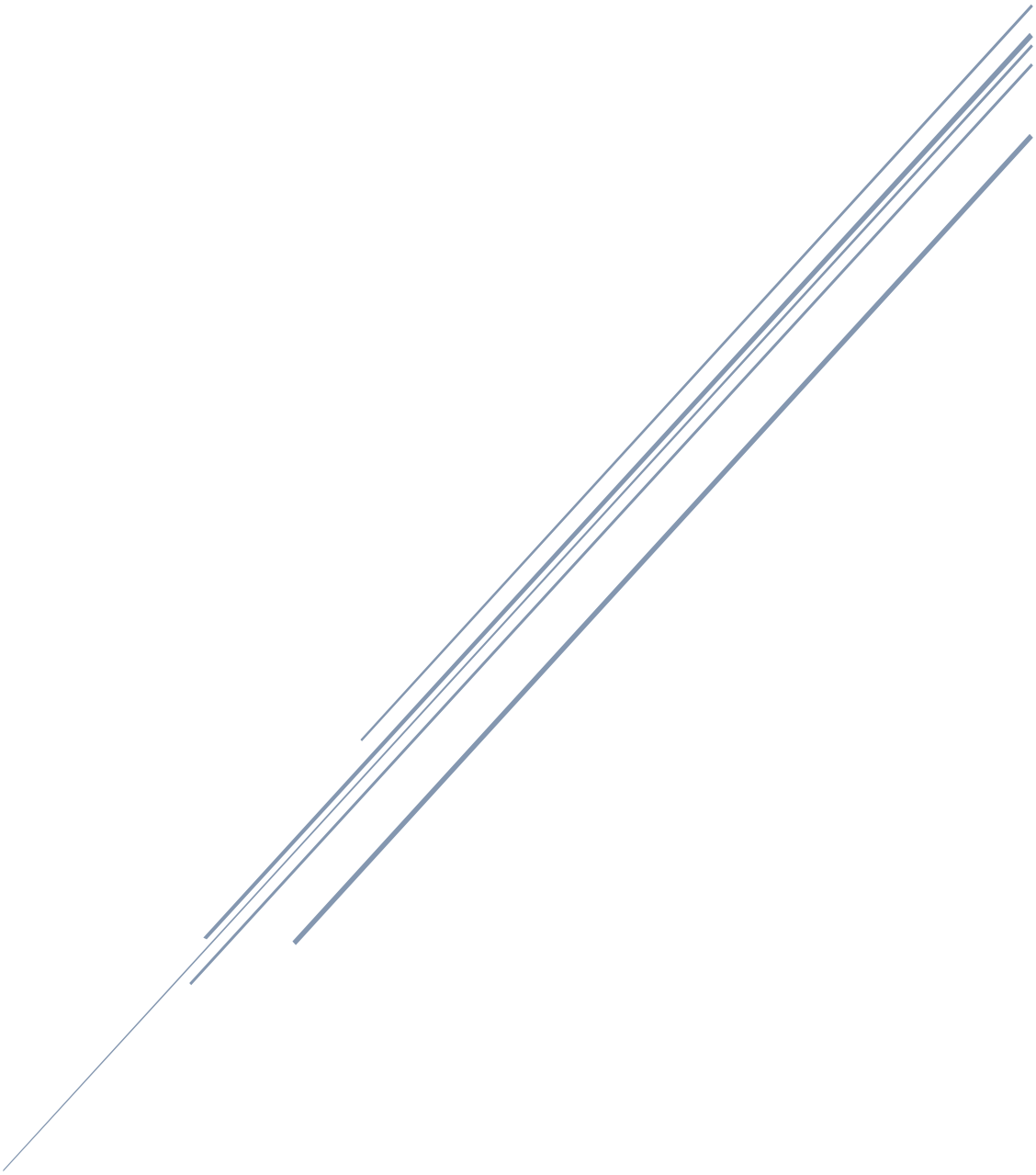
TIC-TAC-TOE CODE EXPLANATION

ITEC30031: Wireless & Mobile Comms



Ruairidh Taylor
N0629719

Code features –

- UIImageViews as buttons
- Multiple View Controllers
- Score tracking
- Audio
- Hard AI
- PDF Viewer
- UI status bar colour changed

From the Image you can see that the code is made up of 2 primary pages, the first being playing against your friend locally on the same phone, and the second is playing against the computer. I have also implemented a UI view within the rules view controller where a pdf of the rules is displayed.

Core Game Play

Once that has all finished the game will then check for a winner this is done by using a function that checks each possible –

```
var game = [0, 0, 0, 0, 0, 0, 0, 0, 0]
let winningCombos = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]]

for combination in winningCombos{
            if game[combination[0]] != 0 && game[combination[0]] == game[combination[1]] && game[combination[1]] == game[combination[2]]
```

The game variable has nine 0s that signal an empty board piece, if x takes a piece it will change to a 1, if o takes a piece it will change to a 2. This code checks for a winner by checking the first element in each array to make sure it's not 0 (`combination[0]] != 0`) and then we're checking to see if element 1 is equal to element 2 and if element 2 is equal to element 3 (`game[combination[0]] == game[combination[1]] && game[combination[1]] == game[combination[2]]`) if this is true then there is 3 numbers in a row that are all equal to each other and aren't 0 .

If it determines a winner, it will set the gameInPlay variable to false, so no other game pieces can be placed. It will then determine if x or o won, this is done by checking if the combination is the same as 1 or 2. Let's say X won, it would set one of my image views to say x has won and add 1 to the score of x, it will also hide and unhide certain image views and labels.

```
if game[combination[0]] == 1{
                    xPlayer.isHidden = true
                    oPlayer.isHidden = true
                    winner.image = UIImage(named: "xWins.png")
                    xScore += 1
                    xPlayer.text = "X's Score = " + String(xScore)
                }
                else{
                    xPlayer.isHidden = true
                    oPlayer.isHidden = true
                    winner.image = UIImage(named: "oWins.png")
                    oScore += 1
                    oPlayer.text = "O's Score = " + String(oScore)
                }
                PlayAgain.isHidden = false
                winner.isHidden = false
        }
```

During the game when a winner is being checked it also checks for a draw, it does this by going through the game variable and checking if there are any 0's left, if not it's a draw and does a very similar thing to a winner.

Once a winner or a draw has been determined a play button will be become unhidden, the play again button will reset the scoreboard, switch all the image views back to empty and hide/unhide all the relevant buttons, image views and labels.

UIImageViews as buttons

First, for the player vs player code, we have the image views that I have made to have the same functionality as a button. I have done this by adding a tap gesture recognizer that targets the image view –

```
let tap1 = UITapGestureRecognizer(target: self, action:
#selector(Bpressed1))
Button1.addGestureRecognizer(tap1)
Button1.isUserInteractionEnabled = true
```

When the Image View is tapped it calls a function that checks if that piece has already been used and if the game is still in play if it is it will either put an X image or an O image, let the game know that it's now occupied and then play a sound –

```
@objc func Bpressed1(){
        if (game[0] == 0 && gameInPlay == true){
            if activePlayer == 1{
                Button1.image = UIImage(named: "cross.png")
                game[0] = 1
                xSoundEffect.play()
                activePlayer = 2
            }
            else{
                Button1.image = UIImage(named: "nought.png")
                game[0] = 2
                oSoundEffect.play()
                activePlayer = 1
            }
            checkWinner()
        }
 }
```

Sounds

Sound has also been implemented into the game, I have chosen to add sounds when a piece has been played and when the game board is cleared. The sound has been implemented with the following code –

```
let xMusicFile = Bundle.main.path(forResource: "Cross", ofType:
"mp3")
do{
      try xSoundEffect = AVAudioPlayer(contentsOf: NSURL
(fileURLWithPath: xMusicFile!) as URL)
        }
        catch{
            print(error)
        }
```

UI status bar colour changed

As the background I have chosen to use is quite dark I was having issues seeing the status bar therefor I decided it was a good idea to change it from black to white by using the following code, this has been done on each of the screens –

```
    override var preferredStatusBarStyle:
UIStatusBarStyle {
        return .lightContent
    }
```

Hard AI

Moving onto the player vs computer code it is very similar in a lot of ways such as I have used the same way to check for winners, I have used the same sounds and I have mostly used the same buttons, labels and image views. The main difference is that instead of image views for the game pieces I have used buttons and I have added an AI that will play as O.

The main code for this is –

```
if (game[sender.tag-1] == 0 && gameInPlay == true){

            game[sender.tag-1] = activePlayer

            if (activePlayer == 1){
                sender.setImage(UIImage(named: "cross.png"), for:
.normal)
                xSoundEffect.play()
                activePlayer = 2
                checkForWinner()
            }

            if gameInPlay == true{
                AITurn()
            }
            checkForWinner()
            activePlayer = 1
        }

    }
```

The human will play as X and the computer as O, the code is quite like that of the P vs P game but much more concise since I'm using buttons, so the player will take their turn and then the computer will take their turn if the game is still in play. The computers go is just a list of if statements checking for the best possible move, here is a small snippet of it choosing the best move –

```
//Play sides
        if game[1] == 0 {
            currentMove = 2
        }
        if game[3] == 0 {
            currentMove = 4
        }
        if game[5] == 0 {
            currentMove = 6
        }
        if game[7] == 0 {
```

```
            currentMove = 8
        }

        //Play corners
        if game[0] == 0 {
            currentMove = 1
        }
        if game[2] == 0 {
            currentMove = 3
        }
        if game[6] == 0 {
            currentMove = 7
        }
        if game[8] == 0 {
            currentMove = 9
        }

        //Play center
        if game[4] == 0 {
            currentMove = 5
        }
```

The best possible move for it out of all this code is for it to play in the centrepiece. Therefore if the centrepiece is empty it will go there. Basically it will go down through all of the if statements and if it is true then it will set the current move variable to that number, so the least desirable move is at the top and the most desirable is at the bottom so as it goes down the rows if statements its move is getting better and better until it reaches the end at which point it has found its most desirable position at which point it will claim that square –

```
if game.contains(0){
            game[currentMove-1] = 2
            oSoundEffect.play()
        }
        if currentMove == 1{
            Button1.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 2{
            Button2.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 3{
            Button3.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 4{
```

```
            Button4.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 5{
            Button5.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 6{
            Button6.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 7{
            Button7.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 8{
            Button8.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
        if currentMove == 9{
            Button9.setImage(UIImage(named: "nought.png"), for:
.normal)
        }
```

This code checks to see if it can play and if it can it will play the sound and set the game position to 2 and then change the button image to an O.

PDF Reader

I have also included a pdf reader that includes some rules about ticktacktoe, I used the code you provided in lecture 10 and a view within the view controller to display it so I would still have access to the rest of the page like the back button.

```
// create and add the PDF view
        pdfView = PDFView()
        pdfView.translatesAutoresizingMaskIntoConstraints
= false
        view.addSubview(pdfView)

        // make it take up the full screen
        pdfView.leadingAnchor.constraint(equalTo:
view.leadingAnchor).isActive = true
        pdfView.trailingAnchor.constraint(equalTo:
view.trailingAnchor).isActive = true
        pdfView.topAnchor.constraint(equalTo:
view.topAnchor).isActive = true
```

```
        pdfView.bottomAnchor.constraint(equalTo:
view.bottomAnchor).isActive = true


        // load our example PDF and make it display
immediately
        let url = Bundle.main.url(forResource: "Rules",
withExtension: "pdf")!
        pdfView.document = PDFDocument(url: url)
```