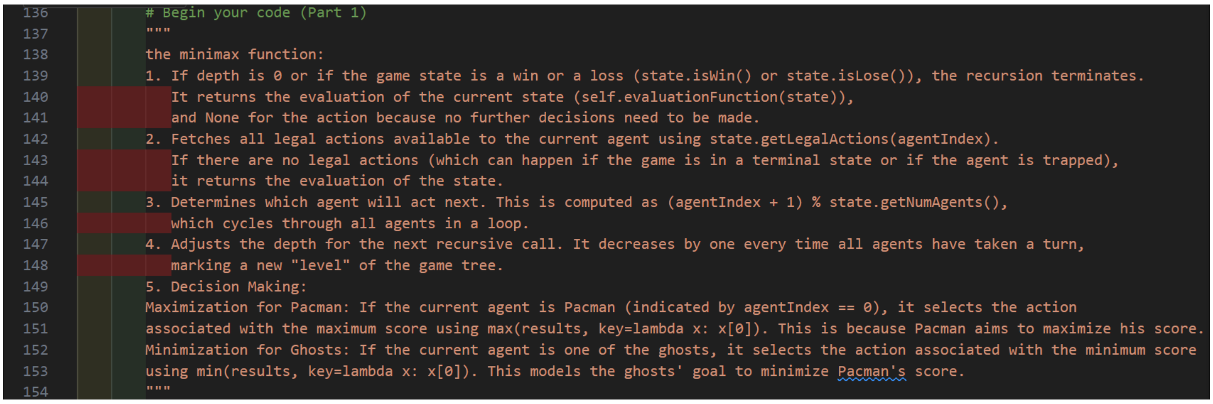
Homework 3: Multi-Agent Search

學號: 111550129　姓名: 林彥亨

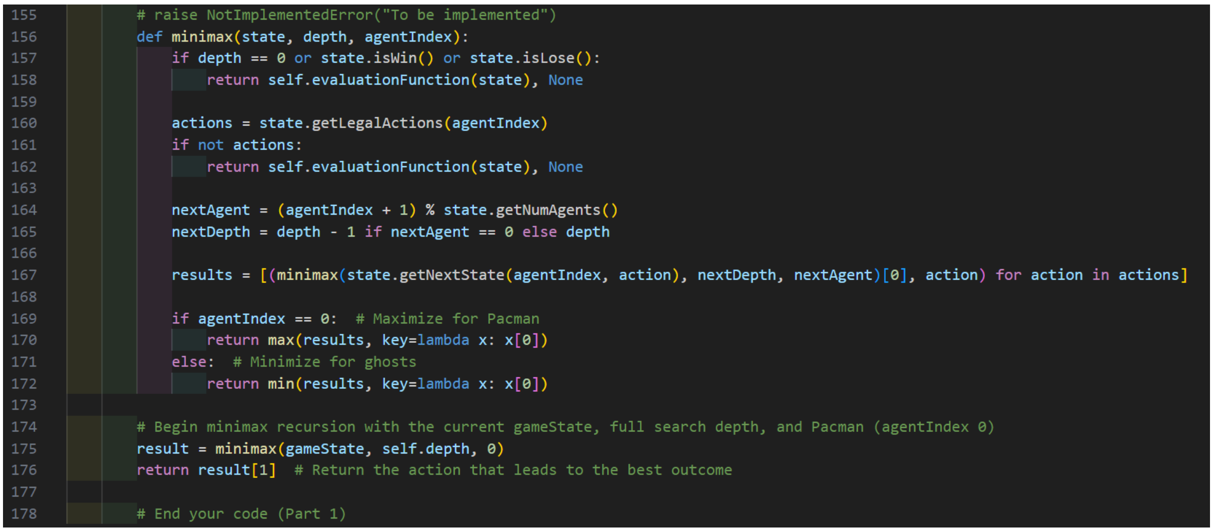
Part 1. Implementation:

Following screenshots are the code from part 1 ~ part 4.

Part 1.

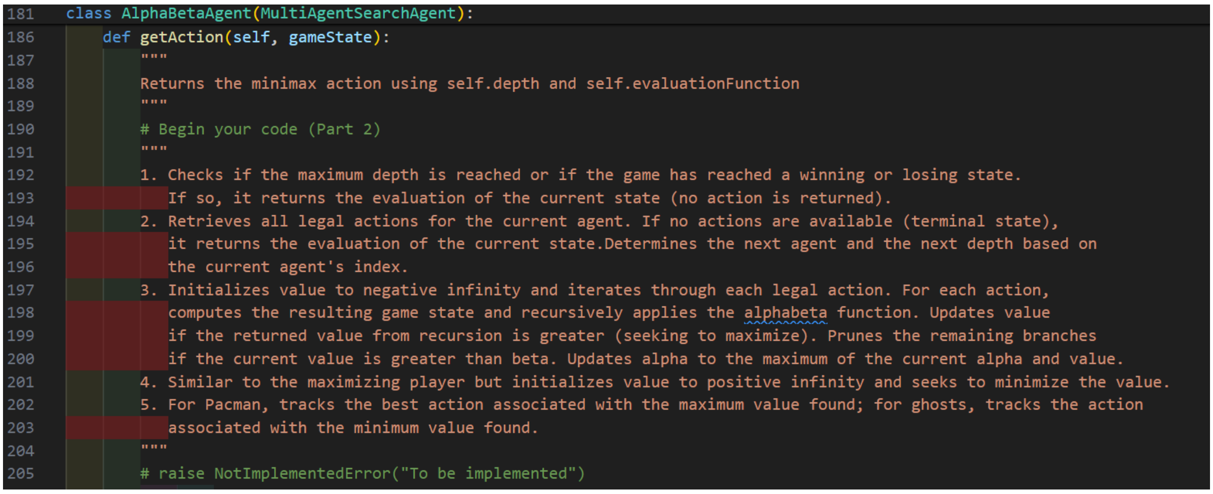


```
        # Begin your code (Part 1)
        """
        the minimax function:
        1. If depth is 0 or if the game state is a win or a loss (state.isWin() or state.isLose()), the recursion terminates.
           It returns the evaluation of the current state (self.evaluationFunction(state)),
           and None for the action because no further decisions need to be made.
        2. Fetches all legal actions available to the current agent using state.getLegalActions(agentIndex).
           If there are no legal actions (which can happen if the game is in a terminal state or if the agent is trapped),
           it returns the evaluation of the state.
        3. Determines which agent will act next. This is computed as (agentIndex + 1) % state.getNumAgents(),
           which cycles through all agents in a loop.
        4. Adjusts the depth for the next recursive call. It decreases by one every time all agents have taken a turn,
           marking a new "level" of the game tree.
        5. Decision Making:
        Maximization for Pacman: If the current agent is Pacman (indicated by agentIndex == 0), it selects the action
        associated with the maximum score using max(results, key=lambda x: x[0]). This is because Pacman aims to maximize his score.
        Minimization for Ghosts: If the current agent is one of the ghosts, it selects the action associated with the minimum score
        using min(results, key=lambda x: x[0]). This models the ghosts' goal to minimize Pacman's score.
        """
```

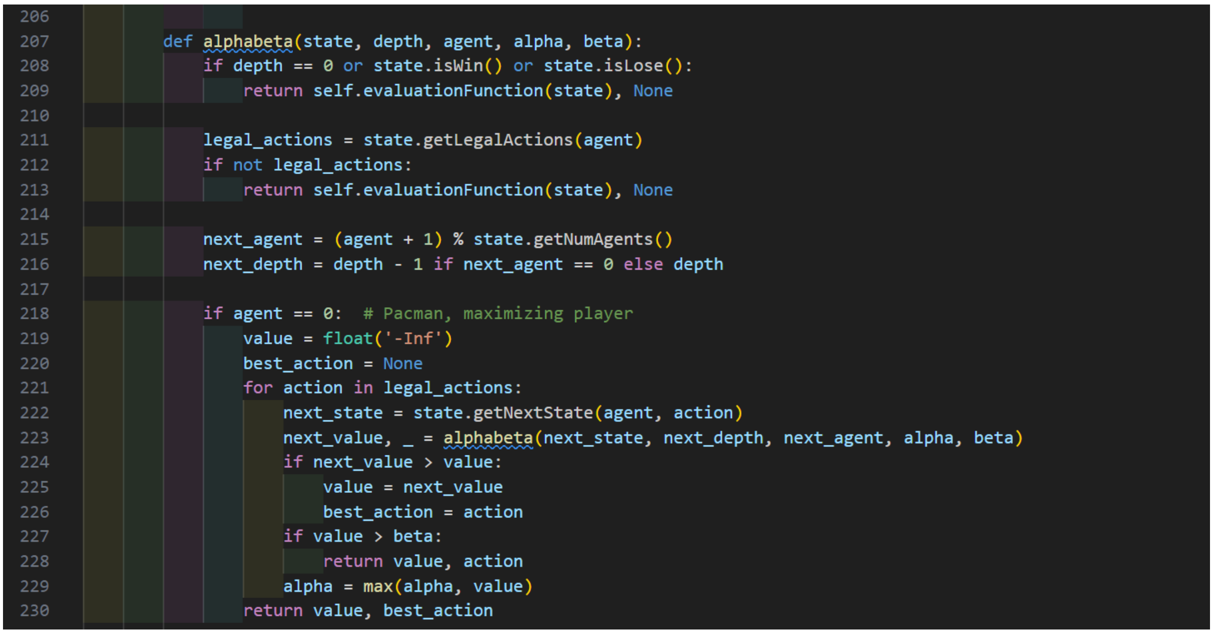


```
        # raise NotImplementedError("To be implemented")
        def minimax(state, depth, agentIndex):
            if depth == 0 or state.isWin() or state.isLose():
                return self.evaluationFunction(state), None

            actions = state.getLegalActions(agentIndex)
            if not actions:
                return self.evaluationFunction(state), None

            nextAgent = (agentIndex + 1) % state.getNumAgents()
            nextDepth = depth - 1 if nextAgent == 0 else depth

            results = [(minimax(state.getNextState(agentIndex, action), nextDepth, nextAgent)[0], action) for action in actions]

            if agentIndex == 0:  # Maximize for Pacman
                return max(results, key=lambda x: x[0])
            else:  # Minimize for ghosts
                return min(results, key=lambda x: x[0])

        # Begin minimax recursion with the current gameState, full search depth, and Pacman (agentIndex 0)
        result = minimax(gameState, self.depth, 0)
        return result[1]  # Return the action that leads to the best outcome

        # End your code (Part 1)
```

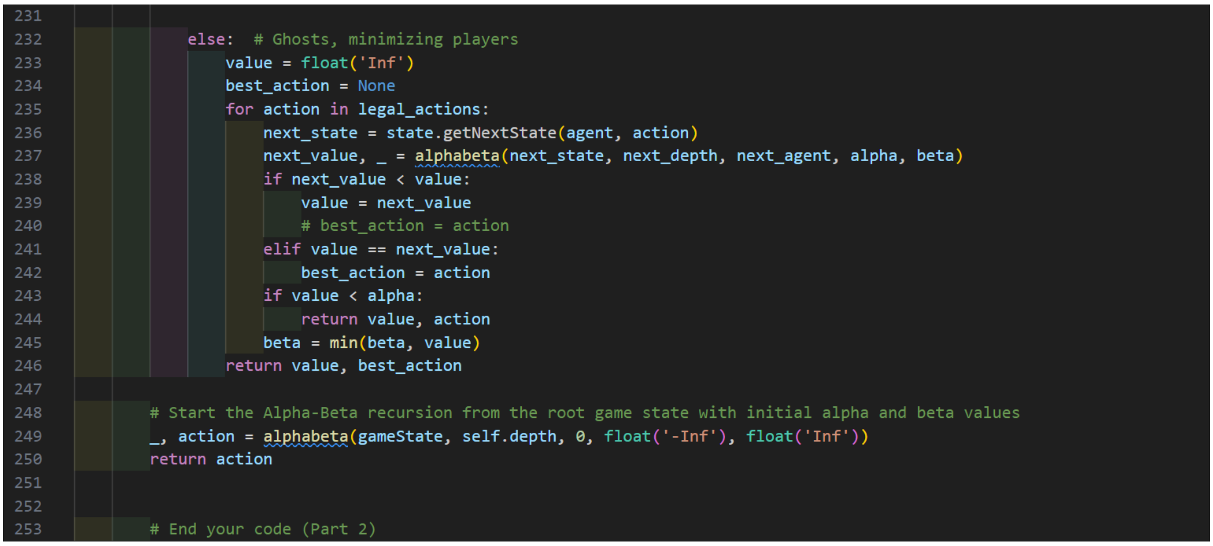
Part 2.



```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    def getAction(self, gameState):
        """
        Returns the minimax action using self.depth and self.evaluationFunction
        """
        # Begin your code (Part 2)
        """
        1. Checks if the maximum depth is reached or if the game has reached a winning or losing state.
           If so, it returns the evaluation of the current state (no action is returned).
        2. Retrieves all legal actions for the current agent. If no actions are available (terminal state),
           it returns the evaluation of the current state.Determines the next agent and the next depth based on
           the current agent's index.
        3. Initializes value to negative infinity and iterates through each legal action. For each action,
           computes the resulting game state and recursively applies the alphabeta function. Updates value
           if the returned value from recursion is greater (seeking to maximize). Prunes the remaining branches
           if the current value is greater than beta. Updates alpha to the maximum of the current alpha and value.
        4. Similar to the maximizing player but initializes value to positive infinity and seeks to minimize the value.
        5. For Pacman, tracks the best action associated with the maximum value found; for ghosts, tracks the action
           associated with the minimum value found.
        """
        # raise NotImplementedError("To be implemented")
```

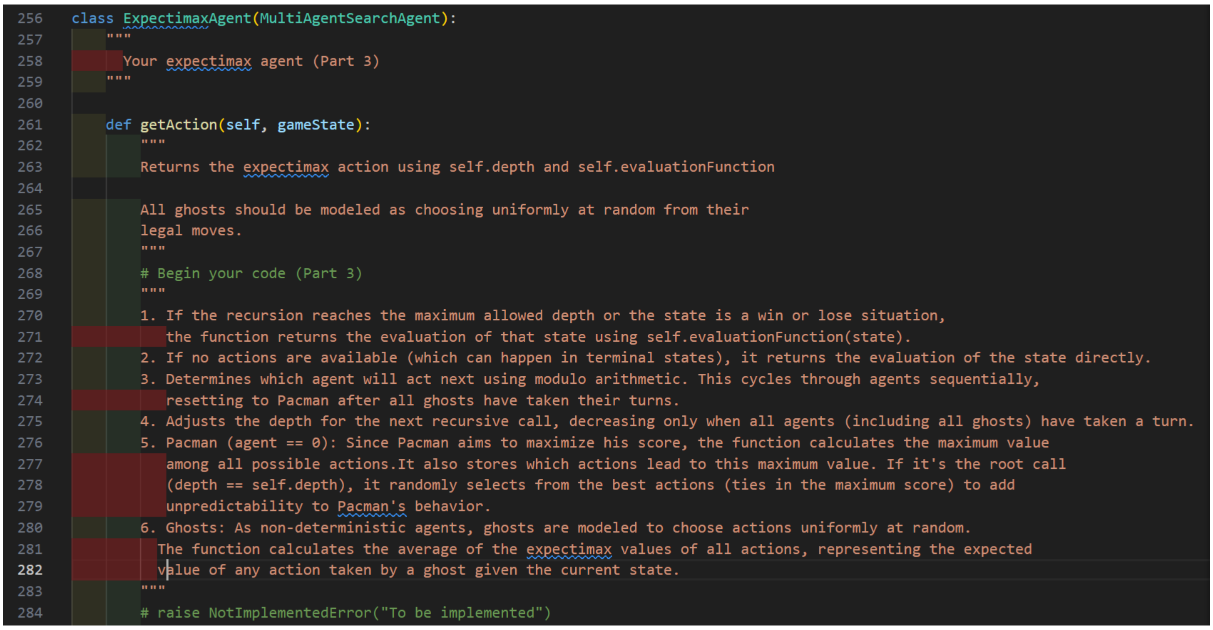


```python

        def alphabeta(state, depth, agent, alpha, beta):
            if depth == 0 or state.isWin() or state.isLose():
                return self.evaluationFunction(state), None

            legal_actions = state.getLegalActions(agent)
            if not legal_actions:
                return self.evaluationFunction(state), None

            next_agent = (agent + 1) % state.getNumAgents()
            next_depth = depth - 1 if next_agent == 0 else depth

            if agent == 0:  # Pacman, maximizing player
                value = float('-Inf')
                best_action = None
                for action in legal_actions:
                    next_state = state.getNextState(agent, action)
                    next_value, _ = alphabeta(next_state, next_depth, next_agent, alpha, beta)
                    if next_value > value:
                        value = next_value
                        best_action = action
                    if value > beta:
                        return value, action
                    alpha = max(alpha, value)
                return value, best_action
```

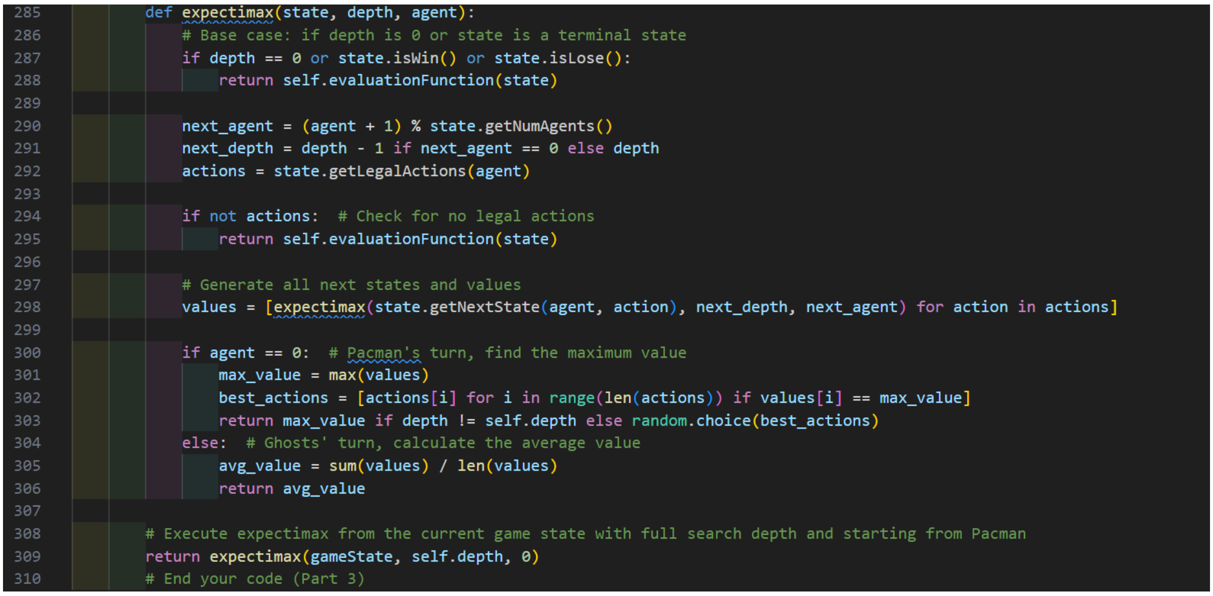


```python

            else:  # Ghosts, minimizing players
                value = float('Inf')
                best_action = None
                for action in legal_actions:
                    next_state = state.getNextState(agent, action)
                    next_value, _ = alphabeta(next_state, next_depth, next_agent, alpha, beta)
                    if next_value < value:
                        value = next_value
                        # best_action = action
                    elif value == next_value:
                        best_action = action
                    if value < alpha:
                        return value, action
                    beta = min(beta, value)
                return value, best_action

        # Start the Alpha-Beta recursion from the root game state with initial alpha and beta values
        _, action = alphabeta(gameState, self.depth, 0, float('-Inf'), float('Inf'))
        return action


        # End your code (Part 2)
```

Part 3.

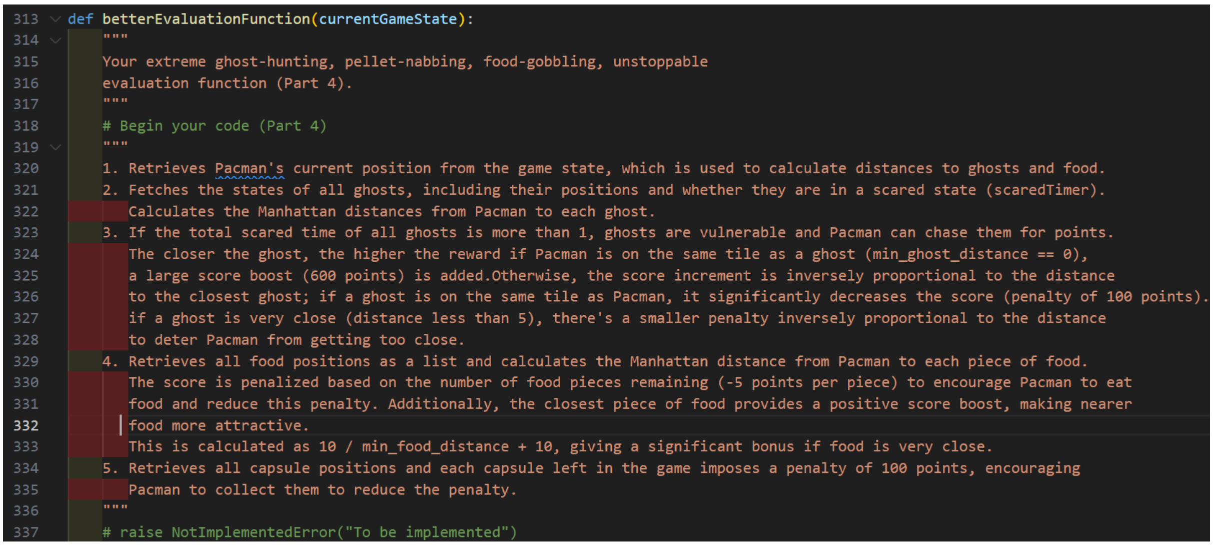


```
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (Part 3)
    """

    def getAction(self, gameState):
        """
        Returns the expectimax action using self.depth and self.evaluationFunction

        All ghosts should be modeled as choosing uniformly at random from their
        legal moves.
        """
        # Begin your code (Part 3)
        """
        1. If the recursion reaches the maximum allowed depth or the state is a win or lose situation,
           the function returns the evaluation of that state using self.evaluationFunction(state).
        2. If no actions are available (which can happen in terminal states), it returns the evaluation of the state directly.
        3. Determines which agent will act next using modulo arithmetic. This cycles through agents sequentially,
           resetting to Pacman after all ghosts have taken their turns.
        4. Adjusts the depth for the next recursive call, decreasing only when all agents (including all ghosts) have taken a turn.
        5. Pacman (agent == 0): Since Pacman aims to maximize his score, the function calculates the maximum value
           among all possible actions.It also stores which actions lead to this maximum value. If it's the root call
           (depth == self.depth), it randomly selects from the best actions (ties in the maximum score) to add
           unpredictability to Pacman's behavior.
        6. Ghosts: As non-deterministic agents, ghosts are modeled to choose actions uniformly at random.
          The function calculates the average of the expectimax values of all actions, representing the expected
          value of any action taken by a ghost given the current state.
        """
        # raise NotImplementedError("To be implemented")
```

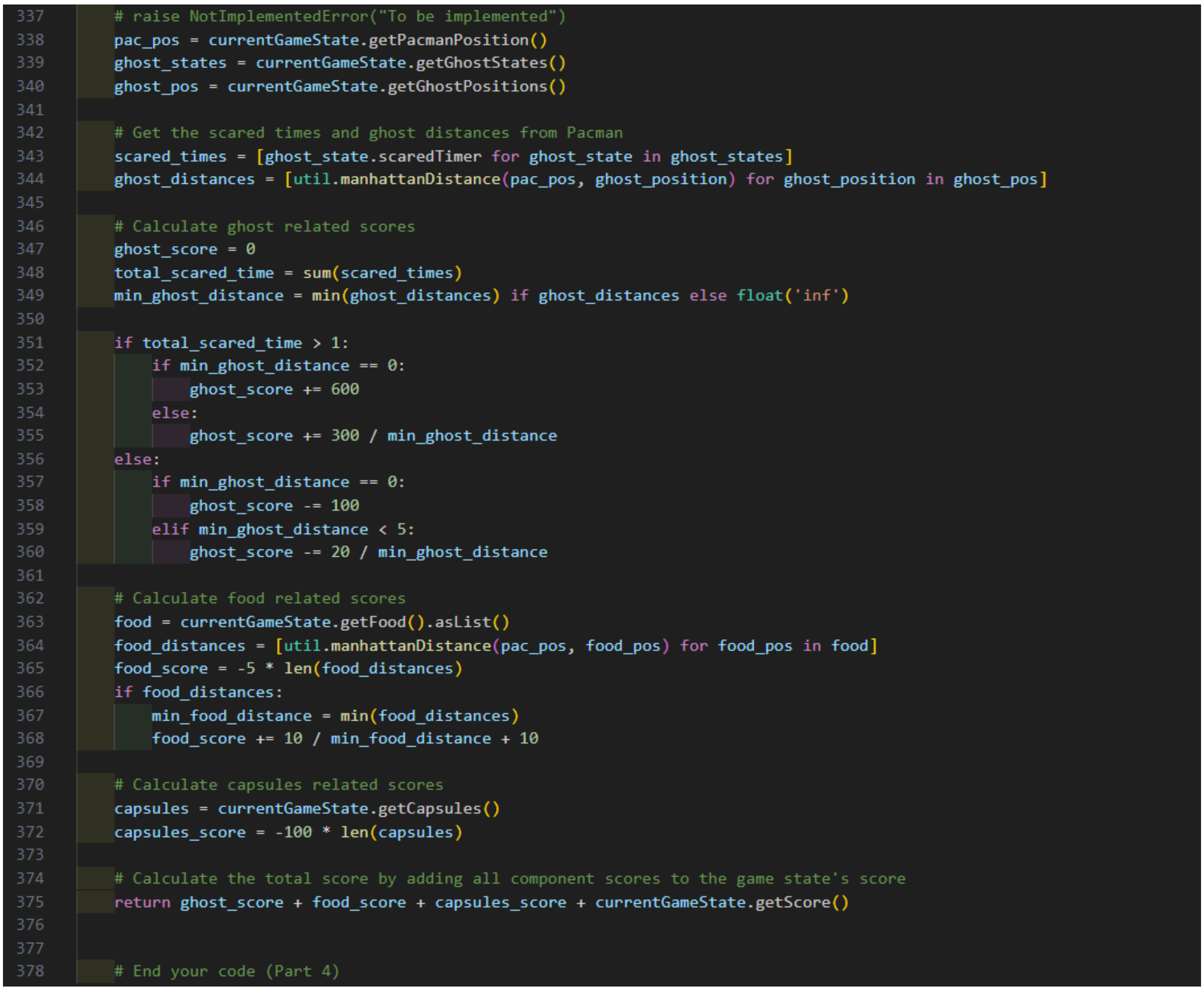


```
        def expectimax(state, depth, agent):
            # Base case: if depth is 0 or state is a terminal state
            if depth == 0 or state.isWin() or state.isLose():
                return self.evaluationFunction(state)

            next_agent = (agent + 1) % state.getNumAgents()
            next_depth = depth - 1 if next_agent == 0 else depth
            actions = state.getLegalActions(agent)

            if not actions:  # Check for no legal actions
                return self.evaluationFunction(state)

            # Generate all next states and values
            values = [expectimax(state.getNextState(agent, action), next_depth, next_agent) for action in actions]

            if agent == 0:  # Pacman's turn, find the maximum value
                max_value = max(values)
                best_actions = [actions[i] for i in range(len(actions)) if values[i] == max_value]
                return max_value if depth != self.depth else random.choice(best_actions)
            else:  # Ghosts' turn, calculate the average value
                avg_value = sum(values) / len(values)
                return avg_value

        # Execute expectimax from the current game state with full search depth and starting from Pacman
        return expectimax(gameState, self.depth, 0)
        # End your code (Part 3)
```

Part 4.



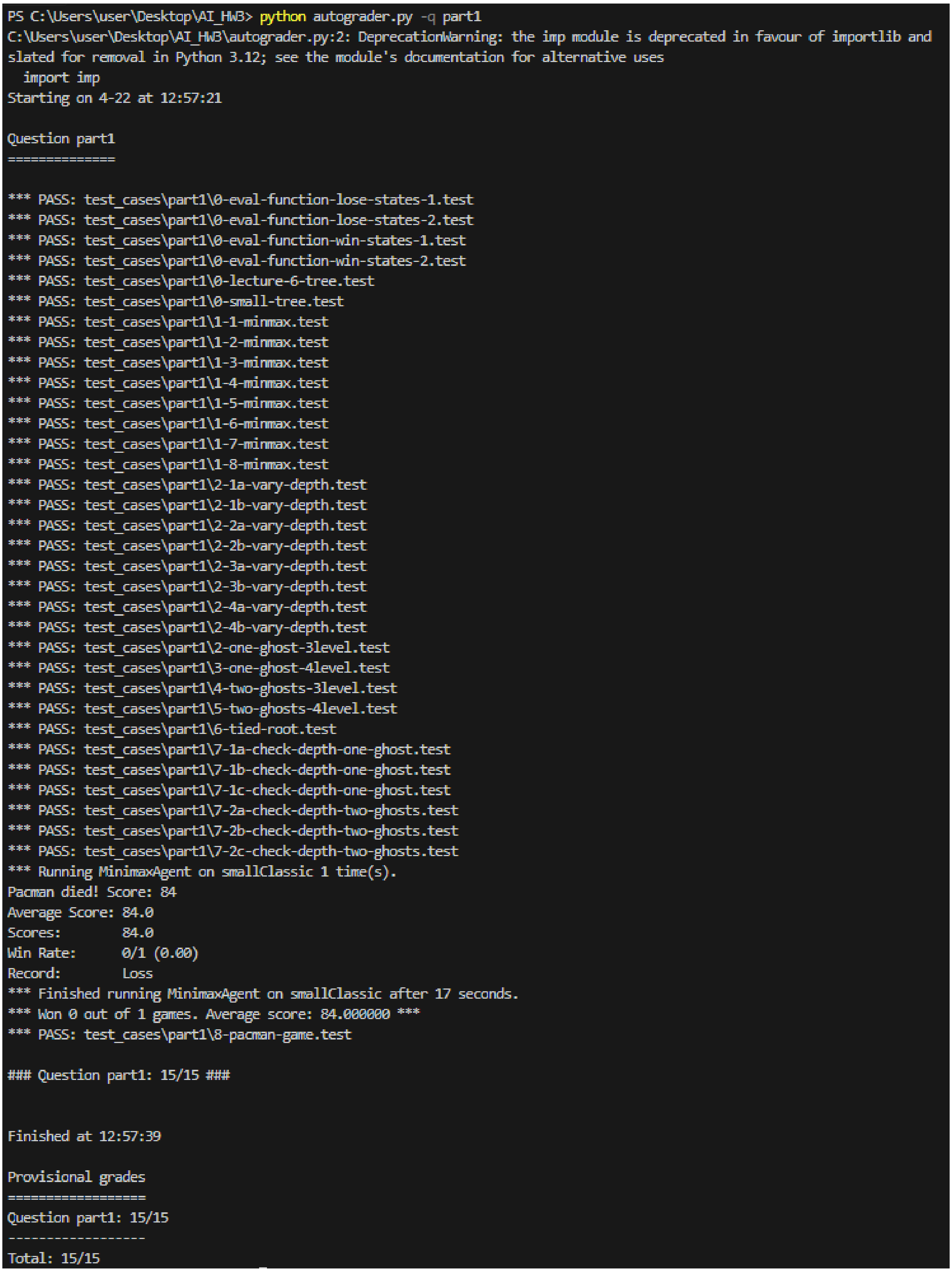
```python
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (Part 4).
    """
    # Begin your code (Part 4)
    """
    1. Retrieves Pacman's current position from the game state, which is used to calculate distances to ghosts and food.
    2. Fetches the states of all ghosts, including their positions and whether they are in a scared state (scaredTimer).
       Calculates the Manhattan distances from Pacman to each ghost.
    3. If the total scared time of all ghosts is more than 1, ghosts are vulnerable and Pacman can chase them for points.
       The closer the ghost, the higher the reward if Pacman is on the same tile as a ghost (min_ghost_distance == 0),
       a large score boost (600 points) is added.Otherwise, the score increment is inversely proportional to the distance
       to the closest ghost; if a ghost is on the same tile as Pacman, it significantly decreases the score (penalty of 100 points).
       if a ghost is very close (distance less than 5), there's a smaller penalty inversely proportional to the distance
       to deter Pacman from getting too close.
    4. Retrieves all food positions as a list and calculates the Manhattan distance from Pacman to each piece of food.
       The score is penalized based on the number of food pieces remaining (-5 points per piece) to encourage Pacman to eat
       food and reduce this penalty. Additionally, the closest piece of food provides a positive score boost, making nearer
       food more attractive.
       This is calculated as 10 / min_food_distance + 10, giving a significant bonus if food is very close.
    5. Retrieves all capsule positions and each capsule left in the game imposes a penalty of 100 points, encouraging
       Pacman to collect them to reduce the penalty.
    """
    # raise NotImplementedError("To be implemented")
```



```python
    # raise NotImplementedError("To be implemented")
    pac_pos = currentGameState.getPacmanPosition()
    ghost_states = currentGameState.getGhostStates()
    ghost_pos = currentGameState.getGhostPositions()

    # Get the scared times and ghost distances from Pacman
    scared_times = [ghost_state.scaredTimer for ghost_state in ghost_states]
    ghost_distances = [util.manhattanDistance(pac_pos, ghost_position) for ghost_position in ghost_pos]

    # Calculate ghost related scores
    ghost_score = 0
    total_scared_time = sum(scared_times)
    min_ghost_distance = min(ghost_distances) if ghost_distances else float('inf')

    if total_scared_time > 1:
        if min_ghost_distance == 0:
            ghost_score += 600
        else:
            ghost_score += 300 / min_ghost_distance
    else:
        if min_ghost_distance == 0:
            ghost_score -= 100
        elif min_ghost_distance < 5:
            ghost_score -= 20 / min_ghost_distance

    # Calculate food related scores
    food = currentGameState.getFood().asList()
    food_distances = [util.manhattanDistance(pac_pos, food_pos) for food_pos in food]
    food_score = -5 * len(food_distances)
    if food_distances:
        min_food_distance = min(food_distances)
        food_score += 10 / min_food_distance + 10

    # Calculate capsules related scores
    capsules = currentGameState.getCapsules()
    capsules_score = -100 * len(capsules)

    # Calculate the total score by adding all component scores to the game state's score
    return ghost_score + food_score + capsules_score + currentGameState.getScore()


    # End your code (Part 4)
```

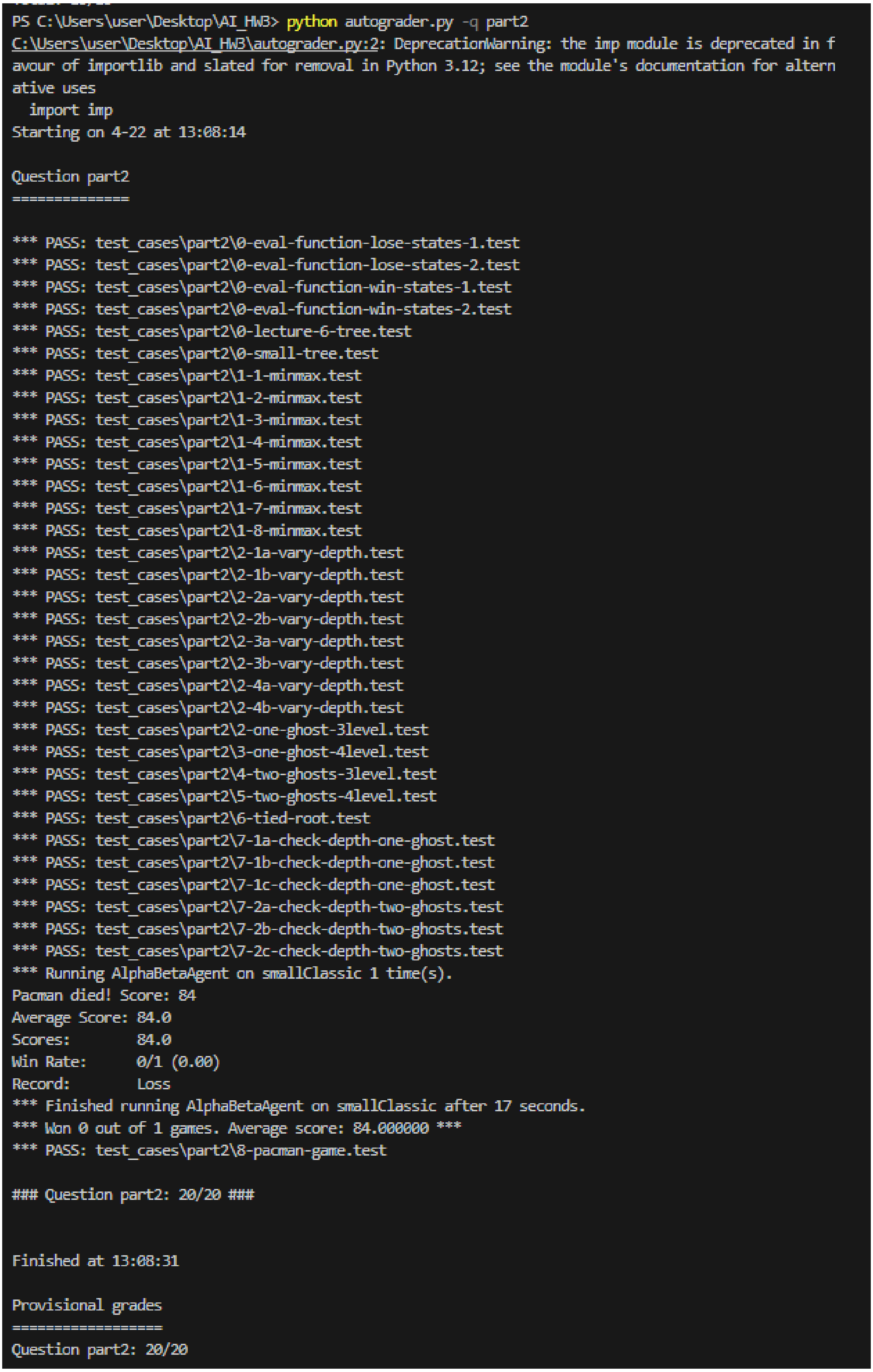
Part 2. Results & Analysis:

Part 1.



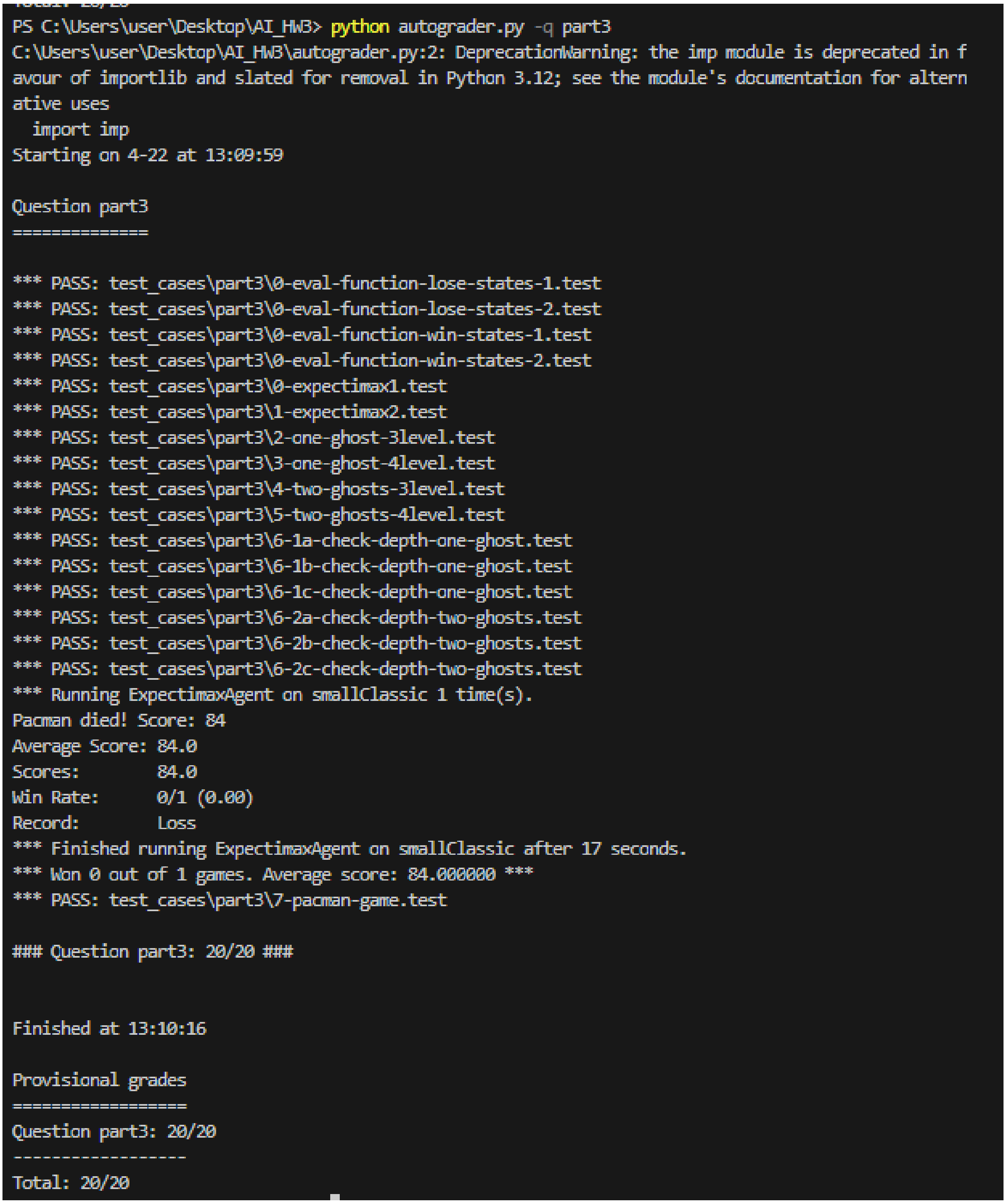
```
PS C:\Users\user\Desktop\AI_HW3> python autograder.py -q part1
C:\Users\user\Desktop\AI_HW3\autograder.py:2: DeprecationWarning: the imp module is deprecated in favour of importlib and
slated for removal in Python 3.12; see the module's documentation for alternative uses
  import imp
Starting on 4-22 at 12:57:21

Question part1
==============

*** PASS: test_cases\part1\0-eval-function-lose-states-1.test
*** PASS: test_cases\part1\0-eval-function-lose-states-2.test
*** PASS: test_cases\part1\0-eval-function-win-states-1.test
*** PASS: test_cases\part1\0-eval-function-win-states-2.test
*** PASS: test_cases\part1\0-lecture-6-tree.test
*** PASS: test_cases\part1\0-small-tree.test
*** PASS: test_cases\part1\1-1-minmax.test
*** PASS: test_cases\part1\1-2-minmax.test
*** PASS: test_cases\part1\1-3-minmax.test
*** PASS: test_cases\part1\1-4-minmax.test
*** PASS: test_cases\part1\1-5-minmax.test
*** PASS: test_cases\part1\1-6-minmax.test
*** PASS: test_cases\part1\1-7-minmax.test
*** PASS: test_cases\part1\1-8-minmax.test
*** PASS: test_cases\part1\2-1a-vary-depth.test
*** PASS: test_cases\part1\2-1b-vary-depth.test
*** PASS: test_cases\part1\2-2a-vary-depth.test
*** PASS: test_cases\part1\2-2b-vary-depth.test
*** PASS: test_cases\part1\2-3a-vary-depth.test
*** PASS: test_cases\part1\2-3b-vary-depth.test
*** PASS: test_cases\part1\2-4a-vary-depth.test
*** PASS: test_cases\part1\2-4b-vary-depth.test
*** PASS: test_cases\part1\2-one-ghost-3level.test
*** PASS: test_cases\part1\3-one-ghost-4level.test
*** PASS: test_cases\part1\4-two-ghosts-3level.test
*** PASS: test_cases\part1\5-two-ghosts-4level.test
*** PASS: test_cases\part1\6-tied-root.test
*** PASS: test_cases\part1\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part1\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part1\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 17 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part1\8-pacman-game.test

### Question part1: 15/15 ###


Finished at 12:57:39

Provisional grades
==================
Question part1: 15/15
------------------
Total: 15/15
```

Part 2.



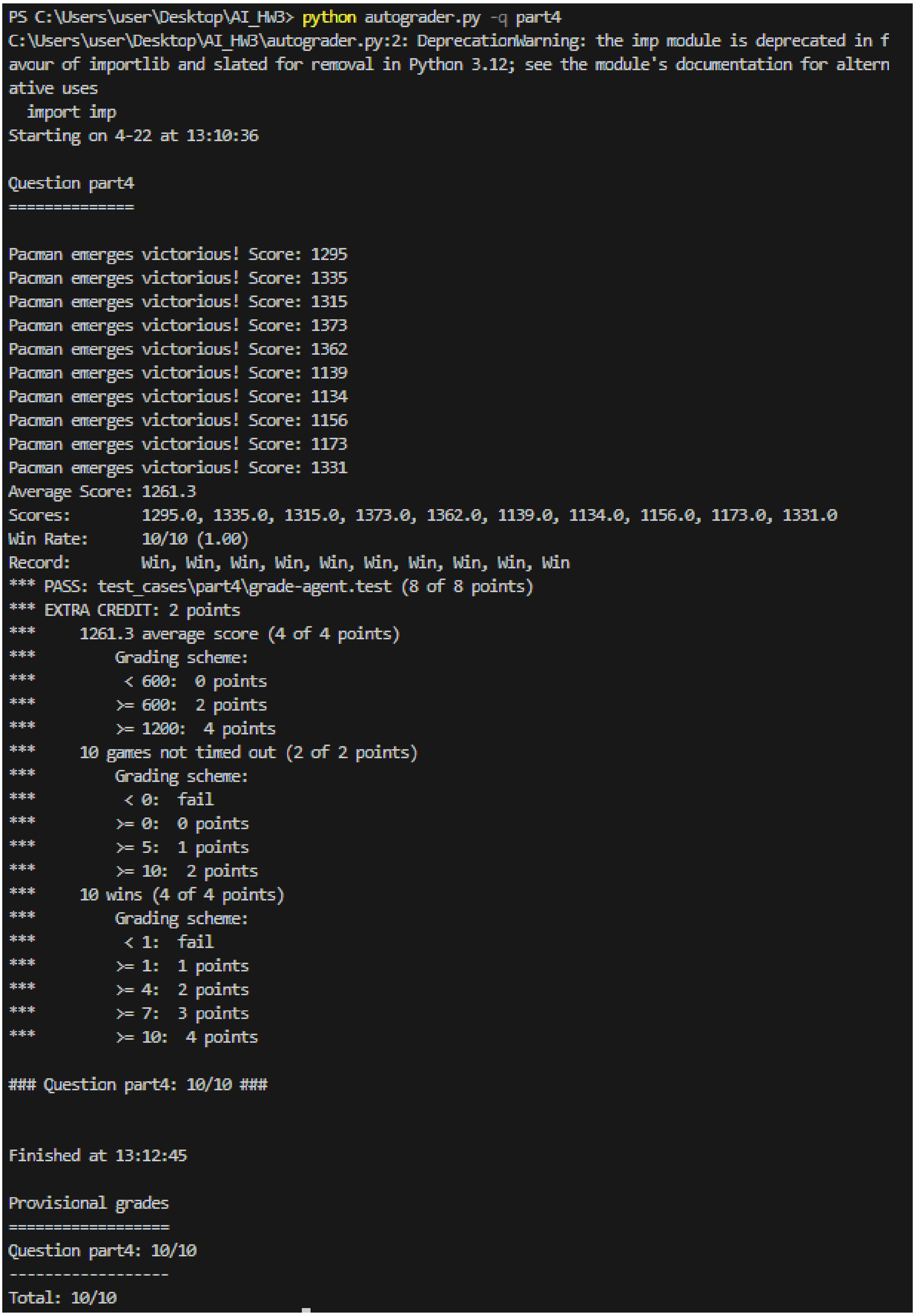
```
PS C:\Users\user\Desktop\AI_HW3> python autograder.py -q part2
C:\Users\user\Desktop\AI_HW3\autograder.py:2: DeprecationWarning: the imp module is deprecated in f
avour of importlib and slated for removal in Python 3.12; see the module's documentation for altern
ative uses
  import imp
Starting on 4-22 at 13:08:14

Question part2
==============

*** PASS: test_cases\part2\0-eval-function-lose-states-1.test
*** PASS: test_cases\part2\0-eval-function-lose-states-2.test
*** PASS: test_cases\part2\0-eval-function-win-states-1.test
*** PASS: test_cases\part2\0-eval-function-win-states-2.test
*** PASS: test_cases\part2\0-lecture-6-tree.test
*** PASS: test_cases\part2\0-small-tree.test
*** PASS: test_cases\part2\1-1-minmax.test
*** PASS: test_cases\part2\1-2-minmax.test
*** PASS: test_cases\part2\1-3-minmax.test
*** PASS: test_cases\part2\1-4-minmax.test
*** PASS: test_cases\part2\1-5-minmax.test
*** PASS: test_cases\part2\1-6-minmax.test
*** PASS: test_cases\part2\1-7-minmax.test
*** PASS: test_cases\part2\1-8-minmax.test
*** PASS: test_cases\part2\2-1a-vary-depth.test
*** PASS: test_cases\part2\2-1b-vary-depth.test
*** PASS: test_cases\part2\2-2a-vary-depth.test
*** PASS: test_cases\part2\2-2b-vary-depth.test
*** PASS: test_cases\part2\2-3a-vary-depth.test
*** PASS: test_cases\part2\2-3b-vary-depth.test
*** PASS: test_cases\part2\2-4a-vary-depth.test
*** PASS: test_cases\part2\2-4b-vary-depth.test
*** PASS: test_cases\part2\2-one-ghost-3level.test
*** PASS: test_cases\part2\3-one-ghost-4level.test
*** PASS: test_cases\part2\4-two-ghosts-3level.test
*** PASS: test_cases\part2\5-two-ghosts-4level.test
*** PASS: test_cases\part2\6-tied-root.test
*** PASS: test_cases\part2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part2\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 17 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part2\8-pacman-game.test

### Question part2: 20/20 ###


Finished at 13:08:31

Provisional grades
==================
Question part2: 20/20
```

Part 3.



```
PS C:\Users\user\Desktop\AI_HW3> python autograder.py -q part3
C:\Users\user\Desktop\AI_HW3\autograder.py:2: DeprecationWarning: the imp module is deprecated in f
avour of importlib and slated for removal in Python 3.12; see the module's documentation for altern
ative uses
  import imp
Starting on 4-22 at 13:09:59

Question part3
==============

*** PASS: test_cases\part3\0-eval-function-lose-states-1.test
*** PASS: test_cases\part3\0-eval-function-lose-states-2.test
*** PASS: test_cases\part3\0-eval-function-win-states-1.test
*** PASS: test_cases\part3\0-eval-function-win-states-2.test
*** PASS: test_cases\part3\0-expectimax1.test
*** PASS: test_cases\part3\1-expectimax2.test
*** PASS: test_cases\part3\2-one-ghost-3level.test
*** PASS: test_cases\part3\3-one-ghost-4level.test
*** PASS: test_cases\part3\4-two-ghosts-3level.test
*** PASS: test_cases\part3\5-two-ghosts-4level.test
*** PASS: test_cases\part3\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent 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 ExpectimaxAgent on smallClassic after 17 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part3\7-pacman-game.test

### Question part3: 20/20 ###


Finished at 13:10:16

Provisional grades
==================
Question part3: 20/20
------------------
Total: 20/20
```

Part 4.



```
PS C:\Users\user\Desktop\AI_HW3> python autograder.py -q part4
C:\Users\user\Desktop\AI_HW3\autograder.py:2: DeprecationWarning: the imp module is deprecated in f
avour of importlib and slated for removal in Python 3.12; see the module's documentation for altern
ative uses
  import imp
Starting on 4-22 at 13:10:36

Question part4
==============

Pacman emerges victorious! Score: 1295
Pacman emerges victorious! Score: 1335
Pacman emerges victorious! Score: 1315
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1362
Pacman emerges victorious! Score: 1139
Pacman emerges victorious! Score: 1134
Pacman emerges victorious! Score: 1156
Pacman emerges victorious! Score: 1173
Pacman emerges victorious! Score: 1331
Average Score: 1261.3
Scores:        1295.0, 1335.0, 1315.0, 1373.0, 1362.0, 1139.0, 1134.0, 1156.0, 1173.0, 1331.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\part4\grade-agent.test (8 of 8 points)
*** EXTRA CREDIT: 2 points
***     1261.3 average score (4 of 4 points)
***         Grading scheme:
***          < 600:  0 points
***         >= 600:  2 points
***         >= 1200:  4 points
***     10 games not timed out (2 of 2 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 5:  1 points
***         >= 10:  2 points
***     10 wins (4 of 4 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 4:  2 points
***         >= 7:  3 points
***         >= 10:  4 points

### Question part4: 10/10 ###


Finished at 13:12:45

Provisional grades
==================
Question part4: 10/10
------------------
Total: 10/10
```