Spring 2022

Introduction to Artificial Intelligence

Homework 3: Multi-Agent Search 109550159 李驊恩

Part I. Implementation

The explanation is in the notation of the code.

Part1. Minimax Search

```python
        # Begin your code (Part 1)

        def MinimaxFunction(gameState,agt,depth):
            ans = []

            # If the game is ended (win or lose)
            if gameState.isWin():
                return self.evaluationFunction(gameState),0
            if gameState.isLose():
                return self.evaluationFunction(gameState),0

            # If it arrive max depth
            if depth == self.depth:
                return self.evaluationFunction(gameState),0
            if agt == gameState.getNumAgents() - 1:
                depth += 1
            if agt == gameState.getNumAgents() - 1:
                nxt_agt = self.index
            else:
                nxt_agt = agt + 1

            # The function that fix the answer with minimax value
            def FixingFunction():
                nxt_val = MinimaxFunction(gameState.getNextState(agt,action),nxt_agt,depth)
                ans.append(nxt_val[0])
                ans.append(action)

            # The function that check the minimax value
            def CheckingFunction():
                prvs_val = ans[0]
                nxt_val = MinimaxFunction(gameState.getNextState(agt,action),nxt_agt,depth)
                if agt == self.index and nxt_val[0] > prvs_val:
                    ans[0] = nxt_val[0]
                    ans[1] = action
                elif agt != self.index and nxt_val[0] <= prvs_val:
                    ans[0] = nxt_val[0]
                    ans[1] = action

            # Find every minimax value
            for action in gameState.getLegalActions(agt):
                if len(ans) == 0:
                    FixingFunction()
                else:
                    CheckingFunction()

            return ans

        return MinimaxFunction(gameState,self.index,0)[1]
        # End your code (Part 1)
```

Part 2: Alpha-Beta Pruning

```python
        # Begin your code (Part 2)
        def AlphaBetaPruning(gameState,agt,depth,alpha,beta):
            ans = []

            # If the game is ended (win or lose)
            if gameState.isWin():
                return self.evaluationFunction(gameState),0
            if gameState.isLose():
                return self.evaluationFunction(gameState),0

            # If it arrive max depth
            if depth == self.depth:
                return self.evaluationFunction(gameState),0
            if agt == gameState.getNumAgents() - 1:
                depth += 1
            if agt == gameState.getNumAgents() - 1:
                nxt_agt = self.index
            else:
                nxt_agt = agt + 1


            for action in gameState.getLegalActions(agt):
                if  len(ans) == 0 :
                    nxt_val = AlphaBetaPruning(gameState.getNextState(agt,action),nxt_agt,depth,alpha,beta)


                    ans.append(nxt_val[0])
                    ans.append(action)

                # Modified the initial value of alpha and beta
                    if agt == self.index:
                        if ans[0] > alpha:
                            alpha = ans[0]
                    else:
                        if ans[0] < beta:
                            beta = ans[0]

                else:
                # If the alpha-beta pruning is correct
                    if ans[0] > beta:
                      if agt == self.index:
                        return ans

                    if ans[0] < alpha :
                     if agt != self.index:
                        return ans

                    prvs_val = ans[0]
                    nxt_val = AlphaBetaPruning(gameState.getNextState(agt,action),nxt_agt,depth,alpha,beta)

                # Max agent
                    if agt == self.index:
                        if nxt_val[0] > prvs_val:
                            ans[0] = nxt_val[0]
                            ans[1] = action
                            if ans[0] > alpha:
                                alpha = ans[0]
                # Min agent
                    else:
                        if nxt_val[0] < prvs_val:
                            ans[0] = nxt_val[0]
                            ans[1] = action
                            if ans[0] < beta:
                                beta = ans[0]

            return ans

        return AlphaBetaPruning(gameState,self.index,0,-float("inf"),float("inf"))[1]
        # End your code (Part 2)
```

Part 3: Expectimax Search

```python
        # Begin your code (Part 3)
        def ExpectivemaxFunction(gameState,agt,depth):
            ans = []

            # If the game is ended (win or lose)
            if gameState.isWin():
                return self.evaluationFunction(gameState),0
            if gameState.isLose():
                return self.evaluationFunction(gameState),0

            # If it arrive max depth
            if depth == self.depth:
                return self.evaluationFunction(gameState),0
            if agt == gameState.getNumAgents() - 1:
                depth += 1
            if agt == gameState.getNumAgents() - 1:
                nxt_agt = self.index
            else:
                nxt_agt = agt + 1

            # The function that fix chance node
            # p = 1 / total legal actions
            def FixingFunction():

                    nxt_val = ExpectivemaxFunction(gameState.getNextState(agt,action),nxt_agt,depth)

                    if(agt != self.index):
                        ans.append((1.0 / len(gameState.getLegalActions(agt))) * nxt_val[0])
                        ans.append(action)
                    else:
                        ans.append(nxt_val[0])
                        ans.append(action)

            # The function that check the minimax value
            def CheckingFunction():
                prvs_val = ans[0]
                nxt_val = ExpectivemaxFunction(gameState.getNextState(agt,action),nxt_agt,depth)

                if agt == self.index:
                    if nxt_val[0] > prvs_val:
                        ans[0] = nxt_val[0]
                        ans[1] = action
                else:
                    ans[0] = ans[0] + (1.0 / len(gameState.getLegalActions(agt))) * nxt_val[0]
                    ans[1] = action

            for action in gameState.getLegalActions(agt):

                if len(ans) == 0 :
                  FixingFunction()

                else:
                  CheckingFunction()

            return ans

        return ExpectivemaxFunction(gameState,self.index,0)[1]

    # End your code (Part 3)
```

Part 4: Evaluation Function

```
    # Begin your code (Part 4)

    # Get the list of the food
    food_list = currentGameState.getFood().asList()

    # Get the total number of the food
    Num_of_food = len(food_list)

    # Get the total number of capsules
    Num_of_cap = len(currentGameState.getCapsules())

    # Get the position of pacman
    pacman_pos = currentGameState.getPacmanPosition()

    # Get the position of ghosts
    ghost_pos = currentGameState.getGhostPositions()

    # Get the state of the ghosts
    ghost_state= currentGameState.getGhostStates()

    # Initialize the distance of the nearest food
    nearest_food = 1

    # The lists of normal ghosts and those can be eaten.
    normal_ghost = []
    weak_ghost = []

    # The lists of distances to normal ghosts and those can be eaten.
    normal_ghost_distance = []
    weak_ghost_distance = []

    # Get the score
    score = currentGameState.getScore()

    # Initialize the evaluation value
    Evaluation_Value = 0

    # Add those scared ghost into the weak ghost list and
    # the other to the normal ghost list
    for ghost in ghost_state:
       if ghost.scaredTimer:
           weak_ghost.append(ghost)
       else:
           normal_ghost.append(ghost)

   # The list of distances of pacman to the food
    food_distances = [manhattanDistance(pacman_pos, food_position) for food_position in food_list]

   # There always be a nearest food while there is food have not been eaten
    if Num_of_food > 0:
      nearest_food = min(food_distances)

   # The list of the distances of pacman and ghosts
    for ghost_position in ghost_pos:
       ghost_distance = manhattanDistance(pacman_pos, ghost_position)

   # If the ghost is too close to the nearest food,
   # set the distance of the nearest food a bigger value

       if ghost_distance <= 1:
           nearest_food = 10000 # Set the value very big if the ghost is super close
                                # because I don't want it to lose
       elif ghost_distance <= 2:
           nearest_food = 3

   # Get manhattan distance
    for ghost in normal_ghost:
       weak_ghost_distance.append(manhattanDistance(pacman_pos,ghost.getPosition()))

    for ghost in weak_ghost:
       weak_ghost_distance.append(manhattanDistance(pacman_pos,ghost.getPosition()))

   # Give value to ghosts that can be eaten
    for i in weak_ghost_distance:
        if i < 3:
            Evaluation_Value += -30 * i
        else :
            Evaluation_Value += -10 * i

   # Give value to ghosts that are normal
    for i in normal_ghost_distance:
        if i <= 1:
            Evaluation_Value += 20 * i
        elif i <= 3:
            Evaluation_Value += 10 * i
        elif i <= 5:
            Evaluation_Value += 5 * i
        else:
            Evaluation_Value += 1 * i

   # Add the value of each weights * features
    Evaluation_Value += 20 / nearest_food
    Evaluation_Value += 100 * score
    Evaluation_Value += (-150) * Num_of_food
    Evaluation_Value += (-30) * Num_of_cap

    return Evaluation_Value
    # End your code (Part 4)

better = betterEvaluationFunction
```

Part II. Results & Analysis

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

Pacman emerges victorious! Score: 1123
Pacman emerges victorious! Score: 1099
Pacman emerges victorious! Score: 1162
Pacman emerges victorious! Score: 1140
Pacman emerges victorious! Score: 1316
Pacman emerges victorious! Score: 1309
Pacman emerges victorious! Score: 1139
Pacman emerges victorious! Score: 1161
Pacman emerges victorious! Score: 1122
Pacman emerges victorious! Score: 1338
Average Score: 1190.9
Scores:        1123.0, 1099.0, 1162.0, 1140.0, 1316.0, 1309.0, 1139.0, 1161.0, 1122.0, 1338.0
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
***     1190.9 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 ###

Finished at 21:45:29

Provisional grades
==================
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.

                   ---      ----      ---
                   |  \    /  + \    /  |
                   | + \--/      \--/ + |
                   |   +     +          |
                   | +     +        +   |
                 @@@@@@@@@@@@@@@@@@@@@@@@@@
               @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
             @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
             @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
             \   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
              \ /  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
               V   \   @@@@@@@@@@@@@@@@@@@@@@@@@@@@
                    \ /  @@@@@@@@@@@@@@@@@@@@@@@@@@
                     V     @@@@@@@@@@@@@@@@@@@@@@@@
                             @@@@@@@@@@@@@@@@@@@@@@
                     /\      @@@@@@@@@@@@@@@@@@@@@@
                    /  \  @@@@@@@@@@@@@@@@@@@@@@@@@
               /\  /    @@@@@@@@@@@@@@@@@@@@@@@@@@@
              /  \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
             /    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
             @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
             @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
               @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                 @@@@@@@@@@@@@@@@@@@@@@@@@@
                     @@@@@@@@@@@@@@@@@@
```