COMP90054 AI PlANNING FOR AUTONOMY---Project 2,2017

Tianyu Xu tianyux2 818236, Junwen Zhang junwenz 791773, Ziyi Zhang ziyiz5 798838

1. Overview

In this project, we implement a Pacman Agent Factory and use the agent to form a team to compete with other teams in the tournament. In the game, our agent has to capture more food in the opponent area as well as defend our food. In the implementation, we use the Game Theoretic Method and Heuristic Search to help the agent make decisions. Both our agents have the ability to attack as well as defend and we keep trading off the two decisions during the game. In general, we make one of our agents pay more attention to attack while the other one pays more attention to defend through adjusting the parameters. Our design purpose is to maximise the heuristic value that our agent can get in the future several steps.

2. Techniques

2.1 Heuristic Search Algorithms

In the project, we use a Pacman-specific heuristic function for evaluating the game state. The heuristic function is a linear combination of our features, heuristic value $=f_1 \times w_1 + f_2 \times w_2 + \cdots + f_n \times w_n$. For applying the technology, it includes two main steps, one is feature selection while the other one is weight adjusting. First, we select some features which affect the agent decisions. Then we assigned some extreme values to those features and use it to compete with baseline team. Through the competition, we filtered the features which have less effect on the decision-making. For adjusting the weights, we use our team to compete with itself and other teams then we mutate the weights from winners to update combinations so that the heuristic function can be closer to perfect heuristic function. Besides, we use parameter "defending food" to adjust the efforts between attack and defence. Meanwhile, we use the distances to nearest food, capsule, teammates, invader and ghost to calculate the corresponding influence factor.

2.2 Game-Theoretic Methods

By using the heuristic function, we evaluated each game state after choosing one action. Then we use Expectimax to simulate the following game procedure to maximise our value after the next 3 steps. In this implementation, we first use the possible actions to simulate all the possible future game states of both aspects, which can form a game tree. Then we use level traverse to implement backward induction. We traverse each element of this level recursively and return the better action to its upper level to find the temporary best action and update the final best action until this level is searched completely. Besides, we set a maximum value for searching depth as well as action time in case timeout problem happened. Meanwhile, we get the noisy distance from the game then use particle filter algorithm to infer the position belief distribution of the opponents. Afterwards, We consider each opponent's action is deterministic but the opposite position is stochastic. The expectation of the opponent's next step is calculated based on the position possibility and value of the new game state.

3. Challenges

During the project, the first challenge is that the Expectimax tree is too large to traverse so we have to do a tradeoff between the search depth and action time. We use Alpha-Beta pruning to ensure that the agent will only consider actions whose previous action is the best action. We set the maximum action time to ensure the agent can search deeply if time available. The second challenge is that the heuristic function weights are hard to adjust resulting from the effect of competitor's policy and map layouts. For solving the problem, we select several representative maps for each competition and teams using different policies for competitors to make our evaluation model more general.

4. Improvements

In our implementation, we find we still have several deficiencies which can be improved in the future. On one hand, the heuristic function we use is not perfect heuristic which can be improved by adjusting the weights through training. On the other hand, we still have a bottleneck in the implementation, in which our agent always wandering over several positions and not enter a new state. A possible solution is to detect the scenario and force the agent to enter a new state which may not be the optimal one.

5. Performance

Our team did some experiments by competing with other teams to test our performance and check the strengths and weaknesses. From Table 1, it is obviously to see that at most time our agent can win the baseline team but may fail in some specific scenarios, in which our agent was stuck in some positions and cannot get out. Besides, in the second case, our team use the action evaluation function to compete with the state evaluation used before. As the table shows, the action evaluation is worse than the state evaluation function from the experiments, which is easily affected by the action. Features used to evaluate the future scores do not relavant to a final winning goal. Meanwhile, our team adjust our weights by competing with the previous version. By adjusting weights of different features, we can avoid some mistakes and dilemma in making decisions. The Version 2 agent scaled down all the distance features by dividing the map size to ensure the maze distance has lower influent factor. The Version 3 agent increase the ghost influent factor to make it less possible to enter dead end for eating some foods.

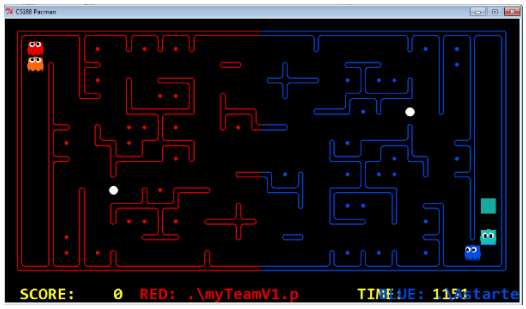

Fig 1 Layout RANDOM1357

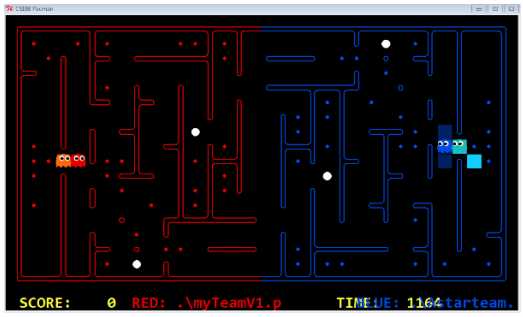

Fig 2 Layout Contest

Game	Opponent Method	Layout RANDOM1357		Layout contest	
		Win:Tie:Fail:	Score	Win:Tie:Fail:	Score
vs Baseline	Reflex	5/0/0	9.8	5/0/0	17.4
vs Version 1	Action Evaluation Function	4/0/1	4.8	3/0/2	2.2
vs Version 2	Different Weights	4/0/1	16	4/0/1	5
vs Version 3	Different Weights	2/0/3	3.2	2/0/3	4.1
vs MCTS	Monte-Carlo Search Tree	4/0/1	12.8	3/0/2	1.2
vs Beans Eater	Game theory & Heuristic Search	3/0/2	-0.4	3/2/0	7

Table 1 Experimental Game Statistics

After setting up the weights through confronting with old versions, we start to compete with our classmates' teams to help optimize our parameters further. Here we choose two teams, one of which uses Monte-Carlo search tree and the other one uses the similar methods as ours but different heuristic functions. As the statistics show, our team has similar performance with the other two teams.

By comparison, we find that the game theoretic methods have a good performance on competing with the opponent like capturing the invading Pacman and escaping from the ghosts' chasing. And it can make some decisions which may not produce the biggest reward at that time but have higher possibilities of survival. And the heuristic search method can make the agent get rewards not only from food but also some other states and actions like capturing Pacman, eating capsules and defending food. However, our team also find some shortcomings of our agent design. The first one is that for routes of a dead end, the agent cannot detect it in advance. The second one is that our features are based on predictions of the opponent's state and its own state, which may cause our agent to make decisions actively.