1 Login

Login using your Unix account, and mount your home:

- goto /Volumes/eurecom/ and run ./create_mount_script.command. Your homes and teaching will be mounted and available in the desktop and finder
- run the two scripts in the Desktop: datas.command and homes.command
- when prompted, provide your windows password
- Both directory will be mounted on the Desktop
- copy the directory “fall-2015/lab_ios/lab2/images” to the Desktop

Note: when running the simulator, you will be prompted “enable developer mode”, press **enable**, and then when prompted to give the password, **cancel**

You can access slides and lab instruction in the course directory.

2 Button-Slot App (<30 minutes)

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

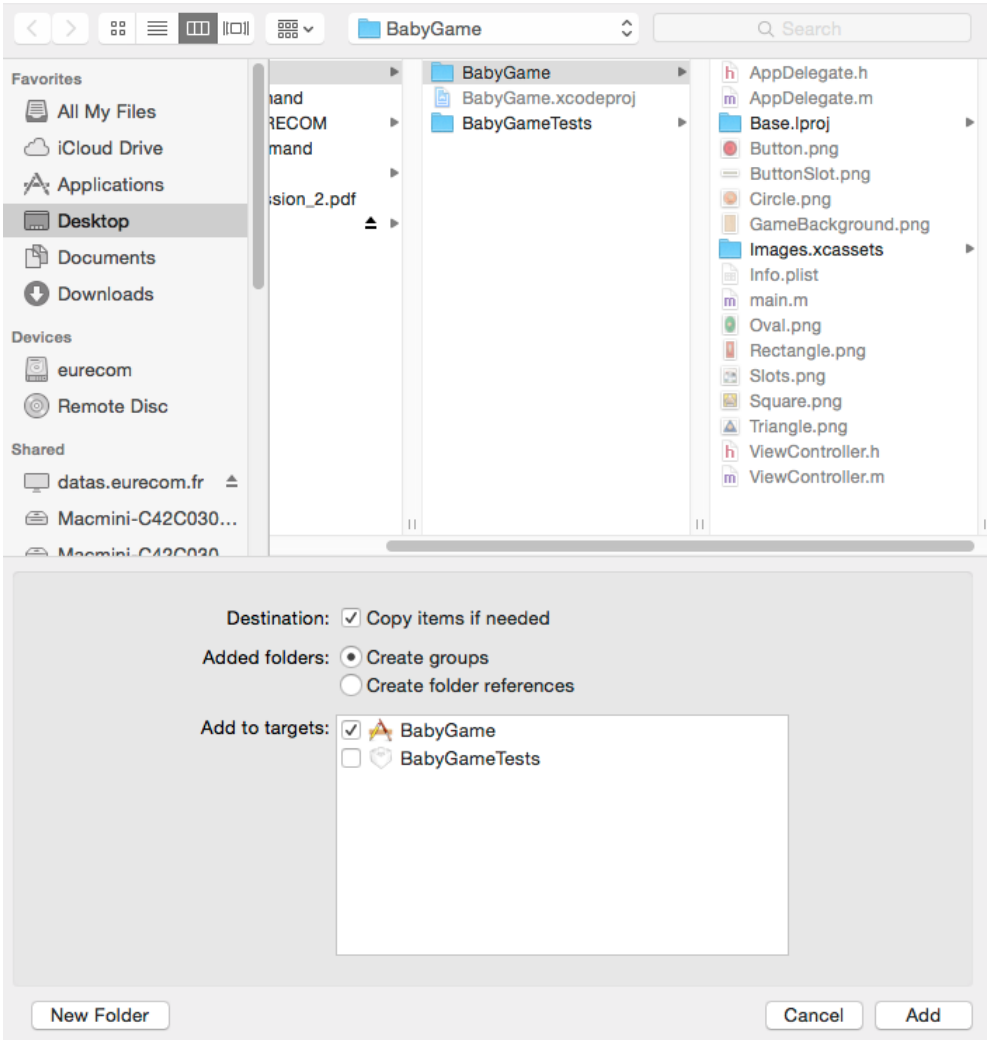
- Product Name: BabyGame
- Organization name: eurecom
- Organization identifier: fr.eurecom
- Device: iPhone
- Languae: objective-c

Select the location, and then create the project.

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

2.1 Import the resources to the project

Select the “**Supporting Files**” folder and with a right click select “**Add Files to BabyGame**”. Select all the images from the directory Resources that you should have in the course directory and select “**Copy items if needed**”. Save the project.

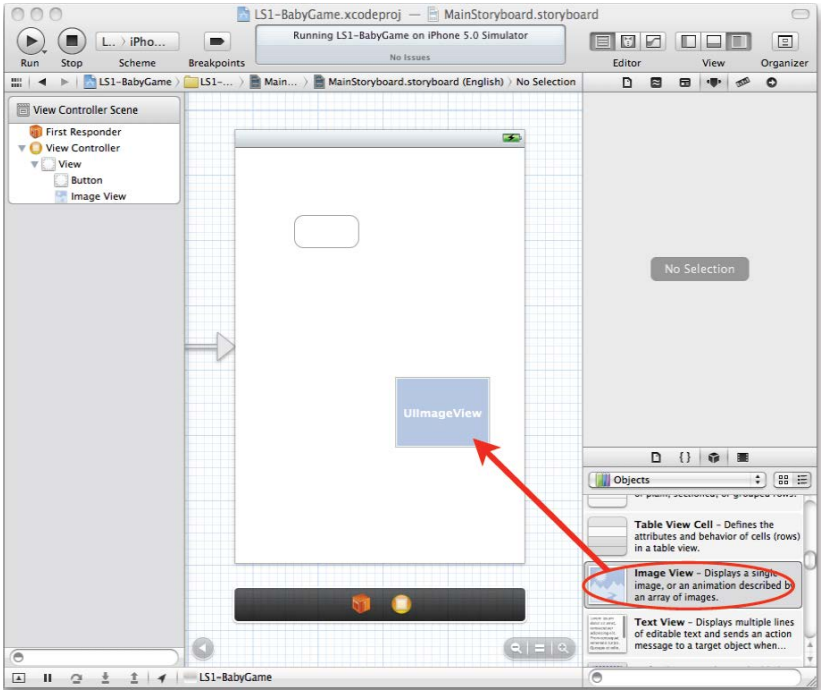


2.2 Build the Storyboard

This is the file that is going to contain all the views of the application and how they interact one to each other. In 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.

2.3 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.



Center both objects with use of constraints.

2.4 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.

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
```

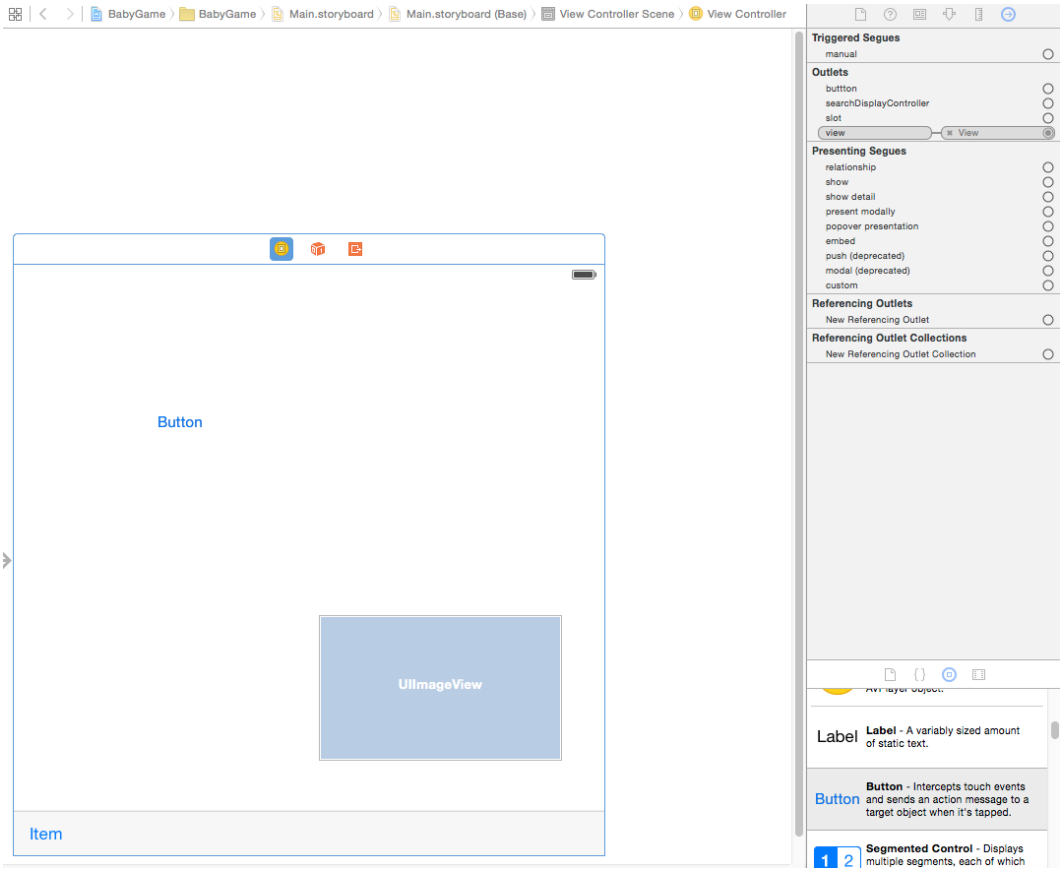
You will notice a little warning sign appearing in the ViewController.m. By clicking the mouse over the warning sign XCode gives you a tooltip with the explanation of the warning. (The same happens when there is an error.)

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:

```
@synthesize button, slot;
```

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

Open again the MainStoryboard file and select the "View Controller". Also open the connection inspector. 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.

2.5 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...).

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:

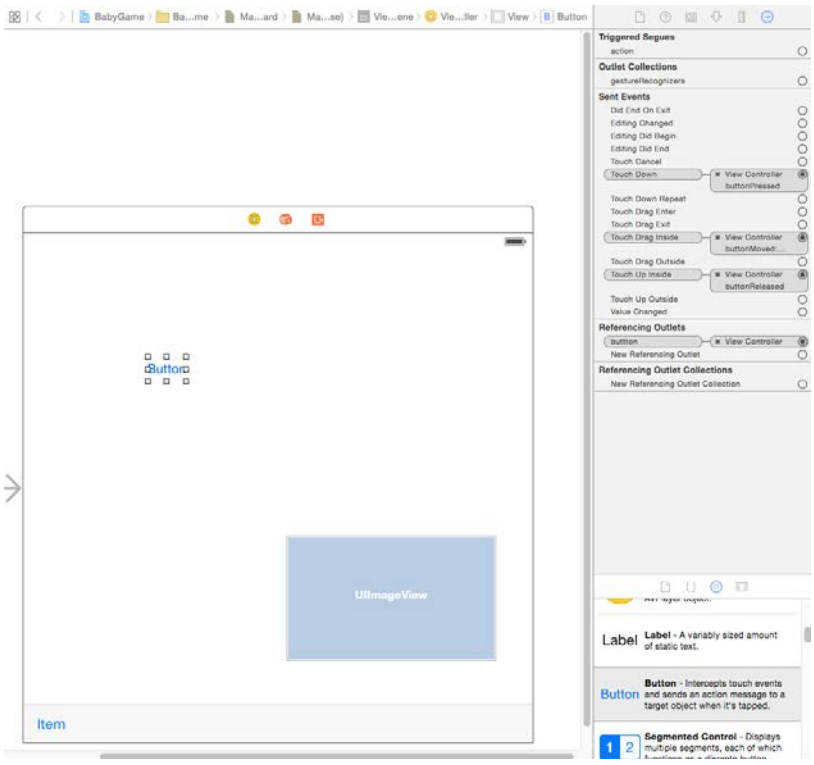
```
-(IBAction) buttonPressed;
-(IBAction) buttonMoved: (id)sender withEvent:(UIEvent *)event;
-(IBAction) buttonReleased;
```

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

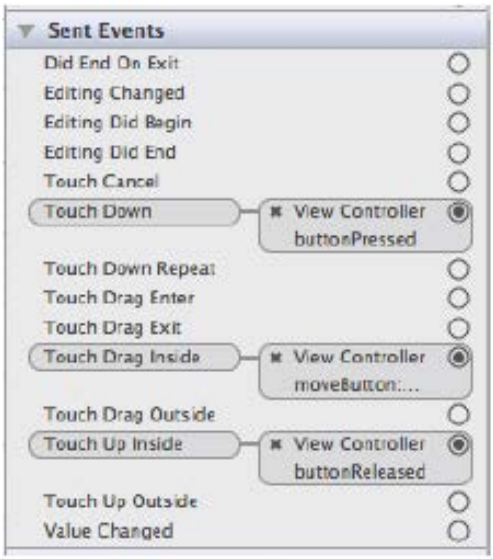
Open again the MainStoryboard file and select the UIButton object. The “Connection Inspector” should show now a list of outlets and actions that can be connected. The interesting actions in our case are:

- **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 :

2.7 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.

2.7.1 -(IBAction) buttonPressed

Let's start by adding the member needed to store the original button position, add that line of code in the ViewController.h file with the other member declarations:

```
CGPoint originalPosition;
```

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

```
-(IBAction) buttonPressed {
originalPosition = button.center;
}
```

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) actionName:(id)sender;

2.7.2 -(IBAction) buttonMoved

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

2.7.3 -(IBAction) 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.**

```
-(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, distanceBetweenPoin, private.

2.8 Beautify the Interface

Go to the MainStoryboard for the last time.

2.8.1 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.

```
UIColor *background = [[UIColor alloc] initWithPatternImage:
[UIImage imageNamed:@"GameBackground.png"]];
self.view.backgroundColor = background;
```

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

2.8.2 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"

2.8.3 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”.

2.9 Test and run the app

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

3 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:

3.1 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.

3.2 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

Note: please zip your project under lab2.name1_name2_date.zip and put in under fall2015/lab_ios/lab2/.