# Mini Contest 2
# Multi-Agent Adversarial Pacman

Korea University 2014250137 Kwon Yong Hwan

## Abstract

This documents explain the method used to make the adversarial pacman game model. Furthermore, results of the simulation with baseline model will be presented and analyzed.

## 1. Introduction

The final project is building an adversarial game, involving two teams competing against each other. Each team have to try to eat the food on the enemy side of the map, while defending the food on their team side. Agent will be in the form of ghosts on their home side and pacmen on their enemy's side. If a pacman is eaten by a ghost before reaching his own side of the board, he will explode into a cloud of food dots that will be deposited back onto the board.

- I made 3 different strategies for this contest.
- First model is designed only for the offense. It doesn't care about the defense of their home.
- Second model has a base class for reflex agents that chooses score-maximizing actions.
- Second model used alpha beta minimax and each agents have a role to offense and defense.
- Third model is designed if agents have eaten more than 5 capsules always return to side.
- Third model's agents participate in both offense and defense.

## 2. Methods

There are lots of information we had to consider for all model such as the location of food pellets we want to eat and defend, what team we're on, what side of the board we're on, location of our opponents, location of capsules, whether the enemies are scared or not. Also, the agent must not approach to the invaders if the agents are scared. We defined these information by using APIs.

### 2.1. First model

Based on the derived information, we select a list of best actions and choose one action randomly among them. We evaluate with the value by mutliplying features and weights. We got the value of features by comparing the locations of the remaining food and that of each agents. 'getFeatures' function returns the list of 'successorScore', 'distancetoFood', and 'distancetoBuddy'. 'getWeights' function returns the value of 'successorScore' as 1000, 'distancetoFood' as -10, 'distancetoBuddy' as 9.

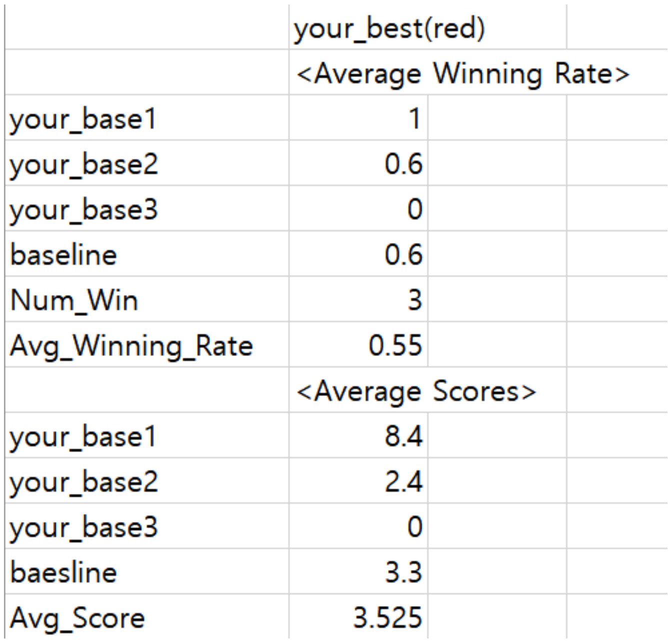
### 2.2. Second model

We use reflexagent to choose score-maximizing actions. We pick action among the actions with the highest Q(s,a). Then, we find the next successor which is a grid position. We evaluate by computing a linear combination of features and feature weights. We return a counter of features for the state by 'getFeatures' functions. I thought weights do not depend on the gamestate so I returned the value in a dictionary shape. Then I used alphaBetaMinimax until the depth become 0. One of the reflexagent take a role of offense. It seeks for food. I modeled offense agent by computing distance to the nearest food. I wanted small numbers for 'closetoevil' to be bad, and big numbers to be good and wanted to maximize the 'smallest dist' by 'getWeights' function so i returned different values to the each variables. Another reflexagent take a role of defense. I computed distance to the invader who is currently in agent's territory. I tried to prevent the agents keep going back and forth.

### 2.3. Third model

I tried to not care how close invaders are to food, only how close they are to getting back to their side. Also, if we have eaten more than 5 capsules, I modeled the agents to return. I got information such as what team we're on with the function 'registerInitialState'. I tried to use as many information i can represent such as the location of enemy pacman or the enemy ghost, the distance between the closest food that we tried to eat or defend, etc. I made a special variable 'defense' and based on its value, we decided whether we have to offense or defense.

## 3. Results



| | your_best(red) |
|---|---|
| | <Average Winning Rate> |
| your_base1 | 1 |
| your_base2 | 0.6 |
| your_base3 | 0 |
| baseline | 0.6 |
| Num_Win | 3 |
| Avg_Winning_Rate | 0.55 |
| | <Average Scores> |
| your_base1 | 8.4 |
| your_base2 | 2.4 |
| your_base3 | 0 |
| baesline | 3.3 |
| Avg_Score | 3.525 |

I chose third model as my best model. I competed my best model with 1st, 2nd, 3rd, baseline model for 10 times each. Average winning rate seems too low but we have to consider that 3rd model is the same model with my best model so its result always be draw. Because my first model is composed of only offensive agents, it was easy to win. While 1 defensive agent is killing every dummy agents of first model, another offensive agents have eaten every foods. My second model seems pretty tough. It also used reflexagents which can both offend and defend. It was in the same way with the baseline model. I couldn't win the baseline with the dramatic score, but it is more than 50 percent as I tried to make.

## 4. Conclusion

In soccer game, mostly preferred formation is 4-3-3 or 3-4-2-1. I think it's perfectly fit with this mini-contest's result. If there's 3 agents for this contest, the best strategy will be 1 denfender, 1 mid fieler(libero), and 1 striker.

## 5. Free discussion

Question 1: In some situations, agents are stuck in contact and why is it happen? Answer : I think it's because both teams cannot compromise each other. One team which has less point think that they will definitely lose when they suicide to the enemy. On the other hand, another team which has higher point would think that they will definitely win if they shut in enemy's agents.

Question 2: What was the hardest point of the final project? Answer : I had to make 3 different algorithms so it was really tough. Also, using APIs from several different files also gave me an ordeal.

Question 3: What algorithm do you think would be the best model for this contest? Answer : I think active learning would be the most appropriate method for this contest however, there are some time limits for the agent's moves so I couldn't tried it.