GAME2011 Winter 2016 Lab1 and Lab2.

Purpose: Get started with iOS game development using Unity.

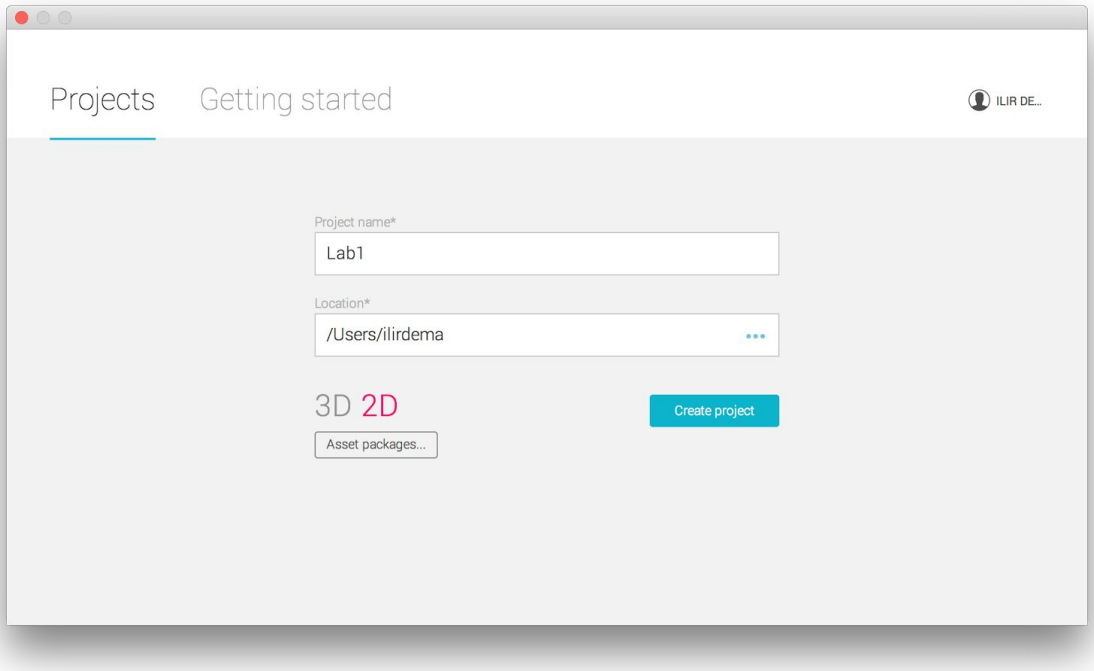
Outcomes:

1. Get started with Unity.
2. Create a simple iOS game using Unity.
3. Add assets to a scene and make use of 2D physics.
4. Script the user input for touch screen.
5. Test your game in Unity

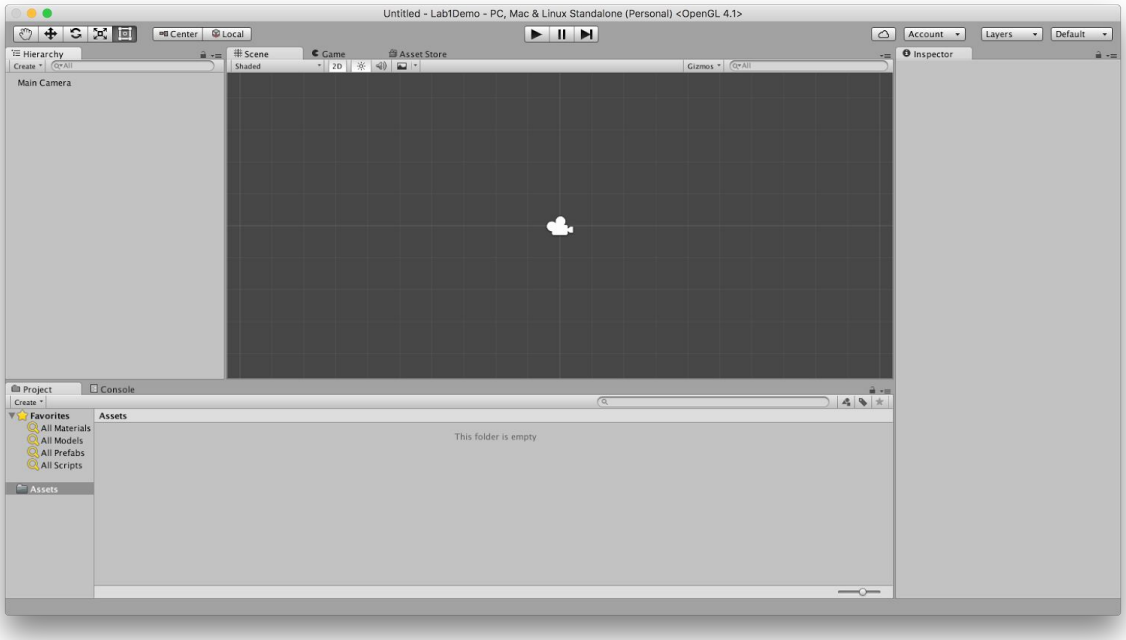
Marking Scheme: Each lab is worth 3% of your course final mark. It will be graded out of 10. In order to earn the grade for both labs, you must show your completed work to your instructor end of the lab class. You may show both completed labs end of the lab class second week. No submissions will be accepted after the end of lab class on second week.

Lab 1.

Using Mac, create a Unity account and log in. Create a new project and name it Lab1xxx where xxx is your first intital and your last name, for example, if your full name is John Smith, the project should be named Lab1JSmith. Below is a screenshot:



Once the project has been created, you should see the following view. If the looks of your screen is a bit different, you may want to select the Default layout from the Window menu.

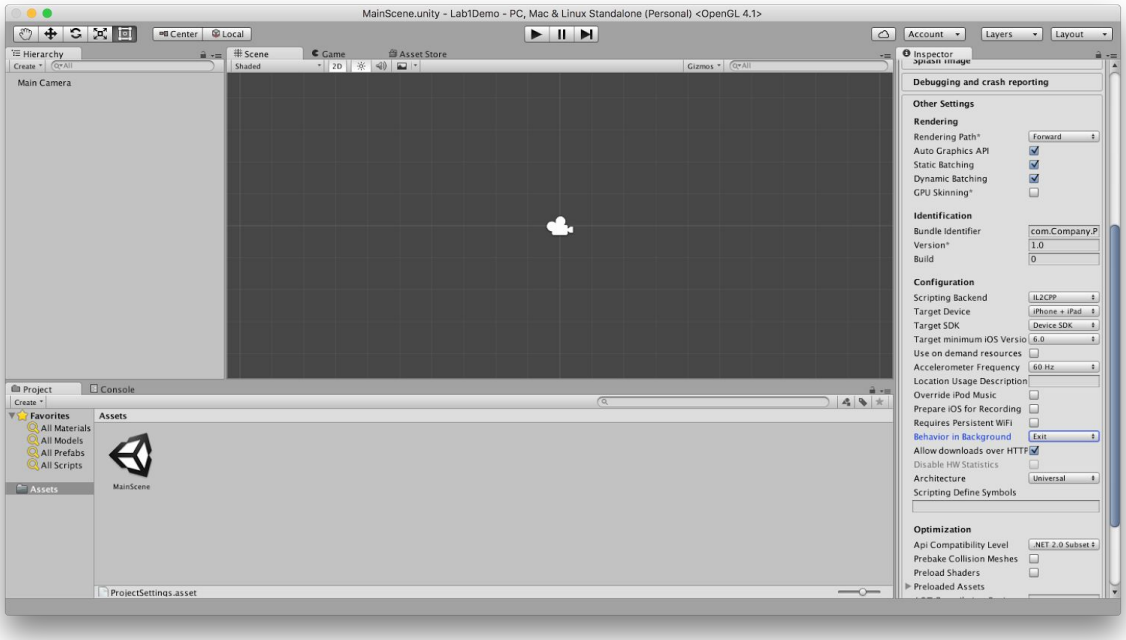


The game that we want to create, is a simple bouncy ball inside a rectangular box, that the user can push in any direction using either the mouse (if we run it in Unity), or a swipe (if we run it on an actual device).
The final product must look as follows:

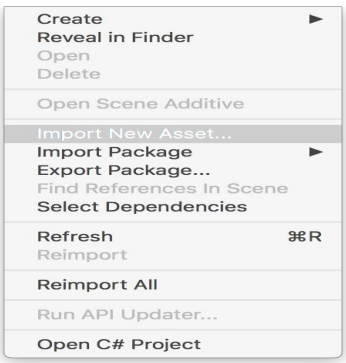
We are going to need two images - a background, and a ball. Although both are provided on Blackboard, you are encouraged to use your own images.
Next, set the build settings to iOS:

Make sure to choose **Exit** from the **Behaviour in Background** just so your games frees the resources when user hits the Home button on the device. Also at this point your can set the Bundle Identifier, but for now we will skip this step as Apple requires that one must have a developers account to do that (and that costs money!). A screenshot is shown below.

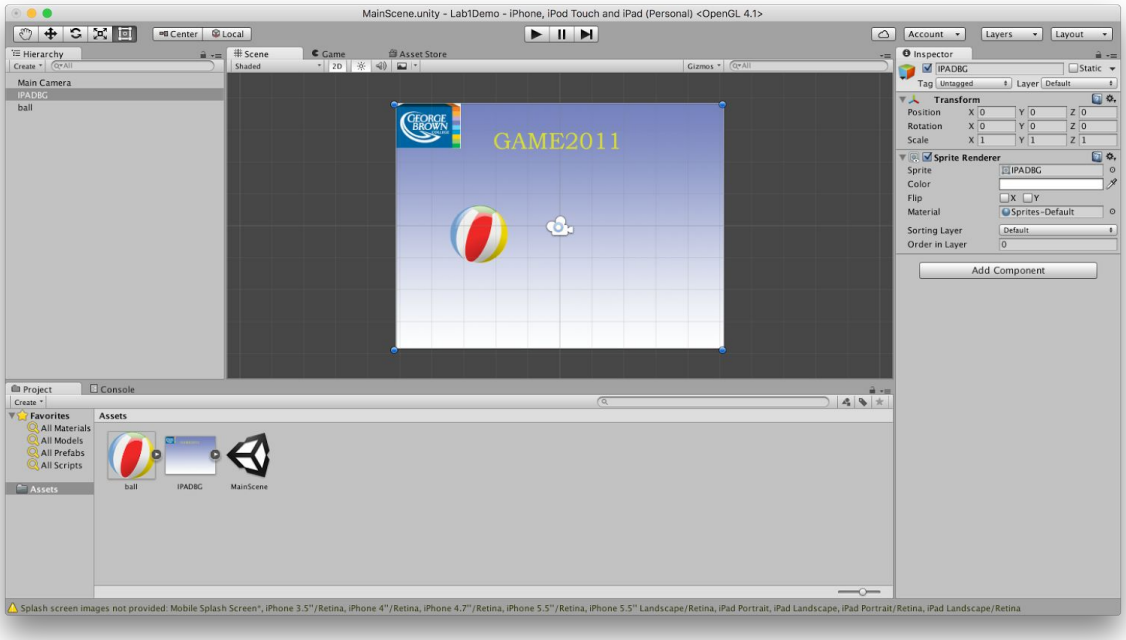
Once you have done that, is is a good idea to hit Build just so you check how your installation of Unity works. Normally Unity will ask you for a destination. Enter whatever you prefer (best keep everything in the project folder, though).



Next, we want to import our assets:



Navigate to the folder where you have your images and import them. Drop the background on the scene and make sure to center it by setting the transform position to (0,0,0). On top of it drop the ball. Its initial position is not important, however do not put it very close to the floor as we want to have it bounce.



For the ball to “fall down” and in general obey the laws of physics as we know it, the ball (and every other gaming object for that matter) must possess a “rigid body”. This is considered a component, so we add it using the **Add Component** button.

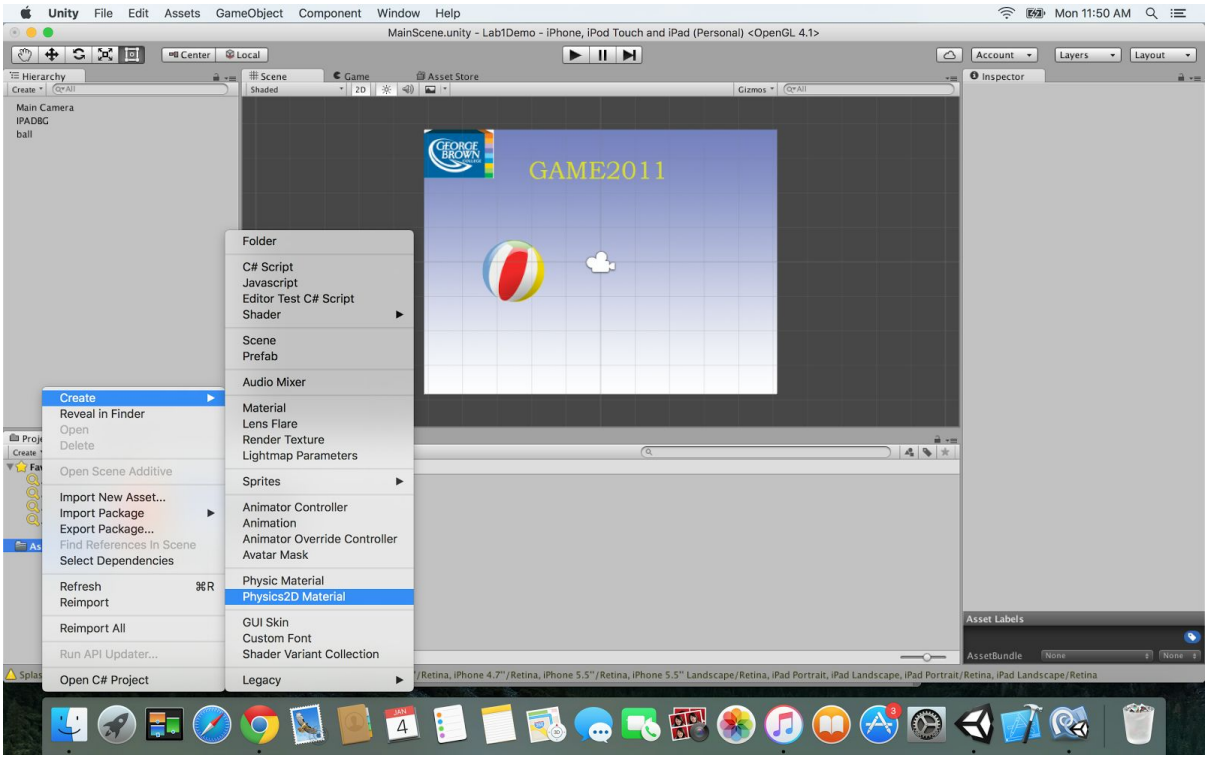
At this point, please take a moment to familiarize yourself with various parameters of Rigidbody2D. Any change on those parameters will affect the behaviour of your object due to the physics engine. Feel free to experiment with various settings later on. For now, set the **Linear Drag** to 0.05. Various forces applied on your game object, act on the rigid body.

This is the earliest point when the game is playable. If you try it, the ball will fall freely and disappear from the scene.

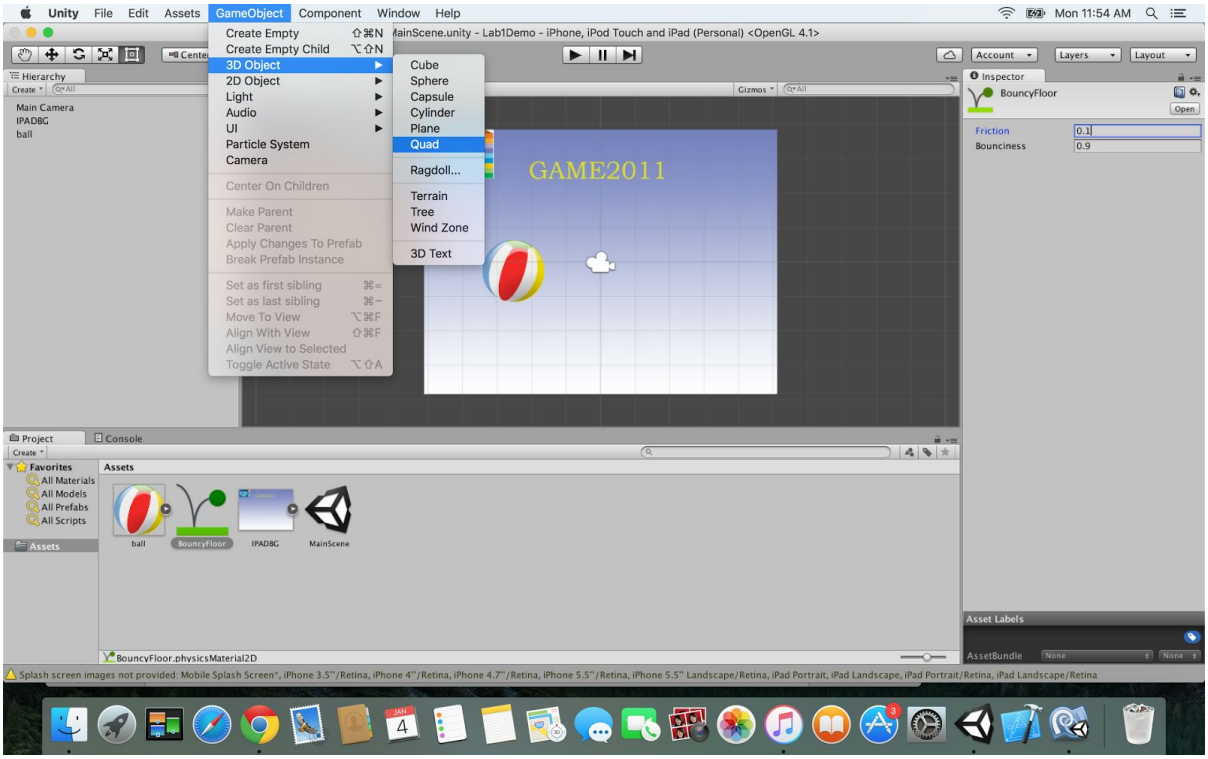
To make it bounce, we need to have a floor, a ceiling, and two left and right walls so the ball can bounce against them. Also the ball and the other elements need to have colliders attached to them. The colliders are components that the physics engine uses to simulate collisions between objects. The colliders “transmit” the forces to the rigid body according to certain rules derived the properties of the collider.

That said, we need to define the materials that will give colliders the desired properties. Although you don’t have to, it is best to create two materials, one for the ball, and another one for the walls.

Right-click on your assets folder, click Create->Physics2DMaterial, name if BouncyFloor, do it again, and name it BouncyBall. Experiement with properties later on. For now you may want to set the Bounciness to 0.9 and Friction to 0.1. Varying them will give different results when you try your game.

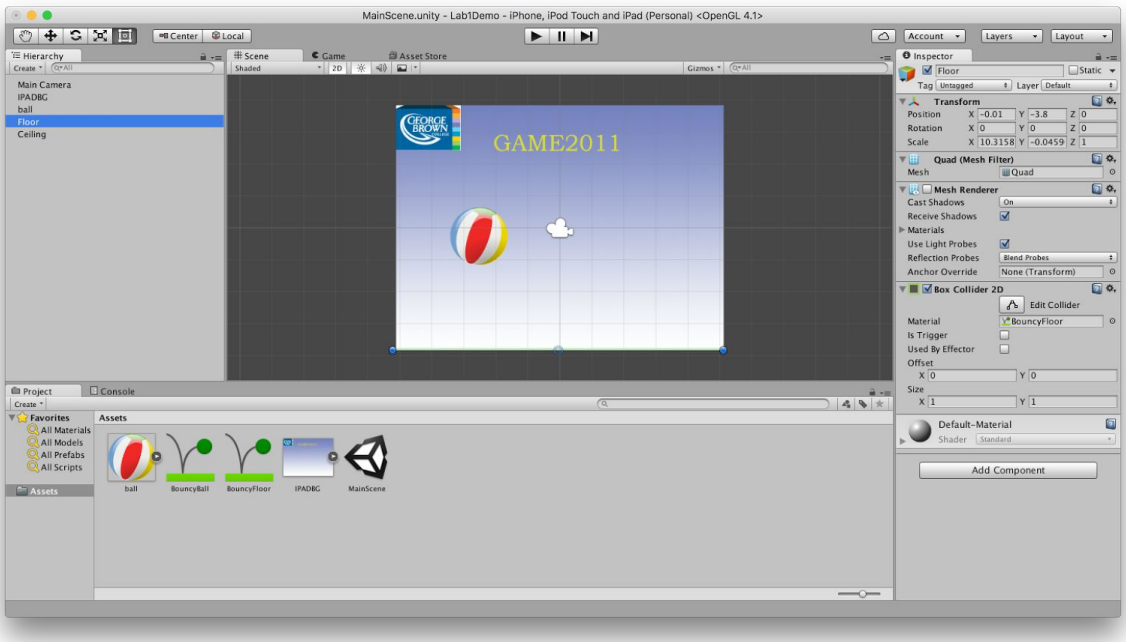


The best way to make the bouncy walls is to use Quad from Physic3D:



Since Quad is a 3D object, we want to remove its Mesh Collider and also uncheck its Mesh Renderer so the floor is invisible when we play the game. Also we want to resize it so really

covers a very thin strip of the bottom of the box. Add a BoxCollider2D. Drag the BouncyFloor from the Assets folder to the Material as shown below:



Duplicate and shift it to the proper position to make other walls.
Add a CircleCollider2D to the ball and also drag the BouncyBall material in the appropriate place.
Now you can try your game and see the ball bounce.

At this point, demo your work to your instructor. Make sure your instructor has given you a mark for your work. **IF YOU LEAVE THE CLASS BEFORE SHOWING YOUR WORK TO YOUR INSTRUCTOR, YOU WILL RECEIVE A ZERO GRADE FOR LAB1.**

This is the end of Lab 1.

Lab2.

Open your project from last Lab. Right click on your assets folder, create a new folder, call is Scripts, right click on the scripts folder, just as shown on the lecture, and create a new C# script called PlayerInput.

Repeat the process and name the other script Ball.

Double click on Ball.cs script and open it in MonoDevelop.

We want to code to implement the following behaviour:

1) The user clicks and drags or swipes the screen at the current ball location.
2) What are the properties of the swipe? Can you think how to represent it? Hint: it has a magnitude and a direction.
3) The swipe will determine the force that acts on the ball. That is, whenever user swipes or clicks, **we need to convert the user input to a force acting on the Rigidbody2D component of the ball.** The force must be proportional to the swipe.
4) To do that, we need to get a hold of the component once the game starts. So, define a Rigidbody2D class variable, and store in it the appropriate component of the ball.
5) Do we need an Update() method? Why?
6) Design and write the code for the ReceiveInput method. You may use some of the code developed in the lecture.
7) Finally, using Add Component, add this script to the ball game object.

Double click on the PlayerInput script and open it in MonoDevelop.

1) Create the SimpleTouch structure like we did in the lecture.
2) Declare the float variables SwipeTime and SwipeDistance. Note we will not make use of SwipeTime, however it is up to you to use it if you like.
3) Declare a Touch variable. Note Touch struct variables contain information for the raw touch input. You may want to consult Unity documentation for more information about it: http://docs.unity3d.com/ScriptReference/Touch.html.
4) Also declare a Ball variable (public).
5) Declare a SimpleTouch variable. Both Touch and SimpleTouch must be private.
6) Create the CalculateTouchInput method with the following signature:
7) private void CalculateTouchInput(SimpleTouch CurrentTouch)
8) Make use of the code provided in the lecture. Do the necessary modifications to invoke the ball's ReceiveInput in case the user touch collides with the ball. Pass in the computed magnitude and direction of the current swipe.
9) Create a new gaming object in the hierarchy and call it PlayerInput. Add the PlayerInput script as a component and make sure to drag and drop the ball gaming object in the appropriate slot for the Ball variable. Below is a screenshot of the completed procedure.

Try your game. You should be able to shift the ball using click and drag.

Optionally, people who have developers account and an Apple device handy, may do a Build & Run and try their game on an actual device.

At this point, demo your work to your instructor. Make sure your instructor has given you a mark for your work. **IF YOU LEAVE THE CLASS BEFORE SHOWING YOUR WORK TO YOUR INSTRUCTOR, YOU WILL RECEIVE A ZERO GRADE FOR LAB 2.**