Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

VISUAL PROGRAMMING

Project Documentation

Authors:

Eroll Sakipi(211590)

Drin Kadriu(211505)

Mentors:

PhD Dejan Gjorgjevikj

MSc Stefan Andonov

Contents

Abstract

This project is a .NET-based chess game that allows two players to play a game of chess with all the rules of modern chess. The project is divided into two main parts:

1. **Backend**: Implements the entire game logic.
2. **Frontend**: User interface built as a WPF (Windows Presentation Foundation) application.

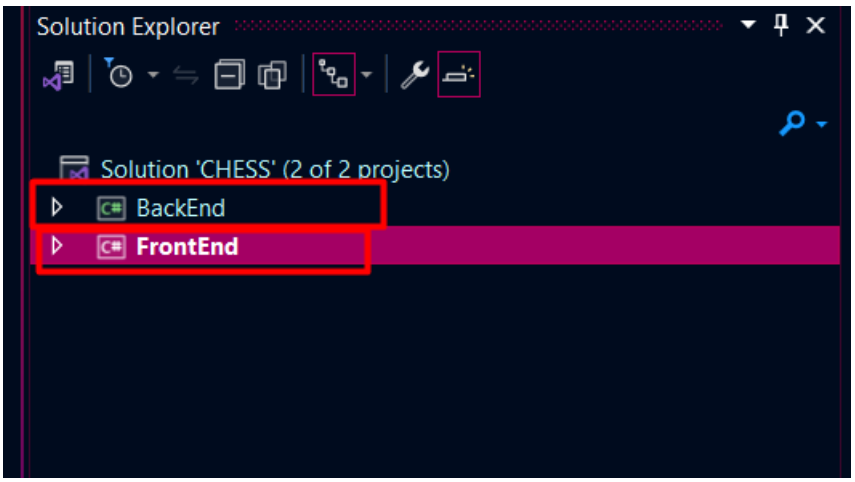

Figure 1.Project Structure

Backend

The backend contains the core logic for the chess game, including move validation, piece movements, game state management, and rule enforcement. It is organized into several folders and files as follows:

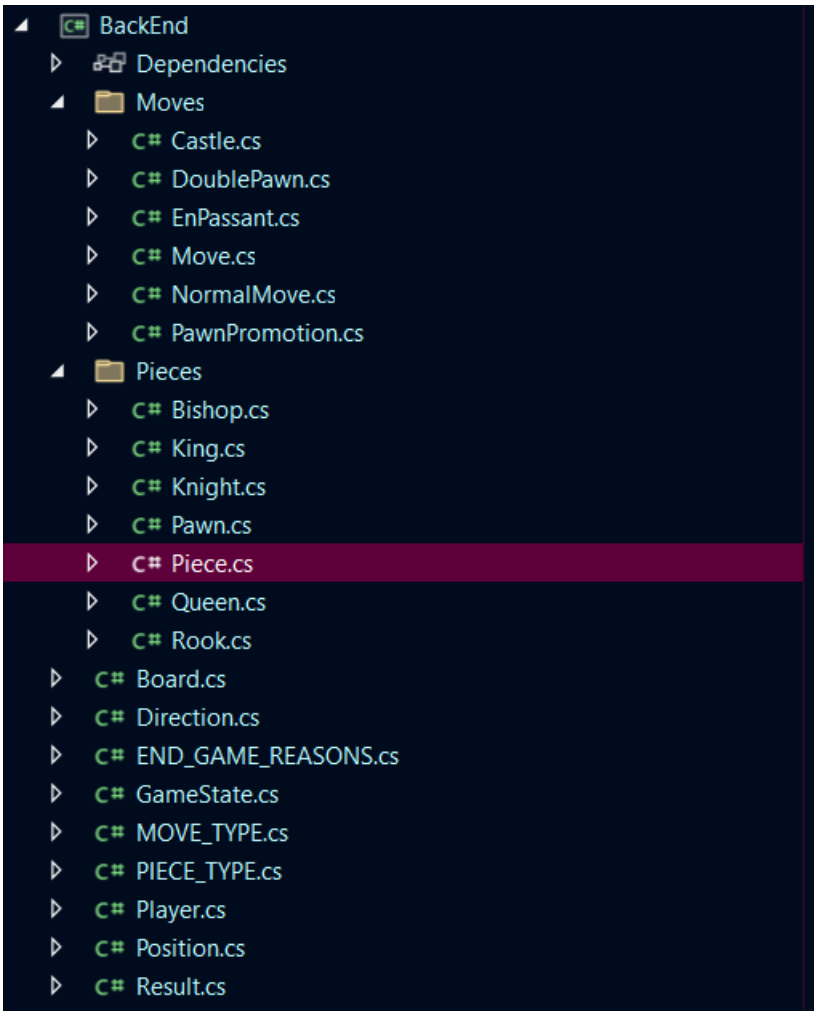

Figure 2.Backend Structure

Moves

- **Move.cs**: The base class for all move types.

```
namespace Backend
{
    31 references
    public abstract class Move
    {
        7 references
        public abstract MOVE_TYPE MoveType { get; }

        21 references
        public abstract Position fromPosition { get; }
        19 references
        public abstract Position toPosition { get; }


        11 references
        public abstract void Execute(Board board);

        5 references
        public virtual bool isLegal(Board board)
        {
            //if executing this move doesnt leave  king in check
            Player player = board[fromPosition].Color;

            Board copied=board.Copy();
            Execute(copied);
            return !copied.IsInCheck(player);
        }
    }
}
```

Figure 3.Move Base Class

- **Castle.cs**: Handles the logic for castling moves.
- **DoublePawn.cs**: Implements logic for double pawn moves.
- **EnPassant.cs**: Contains logic for the En Passant move.
- **NormalMove.cs**: Implements the logic for regular moves.
- **PawnPromotion.cs**: Handles pawn promotion logic.

Pieces

- **Piece.cs**: Base class for all chess pieces.

```
namespace Backend
{
    31 references
    public abstract class Piece
    {

        11 references
        public abstract PIECE_TYPE type { get; }
        34 references
        public abstract Player Color { get; }
        18 references
        public bool hasMoved { get; set; } = false;


        7 references
        public abstract Piece Copy();


        9 references
        public abstract IEnumerable<Move> getAllMoves(Position from, Board board);

        //find all reachable positions on given direction

        1 reference
        protected IEnumerable<Position> MovePosInDir(Position from, Board board, Direction dir)
        {
            for (Position pos = from + dir; Board.isInside(pos); pos += dir)
            {

                if (board.isEmpty(pos))
                {
                    yield return pos;
                    continue;
                }

                Piece piece = board[pos];
                if (piece.Color != Color)
                {
                    yield return pos;

                }
                yield break;

            }
        }


        3 references
        protected IEnumerable<Position> MovePosInDirs(Position from, Board board, Direction[] dirs)
        {
            return dirs.SelectMany(dir => MovePosInDir(from, board, dir));
        }

        3 references
        public virtual bool CanCaptureOpponentKing(Position from ,Board board) //detect check in a moment
        {
            return getAllMoves(from, board).Any(move =>
            {
                Piece piece = board[move.toPosition];
                return piece != null && piece.type == PIECE_TYPE.King;
            });

        }
    }
}
```

Figure 4.Piece Base Class

- **Bishop.cs**: Defines the behavior of the Bishop piece.
- **King.cs**: Defines the behavior of the King piece.
- **Knight.cs**: Defines the behavior of the Knight piece.
- **Pawn.cs**: Defines the behavior of the Pawn piece.
- **Queen.cs**: Defines the behavior of the Queen piece.
- **Rook.cs**: Defines the behavior of the Rook piece.

Game Logic

- **Board.cs**: Represents the chess board and manages piece placement.

  - ✓ *introw,intcol* -> Gets or sets the piece at the specified board coordinates.
  - ✓ *getPlayerSkipPosition(Player player)*-> Returns the skip position for the specified player.
  - ✓ *setPlayerSkipPosition(Player player, Position position)*-> Sets the skip position for the specified player.
  - ✓ *Positionposition:* Gets or sets the piece at the specified board position.
  - ✓ *Init()*-> Initializes and returns a new chess board with pieces.
  - ✓ *addPieces()*-> Adds the initial set of pieces to the board.
  - ✓ *isInside(Position pos)*->Checks if a position is within the board boundaries.
  - ✓ *isEmpty(Position pos)*->Checks if a specified position on the board is empty.
  - ✓ *NonEmptyPositions()*-> Returns an enumerable of all non-empty positions on the board.
  - ✓ *NonEmptyPositionsFor(Player player)*->Returns an enumerable of all non-empty positions for the specified player.
  - ✓ *IsInCheck(Player player)*->Checks if the specified player is in check.
  - ✓ *Copy()*->Creates and returns a copy of the board.

- **Direction.cs**: Utility class for defining movement directions.
- **END_GAME_REASONS.cs**: Enumerates possible reasons for game termination.
- **GameState.cs**: Maintains the current state of the game.

  - ✓ *GameState(Board board, Player currentPlayer)*-> Initializes the game state with the given board and current player.
  - ✓ *LegalMovesForPiece(Position position)*-> Returns legal moves for a piece at the specified position.
  - ✓ *makeMove(Move move)*-> Executes the given move and updates the game state.
  - ✓ *AllLegalMovesForPlayer(Player player)*->Returns all legal moves for the specified player.
  - ✓ *CheckForGameOver()*-> Checks and updates the game state if the game is over.
  - ✓ *isGameOver()*-> Returns true if the game is over, otherwise false.

- **MOVE_TYPE.cs**: Enumerates different types of moves.
- **PIECE_TYPE.cs**: Enumerates different types of chess pieces.
- **Player.cs**: Represents a player in the game.
- **Position.cs**: Utility class for managing positions on the board.
- **Result.cs**: Stores the result of a game.

Frontend

The frontend is built using WPF, providing a graphical interface for the players. It includes various XAML files and their corresponding C# code-behind files to manage the UI.

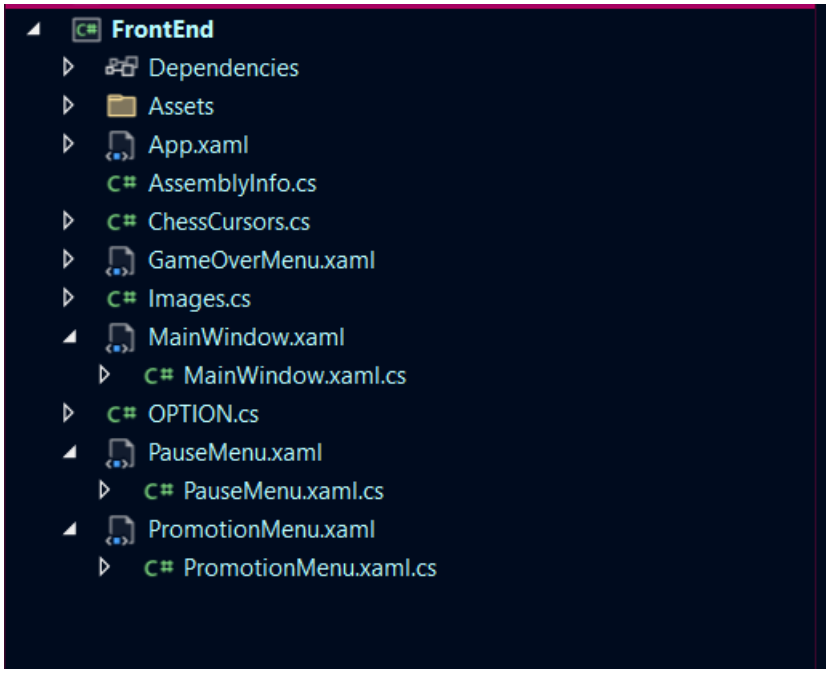

Figure 5.UI-Structure

User Interface Components

- **Assets**: Folder containing game assets such as images and icons.
- **App.xaml & App.xaml.cs**: Entry point of the WPF application.
- **AssemblyInfo.cs**: Contains assembly-level attributes.
- **ChessCursors.cs**: Manages custom cursors for the game.

- **GameOverMenu.xaml & GameOverMenu.xaml.cs**: UI for the game-over screen.

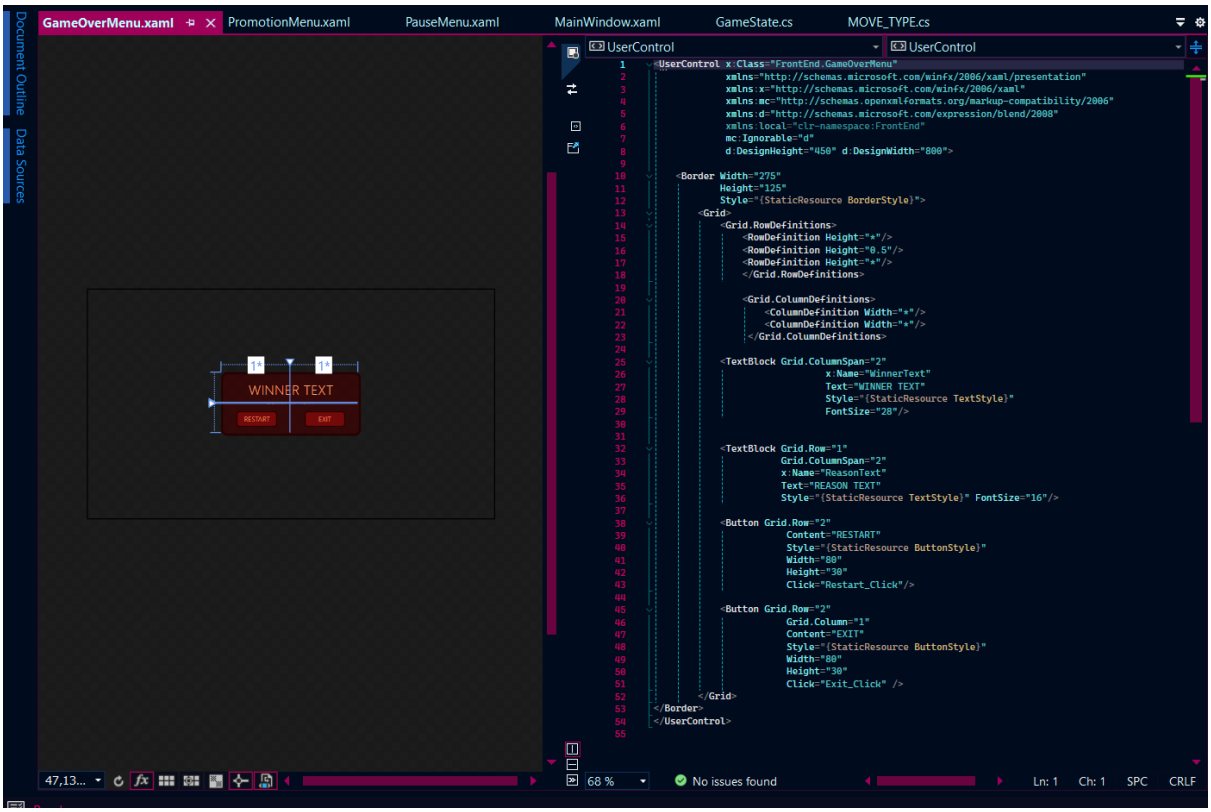

Figure 6.GameOverMenu

- **MainWindow.xaml & MainWindow.xaml.cs**: The main window of the application where the game is played.

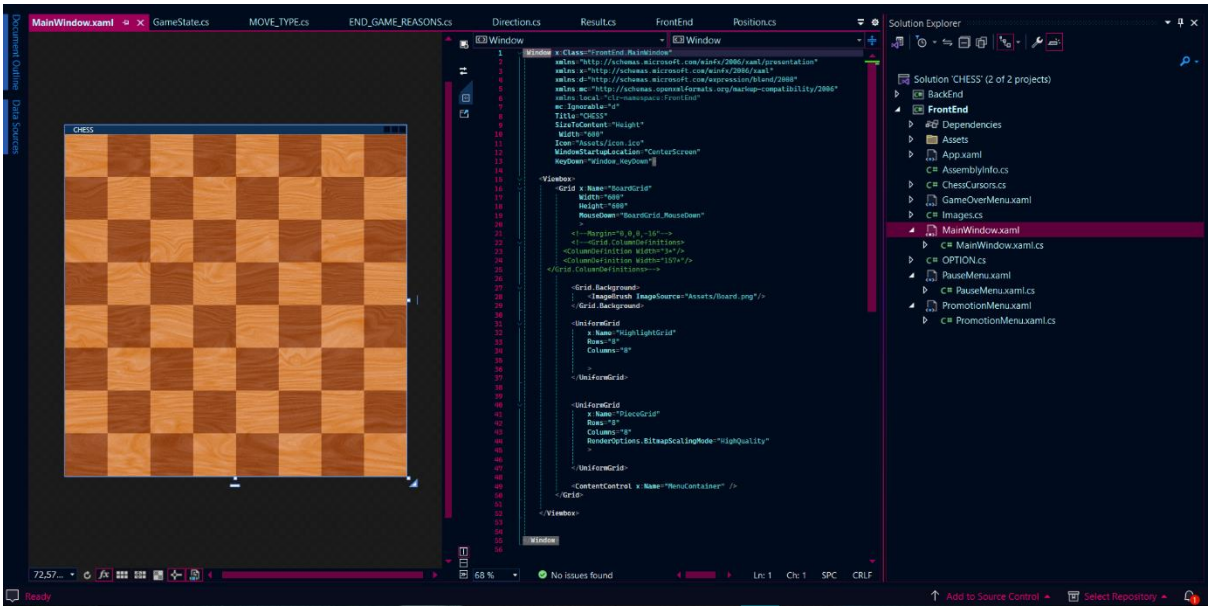

Figure 7.MainWindow-UserControl

- **OPTION.cs**: Manages game options.
- **PauseMenu.xaml & PauseMenu.xaml.cs**: UI for the pause menu.

*PauseMenu is invoked by pressing **ESC key(Escape)**

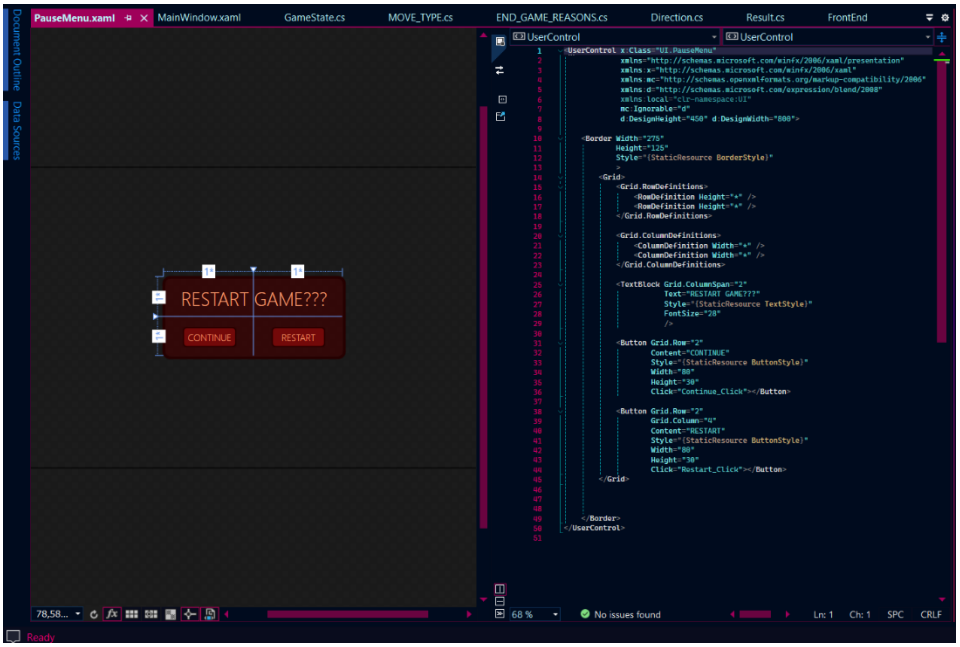

Figure 8.PauseMenu

- **PromotionMenu.xaml & PromotionMenu.xaml.cs**: UI for pawn promotion.

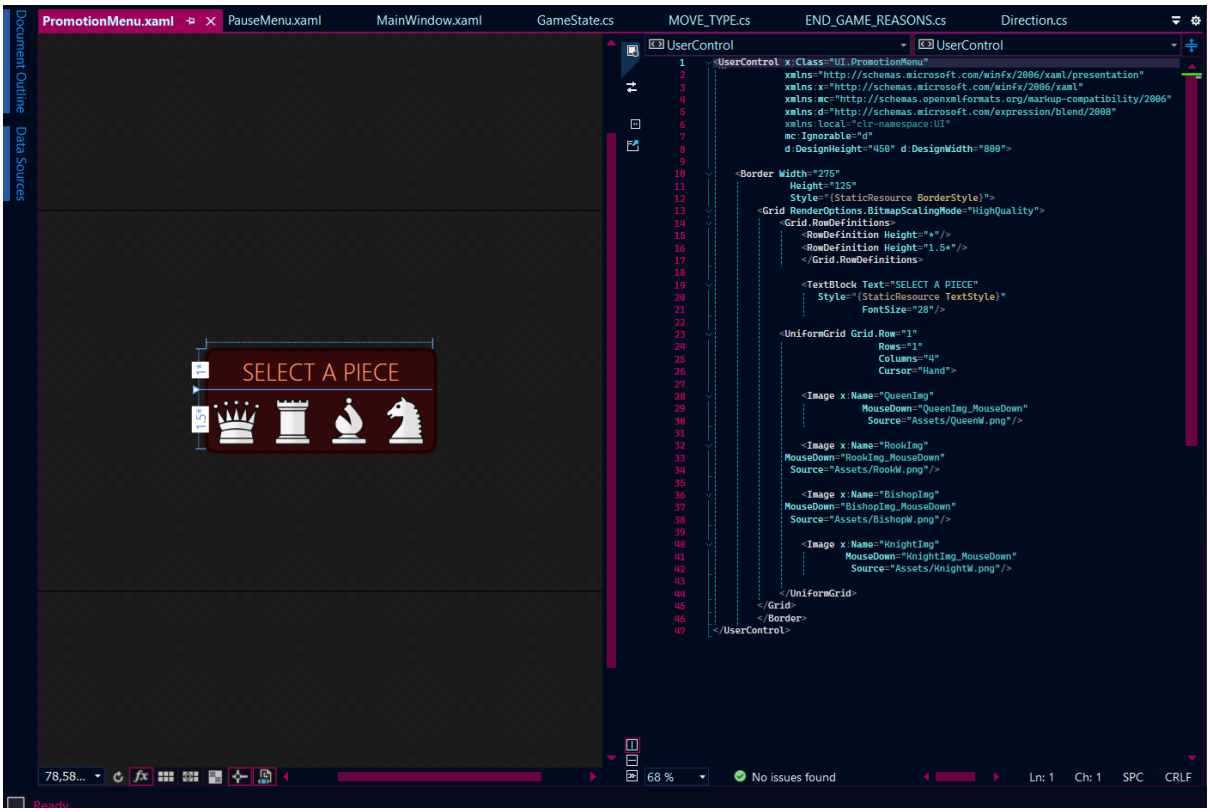

Figure 9.PromotionMenu

List of all images