Γραφικά Υπολογιστών

Ιόνιο Πανεπιστήμιο
Τμήμα Πληροφορικής

Στέργιος Παλαμάς, Επίκουρος Καθηγητής

Μάθημα 6:

Προγραμματισμός στο Unity

Unit 1
Summary

In this Unit, you will program a car moving side-to-side on a floating road, trying to avoid (or hit) obstacles in the way. In addition to becoming familiar with the Unity editor and workflow, you will learn how to create new C# scripts and do some simple programming. By the end of the Unit, you will be able to call basic functions, then declare and tweak new variables to modify the results of those functions.

Το Tutorial στο Unity-Learn

Υλικό που θα χρειαστεί στο Project
Prototype 1 - Starter Files.zip

Το κατεβάζετε και το αποσυμπιέζετε σε έναν φάκελο του υπολογιστή σας

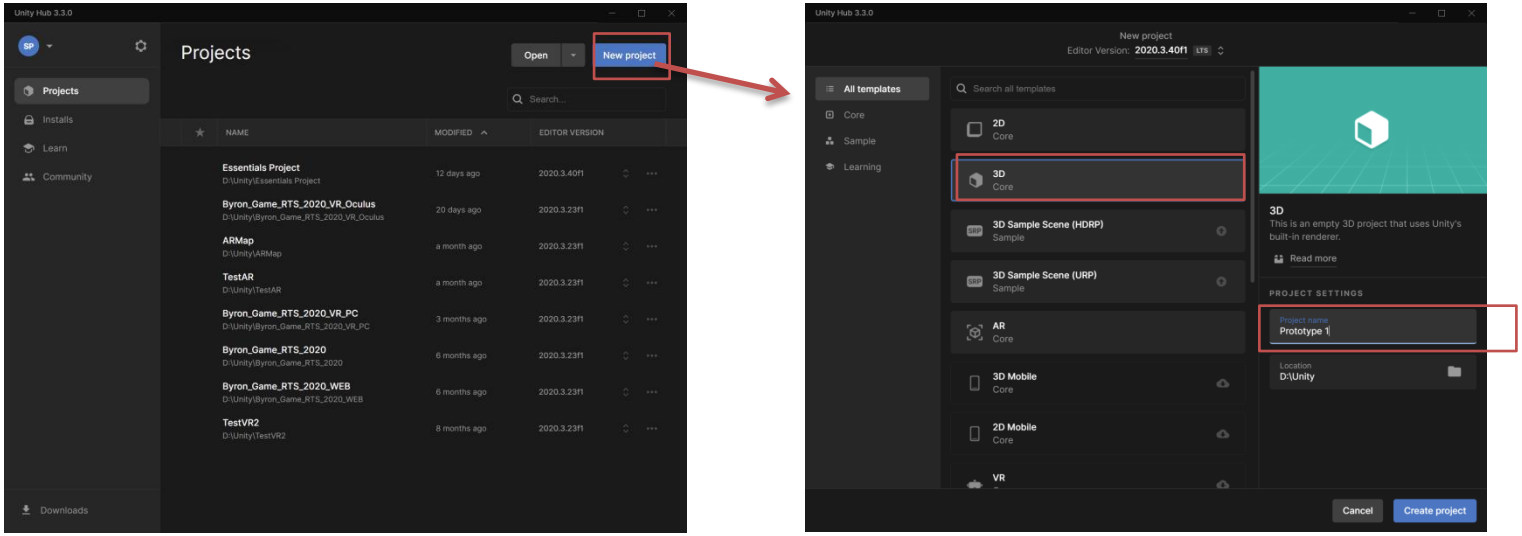
1.Make a course folder and new project

Ανοίγουμε το **Unity Hub** και επιλέγουμε την καρτέλα **Projects**

Επιλέγουμε **New** για να ξεκινήσουμε νέο Project

Επιλέγουμε το **3D** template, ονομάζουμε το project "Prototype 1", και αφήνουμε το Unity να δημιουργήσει το folder του Project στην default θέση

Πατάμε **Create**, και περιμένουμε το Unity να ξεκινήσει τον Editor.



2.Import assets and open Prototype 1

1. Click to download the Prototype 1 Starter Files, then **extract** the compressed folder. **Windows**: Right-click on the file > Extract All **Mac**: Double-click on the file

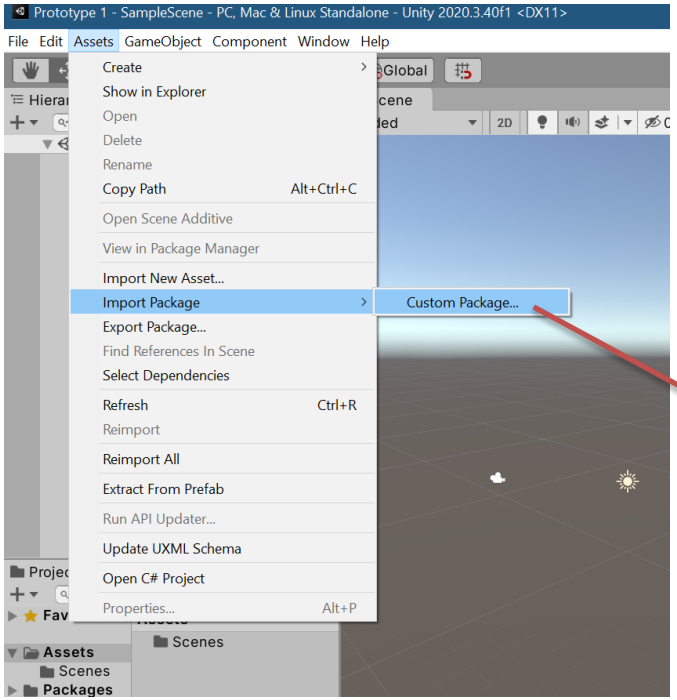
2. From the top menu in Unity, select **Assets** > **Import Package** > **Custom Package**, **then navigate to the folder you extracted and select the Prototype-1_Starter-Files.unitypackage file.**

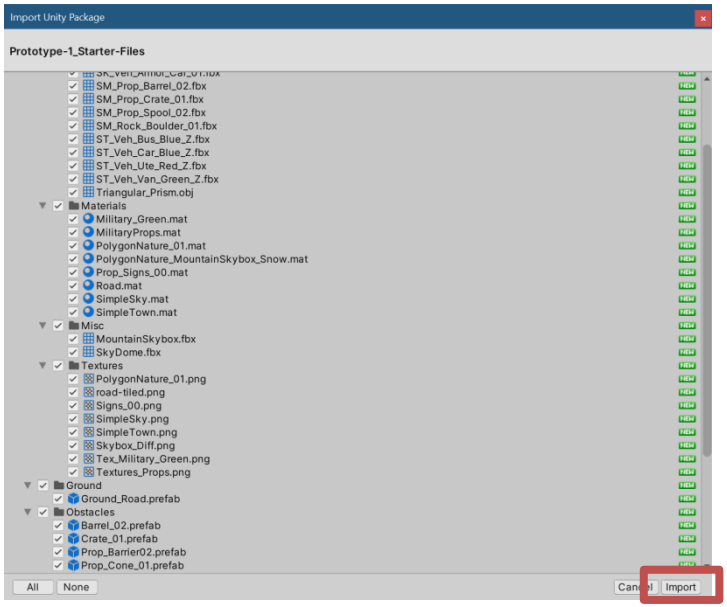
3. In the **Import Unity Package** window that pops up, select **Import** and wait for the assets to import.

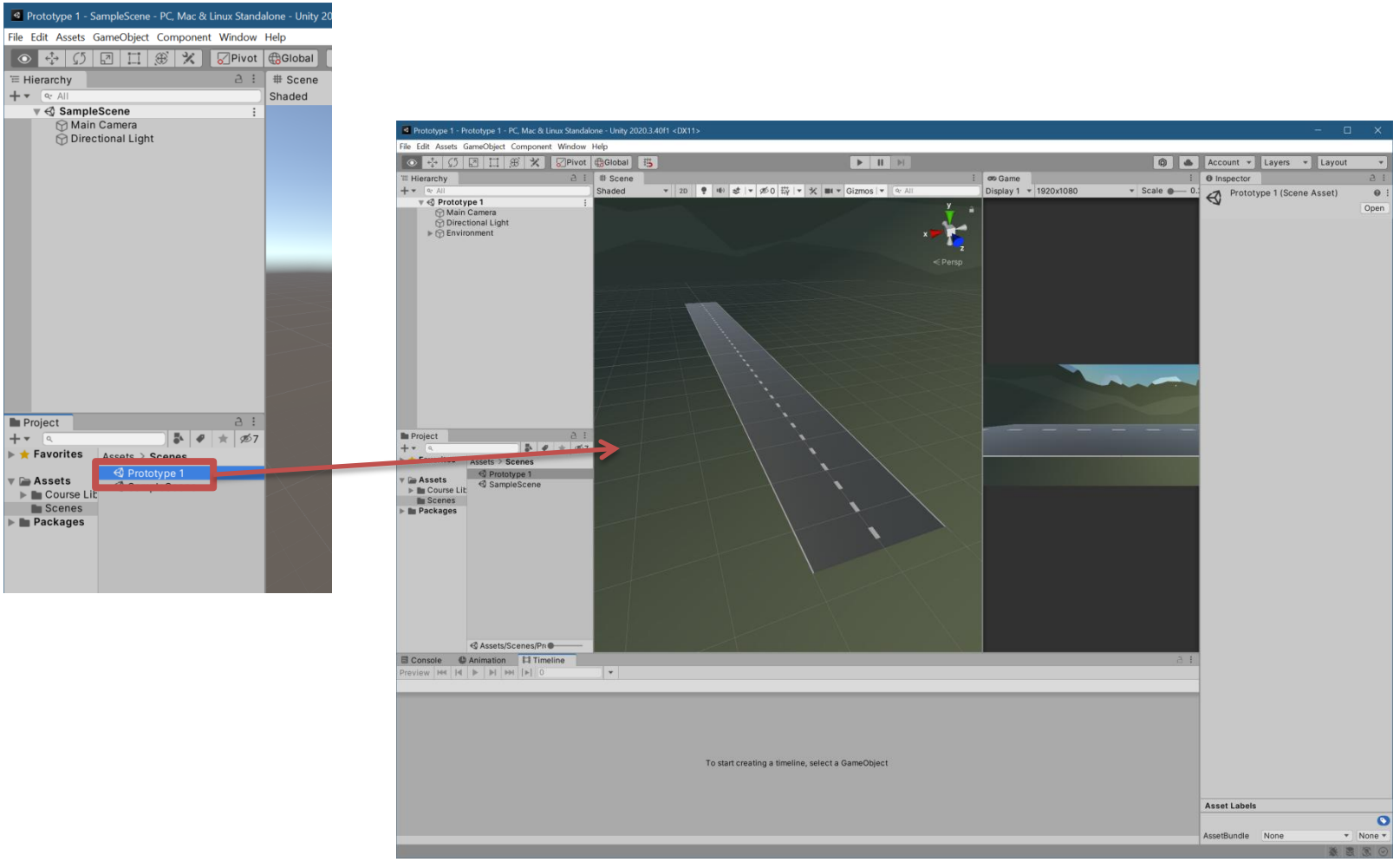
4. In the **Project** window, in *Assets* > *Scenes* > double-click on the **Prototype 1 scene** to open it

5. Delete the **Sample Scene** without saving

6. **Right-click + drag** to look around at the start of the road


Prototype 1 - SampleScene - PC, Mac & Linux Standalone - Unity 2020.3.40f1 <DX11>
File Edit Assets GameObject Component Window Help
Create
Show in Explorer
Open
Delete
Rename
Copy Path Alt+Ctrl+C
Open Scene Additive
View in Package Manager
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport
Reimport All
Extract From Prefab
Run API Updater...
Update UXML Schema
Open C# Project
Properties... Alt+P
Custom Package...
Assets
Scenes
Packages


Import Unity Package
Prototype-1_Starter-Files
SM_Prop_Barrel_02.fbx
SM_Prop_Crate_01.fbx
SM_Prop_Spool_02.fbx
SM_Rock_Boulder_01.fbx
ST_Veh_Bus_Blue_Z.fbx
ST_Veh_Car_Blue_Z.fbx
ST_Veh_Ute_Red_Z.fbx
ST_Veh_Van_Green_Z.fbx
Triangular_Prism.obj
Materials
Military_Green.mat
MilitaryProps.mat
PolygonNature_01.mat
PolygonNature_MountainSkybox_Snow.mat
Prop_Signs_00.mat
Road.mat
SimpleSky.mat
SimpleTown.mat
Misc
MountainSkybox.fbx
SkyDome.fbx
Textures
PolygonNature_01.png
road-tiled.png
Signs_00.png
SimpleSky.png
SimpleTown.png
Skybox_Diff.png
Tex_Military_Green.png
Textures_Props.png
Ground
Ground_Road.prefab
Obstacles
Barrel_02.prefab
Crate_01.prefab
Prop_Barrier02.prefab
Prop_Cone_01.prefab
All
None
Import


Prototype 1 - SampleScene - PC, Mac & Linux Standalone - Unity 20
File Edit Assets GameObject Component Window Help
Pivot
Global
Hierarchy
Scene
Shaded
SampleScene
Main Camera
Directional Light
Project
Favorites
Assets
Course Lib
Scenes
Packages
Prototype 1
Prototype 1 - Prototype 1 - PC, Mac & Linux Standalone - Unity 2020.3.40f1 <DX11>
Account
Layers
Layout
Game
Display 1
1920x1080
Scale
Inspector
Prototype 1 (Scene Asset)
Open
Gizmos
Persp
SampleScene
Console
Animation
Timeline
Preview
To start creating a timeline, select a GameObject
Asset Labels
AssetBundle
None

3.Add your vehicle to the scene

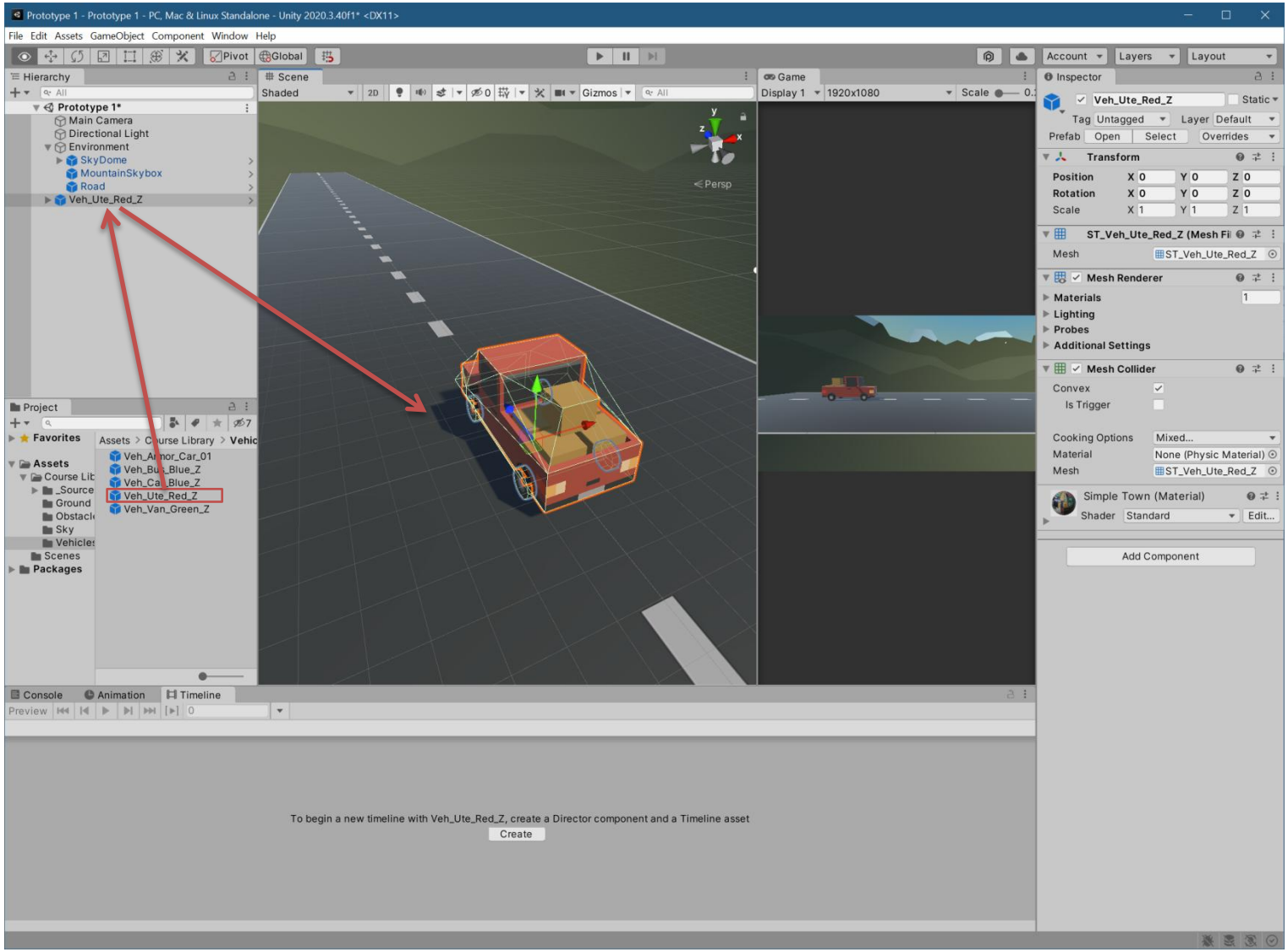
1. In the **Project Window**, open *Assets* > *Course Library* > *Vehicles*, then drag a vehicle into the **Hierarchy**

2. **Hold right-click + WASD** to fly to the vehicle, then try to rotate around it

3. With the vehicle selected and your mouse in the Scene view, **Press F** to focus on it
4. then use the **scroll wheel** to zoom in and out and **hold the scroll wheel** to pan

5. **Hold alt+left-click** to rotate around the focal point or **hold alt+right-click** to zoom in and out

6. If anything goes wrong, press **Ctrl/Cmd+Z** to Undo until it's fixed


Prototype 1 - Prototype 1 - PC, Mac & Linux Standalone - Unity 2020.3.40f1* <DX11>
File Edit Assets GameObject Component Window Help
Pivot
Global
Account
Layers
Layout
Hierarchy
Prototype 1*
Main Camera
Directional Light
Environment
SkyDome
MountainSkybox
Road
Veh_Ute_Red_Z
Scene
Shaded
2D
Gizmos
Persp
Game
Display 1
1920x1080
Scale
Inspector
Veh_Ute_Red_Z
Static
Tag Untagged
Layer Default
Prefab Open Select Overrides
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
ST_Veh_Ute_Red_Z (Mesh Fi
Mesh ST_Veh_Ute_Red_Z
Mesh Renderer
Materials 1
Lighting
Probes
Additional Settings
Mesh Collider
Convex
Is Trigger
Cooking Options Mixed...
Material None (Physic Material)
Mesh ST_Veh_Ute_Red_Z
Simple Town (Material)
Shader Standard Edit...
Add Component
Project
Favorites
Assets
Course Lib
_Source
Ground
Obstacl
Sky
Vehicles
Scenes
Packages
Assets > Course Library > Vehic
Veh_Armor_Car_01
Veh_Bus_Blue_Z
Veh_Car_Blue_Z
Veh_Ute_Red_Z
Veh_Van_Green_Z
Console
Animation
Timeline
Preview
To begin a new timeline with Veh_Ute_Red_Z, create a Director component and a Timeline asset
Create

4.Add an obstacle and reposition it

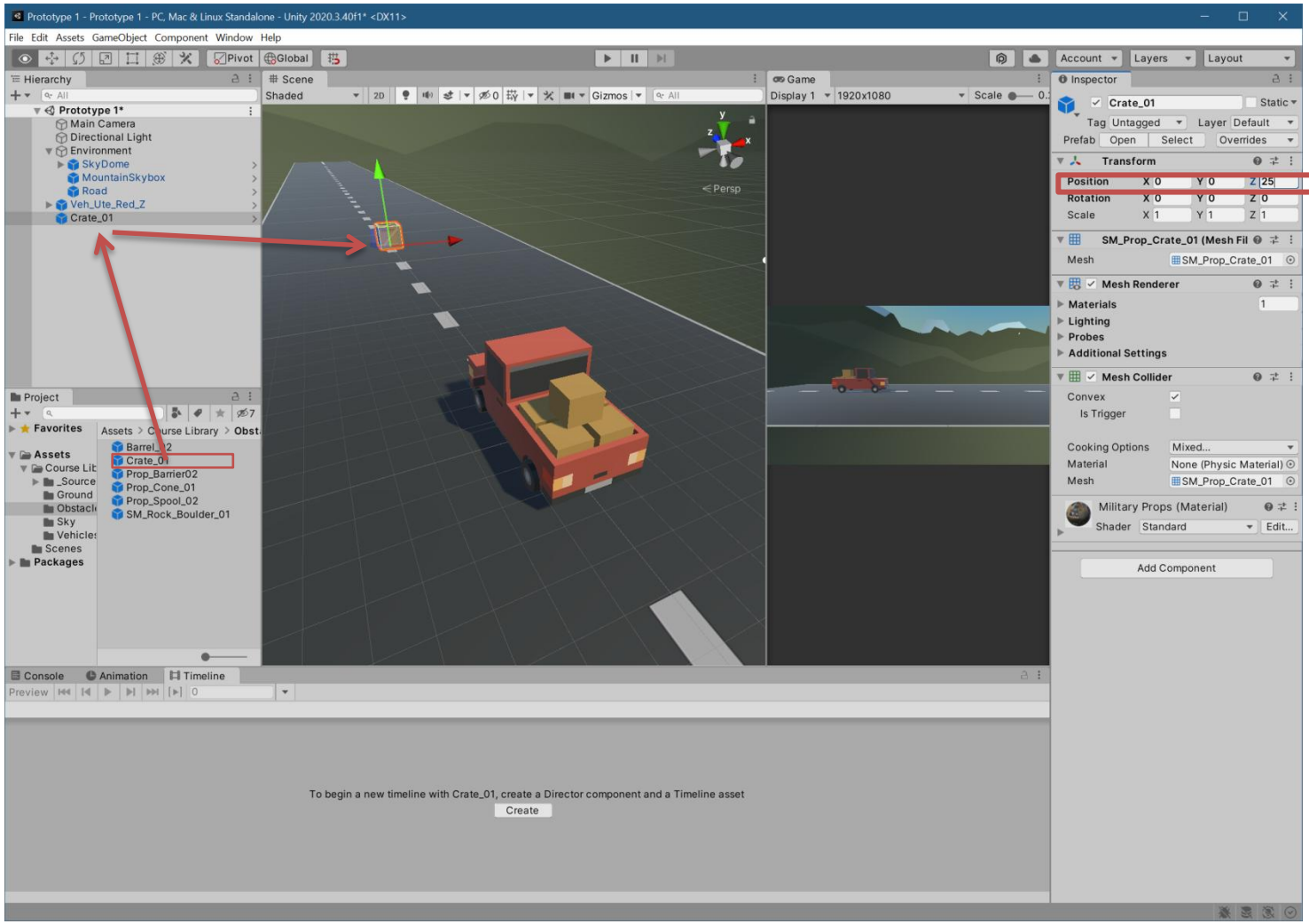
The next thing our game needs is an obstacle! We need to choose one and position it in front of the vehicle.

1. Go to *Course Library* > *Obstacles* and **drag an obstacle** directly into the **Scene view**

2. In the Inspector for your obstacle, in the top-right of the Transform component, click the more options button > ***Reset Position*** **Note**: The more options button may appear as three vertical dots or a gear icon, depending on your version of Unity

3. In the **Inspector,** change the **XYZ Location** to **x=0, y=0, z=25**

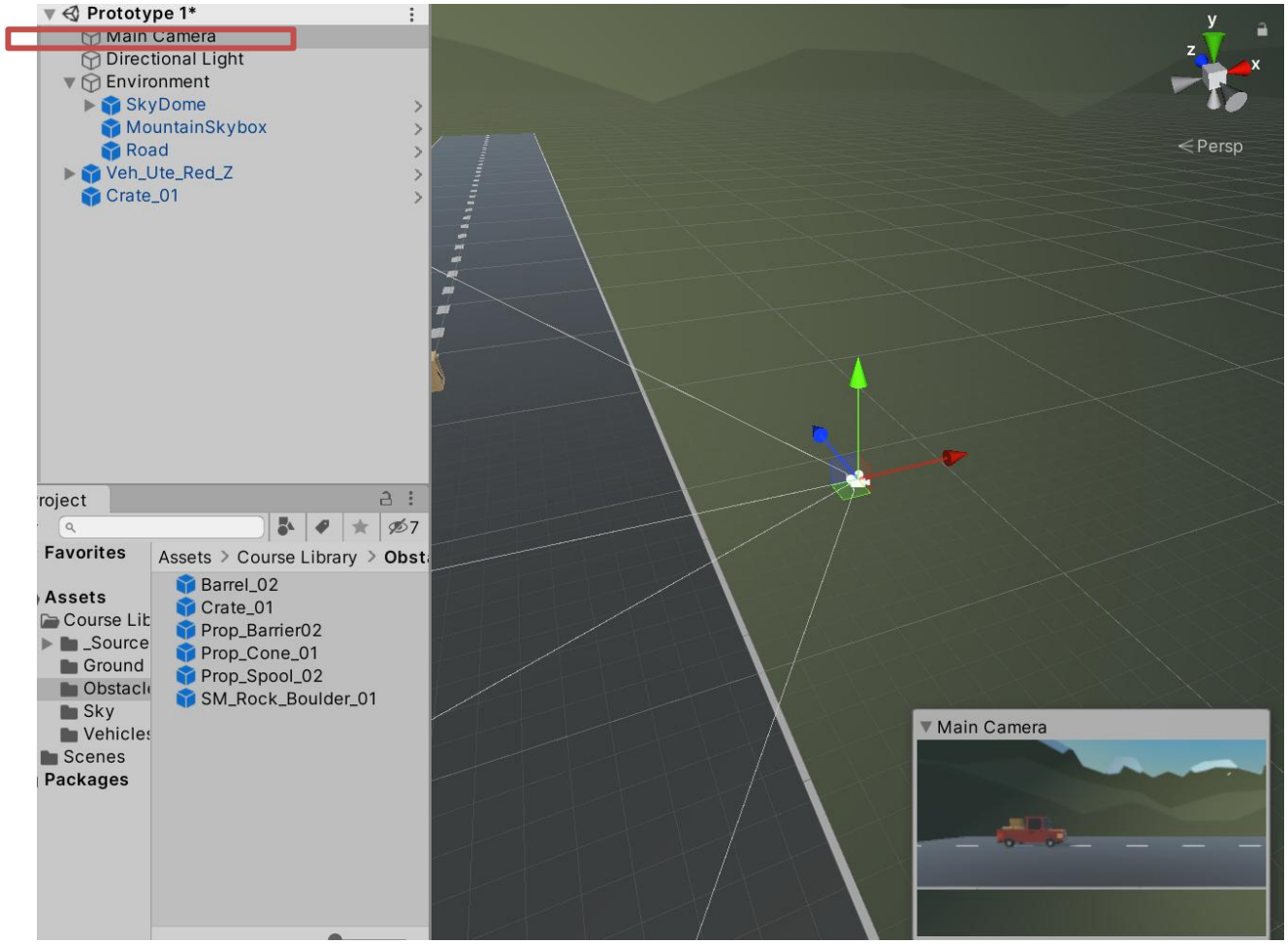
4. In the Hierarchy, *Right-click* > ***Rename*** your two objects as "Vehicle" and "Obstacle"


Prototype 1 - Prototype 1 - PC, Mac & Linux Standalone - Unity 2020.3.40f1* <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Prototype 1*
Main Camera
Directional Light
Environment
SkyDome
MountainSkybox
Road
Veh_Ute_Red_Z
Crate_01
Scene
Shaded
Persp
Game
Display 1
1920x1080
Scale
Inspector
Crate_01
Static
Tag Untagged
Layer Default
Prefab Open Select Overrides
Transform
Position X 0 Y 0 Z 25
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
SM_Prop_Crate_01 (Mesh Fil
Mesh SM_Prop_Crate_01
Mesh Renderer
Materials 1
Lighting
Probes
Additional Settings
Mesh Collider
Convex
Is Trigger
Cooking Options Mixed...
Material None (Physic Material)
Mesh SM_Prop_Crate_01
Military Props (Material)
Shader Standard Edit...
Add Component
Project
Favorites
Assets
Course Library
Barrel_02
Crate_01
Prop_Barrier02
Prop_Cone_01
Prop_Spool_02
SM_Rock_Boulder_01
Console
Animation
Timeline
Preview
To begin a new timeline with Crate_01, create a Director component and a Timeline asset
Create

5.Locate your camera and run the game

1. Select the **Camera** in the hierarchy, then **press F** to focus on it

2. Press the **Play button** to run your Game, then press Play again to **stop** it

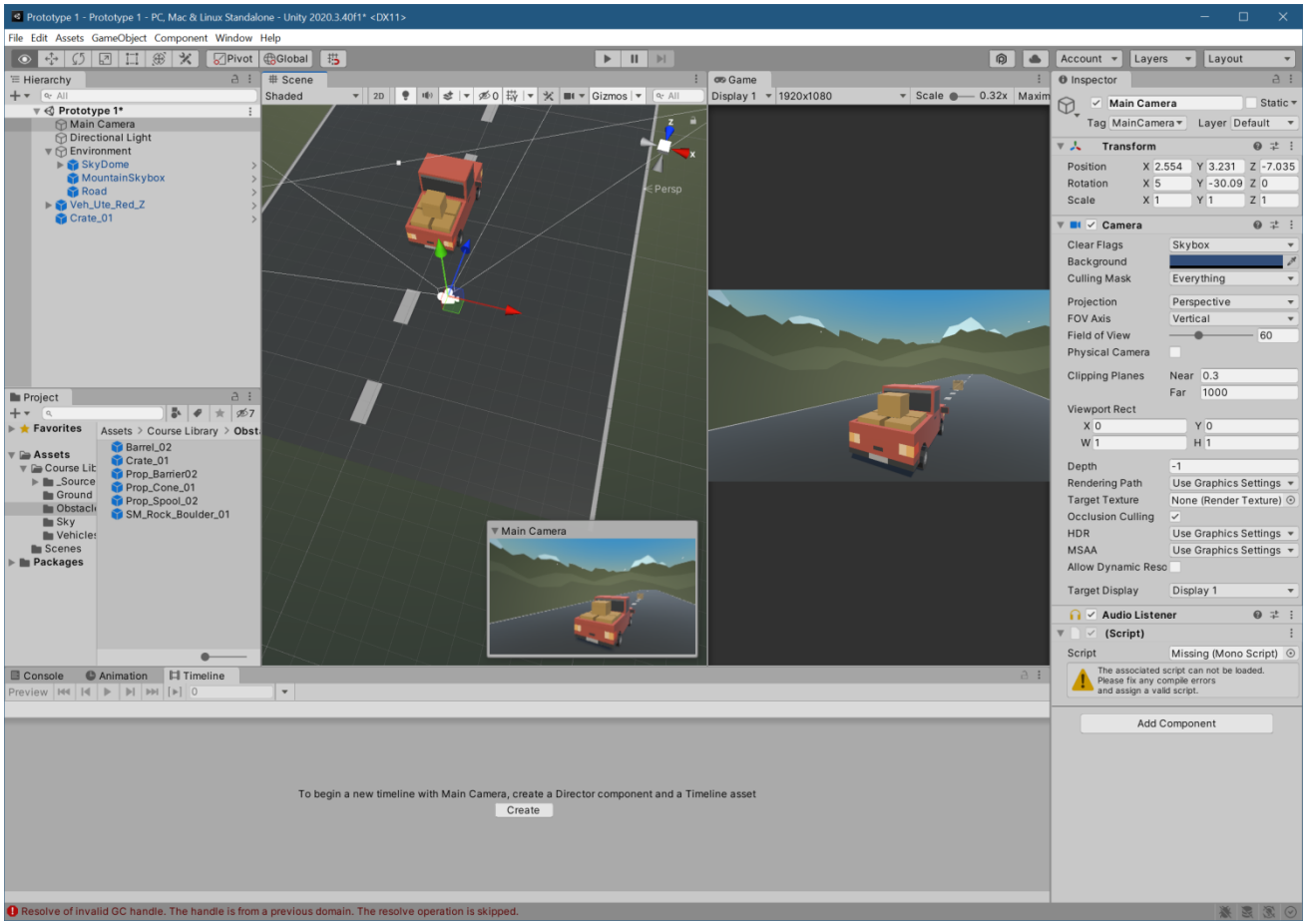


6.Move the camera behind the vehicle

In order for the player to properly view our game, we should position and angle the camera in a good spot behind the vehicle

1. Use the **Move** and **Rotate tools** to move the camera behind the vehicle looking down on it

2. **Hold Ctrl/Cmd** to move the camera by whole units



Lesson 1.2 - Pedal to the Metal

In this lesson you will make your driving simulator come alive. First you will write your very first lines of code in C#, changing the vehicle's position and allowing it to move forward. Next you will add physics components to your objects, allowing them to collide with one another. Lastly, you will learn how to duplicate objects in the hierarchy and position them along the road.

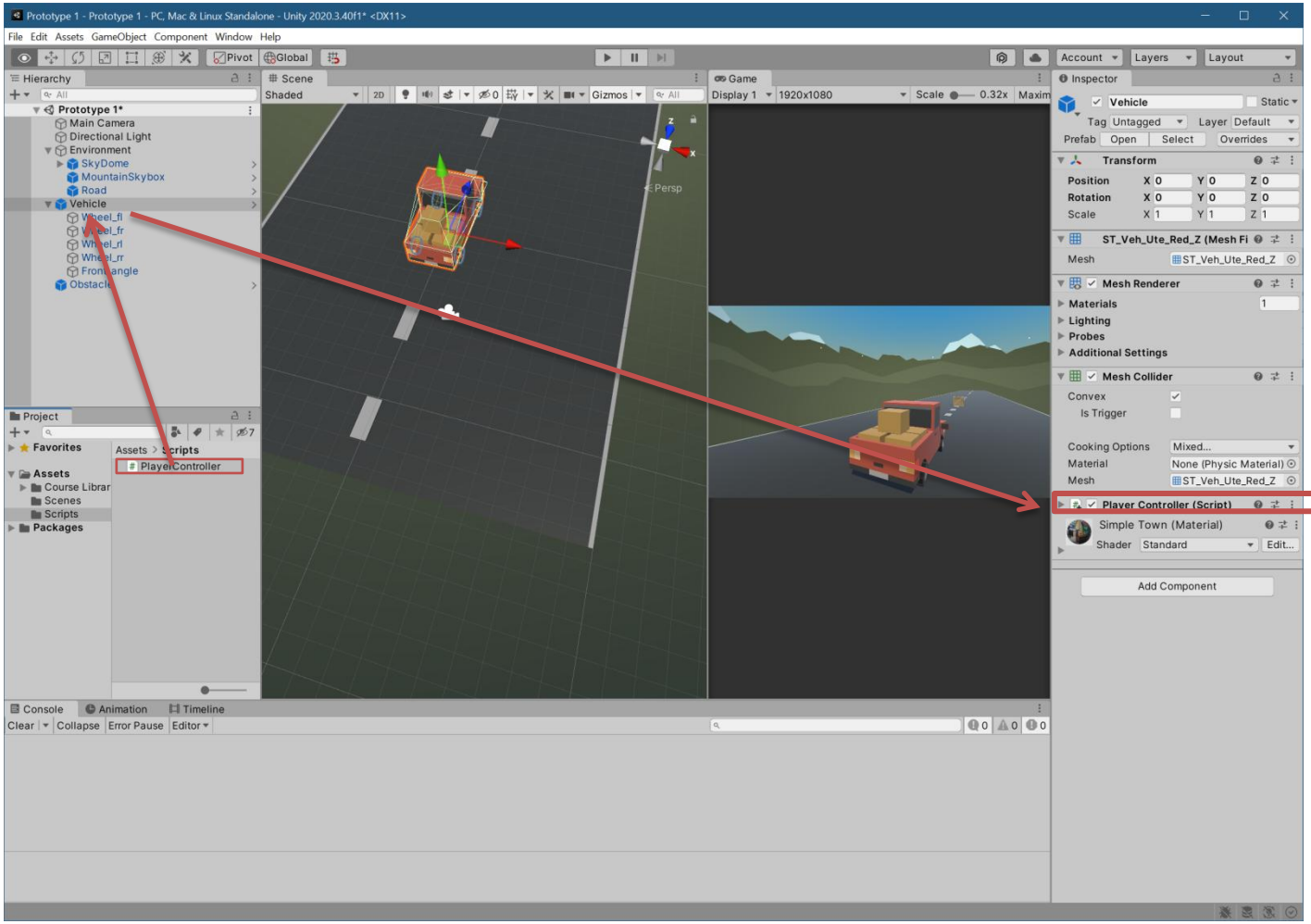
1.Create and apply your first script

1. In the Project window, *Right-click* > *Create* ***> Folder*** named "Scripts"

2. In the "Scripts" folder, *Right-click* > *Create* > ***C# Script*** named "PlayerController"

3. **Drag** the new script onto the **Vehicle object**

4. **Click** on the Vehicle object to make sure it was added as a **Component** in the Inspector


Prototype 1 - Prototype 1 - PC, Mac & Linux Standalone - Unity 2020.3.40f1* <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Prototype 1*
Main Camera
Directional Light
Environment
SkyDome
MountainSkybox
Road
Vehicle
Obstacle
Scene
Shaded
Persp
Game
Display 1
1920x1080
Scale 0.32x
Inspector
Vehicle
Static
Tag Untagged
Layer Default
Prefab Open Select Overrides
Transform
Position
Rotation
Scale
ST_Veh_Ute_Red_Z (Mesh Fi
Mesh
ST_Veh_Ute_Red_Z
Mesh Renderer
Materials
Lighting
Probes
Additional Settings
Mesh Collider
Convex
Is Trigger
Cooking Options
Mixed...
Material
None (Physic Material)
Mesh
ST_Veh_Ute_Red_Z
Player Controller (Script)
Simple Town (Material)
Shader
Standard
Edit...
Add Component
Project
Favorites
Assets
Course Librar
Scenes
Scripts
Packages
Assets > Scripts
PlayerController
Console
Animation
Timeline
Clear
Collapse
Error Pause
Editor

2.Add a comment in the Update() method

1. **Double-click** on the script to open it in **Visual Studio**

2. In the ***Update()*** method, add a comment that you will: ***// Move the vehicle forward***

```
void Update()
{
  // Move the vehicle forward
}
```

3.Give the vehicle a forward motion

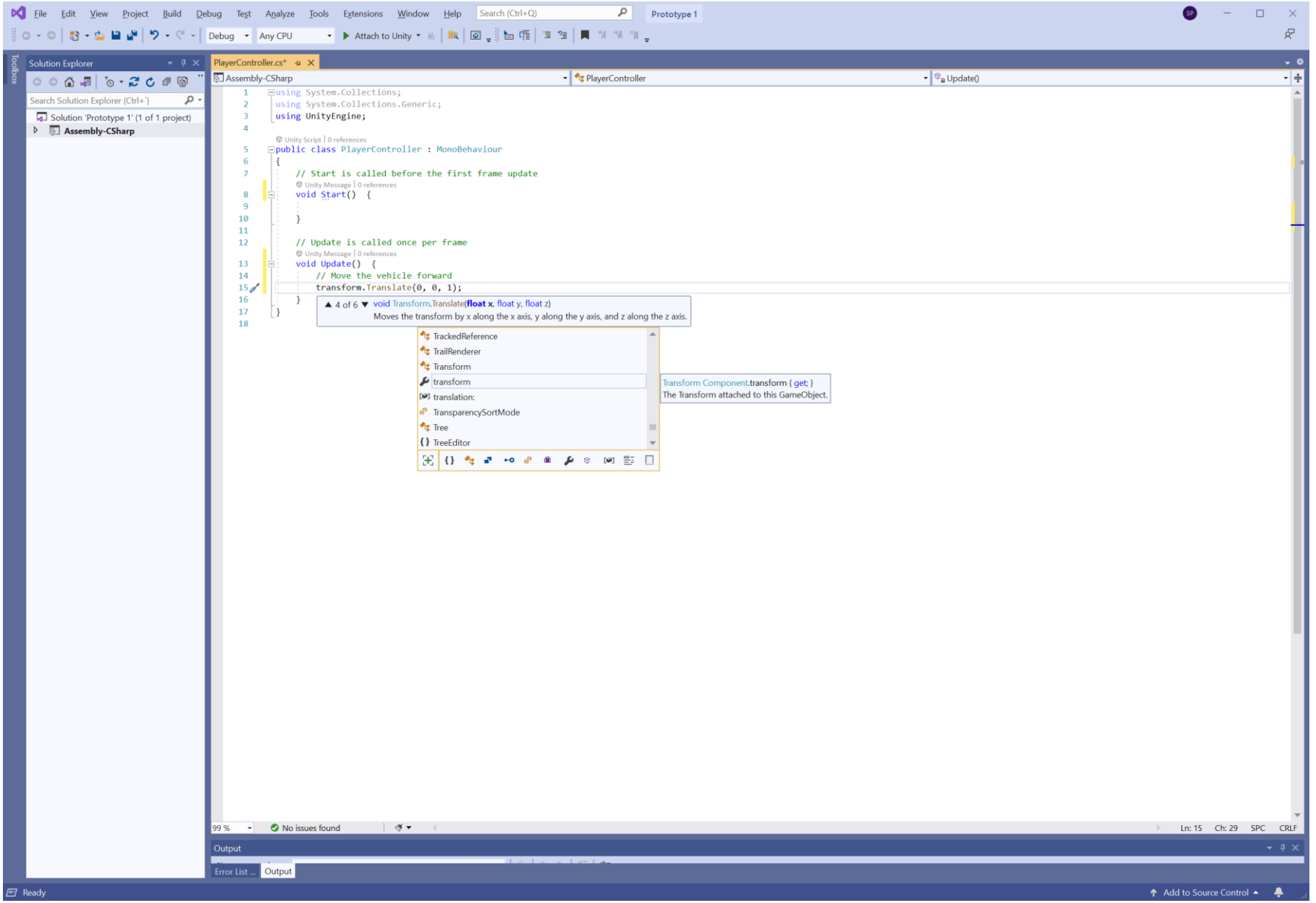
1. Under your new comment, type ***transform.tr***, then select **Translate** from the autocomplete menu

2. Type **(**, add **0, 0, 1** between the parentheses, and complete the line with a semicolon (**;**)

3. Press **Ctrl/Cmd + S** to save your script, then run your game to test it

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(0, 0, 1);
}
```

Είναι το transform component του GameObject που έχει το Script στα components του


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Prototype 1
Debug
Any CPU
Attach to Unity
Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+')
Solution 'Prototype 1' (1 of 1 project)
Assembly-CSharp
PlayerController.cs*
Assembly-CSharp
PlayerController
Update()
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Unity Script | 0 references
public class PlayerController : MonoBehaviour
{
// Start is called before the first frame update
Unity Message | 0 references
void Start() {
}
// Update is called once per frame
Unity Message | 0 references
void Update() {
// Move the vehicle forward
transform.Translate(0, 0, 1);
}
}
▲ 4 of 6 ▼ void Transform.Translate(float x, float y, float z)
Moves the transform by x along the x axis, y along the y axis, and z along the z axis.
TrackedReference
TrailRenderer
Transform
transform
translation:
TransparencySortMode
Tree
TreeEditor
Transform Component.transform { get; }
The Transform attached to this GameObject.
99 %
No issues found
Ln: 15 Ch: 29 SPC CRLF
Output
Error List ...
Output
Ready
Add to Source Control

4.Use a Vector3 to move forward

Delete the 0, 0, 1 you typed and use auto-complete to **replace it** with ***Vector3.forward***

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(0, 0, 1 Vector3.forward);
}
```

5.Customize the vehicle's speed

Right now, the speed of the vehicle is out of control! We need to change the code in order to adjust this.

Add **** Time.deltaTime*** and run your game
Add **** 20*** and run your game

Ένας συντελεστής ταχύτητας του οχήματος.

```
void Update()
{
  // Move the vehicle forward
  transform.Translate(Vector3.forward * Time.deltaTime * 20);
}
```

Το Time.deltaTime είναι ο χρόνος που πέρασε από την προηγούμενη εκτέλεση του Update (ουσιαστικά ο χρόνος που χρειάστηκε το frame για να σχεδιαστεί). Με αυτό τον τρόπο κάνουμε πιο ομαλή την κίνηση αφού προσαρμόζουμε την απόσταση που θα μετακινήσουμε στο χρόνο που πέρασε από την προηγούμενη κίνηση (δηλαδή στο frame-rate με το οποίο τρέχει η σκηνή μας).

6.Add RigidBody components to objects

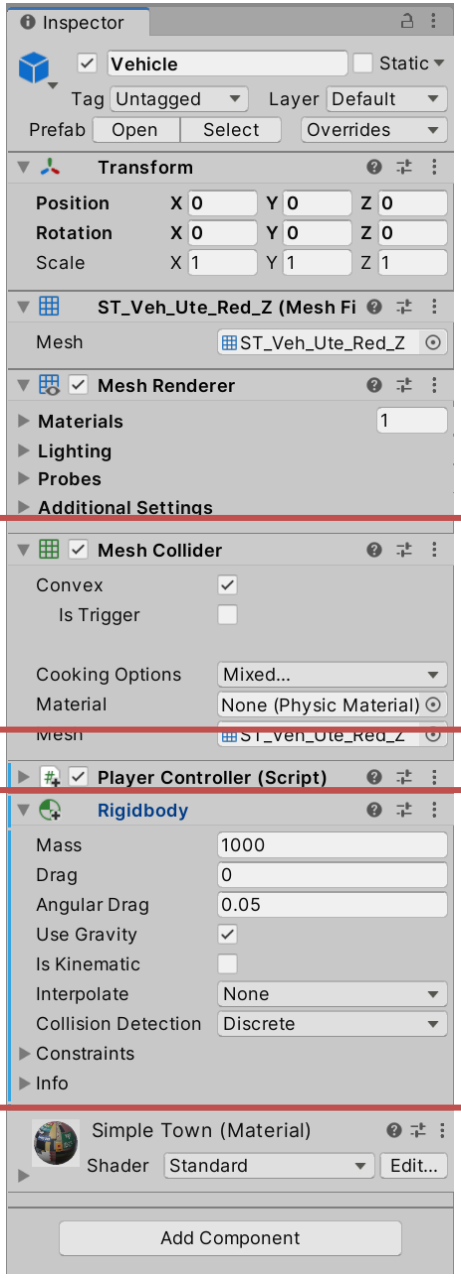
Right now, the vehicle goes right through the box! If we want it to be more realistic, we need to add physics.

1. Select the **Vehicle**, then in the hierarchy click **Add Component** and select **RigidBody**
2. Select the **Obstacle**, then in the hierarchy click **Add Component** and select **RigidBody**
3. In the RigidBody component properties, increase the **mass** of vehicle and obstacle to be about what they would be in **kilograms** and test again

7.Duplicate and position the obstacles

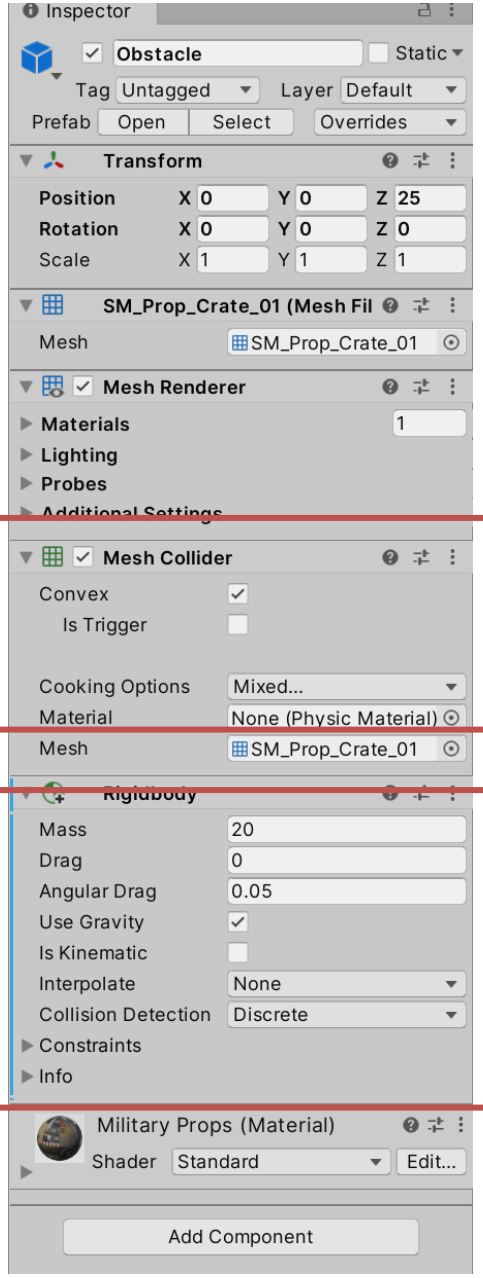
1. Click and drag your obstacle to the **bottom of the list** in the hierarchy
2. Press **Ctrl/Cmd+D** to duplicate the obstacle and move it down the **Z axis**
3. Repeat this a few more times to create more obstacles
4. After making a few duplicates, select one in the hierarchy and **hold ctrl + click** to select multiple obstacles, then **duplicate** those

Προσθέτουμε RigidBody component και στο όχημα και στο κιβώτιο . Βάζουμε μάζα 1000 (κιλά) και 20 (κιλά) αντίστοιχα. Ενεργοποιούμε το use gravity.



Τα MeshColliders επιτρέπουν την ακριβή ανίχνευση σύγκρουσης στο σχήμα των αντικειμένων

Τα Rigidbody components προσδίδουν κανόνες φυσικής στα αντικείμενα (physics).



Lesson 1.3 - High Speed Chase

Summary

Overview:

Keep your eyes on the road! In this lesson you will code a new C# script for your camera, which will allow it to follow the vehicle down the road and give the player a proper view of the scene. In order to do this, you'll have to use a very important concept in programming: variables.

Project Outcome:

The camera will follow the vehicle down the road through the scene, allowing the player to see where it's going.

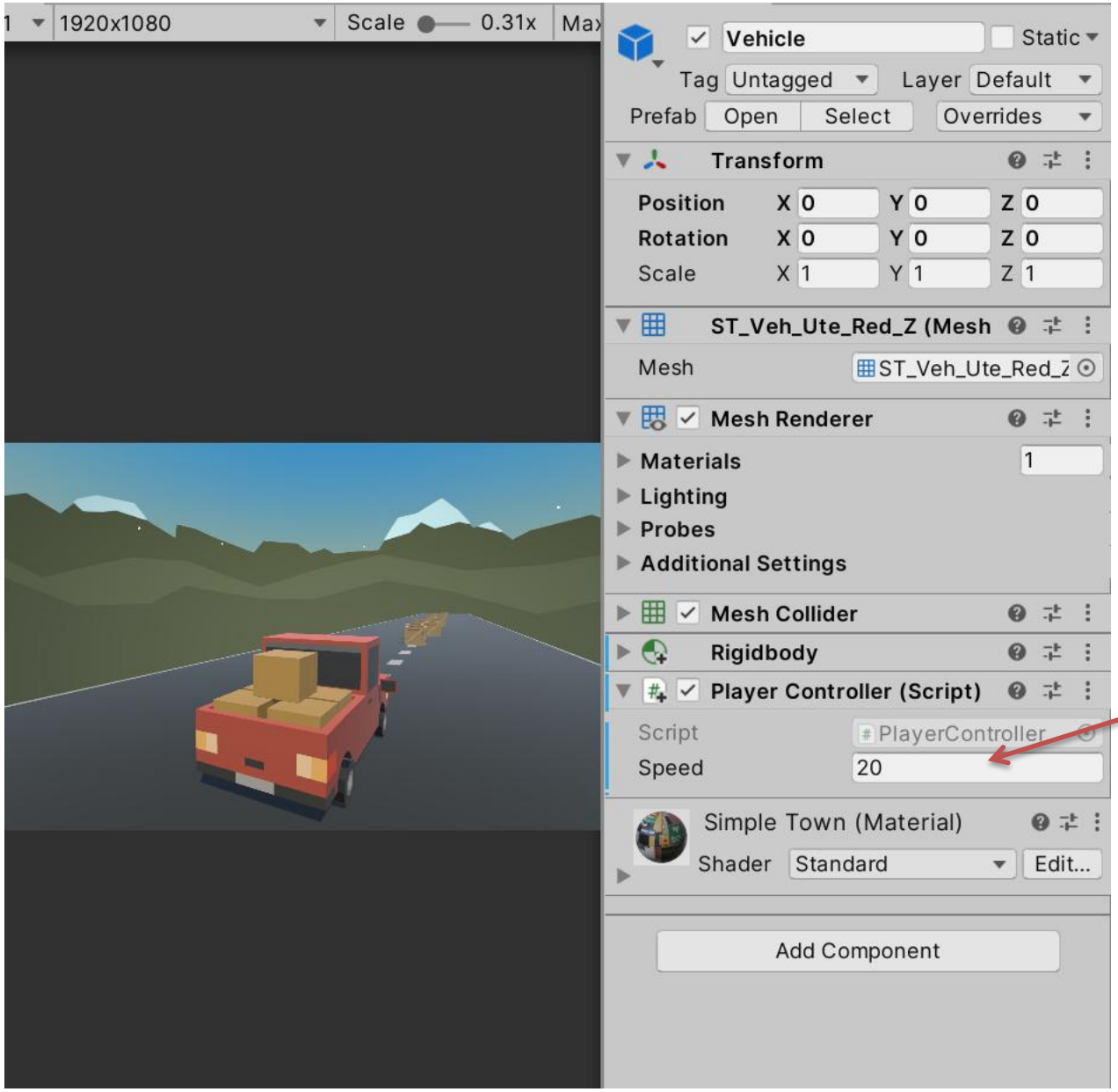
1.Add a speed variable for your vehicle

We need an easier way to change the vehicle's speed and allow it to be accessed from the inspector. In order to do so what we need is something called a variable.

1. In PlayerController.cs, add ***public float speed = 5.0f;*** at the top of the **class**
2. Replace the **speed value** in the Translate method with the **speed variable**, then test
3. **Save** the script, then edit the speed value in the **inspector** to get the speed you want

Η public μεταβλητή speed θα εμφανίζεται στον editor και θα μπορούμε να την τροποποιούμε από εκεί

```
public float speed = 20;

void Update()
{
  transform.Translate(Vector3.forward * Time.deltaTime * 20 speed);
}
```



Η public μεταβλητή speed θα εμφανίζεται στον editor και θα μπορούμε να την τροποποιούμε από εκεί

2.Create a new script for the camera

The camera is currently stuck in one position. If we want it to follow the player, we have to make a new script for the camera.

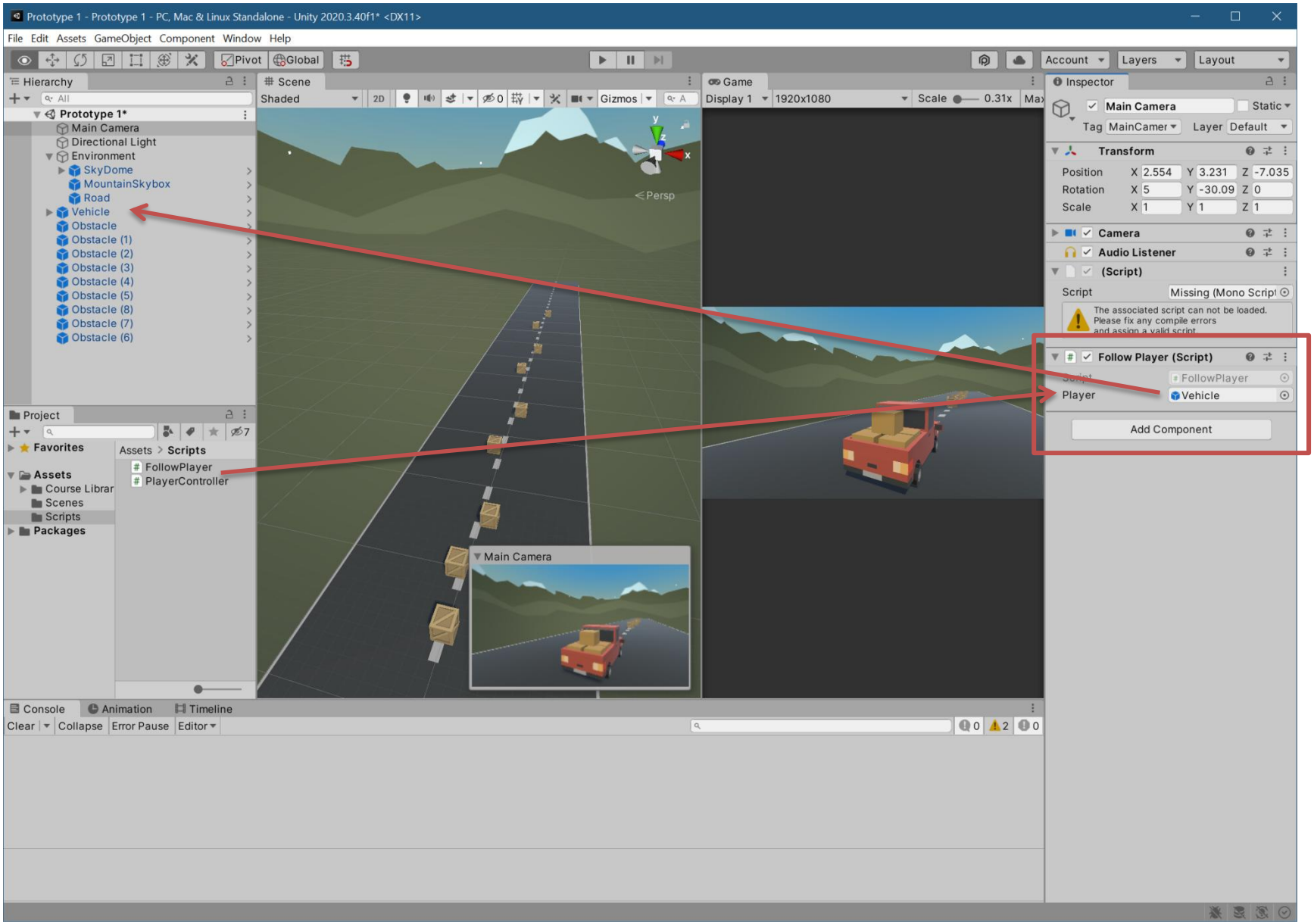
1. Create a new **C# script** called FollowPlayer and attach it to the **camera**
2. Add ***public GameObject player;*** to the top of the script
3. Select the **Main Camera, then**, **drag** the player object onto the **empty player variable** in the Inspector
4. In ***Update()***, assign the camera's position to the player's position, then test

Η public μεταβλητή player θα μα επιτρέψει στον editor να συνδέσουμε την κάμερα με το όχημα

```
public GameObject player;

void Update()
{
  transform.position = player.transform.position;
}
```

Βάζουμε την κάμερα στην ίδια θέση με το όχημα

3.Add an offset to the camera position


Prototype 1 - Prototype 1 - PC, Mac & Linux Standalone - Unity 2020.3.40f1* <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Prototype 1*
Main Camera
Directional Light
Environment
SkyDome
MountainSkybox
Road
Vehicle
Obstacle
Obstacle (1)
Obstacle (2)
Obstacle (3)
Obstacle (4)
Obstacle (5)
Obstacle (8)
Obstacle (7)
Obstacle (6)
Scene
Persp
Main Camera
Game
Display 1
1920x1080
Scale
0.31x
Inspector
Main Camera
Static
Tag MainCamer
Layer Default
Transform
Position X 2.554 Y 3.231 Z -7.035
Rotation X 5 Y -30.09 Z 0
Scale X 1 Y 1 Z 1
Camera
Audio Listener
(Script)
Script Missing (Mono Script
The associated script can not be loaded.
Please fix any compile errors
and assign a valid script.
Follow Player (Script)
FollowPlayer
Player Vehicle
Add Component
Project
Favorites
Assets
Course Librar
Scenes
Scripts
Packages
Assets > Scripts
FollowPlayer
PlayerController
Console
Animation
Timeline
Clear
Collapse
Error Pause
Editor

3.Add an offset to the camera position

In the line in the Update method add ***+ new Vector3(0, 5, -7)***, then test

```
public GameObject player;

void Update()
{
  transform.position = player.transform.position + new Vector3(0, 5, -7);
}
```

4.Make the offset into a Vector3 variable

1. At the top of **FollowPlayer.cs**, declare ***private Vector3 offset;***
2. Copy the ***new Vector3()*** code and **assign** it to that variable
3. **Replace** the original code with the ***offset*** variable
4. **Test** and **save**

```
public GameObject player;
private Vector3 offset = new Vector3(0, 5, -7);

void Update()
{
  transform.position = player.transform.position + new Vector3(0, 5, -7) offset;
}
```

5.Smooth the Camera with LateUpdate

You may have noticed that the camera is kind of jittery as the car drives down the road - let's fix that.

1. Test your prototype to notice the jittering camera as the vehicle drives.

2. In **FollowPlayer.cs**, replace ***Update()*** with ***LateUpdate().***

3. **Save** and **test** to see if the camera is less jittery.

```
void LateUpdate()
{
  transform.position = player.transform.position + offset;
}
```

Lesson 1.4 - Step into the Driver's Seat

Summary

Overview:

In this lesson, we need to hit the road and gain control of the vehicle. In order to do so, we need to detect when the player is pressing the arrow keys, then accelerate and turn the vehicle based on that input. Using new methods, Vectors, and variables, you will allow the vehicle to move forwards or backwards and turn left to right.

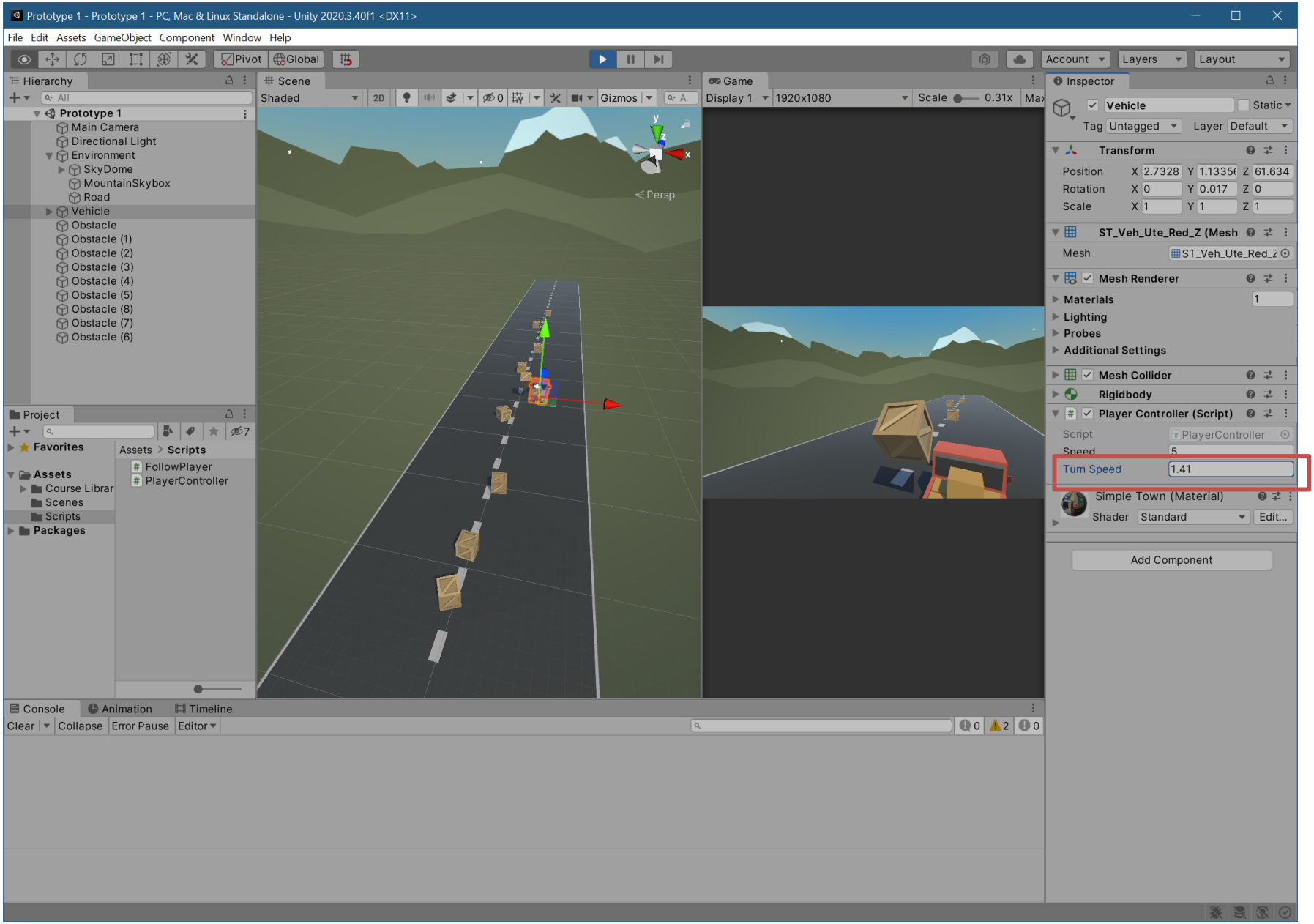
Project Outcome:

When the player presses the up/down arrows, the vehicle will move forward and backward. When the player presses the left/right arrows, the vehicle will turn.

1.Allow the vehicle to move left/right

1. At the top of PlayerController.cs, add a ***public float turnSpeed;*** variable
2. In ***Update()***, add ***transform.Translate(Vector3.right * Time.deltaTime * turnSpeed);***
3. Run your game and use the ***turnSpeed*** **variable slider** to move the vehicle left and right

```
public float turnSpeed;

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


Prototype 1 - Prototype 1 - PC, Mac & Linux Standalone - Unity 2020.3.40f1 <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Prototype 1
Main Camera
Directional Light
Environment
SkyDome
MountainSkybox
Road
Vehicle
Obstacle
Obstacle (1)
Obstacle (2)
Obstacle (3)
Obstacle (4)
Obstacle (5)
Obstacle (8)
Obstacle (7)
Obstacle (6)
Scene
Persp
Game
Display 1
1920x1080
Inspector
Vehicle
Transform
Position
Rotation
Scale
ST_Veh_Ute_Red_Z (Mesh
Mesh Renderer
Materials
Lighting
Probes
Additional Settings
Mesh Collider
Rigidbody
Player Controller (Script)
Script
PlayerController
Speed
Turn Speed
1.41
Simple Town (Material)
Shader
Standard
Edit...
Add Component
Project
Favorites
Assets
Course Librar
Scenes
Scripts
Packages
Assets > Scripts
FollowPlayer
PlayerController
Console
Animation
Timeline
Clear
Collapse
Error Pause
Editor

2.Base left/right movement on input

Currently, we can only control the vehicle's left and right movement in the inspector. We need to grant some power to the player and allow them to control that movement for themselves.

1. From the top menu, click **Edit** > **Project Settings**, select **Input Manager** in the left sidebar, then expand the **Axes** fold-out to explore the inputs

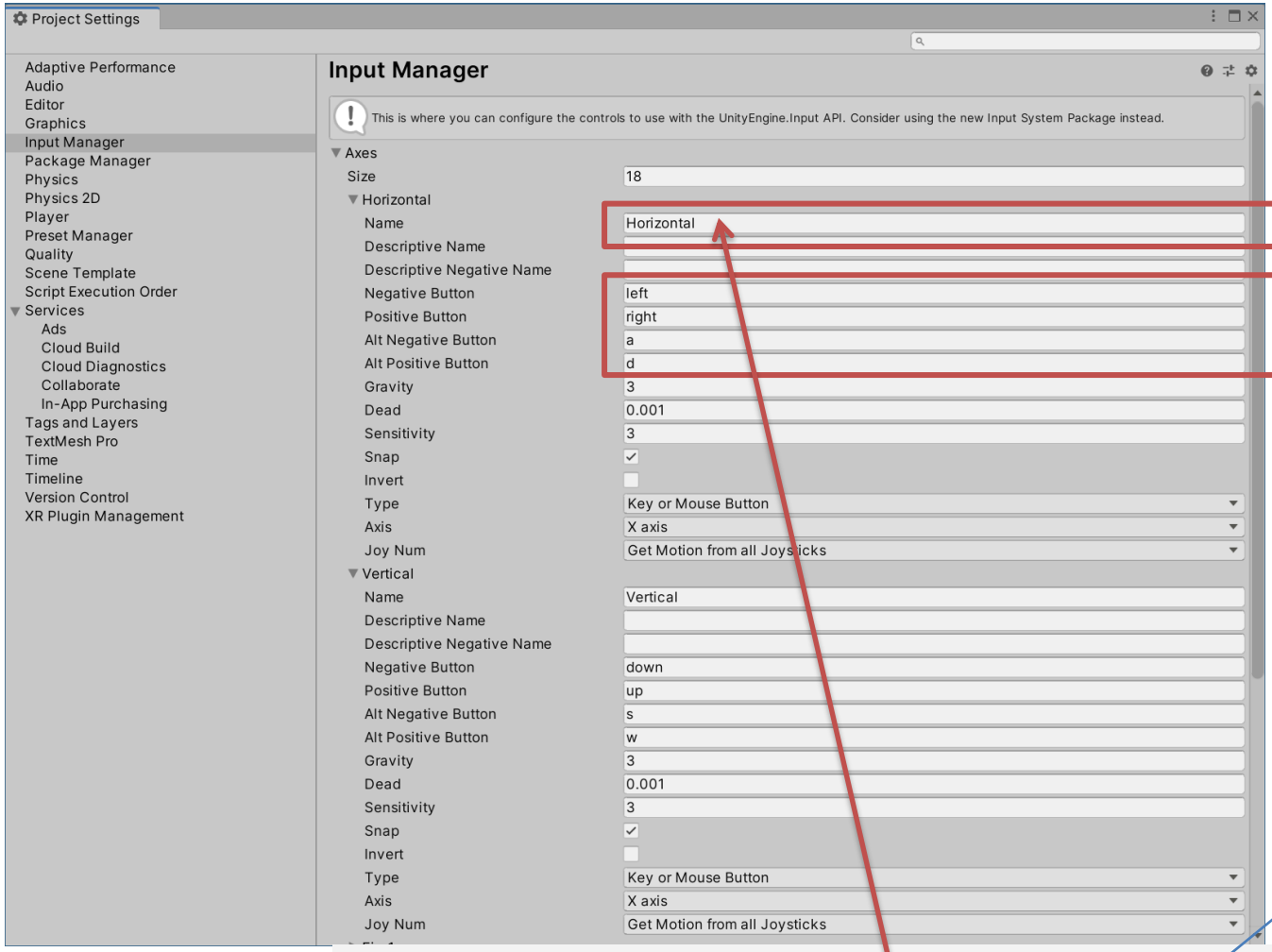
2. In **PlayerController.cs**, add a new ***public float horizontalInput*** variable

3. In ***Update***, assign ***horizontalInput = Input.GetAxis("Horizontal");***, then test to see it in inspector

4. Add the ***horizontalInput*** variable to your left/right **Translate method** to gain control of the vehicle

5. In the Inspector, edit the ***turnSpeed*** and ***speed*** variables to tweak the feel

```
public float horizontalInput;

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

  transform.Translate(Vector3.forward * Time.deltaTime * speed);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```



Διεύθυνση

Πλήκτρα + και -

Επιστρέφει τιμή από -1 έως 1
Ανάλογα με το ποιο πλήκτρο πατιέται

Public για να βλέπουμε την τιμή της στον Editor

```
public float horizontalInput;

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

  transform.Translate(Vector3.forward * Time.deltaTime * speed);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

3.Take control of the vehicle speed

We've allowed the player to control the steering wheel, but we also want them to control the gas pedal and brake

1. Declare a new public ***forwardInput*** variable
2. In ***Update***, assign ***forwardInput = Input.GetAxis("Vertical");***
3. Add the ***forwardInput*** variable to the **forward Translate method**, then test

```
public float horizontalInput;
public float forwardInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  forwardInput = Input.GetAxis("Vertical");

  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);
}
```

4.Make vehicle rotate instead of slide

There's something weird about the vehicle's movement… it's slides left to right instead of turning. Let's allow the vehicle to turn like a real car!

1. In ***Update***, call ***transform.Rotate(Vector3.up, horizontalInput)***, then test
2. **Delete** the line of code that **translates Right**, then test
3. Add **** turnSpeed * Time.deltaTime***, then test

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

  transform.Translate(Vector3.forward * Time.deltaTime * speed * forwardInput);
  transform.Rotate(Vector3.up, turnSpeed * horizontalInput * Time.deltaTime);
  ~~transform.Translate(Vector3.right * Time.deltaTime * turnSpeed * horizontalInput);~~
}
```

Unit 2
Summary

In this Unit, you will program a top-down game with the objective of throwing food to hungry animals - who are stampeding towards you - before they can run past you. In order to do this, you will become much more familiar with some of the most important programming and Unity concepts, including if-then statements, random value generation, arrays, collision detection, prefabs, and instantiation. In completing this Unit, you will learn how to program a basic game with the ability to launch projectiles and maneuver the player to keep the game alive..

Το Tutorial στο Unity-Learn

Υλικό που θα χρειαστεί στο Project
Prototype 2 - Starter Files.zip

Το κατεβάζετε και το αποσυμπιέζετε σε έναν φάκελο του υπολογιστή σας

Lesson 2.1 - Player Positioning

Summary

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.

Materials

Prototype 2 - Starter Files.zip

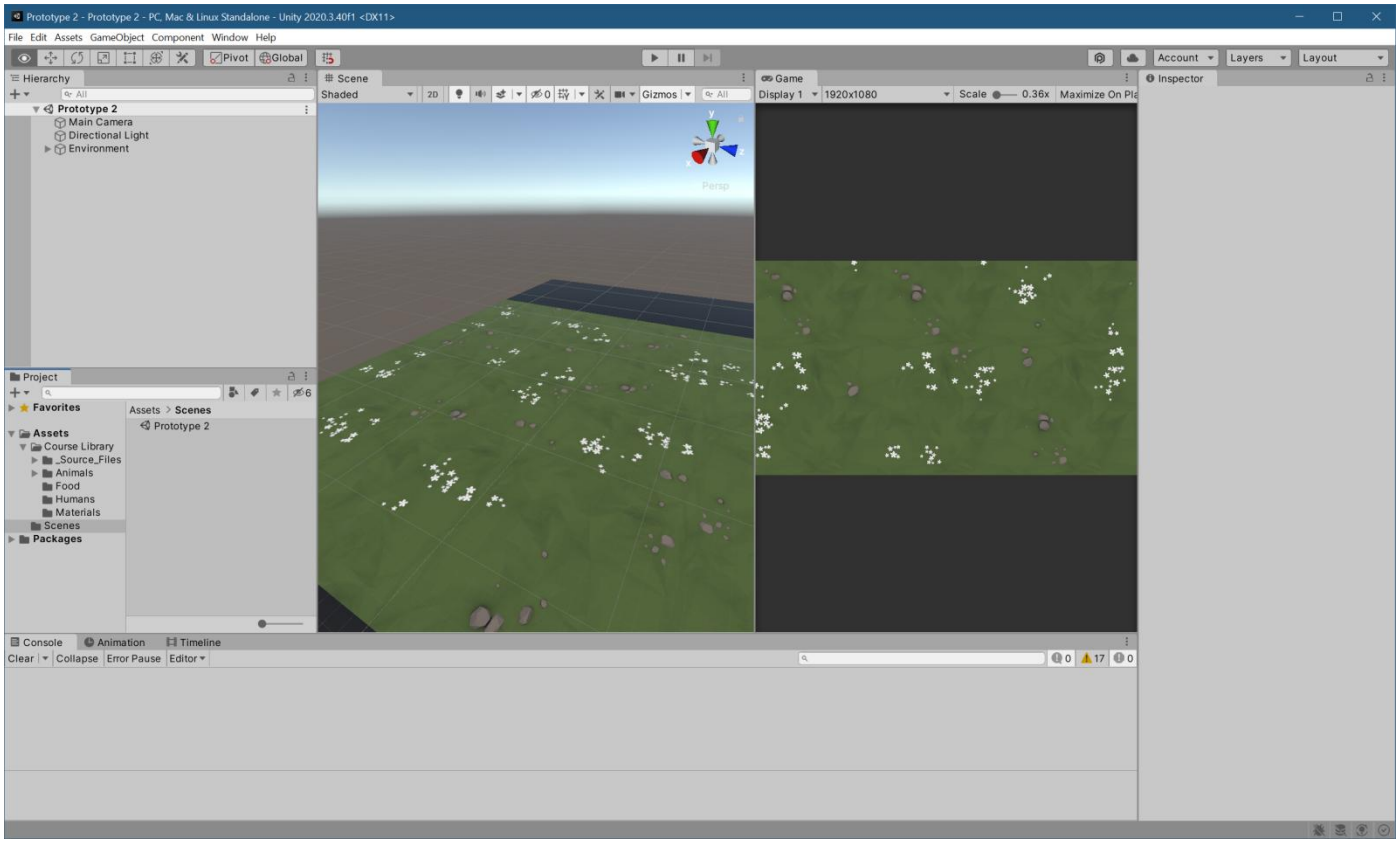
1.Create a new Project for Prototype 2

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**

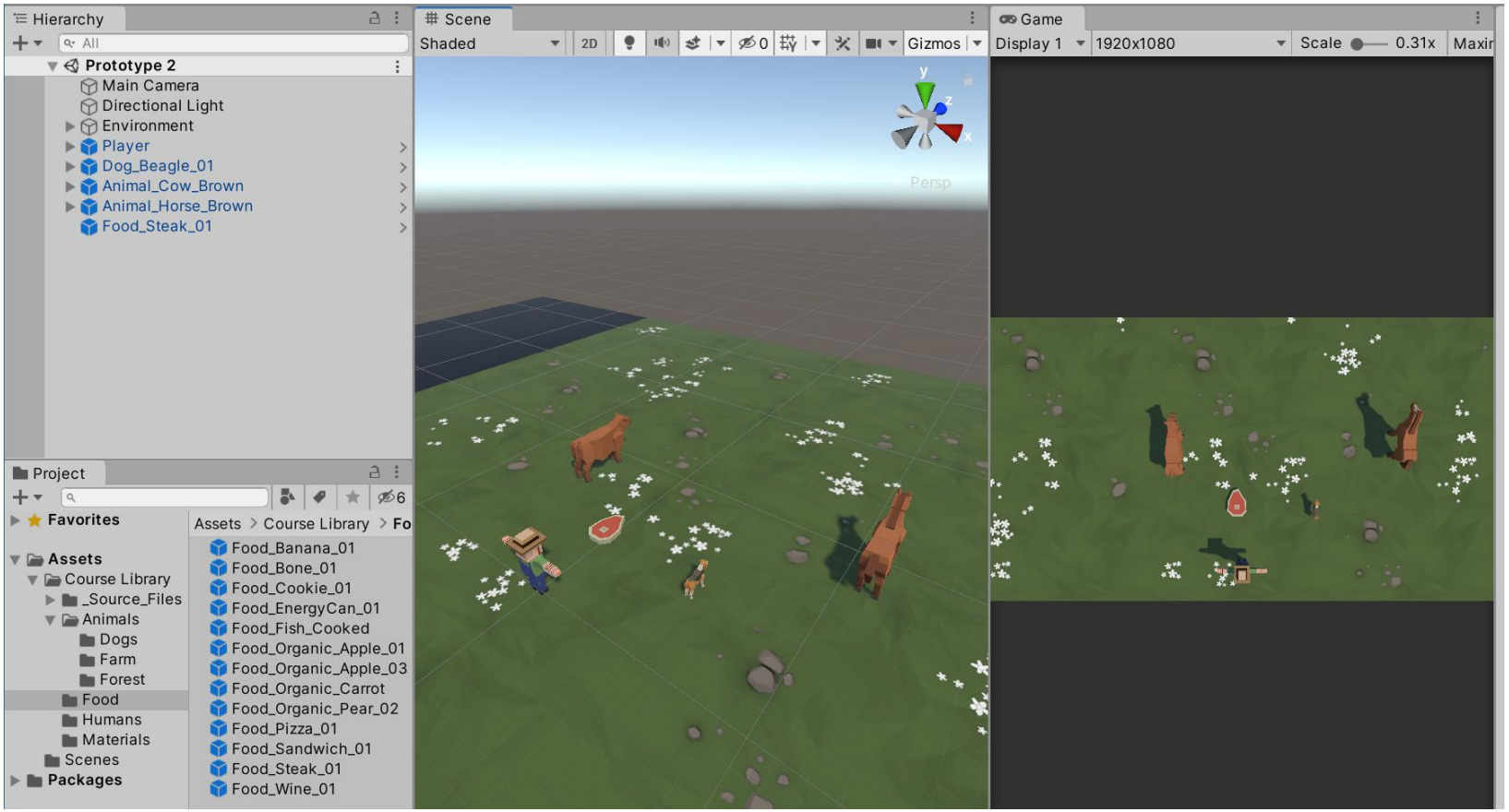
3. From the Project window, open the **Prototype 2** scene and **delete** the SampleScene



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 character "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



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.

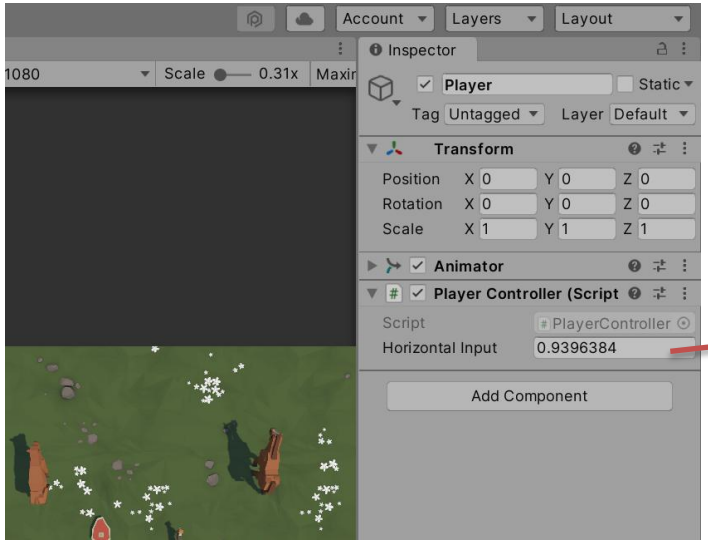
1. In your **Assets** folder, create a "Scripts" folder, and a "PlayerController" script inside

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

```
public float horizontalInput;

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



Πατώντας τα πλήκτρα για δεξιά-αριστερά πρέπει να μεταβάλλεται η τιμή του Horizontal Input

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***

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

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

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**

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

Αν ξεπεραστεί το όριο (-10) , ακινητοποιούμε το X στο -10 και διατηρούμε ίδιες τις άλλες δύο διαστάσεις

6.Clean up your code and variables

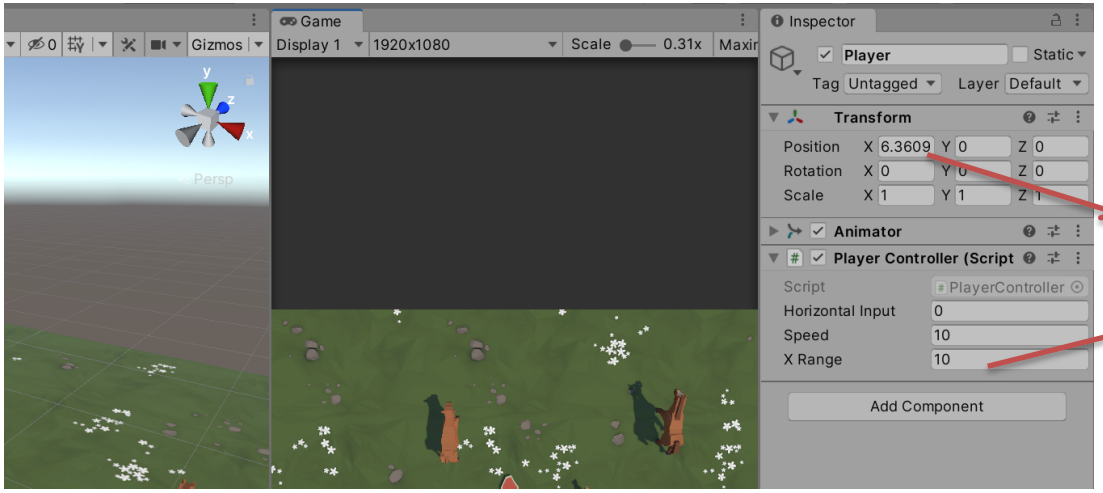
We need to make this work on the right side, too, then clean up our code.

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

```
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);
  }
}
```



Παρατηρήστε πως κινώντας τον παίκτη δεξιά – αριστερά, το X δεν ξεπερνάει ποτέ τα όρια -xRange ←-→ xRange

Lesson 2.2 - Food Flight

Summary

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.

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.

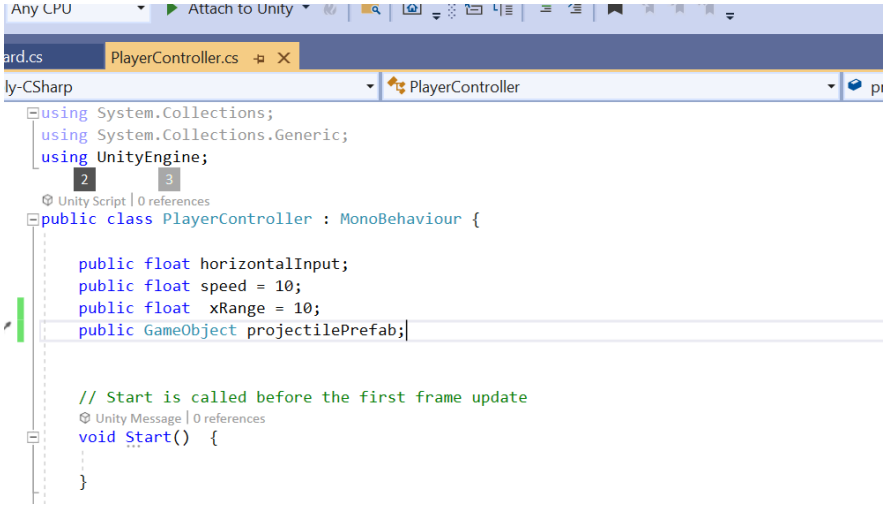
1. Create a new "MoveForward" script, **attach** it to the food object, then open it
2. Declare a new ***public float speed*** variable***;***
3. In ***Update()***, add ***transform.Translate(Vector3.forward * Time.deltaTime * speed);***, then **save**
4. In the **Inspector**, set the projectile's **speed**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

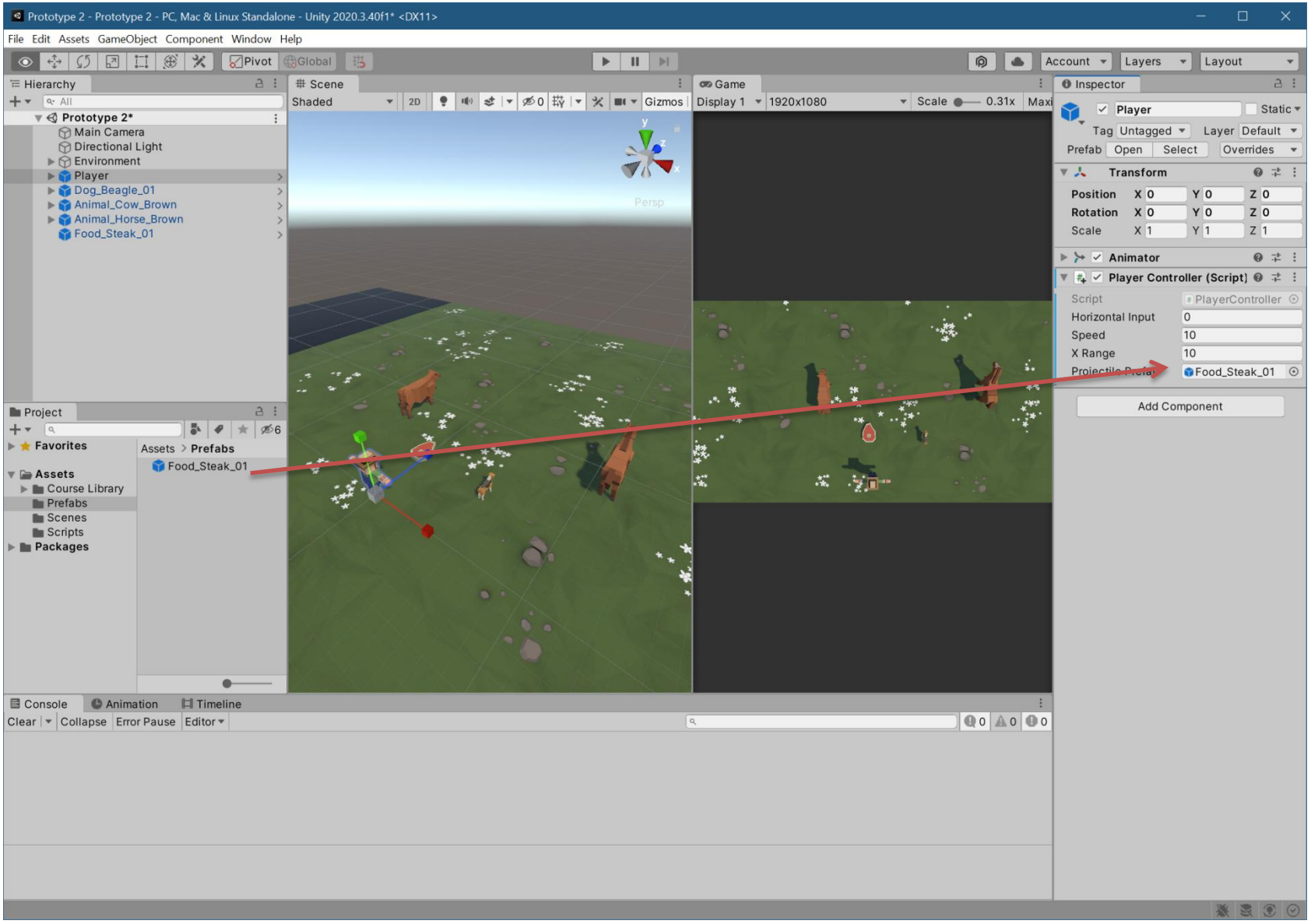
```
public float speed = 40.0f;

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

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




Prototype 2 - Prototype 2 - PC, Mac & Linux Standalone - Unity 2020.3.40f1* <DX11>
File Edit Assets GameObject Component Window Help
Pivot
Global
Account
Layers
Layout
Hierarchy
Prototype 2*
Main Camera
Directional Light
Environment
Player
Dog_Beagle_01
Animal_Cow_Brown
Animal_Horse_Brown
Food_Steak_01
Scene
Shaded
2D
Gizmos
Persp
Game
Display 1
1920x1080
Scale
0.31x
Inspector
Player
Static
Tag Untagged
Layer Default
Prefab Open Select Overrides
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Animator
Player Controller (Script)
Script PlayerController
Horizontal Input 0
Speed 10
X Range 10
Food_Steak_01
Add Component
Project
Favorites
Assets
Course Library
Prefabs
Scenes
Scripts
Packages
Assets > Prefabs
Food_Steak_01
Console
Animation
Timeline
Clear
Collapse
Error Pause
Editor

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)) {

1. Inside the if-statement, add a comment saying that you should ***// Launch a projectile from the player***

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

4.Launch projectile on spacebar press

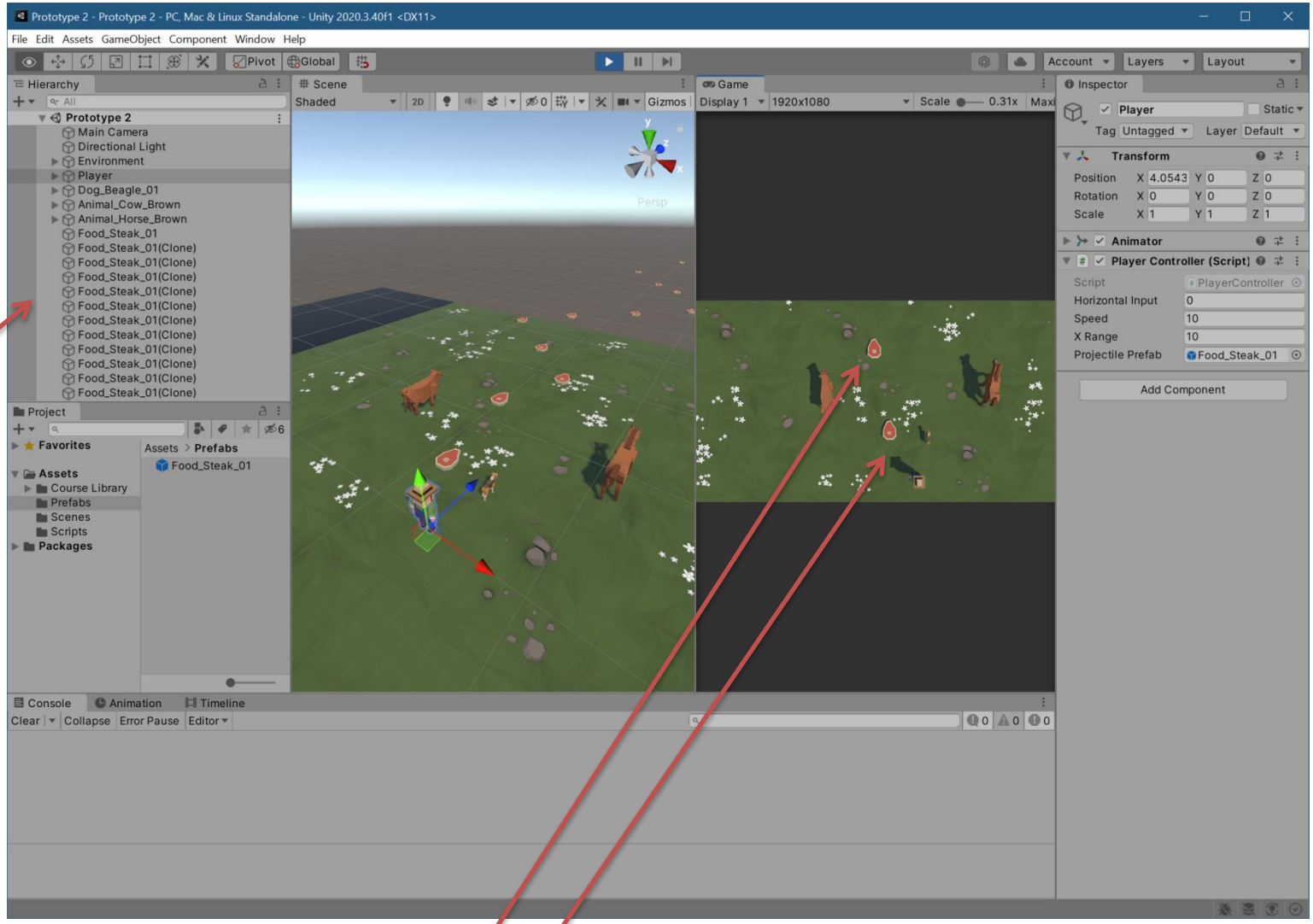
We've created the code that tests if the player presses spacebar, but now we actually need spawn a projectile when that happens

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

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

Κάθε φορά που θα πατιέται το SPACE θα δημιουργείται στη σκηνή ένα αντικείμενο τύπου projectilePrefab , στη θέση του παίκτη (transform.positiojn) και με την περιστροφή που έχει το projectilePrefab (projectilePrefab.transform.rotation)

Projectiles που προστέθηκαν με πάτημα του SPACE



Με το που δημιουργείται ένα νέο projectile , επειδή έχει ενσωματωμένο τον κώδικα από το script ‘moveForward.cs’ θα κινείται με σταθερή ταχύτητα προς τα εμπρός

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.

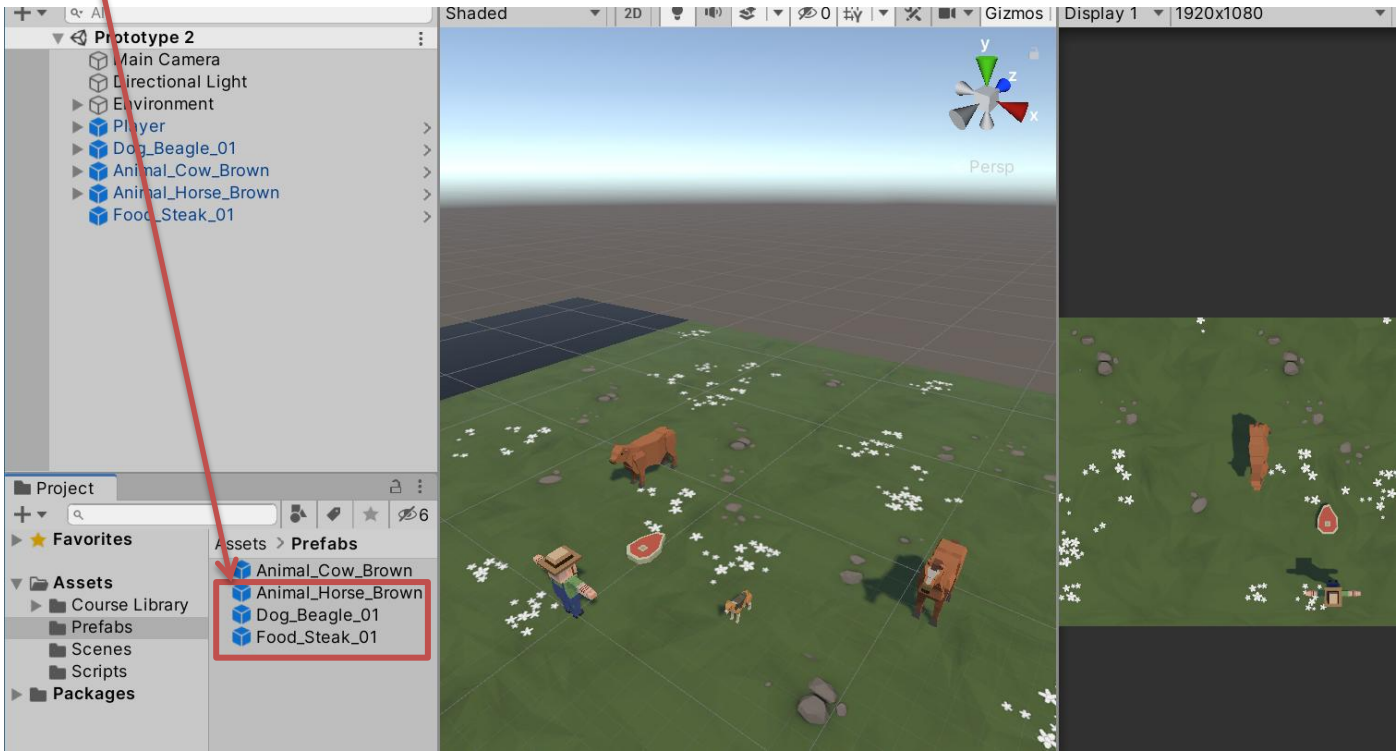
1. **Rotate** all animals on the Y axis by **180 degrees** to face down

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



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**

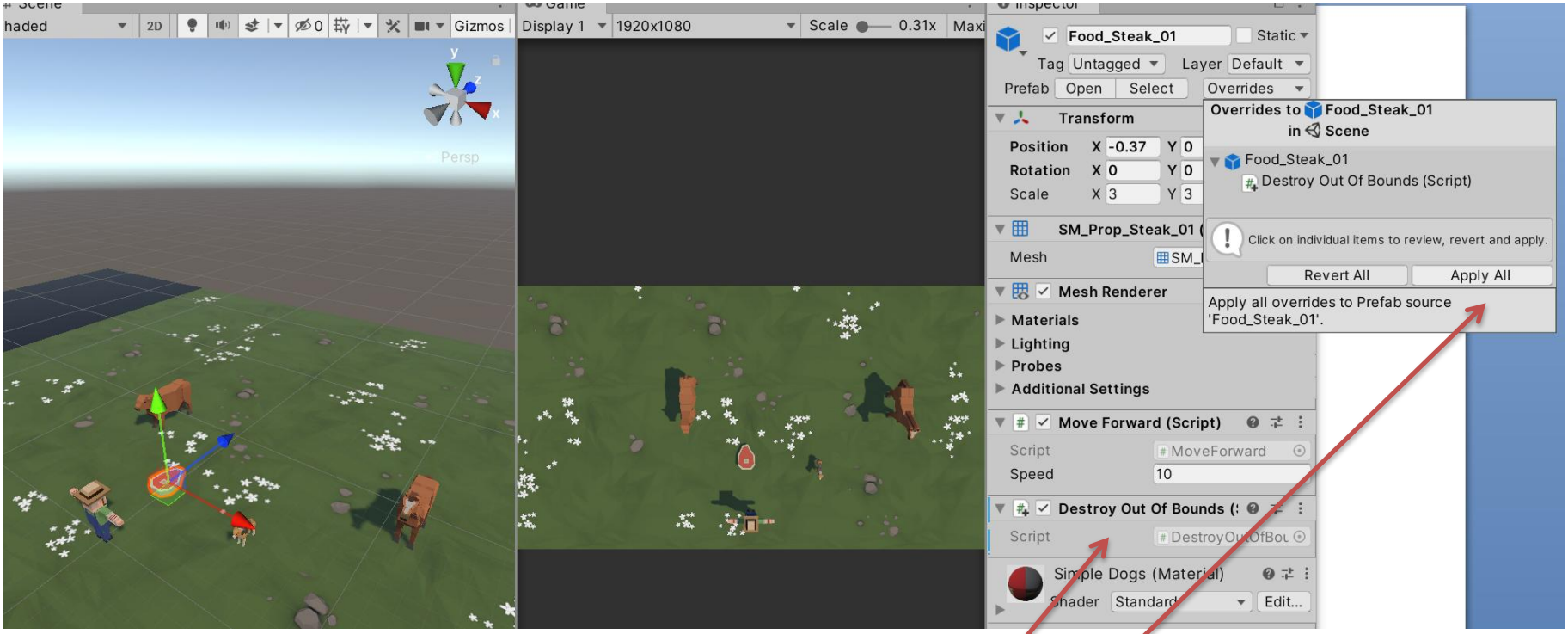
2. Add a new ***private float topBound*** variable and initialize it ***= 30;***

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

```
private float topBound = 30;

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

Όταν το Z του αντικειμένου που φέρει αυτό τον κώδικα ξεπεράσει το topBound , καταστρέφεται.



Έχοντας προσθέσει στη μπριζόλα το DestroyOutOfBounds script, πατάμε Overides – Apply All για να **ενσωματωθεί στο prefab**

7.Destroy animals off screen

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 **else-if statement** to check if objects are beneath ***lowerBound***: ***else if (transform.position.z > topBound)***

2. **Apply** the script to all of the animals, then **Override** the prefabs

```
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);
  } }
```

Κάνουμε προσθήκες στο DestroyOutOfBounds script ώστε να εξυπηρετεί και την καταστροφή των prefabs ζώων όταν βγουν από το ορατό πεδίο. Έτσι με το ίδιο script χειριζόμαστε και τα projectiles και τα ζώα

Lesson 2.3 - Random Animal Stampede

Summary

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.

1.Create a spawn manager

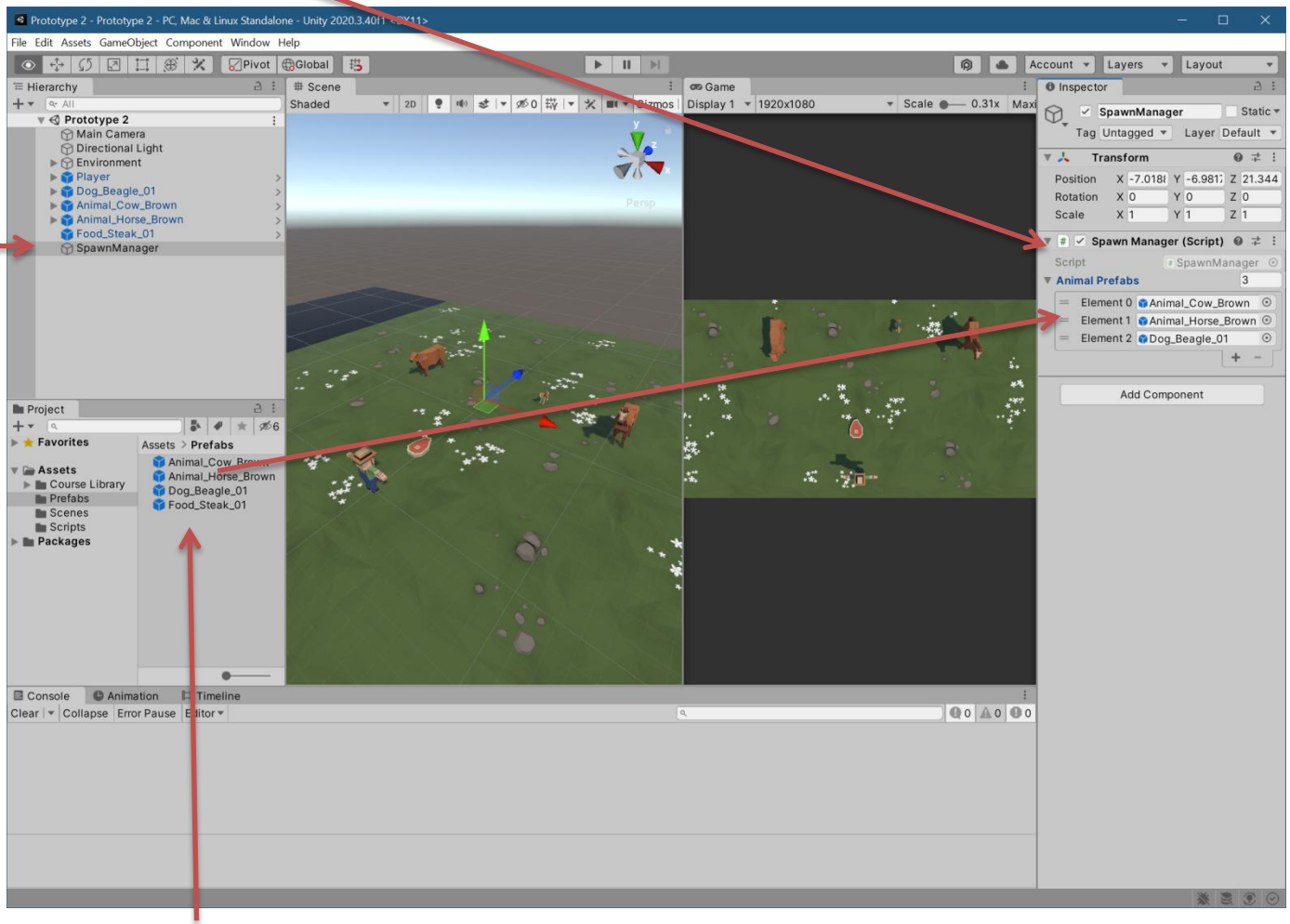
If we are going to be doing all of this complex spawning of objects, we should have a dedicated script to manage the process, as well as an object to attach it to.

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.

Έχουμε τοποθετήσει το Script SpawnManager σαν component του αντικειμένου SpawnManager

Έχουμε βάλει σαν μέγεθος του array 3 και έχουμε «σύρει» τα 3 prefabs που έχουμε φτιάξει για τα ζώα

Ένα κενό Object για να τρέχει το Script SpawnManager



ΠΡΟΣΟΧΗ: Στο array του Scripr SpawnManager πρέπει να τοποθετήσουμε τα Prefabs και όχι τα instances που έχουμε στη σκηνή

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

```
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);
  }
}
```

Κάθε φορά που πατιέται το πλήκτρο S , θα δημιουργείται ένα instance του ζώου animalPrefabs[animalIndex] , στη θέση 0,0,20 και με περιστροφή αυτή που έχει το αντίστοιχο prefab

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**

```
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); }}
```

Επιλέγουμε τυχαία μια θέση του πίνακα (και επομένως ποιο ζώο θα γίνει spawn) από 0 μέχρι το μέγεθος του πίνακα (το Max είναι exclusive)

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***

```
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); }}
```

Επιλέγουμε τυχαία μια θέση X (μεταξύ μιας περιοχής) από την οποία θα γίνει spawn το επόμενο ζώο

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"

Lesson 2.4 - Collision Decisions

Summary

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.

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 start by writing 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

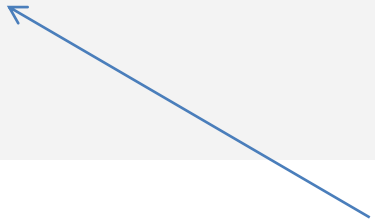
```
void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    SpawnRandomAnimal();
    int animalIndex … (Cut and Pasted Below) }}

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);
 }
```

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

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

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

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



Η function SpawnRandomAnimal θα καλείται κάθε «spawnInterval" sec ενώ θα υπάρξει μια καθυστέρηση έναρξης 2 sec.

Προσοχή! !!!

Η InvokeRepeating ΠΡΕΠΕΙ ΝΑ ΜΠΕΙ ΣΤΟ Start ώστε να κληθεί μόνο μια φορά !

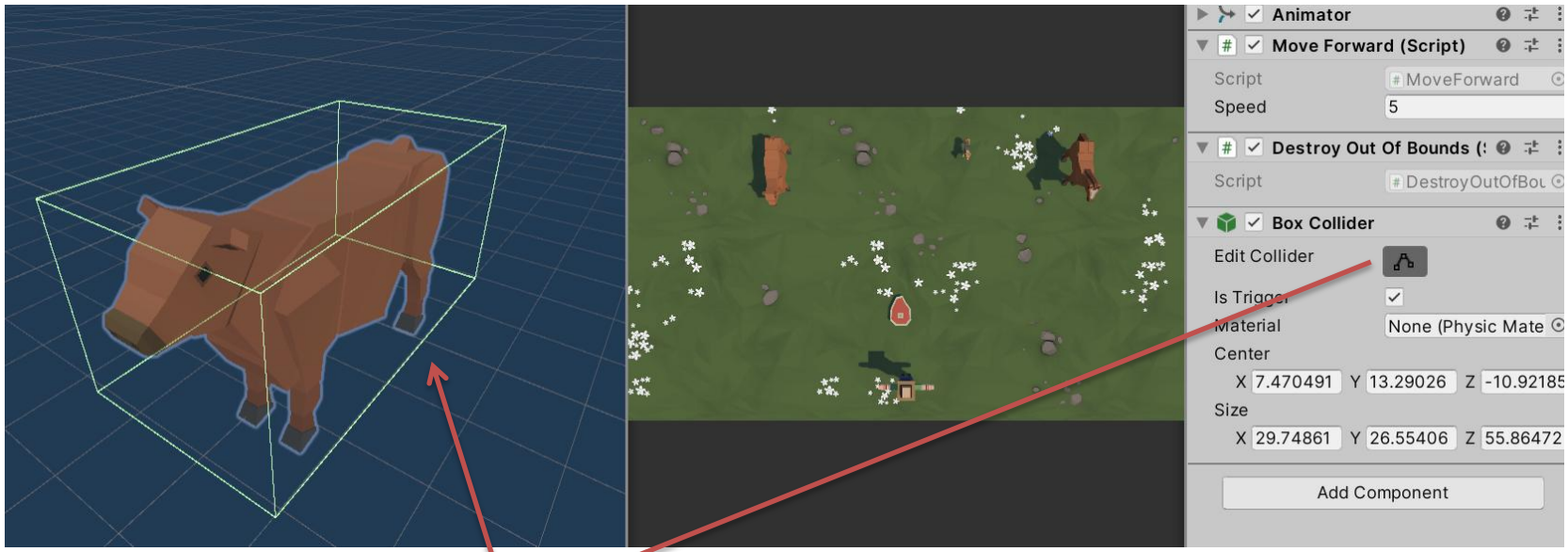
Αν τη βάλουμε στο Update τότε σε κάθε frame θα ξεκινάει μια νέα InvokeRepeating !!!!!

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"

Στο prefab του projectile (Μπριζόλα) ΠΡΕΠΕΙ ΝΑ ΒΑΛΟΥΜΕ ΕΠΙΠΛΕΟΝ RigidBody!



Ρυθμίζουμε το μέγεθος του BoxCollider ώστε να περικλείει το κάθε ζώο

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);***

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

Βάζοντας αυτό το script στα prefabs των ζώων και επειδή πλέον τα ζώα αλλά και η μπριζόλα έχουν boxColliders και επιπλέον η μπριζόλα έχει και RigidBody , κάθε φορά που το ζώο συγκρούεται με μια μπριζόλα θα καλείται η OnTriggerEnter . Εκεί καταστρέφουμε τόσο το ίδιο το ζώο (gameObject) όσο και το αντικείμενο με το οποίο συγκρούστηκε (other.gameObject)

5.Trigger a “Game Over” message

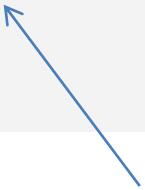
The player can defend their field against animals for as long as they wish, but we should let them know when they’ve lost with a “Game Over” message if any animals get past the player.

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)*

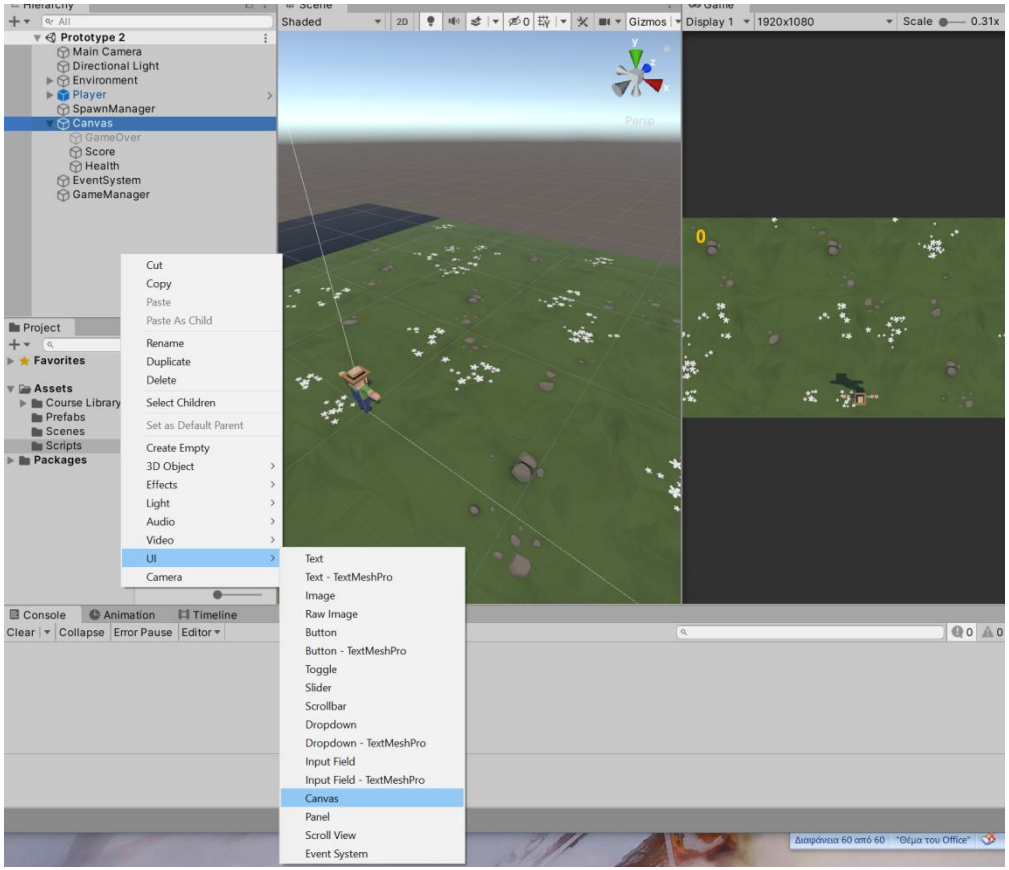
```
void Update() {
  if (transform.position.z > topBound)
  {
    Destroy(gameObject);
  } else if (transform.position.z < lowerBound)
  {
    Debug.Log("Game Over!");
    Destroy(gameObject);
  }
}
```



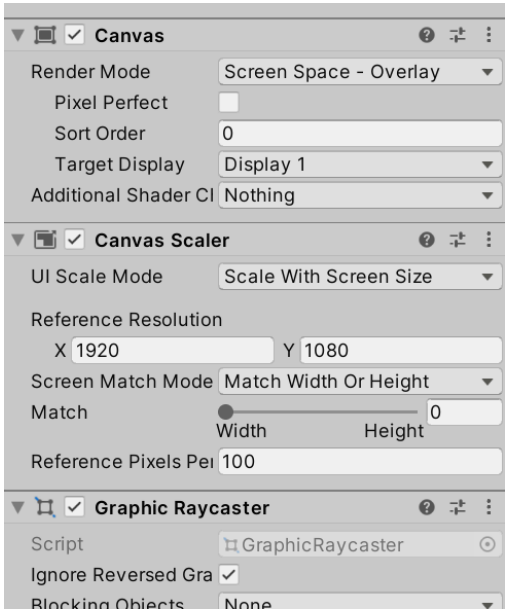
Το μήνυμα θα εμφανιστεί στην Console του Unity

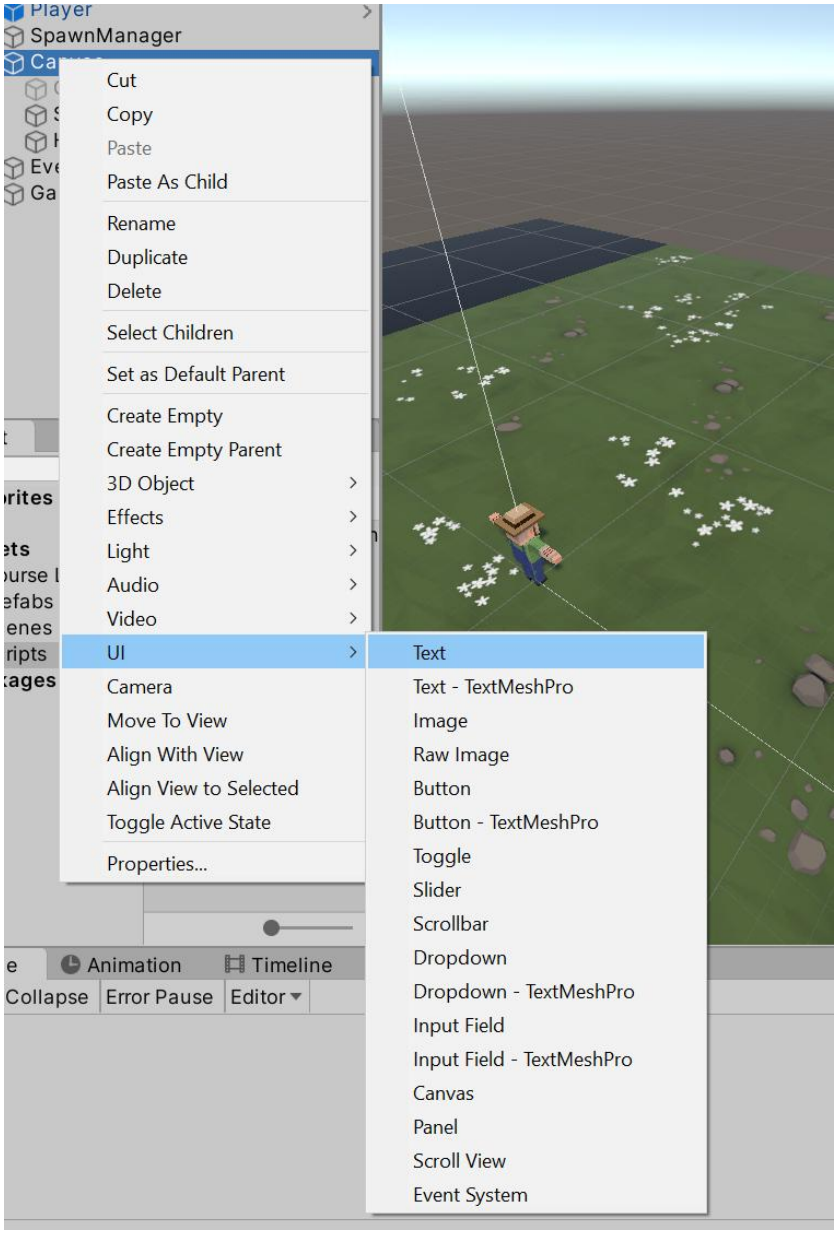
Επέκταση : Score, Υγεία και Game Over

Με τις κατάλληλες προσθήκες θα προσθέσουμε έναν μετρητή βαθμών ώστε κάθε πετυχημένη βολή να δίνει πόντους επιτυχίας στον παίκτη. Επίσης με κάθε ζώο που ξεφεύγει θα αφαιρούνται πόντοι υγείας από τον παίκτη. Όταν η υγεία μηδενιστεί θα εμφανίζεται το Game Over και ο παίκτης θα εξαφανίζεται.

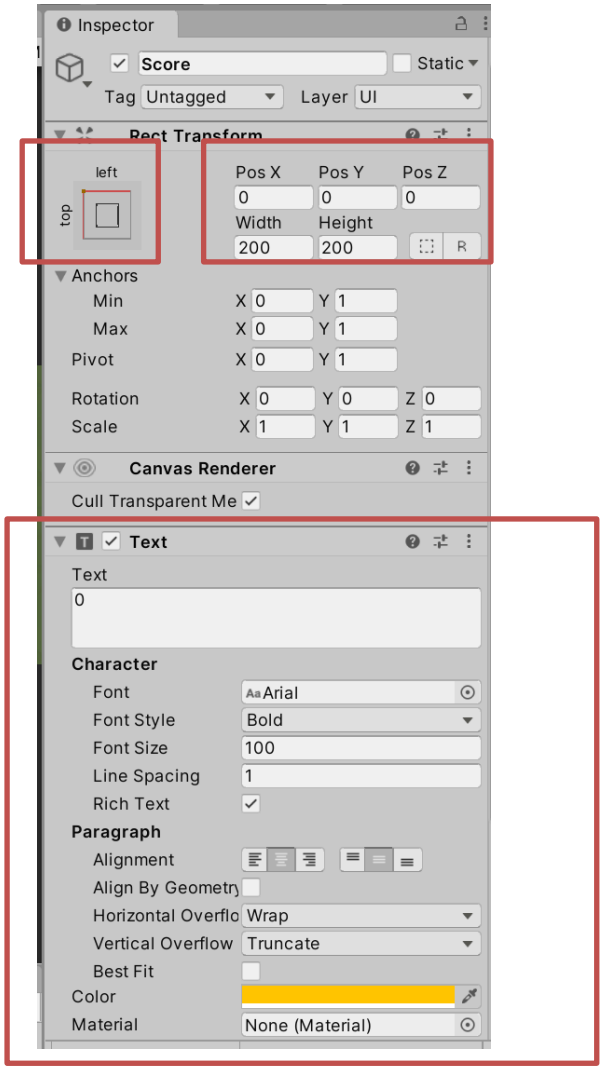


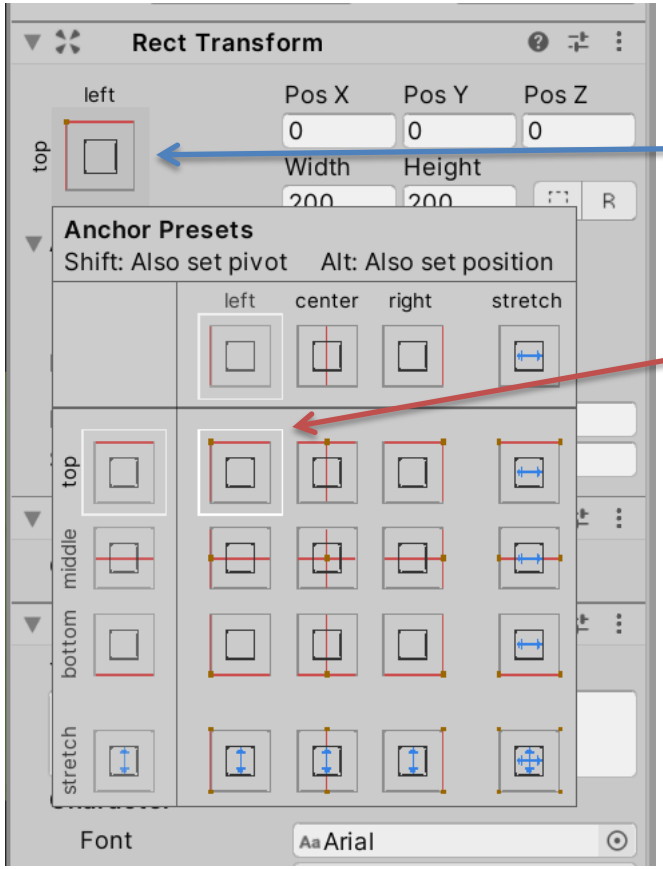
1. Δημιουργούμε στη σκηνή μας ένα αντικείμενο Canvas που είναι η βάση για να φτιάξουμε User Interface
2. Ρυθμίζουμε τα Canvas και Canvas Scaler όπως παρακάτω



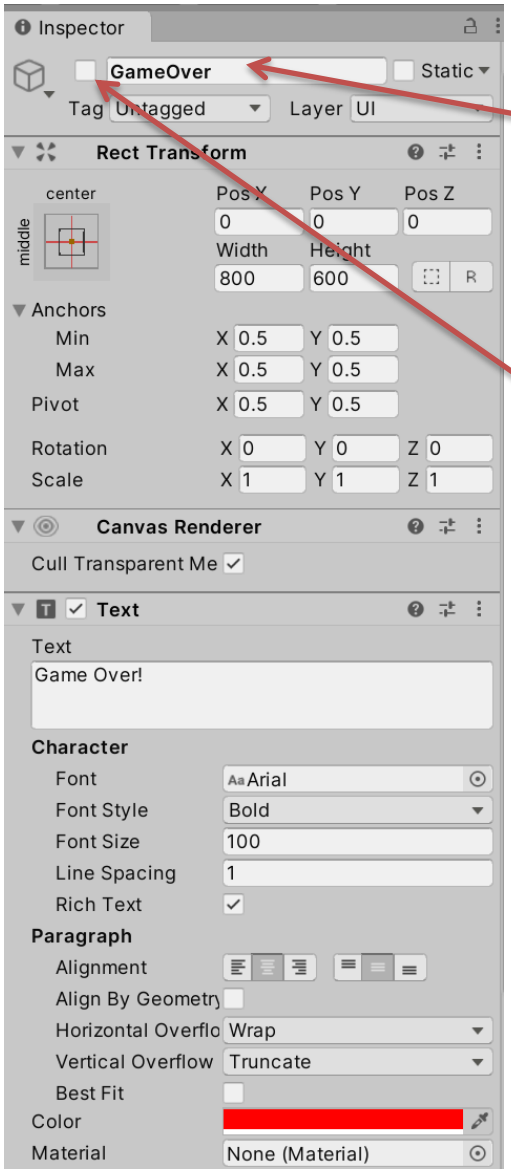


3. Με δεξί click στο αντικείμενο Canvas δημιουργούμε στο εσωτερικό του ένα αντικείμενο τύπου Text και το ονομάζουμε Score
4. Ρυθμίζουμε τις παραμέτρους του Score όπως παρακάτω



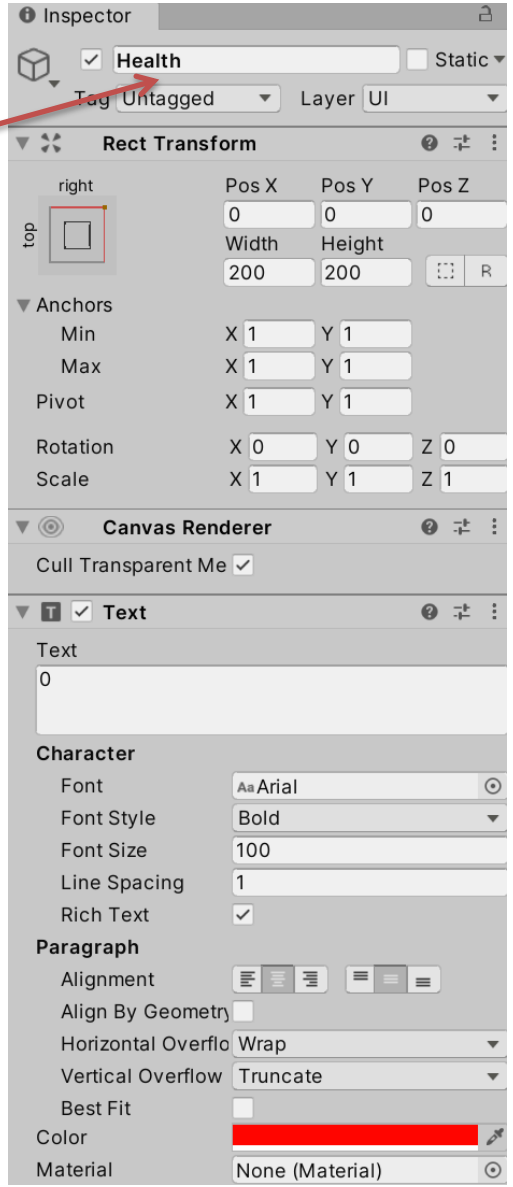


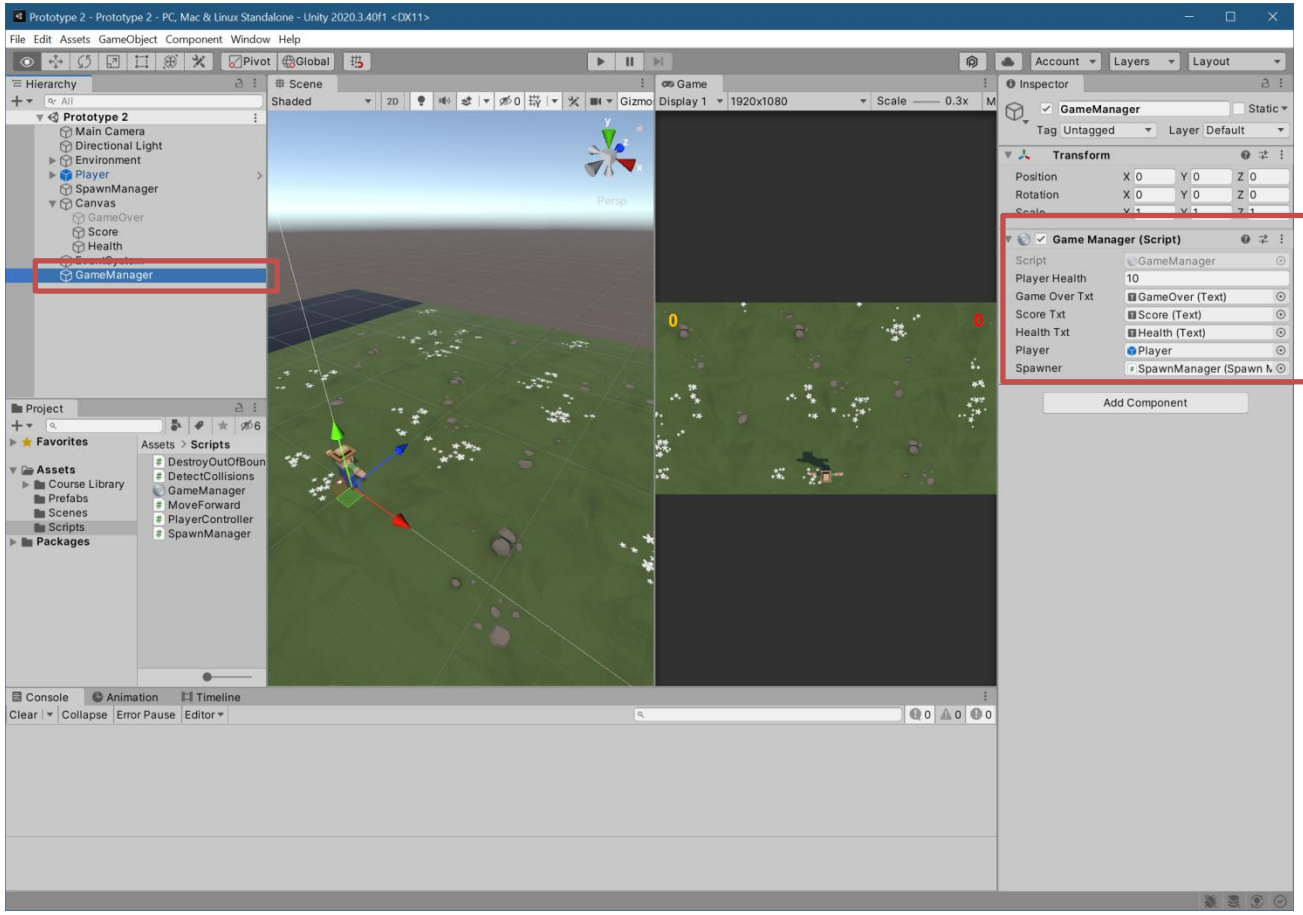
Λεπτομέρεια:
Για να ρυθμίσουμε τη θέση του Score μέσα στο Canvas πατάμε στο σημείο που φαίνεται στην εικόνα δεξιά **και με κρατημένο το πλήκτρο Shift και το Alt επιλέγουμε το πάνω αριστερά εικονίδιο** ώστε η pivot και η θέση του Score να μετριέται από πάνω αριστερά στο Canvas.



5. Με τον ίδιο τρόπο προσθέτουμε άλλα δύο Text ΜΕΣΑ στο Canvas , τα ονομάζουμε GameOver και Health και τα ρυθμίζουμε όπως φαίνεται στις εικόνας.

Προσοχή αρχικά το GaneOver το απενεργοποιούμε ώστε να είναι αόρατο και να το εμφανίσουμε όταν χρειαστεί.





6. Φτιάχνουμε στη σκηνή **ένα Empty Object και το ονομάζουμε GameManager** . Στη συνέχεια **φτιάχνουμε ένα νέο Script και το ονομάζουμε και αυτό GameManager** και το **αναθέτουμε στα Components του GameManager αντικειμένου.**

Προσοχή : Στην εικόνα το GameManager component **εμφανίζεται στην τελική του μορφή με τις public μεταβλητές που δεν τις έχουμε προσθέσει ακόμα στον κώδικα.**

Ανοίγουμε το Script GameManager.cs και γράφουμε τον κώδικα

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

Unity Script (1 asset reference) | 4 references
public class GameManager : MonoBehaviour {
    private int userScore = 0;

    public int playerHealth = 100;
    public Text gameOverTxt;
    public Text scoreTxt;
    public Text healthTxt;
    public GameObject player;
    public SpawnManager spawner;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()     {
        healthTxt.text = playerHealth.ToString();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()    {

    }

    1 reference
    public void addScore(int addScore) {
        userScore += addScore;
        scoreTxt.text = userScore.ToString();
    }

    1 reference
    public void induceDamage(int addDamage) {
        playerHealth -= addDamage;
        if (playerHealth<=0) {
            Destroy(player);
            healthTxt.gameObject.SetActive(false);
            gameOverTxt.gameObject.SetActive(true);
            spawner.stopSpawninging();
        } else {
            healthTxt.text = playerHealth.ToString();
        }
    }
}
```

Public μεταβλητή που θα κρατάει την υγεία του παίκτη. Την αρχική της τιμή τη βάζουμε 100 αλλά μπορούμε να τη ρυθμίσουμε και από τον Editor

Μεταβλητές τύπου Text που θα τις συνδέσουμε στον Editor με τα αντίστοιχα αντικείμενα Text του Canvas ώστε να μπορούμε να αλλάζουμε από τον κώδικα εδώ τις τιμές που εμφανίζουν,,

Μεταβλητή gameObject που θα τη συνδέσουμε στον Editor με το ανθρωπάκι που ελέγχει ο παίκτης ώστε να έχουμε πρόσβαση σε αυτό από τον κώδικα εδώ.

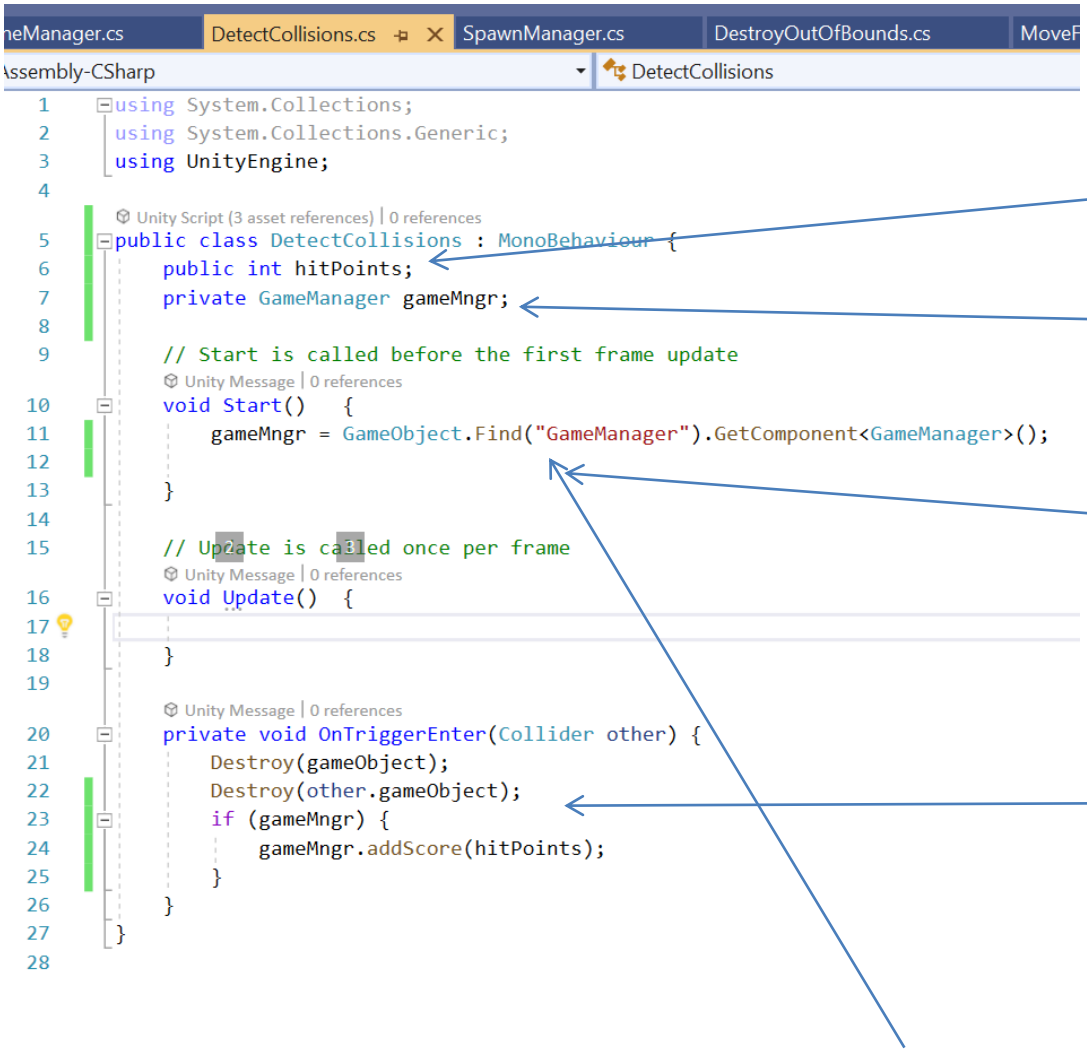
Μεταβλητή τύπου SpawnManager (εμείς έχουμε ορίσει αυτή την κλάση στον αντίστοιχο κώδικα που γράψαμε στο script SpawnManager) που θα τη συνδέσουμε στο αντίστοιχο αντικείμενο της σκηνής στον Editor ώστε να έχουμε πρόσβαση σε αυτό.

Όταν ξεκινάει η εκτέλεση εμφανίζουμε την αρχική υγεία του παίκτη

Μέθοδος που θα καλείται σε κάθε επιτυχημένη βολή για να αυξήσει το Score του παίκτη και να εμφανίσει στην οθόνη το νέο Score

Μέθοδος που θα καλείται κάθε φορά που ένα ζώο δραπετεύει και θα αφαιρεί πόντους από την υγεία του παίκτη. Όταν η υγεία μηδενιστεί θα εξαφανίζεται ο παίκτης, θα εμφανίζεται το μήνυμα Game Over και θα σταματήσει και το spawning νέων ζώων με κλήση της μεθόδου stopSpawning στο αντικείμενο Spawner.

Ανοίγουμε το Script DetectCollisions.cs και τροποποιούμε τον υπάρχοντα με τον κώδικα



Public μεταβλητή όπου θα μπορούμε να ρυθμίζουμε από τον editor πόσους πόντους δίνει κάθε ζώο.

Μεταβλητή όπου θα αποθηκεύσουμε ένα σύνδεσμο στο GameManager αντικείμενο του παιχνιδιού για να έχουμε πρόσβαση στις μεθόδους του εδώ.

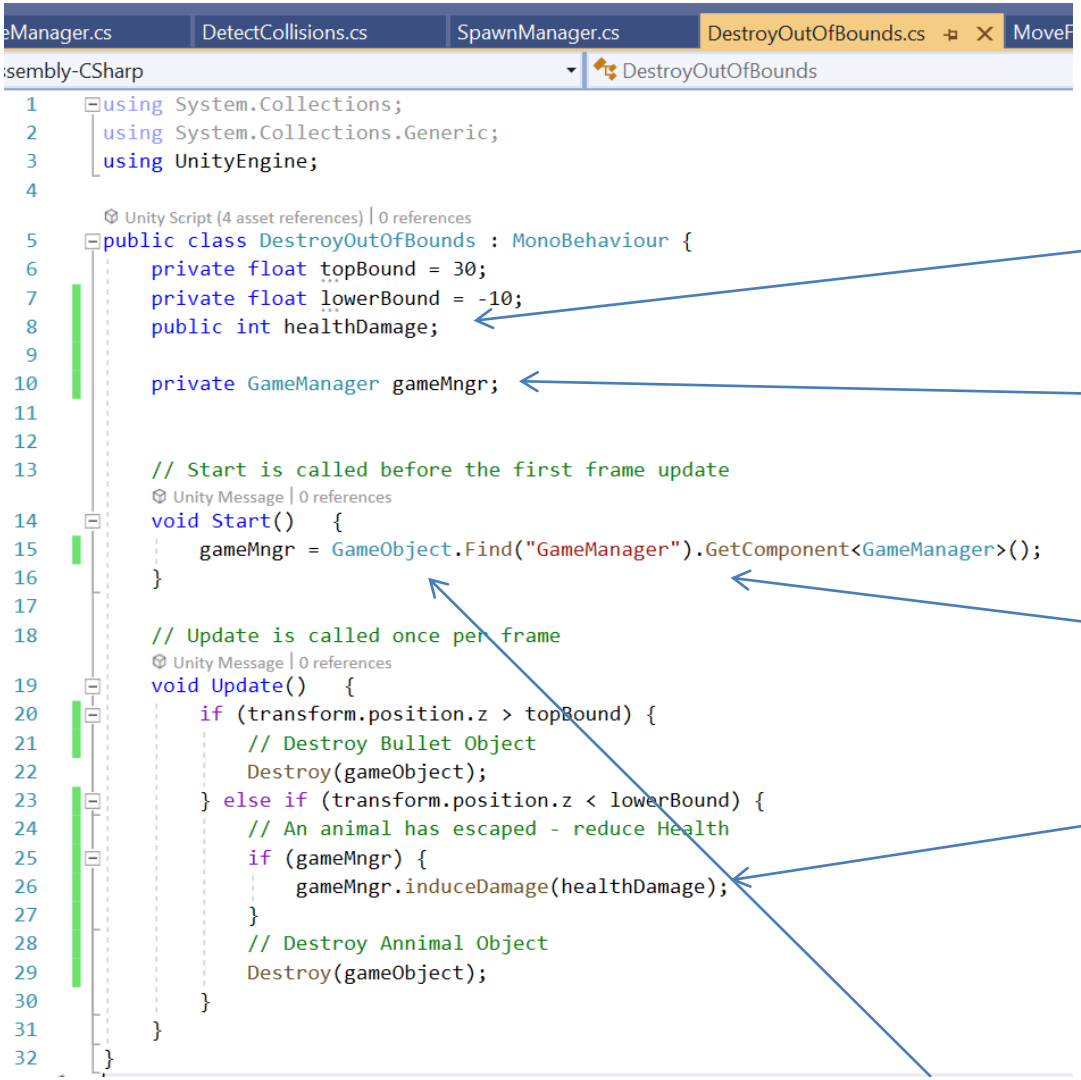
Όταν ξεκινάει η εκτέλεση βρίσκουμε το αντικείμενο GameManager στη σκηνή και κρατάμε ένα reference στον κώδικα του component GameManager που περιέχει

Κάθε φορά που ανιχνεύεται σύγκρουση καταστρέφουμε τόσο το ζώο όσο και τη μπριζόλα-βολή και προσθέτουμε στο σκορ του παίκτη τους αντίστοιχους πόντους.

ΠΡΟΣΟΧΗ ΣΗΜΑΝΤΙΚΟ: Τα ζώα δημιουργούνται δυναμικά (on the fly) κατά την εκτέλεση του παιχνιδιού. Δε μπορούμε λοιπόν από πριν να έχουμε συνδέσει το αντικείμενο GameManager της σκηνής μέσω του Editor στην αντίστοιχη μεταβλητή.

Πρέπει να το κάνουμε στο Start() μέσω της εντολής GameObject.Find που αναζητεί στη σκηνή το αντικείμενο με το όνομα “GameManager” και εφόσον το βρει , μέσω του GetComponent αποκτούμε πρόσβαση στον κώδικα GameManager που έχει αυτό ενσωματωμένο ώστε να μπορούμε να καλέσουμε μεθόδους του (την AddScore εδώ)

Ανοίγουμε το Script DestroyOutOfBounds.cs και τροποποιούμε τον υπάρχοντα με τον κώδικα




```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (4 asset references) | 0 references
public class DestroyOutOfBounds : MonoBehaviour {
    private float topBound = 30;
    private float lowerBound = -10;
    public int healthDamage;

    private GameManager gameMngr;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()   {
        gameMngr = GameObject.Find("GameManager").GetComponent<GameManager>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()   {
        if (transform.position.z > topBound) {
            // Destroy Bullet Object
            Destroy(gameObject);
        } else if (transform.position.z < lowerBound) {
            // An animal has escaped - reduce Health
            if (gameMngr) {
                gameMngr.induceDamage(healthDamage);
            }
            // Destroy Annimal Object
            Destroy(gameObject);
        }
    }
}
```

Public μεταβλητή όπου θα μπορούμε να ρυθμίζουμε από τον editor πόσους πόντους ζημιάς προκαλεί κάθε ζώο που ξεφεύγει

Μεταβλητή όπου θα αποθηκεύσουμε ένα σύνδεσμο στο GameManager αντικείμενο του παιχνιδιού για να έχουμε πρόσβαση στις μεθόδους του εδώ.

Όταν ξεκινάει η εκτέλεση βρίσκουμε το αντικείμενο GameManager στη σκηνή και κρατάμε ένα reference στον κώδικα του component GameManager που περιέχει

Όταν ένα ζώο βγει εκτός ορατής περιοχής (έχει ξεφύγει λοιπόν) ,**καλούμε την αντίστοιχη method InduceDamage που έχω ορίσει στο GameManager** ώστε να αφαιρεθούν οι αντίστοιχοι πόντοι.

ΠΡΟΣΟΧΗ ΣΗΜΑΝΤΙΚΟ: Τα ζώα και οι μπριζόλες δημιουργούνται δυναμικά (on the fly) κατά την εκτέλεση του παιχνιδιού. Δε μπορούμε λοιπόν από πριν να έχουμε συνδέσει το αντικείμενο GameManager της σκηνής μέσω του Editor στην αντίστοιχη μεταβλητή.

Πρέπει να το κάνουμε στο Start() μέσω της εντολής GameObject.Find που αναζητεί στη σκηνή το αντικείμενο με το όνομα “GameManager” και εφόσον το βρει , μέσω του GetComponent αποκτούμε πρόσβαση στον κώδικα GameManager που έχει αυτό ενσωματωμένο ώστε να μπορούμε να καλέσουμε μεθόδους του (την InduceDamage εδώ)

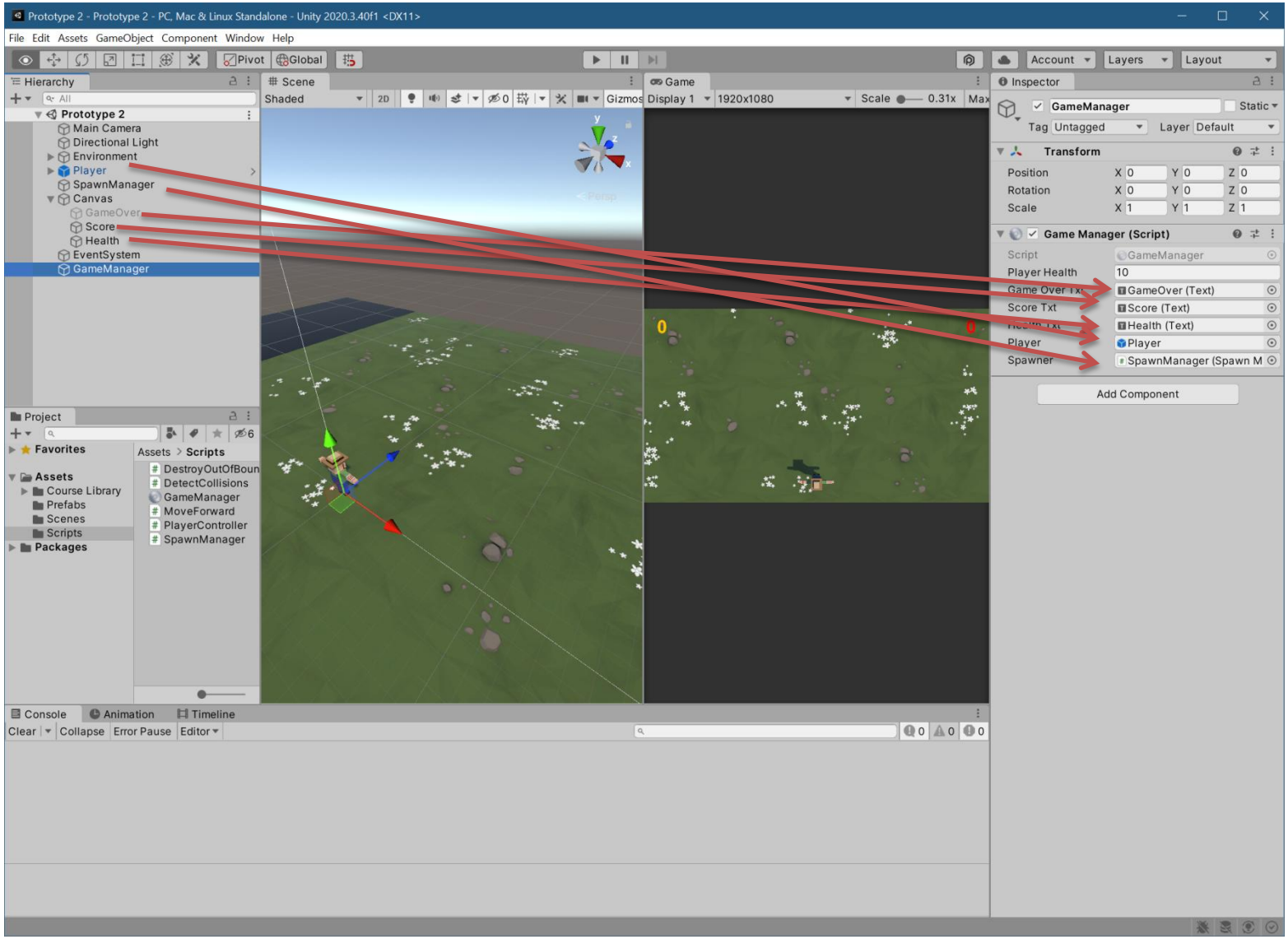
Ανοίγουμε το Script SpawnManager.cs και τροποποιούμε τον υπάρχοντα με τον κώδικα

Assembly-CSharp | SpawnManager | animalPrefabs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
                2                3
    Unity Script (1 asset reference) | 1 reference
5   public class SpawnManager : MonoBehaviour {
6
7       public GameObject[] animalPrefabs;
8       public int animalIndex;
9       private float spawnRangeX = 20;
10      private float spawnPosZ = 20;
11      private float spawnInterval = 1.5f ;
12      private float startDelay = 2;
13
14
15      // Start is called before the first frame update
        Unity Message | 0 references
16      void Start() {
17          InvokeRepeating("SpawnRandomAnimal", startDelay, spawnInterval);
18      }
19
20      // Update is called once per frame
        Unity Message | 0 references
21      void Update() {
22
23
24
25      }
26
        0 references
27      void SpawnRandomAnimal() {
28          Vector3 spawnPos = new Vector3(Random.Range(-spawnRangeX, spawnRangeX), 0, spawnPosZ);
29          animalIndex = Random.Range(0, animalPrefabs.Length);
30          Instantiate(animalPrefabs[animalIndex], spawnPos, animalPrefabs[animalIndex].transform.rotation);
31      }
32
        1 reference
33      public void stopSpawninging() {
34          CancelInvoke();
35      }
36  }
37
```

Προσθέτουμε method stopSpawning που θα τη καλούμε από το GameManager όταν τελειώσει το παιχνίδι.

Στον editor επιλέγουμε το αντικείμενο GameManager και στο αντίστοιχο component του συνδέουμε στις μεταβλητές τα αντικείμενα Text του Canvas καθώς και το αντικείμενο Player και SpawnManager. Τέλος ρυθμίζουμε και την αρχική Υγεία του παίκτη.



Ανοίγουμε ένα ένα τα Prefabs των ζώων και ρυθμίζουμε τους πόντους ζημιάς και τους πόντους σκορ του καθενός. Αποθηκεύουμε τις αλλαγές στο prefab και συνεχίζουμε στο επόμενο ζώο