Reuben Wattenhofer, Parker Skarzynski, and Cody Krueger

Professor Moore

CIS 365 - 01 Artificial Intelligence

31 JAN 2019

Group Name: Croissant  
Halite Bot Report

We began our development by meeting together and ideating possible strategies after we all became familiarized with the typical bot structure, code, game rules and tutorial.  Really, even our initial strategies were influenced by tests of the most basic tutorial bot (which we did during our first meeting to get an idea of what we were working with), demonstrating how much sensor data played a role in our developments.  We broke the bot behavior into several groups: mining, returning to dropoffs, building dropoffs, inspiration, and combat. In addition, we realized that the bot behavior should change depending on how far the game has progressed.

## Mining

    We decided that mining involved two things: finding the richest area and not crashing into other ships.

## Finding Rich Areas

We decided that we needed an advantage in the early game to get a head start on the opponent, so we determined that it would be beneficial to analyze the grid before the game began. We decided that breaking the grid into four quadrants would be feasible computationally within the time limits, and a great way to find areas rich with halite to focus our efforts. For each of the four quadrants we would add up the halite in each cell and calculate the average. After calculating the average for each quadrant we would send our bots into the quadrant with the most halite. This proved successful in early game, but as the game progressed we noticed that this would essentially isolate the bots to one area which caused us to slowly lose our advantage. We believe this was due to the large distance of empty cells that ships had to cover to the shipyard as the game progressed. This in turn caused us to lose time and halite when the ships were headed to the shipyard. We attempted to have the quadrants recalculated after a certain amount of halite was collected, but that did not solve the issue. In late game, this would cause a large ring of empty cells to form around the shipyard. This made travel very inefficient so ultimately we decided to scrap the pre-game analysis. We spent time working on ways to fix this issue which included making smaller quadrants and only sending a few bots out. However, these methods did not resolve the issue either. If we had more time we would’ve liked to experiment with adding dropoff points.

Ultimately, a stop-gap method was used to mine intelligently.  Mining ships look at adjacent tiles (including the current tile) and move to the richest one.  One problem observed almost immediately was that ships would “bounce” back and forth between tiles as they mine.  This was because if two adjacent tiles with similar halite amounts were encountered, the ship would jump to the adjacent one as soon as the current tile’s halite dropped.  To avoid this, we weighted the ship’s current tile halite by 2 in our algorithm. This eliminated the problem and made the ships much more efficient, in general, when mining.  To be fair, this technique was in the Youtube tutorial videos linked in the project description, so it’s not something we came up with by ourselves.

## Collision

    Collisions during mining (and navigation in general) were the biggest problem we faced at the beginning.  At about 40 turns in, the bot would often “die” since it would accidentally crash its ships into one another since it only decided based on where ships were, vs where they were moving to.  To make matters worse, as soon as a full ship crashed on the way back, multiple ships would be drawn to this point compounding the problem, eventually resulting in death. To fix this, we put every planned movement for the turn in a list, and made sure no other ship would move to that location.  This, however, led to other problems that will be discussed in the next section.

Returning to Dropoffs

We divided this into the following subsections: finding the nearest dropoff, avoiding collisions, and efficient navigation.  We did not end up using dropoffs, so the shipyard location was always used. We also used the same collision detection system as for mining.

Our collision detection led to a problem.  We noticed that simply avoiding ships could result in what was witnessed the first day of class- the bots would get stuck with ships waiting in line around their base.  To avoid this we came up with the idea of ship swapping.

To swap ships, we had to consider a few scenarios.  First, we only want to swap with a ship that is closer to the dropoff, and that has a smaller halite load.  The “smaller halite load” condition lets closer, richer ships port first, thought ultimately it probably does little to improve the bot.  Second, we have to make sure both swapping ships are ships that we own; not checking for this resulted in the bot crashing during a few games.  Third, the ship we want to swap with can’t have already moved. Issuing two commands to a single ship will crash the bot, and it did crash before this check was implemented.

Another problem was that the ships would often travel through valuable areas on its way back, losing resources.  We did not address this, since it shouldn’t happen often if the ships are mining intelligently, so we intended that this issue would be solved by that strategy.

To make navigation to the shipyard as efficient as possible, we created a “move order” list for returning ships.  Ships closest to the shipyard would move first. This prevents the problem where a ship farther away tries to move into a tile currently occupied by a ship that hasn’t moved yet.  The ship will abandon its attempt to move that turn, stretching the line out and wasting time for the ships behind it as a result.

## Building Dropoffs

    We did not implement this, which is partly why our bot almost always lost on large maps.

As beneficial as this improvement this would have been, it was just out of reach for the time we had available.

Inspiration

    In the end we did not do anything to take advantage of this, though initially we considered that we could use this to mine more quickly and possibly steal resources close to our opponent.  We determined however that without knowing the enemies movements and the distance typically to an opponent’s base this would not result in sufficient benefit to warrant its cost.

Combat

At first, we were not planning on doing anything for this.  However, we realized that during the endgame we might be able to afford to lose a few ships to give ourselves an advantage.  Near the end of the game, we find the richest player. Up to 6 ships with less than half a load and close enough to reach a dropoff belonging to that player will move to blockade the closest dropoff.  Locations up to three tiles north and south of the dropoff would be chosen as targets. We dubbed this the “crusader” behavior. We did not bother with locations to the left or right since most observed bots did not randomize their returning ships’ movements.  The result is that most returning ships will cluster to the north or south of a dropoff as they return.

The intended result was to force the enemy ships to crash into the “crusader” ships blocking their path.  This would not only prevent those resources from being deposited, it would also increase the cost of movement for every ship trying to get into port.  In reality, most bots did not try to crash into our ships; the ships would simply wait in line, and would never be able to deposit. This was better than we expected, but in the end the bot did not improve its ranking by using this behavior.  If anything, losing our ships’ resources, even though we tried to minimize the loss by choosing mostly-empty ships, hurt us more often than the blockade hurt the other player. We would have like to try this earlier in the game, but it was always a question of whether this would be able to hurt other players enough to give us an advantage- stop one from mining, means you lower your mining and 1 other while the other 2 opponents keep mining.

### Early, Mid, and Late Game Behavior

    We wanted the bot to make the best economic choices throughout the match.  To do this we came up with the idea of breaking the game into stages. During “early game”, the bot is required to have 10 ships on the map.  If ships are destroyed they are replaced. During “late game”, defined as the last 30 turns, the bot is not allowed to make any more ships since they will likely not be able to reach any mining areas in time.  This is also around the time when the “crusade” starts. During “endgame”, defined as the last 20 turns, every ships is required to return to the shipyard. They will crash into any other ship on the shipyard, since that’s faster than waiting for the ship to get out of the way.

### Conclusion

    The most novel behavior, and the funnest to code, was the “crusade.”  Only a few other bots that we observed tried aggressive endgame strategies.  Even though it didn’t end up improving the bot’s performance, it was a lot of fun to watch and we kept it in.

    One of the most frustrating events was when our bot was killed almost immediately when we entered the official tournament- the neighbor bot sent a ship right to our home port and our bot completely quit.  He did not do it to anyone else, just us! But that really just led to more improvements of the swap function and the idea of the “crusade”.

    The “navigation home” behavior was the most difficult to implement, since it required ship swapping and collision detection to work properly.