smite

Vivek Myers, Mihir Patel, Nikhil Sardana, Vinjai Vale

February 1, 2019

1 Overview

Battlecode is a three-week-long AI competition run by MIT students during MIT IAP. The competition changes every year, but the premise stays the same: two teams of robots facing off on a map. You program robots to move, attack and communicate to make sure your team survives and the other doesn't. Teams face off in a double elimination tournament and the final rounds are held at MIT in early February.

1.1 Battlecode 2019

1.1.1 Resources

This year, Battlecode required teams to mine two different types of resources: *fuel* and *karbonite*. Karbonite was used for building units. Fuel was used for building units, along with attacking, moving, and pretty much doing anything. Every map contained a number of fuel and karbonite *depots*. Depots never ran out of karbonite or fuel, but collecting resources from a depot required your pilgrim to stand on the spot for ten turns.

The more depots you control, the more fuel and karbonite your team gains, so you can build units faster. Battlecode 2019 was a game of economy over anything else.

1.1.2 Units

The units for Battlecode 2019 were as follows:

- **Castles:** One to three per team. Cannot move. Originally could not attack, buffed after Sprint tournament to attack with radius $r = 8$. Cannot be built, and once all your castles are dead, you lose.

- **Churches:** Immovable units that cannot attack. Can only be built by pilgrims. Serve as drop-off points for resources (pilgrims must stand next to a church or castle to drop off fuel/karbonite and add to the global store).
- **Pilgrims:** The resource-collectors of Battlecode 2019. Pilgrims stood on karbonite/fuel depots and brought back resources to churches/castles to deliver them to the global stockpile.
- **Prophets:** Long-range attacking units. Low health, medium damage, but could attack from further than other combats could even see.
- **Preachers:** Short-range, short-sighted, high-health attack units. Attacked with AoE damage—if a preacher attacked a square, the nine locations adjacent to the attack target would receive equal damage.
- **Crusaders:** Cheap, fast, but short-range attack units. No special AoE attack.

1.1.3 Quirks

A few more interesting differences from previous years:

- On any given turn, a unit could move *or* attack. Therefore, in a 1 vs. 1 match-up, the unit that first stepped into the enemy's attack radius would always lose. There was very little micro in this game.
- Since it cost fuel to move, launch large-scale attacks was *extraordinarily* expensive. With hundreds of units late-game, it wasn't uncommon to see 20,000 fuel go to waste just moving units across the map to meet the enemies.

2 Sprint Tournament

In the beginning, there were prophets. And they spread their gospel far and wide, across barren lands, past karbonite and fuel, over both passable and impassible terrain. They stalked with killer intent towards the enemy castle, superior to the costly crusader and the myopic preacher. And they came in droves, for we sent them—one after another after another—straight into the heart of the enemy.

We were, at our core, a prophet rush bot. We had the fundamentals down. Mihir and Nikhil have competed for the past two years. We know you start with pathfinding, resource gathering, basic communications, and functional attacking code. Code that doesn't change when **Teh Devs** rebalance. Code that serves as a foundation—that you write once and call a hundred times.

Our one unique feature was that we didn't *just* flood prophets; instead, the attackers paused every few steps, just in case the enemy was *also* pumping prophets our way. In a prophet vs. prophet battle, the first to move in range of the opponent dies. This defensive aggression gave us a distinctive edge over some of the top prophet rushes at the time, like **Ninjadoggy**. But we were still a half-tuned rush bot. Predictably, we placed *okay* in the Sprint tournament. Something had to change.

3 Seeding Tournament

We realized early on that rushes would be nerfed after the Sprint Tournament. Indeed, through **Teh Devs** God descended from the sky and plopped cannons on every castle. We didn't predict, however, that the balance would shift so suddenly to favor economy. Rushes were obsolete in a day. **Oak's Last Disciple** and other teams developed **lattice bots**, placing attacking units in a static grid around the castle. Since creating, attacking, and moving robots cost fuel, this tactic worked remarkably well. It's a very simple strategy, born from a very simple question: what if we did ... *nothing?* The less robots move, the less fuel we burn. The less robots attack, the less fuel we burn. The more fuel saved, the more fuel we can spend building more robots. To do nothing. Except sit there. (And sometimes attack, if an enemy was foolish enough to walk into a line of bored prophets.) Until turn 1000, when their unit health would win the tiebreaks. And so, with this tactic in mind, we set out creating a wonderful *do-nothing* bot. We sat down with 48 hours to go before the seeding submission deadline, starting out by completely tearing down the sprint bot's economy and attacking. When we began anew, we started with just the bare bones of pathing, early-game castletalk to identify where the friendly castles are, and a slew of helper functions.

3.1 On the Economics of Battlecode

Battlecode requires a centralized economy. This is fairly difficult when all your communications must go through signals (which cost fuel), and castle-talks, which can only be sent *to* castles. For the sprint tournament, our pilgrims individually determined which karbonite/fuel deposit was closest, then moved towards the depot. This, as you can imagine, leads to problems. Two pilgrims see an unnocupied resource. They both run for it. The closer pilgrim wins. The pilgrim two steps behind just wasted twenty-five turns of his life chasing a deposit he could never reach.

Castles, we determined, should manage all pilgrims. Centralization prevents waste. When a pilgrim is birthed from a castle, the castle imprints upon the infant his destiny. *"You, my dear puritan, shall journey to (24,13)."* And off the pilgrim goes, trekking to his resting-spot, filling up his sack with fuel and karbonite, and rushing back to his mother-castle.

Of course, if the depot is too far away, the pilgrim, loyal as ever to the castle, will make the trek to his depot, and mine the fuel or harvest the karbonite, and journey back, blissfully unaware he has wasted more fuel in the process than he could ever hope to carry. And so we build the church.

3.2 A Note on Communications and Clusters

Encoding exact locations takes twelve bits: six for the x-coordinate, and six for the y-coordinate. This is fine for sixteen-bit signals, but not so fine for eight-bit castletalk.

Our sprint bot included a simple encoding algorithm to effectively downsample locations into square zones. The map is tiled with 256 such zones, allowing us to communicate a zone in eight bits. This was how we communicated castle locations in the first two turns of the game, in order for each castle to know where the other friendly castles are. Castles could then compute the enemy locations by reflecting over the map's axis of symmetry.

From the first day of competition, we immediately noticed that karbonite and fuel deposits tend to crop up in clumps, rather than randomly dispersed around the maps. At the time of the sprint tournament, we weren't sure how to use this. One of our key insights was that nearly all points of contention on the map are centered at these *resource clusters*. There aren't many of these clusters: in fact, through the hundreds of seeds we'd come across, there were never more than 20-25 clusters on the map. This means that if every unit identifies an array of the resource clusters upon its birth, we can encode locations of interest in just five bits, by communicating the index of the cluster we're talking about in the array of clusters.

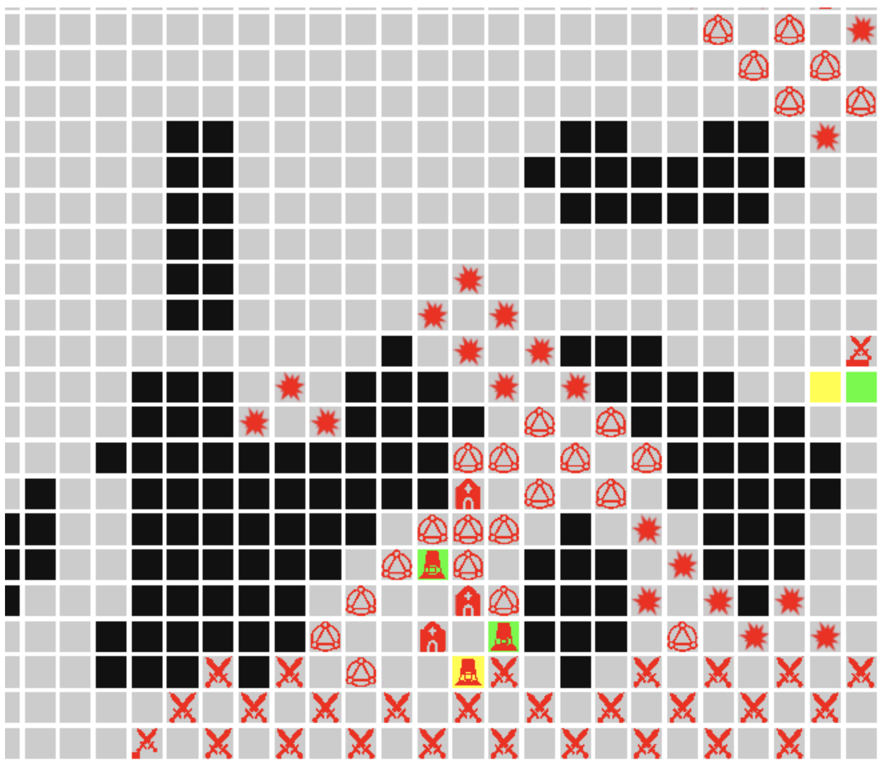
We use simple BFS to compute these clusters, starting at a karbonite/fuel tile and searching for other tiles $r^2 \leq 9$ away, adding them to a growing search queue. We repeat the process, popping off the search queue and adding to a list. We stop when search yields no other tiles, at which point we call the list of karbonite/fuel squares a *cluster*. Every unit that runs this BFS gets the exact same array of clusters in the same order, so we can safely communicate just the index of the cluster of interest.

This may seem like a small point, but clusters ended up forming much of the backbone for our later strategies, including:

- Pilgrim missions: Castles tell certain pilgrims to travel to an unexplored cluster and build a church.
- Harassers: Crusaders/Prophets that rush to enemy clusters and prevent them from colonizing resources on their side of the map.
- Communication: Since most squares are of little importance, destinations usually fall on or near cluster centroids. Since there are less than 32 clusters, and every unit has access to the entire map (and can calculate the clusters exactly the same), we simply need to signal the cluster i.d. (5 bits), rather than an exact location ($6x + 6y = 12$) most of the time.

3.3 Lattice

Now came the great lattice, as our prophets prepared to defend our holy castles. Since each lattice was constructed surrounding either a castle or a church, it was decided the castle/church should dictate its defenses. Thus, each castle/church used its magnificent viewpoint to determine where units could be placed starting in a checkerboard pattern from the center of the map. As units were created, each was assigned a defensive position in this great lattice.



3.4 Preacher Rush

As we'd just developed our new economy bot and hadn't had much time to scrimmage others, we were worried about losing in an economy vs. economy war against a greedy enemy. So an hour before the deadline, we decided to preacher rush on small one-castle maps. Unfortunately, we didn't get to test this very much... and this had dire consequences.

3.5 The Seeding Tournament

To this day, nobody knows how the seeding tournament bracket worked. Probably not even **Teh Devs**. But what we *do* know is that the preacher rush was a bad decision. All but one of the map losses we experienced in the seeding tournament (and in scrims for the next two days before we updated our bot) was due to a bug in the rush. In particular, we'd send out our three preachers, and we attempted to leave one pilgrim behind to mine karbonite. Except for some reason, the pilgrim insisted on mining fuel—every single time. So we'd do significant damage with our rush, but we never transitioned out of it, and eventually lost. Indeed, we had succeeded in our attempt to build a *do-nothing* bot.

On the bright side, our rewrite of the economy appeared to be very strong. We occasionally lost games because very greedy opponents would rush the center with their first few pilgrims, but in general it seemed like we had the core of our macro down.

3.6 The Meta

Going into the tournament, the general expectation was that every good team would simply sit around and turtle. The Battlecode discord was full of complaints that the meta was stale.

Indeed, most of the top teams were turtle bots, with a few notable exceptions. Teams like **Flying Soba Monster** showed up with the strategy of continuously flooding crusaders across the map. This drained the fuel of the turtles, because attacking with prophets is expensive and it takes four prophet shots to kill a crusader. Once the turtle ran out of fuel, it was not uncommon to see a single crusader cleave through six or seven prophets, who had turned from a formidable wall into sitting ducks. A few teams like **Oak's Last Disciple** were careful and included a fuel check to not overproduce prophets; most simply crumpled to the endless crusader onslaught.

Thankfully, **Teh Devs** didn't change any specs after the seeding tournament, despite a nontrivial amount of requests. Their decision would prove to be very wise.

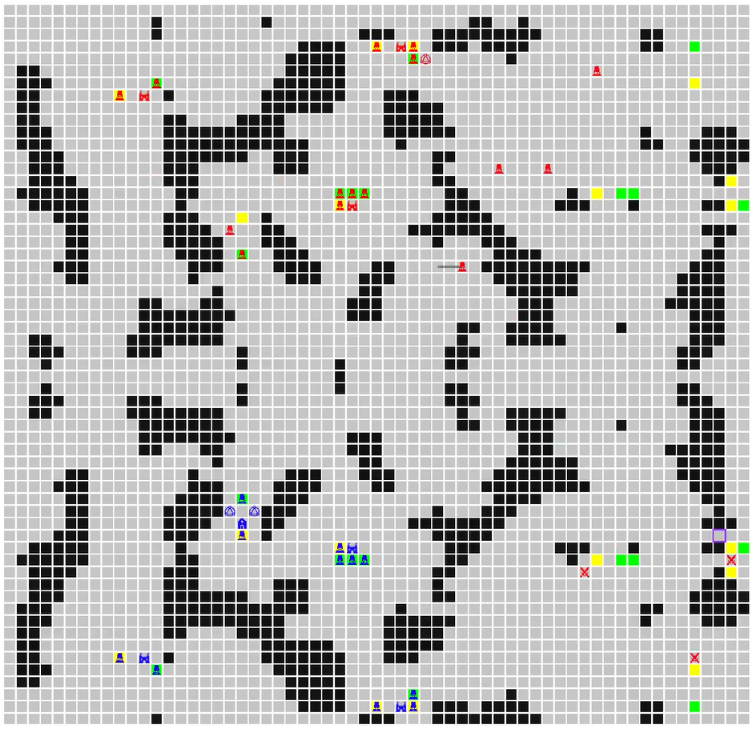
4 Qualifying Tournament

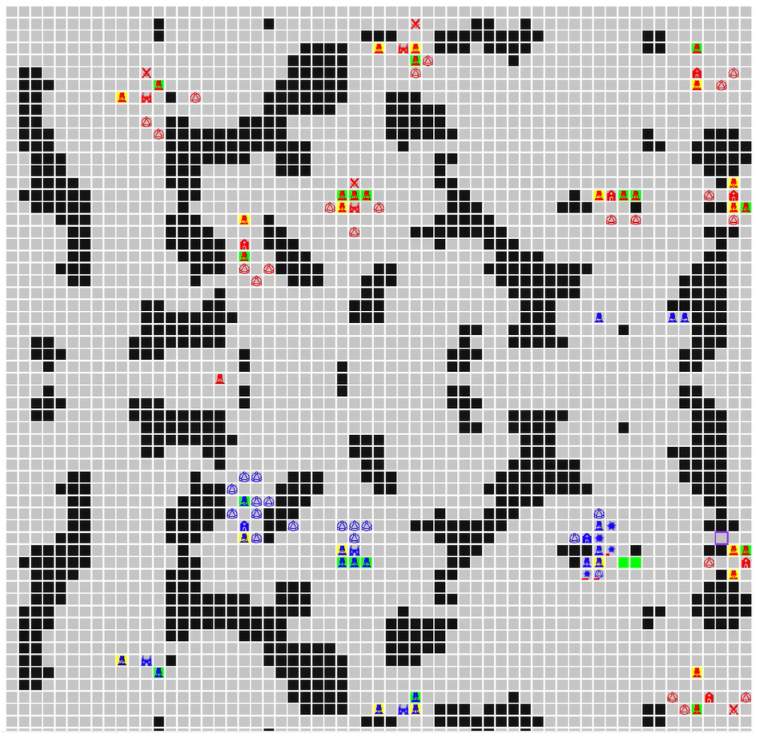
4.1 Resource Harass

Another key insight is as follows: because the map is symmetric, if we can control even one resource cluster on the enemy side of the map, that creates an asymmetry in resource incomes that should, with optimal play, lead to a win (provided we also secure every cluster on our side). We needed a way to counter extremely greedy economies like **Oak's Last Disciple** and **Knights of Cowmelot**, but were worried about implementing an equally greedy economy ourselves, because then we'd be vulnerable to rush bots.

We settled on building up to three units *as early as possible*, and sending them sprinting across the map to the optimal enemy resource clusters. We call these robots **harassers**. Now, there's one keyword here: *as early as possible.* We can't possibly send out a harasser to enemy resource clusters on turn 1: each castle has no idea the other exists, has no idea who is closer to which cluster, or where the enemy castles are located. When castles begin the game, they know the location of all the deposits, and they know there's an enemy at their reflection. That's it. How do we determine the three (or fewer, if < 3 exist) enemy clusters we have the best shot of taking?

Our castles already spent the first three turns castle-talking their locations. This allows each castle to know exactly *how many* castles exist and *where* they are located. With this information, and this information *only*, each castle performs computation *independently* to determine whether it is the correct castle to produce an optimal harasser. Each castle filters out all clusters on our side of the map, along with those too close to enemy castles. Then, each castle computes the minimum distance from each cluster to the closest castle. For the closest three clusters, if I'm the closest castle, I produce a harasser. If I'm not the closest castle, I don't just continue with my standard pilgrim-production. Instead, I `return` immediately, halting all production for one turn. Otherwise, it's very difficult for the harass-producing castle to have enough karbonite to produce the attack unit. With this production-halting hack, our harassers emerge from castles around turn 4-6, early enough to beat enemies to most clusters.


Harass in action (red, bottom right).


The harass pays off.

4.2 Unit Composition and Turtle Formation

The game, we quickly realized, was won on unit health. In this regard, prophets seemed awful as they were very expensive for only 20 hit points. As a result, we upgraded our composition to make all units spawning in the rear of our clusters be crusaders, and in the front we spawned units with a 2/3 ratio between preachers and prophets, skewing prophets. This meant we had a massive amount of preachers, who each had 60 hit points, and were able to gain a huge edge on win conditions. Surprisingly, almost no one copied our strategy. The only downside of this was against aggressive prophets, who would chew threw this weaker lattice. However, given the fuel constraints to organize a late game push, we were confident no one would have the fuel to launch an attack and manage to kill that many preachers, which would force the opponent to burn 150 fuel each.

4.3 Emergency Defense

While our economy-focused bot won on unit health at the end of the game, it was still vulnerable to early game attacks. To mitigate this problem, we added emergency defense code to all of our churc inhes and castles. When these units saw enemy units, they would immediately spawn crusaders, prophets, or preachers (depending on the type and distance of the enemies). This code allowed us to survive early in the game but had a major unintended consequence. Especially later in the game, defensive units would be overproduced, either being fed into enemy lines, wasting resources and stalling global production, or simply using so much fuel to be produced that they could no longer attack.

4.4 Knights of Cowmelot's Church Lightning

With hours left before qualifying, **Knights of Cowmelot** sent us a scrimmage to test their "secret strategy". This turned out to be the feared church lightning. We had given some though to turn order queues, but never quite figured this out and it came as a big surprise. It turned out however, our defense was actually very well equipped to handle this. Our matrix was heavily composed of preachers, which, with their AoE, usually made quick work of the church lightning provided it didn't break our cluster and get all the way inside. As long as we hardened our outer layer, we would be safe. While we weren't able to accomplish that for qualifying, we did for the final tournament and were able to successfully defend any church lightning we saw in qualifying.

As a secondary factor to this, we saw how this approach can actually be used to dump all of our resources on the final turn. We desperately coded this in and added it to our system with hours left. While it was somewhat buggy and only worked if a church or pilgrim could see a castle, it did provide a sizable boost for us.

4.5 The Qualifying Tournament

Generally, our bot performed as expected on the qualifying tournament. We did however lose to DOS two times, resulting in us placing third. DOS used a very aggressive, sustained preacher attack strategy, draining our resources with emergency defense until we eventually lost castles.

5 Final Tournament

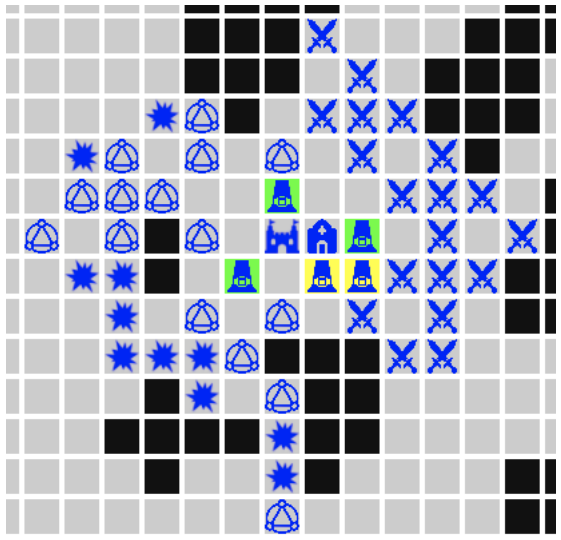
5.1 Debugging

At this point, our bot was extremely solid against most opponents when it actually worked (with the notable exception of **DOS**), but we were still plagued by bugs. The scrimmage server proved incredibly useful; every single loss could be traced to a major bug, and we found bugs in most of our wins too.

Most notably, one of our most severe issues involved our units getting stuck in the adjacent spawn places for our castle and clogging the whole thing up, preventing us from building more units or turning in resources. After days of looking at this, we discovered when updating our attack code that we had accidentally always set the attack flag to all units. This made them use our fuel efficient pathing which only moved 1 square at a time, thus making it very, very easy to clog.

5.2 Dense Lattice

Following the revelations of this new wave of fanaticism driven by churches and pilgrims spreading faith quickly in just a turn, we decided our system needed an upgrade. Namely, we decided to mimic the dense lattice which had units placed on $(x + y) \mod 2$ and $x \mod 2$, constructing an iron grid which these church pilgrim beams couldn't break through. However, we wanted to still keep our checkerboard sparser lattice to gain map control quickly. Thus, we created a way to dynamically switch lattices. In particular, we decided to mark clusters as contested if they had to emergency build a certain amount of times. With this new approach, we could switch to dense lattices in our frontier clusters, helping in both church pilgrim strikes and border skirmishes. We also decided to harden all of our castles past turn 500 to prevent late game cheesing.



5.3 Updating Harass

We noticed that although the prophet harass was effective, by replacing prophets with crusaders, we could get to resource clusters faster. Furthermore, we programmed the crusaders to hunt down and kill enemy units they saw, allowing them to win against enemy harass prophets and also to kill pilgrims, often gaining large kill streaks.

5.4 Aggressive churches

The day before the final submission deadline, we noticed that we would often lose to a team, **Gisthekey**, who whould build agressive churches near resource clusters they wanted to take and spawn units to destroy our churches that were there. We decided to implement this strategy, having pilgrims that lost the race to a resource cluster build aggressive churches near the clusters.

5.5 Prophet Scouts

One of the features we'd always planned on implementing but never had gone around to was scouting. Scouting is essential in a game like this, where responding to the enemy is key. There is no dominant strategy but rather counters for every approach. As such, instead of tuning to be more defensive against rush bots like **DOS**, we decided to send one prophet scout on 1 v 1 maps to the enemy castle. This scout would sit right outside castle attack range and shoot at any units it saw. Specifically, if it ever saw a preacher rush, it would immediately scream back to the castle, which would produce 3 more prophet scouts who could effectively kite and out maneuver preachers in the middle of the field. At best, it saved us the game, and at worst, it was almost no loss. This turns out to have a secondary purpose, in which pilgrims couldn't actually mine any tiles in front of the castle which were in the prophet's attack range until it was killed, giving us valuable economic suppression.

5.6 Breaking Enemy Communications, and Our Own Security

Something that had always worried us was the possibility of another team sending malicious signals similar to our own in an attempt to control our units. To prevent this attack, we wrapped every signal read and write with calls to decrypt and encrypt functions. These functions used the fuel structure and size of the current map to adjust signals by a certain amount, making our robots use different signals each game (based on the map), and eliminating the risk of someone scrimmaging us, recording our signals, and using them against us. We also made a rudimentary effort to decode enemy signals (such as those of **DOS**) but did not end up using them.

5.7 The Meta

This year is unique in Battlecode history as nearly every finalist had a unique combination of strategies that was viable and strong in its own way. Below is a selection of some of the teams we scrimmaged and/or saw in the Qualifiers. It doesn't cover every finalist, but provides an overview of the diversity of viable strategies that are present.

DOS: preacher aggression

Gisthekey: aggressive churches and church lightning

Knights of Cowmelot: strong center jostling, church lightning, and crusader harass

NP-cgw: prophet concave attack and church lightning, prophet escorts on missions

Justice of the War: send missions from the nearest church (castle-coordinated), amass prophets mid-game to reclaim resource clusters

Big Red Battlecode: a much better version of our sprint bot, with endless advancing squads of prophet concaves

Team Barcode: makes a full wall of prophets at the center of the map

Wololo: aggressive preacher/prophet squads

Standard Tech: dense prophet lattice, crusader harass

Chicken: aggressive churches, dense lattice, crusader harass

We decided to take a more middle of the road approach, focusing on a highly efficient economy with strong suppression and harass. To that end, our scrim page before the final tournament and after qualifying had a record of 141-33. We were able to maintain at least a 60% win-rate against every finalist that was open to scrimmages before the finals.

We believe the meta is not set at all. If given another week, we see turtling fading to non-existence as teams are able to construct competent early/mid-game economy attacks with techniques like aggro churches and church pilgrim strikes. A solid midgame bot, capable of suppressing enemy expansions and forcing them off key clusters while aggressively expanding seems to be the general trend to what this game would go to.

All in all, this new dynamic meta has been very different from traditional Battlecode and has been a very fun experience. We look forward to competing in at least 3 more years of Battlecode and are excited for the final tournament.