Homework 3: Multi-Agent Search

109550135 范恩宇

Part I. Implementation :

↓ ↓ Minimax Search ↓ ↓

```python
        # Begin your code (Part 1)
        """
            First , define a "minimax" function . In this function , it returns the utility when the game ends
        or defined depth is reached ,and  does maximization for "Pacman" and minimization for "ghosts"
        depending on the number of agent .
            In pacman maximization part , it finds the max value of the results done by minimax()
        with agent(=1) , current depth , and child game state . Similiar in ghost minimization part , it
        finds the min value with the same function and parameteres except for a changing "agent" , also
        increase depth when agent+1 = 0 .
            Then , perform maximum action for root(pacman) by traversing through pacman's legal move and using
        minimax() during the process .

        """
        def minimax(agent, depth, gameState):
            if gameState.isWin() or gameState.isLose() or depth == self.depth:
                return self.evaluationFunction(gameState)
            if agent == 0:
                return max(minimax(1, depth, gameState.getNextState(agent, act)) for act in gameState.getLegalActions(agent))
            else:
                next_agent = agent + 1
                if gameState.getNumAgents() == next_agent:
                    next_agent = 0
                if next_agent == 0:
                    depth += 1
                return min(minimax(next_agent, depth, gameState.getNextState(agent, act)) for act in gameState.getLegalActions(agent))

        mx = float("-inf")
        move = Directions.WEST
        for agent_st in gameState.getLegalActions(0):
            utility = minimax(1, 0, gameState.getNextState(0, agent_st))
            if utility > mx or mx == float("-inf"):
                mx = utility
                move = agent_st

        return move
        # End your code (Part 1)
```

↓ ↓ Alpha-Beta Pruning ↓ ↓

```python
        # Begin your code (Part 2)
        """
            Similar to Minimax , but set two values , alpha(a) and beta(b) as the standard to determine
        if we should do pruning or not .
            First , define a maximize and a minimize function that calculates value with Max(Min) 's best
        opertion on path to root (alpha(a) and beta(b) ).
            Then , define a "alphabetaprune()" which returns the utility if the game ends or defined depth
        is reached , and maximize(minimize) pacman(ghost) just like minimax .
            Finally , perform maximum action for root(pacman) by traversing through pacman's legal move and using
        alphabetaprune() during the process.

        """
        def maximize(agent, depth, game_state, a, b):
            v = float("-inf")
            for act in game_state.getLegalActions(agent):
                v = max(v, alphabetaprune(1, depth, game_state.getNextState(agent, act), a, b))
                if v > b:
                    return v
                a = max(a, v)
            return v

        def minimize(agent, depth, game_state, a, b):
            v = float("inf")
            next_agent = agent + 1
            if game_state.getNumAgents() == next_agent:
                next_agent = 0
            if next_agent == 0:
                depth += 1

            for act in game_state.getLegalActions(agent):
                v = min(v, alphabetaprune(next_agent, depth, game_state.getNextState(agent, act), a, b))
                if v < a:
                    return v
                b = min(b, v)
            return v

        def alphabetaprune(agent, depth, game_state, a, b):
            if game_state.isWin() or game_state.isLose() or depth == self.depth:
                return self.evaluationFunction(game_state)

            if agent == 0:
                return maximize(agent, depth, game_state, a, b)
            else:
                return minimize(agent, depth, game_state, a, b)

        alpha = float("-inf")
        beta = float("inf")
        utility = float("-inf")
        move = Directions.WEST
        for agent_st in gameState.getLegalActions(0):
            ghost_val = alphabetaprune(1, 0, gameState.getNextState(0, agent_st), alpha, beta)
            if ghost_val > utility:
                utility = ghost_val
                move = agent_st
            if utility > beta:
                return utility
            alpha = max(alpha, utility)

        return move
        # End your code (Part 2)
```

↓ ↓ Expectimax Search ↓ ↓

```python
        # Begin your code (Part 3)
        """
            Also similar to Minimax , define a expectimax function that returns the utility if the game ends
        or defined depth is reached , and does maximization for pacman when "agent" be 0, but chooses the
        branch by max expected utility for ghosts(chance) when "agent" not be 0 .
            Finally , perform maximum action for root(pacman) by traversing through pacman's legal move and using
        expectimax() during the process.

        """
        def expectimax(agent, depth, gameState):
            if gameState.isLose() or gameState.isWin() or depth == self.depth:
                return self.evaluationFunction(gameState)
            if agent == 0:
                return max(expectimax(1, depth, gameState.getNextState(agent, act)) for act in gameState.getLegalActions(agent))
            else:
                next_agent = agent + 1
                if gameState.getNumAgents() == next_agent:
                    next_agent = 0
                if next_agent == 0:
                    depth += 1
                return sum(expectimax(next_agent, depth, gameState.getNextState(agent, act)) for act in gameState.getLegalActions(agent)) / float(len(gameState.getLegalActions(agent)))

        mx = float("-inf")
        move = Directions.WEST
        for agent_st in gameState.getLegalActions(0):
            utility = expectimax(1, 0, gameState.getNextState(0, agent_st))
            if utility > mx or mx == float("-inf"):
                mx = utility
                move = agent_st

        return move
        # End your code (Part 3)
```

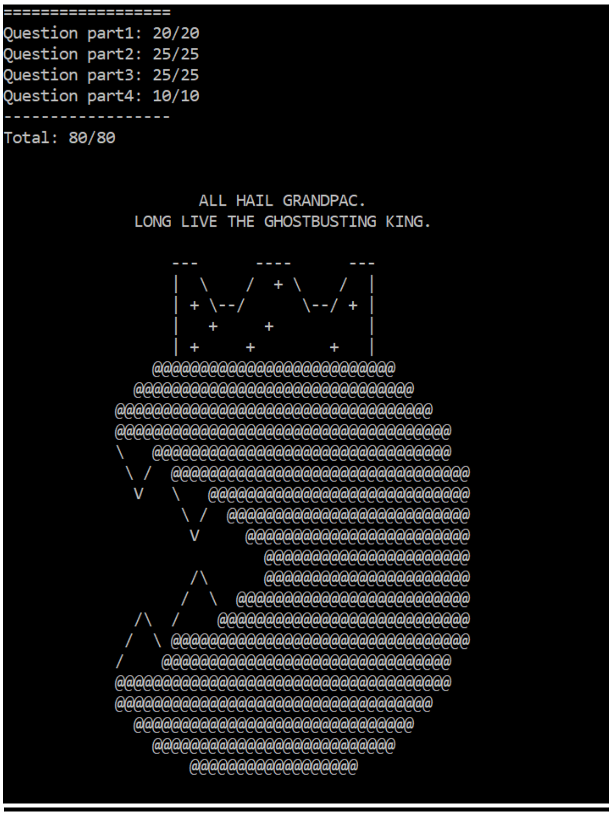
↓ ↓ Evaluation Function ↓ ↓

```python
    # Begin your code (Part 4)
    """
        First , calcultate the distance between pacman & ghosts  , and
    get "g_dist" . In addition , check the proximity of ghosts ( within distance = 1)
    around pacman , and get "g_proximity".
        Second , calculate the distance to the closest food and get "min_fdst"
    as the result.
        Finally , get the number of capsules available (cap_num) . The combination of
    all former calculated results and score we get will be the result .

    """
    pacman_pos = currentGameState.getPacmanPosition()
    g_dist = 1
    g_proximity = 0
    for ghost_state in currentGameState.getGhostPositions():
        dist = util.manhattanDistance(pacman_pos, ghost_state)
        g_dist += dist
        if dist <= 1:
            g_proximity += 1

    food = currentGameState.getFood()
    food_List = food.asList()
    min_fdist = -1
    for fd in food_List:
        dist = util.manhattanDistance(pacman_pos, fd)
        if min_fdist == -1 or min_fdist >= dist:
            min_fdist = dist

    cap = currentGameState.getCapsules()
    cap_num = len(cap)

    return currentGameState.getScore() - (1 / float(g_dist)) - g_proximity + (1 / float(min_fdist))  - cap_num
    # End your code (Part 4)
```

Part II. Results & Analysis :

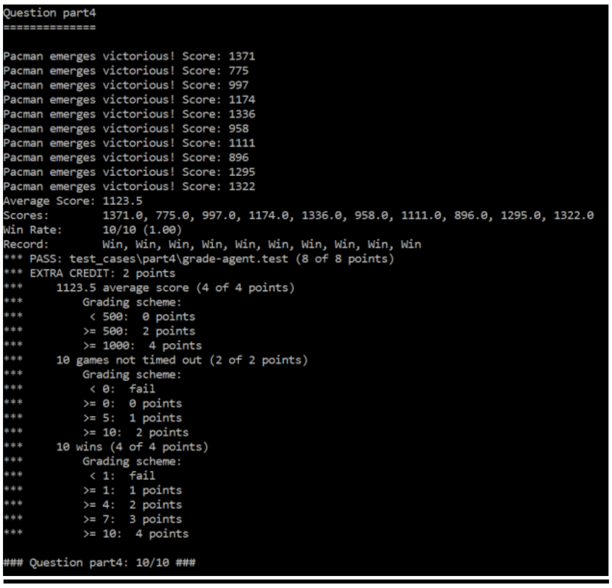
Results:

↓ ↓ Full Results ↓ ↓



```
==================
Question part1: 20/20
Question part2: 25/25
Question part3: 25/25
Question part4: 10/10
------------------
Total: 80/80

ALL HAIL GRANDPAC.
LONG LIVE THE GHOSTBUSTING KING.
```

↓ ↓ Evaluation Function ↓ ↓



```
Question part4
==============

Pacman emerges victorious! Score: 1371
Pacman emerges victorious! Score: 775
Pacman emerges victorious! Score: 997
Pacman emerges victorious! Score: 1174
Pacman emerges victorious! Score: 1336
Pacman emerges victorious! Score: 958
Pacman emerges victorious! Score: 1111
Pacman emerges victorious! Score: 896
Pacman emerges victorious! Score: 1295
Pacman emerges victorious! Score: 1322
Average Score: 1123.5
Scores:        1371.0, 775.0, 997.0, 1174.0, 1336.0, 958.0, 1111.0, 896.0, 1295.0, 1322.0
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
***     1123.5 average score (4 of 4 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  2 points
***         >= 1000:  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 ###
```

Analysis:

When testing my evaluation function , the pacman performs better when it also considers the distance between itself and ghosts(food) , proximity of ghosts around it , and number of capsules .

Among these four arguments , distance between pacman and food seems to influence score result the most . I think it’s quite reasonable , since the goal of the game is to earn more points , considering about things relating to earning points more will be more important . Of course , other arguments also affect the result , since those three factors should be concerned when we play the game by ourselves , too .