RGS TUTORIAL

SPECS – UPF
BCBT 2012

1. Running RGS framework

To check that RGS is working on your system:

1. Run RGS.exe located in folder C:\Documents and Settings\All Users\Escritorio\Windows 32bit\Binary . Press Control+c and write down in the console “HideGUI” in other to minimize the RGS interface.

2. Run FAAST: C:\Documents and Settings\All Users\Escritorio\Windows 32bit\Drivers\FAAST-0.10 And check that it is recognizing the user. If everything works fine you should see something similar to this:

   A new window will open and the Kinect IR sensor will switch on. The Flexible Action and Articulated Skeleton Toolkit (FAAST) is middleware to facilitate integration of full-body control with games and VR applications using either OpenNI or the Microsoft Kinect for Windows skeleton tracking software.

3. Run IRCKinectDriver.exe located in C:\Documents and Settings\All Users\Escritorio\Windows 32bit\Drivers

4. From Unity open the project UnitySDK located in C:\Documents and Settings\All Users\Escritorio\Windows 32bit\UnitySDK. This project contains all the items we will need in this tutorial.

5. Open the scene “HCI.unity” located in C:\Documents and Settings\All Users\Escritorio\Windows 32bit\UnitySDK\Assets\Scenes\02_HCI. This scene just contains a camera and an avatar. Explore the scene.

6. Press the button “play” to run the scenario.

7. Now you should see that the movements of the user are being mapped into the avatar’s arms.

2. Introduction to Unity

Lets now dive in and take a closer look at each of Unity’s windows and some of their features:

The Toolbar

Not quite a window as such, but a very important part of Unity’s interface nonetheless. The ‘Toolbar’ spreads across the top of the IDE and features several key tools for manipulating the ‘Scene’ and ‘Game’ windows as well as the layout of the interface. It also holds the very important ‘Play’ button.

The Toolbar – Transform Tools

The first component within the Toolbar, located in the top left corner, holds the four ‘Transform Tools’ which you must now burn deep into your memory.

From left to right these are:

The Hand Tool (Keyboard Shortcut: Q)

By default the Hand Tool will allow you to Click and Drag to pan around the scene. Holding Shift will increase the panning speed; using combinations of Alt, Ctrl, LMB, MMB and RMB will allow you to pan, rotate and zoom around your scene. Don’t fret, we’ll explain these in a little more detail shortly.

The Translate Tool (Keyboard Shortcut: W)

The Translate Tool provides the ability to select GameObjects — more on these later — within your scene (or hierarchy) and then use the arrow-ended axis handles to move that object around the scene by clicking and dragging each axis handle. You can also move the selected GameObjects freely over all three axes by clicking and dragging the small square where the axis handles meet.

The Rotate Tool (Keyboard Shortcut: E)

The Rotate Tool provides axis handles that loop around and surround the selected GameObject which when click and dragged allow you to rotate it around either its center or pivot point.

The Scale Tool (Keyboard Shortcut: R)

Very similar to the Translate Tool the Scale Tool provides the block ended axis handles which when Click and Dragged allow you to scale the selected GameObject on that

particular axis. To scale all 3 axis simultaneously simply click and drag the grey cube where all 3 axis handles meet.

The Toolbar – Transform Gizmo

The next component within the Toolbar, highlighted below, is the 'Transform Gizmo', which determines whether the selected object is rotated or scaled relative to its center point or to its pivot point, and whether local or global rotation is used when scaling and rotating the selected GameObject.

The Toolbar – Control Buttons

The component within the center of the Toolbar controls the preview of your game within the Game window. Pressing the 'Play' button will play your game, clicking the 'Pause' button will unsurprisingly pause your game and the third button along allows you to step through the game when it's paused.

The Toolbar – Layers & Layout

The 'Layers' drop down menu located to the right hand side of the screen within the Toolbar allows you to toggle the visibility of particular layers on and off, and the 'Layout' drop down menu provides options regarding the Layout of the windows or views on screen. If you create a custom layout and are particularly fond of it you can use this menu to save your layout so you don't have to create it every time you restart Unity.

So that's pretty much it for the Toolbar, pretty straight forward so far right?

Scene View

The Scene view is where you'll spend a lot of your time building and editing the components of your games within Unity. A fully rendered 3D 'preview' of the currently open scene is displayed inside the window where you can add, edit and remove GameObjects using its Orthographic and Perspective views.

For those coming from a Flash background you can liken this to the *stage*. All objects within this window are considered active GameObjects and are listed within the Hierarchy, which we'll look at shortly.

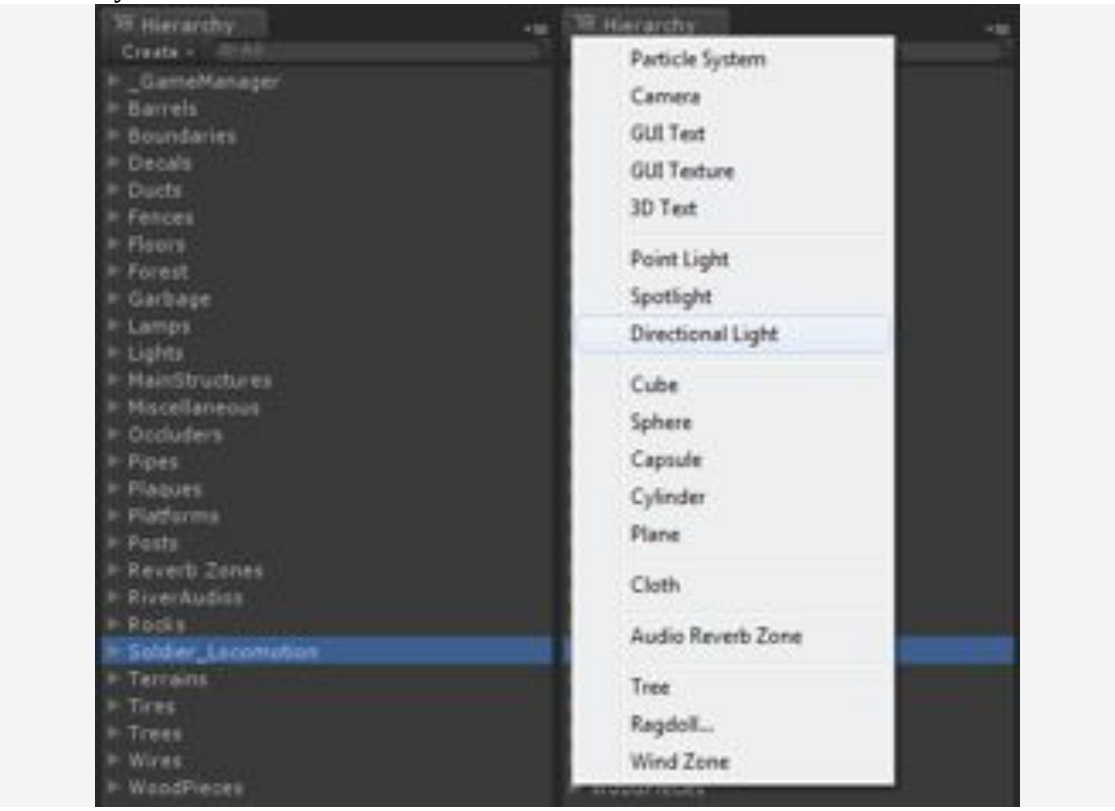
Scene View Control Bar

The control bar along the top of the Scene view provides various view modes such as Textured, Wireframe and Textured Wireframe. It also provides toggles for 'in-game' Lighting, Audio and Elements such as Skyboxes within Scene view. The above image for example has Lighting and Elements toggled on and Audio toggled off.

So that's it for Scene view, let's now take a look at the Hierarchy.

Hierarchy

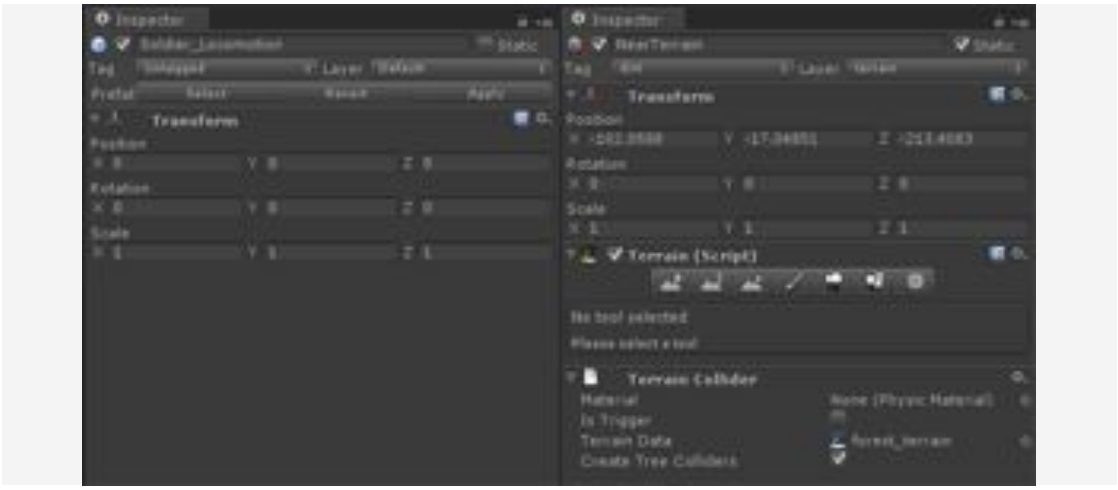
The Hierarchy window provides a list of all the GameObjects within Scene view, i.e. every object in your current scene. When a GameObject is selected within the Hierarchy it's also selected within Scene view and vice versa. You can add new GameObjects to the Hierarchy via the 'Create' menu shown below.



Objects that have children, i.e. are a parent object, have a grey triangle to the left of their name which toggles the display of their children. You can rename a GameObject by selecting it within the Hierarchy and pressing F2 or using the Inspector discussed below. Those shown in blue text within the Hierarchy are Prefabs which will be explained shortly, the rest are simply GameObjects which I'll also explain a bit more about shortly.

Inspector

The Inspector displays information about the selected GameObject including all of its attached components and their adjustable properties. Properties can be assigned or modified by dragging and dropping GameObjects or Prefabs from the Hierarchy or Project views or by simply typing values, checking boxes, etc.

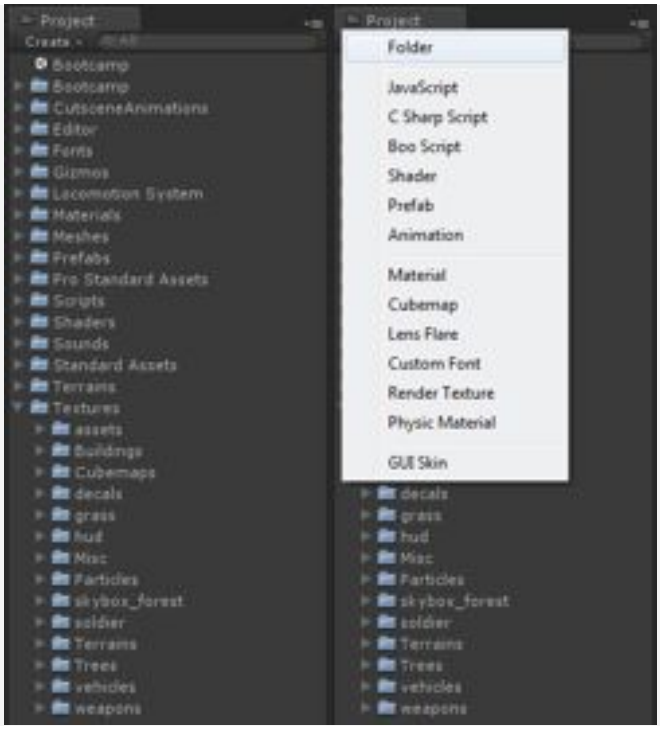


The very top of the Inspector displays the GameObject's name and, to the left, whether it's enabled. On the far right you can define whether the GameObject is static or not, for lighting purposes. Beneath the GameObject's name is its tag, if you have assigned one, and the layer it is part of, if you've assigned one.

A GameObject's Components are listed below and are collapsible via the triangle left of their name. The checkbox directly left of the name defines whether the Component is enabled or not, which comes in very useful when debugging. To the right of the name is a link to the Components Reference as well as a menu for resetting/removing the Component. You can also right-click a Component's name to remove it.

Project Window

Every Unity project has an 'Assets' directory for all the models, textures, scripts, prefabs etc used or made available to the project. You can navigate and explore this directory within Unity's Project window as well as creating new sub-directories and Assets (i.e. scripts) within it by right-clicking or selecting 'Create' as shown below. Assets can also be searched for by using the Search field located in the top-right corner of the window. If you're coming from Flash you can liken this to the *Library* where you keep all your project's assets.



To add assets such as models and textures to your project, from the top menu select *Assets -> Import New Asset*, or simply drag them from your OS's file system to the Project View and the asset is immediately available for use within your project. If assets are altered — e.g. you edit a texture in Photoshop — when you save and return to Unity it will automatically update the asset for you. Awesome? Yes!

You should NEVER move project assets around using your OS's file system since this will break any meta-data associated with the asset, i.e. textures & materials applied to a model, scripts applied to a GameObject. You must ALWAYS use the Project View to organize your assets.

If you wish to locate an asset in Unity within your file system for any reason, like to edit in Photoshop, you can right-click and select 'Show In Explorer'.

Game View

The Game view is likely to become your favorite part of the Unity interface as it's where you play and test your game as if it's actually been published.

When you run your game, using the Play button within the Toolbar discussed earlier, you'll notice that changes within the Game View are reflected within both Scene View, Hierarchy and the Inspector and that it's still all editable. This gives you the great advantage of being able to edit and tweak your game to get the best result whilst it's running!! However, as soon as you stop running the game all changes will be lost.

Lets repeat that:

ALL changes made whilst the game is STILL running WILL BE LOST once the game stops running.

We all fall victim to this; learn quickly or risk going bald fast.

Game View Control Bar

The control bar at the top of the Game view allows you to force an aspect ratio on your game within the Game view, toggle Gizmos, toggle Stats and toggle whether the Game view goes full screen when you click Play.

GameObjects, Components & Prefabs

You may have noticed these three terms in the above sections. *GameObjects*, *Components* & *Prefabs* are key concepts to building games within Unity and it's essential you understand what they are and how you can use them.

GameObjects

Every object within your game is a GameObject. They are containers that contain Components to determine their behavior and appearance. Every GameObject has a Transform Component by default which determines its location, scale and rotation within the game, and more can be added, edited and removed using the Inspector. Within the Hierarchy, GameObjects are white.

Components

Components determine the behavior and appearance of GameObjects they are attached to and are the real workings of any game within Unity. Scripts that you develop can be attached to GameObjects in exactly the same way as Components because that's what they are: Components. They're Components that YOU developed. To add Components to the selected GameObject, use the Component menu.

Prefabs

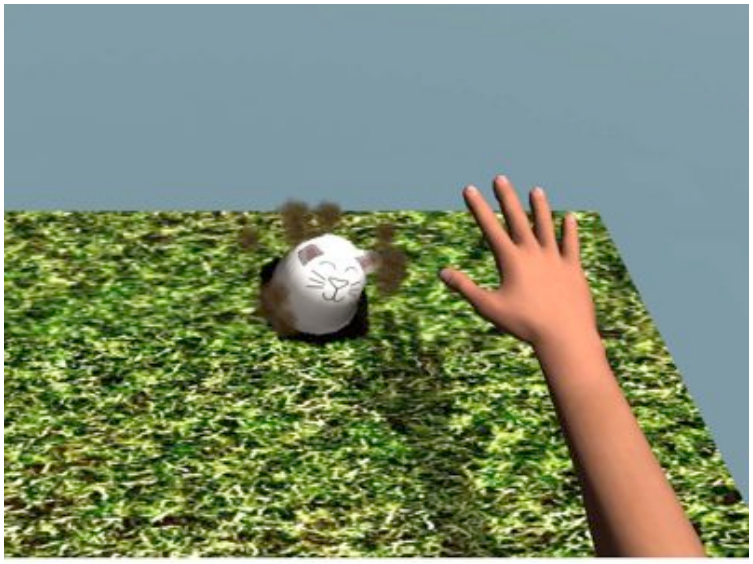
For those who have worked with Flash, Prefabs are very similar to the concept of MovieClips within the Library. If you've never worked with Adobe Flash or don't understand the concept of MovieClips then ignore that last sentence.

Prefabs are re-usable GameObjects stored within the Project window alongside all your other Assets. You can create as many instances of a Prefab within your game as required and any alterations to the original will be reflected within every instance you've created. This is called Inheritance. You can also override an instance's default values allowing them to look more unique, rather than like clones of the original Prefab.
When you first create a new Prefab using the Project Window 'Create -> Prefab' it is just an empty container with no GameObject and therefore can't be added to the Hierarchy or Scene. To create a usable Prefab you must select a GameObject within the Hierarchy and drag and drop it onto the newly created Prefab. Once done, the original GameObject is now an instance of the Prefab and you can add as many instances of the Prefab to your game as you wish. Within the Hierarchy, Prefabs are blue.

Okay, enough theory; let's get started…

3. Building a simple scenario

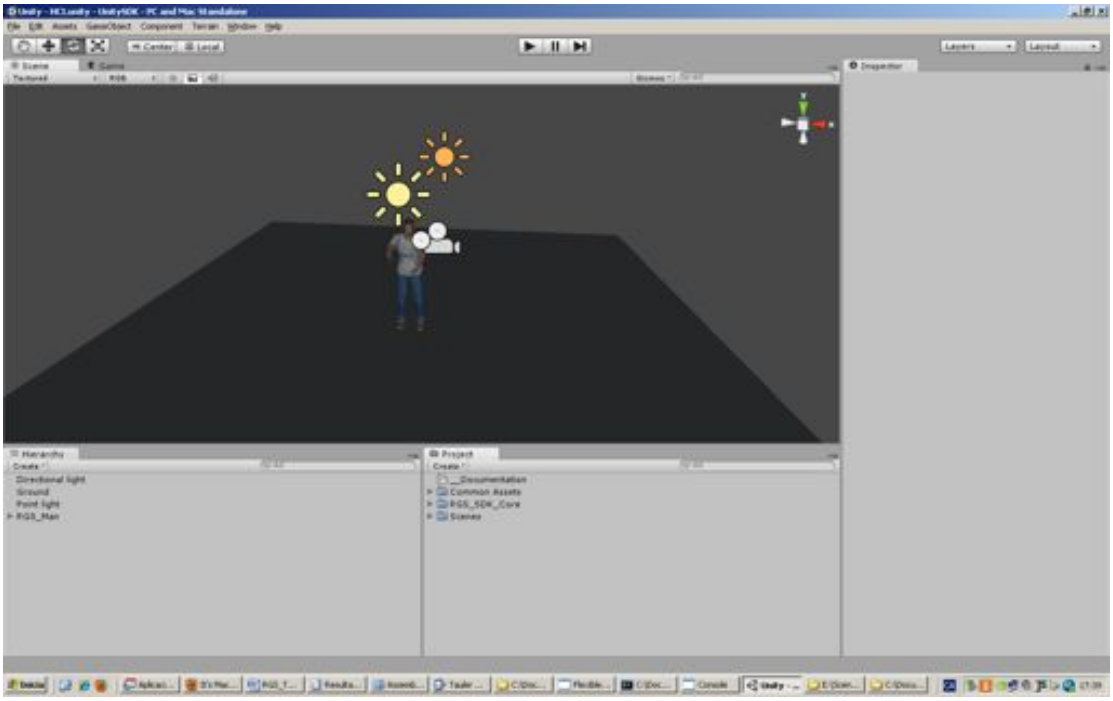
In this chapter we are going to develop a simple scenario where the user has to reach targets. We will create a script in c# to randomize the appearance of the targets in the horizontal plane. Finally, we will learn how to use colliders and basic scripting for making them work.



A. Open a Project and a Scene

To start the tutorial Project, select “File -> Open Project” and browse the UnitySDK folder in C:\Documents and Settings\All Users\Escritorio\Windows 32bit. Once the project is imported, which can take some time for complex projects, open the ‘HCI’ scene within the Project window.

You’re hopefully now looking at the same as below, so welcome to Unity and the RGS demo!



Spend a few minutes navigating around the scene.

There are quite a few keyboard and mouse combos for navigating the Scene view within Unity. The below are for a three-button mouse and can be used when any Transform tool is selected. These are crucial for a fast work-flow, so practice them in your sleep.
To rotate around current pivot point within the scene: **Alt + LMB**
To rotate around on the spot within the scene: **Ctrl + RMB**
To pan around the scene: **Ctrl + Alt + LMB / Alt + MMB**
To zoom in / out of the scene: **Alt + RMB / Mouse Wheel**

With all the above, hold Shift (if you can) to speed them up. If you're using a one- or two-button mouse I'd recommend checking the Unity Scene Navigation Documentation.

Unity's Scene view also features a 'Gizmo' in the top right hand corner which shows the current viewing angle and allows quick navigation to **TOP** (Y) / **SIDE** (X) / **FRONT** (Z) and **PERSPECTIVE** views. Click on each of the gizmo axis to change the view.

B. Importing a Unity package

Go to Assets>ImportPackage and select BCBT_RGSTut_Pack from "UnitySDK" folder.

Once you imported the package of assets succesfully you will see new assets in the project window. Explore the contents.

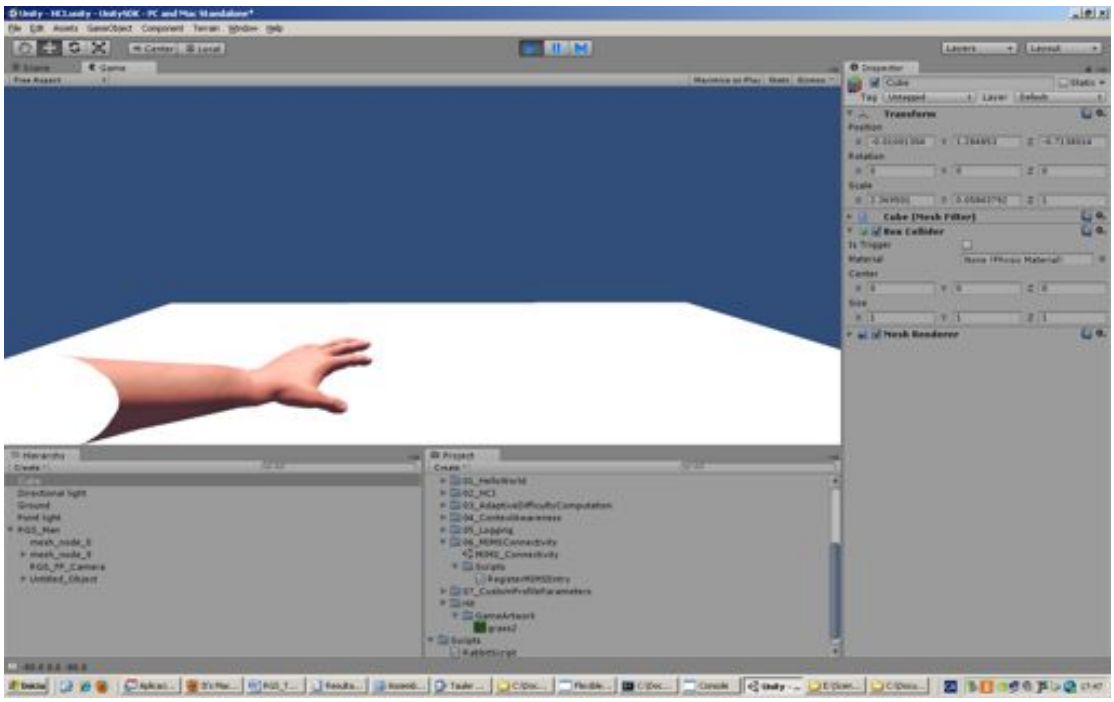
C. Creating new objects: surface

Create a new GameObject with the shape of a cube. Go to the menu and click on GameObject>Create Other>Cube.

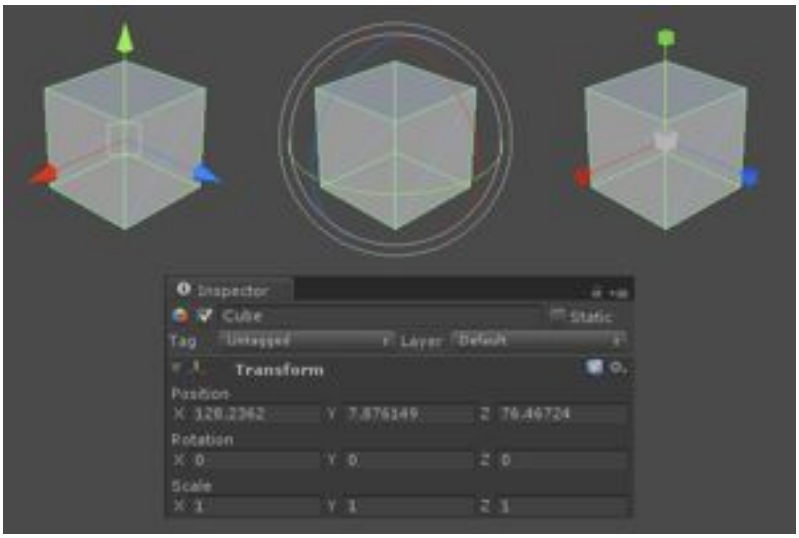
D. Move/Rotate/Scale objects

1. Use these three tools to modify the new cube in order to create the surface of a table.

2. Move and rotate the camera to have a first person view.



3. When you are done click on “play” to check that the position of the table and the camera is correct. Adjust the positions until you like it.



When building your games you’ll likely place lots of different objects within your scene. To do this it’s easiest to use the Transform Tools in the Toolbar to translate, rotate, and scale individual GameObjects using the appearing axis handles, shown below. You can also use shourtcuts:

To move an object: press W
To rotate an object: press **E**
To scale an object: press **R**

You can also type values directly into the Transform Component shown in the Inspector, shown above.

E. Attach a material

Now we are going to add a material to the surface we just created.

1. Go to Assets>Create>Material
2. Give a name to the new Material (i.e. grassMaterial)
3. Select the material and search the picture “Scenes/Hit/GameArtwork/Grass2” in the Project window.
4. Drag and drop the file Grass2 into the Inspector to add the texture.
5. Drag and drop the material from the project window to the surface we created in the scene window. Change the Tiling to 3x3 and Color to gray.

F. Scripting in c#: Programming the basics of the game

We are ready to create the first script of the game. We will use c# to automatize the instantiation of the target in random positions.

1. First we need to create a new c# script (you can also use JavaScript or Boo if you feel more confortable with those languages). Go to Assets>Create>C# Script and name the file "MainGame.cs".

2. In order to generate objects (i.e. targets) during run-time (as the game is being played) we will use the Instantiate function:

```
Instantiate(prefab, new Vector3(i * 2.0F, 0, 0), Quaternion.identity) as Transform
```

A few things to think about:
A. Which object do we instantiate?
B Where do we instantiate it?

Go though the following script to understand what it is doing:

Copy the script and paste it in the new script you created.

```
using UnityEngine;
using System.Collections;

public class MainGame : MonoBehaviour {

//Declare the target, the avatar and an array of angles
    public RabbitScript TargetPrefab;
    public Transform Avatar;
    private float[] angles = new float[10];
    public float timeLimit = 5f;


//Specify the duration of the game
    private float Endtime = 30f;

    //Define the distance from the avatar to the targets (a reachable distance)
    float Radio =0.7f;

   //Define the y position of the target. (It should be greater that the y position of the grass
  surface)
    float Height = 1.314f;


    //This function is authomaticaly called when the application starts
    IEnumerator Start() {

        //Declare an array of predetermined angles variable
        angles[0] = 5f/6f*Mathf.PI; angles[1] = 3f/4f*Mathf.PI; angles[2] = 2f/3f*Mathf.PI;
        angles[3] = 7f/6f*Mathf.PI; angles[4] = 7f/6f*Mathf.PI; angles[5] = 5f/4f*Mathf.PI;
        angles[6] = 4f/3f*Mathf.PI; angles[7] = 5f/6f*Mathf.PI; angles[8] = Mathf.PI; angles[9]
        = Mathf.PI;

        //Initialize the starting time
        float t0 = Time.time;

        //Randomize the order of predefined angles that will determine the targets position
        suffleArrayAngles();
```

```
    for (int j=0; j<10; j++)
    {
        //If time is not over
        if(Time.time-t0 < Endtime)
        {
                //Define the position of the next target given the avatar's position.
                Vector3 position= new Vector3 (Avatar.transform.position.x
                +Mathf.Sin(angles[j])*Radio, Height, Avatar.transform.position.z +
                Mathf.Cos(angles[j])*Radio);

                //Instatiate the target at the given position
                RabbitScript rabbit = Instantiate(TargetPrefab, position, Quaternion.identity) as
                RabbitScript;
                rabbit.SetTimeLimit(timeLimit);


                //Wait some secs for the player to reach the target and then destroy the target
                yield return new WaitForSeconds(5f);
                Destroy(rabbit);
        }
    }

  yield return 0;

  //Stop the game
  Application.Quit();
  }

}
```

3. Finally, add the following function to the class:

```
        void suffleArrayAngles(){

          int num;
          float temp;

           for(int i=0; i<10;i++){
             num = Random.Range(0,9);
             temp = angles[i];
             angles[i] = angles[num];
             angles[num] = temp;
           }
        }
```

G. Connecting Variables

Connecting variables via the GUI is a very powerful feature of Unity. It allows variables that would normally be assigned in code to be done via drag and drop in the Unity GUI. This allows for quick and easy prototyping of ideas. As connecting variables is done via the Unity GUI, we always need to expose a variable in our script code so that we can assign the parameter in the Inspector

View.

If you look at the script provied in the last step, you will notice that in line 7 we are declaring a public variable called "TargetPrefab". Here we are exposing the variable in order to be added from the GUI.

1- Attach the script MainGame.cs to the surface object dragging and dropping it over the object in the Hierarchy window.

2- If you select it you will see the "TargetPrefab" variable and the "avatar" variable in the Inspector Window. No objects are attached to them.

3- With "Main" still selected, drag the "Rabbit" prefab from the Hierarchy View onto the "TargetPrefab" variable in the Inspector View. Also assign the avatar of the player (from the Hierarchy window) to the "Avatar" variable.

4- Run the scenario to observe the results.

H. Adding collision detection

If you run the game and tried to hit the rabbits appearing over the grass, you would have seen that no collision between the hands and the targets is being detected. In order to add collision detection, we need
Second, to detect the collision of the avatar's hand we will use a box collider and a few lines of scripting:

1. Create a sphere (Game Object), locate the spheres at the hands position in the Scene. All objects will collide with these spheres. Unselect MeshRender in the Inspector and add to it the component BoxCollider

2. Adjust the box collider to the size of the sphere from Inspector window.

3. Add a rigidBody component and select Is Kinematic option in the Inspector inside the BoxCollider component.

4. Duplicate the sphere. Then, in the Project window drag and drop the 2 spheres on HandLeft and HandRight of the avatar (RGS_Man/Untitled_Object/ J_Dorsal/ L_Collar/ L_Arm/ L_Forearm/ L_Hand in the Hirarchy window). Each of the spheres will be the children of each of the avatar's hands.

5. Add a new c# script to each cube. Study and copy the following script in it:

```
void OnTriggerEnter(Collider other) {
        Destroy(transform.gameObject);
}
```

6. Instantiate a rabbit object, add a BoxCollider to it and select isTrigger. Attach to it the Collision Script we just created.

7. Drag and drop the modified rabbit object from the hierarchy window to the prefab in the project window. Now we have modified the prefab and all

instances of this prefab will have attached a rigidBody component and a Box Collider.

If you want to learn more about colliders visit Unity Manual>Physics
Also give a look to Unity's Script Reference for the class Collision

Play with physics creating several objects that contain a BoxCollider and RigidBody component.

I. Making the Game more fancy: Adding Animations and Sounds

We have added some behavior to the rabbit prefab: RabbitScript.cs. It basically consists on triggering some animations and you will find the script in the rabbit prefab components.

1. Study it to understand what it is doing:

```
//------------Rabbit behavior---------------------
//------- BCBT September 2012 - SPECS -------------
//--------------RGS tutorial----------------------

using Unity.Engine;
using System.Collections;

public class RabbitScript : MonoBehaviour  {

	public static int score = 0;

	public Material starMaterial;
	public Material hitMaterial;

	private float timeOfBirth, timeLimit, timeVisible;
	private const float wiggleFreq = 3f;
	private const float popUpTime = 0.5f;

	private Transform wiggleBone;

	public bool isHit, isMissed;

	//private float origHeight;

	void Start () {
		animation.Play("rabbit_anim");
		timeOfBirth = Time.time;
		wiggleBone = transform.Find("Root/Spine1");


	}

	// Update is called once per frame
	void Update () {

		timeVisible = Time.time - (timeOfBirth + popUpTime);
		wiggleBone.localPosition = new Vector3(-0.35f + 0.95f * (timeVisible / timeLimit),
wiggleBone.localPosition.y, wiggleBone.localPosition.z );
		Vector3 rot = wiggleBone.rotation.eulerAngles;
		rot.z = 270f + Mathf.Sin(timeVisible*Mathf.PI*wiggleFreq) * 15f;
		wiggleBone.rotation = Quaternion.Euler(rot);

	}

	public void Die(){
```

```
                float amount = Mathf.Min(1f, 1f - timeVisible/timeLimit);

                Debug.Log("time: " + timeVisible + "  timelimit: " + timeLimit + "  amount: " +
amount);

                score += 10 + (int) (40 * amount);
                ParticleRenderer pr = GetComponent<ParticleRenderer>();
                pr.material = starMaterial;
                particleEmitter.Emit(10 + (int) (40 * amount));

                Renderer bunnyRenderer = transform.Find("RabbitMesh").renderer;
                bunnyRenderer.sharedMaterial = hitMaterial;
                animation.Play("rabbit_death", PlayMode.StopAll);

        }

        public void SetTimeLimit(float t){
                timeLimit = t - popUpTime;
        }

}
```

2. To add sounds you will need to call the next function from MainGame.cs:

```
target.audio.PlayOneShot(yourSound1);
```

Don't forget to declare the AudioClip of your sound in the beginning of the file:

```
public AudioClip[] yourSound1, yourSound2, yourSound3;
```

Since we are using a public variable they will be visible in the inspector window. You will need to drag and drop the corresponding audio file from the Project Window to the Inspector window. If you don't do this, Unity will not know which audio file you are referring to in those variables. Check Unity's Script Reference if you want to learn more about additional functions and classes.

J. Lighting the scene

There are three different types of Lighting within Unity: Point, Spot and Directional.

To light our map we'll use a Directional light so go ahead and add one to the scene from the top menu by selecting GameObject -> Create Other -> Directional Light.

You may notice a difference as soon as the light is added depending on its direction but reposition it using the Transform and Rotation tools to light your scene properly. It may help to use the Orthographic views, particularly TOP to position and SIDE to rotate as shown above.

Hopefully now when you play the scene is lit a bit better than before.
Play a bit with lights.

K. Testing the game

Play with the different features of Unity and explore its interface. Try to customize the scenario you just build adding more components, such as a TextGUI showing the score.