (which has gaps to enable unit movement) by having a pilgrim build a "church chain" (i.e. a chain of church-pilgrim-church-pilgrim-church-pilgrim-church-(attacking unit, typically a preacher)) in order to ambush attack a castle. This aptly used the turn timer (as newly completed units had the next turn and could thus go right after the preceding unit, and so the church lightning chain steps could be completed almost instantly given enough resources. Church lightning very easily broke stalemates that focused on winning by unit health because this technique (named such because it was created to destroy the castle and enemy units would fire it down almost right after) was able to allow teams to win by having more castles (even if complete castle annihilation wasn't possible).
- *Sparse vs. Dense Lattices:* **smite** adapted this to effectively win Battlecode. The biggest issue with the metagame was how much to focus resources on economy vs. focusing on units and defense. The given lattice which smite focused on started as a "sparse" lattice (i.e. low on units, but still robust enough to give time for reinforcement and cheap enough to enable a solid early economy), and coalesced into a "dense" lattice as soon as the individual units saw enemies. While this strategy was somewhat vulnerable to random strikes of church lightning, it was mostly robust if enemies revealed themselves and lattices condensed before church lightning could commence.

The Origin of Big Red Battlecode, and our own lessons for other teams:

In 2018, during our freshman year, I was already following Battlecode for some time, and with a few friends from Cornell's notoriously hard variant of its popular "DSA with Java" class, CS 2112, we decided to compete in Battlecode 2018, and eventually Battlecode 2019 after our earlier success, under the name Big Red Battlecode. It definitely helped that we could devote a lot of our time early on to the competition: as Cornell's winter break was roughly 5 weeks (covering almost all of January), we didn't have to worry about schoolwork during the most formative period of Battlecode (pre-seeding), and we could focus entirely on polishing the fundamentals of our bot. For the most part, the name didn't just come from Cornell's "Big Red" theme, but more specifically from the in-college "cafe currency" of Cornell: "Big Red Bucks:" since the acronym for our team was the same as Big Red Bucks and "Be Right Back," our main joke for the description was some variant of 'BRB using BRBs instead of working on BRB."

WIth winter break going on, we got the specs and got to work, learning the following lessons along the way (this is both for 2018 and 2019, though 2019 was mostly us taking these lessons and iterating on them fast)

Lesson 1: Build a working bot before writing a complex one (and build one fast)

Stuartj summarized this idea pretty well in his 2017 post-mortem, but the rule still applies here. Focusing on building a working bot, and creating a strategy that you can fully understand and that applies a single rule, or a small set of rules, really well, is definitely helpful in the early competition. The goal isn't to make an initial bot that's *perfect,* but an initial bot that's *improvable,* and as long as a team can do that fast and keep up with the current meta (undeveloped and setup for obsolescence as it might be), they're going to be put at a distinct advantage as it becomes easier and easier to adapt to changes in strategies and find bugs and issues.
As the above post mentioned, while the Sprint tournament is mostly about bragging rights and isn't necessarily as high-stakes compared to other tournaments (and so taking an "L" on sprint is definitely not the end of the world, teams should be motivated to try and conquer Sprint and scrimmage as much as possible. Getting a bot running before most other people do is never a bad thing, and you can always adapt as the meta and the general strategy changes.

Lesson 2: Create a solid infrastructure

One of the big things to learn from Battlecode overall is just "being a good software engineer." Make life easier on yourself by making multiple classes, isolating individual units of functionality into their own functions, documenting everything you do, using version control properly, handling exceptions as much as possible, etc. (the list goes on). Not only is it easier to understand your code as you write, but it's also easier to incrementally and modularly improve specific pieces of your code (pathing, combat micro, macro strategies) or flip strategies completely with a meta change. This made everything, from designing complex strategies to polishing and bugfixing bots on the fly, much faster, which is especially important since you only have a month to write an amazing bot.

Lesson 3: Scrimmage and test constantly

Constantly, constantly do it. It's really the only way to know for sure if your bot is working as expected, if there are no bugs with either your strategy or code, or if the metagame is long past you by then. It pains me to say this (as the main theorycrafter of this team), but theorycrafting *can only get you so far.* The weakness of aggression really became apparent after a few scrimmages, and we'd often run a lot of scrimmages right before and right after major tournaments to sample the meta and understand how our own bot stacked up to other bots.

Lesson 4: Reverse-engineer competitor bots

This seems pretty obvious, but it's *vital* to constantly track the meta and understand what competitors are doing. Reverse-engineering competitor bots (or at least approximating the strategy to some degree based on scrimmage data) makes it easier to benchmark your own bots against the competition. In our case, before the sprint tournament (and for a little while afterwards), we reproduced Volatile's bot and ran our bot against it in order to better build out the defensive economy strategy that helped us in spades later on. It seems weird considering Lesson 3's statement, but even though scrimmages are great, *scrimmage data can also only get you so far*. The website might be down, you might not have seeds that help your bot or you understand what's going on, or the team might not be on auto-accept. In any case, to make sure that your bot is the best it can be, it's *necessary* to reproduce other bots (it also helps with becoming a better programmer in general: building a complex system from observations and test cases alone is like playing an intricate song by ear, and this sort of skill pays off in spades in the long run).

Lesson 5: Get ready to change fast

The competition is fast-paced and frenetic. A lot of the previous lessons kind of build up on this one: the metagame can change in a moment's notice. Whether it be the devs changing the specs to buff certain units or nerf others (especially post-sprint), or the competitors discussing, testing, and discovering new strategies (such as church lightning), the only real constant in this game is relentless change. Not adapting is often the death-knell of any team in terms of competition, so get ready to change big cornerstones of your strategies fast. This is why good code design is quite important: modularizing everything allows you to put some of the best components of your old bots easily into your new ones without requiring hundreds of hours of testing and integration.

Lesson 5.5: Be a generally nice/helpful member

One of the best parts of Battlecode is the community: it's not often that in a coding competition, many of the participants and competitors actually *talk* to each other about the intricacies of the projects they work on even while competition is going on, and newer people can more easily

pick the brains of more experienced teams to learn more about how to build a better and more robust bot. Code is open-sourced at the end of the tournament (including some of the best bots), and the fact that some of the best competition contributions are made *by the community* (so many teams, including us, swear by Clairvoyance and there was an amazing point where the community came together to help the devs fix a Java transpilation issue with Maven) would be *absolutely insane* in a normal hackathon. We learned a lot from some of the top competitors in 2018, especially from the *Bruteforcer* and *Oak's Disciples* teams, and we tried our best when we could to help some newer people and get involved in the discussion on Discord. As a statement to prospective participants: please be nice and helpful and get involved. This competition is very hard to manage from the devs' perspective (especially since they need to maintain a fairly complex tech stack), and even small contributions, from helping newer members with understanding spec changes and debugging issues to talking more closely about the current metagame, are super helpful to everyone involved. We've met really cool people through Battlecode alone, and we hope you do too!