CS 4365.501

Fall 2019

Final Project: Pacman Agent

Bradley Ritschel

Reynu Shirali

Team: The Sneaky Sneks

Final Term Project Report

Introduction

In Pacman "capture the flag", two teams of two agents compete against each other to capture all but two of the other team's food, or more of the other team's food than the other team captures of theirs before 1200 moves have elapsed (300 moves each by 4 agents total). A random maze consisting of walls or open squares is generated. The agents are free to move north, east, west, or south to a coordinate that is not a wall. They can also stay in place if they wish ('stop' move). The teams spawn on two separate sides, split down the middle. The red team spawns on the left side, and the blue team spawns on the right. A team is chosen to make the first move. After the first agent from that team makes their first move, an agent from the other team makes their first move, and then the second agent from the team that was first to move makes their first move, and finally the second agent from the team that was second to move makes its first move; this repeats.

When an agent is on the side that they spawned on ("their side"), they act as a ghost that can kill the opposing agents by colliding with them if they are invading their side. A collision is defined as occurring when a game state exists with the two agents existing on the same coordinate pair. When an agent is invading the other team's side, they are a Pacman that can be killed by the opposing team by a collision. All food that a team wishes to capture is on the other team's side. An agent can carry as many pellets at a time as it pleases. However, if it is killed before it returns to its side, then it drops all those pellets and they return to their original positions. A team need only cross one square on to their home side to capture the pellets they are carrying.

Each side also has two "power capsules" on it that can be activated by "eating" (colliding) with it. These power capsules, like the food that a team wishes to capture, are always located on the enemy side. If a power capsule is eaten, then the enemies become "scared" for 40 moves (4 agents each making 40 moves), meaning that they can be killed anywhere, not just when they are invading the opposing side. If a second power capsule is activated while one is already active, the number of moves that the enemy is scared is reset to 40 moves, they do not sum. If an enemy is killed while it is scared, then it respawns not scared. However, if a new power capsule is activated, then it becomes scared again. When an enemy is killed, they respawn on their side. The red team always spawns in the bottom left corner of the map, and the blue team always spawns in the top right side of the map. The blue side and red side are doubly

mirrored images of each other, meaning to get the map on the blue side, one could take the red side, reflect it over the edge of its right side, and then flip it over its horizontal bisection.

Theoretical Foundation

The baseline team included in the code base provided a good baseline for the construction of reflex agents that choose their next action based on the game state. An evaluation function is used to rate the successor state of each possible move. The evaluation value is the product of a feature and weight vector. The baseline team had one agent that was always on offense, and one agent that was always on defense. Different feature and weight vectors were used for the offensive and defensive agents. The baseline team had some significant limitations, especially since one agent was always on offense and one was always on defense. Both defense and offense are more effective with both agents involved. For example, a single defender can only chase an invader around their side, never killing them unless that invader goes into a dead end. Also, a single agent on offense can never invade a dead end with food in it if there is always an enemy defender present. However, two agents on offense can cause the defending agent to have to choose a single agent to chase, and the other can be free to capture food that is even dangerously located in dead ends. The vectors of features for the baseline team were also incredibly naïve, not considering the position of enemies, whether they were scared, how much food they are carrying, and their distance from their side, all of which should be considered when deciding whether to try to kill an invading enemy, or go offensive and try to capture food, as well as determine the next best possible move if one is already decidedly on offense or defense.

Final Agent Description

Our team, The Sneaky Sneks, is composed of two reflex agents that alternate between offensive (capturing food) and defensive (defending food) modes. Each agent is the same and determines which mode (offensive or defensive) it should be in independent of the other agent. However, it may consider the other agent's position when determining its next best move. For example, it is better for two agents on offense to split up and not attack the same areas, so that a single defender will have a more difficult time. To understand our agent, one must understand how our agent decides whether go on offense or defense, and what features and weights are used to determine the best move when it is in either of the two modes.

When the game begins, the agent defaults to offensive mode, and takes the shortest path to the enemy side. When on offense, the agent checks every move if their team is currently carrying all but two of the enemy's pellets. If they are, then all they need to do is return home without getting killed to win the game, so they go into a third mode, "returning mode" in which they take the shortest path home that avoids enemy defenders.

Otherwise, the agent checks to see if any power capsule for either side is active. If one is active then the agent goes offensive, as there is no point in trying to defend their side if an enemy's capsule is active and they can be killed by any collision and cannot kill the enemy. Oppositely, if our power capsule is active, then we want to go full offensive since we cannot be killed by the enemy team.

Otherwise, we check if we were just killed by the enemy by checking if our current position is our spawning position. If we were just killed, then the agent defaults to defensive mode, since he will be nearest enemies that are already invading our side.

Otherwise, if we are on defense, we determine whether to go offensive depending on who the closest invader is and how many pellets they are carrying, and how close the closest food is that we wish to capture.

Otherwise, if we are on offense and currently on our side and an enemy on our side is closer than the closest pellet we wish to capture, we go defensive.

Once we have decided whether go to offensive or defensive mode, we decide which move is best.

First, if we are on defense and we are on our side, and we are next to an enemy and can kill them with our next move, we just go ahead and kill them.

Otherwise, we consider all moves except staying in place (Stop). We decided that staying in place is never productive, as we want to die as quickly as possible when we are trapped and want to get back in the game, and we want to kill enemies as quickly as possible when they are trapped.

We also remove moves from consideration that would trap us in a dead end allowing an enemy to kill us. We identify these coordinates as “dangerous positions”. At the initialization of the agent, we use an algorithm that considers the locations of all the walls to determine dead ends, which are dangerous positions. When an agent is on offense and considering whether to make a move that enters a dangerous position, an algorithm is used that considers the distance to the nearest enemy and closest food in the dead end to determine if our agent can get the food and get out of the dead end before the enemy defender clogs us in the dead end. If a move into a danger square would clog us, then we remove that action from the actions we are considering.

If we are on defense, we remove actions from consideration that would put us on the enemy’s side, as we cannot kill enemies when we are on their side.

If we are in the “returning” mode for the win, a separate evaluation function is used that determines the path home that will get us their safest and quickest.

Otherwise, we use the defensive evaluation function for each possible action if we are on defense and choose the best rated action, or we use the offensive evaluation function if we are on offense.

Now let’s explore what features and weights are used for the three evaluation functions: offense, defense, and returning.

Features

For offense:

'successorScore': considers what our score would be in the next state, meaning it considers whether we would capture food and the points we would gain from that

'distanceToPartner': considers the distance to our partner if they are also invading the enemy side. We want to attack different regions of the enemy territory so that they have a more difficult time defending all their food.

'distanceToFood': considers the distance to the closest food

'distanceToEnemyGhost': considers the distance to the closest enemy ghost

'riskOfPelletLoss': considers how many pellets we are currently carrying, and our distance from our side

For defense:

'numInvaders': considers how many invaders are on our side

'invaderDistance': considers the distance to the closest invader

'reverse': considers whether we are doing the opposite of our last move, which would be a waste of a move usually

'distToCenter': considers the distance to the center of the map, since we want to defend closer to the boundary of the sides to prevent enemies entering our side and leaving our side

For returning:

'distanceToEnemyGhost': considers the distance to the nearest enemy ghost

'distanceHome': considers the distance to our side

Weights

For offense:

'successorScore': 100

'distanceToPartner': -10,

'distanceToFood': -4,

'distanceToEnemyGhost': -20,

'riskOfPelletLoss': -2

For defense:

'numInvaders': 10,

'invaderDistance': -5,

'reverse': -2,

'distToCenter': -1

For returning:

'distanceToEnemyGhost': -50,

'distanceHome': -10

Observations

Our team performed especially well in three different areas. One major strength of our agents is their ability to avoid getting trapped in a dead end on the opponent's side when they are trying to capture food that may be in a dead end. At the initialization of each agent, grid coordinates that are part of any "dead end" (a group of coordinates with only a single entry point) are identified, and when our agent considers to enter one of these dead ends, a calculation is made using the distance to the nearest enemy and the distance to the nearest food in that dead end to decide if it is safe to pursue the food in the dead end, meaning we will leave the dead end before our enemy has time to clog us in it.

Another major strength is the ability of our agents to decide to return home based on a 'riskOfPelletLoss', which is a function of the number of pellets our agent is currently carrying, their distance from their side, and the distance to the nearest enemy. Instead of trying to capture all pellets in one attack, our agent makes many trips, which decreases the number of pellets per trip, but decreases the likelihood of death, and reduces variance since we are no longer using an "all-or-nothing" attacking strategy.

Perhaps the greatest strength of our agent is its ability to decide when to toggle between offensive and defensive mode. The distance to the nearest pellet we wish to capture, the distance to the nearest enemy, and how many pellets each enemy are currently carrying are the main factors that our algorithm uses to determine which mode to choose for the current game state.

While our agent was successful at identifying simple, straight dead ends, our agent is poor at identifying complex dead ends, where for example there could be a single entry to a large open space that is not identified as a dead end. However, the enemy would still be able to clog us in that space by simply stopping at the entryway.

Another downside of our agent is its lack of ability to anticipate enemy moves. Our agent only considers the current and last game state, and does not make any assumptions about the enemy team's strategy, which implies that our agent is not able to predict the next moves of our enemies, and select our next moves with our predictions in mind.

Another important drawback of our agent is its inability to strategize when to use power capsules. Power capsules do not "stack", meaning the maximum number of moves an enemy's "scared timer" will be set to is 40 moves. Therefore, it makes sense to only use one power capsule at a time and save at least one of them for a dangerous push deep into enemy territory.

Recommendations and Conclusions

Our simulations show that our team beats the baseline team over 90% of the time, which shows that our overall strategy of toggling each agent between defense mode and offense mode continuously is superior. However, many improvements can be made in the future.

Focusing on the three drawbacks of our agent mentioned above, we believe the next iteration of our agent should be able to identify complex dead ends that are not in the shape of straight lines. This would further prevent our enemy from clogging us in dead ends when we are attacking.

Our agent should also learn to predict enemy moves and adjust accordingly. Furthermore, if our agents could learn to act cooperatively instead of independently, then they will have many more opportunities to squeeze the enemy to death by trapping them in a section of the map with two or less entrances.

Finally, a simple strategy could be implemented that decides when to use power capsules. A simple improvement could be to avoid consuming additional power capsules when one is already active.

Overall, we can conclude that implementing a strategy of continuous toggling between offense and defense is superior to the strategy of the baseline team, where a single agent is always on offense, and the other on defense.