交大資工 110550062
朱致伶

Homework 3: Multi-Agent Search

Part I. Implementation (5%):

- **Please screenshot your code snippets of Part 1 ~ Part 4, and explain your implementation.**
- **Part1 (minimaxAgent):**

```
# Begin your code (Part 1)
    """
    * for part(a),
    Take PACMAN as the agent for the most upper layer, and get possible actions.
    With currentState and all the possible actions for PACMAN, all the reachable nextStates are obtained
    Evalute the score of all nextStates, by calling ghost 1.
    Take the index of maxScore, return the action with maxScore.

    * for part(b),
    When reach terminal states(depth=0, win, lose), return the score of the state.
    Classify cases to 2 cases, PACMAN turn and GHOST turn, and assign value to parameters.
    Get possible actions of currentState.
    Obtain all the reachable nextStates, with agentIndex, currentState and all the possible actions.
    Evaluate the score of nextStates, by recursive the next state with minimaxAgent.
    Return the related bestScore, Max for pacman, min for all ghosts.
    Recursve down the depth until return bestScore.
    """
    # part(a)
    legalActions = gameState.getLegalActions(0)
    successors = (gameState.getNextState(0, action) for action in legalActions)
    scores = [self._minimax(successor, 1, self.depth) for successor in successors]
    i = scores.index((max(scores)))
    return legalActions[i]

# part(b)
def _minimax(self, state, agentIndex: int, depth: int):
    if depth == 0 or state.isWin() or state.isLose():
        return self.evaluationFunction(state)

    if agentIndex == 0:  # PACMAN TURN
        selectBestScore = max
        nextAgent = 1
        nextDepth = depth
    else:  # GHOST TURN
        selectBestScore = min
        nextAgent = (agentIndex + 1) % state.getNumAgents()
        nextDepth = (depth - 1) if nextAgent == 0 else depth

    legalActions = state.getLegalActions(agentIndex)
    successors = (state.getNextState(agentIndex, action) for action in legalActions)
    scores = [self._minimax(successor, nextAgent, nextDepth) for successor in successors]
    return selectBestScore(scores)
# End your code (Part 1)
```

交大資工 110550062
朱致伶

- **Part2 (AlphaBetaAgent):**

```
def getAction(self, gameState):
# Begin your code (Part 2)
    """
    * for part(a),
    Call the function of AlphaBetaAgent taking PACMAN as agent, and Return the best action.

    alphabeta is a recursive function implementint Alpha-Beta Pruning.
    If terminal state or max depth reached, return tuple(evaluatedScore, action)

    * for part(b), which is PACMAN turn, maximize the value
    First, Initialize the variable and Get possible actions of currentState,
    Then, go through all possible actions doing
    - Get the nextState with related action
    - Get the bestScore of nextState with recursion
    - Renew the maxScore
    - Do the Pruning if maxScore > beta
    - Renew alpha and maxAction if maxScore>alpha

    * for part(c), which is GHOST turn, minimize the value for all ghosts
    First, Initialize the variable and Get possible actions of currentState,
    Then, go through all possible actions doing
    - Get the nextState with related action
    - Get the bestScore of nextState with recursion
    - Renew the minScore
    - Do the Pruning if minScore < alpha
    - Renew beta and minAction if minScore>beta
    """
    # part(a)
    return self.alphabeta(gameState, 0, self.depth, -math.inf, math.inf)[1]

def alphabeta(self, gameState, agentIndex, depth, alpha, beta):
    if depth == 0 or gameState.isWin() or gameState.isLose():
        return tuple([self.evaluationFunction(gameState), None])

    # part(b): PACMAN turn, maximize the value
    if agentIndex == 0:
        maxScore = -math.inf
        maxAction = None
        legalActions = gameState.getLegalActions(agentIndex)

        for action in legalActions:
            successor = gameState.getNextState(agentIndex, action)
            score = self.alphabeta(successor, 1, depth, alpha, beta)[0]
            maxScore = max(maxScore, score)
            if maxScore > beta:
                return tuple([maxScore, action])
            maxAction = action if alpha<maxScore else maxAction
            alpha = max(alpha, maxScore)
        return tuple([maxScore, maxAction])

    # part(c): GHOST turn, minimize the value for all ghosts
    else:
        minScore = math.inf
        minAction = None
        legalActions = gameState.getLegalActions(agentIndex)
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        nextDepth = depth if nextAgent > 0 else depth - 1
        for action in legalActions:
            successor = gameState.getNextState(agentIndex, action)
            score = self.alphabeta(successor, nextAgent, nextDepth, alpha, beta)[0]
            minScore = min(minScore, score)
            if minScore < alpha:
                return tuple([minScore, action])
            minAction = action if beta>minScore else minAction
            beta = min(beta, minScore)
        return tuple([minScore, minAction])
# End your code (Part 2)
```

- **Part3 (ExpectimaxAgent):**

```
# Begin your code (Part 3)
    """
    * for part(a),
    Take PACMAN as the agent for the most upper layer, and get possible actions.
    Obtain all the reachable nextStates with currentState and all the possible actions for PACMAN.
    Evalute the score of all nextStates by calling ghost 1.
    Take the index of maxScore, return the action with maxScore.

    * for part(b),
    When reach terminal states(depth=0, win, lose), return the score of the state.
    Get possible actions of currentState.
    Obtain all the reachable nextStates, with agentIndex, currentState and all the possible actions.
    Assign related value to nextAgent and nextDepth.
    Evaluate the score of nextStates, by recursive the next state with ExpectimaxAgent.
    Return the related bestScore, Max for pacman, AverageScore for all ghosts.
    Recursve down the depth until return bestScore.
    """
    # part(a)
    legalActions = gameState.getLegalActions(0)
    successors = (gameState.getNextState(0, action) for action in legalActions)
    scores = [self._expected(successor, 1, self.depth) for successor in successors]
    i = scores.index((max(scores)))
    return legalActions[i]

# part(b)
def _expected(self, state, agentIndex: int, depth: int):
    if depth == 0 or state.isWin() or state.isLose():
        return self.evaluationFunction(state)

    legalActions = state.getLegalActions(agentIndex)
    successors = (state.getNextState(agentIndex, action) for action in legalActions)
    nextAgent = (agentIndex + 1) % state.getNumAgents()
    nextDepth = depth if nextAgent > 0 else depth - 1
    scores = [self._expected(successor, nextAgent, nextDepth) for successor in successors]
    if agentIndex == 0:  # pacman turn
        return max(scores)
    else:  # ghost turn
        return sum(scores) / len(legalActions)
# End your code (Part 3)
```

- **Part4 (betterEvaluationFunction):**

```
betterEvaluationFunction takes ghost-hunting and food-gobbling into account.
Get originalScore with original EvaluationFunction.
* for ghostPenalty:
When Pacman eats the big dot, Pacman has to be awared of the timer counted to 0,
since ghosts are able to eat pacman if the two encounterd.
Get distanceFromUnscared with manhattan distance between ghost and pacman, and scaredTimer.
Adapt the score by minusing ghostPenalty, which is the sum of (300 / square of distanceFromUnscared).

* for ghostBonus:
When Pacman eats the big dot, Pacman has the ability to eat ghosts.
Get distanceFromScared with manhattan distance between ghost and pacman, and scaredTimer.
Adapt the score by adding ghostBonus, which is the sum of (190 / square of distanceFromScared).

* for foodBonus:
Since the goal for Pacman is to eat all dots, dots can be regard as bonus.
Get manhattanNearestFood with manhattan distance between dots and pacman position, and all the remained dots.
Adapt the score by adding foodBonus, which is the sun of (9 / distance in manhattanNearestFood).

Therefore, score = originalScore - ghostPenalty + ghostBonus + foodBonus.
"""
# Begin your code (Part 4)
state = currentGameState
currentScore = state.getScore()
if state.isWin() or state.isLose():
    return currentScore
position = state.getPacmanPosition()
ghosts = state.getGhostStates()

ghostDistances = [manhattanDistance(position, ghost.configuration.pos) for ghost in ghosts]
scaredTimers = [ghost.scaredTimer for ghost in ghosts]
distFromUnscared = [dist for dist, timer in zip(ghostDistances, scaredTimers) if timer == 0]
distFromScared = [dist for dist, timer in zip(ghostDistances, scaredTimers) if timer > 2]
ghostPenalty = sum((300 / dist ** 2 for dist in distFromUnscared), 0)
ghostBonus = sum((190 / dist for dist in distFromScared), 0)

foods = state.getFood().asList()
manhattanDistances = [(manhattanDistance(position, food), food) for food in foods]
manhattanNearestFood = [food for dist, food in sorted(manhattanDistances)[:5]]
foodBonus = sum(9 / d for d, f in manhattanNearestFood)

score = currentScore - ghostPenalty + ghostBonus + foodBonus # + capsuleBonus
return score
# End your code (Part 4)
```

交大資工 110550062
朱致伶

Part II. Results & Analysis (5%):

- **Please screenshot the results. For instance,** the result of the **autograder** and any observation of your **evaluation function**.

Part 1 (MinimaxAgent):

```
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 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part1\8-pacman-game.test

### Question part1: 20/20 ###
```

交大資工 110550062
朱致伶

Part 2 (AlphaBetaAgent):

```
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 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part2\8-pacman-game.test

### Question part2: 25/25 ###
```

Part 3 (ExpectimaxAgent):

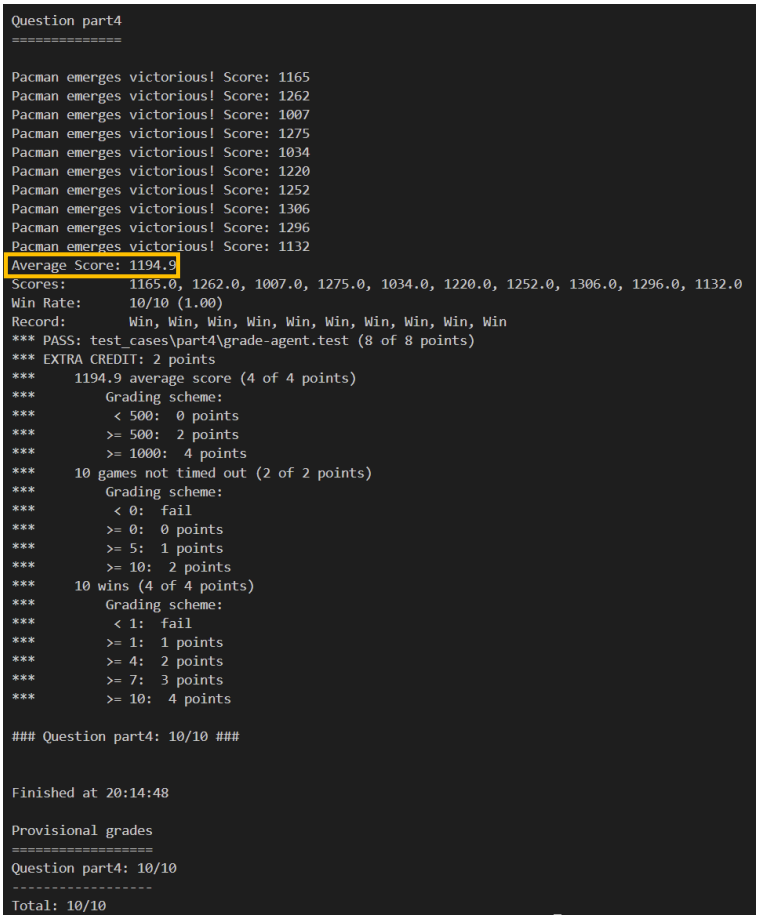
```
Question part3
==============

*** 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 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part3\7-pacman-game.test

### Question part3: 25/25 ###
```

交大資工 110550062
朱致伶

Part 4 (betterEvaluationFunction):



betterEvaluationFunction takes ghost-hunting and food-gobbling into account, compared with evaluationFunction.
Therefore, **score = originalScore - ghostPenalty + ghostBonus + foodBonus.**
Detailed description is in the above Part4 screenshot.

I tested the weight of ghostPenalty, ghostBonus and foodBonus, in the range of 250 to 350, 150 to 250, and 10 to 30, respectively.
When determining the weight of ghostPenalty, ghostBonus and foodBonus, they have weight in the value of 300, 200 and 10 respectively.
Since if the Pacman is eaten by any of the ghosts, the game is over. Therefore, ghostPenalty has the heaviest weight among the three.
When the ghost is eaten, it restarts from the rectangle in the middle, which is better for Pacman to eat the remaining dots. Therefore, GhostBonus has high weight related to foodBonus.
For foodBonus, since there are many dots in the space, each of them cause mere change to the current state. It has the lowest weight, which is only 10.

交大資工 110550062
朱致伶

Finish: