Implementation and Design of an Othello Game Variant

Silent0Wings

August 15, 2024

Abstract

This report details the implementation and design of a variant of the Othello game. The project involves creating a playable game using object-oriented principles, adhering to the provided UML diagrams, and modifying the design where necessary. The game is implemented with a focus on flexibility, enabling various starting positions and rules adjustments. This report outlines the structure of the code, the functionality of each component, and the logic behind the game mechanics.

Contents

Chapter 1

Introduction

The objective of this project is to implement a variant of the Othello game following the guidelines provided. The implementation emphasizes object-oriented programming concepts, particularly the use of classes and polymorphism. The game features a customizable starting position, unplayable squares, and ASCII-based board rendering.

Chapter 2

Project Overview

2.1 Core Components

The project consists of several core components, each represented by a class in the implementation:

- **Piece:** Represents the individual pieces on the Othello board, including white, black, and empty positions.
- **Board:** Manages the game board, including initialization, legal moves, and display functions.
- **Player:** Handles player attributes such as name and assigned symbol (white or black).
- **Game:** Controls the overall game flow, including turn management, game state saving/loading, and win conditions.
- **Position:** Represents the positions on the board, determining if they are playable or unplayable.
- **UnplayablePosition:** A derived class from Position, representing squares where pieces cannot be placed.

Chapter 3

Design and Implementation

3.1 Class Structure and UML

The initial design was based on the provided UML diagrams, with modifications made to enhance functionality and meet the specific requirements of the variant Othello game. The key classes and their interactions are detailed below.

3.1.1 Piece Class

The `Piece` class represents the different states of a board position, including methods to set or switch between white, black, and empty.

```
class Piece {
public:
    char Current;
    const char Black = 'B';
    const char White = 'W';
    const char Empty = '-';

    Piece();
    Piece(char C);
    void Set_Empty();
    void Set_White();
    void Set_Black();
    void Switch();
    bool Is_Empty();
    bool Is_White();
    bool Is_Black();
    bool Equal(Piece* temp);
    char* Get_Current();
    char Get_Symbole();
};
```

3.1.2 Board Class

The `Board` class is responsible for managing the state of the game board, including legal move determination, board initialization, and rendering the current state.

```
class Board {
public:
    size_t Board_Size;
    Piece* The_Board[8][8];
    std::string Board_Frame;
    char Current_Turn;

    Board();
    Board(int arr[8][8]);
    void Reset_Board();
    void Initialise_Board();
    void Initialise_Board_Legal_Moves();
    void Initialise_Board_Frame();
    bool Board_Is_Empty();
    int Number_Blacks();
    int Number_Whites();
    bool Full_Board();
    bool MakeMove(int X, int Y);
    void Draw_Board();
};
```

3.1.3 Game Class

The `Game` class manages the game loop, player turns, and game state. It interacts with the `Board` and `Player` classes to control the game's progression.

```
class Game {
public:
    bool Draw;
    bool Forfeit;
    bool Running;
    GameLogic Current_Game_Logic;

    Game();
    Game(std::string temp_path);
    bool Save_Game(std::string temp_path);
    static bool Load_Game(std::string temp_path);
    void Play();
    void Handle_Game_Type();
    void Process_End();
    void Set_Start_Configuration();
```

```
    void Quit();
    void Print_Commands();
};
```

3.2 Game Logic and Flow

The game follows a traditional Othello structure with modifications to starting positions and unplayable squares. The main game loop is handled by the `Game::Play()` method, which manages player input, move validation, and win conditions.

3.2.1 Handling Player Turns

Players alternate turns, with the game checking for valid moves and updating the board accordingly. If no valid moves are available, the player may forfeit their turn or save the game.

Chapter 4

File Management

The game includes functionality for saving and loading the current game state. The state is saved to a text file, with the board layout and player turns recorded for later retrieval.

Chapter 5

Conclusion

This project demonstrates the application of object-oriented programming principles to implement a variant of the Othello game. The final product is a flexible, text-based game that can be customized and extended further. Future improvements could include a graphical user interface or networked multiplayer support.