Tutorial Week 3

Created by Jordan Woo
Presented by Matthew Graber

DOWNLOAD

1. Go to **https://ter.ps/XRWeek3**
2. Unzip downloaded Proj3 folder
3. Go to Assets/Scenes/ and open SceneFinal.unity

OR (if you know git)

1. Go to github.com/umdxrclub/Tutorials
2. Git pull
3. Open project 3

Vector3

Constructor:

```
new Vector3(x, y, z);
```

Instance members:

```
vec.x, vec.y, vec.z;
```

Multiplying a vector with scalar:

```
vec = new Vector3(1, 2, 3);
vec = vec * 4;
Debug.Log(vec);
// prints (4, 8, 12)
```

Input Axes

Locate input axes:
Edit->Project Settings->Input

An input axis represents a **direction using two keys** (e.g. vertical axis represented by up and down keys)

In code:

```
Input.GetAxis(“Vertical”);
// returns number between
1 and -1 (up 1, down -1)
```

Time.deltaTime

- Returns **time elapsed since last frame.**
- If your game is running around 30 FPS, it will return around **1/30 of a second, or 0.0333**
- Used to define values over a time period (e.g. 1 meter per second)
- Example:
  - We want to move 1 meter per second
  - `transform.Translate(1, 0, 0);`
  - Moves 1 meter per FRAME (way too fast)
  - `transform.Translate(1 * Time.deltaTime, 0, 0);`
  - Now moves 1 /30 of meter per frame, or 1 meter per second

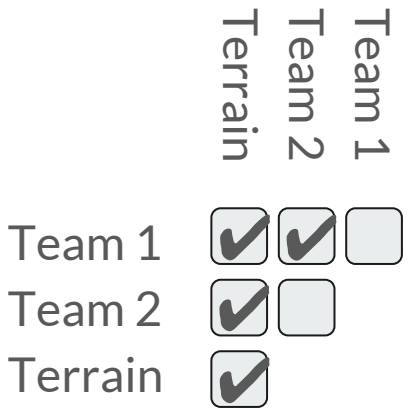
Layers

- Say you have a red team and blue team with two players each.
- We want players on the **same team to be able to walk through each other**, but collide with everything else.
- By default in unity, everything collides with everything else. We get around this by **using layers**.
- The layer of a gameobject is found in the inspector in the top-right corner.
- Layers can be accessed and changed through code using:
  - `gameObject.layer;`
- Layers in code are represented as ints and can be accessed through
  - `LayerMask.NameToLayer("LayerName");`

Collision Matrix

- We can define which layers collide with which other layers by using **the Collision Matrix.**
- Found by doing Edit->Project Settings->Physics
- For previous example - we would split up game into three layers: Team 1, Team 2, and Terrain
- Team 1 would collide with Team 2 and Terrain, but not itself.
- Team 2 would collide with Team 1 and Terrain, but not itself.
- Terrain would collide with everything.

Checkmark indicates collision is enabled between two layers.



OnCollisionEnter()

OnCollisionEnter(), like Start() and Update(), is a **Callback:** it is called automatically by Unity.

Method signature:

```
public void
OnCollisionEnter(Collision
collision)
```

Called the first frame that another object collides with this object.

Collision

Object that represents a collision that is passed by OnCollisionEnter

Instance members:

```
Collision.gameObject
Collision.rigidBody
Collision.collider
Collision.relativeVelocity
```

And others that can be found using scripting reference.

Tags

- Tags are a way to arbitrarily **mark certain objects.**
- Common uses are to tag the player, the main camera, or enemies.
- Tags can be set at the top left of the inspector.
- Tags can be checked in code using:
  - `gameObject.CompareTag("Enemy");`

Prefabs

- Prefabs are essentially **reusable gameobject templates**.
- They're commonly used for spawned objects.
- Prefabs can be created at any time by creating a gameobject, modifying it, then dragging it into the project panel.
- Prefabs can be spawned using
  - `Instantiate(prefab, position, rotation);`

Triggers

- Triggers are colliders that are **marked is trigger** in inspector.
- Triggers differ from colliders in that they **do not collide with anything**.
- Triggers have their own callback methods for overlaps.
- Triggers are often used for item pickups or detecting if the player has moved into a certain area.

OnTriggerEnter()

- OnTriggerEnter() is called the **first frame** when a trigger overlaps with a collider.
- Method signature:
  - `public void OnTriggerEnter(Collider other)`
- Its argument is the collider that this object has overlapped.
- OnTriggerEnter() is called for both objects that overlap.