110550126 曾家祐

Homework 3: Multi-Agent Search

Part I. Implementation (5%):

- **Part1: minimax search**

1. **Construct and call minimax function**
2. **In the function, first check if we reach the end or not.**
   Than calculate the next state, including state ,depth, agent etc.
   According to the agent we look at, determine choose the max or min value
3. **Return the value and action**

```python
class MinimaxAgent(MultiAgentSearchAgent):
    """
    Your minimax agent (Part 1)
    """
    def getAction(self, gameState):
        # Begin your code (Part 1)
        """
        we should return the action that the pacman will take.

        the architecture of my implement is use recurrsive dfs method to build an search tree.

        we construct a function of minimax with parameters: state, depth, agent.
        and return the value and which action should take (construct the tree need value, answer need action)

        first we detect the tree is finish or not depand on depth or the game is end,
        and return action(stop,value)

        second, we compute the next parameters that need to pass into next iterate function,

        based on the agent (pacman or ghost)
        we chooose the max value or min value in all possible state
        and we return the value with its correesponding action.
        """
```

```python

        def minimax(state,depth,agent):
            if depth==0 or state.isLose() or state.isWin():
                return 0,self.evaluationFunction(state)

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

            legal_act = state.getLegalActions(agent)
            next_state = [state.getNextState(agent, act) for act in legal_act]
            next_value = [minimax(nextstate,next_depth,next_agent)[1] for nextstate in next_state]

            if agent ==0: #pacman
                ans_value = max(next_value)
            else: #ghost
                ans_value = min(next_value)
            possible_index = []
            for i in range(len(next_value)):
                if next_value[i]==ans_value:
                    possible_index.append(i)
            ans_index = random.choice(possible_index)

            return legal_act[ans_index] , ans_value
        return minimax(gameState,self.depth,0)[0]
        # End your code (Part 1)
```

- **Part2: Alpha-Beta Pruning**

1. **Construct and call alphabata function**
2. **In the function, first check if we reach the end or not.**
   Than calculate the next state, including state ,depth, agent etc.
   According to the agent we look at, determine choose the max or min value. Moreover by the upper bound and lower bound (alpha, beta), we can finish the function earlier to speed up the execution time.
3. **Return the value and action**

```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning (Part 2)
    """

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

        # Begin your code (Part 2)
        """
        we should return the action that the pacman will take.

        the architecture of my implement is use recurrsive dfs method to build an search tree.

        we construct a function of alphabeta with parameters: state, depth, agent, alpha beta.
        and return the value and which action should take (construct the tree need value, answer need action)

        first we detect the tree is finish or not depand on depth or the game is end,
        and return action(stop,value)

        second, we compute the next parameters that need to pass into next iterate function,

        based on the agent (pacman or ghost)

        if the agent is pacman(want to make value bigger)
        we initial the answer value be -inf
        for every next state, we can update alpha, beta and answer
        if next state value > answer, make answer =next state value
        if answer > upper bound(beta) return value and action
        update the lower bound(alpha)
        if the iteration finish return the answer value and action

        if the agent is ghost(want to make value bigger)
        we initial the answer value be inf
        for every next state, we can update alpha, beta and answer
        if next state value < answer, make answer =next state value
        if answer < lower bound(alpha) return value and action
        update the upper bound(beta)
        if the iteration finish return the answer value and action
        """
```

```python
        def alphabeta(state,depth,agent,alpha,beta):
            if depth==0 or state.isLose() or state.isWin():
                return 0,self.evaluationFunction(state)

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

            legal_act = state.getLegalActions(agent)
            if agent ==0:
                ans_value = float('-Inf')
                for action  in legal_act:
                    next_state = state.getNextState(agent, action)
                    next_value = alphabeta(next_state,next_depth,next_agent,alpha,beta)[1]
                    if ans_value < next_value:
                        ans_value = next_value
                        ans_action = [action]
                    elif ans_value ==next_value:
                        ans_action.append(action)
                    if ans_value > beta:
                        return  action,ans_value
                    alpha = max(alpha,ans_value)
                return random.choice(ans_action), ans_value
            else:
                ans_value = float('Inf')
                for action  in legal_act:
                    next_state = state.getNextState(agent, action)
                    next_value = alphabeta(next_state,next_depth,next_agent,alpha,beta)[1]
                    if ans_value > next_value:
                        ans_value = next_value
                        ans_action = [action]
                    elif ans_value==next_value:
                        ans_action.append(action)
                    if ans_value < alpha:
                        return  action,ans_value
                    beta = min(beta,ans_value)
                return random.choice(ans_action), ans_value

        return alphabeta(gameState,self.depth,0,float('-Inf'),float('Inf'))[0]
        # End your code (Part 2)
```

- **Part 3: Expectimax Search**
  1. **Construct and call Expectimax function**
  2. **In the function, first check if we reach the end or not.**
     Than calculate the next state, including state ,depth, agent etc.
     According to the agent we look at, from the agent we look at,
     Pac-man: choose the max value and its corresponding action
     Ghost : calculate and return the average value that ghost may act

3. **Return the value and action**

```python
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)
        """
        we construct a function of expectimax with parameters: state, depth, agent.
        and return the value and which action should take (construct the tree need value, answer need action)

        first we detect the tree is finish or not depand on depth or the game is end,
        and return action(stop,value)

        second, we compute the next parameters that need to pass into next iterate function,

        based on the agent (pacman or ghost)
        if the agent is pacman the process is same as the part1

        but for the ghost, since the ghosts' action is random we need to return the average value
        to simulate the real action of ghost
        """
        def expectimax(state,depth,agent):
            if depth ==0 or state.isWin() or state.isLose():
                return 0,self.evaluationFunction(state)

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

            legal_act = state.getLegalActions(agent)

            next_state = [state.getNextState(agent, act) for act in legal_act]
            next_value = [expectimax(nextstate,next_depth,next_agent)[1] for nextstate in next_state]
            if agent ==0:
                ans_value = max(next_value)
                for i in range(len(next_value)):
                    if next_value[i] == ans_value:
                        return legal_act[i], ans_value
            else:
                return 0, sum(next_value)/len(next_value)

        return expectimax(gameState,self.depth,0)[0]

        # End your code (Part 3)
```

4. **Part 4: Evaluation Function**
    1. **Get information about food pac-man and ghost**
    2. **Calculate the score of the game state**

       I divide the score into three part:

1. ghost score:

if ghost is scared: we want to eat the ghost -> the closer to ghost, the higher score

if ghost is not scared: we want to leave the ghost -> the farer to ghost, the higher score

2. food score:

the more food is on the map the lower score we get

the closer to food the higher score we get

3. capsules score:

the more food is on the map the lower score we get

the closer to food the higher score we get

note that the weight of capsules is greater than food

3. Return the total value

```
def betterEvaluationFunction(currentGameState):

    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (Part 4).
    """
    """
    first, from the gamestate, wecan get information about the food, pac-man and ghost
    calculate the score of the state from three part
    1.ghost score:
        if ghost is scared: we want to eat the ghost -> the closer to ghost, the higher score
        if ghost is not scared: we want to leave the ghost -> the farer to ghost, the higher score
    2.food score:
        the more food is on the map the lower score we get
        the closer to food the higher score we get
    3.capsules score:
        the more food is on the map the lower score we get
        the closer to food the higher score we get
        note that the weight of capsules is greater than food
    """

```

```python
    # Begin your code (Part 4)
    pac_pos = currentGameState.getPacmanPosition()


    ghost_states = currentGameState.getGhostStates()
    ghost_pos = currentGameState.getGhostPositions()

    scared_times = [g_s.scaredTimer for g_s in ghost_states]
    ghost_distance = [util.manhattanDistance(pac_pos,g_p)for g_p in ghost_pos]

    ghost_score = 0
    scared_time = sum(scared_times)
    min_ghost_distance = min(ghost_distance)

    if 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

    food = currentGameState.getFood()
    food = food.asList()
    food_distance = [util.manhattanDistance(pac_pos,f_p) for f_p in food]

    food_score = -5*len(food_distance)
    if len(food_distance)>0:
        min_food_distance = min(food_distance)
        food_score +=10/min_food_distance+10

    capsules = currentGameState.getCapsules()
    #print(capsules_distance)
    capsules_score = len(capsules)*(-100)
    return food_score+ghost_score+currentGameState.getScore()+capsules_score
    # End your code (Part 4)

# Abbreviation
better = betterEvaluationFunction
```

Part II. Results & Analysis (5%):

- **Overall result：**

```
Finished at 3:15:01

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

- **Part1:**

```
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 ###
```

- **Part2:**

```
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:                              depth.test
          Open file in editor (ctrl + click)
*** PASS:                              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 ###
```

- **Part3:**

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

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

- **Part4:**

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

Pacman emerges victorious! Score: 1284
Pacman emerges victorious! Score: 1331
Pacman emerges victorious! Score: 1350
Pacman emerges victorious! Score: 1135
Pacman emerges victorious! Score: 1356
Pacman emerges victorious! Score: 1348
Pacman emerges victorious! Score: 1227
Pacman emerges victorious! Score: 1173
Pacman emerges victorious! Score: 1286
Pacman emerges victorious! Score: 1282
Average Score: 1277.2
Scores:        1284.0, 1331.0, 1350.0, 1135.0, 1356.0, 1348.0, 1227.0, 1173.0, 1286.0, 1282.0
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
***     1277.2 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 ###
```