2.3 Random Animal Stampede

Steps:

Example of project by end of lesson

Length: 50 minutes

Overview: Our animal prefabs walk across the screen and get destroyed out of bounds, but they don't actually appear in the game unless we drag them in! In this lesson we will allow the animals to spawn on their own, in a random location at the top of the screen. In order to do so, we will create a new object and a new script to manage the entire spawning process.

Project Outcome: When the user presses the S key, a randomly selected animal will spawn at a random position at the top of the screen, walking towards the player.

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

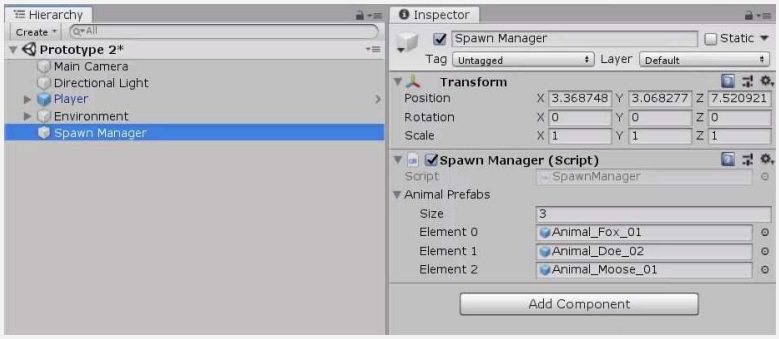
- Create an empty object with a script attached
- Use arrays to create an accessible list of objects or values
- Use integer variables to determine an array index
- Randomly generate values with Random.Range in order to randomize objects in arrays and spawn positions
- Change the camera's perspective to better suit your game

Step 1: Create a spawn manager

If we are going to be doing all of this complex spawning of objects, we should have a dedicated script to manage the process, as well as an object to attach it to.

1. In the hierarchy, create an **empty object** called "Spawn Manager"
2. Create a new script called "SpawnManager", attach it to the **Spawn Manager**, and open it
3. Declare new ***public GameObject[ ] animalPrefabs;***
4. In the inspector, change the **Array size** to match your animal count, then **assign** your animals by **dragging** them in

- **Tip:** Empty objects can be used to store objects or used to store scripts
- **Warning:** You can use spaces when naming your empty object, but make sure your script name uses PascalCase!
- **New:** Arrays



Step 2: Spawn an animal if S is pressed

We've created an array and assigned our animals to it, but that doesn't do much good until we have a way to spawn them during the game. Let's create a temporary solution for choosing and spawning the animals.

1. In ***Update()***, write an if-then statement to **instantiate** a new animal prefab at the top of the screen if **S** is pressed
2. Declare a new ***public int animalIndex*** and incorporate it in the **Instantiate** call, then test editing the value in the Inspector

- **New:** Array Indexes
- **Tip:** Array indexes start at 0 instead of 1. An array of 3 animals would look like [0, 1, 2]
- **New:** Integer Variables
- **Don't worry:** We'll declare a new variable for that Vector3 and that index later

```
public GameObject[] animalPrefabs;
public int animalIndex;

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20),
    animalPrefabs[animalIndex].transform.rotation);
  }
}
```

Step 3: Spawn random animals from the index

We can spawn animals by pressing S, but doing so only spawns an animal at the array index we specify. We need to randomize the selection so that S can spawn a random animal based on the index, without our specification.

1. In the if-statement checking if S is pressed, generate a random ***int animalIndex*** between 0 and the length of the array
2. Remove the global ***animalIndex*** variable, since it is only needed locally in the **if-statement**

- Tip: Google "how to generate a random integer in Unity"
- New: Random.Range
- New: .Length
- New: Global vs Local variables

```
public GameObject[] animalPrefabs;
public int animalIndex;

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    int animalIndex = Random.Range(0, animalPrefabs.Length);
    Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20),
            animalPrefabs[animalIndex].transform.rotation); }}
```

Step 4: Randomize the spawn location

We can press S to spawn random animals from animalIndex, but they all pop up in the same place! We need to randomize their spawn position, so they don't march down the screen in a straight line.

1. **Replace** the X value for the Vector3 with ***Random.Range(-20, 20)***, then test
2. Within the **if-statement**, make a new local ***Vector3 spawnPos*** variable
3. At the top of the class, create ***private float*** variables for ***spawnRangeX*** and ***spawnPosZ***
4. Add **comments** to your code to clean it up

- Tip: Random.Range for floats is inclusive of all numbers in the range, while Random.Range for integers is exclusive!
- Tip: Keep using variables to clean your code and make it more readable

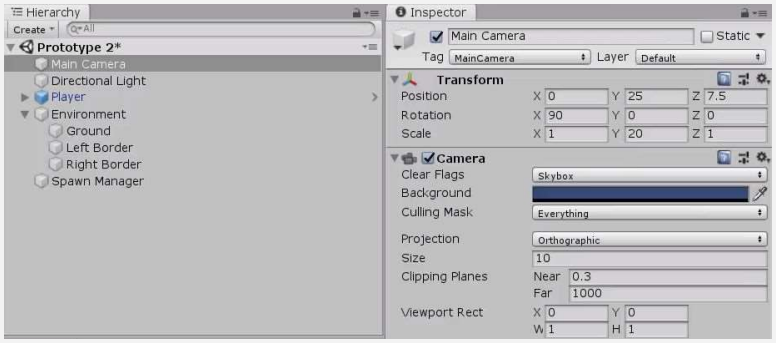
```
private float spawnRangeX = 20;
private float spawnPosZ = 20;

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    // Randomly generate animal index and spawn position
    Vector3 spawnpos = new Vector3(Random.Range(-spawnRangeX, spawnRangeX), 0,
spawnPosZ);
    int animalIndex = Random.Range(0, animalPrefabs.Length);
    Instantiate(animalPrefabs[animalIndex], spawnpos,
    animalPrefabs[animalIndex].transform.rotation); }}
```

Step 5: Change the perspective of the camera

Our Spawn Manager is coming along nicely, so let's take a break and mess with the camera.Changing the camera's perspective might offer a more appropriate view for this top-down game.

1. Toggle between **Perspective** and **Isometric** view in Scene view to appreciate the difference
2. Select the **camera** and change the **Projection** from "Perspective" to "Orthographic"
3. **Save** and quit

- **New:** Orthographic vs Perspective Camera Projection
- **Tip:** Test the game in both views to appreciate the difference



End of Lesson

Recap

New Functionality

- The player can press the S to spawn an animal
- Animal selection and spawn location are randomized
- Camera projection (perspective/orthographic) selected

New Concepts & Skills

- Spawn Manager
- Arrays
- Keycodes
- Random generation
- Local vs Global variables
- Perspective vs Isometric projections

Extension

Related tutorials:

-

Challenge ideas:

-