Team Pacmen:

Colin Xie

Max Leonetti

Zachary Lind

Pacman Capture the Flag Report Write-up

Introduction/Purpose of the Project

The goal of this project is to create 2 efficient agents in a contest to capture the flag with the pacman game as the medium and environment. The nature of the game begins with two teams(red vs blue) in a 1v1 scenario. Each team has two ghost agents navigating through their own side and changing into pacman agents once they cross the middle border. Whichever team is able to eat all the pellets while avoiding or killing agents on the opposite team will be the winner of the round. If no team is able to capture all the pellets on the opposing team, the winner of the round will also be determined by the top scoring team once the time limit is reached. The teams will compose of competing classmates as well as a baseline agent provided by the instructors.

Fundamental problem that you are trying to solve

The fundamental problem we are trying to solve in this project is finding the best method that can consistently beat the opposing player's agents. This problem can be split up into two main phases of our project: beating the baseline and tournament play. For beating the baseline agent, we focused on maximizing the percentage of games won by our agent against the baseline during the test runs. For tournament play, we would then have to find ways to improve this agent for performance against the other competing team's respective agents.

Problem Model/Representation

We chose to represent our problem as a series of smaller subproblems for our agent to overcome. For each of these problems, we would come up with features that we felt would help remedy the problem, implement these features into our model, and then test various weights to maximize the performance of these features. Some of the main subproblems we identified during our project included:

- Balancing pellet searching vs avoiding ghosts for our offensive agents: we needed to make sure our agent would avoid ghosts while hunting pellets, but to not be too scared or else it would play to passively
- Capsule behavior for attacking agents: We want our agent to seek out capsules and act aggressively after consuming the capsules

- Defensive agent positioning: We need a way for our defensive agent to stay in the area most likely for the enemy agent to cross over at, without wandering randomly or acting too rigid

Computational Strategies

Our team considered using a hybrid agent algorithm for both pacman agents instead of the default attack and defensive algorithms. The hybrid agents would prioritize attacking by going after the capsules first so the enemy agents would be scared and avoid our agents while capturing more pellets during that vulnerability period. Once the scared timer runs out our agents would either retreat to their own side and defend or capture another pellet on the map starting the cycle over. Since there were some issues implementing this strategy, we settled with a defense agent that guards the entrance targeting approaching enemies while an attack agent attempts to prioritize food a bit more than capsule on the map.

Algorithmic Choices

Our main algorithm choice was a feature-weight representation of our main problem model, and a reflex agent to evaluate actions based on these feature-weight representations. Our reflex agent would take a linear representation of these features, multiplied by their respective weight values, and judge the values of the possible actions it could take based on this linear representation. It would then pick the action with the highest value, and execute that action. With this method, the bulk of the coding was in coming up with features that could accurately represent the subproblems highlighted in the problem model section. Some of the most useful features ended up being:

- Maze distance from attack agent to closest food - prioritize short distances
- Maze distance from attack agent to closest guard - prioritize long distances
- Maze distance from defense agent to closest enemy - prioritize short distances
- SuccessorScore - incentivize agent to keep moving

We came up with many more features than this during the span of our project, things like prioritizing running from ghosts over food when in close proximity, switching behaviors after eating capsules to attack scared ghosts, but most of these had unintended side effects and overall lowered the performance of our agent, and were scrapped.

Obstacles

Some issues our team faced included our pacman agents attempt at capturing capsules. When the agents reaches a position adjacent to the capsule, it would pause and refuse to eat the capsules since our function had a negative scared distance reward, which would prioritize chasing scared ghosts after the capsule was eaten, however, this feature would always lower the overall score when compared to not eating the capsule, so our agent would avoid eating the capsules.

Another major obstacle was the adjustments of certain weights where changing one value of a feature drastically would affect another feature. For example, sometimes the attacking pacman agent would be too afraid of the enemy ghost so it would run away from the nearby enemy ghosts rather than approach the food. However if the two features had the same weight values, the pacman agent would idle by and eventually be eaten. To combat this, we made sure that there was enough difference in weight between food and ghost priority where our attacking agent would focus a bit more on getting the food rather than avoiding ghosts.

Evaluation of Agents

Our initial main metric for evaluating the performance of our agent was in test runs against the provided baseline agent. Our first goal was to beat the baseline consistently, so this was the most logical course of action. After the addition of a new feature, or a major change in a weight value, we would run our agent against the baseline around 200~ runs in random configurations and base its performance on the percentage of runs it won. By the end of our baseline performance testing, our agent was beating the baseline on an average of 90-100% of all runs. For testing our performance against other teams agents, we watched games against various opponents in the tournament after addition of new features, and adjusted weights/removed features depending on how it performed. In the end, we didn't have much time to improve on the baseline beating agent, but we ended up with a placement of 28th which we were happy with.

Lessons Learned

From this project, our team learned that certain small adjustments of certain weights can drastically affect the performance of our agents in certain situations such as prioritizing food but sacrifice performance in other fields such as ghost avoidance. Also, some ideas and features we thought were simple to implement proved to be difficult with unexpected behavior of agents when manually adjusting feature weights. In the future, we would like to implement a Q-learning algorithm that would update weight values after every game during a training process depending on wins and losses.

Contributions

Maxwell Leonetti - Baseline shell agent implementation, feature addition, performance runs, weight testing against baseline agent
Colin Xie - Feature creation, weight testing, theorycrafting, analysis of tournament performance
Zachary Lind - Weight testing, feature addition, performance runs agent vs itself