Hanoi University of Science and Technology
University of Information and Communication Technology

PACMAN PROJECT

Phạm Ngọc Quân 20210704
Trần Quốc Đệ 20210179
Phạm Quang Nguyên Hoàng 20214901

December, 2022

CONTENTS

I. ABSTRACT

The explosion of Artificial Intelligence (AI) has resulted in various real-world applications, such as autonomous vehicles, recommendation systems, financial services, and cancer detection. Games have also been an important test of Artificial Intelligence since the 1950s, with the examples of Deep Blue and AlphaZero in Chess or AlphaGo in Go. Pacman has also been a game studied a lot during the development of AI. It is a famous Atari game developed back in 1979 by a nine-person team and then released in 1980 by the former Japanese developer and publisher of arcade video games Namco.

II. PACMAN ENVIRONMENT

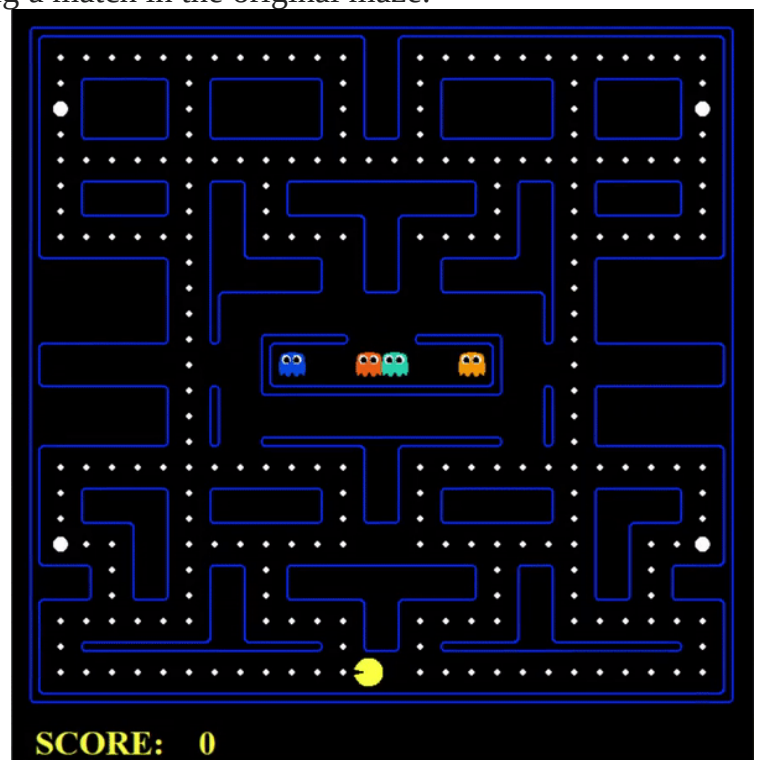
The rule of Pacman is also well-known, but in this problem, we have modified it a little for testing performance efficiency. Firstly, we consider some fundamental component of Pacman environment:

- **Pacman**(the yellow circle with a mouth in the above figure) in this game represents an agent who interacts with the environment (maze) as a move. The mission of Pacman is going to eat as more food (small white dots) as possible while avoiding ghosts.
- **Ghost**(the other two agents with eyes in the above figure) is an adversarial agent of Pacman, which kill Pacman if they collide on the path.
- **Layout**: we obtain many layouts of the maze which define the starting point of Pacman and the ghost, along with the food across the path.
- **Food**: Food was spawned all along the path.Eating food count the point for the game. Especially, a big white dot at the corner are *capsules*, which give Pac-Man the power to eat ghosts in a limited time window.

We briefly explain the rule of Pacman-game as follow:

- The ghosts are all controlled by the same kind of **AI** and there is not a fixed number of ghosts.
- Eating a dot rewards 10 points, each step will decrease by 1 point and the game ends when all the dots have been cleared or when a ghost eats Mr.Pacman, with no extra lives.
- When you eating **capsule**, the ghost are going to be scared. Being scared, ghost will have its speed halved and no more chasing player but run away from pacman agent.
- Eating a **scared ghost** rewards **200** points.
- **Winning** the game rewards **500** points.
- **Losing** the game will subtract 500 points.

In our file, that allow to modify a maze, so there are different kind of maze with different level of difficulties. The video below displaying one of an agent we are going to implement playing a match in the original maze.



III. ALGORITHMS

We consider 4 agents: **Reflex Agent**, **Minimax Agent**, **AlphaBeta Agent** and **Expectimax Agent**.

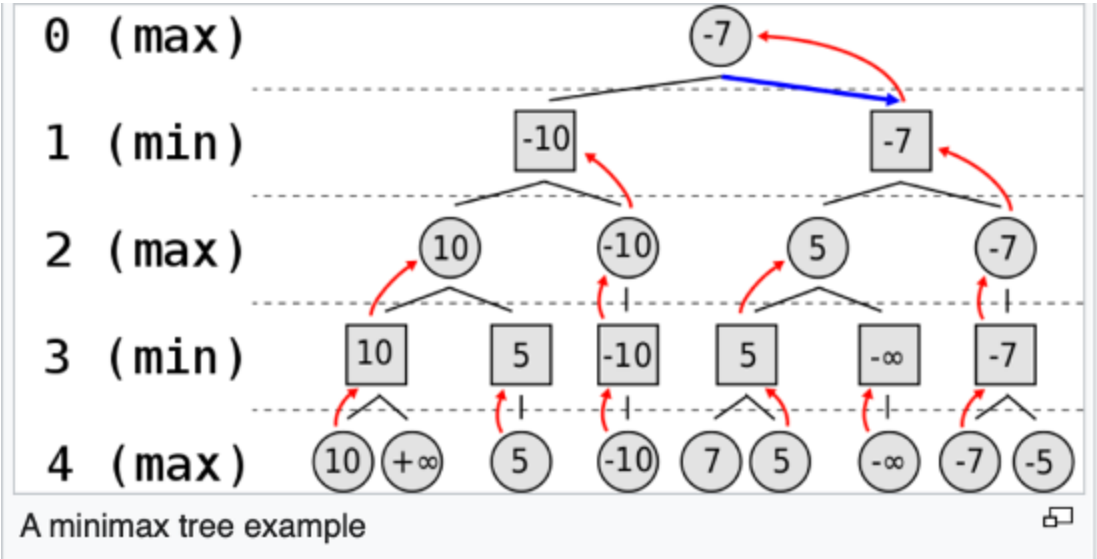
- **Reflex Agent** :

We use greedy algorithms to make decisions in the current state for Mr.Pacman. We define an "evaluationFunction" to estimate the score of the next step after executing an action. And we choose the action among all the legal actions for Mr.Pacman corresponding to the highest score of "evaluationFunction".
For this agent, we have 2 types of ghosts : RandomGhost and DirectionalGhost. For RandomGhost, the ghosts are moving randomly and for DirectionalGhost, the ghosts are trying to catch Mr.Pacman.

- **Minimax Agent** :

We illustrate every state of the game in a game tree. In the game tree, we divide all of the nodes into 2 types: Max node corresponding to Mr.Pacman and Min node corresponding to Ghosts.

This following figure is an example of depth-limited minimax model with 1vs1 game (1 layer of max then 1 layer of min). But our game is harder (1 Pacman vs multiple ghosts) then we need to add more min layers in the model.

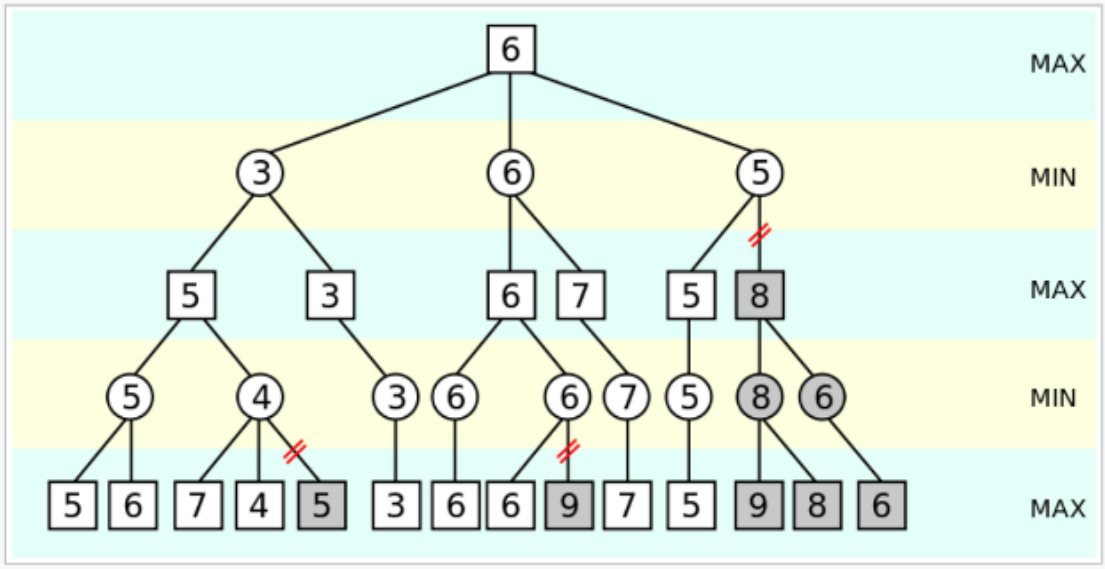


A minimax tree example

The algorithm of Minimax Agent is : We want to find the action for Mr.Pacman at level 0. Consider a depth k, for example k = 4. We compute the estimated score of every game state at that depth. Then we compute the value for Min nodes and Max nodes backward. For the Min node, we find the min value of all its children and assign that value to the Min node. For the Max node, we find the max value of all its children and assign that value to the Max node. The bigger the depth k is, the better performance of Mr.Pacman.

- **AlphaBeta Agent** : (Improvement of naive Minimax)

This agent is based on the model of Minimax Agent but we are adding an idea of branch and bound algorithm to reduce the number of branches in computing.

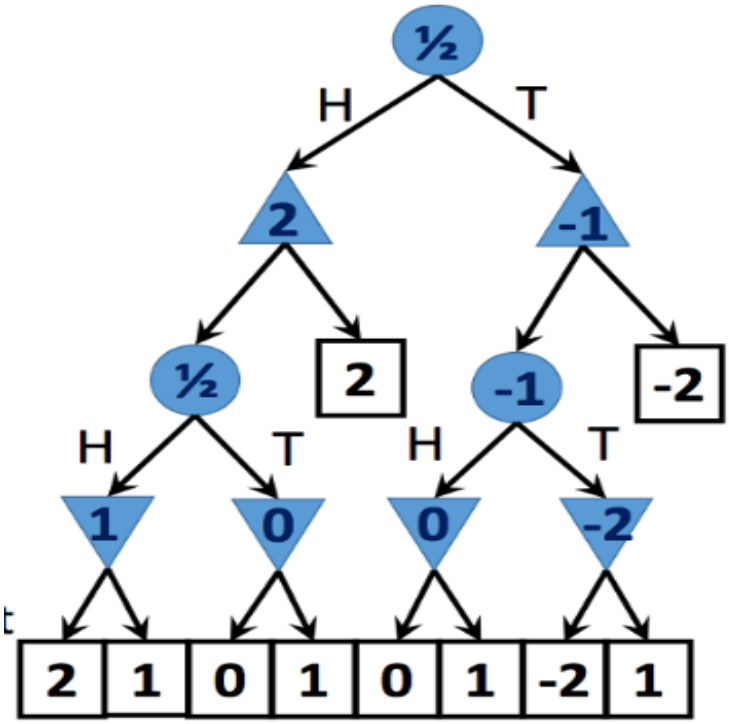


An illustration of alpha–beta pruning. The grayed-out subtrees don't need to be explored (when moves are evaluated from left to right), since it is known that the group of subtrees as a whole yields the value of an equivalent subtree or worse, and as such cannot influence the final result. The max and min levels represent the turn of the player and the adversary, respectively.

The algorithm maintains two values, alpha and beta, which respectively represent the minimum value that the Max node is assured of and the maximum value that the Min node is assured of. Initially, alpha is negative infinity and beta is positive infinity, i.e. both types of nodes start with their worst possible score. Whenever the maximum score that the Min node (i.e. the "beta" player) is assured of becomes less than the minimum score that the Max node (i.e., the "alpha" player) is assured of (i.e. beta < alpha), the Max node need not consider further descendants of this node, as they will never be reached .

- **Expectimax Agent** :

The Expectimax Agent is a variation of Minimax Agent.
In the Expectimax Agent, the Pacman agent no longer assumes ghost agents take actions that minimize score. Instead, Pacman tries to maximize his expected utility and he is playing against multiple Random Ghosts.



IV. IMPLEMENTATION

- For Reflex Agent

The key point is building a good 'evaluationFunction'. The 'evaluationFunction' implemented in the code is based on the distance to the nearest food and the distance to the nearest ghost (here, distance means Manhattan distance). If Mr.Pacman is near the food, he will move in the direction of the food and if he is near the ghosts, he will run away from them.

- Pseudocode for depth-limited Minimax Agent:

```
function minimax( node, depth, maximizingPlayer ) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max( value, minimax( child, depth − 1, FALSE ) )
        return value
    else (* minimizing player *)
        value := +∞
```

```
        for each child of node do
            value := min( value, minimax( child, depth − 1, (TRUE |if next layer
    if max ,else FALSE) )
        return value

(* Initial call *)
minimax( origin, depth, TRUE )
```

- Pseudocode for AlphaBeta Agent :

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if value ≥ β then
                break (* β cutoff *)
        return value
    else  (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, (TRUE | if next
    layer is max , else FALSE))
            β := min(β, value)
            if value ≤ α then
                break (* α cutoff *)
        return value

(* Initial call *)
alphabeta(origin, depth, −∞, +∞, TRUE)
```

- Pseudocode for Expectimax Agent :

```
function  expectimax( node, depth, maximizingPlayer ) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
```

```
o             value := max( value, minimax( child, depth − 1, FALSE ) )
o         return value
o     else (* Random player *)
o         List_of_chance_node = []
o         for each child of node do
o
o             value := expectimax( child, depth − 1,
o  (TRUE |if next layer if max ,else FALSE
o             List_of_chance_nodes.append(value)
o         return expectation(list_of_chance_nodes)

o  (* Initial call *)
o  expectimax( origin, depth, TRUE )
```

V. RESULT

Overall, in different maps, the scores, winrate and runtime of Agents is slightly different, but the ratio between these features is similar to each other. Due to the limit of training time and hardware, we only consider the default map named MediumClassic on detail, and give some insteresting statistics from these results. Of course, we also give some other maps in folder "layouts".

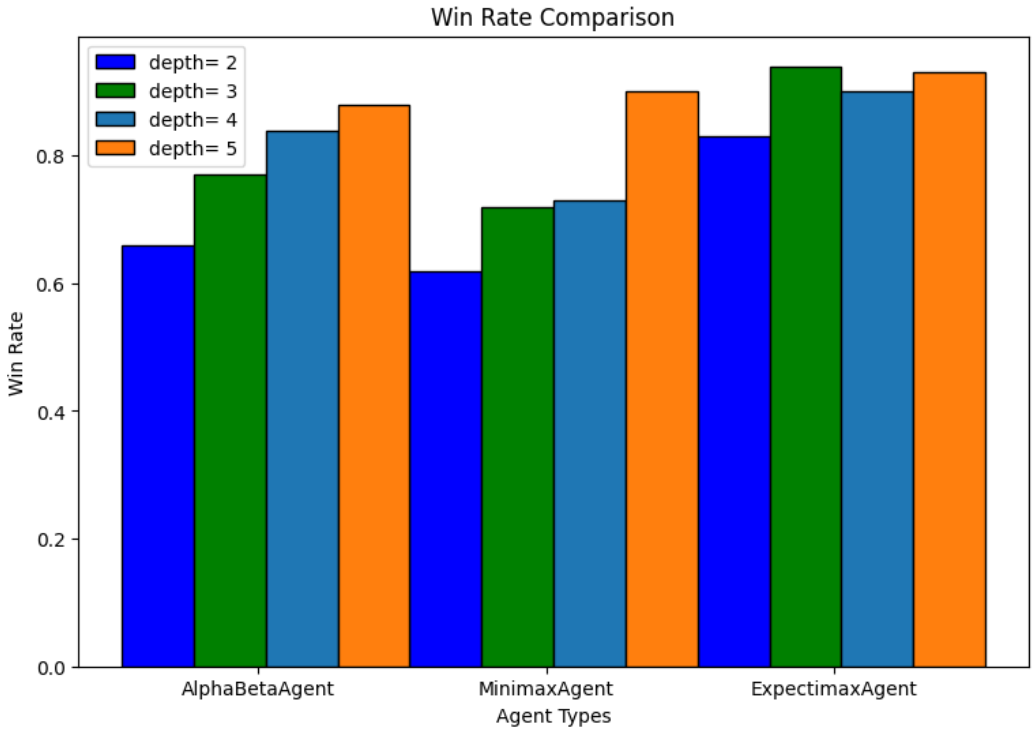
5.1: Experiment procedure

A computer with the following specification was used:

- Processor: Apple M1 Pro
- RAM: 16GB
- Hard Drive: 512GB
- OS: macOS Ventura 13.0

5.2: Winrate Comparison

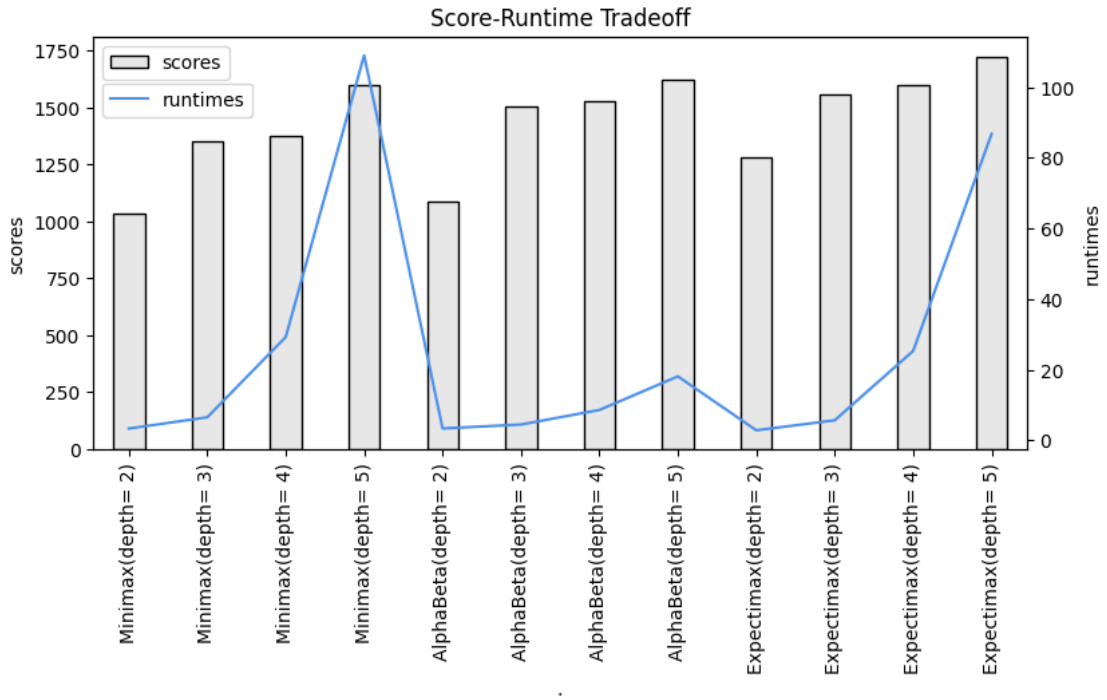
Firstly, we compare winrate between 3 Agents in four different depth – the most essential characteristics of Agents. The performance of ReflexAgent is significant lower than other Agents, so we will consider its performance later in our report.



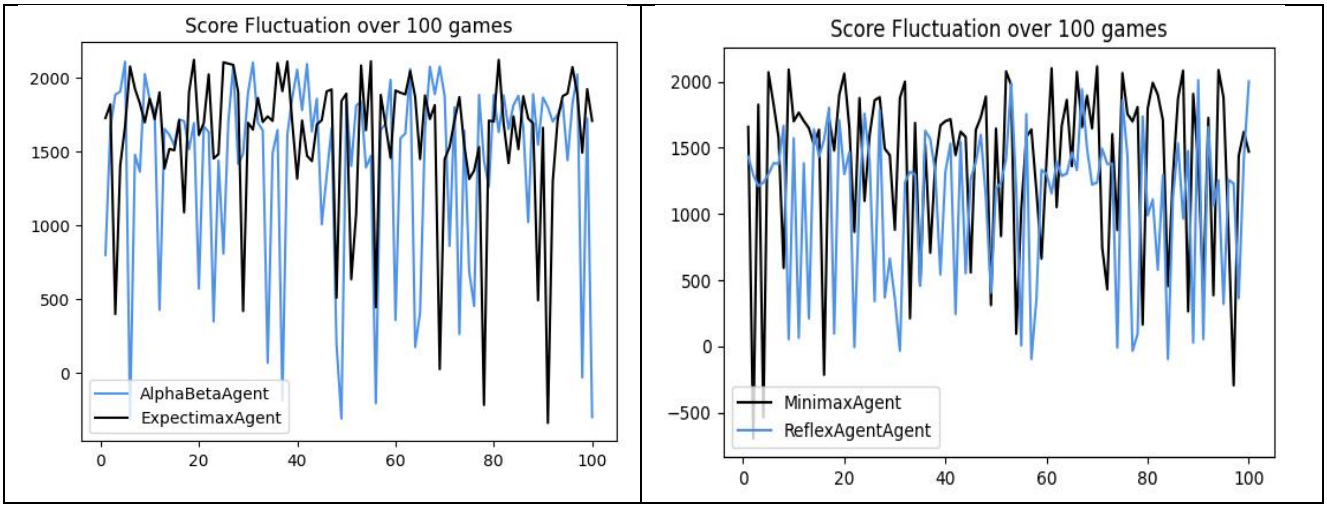
From the chart above, there are not any difference between AlphaBeta Agent and Minimax Agent which share the same depth. The performance of ExpectimaxAgent is slightly higher than these two Agents, we will explain later on 5.3

5.3: Score-Runtime Tradeoff

On the other hand, in general the winrate of Agents increase along with the increasing of its depth in both three Agent type. However, in practice, Agents with the highest winrate does not mean the best performance. The reason is the dramatic growth of runtime goes along with the slightly increase of win rate, score in high-depth Agent. It is called “Score-Runtime Tradeoff”. After training 500 games for each Agent in different depth, we calculate runtime and scores average and using for visualize this phenomenon in the double-axis chart below:



According to these chart above, we can see that between depth-4 and depth-5 Agents, the increase of scores is not propotional to the increase of runtime. On the other hand, despite the lower in runtime, the score average of depth-3 Agent is similar to depth-4 Agent. Therefore, we will choose depth-3 Agent to represent for all 3 types of Agent. We will consider 3 Agent above as well as RelexAgent in detail and visualize in the chart below:



5.4: Explain the result

The performance of agents are in the following order:

Expectimax Agent > Reflex Agent (random ghost)

Expectimax Agent > AlphaBeta Agent > Minimax Agent > Reflex Agent (directional ghost)

Reason why expectimax better than reflex (random ghosts) or minimax better than reflex (directional ghosts) is obvious. Intuitively, with expectimax and minimax, Mr.Pacman can predict the near future so it helps Mr.Pacman make better decision than just take only the current situation into account in reflex agent.

Reason why expectimax better than alpha beta is : In alpha beta agent, the ghosts are moving optimally, not random as in expectimax agent.

Reason why AlphaBeta > Minimax: Because we prune branches in GameState Tree, then the running time for alphabeta is smaller than minimax, which means the performance is better. Theoretically, winning rate and score of these 2 agent are the same (but in the implementation, there are a lot of random issues, that why we need to run a lot of games to see the similarity of them)

VI. CONCLUSION AND POSSIBLE EXTENSIONS

In conclusion, we have successfully solved our problems using four classical Agents: ReflexAgent, MinimaxAgent, AlphaBetaAgent and ExpectimaxAgent with a reasonable running time. It provides a trade-off value between good running time and their performance, including scores and win rate. Out of our algorithms, we conclude that using Expectimax version brings the best result. By achieving this project, we also gained a lot of experience about team work, understood the algorithms described in the class, and understood how to implement them clearly. This project is the first project for us at the university and finishing it boosted our confidence a lot.

If we had more time, we will write my custom distance function using BFS, instead of using Mahattan distance version. These function will help Pacman agents improve their performance in some diffucult maps, which contains a lot of walls and impassable paths. Moreover, we will increase the efficiency of EvaluationFunction, which is the core-function for all agents, by adding more features such as nearest Capsule distance, the amount of food, capsule remaining,… and return the linear combination of these features. And then we will find the optimal weight for each feature by training the agents or by hyperparameter tuning throughout numerous of experiments.

VII. LISTS OF TASKS

7.1: Programming tasks

Trần Quốc Đệ: ReflexAgent, ExpectimaxAgent

Phạm Ngọc Quân: AlphaBetaAgent

Phạm Quang Nguyên Hoàng: MinimaxAgent

7.2: Analytic tasks

Phạm Quang Nguyên Hoàng: Abstract, Pacman Environment

Trần Quốc Đệ: Algorithms, Implementation

Phạm Ngọc Quân: Result, Conclusion and Possible Extensions

REFERENCES

[1] Project 2: Multi-Agent Search - UC Berkeley CS188 Intro to AI.

https://ai.berkeley.edu/multiagent.html

[2] Minimax – Wikipedia

https://en.wikipedia.org/wiki/Minimax

[3] Alpha-Beta Pruning – Wikipedia

https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning