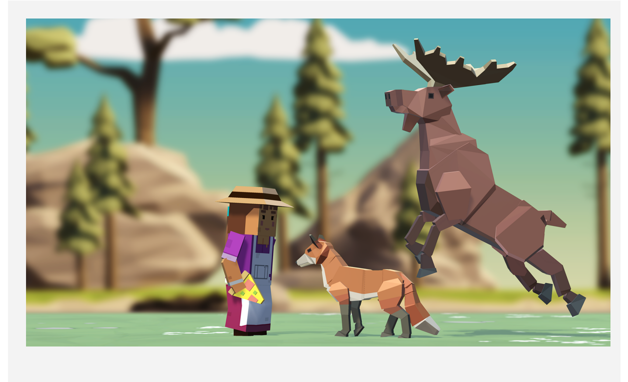
# Create with Code

## Unit 2 Lesson Plans



© Unity 2021

# 2.1 Player Positioning

**Steps:**

*Step 1: Create a new Project for Prototype 2*

*Step 2: Add the Player, Animals, and Food*

*Step 3: Get the user's horizontal input*

*Step 4: Move the player left-to-right*

*Step 5: Keep the player inbounds*

*Step 6: Clean up your code and variables*

*Example of project by end of lesson*

**Length:** 60 minutes

**Overview:** You will begin this unit by creating a new project for your second Prototype and getting basic player movement working. You will first choose which character you would like, which types of animals you would like to interact with, and which food you would like to feed those animals. You will give the player basic side-to-side movement just like you did in Prototype 1, but then you will use if-then statements to keep the Player in bounds.

**Project Outcome:** The player will be able to move left and right on the screen based on the user's left and right key presses, but will not be able to leave the play area on either side.

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

- Adjust the scale of an object proportionally in order to get it to the size you want
- More comfortably use the GetInput function in order to use user input to control an object
- Create an if-then statement in order to implement basic logic in your project, including the use of greater than (>) and less than (<) operators
- Use comments and automatic formatting in order to make their code more clean and readable to other programmers

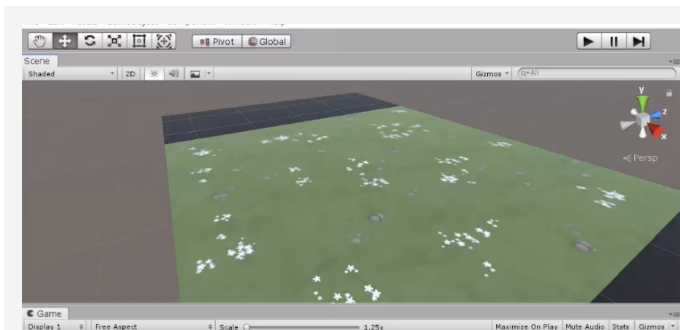
© Unity 2021

## Step 1: Create a new Project for Prototype 2

*The first thing we need to do is create a new project and import the Prototype 2 starter files.*

1. Open **Unity Hub** and create an empty "Prototype 2" 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 2 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. From the Project window, open the **Prototype 2** scene and **delete** the SampleScene
4. In the top-right of the Unity Editor, change your **Layout** from Default to your custom layout

- **Don't worry:** Unit 2 has far more assets than Unit 1, so the package might take a while to import.



## Step 2: Add the Player, Animals, and Food

*Let's get all of our objects positioned in the scene, including the player, animals, and food.*

1. If you want, drag a different **material** from *Course Library > Materials* onto the Ground object
2. Drag 1 **Human**, 3 **Animals**, and 1 **Food** object into the Hierarchy
3. Rename the human "Player", then **reposition** the animals and food so you can see them
4. Adjust the XYZ **scale** of the food so you can easily see it from above

- **New Technique:** Adjusting Scale
- **Warning:** Don't choose people for anything but the player, they don't have walking animations
- **Tip:** Remember, dragging objects into the hierarchy puts them at the origin

## Step 3: Get the user's horizontal input

*If we want to move the Player left-to-right, we need a variable tracking the user's input.*

1. In your **Assets** folder, create a "Scripts" folder, and a "PlayerController" script inside
2. **Attach** the script to the Player and open it
3. At the top of PlayerController.cs, declare a new ***public float horizontalInput***
4. In ***Update()***, set ***horizontalInput = Input.GetAxis("Horizontal")***, then test to make sure it works in the inspector

- **Warning:** Make sure to create your Scripts folder inside of the assets folder
- **Don't worry:** We're going to get VERY familiar with this process
- **Warning:** If you misspell the script name, just delete it and try again.

```
public float horizontalInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
}
```

## Step 4: Move the player left-to-right

*We have to actually use the horizontal input to translate the Player left and right.*

1. Declare a new ***public float speed = 10.0f;***
2. In ***Update()***, Translate the player side-to-side based on ***horizontalInput*** and ***speed***

- **Tip:** You can look at your old scripts for code reference

```
public float horizontalInput;
public float speed = 10.0f;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  transform.Translate(Vector3.right * horizontalInput * Time.deltaTime * speed);
}
```

# Step 5: Keep the player inbounds

*We have to prevent the player from going off the side of the screen with an if-then statement.*

1. In ***Update()***, write an **if-statement** checking if the player's left X position is **less than** a certain value
2. In the if-statement, set the player's position to its current position, but with a **fixed X location**

- **Tip:** Move the player in scene view to determine the x positions of the left and right bounds
- **New Concept:** If-then statements
- **New Concept:** Greater than > and Less Than < operators

```
void Update() {
  if (transform.position.x < -10) {
    transform.position = new Vector3(-10, transform.position.y, transform.position.z);
  }
}
```

# Step 6: Clean up your code and variables

*We need to make this work on the right side, too, then clean up our code.*

1. Repeat this process for the **right side** of the screen
2. Declare new ***xRange*** variable, then replace the hardcoded values with them
3. Add **comments** to your code

- **Warning:** Whenever you see hardcoded values in the body of your code, try to replace it with a variable
- **Warning:** Watch your greater than / less than signs!

```
public float xRange = 10;

void Update()
{
  // Keep the player in bounds
  if (transform.position.x < -10 -xRange)
  {
    transform.position = new Vector3(-10 -xRange, transform.position.y, transform.position.z);
  }
  if (transform.position.x > xRange)
  {
    transform.position = new Vector3(xRange, transform.position.y, transform.position.z);
  }
}
```

© Unity 2021

# Lesson Recap

| | |
|---|---|
| **New Functionality** | • The player can move left and right based on the user's left and right key presses<br>• The player will not be able to leave the play area on either side |
| **New Concepts and Skills** | • Adjust object scale<br>• If-statements<br>• Greater/Less than operators |
| **Next Lesson** | • We'll learn how to create and throw endless amounts of food to feed our animals! |

© Unity 2021

# 2.2 Food Flight

**Steps:**

*Step 1: Make the projectile fly forwards*

*Step 2: Make the projectile into a prefab*

*Step 3: Test for spacebar press*

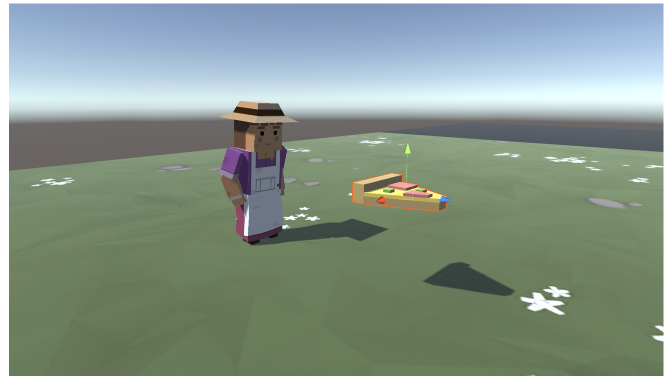
*Step 4: Launch projectile on spacebar press*

*Step 5: Make animals into prefabs*

*Step 6: Destroy projectiles offscreen*

*Step 7: Destroy animals offscreen*

*Example of project by end of lesson*



**Length:** 70 minutes

**Overview:** In this lesson, you will allow the player to launch the projectile through the scene. First you will write a new script to send the projectile forwards. Next you will store the projectile along with all of its scripts and properties using an important new concept in Unity called Prefabs. The player will be able to launch the projectile prefab with a tap of the spacebar. Finally, you will add boundaries to the scene, removing any objects that leave the screen.

**Project Outcome:** The player will be able to press the Spacebar and launch a projectile prefab into the scene, which destroys itself when it leaves the game's boundaries. The animals will also be removed from the scene when they leave the game boundaries.

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

- Transform a game object into a prefab that can be used as a template
- Instantiate Prefabs to spawn them into the scene
- Override Prefabs to update and save their characteristics
- Get user input with GetKey and KeyCode to test for specific keyboard presses
- Apply components to multiple objects at once to work as efficiently as possible

© Unity 2021

# Step 1: Make the projectile fly forwards

*The first thing we must do is give the projectile some forward movement so it can zip across the scene when it's launched by the player.*

1. Create a new "MoveForward" script, **attach** it to the food object, then open it
2. Declare a new ***public float speed*** variable***;***
3. In ***Update()***, add ***transform.Translate(Vector3.forward * Time.deltaTime * speed);***, then **save**
4. In the **Inspector**, set the projectile's **speed** variable, then test

- **Don't worry:** You should all be super familiar with this method now… getting easier, right?

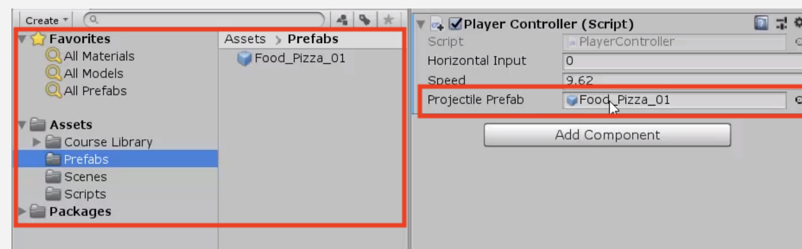
```
public float speed = 40;

void Update() {
  transform.Translate(Vector3.forward * Time.deltaTime * speed);
}
```

# Step 2: Make the projectile into a prefab

*Now that our projectile has the behavior we want, we need to make it into a prefab it so it can be reused anywhere and anytime, with all its behaviors included.*

1. Create a new "Prefabs" folder, drag your food into it, and choose **Original Prefab**
2. In PlayerController.cs, declare a new ***public GameObject projectilePrefab;*** variable
3. **Select** the Player in the hierarchy, then **drag** the object from your Prefabs folder onto the new **Projectile Prefab box** in the inspector
4. Try **dragging** the projectile into the scene at runtime to make sure they fly

- **New Concept:** Prefabs
- **New Concept:** Original vs Variant Prefabs
- **Tip:** Notice that this your projectile already has a move script if you drag it in



## Step 3: Test for spacebar press

*Now that we have a projectile prefab assigned to PlayerController.cs, the player needs a way to launch it with the space bar.*

1. In PlayerController.cs, in ***Update()***, add an **if-statement** checking for a spacebar press: ***if (Input.GetKeyDown(KeyCode.Space)) {***
2. Inside the if-statement, add a comment saying that you should ***// Launch a projectile from the player***

- **Tip:** Google a solution. Something like "How to detect key press in Unity"
- **New Functions:** Input.GetKeyDown, GetKeyUp, GetKey
- **New Function:** KeyCode

```
void Update()
{
  if (Input.GetKeyDown(KeyCode.Space))
  {
    // Launch a projectile from the player
  }
}
```

## Step 4: Launch projectile on spacebar press

*We've created the code that tests if the player presses spacebar, but now we actually need spawn a projectile when that happens*

1. Inside the if-statement, use the **Instantiate** method to spawn a projectile at the player's location with the prefab's rotation

- **New Concept:** Instantiation

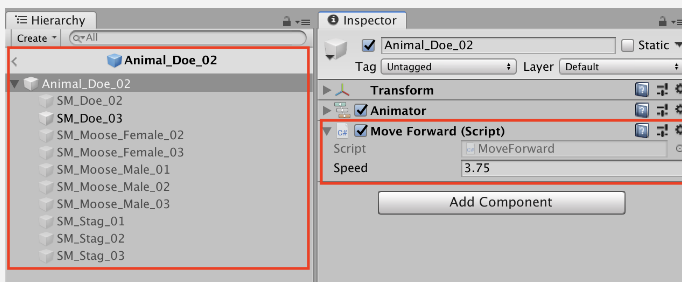
```
if (Input.GetKeyDown(KeyCode.Space))
{
  // Launch a projectile from the player
  Instantiate(projectilePrefab, transform.position, projectilePrefab.transform.rotation);
}
```

# Step 5: Make animals into prefabs

*The projectile is now a prefab, but what about the animals? They need to be prefabs too, so they can be instantiated during the game.*

1. **Rotate** all animals on the Y axis by **180 degrees** to face down
2. **Select** all three animals in the hierarchy and *Add Component > **Move Forward***
3. Edit their **speed values** and **test** to see how it looks
4. Drag all three animals into the **Prefabs folder**, choosing "Original Prefab"
5. **Test** by dragging prefabs into scene view during gameplay

- **Tip:** You can change all animals at once by selecting all them in the hierarchy while holding Cmd/Ctrl
- **Tip:** Adding a Component from inspector is same as dragging it on
- **Warning:** Remember, anything you change while the game is playing will be reverted when you stop it




# Step 6: Destroy projectiles offscreen

*Whenever we spawn a projectile, it drifts past the play area into eternity. In order to improve game performance, we need to destroy them when they go out of bounds.*

1. Create "DestroyOutOfBounds" script and apply it to the **projectile**
2. Add a new ***private float topBound*** variable and initialize it ***= 30;***
3. Write code to destroy if out of top bounds ***if (transform.position.z > topBound) { Destroy(gameObject); }***
4. In the Inspector **Overrides** drop-down, click **Apply all** to apply it to prefab

- **Warning:** Too many objects in the hierarchy will slow the game
- **Tip:** Google "How to destroy gameobject in Unity"
- **New Function:** Destroy
- **New Technique:** Override prefab

```
private float topBound = 30;

void Update() {
  if (transform.position.z > topBound) {
    Destroy(gameObject);   }}
```

# Step 7: Destroy animals offscreen

*If we destroy projectiles that go out of bounds, we should probably do the same for animals. We don't want critters getting lost in the endless abyss of Unity Editor...*

1. Create a new **private float lowerBound** variable and initialize it = -10;
2. Create **else-if statement** to check if objects are beneath ***lowerBound***:
   ***else if (transform.position.z > topBound)***
3. **Apply** the script to all of the animals, then **Override** the prefabs

- **New Function:** Else-if statement
- **Warning:** Don't make topBound too tight or you'll destroy the animals before they before they can spawn

```
private float topBound = 30;
private float lowerBound = -10;

void Update() {
  if (transform.position.z > topBound)
  {
    Destroy(gameObject);
  } else if (transform.position.z < lowerBound) {
    Destroy(gameObject);
  }
}
```

## Lesson Recap

| | |
|---|---|
| **New Functionality** | • The player can press the Spacebar to launch a projectile prefab,<br>• Projectile and Animals are removed from the scene if they leave the screen |
| **New Concepts and Skills** | • Create Prefabs<br>• Override Prefabs<br>• Test for Key presses<br>• Instantiate objects<br>• Destroy objects<br>• Else-if statements |
| **Next Lesson** | • Instead of dropping all these animal prefabs onto the scene, we'll create a herd of animals roaming the plain! |

# 2.3 Random Animal Stampede

**Steps:**

*Step 1: Create a spawn manager*

*Step 2: Spawn an animal if S is pressed*

*Step 3: Spawn random animals from array*

*Step 4: Randomize the spawn location*

*Step 5: Change the perspective of the camera*

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

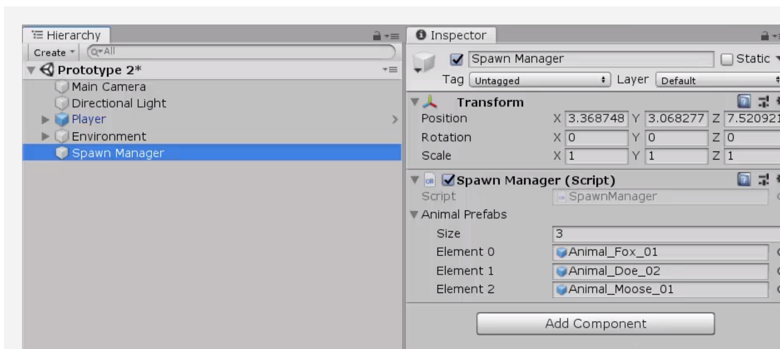
# Step 1: Create a spawn manager

*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 "SpawnManager"
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 from the Project window into the empty slots

   **Note:** Make sure you drag them from the **Project** window; not the Hierarchy! If you're going to spawn objects, you need to make sure you're using Prefabs, which are stored in the Project window.

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

*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 Function:** Random.Range
- **New Function:** .Length
- **New Concept:** 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); }}
```

© Unity 2021

# 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***

- **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

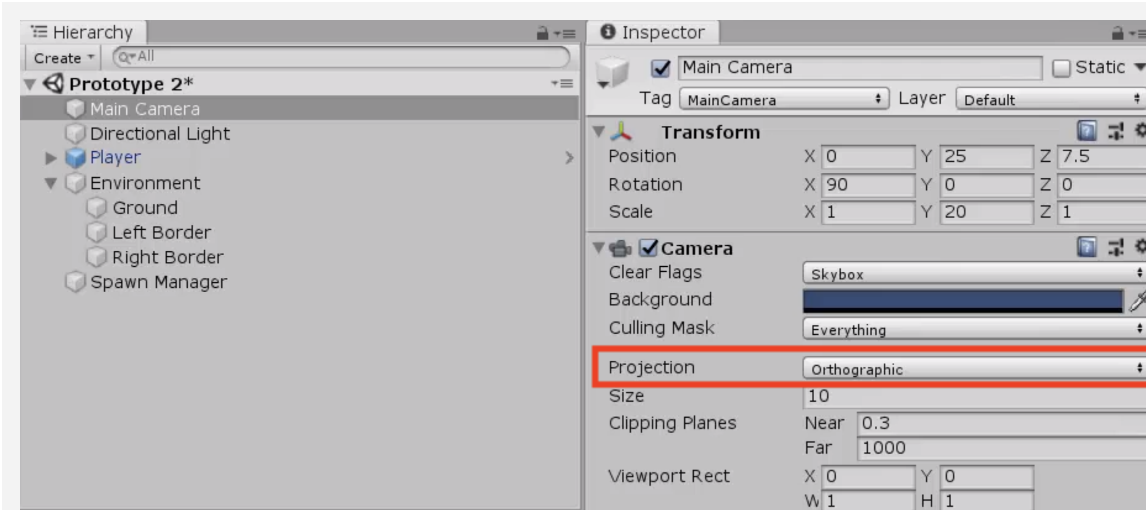
```
private float spawnRangeX = 20;
private float spawnPosZ = 20;

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"

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




## 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 and Skills**

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

**Next Lesson**

- Using collisions to feed our animals!

# 2.4 Collision Decisions

**Steps:**

*Step 1: Make a new method to spawn animals*

*Step 2: Spawn the animals at timed intervals*

*Step 3: Add collider and trigger components*

*Step 4: Destroy objects on collision*

*Step 5: Trigger a "Game Over" message*

*Example of project by end of lesson*

**Length:** 50 minutes

**Overview:** Our game is coming along nicely, but there are are some critical things we must add before it's finished. First off, instead of pressing S to spawn the animals, we will spawn them on a timer so that they appear every few seconds. Next we will add colliders to all of our prefabs and make it so launching a projectile into an animal will destroy it. Finally, we will display a "Game Over" message if any animals make it past the player.

**Project Outcome:** The animals will spawn on a timed interval and walk down the screen, triggering a "Game Over" message if they make it past the player. If the player hits them with a projectile to feed them, they will be destroyed.

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

- Repeat functions on a timer with InvokeRepeating
- Write custom functions to make your code more readable
- Edit Box Colliders to fit your objects properly
- Detect collisions and destroy objects that collide with each other
- Display messages in the console with Debug Log

© Unity 2021

## Step 1: Make a new method to spawn animals

*Our Spawn Manager is looking good, but we're still pressing S to make it work! If we want the game to spawn animals automatically, we need to write our very first custom function.*

1. In **SpawnManager.cs**, create a new ***void SpawnRandomAnimal() {}*** function beneath ***Update()***
2. Cut and paste the code from the **if-then statement** to the **new function**
3. Call ***SpawnRandomAnimal();*** if **S** is pressed

- **New Concept:** Custom Void Functions
- **New Concept:** Compartmentalization / Abstraction

```
void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    SpawnRandomAnimal();
    int animalIndex ... (Cut and Pasted Below) }}

void SpawnRandomAnimal() {
  int animalIndex = Random.Range(0, animalPrefabs.Length);
  Vector3 spawnpos = new Vector3(Random.Range(-xSpawnRange,
    xSpawnRange), 0, zSpawnPos);
  Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20) spawnpos,
    animalPrefabs[animalIndex].transform.rotation);
 }
```

## Step 2: Spawn the animals at timed intervals

*We've stored the spawn code in a custom function, but we're still pressing S! We need to spawn the animals on a timer, so they randomly appear every few seconds.*

1. In ***Start()***, use ***InvokeRepeating*** to spawn the animals based on an interval, then **test**.
2. Remove the **if-then statement** that tests for **S** being pressed
3. Declare new ***private startDelay*** and ***spawnInterval*** variables then playtest and tweak variable values

- **Tip:** Google "Repeating function in Unity"
- **New Function:** InvokeRepeating

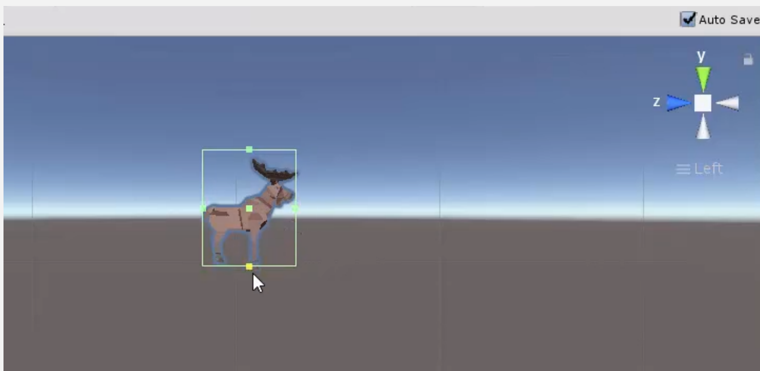
```
private float startDelay = 2;
private float spawnInterval = 1.5f;

void Start() {
  InvokeRepeating("SpawnRandomAnimal", startDelay, spawnInterval); }

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    SpawnRandomAnimal(); } }
```

# Step 3: Add collider and trigger components

*Animals spawn perfectly and the player can fire projectiles at them, but nothing happens when the two collide! If we want the projectiles and animals to be destroyed on collision, we need to give them some familiar components - "colliders."*

1. Double-click on one of the **animal prefabs**, then *Add Component > Box Collider*
2. Click **Edit Collider**, then **drag** the collider handles to encompass the object
3. Check the "**Is Trigger**" checkbox
4. Repeat this process for each of the **animals** and the **projectile**
5. Add a **RigidBody component** to the projectile and uncheck "use gravity"

- **New Component:** Box Colliders
- **Warning:** Avoid Box Collider 2D
- **Tip:** Use isometric view and the gizmos to cycle around and edit the collider with a clear perspective
- **Tip:** For the Trigger to work, at least one of the objects needs a rigidbody component




# Step 4: Destroy objects on collision

*Now that the animals and the projectile have Box Colliders with triggers, we need to code a new script in order to destroy them on impact.*

1. Create a new **DetectCollisions.cs** script, add it to each animal prefab, then **open** it
2. Before the final **}** add ***OnTriggerEnter*** function using **autocomplete**
3. In ***OnTriggerEnter***, put ***Destroy(gameObject);***, then test
4. In ***OnTriggerEnter***, put ***Destroy(other.gameObject);***

- **New Concept:** Overriding Functions
- **New Function:** OnTriggerEnter
- **Tip:** The "other" in OnTriggerEnter refers to the collider of the other object
- **Tip:** Use VS's Auto-Complete feature for OnTriggerEnter and any/all override functions

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

© Unity 2021

## Step 5: Trigger a "Game Over" message

*The player can defend their field against animals for as long as they wish,  but we should let them know when they've lost with a "Game Over" message if any animals get past the player.*

1. In DestroyOutOfBounds.cs, in the **else-if condition** that checks if the animals reach the bottom of the screen, add a Game Over messsage: ***Debug.Log("Game Over!")***
2. Clean up your code with **comments**
3. If using Visual Studio, Click *Edit > Advanced > **Format document*** to fix any indentation issues (On a **Mac**, click *Edit > Format > Format Document)*

- **New Functions:** Debug.Log, LogWarning, LogError
- **Tip:** Tweak some values to adjust the difficulty of your game. It might too easy!

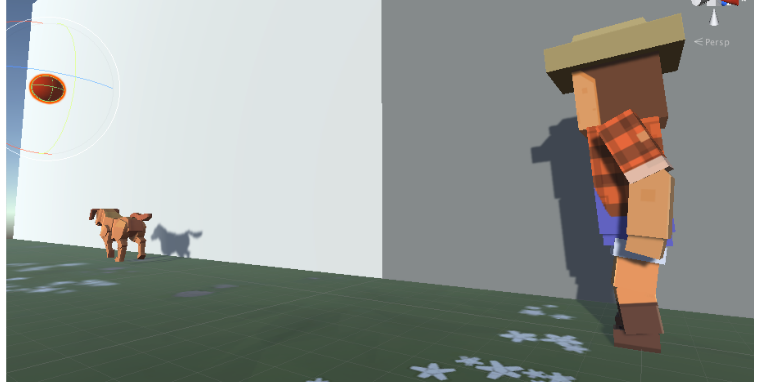
```
void Update() {
  if (transform.position.z > topBound)
  {
    Destroy(gameObject);
  } else if (transform.position.z < lowerBound)
  {
    Debug.Log("Game Over!");
    Destroy(gameObject);
  }
}
```

## Lesson Recap

| | |
|---|---|
| **New Functionality** | • Animals spawn on a timed interval and walk down the screen<br>• When animals get past the player, it triggers a "Game Over" message<br>• If a projectile collides with an animal, both objects are removed |
| **New Concepts and Skills** | • Create custom methods/functions<br>• InvokeRepeating() to repeat code<br>• Colliders and Triggers<br>• Override functions<br>• Log Debug messages to console |

# Challenge 2

## Play Fetch



**Challenge Overview:** Use your array and random number generation skills to program this challenge where balls are randomly falling from the sky and you have to send your dog out to catch them before they hit the ground. To complete this challenge, you will have to make sure your variables are assigned properly, your if-statements are programmed correctly, your collisions are being detected perfectly, and that objects are being generated randomly.

**Challenge Outcome:**

- A random ball (of 3) is generated at a random x position above the screen
- When the user presses spacebar, a dog is spawned and runs to catch the ball
- If the dog collides with the ball, the ball is destroyed
- If the ball hits the ground, a “Game Over” debug message is displayed
- The dogs and balls are removed from the scene when they leave the screen

**Challenge Objectives:** In this challenge, you will reinforce the following skills/concepts:

- Assigning variables and arrays in the inspector
- Editing colliders to the appropriate size
- Testing xyz positions with greater/less than operators in if-else statements
- Randomly generating values and selecting objects from arrays

**Challenge Instructions:**

- Open your **Prototype 2** project
- **Download** the "Challenge 2 Starter Files" from the Tutorial Materials section, then double-click on it to **Import**
- In the *Project Window > Assets > Challenge 2 >* ***Instructions*** folder, use the "Challenge 2 - Instructions" and “Outcome” video as a guide to complete the challenge

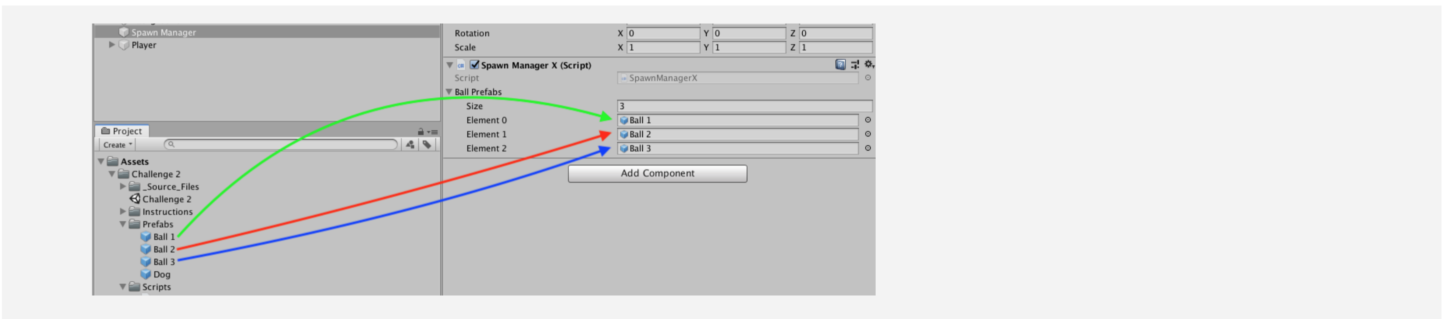
© Unity 2021

| | Challenge | Task | Hint |
|---|---|---|---|
| 1 | Dogs are spawning at the top of the screen | Make the balls spawn from the top of the screen | Click on the Spawn Manager object and look at the “Ball Prefabs” array |
| 2 | The player is spawning green balls instead of dogs | Make the player spawn dogs | Click on the Player object and look at the “Dog Prefab” variable |
| 3 | The balls are destroyed if anywhere near the dog | The balls should only be destroyed when coming into direct contact with a dog | Check out the box collider on the dog prefab |
| 4 | Nothing is being destroyed off screen | Balls should be destroyed when they leave the bottom of the screen and dogs should be destroyed when they leave the left side of the screen | In the DestroyOutOfBounds script, double-check the lowerLimit and leftLimit variables, the greater than vs less than signs, and which position (x,y,z) is being tested |
| 5 | Only one type of ball is being spawned | Ball 1, 2, and 3 should be spawned randomly | In the SpawnRandomBall() method, you should declare a new random ***int index*** variable, then incorporate that variable into the Instantiate call |

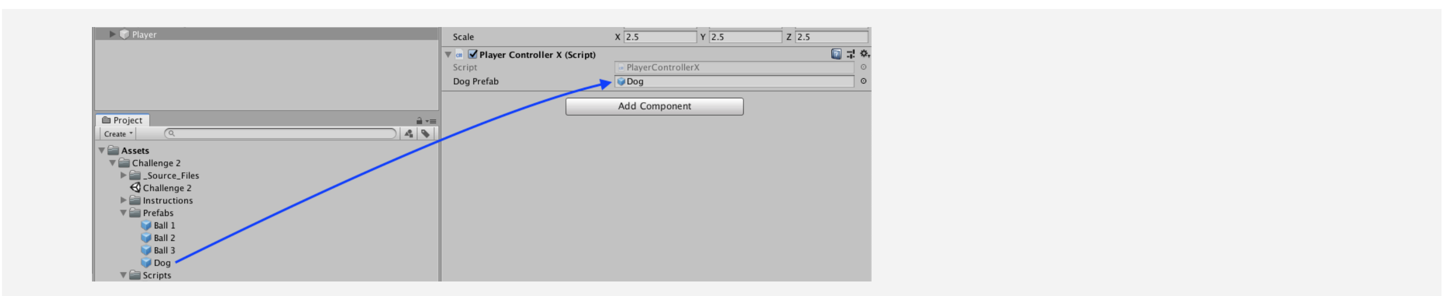
| | Bonus Challenge | Task | Hint |
|---|---|---|---|
| X | The spawn interval is always the same | Make the spawn interval a random value between 3 seconds and 5 seconds | Set the spawnInterval value to a new random number between 3 and 5 seconds in the SpawnRandomBall method |
| Y | The player can “spam” the spacebar key | Only allow the player to spawn a new dog after a certain amount of time has passed | Search for `Time.time` in the Unity Scripting API and look at the example. And don’t worry if you can’t figure it out - this is a *very difficult* challenge. |

# Challenge Solution

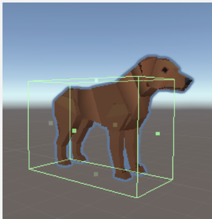
**1** Select the Spawn Manager object and expand the “Ball Prefabs” array, then drag the **Ball 1, 2, 3** prefabs from *Assets > Challenge 2 > Prefabs* onto **Element 0, 1, 2**




**2** Select the Player object and drag the **Dog** prefab from *Assets > Challenge 2 > Prefabs* onto the “Dog Prefab” variable




**3** Double-click on the Dog prefab, then in the Box Collider component, click **Edit Collider**, and reduce the collider to be the same size as the dog



**4** In DestroyOutOfBoundsX.cs, make the leftLimit a negative value, change the greater than to a less than when testing the x position, and test the y value instead of the z for the bottom limit

```
private float leftLimit = -30;
private float bottomLimit = -5;

void Update() {
  if (transform.position.x > < leftLimit) {
    Destroy(gameObject);
  } else if (transform.position.z y < bottomLimit) {
    Destroy(gameObject);
  }
}
```

© Unity 2021

**5** In the SpawnRandomBall() method, declare a new random ***int index*** variable between 0 and the length of the Array, then incorporate that index variable into the the Instantiate call

```
void SpawnRandomBall ()
{
  // Generate random ball index and random spawn position
  int index = Random.Range(0, ballPrefabs.Length);
  Vector3 spawnPos = new Vector3(Random.Range(spawnXLeft, spawnXRight), spawnPosY, 0);

  // instantiate ball at random spawn location
  Instantiate(ballPrefabs[0 index], spawnPos, ballPrefabs[0 index].transform.rotation);
}
```

# Bonus Challenge Solution

**X1** In SpawnManagerX, the "InvokeRepeating" method will not work to accomplish this, since it is only capable of calling a single, unchanging method at a pre-set spawnInterval. Instead, we could use the simpler "Invoke" method (which does not specify a spawnInterval), and then in the in SpawnRandomBall() method, randomly reset ***startDelay*** using Random.Range() and re-call the SpawnRandomBall() method again from within the method itself.

```
private float spawnInterval = 4.0f;

void Start ()
{
  InvokeRepeating("SpawnRandomBall", startDelay, spawnInterval);
}

void SpawnRandomBall ()
{
  startDelay = Random.Range(3, 5);
  ...
  Invoke("SpawnRandomBall", startDelay);
}
```

**Y1** In PlayerControllerX.cs, declare and initialize new fireRate and nextFire variables. Your "fireRate" will represent the time the player has to wait in seconds, and the nextFire variable will indicate the time (in seconds since the game started) at which the player will be able to fire again (starting at 0.0)

```
public GameObject dogPrefab;
private float fireRate = 1; // time the player has to wait to fire again
private float nextFire = 0; // time since start after which player can fire again
```

**Y2** In the if-statement checking if the player pressed spacebar, add a new condition to check that Time.time (the time in seconds since the game started) is *greater* than nextFire (which represents the time after which the player is allowed to fire. If so, nextFire should be *reset* to the current time plus the fireRate.

```
// On spacebar press, if enough time has elapsed since last fire, send dog
if (Input.GetKeyDown(KeyCode.Space) && Time.time > nextFire)
{
  nextFire = Time.time + fireRate; // reset nextFire to current time + fireRate
  Instantiate(dogPrefab, transform.position, dogPrefab.transform.rotation);
}
```

# Unit 2 Lab

## New Project with Primitives

**Steps:**

*Step 1: Create a new Unity Project*

*Step 2: Create a background plane*

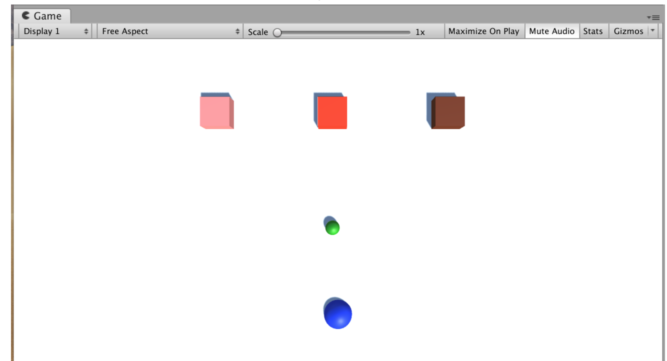
*Step 3: Create primitive Player and material*

*Step 4: Position camera based on project type*

*Step 5: Enemies, obstacles, and projectiles*

*Step 6: Export a Unity Package backup file*

*Example of progress by end of lab*




**Length:** 60 minutes

**Overview:** You will create and set up the project that will soon transform into your very own Personal Project. For now, you will use “primitive” shapes (such as spheres, cubes, and planes) as placeholders for your objects so that you can add functionality as efficiently as possible without getting bogged down by graphics. To make it clear which object is which, you will also give each object a unique colored material.

**Project Outcome:** All key objects are in the scene as primitive objects with the camera positioned properly for your project type.

**Learning Objectives:** By the end of this lab, you will be able to:
- Create a simple plane as a background for your project
- Position the camera, background, and player appropriately depending on the type of project you are creating
- Create primitive shapes to serve as placeholders for your GameObjects
- Create new colored materials and apply them to distinguish GameObjects

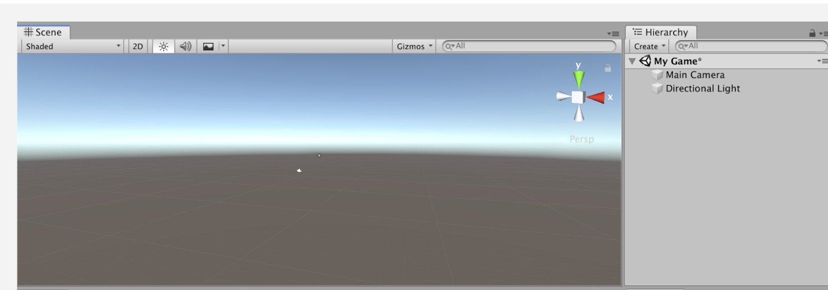
© Unity 2021

# Step 1: Create a new Unity Project

*Just like we did with the Prototype, the first thing we need to do is create a new blank project*

1. Open **Unity Hub** and create an empty project named "Personal Project" in your course directory on the correct Unity version
   If you forget how to do this, refer to Lesson 1.1, step 1.
2. After Unity opens, select your custom **Layout**
3. In the Project window, *Assets > Scenes*, rename "**SampleScene**" to "My Game"

- **Tip:** If there are multiple people with the same name using the computer, might want to add last initial
- **Don't worry:** There will just be a Main camera and directional light in there

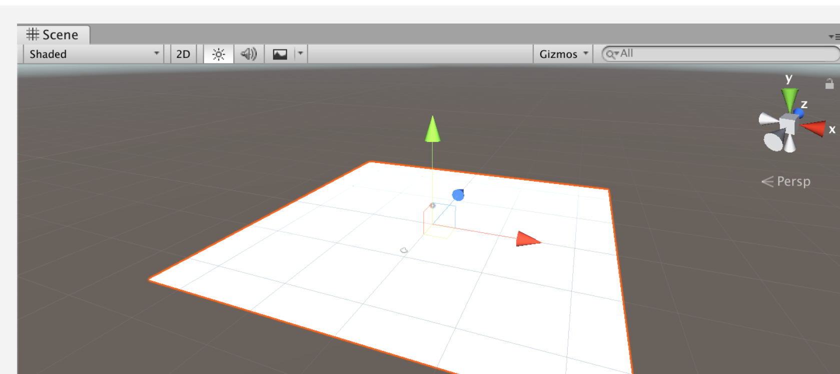



# Step 2: Create a background plane

*To orient yourself in the scene and not feel like you're floating around in mid-air, it's always good to start by adding a background / ground object*

1. In the Hierarchy, *Right-click > 3D Object > **Plane*** to add a plane to your scene
2. In the Plane's Inspector, in the top-right of the Transform component, click on the three dots icon ***> Reset***
   **Note**: the three dots will appear as a gear icon in older versions of Unity.
3. Increase the **XYZ scale** of the plane to (5, 1, 5)
4. Adjust your position in Scene view so you have a good view of the Plane

- **Explanation:** Working with **primitives** - these are simple objects that allow you to work faster



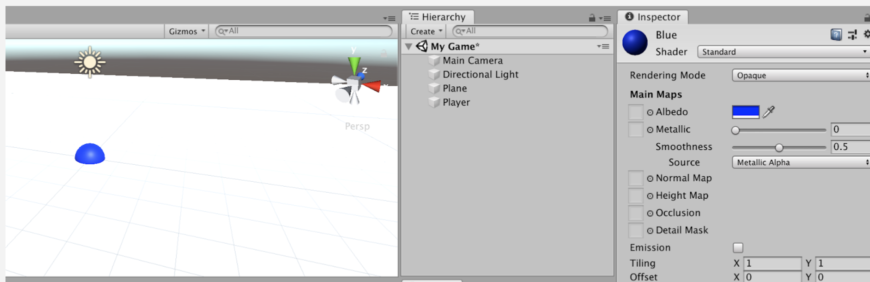

© Unity 2021

# Step 3: Create primitive Player and material

*Now that we have the empty plane object set up, we can add the star of the show: the player object*

1. In the Hierarchy, *Right-click > 3D Object > **Sphere***, then rename it "Player"
2. In Assets, *Right-click > Create > **Folder*** named "Materials"
3. Inside "Materials", *Right-click > Create > **Material*** and rename it "Blue"
4. In Blue's Inspector, click on the **Albedo color** box and change it to a blue
5. **Drag** the material from your Assets onto the Player object

- **Tip**: Using primitives doesn't let graphics distract you and get in the way of core features,
- **Explanation:** Albedo is a reference to astronomical light reflection properties - but it's basically just the material's color
- **Warning:** Stick with blue right now so it's easy to follow - you'll be replacing it later



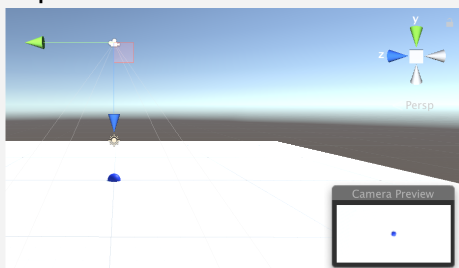

# Step 4: Position camera based on project type

*Now that we have the player in there, we need the best view of it, depending on our type of project*

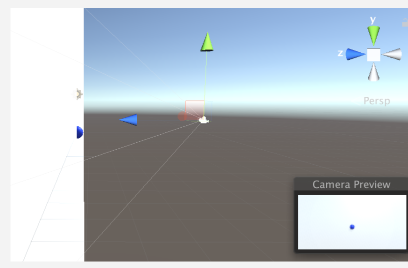
1. For a **top-down** game, position the camera at (0, 10, 0) directly over the player and rotate it 90 degrees on the **X axis**
2. For a **side-view** game, rotate the **Plane** by -90 degrees on the **X axis**
3. For a **third-person** view game, move the camera up on the **Y and Z axes** and increase its **rotation on the X axis**

- **Tip:** Side view looks like top view, but it'll make a big diff when you apply gravity
- **Don't worry:** You might not know exact view yet - just go with what's in your design doc

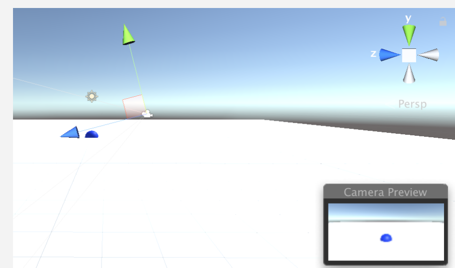
Top-down view




Side-view




Isometric view



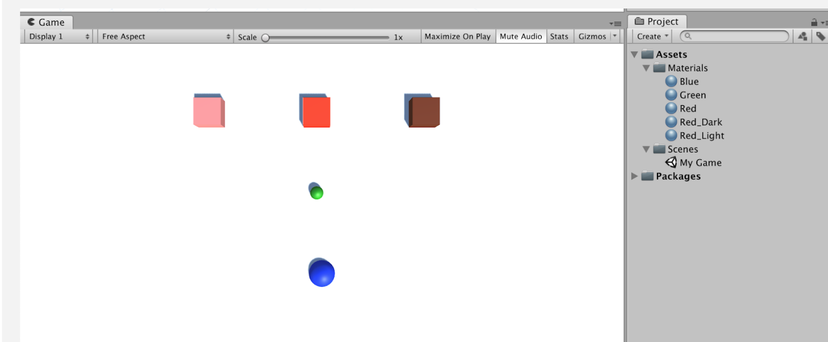

© Unity 2021

# Step 5: Enemies, obstacles, and projectiles

*Now that we know how to make primitives, let's go ahead and make one for each object in our project*

1. In the Hierarchy, create new **Cubes**, **Spheres**, and **Capsules** for all other main objects, **renaming** them, **repositioning** them, and **scaling** them
2. In your Materials folder, create **new materials** for as many colors as you have unique objects, editing their color to match their name, then **apply** those materials to your objects
3. Position all of your objects in locations relative to each other that make sense

- **Tip:** If you plan on having variants of certain objects (e.g. multiple animals), create dark/light shades of the same color
- **Tip:** Good to make enemies red - easy if everyone uses the same conventions

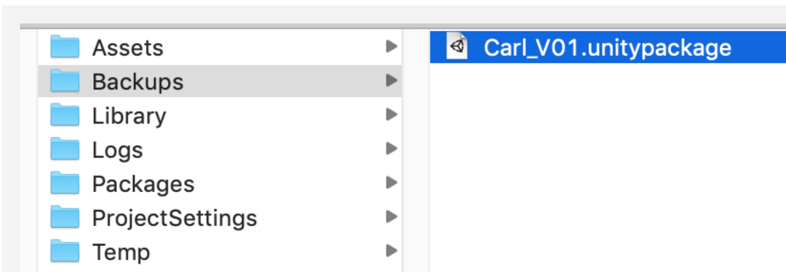



# Step 6: Export a Unity Package backup file

*Since we're going to be putting our hearts and souls into this project, it's always good to make backups*

1. **Save** your Scene
2. In the Project window, Right-click on the *"Assets" folder > **Export Package***, then click Export
3. Create a **new "<u>Backups</u>" folder** in your Personal Project folder, then **save** it with your name and the version number (e.g. Carl_V0.1.unitypackage")

- **Explanation:** The "include dependencies" checkbox will include any files that are tied to / used by anything else we're exporting
- **Tip:** This is the same file type that you *imported* at the start of Prototype 1



## Lesson Recap

| | |
|---|---|
| **New Progress** | ● New project for your Personal Project<br>● Camera positioned and rotated based on project type<br>● All key objects in scene with unique materials |
| **New Concepts and Skills** | ● Primitives<br>● Create new materials<br>● Export Unity packages |

© Unity 2021

# Quiz Unit 2

| QUESTION | CHOICES |
|---|---|
| **1** If it says, “`Hello there!`” in the console, what was the code used to create that message? | a. Debug(“Hello there!”);<br>b. Debug.Log("Hello there!");<br>c. Debug.Console(“Hello there!”);<br>d. Debug.Log(Hello there!); |
| **2** If you want to destroy an object when its ***health reaches 0***, what code would be best in the blank below? | a. health > 0<br>b. health.0<br>c. health < 1<br>d. health < 0 |
| **3** The code below creates an error that says, “error CS1503: Argument 1: cannot convert from 'UnityEngine.GameObject[]' to 'UnityEngine.Object'”. What could you do to remove the errors? | a. On line 1, change “GameObject[]” to “GameObject”<br>b. On line 1, change “enemyPrefabs” to “enemyPrefabs[0]”<br>c. On line 3, change “Start()” to “Update()”<br>d. On line 5, change “enemyPrefabs” to “enemyPrefabs[0]”<br>e. Either A or D<br>f. Both A and D together<br>g. Both B and C together |

Question 2 code:

```
private int health = 0;

void Update() {
  if (__________) {
    Destroy(gameObject);
  }
}
```

Question 3 code:

```
1. public GameObject[] enemyPrefabs;
2.
3. void Start()
4. {
5.     Instantiate(enemyPrefabs);
6. }
```

**4** Which comment best describes the following code?

```
public class PlayerController : MonoBehaviour
{
  // Comment
  private void OnTriggerEnter(Collider other) {
    Destroy(other.gameObject);
  }
}
```

a. // If player collides with another object, destroy player
b. // If enemy collides with another object, destroy the object
c. // If player collides with a trigger, destroy trigger
d. // If player collides with another object, destroy the object

**5** If you want to move the character **up continuously** as the player presses the **up arrow**, what code would be best in the two blanks below:

```
if (Input.__________(__________))
{
    transform.Translate(Vector3.up);
}
```

a. GetKey(KeyCode.UpArrow)
b. GetKeyDown(UpArrow)
c. GetKeyUp(KeyCode.Up)
d. GetKeyHeld(Vector3.Up)

**6** Read the documentation from the Unity Scripting API and the code below. Which of the following are possible values for the randomFloat and randomInt variables?

public static float **Range**(float **min**, float **max**);

## Description

Return a random float number between `min` [inclusive] and `max` [inclusive] (Read Only).

Note `max` is inclusive. `Random.Range(0.0f, 1.0f)` can return `1.0` as the value. The Random.Range distribution is uniform. Range is a Random Number Generator.

public static int **Range**(int **min**, int **max**);

## Description

Return a random integer number between `min` [inclusive] and `max` [exclusive] (Read Only).

Note `max` is exclusive. `Random.Range(0, 10)` can return a value between 0 and 9. Return `min` if `max` equals `min`. The Random.Range distribution is uniform. Range is a Random Number Generator.

a. randomFloat = 100.0f; randomInt = 0;
b. randomFloat = 100.0f; randomInt = 100;
c. randomFloat = 50.5f; randomInt = 100;
d. randomFloat = 0.0f; randomInt = 50.5;

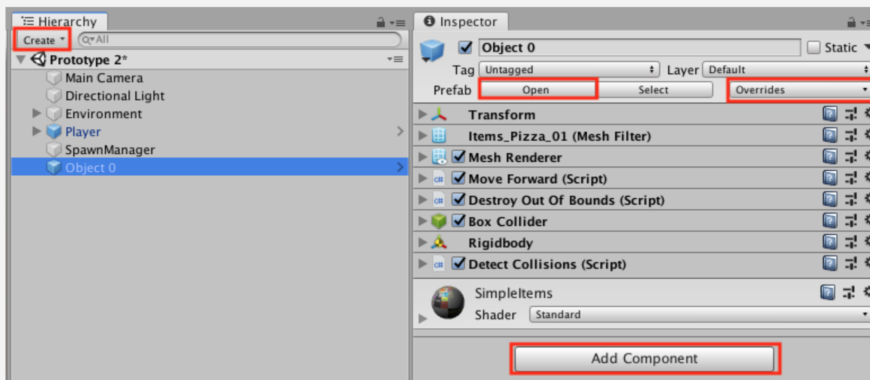
```
float randomFloat = Random.Range(0, 100);
int randomInt = Random.Range(0, 100);
```

**7** You are trying to randomly spawn objects from an array. However, when your game is running, you see in the console that there was an "error at Assets/Scripts/SpawnManager.cs:5. IndexOutOfRangeException: Index was outside the bounds of the array." Which line of code should you edit in order to resolve this problem and still retain the random object functionality?

```
1. public GameObject[] randomObjects;
2.
3. void SpawnRandomObject() {
4.   int objectIndex = Random.Range(0, 3);
5.   Instantiate(randomObjects[objectIndex]);
6. }
```

a. Line 2
b. Line 3
c. Line 4
d. Line 5

**8** If you have made changes to a prefab in the scene and you want to apply those changes to all prefabs, what should you click?




a. The "Create" drop-down at the top of the Hierarchy
b. The "Open" button at the top of the Inspector
c. The "Overrides" drop-down at the top of the Inspector
d. The "Add Component" button at the bottom of the Inspector

**9** Read the documentation from the Unity Scripting API below. Which of the following is a correct use of the InvokeRepeating method?

public void **InvokeRepeating**(string **methodName**, float **time**, float **repeatRate**);

**Description**

Invokes the method `methodName` in `time` seconds, then repeatedly every `repeatRate` seconds.

a. InvokeRepeating("Spawn, 0.5f, 1.0f");
b. InvokeRepeating("Spawn", 0.5f, 1.0f);
c. InvokeRepeating("Spawn", gameObject, 1.0f);
d. InvokeRepeating(0.5f, 1.0f, "Spawn");

**10** You're trying to create some logic that will tell the user to speed up if they're going too slow or to slow down if they're going too fast. How should you arrange the lines of code below to accomplish that?

```
1. Debug.Log(speedUp); }
2. else if (speed > 60) {
3. private string speedUp = "Speed up!";
4. void Update() {
5. Debug.Log(slowDown); }
6. if (speed < 10) {
7. private float speed;
8. private string slowDown = "Slow down!";
9. }
```

a. 4, 6, 1, 2, 5, 9, 7, 8, 3

```
void Update()
{

   if (speed < 10)
{Debug.Log(speedUp); }

else if (speed > 60) {
Debug.Log(slowDown); }
}

private float speed;
private string slowDown =
"Slow down!";
private string speedUp =
"Speed up!";
```

b. 6, 1, 2, 5, 7, 8, 3, 4, 9

```
if (speed < 10) {
Debug.Log(speedUp); }
else if (speed > 60) {
Debug.Log(slowDown); }
private float speed;
private string slowDown =
"Slow down!";
private string speedUp =
"Speed up!";
void Update() {
}
```

c. 7, 8, 3, 4, 6, 5, 2, 1, 9

```
private float speed;
private string slowDown =
"Slow down!";
private string speedUp =
"Speed up!";
void Update() {
if (speed < 10) {
Debug.Log(slowDown); }
else if (speed > 60) {
Debug.Log(speedUp); }
}
```

d. 7, 8, 3, 4, 6, 1, 2, 5, 9

```
private float speed;
private string slowDown =
"Slow down!";
private string speedUp =
"Speed up!";
void Update() {
if (speed < 10) {
Debug.Log(speedUp); }
else if (speed > 60) {
Debug.Log(slowDown); }
}
```

# Quiz Answer Key

| # | ANSWER | EXPLANATION |
|---|---|---|
| 1 | B | Debug.Log() prints messages to the console and can accept String parameters between quotation marks, such as “Hello there!” |
| 2 | C | Since the “health” variable is an int, anything less than 1 would be “0”. The sign for “less than” is “<”. |
| 3 | E | “GameObject[]” is a GameObject array. You cannot instantiate an array, but you *can* instantiate an object inside an array. So you could either remove the array and have Instantiate use an individual object (option A) or you could use an GameObject index of that Array (option D), but both would not work. |
| 4 | D | Since it’s inside the PlayerController class, and it is destroying ***other****.gameObject*, it is destroying something that the player collides with. |
| 5 | A | “Input.GetKey” tests for the user holding down a key (as opposed to KeyKeyDown, which test for a single press down of a Key). |
| 6 | A | As it says in the documentation, Random.Range does *not* include the maximum value for integers, but *does* include the maximum value for floats. This means that randomInt *cannot* be 100, but randomFloat can be. |
| 7 | C | Line 4, which generates the objectIndex, must be generating an index value that is too high for the number of objects in the array. The best thing to do would be to change it to “Random.Range(0, randomObjects.Length); |
| 8 | C | The “Override” drop-down will allow you to apply any changes you’ve made to your individual prefab to the original prefab object. |
| 9 | B | According to the Scripting API, InvokeRepeating requires a string parameter, then two floats. |
| 10 | D | All variables should be declared first, then the void method, then the if-condition telling them to speed up, then the else condition telling them to slow down. |

© Unity 2021

# Bonus Features 2 - Share your Work

**Steps:**

*Step 1: Overview*

*Step 2: Easy: Obstacle pyramids*

*Step 3: Medium: Oncoming vehicles*

*Step 5: Hard: Camera switcher*

*Step 6: Expert: Local multiplayer*

*Step 7: Hints and solution walkthrough*

*Step 8: Share your work*

**Length:** 60 minutes

**Overview:** In this tutorial, you can go way above and beyond what you learned in this Unit and share what you've made with your fellow creators.

There are four bonus features presented in this tutorial marked as Easy, Medium, Hard, and Expert. You can attempt any number of these, put your own spin on them, and then share your work!

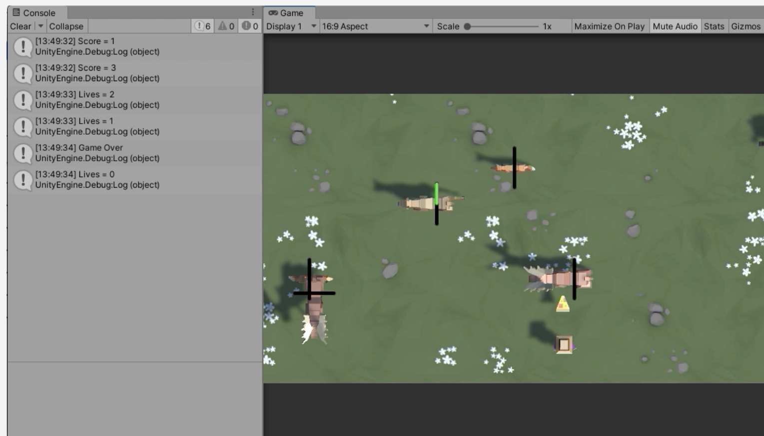
This tutorial is entirely optional, but highly recommended for anyone wishing to take their skills to a new level.

## Step 1: Overview

This tutorial outlines four potential bonus features for the Feed the Animals Prototype at varying levels of difficulty:

- **Easy**: Vertical player movement
- **Medium**: Aggressive animals
- **Hard**: Game user interface
- **Expert**: Animal hunger bar

Here’s what the prototype could look like if you complete all four features:




The Easy and Medium features can probably be completed entirely with skills from this course, but the Hard and Expert features will require some additional research.

Since this is optional, you can attempt none of them, all of them, or any combination in between. You can come up with your own original bonus features as well!

Then, at the end of this tutorial, there is an opportunity to share your work.

We highly recommend that you attempt these using relentless Googling and troubleshooting, but if you do get completely stuck, there are hints and step-by-step solutions available below.

Good luck!

© Unity 2021

## Step 2: Easy: Vertical player movement

Allow the player to move forward and backwards within a certain range.
This makes the game a bit more dynamic and allows for the addition of other features.

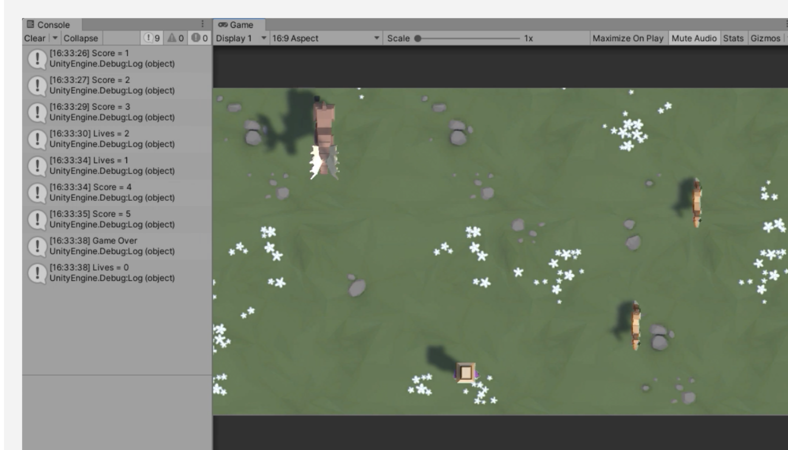
## Step 3: Medium: Aggressive animals

Have animals that also spawn from the left and right side of the screen. If one of them hits you, "Game Over" should be logged to the console.
This will make the game much more exciting and requires the player to stay on their toes, especially if vertical movement is also implemented.

© Unity 2021

## Step 5: Hard: Game user interface

At the start of the game, display in the console that the player's Lives = 3 and Score = 0. If the player feeds an animal, increase and display the Score. If the player misses an animal or is hit by one, decrease and display the Lives. When the number of Lives reaches 0, log "Game Over" in the console.




## Step 6: Expert: Animal hunger bar

Display a "hunger bar" on top of each of the animals. Then, each time you feed one of them, the hunger bar fills up a little. Each animal should require different amounts of food to successfully "feed" them. They should only disappear after their hunger bars are full.

## Step 7: Hints and solution walkthrough

**Hints:**

- Easy: Vertical player movement
  - Look at how we are doing the left and right movement range.
- Medium: Aggressive animals
  - Look at how we are currently spawning animals and doing collisions.
- Hard: Game user interface
  - You will need to update the score and lives in the DetectCollisions script, and update the lives in the DestroyOutOfBounds script.
- Expert: Animal hunger bar
  - You will need to add a UI Slider object in World space Render Mode as a prefab for each animal, then set the slider's value through a script every time the animal is fed.

**Solution walkthrough**

If you are really stuck, download the step-by-step solution walkthrough.
Note that there are likely many ways to implement these features - this is only one suggestion.

## Step 8: Share your work

Have you implemented any of these bonus features? Have you added any new, unique features? Have you applied these new features to another project?

We would love to see what you've created!

Please take a screenshot of your project or do a screen-recording walking us through it, then post it here to share what you've made.

We highly recommend that you comment on at least one other creator's submission. What do you like about the project? What would be a cool new feature they might consider adding?