COSE361(03)
Final Project 2/2 Report

Choi Jinhyeok, Student No. 2022320006 [1]

1. Introduction

The final project(2/2) is an implementing of two-agent team against opponent's team. The map is divided into 2 side, red and blue. Each agent is a pacman or ghost, depending on the agent's location. If my team is red, my agent is ghost when in the red side, and pacman in the blue side. We should eat foods on the enemy's side as much as possible, and protect foods in our side against enemy pacman.

My implementation is based on the baseline code, because the evaluation method for the successor is quite efficient. Thus, I use the baseline code for base logic, but also add some other features so that the agent can consider other important information.

2. Methods

Details about my implemented 3 agents

2.1. your_baseline1

Main difference of my agents is that the offensive agent can be defensive under some conditions. Offensive agent can be a defensive agent when the score is high enough. The threshold is half of the number of the protecting foods. It also consider the features 'distanceToGhost', 'distanceToCapsule', 'distanceToHome'.

'distanceToGhost' is the feature about the distance to the ghost. When the ghosts are scared, the feature is 50. Otherwise, the feature is the minimum distance to the ghost, especially -9999 when the distance to ghost is 5 or less.

'distanceToCapsule' is the feature about the minimum distance to the capsule. It is same as the actual minimum distance to capsule. The weight is -10, which is more important than food.

'distanceToHome' is the feature for safe pacman returning. When pacman eat some foods, it is important to return agent's side to get the points. Thus, it was implemented to calculate the distance to the home as the feature, when the pacman ate more than limit number of food, so that the pacman do not goes far away from home. When there is no opponent's ghost, or all opponent agent is pacman, this feature will be ignored to free eat.

Defensive agent is also based on the baseline, but it will walk around near the capsule position. For this, new feature 'disDiffToCapsule' is added.

'disDiffToCapsule' is the feature about the difference of distance to capsule of agent and enemy pacman. Defending the capsule is important because eating capsule can make enemy agents scared. Thus, defensive agent will try to keep his distance to capsule less than the enemy's. First, find the minimum distance between enemy pacman and capsule with the target capsule. When the enemy's distance is large enough, just keep closer to the target capsule than the enemy. Otherwise, or when the enemy is close to the capsule, the agent's distance to the capsule become much important with higher weight.

2.2. your_baseline2

This is an improved version of 'your_baseline1'. Both offensive and defensive agent has modified with some new features or improved way to compute their features.

First, the offensive agent will become a defensive agent when the agent is on his side and the all of opponent's agent is pacman. This is the case when the protecting foods is more important than attacking.

Second, 'distanceToHome' feature has modified to include the number of foods that pacman is carrying. The new feature is multiple of the distance to the home and the number of foods carrying. It makes the returning with larger number of food is more important.

The new feature 'eatCapsule' is included for offensive agent to eat capsule. If the next position of the agent is the same as the capsule position, or it is guaranteed to eat capsule, the agent will always eat the capsule. This is because there is no other threat of the ghost, and it is more important to eat capsule than one single food.

The 'disDiffToCapsule' feature was also adjusted to consider more specific circumstances. When the opponent's distance to capsule is large enough, the feature is just the agent's distance to capsule. When the opponent is close to the capsule some extent, the distance difference is matter. If

the agent is closer than the opponent's pacman, the feature is the agent's distance to capsule with the corresponding weights. If the opponent's pacman is closer than the agent, the feature is worst case value. When the opponent is much close to the capsule, the feature is square of the distance to the capsule with weight, so that the distance become more important.

2.3. your_baseline3

This is an upgrade version of 'your_baseline2', which has the best performance among the three agents. It was focused to improve offensive agent to survive from the ghosts, has some new features 'eatGhost', 'eatInvader' as well. The defensive agent is same as the 'your_baseline2'.

Computation method for feature 'distanceToGhost' was improved. Surviving is the most important thing of the offensive agent, thus the feature has high value than other features. Large distance to the ghost is preferred, so the feature must be proportional to the distance. If the feature has extremely high value where threshold for having the feature (i.e. like a stair function), it can make the agent to stop on the threshold while the ghost is approaching. The feature 'distanceToGhost' is designed to be zero when the agent is not a pacman, so in some cases, agent will stop on the half line to remain as a pacman. This is because the smaller distance to the ghost has higher evaluation value rather than to move agent's side making feature is zero. Thus, the feature was adjusted as following equation.

$$distanceToGhost = \begin{cases} \frac{-9}{x+\alpha} & (x \leq 10) \\ \frac{9}{100}x - \frac{9}{5} & (10 < x \leq 20) \\ 0, & elsewhere \end{cases} \quad (1)$$

Where the x is the minimum distance to the ghost, and the α is adjustment factor. This equation make the feature value function continuous and smooth, so that the above problem can be resolved. The weight of the feature is 10.

New feature 'eatGhost' is for eating scared ghost. If the next position generated by the action is same as the position of the sacred ghost, the agent will always eat the ghost.

'eatInvader' is similar to this. When the offensive agent is offensive mode, there are some cases that the agent ignore adjacent opponent pacman, and just go to the opponent's side. Although it's purpose is to eat foods now, there was a chance to eat opponent pacman. Thus the agent was modified to eat the right beside pacman. If the offensive agent is not a pacman, and the next position is same as the opponent pacman, the agent will always eat the pacman, not changing his offensive mode.

Moreover, when the offensive agent is on the defense mode, the agent will ignore protecting capsule. The offensive agent does not have a 'disDiffToCapsule' feature, but the defensive agent only. This is because when the all agents are on the defense mode, it is efficient to disperse two agents.

This agent has higher performance among 3 agents, thus this was choosed as the 'your_best'.

3. Results

	your_best(red)
	<Average Winning Rate>
your_base1	0.9
your_base2	0.8
your_base3	-0.2
baseline	1.0
Num_Win	3.0
Avg_Winning_Rate	0.625
	<Average Scores>
your_base1	2.9
your_base2	1.0
your_base3	-0.5
baesline	4.5
Avg_Score	1.975

This is an $output.csv$ file. In this result, 'your_best' is same as the 'your_base3'. The result shows that 'your_best' wins every time against 'baseline', also wins with high winning rate against 'your_baseline1' and 'your_baseline2'.

4. Conclusion & Free Discussion

Q: What if the defensive agent also can be a offensive agent in some conditions?

A: It is my big interest, because it means fully-reflex agent. I think it is more efficient in some cases such as when all opponent agents are scared, because it is possible to eat more food in time. However, the map is not big for two food-eating pacman, thus it is not guaranteed to higher performance. Rather, the more complex circumstance is possible, so a lot of features can be required to manage it. I want to design fully-reflex agent and to evaluate which agent is better, but it is not easy due to other assignments and exams. I will try it after the semester.