# Button-Slot App (<30 minutes)

Create a new project and select a “single view application”

* Product Name: BabyGame
* Organization name: eurecom
* Organization identifier: fr.eurecom
* Device: iPhone
* Language: objective-c and swift
* UI: Storyboard

Select the location, and then create the project.

The project should already compile and show a white screen with the top navigation bar.

## Import the resources to the project

Download the required images from the following link: <https://my.eurecom.fr/jcms/p0_2031336/fr/ios-lab2-images>

Right click on the “**BabyGame folder”** inside the BabyGame project, select “**Add Files to BabyGame**”, and then select the “**iOS\_lab2\_images**” files and extend the options and check “**copy items if needed**”. Save the project.



## Build the Storyboard

This is the file that is going to contain all the views of the application and how they interact one to each other. On the left-hand side, we can see that for this particular “View Controller” we have already a “View” associated to it. This is the main view of the application and where we will be adding the controls for this project.

## Add Objects to the view

Open the MainStoryboard file and Drag & Drop a “Button” objects to the view and a “ImageView” object as shown in the figure.

In the newest version of Xcode the *Object Library* is located at the top of the page and referenced with a “+” button.



Center both objects with the use of constraints – sometimes constraints are not enough considering that you are modifying even the frame of the object, so you may still receive a warning from Xcode.

**Note:** If you encounter problems moving the object within the view, then select the object in the view controller scene and place it where you want in the view.

* <http://stackoverflow.com/questions/8875543/why-cant-i-move-ios-storyboard-elements-around>

## Add the IBOutlets

We are going to need to access to those objects from the source code, so we need to create two IBOutlets to associate to them.

**For objective-C:** Open the file ViewController.h and add the following lines of code:

@interface ViewController : UIViewController {

// Declaration of the IBOutlets

IBOutlet UIImageView \* slot;

IBOutlet UIButton \* button;

}

// Properties for the members

@property(nonatomic, retain) UIImageView \* slot;

@property(nonatomic, retain) UIButton \* button;

@end

You will notice a little warning sign appearing in the ViewController.m. By clicking the mouse over the warning sign XCode gives you a tooltip with the explanation of the warning. (The same happens when there is an error.)

In this case the warning is telling us that we declare properties without synthesize the variables. To **synthesize** them we need to open the file ViewController.m and add the following line at the beginning of the @implementation:

**For Swift**, open the ViewController.swift, and add the following code inside the class

@synthesize button, slot;

@IBOutlet weak var slot: UIImageView!

@IBOutlet weak var button: UIButton!

Now, we need **to associate the IBOutlets with Objects in the view.**

**Note that you can even do *ctrl + drag* straight from the object you just created onto the code, so that the creation of the *IBOutlet* is easier and straight forward.**

Open again the Main.Storyboard file and select the “View Controller”. Also open the connection inspector (left menu, with arrow). As the associated file to that controller is the file we just modified the “Connection Inspector” should show us the two IBOutlets we just add in the code, i.e. slot and button.



Drag & Drop the little circle at the right of “button” and “slot” to the corresponding object in the view. The Connection Inspector should start showing the connection in that way:



From now-on every time we modify “button” and “slot” within the code, we will be modifying the interface, and thus the view.

## Create an Action

An action is a function that will be called by the interface when an event happens (a button has been clicked, the finger is touching the screen, etc...).

**Objective-C :** Actions can have any of those three declaration structures:

-(IBAction) actionName;

-(IBAction) actionName:(id)sender;

-(IBAction) actionName:(id)sender withEvent:(UIEvent \*)event;

We will use one or the other depending on if we are interested on knowing who is the event sender (in the case the same action manages different objects for example) or we need details of the event (in the case we need to know the position of the finger for example)

Open the ViewController.h file and add the following action declaration after the @property declarations:

-(IBAction) buttonPressed;

-(IBAction) buttonMoved: (id)sender withEvent:(UIEvent \*)event;

-(IBAction) buttonReleased;

**For Swift**, open the ViewController.swift file and add the following functions.

@IBAction func buttonPressed () {}

@IBAction func buttonMoved(sender: UIButton, forEvent event: UIEvent) {}

@IBAction func buttonReleased() {}

## Association of an Action to the signal of the objects (Target-Action)

Open again the Main.Storyboard file and select the UIButton object. The “Connection Inspector” should show now a list of outlets and actions that can be connected. The interesting actions in our case are:

* **Touch Down, Touch Drag Inside and Touch Up Inside.**

To link those event actions with the corresponding coded action, we will drag & drop the little circle on the right-hand side of each action to the View Controller icon, i.e. the button.



Once the link dropped a popup menu will show the available actions in the ViewController, we will link the events to the following actions:

* **Touch Down -> buttonPressed**
* **Touch Drag Inside -> buttonMoved**
* **Touch Up Inside -> buttonReleased**

The Connection Inspector should show the following:

## Coding the actions

Here is what the actions are supposed to do:

1. When we press the button, we will store its current position, to be able to come back if we need to.
2. When we are dragging the finger, we will update the position of the button with the current finger position
3. When we release the button, we will check how close we are to the slot. If we are close enough, we will leave the button there, otherwise the button will go back to its original position.

### buttonPressed

Let’s start by adding the member needed to store the original button position.

**For Objective-C :** add that line of code in the ViewController.h file with the other member declarations:

Now we add at the end of the file ViewController.m the following code for the **buttonPressed** action:

CGPoint originalPosition;

since there is only one button in our view, there is no ambiguity. However, if there were more buttons, we need to include the sender in the function declaration, i.e

-(IBAction) buttonPressed {

originalPosition = button.center;

}

-(IBAction) actionName:(id)sender;

**For Swift,** add the following as a variable declaration in the ViewController.swift

var originalPosition: CGPoint!

and the following for the function

@IBAction func buttonPressed () {

originalPosition = button.center

}

### buttonMoved

The dragging action is a bit more difficult as we need to extract the finger position from the UIEvent.

**For Objective-C:** Add the following code after the **buttonMoved**:

This function gets all the touches and for each one of them calculates the coordinates in the main view and sets them to the button.

-(IBAction) buttonMoved: (id)sender withEvent:(UIEvent \*)event {

for(UITouch \* touch in [event allTouches]) {

button.center = [touch locationInView:self.view];

}

}

**For Swift,** add the following code to ViewController.swift

@IBAction func buttonMoved(sender: UIButton, forEvent event: UIEvent) {

let touches = event.allTouches

for touch in touches!

{

button.center = touch.location(in: self.viewIfLoaded)

}

}

### buttonReleased

The last action will have the following code when **buttonReleased**. The first function needs to calculate the distance in between two points and is then used to define the behavior in **buttonReleased.**

**For Objective-c:**

-(CGFloat) distanceBetweenPoint:(CGPoint) point1 andPoint:(CGPoint)point2

{

CGFloat dx = point2.x - point1.x;

CGFloat dy = point2.y - point1.y;

return sqrt(dx\*dx + dy\*dy );

}

-(IBAction) buttonReleased {

if ([self distanceBetweenPoint:button.center andPoint:slot.center] < 100)

button.center = slot.center;

else

button.center = originalPosition;

}

make the function, *distanceBetweenPoint*, private.

**For Swift,**

@IBAction func buttonReleased() {

if(distanceBetweenPoint(point1: button.center, point2: slot.center) < 100) {

button.center = slot.center

}

else{

button.center = originalPosition

}

}

private func distanceBetweenPoint(point1:CGPoint, point2:CGPoint) ->CGFloat {

let dx: CGFloat = point2.x - point1.x

let dy: CGFloat = point2.y - point1.y

return sqrt(dx\*dx+dy\*dy)

}

## Beautify the Interface

Go to the Main.Storyboard for the last time.

### Add the background

If you select the “View” you will see in the Attribute Inspector that it has an attribute background and that this can be either a color or a texture. We can customize the displayed texture directly from the source code.

**For Objective-C:**

Open the file “ViewController.m” and add the following code at the end of viewDidLoad:

UIColor \*background = [[UIColor alloc] initWithPatternImage:

[UIImage imageNamed:@"GameBackground.png"]];

self.view.backgroundColor = background;

**For Swift:**

let image = UIImage(named: "GameBackground.png")

let background = UIColor.init(patternImage: image!)

self.view.backgroundColor = background

### add the slot image

Go to the storyboard. Select the UIImageView, look at the Attribute Inspector and do the following changes:

* Select “ButtonSlot.png” on the Image field.
* Select AspectFit in the Mode field.
* Then, go to the menu entry “Editor”, and click “Size to Fit Content”

### Add the button image

Select the UIButton, look at the Attribute Inspector and do the following changes:

* Change the Type to “Custom”
* With the State Config to “Default”, change the Background to “Button.png”
* With the State Config to “Highlighted”, change the Background to “Button.png” (That stops the button from becoming darker when we are moving it)

Now select the menu entry “Editor/Size to Fit Content”

**Note:** When trying the application, it may happen that the button and/or slot are under each other or the background. This means that first, you need to reorder the objects in the view. For this, change the order of the items in the view hierarchy to show the objects in that order:



Then, select the button and go to the menu entry “Editor/Arrange/Send to Front”.

In addition, make sure that the button and slot are centered for all type of screen.

## Test and run the app

Run the simulator and make sure that the babygame is working as expected.

# BabyGame

You need now to modify this application to create a game using the other images in the Resources directory. The resulting game should look like this:

## Extensions

* Remove the top status bar to win some space (Tip: Attribute Inspector)
* Add animations to the game pieces so that they come back to their place in a nice way. Look at the **help of animateWithDuration of UIView.**
* Show a congrats message using a UIAlertView
* Add a restart button with the congrats message.

## Optional Extensions (bonus)

* Add a 3D effect to the pieces making them look bigger when you select them.
* Shake the phone to restart the game at any time. (Look at *motionEnded* of UIResponder)
* Add music

# Submit your project

When your project is ready, proceed as follows:

1. Click on Product > Clean Build Folder
2. Click on Product > Build
3. Click on File > Save as Workspace > Select where to save it

Then, once you have also answered all the questions at the end of each lab, in this case both HelloWorld and BabyGame have some questions to be answered, you will compress the generated folder + the answers to the lab questions into a zip folder and upload it to the server as follows:

* **Name is as: projectname\_name1\_name2\_date.zip**
* **Copy in** https://nextcloud.eurecom.fr/s/3zeCtMEQsx3s4JB