Lesson 1: Creating a Board

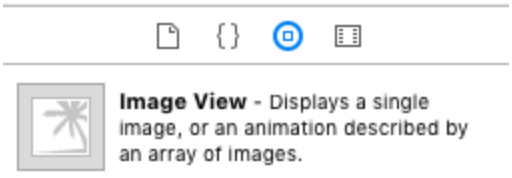
1-1 Welcome to XCode

1) Open up XCode.
2) Create a new Xcode Project
3) Choose Single View Application
4) Enter a Product Name
    a) Type in “WahiawaMiddleSchool” for the Organization
    b) Double Check that the Language is in “Swift”
    c) Double Check that the Devices says “Universal”
5) Save your Project under:
    a) GoogleDrive
    b) iOS Apps 2016-2017
    c) <your folder name> (Make one if you don’t have one yet)

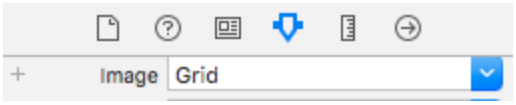
1-2 Add Images to Asset Folder

1) On the Mac, find the folder called TicTacToe Images (it should be under the Google Drive Folder)
2) Drag your 3 images into the Assets Folder

1-3 Add Grid



1) Click on Main.storyboard (on the left)
2) Search for Image View in Object Library (Bottom Right Corner)
    a) Drag onto screen
    b) Resize
3) Go to the Attributes Inspector at the top right. (You might have to open “Utilities”.
    a) Set the image to “Grid”



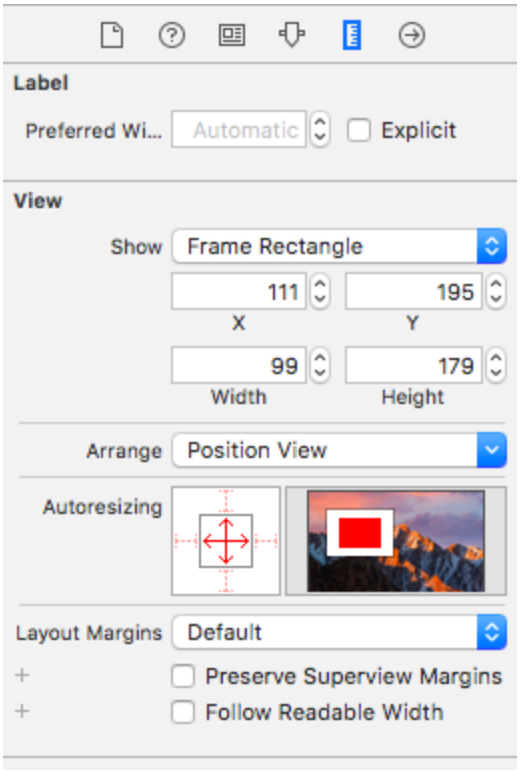
4) *Simulate! You should be able to see the Grid!*

What does this do?
You should be able to save your project in the Google Drive Folder.
You should be able to add images to your Asset Folders. (If you want more images, simply drag more.)
You should be able to create an image on the screen.

Lesson 1a: Constraints

1a-1)
http://www.appcoda.com/auto-layout-guide/
https://developer.apple.com/videos/play/wwdc2016/222/ (Must use Safari)

What's New in Auto Layout
https://developer.apple.com/videos/play/wwdc2016/236/ (Must use Safari)



There are 6 choices: Pin-Up, Pin-Left, Pin-Right, Pin-Down, Scale Up/Down, Scale Left/Right.
Click on Scale Up/Down and Scale Left/Right for each object.
Click on the appropriate Pin buttons. For example, for the upper left object, click on Pin-Up and Pin-Left, whereas for the bottom middle object, click on Pin-Down.

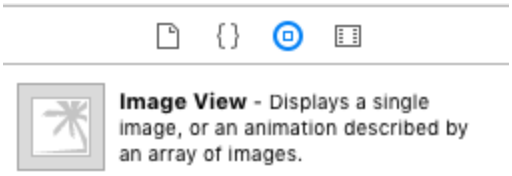
Simulate using different screen sizes. Flip the screen. Do the images scale properly?

What does this do?
You should be able to automatically scale the images to different screen sizes.

Lesson 2a: Adding Image Views, Buttons, and Labels

2-1 Add UI Image View

1) Click on Main.storyboard (on the left)
2) Search for UI Image View in Object Library (Bottom Right Corner)
3) Drag a UI Image View into upper left square
4) Resize UI Image View to fit into square
5) Make 8 More UI Image Views
    a) Either Add each UI Image View Individually
    b) Or, Copy and Paste
6) Make Sure all 9 UI Image Views are there and in order.
7) Simulate. Nothing new should have happened. (You can try and change one of these Image-Views to x-image to see what it'll look like.)



2-2 Add Buttons

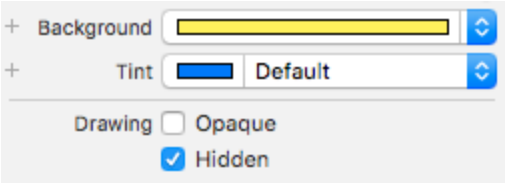
1) Click on Main.storyboard (on the left)
2) Search for Button in Object Library (Bottom Right Corner)
3) Drag a Button into upper left square
4) Resize Button to fit into square
5) Go to Attributes and Erase the Title
6) Make 8 More Buttons
    a) Either Add each Button Individually
    b) Or, Copy and Paste
7) Make Sure All 9 Buttons are there and in order. (Rename?)
8) *Simulate. You should see 9 buttons.*

What does this do?
You should now have 9 place-markers for 9 images. This is where the x and o-images will go.
You should also have 9 touch buttons for your phone to process. Later on, if you touch a button, that square should show an image.

Lesson 2b: Adding Image Views, Buttons, and Labels

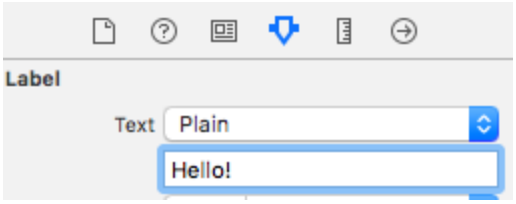
2-3 Add a Reset Button

1) Click on Main.storyboard (on the left)
2) Search for Button in Object Library (Bottom Right Corner)
3) Drag a Button into the middle-ish
4) Resize it a little
5) Rename the Title: “New Game”
6) Under Attributes (Looks like cursor pointing down), Scroll Down until you find Background
    a) Change the Background Color
7) *Simulate. You should see a beautiful New Game Button.*
8) Scroll down and click on the “Hidden” Checkbox
9) *Simulate. The New Game Button should now be hidden.*



2-4 Add a Label

1) Click on Main.storyboard (on the left)
2) Search for Label in Object Library (Bottom Right Corner)
3) Drag a Label into the upper-ish
4) Resize it to fill the screen left and right
5) Center the Alignment
6) Change the Color of the Background
    a) Change the Color of the Text (to white)
7) *Simulate. There should be a Label that says “Label”.*
8) Go to the Attributes Inspector and rename Label to “Hello!”
9) *Simulate. There should be a Label that says “Hello!”*

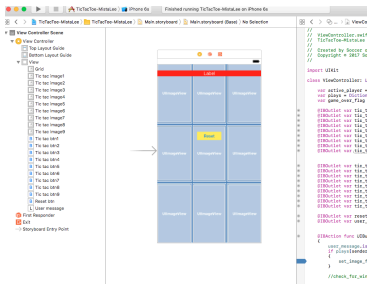


What does this do?

You can customize buttons for color, text, etc. You can even hide them if you need, then bring them back when you want.

You can also create labels around the frame to give messages to your user. These messages can be changed later on.

Lesson 3: Inserting Swift Code (Getting Harder)



3-1 Insert UI Image View into ViewController.swift code

1) Close Left (Navigator) and Right (Utilities). Should have three windows open:
    a) View Controller Scene
    b) Storyboard
    c) ViewController.swift
2) Control Drag the first Image View into code (into the Class. . .)
3) Rename it tic_tac_image1
    a) Select Strong
4) *Simulate. Nothing New should happen.*
5) Do the same for each of the other 8 UI Image Views
6) You can check at the end by moving your cursor over the grey circles to see which ones they are connected to
7) *Simulate. Nothing New should happen.*

```
@IBOutlet var tic_tac_image1: UIImageView!
@IBOutlet var tic_tac_image2: UIImageView!
@IBOutlet var tic_tac_image3: UIImageView!
@IBOutlet var tic_tac_image4: UIImageView!
@IBOutlet var tic_tac_image5: UIImageView!
@IBOutlet var tic_tac_image6: UIImageView!
@IBOutlet var tic_tac_image7: UIImageView!
@IBOutlet var tic_tac_image8: UIImageView!
@IBOutlet var tic_tac_image9: UIImageView!
```

3-2 Insert Buttons into ViewController.swift code

1) Close Left (Navigator) and Right (Utilities). Should have three windows open:
    a) View Controller Scene
    b) Storyboard
    c) ViewController.swift
2) Control Drag the first Button into code (into the Class. . .)
    a) Choose Outlet
    b) Rename it tic_tac_btn1
    c) Select Strong
3) *Simulate. Nothing New should happen. You might be able to "click" on a button, but no action yet.*
4) Do the same for each of the other 8 Buttons
5) You can check at the end by moving your cursor over the grey circles to see which ones they are connected to
6) *Simulate. Nothing New should happen.*
7) For each button, change the tag to 1-9, respectively

```
@IBOutlet var tic_tac_btn1: UIButton!
@IBOutlet var tic_tac_btn2: UIButton!
@IBOutlet var tic_tac_btn3: UIButton!
@IBOutlet var tic_tac_btn4: UIButton!
@IBOutlet var tic_tac_btn5: UIButton!
@IBOutlet var tic_tac_btn6: UIButton!
@IBOutlet var tic_tac_btn7: UIButton!
@IBOutlet var tic_tac_btn8: UIButton!
@IBOutlet var tic_tac_btn9: UIButton!
```

3-3 Insert New Game Button and Label into ViewController.swift code

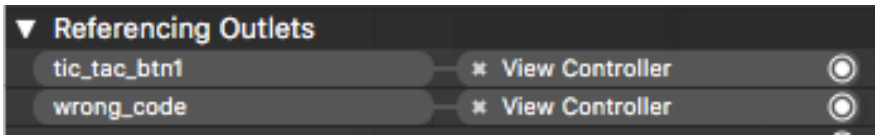
1) Control Drag New Game Button into code: name it new_game_btn
2) Control Drag Label into Code: name it User_message

```
@IBOutlet var new_game_btn: UIButton!
@IBOutlet var user_message: UILabel!
```

What does this do?

This connects your storyboard with your ViewController code. It's tricky at first, but it is really powerful because it lets you create the objects in your storyboard, and then link each object with pieces of code that you'll write.

Note!!! If you Control Drag an outlet, but made a mistake, you can’t just delete the code. Go back to View Controller Scene and see if you have two Outlets. Right-click each button in the View Controller Scene. If it has the wrong outlet, click the x / delete.



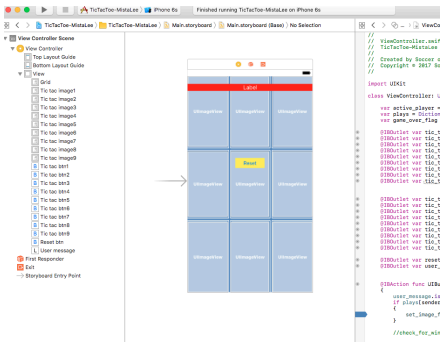
Lesson 4a: Creating variables and Dictionaries

4-1 Create plays Dictionary and game_over_flag

1) Towards the top under import UIKit
2) var active_player = 1 // Player X
3) var plays = Dictionary <Int,Int>()
4) var game_over_flag = false

```
import UIKit

var active_player = 1 //player X
var plays = Dictionary <Int,Int>()
var game_over_flag = false
```



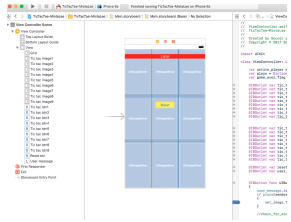
What does this do?

Variables are values that can change (or vary) as your program does stuff. For example, our variable active_player is first set to 1, so it will be player X's turn. After she clicks, the variable active_player should change to 2, making it player O's turn. This should be reset to 1 and back to 2 each time somebody makes a move. Similarly, the game_over_flag keeps track if the game is still on (game_over_flag = false → moves are possible) or if the game is off (game_over_flage = true → moves are no longer possible).

Dictionaries are more complicated collections of values that have a key and a value <key:value>. For example, if I have a dictionary called poke_dict = [5: "apples", 8: "oranges", 2: "mangos"], I can then get mangos by checking poke_dict[2]

Plays is a dictionary that keeps track of each button (sender) and its tag. For the time being, our plays dictionary is empty. Remember how we set each button to be a tag, from 1-9? (The first button's tag is 1, the second button's tag is 2, etc.) We'll use the plays dictionary to keep track of each button's value, whether it is player 1 or player 2. In the beginning the dictionary is empty because nobody has played yet. Let's say player 1 chooses the middle square (tag 5). The dictionary should now become plays = [5: 1]. Let's say player 2 choose the top-left square. The dictionary should now become playes = [5:1, 1:2]. The dictionary gets bigger and bigger each time a player chooses a square.

Lesson 4b: Creating UIButtonClicked Action (Hard!!!)



4-2 Create UIButtonClicked Action

1) Close Left and Right. Should have three windows open:
    a) View Controller Scene
    b) Storyboard
    c) ViewController.swift
2) Control Drag the first Button into code (into the class ViewController)
    a) (Remember, all functions will go into the class ViewController)
    b) Choose Action
    c) Name it UIButtonClicked
    d) Click on UIButton
    e) It should read @IB>,Action func UIButtonClicked(_ sender: UIButton)
    f) Simulate. Nothing New should happen.
    g) Add the code user_message.text = “Ouch” into your action.
    h) Simulate. If you click the first Button, the label should now say “Ouch”

```
class ViewController: UIViewController {

    @IBAction func UIButtonClicked(_ sender: UIButton)
    {
    user_message.text = "Ouch"

    }
```

3) Link all 8 remaining buttons to the action
    a) Drag click each button into the action UIButtonClicked function box
    b) (You don’t create new code. You are linking each button the code that is already written above.
    c) *Simulate. If you click any Button, the label should now say “Ouch”*

4) In the code
    a) In @IBAction func UIButtonClicked
    b) Slash//slash the user_message.text = “Ouch”
    c) Add the following code:

```
@IBAction func UIButtonClicked(_ sender: UIButton)
{
    //user_message.text = "Ouch"
    if plays[sender.tag] == nil && game_over_flag == false
    {
        user_message.text = "Ooomph"
    }
}
```

    d) Simulate. If you click any button, it should now say “Ooomph” instead of “Ouch”

What does this mean?
Now that you have buttons and linked them to your ViewController code, we want to make an action to determine what that button will do. Anything within the curly{} braces of the @IBAction func UIButtonClicked will happen.
In this case, we first made it say “Ouch.”
However, we want the button to have an action only if nobody has already played there. Thus, we check to see if the dictionary for that spot has a player. If it doesn’t have a player (1 or 2), then we say “Ooomph”

```
if plays[sender.tag] == nil
```

Similarly, we want it to happen only when the game is still on, so we check the game_over_flag

```
&& game_over_flag == false
```

Lesson 5a: Setting Images at Each of the 9 Spots (Error Codes Galore!!!)

1) Call check_for_win()
    a) In the @IBAction function UIButtonClicked. . .
    b) Call the function check_for_win()

```
class ViewController: UIViewController {

    @IBAction func UIButtonClicked(_ sender: UIButton)
    {
        //user_message.text = "Ouch"
        if plays[sender.tag] == nil && game_over_flag == false
        {
        user_message.text = "Ooomph"
        }
        check_for_win()

    }
```

    c)
    d) Simulate. It should fail because you don't have a check_for_win() function yet.
    e) Slash//slash it out.
2) Add set_image_for_spot(spot: sender.tag, player:1)
    a) In the @IBAction function UIButtonClicked

```
17 class ViewController: UIViewController {
18
19     @IBAction func UIButtonClicked(_ sender: UIButton)
20     {
21         //user_message.text = "Ouch"
22         if plays[sender.tag] == nil && game_over_flag == false
23         {
24             user_message.text = "Ooomph"
25             set_image_for_spot(spot:sender.tag, player:1)
26         }
27         //check_for_win()
28
29     }
```

    b) Simulate. Oops! Once again, it should fail because you don't have a function called set_image_for_spot() yet. Don't slash//slash it out.

What does this mean?
We now want the code to change the image for each box to the appropriate player. For the time being, we'll set all of them to player 1 (x-image).
It will check the value of the tag of the button (sender.tag) in the dictionary.
If there is no image there, it will either place an x-image (for player 1) or an o-image (for player 2)

Lesson 5b: Setting Images at Each of the 9 Spots (Super Hard!!!)

3) Create a function called set_image_for_spot()
    a) After your @IBAction func UIButtonClicked. . .

```
class ViewController: UIViewController {

    @IBAction func UIButtonClicked(_ sender: UIButton)
    {
        //user_message.text = "Ouch"
        if plays[sender.tag] == nil && game_over_flag == false
        {
            user_message.text = "Ooomph"
            set_image_for_spot(spot:sender.tag, player:1)
        }
        //check_for_win()

    }

    func set_image_for_spot(spot:Int, player:Int)
    {

    }
```

    b) *Simulate. It shouldn't have an error code any more.*

4) Determine if it is "x-image" (player 1) or "o-image" (player 2)

```
func set_image_for_spot(spot:Int, player:Int)
{
    var player_mark = ""
    if (player == 1)
    {
        player_mark = "x-image"
    }
    else if (player == 2)
    {
        player_mark = "o-image"
    }
    plays[spot] = player
}
```

    a) *Simulate. Nothing New should happen.*

What does this mean?
Remember our code in the button action?

```
set_image_for_spot(spot:sender.tag, player:1)
```

We want to set each of the 9 squares to match player 1, which is the x-image.
How does this work? Well, if the player==1, then we'll use "x-image" and if the player==2, we'll use "o-image"
We'll also change the plays dictionary for that spot to the appropriate player, so that it can keep track.

```
plays[spot] = player
```

Lesson 5c: Setting Images at Each of the 9 Spots (Super Super Hard!!! But oh-so fun!)

5) Continue programming the set_image_for_spot
   a) Create a switch to change the image of each of the 9 spots

```
        plays[spot] = player

        switch spot
        {
            case 1:
                tic_tac_image1.image = UIImage(named:player_mark)

        }
    }
```

   b) Simulate. It should have an error code.
   c) Add a default case.

```
        plays[spot] = player

        switch spot
        {
            case 1:
                tic_tac_image1.image = UIImage(named:player_mark)
            default:
                tic_tac_image5.image = UIImage(named:"Grid")
        }
    }
```

   d) Simulate. Click on the first button. It should change to x-image. What happens if you click on any other button? Why?

6) Do all 9 cases

```
        switch spot
        {
            case 1:
                tic_tac_image1.image = UIImage(named:player_mark)
            case 2:
                tic_tac_image2.image = UIImage(named:player_mark)
            case 3:
                tic_tac_image3.image = UIImage(named:player_mark)
            case 4:
                tic_tac_image4.image = UIImage(named:player_mark)
            case 5:
                tic_tac_image5.image = UIImage(named:player_mark)
            case 6:
                tic_tac_image6.image = UIImage(named:player_mark)
            case 7:
                tic_tac_image7.image = UIImage(named:player_mark)
            case 8:
                tic_tac_image8.image = UIImage(named:player_mark)
            case 9:
                tic_tac_image9.image = UIImage(named:player_mark)
            default:
                tic_tac_image5.image = UIImage(named:"Grid")
        }
```

   a) Simulate! Each time you click a spot, it should change to the x-image!

What does this mean?
A switch is like a series of if statements, checking each and every value for “spot”.
If spot == 1, then tic_tac_image1 will change to a new picture. If spot == 7, then tic_tac_image7 will change to a new picture.
Switches need a default case because it needs to check ALL possible values. It’s like an else statement. In our program, the default sets tic_tac_image5 to “Grid”. Once all 9 cases are programmed, there are no other numbers, so the default never gets accessed.

Lesson 6a: Check for Win!

1) (You can collapse a function by holding option-command-left_arrow
2) After your set_image_for_spot function, create a check_for_win() function

```
func set_image_for_spot(spot:Int, player:Int)
{...}

func check_for_win()
{

}
```

   a) *Simulate. Nothing New should happen.*

3) Un-slash//slash check_for_win() in your @IBAction function

```
@IBAction func UIButtonClicked(_ sender: UIButton)
{
    //user_message.text = "Ouch"
    if plays[sender.tag] == nil && game_over_flag == false
    {
        user_message.text = "Ooomph"
        set_image_for_spot(spot:sender.tag, player:1)
    }
    check_for_win()

}
```

4) Create a who_won dictionary

```
func check_for_win()
{
    let who_won = ["Player 1" : 1, "Player 2" : 2]
}
```

   a) Created a dictionary to determine if Player 1 or Player 2 won

   b) *Simulate. Nothing New should happen. (Will give you a yellow caution code.)*

5) Create a For Loop (Hard, but fun!!!!)

```
func check_for_win()
{
    let who_won = ["Player 1" : 1, "Player 2" : 2]

    for(key,value) in who_won
    {
        if ((plays[1]==value && plays[2]==value && plays[3]==value)) // top across
        {
            user_message.text = "Looks like \(key) won!"
            game_over_flag = true
        }
    }
}
```

   a) *Simulate!!! If you click all three squares along the top row, what happens?!?*

What does this mean?
We first created a dictionary called who_won. The keys of this dictionary are the names of the players, in this case, “Player 1” and “Player 2.” This matches with the values 1 and 2 respectively.
We then checked to see if the value of the top three squares match. Remember how we kept that information stored in the plays dictionary?
For example, if the plays = [1:1, 2:1, 3:1], then that means that the top row is X-X-X, so Player 1 has three in a row! In contrast, if the plays = [1:1, 2:1, 3:2], then that means that the top row is X-X-O, which is not a win.
If there is a winner, the user_message is changed to say who won. Recall that the \(key) is either “Player 1” or “Player 2,” as determined by our who_won dictionary.
Lastly, once somebody wins, the game_over_flag is set to true, meaning that no more buttons can be pushed.

Lesson 6b: Check for Win!

1) Repeat by adding more combinations to account for all different sets of three.
    a) Add a new set of three

```
func check_for_win()
{
    let who_won = ["Player 1" : 1, "Player 2" : 2]

    for(key,value) in who_won
    {
        if ((plays[1]==value && plays[2]==value && plays[3]==value)) // top across
        || ((plays[4]==value && plays[5]==value && plays[6]==value)) // middle across
        {
            user_message.text = "Looks like \(key) won!"
            game_over_flag = true
        }
    }
}
```

    b) || stands for “or”
    c) Simulate. The second row should now work!
    d) Can you figure out all 8 combinations?

```
func check_for_win()
{
    let who_won = ["Player 1" : 1, "Player 2" : 2]

    for(key,value) in who_won
    {
        if ((plays[1]==value && plays[2]==value && plays[3]==value)) // top across
        || ((plays[4]==value && plays[5]==value && plays[6]==value)) // middle across
        || ((plays[7]==value && plays[8]==value && plays[9]==value)) // bottom across
        || ((plays[1]==value && plays[4]==value && plays[7]==value)) // right down
        || ((plays[2]==value && plays[5]==value && plays[8]==value)) // middle down
        || ((plays[3]==value && plays[6]==value && plays[9]==value)) // left down
        || ((plays[1]==value && plays[5]==value && plays[9]==value)) // diagonal 1
        || ((plays[3]==value && plays[5]==value && plays[7]==value)) // diagonal 2
        {
            user_message.text = "Looks like \(key) won!"
            game_over_flag = true
        }
    }
}
```

    e) Simulate. Check all combinations!
    f) Change the set_image_for_spot_parameters

```
set_image_for_spot(spot:sender.tag, player:2)
```

    g) Simulate. What images shows up now? Who wins?

What does this mean?
This is the crux of the “game” that determines who wins. There are 9 squares, each one tagged 1-9. Each time a player chooses a square, the value of that square is saved in the plays dictionary. The check_for_win() function checks if sets of 3 are made.

Lesson 7a: Create a New Game Button / Action

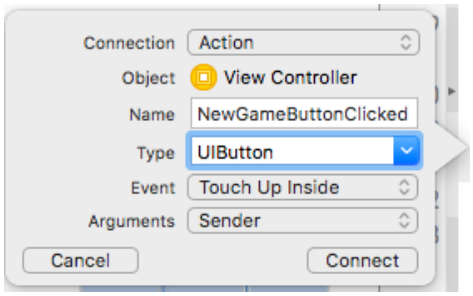
1) In check_for_win() function, unhide new_game_btn

```
func check_for_win()
{
    let who_won = ["Player 1" : 1, "Player 2" : 2]

    for(key,value) in who_won
    {
        if ((plays[1]==value && plays[2]==value && plays[3]==value)) // top across
        || ((plays[4]==value && plays[5]==value && plays[6]==value)) // middle across
        || ((plays[7]==value && plays[8]==value && plays[9]==value)) // bottom across
        || ((plays[1]==value && plays[4]==value && plays[7]==value)) // right down
        || ((plays[2]==value && plays[5]==value && plays[8]==value)) // middle down
        || ((plays[3]==value && plays[6]==value && plays[9]==value)) // left down
        || ((plays[1]==value && plays[5]==value && plays[9]==value)) // diagonal 1
        || ((plays[3]==value && plays[5]==value && plays[7]==value)) // diagonal 2
        {
            user_message.text = "Looks like \(key) won!"
            game_over_flag = true
            new_game_btn.isHidden = false
        }
    }
}
```

a) Simulate. After somebody wins, the New Game Button should appear, in all its glory!

2) Create a NewGameButtonClicked Action




```
@IBAction func UIButtonClicked(_ sender: UIButton)
{}

@IBAction func NewGameButtonClicked(_ sender: UIButton)
{
    new_game()
}
```

a) Control Drag the New Game Button into code (into the class ViewController, beneath the UIButtonClicked Action)
   i) Choose Action
   ii) Name it NewGameButtonClicked
   iii) Click on UIButton
   iv) It should read @IBAction func NewGameButtonClicked(_ sender: UIButton)

b) Add the code new_game() inside
   i) Each time the button is clicked, new_game() is called

c) Simulate. Should have an error code because there is no new_game() function.

What does this mean?
We want to have the New Game Button show up after somebody wins.
We want the button to have its own Action, which is different from the 9 squares Action.

Lesson 7b: Create a New Game Button / Action

3) Create a new function called new_game()

```
func check_for_win()
{...}

func new_game()
{

}
```

a) Simulate. No error code!!!

4) Reset the values so that a new game can start.

```
func new_game()
{
    game_over_flag = false
    new_game_btn.isHidden = true
    plays = [:]
    tic_tac_image1.image = nil
    tic_tac_image2.image = nil
    tic_tac_image3.image = nil
    tic_tac_image4.image = nil
    tic_tac_image5.image = nil
    tic_tac_image6.image = nil
    tic_tac_image7.image = nil
    tic_tac_image8.image = nil
    tic_tac_image9.image = nil
}
```

a) This resets the game_over_flag so we can push buttons
b) This hides the New Game Button
c) This clears the plays dictionary
d) This clears all images from the 9 squares
e) Simulate. Wow! The board clears, the New Game Button disappears, and you can play again!

What does this mean?
How do we make a new game? Well, we need to reset all values to their original values.

Lesson 8a: Create the 2nd Player

1) Remember our active_player variable at the top?

```
var active_player = 1 //player X
```

2) Let's make code that keeps track of whose turn it is and what should happen.
3) Recode your UIButtonClicked Action

```
@IBAction func UIButtonClicked(_ sender: UIButton)
{
    //user_message.text = "Ouch"
    if active_player == 1
    {
        if plays[sender.tag] == nil && game_over_flag == false
        {
            user_message.text = "Ooomph"
            set_image_for_spot(spot:sender.tag, player:1)
            active_player = 2
        }
    }

    check_for_win()

}
```

   a) *Simulate. This should make it so that you can only put 1 "x". Why? Because it is now Player 2's turn.*

4) Let's have the game notify us that it is Player 2's turn.

```
if active_player == 1
{
    if plays[sender.tag] == nil && game_over_flag == false
    {
        //user_message.text = "Ooomph"
        set_image_for_spot(spot:sender.tag, player:1)
        active_player = 2
        user_message.text = "Player 2's Turn!"
    }
}
```

   a) *Simulate. The label should now tell you it's Player 2's Turn.*

5) Add a new if statement for player_turn == 2

```
if active_player == 1
{...}
if active_player == 2
{
    if plays[sender.tag] == nil && game_over_flag == false
    {
        set_image_for_spot(spot:sender.tag, player:2)
        active_player = 1
        user_message.text = "Player 1's Turn!"
    }
}
```

   a) *Simulate. You should have your first game now! Don't you feel like a rock star?*
   b) *Simulate. What happens when you get a tie?*

What does this mean?
We use the variable active_player to keep track whose turn it is.
Then, depending of if it is Player 1's turn or Player 2's turn, we add the appropriate image.
We switch between Player 1 and Player 2
Lastly, we take advantage of the label to tell the User who should go next.

Lesson 8b: Create the 2nd Player

6) Modify the label to reflect whose turn it is when you start a New Game

```
func new_game()
{
    game_over_flag = false
    new_game_btn.isHidden = true
    plays = [:]
    tic_tac_image1.image = nil
    tic_tac_image2.image = nil
    tic_tac_image3.image = nil
    tic_tac_image4.image = nil
    tic_tac_image5.image = nil
    tic_tac_image6.image = nil
    tic_tac_image7.image = nil
    tic_tac_image8.image = nil
    tic_tac_image9.image = nil
    if active_player == 1
    {
        user_message.text = "Player 1's Turn"
    }
    if active_player == 2
    {
        user_message.text = "Player 2's Turn"
    }
}
```

a) *Simulate. When you press the New Game Button, it should tell you whose turn it is.*

What does this mean?
If statements are useful to change values based on other values. In this case, the user_message changes based on whose turn it is.

Lesson 9: Cat's Games

1) Create a check_for_tie() function

```
func check_for_win()
{...}

func check_for_tie()
{

}
```

   a) *Simulate. Nothing New should happen.*

2) Create a Cat's Game Scenario

```
func check_for_tie()
{
    if plays.count == 9
    {
        user_message.text = "Cat's Game. Meow"
        game_over_flag = true
        new_game_btn.isHidden = false
    }
}
```

   a) *Simulate. Try to get a Cat's Game. Still Nothing New should happen.*

3) Call this code right before the check_for_win() call

```
@IBAction func UIButtonClicked(_ sender: UIButton)
{
    //user_message.text = "Ouch"
    if active_player == 1
    {...}
    if active_player == 2
    {...}
    check_for_tie()
    check_for_win()

}
```

   a) *Simulate. Try to get a Cat's Game. Voila!*

What does this mean?
Remember the dictionary called plays? It keeps track of all 9 squares. If all 9 squares have been played, and there is now winner, then it is a Cat's Game. Can you figure out why we check_for_tie() before we check_for_win()? Try flipping the order and seeing what happens?

Lesson 10: Extras / Extensions

1) Can you figure out how to make the word “button” disappear?
2) Add Points (score1 and score2, with a label/message to show it after each game)
3) Randomize who starts
4) Change the pictures
5) Make Sounds
6) Add Names
7) Add lines to show the winning combination.
8) Add AI (Yikes! Don’t know how to do this myself. . . .yet. . .)

Next Games:
Rock Paper Scissors
Guess the Number
Choose your own adventure
Who goes first?