Swift Connect 4

Kevin Singpurwala

Student ID: 16359146

Module: COMP47390 Swift programming 2022-2023

Supervisor: Dr. Guénolé Silvestre

UCD School of Computer Science

University College Dublin

December 21, 2022

Table of Contents

1 Swift Connect 4 Project 2

1.1 Overview 2

1.2 First Steps 2

1.3 Left To Make functionality 4

1.4 Working functionality 7

Chapter 1: **Swift Connect 4 Project**

1.1 Overview

We set out to make a connect 4 board game swift iOS application using storyboard with a Model View Controller architecture. The base project already gave us a good start. It contained a gameSession protocol with all the common methods that we would need such as isValidMove which checks if a discDrop is valid for a given column. The αC4 ai provided us with an ai that could play connect 4. Furthermore, we were provided some ViewController class that demonstrated how to implement the protocol and a few other pieces such as the printed board, boardInit function, etc. [1]

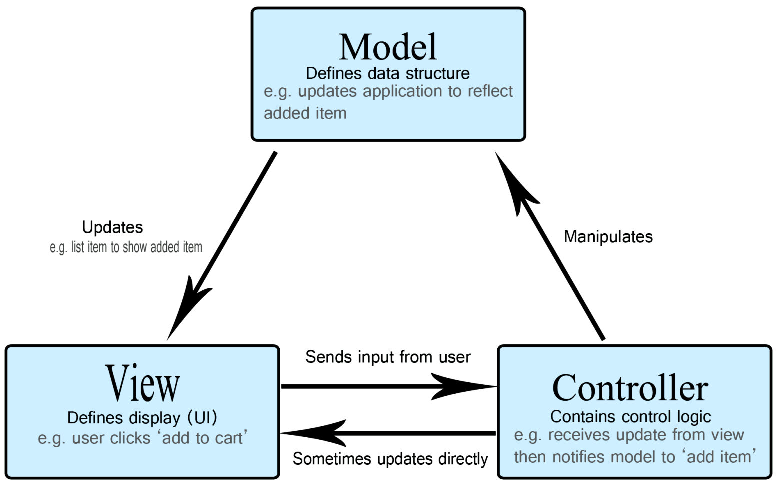

Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Figure 1.1: MVC- the mozilla foundation

1.2 First Steps

The first thing I did was create the model, then the view and finally the viewController. Some of the items in viewController would better be moved to view however near the end in time trouble I added all remaining functionality in viewController. Ideally the viewController should not be performing any calculations. This should be done in the view. The viewController is simply a means of presenting the view and the model contains the data structure of the game such as the number of Columns.

1.2.1 Model

The connect 4 model file contains a Connect4Model swift class that provides utility functions for calculating and returning various values related to a Connect 4 game board. The Connect4Model class has four methods for calculating and returning the grid width, disc radius, initial tile position, and hole positions on the board. The getGridWidth method takes a boardWidth argument and returns the width of each grid square in the board. The getDiscRadius method takes a boardWidth argument and returns the radius of the discs in the game. The getInitTile method takes a boardSize and middle point as arguments and returns the point at which the top-left corner of the board should be located. The createHolesHelper method takes a boardSize, boardBeginning point, column, and row integer values as arguments and returns the point at which the center of the disc at the specified column and row should be placed. The Connect4Model class also has two private helper methods, goRight and goUp, which are used to calculate the horizontal and vertical offsets for the hole positions on the board.

1.2.2 The View

The view is divided in to the connect4 View and the discView.

The DiscView class represents a graphical disc that can be displayed in an iOS user interface. It is a subclass of UIView and has a single instance variable called discDropped which is a tuple containing information about the disc such as its color, index, and action. The DiscView class also has a private instance variable called discLabel which is a UILabel object used to display text on the disc. The DiscView class has an initializer that takes a CGRect object and an optional tuple as arguments. It initializes the discDropped instance variable with the tuple if it is provided, or with default values if it is not. The initializer also sets the discLabel object's frame, center, and text alignment properties. It also sets the background color and border properties of the DiscView object based on the color of the disc. The DiscView class has a displayLabel method that takes an optional array of integers as an argument and sets the discLabel object's text to the index of the disc. The method then adds the discLabel object as a subview of the DiscView object.

The Connect4View class is a subclass of UIView, which is a basic view class provided by the UIKit framework. It represents a graphical view of a Connect 4 game board and has properties that store information about the board, such as its width, center point, and colliders. It also has a private CAShapeLayer object called board that is used to render the board. The Connect4View class has a boardSources property that is a weak reference to an object that conforms to the Connect4DataSource protocol, which defines a set of methods that provide information about the board such as its start point, disc radius, and grid width. The Connect4View class also has an override of the draw method, which is called when the view needs to be redrawn. In this method, the board object is removed and a new UIBezierPath object is created and populated with circles representing the holes in the board and a rounded rectangle representing the board itself. A CAShapeLayer object is created, its path property is set to the UIBezierPath object, and its fillRule and fillColor properties are set. The CAShapeLayer object is then added as a sublayer to the Connect4View object's layer property. [2] [3] [4]

1.2.3 View Controller

The Connect4ViewController still needs a lot of work. The connect4 Model and the connect 4 View or more less complete.

The Connect4ViewController is responsible for managing the Connect 4 game interface. It has a number of properties, including a resultLabel for displaying messages to the user, a Connect4View for displaying the game board, and a Connect4Model for handling game logic. The Connect4ViewController class also conforms to the Connect4DataSource protocol, which defines a set of methods for providing information about the game board.

The Connect4ViewController class has a number of methods for handling game events. These include methods for handling player turns, resetting the game, and displaying messages to the user. The class also has a tap method that is called whenever the user taps the screen, and a playerTurn method that is called whenever it is the player's turn to make a move. Additionally, the Connect4ViewController class has a number of private helper methods for handling game logic and displaying messages to the user.

1.3 Left To Make functionality

1.3.1 Add disc "dropped in" effect

In the current implementation, discs are simply rendered on screen, not dropped in with a nice visual effect which would be more ideal. We can add that code to the viewController with something like this using gravity and bounce.

```
b2Vec2 gravity(0.0f, -10.0f);
b2World world(gravity);

b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
bodyDef.position.Set(x, y);

b2CircleShape shape;
shape.m_radius = radius;

b2FixtureDef fixtureDef;
fixtureDef.shape = &shape;
fixtureDef.density = 1.0f;
fixtureDef.restitution = 0.8f; // bounce coefficient

b2Body* body = world.CreateBody(&bodyDef);
body->CreateFixture(&fixtureDef);
```

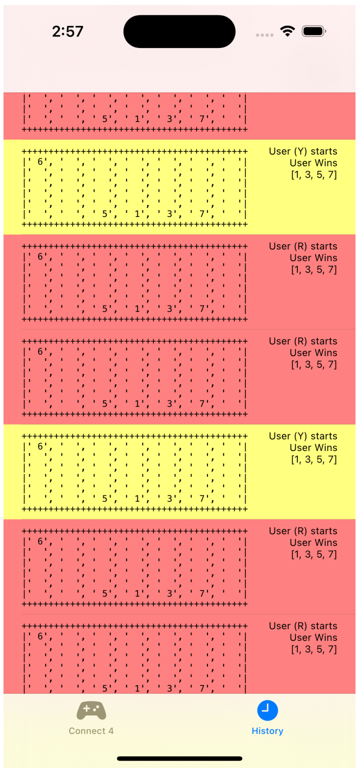
1.3.2 Persistence for History

The history tab navigation works to bring up the history view but the history view itself and the replay functionality is not implemented. The history view also brings up a default board that it stores in history instead of the board that is meant to be used implement in the connect4 view controller. To further this process one would need to replace these cells in table view controller with for example UIImageView() which would take a picture of the completed game when case .ended is true. To add a UIImage cell to table view something like: To include Swift code in a

LaTeX document, you can use the listings package. Here is an example of how you can do this:

```swift
class MyTableViewCell: UITableViewCell {
    @IBOutlet weak var myImageView: UIImageView!
}

class MyTableViewController: UITableViewController {
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
    IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
        "MyTableViewCell", for: indexPath) as! MyTableViewCell
        let image = UIImage(named: "myImage")
        cell.myImageView.image = image
        return cell
    }
}
```



User (Y) starts
User Wins
[1, 3, 5, 7]

User (R) starts
User Wins
[1, 3, 5, 7]

User (R) starts
User Wins
[1, 3, 5, 7]

User (Y) starts
User Wins
[1, 3, 5, 7]

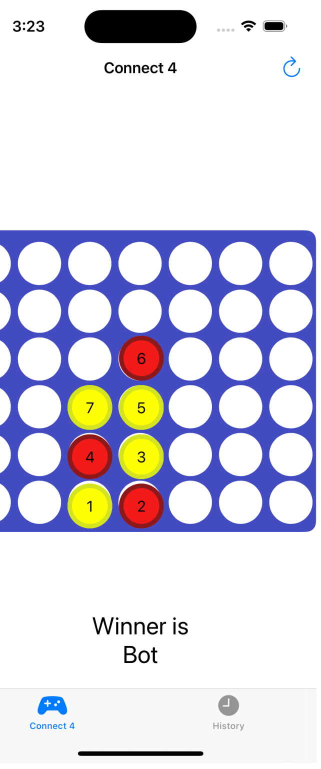
User (R) starts
User Wins
[1, 3, 5, 7]

User (R) starts
User Wins
[1, 3, 5, 7]

Figure 1.2: Game History View

1.3.3 Board array mismatch -> Due to this,the win condition does not work. ".ended" called when it shouldn't be

In figure 1.3 we see that the label states that game is over and that bot won. This is because the board I generated on screen is not the same as the one being stored in CoreData. CoreData has a board as in figure 1.2 where bot wins with a static (1,3,5,7) horizontal 4 in a row. To fix this issue we would have to update our persistence and make it so that Coredata receives our new board array from the GUI. Then it can register it on the history tab as further discussed in section 1.3.2.



Winner is
Bot

Figure 1.3: Failing Game Ended Example

1.3.4 Reset board, discs and Label

Ideally board, discs and Label should be reset. You can get away with not resetting board, but discs and label must be reset.

Something like what is below can reset the discs from connect4

```swift
// Remove the disk from the view controller
  disk.removeFromSuperview()

// ... (reset the game) ...

// Add the disk back to the view controller
  view.addSubview(disk)
```

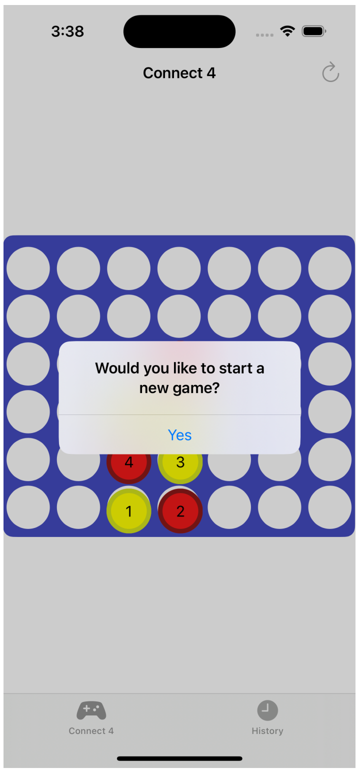


Figure 1.4: Working Label Example

For the label we already have the reset working in the viewController. We press the rest button on the top right and go from figure 1.3 to 1.4. The label is reset correctly and after you press yes the label will display whose move it is again. This is the game loop. A new game will be added to history but as seen in Section 1.3.2, it is not the correct game. The obvious problem with figure 1.4 is that the discs are not reset. some gameSession.resetGame() function needs to be called. Or reset the whole connect4 view after the game is saved to coredata.

1.3.5 make board work with game logic (output) win conditions

The protocol thinks that the game is over and the ai does not work properly with the GUI board created. This is because our board is created and the protocol does not know about it, nor does coredata. Fixing the array mismatch issue should resolve this problem as well.

1.4 Working functionality

In figure 1.5, we see that the connect 4 view and disc view are rendered on screen properly. The board was made with UIBezierPath as where the discs. The discs have the index number which is the order of which they were dropped in. The discs drop in to the board properly in to the "holes" when the column to drop in to is specified by clicking over that column area§. The discs have an outer and an inner colour so that the styling is better. The rectangle connect 4 blue rectangle has "holes" placed in it where coins will be shown on top after. We also see that the discs toggle correctly between red and yellow. The tab bar Controller is implemented and can be used to change views between the "history" (table view controller) and the "connect 4" connect4viewController

Game. The "label" the bottom of the screen also works and indicates whose move it is. Either player or bot. Currently bot is not working so both are human players, differentiated by disc colour.

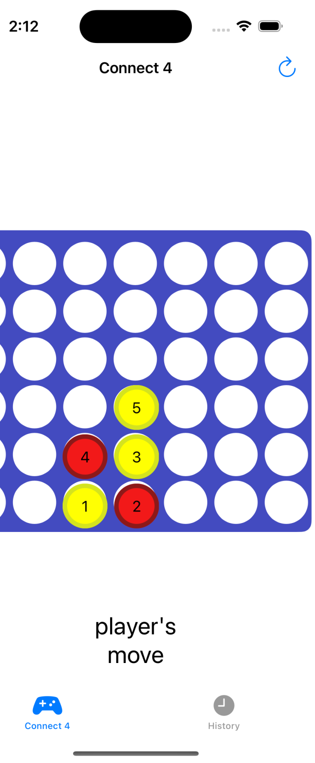


Figure 1.5: Working Game View Example

Bibliography

[1] Mozilla, *Mozilla mvc*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC.

[2] R. Chuang, *Github*. [Online]. Available: https://github.com/5j54d93/Connect-4-iOS-Game.

[3] G. Theodoropoulos, *Bezier paths introduction*. [Online]. Available: https://www.appcoda.com/bezier-paths-introduction/.

[4] K. H, *Github*. [Online]. Available: https://github.com/kenrick95/c4.