# Contest : Pacman Capture the Flag

Team : PacmanGo

Chang Zhou, Weihan Dai, Yuan Gao

## ABSTRACT

Pacman Capture the Flag is a variant of a popular game called Ms.Pacman. This paper details the approaches taken to design agents to play this multi-player game. The goal of the team is to get as much food as possible on the opponent's territory and bring the food back within limited time and steps, while defending food on their own territory. The uncertainty of opponent's policy and location is the challenge of the game. We designed two teams PacmanGo and GoTeam. PacmanGo is composed of one offensive agent based on Minimax, and one defensive agent which uses particle filter to approximate opponent's locations. GoTeam is composed of two feature-based offensive agents, which will collaborate while attacking. Our teams can perform well when competing with other teams with average winning rate of 56.25%.

## KEYWORDS

Pacman; Multi-player; Minimax; Particle Filter

## 1 INTRODUCTION

The reason why we choose capture the flag as our final project is that, the game itself is fun and there are plenty of strategies to win. In the following part, we'll briefly introduce the rule.

The map is divided into 2 halves: red(left) and blue(right). Each team with two agents must defend their own color territory while trying to eat food on the other side. Red agents must defend food on red side and try to eat food on the blue side. When on the red side, red agents are ghosts. When on the blue side, red agents are Pacmans.

When a Pacman eats a food, the food is stored up inside the Pacman and removed from the board. When it returns to its own side, it deposits the food, earning 1 point per food delivered. If the Pacman is eaten before returning to its own side by a ghost, it loses all the food it is carrying and re-spawn to the start point as a ghost. All the food become a cloud of food on the board. No point rewards for eating Pacmans.

Agent can only observe enemy agents position and direction if they are within 5 Manhattan distance from our agent. Otherwise, we will always get noisy distance reading. And this game is limited to 1200 moves(300 moves for each agent), and the computation time for each agent is 1 second.

The algorithm we are using is minimax algorithm with alpha-beta pruning. To deal with the noise in distance reading, we implemented particle filter, which is an established method for approximate inference of Hidden Markov Model. When test against two test agents baselineTeam and GoTeam, our agent won 19 games out of 20.

## 2 APPROACH

Each team can control two agents. On our territory, they are in form of ghost (defensive agent), while on opponent's territory, they are in form of pacman (offensive agent). We have designed different strategies for the two types of agent.

### 2.1 Development of offensive agent

The goal of the offensive agent (pacman) is to return enemy's food safely. When making a choice for the next action, the offensive agent uses miniMax algorithm to choose actions that maximize the score. The score is calculated by evaluating a given game state with various heuristics. We have developed following strategies (heuristics) to guide the agent.

*2.1.1 Avoid enemy ghosts.* Although the first priority of our pacman is to return enemy's food, we want to do it in a safe way. Our pacman will try to avoid enemy ghosts by keeping distance to the nearest enemy. When enemy ghost is very close (maze distance <= 5) to our pacman, we choose to rush back to our side to dump the food we are carrying.

*2.1.2 Don't be too greedy.* If our pacman was caught by enemy ghosts, we will lose all food carried by that pacman. Thus we want our pacman to regularly return food to base and come back to eat more food. Currently we set a carrying-limit which equals to 6 at the beginning of the game, and this number decreases according to how much time is left in the game.

*2.1.3 Avoid obvious dead end.* There are many dead ends on the map. Our pacman should be able to 'see' an obvious dead end and avoid that, because run into a dead end will waste a step and increase the chance of being caught by the enemy. To help the pacman avoid dead end, we scan the board at the very beginning of the game. For each legal position (not wall), we calculate its degree (how many directions can move when standing on that position) and store the result in a dictionary for later reference. When making a choice for the next move, our pacman should select a next position with a higher degree.

**Table 1: Heuristics of offensive agent**

| Feature | Weight |
|---|---|
| Board score | 1000 |
| Remaining food | -100 |
| Distance to closest food | -5 |
| Distance to enemy ghost | 20 |
| Distance to unseen ghost | 1 |
| Distance to scared ghost | -2 |
| Freedom to move | 1 |

### 2.2 Development of defensive agent

The goal of the defensive agent (ghost) is to defend our food. When making a choice for the next action, the defensive agent simply

select an action from all legal actions that maximize the score. The score is calculated by evaluating a given gameState with various heuristics. We have developed following strategies (heuristics) to guide the agent.

*2.2.1 Chase enemy pacman.* Our ghost will keep monitoring invaders and try to send them back. For enemies that are insight, their exact location and direction are known, and our ghost will move towards the enemy to shorten the distance between them. For enemies that are unseen (only know a noise distance), we use particle filters to estimate their current location and move towards them.

*2.2.2 Follow potential enemies.* Our ghost should be able to follow potential enemies's move, and stop them from invading. To this end, we use particle filters to estimate enemy agents' location and make our ghost move according to the enemy's position.

*2.2.3 Stay close to frontline.* If there are no invaders, our ghost should stay close to frontline and patrol. This is more efficient than staying at rear.

**Table 2: Heuristics of defensive agent**

| Feature | Weight |
|---|---|
| Number of invaders insight | -1000 |
| Number of unseen invaders | -500 |
| Distance to invaders insight | -10 |
| Distance to unseen invaders | -5 |
| Distance to potential enemies | -2 |
| On Defense | 100 |
| Distance to frontline | 1 |

## 2.3 Strategy of test agent GoTeam

As our final team is one defensive agent and one offensive agent, we implemented two offensive reflex, feature-based agents to test.

To make the two agent collaborate, we divide the food on the map into two parts. Each agent would approach the closest food in their own foodList. Once they had consumed all the food in their foodList, they would return the food back to home and went after the food left on the map.

To avoid the enemy ghost, we made the agent to retreat to middle line regardless of whether it has eaten up its foodList to secure the food they have eaten. Eat capsule first if it is close to the capsule or its teammate is close to the capsule. This is also a way to collaborate.

Our test agent GoTeam is adventurous as to the distance to the opponent. Although we can see the actual distance within 5 steps, we only make moves when the opponent is one step away from us. The advantage of it is the fact that we can eat as much food as possible. The disadvantage is somehow obvious, sometimes the agent will run into a dead end and caught by the enemies.

To determine the weight given to each of the above goals, we used the human thinking way. Cause it is easy to rank the offensive agent's priority: 1. avoid ghost enemies, 2. eat adjacent scared ghosts, 3. eat closest food. We set different feature and weight to reflect these priorities, as shown in Table 1.

**Table 3: Heuristics of GoTeam agent**

| Feature | Weight |
|---|---|
| successorScore | 100 |
| eatInvader | 5 |
| invaderOneStep | 1 |
| distanceToFood | -1 |
| distanceToShelter | 10 |
| eatCapsule | 10 |
| dangerousGhostOneStep | -20 |
| eatScaredGhostFirst | 1 |
| safeGhostOneStep | 0.1 |
| stopped | -10 |
| eatFoodFirst | 1 |

## 2.4 Locate unseen enemies

One challenge in this game is that when enemy agents are far away (manhattan distance > 5), their exact positions are unknown. What we only know is a distance reading with noise up to . This is a classic hidden Markov Model (HMM), where we cannot make direct observations of the current state. To locate unseen enemies, we applied particle filters for approximate inference of the HMM [1] [2] [3]. Briefly, we use a particle to represent a possible state (a legal location on the board). At time elapse step, we update the particle distribution according to enemy's moves. We expect the enemy agents will move towards our food, thus the moves that shorten the distance between enemy agent and our food will have a higher probability (we use 0.9). At observation step, we update the particle distribution according to current observation and normalize the probabilities. When the enemy agent is insight, we know its exact position. Otherwise we use the expected location which has the highest probability. Using this method, we were able to estimate enemy positions with high precision. This is critical to improve the efficiency of our defensive agent. In Figure 1, the depth of red color represents the approximate locations of enemies. We can see that particle filter can almost locate every unseen enemies.

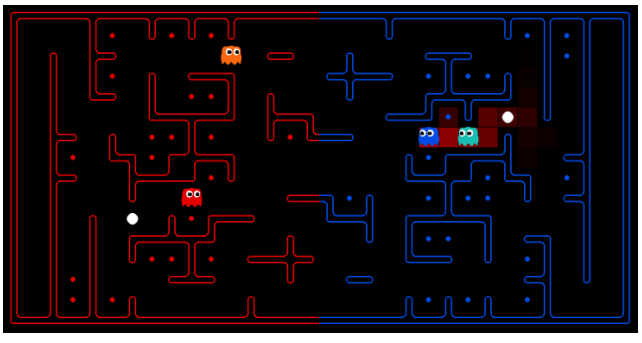

**Figure 1: Defensive agent with `Particle Filter`**

## 3 RESULTS AND TESTING

We did the test for our test agent GoTeam, and our final agent PacmanGo with/without particle filter on default map. Following are the results after running 20 games against each opponent team.

**Table 4: PacmanGo vs BaselineTeam result after 20 games**

| vs BaselineTeam | Win | Tie | Lose |
|---|---|---|---|
| PacmanGo without Particle Filter | 85% | 5% | 10% |
| PacmanGo with Particle Filter | 95% | 5% | 0% |

**Table 5: PacmanGo vs GoTeam result after 20 games**

| vs GoTeam | Win | Tie | Lose |
|---|---|---|---|
| PacmanGo without Particle Filter | 70% | 5% | 25% |
| PacmanGo with Particle Filter | 95% | 0% | 5% |

From the test results we can see that the winning rate against GoTeam and BaselineTeam improved after implementing the particle filter.

Then we used our final agent(with Particle Filter) against other teams. Following are the results.

**Table 6: PacmanGo vs Other teams result after 20 games**

| PacmanGo vs | Win | Tie | Lose |
|---|---|---|---|
| Team1 | 0% | 35% | 65% |
| Team3 | 55% | 10% | 35% |
| Team4 | 0% | 60% | 40% |
| Team5 | 55% | 0% | 45% |

We have assessed the replay and observed several problems. One of them is deadlock between our offensive agent and opponents defensive agent(or our defensive agent and opponents offensive agent). These agents move back and forth near the border line, because the enemy ghost is in the way to the nearest food that our offensive agent wants to eat. This usually results in a tie game.

Similarly, we utilized GoTeam to compete against other teams.

**Table 7: GoTeam vs Other teams result after 20 games**

| GoTeam vs | Win | Tie | Lose |
|---|---|---|---|
| Team1 | 30% | 30% | 40% |
| Team3 | 90% | 0% | 10% |
| Team4 | 70% | 0% | 30% |
| Team5 | 15% | 0% | 85% |

From the results, we noticed that the winning rate increases dramatically against Team 1, Team 3 and Team 4 comparing with PacmanGo's result. That's because GoTeam has better offense collaboration than PacmanGo.

A common issue with PacmanGo and GoTeam is fixed roles of agents. As a result, situations like these may occur in PacmanGo, after the offensive agent re-spawn at the start point and invaders observed in our side, the offensive agent will not assist the defensive agent so the enemy got away with food. Similarly, when there is no invaders, the defensive agent just stay and not attack, which is conservative. On the contrary, GoTeam is way more adventurous, but it lacks defense action, once it is caught by enemy and re-spawn at home it will not defend only if the enemy is very close.

## 4 DISCUSSION

There are still a lot of place that can be strengthened in our work.

To begin with, we considered the factor that the time and steps are limit, and fixed- role-agent cannot deal with all situations in a dynamic world so our agent should change roles dynamically. We could make agent switch between different strategies, like Attack, Defense, Run, Chase, and let the agent switch role between offense and defense based on how far the game has progressed. Besides, when there is a deadlock, we could have made the agent switch plan or path. At last, our minimax depth is 1 so far. When the depth is 2, the execution time is too long. To solve the problem we may run the minimax in a simulate gameState to improve the depth to 2 or 3.

As to our test agent GoTeam, in some situations, it received better result than our final agent because of its collaborative offense. It can be improved also by changing strategy based on gameState, like when the board score is enough to win it can return home to defend the territory, using the strategy of our final agent.

## ACKNOWLEDGMENTS

We gratefully acknowledge John DeNero and Dan Klein in UC Berkeley for developing the contest baseline code.

This work was performed by Chang, Weihan, and Yuan.

Chang found the idea of the project. He implemented Minimax and Particle Filter in our final agent. He wrote the approach of our final agent part. Weihan arranged the code, meeting note, implemented agent test and designed the heuristics of final agent. He wrote the introduction, test and result part in this paper. Yuan implemented test agent GoTeam, organize the team meetings and took charge of the presentation PPT. He wrote the rest part of the paper and combine the paper using latex.

We want to express our gratitude to Prof.Stacy, what he taught in class not only gives us inspiration, but also is very useful in our future study and work. Also, we would like to thank our TA Dan and Navya, who have been assisting us in solving problems during the course. And special thanks to Pedro, Nutchanon, Dan, Sophie for giving us wonderful lectures.

## REFERENCES

[1] UC Berkeley. 2017. Particle Filters and Applications of HMMs. (Dec. 2017). Retrieved December 1, 2017 from http://ai.berkeley.edu/slides/Lecture%2015%20--%20Particle%20Filters%20and%20Applications%20of%20HMMs/SP14%20CS188%20Lecture%2015%20--%20Particle%20Filters%20and%20Applications%20of%20HMMs.pptx

[2] S. Thrun. 2002. Particle Filters in Robotics. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*.

[3] Wikipedia. 2017. Particle Filter. (Dec. 2017). Retrieved December 11, 2017 from https://en.wikipedia.org/wiki/Particle_filter