AI Foundations – Halite Project

Andy Vuong, Matt Schuch, and Steven Bouwkamp

We started out with of one of Travis Erdman’s Python starter bots – namely OverkillBot.py.1 One of the more fortunate aspects for this project was the ability to see many different strategies – both ones that were dominating and ones that were lackluster. From this we got a good sense of what direction to head and began developing strategies. An example of this is realizing that practically all of the bots have two general states: before and after attacking.

What we ended up doing was making more *intelligent* decisions for where to move our pieces based on the state of the game, namely whether we are in attack mode or not. The first thing that we focused on what targeting high production zones – to do this we created a heuristic to calculate the value of any given square that was in the game. The square’s production, strength, distance, and how hard it was to get to it all contributed to our decision of where to go.

The next step was to figure out how to get our squares to the destination that we wanted. This is where vital aspect of efficiency of an AI came into play. We first thought that a breadth-first search from the starting square to the target was our best option because we wanted our squares to choose the best path. What we discovered was that we simply didn’t have enough time to discover these paths as we gained more territory and only had one second to make all of our moves.

Going away from this, many participants of the Halite challenge dove into machine learning strategies and laid out the foundations for anyone else to venture down.2 This is something that was explored and a machine learning based bot got into the Gold ranks for us; however, this came at a price which was the steep difficulty of understanding what is going on behind the various formulas that are being used. There are two choices that can be made, either go through the math of all the formulas, or simply change and tinker with parameters and different functions to see what sticks, the latter was chosen.

Even with that though there is the cost of time. When lowering the learning rate or step size a smaller value might get you closer to your minima, but it greatly increases the amount of time to train. Likewise, changing and testing different gradient decent algorithms and loss calculation functions (mean squared error and categorical cross entropy) can’t really all be done at once because there needs to be reference point to base the changes on and making too many makes that much more difficult.

Another interesting observation from using machine learning was the ability to mimic strategies from top tier players. By downloading all of their recent games and using that as our data set we would only train on the moves they would make – whether they lost or won didn’t matter. The result of this seemed to be much better than just a mashup of 5,000 diamond replays and just picking the winner every single time because having the one strategy minimizes how much contradicting data we would have (one strategy says north is correct and south is bad while another says that south is correct and north is bad).

The player that we chose to emulate, nmalguti34, seemed to have a very stable strategy and our bot could predict it based off of the replays with about a 70% accuracy. However, with our raw Python bot (non-machine learning) some parameters were altered for it and we ran it against three other versions of itself for about 10,000 games. We were able to mimic its strategy with almost 95% accuracy, but since it was data from lower tier bots the results were sub-optimal.