Homework 3: Multi-Agent Search

Part I. Implementation (5%):

Part 1 Minimax

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

        def countmax(depth, gameState, index_agent):
            '''
            depth = the depth this state is in
            gameState = the state that is going to be execute
            index_agent = the index of the agent (pacman = 0, ghosts = 1~(num_agents-1))
            bestscore is either min/max score in ghost/pacman
            bestmove only matters when it comes to pacman(index_agent=0)
            countmax returns the bestscore and bestmove
            '''
            num_agents = gameState.getNumAgents() # number of agents
            legal_moves = gameState.getLegalActions(index_agent) # legal moves of the executing state
            if not len(legal_moves): # no legal_moves = done with game at the state (win or lose)
                return self.evaluationFunction(gameState), "0" # return the points
            '''
            in each condition, scores means the scores of all childstates
            take all possible childstates from legalmove and count the min/max of it
            '''
            if depth == 1 and index_agent == num_agents-1:
                '''
                depth = 1 and index = num_agents-1 means it is the terminal state
                it is also the leaf process of the recursion function call
                simply uses the evaluationFunction to calculate the scores since it's the terminal tate
                return best_score, which is the minimum score of all posible scores
                '''
                scores = []
                for action in legal_moves:
                    GameState = gameState.getNextState(index_agent, action)
                    scores.append(self.evaluationFunction(GameState))
                best_score = min(scores)
                best_indices = [index for index in range(len(scores)) if scores[index] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score, legal_moves[chosen_idx]

            elif index_agent == 0:
                '''
                index_agent = 0 means it is the time when pacman(max_player)'s time
                countmax(depth, Gamestate, 1) is used because the ghost with index 1 starts
                return best_score (max score of scores)
                and bestMove (which matters because the original get_action function relies on it)
                '''
                scores = []
                for action in legal_moves:
                    GameState = gameState.getNextState(0, action)
                    scores.append(countmax(depth, GameState, 1))
                best_score = max(scores)[0]
                best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score, legal_moves[chosen_idx]

```

```python
            elif index_agent == num_agents-1:
                '''
                index_agent = num_agents-1 but depth ≠ 1 means the childstate is pacman in the next depth
                thus, countmax(depth-1, Gamestate, 0) is used
                instead of countmax(depth, Gamestate, index_agent+1)
                return best_score, which is the minimum of scores.
                '''
                scores=[]
                for action in legal_moves:
                    GameState = gameState.getNextState(index_agent,action)
                    scores.append(countmax(depth-1, GameState, 0))
                best_score = min(scores)[0]
                best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score, legal_moves[chosen_idx]
            else:
                '''
                those 0 < index_agent < num_agent-1 (no matter the depth) would be execute here
                countmax(depth, Gamestate, index_agent+1) is used since there is no need to change depth
                and the next agent to be executed is index_agent+1
                '''
                scores=[]
                for action in legal_moves:
                    GameState = gameState.getNextState(index_agent, action)
                    scores.append(countmax(depth, GameState, index_agent+1))
                best_score = min(scores)[0]
                best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score, legal_moves[chosen_idx]

        # take the best move from function countmax recursively
        best_score, bestmove = countmax(self.depth,gameState,0)
        return bestmove
        # End your code (Part 1)
```

Part 2 AlphaBeta

```python
        # Begin your code (Part 2)

        def countmax(depth, gameState, index_agent, alpha_beta):
            # alpha_beta[0] = alpha
            # alpha_beta[1] = beta
            num_agents = gameState.getNumAgents()
            legal_moves = gameState.getLegalActions(index_agent)
            if not len(legal_moves): # no legal_moves = done with game at the state (win or lose)
                return self.evaluationFunction(gameState),"0" # return the points
            if depth == 1 and index_agent == num_agents-1:
                '''
                this is a min_player, so update beta and prune if best_score(v) is less than alpha
                the updated alpha_beta list will be changed in the function that call this function
                because list is mutable
                '''
                scores = []
                best_score = float("inf")
                for action in legal_moves:
                    GameState = gameState.getNextState(index_agent, action)
                    thisScore = self.evaluationFunction(GameState)
                    scores.append(thisScore)
                    best_score = min(best_score, thisScore)
                    if best_score < alpha_beta[0]:
                        return best_score, action
                    alpha_beta[1] = min(alpha_beta[1], best_score)
                best_indices = [index for index in range(len(scores)) if scores[index] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score, legal_moves[chosen_idx]

```

```python
            elif index_agent == 0:
                '''
                this is a max_player, so update alpha and prune if best_score is larger than beta
                the updated alpha_beta list will be changed in the function that call this function
                because list is mutable
                ab is used in order to not chanage the beta's value
                (since this is a max_player, value of beta shouldn't be changed)
                '''
                scores = []
                best_score = -float("inf")

                for action in legal_moves:
                    GameState = gameState.getNextState(0, action)
                    ab = [alpha_beta[0], alpha_beta[1]]
                    thisScore = countmax(depth, GameState, 1, ab)
                    scores.append(thisScore)
                    best_score = max(best_score,thisScore[0])
                    if best_score > alpha_beta[1]:
                        return best_score, action
                    ab[0] = max(ab[0], best_score)
                    alpha_beta[0] = ab[0]
                best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score,legal_moves[chosen_idx]

```

```python
            elif index_agent == num_agents-1:
                '''
                this is a min_player, so update beta and prune if best_score(v) is less than alpha
                the updated alpha_beta list will be changed in the function that call this function
                because list is mutable
                '''
                scores = []
                best_score = float("inf")
                for action in legal_moves:
                    ab = [alpha_beta[0], alpha_beta[1]]
                    GameState = gameState.getNextState(index_agent, action)
                    thisScore = countmax(depth-1, GameState, 0, ab)
                    scores.append(thisScore)
                    best_score = min(best_score, thisScore[0])
                    if best_score < alpha_beta[0]:
                        return best_score, action
                    ab[1] = min(ab[1], best_score)
                    alpha_beta[1] = ab[1]
                best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                return best_score,legal_moves[chosen_idx]
```

```python
        else:
            '''
            those 0 < index_agent < num_agent-1 (no matter the depth) would be execute here
            this is a min_player, so update beta and prune if best_score(v) is less than alpha
            the updated alpha_beta list will be changed in the function that call this function,
            because list is mutable
            '''
            scores = []
            best_score = float("inf")
            for action in legal_moves:
                ab = [alpha_beta[0], alpha_beta[1]]
                GameState = gameState.getNextState(index_agent, action)
                thisScore = countmax(depth, GameState, index_agent+1, ab)
                scores.append(thisScore)
                best_score = min(best_score, thisScore[0])
                if best_score < alpha_beta[0]:
                    return best_score, action
                ab[1] = min(ab[1], best_score)
                alpha_beta[1] = ab[1]
            best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
            chosen_idx = random.choice(best_indices) # pick randomly among the best
            return best_score,legal_moves[chosen_idx]
    #alpha_beta is sent as a parameter, which is new from minimax
    best_score, bestmove = countmax(self.depth, gameState, 0, [-float("inf"), float("inf")])
    return bestmove
    # End your code (Part 2)
```

Part 3 Expectimax

```python
    # Begin your code (Part 3)

    def countmax(depth, gameState, index_agent):
        num_agents = gameState.getNumAgents()
        legal_moves = gameState.getLegalActions(index_agent)
        if not len(legal_moves): # no legal_moves = done with game at the state (win or lose)
            return self.evaluationFunction(gameState), "0" # return the points
        if depth == 1 and index_agent == num_agents-1:
            scores = []
            for action in legal_moves:
                GameState = gameState.getNextState(index_agent,action)
                scores.append(self.evaluationFunction(GameState))
            total = 0
            for value in scores: total += value
            best_score = total/len(scores)
            '''
            instead of finding the minimum of the scores,
            find the average score as the best score
            bestmove is not importent here,
            so just return legal_moves[0] is okay
            '''
            return best_score, legal_moves[0]

```

```python
            elif index_agent == 0:
                scores = []
                for action in legal_moves:
                    GameState = gameState.getNextState(0, action)
                    scores.append(countmax(depth, GameState, 1))
                best_score = max(scores)[0]
                best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
                chosen_idx = random.choice(best_indices) # pick randomly among the best
                '''
                the max_player, which is the pacman, remains the same as minimaxAgent
                '''
                return best_score, legal_moves[chosen_idx]

            elif index_agent == num_agents-1:
                scores = []
                for action in legal_moves:
                    GameState = gameState.getNextState(index_agent, action)
                    scores.append(countmax(depth-1, GameState, 0))
                total = 0
                for value in scores:
                    total += value[0]
                best_score = total/len(scores)
                '''
                instead of finding the minimum of the scores,
                find the average score as the best score
                bestmove is not importent here, so just return legal_moves[0] is okay
                '''
                return best_score, legal_moves[0]

            else:
                '''
                those 0 < index_agent < num_agent-1 (no matter the depth) would be execute here
                '''
                scores = []
                for action in legal_moves:
                    GameState = gameState.getNextState(index_agent, action)
                    scores.append(countmax(depth, GameState, index_agent+1))
                total = 0
                for value in scores:
                    total += value[0]
                best_score = total/len(scores)
                '''
                instead of finding the minimum of the scores,
                find the average score as the best score
                bestmove is not importent here, so just return legal_moves[0] is okay
                '''
                return best_score,legal_moves[0]

        best_score, bestmove = countmax(self.depth, gameState, 0)
        return bestmove
        # End your code (Part 3)
```

Part II. Results & Analysis (5%):

```
AI_HW3 — -zsh — 80×24
***          >= 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 20:33:28

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