Dakota Sliter

Austin Ackerman

Noah Betz

**AI Project 1**

**Introduction**:

Overall, this project was challenging from a programming aspect, but our group found it surprisingly fun to experiment with our agent behavior. Logistically, we had some issues getting together synchronously to work on the project, so we ended up working asynchronously most of the time and giving one another a summary of what we worked on. This included issues we encountered and solutions we came up with. We also threw around many ideas on how to approach agent behavior and discussed which ones we’d like to attempt implementation of. In the end, we settled on a few key behavioral features for our agents, building off of improvedTeam.py:

* One agent dedicated to defense, one for offense (in the beginning of the match)
* Fix issue with offensive agent cowering in dead end corners when enemy ghost was near
* Add points of interest for defensive agent to stay near (patrol)
* Optimize feature weights

We also established some nice-to-have agent behaviors, although we were not able to successfully implement these, either due to difficulty or time constraints:

* Defensive agent dynamically patrols or waits in areas of high-density food
* Strategic usage of the capsule to kill enemies
* Add feature to prioritize offense agent moving to areas of high-food density over lower densities. For example, if there is 1 food 3 moves away, but 3 food 5 moves away, attempt to prioritize pursuing the 3 food cluster
* Simulate likely enemy paths to either avoid or chase better
* Write genetic algorithm that finds the optimum weights of our agent features

This writeup will be broken into three main phases: *Exploration*, *Experimentation*, and *Optimization*.

**Exploration:**

To get started on the project, we first made a Discord server for all our communication purposes. We then started a GitHub repository with the base project structure and everyone set up their hosts for remote pulling and pushing. After running a few sample games to observe agent behavior, we took to analyzing the starter code line-by-line and commenting what each did. This served the purpose of familiarizing us all with the overall strategy employed by improvedTeam.py and getting an idea of what types of methods and properties were available to us. The commented code may seem a bit redundant in places, but it did help to understand the task better.

Next, we created a shared Google Document to hold some ideas we had about how to approach our agents’ behavior. We decided early on that we wanted one agent to remain on defense to avoid giving up any points in the beginning, so our other agent can focus on building up a lead. One idea was to implement a type of behavioral switch in our offense agent, such that when our point lead is greater than a set amount, the agent returns home and begins patrolling friendly food areas. Most of the ideas we came up with are summarized above in the introduction section of this document.

Observing how the games played out, it was clear that the improvedTeam.py offense agent “cowering” problem was a high priority fix for us. We proposed solutions of adding checks to prevent entering dead ends, even for food, as well as maintaining a state variable of last seen enemy positions. This would allow our agent to choose actions that avoid areas where the enemy is likely to be headed, and to stay away from fatal corridors. Additionally, we decided that the defensive agent could be improved if it prioritized certain areas of the map for patrol and didn’t stray too far from those points of interest, even to pursue an invading enemy. This way, an enemy agent would fail to lure the defense away from key chokepoints or food sources.

We now had a long list of ideas for our agent behavior, but the list was longer than we had time to experiment. As such, we rated the ideas based on priority:

1. Fixing the cowering behavior
2. Add points of interest for defense agent to stay close to
3. Offense agent preference for more food-dense areas
4. Switch offense to defense after a significant point lead
5. Target capsules strategically

Next, we began to experiment with implementation of these features and optimization of weightings.

**Experimentation:**

After sufficient planning and discussion, we decided to start coding up some behaviors. We started with fixing offensive agents from getting stuck in corners. Our first idea to fix this was crude and only somewhat effective: simply prevent the agent from taking the “Stop” action if on offense, and choose a different bestAction. This has the effect of preventing our agent from sitting in corners, but they would still get stuck in narrow corridors with only 2 directions to go. The agent would simply move back and forth, but running in circles is better than cowering. We discussed how we could further improve this behavior, and came up with the idea to add a feature that is based on the agent’s distance to it’s home side. In this case, the feature competes with fleeEnemy, so that the offensive agent’s action prioritizes getting closer to home when enemies are near. In practice, this seems to have fixed pacing and cowering, but an issue exists where the agent will sometimes run right into the enemy, dying, and dropping their food. We decided to minimize the risks posed by this imperfect behavior by limiting maximum food carrying capacity to 3 before the agent begins to path back home. We brainstormed some alternative ideas to avoid the enemy but did not implement them successfully. One such idea was to keep a list on the agent object that tracked every move it made, and when an enemy was within a certain distance, choose actions that take the agent back down the path it arrived from. The thinking here was that if we assume the enemy can come from four different directions at most, and our agent entered its current position from a single direction, then in the best case our agent has 3 “safe” directions to go if its path is reversed, giving a 75% chance of evasion. In the worst case, we have 2 directions to go, giving a 50% chance of evasion (action to action, not overall). This assumes a somewhat random distribution of enemy and player positions, and only accounts for the closest enemy “chasing” agent in each case. Of course, agent distributions are not random, and we decided that this behavior would not be a significant improvement.

Now we began working on the defensive agent. Observing it in action, the base behavior from improvedTeam.py was pretty sufficient for dealing with the base offense agent. However, we are going to be facing off against other teams’ AI, so we needed to make some improvements if we want to win (or at least make it past the first round). After clarifying that the tournament map was the same as the base maze we are running our tests in, we decided that we could analyze the maze for points of interest that offer a strategic advantage for our defense agent. With some trial-and-error experimentation, we grabbed the tuple values of these POIs and added our agent’s distance to them as a feature in getFeatures. We weighted it slightly higher than invaderDistance, to avoid our agent getting pulled away from these chokepoints by a fleeing enemy. In practice runs, this seems to work well, resulting in an agent that seems to patrol the different chokepoints until an enemy gets close, and ensures that improvedTeam.py never scores. This behavior is hard-coded, so it will not work dynamically in different maze layouts. We acknowledge this issue and came up with an idea on how to improve it: dynamically generate points of interest by looking at all the food we need to defend, calculated food densities of each grid position, and then choose the top 2 or 3 positions as our defensive POIs. While we were not able to implement this idea, we believe we could work out the solution given more time. An additional, minor change we made to the defensive agent was to ensure it never went back into the spawn area, since there is no reason those actions would ever be beneficial.

Another major idea we had for our offense agent was to add some sort of state tracker to it that would allows us to switch from offense to defense and back dynamically, based on our score. One way we think we could implement this is to set a simple boolean, self.isOffense, and check it’s value in each getFeatures. Based on this value, we could use the same agent class and simply control how features are decided using simple logic branches, adding some checks in chooseAction() as well. Against the improvedTeam.py, this additional behavior doesn’t seem necessary since our defensive agent has no problem holding off their single offense agent. However, against a more dynamic strategy, such as two attackers or “rushers”, such behavior may be useful. The key would be evaluating a “safe” score lead at which to take up defense. Another consideration is that we may need to address the potential issue of agent “clumping”, where each agent will likely choose similar locations to patrol, resulting in a less effective defense against multiple attacking enemies. We could do this by adding a feature, perhaps

Ideas remaining:

* Switching offensive agent to defense centered around a different node than the other agent
* making it so when the offensive agent is in defense land it can go after guys, or will consider waiting breifly (around a corner or at edge?) for ghost to enter
* make it so when both agents are defending, the closer of the two goes after the guy and the other ignores him
* make it so offensive agent doesn’t take dead end paths when running away
* make it so offensive agent remembers what the last thing did
* make them pay attention to the white dot

Notes:

* adding a feature that checks for how close the nearest 3 food are on offense didn’t help the decision making at all, possibly even made it worse.
  + Idea was that it would prioritize a move that is closer to multiple food than 1 food so that it could spend less time, but in practice it didn’t do much.
  + I suspect issue was that it was throwing the weight scale out of whack. Initially it was a very significant skew because I did it based on each foods distance which made it start to care more about being between food than being on food or running away. After heavy tweaking I got it down to being less impactful than where the closest food is, but it still was still causing it was making it harder for me to prevent the pacman from entering into dead end paths that contain food while running away
  + could possibly have been better implemented using a genetic algorithm to determine weights, but this would have taken way too long as it already takes 15 or more minutes to run a single test (individual) 30 times against a single opponent.
* Removed the ability to stop in place on offense because it generally isn’t very useful
* removed the check for if pacman or ghost on offensive agent because it didn’t work
* added feature that weighs staying near the closest chokepoint on the default map highly so that the defensive agent hovers in the area. This combined with the 2 removal fixes already improved the capability of the AI from being very bad to being better than anything else, particularly on the red team. On the red team, the AI won 93% and tied 7% against the improvedteam agents, and it scored a median of +6 points and an average of 7.33 points. Against the baselineTeam agents, our AI still never and had the same median and average scores, but it had 2 more ties than against improvedTeams. Against itself the red side lost 3 games and tied 2, and it only scored an average of 3 points and a median of 1 point per game. This was a lot more even, but there seems to be a slight favoritism towards the red side for some reason. I believe this is because the position of the POIs might be slightly off on the blue side compared to the red.