



T.C.

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE 4082 – Assignment 3

Report

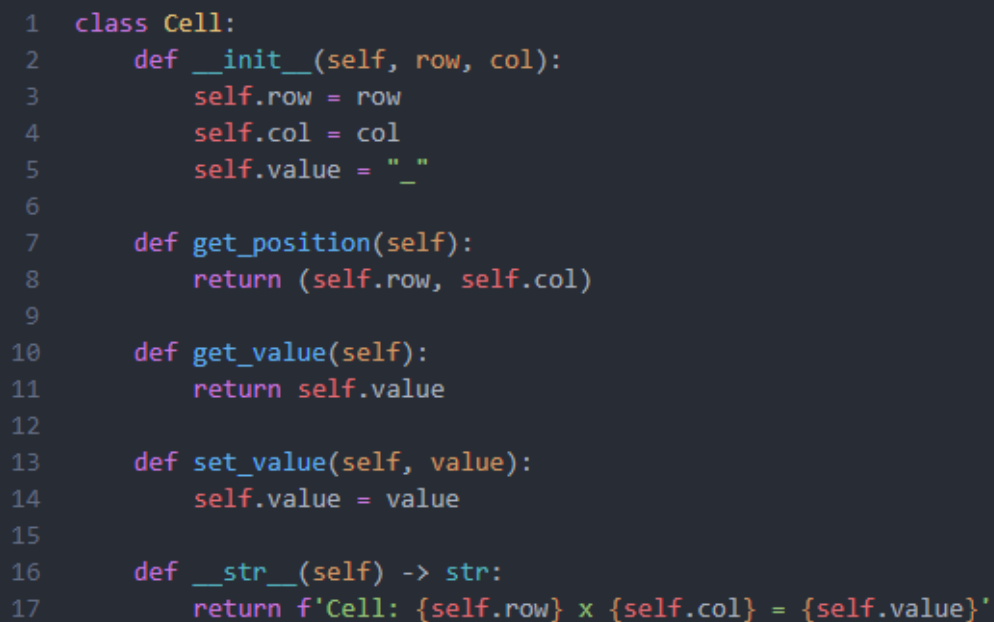
Group Members

Ömercan Göktaş – 150119671

1. Cell Class

This "Cell" class represents a cell on a game board or matrix. The class creates a cell with a specific row and column position and initially sets the cell's value to "_". For example, by using this class in a game board, you can represent each cell and track the progress of the game. The class provides essential functionalities for each cell on the game board or matrix.


- **get_position(self):** This method returns the position of the cell as a tuple (row, column).

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and defines a class named 'Cell'. The code is as follows:

```
1 class Cell:
2     def __init__(self, row, col):
3         self.row = row
4         self.col = col
5         self.value = "_"
6
7     def get_position(self):
8         return (self.row, self.col)
9
10    def get_value(self):
11        return self.value
12
13    def set_value(self, value):
14        self.value = value
15
16    def __str__(self) -> str:
17        return f'Cell: {self.row} x {self.col} = {self.value}'
```

2. Board Class

The Board class is designed to represent a game board. This constructor method takes basic information to create a game board and determines the size of the board. Each cell on the board is initially empty and represented by an instance of the Cell class. The `self.made_moves` list is used to track the moves made during the game.



```

1  class Board:
2      def __init__(self, height=5, width=5):
3          self.height = height
4          self.width = width
5          self.cells = []
6          self.made_moves = []

```

- **create_board(self):** This method's purpose is to create the game board. It achieves this by iterating through each row and column, creating a new instance of the Cell class for each cell with specific row and column values. These cells are then added to the board.
- **fill_corner_cells(self, value="S"):** This method's purpose is to fill the corner cells of the game board with a specified value, defaulting to "S". The corner cells refer to the cells located at the four corners of the board. The method achieves this by using the change_cell_value method to set the value of each corner cell.
- **add_cell(self, cell):** This method appends a provided cell to the list of cells on the game board.
- **print_board(self):** This method prints the current state of the game board, displaying the row and column numbers along with the values of each cell
- **change_cell_value(self, row, col, value):** This method changes the value of a specified cell on the game board. It searches for the cell based on provided row and column values, updating its value if the cell is empty.
- **get_available_moves(self):** This method returns the available moves on the game board by calling the private method __available_moves(). It serves as an interface for obtaining the list of possible moves during the gameplay.
- **__available_moves(self):** This private method, generates a list of available moves on the game board. It iterates through each cell on the board, and if the cell is empty (with a value "_"), it appends two dictionaries to the available_moves list. Each dictionary represents a possible move, including the cell's position and two potential values, "O" and "S". The method returns the list of available moves, providing options for both "O" and "S" values in empty cells.

- **display_available_moves(self):**This method prints the available moves on the game board.
- **check_if_move_is_valid(self, row, col):**This method checks if a specified cell on the game board is empty, returning True if it is and False otherwise.
- **get_neighbours(self, row, col):**This method returns a dictionary containing the neighbors of a specified cell on the game board. It checks eight possible neighboring positions, including top left, top, top right, left, right, bottom left, bottom, and bottom right. If a neighbor exists at a given direction, its position is included in the dictionary; otherwise, it is set to None. This method is essential for examining the surroundings of a cell during gameplay, aiding in various game logic implementations.
- **display_neighbours(self, row, col):** This method prints the neighbors of a given cell on the game board.
- **check_score_for_move(self, row, col, value):** This method evaluates the potential score for a move at a given position on the game board, considering the placement of either "S" or "O." It examines the neighbors in different directions, assigning a score of 1 for each detected scoring pattern. The result is a list of dictionaries containing information about scoring directions, including score, value, cell, direction, and path. If no scoring move is identified, the method returns None, aiding strategic decision-making in the game.
- **get_best_move(self):**This method determines the best move for the AI player by evaluating available moves on the game board. It iterates through the moves, calculates their scores using the evaluate method, and selects the move with the highest score. If no positive-score move is found, it randomly selects a move. The method returns a dictionary with the best move's details: row, column, value to play ("S" or "O"), and associated score.
- **get_random_move(self):**This method selects a random move from the available moves on the board, calculates its score, and returns a dictionary with the move's details: row, column, value, and score. If the move has no score, it sets the score to 0.
- **move(self, row, col, value):** This method validates and executes a move, updating the cell with the provided value and tracking the move. It returns True for a valid move and False otherwise.

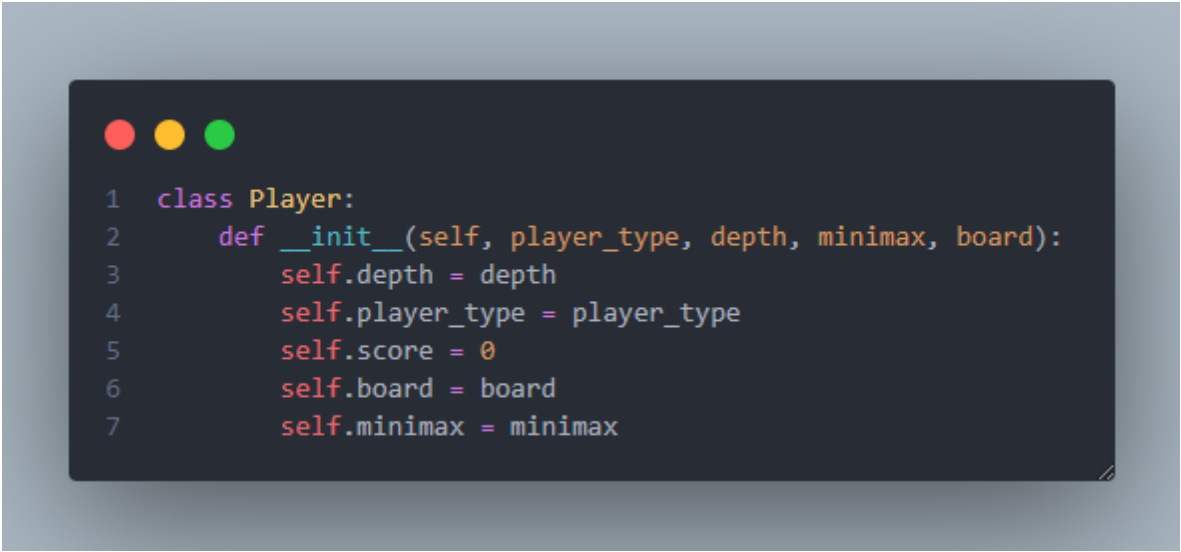
- **undo_move(self, row, col):** This method undoes the move at a specified position by resetting the cell value to "_".
- **is_game_over(self):** This method checks if the game is over by verifying if there are any available moves remaining on the board.
- **evaluate(self, row, col, value):** This method evaluates a move at a specified position, returning the row, column, value, and the obtained score.
- **alpha_beta_pruning(self, depth, alpha, beta, maximizing_player, row=None, col=None, value=None):** This method employs the alpha-beta pruning algorithm to evaluate the game state, utilizing a minimax approach. It explores potential moves up to a specified depth, returning the evaluation score for the current state. The maximizing_player parameter indicates whether the method is evaluating for the maximizing player (True) or the minimizing player (False). Considering the current move (row, col, value) during recursive exploration, the method returns the evaluation score using the evaluate method when the depth is 0 or the game is over. The algorithm efficiently prunes unnecessary branches based on alpha-beta values, optimizing the search space.

3. Player Class

The "Player" class encapsulates the characteristics and playing strategy of a participant in a game. Key attributes include:

- **player_type:** Represents the player's role in the game, such as 'max' or 'min'.
- **depth:** Specifies the depth of the minimax algorithm that the player utilizes.
- **score:** Maintains the player's score throughout the game.
- **board:** References the game board on which the player makes moves.
- **minimax:** Holds the minimax algorithm instance for decision-making.

These attributes collectively enable the representation of each player's strategy, facilitate move execution, and track the player's score during the course of the game. The class serves as a crucial component for managing player-related functionalities in the gaming environment.



```
1 class Player:
2     def __init__(self, player_type, depth, minimax, board):
3         self.depth = depth
4         self.player_type = player_type
5         self.score = 0
6         self.board = board
7         self.minimax = minimax
```

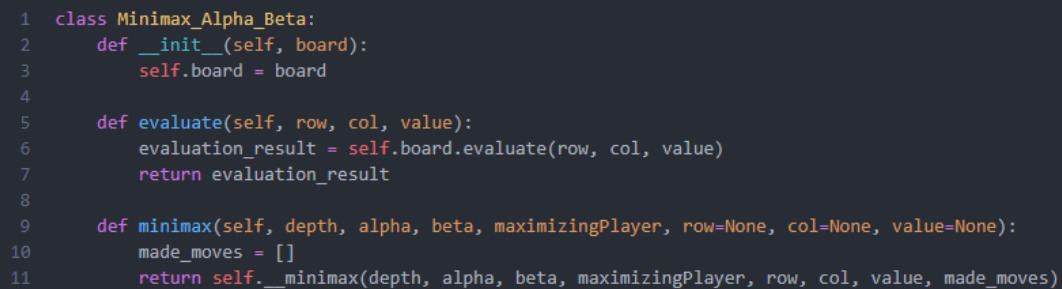
play(self):The play method in the "Player" class serves as the entry point for the player to make a move in the game. It checks the player's type ('max' or 'min') and calls the private `__play` method to execute the move strategy.

__play(self):is a method in the "Player" class that manages the execution of a player's move. It utilizes the minimax algorithm to determine the best move based on the current game

state. The method then executes the chosen move on the game board, checks if the move results in a score, and updates the player's score accordingly. If no score is gained, it makes a random move and checks for a score again. Overall, `__play` handles the decision-making process for a player during the game.

4. Minimax Class

The "Minimax_Alpha_Beta" class is designed to facilitate decision-making in a game by implementing the minimax algorithm with alpha-beta pruning.



```
1 class Minimax_Alpha_Beta:
2     def __init__(self, board):
3         self.board = board
4
5     def evaluate(self, row, col, value):
6         evaluation_result = self.board.evaluate(row, col, value)
7         return evaluation_result
8
9     def minimax(self, depth, alpha, beta, maximizingPlayer, row=None, col=None, value=None):
10        made_moves = []
11        return self.__minimax(depth, alpha, beta, maximizingPlayer, row, col, value, made_moves)
```

- **evaluate(row, col, value):** Evaluates the current state of the game at a specific move and returns the score.
- **minimax(depth, alpha, beta, maximizingPlayer, row=None, col=None, value=None):** Initiates the minimax algorithm with alpha-beta pruning. It explores available moves up to a specified depth, considering the maximizing or minimizing player. The result includes the score, the move made, and the sequence of moves leading to that state.
- **def __minimax(self, depth, alpha, beta, maximizing_player, row=None, col=None, value=None, current_moves=None):**

1. max_eval Part (maximizing_player=True):

This section represents the situation where the player is attempting to maximize. It evaluates each available move, traversing through them within the alpha-beta window to prune unnecessary branches and optimize the search space.

```
1 def __minimax(self, depth, alpha, beta, maximizing_player, row=None, col=None, value=None, current_moves=None):
2     if depth == 0 or self.board.is_game_over():
3         score = self.evaluate(row, col, value)['score']
4         return {'score': score, 'move': {'row': row, 'col': col, 'value': value}, 'moves': current_moves.copy()}
5
6     if maximizing_player:
7         max_eval = {'score': -sys.maxsize, 'move': None, 'moves': current_moves.copy()}
8         for move in self.board.get_available_moves():
9             row = move['available_move'][0]
10            col = move['available_move'][1]
11            value = move['value']
12
13            self.board.move(row, col, value)
14            current_moves.append({'move': (row, col), 'value': value, 'player': 'max'})
15            eval = self.__minimax(depth - 1, alpha, beta, False, row, col, value, current_moves)
16            self.board.undo_move(row, col)
17
18            if eval['score'] == 0:
19                random_move = self.board.get_best_move()
20                row = random_move['row']
21                col = random_move['col']
22                value = random_move['value']
23                eval['score'] = random_move['score']
24                current_moves.pop()
25                current_moves.append({'move': (row, col), 'value': value, 'player': 'max'})
26                eval['moves'] = current_moves.copy()
27
28            check_score = self.board.check_score_for_move(row, col, value)
29            if check_score != None and len(check_score) > 0:
30                max_eval['score'] = len(check_score)
31                max_eval['move'] = {'row': row, 'col': col, 'value': value}
32                max_eval['moves'] = current_moves.copy()
33
34            if eval['score'] > max_eval['score']:
35                max_eval['score'] = eval['score']
36                max_eval['move'] = {'row': row, 'col': col, 'value': value}
37                max_eval['moves'] = eval['moves']
38
39            alpha = max(alpha, eval['score'])
40
41            current_moves.pop()
42
43            if beta <= alpha:
44                break
45        return max_eval
```

This code implements the minimax algorithm with alpha-beta pruning. The max_eval dictionary contains information about the current state, including the score, the move made, and the list of moves played. The code iterates over all available moves using a for move in self.board.get_available_moves(): loop. For each move, it makes the move on the game board and appends it to the current_moves list. After making the move, it recursively calls the self.__minimax function to evaluate the state at a lower level. The score obtained from

the evaluation is assigned to the eval variable. If the obtained score is 0, indicating a non-scoring move, the player makes a random move, and the score of this move updates eval. The score of the played move is checked using the check_score_for_move function, and if the move results in a score, max_eval is updated. If the eval score is greater than the current max_eval score, max_eval is updated. The alpha value is updated when the highest score is achieved. Alpha-beta pruning is performed when necessary to cut unnecessary branches in the search space.

2. min_eval Part (maximizing_player=false):

This section represents the situation where the player is attempting to minimize. Again, it evaluates each available move, traversing through them within the alpha-beta window to find the best move that minimizes the score.

```
1  else:
2      min_eval = {'score': -sys.maxsize, 'move': None, 'moves': current_moves.copy()}
3      for move in self.board.get_available_moves():
4          row = move['available_move'][0]
5          col = move['available_move'][1]
6          value = move["value"]
7
8          self.board.move(row, col, value)
9          current_moves.append({'move': (row, col), 'value': value, 'player': 'min'})
10         eval = self.__minimax(depth - 1, alpha, beta, True, row, col, value, current_moves)
11
12         if -eval['score'] < -min_eval['score']:
13             min_eval['score'] = eval['score']
14             min_eval['move'] = {'row': row, 'col': col, 'value': value}
15             min_eval['moves'] = eval['moves']
16
17         beta = min(beta, -eval['score'])
18         self.board.undo_move(row, col)
19         current_moves.pop()
20
21         if beta <= alpha:
22             break
23     return min_eval
```

The min_eval dictionary in this code represents the state information for the minimizing player during the alpha-beta pruning process. The loop for move in self.board.get_available_moves(): iterates over all available moves.

For each move, the code makes the move on the game board and appends it to the current_moves list. It then recursively calls the self.__minimax function to evaluate the state at a lower level. The obtained score from the evaluation is assigned to the eval variable. If the negative of the eval score is less than the current min_eval score, min_eval is updated. Beta is updated when the lowest score is achieved. Alpha-beta pruning is performed when necessary to cut unnecessary branches in the search space. The move is then undone on the board, and the current move is removed from the current_moves list to backtrack and explore other possibilities.

5. Game Modes

Firstly, we run the main function.

The program welcomes the user and prompts them to specify the dimensions of the game board.

After creating the board, it displays the menu for game mode selection, where the user can choose to play Human vs AI, AI vs AI, Human vs Human, or exit the game.

```
hon/Python312/python.exe "c:/Users/Utku/Desktop/ai/AI-Projects/Project - 3/main.py"
Welcome to the Game!
Please specify the board's height (1-10): 5
Please specify the board's width (1-10): 5
  1 2 3 4 5
1 S _ _ _ S
2 _ _ _ _ _
3 _ _ _ _ _
4 _ _ _ _ _
5 S _ _ _ S
Please select an option:
1. Human vs AI
2. AI vs AI
3. Human vs Human
4. Exit
Enter your choice (1-4): 
```

1. Human vs AI

In the SOS game, when a person plays against AI (Artificial Intelligence), the AI uses a special plan called "minimax" and a "depth" level. "Minimax" is like a smart guide for the AI to make good moves. "Depth" means how many steps ahead the AI thinks about the game. The more the depth, the smarter the AI plays, but it also takes more time to think. The human player tries to beat the AI by making clever moves and guessing what the AI will do next and allows playing up to 10x10 games.

Human vs Min Ai(7 depth)

<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td></td><td></td><td>S</td></tr><tr><td>2</td><td></td><td>S</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>human</p>	1	2	3	4	5	1	S			S	2		S			3					4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td></td><td></td><td>S</td></tr><tr><td>2</td><td></td><td>S</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>0</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Human</p>	1	2	3	4	5	1	S			S	2		S			3				0	4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td></td><td>0</td><td></td></tr><tr><td>2</td><td>0</td><td>S</td><td>0</td><td></td></tr><tr><td>3</td><td>S</td><td></td><td></td><td>0</td></tr><tr><td>4</td><td></td><td>S</td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Human</p>	1	2	3	4	5	1	S		0		2	0	S	0		3	S			0	4		S			5	S			S
1	2	3	4	5																																																																																								
1	S			S																																																																																								
2		S																																																																																										
3																																																																																												
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S			S																																																																																								
2		S																																																																																										
3				0																																																																																								
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S		0																																																																																									
2	0	S	0																																																																																									
3	S			0																																																																																								
4		S																																																																																										
5	S			S																																																																																								
<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td></td><td></td><td>S</td></tr><tr><td>2</td><td></td><td>S</td><td>0</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>0</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>ai</p>	1	2	3	4	5	1	S			S	2		S	0		3				0	4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td></td><td>0</td><td></td></tr><tr><td>2</td><td></td><td>S</td><td>0</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>0</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Ai</p>	1	2	3	4	5	1	S		0		2		S	0		3				0	4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td></td><td>0</td><td></td></tr><tr><td>2</td><td>0</td><td>S</td><td>0</td><td></td></tr><tr><td>3</td><td>S</td><td></td><td></td><td>0</td></tr><tr><td>4</td><td></td><td>S</td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Ai</p>	1	2	3	4	5	1	S		0		2	0	S	0		3	S			0	4		S			5	S			S
1	2	3	4	5																																																																																								
1	S			S																																																																																								
2		S	0																																																																																									
3				0																																																																																								
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S		0																																																																																									
2		S	0																																																																																									
3				0																																																																																								
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S		0																																																																																									
2	0	S	0																																																																																									
3	S			0																																																																																								
4		S																																																																																										
5	S			S																																																																																								

Human vs Max Ai(7 depth)

<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td>0</td><td>S</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Human</p>	1	2	3	4	5	1	S	0	S		2					3					4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td>0</td><td>S</td><td>0</td></tr><tr><td>2</td><td></td><td></td><td>S</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Human</p>	1	2	3	4	5	1	S	0	S	0	2			S		3					4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td>0</td><td>S</td><td>0</td></tr><tr><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>S</td><td></td><td>S</td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td>0</td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Human</p>	1	2	3	4	5	1	S	0	S	0	2					3	S		S		4				0	5	S			S
1	2	3	4	5																																																																																								
1	S	0	S																																																																																									
2																																																																																												
3																																																																																												
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S	0	S	0																																																																																								
2			S																																																																																									
3																																																																																												
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S	0	S	0																																																																																								
2																																																																																												
3	S		S																																																																																									
4				0																																																																																								
5	S			S																																																																																								
<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td>0</td><td>S</td><td>0</td></tr><tr><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Ai</p>	1	2	3	4	5	1	S	0	S	0	2					3					4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td>0</td><td>S</td><td>0</td></tr><tr><td>2</td><td></td><td></td><td>S</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>0</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Ai</p>	1	2	3	4	5	1	S	0	S	0	2			S		3				0	4					5	S			S	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>S</td><td>0</td><td>S</td><td>0</td></tr><tr><td>2</td><td></td><td>0</td><td></td><td></td></tr><tr><td>3</td><td>S</td><td></td><td>S</td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td>0</td></tr><tr><td>5</td><td>S</td><td></td><td></td><td>S</td></tr></table> <p>Ai</p>	1	2	3	4	5	1	S	0	S	0	2		0			3	S		S		4				0	5	S			S
1	2	3	4	5																																																																																								
1	S	0	S	0																																																																																								
2																																																																																												
3																																																																																												
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S	0	S	0																																																																																								
2			S																																																																																									
3				0																																																																																								
4																																																																																												
5	S			S																																																																																								
1	2	3	4	5																																																																																								
1	S	0	S	0																																																																																								
2		0																																																																																										
3	S		S																																																																																									
4				0																																																																																								
5	S			S																																																																																								

2. AI vs AI

This section details a series of AI vs AI game simulations conducted to compare the efficacy and strategic depth of two artificial intelligence agents in games ranging from 5x5 to 7x7 grids. Each AI was assigned specific 'depth' and 'max/min' settings, which influenced their lookahead in the game's decision tree. Usually, min player was defeated by max. The 'max' player's ability to calculate and evaluate more potential future moves provided a substantial advantage over the 'min' player, leading to a higher frequency of wins for the 'max' player in these simulations.

5 x 5 Game

Please specify the AI-1's depth (1-10): 6	AI-1 (max) player score: 10
Please specify the AI-1's type (max/min): max	AI-2 (min) player score: 6
Please specify the AI-2's depth (1-10): 6	AI-1 (max) player won!!!

Please specify the AI-1's depth (1-10): 7	AI-1 (max) player score: 11
Please specify the AI-1's type (max/min): max	AI-2 (max) player score: 9
Please specify the AI-2's depth (1-10): 5	AI-1 (max) player won!!!

Please specify the AI-1's depth (1-10): 7	AI-1 (min) player score: 3
Please specify the AI-1's type (max/min): min	AI-2 (max) player score: 11
Please specify the AI-2's depth (1-10): 5	AI-2 (max) player won!!!

6 x 6 Game

Please specify the AI-1's depth (1-10): 6	AI-1 (max) player score: 18
Please specify the AI-1's type (max/min): max	AI-2 (min) player score: 6
Please specify the AI-2's depth (1-10): 6	AI-1 (max) player won!!!

Please specify the AI-1's depth (1-10): 7	AI-1 (max) player score: 8
Please specify the AI-1's type (max/min): max	AI-2 (max) player score: 16
Please specify the AI-2's depth (1-10): 5	AI-2 (max) player won!!!

Please specify the AI-1's depth (1-10): 7	AI-1 (min) player score: 3
Please specify the AI-1's type (max/min): min	AI-2 (max) player score: 19
Please specify the AI-2's depth (1-10): 5	AI-2 (max) player won!!!

7 x 7 Game

Please specify the AI-1's depth (1-10): 6	AI-1 (max) player score: 22
Please specify the AI-1's type (max/min): max	AI-2 (min) player score: 11
Please specify the AI-2's depth (1-10): 6	AI-1 (max) player won!!!

Please specify the AI-1's depth (1-10): 7	AI-1 (max) player score: 21
Please specify the AI-1's type (max/min): max	AI-2 (max) player score: 21
Please specify the AI-2's depth (1-10): 5	It's a tie!

Please specify the AI-1's depth (1-10): 7	AI-1 (min) player score: 8
Please specify the AI-1's type (max/min): min	AI-2 (max) player score: 26
Please specify the AI-2's depth (1-10): 5	AI-2 (max) player won!!!
Please specify the AI-2's type (max/min): max	

3. Human vs Human

In the "Human vs Human" game mode, players alternate making moves on the game board by entering the row, column, and value for their moves. The game concludes when the board is full or there are no more valid moves. The final scores of both players are then compared, and the winner is determined based on the higher score. If the scores are equal, the game results in a tie.

```
Human-2's turn:
Please enter the row: 4
Please enter the column: 4
Please enter the value: 0
  1 2 3 4 5
1 S O S O S
2 O O _ O O
3 S _ _ _ S
4 O O _ O O
5 S O S O S
```

```
Human-1's turn:
Please enter the row: 3
Please enter the column: 3
Please enter the value: S
  1 2 3 4 5
1 S O S O S
2 O O _ O O
3 S _ S _ S
4 O O _ O O
5 S O S O S
```

```
Human-2's turn:
Please enter the row: 3
Please enter the column: 2
Please enter the value: 0
  1 2 3 4 5
1 S O S O S
2 O O _ O O
3 S O S _ S
4 O O _ O O
5 S O S O S
```

```
Human-1's turn:
Please enter the row: 3
Please enter the column: 4
Please enter the value: 0
  1 2 3 4 5
1 S O S O S
2 O O _ O O
3 S O S O S
4 O O _ O O
5 S O S O S
```

```
Human-2's turn:
Please enter the row: 2
Please enter the column: 3
Please enter the value: 0
  1 2 3 4 5
1 S O S O S
2 O O O O O
3 S O S O S
4 O O _ O O
5 S O S O S
```

```
Human-1's turn:
Please enter the row: 4
Please enter the column: 3
Please enter the value: 0
  1 2 3 4 5
1 S O S O S
2 O O O O O
3 S O S O S
4 O O O O O
5 S O S O S
```

```
Human-1 player score: 12
Human-2 player score: 8
Human-1 player won!!!
```