Pacman Contest Documentation

Team: ExampleTeam

Version 1.0

22/05/2018

Author: Sean Lee Jin Xiang s3301214

TABLE OF CONTENTS

1 INTRODUCTION

This report entails methods and details on how we created a competitive agent for Pac Man Capture the Flag game. It is a multi-player game with two agents on each team. The goal is to get food in the opponents' territory and bring the food back to your home side, while defending the opponents which are trying to do the same.

2 AGENTS

We will describe the strategies and algorithms used in the implementation of the two agents developed. All our agents use a A* path finder algorithm to find the best path to a set goal, the path accounts for enemies that can kill us while retreating to our side.

2.1 Defensive Agent

Our team prioritized defence over offence. So, we implemented a dedicated defensive agent. We needed to find a way to successfully defend against attacks and after we successfully eat or kill an enemy, it would make it easier for us to invade the enemy's territory when one of more of their agents are sent back to their respawn points. So, initially our agents start defending to try to locate enemies to get more information on whether it's safe to attempt to invade the enemy's territory. Our agents only ever attacked when it was free to attack, otherwise it stayed on its assigned enemy and defended.

We implemented this by setting a default defend position, where one defender will defend the closest food to the enemy and the other will sit a few spaces away to try to observe the enemy's position and whilst on defence, our strategy was purely to reduce the distance between our agent and the enemy's agent we were pursuing.

After implementing this, we observed that we were getting a high number of ties. This is probably since, if the agents were body blocking or protecting our food successfully, the opponent's agents will never try to invade our side, and vice versa. We found this to work extremely well on maps which had a clear boundary line but suffered harshly on maps with multiple obstacles in the middle sections of the maps.

As having a lot of ties seem to be drastically pulling our score down, we had to implement a way to successfully get ahead by at least one point then set our agents to go full defensive. Here is a brief explanation of our defence strategy:

- Default Defend Positions: If we can't observe an enemy, we set default defend positions to each defender. We get the closest food to the enemy and set the first agent's default defend position to the closest exit to that food, which is calculated by getting the boundary list and using the mazeDistance function to see which is the closest one to that food. If the default defence position is next to the side of the map, move away a few spaces from it to cover more of the boundary. If there is a power capsule that is close to the boundary, we set the second agent to protect that capsule from the enemy. If not, the second agent will be defending the side of our first agent with the biggest area and sit 6 spaces away.
- Body Block: If we can observe an enemy, we make the defence agents follow the enemy up and down the boundary without crossing over. First, we get the

closest exit to that enemy and find the best way to move to that exit using our A* implementation of a path finder and if we are on the boundary. With the enemy on the opposite side, we take a step back from the boundary, to lure or bait the enemy into crossing the boundary and when they cross over we eat them.

- Defend: We made it so that the closer defender to the enemy that has invaded our territory will pursue that attacker and try to kill it, while the other agent is set to body block the enemy agent that's closest to them but furthest from the teammate, this is because we don't want the same defender body blocking or perusing the same agent and leaving the other agent to do as he pleases in our territory.

2.2 Offensive Agent

Our offensive agent simply works by grabbing the closest food to our agent, as well as calculating if will we be able to return after grabbing that food using the A* pathfinder algorithm. We first, try and get the enemy positions and after we established that it is safe, we look to see if there is any food we can grab easily. We also check if the enemies are scared, and if they were, we try and invade their territory. If there is any indefensible food pellet's, we always attacked that regardless as it was free points. Our agent always retreats to deposit the pellets after it had collected 3 pellets and if they ever returned prematurely before that, the current pellets collected counter resets to 0, allowing the agent to stay on offense for longer periods of time.

After a certain number of moves, and we are still tied with the other team, we realise that if we do not try to get some points on the board, it would ultimately resort into a tie. And after viewing the preliminary contest results that were ran daily, we seem to notice that having a high number of ties was the reason we were scoring so low on the leader boards. So, we set it that after a certain number of moves were taken and we still have a score of less than or equal to zero, we must try and invade get some points on the board. Here is a brief explanation of our offence strategy:

- Easy Food: This tells our agents if there are any easy foods we can steal from the enemy, it calculates it by checking the enemy positions and if they are on our territory, do not invade and if the mazeDistance from our agents to the enemy is less or equal to the distance to the food, that the enemy is close and its not worth it to try and invade. If none of this is true then it will try to steal the closest food to the agent, using the A* path finder algorithm we implemented**.** Also, if we are currently in invade mode, it will stop the agent from getting foods that is too far away from the current food that that agent has just eaten because it increases the risk that it will get captured while holding a lot of points that have yet to be deposited into our territory.
- Retreat: After we have successfully grab a food in the enemy's territory, we must get back to our side of the map. We calculate where is the closest exit and then using the A* path finder, we calculate if its safe to get there without running into an enemy that threatens to kill us if not we find another path.
- Invade: If getting an easy food fails, we must find a way to try and invade even if it's risky, to avoid all the ties. We find the distance to the closest food or capsule, if the capsule is closer, we will try and get the capsule and we try and get the enemy positions, if they are scared we will try and kill them then after

we have eaten a capsule, we check if are we able to grab the next food and return safely to our side. If its unsafe we will just retreat to our side and deposit the food. If it's safe we can continue eating food.

3 ASSUMPTIONS

We made several assumptions on our enemy's behaviour to try to predict the best solution that would pull us ahead of our competitors. These are the assumptions we made:

- If the enemy is in our territory and we seen them eat a food pellet, we assumed that they were attempting to return to their territory, to deposit that pellet, regardless of their knowledge of the current game state and how many pellets they had already collected.
- We assumed that the enemy would always at least send one agent to attempt to collect the food and/or capsule that was closest to their current position.
- We always sent an agent to a food/power-up capsule if it was closer to our agents starting position in comparison to the enemy's position.
- We assumed that the enemy will always head towards a capsule over a food pellet if the capsule is of equal or less distance to the enemy's current position.

4 ANALYSIS OF STRENGTH AND WEAKNESSES

4.1 Strengths

- Our agents will recalculate a viable path to the goal if it encountered or assumed there was a blockage in the path. E.g. Didn't go down a path where there was an enemy previously seen or could possible kill our agent.
- Our agents were never 'too' greedy, we always returned after 3 pellets were eaten. Instead of collecting a lot of pellets and risk being eaten by the enemy.
- We only ever sent in an agent to attack whilst it was undefended or had no enemy to defend. If it knew there was an enemy either in, or heading to our side of the map, one of our agents returned to slow down the enemy's attack.
- Our agents were able to determine which of them was more likely to successfully defend against an enemy. E.g. The closest agent was sent but factored in the distance to the second agent also which meant not always the closest defended.

4.2 Weaknesses

- Our agents have no way of evaluating if it was wise to return to our side or continue eating pellets. We returned as soon as our agent ate 3 pellets.
- Our agents didn't determine if performing an action was safe. E.g. We didn't check if we were entering a dead end without a guaranteed escape route or a long alley which would increase our chances of being blocked and killed.
- As the agents were unable to share information, an agent would try to eat an enemy invader and traverse to that enemy's last known location regardless if a friendly agent had just eaten that same invader.
- Agents had no reasoning behind whether we should head to a food or capsule, we only ever were eating the closest food to our current location. This meant that in the possibility of getting trapped by an enemy and if we had an option of eating a capsule, we weren't taking it.

5 TECHNIQUES TRIED

- Min/Max algorithm to determine the best action that leads to our goal.
- Creating an agent that determined its greediness based on the current state of the game i.e. Returning to deposit pellets if we were a certain score behind the enemy's current score or returning to deposit pellets if an enemy was within a certain number of cells from our current position
- Tried different food counters. E.g. How many pellets do we try to collect before returning to deposit them, 5? 3? 1?
- Tried different defensive strategies: Leave an agent on the boundary to always cut off the shortest path back for the enemy, whilst the second chases them through our territory.
- Tried double offensive strategies from the start of the game.
- Purely reducing the maze distance to a target was not viable, agent typically found themselves stuck in a situation that lead to a guaranteed death.

6 TECHNIQUES DISCUSSED

- Performing our pathfinding function on the successor state to check whether we would be guaranteed a path to return safely to our side of the map. This function would have included minmax which would predict the best route for an enemy to reach us from scoring.
- Using breadth first search to find the shortest path to an agent's goal.

7 LAST MINUTE IMPROVEMENTS

After viewing the number of ties we have based on the preliminary contest results. We observed that we seem to be doing well against the top teams but losing to the bottom ones. We figured this is because our defensive agent is not performing well against teams that just run a greedy algorithm and just constantly try to grab a food and retreat with no defensive functions. We then improved on our defensive and offensive agents and placed them in more strategic areas of the boundary to block them better and, we tweaked our A* path finder to not only calculate the best path to the goal but also if it's safe to get to the goal and back to our side without getting blocked or eaten by the enemy.

After doing so, we were able to steal some food quickly and score some points before going into defensive mode. We then shot back up in the rankings and during the last preliminary contest and placed just 3 pts behind staff_top.

8 CONCLUSIONS

We feel there are still a lot of ways that we can improve our agents, but it was hard to see how our agents reacted to the other teams because there was not a quick was to test our code against them. We had to wait for the preliminary contest results to check how our agents perform and can't do minor tweaks here and there to test it out.

Modelling our agents separately into offensive and defensive areas was probably the best way to tackle this assignment. As narrowing down the problem's associated with each into smaller subproblems allowed us to concentrate our strategies independently and perfect them. Overall, it was a very challenging but fun assignment.