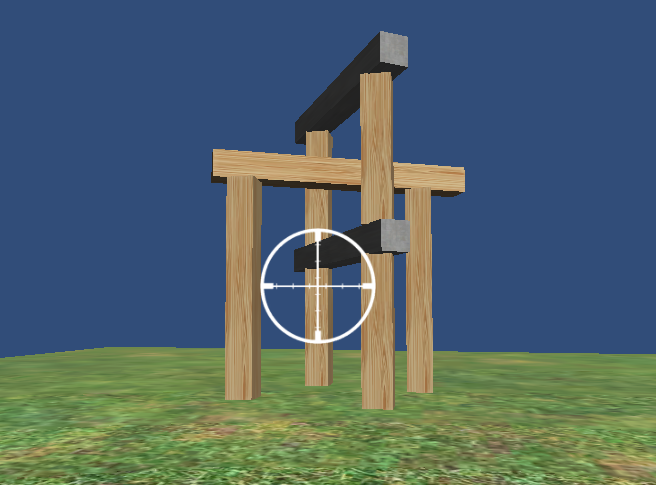
**TEMASEK POLYTECHNIC**

**SCHOOL OF INFORMATICS & IT**

**DIPLOMA IN GAME DESIGN & DEVELOPMENT**

To see the additional comments and resources, make sure you select **All Markup** in the **Review/Tracking** pane



*These are extended notes to the tutorial at*:

<https://code.tutsplus.com/tutorials/working-with-unity3d-physics--mobile-11771>

*It’s from 2012, but the physics is the same, and the code has been updated where necessary.*

In this tutorial, we’ll use the physics engine in Unity to build a 3D knockdown game similar to [BoomBlox](https://youtu.be/TV7zFHl2fX4?t=22) and [Angry Birds](https://youtu.be/2BqfjGDsHUs?t=31).

We’ll learn how to give objects different physical properties, make them capable of colliding, and even allow them to be destroyed if the collisions are strong enough.

**Project Setup**

* Create a new project called **CannonBall**.
* Create a directional light.
* Create four folders inside the Assets folder:   
   Scenes  
   Scripts  
   Prefabs  
   Images
* Save the scene in the Scenes folder as **Main.scene**.

**Colliders**

Colliders are what a physics engine uses to perform hit detection.

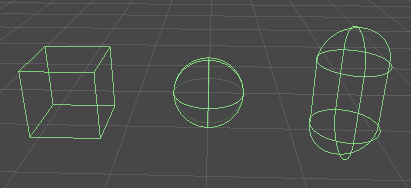
Unlike mesh objects, they know when they’ve come in contact with each other.

They are simple shapes like boxes, spheres, or capsules that are assigned to your GameObjects and follow them around.

Conveniently, whenever a GameObject is created, it is assigned an appropriate collider.

Primitive game objects (cube, sphere, etc.) will be assigned the appropriate primitive collider. More complex meshes are assigned a mesh collider.

A Cube gets a BoxCollider, a Sphere gets a SphereCollider, a Cylinder gets a CapsuleCollider, and so on.



We’ll eventually need some blocks to knock down:

* Create a Cube.
* Rename the Cube to **WoodBlock**.
* Set the Block’s position to 0, 0, 0. This will center the Block in the world.
* Download [Wood.jpg](https://cdn.tutsplus.com/mobile/authors/legacy/Ian%20Zamojc/2012/08/06/Wood.jpg) *(available on LMS)* to the Images folder. This will make it a texture.
* Drag the Wood texture onto the Block in the Scene View to automatically create a Wood material and apply it to the Block.
* Drag the Block from the Hierarchy panel into the Prefabs folder to turn it into a Prefab.

If you press Play, the block won’t do anything.

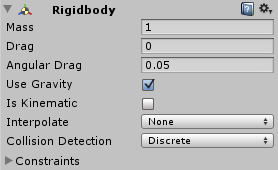
Though it has a collider, it lacks a rigidbody, so isn’t affected by any physical forces.

**Rigidbodies**

A rigidbody is the the most critical element in a physics engine.

Any GameObject the rigidbody is attached to is included in the physics simulation.

* Select the Block prefab in the Project panel.
* Click Component > Physics > Rigidbody.



By default, a rigidbody is affected by gravity and air resistance, also known as [drag](https://www.real-world-physics-problems.com/drag-force.html).

If we press Play, the block will start to fall, accelerate, and eventually hit [terminal velocity](https://www.youtube.com/watch?v=pr78Db9qqeI) when the force of gravity and drag become equal.

**Build a Structure**

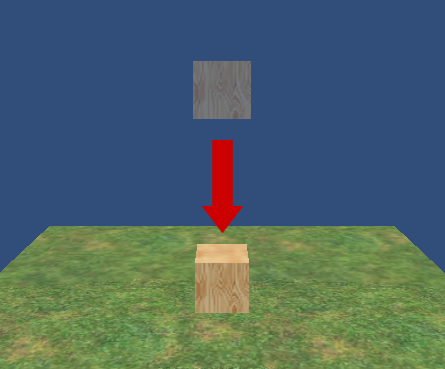
We’ll need to to create a few more elements to build a proper level.

First, add some ground so the the block has something to land on.

* Create a Plane.
* Rename the Plane to **Ground.**
* Set the Ground’s position to 0, 0, 0.
* Download [Grass.jpg](https://cdn.tutsplus.com/mobile/authors/legacy/Ian%20Zamojc/2012/08/06/Grass.jpg) to the Images folder. This will make it a texture.
* Drag the Grass texture onto the Ground in the Scene View.

The Ground will automatically be given a MeshCollider which will prevent any rigidbodies from passing through it.

Press Play and the Block should fall and settle on top of the Ground.



Now we need a structure to knock down.

Select the Block and press **Ctrl+D** in Windows, or **Cmd+D** in OSX, to duplicate the Block a few times.

Use the scale and move tools to stretch and position the blocks in roughly the same configuration as the picture below.



**NOTE:**  It’s a good idea to use precise numbers for transformations.

Blocks should rest against each other, but not overlap. Overlaps will cause the physics engine to freak out and do unpredictable things.

**Camera Controls**

Now that we’ve created our structure, let’s write a script that will allow us to move the camera so we can admire our creation from all angles.

* Create a new C# script in the Scripts folder.
* Rename the script **Cannon** (because eventually our camera will be doing the shooting).
* Drag the script onto the Main Camera.
* Double click the script to edit it.

This will cause the camera to orbit the center of the world, **as well as tilt up and down:**

public class Cannon : MonoBehaviour

{

public GameObject projectilePrefab;

void LateUpdate()

{

float x = Input.GetAxis("Mouse X") \* 2;

float y = -Input.GetAxis("Mouse Y");

// vertical tilting

float yClamped = transform.eulerAngles.x + y;

transform.rotation = Quaternion.Euler(

yClamped,

transform.eulerAngles.y,

transform.eulerAngles.z);

// horizontal orbiting

transform.RotateAround(

new Vector3(0, 3, 0),

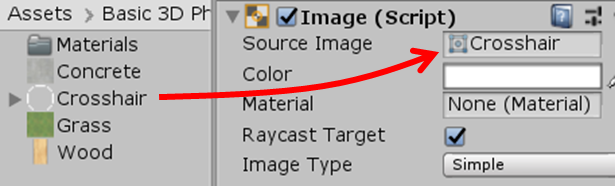
Vector3.up, x);

}

}

Now add a crosshair to the camera.

* Download [Crosshair.png](https://cdn.tutsplus.com/mobile/authors/legacy/Ian%20Zamojc/2012/08/06/Crosshair.png) to the Images folder.
* Click on Crosshair.png in the Images folder, then in the Inspector sets its texture type to **Sprite (2D and UI)**.
* Add a **Canvas** to the scene.
* Right click on the Canvas object in the Hierarchy, select **UI** from the dropdown menu, then select **Image**. A white square (the image you just created) should appear at the center of the screen.
* Rename the Image object to **Crosshair**.
* Click on Crosshair, then drag Crosshair.png to the **Source Image** property of the Crosshair object’s Image script.



You should now have a crosshair at the center of your game view.



**Shooting Cannonballs**

Now, we need to knock the blocks down to see the physics in action.

* Create a Sphere.
* Rename the Sphere to **Cannonball**.
* Set the Cannonball’s position to 0, 0, 0.
* With the Cannonball selected, click Component > Physics > Rigidbody.
* Drag Cannonball from the Hierarchy panel to the Prefabs folder to turn it into a Prefab.

Since we’re going to be shooting cannonballs directly from the camera, we can edit the existing Cannon script.

First, add a public attribute for the projectile prefab.

public class Cannon : MonoBehaviour

{

public GameObject projectilePrefab;

When the “[Fire1](https://answers.unity.com/questions/957016/space-shooter-tutorial-where-is-fire1.html)” button is pressed, a Cannonball prefab must be instantiated, positioned at the camera, and then moved forward by adding a force to it.

Inside FixedUpdate, call the rigidbody’s [AddRelativeForce](https://docs.unity3d.com/ScriptReference/Rigidbody.AddRelativeForce.html) function to apply the force to the cannonball.

Before applying a force to the cannonball, watch this [video](https://www.youtube.com/watch?v=_LdcxCdB-s8) for a brief explanation of what a force actually is.

void FixedUpdate()

{

if (Input.[GetButtonDown](https://docs.unity3d.com/ScriptReference/Input.GetButtonDown.html)("Fire1"))

{

GameObject projectile = [Instantiate](https://learn.unity.com/tutorial/instantiate)(

projectilePrefab,

transform.position,

transform.rotation);

Rigidbody rb = projectile.GetComponent<Rigidbody>();

rb.AddRelativeForce(new Vector3(0, 0, 2000));

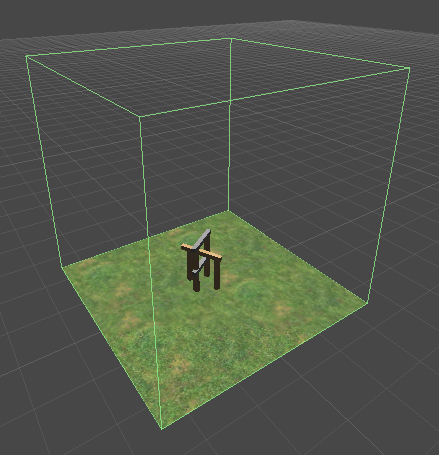
}

}

**Boundaries**

You may have noticed that if a cannonball is fired far enough, it can fall off the edge of the ground plane. This cannonball will continue to exist as long as the game keeps running and its physics will continue to be calculated, eventually slowing things down. We need to create a boundary around the level and destroy any game objects that leave this boundary.

* Create an empty GameObject.
* Rename it to **Boundary**.
* Set the Boundary’s x, y, and z position to 0.
* With the Boundary selected, click Component > Physics > Box Collider.
* In the inspector, make sure **IsTrigger** is checked.
* Set the Box Collider’s Center to 0, 25, 0, or similar to below.
* Set the Box Collider’s Size to 50, 50, 50, or similar to below.



Now we need to create the script that will destroy any objects that stray outside the boundary.

* Create a new C# script in the Scripts folder.
* Rename the script to **Boundary**.
* Drag the script onto the Boundary object in the Hierarchy panel.
* Edit the script and add the code below.

public class Boundary : MonoBehaviour

{

void OnTriggerExit(Collider other)

{

Destroy(other.gameObject);

}

}

**Causing Destruction**

We need a way to win our level. To do this, our blocks need to be destroyed if they take enough damage.

* Create a new C# script in the Scripts folder.
* Rename the script to **Block**.
* Drag the script onto the Block prefab in the Prefabs folder.
* Double click the script in the Project panel to edit it.

In the script, we give the prefab a public health property which can be adjusted in the editor. This allows different blocks to have different amounts of health.

public class Block : MonoBehaviour

{

public float health = 20;

}

When a collision is detected, the magnitude of the impact is measured. The greater the magnitude, the more damage is done. Anything above a light tap is subtracted from the block’s health.

If the block’s health drops below 0, the block destroys itself. It then checks to see how many other blocks are remaining in the scene. If there is only one block left, the game is over and it reloads the scene to play again.

void OnCollisionEnter(Collision collision)

{

// apply collision damage

if (collision.relativeVelocity.magnitude > 0.5)

{

health -= collision.relativeVelocity.magnitude;

}

if (health <= 0) // destroy if health is too low

{

Destroy(gameObject);

// restart the scene if this was the last box

GameObject[] boxes =

GameObject.FindGameObjectsWithTag("Box");

if (boxes.Length <= 1)

{

SceneManager.LoadScene("Main");

}

}

}

**Concrete Blocks**

Wooden blocks are light and relatively weak, making the structure too easy to destroy. We need to create another type of block, one that’s both heavier and stronger.

* In the Project panel, duplicate the WoodBlock prefab (**Ctrl+D** in Windows, **Cmd+D** in OSX).
* Rename the duplicate to **ConcreteBlock**.
* Download [Concrete.jpg](https://cdn.tutsplus.com/mobile/authors/legacy/Ian%20Zamojc/2012/08/06/Concrete.jpg) *(available on LMS)* to the Images folder
* Drag Concrete.jpg into the Images folder to make it a texture.
* Drag the Concrete texture on to the ConcreteBlock prefab in the Project panel.
* With the prefab selected, use the Inspector to update the Health to 50.
* In the Rigidbody component, increase the Mass to 5 to make it heavier.
* Drag the ConcreteBlock prefab into the Scene.

Try replacing some of the cross members with concrete blocks. The concrete blocks should be harder to knock over, fall with greater impact, and be harder to destroy with cannonballs.

