Project Report

Connect4FX Game

Yassr Shaar

Student ID: 14328571

MSc. in Computer Science NL
Module Code: COMP20300
Module Title: Java programming
Lecturer: Dr Simon Caton

UCD School of Computer Science
University College Dublin
December 18, 2020

Table of Contents

Introduction

The game chosen for this JavaFX project was Connect4.
Connect4 is a two player, turn-based board game that is typically comprised of a board that is seven columns in length and six rows in height.

This game was chosen as it offered a simple concept of gameplay, the players only need to choose a column where a disc would drop, however, the complexity of the game comes with the different ways a player can win. This level of complexity coupled with the desire to "make more" of this traditional game is what compelled me towards the project.

Game Implementation

This project's implementation of Connect4 provides further options to players which include customisation of the player name, colour, board size and finally the options to pause or play music. This is a very visual implementation of the traditional game and is aimed to be aesthetically pleasing and enjoyable.

The game rules are in line with the traditional game in that any player is capable of winning by having a combination of four discs of the same colour in any of the following directions:

- Horizontally
- Vertically
- Diagonally (to the left or right)

This implementation does also take into consideration the possibility of a draw occurring where neither player wins and there are no empty tiles on the board to place a disc.

Connect4 trivia

Connect4 trivia that I discovered purely through the designing of this game (quite possibly the creators of the game did not even realise this).

The original colours for Connect4 are Yellow & Red.
When we look at a HEX colour picker, we can see something very interesting:

When we select #ff0 we get the colour **Yellow** & if we were to **Connect4** 0's (discs) to that hex colour #ff0000 we get **Red**!

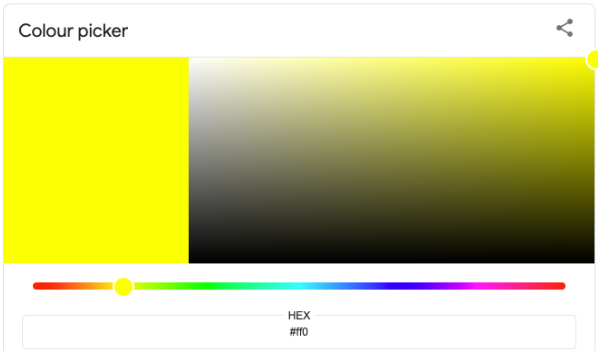

Figure 1 - Hex Colour picker Red

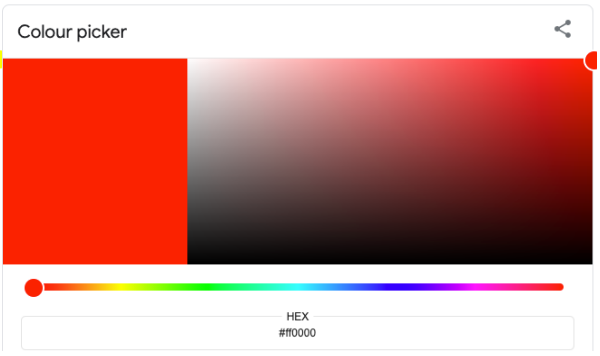

Figure 2 - Hex Colour picker Yellow

Class Structure

GameMain.java

The class structure of the JavaFX game begins with the GameMain. This is where we create our Stage and scene which we call to it the Main Menu preference method.

MainMenu.java

The Main Menu preference method returns a GridPane which is filled with customisation and graphics tailored to the player's choices. Here the players can choose their name & colour for a basic game, however, they can also choose to have the background music playing or not, they can also optionally increase the size of the board to create a more complex game.

Whether the players chose to customise the game or go with the default settings, clicking the Start Game button will launch the game. This button performs the following:

Figure 3 - main menu display

1. The list of Players is cleared. This ensures that when a new game is started it will take's into consideration new players added.
2. Sets the music button image to display a No music sign. (this is relevant as the music stops during the end screen to allow for a different music clip to play)
3. Sets the Column and Row based on the player's choice
4. Creates the two players and takes in their name and colour choices.
5. Changes Scenes by calling StartGame.

StartGame is a method within MainMenu which is instructed to firstly call the method within Disc which clears the board of any previous game attempts and retrieves the required elements to be displayed within the game. This includes a grid for the board and the discs that will be used to play the game along with the selector that aids with identifying which column the player is about to drop a disc in. The method will return a gameroot pane which will result in changing scenes to the main game.

GameDesign.java

The logic for the main part of the game is constructed within he GameDesign class while any actions that occur related to the game discs occurs within the Disc class.

GameDesign firstly holds the 'not so constant' constants for the diameters of the game. This would mainly include the Columns, rows and tile size (which is a nonadjustable size used to keep a consistent sizing of the board). Most importantly the player1Move Boolean is declared here and asserted to be true.

Some fundamentals are created here such as the board shape. This includes some visualisation elements to make the board appear 3D. Such things as punching holes in the board shape and sliding the disc behind them to appear like the traditional game in 3D. Also, the choice of lighting and shadows amplifies this effect.
The selector is created within this method to allow for a visual representation of where the mouse is hoovering on the board and which column would the disc drop on if the player

were to click down. This would return a list of rows within the column for which the selector is hovering over.

The createPlayer method takes in the player's name and colour when they are assigned in the MainMenu class. The name is checked and manipulated to best suit the structure of the game. This is done by removing any spaces in the name (helps with saving the leader board), checking if the name is empty, if it is then a default name is given such as Player1; ensuring that the player name is no longer than 8 characters (for visual purposes).

The colour is also checked to determine if no colour is selected or returned, by default the colour RED will be assigned for player 1 and YELLOW for player 2. Since the Colour picker is used to retrieve the player colour choices this should never be an issue, but the precautions are being taken here.

The final two aspects of the game design are focused on consideration of whether the game has been won or if there are further discs to be played.

Whenever a disc is played, the gameEnd method is called and takes in the column and row of where the new disc landed. This method then uses the new disc coordinates to create 4 lists which take 3 coordinates off of where this disc is placed. The 4 lists are the coordinates of points vertical, horizontal and diagonal (from bottom left and bottom right) of our disc. Once these are collected the checkWin method is called to determine if any of these lists would result in a win.

The Checkwin method takes the list of points from gameEnd and determines if the 4 coordinates are for discs of the same colour. If they are then the player who placed that last disc is the winner.

Disc.java

The logic for how the win is handled and the discs dropping is handled with in the Disc class. Firstly, the constructor for Disc checks firstColour, which is a Boolean to determine which player's turn once that is known, the colour is assigned to the disc to be dropped.
The centerX and centerY are also set here so that the discs are dropped in the correct location which would be the Circles previously cut-out from the board.

The first method would be the dropDisc method which manages everything that occurs that allows a disc to drop and what happens WHILE the disc is dropping.

A grid is created which stores the column and row of where the disc being dropped has landed. This is critical to ensure that the discs stack on top of each other and that we can find the game winner.

Figure 4 - Can't place another disc here

Firstly, row is counted backwards with 0 being the very top of the board where the discs come from. This allows for the discs to drop from the 0 to the set number of Rows determined by the place. If the row is greater than 0 that means that the player can keep dropping discs within this column, however once it gets to 0 the player will be prompted that they are no longer allowed to drop discs in this column.

An animation is created to aid with the visual aspect of the game play. This animation simply displays the dropping of the disc from the top of the board down to the last open position in the column selected. A duration of 0.5 seconds is assigned to this action, during which a malicious player may try to drop a few more discs by spamming the mouse click. To counteract this, we disable the gameboard until the disc is finished it's dropped animation and the turn has changed to the next player. While the disc is dropping a sound clip is played that adds to the immersion of the gameplay.

A number of critical checks and assignments occur as soon as the disc has finished it's drop animation. Firstly we check if the game has ended, in which case we stop the main gameplay music clip and play a victory cheer sound clip, followed by the method gameOver() being called.

If the game hasn't ended then we check if the game is still continuing to play, in this case we would re-enable the gameboard and change the player turn to allow the next player to take their turn.

Here we also check if the game is a draw, this is done with a simple counter that adds every move played to its ArrayList then comparing the size of the ArrayList with the number returned when Columns is multiplied by Rows. If this is true then a Draw is declared which plays a specific sound clip and passes the method GameOver from EndScreen.

Disc also contains the player name displayed at the bottom of the screen as part of the turn's animation. With the drop of every disc and change of turn the name at the bottom of the screen will also change to reflect the player's name.

Finally Disc houses the ClearAll() method, a powerful method that wipes the board and resets all parameters associated with it to enable a new game to be played.

EndScreen.java

Once the game is deemed as "over" the method GameOver() is called from within EndScreen. GameOver will first wait 3.5 seconds to allow the celebration sound track to end before playing the end of game song. During which it will write to the leaderboard score to the board, declaring the winner and loser of the game and it will set up the new scene for the end scene.

The endPane method creates all of the elements that will be displayed on the end screen, from the winner and loser to the full leader board along with two buttons, one to display a non-interactive view of the final board and another button to restart the game and head back to the main menu.

The winner and loser name and colour are retrieved and stored to be displayed on the screen along with a message for each player. Beneath that we display a TextArea that houses the leader board for all previous games including the one that just finished. Every new game that ends has its winner and loser names added to the end of the list.
Below the leader board are the buttons for reviewing the final board, which is made to be non-interactive so that no further changes can be made, along with the main menu button to start a new game.

Depending on the outcome of the game, it is possible to have a DRAW where no player wins the game. This is taken into consideration and the graphics on the end screen will behave

accordingly to this outcome. (Below are displays from choosing the **Review Final Board!** button).

Normal game outcome:

Figure 5 - Review final board normal game

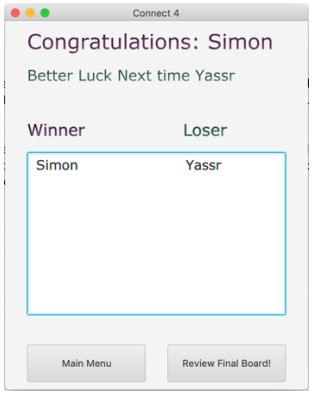

Figure 6 - Review board normal game

Draw:

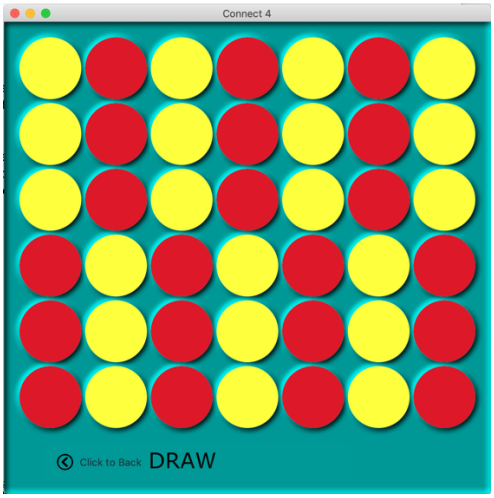

Figure 7 – Review final board Draw

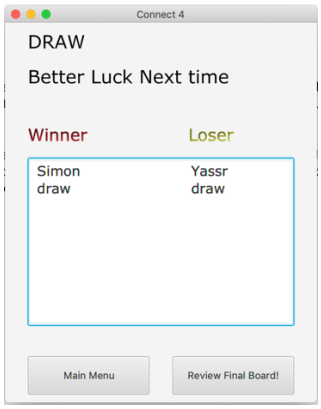

Figure 8 - Leaderboard Draw

The method printLeaderTextArea() is called during the creation of endPane() to read the data that is stored within the Leaderboards.txt file and write the results of the file into the Leader board textArea. To allow for multiple iterations of the game without the read function spamming the textArea with the same results every time the method is called, we clear the ArrayList which contains the results before reading the leaderboard.txt file.

Leaderboard.java

Creates an ArrayList that will contain Strings to be read from the leaderboard and passed onto EndScreen. Firstly, if the file leaderboard.txt does not exist then a new file with that name is created. Depending if the game is a Draw then the words “Draw Draw” are written into the text file, alternatively the winner and loser of the game are written into the text file with a space between them. This space between the names is used in the readFile method to split each sentence read in from the text file into input [0] for the winner and input [1] for the loser. An if statement is in place to tell the reader that if a line is blank then simply ignore it, this is to resolve any issues that may occur when creating a blank file that could have an empty string at the start.

Player.java

The player class is used to create a player and store their name and colour. It is a minimalist class that focuses on providing a constructor to create new players and the getters/setters necessary for manipulating the created players.

Music.java

This class focuses on creating functions that handle music throughout the game.
The constructor takes in a sting for a sound clip location along with a Boolean to determine if the song should be on or off. The methods implemented are the Play method for playing a soundtrack based on a number of times passed to it to repeat. The Stop function to end any sound clip and finally the loop functionality to loop a song continuously until told to stop.

Class Diagram

Game classes:

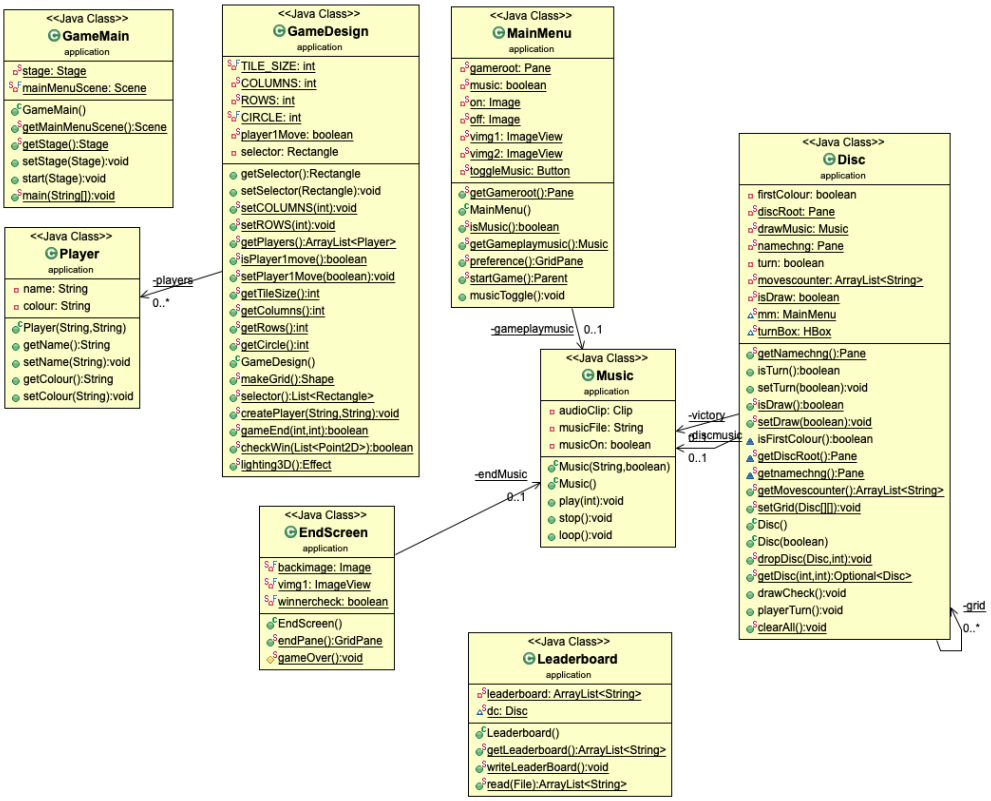

Figure 9 - Class diagram

The class diagram in figure 9 displays all of the classes that make up the Connect4FX game. The relationships between the classes aren't accurately captured within the diagram as the structure of the game was initially designed with a narrow outlook of abstract methodologies and object-oriented mindset. Which lead to the blind development of static variables and methods.

These static variables and method are represented at a global state and are therefore much more difficult to reason with and unit test. This was a harsh awakening during the development of the project at a time which was unfortunately too short of notice to untangle the binds of static dependencies.

A reconsideration of this project would be to avoid static variables unless extremely necessary.

Test & Game Classes

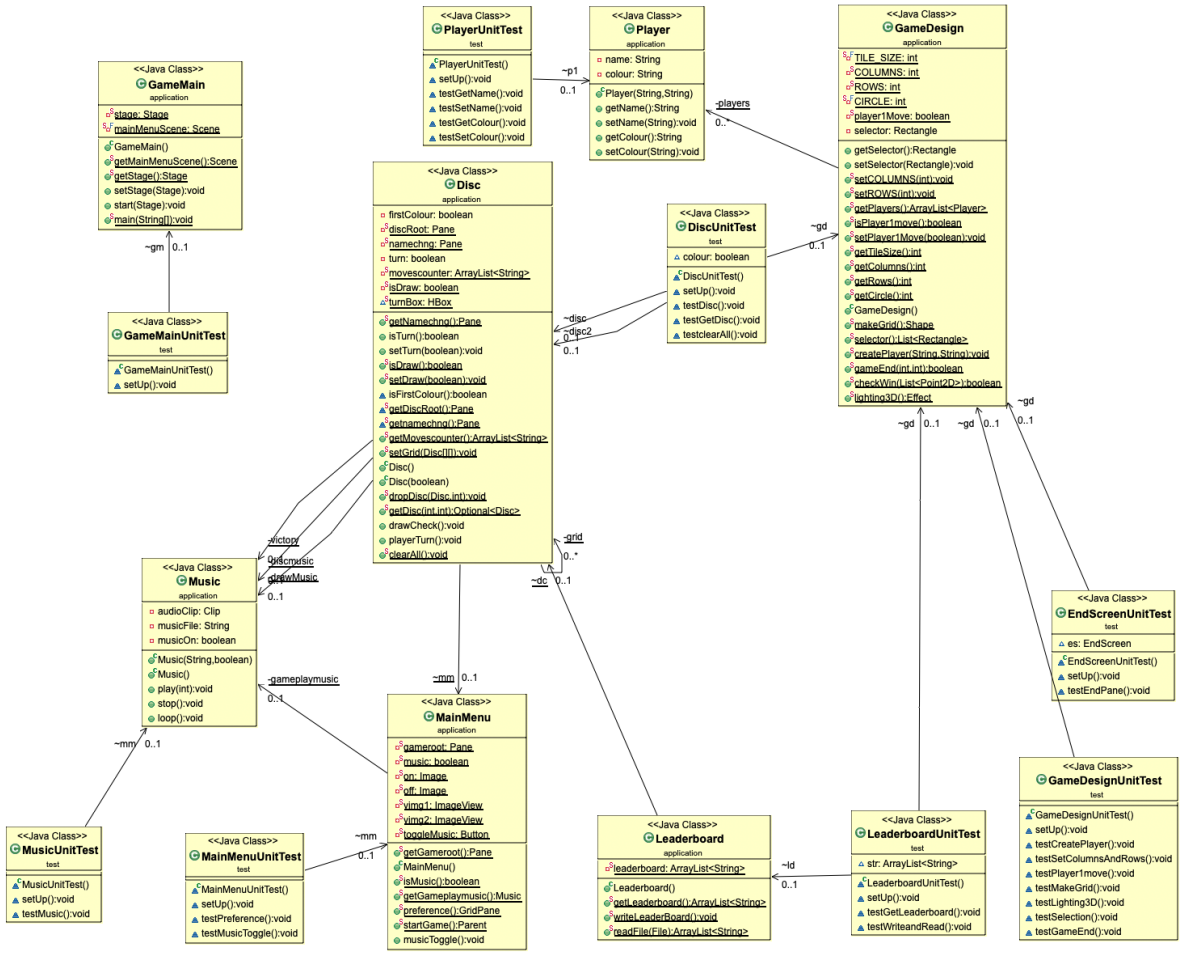

Figure 10 - Class and Unit Tests Diagram

The diagram in Figure 10 represents the relationship between the project classes and their respective Junit tests. There are much clearer relationships created here as each test is designed for its respective class.

Encapsulation

Encapsulation was used prominently throughout the project to create variables within specific methods that use them, hiding them from other classes and making them only accessible through the calling of those methods. This can be seen in every class of the project, notably for creating Lists that append details within the method to be returned and in the creation of objects such as Shapes, Panes and labels.

Model View Controller

despite not implementing MVC as traditionally would be expected, the project does borrow some design considerations when approaching the handling of some elements. This can be more prominently seen with regards to the displaying of the End Screen of application. The creation of the model elements such as the pane, buttons, text and labels is all handled within a single method dedicated to the View, while the logic behind it is controlled within the body of the main method.

Game Unit Testing

GameDesignUnitTest.java

This test starts off by running a quick instance of the game as do most tests, using JFXPanel().

- testCreatePlayer(): The test creates 4 players and checks that their names and colours are being handled as expected.
- testSetColumnsAndRows(): provides an input for columns and rows and ensures they are handled.
- testPlayer1move(): Checks that the first player is recognised.
- testMakeGrid(): ensures that the function makeGrid is returning a grid.
- testLighting3D(): Tests the 3D effects applied.
- testSelection(): The selector first value would return the top left corner of the board.
- testGameEnd(): Checks if a game is declared as winnable.

DiscUnitTest.java

Create two players and two instances of Disc with the varying constructor.

- testDisc(): checks the two types of constructors
- testGetDisc(): checks the retrieval of a disc
- testclearAll(): checks that the elements that are cleared are in fact empty.

MainMenuUnitTest.java

Sets up a main menu instance.

- testPreference(): checks for a return of the GridPane
- testMusicToggle(): tests the music toggle Boolean.

EndScreen.java

Creates an instance of EndScreen and two players.

- testEndPane(): ensures the endpane is returned and it is not null
- testprintLeaderTextArea(): ensures that a TextArea is returned and not null

PlayerUnitTest.java

Creates a player and tests the getters and setters.
This is done by giving new values for name and colour and asserting that the getter for player would equal the new values.

LeaderboardUnitTest.java

Creates two players which are saved to a text file. To run this test successfully, it will be required to clear the Leaderboard.txt file before running this test.

Assuming the leader is clear then the response should be a simple capture of the first results in the text file which where the two players added.

Improvements over Traditional game

This game is far superior to its traditional counter as features many impressive and fun elements that enrich the user experience. The Grid size can be adjusted for a more tricky and fun game. Players can choose any colours they desire with no restrictions! This creates fun game modes such as “Who’s who??” where both players choose the same colour or an even further step up would be both choosing invisible disc colours. Finally, MUSIC to add to the atmosphere of the fantastic game!