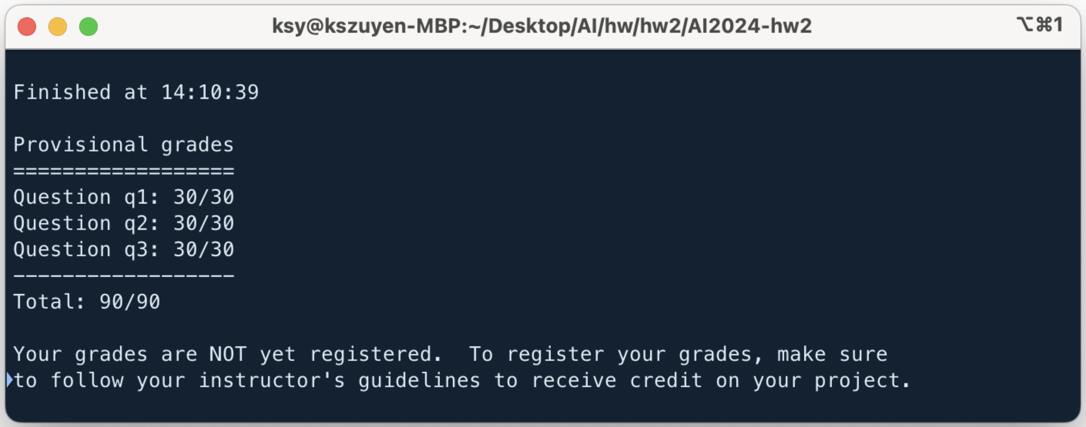
2024 AI - HW#2 Multi-Agent Search

生醫電資所 碩一 r12945040 郭思言

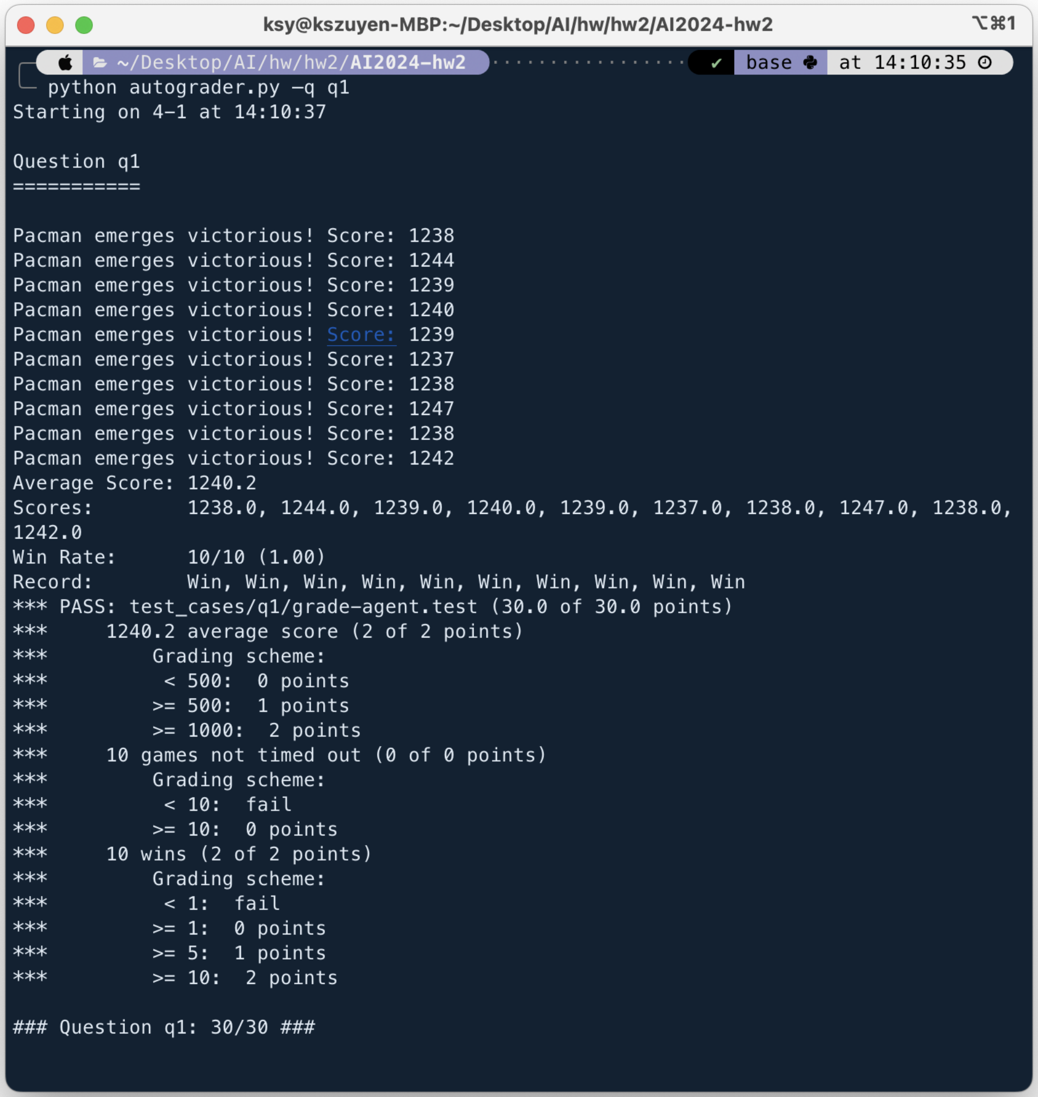
Show your autograder results and describe each algorithm



ksy@kszuyen-MBP:~/Desktop/AI/hw/hw2/AI2024-hw2

```
Finished at 14:10:39

Provisional grades
==================
Question q1: 30/30
Question q2: 30/30
Question q3: 30/30
------------------
Total: 90/90

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

- **Q1. Reflex Agent**



```
ksy@kszuyen-MBP:~/Desktop/AI/hw/hw2/AI2024-hw2                                    ⌥⌘1
   ~/Desktop/AI/hw/hw2/AI2024-hw2 ·················  ✓  base   at 14:10:35
╰─ python autograder.py -q q1
Starting on 4-1 at 14:10:37

Question q1
===========

Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1240
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1237
Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1247
Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1242
Average Score: 1240.2
Scores:        1238.0, 1244.0, 1239.0, 1240.0, 1239.0, 1237.0, 1238.0, 1247.0, 1238.0,
1242.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q1/grade-agent.test (30.0 of 30.0 points)
***     1240.2 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (0 of 0 points)
***         Grading scheme:
***          < 10:  fail
***         >= 10:  0 points
***     10 wins (2 of 2 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  0 points
***         >= 5:  1 points
***         >= 10:  2 points

### Question q1: 30/30 ###
```

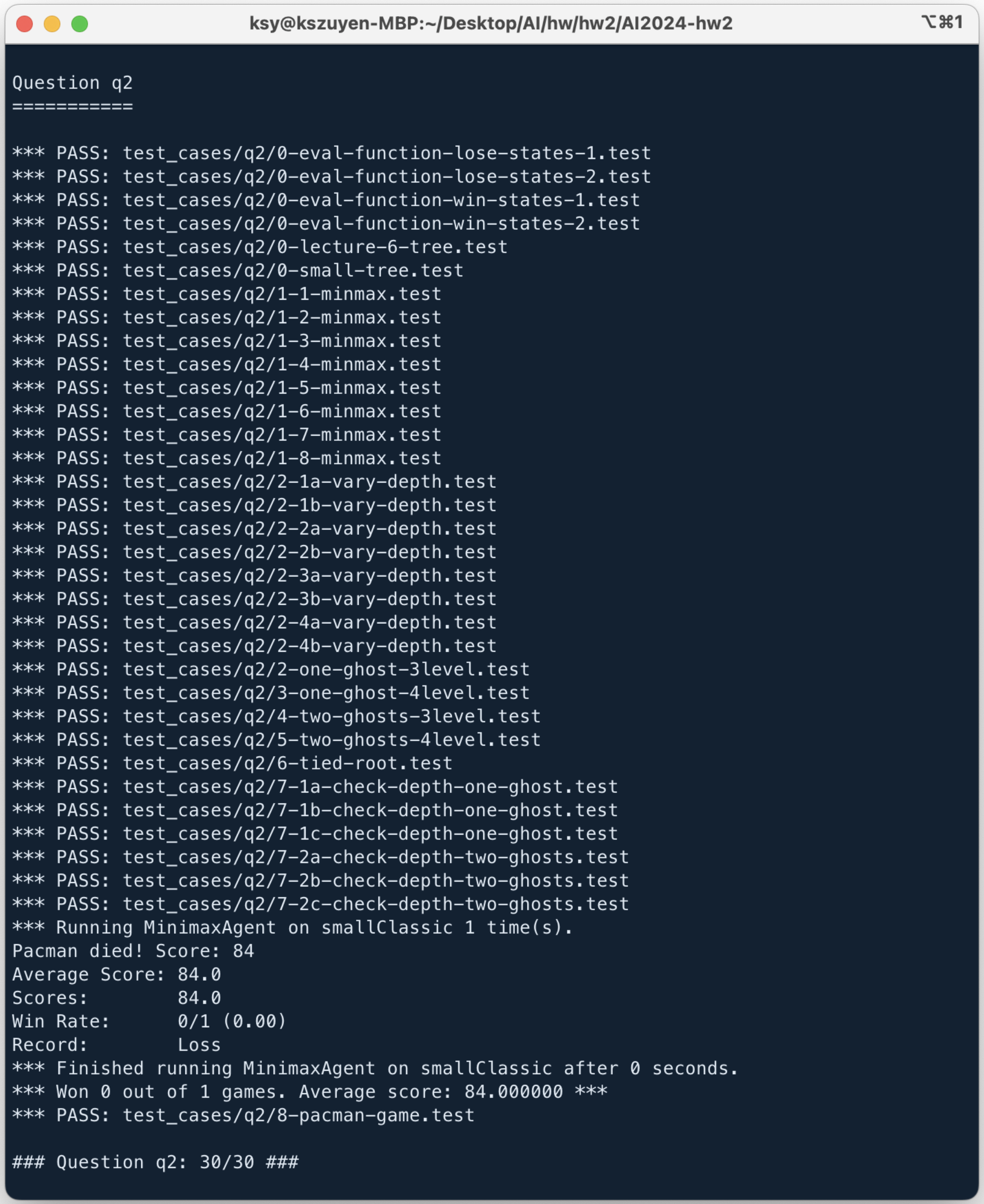
The reflex agent only considers the immediate state of the game when making decisions. It does not plan ahead or consider future consequences of its actions. The effectiveness of this agent depends heavily on the quality of the evaluation function, which should capture the key aspects of the game that lead to success, such as avoiding ghosts and eating food pellets.Here's a breakdown of how it works:

1. **Initialization**: The agent is provided with a `GameState` object, which represents the current state of the game. This includes information such

as Pac-Man's position, the positions of ghosts, the layout of the maze, and the locations of food pellets.

2. **Legal Moves**: The agent first collects all legal moves it can make in the current state. These moves typically include moving in any direction (North, South, East, West) or staying still (Stop).
3. **Evaluation Function**: For each legal move, the agent calculates a score using an evaluation function. This function examines the successor state that would result from taking the move and assigns a score based on how favorable that state is.
4. **Choosing the Best Move**: After calculating scores for all legal moves, the agent selects one of the best moves. If there are multiple moves with the same highest score, it randomly chooses one of them.
5. **Executing the Move**: The agent returns the chosen move, which is then executed in the game. The game advances to the next state, and the process repeats.
6. **Game Progression**: As the game progresses, the agent continues to make decisions at each step based on the current state of the game, always aiming to maximize its score according to the evaluation function.

- **Q2. Minimax**



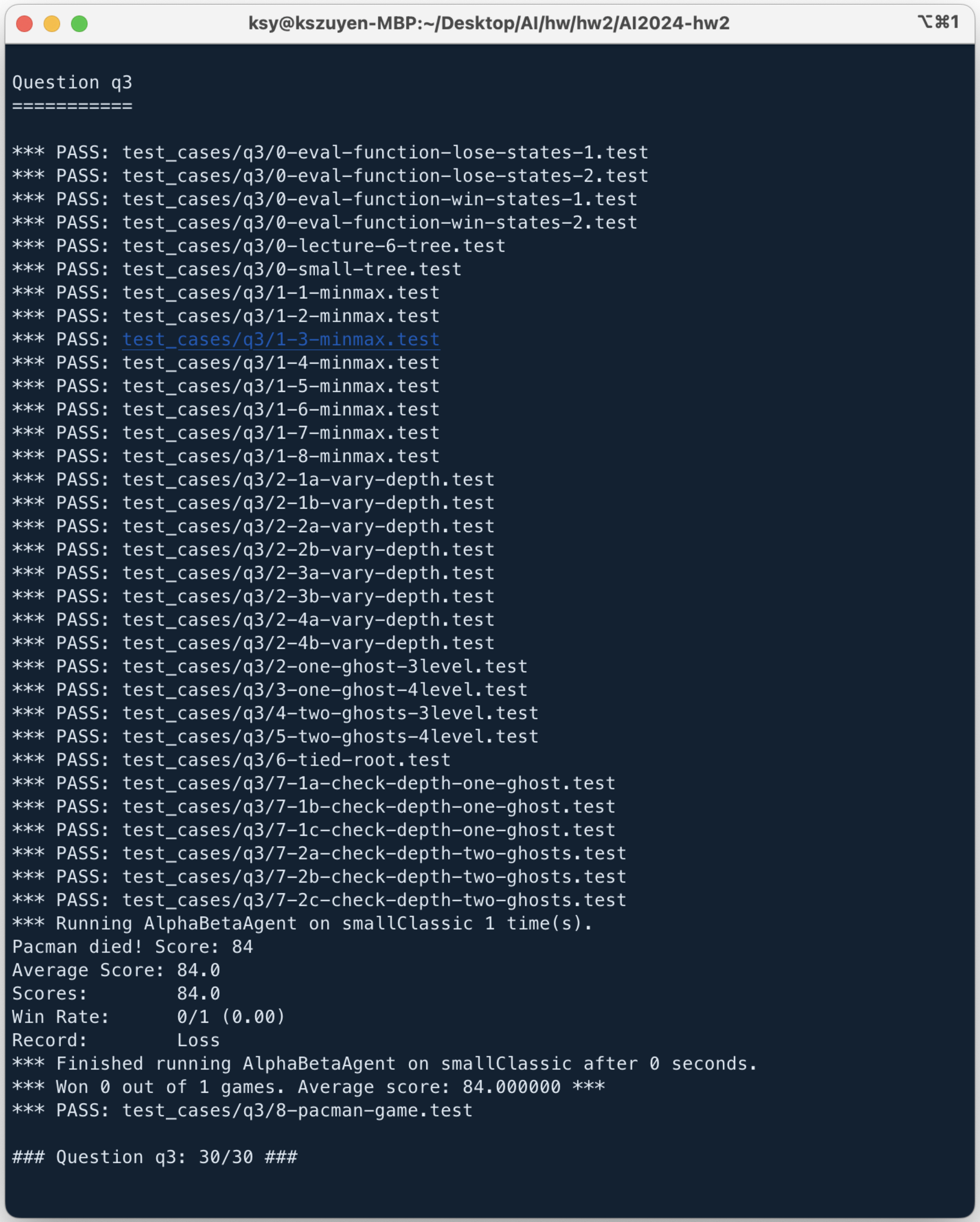
```
Question q2
===========

*** PASS: test_cases/q2/0-eval-function-lose-states-1.test
*** PASS: test_cases/q2/0-eval-function-lose-states-2.test
*** PASS: test_cases/q2/0-eval-function-win-states-1.test
*** PASS: test_cases/q2/0-eval-function-win-states-2.test
*** PASS: test_cases/q2/0-lecture-6-tree.test
*** PASS: test_cases/q2/0-small-tree.test
*** PASS: test_cases/q2/1-1-minmax.test
*** PASS: test_cases/q2/1-2-minmax.test
*** PASS: test_cases/q2/1-3-minmax.test
*** PASS: test_cases/q2/1-4-minmax.test
*** PASS: test_cases/q2/1-5-minmax.test
*** PASS: test_cases/q2/1-6-minmax.test
*** PASS: test_cases/q2/1-7-minmax.test
*** PASS: test_cases/q2/1-8-minmax.test
*** PASS: test_cases/q2/2-1a-vary-depth.test
*** PASS: test_cases/q2/2-1b-vary-depth.test
*** PASS: test_cases/q2/2-2a-vary-depth.test
*** PASS: test_cases/q2/2-2b-vary-depth.test
*** PASS: test_cases/q2/2-3a-vary-depth.test
*** PASS: test_cases/q2/2-3b-vary-depth.test
*** PASS: test_cases/q2/2-4a-vary-depth.test
*** PASS: test_cases/q2/2-4b-vary-depth.test
*** PASS: test_cases/q2/2-one-ghost-3level.test
*** PASS: test_cases/q2/3-one-ghost-4level.test
*** PASS: test_cases/q2/4-two-ghosts-3level.test
*** PASS: test_cases/q2/5-two-ghosts-4level.test
*** PASS: test_cases/q2/6-tied-root.test
*** PASS: test_cases/q2/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q2/8-pacman-game.test

### Question q2: 30/30 ###
```

The goal of the minimax algorithm is to find the best move for the maximizing player (Pac-Man) assuming that the minimizing player (the ghosts) will also make optimal moves.

Here's a brief overview of how the algorithm works in this implementation:

1. The `getAction` method is called to determine the best move for Pac-Man from the current game state (`gameState`). It uses the `self.depth` attribute to limit the depth of the search and the `self.evaluationFunction` to evaluate the game state.
2. The `miniMax` function is a helper function that performs the minimax algorithm. It iterates over all legal moves for Pac-Man and calls the `minValue` function to find the best move for the ghosts.
3. The `maxValue` function is called when it's Pac-Man's turn to move. It recursively explores all possible moves up to a certain depth (`self.depth`) and returns the maximum value found.
4. The `minValue` function is called when it's the ghosts' turn to move. It recursively explores all possible moves for each ghost and returns the minimum value found.
5. The algorithm alternates between maximizing and minimizing players until it reaches the specified depth (`self.depth`). At this point, it uses the `evaluationFunction` to evaluate the leaf nodes and returns the best move for Pac-Man.

- **Q3. Alpha-Beta Pruning**



ksy@kszuyen-MBP:~/Desktop/AI/hw/hw2/AI2024-hw2

```
Question q3
===========

*** PASS: test_cases/q3/0-eval-function-lose-states-1.test
*** PASS: test_cases/q3/0-eval-function-lose-states-2.test
*** PASS: test_cases/q3/0-eval-function-win-states-1.test
*** PASS: test_cases/q3/0-eval-function-win-states-2.test
*** PASS: test_cases/q3/0-lecture-6-tree.test
*** PASS: test_cases/q3/0-small-tree.test
*** PASS: test_cases/q3/1-1-minmax.test
*** PASS: test_cases/q3/1-2-minmax.test
*** PASS: test_cases/q3/1-3-minmax.test
*** PASS: test_cases/q3/1-4-minmax.test
*** PASS: test_cases/q3/1-5-minmax.test
*** PASS: test_cases/q3/1-6-minmax.test
*** PASS: test_cases/q3/1-7-minmax.test
*** PASS: test_cases/q3/1-8-minmax.test
*** PASS: test_cases/q3/2-1a-vary-depth.test
*** PASS: test_cases/q3/2-1b-vary-depth.test
*** PASS: test_cases/q3/2-2a-vary-depth.test
*** PASS: test_cases/q3/2-2b-vary-depth.test
*** PASS: test_cases/q3/2-3a-vary-depth.test
*** PASS: test_cases/q3/2-3b-vary-depth.test
*** PASS: test_cases/q3/2-4a-vary-depth.test
*** PASS: test_cases/q3/2-4b-vary-depth.test
*** PASS: test_cases/q3/2-one-ghost-3level.test
*** PASS: test_cases/q3/3-one-ghost-4level.test
*** PASS: test_cases/q3/4-two-ghosts-3level.test
*** PASS: test_cases/q3/5-two-ghosts-4level.test
*** PASS: test_cases/q3/6-tied-root.test
*** PASS: test_cases/q3/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q3/8-pacman-game.test

### Question q3: 30/30 ###
```

Alpha-beta pruning is an optimization technique to reduce the number of nodes evaluated in the search tree.

1. **Alpha-Beta Pruning**:

    - Alpha represents the best (highest) value that the maximizing player (Pac-Man) can currently achieve.
    - Beta represents the best (lowest) value that the minimizing player (ghosts) can currently achieve.
    - During the search, if at any point the current node's value exceeds the alpha (for the maximizing player) or beta (for the minimizing player) bounds, the algorithm can prune the subtree below that node, as it will not affect the final decision.
2. **`maxValue` and `minValue` Functions**:
    - They recursively explore all possible moves up to a certain depth (`self.depth`) and update the alpha and beta values accordingly.
3. **Pruning Condition**:
    - If the current score exceeds the beta value in a maximizing node or falls below the alpha value in a minimizing node, the algorithm prunes the subtree below that node, as it will not affect the final decision.
4. **Final Move Selection**:
    - Once the search is complete, the algorithm selects the best move for the current player based on the utility values propagated from the leaf nodes.

Describe the idea of your design about evaluation function in Q1.

The evaluation function encourages Pac-Man to prioritize eating food pellets and power pellets while avoiding ghosts. It considers several factors to determine the score:

1. **Pac-Man Position and Ghosts**:
    - Calculates the distance between Pac-Man's new position (`newPos`) and each ghost's position. If Pac-Man is too close to a ghost (`nearestGhost <= 1`), then return a very low score (`maxInt`) to discourage this move.
2. **Food**:

- It calculates the distance between Pac-Man's new position and the nearest food pellet (`nearestFood`). It favors moves that bring Pac-Man closer to food by subtracting this distance from a maximum value (`maxInt`).
- If Pac-Man's new position overlaps with a food (i.e., there is food at the new position) and Pac-Man is not too close to a ghost, it returns a very high score (`maxInt`). This encourages Pac-Man to eat food when safe.

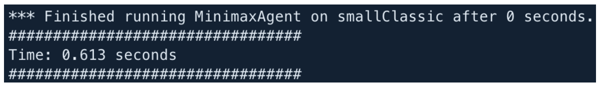
Demonstrate the speed up after the implementation of pruning.

Run **Minimax** and **AlphaBetaPruning** each, and keep track of the time:

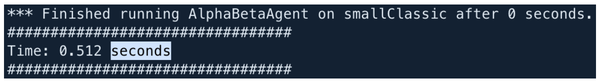
	Minimax	AlphaBetaPruning
1	0.613	0.512
2	0.631	0.531
3	0.623	0.525
4	0.612	0.511
5	0.623	0.522
Average	**0.6204**	**0.5202**

From the data, we can see that on average, Alpha-Beta Pruning is faster than Minimax. The average time for Minimax is approximately 0.6204 seconds, while the average time for Alpha-Beta Pruning is approximately 0.5202 seconds. This demonstrates the speedup achieved by implementing Alpha-Beta Pruning over Minimax.

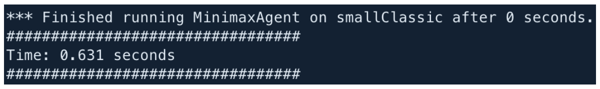
Supplementary images:



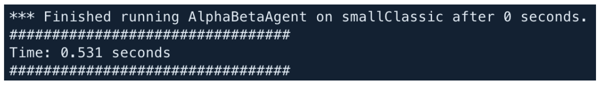
```
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
#################################
Time: 0.613 seconds
#################################
```



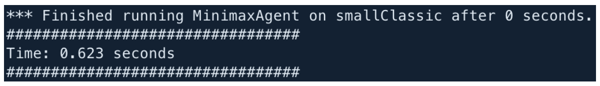
```
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
#################################
Time: 0.512 seconds
#################################
```



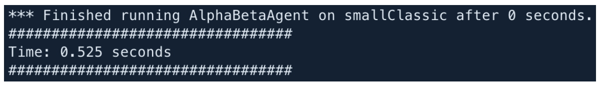
```
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
#################################
Time: 0.631 seconds
#################################
```



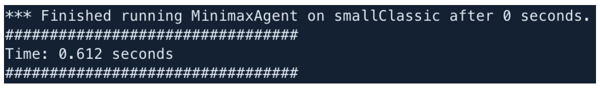
```
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
#################################
Time: 0.531 seconds
#################################
```



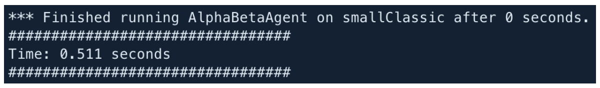
```
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
#################################
Time: 0.623 seconds
#################################
```



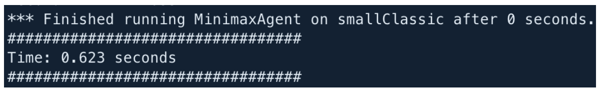
```
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
#################################
Time: 0.525 seconds
#################################
```



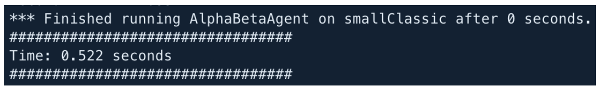
```
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
#################################
Time: 0.612 seconds
#################################
```



```
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
#################################
Time: 0.511 seconds
#################################
```



```
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
#################################
Time: 0.623 seconds
#################################
```



```
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
#################################
Time: 0.522 seconds
#################################
```