Procedural Level Generation

Lets start from where we left off (see the demo for Basic Scripting under the Instantiation scene).

```
void Awake () {
       Vec;;l00ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp
ppppppppppppppppppppppppppppppppppppptor3 currentPosition = Vector3.zero;
           Transform trans = gameObject.GetComponent<Transform>();
// draw a bunch of platforms!
       for (int i = 0; i< 10; i++) {
              Instantiate (basePlatform, currentPosition, trans.rotation);
                                currentPosition+= new Vector3(0,0,50f);
           }
    }
```

How can we make this go FOREVER? Use the update function instead of a loop!

```
  public GameObject basePlatform;
    Vector3 currentPosition = Vector3.zero;

       // Update is called once per frame
       void Update () {
        Instantiate(basePlatform, currentPosition, transform.rotation);
        currentPosition += new Vector3(0, 0, 50f);
       }
}
```

Play and see what happens.

Take a look at how many cubes you have now. A LOT… more and more as we go along. Unity is pretty good at not letting this affect rendering speed but you and I know we would eventually run out of memory here.

How to fix?

1. We need to destroy the platforms behind us.
2. We only need to render the platforms in front of us as far as we can see – the really far away ones we don't need to render yet.

Destroy Platforms

To destroy an object, use the Destroy method.

https://docs.unity3d.com/ScriptReference/Object.Destroy.html

public static void **Destroy**(Object **obj**, float **t** = 0.0F);

Parameters

obj	The object to destroy.
t	The optional amount of time to delay before destroying the object.

Oops we didn't save a reference to the object after instantiating it!

We could do that and use the GenerateLevelScript to destroy the objects. Or we could write a script for the objects to destroy themselves. Lets make a script for this and attach it to the platform prefab since that's easier.

```
public class DestroyAtDistance : MonoBehaviour {

    public float distance=100;

       // Update is called once per frame
       void Update () {
      if(  Vector3.Distance(Camera.main.transform.position, transform.position) >
distance)
         Destroy(gameObject);
       }
}
```

You can use Vector3.Distance to calculate distances and Camera.main.transform.position is the position of the main camera (the player). Now, play and see what happens.

Yay now we only have 3 cubes! But wait, we get to the end of the last one and we never get any more. Oh no! Let's fix that in the GenerateLevel script.

```
public float distanceToInstantiate = 100;
       // Update is called once per frame
       void Update () {
        if (Vector3.Distance(Camera.main.transform.position, currentPosition) <
distanceToInstantiate){
            Instantiate(basePlatform, currentPosition, transform.rotation);
            currentPosition += new Vector3(0, 0, 50f);
        }

}
```

It worked! Except now we can see the new platforms popping in at a distance. I think we need to set `distanceToInstantiate` to be more than 100, but then it will get destroyed too early. There are many ways to deal with this. One way to do this is to see whether we stepped on the platform yet or not.

First lets add a variable to Destroy at Distance to keep track of whether we stepped on it already or not.

```
public float distance=100;
   public  bool playerStepped = false;

       // Update is called once per frame
       void Update () {
        if (Vector3.Distance(Camera.main.transform.position, transform.position) >
distance && playerStepped)
        {
            Destroy(gameObject);
        }
       }

    public void SetPlayerStepped(bool step){
        playerStepped = step;
    }
```

To call `SetPlayerStepped`, we need to use the OnCharacterColiderHit function which ONLY gets called on a character collider. We need to add a script to the CharacterCollider to do this

```
public class StepOnPlatform : MonoBehaviour {


    void OnControllerColliderHit(ControllerColliderHit other)
    {
        if(other.gameObject.tag.Equals("platform"))
        other.gameObject.GetComponent<DestroyAtDistance>().SetPlayerStepped(true);

    }
}
```

And we need to tag our platform prefab as a platform. We will need to create a platform tag for that (top of inspector).

Play! Yay it works!

Adding Some Randomness to Platform Placement

Let's make the platform positions offset randomly to make it a little less monotonous.

In GenerateLevel.cs

```
void Update () {
        if (Vector3.Distance(Camera.main.transform.position, currentPosition) <
distanceToInstantiate){
           Instantiate(basePlatform, currentPosition, transform.rotation);
           currentPosition += new Vector3(Random.Range(-2, 3), Random.Range(-1, 2), 49f);
        }
}
```

Activity

Create an obstacle prefab that your character has to jump over. Randomly populate it on the platforms.

Upload a screenshot of your game to today's discussion thread.