AD41700 Computer Games
Prof. Fabian Winkler
Fall 2013

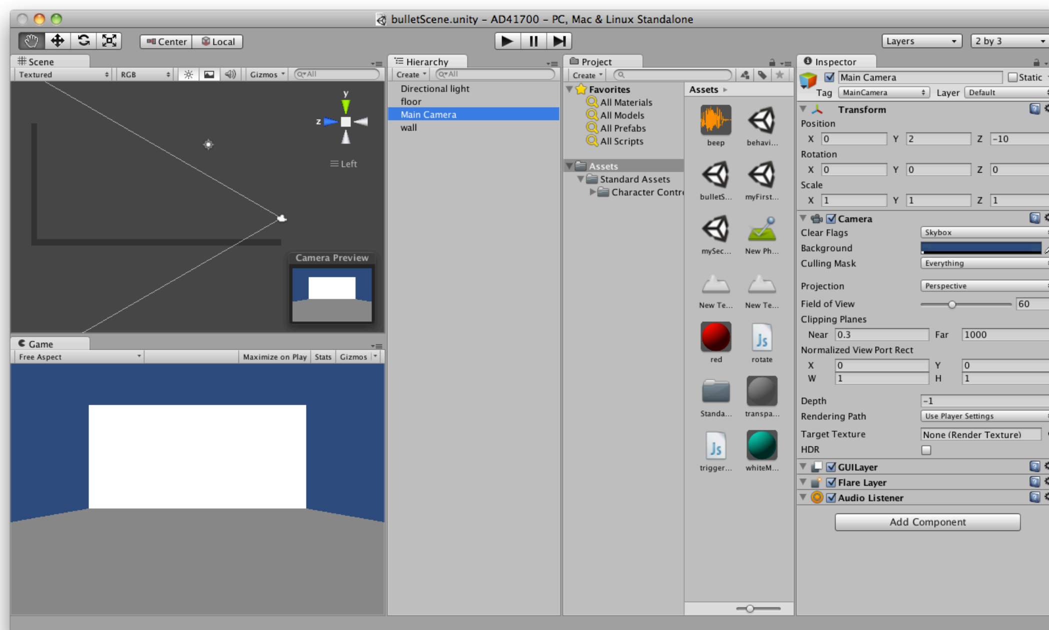
Creating Bullets in Unity3D (vers. 4.2)

I would like to preface this workshop with Celia Pearce's essay *Beyond Shoot Your Friends* (download from: http://www.gardensandmachines.com/AD41700/Readings_F13/pearce2_pass.pdf) and the idea that shooting at things in computer games can be used for so many more things than to kill, destroy or 'shoot your friends.'

This is also the premise of this workshop, to challenge the meaning and use of shooting, specifically in the loaded environment of the First Person Shooter game and to experiment with this game mechanic in more imaginative, nuanced, engaging and meaningful ways.

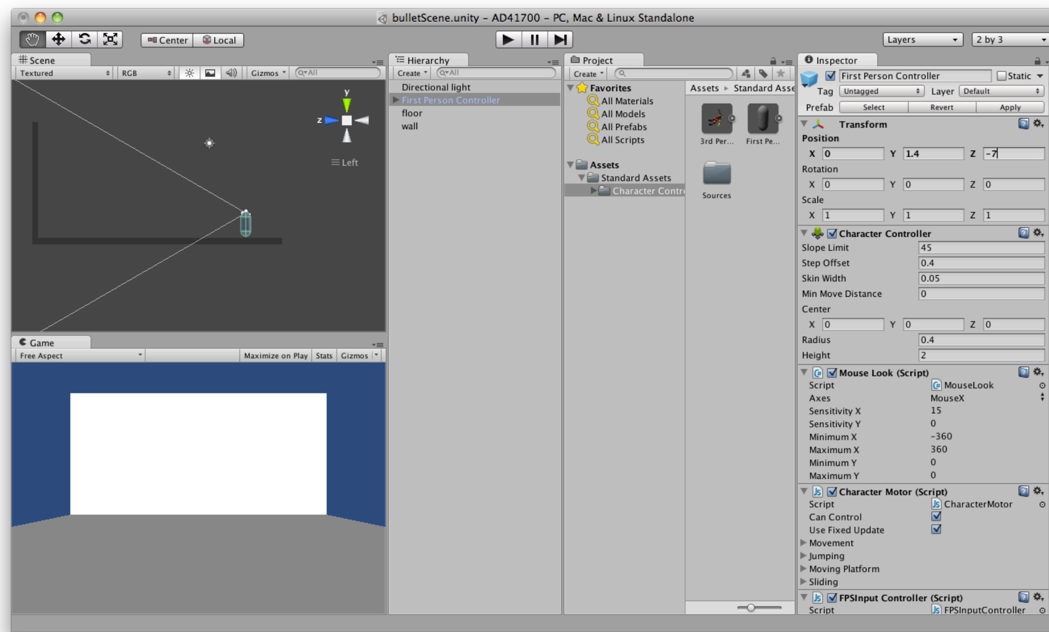
Creating Bullets Using Prefabs

We start by creating a floor and a wall in a new scene, like this:



In the next step we place a first person controller in the scene, so we can move around

freely (don’t forget to delete the main camera after placing the first person controller). If you don’t have the Character Controller Package already imported into your project (from previous workshops) import it now by going to: `Assets > Import Package > Character Controllers`.



Creating Custom Prefabs

We will use the concept of Prefabs, reusable assets, for creating bullets on the fly.

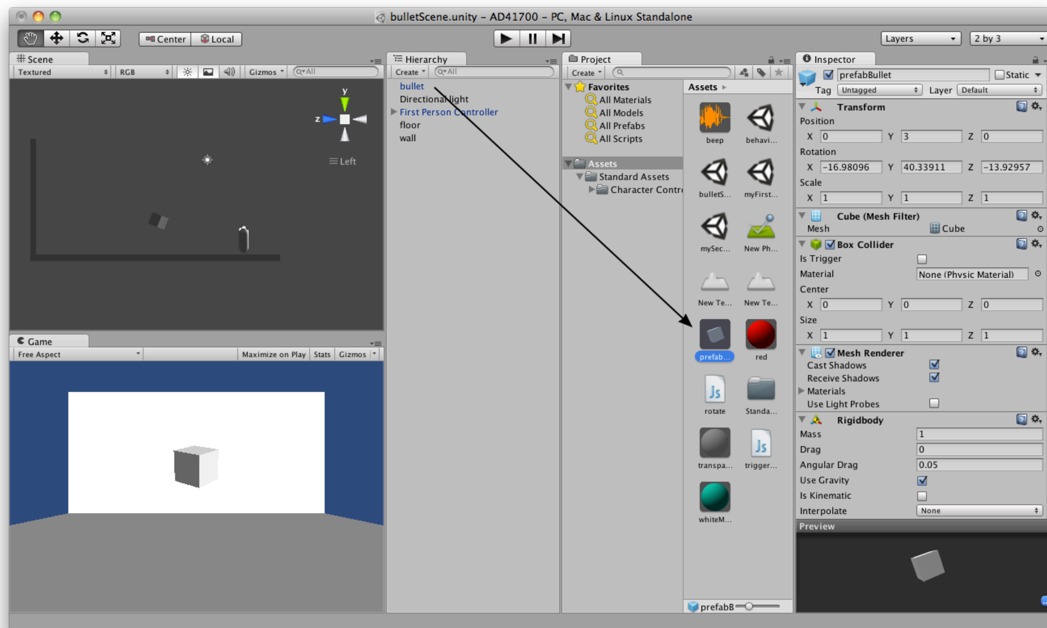
I create a simple cube (which we will use as the projectile/bullet later) and turn it into a prefab:

`Game Object > Create Other > Cube…`

Attach a rigid body component to the cube. While it is selected in the Hierarchy window got to: `Component > Physics > RigidBody`

I rename the cube game object in the Hierarchy window to “bullet” and I also create a Prefab called “prefabBullet” (you can name these things whatever you want):
`Asset > Create > Prefab`

The Prefab shows up in the Project window. I just need to drag and drop the bullet game object from the Hierarchy window onto the bullet prefab in the Project window to assign it.



Since all the data about the bullet game object is now in the bullet prefab in the Project window, you can delete the bullet game object in the Hierarchy window.

Now, I create a JavaScript which, upon being triggered, creates new instances of the bullet prefab:

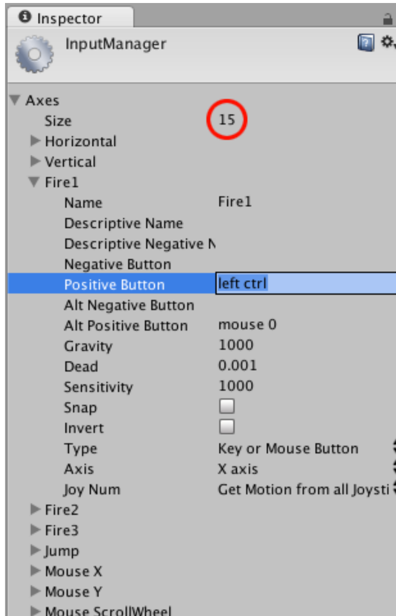
```
var prefabBullet : Transform;

function Update()
{
      if (Input.GetButtonDown("Fire1")) // this is left ctrl.
      {
       var instanceBullet = Instantiate(prefabBullet,
       transform.position, Quaternion.identity);
      }
}
```

Here is some more background on the Instantiate function:
http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html

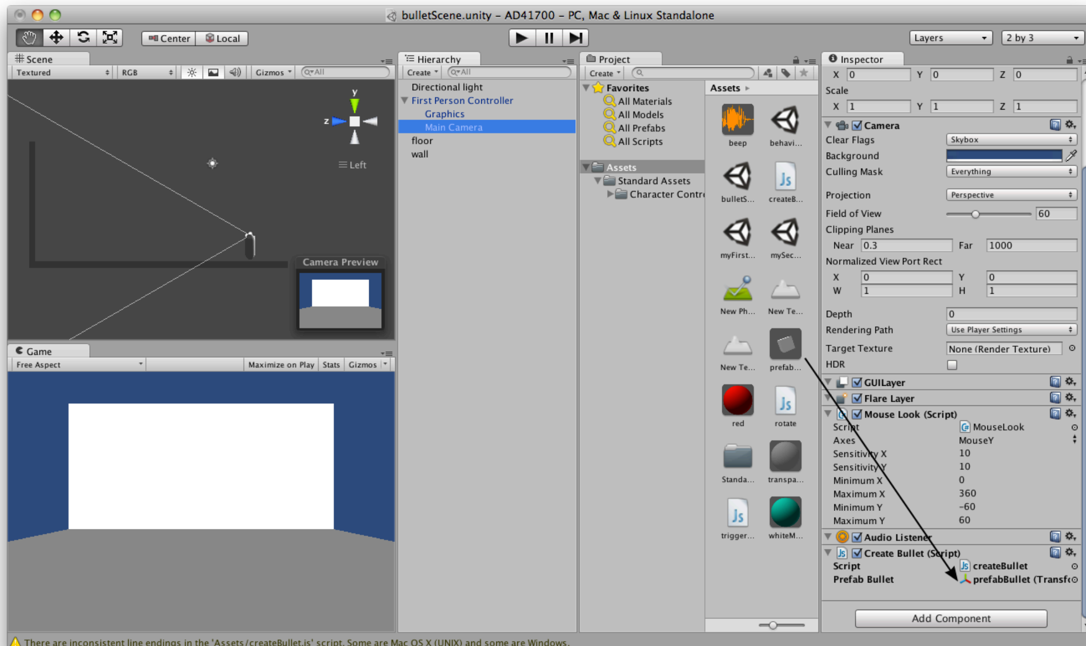
The trigger for creating new instances of the bullet prefab is the “Fire1” button. You can find out about which key this button is mapped to by going to the project settings: `Edit > Project Settings > Input` expanding the properties of Fire1 in the list in the Inspector gives you information about the assigned keys. It also allows you to change keys associated with certain input types, e.g. for moving around, firing, jumping, etc…

You can also add additional inputs (key or joystick-based) by increasing the size of the Input Manager’s Input Axes (in the Inspector under Axes > Size). By default these new properties are created as “jump” keys, just open up each of them and give them new

names and change their positive button value to the key you would like to assign to each of them.

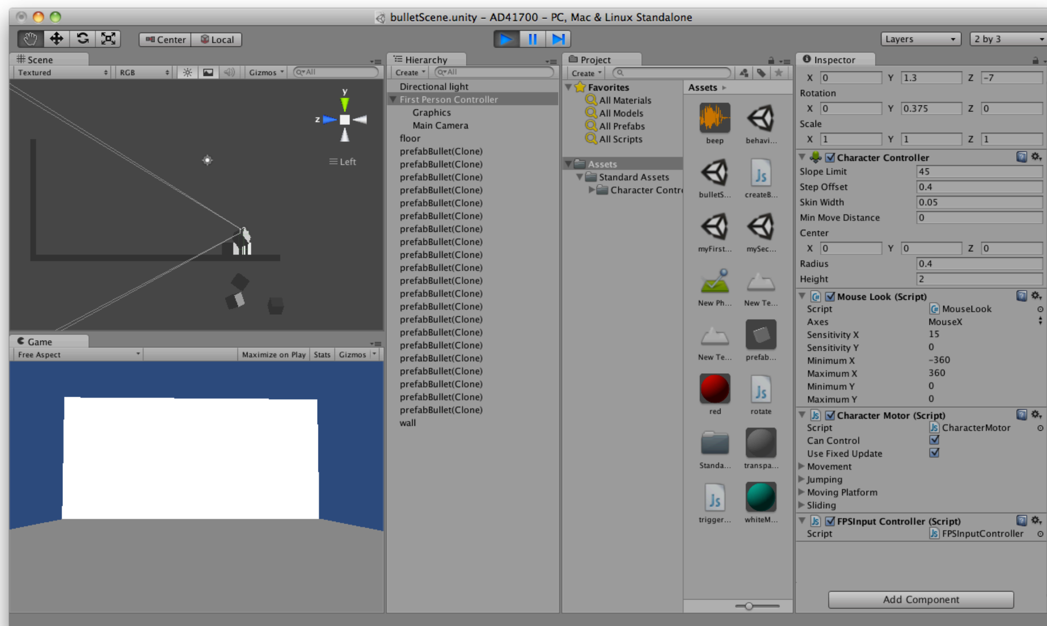


Back to the createBullet script, I attach it to the first person controller and make sure that the prefab “prefabBullet” is assigned to the prefabBullet variable in the script (I assign it in the Inspector): drag and drop the bullet prefab from the Project window onto the prefabBullet variable in the Inspector:



Now you can already try out the new script and you’ll see that cubes are being created in the position of the camera. Because they have no forward pointing force attached to them they simply stay in place and are pushed away by new instances of the prefab.

You also see in the Hierarchy window how all the instances of the prefab bullet are listed as you create them:



In order for theses bullets to have a direction in which they are moving, we need to add a forward pointing force to the script:

```
var prefabBullet : Transform;
var forwardForce = 1000;

function Update()
{
 if (Input.GetButtonDown("Fire1"))
 {
  var instanceBullet = Instantiate (prefabBullet, transform.position,
  Quaternion.identity);

  instanceBullet.rigidbody.AddForce(transform.forward *
  forwardForce);
 }
}
```

We are ready to try out the script and should see the cubes shooting out forward from the camera position rather than just dropping down. Also the initial angle of the newly created bullets is dependent on the viewing angle of the camera (i.e. looking up and shooting makes the bullets fly up).

Here is a short video from the official Unity3D tutorials that also talks about instantiation:
http://unity3d.com/learn/tutorials/modules/beginner/scripting/instantiate

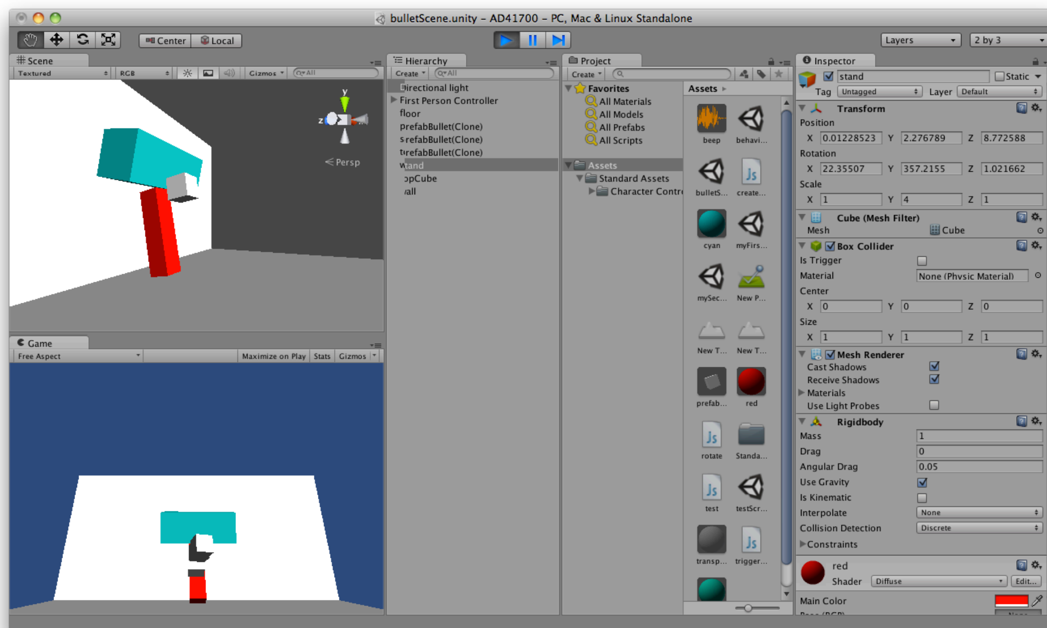
Of course you can also create your own custom trajectories by playing around with the vector of the AddForce function, like this:

```
var prefabBullet : Transform;
var forwardForce = 1000;
var upwardForce = 500;


function Update()
{
 if (Input.GetButtonDown("Fire1"))
 {
  var instanceBullet = Instantiate (prefabBullet, transform.position,
  Quaternion.identity);

  instanceBullet.rigidbody.AddForce(0, upwardForce,
  forwardForce);
 }
}
```

The Unity3D website has a very good primer tutorial in vector math, which you can watch here: http://unity3d.com/learn/tutorials/modules/scripting/lessons/vector-maths-dot-cross-products

Detecting Collision between Bullets and Gameobjects

I added two more cubes to the previous scene – the red one called "stand", the turquoise one called "top_cube." Both new game objects have rigid body components added to them. The idea is that when one of the bullets is hitting them that they would fall to the floor, like this:



Next we would like to detect the collision between the bullet and the red "stand". We can approach this in two ways:

1) from the perspective of the red stand – i.e. do something whenever something hits the stand.
2) from the perspective of the bullet – i.e. do something whenever the bullet hits something.

1) Collision with the stand

I can write the the following script ("collStand") using Unity's built-in *OnCollisionEnter* function and attach it to the stand:

```
private var numberOfHits : int = 0;

function OnCollisionEnter(theCollision : Collision)
{
        numberOfHits++;
        print("The stand has been hit " + numberOfHits + " times");

        // You can use this line to figure out which game object
        // the stand collided with (just uncomment to try out):
        // print(theCollision.gameObject.name);
}
```

Now the counter variable "numberOfHits" counts every collision that occurs with the stand (it does not need to be from a bullet, e.g. it could also be from the turquoise box sitting on top of the stand, or the stand hitting the wall behind it). If you uncomment the last line of code, you can figure out which game object specifically the stand collided with: `theCollision.gameObject.name`

Let's trigger a sound this time when we hit the stand. I prepared a sound file called "bang.aiff" (remember Unity only likes uncompressed sound – .aiff or .wav or ogg/vobis. Import the sound file into your Project window:

`Asset > New Asset…`

You can leave the 3D sound option checked in the Inspector, this will make the metal bang sound more convincingly coming from the stand when you hit it (i.e. louder when you are closer and quieter when farther away from it).

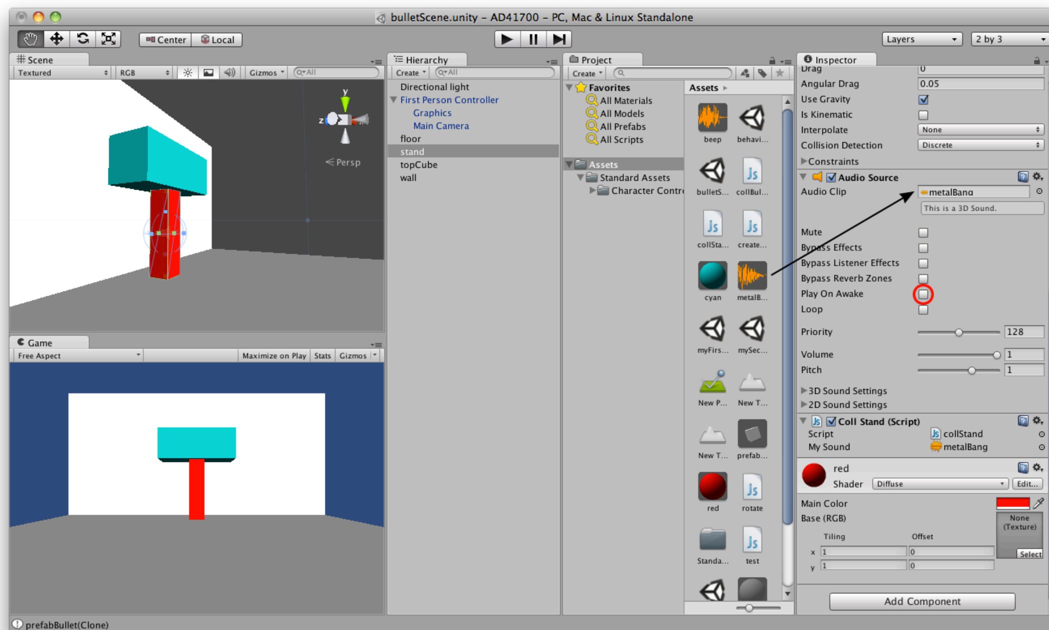
I change the " collStand" script to include two lines that trigger the audio playback:

```
var mySound : AudioClip;

private var numberOfHits : int = 0;

function OnCollisionEnter(theCollision : Collision)
{
        numberOfHits++;
        print("The stand has been hit " + numberOfHits + " times");

        audio.PlayOneShot(mySound);

        // You can use this line to figure out which game object
        // the stand collided with (just uncomment to try out):
        // print(theCollision.gameObject.name);
}
```

Don’t forget to drag and drop the sound asset onto the variable “mysound” in the Inspector after selecting the stand, so the correct sound is assigned to the script.

Finally, also attach an audiosource component to the stand, so the sound can be played in the scene `Component > Audio > AudioSource`.

In the Inspector, choose “bang” as the Audio Clip and uncheck “Play On Awake”:

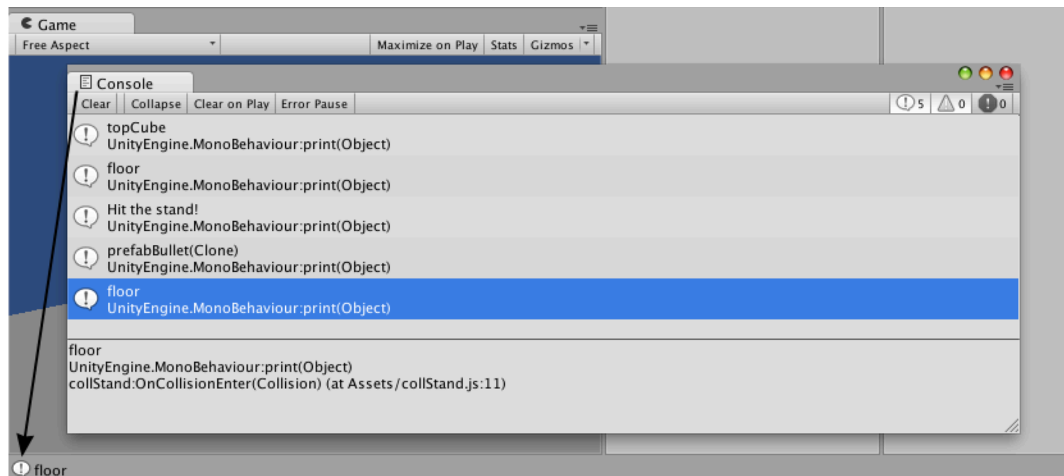


2) Collision with the Bullet

This works pretty much the same way as with the stand. We can write another script “collBullet” and attach it to the prefab – in this case to know if the bullet really hit the stand:

```
function OnCollisionEnter(theCollision : Collision)
{
        if (theCollision.gameObject.name == “stand”) {
        print("I hit the stand!");
      }
}
```

Open up the Console window to see all your text feedback from the interactions with the game objects in your scene:



Housecleaning

Now that we know how to create game objects dynamically at runtime it is a good idea to think about removing them eventually as well. With the scripting experience you have gained so far you can figure out different events that might remove the cube bullets from your scene (e.g. when they collide with another game object), in the following script ("destroyBullets"), I use a timer that automatically removes (destroys) the game objects after a certain amount of time:

```
var timeRemaining = 3.0;

function Update()
{
	timeRemaining -= Time.deltaTime;

	if (timeRemaining <= 0.0)
	{
		Destroy(gameObject);
	}
}
```

Simply attach this script to the bullet prefab and watch the cubes disappear after 3 seconds.

Hit Counter with Onscreen Text

Since some of the information in the Console window might be helpful for a player as well, I setup a very simple GUI (on screen text) that displays the number of hits the stand has taken.
The Unity Reference Manual has a nice introduction to UnityGUI, an easy to use set of commands that produce GUI controls. The following examples show some of its functionality. For a full overview go to:
http://docs.unity3d.com/Documentation/Components/GUIScriptingGuide.html

In order to create an onscreen text with the UnityGUI we need to create an empty game object and then attach the script below.

```
GameObject > Create Empty
```

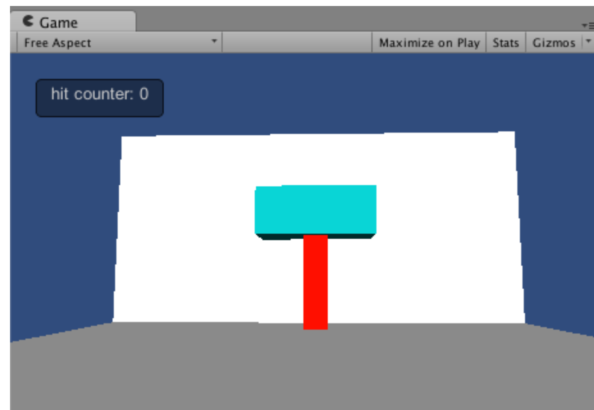
This script, which I titled "guiScript" initially just writes some text into a semitransparent text box.

```
var hitCounter : int = 0;

function OnGUI () {

      // Make a background box with text
      GUI.Box (Rect (10,10,100,90), "hit counter: " + hitCounter);
}
```

You can make this GUI appear and disappear by enabling or disabling the empty game object's GUIScript component. This is because the OnGUI() function works very similar to an Update() function in that it is being executed every frame of the game for as long as its script is enabled. You can toggle the guiScript's enable property using the following script that you should also attach to the very same empty game object (called "enableGUI"):

```
var targetScript : guiScript;

function Start() {
targetScript = GetComponent(guiScript);
targetScript.enabled = false;
}

function Update () {

 // If the 'g' button is pressed:
 if(Input.GetKeyDown(KeyCode.G)) {
      if(targetScript.enabled == false) {
          targetScript.enabled = true;
      } else {
            targetScript.enabled = false;
      }
 }
}
```

Now, pushing the ‘g’ button on the keyboard toggles the GUI on and off.



In general, the syntax for accessing public variables, properties and functions in another script attached to the same game object goes like this:

```
var script : ScriptName;
script = GetComponent("ScriptName");
script.DoSomething ();
```

See also:
http://docs.unity3d.com/Documentation/ScriptReference/Component.GetComponent.html

Accessing Variables from Other Scripts

Because I would like to update the guiScript’s hitCounter variable from within the collStand script (the one attached to the stand game object, see p. 7) I need to add a couple of lines of code to the collStand script, like this:

```
var mySound : AudioClip;
//private var numberOfHits : int = 0;
private var otherScript : guiScript;

function Start(){
      otherScript = GameObject.Find("GUI").GetComponent(guiScript);
}


function OnCollisionEnter(theCollision : Collision)
{

        audio.PlayOneShot(mySound);

        otherScript.hitCounter++;

        // You can use this line to figure out which game object
        // the stand collided with (just uncomment to try out):
        // print(theCollision.gameObject.name);
}
```

Now we can easily access the "hitCounter" variable in the "guiScript" from within the "collStand" script. For this purpose, we made use of the `GameObject.Find()` function which allows you to find game objects by their name (i.e. the name that you give them in a scene) and then help you access their components using the `GetComponent` function (such as scripts, sound, rigidbodies, renderer/materials, etc.), more documentation about the `GameObject.Find()` function is here:
http://docs.unity3d.com/Documentation/ScriptReference/GameObject.Find.html

And this is what the final scene looks like with the GUI counter: