Week 11: Programming (Part 3 - Obstacles)

Welcome to Week 11!

Continuation of programming obstacles

11.1—Introduction

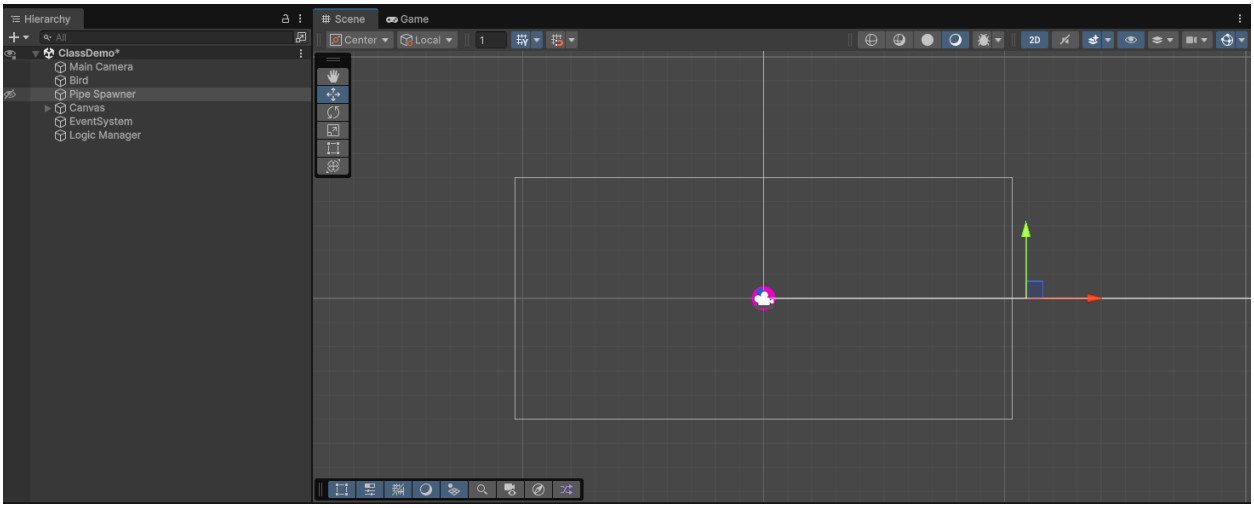
This week continues development of the final Unity project started last week. The focus shifts from player control to level challenge by introducing obstacles for the bird to navigate. Specifically, vertical pipe obstacles will be created, animated, spawned at timed intervals, and removed once they leave the visible game area. These systems work together for form the core challenge mechanics of the Flying Bird game.

This week continues the development of the final Unity project by continuing to expand the pipe spawning system. The focus shifts from creating a single obstacle to dynamically generating them during gameplay.

11.2—Pipe Spawner

To control how often pipe obstacles appear, an empty Game Object named “Pipe Spawner” is added to the scene. This GameObject is positioned to the right side of the camera’s view so the pipes enter the screen from offscreen.

Only the X value of the Transform component is adjusted during the positioning, while the Y and Z values remain at zero. This ensures consistent vertical placement logic later.



11.3—Pipe Spawner Scripting

A new script name “PipeSpawner” is created and attached to the Pipe Spawner GameObject. This script controls when and how often pipe prefabs are instantiated.

A public GameObject variable named “pipe” is declared to store a reference to the pipe prefab. In the Update method, Unity’s built-in Instantiate() function is used to spawn the pipe at the spawner’s position and rotation.

```
public class PipeSpawner : MonoBehaviour
{
    public gameObject pipe;
    void Start()
    {

    }

    void Update()
    {
        Instantiate(pipe, transform.position, transform.rotation);
    }
```

Testing this script reveals that pipes spawn continuously every frame. To fix this, a timing system is introduced using two float variables: “spawnRate”, which defines the delay between spawns, and “timer”, which tracks elapsed time.

```
public class PipeSpawner : MonoBehaviour
{

    public gameObject pipe;

    public float spawnRate =2f
```

```
    public float timer =0;

    void Start()
    {

    }

    void Update()
    {
        if(timer<spawnRate)
        {
            timer+=Time.deltaTime;
        }
        else
        {
            Instantiate(pipe, transform.position, transform.rotation);
            timer=0;
        }
    }
}
```

To avoid duplicating code and to allow an initial pipe to spawn immediately, the instantiation logic is moved into a separate method named SpawnPipe(). This method is called within the Start() and Update() function.

```
public class PipeSpawner : MonoBehaviour

{

    public gameObject pipe;
    public float spawnRate =2f
    public float timer =0;

    void Start()
    {
        SpawnPipe();
    }

    void Update()
    {
        if(timer<spawnRate)
        {
            timer+=Time.deltaTime;
        }
        else
        {
            SpawnPipe();
            timer=0;
        }
    }

    void SpawnPipe()
    {
        Instantiate(pipe, transform.position, transform.rotation);
    }
```

```
}
```

11.4—Pipe Spawner Variation scripting

At this stage, pipe spawn at regular intervals but always at the same vertical position. This makes game play predictable and removes difficulty. To introduce variation, a vertical offset is added.

A public float variable named “heightOffset” is declared and set to a value of 3. Inside the SpawnPipe() method, two local variables, “lowestPoint” and “highestPoint”, are calculated based on this offset. These values define the vertical range in which pipes may appear.

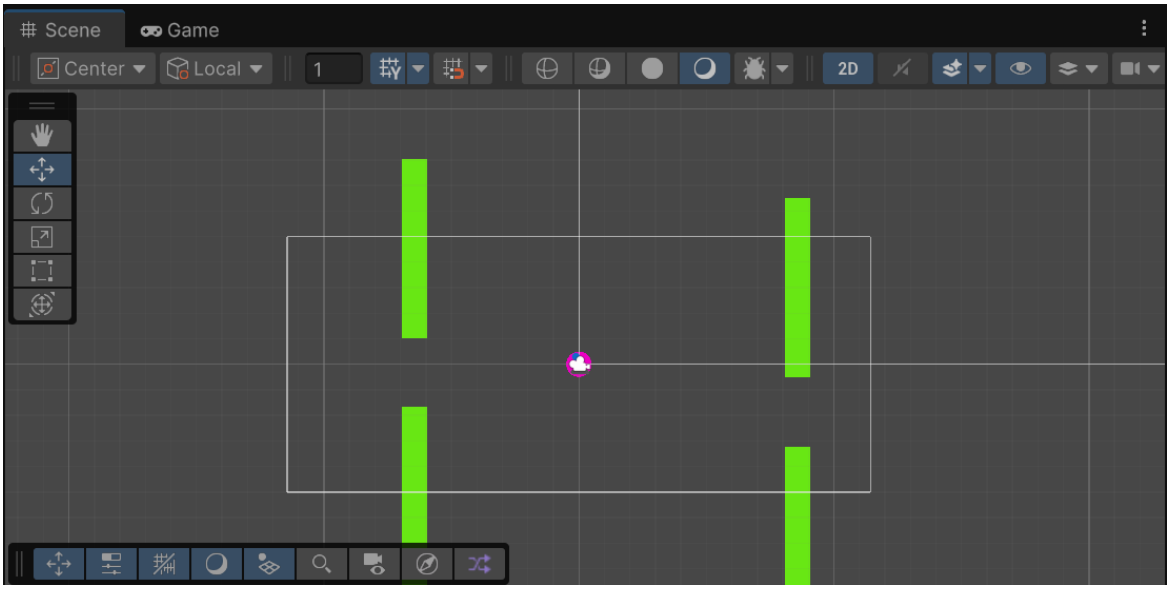
The Y position of the spawned pipe is then randomized using Random.Range().

```
void SpawnPipe()

    {
        float lowestPoint = transform.position.y -heightOffset;
        float highestPoint = transform.position.y+heightOffset;

        Instantiate(pipe, new Vector3(transform.position.x,
Random.Range(lowestPoint, highestPoint),0), transform.rotation);
    }
```

After testing, pipes should now appear at varying heights, creating a more challenging and engaging gameplay experience.



11.5—Completed Scripts

Pipe Spawner Script:

```
public class PipeSpawner : MonoBehaviour
```

```
{

    public gameObject pipe;
    public float spawnRate =2f
    public float timer =0;

    void Start()
    {
        SpawnPipe();
    }

    void Update()
    {
        if(timer<spawnRate)
        {
            timer+=Time.deltaTime;
        }
        else
        {
            SpawnPipe();
            timer=0;
        }
    }

    void SpawnPipe()
    {
        float lowestPoint = transform.position.y -heightOffset;
        float highestPoint = transform.position.y+heightOffset;

        Instantiate(pipe, new Vector3(transform.position.x,
Random.Range(lowestPoint, highestPoint),0), transform.rotation);
    }
```

11.6—Exercises

Reinforcement

The reinforcement section is used to check the understanding of the information that was presented in the PowerPoint/document. Please answer the questions in your own words.

R—11.1 What does the Unity's Instantiate() function do?

R—11.2 Why does placing the Instantiate() function directly inside Update() cause too many pipes to spawn?

R—11.3 Why is Time.deltaTime used when updating the timer?

R—11.4 Why was the pipe spawning logic moved into a separate SpawnPipe() function?

R—11.5 What does the Random.Range() return?

R—11.6 How does vertical randomization increase replayability?

R—11.7 What potential bugs could occur if the timer logic is written incorrectly?

Project

The project section's tasks are used in work toward creating the final Flying Bird game project for the end of the course.

P—11.1 Create an empty GameObject named "Pipe Spawner" and position it to the right of the camera view.

P—11.2 Create a C# script that uses Unity's Instantiate() function to spawn Pipe Spawner pefab and the spawner's position and rotation.

P—11.3 Refactor the spawning logic into a SpawnPipe() function to improve code organization.

P—11.4 Update the SpawnPipe() function so it includes a vertical variation by using a heightOffset variable and Random.Range() to spawn pipes at different Y positions.