**Module: Software Development 1**

**Module Code: SET1110**

**Module Leader: John Owens**

**Matriculation Number: 40312877**

**Coursework Submission: Connect Four Game**

**Date of Submission: 1 December 2017**

Contents

[Design Report 3](#_Toc499848119)

[Grid Representation 4](#_Toc499848120)

[Grid Display 5](#_Toc499848121)

[Four Discs in a Row Check 6](#_Toc499848122)

[Prevent Adding Discs to a Full Column 6](#_Toc499848123)

[Advanced Features 6](#_Toc499848124)

[Save Game 6](#_Toc499848125)

[Check for Diagonals 6](#_Toc499848126)

[Figure 1 Connect4 Class Diagram 3](#_Toc499719346)

[Figure 2 Blank Board 4](#_Toc499719347)

[Figure 3 Game in Play 4](#_Toc499719348)

# Design Report

The Connect4 game has a main class called Connect4. Where required, it can call the methods contained within the classes of Board, Connect4Board, Counter and FourCheck classes.

The program uses a 2D array to create the six by seven grid required for the game. This is updated every time a counter is placed in a column. The game has a choice of the red player winning or the yellow player winning or drawing. It uses a Boolean function so that the game starts off as a player winning equalling false. When the game detects a winning combination, it changes this Boolean function to true, allowing the identification of the winner or whether the game is completed as a draw.

There is a method to identify a drawn game.

There is a method that reports to the main that a win has been detected. This is fed by the various methods that check for a win – up, across and the four diagonals. The game uses a Do While loop to allow a choice as to whether to play again.

The program was built in the order of

1. Create and display the board.
2. Locate a counter on the board and prompt if out of range.
3. Prompt a player if the column they are using is already full.
4. How to check for a draw.
5. How to check for a win in a row.
6. How to check for a win in a column
7. Change 5 and 6 to create a win method which checks with all winning methods as this would be required for including checking diagonals.
8. How to check for diagonals.

Also added was a menu option to allow another game to be played.

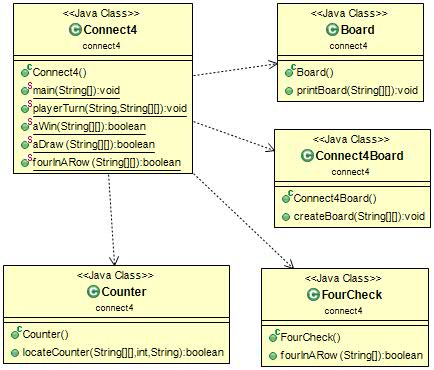


Figure 1 Connect4 Class Diagram

# Grid Representation

The grid is represented using a 2D six by seven array. This uses a null space contained within square brackets [\u0000] to represent each space in the grid. In the grid this looks like [ ] (figure 2) but allows the placement if a token inside, [ R ] or [ Y ] replacing the null. This is not ideal as the shape of the grid can change as the counters are placed in it (figure 3). This makes it harder to judge precisely which column the counters on the board are in and therefore to make a logical next turn. A loop is used for the height and another loop for the width.

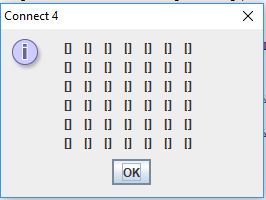


Figure 2 Blank Board

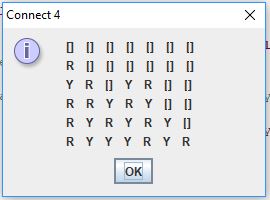


Figure 3 Game in Play

# Grid Display

The grid is created by using the class Connect4Board. It is printed using the class Board. These both use a For loop inside a For loop to build a 2D array to generate the grid. This is updated and re-displayed after each player’s turn. An issue that had to be resolved when first trying to place a disc in the grid was that on each attempt, the player was informed that the column was full. There were two reasons for this:

1. The code for the 2D array in all the classes where square brackets are used needed to have a null space inserted into them to allow that to be replaced by the counter as it was placed. Before that was put in place, a grid of six by seven with null written in every space appeared.
2. In the class Counter, the integer values of ARRAY\_BOARD\_BASE and ARRAY\_BOARD\_HEIGHT required swapping around.

# Four Discs in a Row Check

The FourCheck class tells the program what four counters in sequence looks like. It creates arrays that are four boxes in length. It checks these boxes to see if there are any null values in them or if two For loop arrays contents don’t tally. This prevents four counters of any colour in a sequence creating a win. The main class then calls this and uses it with the methods for checking against columns rows and the four diagonals.

# Prevent Adding Discs to a Full Column

The class Counter tells the program the depth of the board and allows a counter to be placed in the column if there is a null space in it. In the Connect4 class, the method playerTurn calls this. Once the method has identified whether it is the red of yellow players turn, it asks the player to choose a column between zero and six to place the counter in. If the column number chosen is outside this range, the player is prompted to try again. Counter uses the 2D array and uses a for loop to check each column. It then checks to see if there is a null space in the chosen column. Counter does a Boolean check to see if the outcome for the chosen column is true. If it is, the counter is placed in the first (lowest) available null space. This in turn is updated to the main part of the program and onto the grid. The relevant array box is now filled. If it is false, the player is prompted to choose another column.

# Advanced Features

## Save Game

Not successful.

## Check for Diagonals

In the Connect4 game there are four diagonals to check for. Lower left to upper right, lower right to upper left, upper left to lower right and upper right to lower left. This is done using a For loop inside a For loop. This takes the elements for checking a row and a column and iterates across them together. The relevant row and column checks either increment or decrement as they move across the board and create a diagonal array. These are all checked against the FourCheck class and return true if detected.