Zion Mantey

Raman Srivastava

Pacman CTF Report

Pacman CTF is a modification of the original Pacman game to include two teams fighting against each other to collect the most food. Each team consists of two agents with the play space divided evenly in half. Each agent is a ghost on its side of the board while it is a Pacman while on the enemy’s side. The goal of the game is to collect the food pellets scattered across the board and return to your own side to capture them. A ghost may eat a Pacman forcing them to drop any collected pellets and respawn at their start point. Our solution to this game is to assign a numeric value to each possible move and evaluate its viability using vectors of weights and features. The move that produces the highest value is the most desirable and is chosen.

Our agent is based on the reflex agent which provides the framework for the feature-weight vector system. This system checks each possible action for a certain set of features and then multiplies each feature value by a corresponding weight. For example, these features can be distances from an enemy, food, or if the action is backtracking to the previous location. A function then compares all the values for each legal action from the current state and choses the one with the highest value to be used in the game. This solution is good at selecting a decent action quickly but is it vulnerable to making near sighted mistakes such as getting chased into a dead end. This limits the agents as they may not make the best decision in the long term and may cause them to make a suboptimal action when there is a better choice.

Each time an agent is asked to choose an action, it checks all the available legal actions for their score and returns the highest one. If there is a tie, it chooses one of the highest randomly. The team consists of two different agents that are similar with the differences being the weights associated with the features and the offensive agent has one additional feature. We decided to have two agents because it would help alleviate the situation where both agents would be focusing on the same task and therefore wasting time when they could be accomplishing different goals at the same time. The agents that we decided to implement is one that focuses collecting food pellets while the other one prefers to defend food pellets from the other team. This allows for one agent to focus on increasing the score for the team while the second agent will try to prevent the other team from scoring.

The first feature that is considered is the score, which is the current score of the game added with the number of pellets that the agent is carrying. The more offensive agent has a higher weight for the score feature so that it will value food pellets over the other features. The next two features are the distance to the closest food and power capsule. These act as the default behavior of the agents trying to minimize the distance to collectable food and power ups. The distance from the agent to its starting position is the next feature which is used to return to the safe side when it is being chased or it is cashing in collected food. Then the next three features allow the agents to avoid making actions that lead to a dead end, backtrack to the previous position, or stay still. The implementation of dead-end avoidance is primitive as it only checks for positions that only have 2 legal actions of a backtrack and stopping. This method is not strong enough to avoid a dead end that is a hallway or has space at the end of it. Fortunately, the distance to the nearest ghost mitigates this issue by making the agent prefer to go to the safe side of the stage when a ghost is nearby. The feature works by making the agent return to its starting position if a ghost is within 4 tiles of it and if it is within 3 it will also try to maximize the distance from the ghost. This feature enables the agent to go to safety if a ghost is nearby but will not run directly into one to do so. Finally, the last feature shared with both agents is the distance to the closest enemy Pacman. This feature allows for the agents to chase an enemy that is trying to collect food pellets. The offensive agent has a lower weight for this feature to make it prefer collecting food instead of defending. The offensive agents additional feature makes it avoid the last 8 positions it was in to avoid a stalemate with another agent at the middle of the board. This feature forces the offensive agent to explore more as it refuses to make an unsafe action to cross the middle line into a waiting ghost. By utilizing all these features and weights to describe the value of each game state, the agents can decide which action is the best to make at each point in the game.

The strengths of this approach are that it is easily modifiable to add or change features to the agents to alter their behavior. The most difficult part of implementing a new feature is deciding what the feature should be. Once that is done, it is straightforward to collect the needed information from the game state. Finally, the adjustment of the weight of the feature is done through trial and error mostly. One downside of this approach is that it is simple with a limited number of implemented features and does not learn from each match of a game. The agent could use the multi match style of the game to learn each opponent to optimize its behavior over the course of the tournament.

We have observed that the agent does well against its peers as it was able to get first place in the friendly tournament and second in the preliminary. Unfortunately, it is difficult to measure the impact of each change to the agent after it was able to easily beat the baseline team consistently. This made it hard to determine if a certain feature was helping the agent or if the weight needed to be changed. One behavior that was hard to remove in this implementation was both teams stalling at the center of the board and refusing to cross the line eventually ending in a tie. We attempted to avoid this behavior by making the offensive agent avoid the last 8 positions it has been in, which helped a little bit. However, it can sometimes just lead to the agents stalling over a larger area instead of truly solving the problem.

Pacman CTF is an adversarial game where two teams of two agents are competing to collect the most food pellets in a limited amount of time. Our approach utilizes the reflex agent framework given in the codebase to give each possible action a numerical score using a feature and weight system. The features of an action define how desirable it is while the weights define how important each feature is compared to all the other ones. This solution worked quite well according to the performance in the two tournaments, but it could still be improved with additional features. For example, one feature might be to consider the amount of time left in the game to decide if the agent should return to the safe side to capture the pellets it has already collected. In the current form, the agents do not consider the remaining time left and may lose out on points by collecting more and not being able to capture them by the time limit. Another technique that could be added it the use of game trees to look farther into the future of the game to make more informed decisions. The current implementation only looks one step ahead which is a partial cause to its limitation of getting trapped by larger dead ends. With a game tree, the agents would use more information to make a better decision in the long term to make it much better.