unity

# 5.1 Clicky Mouse

**Steps:**

*Step 1: Create project and switch to 2D view*

*Step 2: Create good and bad targets*

*Step 3: Toss objects randomly in the air*

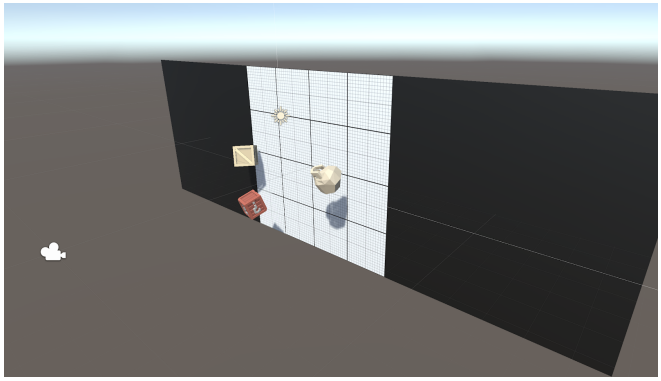
*Step 4: Replace messy code with new methods*

*Step 5: Create object list in Game Manager*

*Step 6: Create a coroutine to spawn objects*

*Step 7: Destroy target with click and sensor*

*Example of project by end of lesson*



**Length:** 60 minutes

**Overview:** It's time for the final unit! We will start off by creating a new project and importing the starter files, then switching the game's view to 2D. Next we will make a list of target objects for the player to click on: Three "good" objects and one "bad". The targets will launch spinning into the air after spawning at a random position at the bottom of the map. Lastly, we will allow the player to destroy them with a click!

**Project Outcome:** A list of three good target objects and one bad target object will spawn in a random position at the bottom of the screen, thrusting themselves into the air with random force and torque. These targets will be destroyed when the player clicks on them or they fall out of bounds.

**Learning Objectives:** By the end of this lesson, you will be able to:

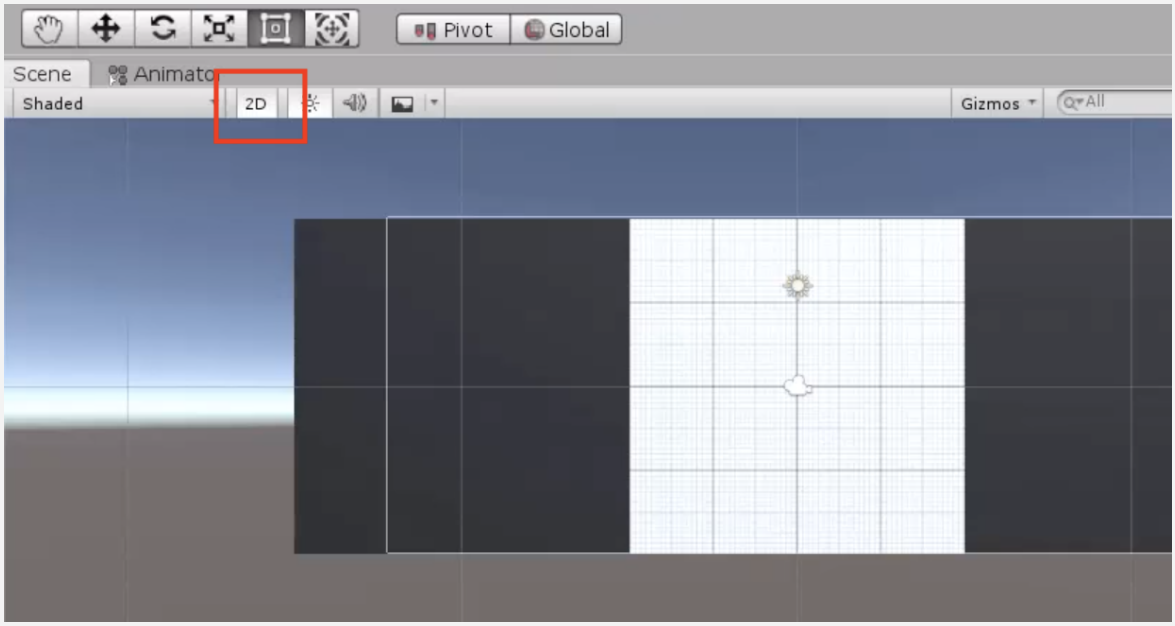
- Switch the game to 2D view for a different perspective
- Add torque to the force of an object
- Create a Game Manager object that controls game states as well as spawning
- Create a List of objects and return their length with Count
- Use While Loops to repeat code while something is true
- Use OnMouseDown to enable the player to click on things

# Step 1: Create project and switch to 2D view

*One last time… we need to create a new project and download the starter files to get things up and running.*

1. Open **Unity Hub** and create an empty "Prototype 5" project in your course directory on the correct Unity version.
   If you forget how to do this, refer to the instructions in Lesson 1.1 - Step 1
2. Click to download the Prototype 5 Starter Files, **extract** the compressed folder, and then **import** the .unitypackage into your project.
   If you forget how to do this, refer to the instructions in Lesson 1.1 - Step 2
3. Open the **Prototype 5** scene, then delete the **sample scene** without saving
4. Click on the **2D icon** in Scene view to put Scene view in **2D**
5. (optional) Change the texture and color of the **background** and the color of the **borders**

- **New Concept:** 2D View
- **Demo:** Notice in 2D view: You can't rotate around objects or move them in the Z direction

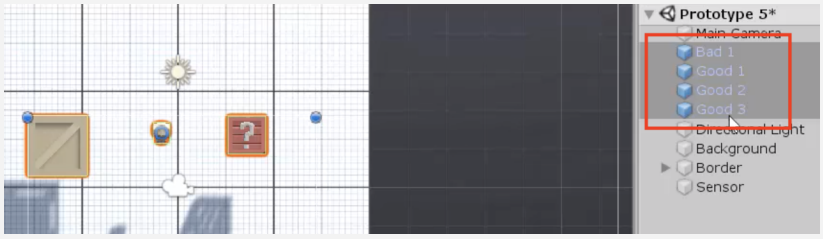


# Step 2: Create good and bad targets

*The first thing we need in our game are three good objects to collect, and one bad object to avoid. It'll be up to you to decide what's good and what's bad.*

1. From the **Library**, drag 3 "good" objects and 1 "bad" object into the Scene, rename them "Good 1", "Good 2", "Good 3", and "Bad 1"
2. Add **Rigid Body** and **Box Collider** components, then make sure that Colliders surround objects properly
3. Create a new Scripts folder, a new "Target.cs" script inside it, attach it to the **Target objects**
4. Drag all 4 targets into the **Prefabs folder** to create "original prefabs", then **delete** them from the scene

- **Tip:** The bigger the collider boxes, the easier it will be to hit them
- **Tip:** Try selecting multiple objects and applying scripts/components - very handy




# Step 3: Toss objects randomly in the air

*Now that we have 4 target prefabs with the same script, we need to toss them into the air with a random force, torque, and position.*

1. In **Target.cs**, declare a new ***private Rigidbody targetRb;*** and initialize it in **Start()**
2. In **Start()**, add an **upward force** multiplied by a **randomized speed**
3. Add a **torque** with randomized **xyz values**
4. Set the **position** with a randomized **X value**

- **New Function:** AddTorque
- **Tip:** Test with different values by dragging them in during runtime
- **Don't worry:** We're going to fix all these hard-coded values next

```
private Rigidbody targetRb;

void Start() {
  targetRb = GetComponent<Rigidbody>();
  targetRb.AddForce(Vector3.up * Random.Range(12, 16), ForceMode.Impulse);
  targetRb.AddTorque(Random.Range(-10, 10), Random.Range(-10, 10),
  Random.Range(-10, 10), ForceMode.Impulse);
  transform.position = new Vector3(Random.Range(-4, 4), -6); }
```

## Step 4: Replace messy code with new methods

*Instead of leaving the random force, torque, and position making our Start() function messy and unreadable, we're going to store each of them in brand new clearly named custom methods.*

1. Declare and initialize new private float variables for ***minSpeed, maxSpeed, maxTorque, xRange,*** and ***ySpawnPos***;
2. Create a new function for ***Vector3 RandomForce()*** and call it in ***Start()***
3. Create a new function for ***float RandomTorque()*** and call it in ***Start()***
4. Create a new function for ***RandomSpawnPos()***, have it return a new ***Vector3*** and call it in ***Start()***

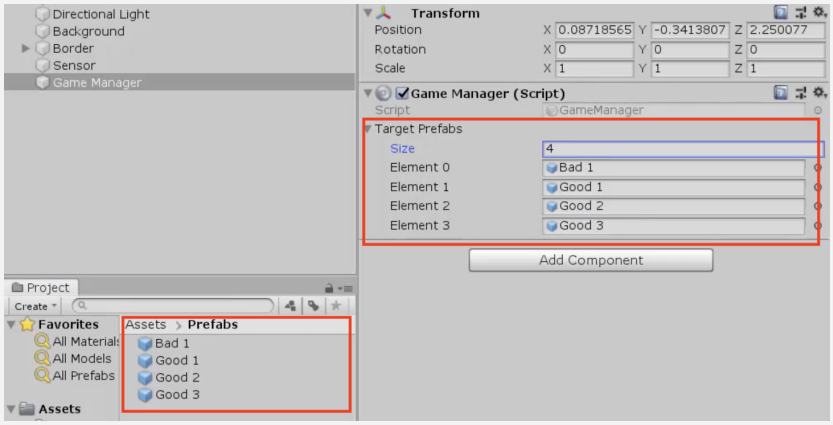
```
private float minSpeed = 12;
private float maxSpeed = 16;
private float maxTorque = 10;
private float xRange = 4;
private float ySpawnPos = -6;

void Start() {
  ...
  targetRb.AddForce(... RandomForce(), ForceMode.Impulse);
  targetRb.AddTorque(... RandomTorque(), RandomTorque(), RandomTorque(),
    ForceMode.Impulse);
  transform.position = ... RandomSpawnPos();
}

Vector3 RandomForce() {
  return Vector3.up * Random.Range(minSpeed, maxSpeed);
}
float RandomTorque() {
  return Random.Range(-maxTorque, maxTorque);
}
Vector3 RandomSpawnPos() {
  return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
}
```

# Step 5: Create object list in Game Manager

*The next thing we should do is create a list for these objects to spawn from. Instead of making a Spawn Manager for these spawn functions, we're going to make a Game Manager that will also control game states later on.*

1. Create a new "Game Manager" **Empty object**, attach a new **GameManager.cs** script, then open it
2. Declare a new ***public List<GameObject> targets;***, then in the Game Manager inspector, change the list **Size** to 4 and assign your **prefabs**

- **New Concept:** Lists
- **New Concept:** Game Manager
- **Demo:** Feel free to reference old code: We used an array instead of a list to spawn the animals in Unit 2




# Step 6: Create a coroutine to spawn objects

*Now that we have a list of object prefabs, we should instantiate them in the game using coroutines and a new type of loop.*

1. Declare and initialize a new ***private float spawnRate*** variable
2. Create a new ***IEnumerator SpawnTarget ()*** method
3. Inside the new method, ***while(true),*** wait **1 second**, generate a **random index**, and spawn a random **target**
4. In **Start()**, use the ***StartCoroutine*** method to begin spawning objects

- **Tip:** Feel free to reference old code: we used coroutines for the powerup cooldown in Unit 4
- **Tip:** Arrays return an integer with .Length, while Lists return an integer with .Count
- **New Concept:** While Loops

```
private float spawnRate = 1.0f;

void Start() {  StartCoroutine(SpawnTarget());  }

IEnumerator SpawnTarget() {
  while (true) {
    yield return new WaitForSeconds(spawnRate);
    int index = Random.Range(0, targets.Count);
    Instantiate(targets[index]); } }
```

## Step 7: Destroy target with click and sensor

*Now that our targets are spawning and getting tossed into the air, we need a way for the player to destroy them with a click. We also need to destroy any targets that fall below the screen.*

1. In **Target.cs**, add a new method for ***private void OnMouseDown() { }*** , and inside that method, destroy the gameObject
2. Add a new method for ***private void OnTriggerEnter(Collider other)*** and inside that function, destroy the gameObject

- **New Function:** OnMouseDown
- **Tip:** There is also OnMouseUp, and OnMouseEnter, but Down is definitely the one we want
- **Tip:** You could use Update and check if target y position is lower than a certain value, but a sensor is better because it doesn't run all the time

```
private void OnMouseDown() {
  Destroy(gameObject); }

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

## Lesson Recap

| | |
|---|---|
| **New Functionality** | • Random objects are tossed into the air on intervals<br>• Objects are given random speed, position, and torque<br>• If you click on an object, it is destroyed |
| **New Concepts and Skills** | • 2D View<br>• AddTorque<br>• Game Manager<br>• Lists<br>• While Loops<br>• Mouse Events |
| **Next Lesson** | • We'll add some effects and keep track of score! |