Othello Game

Project Overview

This project is an implementation of the Othello (also known as Reversi) game using Python's `tkinter` library for the graphical user interface (GUI) and `pygame` for background music. The game includes features for single-player mode against an AI, two-player mode, options to toggle sound, and instructions on how to play the game.

Step-by-Step Breakdown

1. Initialization and Setup

Libraries Imported

- **tkinter**: For creating the GUI.
- **messagebox**: For displaying dialog boxes.
- **PIL (Pillow)**: For image handling and manipulation.
- **copy**: For copying objects.
- **math**: For mathematical operations.
- **pygame**: For playing background music.

Class Definition

- **OthelloGame**: The main class that contains all methods and attributes necessary to run the game.

Root Window Setup

- **self.root**: Initializes the main window of the application and sets the title to "Othello".

Board Initialization

- **self.board**: An 8x8 list of lists that represents the game board, initialized with empty strings.

Current Player

- **self.current_player**: A string that keeps track of whose turn it is, starting with 'black'.

Splash Screen

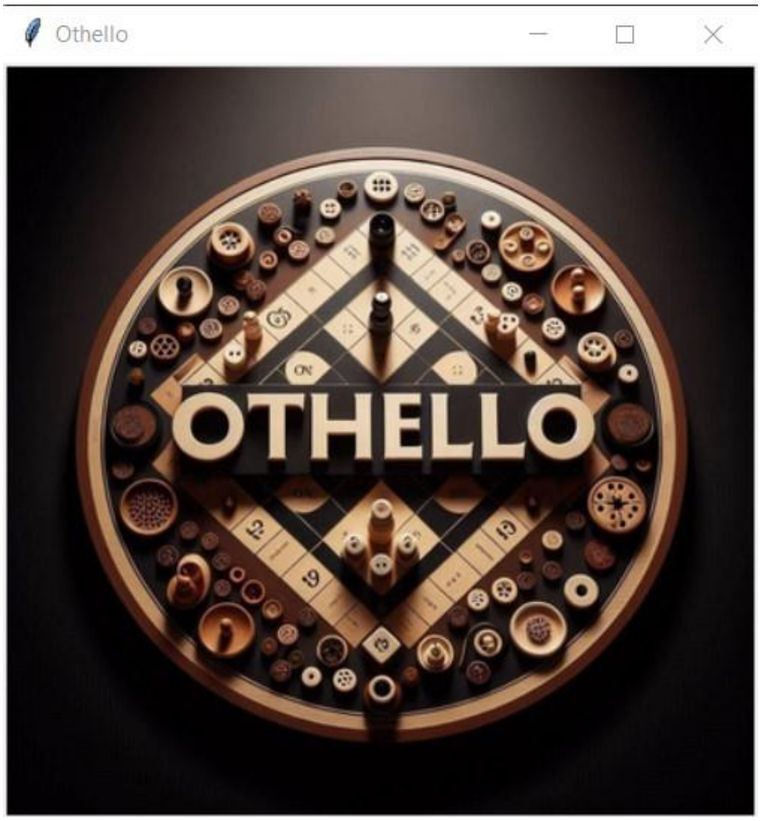
- **self.show_splash_screen()**: Displays the initial splash screen.

Pygame Mixer Initialization

Initializes pygame's mixer to handle background music.

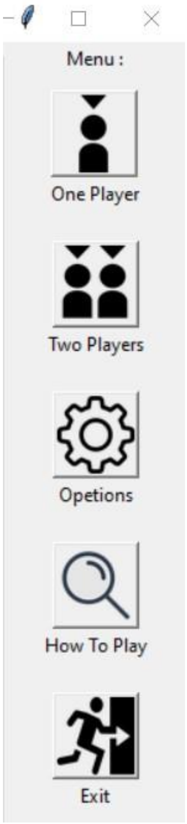
2. Splash Screen and Main Menu

Splash Screen



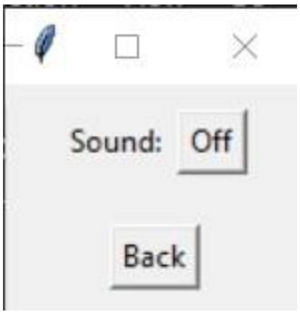
- **show_splash_screen()**: Displays an image as the splash screen and transitions to the main menu when clicked.

Main Menu



- **create_start_screen()**: Displays the main menu with options for one player, two players, settings, how to play, and exit.

3. Options Mode



Sound Toggle

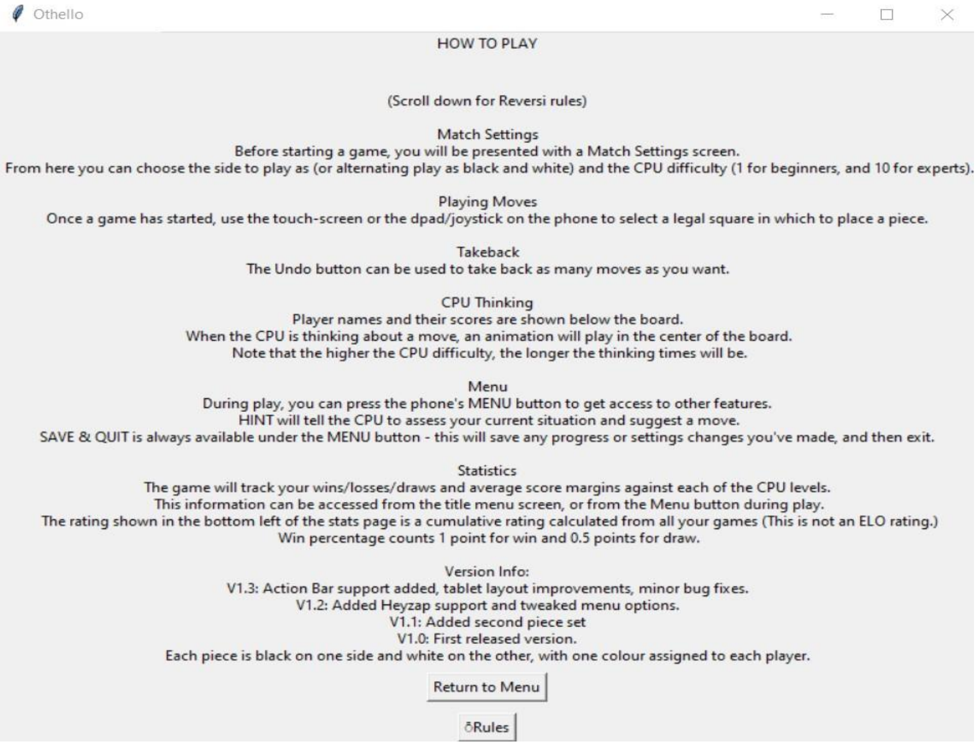
- **opetions_mode()**: Displays the options menu where the user can toggle the background music.
- **toggle_sound()**: Toggles the music on and off by pausing and unpausing the pygame mixer.

Back Button

Returns to the main menu from the options screen.

4. How to Play Mode

Instructions



HOW TO PLAY

(Scroll down for Reversi rules)

Match Settings
Before starting a game, you will be presented with a Match Settings screen.
From here you can choose the side to play as (or alternating play as black and white) and the CPU difficulty (1 for beginners, and 10 for experts).

Playing Moves
Once a game has started, use the touch-screen or the dpad/joystick on the phone to select a legal square in which to place a piece.

Takeback
The Undo button can be used to take back as many moves as you want.

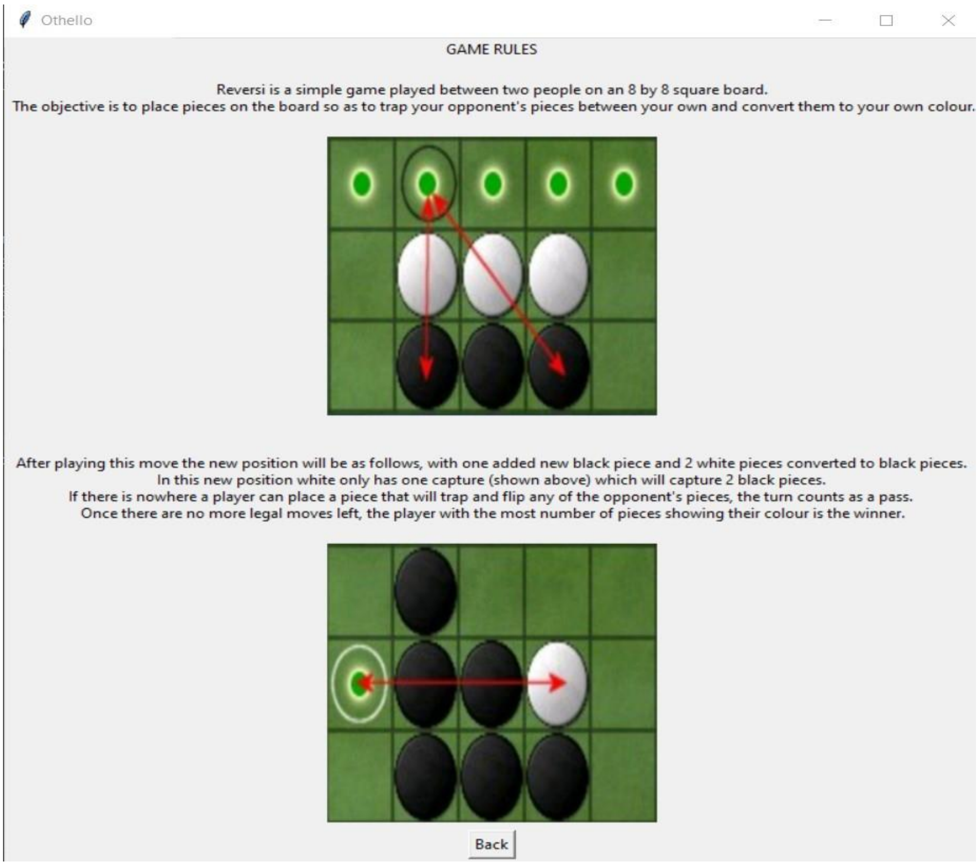
CPU Thinking
Player names and their scores are shown below the board.
When the CPU is thinking about a move, an animation will play in the center of the board.
Note that the higher the CPU difficulty, the longer the thinking times will be.

Menu
During play, you can press the phone's MENU button to get access to other features.
HINT will tell the CPU to assess your current situation and suggest a move.
SAVE & QUIT is always available under the MENU button - this will save any progress or settings changes you've made, and then exit.

Statistics
The game will track your wins/losses/draws and average score margins against each of the CPU levels.
This information can be accessed from the title menu screen, or from the Menu button during play.
The rating shown in the bottom left of the stats page is a cumulative rating calculated from all your games (This is not an ELO rating.)
Win percentage counts 1 point for win and 0.5 points for draw.

Version Info:
V1.3: Action Bar support added, tablet layout improvements, minor bug fixes.
V1.2: Added Heyzap support and tweaked menu options.
V1.1: Added second piece set
V1.0: First released version.
Each piece is black on one side and white on the other, with one colour assigned to each player.

- **how_to_play_mode()**: Displays detailed instructions on how to play the game.



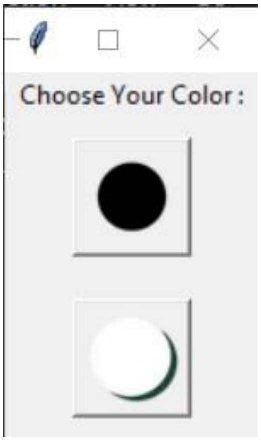
GAME RULES

Reversi is a simple game played between two people on an 8 by 8 square board.
The objective is to place pieces on the board so as to trap your opponent's pieces between your own and convert them to your own colour.

After playing this move the new position will be as follows, with one added new black piece and 2 white pieces converted to black pieces.
In this new position white only has one capture (shown above) which will capture 2 black pieces.
If there is nowhere a player can place a piece that will trap and flip any of the opponent's pieces, the turn counts as a pass.
Once there are no more legal moves left, the player with the most number of pieces showing their colour is the winner.

- **rules_mode()**: Further explains the rules of Othello with text and images.

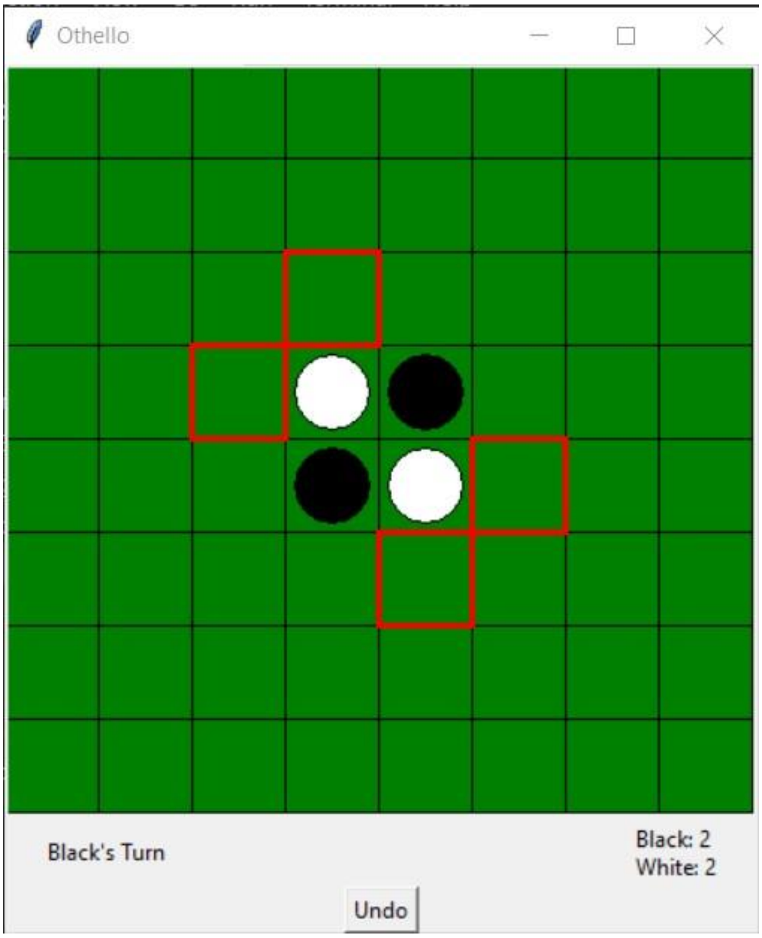
5. Color Choice Screen



Color Selection

- **show_color_choice_screen()**: Allows the user to choose their color (black or white) for single-player mode.

6. Game Initialization



Black's Turn

Black: 2
White: 2

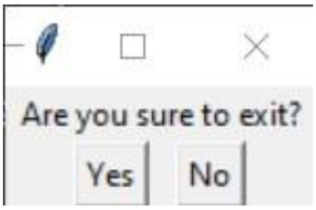
Board Setup

- **start_game(color)**: Sets up the initial positions of the pieces on the board.

Game Screen

- **show_game_screen()**: Displays the game board and updates the status labels to show the current player's turn.

7. Exit Mode



Are you sure to exit?

Confirmation Dialog

- **exit_mode()**: Asks the user for confirmation before exiting the game.

Explanation of the Main Algorithm (Minimax Algorithm)

Overview

The Minimax algorithm is a recursive algorithm used for decision-making and game theory. It provides an optimal move for the player assuming that the opponent also plays optimally. The algorithm is commonly used in two-player zero-sum games like Othello, Chess, and Tic-Tac-Toe.

Minimax Algorithm Steps

1. **Generate the Game Tree**:
    - The game tree represents all possible moves from the current state of the game.
2. **Assign Scores to Terminal States**:
    - Evaluate the terminal states (end of the game) and assign scores. For Othello, this might be the difference in the number of pieces between the two players.
3. **Propagate Scores Up the Tree**:
    - Use the scores of the terminal states to evaluate the non-terminal states. The scores are propagated up the tree to determine the best move.
4. **Maximizing and Minimizing**:
    - The algorithm alternates between maximizing and minimizing. The maximizing player tries to get the highest score, while the minimizing player tries to get the lowest score.

```
function minimax(board, depth, is_maximizing) {
    if (depth == 0 || game_over(board)) {
        return evaluate_board(board);
    }

    if (is_maximizing) {
        let max_eval = -Infinity;
        for (let move of get_possible_moves(board, 'black')) {
            let new_board = make_move(board, move, 'black');
            let eval = minimax(new_board, depth - 1, false);
            max_eval = Math.max(max_eval, eval);
        }
        return max_eval;
    } else {
        let min_eval = Infinity;
        for (let move of get_possible_moves(board, 'white')) {
            let new_board = make_move(board, move, 'white');
            let eval = minimax(new_board, depth - 1, true);
            min_eval = Math.min(min_eval, eval);
        }
        return min_eval;
    }
}
```

Alpha-Beta Pruning

To optimize the performance of the Minimax algorithm, Alpha-Beta pruning is used. It reduces the number of nodes evaluated in the search tree by pruning branches that cannot affect the final decision.

```
function minimax_alpha_beta(board, depth, alpha, beta, is_maximizing) {
    if (depth == 0 || game_over(board)) {
        return evaluate_board(board);
    }

    if (is_maximizing) {
        let max_eval = -Infinity;
        for (let move of get_possible_moves(board, 'black')) {
            let new_board = make_move(board, move, 'black');
            let eval = minimax_alpha_beta(new_board, depth - 1, alpha, beta, false);
            max_eval = Math.max(max_eval, eval);
            alpha = Math.max(alpha, eval);
            if (beta <= alpha) {
                break;
            }
        }
        return max_eval;
    } else {
        let min_eval = Infinity;
        for (let move of get_possible_moves(board, 'white')) {
            let new_board = make_move(board, move, 'white');
            let eval = minimax_alpha_beta(new_board, depth - 1, alpha, beta, true);
            min_eval = Math.min(min_eval, eval);
            beta = Math.min(beta, eval);
            if (beta <= alpha) {
                break;
            }
        }
        return min_eval;
    }
}
```

Functions Used in Minimax

- **evaluate_board(board)**: Evaluates the board and returns a score based on the current state.
- **get_possible_moves(board, player)**: Returns a list of all possible moves for the given player.
- **make_move(board, move, player)**: Makes the move on the board and returns the new board state.
- **game_over(board)**: Checks if the game is over.

Rules of Othello

1. **Starting Position**: The game starts with four pieces placed in the center of the board in a specific pattern: two black and two white pieces diagonal to each other.
2. **Objective**: The goal is to have the majority of pieces turned to display your color by the end of the game.
3. **Making Moves**: Players take turns placing a piece of their color on the board. A valid move must sandwich one or more of the opponent's pieces between the placed piece and another piece of the same color.
4. **Flipping Pieces**: All sandwiched opponent pieces are flipped to the current player's color.
5. **Passing Turns**: If a player cannot make a valid move, they pass their turn.
6. **End of Game**: The game ends when neither player can make a valid move. The player with the most pieces of their color on the board wins.

Conclusion

This Othello game project provides a comprehensive implementation of the classic board game with a graphical interface and background music. The game includes both single-player and two-player modes, as well as detailed instructions and game rules for new players. The use of `tkinter` for the GUI and `pygame` for music integration demonstrates a blend of different Python libraries to create an engaging user experience.

Anna Nami

StudentNumber:40032143