Final Project 2/2

Seongjoon Mun, 2018320147

1. Introduction

This mini contest is a Pacman game where agents control both Pacman and ghosts in team-based strategies. My team will try to eat the food on the far side of the map while defending the food on my home side.

My agents are in the form of ghosts on my home side and Pacman on my opponent's side. Also, I am able to eat my opponent when I am a ghost. If a Pacman is eaten by a ghost before reaching my own side of the board, he will explode into a cloud of food dots that will be deposited back onto the board.

My implementation in this final project includes two agents which are FirstAgent(Offending Agent) and SecondAgent(Defending Agent). The FirstAgent tries to score in a conservative way, which he eats one or two dots on the opponent's side and returns back to the home side. In this way, it is hard to get a high score in a short time but since the goal of this game is to win, he will be able to deliver food to the home side in a safe manner. This agent does not get involved in defending his home side.

On the other hand, the SecondAgent only focuses on defending, which means that he does not try to eat the food on the opponent's side but just tries to eat the opponent who invades our home. The design of SecondAgent is similar to the provided baseline.py code in this assignment.

2. Methods

All the details and explanations are included in the python file that I submitted.

2.1. Your baseline1

The actions are chosen based on the evaluate function which is calculated by linear combination of features and weights. The FirstAgent(Offender) tries to eat one dot in the opponent's side then returns to his home. In this way, it is easier to score than eating many dots at the same time. The evaluate function of this agent is determined by distance to food on the other side, number of remaining foods, and whether this agent is on the opponent side(pacman). Also, this agent does not take part in defending the home side.

On the other hand, the SecondAgent(Defender) tries to eat opponent that is on our side, but tries to stay on our side as long as possible. The evaluate function of this agent is determined by number of invaders, distance to invaders, and whether this agent is on our side.

Below are the features that are related to deciding actions of the FirstAgent.

- **SuccessorScore** : This is calculated by -1 multiplied by length of the foods left on the opponent's side. The weight of this feature is 100, which means that this feature is very important and this forces the pacman to eat the dot when it is near.
- **distanceToFood** : This is the distance between our agent and the closest food of the opponent. The weight of this feature is -1, which implies that pacman should get closer to the food.
- **OnOffense** : When the first agent eats a dot, he has to return to our home. So the closest distance between the first agent and our food is calculated and this agent moves toward that food. Also, when he eats the food, OnOffense feature is updated to 1 which means that he is currently on the opponent's side and has to return home. When this agent arrives home, this feature is updated to 0.

Below are the features that are related to deciding actions of the SecondAgent.

- **numInvaders** : The number of invaders, the weight is -1000 which means this feature is the most important and the SecondAgent tries to eat the invader.
- **onDefense** : Whether this agent is on our side, because of this feature, the agent does not go to the opponent's side and purely does his job as a defender.
- **invaderDistance** : The closest distance to invaders, thanks to this feature, this agent does his best to get closer to the invader.

2.2. Your baseline2

Baseline2 is similar to baseline1 except for that when opponent on their side gets too close to our pacman, the pacman tries to avoid the opponent rather than eating foods nearby.

One feature is added to the FirstAgent.

- **attackerDistance** : The distance between our pacman and the closest opponent in their side is calculated.Then 6*(2/float(distance)) is put in to the attackerDistance feature, which implies that the pacman will strongly try to avoid the opponent when the distance is one or two.

2.3. Your baseline3

There are a few improvements and new functions in baseline3. First, the pacman now tries to eat power capsules which makes him invincible. When he eats this, he does not return home but rather eats all the foods nearby. Also, he doesn't avoid opponents that are scared. However, when he detects opponent nearby that is not scared, he returns home. Also, the function of SecondAgent is improved slightly to roam around and escape when he is trapped in the maze during the beginning stage. The SecondAgent moves towards the closest food in our area when there is no invader. And the weight of successorScore is decreased to put more importance on avoiding opponents. In addition, the function of FirstAgent is improved to return home after he eats the food with 100 percent accuracy, and to move(not stop) when there is an opponent nearby. Also, weight of onDefense is increased to make sure that SecondAgent stays on our side.

Two features are added to the FirstAgent.

- **redistanceToCapsule** : This feature is inversely proportional to the closest distance between power capsule and our pacman agent. So the pacman tries to eat this power capsule due to this feature.
- **isSame** : When there is an attacker in the opponent's side, the pacman is in danger so he has to move for sure. So penalty is given when the pacman doesn't move which actually forces the pacman agent to move.

Two features are added to the SecondAgent.

- **isSame** : When there is no invader on our side, the SecondAgent will try to move rather than stay in one spot in order to prevent wasting time on the same place.
- **distanceToFood** : When there is no invader on our side, the SecondAgent moves toward the closest food in our area to escape the start point of the maze and to get prepared for invasion.

2.4. Your best(2018320147.py)

Your best is similar to baseline3 except for the few improvements. First, the pacman now can return home when he eats all the food on the opponent's side. This is implemented by adding a new variable named flag2. Also, when our pacman detects an opponent nearby, he tries to avoid the opponent but as soon as the opponent gets far away, he eats foods nearby rather than returning back home.

There are no additional features, but just few improvements and variables(flag2) to slightly change our agent's movements.

3. Results

Table 1 is the result of my agent(your best) against baselines. If the number in the table is 2, it means that your best won the baseline by 2 points. 0 is a tie.

Baseline	Yourbaseline1	Yourbaseline2	Yourbaseline3
4	6	0	0
4	2	0	0
3	5	0	0
4	2	0	0
2	9	0	0
8	4	0	0
5	4	0	0
5	3	0	0
3	3	0	0
3	4	0	0

Table 1. Scores of Your best against other baselines.

Average Score : 2.075

Baseline	Yourbaseline1	Yourbaseline2	Yourbaseline3
+4.1	+4.2	0	0

Table 2. Average scores of Your best against other baselines.

Numwin : 2, Average Winning Rate : 0.5

Baseline	Yourbaseline1	Yourbaseline2	Yourbaseline3
1.0	1.0	0	0

Table 3. Average Winning rate of Your best against other baselines.

4. Conclusion and Free Discussion

What improvements can be made to your final agent?

Answer : First, when the opponent becomes invincible after eating power capsule, our defending agent has to avoid the opponent rather than trying to eat it. Also, the method of eating one food and returning back home takes too much time. So the FirstAgent(offender) has to eat more food

before going home if it is possible. In addition, if there is a food nearby and our pacman can die if he eats that food, the pacman has to give up eating that food in order to maintain its life. However, the current agent eats the food and dies so this problem has to be fixed. Moreover, when invader comes into our area and it is closer to our FirstAgent(offender), this agent has to help catch this opponent. In short, the FirstAgent has to help defending and the SecondAgent has to help offending. The role should be switched when it is necessary.