Introduction

The is a project aimed at creating an autonomous Pacman game where Pacman's movements and decision-making are generated by various algorithms. This report provides an overview of the approaches used for different tasks within the game, including pathfinding and decision-making.

Objective

The primary objective of this project is to demonstrate the capability of various algorithms to control Pacman's movements efficiently and intelligently in different simulated game environments.

Algorithms

Part 1: Single Agent Search

Task 1a: Single Food Pellet Search

In Task 1a, the primary objective is to locate a single food pellet within the game map while optimising the shortest route. To fulfil this task, the A* algorithm was judiciously selected. A* algorithm employs a heuristic function, in this case, the Manhattan distance between Pacman and the food pellet. This heuristic is particularly favoured for its admissibility, ensuring it never overestimates the cost-to-go. The algorithm maintains both open and closed lists, containing vital information about the current state, Pacman's position, the Manhattan distance to the food pellet, required actions, and the cost to reach that state. The open list is systematically sorted based on estimated cost, a fusion of the cost to reach a state and the Manhattan distance to the food pellet. This sort order dictates the exploration sequence. Meanwhile, the closed list records visited states, and if a state is encountered again, the algorithm checks for a lower cost to reach it before reintroducing it to the open list.

Task 1b & 1c: Multiple Food Pellet Search

Since task 1b and 1c has similar requirements, which is to search for multiple food pellets dispersed throughout the map, I have used the same approach to solve them. To address these tasks, the A* algorithm is once again employed. However, the primary distinction lies in the dynamic nature of food pellet positions, represented as a list. In each iteration, Pacman targets the nearest food pellet, removing it from the list upon discovery, and subsequently continuing the search.

Part 2: Adversarial Search

Task 2a: Alpha-Beta Pruning Algorithm

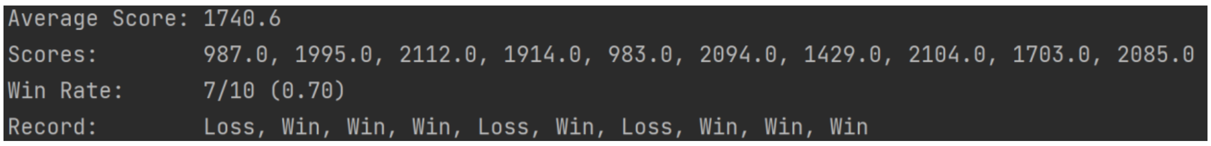
Task 2a necessitates the implementation of the Alpha-Beta pruning algorithm, set with a search depth of 3. This algorithm delves into the game tree through recursive exploration, assessing various game states, while keeping track of alpha (the best value for Pacman) and beta (the best value for the ghosts). It minimises the number of nodes evaluated by cutting off branches that are guaranteed to be worse than the current best option for Pacman. In addition to evaluating positions based solely on current game scores, an auxiliary heuristic function comes into play for tie-breaking purposes. This heuristic rewards positions with shorter Manhattan distances between Pacman and the nearest food pellet, increasing the likelihood of favorable moves in tied-score scenarios.

Task 2b: Expectimax Algorithm

Task 2b introduces the Expectimax algorithm, another variation of the Minimax algorithm. Like Alpha-Beta pruning, Expectimax explores the game tree through recursive traversal. Positions are evaluated using the same heuristic function. However, a critical distinction lies in the treatment of ghost moves. Given that ghosts do not consistently choose the optimal actions, Expectimax considers the range of potential moves for ghosts, enabling a higher win rate through a more probabilistic approach.

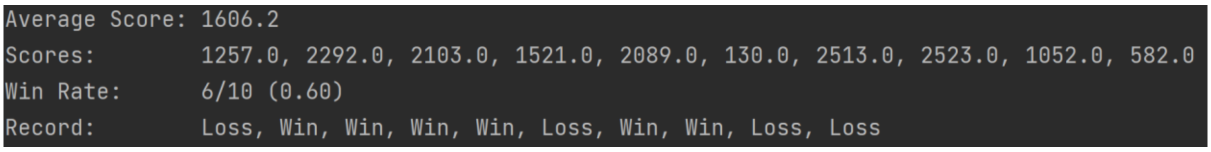
Depth Selection Analysis

A critical parameter in the Expectimax algorithm is the depth of the search tree, a factor that directly impacts the quality of Pacman's decision-making. To pinpoint the optimal depth, an extensive series of experiments was conducted, encompassing depth values ranging from 2 to 15. These experiments rigorously assessed how different depths influences gameplay performance. The results, gathered from 10 game runs for each depth, provide invaluable insights into the trade-offs and advantages inherent in each choice. Out of the 14 depth values tested, three emerged as the most promising candidates: 9, 10, and 12. Consequently, out analysis will focus exclusively on these three depth values.



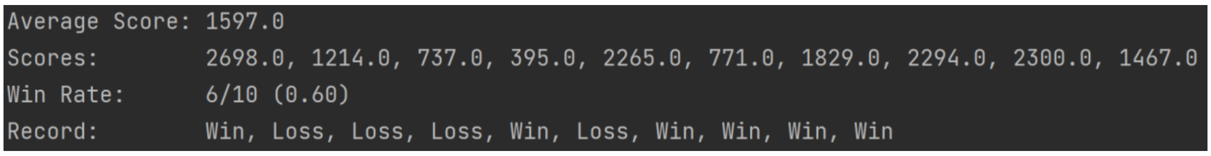
```
Average Score: 1740.6
Scores:        987.0, 1995.0, 2112.0, 1914.0, 983.0, 2094.0, 1429.0, 2104.0, 1703.0, 2085.0
Win Rate:      7/10 (0.70)
Record:        Loss, Win, Win, Win, Loss, Win, Loss, Win, Win, Win
```

Figure 1: Result of the Expectimax algorithm with a depth of 9



```
Average Score: 1606.2
Scores:        1257.0, 2292.0, 2103.0, 1521.0, 2089.0, 130.0, 2513.0, 2523.0, 1052.0, 582.0
Win Rate:      6/10 (0.60)
Record:        Loss, Win, Win, Win, Win, Loss, Win, Win, Loss, Loss
```

Figure 2: Result of the Expectimax algorithm with a depth of 10



```
Average Score: 1597.0
Scores:        2698.0, 1214.0, 737.0, 395.0, 2265.0, 771.0, 1829.0, 2294.0, 2300.0, 1467.0
Win Rate:      6/10 (0.60)
Record:        Win, Loss, Loss, Loss, Win, Loss, Win, Win, Win, Win
```

Figure 3: Result of the Expectimax algorithm with a depth of 10

While both a depth of 12 and a depth of 10 produced higher maximum scores (2698.0 and 2523.0, respectively) compared to a depth of 9, which achieved a maximum score of 2112.0, it is noteworthy that the Expectimax algorithm with a depth of 9 consistently delivered a higher average score, specifically 1740.6, surpassing both a depth of 12 (averaging 1597.0) and a depth of 10 (averaging 1606.2). Furthermore, opting a depth of 9 also resulted in a higher win rate, with Pacman securing victory in 7 out of 10 games. This achievement stands in contrast to a depth of 12 and a depth of 10, which both attained a win rate of 6 out of 10 games. This outcome underscores the enhanced likelihood of Pacman winning when navigating the search tree to a depth of 9.

In summary, while both depths 12 and 10 yield competitive results, it is abundantly clear that a depth of 9 consistently outperforms them across various metrics, establishing it as the preferred choice for the algorithm.

Acknowledgments

- ChatGPT: Discussions pertaining to heuristic functions and A* algorithm implementations for TSP problems, as well as indispensable assistance in proofreading and refining the report
- "Playing Pacman with Multi-Agents Adversarial Search" by David Liu, February 2020: provided inspiration and guidance for part 2 of the assignment.