CS4386 Assignment 1 (Semester B, 2022-2023)

Name: LUO Peiyuan SID: 56642728

Table of Contents

Utility Functions

Board Class

This class stores some utility function related to the game board checking as follows. And each board object stores the current player of the board, state matrix, and winner of this board(if applicable).

```
class Board:
    alfreddLUO
    def __init__(self, player, state):...
    alfreddLUO
    def update_winner(self, winner):...
    alfreddLUO
    def makeMove(self, move):...
    alfreddLUO
    def currentPlayer(self):...
    alfreddLUO
    def check_total_distance_from_sheep_to_wolf(self):...
    alfreddLUO
    def check_num_of_ways_wolf_trapped(self):...
    alfreddLUO
    def check_num_of_sheep_and_num_of_to_be_killed(self):...
    alfreddLUO
    @staticmethod
    def calculate_shortened_distance_by_current_move(org_board, cur_board):...
    alfreddLUO
    def calculate_sheep_scores(self, num_of_sheep_killed, shortened_distance, num_of_trapped_ways, trapped_wolf_num,num_of_to_be_killed_sheep):...
    alfreddLUO
    def calculate_wolf_scores(self,num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_be_killed_sheep)
    alfreddLUO
    def evaluate(self, player, gameEnds, org_board):...
    alfreddLUO
    def check_wolf_trapped_in_this_way(self, i, j):...
    alfreddLUO
    def game_ends(self):...
    alfreddLUO
    def getWolfMoves(self):...
    alfreddLUO
    def getSheepMoves(self):...
```

makeMove function

This method would take the move as input, change the game state, update the next player and return a new board object.

```
def makeMove(self, move):
    [start_row, start_col, end_row, end_col] = move
    matrix2 = copy.deepcopy(self.state)
    matrix2[end_row, end_col] = self.player
    matrix2[start_row, start_col] = 0
    nextPlayer = 2 if self.player == 1 else 2
    return Board(nextPlayer, matrix2)
```

check_total_distance_from_sheep_to_wolf

Return the total distance from sheep to wolf of current state.

Objective: for the later heuristic evaluation function

```
def check_total_distance_from_sheep_to_wolf(self):
    wolf1_location = self.wolf1_location
    wolf2_location = self.wolf2_location
    total_distance = 0
    board = self.state
    for i in range(5):
        for j in range(5):
            if board[i][j] == 1:
                dist_current_to_wolf1 = abs(i - wolf1_location[0]) + abs(j - wolf1_location[1])
                dist_current_to_wolf2 = abs(i - wolf2_location[0]) + abs(j - wolf2_location[1])
                total_distance += (dist_current_to_wolf1 + dist_current_to_wolf2)
    return total_distance
```

calculate_shortened_distance_by_current_move

Return the difference of the total distance of all wolves and sheep between the original board and current board. If the returned value is positive, then the distance between wolf and sheep are shortened. Otherwise, the distance increased.

Objective: for the later heuristic evaluation function

```
alfreddLUO
@staticmethod
def calculate_shortened_distance_by_current_move(org_board, cur_board):
    return org_board.check_total_distance_from_sheep_to_wolf()-cur_board.check_total_distance_from_sheep_to_wolf()
```

check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped

Implemntation: check the current state, and output the number of ways that wolf's next move are trapped (max 4 for each wolf) and number of wolf that is already trapped(max 2).
*trapped: the wolf can't move in this direction
Objective: for the later heuristic evaluation function

```
alfreddLUO *
def check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped(self):
    wolf_locations = [self.wolf1_location, self.wolf2_location]
    num_of_trapped = 0
    trapped_wolf_num = 0
    for wolf_loc in wolf_locations:
        wolf_loc_x = wolf_loc[0]
        wolf_loc_y = wolf_loc[1]
        trapped_ways = [[wolf_loc_x - 1, wolf_loc_y], [wolf_loc_x, wolf_loc_y - 1], [wolf_loc_x, wolf_loc_y + 1],
                        [wolf_loc_x + 1, wolf_loc_y]]
        trapped = False
        for way in trapped_ways:
            if self.check_wolf_trapped_in_this_way(way[0], way[1]):
                num_of_trapped += 1
        if num_of_trapped == 4:
            trapped_wolf_num += 1
            num_of_trapped -= 4
    return num_of_trapped, trapped_wolf_num
```

check_num_of_sheep_and_num_of_to_be_killed

return the number of sheep at this state and the number of sheep that can be killed at next step(one empty column away from wolf).

Objective: for the later heuristic evaluation function

```
def check_num_of_sheep_and_num_of_to_be_killed(self):
    sheep_cnt = 0
    num_of_to_be_killed_sheep = 0
    board = self.state
    for i in range(5):
        for j in range(5):
            tmp_num = 0
            if board[i][j] == 1:
                sheep_cnt += 1
                if i + 2 < 5:
                    if board[i + 2, j] == 1 and board[i + 1, j] == 0:
                        tmp_num += 1
                if i - 2 >= 0:
                    if board[i - 2, j] == 1 and board[i - 1, j] == 0:
                        tmp_num += 1
                if j + 2 < 5:
                    if board[i, j + 2] == 1 and board[i, j + 1] == 0:
                        tmp_num += 1
                if j - 2 >= 0:
                    if board[i, j - 2] == 1 and board[i, j - 1] == 0:
                        tmp_num += 1
                num_of_to_be_killed_sheep += tmp_num if tmp_num <= 1 else 1
    return sheep_cnt, num_of_to_be_killed_sheep
```

calculate_sheep_scores

calculate the heuristic value of the sheep when the board is not yet end but reach the max depth.

Objective: for the later heuristic evaluation function

```
def calculate_sheep_scores(self, num_of_sheep_killed, shortened_distance, num_of_trapped_ways, trapped_wolf_num,num_of_to_be_killed_sheep):
    alfreddLUO
    def calculate_trapped_scores(num):
        # res = 0
        # for i in range(num + 1):
        #     res += 2 * i ** 2
        return num
    return num_of_sheep_killed * (-8000) + shortened_distance * (-100) + calculate_trapped_scores(num_of_trapped_ways)*800 + trapped_wolf_num * 4000 \
        - num_of_to_be_killed_sheep
```

calculate_wolf_scores

calculate the heuristic value of the wolf when the board is not yet end but reach the max depth.

Objective: for the later heuristic evaluation function

```
def calculate_wolf_scores(self,num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_be_killed_sheep):
    alfreddLUO
    def calculate_trapped_scores(num):
        res = 0
        for i in range(num + 1):
            res += 4 * i ** 2
        return res
    # shorten_distance_from_sheep_to_wolf = shorten_distance_from_sheep_to_wolf if shorten_distance_from_sheep_to_wolf == 0 else 1
    return num_of_sheep_killed * 500 + shorten_distance_from_sheep_to_wolf*20 - 0 * calculate_trapped_scores(num_of_trapped_ways) + 300 * num_of_to_be_killed_sheep
```

check_wolf_trapped_in_this_way

check whether wolf is trapped in this way.

Objective: used in check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped function.

```
def check_wolf_trapped_in_this_way(self, i, j):
    board = self.state
    if i < 0 or i > 4 or j < 0 or j > 4:
        return True
    if board[i][j] != 0:
        return True
    return False
```

getWolfMoves

Return the valid moves for wolves.

getSheepMoves

Return the valid moves for sheep.

game_ends function

Return whether the game ends or not.

```python
def game_ends(self):
    board = self.state
    sheep_cnt = 0
    wolf1_exist = False
    wolf2_exist = False
    wolf1_trapped = False
    wolf2_trapped = False
    for i in range(5):
        for j in range(5):
            if board[i][j] == 0:
                continue
            elif board[i][j] == 1:
                sheep_cnt += 1
            elif board[i][j] == 2:
                if wolf1_exist == False:
                    wolf1_exist = True
                    self.wolf1_location = [i, j]
                else:
                    wolf2_exist = True
                    self.wolf2_location = [i, j]
                if self.check_wolf_trapped_in_this_way(i - 1, j) and self.check_wolf_trapped_in_this_way(i,
                                                                                                           j - 1) and self.check_wolf_trapped_in_this_way(
                    i + 1, j) and self.check_wolf_trapped_in_this_way(i, j + 1):
                    if wolf1_exist and not wolf2_exist:
                        wolf1_trapped = True
                    else:
                        wolf2_trapped = True
    if sheep_cnt <= 2:
        self.update_winner(2)
        return True
    if wolf1_exist and wolf2_exist and wolf1_trapped and wolf2_trapped:
        self.update_winner(1)
        return True
    return False
```

Methodology

AB_Minmax

Objective

I use AB_Minmax algorithm as the framework of my code structure.
To simulate several rounds of the game and use the max value of the evaluation result for my turn and find the minimum of the evaluation result for opponent's turn. And by using this, we would see a bigger picture when we make decision. However, with limited time, we should limit the max depth of the ab minmax, and also use alpha beta pruning approach to reduce running time.

Implementation

Max Depth: 4 for Wolf, 3 for Sheep

Location: I use ab minmax function inside getBestMove() function to find the best move using ab minmax method.
All_moves: I generate the valid moves list accoding to player, and if board.currentPlayer() ==2 it would return all valid moves of wolves, else return all valid moves of sheep.
Ending condtion: to end the recursion function of ab minmax, the triggering condition is either game ends(one of the player wins) or currentDepth==maxDepth.

```
alfreddLUO *
def getBestMove(board, maxDepth, player):
    alfreddLUO *
    def ab_minmax(board, player, maxDepth, currentDepth, alpha, beta, org_board):
        gameEnds = board.game_ends()
        if gameEnds or currentDepth == maxDepth:
            return None, None, None, None, board.evaluate(player, gameEnds, org_board), currentDepth, None
        if board.currentPlayer() == 2:
            all_moves = board.getWolfMoves()
        else:
            all_moves = board.getSheepMoves()
        if board.currentPlayer() == player:
            best_start_row, best_start_col, best_end_row, best_end_col = None, None, None, None
            bestScore = -math.inf
            bestScoreDepth = math.inf
            bestScoreBoard = None
            for move in all_moves:
                newBoard = board.makeMove(move)
                _, _, _, _, currentScore_, currentScoreDepth, _ = ab_minmax(newBoard, player, maxDepth,
                                                                            currentDepth + 1, alpha,
                                                                            beta, org_board)
                currentScore = currentScore_
                alpha = max(alpha, currentScore)
                if currentScore > bestScore or (
                        currentScore == bestScore and currentScoreDepth < bestScoreDepth):
                    # print("Yes")
                    bestScore = currentScore
                    bestScoreDepth = currentScoreDepth
                    best_start_row, best_start_col, best_end_row, best_end_col = move[0], move[1], move[2], move[3]
                    bestScoreBoard = newBoard
                    if alpha >= beta:
                        return best_start_row, best_start_col, best_end_row, best_end_col, bestScore, bestScoreDepth, bestScoreBoard
        else:
            best_start_row, best_start_col, best_end_row, best_end_col = None, None, None, None
            bestScore = math.inf
            bestScoreDepth = math.inf
            bestScoreBoard = None
            for move in all_moves:
                newBoard = board.makeMove(move)
                # if player == 2:
                _, _, _, _, currentScore_, currentScoreDepth, _ = ab_minmax(newBoard, player, maxDepth,
                                                                            currentDepth + 1, alpha,
                                                                            beta, org_board)
                # currentScore = currentScore_
                currentScore = currentScore_
                beta = min(beta,currentScore)
                if currentScore < bestScore or (
                        currentScore == bestScore and currentScoreDepth < bestScoreDepth):  # or currentScore == 1000:
                    bestScore = currentScore
                    bestScoreDepth = currentScoreDepth
                    best_start_row, best_start_col, best_end_row, best_end_col = move[0], move[1], move[2], move[3]
                    bestScoreBoard = newBoard
                    if alpha >= beta:
                        return best_start_row, best_start_col, best_end_row, best_end_col, bestScore, bestScoreDepth, bestScoreBoard

        return best_start_row, best_start_col, best_end_row, best_end_col, bestScore, bestScoreDepth, bestScoreBoard

    best_start_row, best_start_col, best_end_row, best_end_col, bestScore, bestScoreDepth, bestScoreBoard = ab_minmax(board, player, maxDepth, 0,
                                                                                                                      -math.inf, math.inf, board)
    # print("Best Move: ", best_start_row, best_start_col, best_end_row, best_end_col, bestScore, bestScoreDepth)
    return best_start_row, best_start_col, best_end_row, best_end_col, bestScore, bestScoreDepth
```

Heuristic Evaluation

I used the heuristic evaluation method for the evaluation function of the game.

Exception: Under the condition that the game ends at this state: if the winner is current player, then return 100000, otherwise -100000. If the game won't end at this state, then return the heuristic function for the player.

```python
alfreddLUO *
def evaluate(self, player, gameEnds, org_board):
    if gameEnds:
        if self.winner == player:
            return 100000
        else:
            return -100000
    if player == 2:
        org_sheep_cnt, org_to_be_killed_sheep = org_board.check_num_of_sheep_and_num_of_to_be_killed()
        cur_sheep_cnt, cur_to_be_killed_sheep = self.check_num_of_sheep_and_num_of_to_be_killed()
        num_of_sheep_killed = org_sheep_cnt - cur_sheep_cnt
        num_of_trapped_ways, trapped_wolf_num = self.check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped()
        # total_distance_from_sheep_to_wolf = self.check_total_distance_from_sheep_to_wolf()
        shorten_distance_from_sheep_to_wolf = Board.calculate_shortened_distance_by_current_move(org_board, self)
        num_of_to_be_killed_sheep = cur_to_be_killed_sheep
        # print("Wolf (tmpScores: ", self.calculate_wolf_scores(num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_
        return self.calculate_wolf_scores(num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_be_killed_sheep)
    elif player == 1:
        org_sheep_cnt, org_to_be_killed_sheep = org_board.check_num_of_sheep_and_num_of_to_be_killed()
        cur_sheep_cnt, cur_to_be_killed_sheep = self.check_num_of_sheep_and_num_of_to_be_killed()
        num_of_sheep_killed = org_sheep_cnt - cur_sheep_cnt
        org_num_of_trapped_ways, org_trapped_wolf_num = org_board.check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped()
        cur_num_of_trapped_ways, cur_trapped_wolf_num = self.check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped()
        delta_num_of_trapped_ways, delta_trapped_wolf_num = cur_num_of_trapped_ways-org_num_of_trapped_ways, cur_trapped_wolf_num-org_trapped_wolf_num
        num_of_to_be_killed_sheep = cur_to_be_killed_sheep
        shorten_distance_from_sheep_to_wolf = Board.calculate_shortened_distance_by_current_move(org_board, self)
        total_scores = self.calculate_sheep_scores(num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, delta_num_of_trapped_ways, delta_trapped_wolf_num,
                                                   num_of_to_be_killed_sheep)
        # print("Sheep (tmpScores: ",total_scores)
        return total_scores
    # print("No")
    return 0
```

Wolf Heuristic Function

Overview

The below is the main calculation related to Wolf Heuristic Function.

```python
if player == 2:
    org_sheep_cnt, org_to_be_killed_sheep = org_board.check_num_of_sheep_and_num_of_to_be_killed()
    cur_sheep_cnt, cur_to_be_killed_sheep = self.check_num_of_sheep_and_num_of_to_be_killed()
    num_of_sheep_killed = org_sheep_cnt - cur_sheep_cnt
    num_of_trapped_ways, trapped_wolf_num = self.check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped()
    # total_distance_from_sheep_to_wolf = self.check_total_distance_from_sheep_to_wolf()
    shorten_distance_from_sheep_to_wolf = Board.calculate_shortened_distance_by_current_move(org_board, self)
    num_of_to_be_killed_sheep = cur_to_be_killed_sheep
    # print("Wolf (tmpScores: ", self.calculate_wolf_scores(num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_
    return self.calculate_wolf_scores(num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_be_killed_sheep)
def calculate_wolf_scores(self,num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, num_of_trapped_ways, org_board,num_of_to_be_killed_sheep):
    '''
    calculate the heuristic value of the wolf when the board is not yet end but reach the max depth.
    Objective:  for the later heuristic evaluation function
    '''
    alfreddLUO
    def calculate_trapped_scores(num):
        res = 0
        for i in range(num + 1):
            res += 4 * i ** 2
        return res
    # shorten_distance_from_sheep_to_wolf = shorten_distance_from_sheep_to_wolf if shorten_distance_from_sheep_to_wolf == 0 else 1
    return num_of_sheep_killed * 500 + shorten_distance_from_sheep_to_wolf*20 - 0 * calculate_trapped_scores(num_of_trapped_ways) + 300 * num_of_to_be_killed_sheep
```

Parameters

Parameters' Importance Ranking

num_of_sheep_killed>num_of_to_be_killed_sheep>shorten_distance_from_sheep_to_wolf

How to distinguish their differences

By providing different weight to these parameters when do calculation, it would make the influence of each parameter differs a lot.

Parameters that are taken into account

num_of_sheep_killed

- meaning: the number of sheep that are killed between the previous taken move and current depth, calculated by the number of the sheep before this turn minus the number of sheep in current state.
- Weight: 500
- Importance: very high, because the more sheep is eaten by wolf, the higher probability the wolf would win.

num_of_to_be_killed_sheep

- meaning: after all the moves made, the number of sheep that can be killed by the wolf in next turn of wolf, calculated by counting the number of sheep that are only one empty column away from the wolf.
- Weight: 300
- Importance: high, because the more sheep can be killed by wolf in next move, the higher probability the wolf would win under this situation. But the weight should be lighter than the 500.

shorten_distance_from_sheep_to_wolf

- meaning: the shortened distance between all wolves and sheep after all the move is made, calculated by the distance of sheep and wolves before this turn minus the distance of sheep in current state.
- Weight: 20
- Importance: high, as the closer the wolf is to sheep, the higher probability the wolf would win.

Sheep Heuristic Function

Overview

The below is the main calculation related to Sheep Heuristic Function.

```python
elif player == 1:
    org_sheep_cnt, org_to_be_killed_sheep = org_board.check_num_of_sheep_and_num_of_to_be_killed()
    cur_sheep_cnt, cur_to_be_killed_sheep = self.check_num_of_sheep_and_num_of_to_be_killed()
    num_of_sheep_killed = org_sheep_cnt - cur_sheep_cnt
    org_num_of_trapped_ways, org_trapped_wolf_num = org_board.check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped()
    cur_num_of_trapped_ways, cur_trapped_wolf_num = self.check_num_of_ways_wolf_trapped_and_num_of_wolf_trapped()
    delta_num_of_trapped_ways, delta_trapped_wolf_num = cur_num_of_trapped_ways-org_num_of_trapped_ways, cur_trapped_wolf_num-org_trapped_wolf_num
    num_of_to_be_killed_sheep = cur_to_be_killed_sheep
    shorten_distance_from_sheep_to_wolf = Board.calculate_shortened_distance_by_current_move(org_board, self)
    total_scores = self.calculate_sheep_scores(num_of_sheep_killed, shorten_distance_from_sheep_to_wolf, delta_num_of_trapped_ways, delta_trapped_wolf_num,
                                               num_of_to_be_killed_sheep)
    # print("Sheep (tmpScores: ",total_scores)
    return total_scores
def calculate_sheep_scores(self, num_of_sheep_killed, shortened_distance, num_of_trapped_ways, trapped_wolf_num,num_of_to_be_killed_sheep):
    '''
    calculate the heuristic value of the sheep when the board is not yet end but reach the max depth.

    Objective:  for the later heuristic evaluation function
    '''
    alfreddLUO
    def calculate_trapped_scores(num):
        # res = 0
        # for i in range(num + 1):
        #     res += 2 * i ** 2
        return num
    return num_of_sheep_killed * (-8000) + shortened_distance * (-100) + calculate_trapped_scores(num_of_trapped_ways)*800 + trapped_wolf_num * 4000 \
        num_of_to_be_killed_sheep
```

Parameters

Parameters' Importance Ranking

num_of_sheep_killed > trapped_wolf_num > shorten_distance_from_sheep_to_wolf >num_of_trapped_ways > num_of_to_be_killed_sheep

How to distinguish their differences

By providing different weight to these parameters when do calculation, it would make the influence of each parameter differs a lot.

num_of_sheep_killed:

- meaning: the number of sheep that are killed between the previous taken move and current depth, calculated by the number of the sheep before this turn minus the number of sheep in current state.
- Weight: -20000
- Importance: very high, because the less sheep is eaten by wolf, the higher probability the sheep would win.

trapped_wolf_num:

- meaning: the number of wolves that are trapped by sheep(can't move any more). Each wolf has four ways (above, below, right, left). If all four ways of a wolf are trapped, then we say the wolf is trapped.
- Weight: 4000
- Importance: high, the more wolves that are trapped, then the less dangerous for sheep, and the more probability for sheep to trap the wolf and wins.

shorten_distance_from_sheep_to_wolf:

- meaning: the shortened distance between all wolves and sheep after all the move is made, calculated by the distance of sheep and wolves before this turn minus the distance of sheep in current state.
- Weight: -100
- Importance: high, as the farer the sheep is wolves, the higher probability the wolf would win, because the wolf would take more moves to get close to sheep.

num_of_trapped_ways:

- meaning: the number of ways that the wolf' move is trapped. Each wolf has four ways (above, below, right, left). If a ways is trapped then the wolf can't move in this direction.
- Weight: 800
- Importance: high, the more ways that the wolves are trapped, then the more limit for wolves' move, then the less probability for wolf to win, and the more probability for sheep to trap the wolf and wins.

num_of_to_be_killed_sheep:

- meaning: after all the moves made, the number of sheep that can be killed by the wolf in next turn of wolf, calculated by counting the number of sheep that are only one empty column away from the wolf.
- Weight: 1
- Importance: very low. Because for wolf's turn, it needs to wait for another turn to kill the to-be-killed sheep, which increase the uncertainty of this value. Hence, I only use this value to differ slightly.

APPENDIX

My source code is already pushed to my personal github:
https://github.com/alfreddLUO/AI-Algorithm-for-Wolves-Eats-Sheep-Game.git