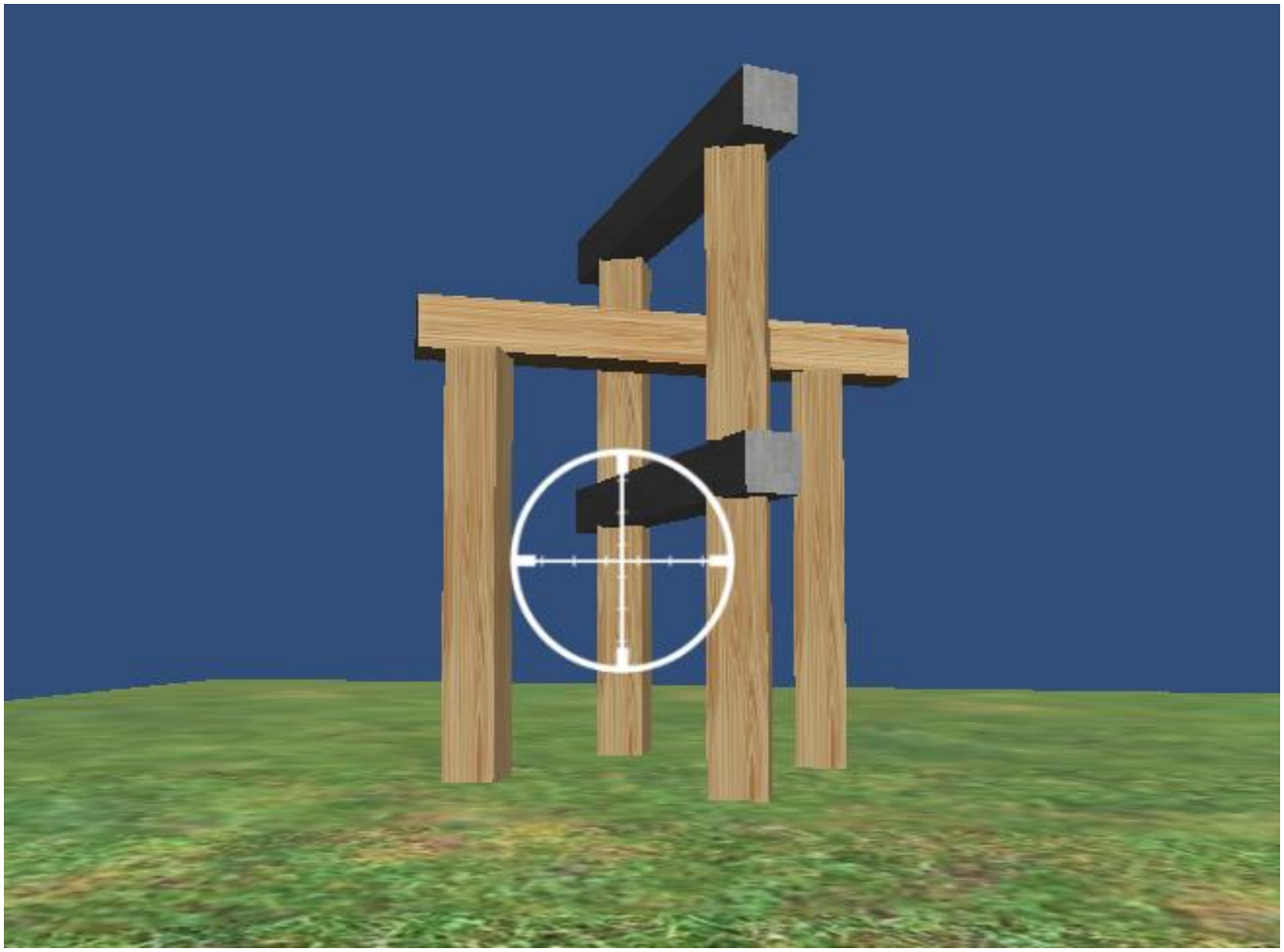
Working with Unity3D Physics

This tutorial will teach you how to build a knock down game with Unity3D! Along the way, you'll learn about the importance of using a physics engine and how doing so will save countless hours of manual animation. Read on!

What modern game engine would be complete without a physics engine? Every current game engine, be it 3D or 2D, has a physics library of some sort, and Unity is no exception. Realtime physics are ideal for simulating complex interactions between objects in your game. A physics engine can save a lot of the manual coding and animation to achieve realistic movement, can make performing hit detection a breeze, and can quickly introduce a host of new gameplay mechanics to your games.

In this tutorial, we'll use the physics engine in Unity to build a 3D knockdown game similar to BoomBlox and Angry Birds. 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.



What is a Physics Engine?

A physics engine works by simulating how objects react with each other when forces are applied to them. These forces can be constant, like gravity or the momentum of a vehicle, while others are brief and powerful, like explosions. A physics simulation is sometimes called a “sandbox” because only objects within the simulation are impacted. Indeed, not every object in your game needs to be part of the simulation. This is important since player movements often need to be unrealistic while still responding realistically to collisions.

Project Setup

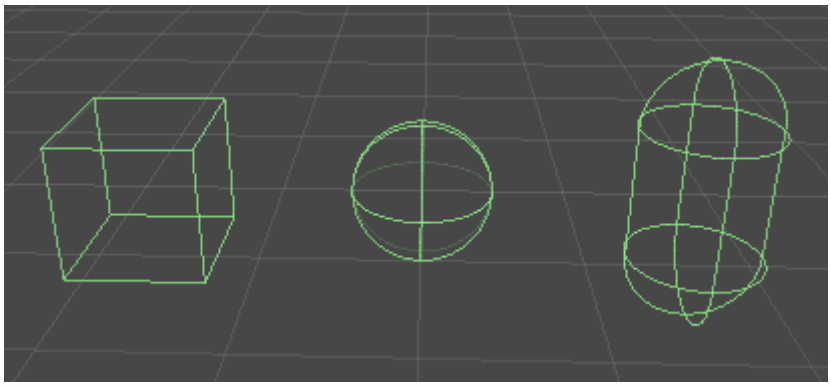
- Click File > New Project
- Browse to a suitable folder and name the project CannonBowl
- Click Create

- Click GameObject > Create Other > Directional Light
- Click File > Save Scene
- Name the scene **Main**
- Click Save

Colliders

Colliders are what physics engines use 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. You can think of them like something of a "force field".

Conveniently, whenever a GameObject is created, it is automatically assigned an appropriate 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:

- Click GameObject > Create Other > 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
- Drag Wood.jpg into your Project panel to make it a texture
- Drag the Wood texture onto the Block in the Scene View, this will automatically create a Wood material for us and apply it to the Block

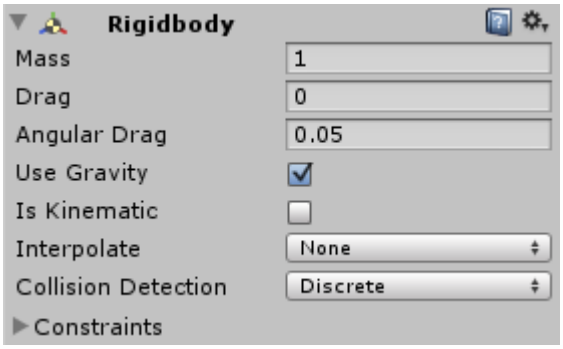
- Drag the Block from the Hierarchy panel into the Project panel to turn it into a Prefab

If we press Play, the block won't do anything. While it has a collider, it lacks a rigidbody, so it isn't affected by any physical forces.

Rigidbodies

A rigidbody is the the most critical element in a physics engine. Any GameObject it is attached to is included in the 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. If we press Play, the block will start to fall, accelerate, and eventually hit terminal velocity when the force of gravity and drag equalize.

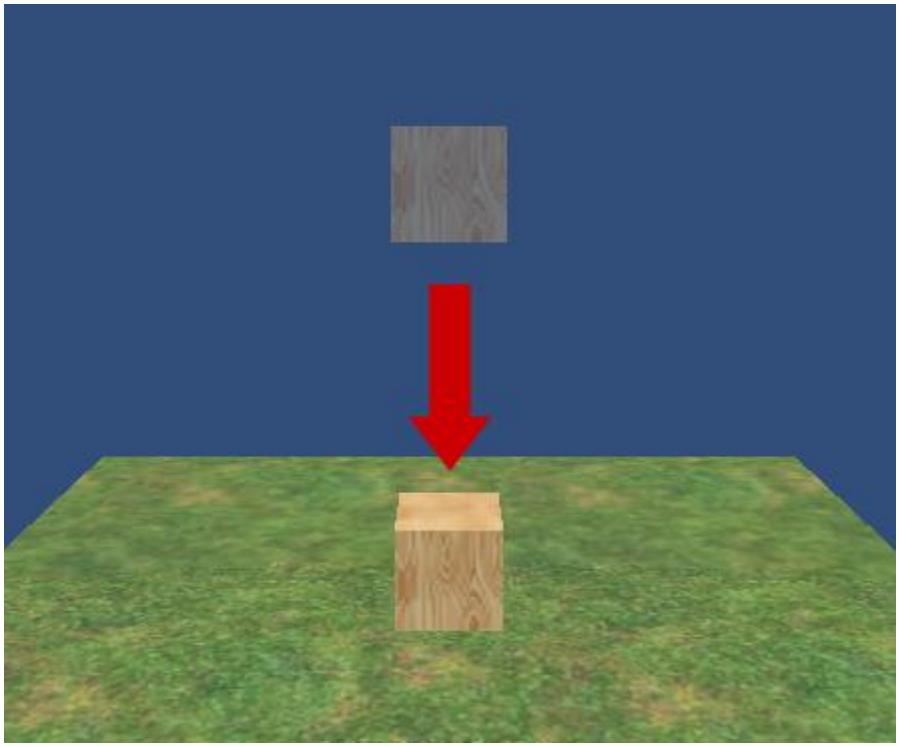
Build a Structure

We'll need to to create a few more elements in order to build a proper level. First, let's add some ground so the the block has something to land on.

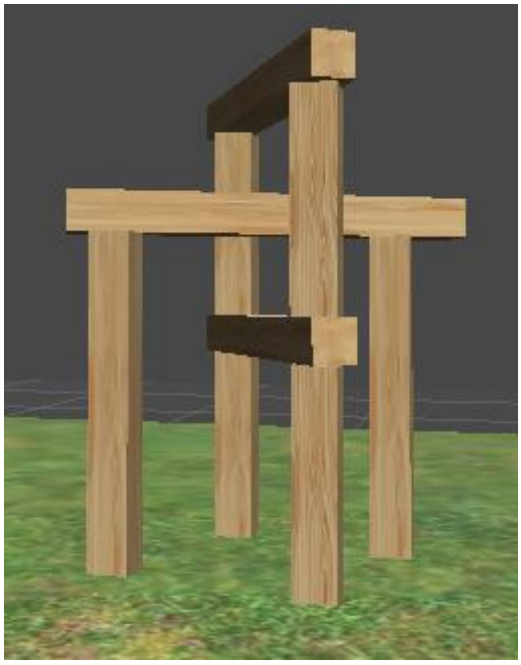
- Click GameObject > Create Other > Plane
- Rename the Plane to **Ground**
- Set the Ground's position to 0, 0, 0
- Download Grass.jpg

- Drag Grass.jpg into your Project panel to 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 your 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 beautiful structure, let's write a script that will allow us to move the camera so we can admire our creation from all angles.

- Click Assets > Create > C# Script
- Rename the script to **Cannon** (because eventually our camera will be doing the shooting)
- Drag the script onto the Main Camera
- Double click the script to edit it

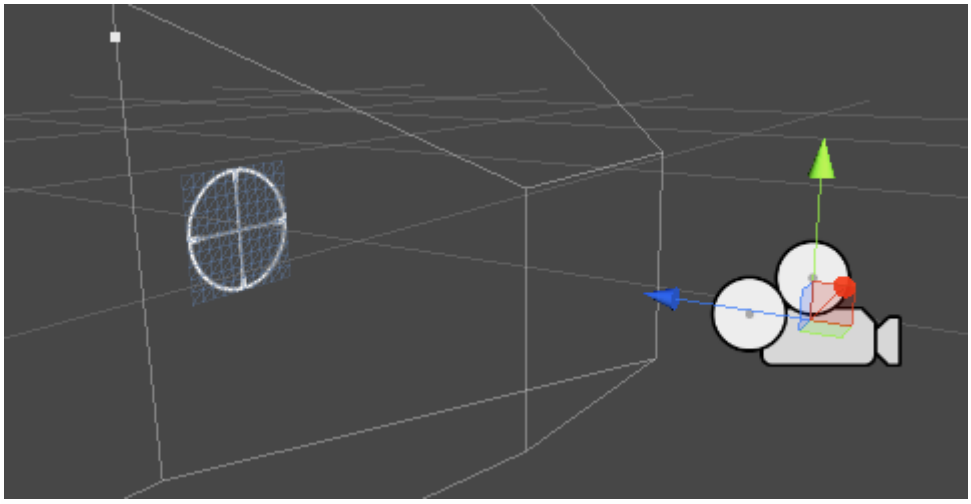
The following script will cause the camera to orbit the center of the world, as well as tilt up and down:

```
01  public class Cannon : MonoBehaviour {
02      void LateUpdate() {
03          float x = Input.GetAxis("Mouse X") * 2;
04          float y = -Input.GetAxis("Mouse Y");
05
06          // vertical tilting
07          float yClamped = transform.eulerAngles.x + y;
```

```
08  transform.eulerAngles.z);
09
10          // horizontal orbiting
11          transform.RotateAround(new Vector3(0, 3, 0), Vector3.up, x);
12      }
13  }
```

As a final touch, let's make it easier to aim by adding a crosshair to our camera:

- Click GameObjects > Create > Plane
- Rename the Plane to **Crosshair**
- Set the Crosshair's position to 0, 0, 0
- Download Crosshair.png
- Drag Crosshair.png into the Project panel
- Drag the Crosshair texture onto the Crosshair plane in the Scene panel
- In the Inspector, right click the MeshCollider component and remove it so that it won't affect other physics objects in the scene



Shooting Cannonballs

Being able to look at our structure is okay, but this is supposed to be about physics! We need a way to knock it down to see the physics in action. What we need is something to shoot!

- Click GameObject > Create Other > Sphere
- Rename the Sphere to **Cannonball**
- Set the Cannonball's position to 0, 0, 0

- With the Cannonball selected, click Components > Physics > Rigidbody
- Drag Cannonball from the Hierarchy panel to the Project panel to turn it into a Prefab

Since we're going to be shooting cannonballs directly from the camera, we can edit our existing Cannon script. First, we add a public attribute at the top of the class for our projectile prefab.

```
1  public class Cannon : MonoBehaviour {
2      public GameObject projectilePrefab;
```

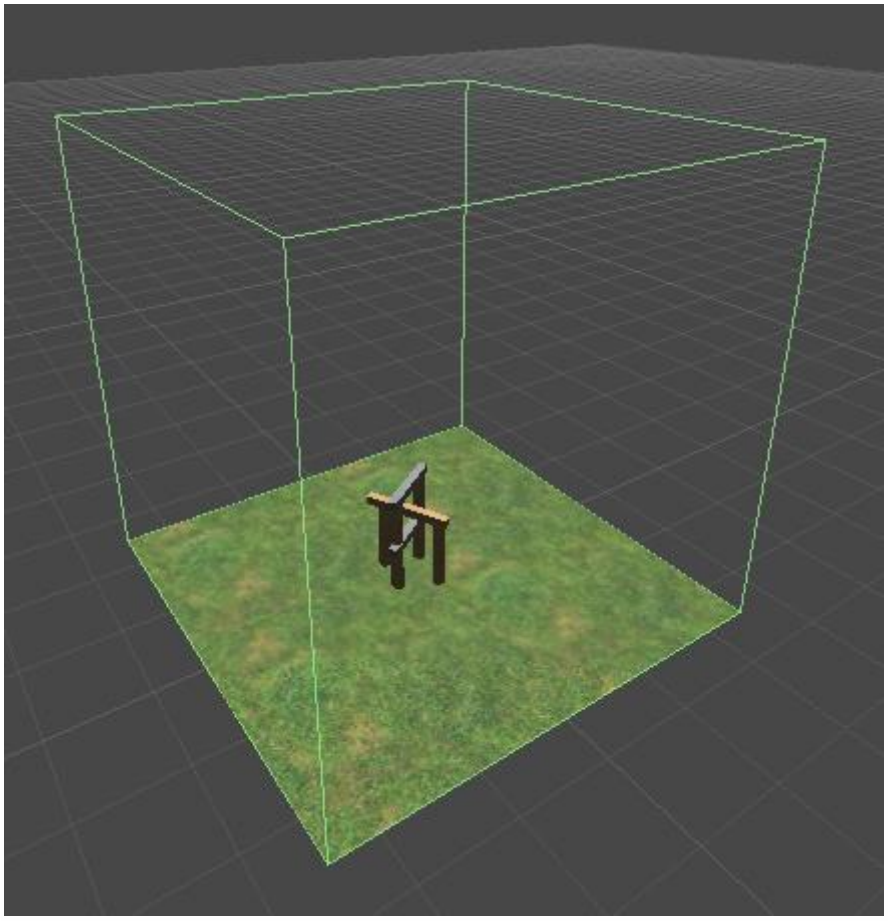
We add a FixedUpdate method to listen for the "Fire1" button to be pressed and then instantiate a Cannonball prefab, position it at the camera, and then add a force to it to move it forward.

```
1  void FixedUpdate () {
2      if (Input.GetButtonDown("Fire1")) {
3          GameObject projectile = Instantiate(projectilePrefab, transform.position,
   GameObject;
4          projectile.rigidbody.AddRelativeForce(new Vector3(0, 0, 2000));
5      }
6  }
```

Boundaries

You may have noticed that if a cannonball is fired far enough, it can fall off the edge of our 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.

- Click GameObject > Create Empty
- Rename it to **Boundary**
- Set the Boundary's x, y, and z position to 0
- With the Boundary selected, click Components > Physics > Box Collider
- In the insepctor, make sure Is Trigger is checked
- Set the Box Collider's Center to 0, 25, 0
- Set the Box Collider's Size to 50, 50, 50



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

- Click Assets > Create > C# Script
- Rename the script to **Boundary**
- Drag the script onto the Boundary object in the Hierarchy panel
- Edit the script and add the following code

```
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 from impacts.

- Click Assets > Create > C# Script
- Rename the script to **Block**
- Drag the script onto the Block prefab in the Project panel
- 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 that was 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.

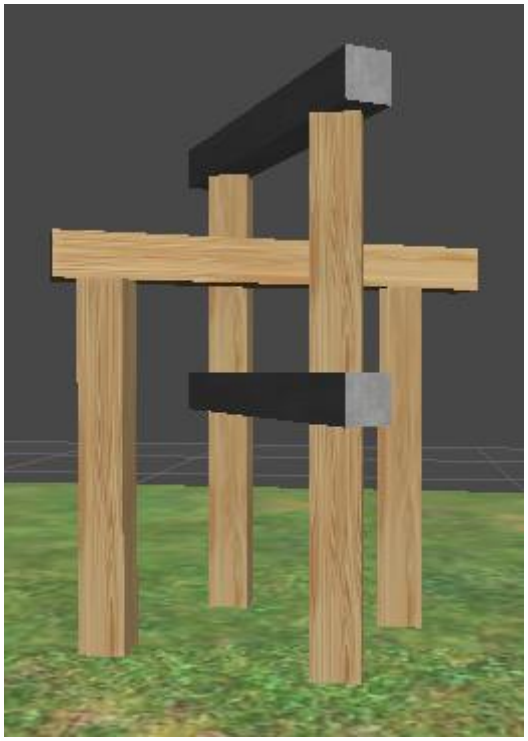
```

    void OnCollisionEnter(Collision collision) {
        // apply collision damage
        if (collision.relativeVelocity.magnitude > 0.5) {
            health -= collision.relativeVelocity.magnitude;
        }

        // destroy if health is too low
        if (health <= 0) {
            Destroy(gameObject);

            // restart the scene if this was the last box
            GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
            if (boxes.Length <= 1) {
                Application.LoadLevel("Main");
            }
        }
    }

```

Concrete Blocks

So far, we've only been using wooden blocks. They're light and relatively weak, making the structure too easy to destroy and fairly predictable in how it will move. 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
- Drag Concrete.jpg into the Project panel
- 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 great impact, and be harder to destroy with cannonballs.



Finished Game

Below is the finished game running in the Unity Web Player. Use the mouse to orbit the camera and press Ctrl or Left Mouse button to shoot cannonballs.