DOTS AND SQUARES

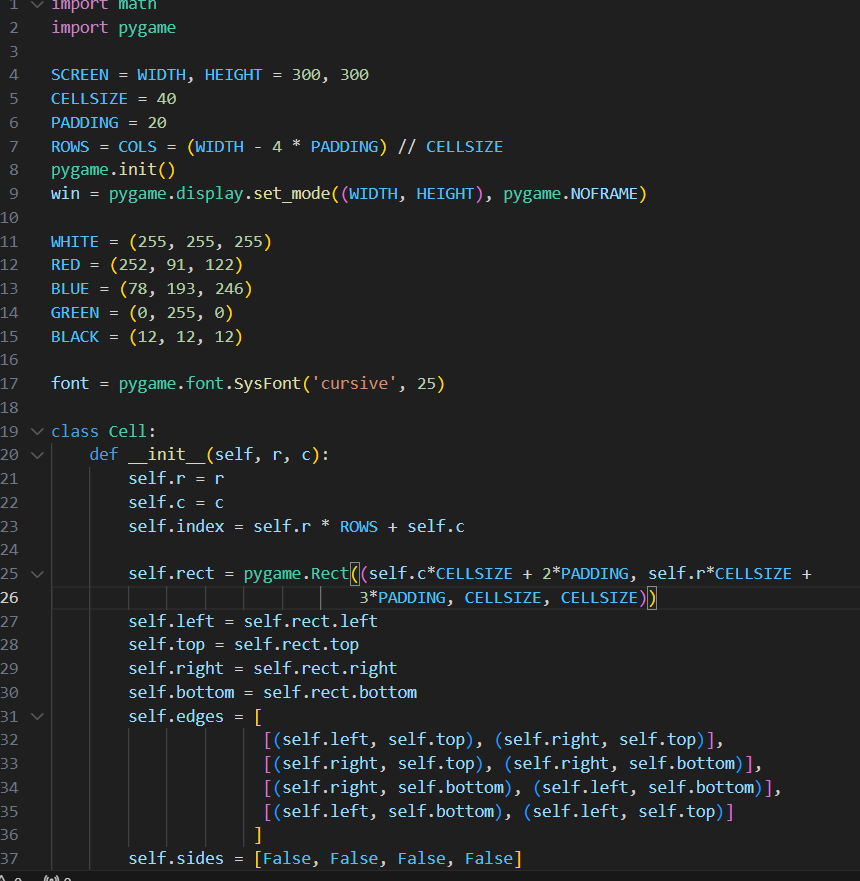
Nihaal Asif

**Introduction:**

Our Group chose to develop “Dots And Boxes” game in python. The basic idea of this game is that it is played by 2 players. There is a grid of NxN cells, each player takes a turn to draw a line between 2 coordinates. The player to complete one cell/box (4 sides each) gets a point. Our implementation consists of one player controlled by the user and the other controlled by AI. We have used the Min-Max algorithm with Alpha beta pruning for the AI controlled player.The game is developed using Pygame library.

**Initialization:**

The initialization section of the code sets up the fundamental parameters and resources necessary for the game. Firstly, it defines constants such as SCREEN, WIDTH, HEIGHT, CELLSIZE, and PADDING, determining the dimensions of the game window, size of each cell in the grid, and spacing between cells. These constants ensure consistency and scalability of the game interface. Following this, Pygame is initialized with pygame.init(), preparing the environment for game development. The window is created using pygame.display.set\_mode(), specifying its dimensions and removing the frame with pygame.NOFRAME to provide a clean interface. Next, color constants and a font object are defined for rendering text and shapes within the game. This initialization process establishes the foundational elements required for the subsequent functioning of the game, laying the groundwork for further development and interaction.



**Helper functions:**

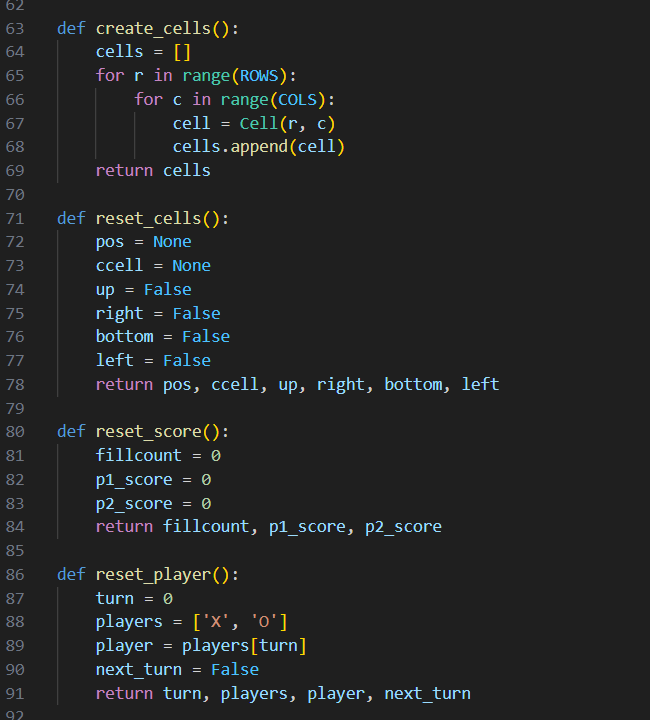
The helper functions in the code serve various purposes to facilitate the management of game state, player actions, and AI decision-making:

1**. create cells():** This function generates and returns a list of Cell objects representing each cell in the game grid. It iterates over the rows and columns, creating a Cell object for each grid position and appending it to the list.

2. **reset\_cells():** This function resets the state of all cells in the game grid. It initializes variables such as pos (mouse position), ccell (current cell), and boolean flags for arrow key presses (up, right, bottom, left) to None or False, indicating that there is no active selection or input.

3. r**eset\_score():** This function resets the game scores and the count of filled cells. It initializes variables such as fillcount (count of filled cells), p1\_score (score of Player 1), and p2\_score (score of Player 2) to zero.

4. **reset\_player():** This function resets the player-related variables. It sets the initial player turn to 0 (Player 1), initializes a list of player symbols ('X' and 'O'), and sets the current player to the symbol at the initial turn index. Additionally, it sets the next\_turn flag to False, indicating that it's not yet the next player's turn.



**Logical functions:**

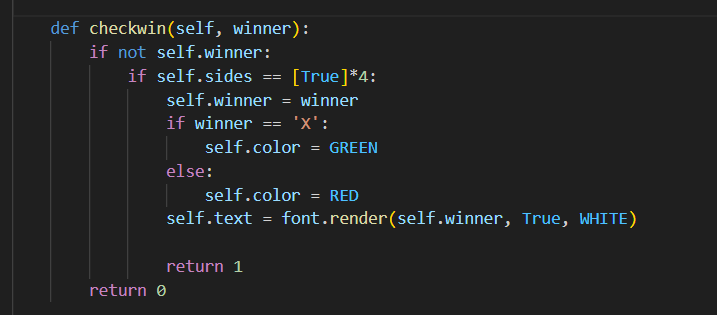
1. **Checkwin function():**This method is a part of the `Cell` class and is responsible for checking if a player has won after making their move. It takes the parameter `winner`, which represents the symbol ('X' or 'O') of the player who made the last move. Here's what it does:

- It first checks if there's no winner already (`self.winner` is `None`).

- Then, it checks if all four sides of the cell are marked (`self.sides == [True]\*4`). If they are, it means the cell is completely enclosed, and the current player has won.

- If the player has won, it sets the `winner` attribute of the cell to the symbol of the winning player (`winner`), changes the color of the cell accordingly (green for 'X' and red for 'O'), and renders the winning symbol text onto the cell surface.

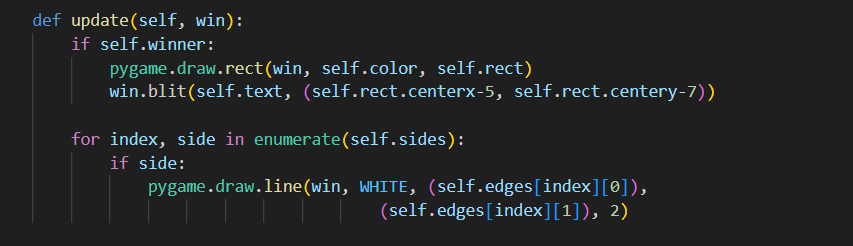
- Finally, it returns 1 if the player has won (indicating that the game should increment the score and check for game over), and 0 otherwise.



1. **Update function():** This method is also a part of the `Cell` class and is responsible for updating the appearance of the cell on the game window. Here's what it does:

- If there's a winner (`self.winner` is not `None`), it draws a colored rectangle representing the cell (green or red based on the winner) and renders the winning symbol text onto the cell surface.

- It then iterates over each side of the cell (`self.sides`) and draws a line between the corresponding edge points (`self.edges`). If a side is marked (`True`), it draws a white line; otherwise, it skips drawing the line, leaving the cell boundary open.



**Min-Max Implementation:**

The `minimax` function implements the minimax algorithm, a recursive algorithm used for decision-making in two-player, zero-sum games like Tic-Tac-Toe. Let's break down the function step by step:

**Parameters:**

**- cells:** A list of Cell objects representing the current state of the game grid.

- **depth**: An integer representing the depth of the search tree. It determines how many moves ahead the algorithm should consider.

**- alpha, beta:** Alpha and beta are used for alpha-beta pruning, an optimization technique to reduce the number of nodes evaluated by the minimax algorithm.

**- maximizingPlayer:** A boolean flag indicating whether the current player is maximizing ('X') or minimizing ('O').

**Algorithm:**

**1. Base Case:**

If the `depth` is 0 or the game is over, return None (no move) and the evaluation of the current game state using the `evaluate` function.

**2. Maximizing Player's Turn:**

If it's the maximizing player's turn, the algorithm maximizes the score. It initializes `maxEval` to negative infinity and `bestMove` to None.

* 1. It iterates over each cell in the grid. For each cell, it checks if making a move (marking a side) is valid.
  2. If it is a valid move, it recursively calls the `minimax` function with a reduced depth and updates `maxEval` and `bestMove` if a better move is found.
  3. Alpha-beta pruning is applied to reduce unnecessary exploration of nodes.

**3**. **Minimizing Player's Turn:**

3.1. If it's the minimizing player's turn, the algorithm minimizes the score. It initializes `minEval` to positive infinity and `bestMove` to None.

3.2. It follows a similar process as the maximizing player's turn, but this time it minimizes the score.

**4. Returning Best Move:**

After exploring all possible moves, the function returns the `bestMove` and the corresponding evaluation (`maxEval` for the maximizing player, `minEval` for the minimizing player).



**Other:**

**Evaluation Function:**

The `evaluate` function is a simple heuristic evaluation function used to evaluate the game state. It calculates the difference between the scores of Player 1 ('X') and Player 2 ('O') based on the number of cells each player has won.

**Graphical interface:**

**1. Game Window:**

Initialized using Pygame's `set\_mode()` function, providing the main display area for the game.

**2. Grid Drawing:**

Horizontal and vertical lines drawn at regular intervals to create the grid layout using Pygame's `draw.line()` function.

**3. Cell Rendering:**

Each cell in the grid is represented by a `Cell` object, which encapsulates its position, state, and appearance. Pygame's drawing functions (`draw.rect()`, `draw.line()`, `blit()`) are used to render shapes, lines, and text onto the game window surface.

**4. Player Interaction:**

Keyboard input is used for the human player ('X'), with arrow keys indicating the sides to be marked on the current cell. The GUI visually reflects the player's actions by updating the appearance of the marked sides.

**5. AI Player Interaction:**

For the AI player ('O'), the minimax algorithm is used to determine the optimal move. After calculating the best move, the GUI updates the state of the game grid accordingly.

**6. Score Display:**

Elements for displaying the scores of each player are included in the GUI. Text rendering functions are used to display the scores on the game window surface.

**7. Game Over Message:**

When the game ends, a game over message is displayed on the GUI along with the winner. Text rendering functions are used to display the message on the game window surface.

These GUI components work together to create an interactive and visually appealing interface for the game, enhancing the overall gameplay experience for users.