Pac-Man Contest Report

Team: HelloWorld

Team member: Meng Qi 791890, Linlin Cong 790432, Zelong Cong 732735

1 Introduction

This report is designed for AI Planning for Autonomy project 2. The aim of the project is to implement a Pac Man Agent, which is able to compete with other Pac man agent automatically under different scenarios.

2 Approaches

2.1 General Strategies

We have an offensive agent and a defensive agent.

The strategies for the offensive agent are: 1) Give up last two foods. 2) Greedy eat pacman when ally not around. 3) Greedy eat foods when safe. 4) Greedy eat capsule when the agent is half distance closer to it than ghost. 5) Other situation: UCT Algorithm trade off score, escape, capsule.

The strategies for the defensive agent are: 1) Patrol along the boundary of our territory. 2) Get to the position where the food disappears immediately. 3) Patrol around the dots if there less than 5 dots left. 4) Chase and eat the enemies in in its sight

2.2 Technologies - UCT

Assumptions:

- The agent need to get as much scores as possible while avoiding the ghosts in its sight.
- Goals and corresponding awards are: eat invaders (1), avoid a ghost (1), eat a capsule (0.002) and get 3 scores (0.5).
- In most situations, the agent can find one or more goals within 20 moves.
- If there is only one legal move, we return that move without doing the simulation.
- If there are only 2 foods left, we then find the shortest path to return home.

Rational

The UCT class initializes two counters: self.plays and self.wins. The former stores all the positions the agent experienced during simulations as keys. For each key, the corresponding value is the number of times the agent has been to that position during simulations. The later also stores all the positions the agent experienced during simulations as keys. However, for each key, the corresponding value is the number of times the agent has found a goal on that position during simulations.

For each action, the time of UCT simulation process is restricted to 0.5s. Each simulation allows the agent to explore at most 20 game states. Once the agent finds the goal, the simulation will stop and the corresponding reward will be added to each position in self.wins, which is the path to the goal. Lastly, all the positions stored in self.plays will be added 1 no matter the agent found the goal or not. During the simulation, we apply UCB1 policy to balance whether to encourage more exploration or not.

When the time slot is over, we get all legal successor states of current state. For each successor state, we use its position to find its corresponding values in self.plays and self.wins. Then the winning rate is calculated by self.wins divided by self.plays. Finally, we select the successor state with the highest winning rate and return the corresponding move.

Strengths and weaknesses

The strength of UCT lies in the fact that it combines MCTS with UCB1, which makes the choice between exploration and exploitation more reasonable. Specifically, when using MCTS only, our selection of next move in the simulation is based on random choice, which creates an exploration vs. exploitation dilemma. After applied the UCB1 policy, the fear of missing out can be alleviated by a calculation that balance the dilemma, thus the decision of the next move can be made more precisely.

However, UCT is not perfect. It should be noted that the simulation stage makes our UCT strategy to take much longer time than using just classical planning. If the number of simulation times is not big enough, the agent may not be able to find the goal and the winning rate will be 0.

2.3 Technologies - BFS

Assumptions

- BFS leads the agent to the nearest food when there is no other interference.
- There is always a goal exist when using BFS and it can always find a path to it.

Strengths and weaknesses

The biggest strength of BFS is that it is complete and is always able to find the path to a goal. In other words, our agent will never get trapped in useless explorations using BFS. However, BFS requires a large amount of memory when the search space is big. In our experiments, we found that it required a lot of time when the maze was large and the nearest goal (food) was quite far away from the current position of the agent.

2.4 Technologies - Greedy

Assumptions

- Greedy leads the agent to the nearest goal when there is no other interference.
- Greedy can consume less time and computational resource than BFS.
- Greedy chooses next legal successor which is closer to the goal (maze distance).

Strengths and weaknesses

The biggest strength of Greedy is that it takes much less time and less computational resources than BFS. Also, it is quite easy to be implemented. However, due to its simplicity, it cannot be applied in every circumstance e.g. trade-off between eating a capsule or a dot.

2.5 Challenges

The biggest challenge was to adjust awards of four goals in UCT simulation, for example, the agent was too aggressive to avoid the ghost. This was because the wining rate for eating capsule (or food) after simulations became much higher than that of avoiding ghost. Consequently, we found the best awards by adjusting them several times.

Eat an invader (1.0), avoid a ghost (0.7), eat a capsule (0.02) and get 4 scores (0.4)

The other challenge is to prevent the agent from entering alleys without food. We solved this problem by adding function takeToEmptyAlley( ).

3 Experiments and Performance

3.1 Performance

Compete with baseline team

The performance of competing with baseline team is good. When using MCTS and Greedy only, our team could win more than half of the opponents' dots when the time was up. However, the agent sometimes moved randomly when there is no ghost nearby, which was caused by the random selection of explored nodes in MCTS. After applying the UCB1 policy, the performance was largely improved. Our team was able to beat its opponents before time was up and the random move was not observed.

Compete with staff team

The performance of competing with staff team varied on different mazes. Specifically, our agent timed out on some mazes when it trying to use BFS to eat opponents. This may be caused by the fact that BFS requires much more time and computational resources when applied on big mazes. Therefore, we changed it to Greedy Algorithm.

3.2 Possible improvements

One possible improvement is to predict the next move of the opponents when they are in our view. This can be achieved by using our getting UCT action function. However, it should be noted that doing simulation for opponents might make the whole choice action stage to take longer time and thus lead to timeout.

4 Conclusion

The report presented the techniques, challenges and performance etc. of the Pac Man contest project. We implemented UCT and Greedy to support our strategy for the pacman agent. The observed performance from experiments seems to be in line with our expectations and UCB1 contributed a great extent to the optimization of our strategy. Further improvements could be using UCT to predict the next move of opponents as long as the simulation rounds is carefully decided.