Assignment 4

Roberto Oregon [raoregon@ucsc.edu](mailto:raoregon@ucsc.edu)

John Lara

Daniel Tsai

**Fundamental problems we tried to solve**: Our first problem was what type of agent we needed to build, since we essentially had 3 options – Attacker+Attacker, Attacker+Defender, and ]agents that could switch to either depending on the situation. We decided on an Attacker and Defender since that seemed to be what the baseline agents did as well and we needed somewhere to start with. Now that we had a direction, we got to brainstorming how our agent would work and what it was likely to go up against. We decided to just dump out all of our ideas – good and bad, and filter through them as we came up with them.

We decided that our Defender needed: to keep track of our number of pellets/food in case we implemented an offense/defense approach, location of our food, defend a specific area of our map, potentially check the game state and see if the defending location was less effective than another, and lastly to switch to offense given certain conditions were met

Next, for our attacker we came up with the following ideas: to ensure that our attacker prioritized food over its own life to get our team as far as it could, to prioritize a specific side of the map (ideally either an area with higher food density or without a guard), to eat enemy scared ghosts after an allotted amount of time ( in order to get the most benefit from their scared state – assuming their ghost attempted to stay alive), and lastly to prioritize power pellets.

We believed solving each of these problems in succession would lead to ai that could beat other teams quickly and efficiently.

**How you modeled these problems**: Since our goal was to work on each ai feature implementation, we pitted our ai against the baseline with each change in features. Essentially, we would run 100 games with our ai as it was and then another 100 with a new feature added – if it didn’t improve our victory percentage we either dropped the feature or left it to be worked on later. Simultaneously we watched how our AI did against the opposing teams during the nightly tournaments. Each team potentially had a different strategy than the baseline so we needed to know how our agent (even if it was beating the baseline) did against their strategies. Lastly, We tried different approaches to our testing depending on the features we were implementing. For example, if working on the defensive agent, we adjusted the baseline agent to be two attackers and our agents to be two defenders. While these tests would obviously not be viable during nightly tournaments, it let us see which features were more effective and which features could be dropped.

**Your representations of the problems**: IDK what they mean by this :D

**The computational strategy used to solve each problem**: To start, we took the code from the baseline agents, learned how the code worked and implemented our changes. All of our original ideas required too much calculation time given that we were only allotted 1.5 seconds per move so we dropped expensive ideas like qlearning and alphabeta pruning. The baseline agents code also had a relatively simple strategy: given certain conditions, assign weights to actions and choose the action with the highest potential score.

To build our defensive agent, we created code that heavily prioritized defensive actions. Staying on its side of the map was prioritized as well as invaders and potential invaders. The original baseline code had the agent wandering around our side of the map, but as we quickly saw with the tournaments, having our agent actively head towards the nearest potential attacker greatly increased our defense potential. We adjusted weights accordingly to specific situations: EX: actions that bring agent closer to enemies > actions that don’t. Actions that close the distance with invaders > actions that don’t.

**Algorithmic choices you made in your implementation:** Our endgoal was to implement an agent that could switch from attacker to defender depending on the game state so we immediately sectioned off code so that attacker algorithms weren’t implemented while “playing” defense. Our agents were constantly checking the gamestate and therefore using up a lot of calculation time so any minimizing that we could do was ideal. Most of our implementations were via if statements ie If “gamestate condition” calculate “new objective”, EX: if enemy invaded, calculate distance to enemy, change weights of actions to prioritize closing the distance and killing enemy. How we measured each gamestate condition was by running for loops and checking the amount of food left on the board each turn and then setting up an X Y variable for the layout so that we could measure distances and certain condtions accurately – this was expensive computationally but it allowed us to implement specific strategies that ultimately were effective. EX since we calculated the line of scrimmage each game, we were able to prioritize our defender to stay one space away from it – this often baited our opponents into entering our territory so that we could eat them.

**Any obstacles you encountered while solving the problem:** Most of our ideas required accurate knowledge of where our enemies were, what portion of the map we were on and how to get our agent to prioritize the actions we wanted. Our agent would need to spend a lot of calculation time just to check if a specified event was happening often for that calculation to not be needed if the conditions weren’t met, essentially increasing our turn times for no reason. Our agents relied heavily on score values to cause an action, but this would sometimes lead to stalemates with other agents or misinterpreted actions, EX: if our agent is on offense, it prioritized food over its own safety to the point of repeatedly running into enemy defense agents rather than food that was potentially safer. EX: if the scores weren’t accurately weighed properly (ie different actions had to have different weights, sometimes hundreds of times more), then our agent might prioritize an action like stopping just before an enemy agent can be eaten because the agent would prefer to stay on our side of the board than eat an enemy.

**Evaluation of your agent:** We ended up focusing more on the defensive agent since having a better defender immediately led to better results vs small incremental victories that the attacking agent got us. EX: implementing better defensive strategies earned us an averaged 87% winrate against the baseline whereas better offensive strategies only earned us an increase of about 3%. We also were only able to implement about 6 of 12 of the strategies that we originally wanted (although some of the ones not implemented were due to them not being viable strategies). Regardless, we were able to consistenly average in the top 15 for about half of the nightly tournaments so we were happy with our agent (especially when it broke top 10).

**Lessons learned during the project:** Organizing a group of three to handle the same code was definitely a challenge. Too many cooks spoil the pot so we ended up just having different members work on code separately and ask the other members for help on implementing their goals. EX: “I’m trying to get the agent to do x, how did you get it to do y?” “ i used this line of code to do y, why not try z to get x?” “genius”