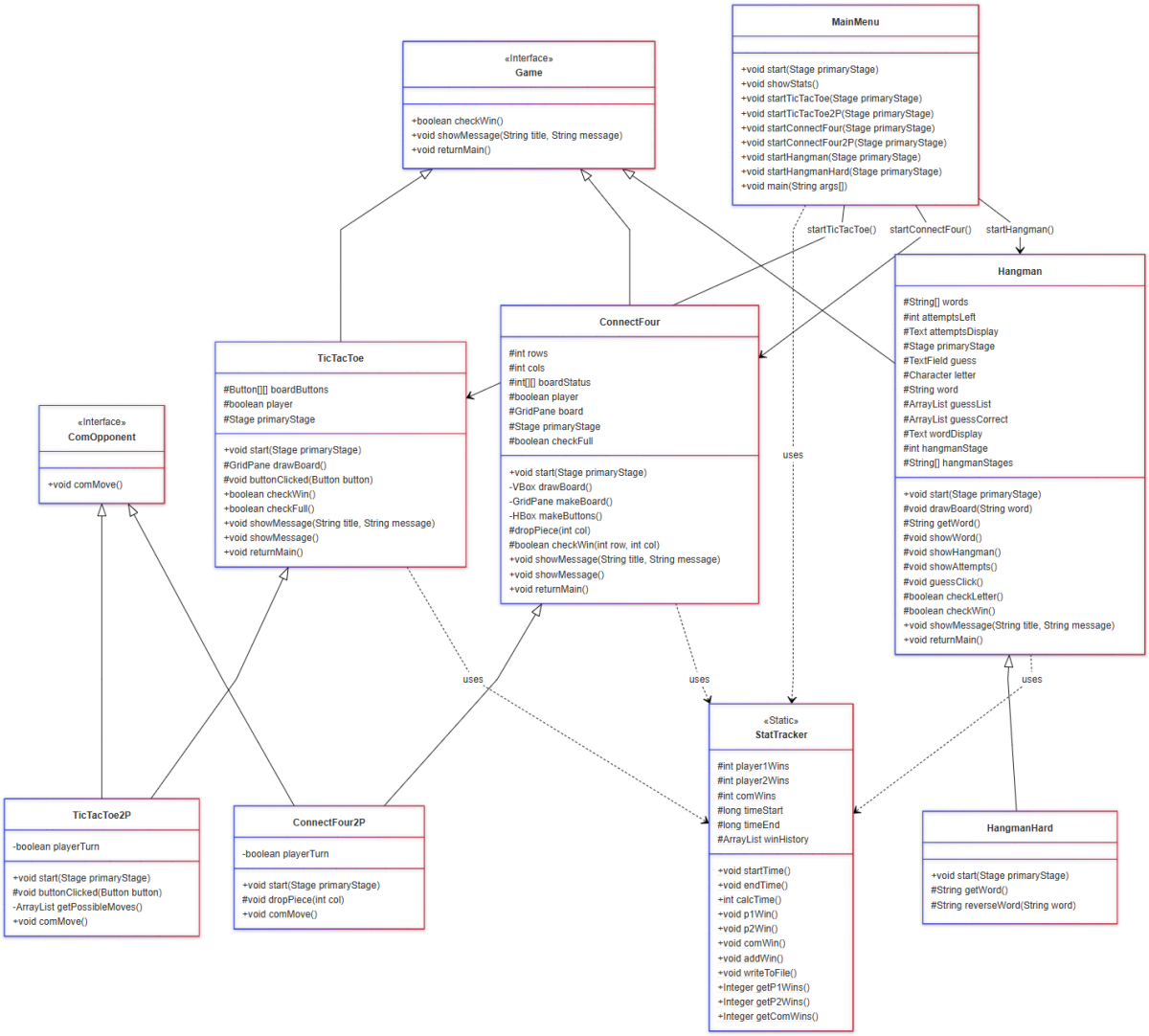
UML Class Diagram



This program works around a central class, MainMenu, which is the entry point for the program and allows the user to enter any of the following three games:

1. Tic-Tac-Toe
    a. 1 player, against the computer
    b. 2 players, against another player
2. Connect Four
    a. 1 player, against the computer
    b. 2 players, against another player
3. Hangman

a. Normal mode
b. Hard mode, where all the words are reversed

Each game is its own class, and each alternative mode is another class that extends the main game class. Each main game class implements interface Game and each child class that allows for a computer opponent implements interface ComOpponent.

There is also a static class, StatTracker, that keeps track of all the player wins and win times, and stores them in static variables and a file called winhistory.txt. More information about each class below.

KEY:

+ public

- private

protected

Interface Game

An interface designed to be implemented in the three game classes, TicTacToe, ConnectFour, and Hangman.

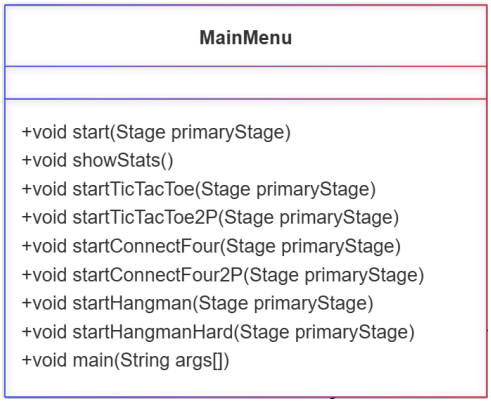
- +boolean checkWin() makes sure every game class has a method to check for the win condition
- +void showMessage() makes sure every game has the same kind of method to show the user messages
- +void returnMain() makes sure every game has the option to return to the main menu

Interface ComOpponent

An interface designed to be implemented in the two classes for a computer opponent, TicTacToe2P and ConnectFour2P.

- +void comMove() makes sure the computer has a method to make its move

Public Class MainMenu



This is the class that draws the main menu GUI and acts as the point where the player can enter the rest of the games. It extends the application class and implements the Game interface.

- +void Start(Stage primaryStage) overrides the start method from the application class, it contains all the UI nodes for the main menu.
- +void showStats() draws another menu to show each player's total wins through the StatTracker class, it calls a separate window by creating a new scene. More specific info under the StatTracker section.
- All the +start[GameTitle](Stage primaryStage) classes switch the primaryStage to the one in each game class' respective start methods, depending on the button the user clicked and the radio button they selected.
- The +main(String args[]) method launches the main menu

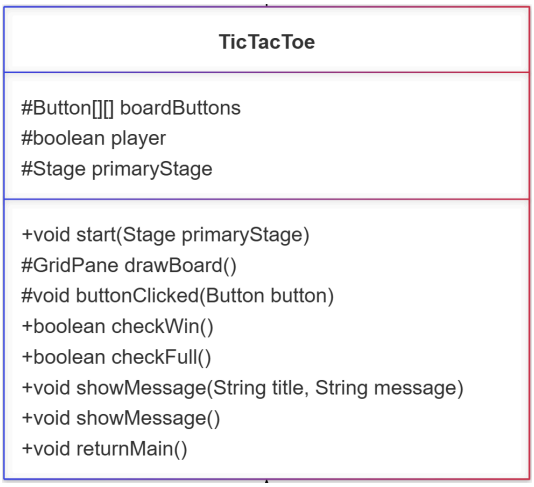
Public Class StatTracker

This static class is a class that contains static variables to store the global win count for player 1, player 2, and the computer. It also times how many seconds it takes for each win and records them in a text file. Classes TicTacToe, ConnectFour, and Hangman all use the methods in this class, they all start with a call to startTime() and end with endTime() and depending on who won, p1Win(), p2Win(), or comWin(), as well as writeToFile() to store the win in winhistory.txt. The MainMenu class uses getP1Wins(), getP2Wins(), and getComWins() to get and display the win counts with the showStats().

- #int player1Wins, player2Wins, and comWins are the int variables that hold the win counts
- #long timeStart and timeEnd hold the start time and end times for the timer
- #ArrayList winHistory is the list of wins that have been written to the text file
- +void startTime() and endTime() both get the current system time in milliseconds, int calcTime() subtracts the start time from the end time and converts it to seconds before returning it
- +void p1Win(), p2Win(), and comWin() all add to the win static variable associated with all the possible player wins, and void addWin() calculates the time it took to get a win through calcTime() and writes a string saying who won and how long it took to text file winhistory.txt through writeToFile()

- +Integer getP1Wins(), getP2Wins(), and getComWins() gets the value of the respective win integer. It returns Integer instead of int because the method toString() doesn't work on primitive types, the Integers need to be converted to strings to be displayed properly
- +ArrayList getWinHistory() returns the ArrayList storing all the wins written to the text file winhistory.txt

Public Class TicTacToe implements Game



The superclass for the TicTacToe family, it houses the main game logic and draws the UI used for the game.

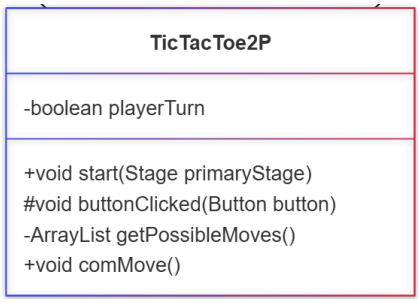
- #Button[][] boardButtons is a 2D array meant to hold the buttons that will be used in the playing field
- #boolean player is a boolean meant to determine which player's turn it is, set to true at the start, which is player 1/X's turn, when it is false that means it is player 2/O's turn.
- #Stage primaryStage is meant for the constructor:

```
public TicTacToe(Stage primaryStage){
    this.primaryStage = primaryStage;
}
```

  This constructor is established so the primaryStage can be passed around as the program switches main scenes.
- +void start(Stage primaryStage) overrides the start method from the application class, it initializes the JavaFX GUI by running drawBoard() and connecting it to a scene to be displayed

- #GridPane drawBoard() is a method that draws the game board, using two nested for loops to create a button for each space in the 3x3 playing grid, and a final button below the playing grid to allow the player to return to the menu.
- #void buttonClicked(Button button) takes a given button and assigns it the actions of taking a turn if the user clicks on an unused button, changing the text in the button to either "X" or "O," depending on whose turn it is. It also checks if the player got three in a row through checkWin(), and then checking if the board is full through checkFull().
- +boolean checkWin() checks the entire board through for loops to see if there are any instances of either player getting three in a row by checking text in the button, and +boolean checkFull() checks the entire board to see if each button has text in it.
- +void showMessage(String title, String message) shows an informational alert with the given information
- +void showMessage() overloads the previous method and shows an informational alert about the game ending in a tie.
- +void returnMain() returns the primaryStage to class MainMenu.

Public Class TicTacToe2P extends TicTacToe

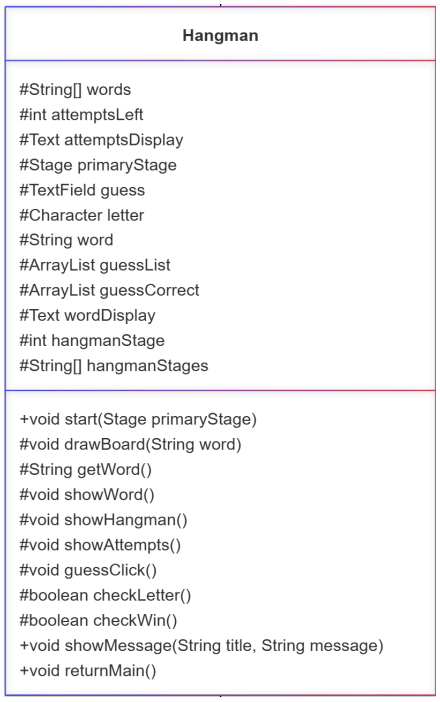


Child class of TicTacToe, implements interface ComOpponent. Overloads the method that handles the button click and has a new method to generate moves to play against the player.

- -boolean playerTurn is a new boolean that keeps track of if it's the player's turn or the computer's turn
- +void start(Stage primaryStage) calls the primaryStage of its superclass
- #void buttonClicked(Button button) is largely the same as the method its overriding, except it doesn't use an if statement to check which turn it is as only player 1 will be clicking buttons.

- -ArrayList getPossibleMoves() is a method that returns an ArrayList of all the possible moves the computer could make by looping through the playing area and finding the empty spaces.
- +void comMove() uses getPossibleMoves() and picks one of the moves at random to make against the player. Checks if the computer placed a winning move or tied the game with methods checkWin() and checkFull().

Public Class Hangman implements Game



The parent class of the Hangman family, extends class Application and implements interface Game. It houses the main game logic and draws the UI.

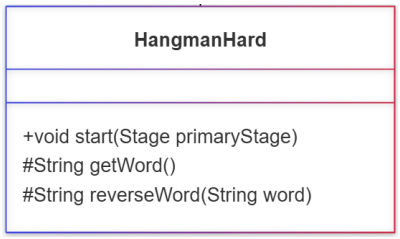
- #String[] words is a static final list that houses all the possible words that could be used #for the game.
- #int attemptsLeft is a static int that determines how many attempts the player has left to guess.
- #Stage primaryStage is defined for the constructor, so that the primaryStage can be passed around the different classes:

```
protected Hangman(Stage primaryStage){
    this.primaryStage = primaryStage;
}
```

- #TextField guess is the text field the user will use to input their guess
- #Character letter is the character taken from the player's input
- #String word is the word that gets chosen for the game
- #ArrayList guessList is an arraylist of guesses the player has made
- #ArrayList guessCorrect is an arraylist of correct guesses the player has made
- #Text wordDisplay is the text that displays the possible letters and the guessed letters of the word chosen by the program.
- #int hangmanStage is an integer that stores the index of the ascii art in String[] hangmanStages that corresponds to the amount of attempts the player has left.
- #String[] hangmanStages is a string array that houses ascii art of 7 stages of a hangman game to be displayed.
- +void start(Stage primaryStage) overrides the start method from the application class, gets the value of #String word through String getWord(), sets int attemptsLeft to 6 and int hangmanStage to 1, initializes the two arraylists storing guesses, and uses method drawBoard() to draw the board.
- #void drawBoard(String word) takes the given word and draws the game area around it, sets the text area that houses the hangman ascii art, the display of how many spaces the word has and how many letters have been guessed, the text field for the user to enter their guess, and the buttons to submit the guess and return to the main menu.
- #String getWord() gets a random word from String[] words.
- #void method showWord() sets the text wordDisplay to a series of underscores that corresponds to the amount of letters are in the chosen word, sets any letters that have been correctly guessed to show.
- #void method showHangman() shows the current stage from String[] hangmanStages to display in the text area and void showAttempts() updates the text that says how many attempts the player has left
- #void guessClick() uses an if-else block to make sure the input is a single letter, adds it to the arraylist of guesses, checks if it was correct through checkLetter(), and checks if the user has guessed the entire word.

- #boolean checkLetter(Character letter) checks if the given letter is present in the word chosen by the program, and boolean checkWin() checks if the entire word has been successfully guessed by comparing the arraylist of correct guesses to the word.
- +void showMessage(String title, String message) shows an informational alert with the given information
- +void returnMain() returns the primaryStage to class MainMenu.

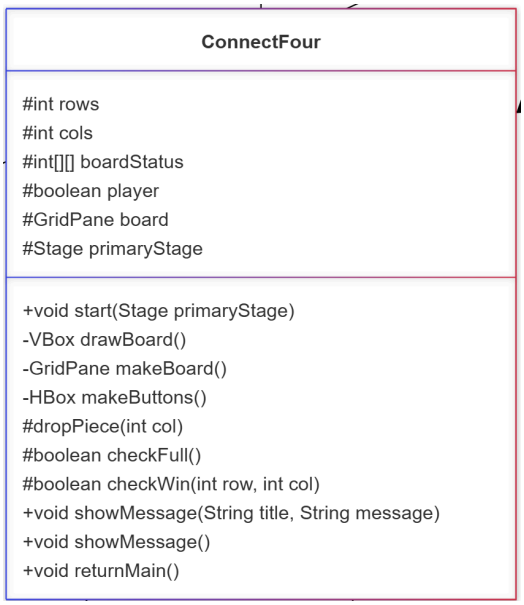
Public Class HangmanHard extends Hangman



Extends class Hangman, implements interface ComOpponent. Overrides the method getWord() to reverse the word using the new recursive method reverseWord().

- String getWord() is the same as the method it overrides, except it calls the recursive method reverseWord(String word) to reverse the chosen word.
- String reverseWord(String word) is a recursive method that reverses the given string.

Public Class ConnectFour implements Game



The parent class of the ConnectFour family, extends class Application and implements interface Game. Houses the main game logic and draws the UI.

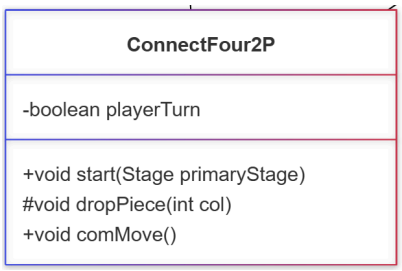
- #int rows is a static variable that contains the amount of rows the gameboard has
- #int cols is a static variable that contains the amount of columns the gameboard has
- #int[][] boardStatus is a 2D integer array that contains the information on the contents of the gameboard. If a space has a 0, it is empty, 1, a red piece, and a 2, a yellow piece.
- #Stage primaryStage is declared for the class constructor so that the primaryStage can be passed around between the main javaFX classes:

```
public ConnectFour(Stage primaryStage){
    this.primaryStage = primaryStage;
}
```

- +void start(Stage primaryStage) draws the game board UI with method drawBoard()
- -VBox drawBoard() sets up the game board and the buttons using method makeBoard to create the connect 6x7 connect four grid and method makeButtons() to make the buttons used to drop the pieces. It also sets their alignment and adds a button to return to the main menu
- -GridPane makeBoard() draws the circles that make up the gameboard with two nested for loops to fill up the 6x7 board and add them to a GridPane
- -HBox makeButtons() creates an HBox of buttons that corresponds to each column on the connect four board
- #void dropPiece() drops a piece in the selected column by looping through the column from bottom to top until it finds an empty space to place the piece, and then depending on whose turn it is it drops a red or yellow piece. It then checks if it was a winning piece or if the board is full with methods checkWin(int row, int col) and checkFull().
- #boolean checkFull() loops through the board to make sure there are no empty spaces
- #boolean checkWin(int row, int col) takes the given space and uses for loops to check both horizontal and vertical directions, as well as both diagonal directions to see if the piece was the fourth in a row and therefore was a winning piece.
- +void showMessage(String title, String message) shows an informational alert with the given information

- +void showMessage() overloads the previous method and shows an informational alert about the game ending in a tie.
- +void returnMain() returns the primaryStage to class MainMenu.

Public Class ConnectFour2P extends ConnectFour



Extends class ConnectFour, implements interface ComOpponent. Largely similar to ConnectFour, with a few modifications to the dropPiece(int col) method and a new comMove() for random move generation.

- -boolean playerTurn is a boolean to track if it is the player's turn or the computer's turn
- +void start(Stage primaryStage) calls the start method from the superclass
- #void dropPiece(int col) overrides a method from the parent class, it is largely the same, just without if-else blocks checking which player is doing it, as it is exclusively for the player now.
- +void comMove() randomly generates a column to place a piece in, and then drops the piece using the same code used in both dropPiece() methods.