# Game Base

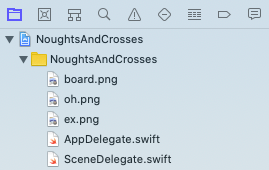
The coursework asks you to extend a basic noughts and crosses game developed in tutorials. The coursework gives you some ideas for how you might extend it but gives you a lot of flexibility.

In this workbook you will develop the basic working game. At the end are a couple of initial suggestions as to how it could be extended.

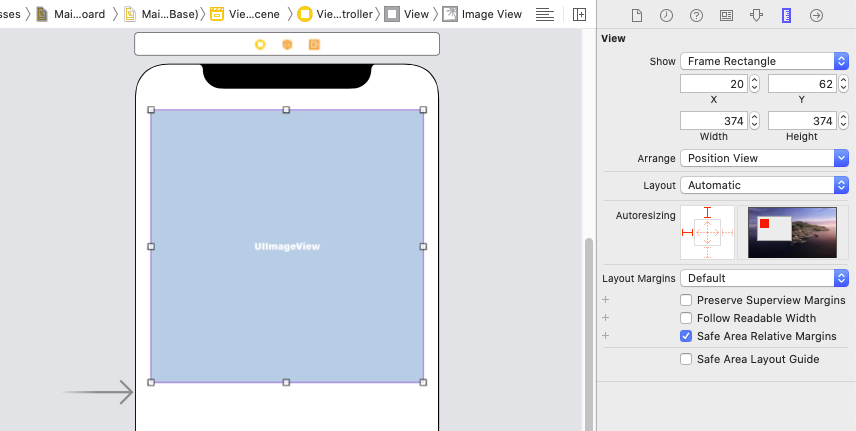
## Developing the Game Base

First, we need to create the image resources necessary for the game – a board, a X and a O.

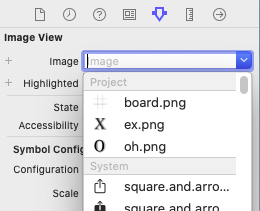
1. Use tables in Word or something similar to create a three-by-three board, a X and a O.
2. Screen grab these to create the images – shift-cmd-4 will do this on OSX, or Snip & Sketch on Windows.
3. Create a new Single View Application
4. Using either File > Add Files to… or drag and drop, add the image files to the project

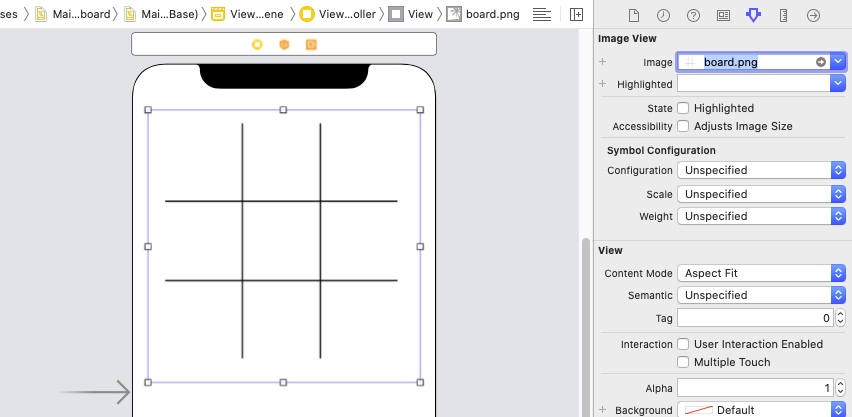


1. Add an Image View to the storyboard, make it square using the Size Inspector

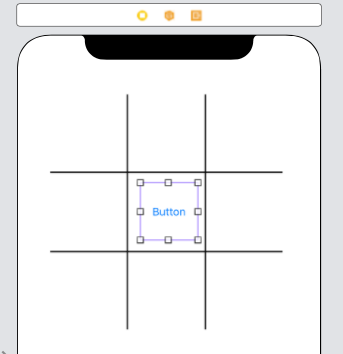


1. Click on the Image View
2. Select the Attributes Inspector
3. Change the Image to be the board image

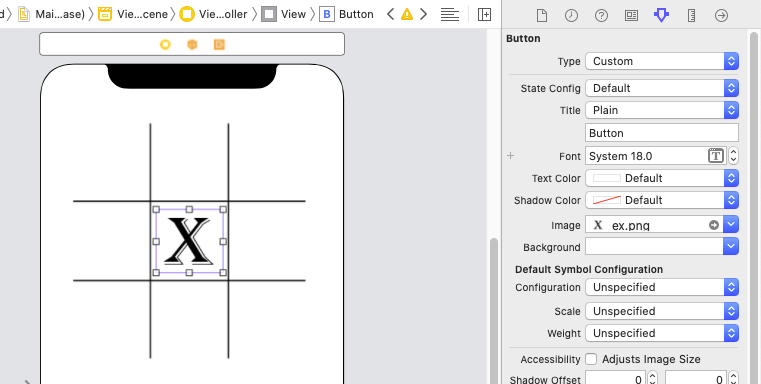




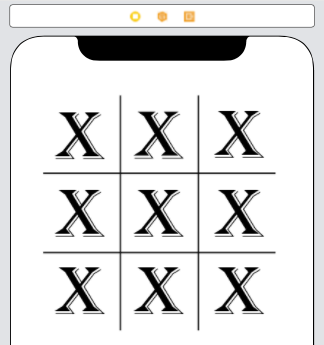
1. Add a Button to the middle square of the board grid and resize it so it fits nicely



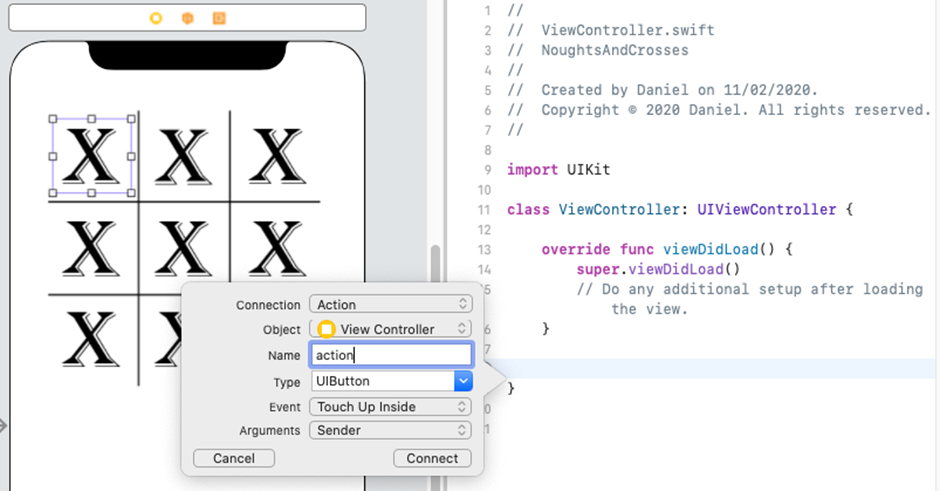
1. Click on the Button
2. Select the Attributes Inspector
3. Change the Image to be the X image



1. Copy and paste the button into the other squares of the board



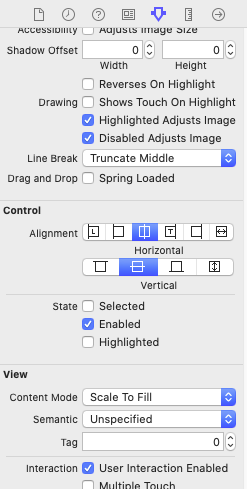
1. Click on one Button
2. Drag an Action to the ViewController class in the Swift code
3. Name it action, make its Type UIButton



1. Connect all the other buttons to the same Action – drag the connection to the same piece of code. When any of the buttons is clicked, the same function will be executed.

In order to distinguish the individual buttons when the function is executed, we need to set a Tag.

1. Click on the top-left button.
2. Select the Attributes Inspector.
3. Scroll down the Attributes Inspector until the Tag field becomes visible



1. Set the Tag to 1
2. Repeat steps 27-31 for each of the buttons, incrementing the Tag value by 1 each time. The last button will be tagged 9.

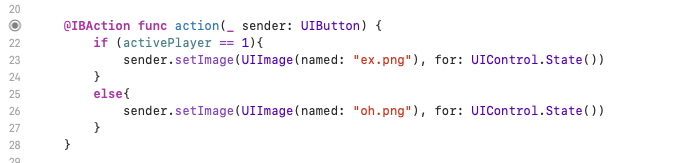
|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

We now need to add some code to track which players turn it is and to change the button to show the appropriate image, player 1 is Xs and player 2 is Os.

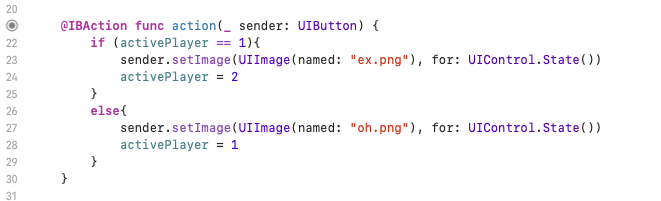
1. Add a variable to the ViewController class to hold the currently active player



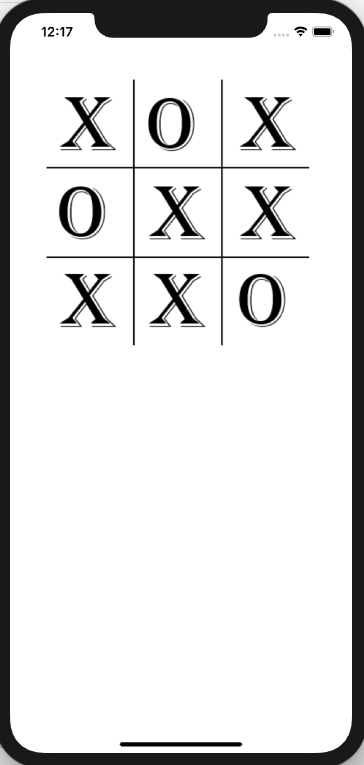
1. Add code to swap the button image depending on which player is active. Make sure the file names are the names of your image files!



1. Add the following additional code (lines 24 and 28) to the action function to swap active player

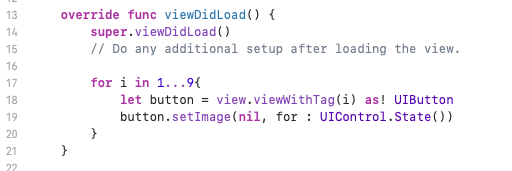


1. Run the simulator – we will find that when we click on the buttons we get an X or a O depending on whose turn it is.



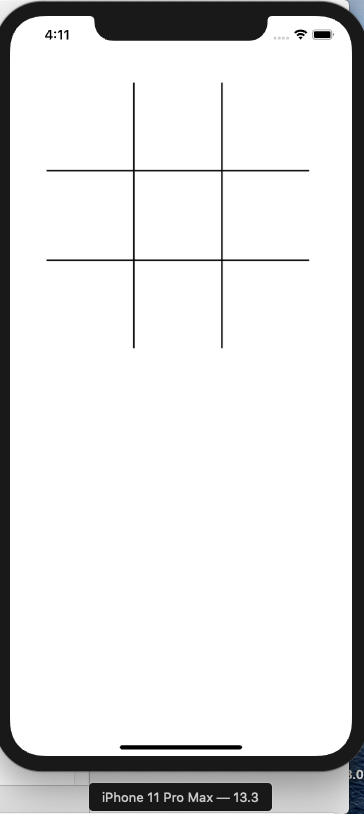
However, we obviously don't want the board to start full of Xs.

1. Add the following code to the viewDidLoad function, to empty the board when the view is first loaded



Note that we are using the tag value to refer to each individual button in turn

1. Run the simulator – we have an empty board, we can take turns, but we can still overwrite our opponent's pieces.



1. We need a data structure to represent to board, so we can track moves, add an array

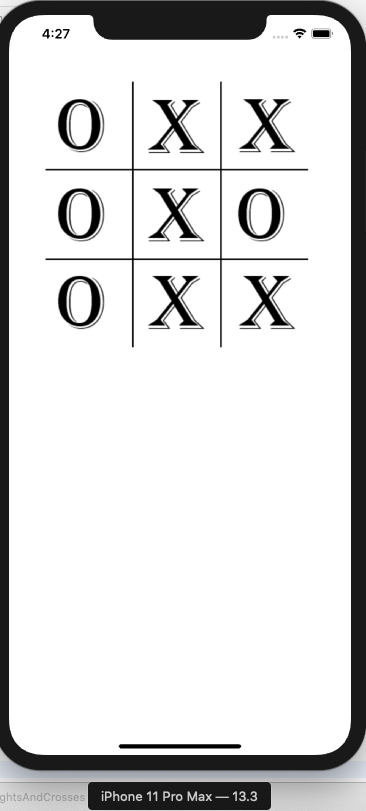


Each element in the array represents a square of the board. Each time a player takes a turn we need to record their move in the gameState array. Later we will use this array to check if anyone has won, so we need to record which player it was that took the turn

1. Add the following additional code (lines 30 and 32) to the action function



1. We are first checking (line 30) of the square the player has clicked on is empty (i.e. == 0), in which case it is a legal move. If it is, then we update the gameState to show that the player has a piece in that square (line 32), then, as before, set the appropriate image for that square and swap the active player.
2. Run the simulator, we can no longer overwrite our opponent's pieces. However, the game cannot determine if someone has won, we just keep playing until the board is full.



In order to check to see if someone has won, we first need a representation of the winning game states and then we just need to check if one of them has occurred each time a player takes a move.

The gameState variable represents the current state of the board. It has an item for each square (associated by tag number). Each item can be either 0 – empty; 1 – X; or 2 – O. So, if the gameState is [1, 2, 1, 2, 2, 1, 2, 1, 1], this represents

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 1 |
| 2 | 2 | 1 |
| 2 | 1 | 1 |

Or

|  |  |  |
| --- | --- | --- |
| X | O | X |
| O | O | X |
| O | X | X |

In this case X has won, and the winning squares (tag numbers) are 3, 6 and 9. So [3, 6, 9] is a representation of one possible winning state. We can enumerate all the winning states in this way and then check each time a player takes a move to see if a winning state is present in the game state.

1. Add the following data structure in the ViewController class (note we can use a constant because the winning states will never change)



1. We will also add another variable to track whether the game is still active, i.e. no one has won yet. Add the following to the ViewController class



1. Now we will add the code to check for the win. Add the following code (lines 47 to 57) to the action function. It needs to be included within our initial if statement (that checks if a move is legal)



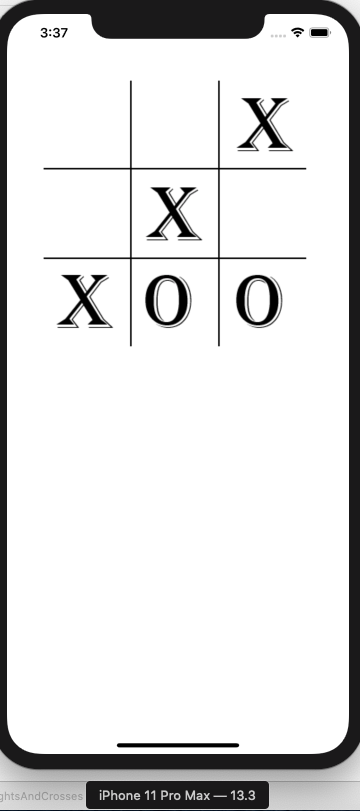
So, looking at the code, we are taking each winning state in turn. We are checking if the first board square in the combination is not empty, then checking to see if all three squares in the combination have the same (non-zero) number in them (i.e. one of the players has a piece in all three of the squares required for that winning combination.

If someone has won, we are setting the gameActive state to false so that no further moves can be made. However, we are not checking anywhere to see if the game is active. We need to check is the game is active before we check is a move is valid.

1. Add the following code to the action function. We are wrapping the whole of the code in a condition (line 34) that checks if the game is active.



1. Run the simulator – have a game of Os and Xs!



## Extending the Game Base

We now have a very basic, but playable game. This is the basic platform you need to develop further for the coursework. The coursework contains several suggestions for further development.

Here are a couple of simple ideas to get you started:

Add a single label which says whose turn it is next, or who has won, or that it is a draw (you will need to check for a draw).

Add a reset button that is hidden until the game is won or drawn. Hint: buttons have an isHidden property which takes a Boolean value, for example resetButton.isHidden = true

Add a best of three games or best of five games functionality.