



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**  

---

**UNIVERSITY OF PIRAEUS**

**Απαλλακτική Ατομική Εργασία 2018-2019**  
**Πανεπιστήμιο Πειραιώς**  
**Τεχνολογίες Ανάπτυξης Ηλεκτρονικών Παιχνιδιών**  
**Διεκπεραίωση Εργασίας στο Εργαλείο**  
**Unity**

## **Στοιχεία Φοιτητή**

Όνομα Φοιτητή

Επώνυμο Φοιτητή

Αριθμός Μητρώου

Email

Ημερομηνία Παράδοσης

Κωνσταντίνος

Κουσουννής

p14086

kwstas654321@gmail.com

18/2/2019

## Εκφώνηση Εργασιών

Το μάθημα θα εξεταστεί αποκλειστικά με την πραγματοποίηση απαλλακτικής εργασίας. Υπάρχουν 2 εναλλακτικές  
Α) Εναλλακτική Επιλέγονται για ανάπτυξη μεταξύ των παρακάτω Tutorials : 1)Adventure Game Tutorial (20  
μονάδες) ( <https://unity3d.com/learn/tutorials/projects/adventure-game-tutorial> ) 2 Space Shooter Tutorial (40  
μονάδες) ( <https://unity3d.com/learn/tutorials/s/space-shootertutorial> ) 3) Survival Shooter Tutorial (40 μονάδες)  
(<https://unity3d.com/learn/tutorials/s/survivalshooter-tutorial> ) 4) 2D Game Kit (60 μονάδες) ( <https://unity3d.com/learn/tutorials/s/2d-game-kit> ) 5) 3D Game Kit (80 μονάδες)  
(<https://unity3d.com/learn/tutorials/s/3d-game-kit> )

Β Εναλλακτική Το αντικείμενο της εργασίας είναι η δημιουργία ενός 3D χώρου με ένα μεσαιωνικό χωριό και της  
περιοχής γύρω από αυτόν (περιβάλλον χώρος, δένδρα/δάσος, λόφοι, ουρανός, κ.τ.λ.) σε Unity3D και η ανάπτυξη  
ενός Role Playing Game, ή First Person Shooter, ή Third Person Shooter, ή Game, που λαμβάνει χώρα στο χωριό.

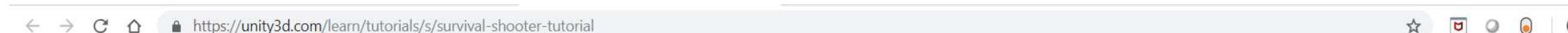
Για την εργασία μου επέλεξα να πραγματοποιήσω τα παρακάτω Tutorials:

- 1)Survival shooter Tutorial .
- 2)Space Shooter Tutorial .
- 3)Adventure Game Tutorial .

**Σε Unity 5.5 έκδοση**

## Survival Shooter Tutorial

Αρχικά πηγαίνουμε στο αντίστοιχο link της Unity και κάνουμε import τα στοιχεία μας.



For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Tutorials > Survival Shooter tutorial

# Survival Shooter tutorial

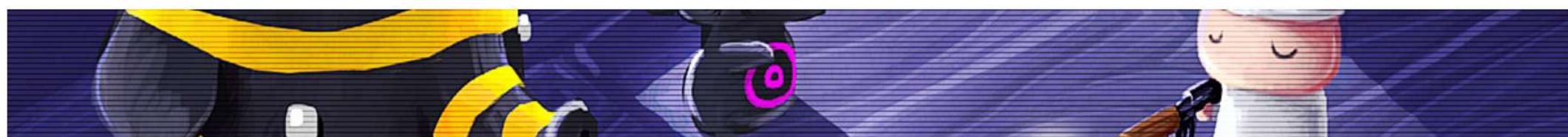
Learn how to make an isometric 3D survival shooter game.

Learn how to make an isometric survival shooter game with this project from Unite training day 2014. NOTE: The original project shown in the videos was created with Unity 4.6 and can be followed with that version, but we recommend that you use Unity 5.1 or greater, and read the Upgrade Guide pdf located in the root of the project files.

[Click here to download the 5.x version of the project.](#)

Questions? [Ask in the official forum thread here.](#)

You can download the slides [here](#).



Κάνουμε add τα στοιχεία μας στο Unity εργαλείο μας . Πατάμε Add to MyAssets

For quick access, place your bookmarks here on the Bookmarks bar. Import bookmarks now...

unity Asset Store

3D 2D Add-Ons Audio Templates Tools VFX

All Assets ▾ Type here to search assets

Plus/Pro Impressive New Assets Bundles 33% Off Shop On

Home > Essentials > Tutorial Projects > Survival Shooter Tutorial

UNITY TECHNOLOGIES

Survival Shooter Tutorial

FREE

Add to My Assets

IMPORTANT!

For users of Unity 5, please read the 'Upgrade to 5' pdf included in the root of the Assets folder to help you follow along with the accompanying [video tutorial series](#).

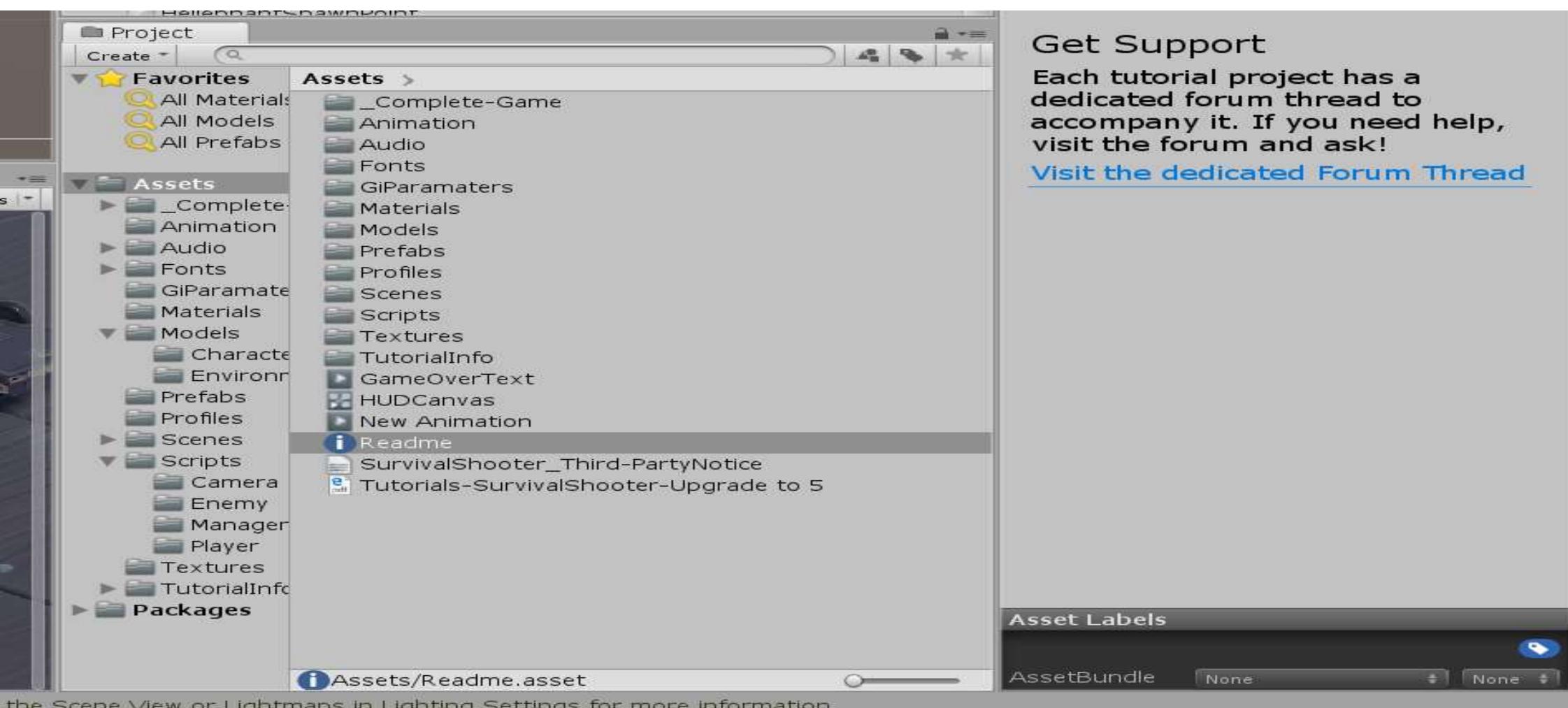
A top-down survival shooter project to help you learn Unity!

This project was used to teach Unite training day 2014 and updated for Unity 5 in 2015.

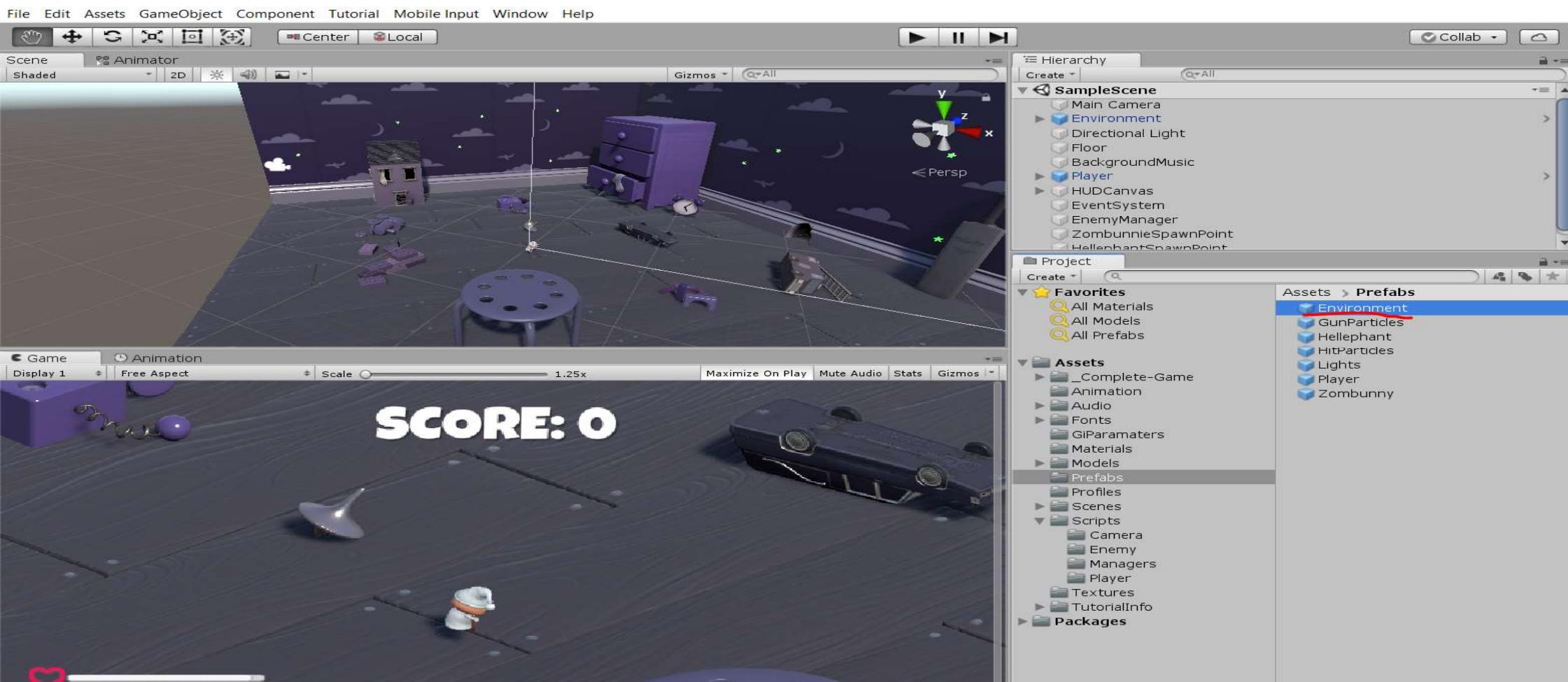
▼ Package contents 79.8 MB

Feedback

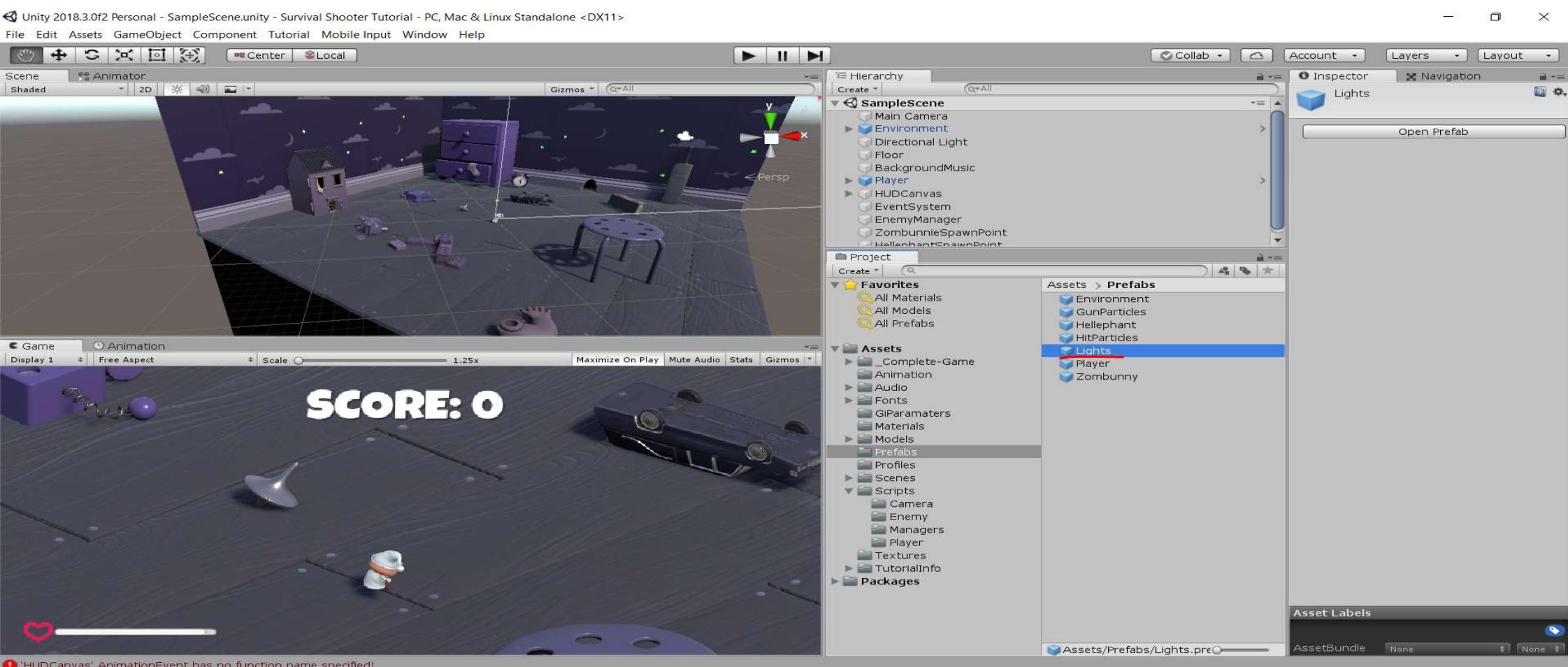
Κάνουμε add τα στοιχεία μας στο Unity εργαλείο μας . Πατάμε Add to MyAssets



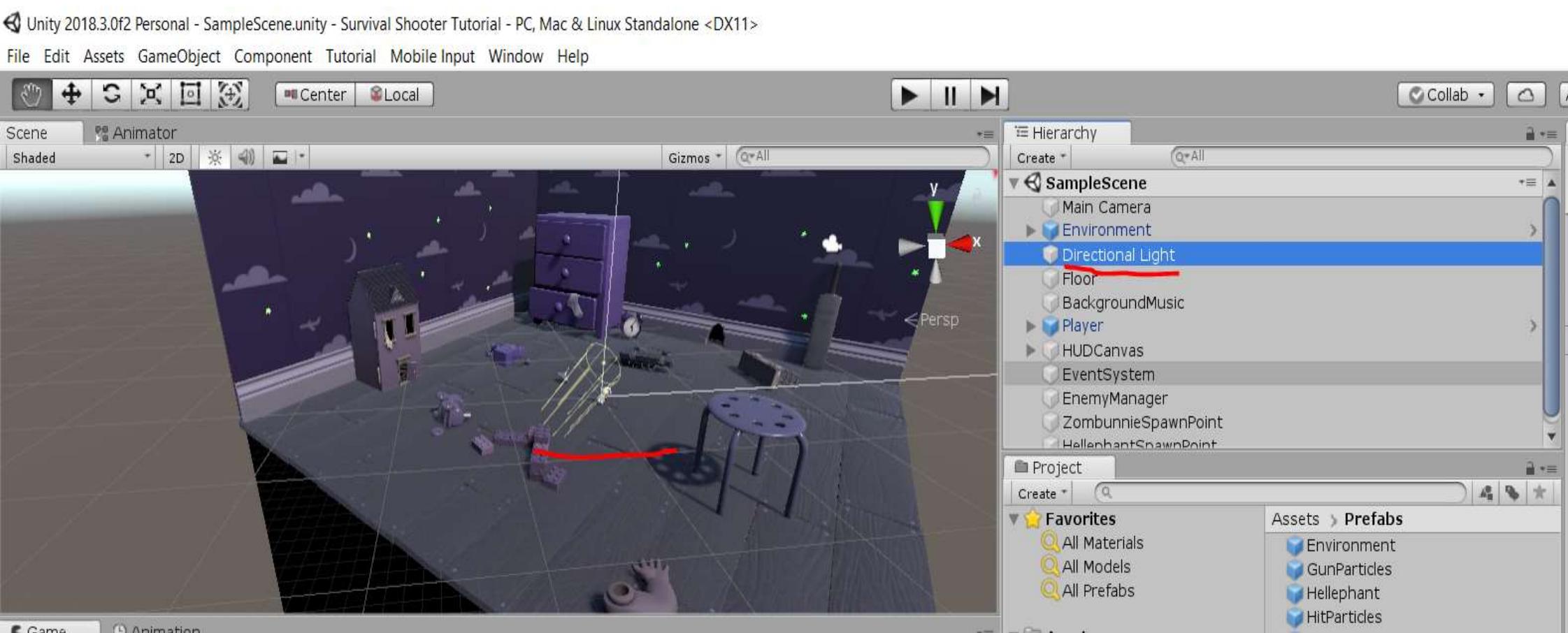
Στην συνέχεια επιλέγουμε Assets->Prefabs->Environment . Και παίρνουμε το environment και το σέρνουμε μέσα στο Hierarchy . Αυτό είναι ένα έτοιμο γραφικό περιβάλλον το οποίο μας εμφανίζει το χώρο στον οποίο θα κινείται ο χαρακτήρας μας.



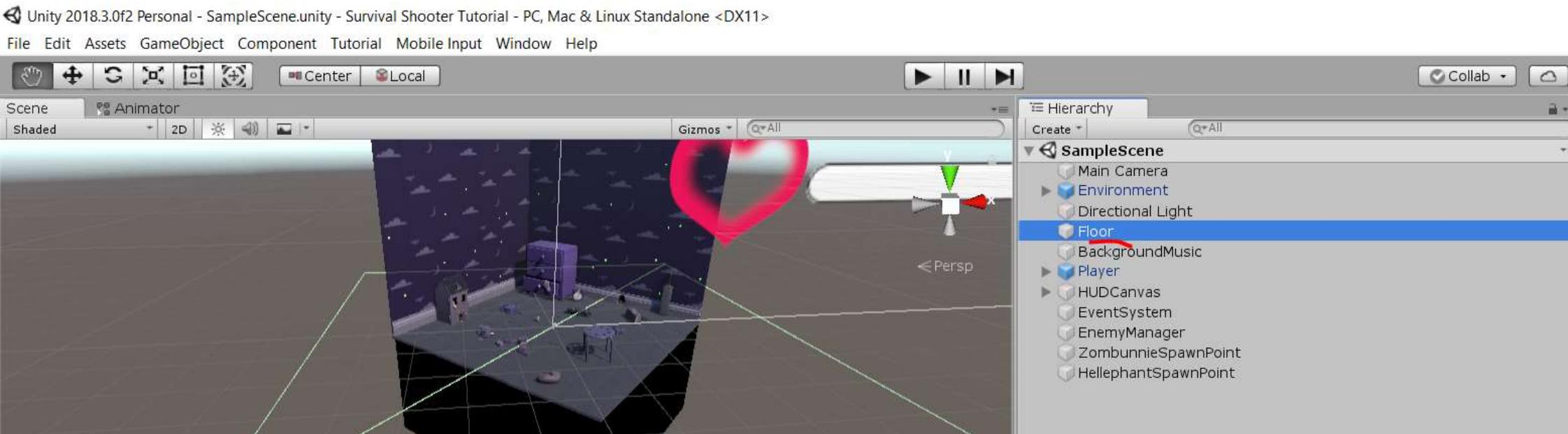
Στην συνέχεια Assets->Prefabs->Lights και σέρνουμε στο project μας τον φωτισμό για το παιχνίδι.



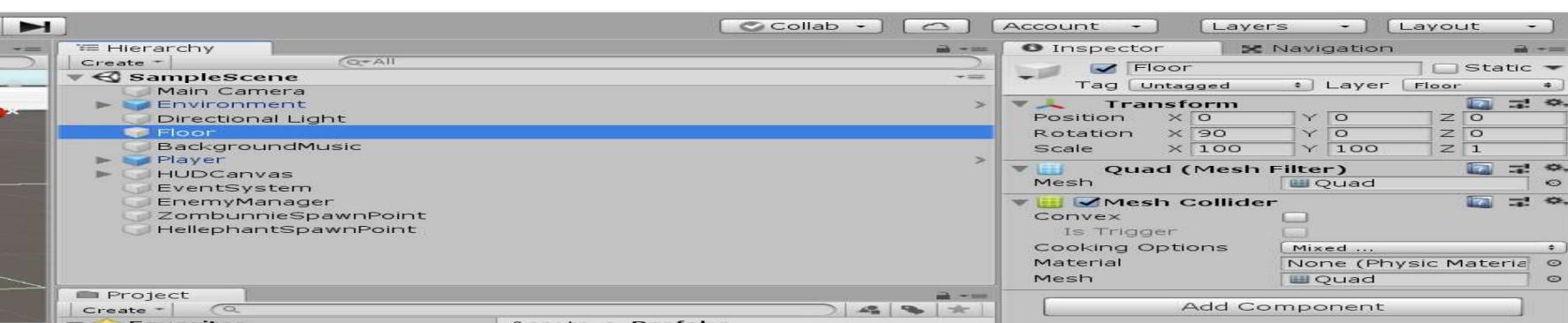
Βλέπουμε τον φωτισμό στο παιχνίδι μας. Το ονομάζουμε Directional Light.



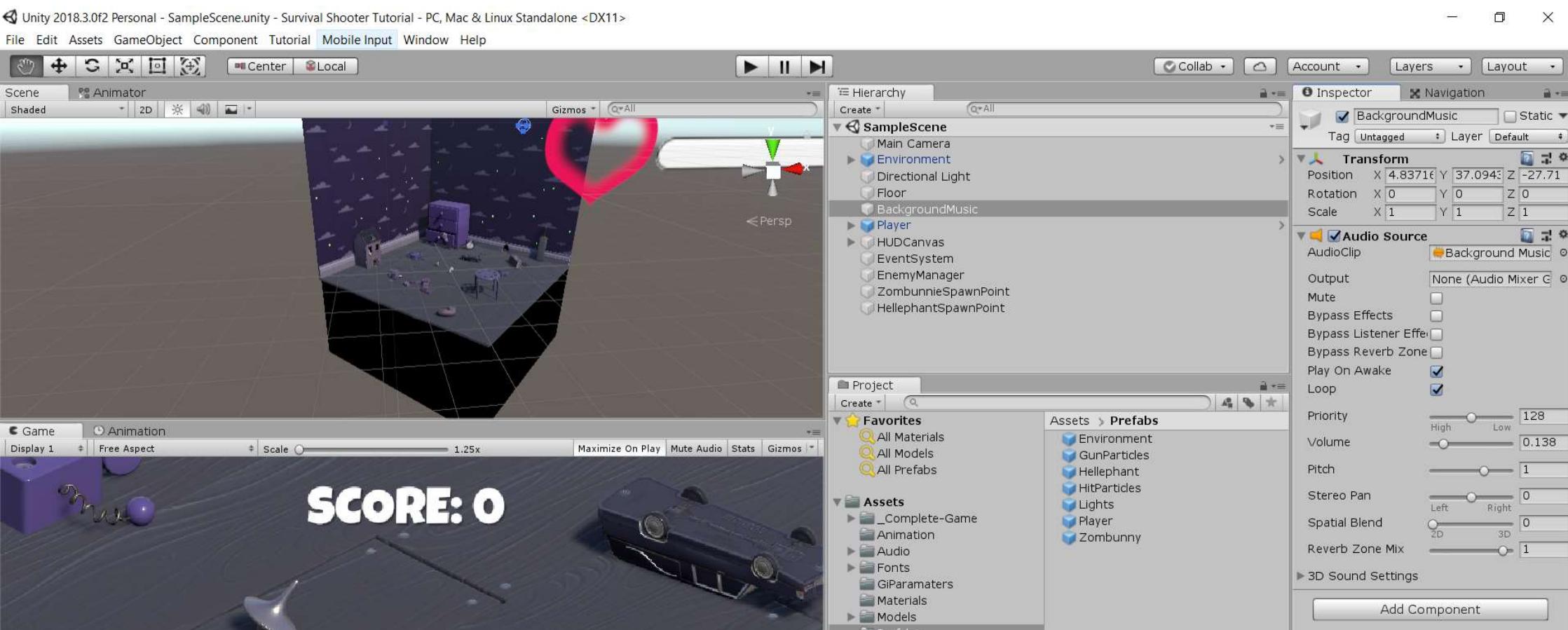
Στην συνέχεια χρειαζόμαστε μια μέθοδο με την οπία θα εντοπίζουμε την κίνηση του παίχτη μάς. Επιλέγουμε **Game Object ->3D object -> Quad** το οποίο το ονομάζουμε **floor** στο γραφικό μας περιβάλλον.



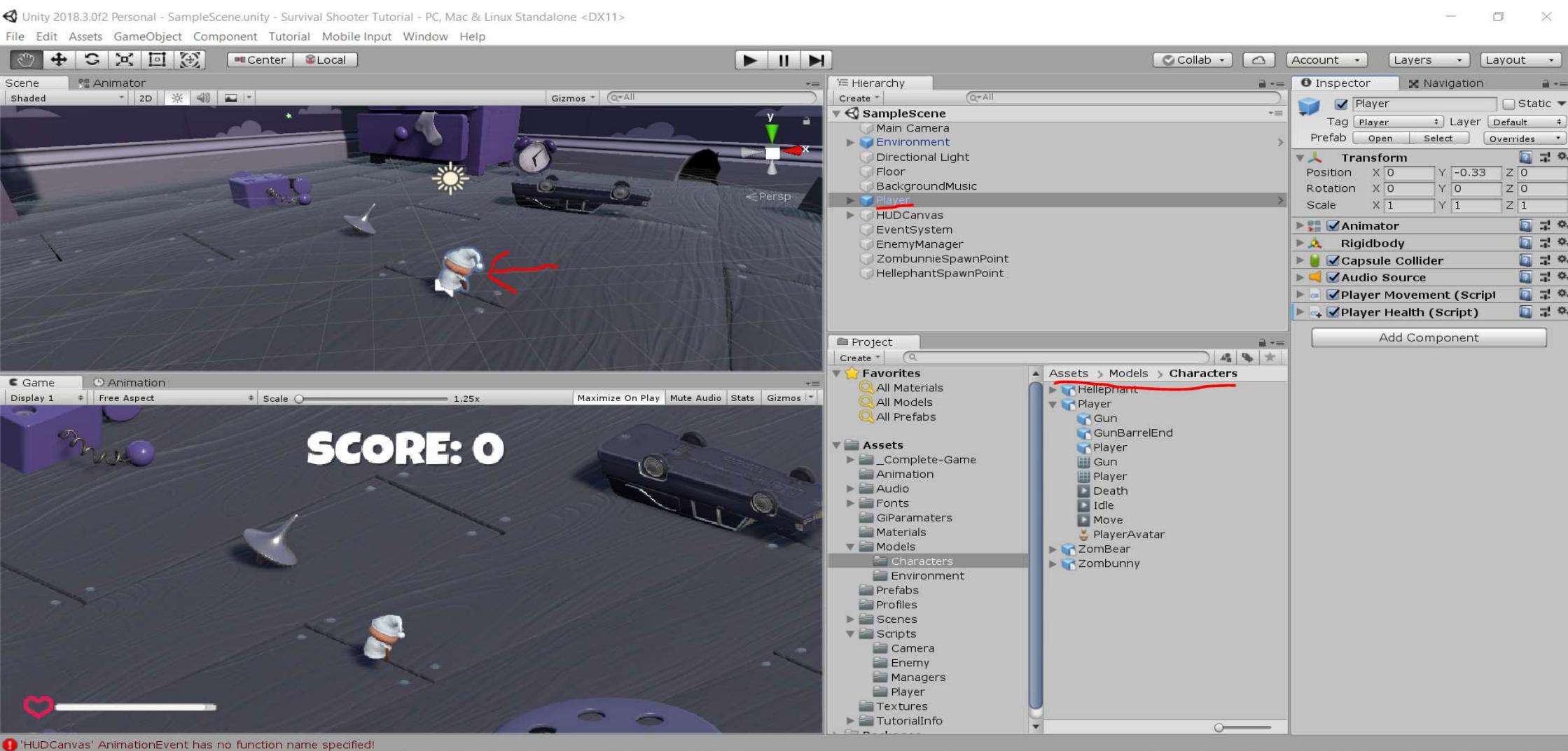
Ορίζουμε τις αντίστοιχες τιμές.



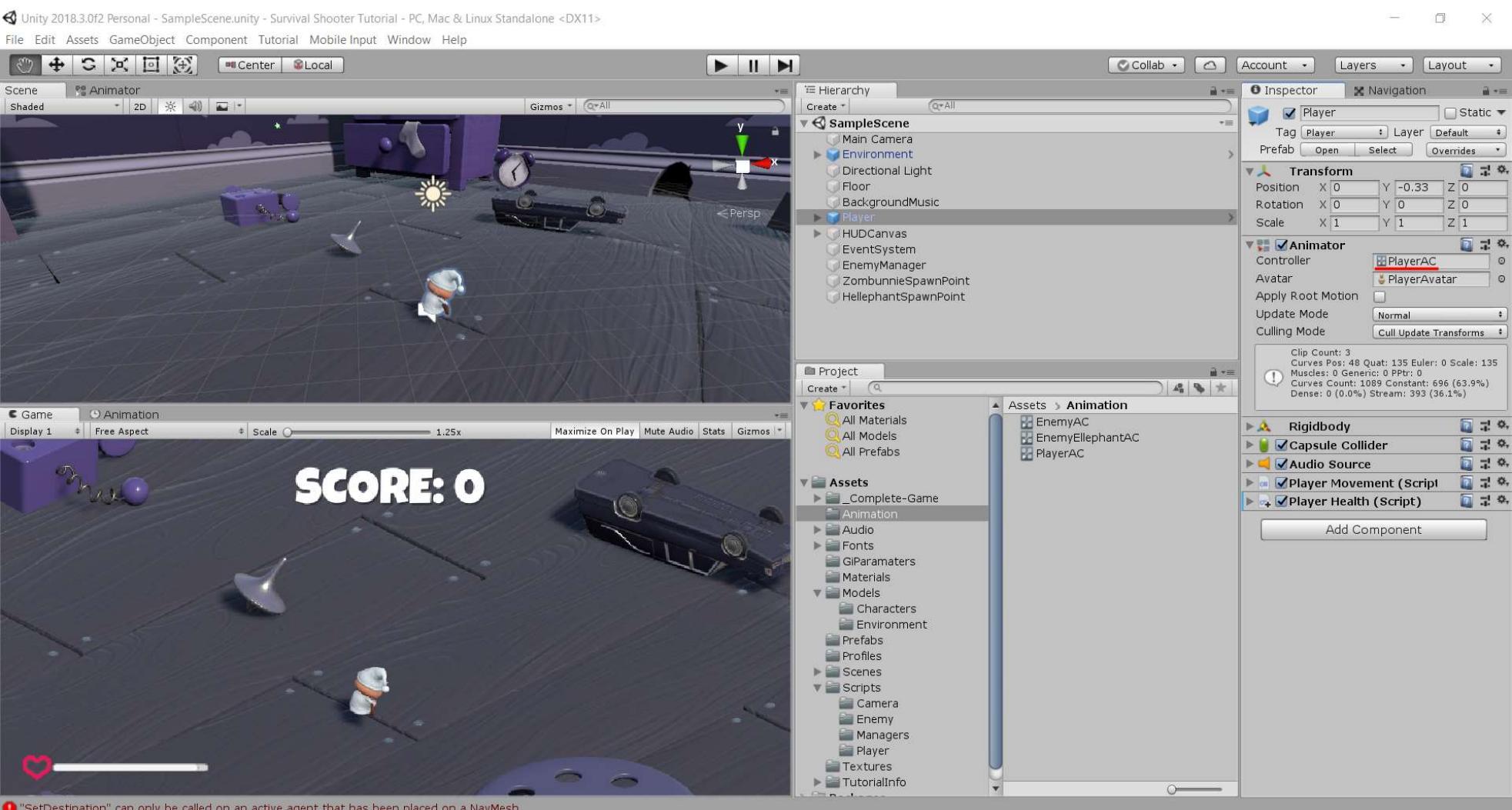
Στην συνέχεια βάζουμε ήχο στο περιβάλλον για να ακούει ο χρήστης κατά την διάρκεια του παιχνιδιού . Αντίστοιχα **Game Object ->Audio** το ονομάζω **BackgroundMusic** **Add Component Audio Source** και στην συνέχεια επιλέγουμε να βάλουμε τον έτοιμο ήχο από το project που κάναμε Import. Επιπλέον επιλέγουμε να κάνουμε check **Play on Awake** στην ουσία να ξεκινάει κατά το ξεκίνημα του παιχνιδιού ο ήχος



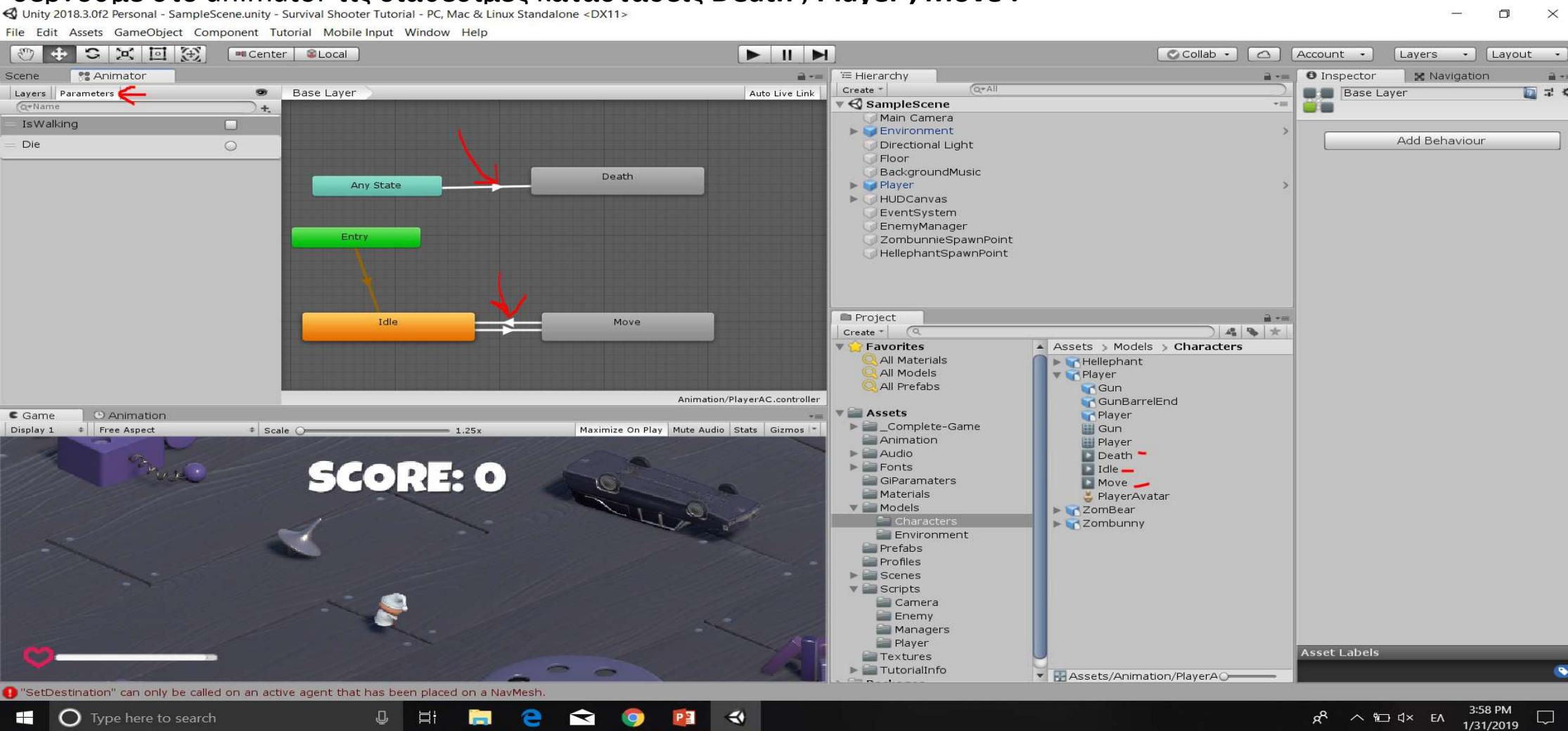
Στην συνέχεια θέλουμε να εισάγουμε στο παιχνίδι μας τον παίχτη του παιχνιδιού **Player**. Πηγαίνουμε στα Assets->Models->Characters και σέρνουμε το παίχτη μας μέσα στο project μας



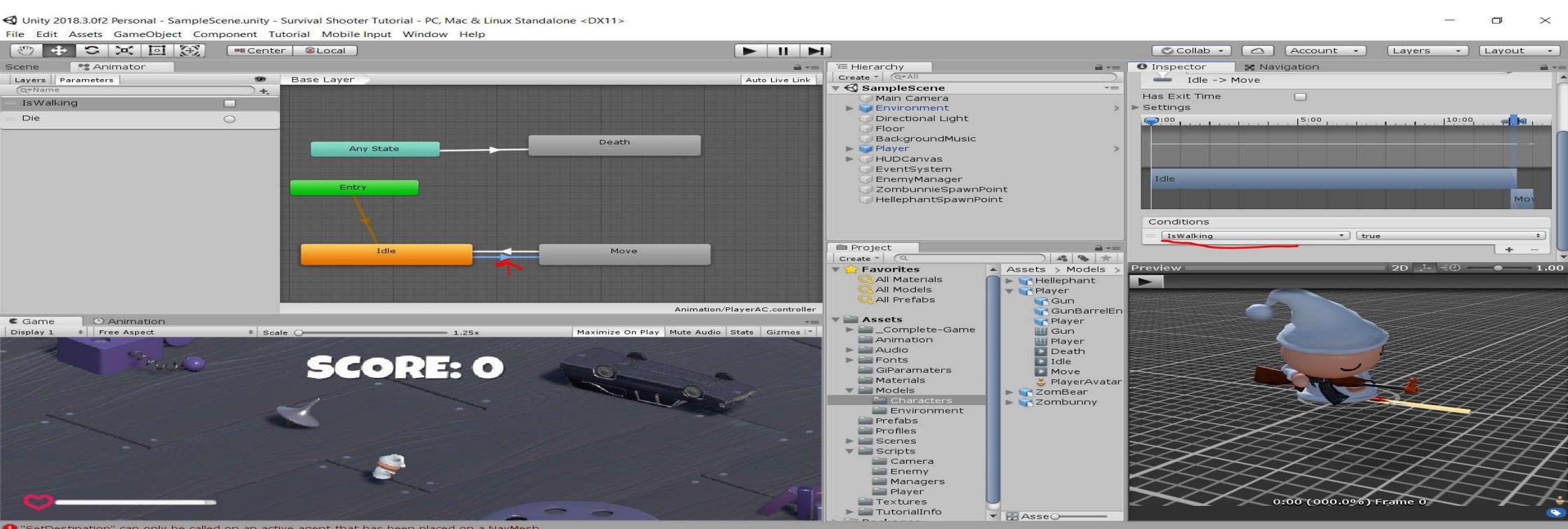
Στην συνέχεια θέλουμε να δώσουμε κίνηση στον παίχτη μας . Για να το κάνουμε επιλέγουμε μέσα στα **Assets** να δημιουργήσουμε μια επαναλαμβανόμενη κίνηση άρα δεξιά κλικ **Create->Animator Override Controller** και το ονομάζουμε **PlayerAC** . Στην συνέχεια το σέρνουμε πάνω στο **player** στο project μου .



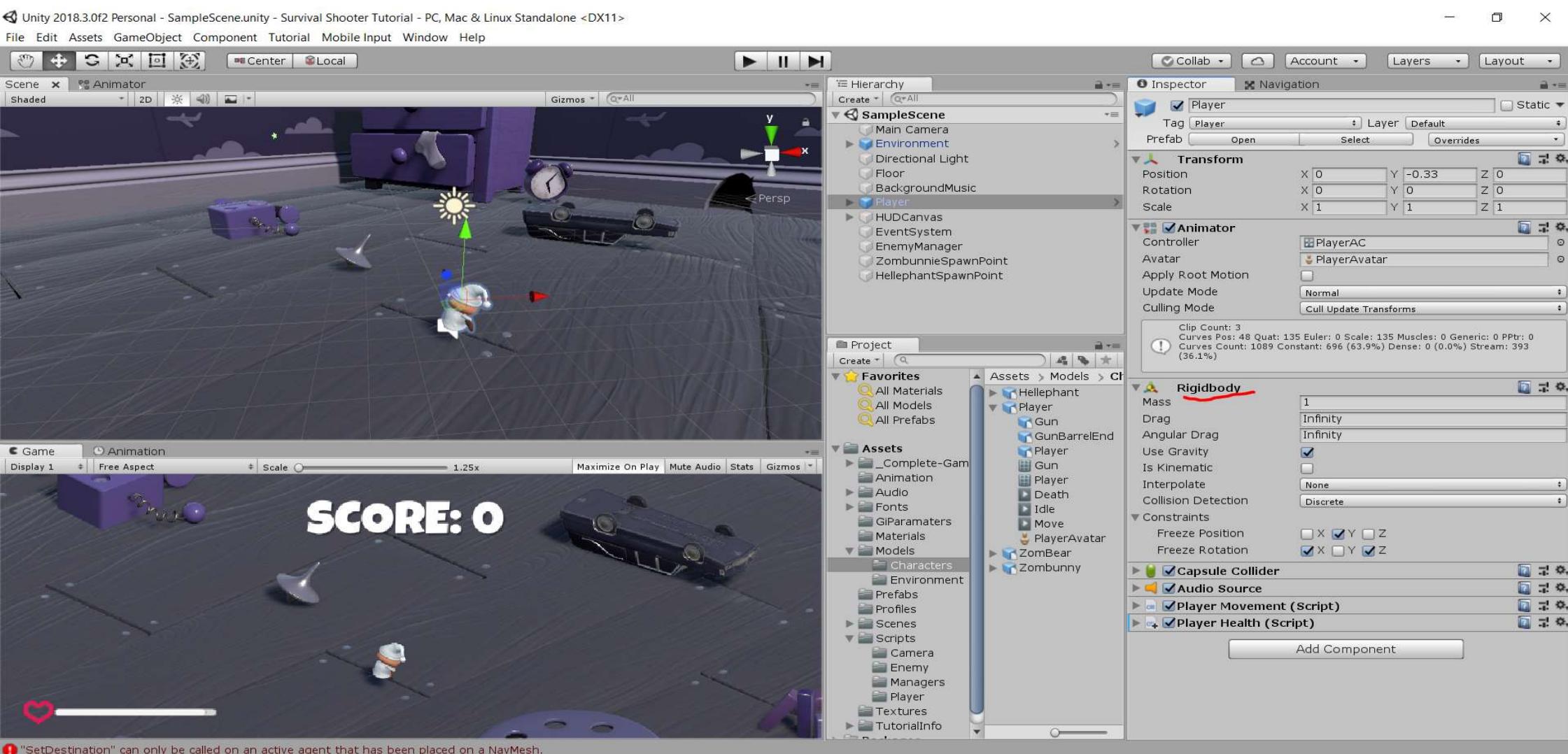
Στην συνέχεια επιλέγουμε την καρτέλα **Animator** και δημιουργούμε την κίνηση του παίχτη μας . Δημιουργούμε δυο παραμέτρους τις οποίες τοποθετούμε στα βελάκια και στην ουσία μεταβαίνουμε από την μια κατάσταση στην άλλη . Αυτές τις καταστάσεις τις παίρνουμε έτοιμες από τις κινήσεις το Player στο **Asset->Models->Characters** σέρνουμε στο animator τις διαθέσιμες καταστάσεις **Death , Player , Move** .



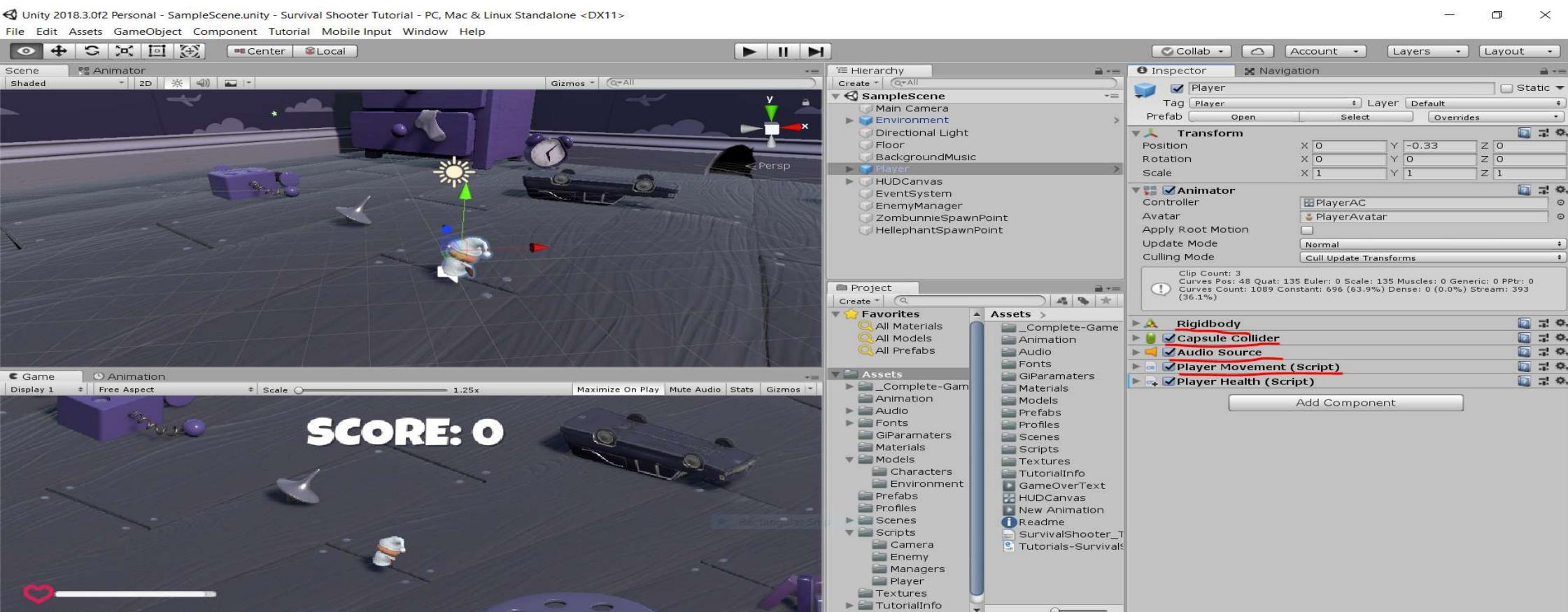
Για κάθε αντίστοιχη μετάβαση αντιστοιχίζουμε στο βελάκι την αντίστοιχη παράμετρο δηλαδή στην αρχή ο παίχτης μας παραμένει ακίνητος αν πληκτρολογήσει ο χρήστης κάποια κίνηση μεταβαίνει στην κατάσταση **Move**, οπότε στο βελάκι βάζουμε την παράμετρο μας **IsWalking** όταν είναι true δηλαδή πατάμε το πληκτρολόγιο μας, σημαντικό κάνουμε **unchecked** το exit time γιατί αλλιώς δεν θα κινείτε όταν πατάμε το πληκτρολόγιο αλλά όταν περνάει ένα χρονικό διάστημα για αυτό τον λόγο το κάνουμε **unchecked** για να πραγματοποιεί κίνηση όταν πατάμε το πληκτρολόγιο. Με την αντίστοιχη διαδικασία βάζουμε την παράμετρο **IsWalking false** όταν ο χρήστης αφήνει το πληκτρολόγιο και τέλος βάζουμε την παράμετρο **Die** να συμβαίνει όταν ο παίχτης βρίσκεται σε οποιαδήποτε κατάσταση.



Στην συνέχεια επιλέγουμε Add Component πάνω στον Player και επιλέγουμε **Rigidbody** και βάζουμε τις αντίστοιχες τιμές που χρειάζονται για την κίνηση μας.



Στην συνέχεια επιλέγουμε Add Component πάνω στον Player και επιλέγουμε **Rigidbody** και βάζουμε τις αντίστοιχες τιμές που χρειάζονται για την κίνηση μας. Με την ίδια διαδικασία κάνουμε add ένα **Capsule Collider** για να μπορεί ο παίχτης μας να συγκρούεται με τα αντικείμενα που υπάρχουν μέσα στο περιβάλλον που κινείτε. Επιπλέον κάνουμε **Add Audio Source** και τοποθετούμε τον ήχο από τα Asset μας **Player Hurt** κάνουμε uncheck. Τέλος πατάμε add component για να εισάγουμε ένα c# script.



Στην συνέχεια ανοίγουμε το script και έχουμε γράψει τον αντίστοιχο κώδικα για την κίνηση μας.

The screenshot shows the Unity Editor interface with the code editor window open. The script file is named 'PlayerMovement.cs'. The code defines a class 'PlayerMovement' that inherits from 'MonoBehaviour'. It includes fields for speed, movement direction, animator, rigidbody, floor mask, and camera ray length. The 'Awake' method initializes these fields. The ' FixedUpdate' method stores input axes. The code editor has syntax highlighting and a status bar at the bottom showing '100 %' zoom.

```
PlayerMovement.cs  X
PlayerMovement
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 6f;           // The speed that the player will move at.

    Vector3 movement;                // The vector to store the direction of the player's movement.
    Animator anim;                  // Reference to the animator component.
    Rigidbody playerRigidbody;      // Reference to the player's rigidbody.
    int floorMask;                  // A layer mask so that a ray can be cast just at gameobjects on the floor layer.
    float camRayLength = 100f;        // The length of the ray from the camera into the scene.

    void Awake()
    {
        // Create a layer mask for the floor layer.
        floorMask = LayerMask.GetMask("Floor");

        // Set up references.
        anim = GetComponent<Animator>();
        playerRigidbody = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        // Store the input axes.
    }
}
```

Server Explorer    Toolbox

Output

Show output from:

## Ανάλυση Κώδικα PlayerMovement.cs

public float speed = 6f; Είναι για το πόσο γρήγορα θα κινείτε ο παίχτης μας.

Vector3 movement; Για να αποθηκεύσουμε την κίνηση που θέλουμε να εφαρμοστεί στο παίχτη μας.

Animator anim; Κάνουμε αναφορά για να πάρουμε το **animator component** που δημιουργήσαμε πριν.

Rigidbody playerRigidbody; Αναφορά στο Rigidbody του παίχτη.

int floorMask; Εκεί που πατάει ο παίχτης.

Στην συνέχεια έχουμε την μέθοδο awake η οποία παίρνει τις τιμές μας από το κανονικά project μας.

Μέθοδος Fixed update για να κινείτε ο παίχτης μας στιγμιαία.

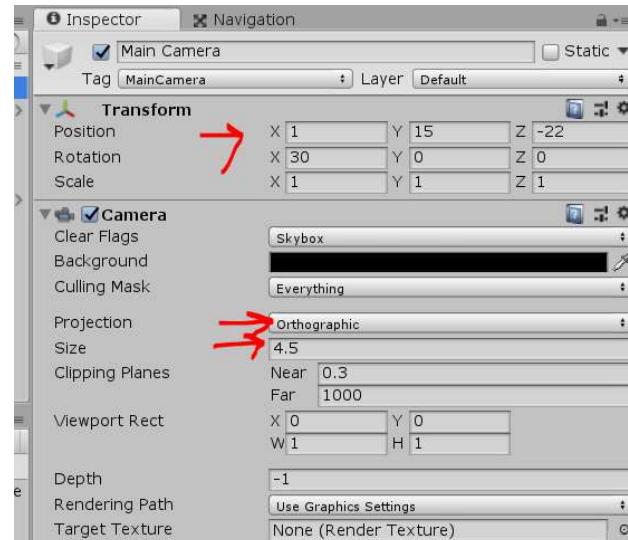
Μέθοδος Move για να κινείται ο παίχτης μου με σταθερή ταχύτητα και να μην υπάρχει αυξομείωση.

Μέθοδος Turning είναι για να κινείτε ο χαρακτήρας μας με βάση την κάμερα μας.

Μέθοδος Animating είναι για την αλληλεπίδραση παίχτη με χρήστη δηλαδή να κινείτε.

## Ρύθμιση Κάμερας

Ρυθμίζουμε την κάμερα μας



## Δημιουργούμε ένα script το οποίο το ονομάζουμε Camera Follow.cs

The screenshot shows the Unity Editor interface with the code editor window open. The title bar says "CameraFollow.cs" and "PlayerMovement.cs". The code editor displays the following C# script:

```
using UnityEngine;
using System.Collections;

public class CameraFollow : MonoBehaviour
{
    public Transform target; // The position that that camera will be following.
    public float smoothing = 5f; // The speed with which the camera will be following.

    Vector3 offset; // The initial offset from the target.

    void Start()
    {
        // Calculate the initial offset.
        offset = transform.position - target.position;
    }

    void FixedUpdate()
    {
        // Create a position the camera is aiming for based on the offset from the target.
        Vector3 targetCamPos = target.position + offset;

        // Smoothly interpolate between the camera's current position and it's target position.
        transform.position = Vector3.Lerp(transform.position, targetCamPos, smoothing * Time.deltaTime);
    }
}
```

The Unity interface includes the Server Explorer, Toolbox, and Solution Explorer on the left, and the Properties panel on the right.

Δημιουργούμε αυτό το script για να είναι ποιο απαλή η οπτική γωνία που έχει ο παίχτης του παιχνιδιού μας

public float smoothing = 5f; Η ταχύτητα με την οποία θα ακολουθεί η κάμερα μας

Vector3 targetCamPos = target.position + offset; Η θέση που βρίσκεται η κάμερα μας

transform.position = Vector3.Lerp(transform.position, targetCamPos, smoothing \* Time.deltaTime); Η πραγματική κίνηση της κάμερα μας.

Στην συνέχεια τοποθετούμε στην κάμερα ποιον επιθυμεί να ακολουθήσει.



## Δημιουργία Αντιπάλου στο παιχνίδι μας:

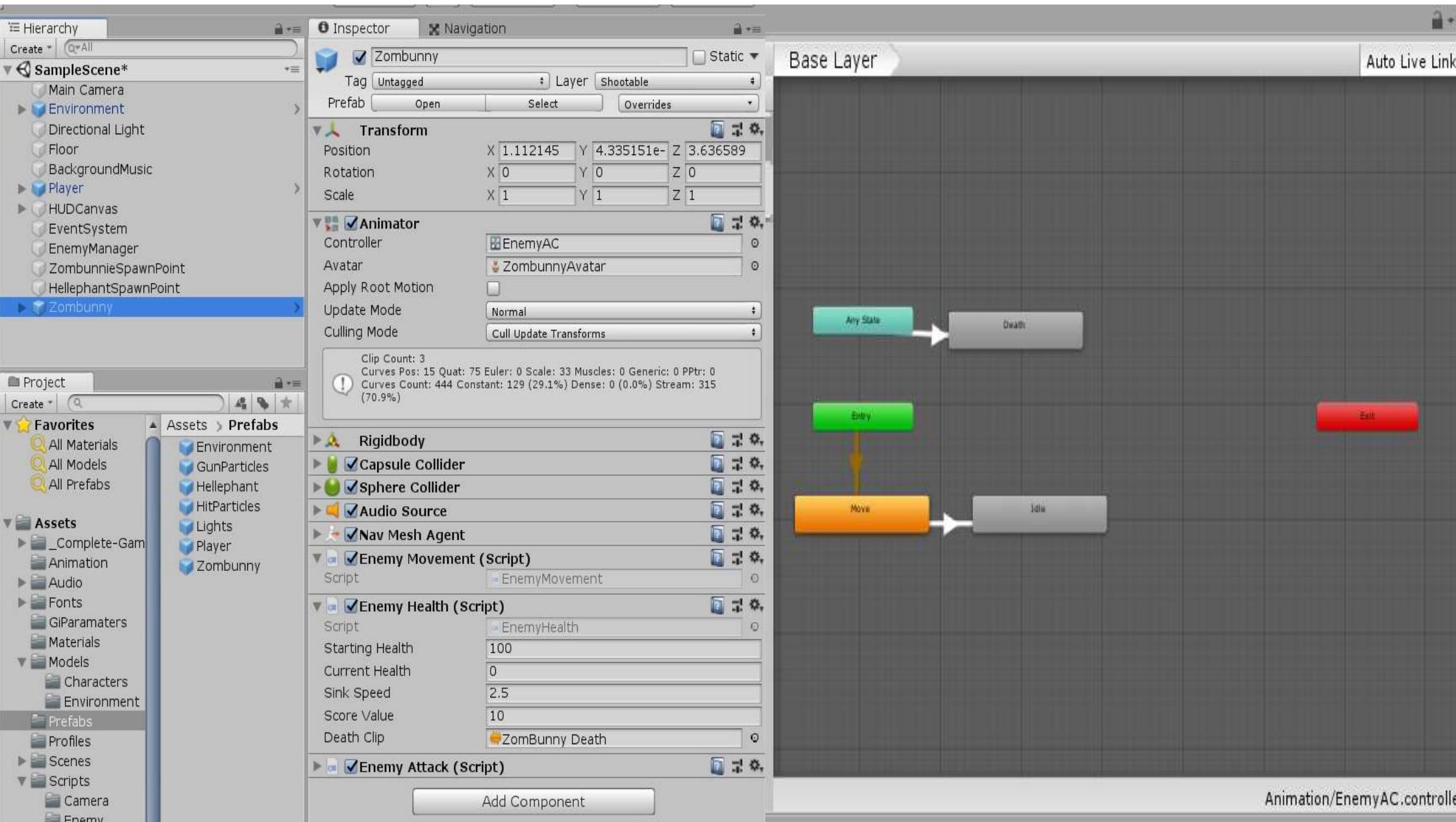
Παίρνουμε ένα **zombunny** από το έτοιμο project μου **Assets->Models->Characters**

Στην συνέχεια θέλουμε να βάλουμε στον αντίπαλο μας να έχει την δυνατότητα να επιτίθεται .Οπότε πηγαίνουμε Assets->Prefabs και να τοποθετήσουμε στο **zombunny** μας να επιτίθεται . Επιλέγουμε το στην επιλογή Layer shootable .Κάνουμε add component έναν Rigidbody ,στην συνέχεια θέλουμε το **zombunny** μας να μπορεί να συγκρούεται με αντικείμενα που βρίσκονται μέσα στο περιβάλλον μας , άρα κάνουμε add ένα Capsule Collider .Επιπλέον χρειαζόμαστε ένα **sphere collider** για να υπάρχει μια περίμετρος στην όποια όταν εισέρχεται ο κανονικός παίχτης θα του επιτίθεται .

Επιπρόσθετα τοποθετούμε ένα audio source για να παράγει έναν ήχο όταν δέχεται ζημία από τον player του παιχνιδιού μας .

Όσο αναφορά την λειτουργία zombie δηλαδή να ακολουθεί τον παίχτη μας και να προσπαθεί να τον σκοτώσει θα πρέπει να κάνουμε add το **nav mesh Agent** το οποίο αξιοποιεί το επίπεδο που δημιουργήσαμε στην αρχή για να διαβάσει το έδαφος και να πάρει την θέση του παίχτη μας.

Τέλος θέλουμε να δημιουργήσουμε την κίνηση στον **zombunny** μας θα δημιουργήσουμε την κίνηση με την ίδια ακριβώς διαδικασία όπως δημιουργήσαμε την κίνηση στον player μας.



The screenshot shows the Unity Editor's interface. On the left, the Project and Assets panels are visible, with the Assets panel showing scripts named 'EnemyAttack', 'EnemyHealth', and 'EnemyMovement'. The 'EnemyMovement' script is selected and highlighted with a red underline. In the center, the Inspector window is open, showing the 'EnemyMovement Import Settings' tab with 'C#' selected. Below it, the 'Imported Object' section shows 'EnemyMovement' with another 'C#' icon. The 'Assembly Information' section shows the filename as 'Assembly-CSharp.dll'. The main area displays the C# code for the 'EnemyMovement' script:

```
using UnityEngine;
using System.Collections;

public class EnemyMovement : MonoBehaviour
{
    Transform player;
    //PlayerHealth playerHealth;
    //EnemyHealth enemyHealth;
    UnityEngine.AI.NavMeshAgent nav;

    void Awake ()
    {
        player = GameObject.FindGameObjectWithTag ("Player").transform;
        //playerHealth = player.GetComponent <PlayerHealth> ();
        //enemyHealth = GetComponent <EnemyHealth> ();
        nav = GetComponent <UnityEngine.AI.NavMeshAgent> ();
    }

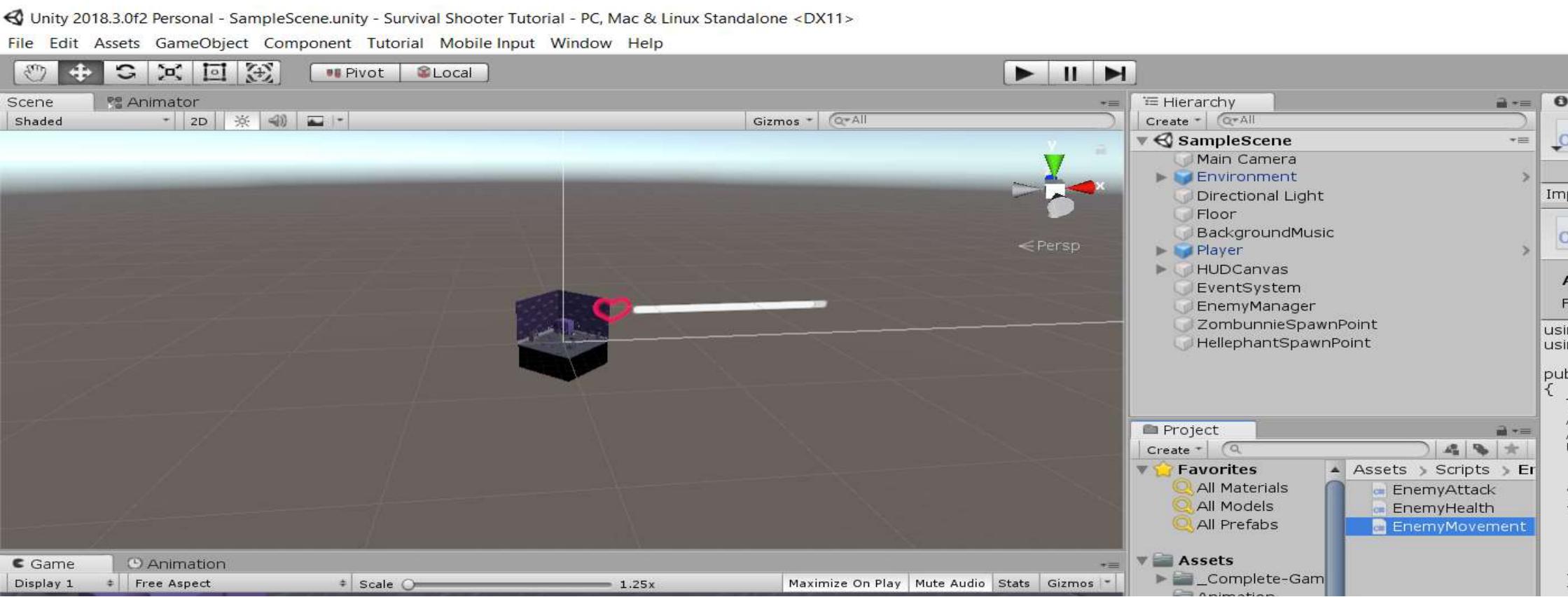
    void Update ()
    {
        //if(enemyHealth.currentHealth > 0 && playerHealth.currentHealth >
        0)
        //{
        //    nav.SetDestination (player.position);
        //}
        //else
        //{
        //    nav.enabled = false;
        //}
    }
}
```

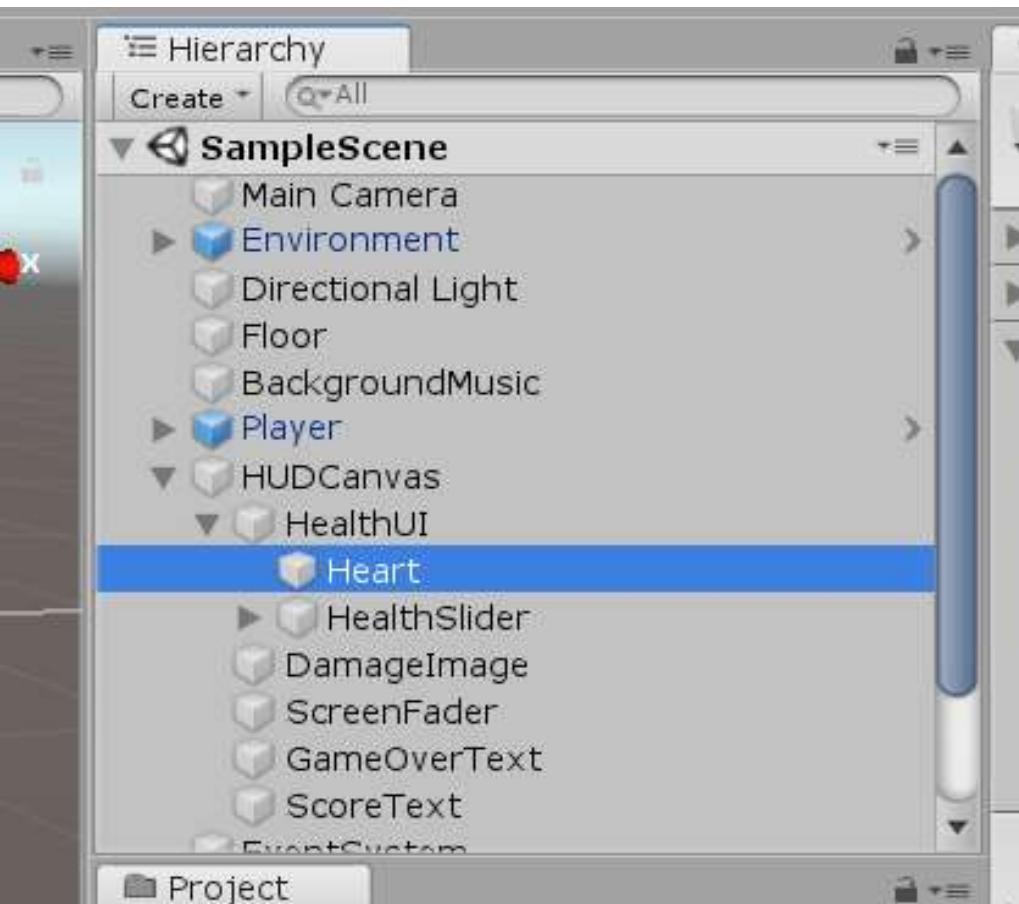
Δημιουργία κίνησης **zombunny** για να κινείται αυτόματα και να ακολουθεί τον **player**.

Με τις εντολές :

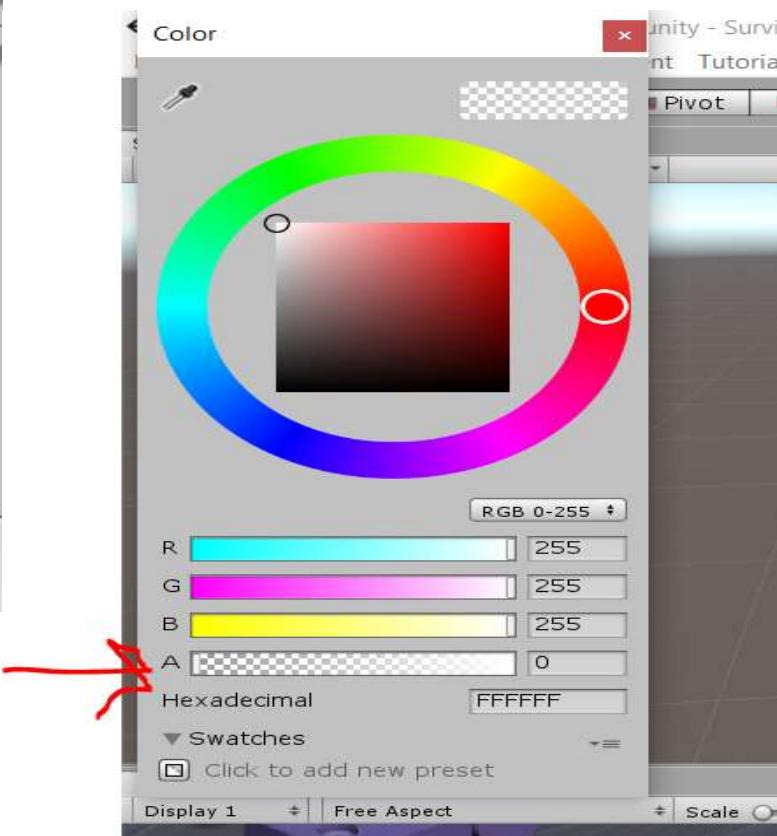
```
player = GameObject.FindGameObjectWithTag ("Player").transform;
nav.SetDestination (player.position);
```

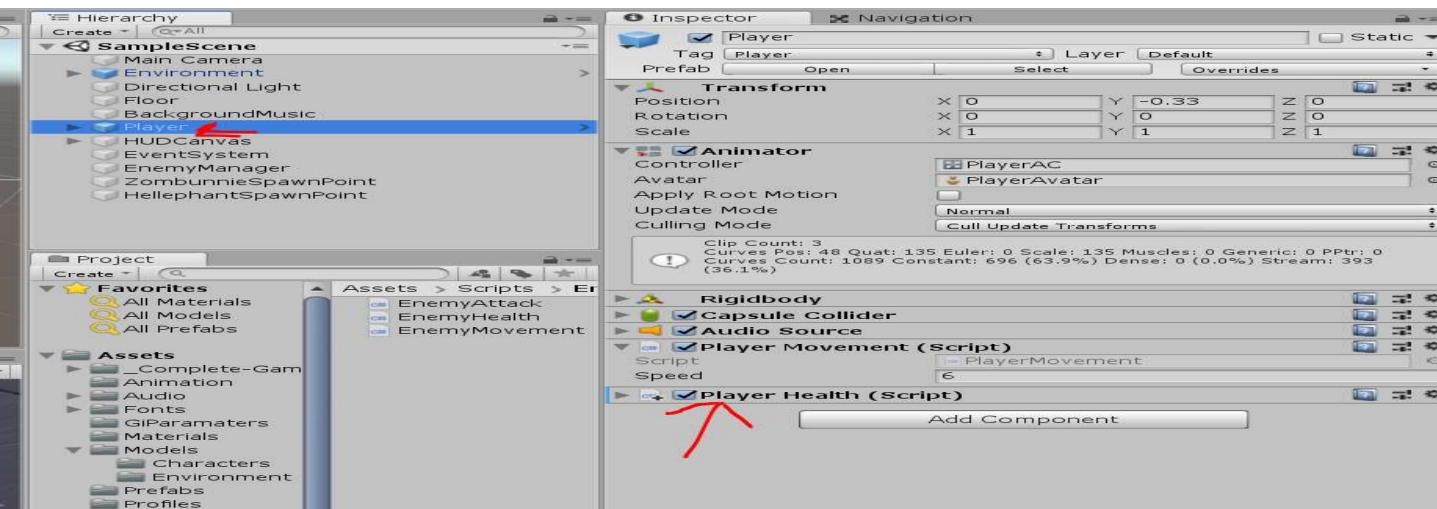
Δημιουργία **Health** στον παίχτη μας . Επιλέγουμε δημιουργία canva . GameObject->UI->Canvas για να εμφανίζει στο παιχνίδι μας την ζωή του player κάτω αριστερά. Επιλέγουμε να βάλουμε το image για την καρδιά μας στον canva για να βλέπουμε που είναι η ζωή του παίχτη μας.





Δημιουργία μπάρας ζωής **HealthSlider** για τον παίχτη μας. Επίσης κάνουμε add ένα DamageImage το οποίο θα εμφανίζει στον παίχτη μας όταν θα δέχεται ζημία το οποίο είναι διαφανές και εμφανίζεται μόνο όταν δέχεται επίθεση με ένα script.





```

void Update ()
{
    if(damaged)
    {
        damageImage.color = flashColour;
    }
    else
    {
        damageImage.color = Color.Lerp (damageImage.color, Color.clear, flashSpeed * Time.deltaTime);
    }
    damaged = false;
}

public void TakeDamage (int amount)
{
    damaged = true;

    currentHealth -= amount;

    healthSlider.value = currentHealth;

    playerAudio.Play ();

    if(currentHealth <= 0 && !isDead)
    {
}

```

Παίρνουμε το αντίστοιχο script Assets->Script->Player και παίρνουμε το script μας PlayerHealth.cs και το σέρνουμε πάνω στον player μας .

Εδώ χρησιμοποιούμε το script για να αλληλοεπιδράσουμε με τον παίχτη του παιχνιδιού μας και τον canva που δημιουργήσαμε. Και δημιουργούμε τον θάνατο του παίχτη μας

```

    void Death ()
    {
        isDead = true;

        //playerShooting.DisableEffects ();

        anim.SetTrigger ("Die");

        playerAudio.clip = deathClip;
        playerAudio.Play ();

        playerMovement.enabled = false;
        //playerShooting.enabled = false;
    }

    public void RestartLevel ()
    {
        SceneManager.LoadScene (0);
    }
}

```

EnemyAttack.cs

```
using UnityEngine;
using System.Collections;

public class EnemyAttack : MonoBehaviour
{
    public float timeBetweenAttacks = 0.5f;
    public int attackDamage = 10;

    Animator anim;
    GameObject player;
    PlayerHealth playerHealth;
    EnemyHealth enemyHealth;
    bool playerInRange;
    float timer;

    void Awake ()
    {
        player = GameObject.FindGameObjectWithTag ("Player");
        playerHealth = player.GetComponent <PlayerHealth> ();
        enemyHealth = GetComponent<EnemyHealth>();
        anim = GetComponent <Animator> ();
    }
}
```

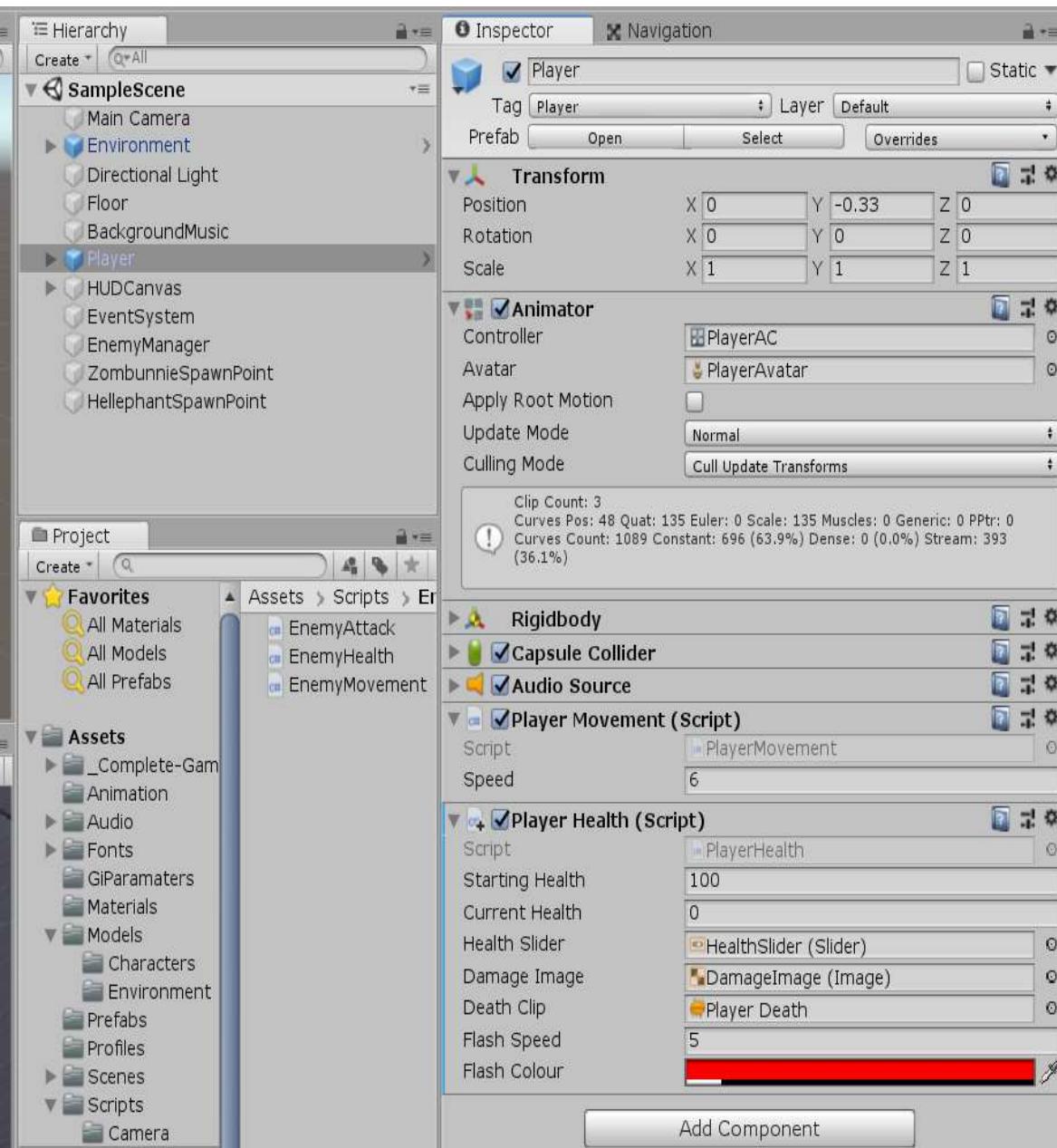
Ανοίγουμε το EnemyAttack.cs .Το zombunny μας όταν έρχεται σε εμβέλεια που είναι κοντά στον Player αρχίζει να επιτίθεται δηλαδή γίνεται true και μειώνει την ζωή του Player ενώ όταν βγαίνει εκτός εμβέλειας ο παίχτης μας σταματάει να του μειώνει την ζωή και το κάνει false .

```
void OnTriggerEnter (Collider other)
{
    if(other.gameObject == player)
    {
        playerInRange = true;
    }
}

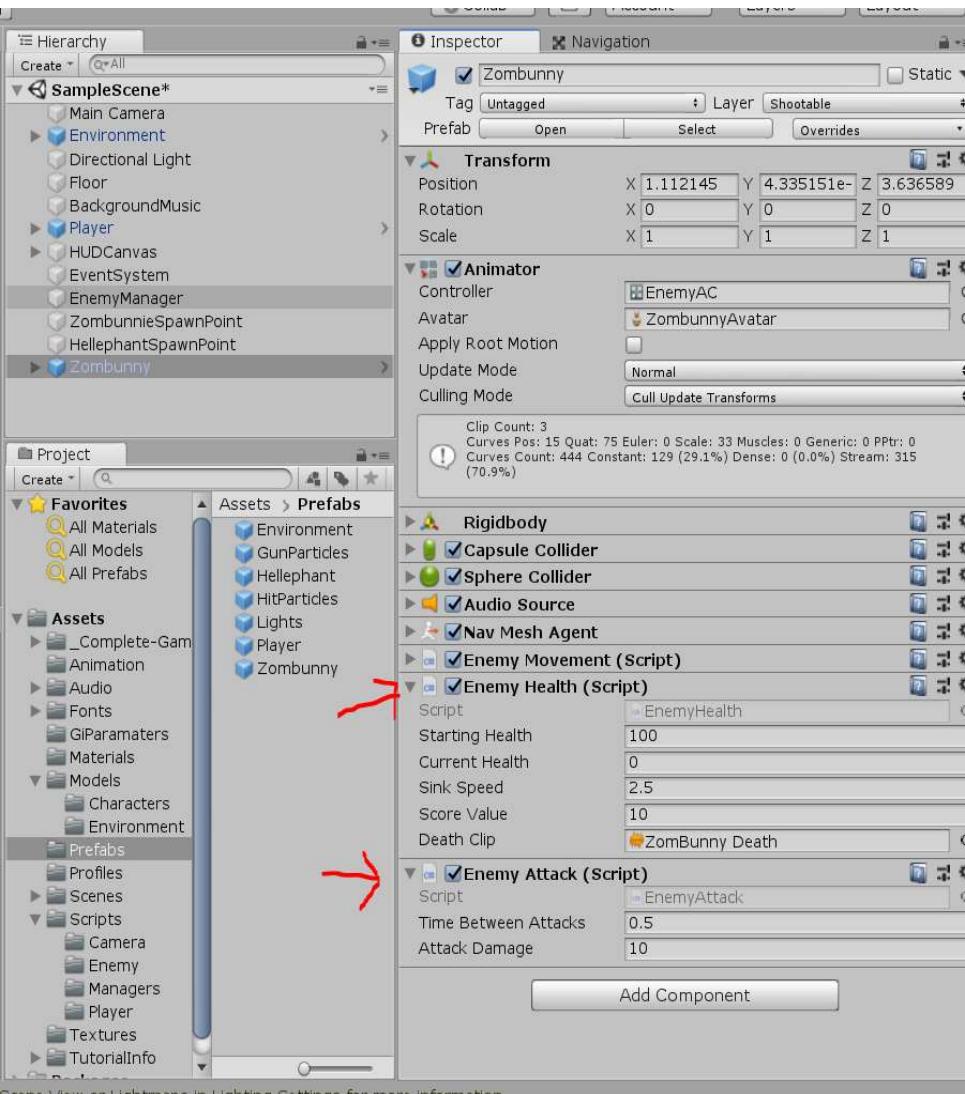
void OnTriggerExit (Collider other)
{
    if(other.gameObject == player)
    {
        playerInRange = false;
    }
}

void Update ()
{
    timer += Time.deltaTime;

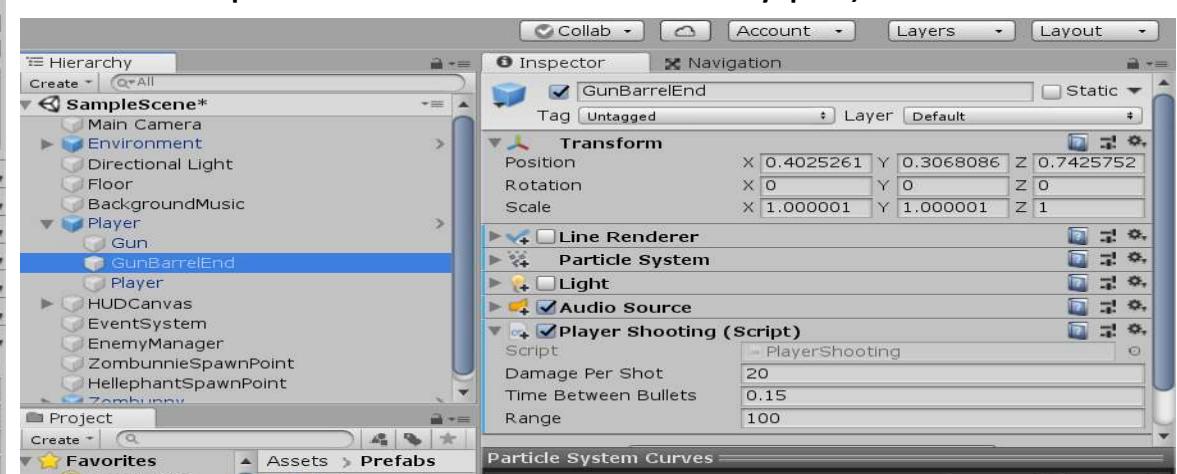
    if(timer >= timeBetweenAttacks && playerInRange && enemyHealth.currentHealth > 0)
    {
        Attack ();
    }
}
```

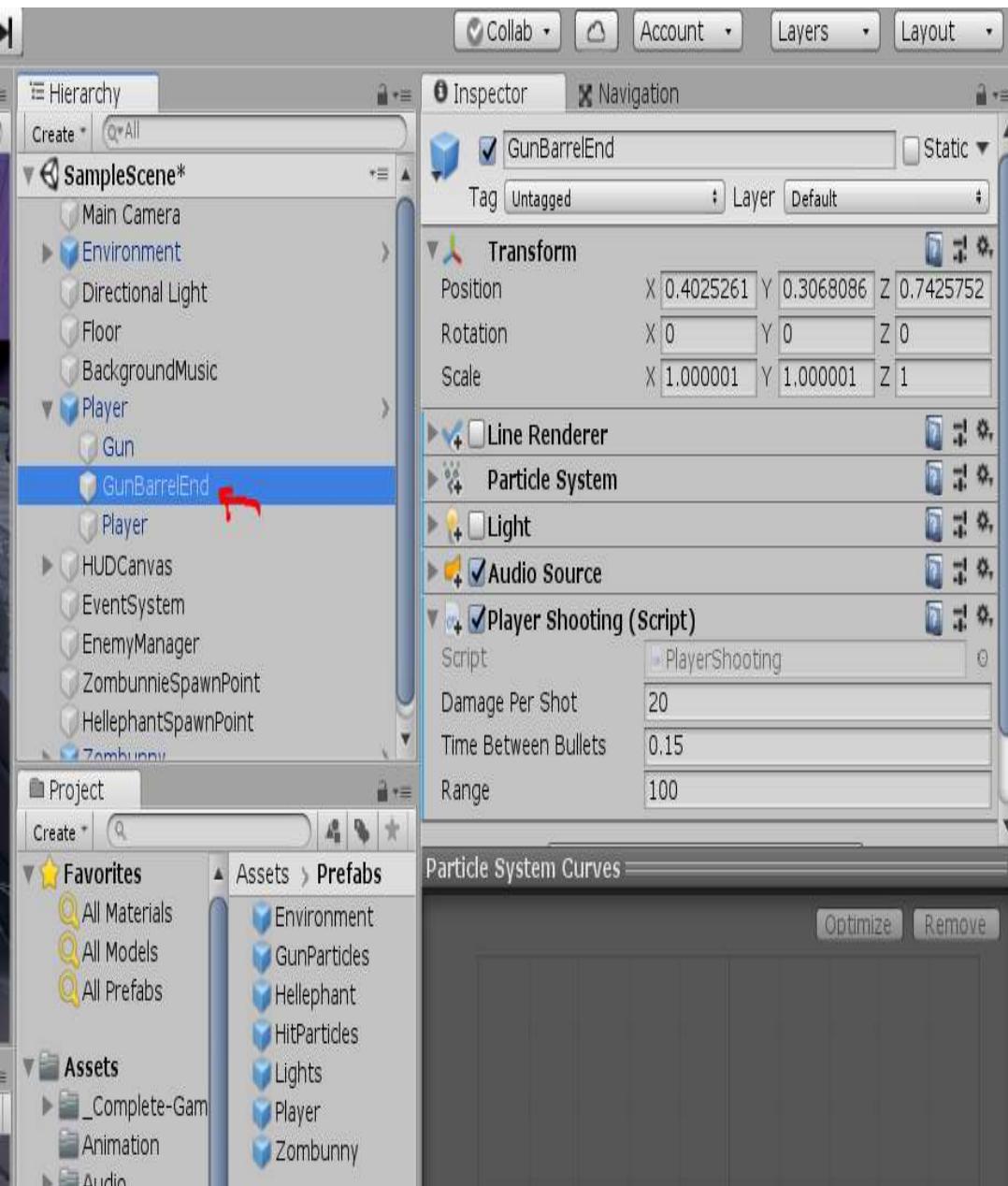


Όπως βλέπουμε ρυθμίζουμε το Flash Colour σε κόκκινο είναι το χρώμα που θα εμφανίζεται όταν ο παίχτης μας δέχεται ζημία.  
Starting Health = 100;  
Current Health = 0;  
HealthSlider για την ζωή του παίχτη μας  
DamageImage για την ζημία που δέχεται ο παίχτης μας.  
Death Clip όταν πεθαίνει ο παίχτης περιέχει το animation του θανάτου του.



Δημιουργούμε την επίθεση του player μας και τον θάνατο του player μας . Βάζουμε τα αντίστοιχα script Enemy Health , Enemy Attack .Τα οποία αντίστοιχα ελέγχουν την μπάρα ζωής του **zombunny** μας και την επίθεση του στον Player.Επιπλέον έχουμε και τον θάνατο από το **zombunny** μας όπου έχουμε βάλει το animation για το θάνατο του zombunny μας.





Παίρνουμε από το έτοιμο project μας **Assets->Prefabs GunBarrelEnd**. Κάνουμε add τα αντίστοιχα **Audio Source Player Shooting** για τον παίχτη μας που πυροβολεί Light για το χρώμα με το οποίο πυροβολεί το όπλο μας. Για το player shooting script το δημιουργούμε εμείς. Αυτό περιέχει την ζημία που ασκεί για κάθε πυροβολισμό τον χρόνο ανάμεσα στον πυροβολεί το όπλο μας . Την ακτίνα που φτάνει ο πυροβολισμός μας .



Επιλέγουμε GameObject->Ui->Text και δημιουργούμε ένα κουτί που θα περιέχει την βαθμολογία μας.

Δημιουργούμε ένα c# file score manager το οποίο κάνει update το σκορ μας στο παιχνίδι κάθε φορά που σκοτώνει έναν αντίπαλο ο παίχτης μας.

ScoreManager.cs

```

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class ScoreManager : MonoBehaviour
{
    public static int score;

    Text text;

    void Awake ()
    {
        text = GetComponent <Text> ();
        score = 0;
    }

    void Update ()
    {
        text.text = "Score: " + score;
    }
}

```

Προσθέτουμε στο EnemyHealth.cs να αλλάζει score κάθε φορά που πεθαίνει το zombunny μας.

EnemyHealth.cs

```

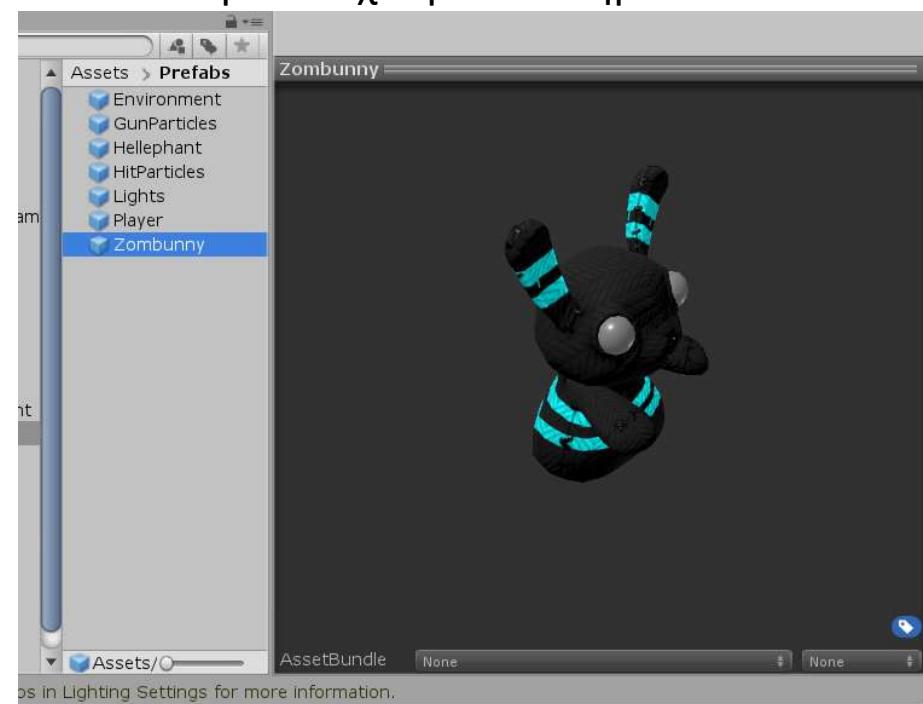
public class EnemyHealth : MonoBehaviour
{
    public int startingHealth = 100;
    public int currentHealth;
    public float sinkSpeed = 2.5f;
    public int scoreValue = 10; ←
    public AudioClip deathClip;
}

```

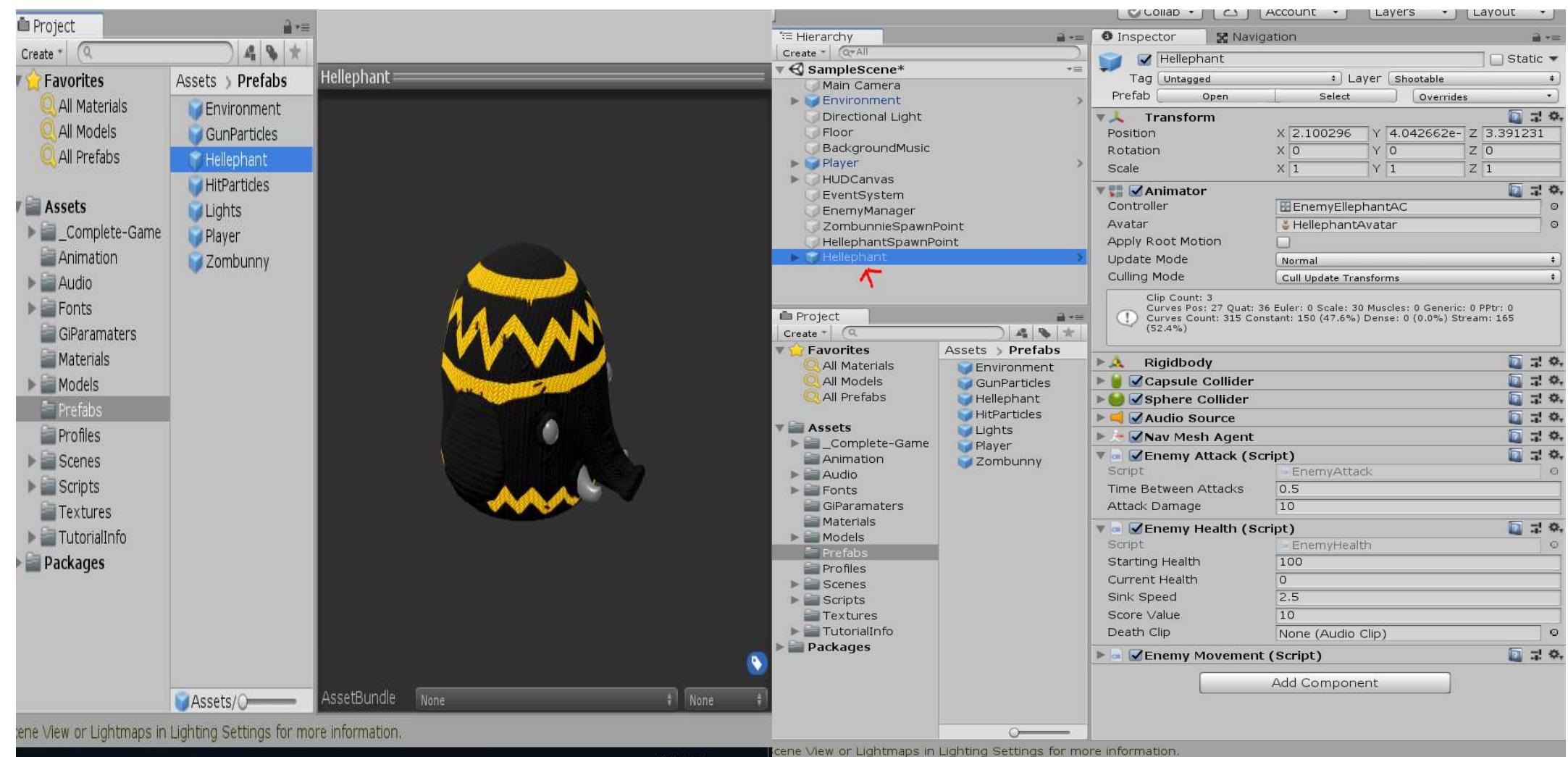
Αυξάνουμε το score κάθε φορά που πεθαίνει ένας αντίπαλος

```
public void StartSinking ()  
{  
    GetComponent <UnityEngine.AI.NavMeshAgent> ().enabled = false;  
    GetComponent <Rigidbody> ().isKinematic = true;  
    isSinking = true;  
    ScoreManager.score += scoreValue;  
    Destroy (gameObject, 2f);  
}
```

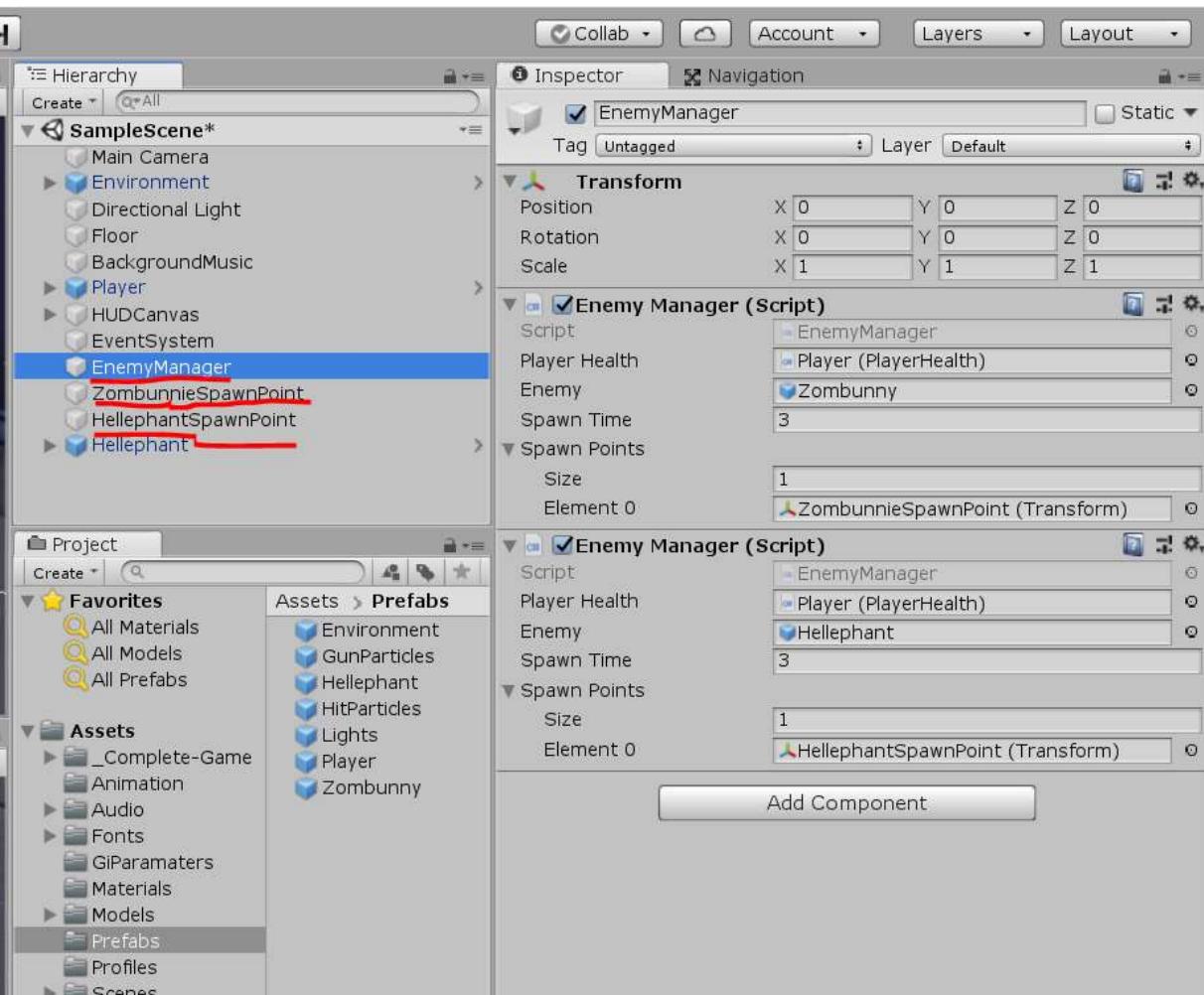
Στην συνέχεια παίρνουμε το zombunny μας και το τοποθετούμε μέσα στο Folder Prefabs και το διαγράφουμε από το παιχνίδι μας καθώς τώρα έχουμε μόνο ένα zombunny ενώ θέλουμε να έχουμε ένα σημείο που τα κάνουμε spawn δηλαδή να δημιουργούνται πολλοί αντίπαλοι.



Με τον ίδιο τρόπο που δημιουργήσαμε το zombunny μας θα δημιουργήσουμε και ένα δεύτερο αντίπαλο ο οποίος θα λέγεται Hellephant



Δημιουργούμε τα σημεία που θα δημιουργούνται οι αντίπαλοι μας στο παιχνίδι. Έχουμε δημιουργήσει τα σημεία που δημιουργούνται οι αντίπαλοι μας ZombunnySpawnPoint ,HellephantSpawnPoint και δημιουργούμε ένα EnemyManager ο οποίος ελέγχει την ροή με την οποία δημιουργούνται οι αντίπαλοι μας.



Δημιουργούμε δυο Script Enemy Manager . βάζουμε κάθε φορά τον αντίστοιχο αντίπαλο που θέλουμε να δημιουργείτε καθώς και να ασκεί ζημία στον player.

```
using UnityEngine;
public class EnemyManager : MonoBehaviour
{
    public PlayerHealth playerHealth;
    public GameObject enemy;
    public float spawnTime = 3f;
    public Transform[] spawnPoints;

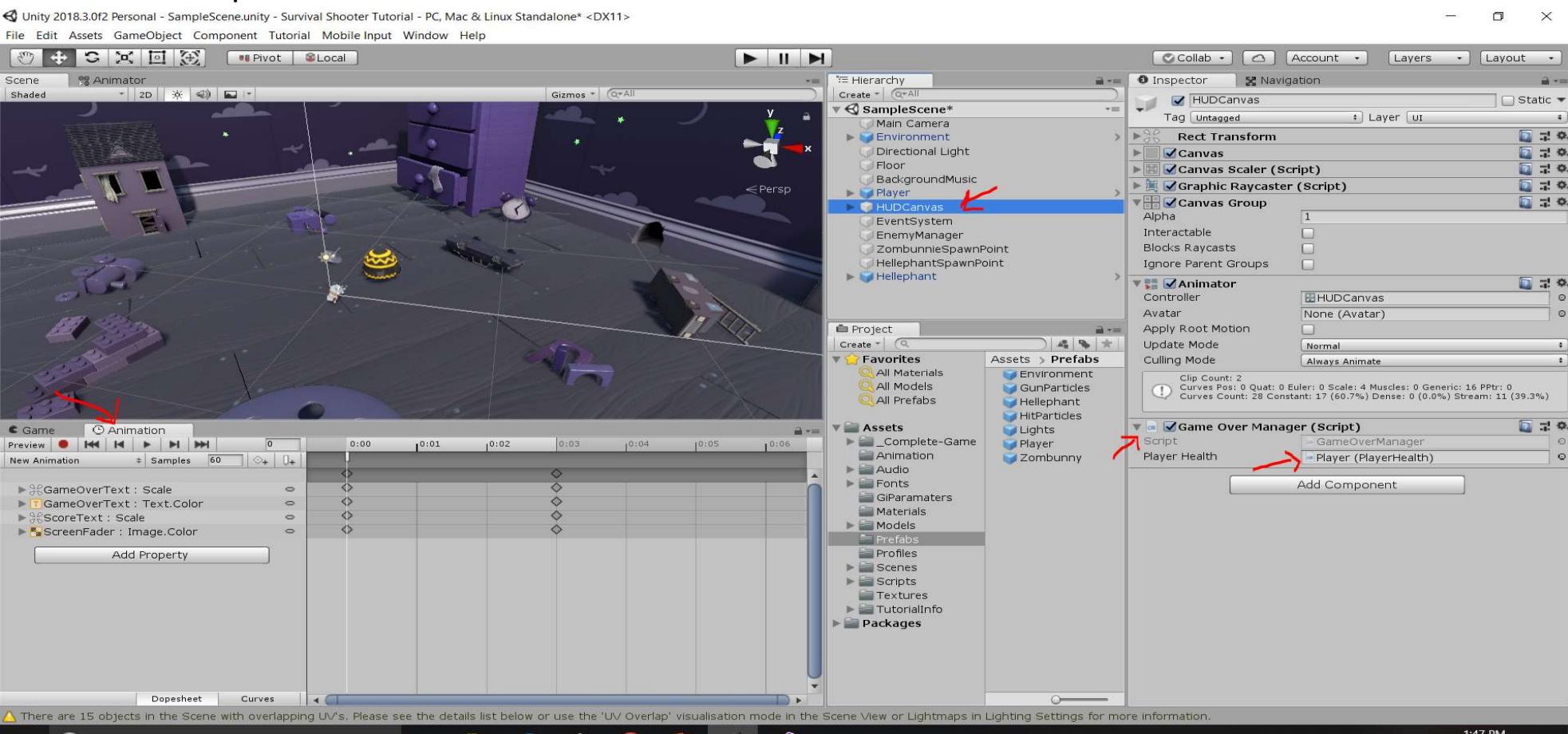
    void Start ()
    {
        InvokeRepeating ("Spawn", spawnTime, spawnTime);
    }

    void Spawn ()
    {
        if(playerHealth.currentHealth <= 0f)
        {
            return;
        }

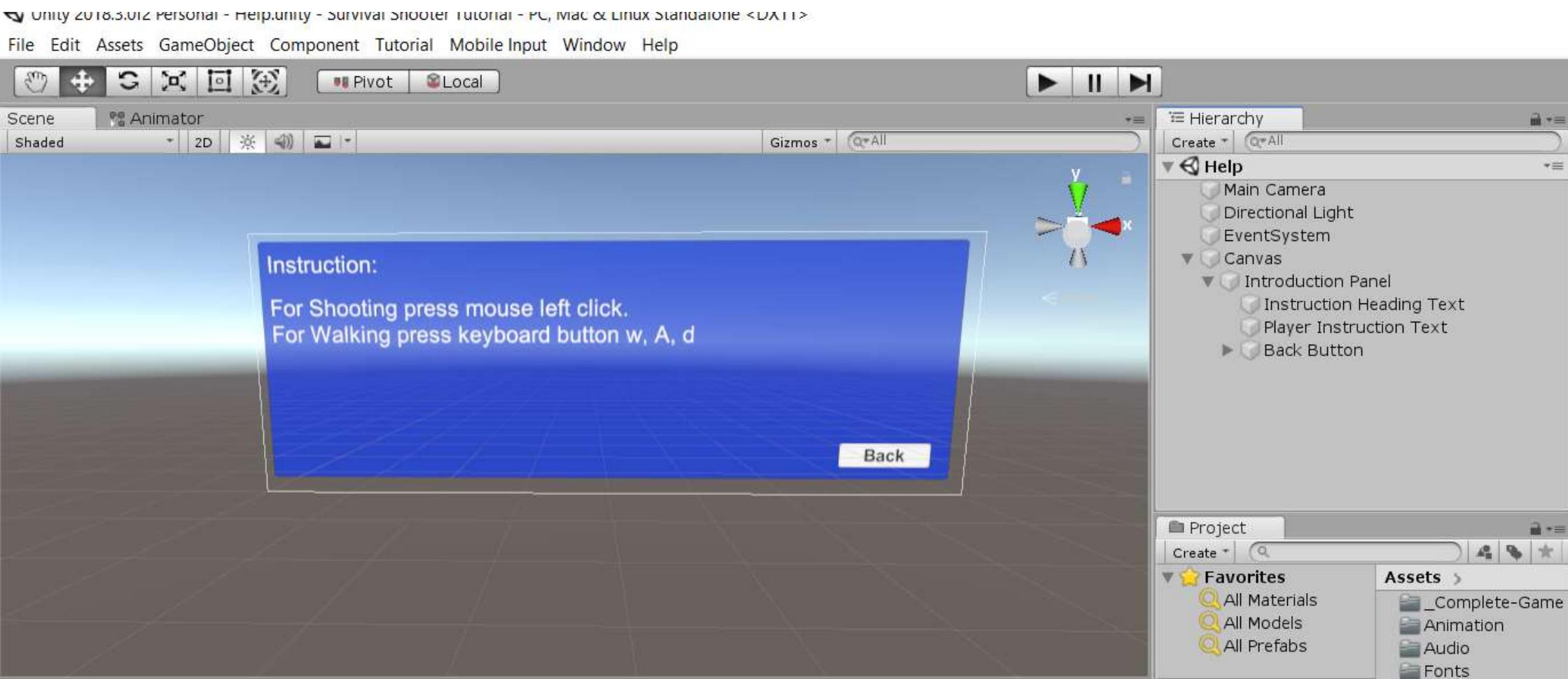
        int spawnPointIndex = Random.Range (0, spawnPoints.Length);

        Instantiate (enemy, spawnPoints[spawnPointIndex].position, spawnPoints[spawnPointIndex].rotation);
    }
}
```

Δημιουργία Game Over στο παιχνίδι μας. Όταν πεθαίνει ο παίχτης μας θέλουμε να μας εμφανίζει μια οθόνη που τελειώνει το παιχνίδι μας. Δημιουργούμε αρχικά το γραφικό μας περιβάλλον GameObject->UI->Text . Δημιουργούμε μια μπλε εμφάνιση και ένα text που λέει Game over . Επιπλέον θέλουμε κάθε φορά που πεθαίνει ο παίχτης μας να εμφανίζεται αυτή η εμφάνιση που δημιουργήσαμε και αυτό θα το πραγματοποιήσουμε με Animation. Window->Animation και δημιουργούμε αυτό το clip .Τέλος παίρνουμε από το Project μας το έτοιμο GameOverScript.cs

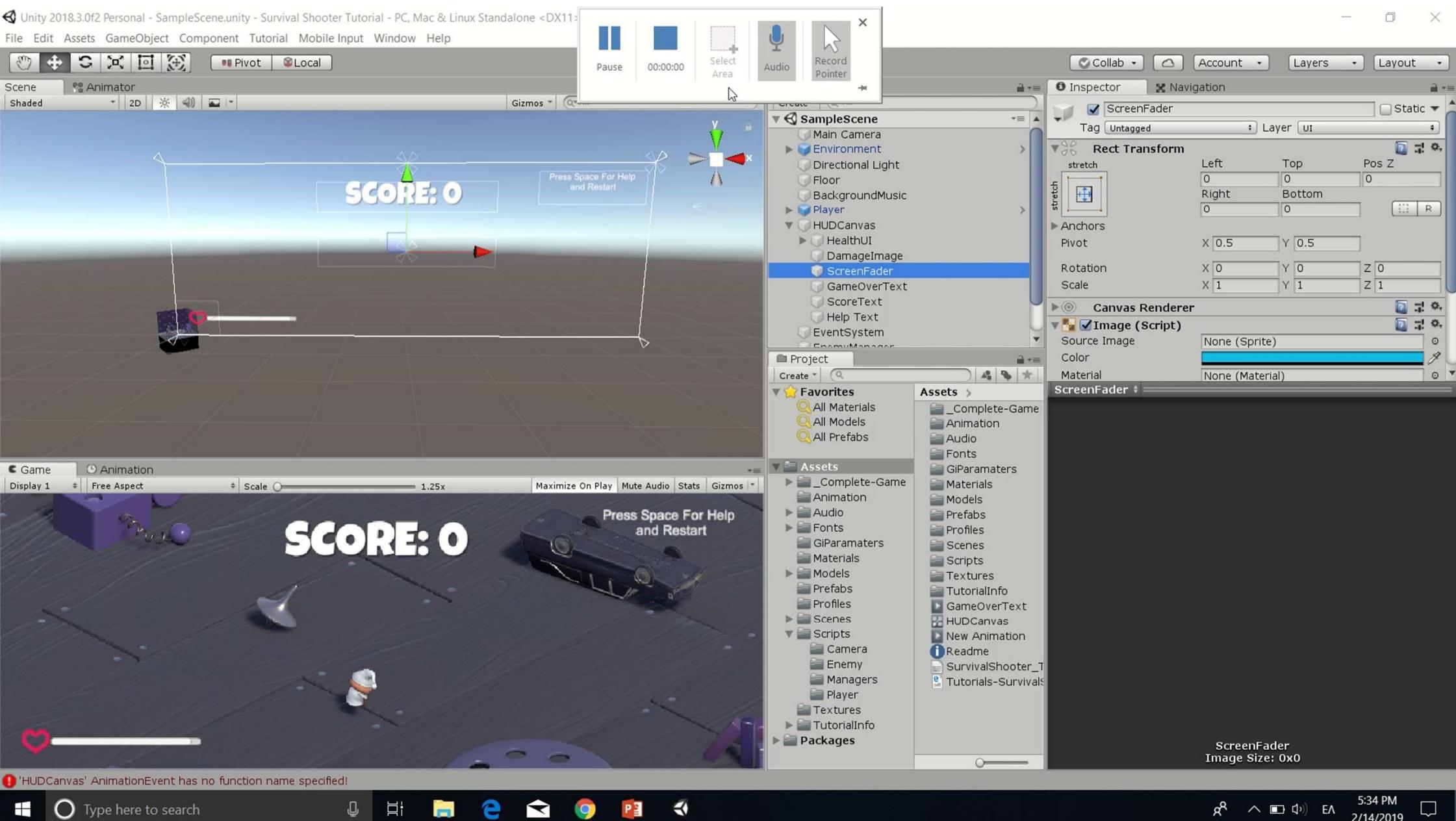


Δημιουργία Help Screen και Restart για τον παίχτη μας . Δημιουργούμε ένα νέο σκηνικό από File ->new Scene και το κάνουμε save σαν Help. Στο νέο σκηνικό επιλέγουμε GameOnject ->UI->Panel και στην συνέχεια επιλέγουμε να βάλουμε δύο text και ένα button για να επιστρέφουμε στο παιχνίδι μας .



# **Video Ηλεκτρονικού Παιχνιδιού**

# **Survival Shooting Game**



# Υλοποίηση δεύτερου tutorial : Space Shooter Tutorial

Αρχικά κατεβάζουμε τα έτοιμο project μας από την ιστοσελίδα της Unity . Και δημιουργούμε ένα νέο project το οποίο το ονομάζομε Space shooter Tutorial.

Tutorials > Space Shooter tutorial

# Space Shooter tutorial

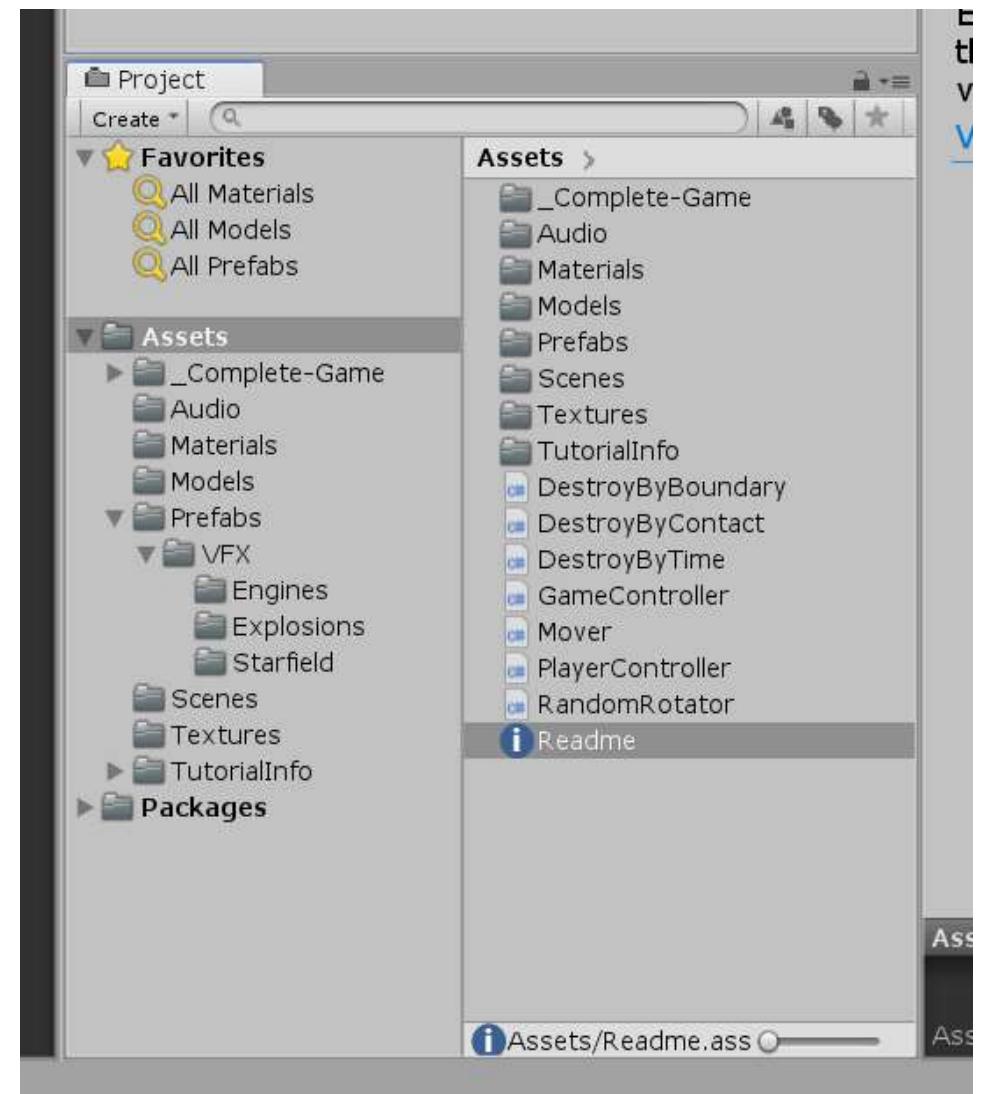
Expand your knowledge and experience of working with arcade style shooter.

Using basic assets provided by Unity Technologies, built-in components and with imported Mesh Models, Audio, Textures and Materials while practicin

Download the assets for free from the Asset Store [here](#).

 Download the Upgrade Guide to Unity 5 [here](#).

Questions? Ask in the official forum [thread here](#)

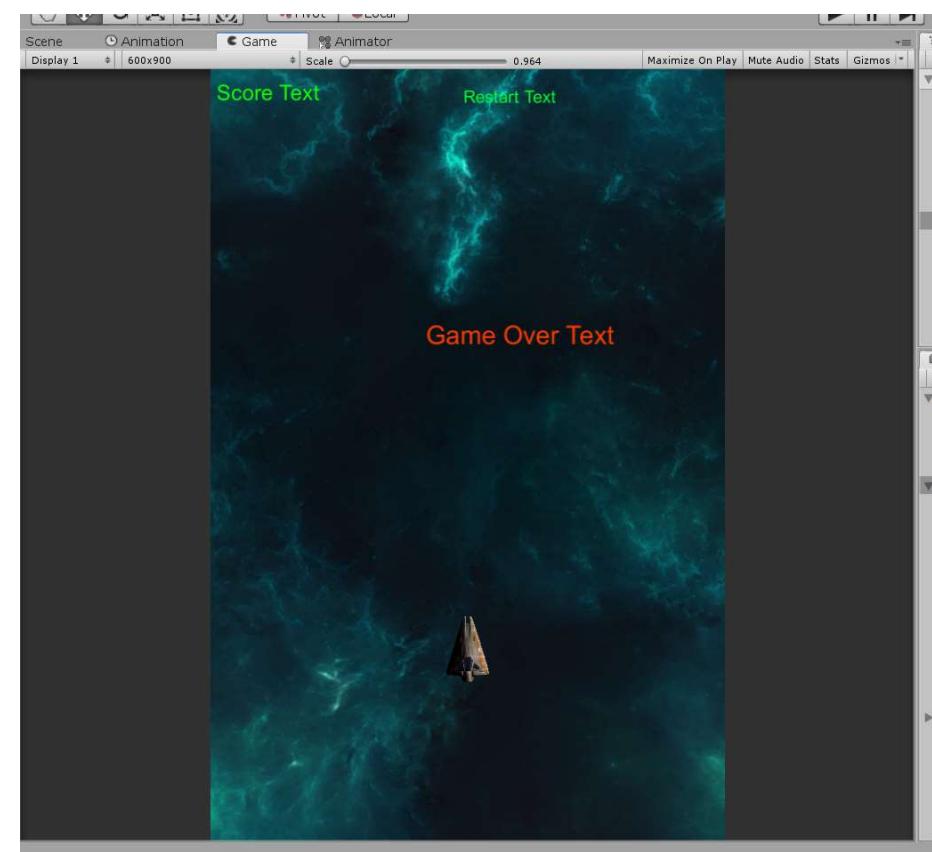
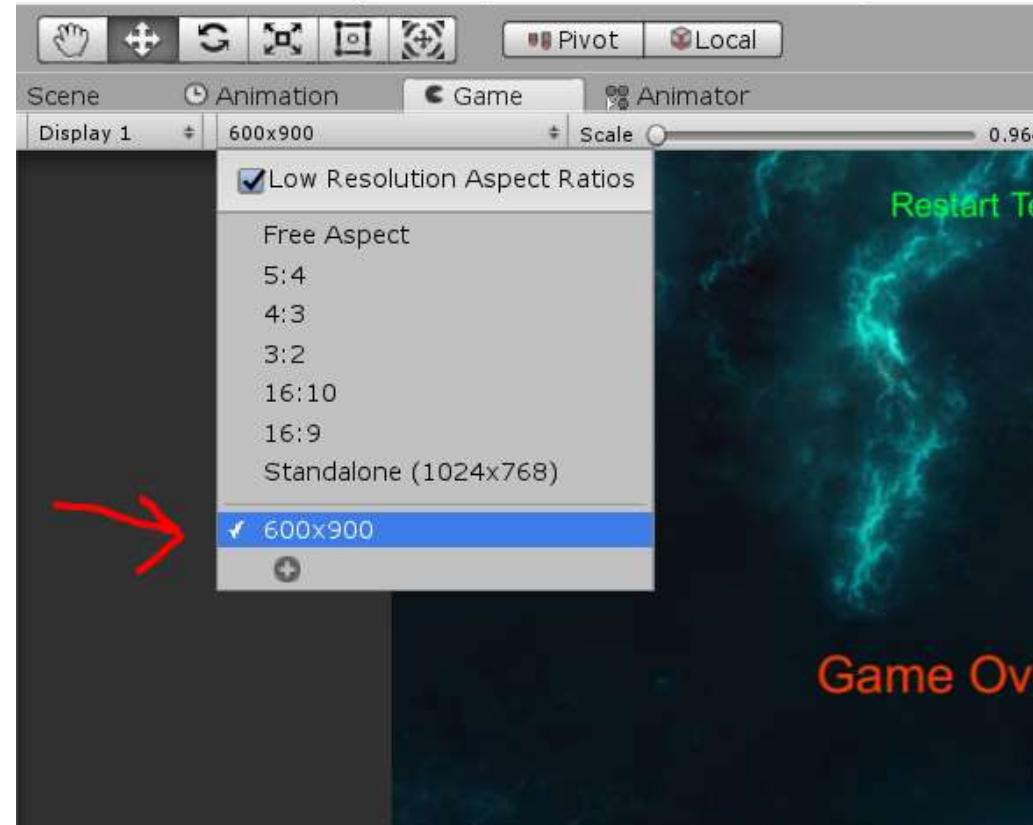


Αρχικά δημιουργούμε το περιβάλλον μας στο δικό μας project. Θέλουμε να βλέπουμε το παιχνίδι μας πανοραμικά σαν 2d παιχνίδι . Επιλέγουμε να αλλάξουμε την ανάλυση μας σε 600 x 900.

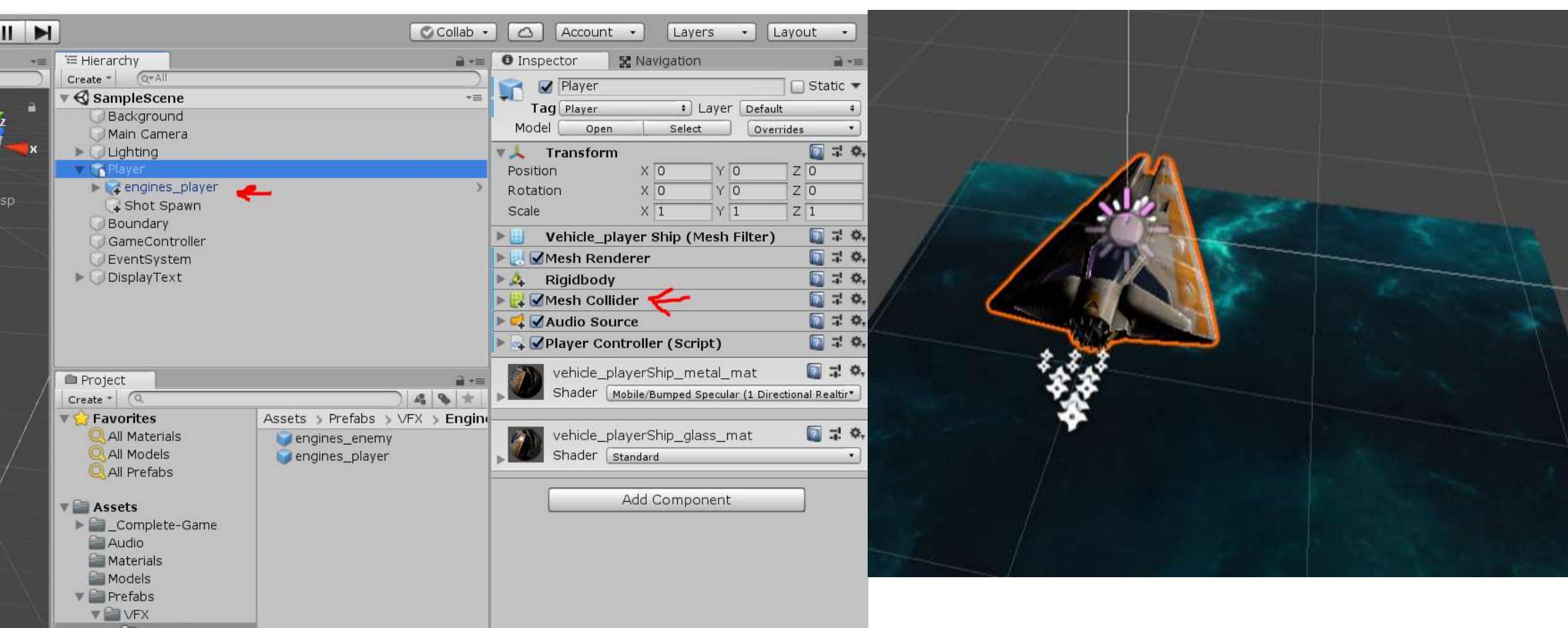
Game Window ->Free Aspect για να αλλάξουμε την ανάλυση μας.

Unity 2018.3.0f2 Personal - SampleScene.unity - Space Shooter Tutorial - PC, Mac & I

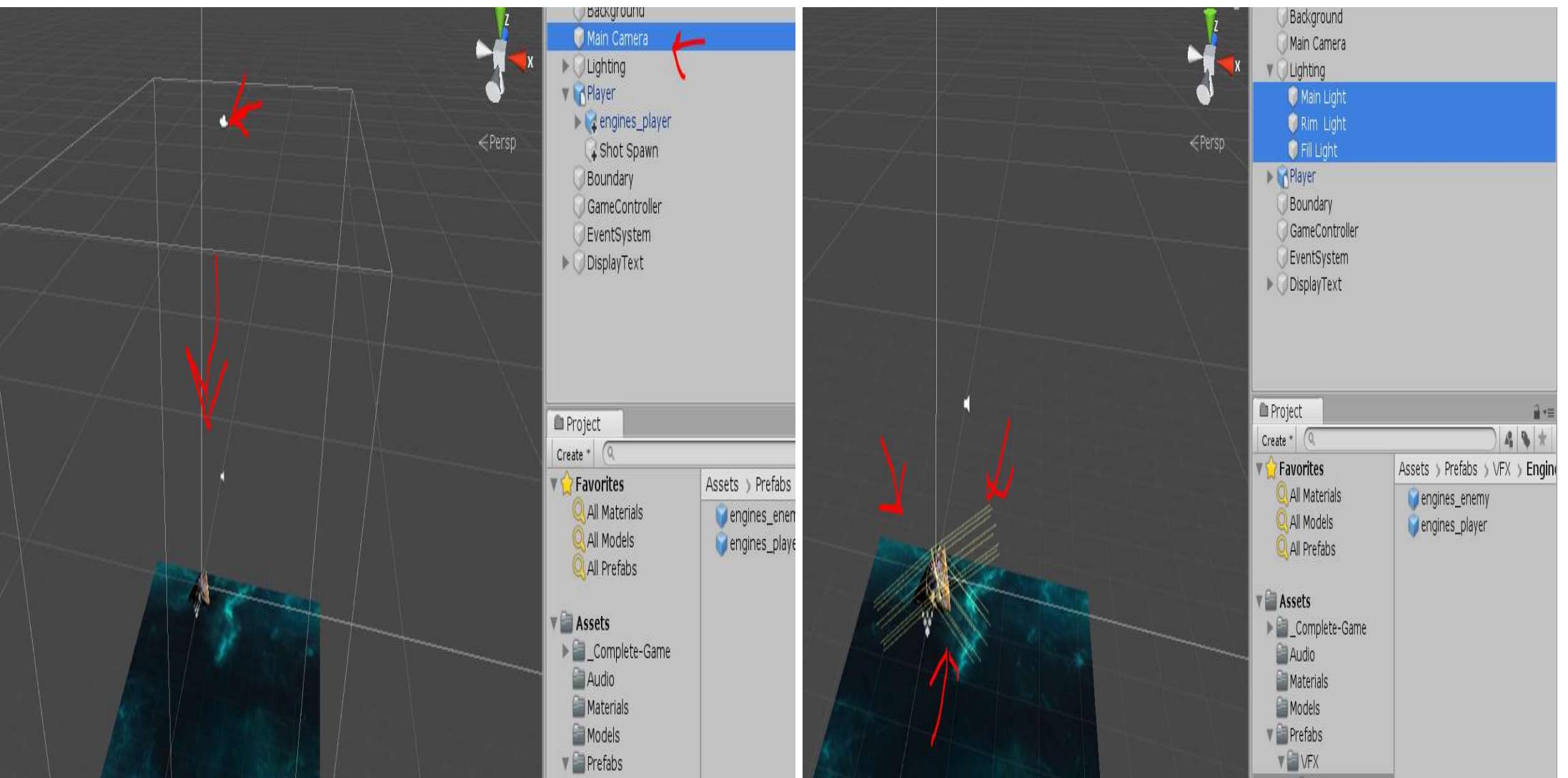
File Edit Assets GameObject Component Tutorial Window Help



Τοποθετούμε τον παίχτη μας Assets->Models και σέρνουμε το διαστημόπλοιο στο Hierarchy . Στην συνέχεια θέλουμε το διαστημόπλοιο μας να μπορεί να έρχεται σε επαφή με αντικείμενα . Άρα κάνουμε add Component Mesh collider και ρυθμίζουμε να παίρνει το ακριβές σχήμα από το διαστημόπλοιο και επιλέγουμε να γίνεται trigger δηλαδή να ενεργοποιείτε κατευθείαν στο παιχνίδι μας . Επιπλέον τοποθετούμε ένα έτοιμο prefab Assets->Prefabs->VFX->engines\_player.



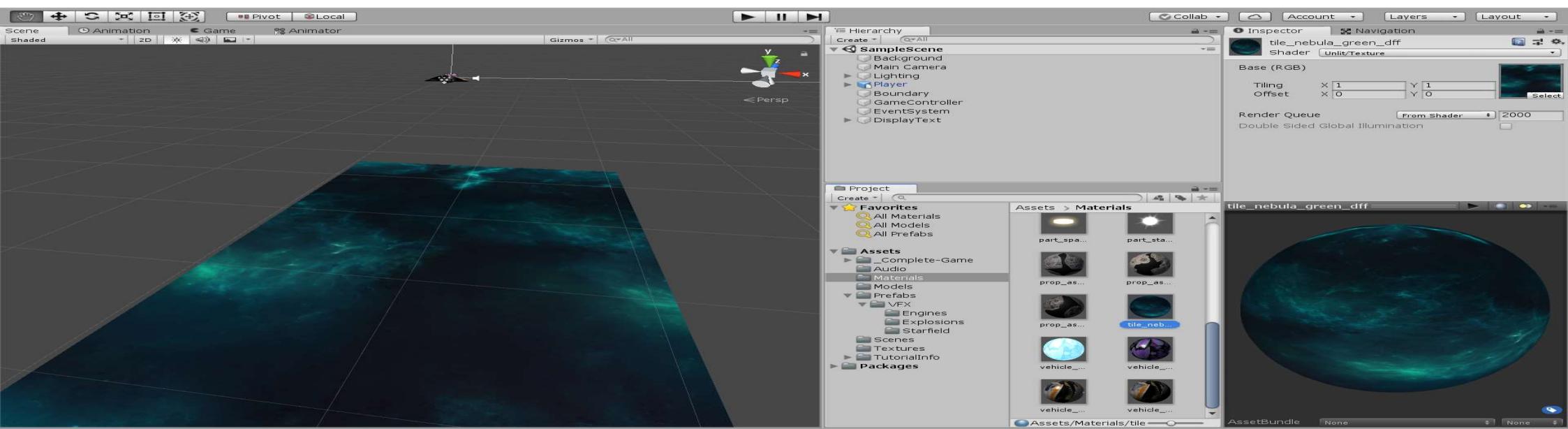
Ρυθμίζουμε τον φωτισμό και την κάμερα ως προ τον τρόπο που θα πέφτουν στο διαστημόπλοιο μας . Για τον φωτισμό έχουμε δημιουργήσει τρία διαφορετικά φώτα τα οποία χτυπάνε το διαστημόπλοιο.





Αυτό που θέλουμε ένα πέτυχουμε με τους τρείς διαφορετικούς φωτισμούς μας είναι το διαστημόπλοιο κατά την διάρκεια του παιχνιδιού να είναι σκοτεινό από την μια πλευρά και να πέφτει λίγο από την άλλη καθώς βρισκόμαστε στο διάστημα και το φως του ήλιου χτυπάει από μια γωνία .

Προσθήκη φόντου για το περιβάλλον του παιχνιδιού μας . Δημιουργούμε ένα quad και φτιάχνουμε ένα texture το οποίο κάνουμε προσθήκη πάνω στο quad μας το σχέδιο το παίρνουμε έτοιμο από το κατεβασμένο project. Επιπλέον προσαρμόζουμε το scale της πίστας μας.



Μετακίνηση του παίχτη μας στο περιβάλλον με το πληκτρολόγιο . Δημιουργούμε ένα c# script πάνω στον παίχτη μας με ονομασία PlayerController.cs. Στο όποιο έχουμε την κίνηση του παίχτη (διαστημόπλοιο) και επιπλέον θα ορίσουμε και τα όρια του παιχνιδιού με σκοπό να μην έχει την δυνατότητα να βγει εκτός του πλαισίου που εμφανίζεται στον παίχτη μας. Επίσης βάζουμε και την ταχύτητα με την οποία θα κινείται

```
public Rigidbody rb; ↖
public float speed;
public float tilt;
public Boundary boundary; ↖

void FixedUpdate()
{
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");

    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.velocity = movement * speed;

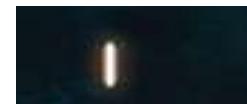
    rb.position = new Vector3
    (
        Mathf.Clamp(rb.position.x, boundary.xMin, boundary.xMax),
        0.0f,
        Mathf.Clamp(rb.position.z, boundary.zMin, boundary.zMax)
    );

    rb.rotation = Quaternion.Euler(0.0f, 0.0f, rb.velocity.x * -tilt);
}
```

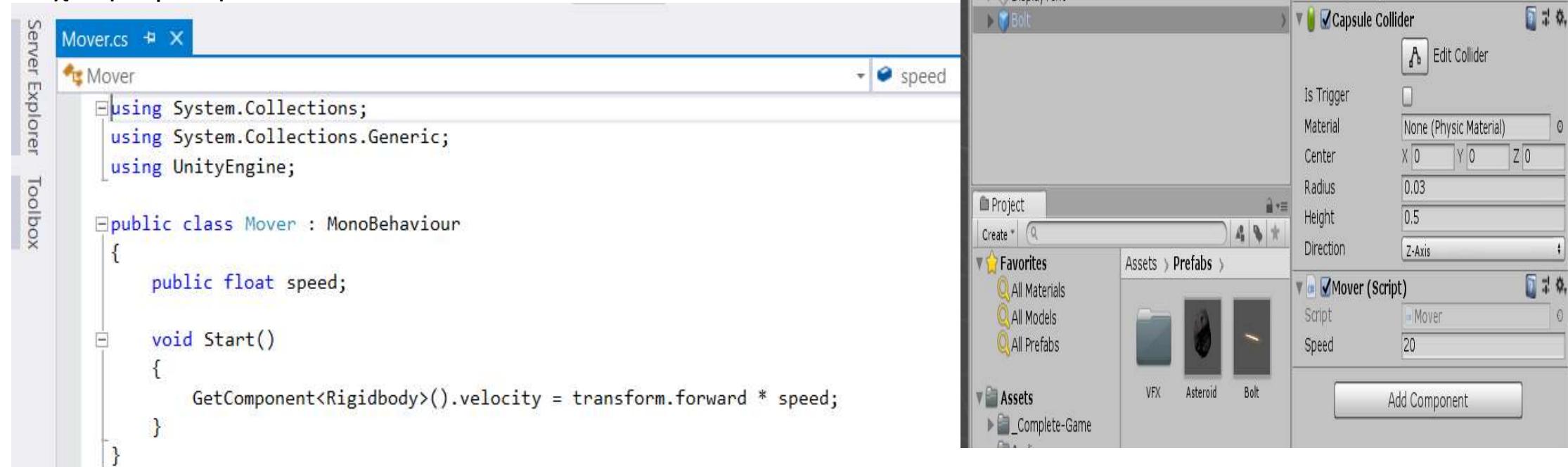


Χρησιμοποιούμε τον τύπο το Euler για να έχουμε μια ποιο ρεαλιστική κίνηση όταν πηγαίνουμε δεξιά και αριστερά τον παίχτη μας.

Δημιουργία επίθεσης στον παίχτη μας , με το να το να έχει την δυνατότητα να πυροβολεί . Δημιουργούμε το πρώτο πυροβολισμό τον οποίο θα χρησιμοποιήσουμε σαν prefab για όλους του πυροβολισμούς μας.  
Δημιουργούμε ένα texture από το project που εγκαταστήσαμε στην αρχή .



Δημιουργούμε το script mover με το οποίο θα μετακινείτε αυτή η ακτίνα μέσα στο παιχνίδι μας καθώς και την ταχύτητα με την οποία θα κινείται.



The screenshot shows the Unity Editor interface. On the left, the code editor displays the `Mover.cs` script:

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

public class Mover : MonoBehaviour
{
    public float speed;

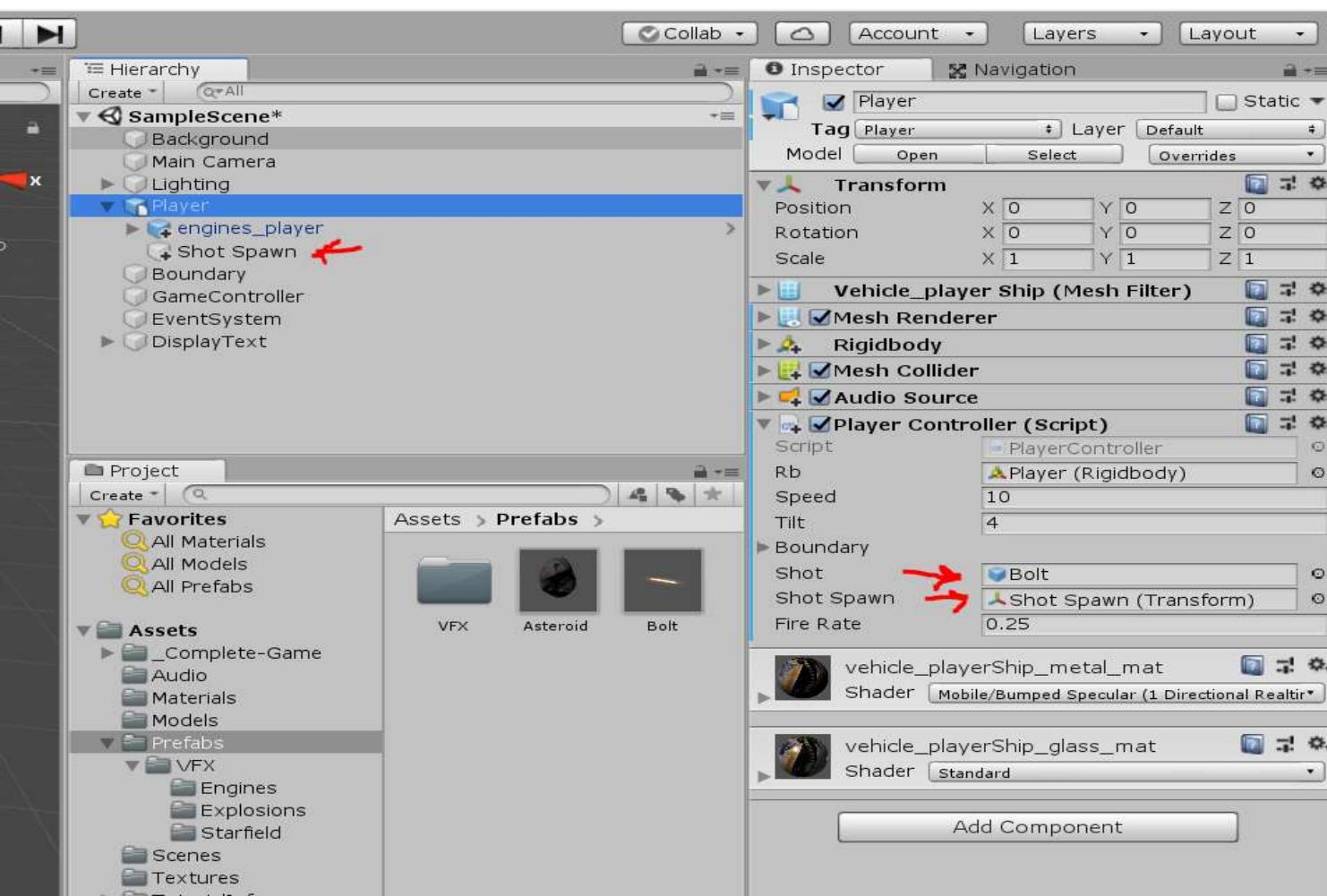
    void Start()
    {
        GetComponent<Rigidbody>().velocity = transform.forward * speed;
    }
}
```

The Inspector panel on the right shows the `Bolt` prefab configuration:

- Rigidbody**: Enabled
- Capsule Collider**: Enabled, with settings:
  - Is Trigger: Off
  - Material: None (Physic Material)
  - Center: X 0, Y 0, Z 0
  - Radius: 0.03
  - Height: 0.5
  - Direction: Z-Axis
- Mover (Script)**: Enabled, with settings:
  - Script: Mover
  - Speed: 20

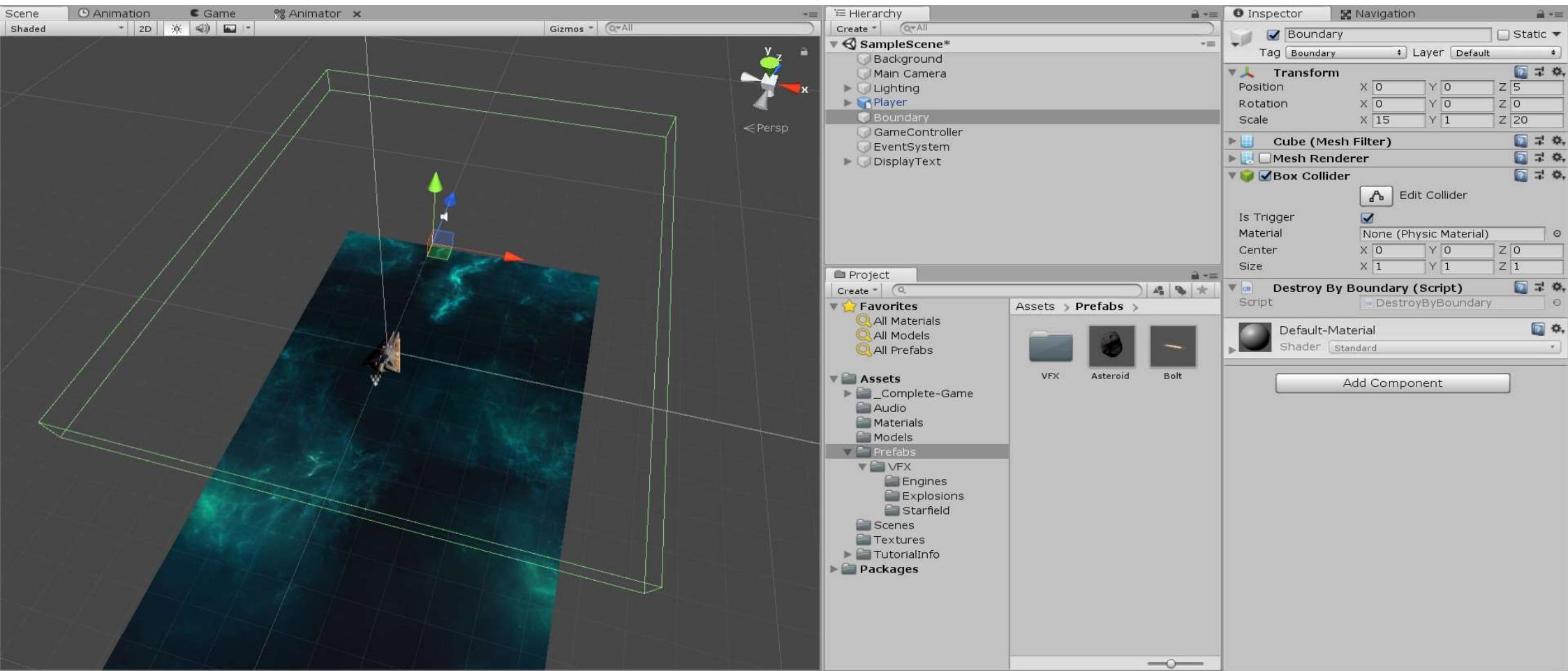
The Project panel at the bottom shows the `Bolt` prefab in the `Prefabs` folder.

Τοποθετούμε αυτόν τον πυροβολισμό μέσα στα Prefabs και θα δημιουργήσουμε το script μας για να μπορούμε να πυροβολήσουμε. Δημιουργούμε ένα Game Object το οποίο ονομάζουμε Shot spawn και θα παράγει τους πυροβολισμούς μας . Στο player Controller script κάνουμε add τον κώδικα μας ο οποίος θα δημιουργεί τους πυροβολισμούς μας .

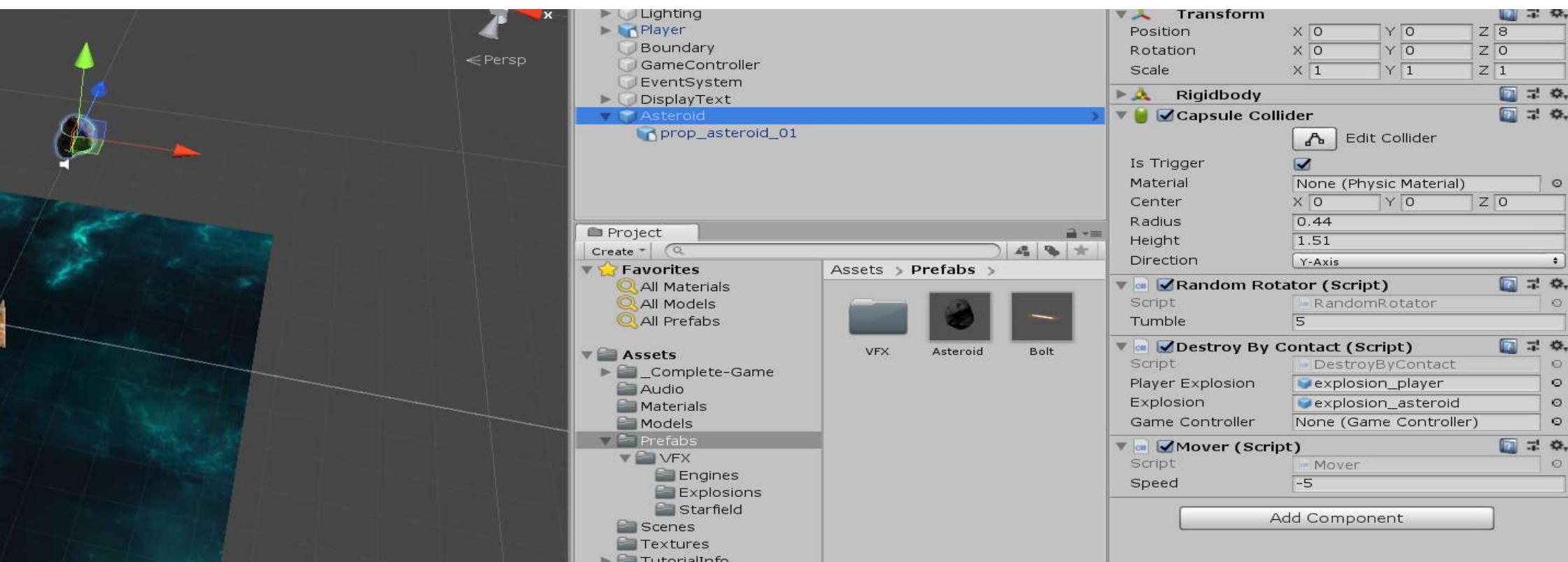


Το πρόβλημα που δημιουργείτε είναι ότι δημιουργούνται στο παιχνίδι μας πολλά GameObject . Αυτά αν δεν τα διαγράφουμε με κάποιο script θα δημιουργούνε κατά την πάροδο του παιχνιδιού μας lag και το παιχνίδι θα κολλάει. Οπότε στην συνέχεια θα δημιουργήσουμε ένα πλαίσιο στο οποίο οτιδήποτε διέρχεται μέσα από αυτό θα διαγράφη από το παιχνίδι μας. Αυτά θα είναι GameObject όπως η ακτίνες laser τις οποίες δημιουργήσαμε και οι μετεωρίτες που θα δημιουργήσουμε στην συνέχεια.

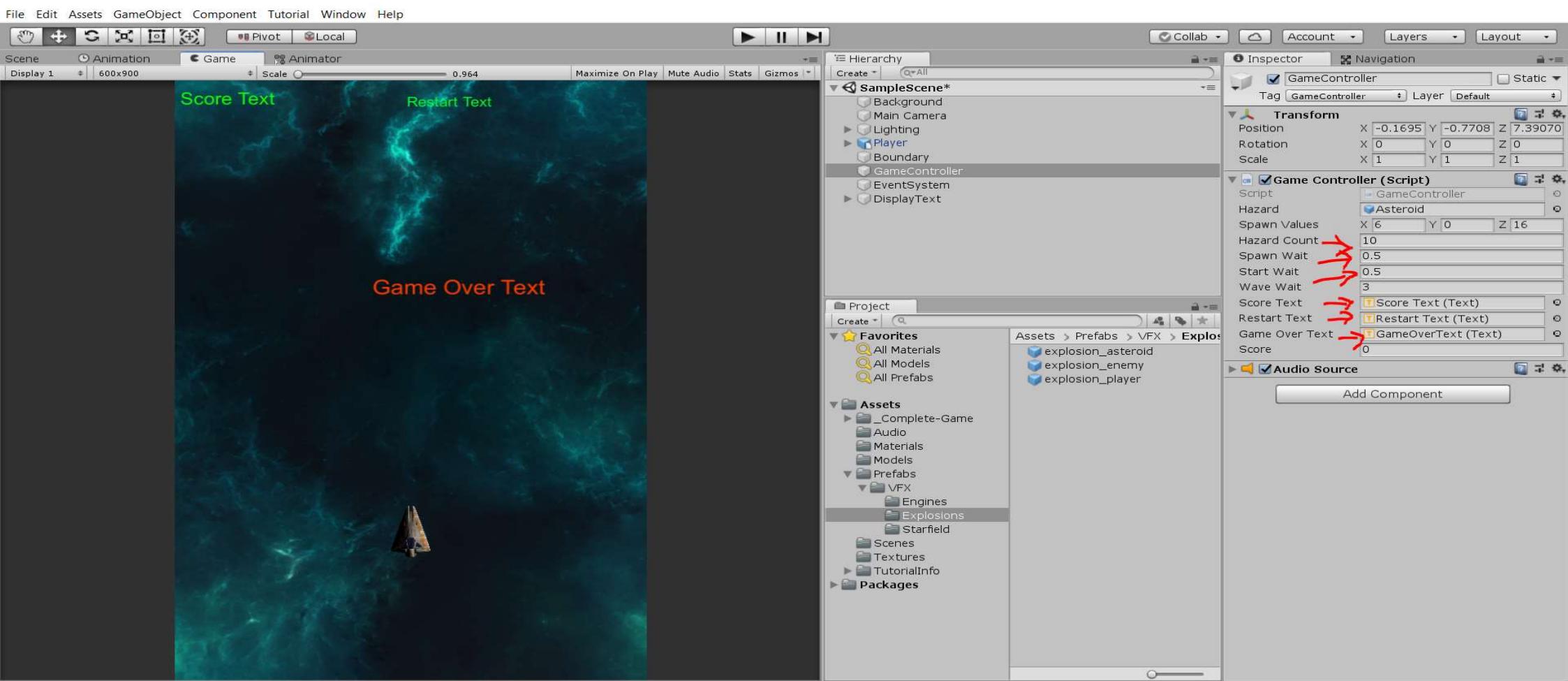
Οπότε δημιουργούμε το πλαίσιο μας που θα είναι αόρατο μέσα στο παιχνίδι μας αλλά οτιδήποτε διέρχεται μέσα από αυτό και είναι GameObject θα διαγραφεί . Δημιουργούμε το script Destroy By Boundary το οποίο διαγραφεί οποιοδήποτε GameObject περνάει από το πλαίσιο μας.



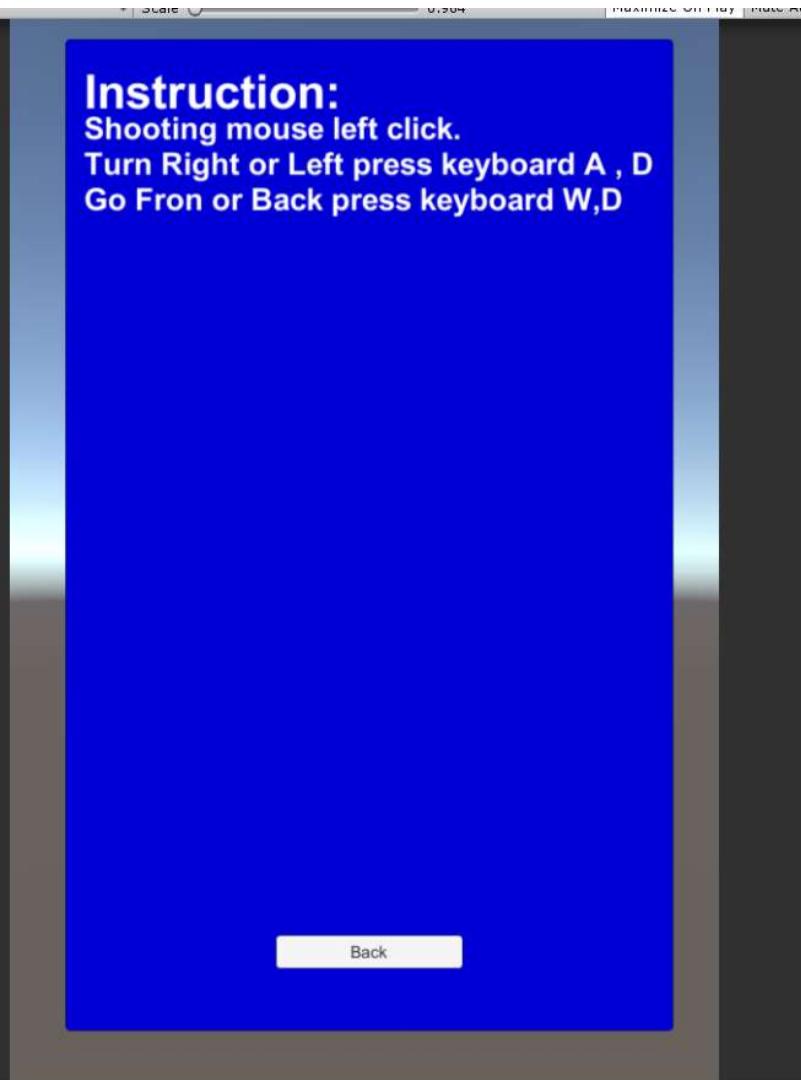
Δημιουργία μετεωρίτων για το παιχνίδι μας .Πηγαίνουμε Asset->Model Prop\_Asteroid . Δημιουργούμε όπως και προηγούμενος τον πρώτο μετεωρίτη μας και στην συνέχεια θα τον χρησιμοποιούμε από τα Prefabs σαν ένα έτοιμο αστεροειδή ο οποίος θα δημιουργείται πολλές φορές .Θέλουμε ο μετεωρίτης να περιστρέφεται . Δημιουργούμε το script μας random rotator με το οποίο θα περιστρέφεται ο μετεωρίτης μας . Στην συνέχεια θέλουμε ο μετεωρίτης να μετακινείτε οπότε δημιουργούμε το Mover script το οποίο δέχεται μια ταχύτητα .Η ταχύτητα του είναι αρνητική διότι θέλουμε να πηγαίνει προς το διαστημόπλοιο μας . Τέλος Δημιουργούμε το script Destroy By Contact έτσι ώστε όταν έρχεται σε επαφή με τον παίχτη να καταστρέφεται ο μετεωρίτης και ο παίχτης μας .Παίρνουμε από τα prefabs ->explosion και τοποθετούμε τις έτοιμες εκρήξεις στο παιχνίδι μας.



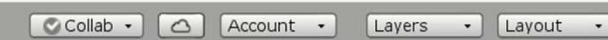
Δημιουργούμε ένα GameController το οποίο θα ρυθμίζει την ροή του παιχνιδιού μας. Δημιουργούμε τρία text τα οποία δείχνουν στο παιχνίδι μας το σκορ, το τέλος του παιχνιδιού και πώς να ξανά ξεκινήσουμε το παιχνίδι μας. Στο script GameController παίρνουμε αυτές τις Score Text και το κάνουμε update.



Δημιουργία Help Screen και Restart για τον παίχτη μας . Δημιουργούμε ένα νέο σκηνικό από File ->new Scene και το κάνουμε save σαν Help. Στο νέο σκηνικό επιλέγουμε GameObject ->UI->Panel και στην συνέχεια επιλέγουμε να βάλουμε δύο text και ένα button για να επιστρέφουμε στο παιχνίδι μας .



# **Video Ηλεκτρονικού Παιχνιδιού Space Shooter Tutorial**



Scene Animation Game Animator

Display 1 + 600x900 Scale 0.964 Maximize On Play Mute Audio Stats Gizmos



Hierarchy Create ⌂ All

SampleScene  
↳ Background  
↳ Main Camera  
↳ Lighting  
↳ Player  
↳ Boundary  
↳ GameController  
↳ EventSystem  
↳ DisplayText

Project Create

Favorites  
↳ All Materials  
↳ All Models  
↳ All Prefabs  
  
Assets  
↳ \_Complete-Game  
↳ Audio  
↳ Materials  
↳ Models  
↳ Prefabs  
↳ Scenes  
↳ Textures  
↳ TutorialInfo  
↳ Packages

\_Complete-Game  
Audio  
Materials  
Models  
Prefabs  
Scenes  
Textures  
TutorialInfo  
DestroyByBoundary  
DestroyByContact  
DestroyByTime  
GameController  
HelpScreenManager  
Mover  
PlayerController  
RandomRotator  
Readme

## Asset Labels

AssetBundle None

# Space Shooter Tutorial

## Follow this Tutorial

This is a tutorial project which has accompanying video instructions:

[Open the instructions in your browser](#)

## Get Support

Each tutorial project has a dedicated forum thread to accompany it. If you need help, visit the forum and ask!

[Visit the dedicated Forum Thread](#)



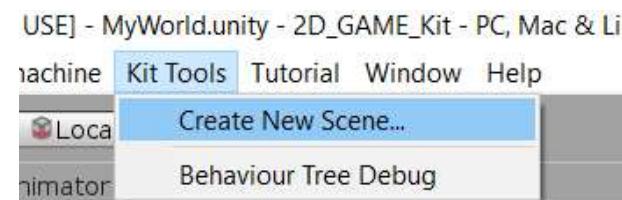
Type here to search



# Υλοποίηση δεύτερου tutorial : 2Game-Kit

Αρχικά κατεβάζουμε τα έτοιμο project μας από την ιστοσελίδα της Unity . Και δημιουργούμε ένα νέο project το οποίο το ονομάζομε 2D\_Game\_Kit.

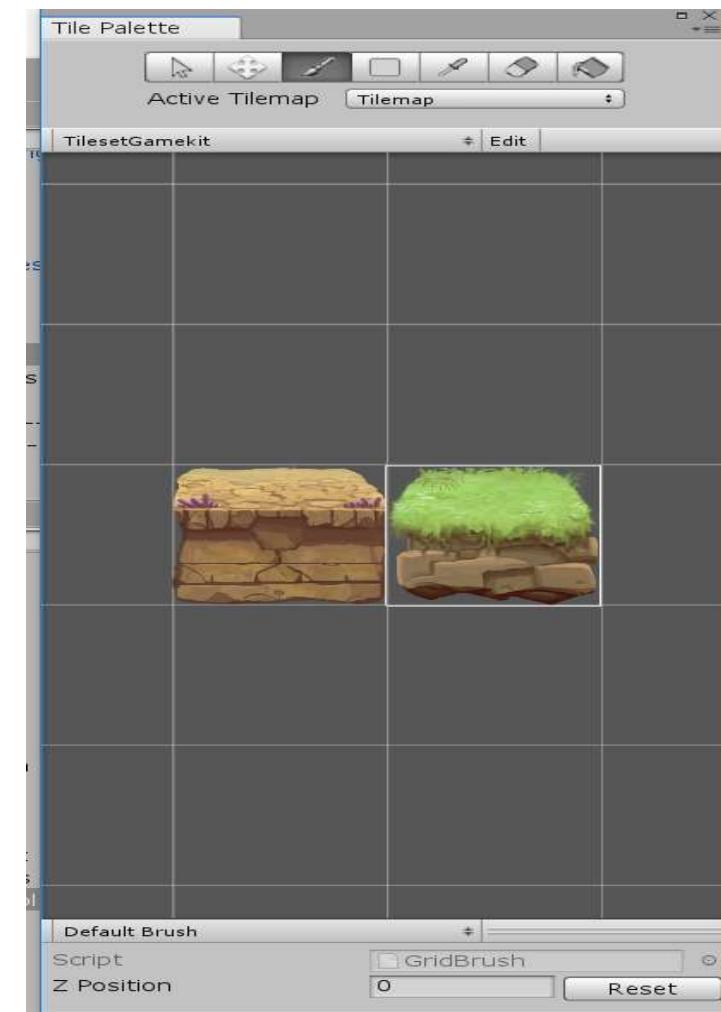
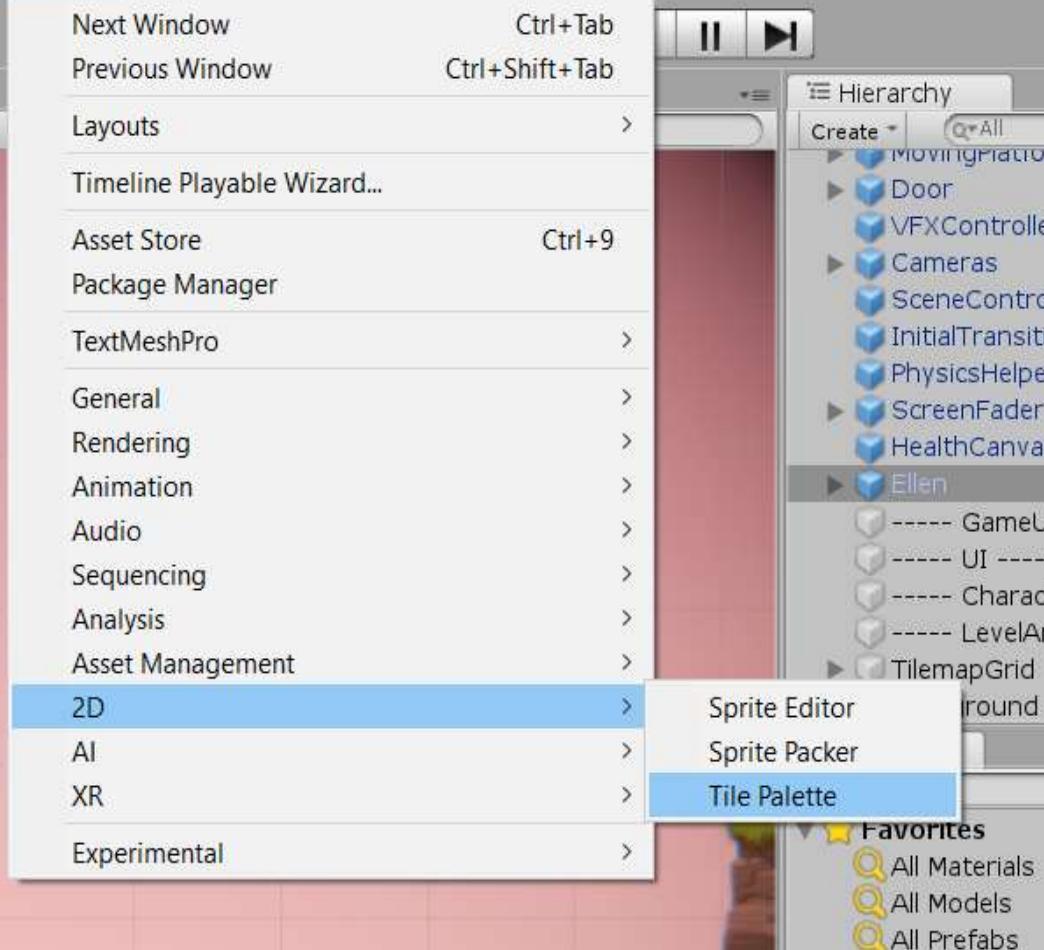
Δημιουργία νέου σκηνικού για την δημιουργία παιχνιδιού . Επιλέγουμε Kit tools->Create New Scene δίνουμε ονομασία στον νέο κόσμο μας **Myworld** . Δημιουργείτε ένα έτοιμο σκηνικό το οποίο θα επεκτείνουμε εμείς.



Στην Συνέχεια θέλουμε να αρχίζουμε να χτίζουμε την πίστα μας. Επιλέγουμε Window->2D->Tile Palete μπορούμε να επιλέξουμε μια από τις δύο επιλογές μας. Για να τοποθετήσουμε στο περιβάλλον μας.

- 2D\_GAME\_Kit - PC, Mac & Linux Standalone <DX11>

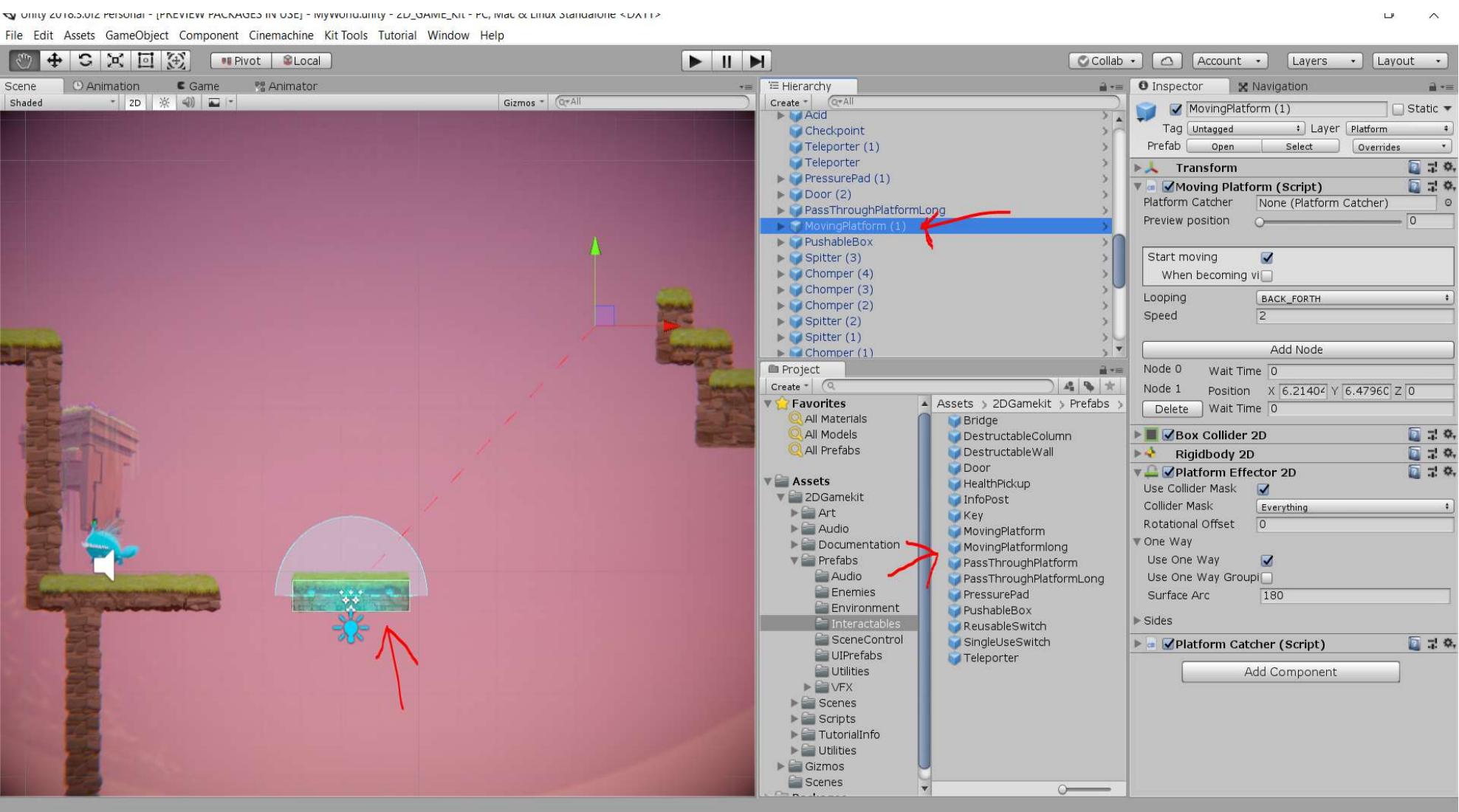
torial **Window** Help



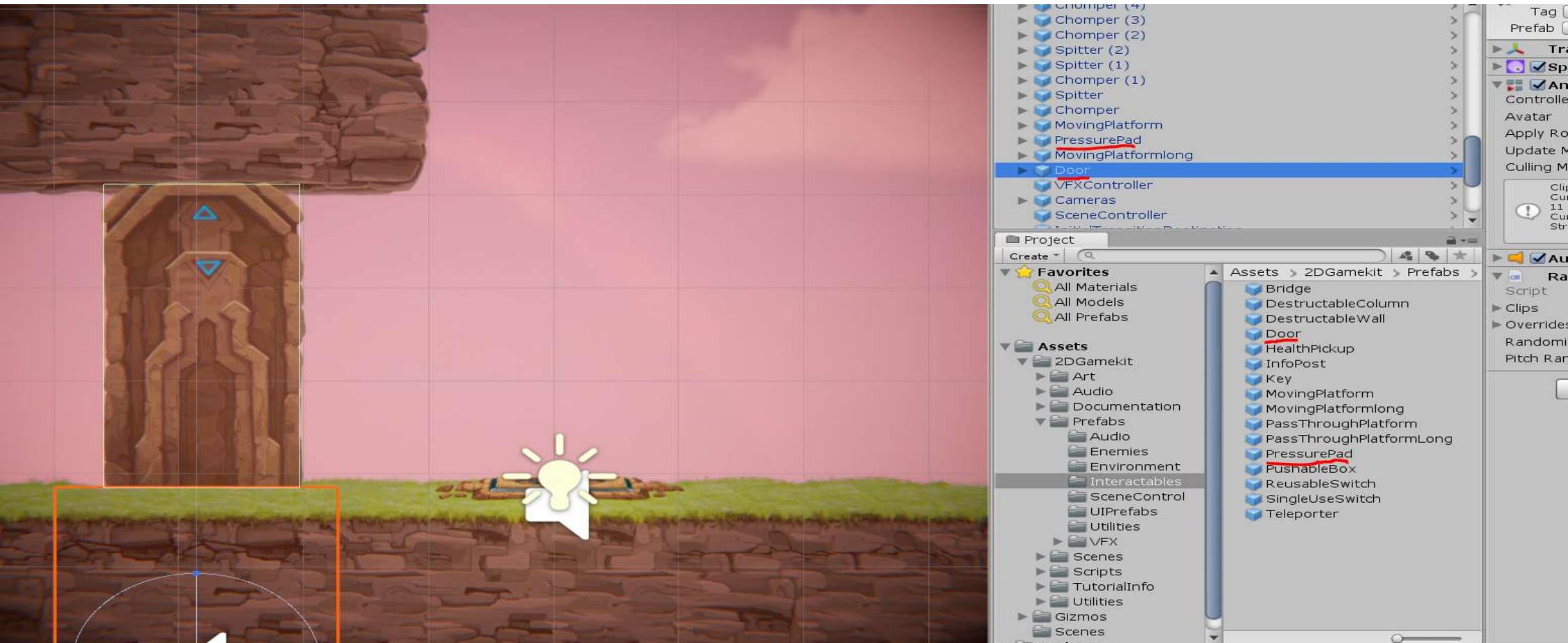
Όπως βλέπουμε έχω δημιουργήσει την πίστα μου στην οποία θα μετακινείτε ο player μου.



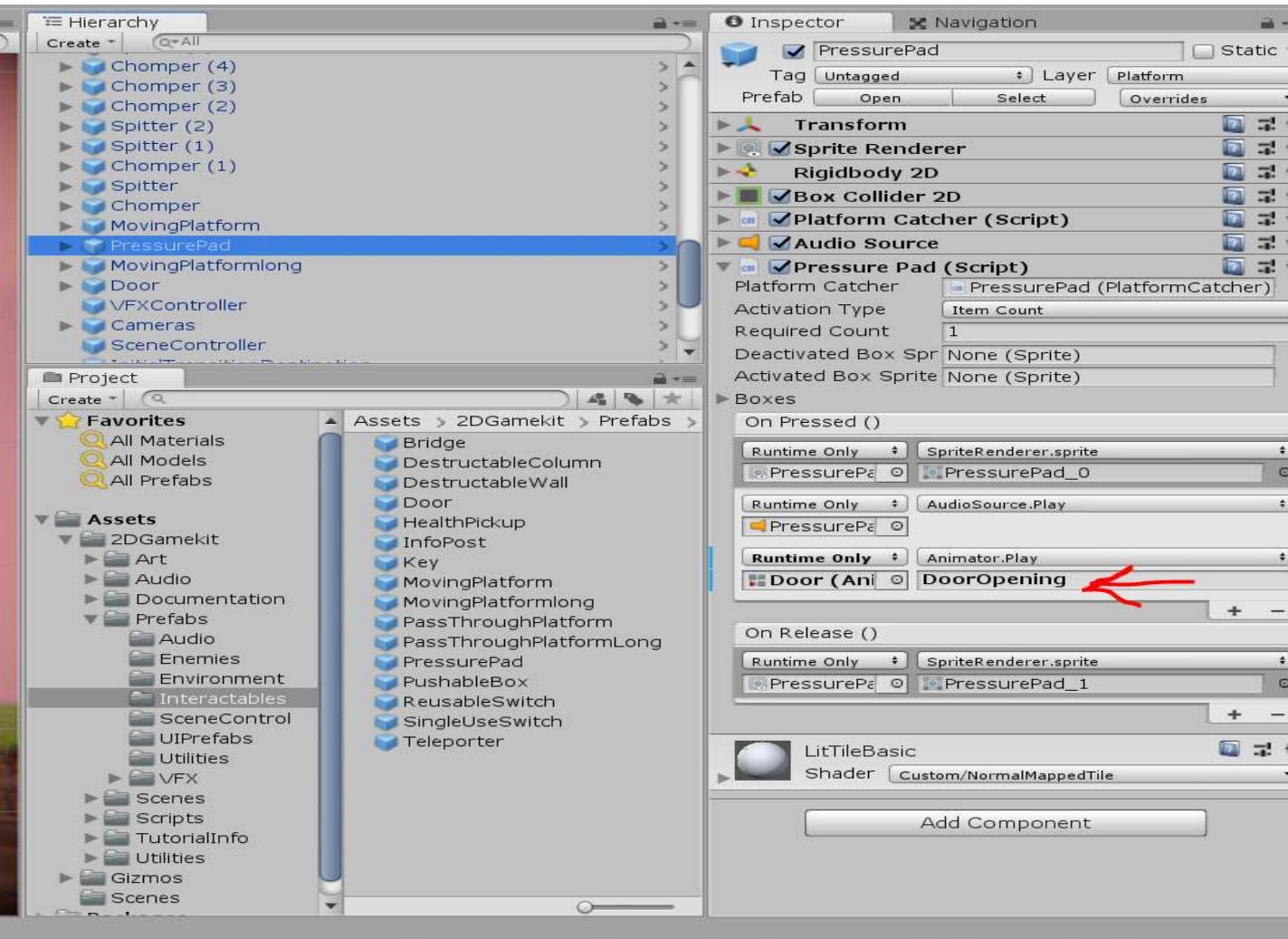
Δημιουργία εμποδίων μέσα στο παιχνίδι μας . Επιλέγουμε Assets->2DGamekit->MovingPlatform .Και επιλέγουμε πώς θέλει να μετακινείτε η πλατφόρμα μας .



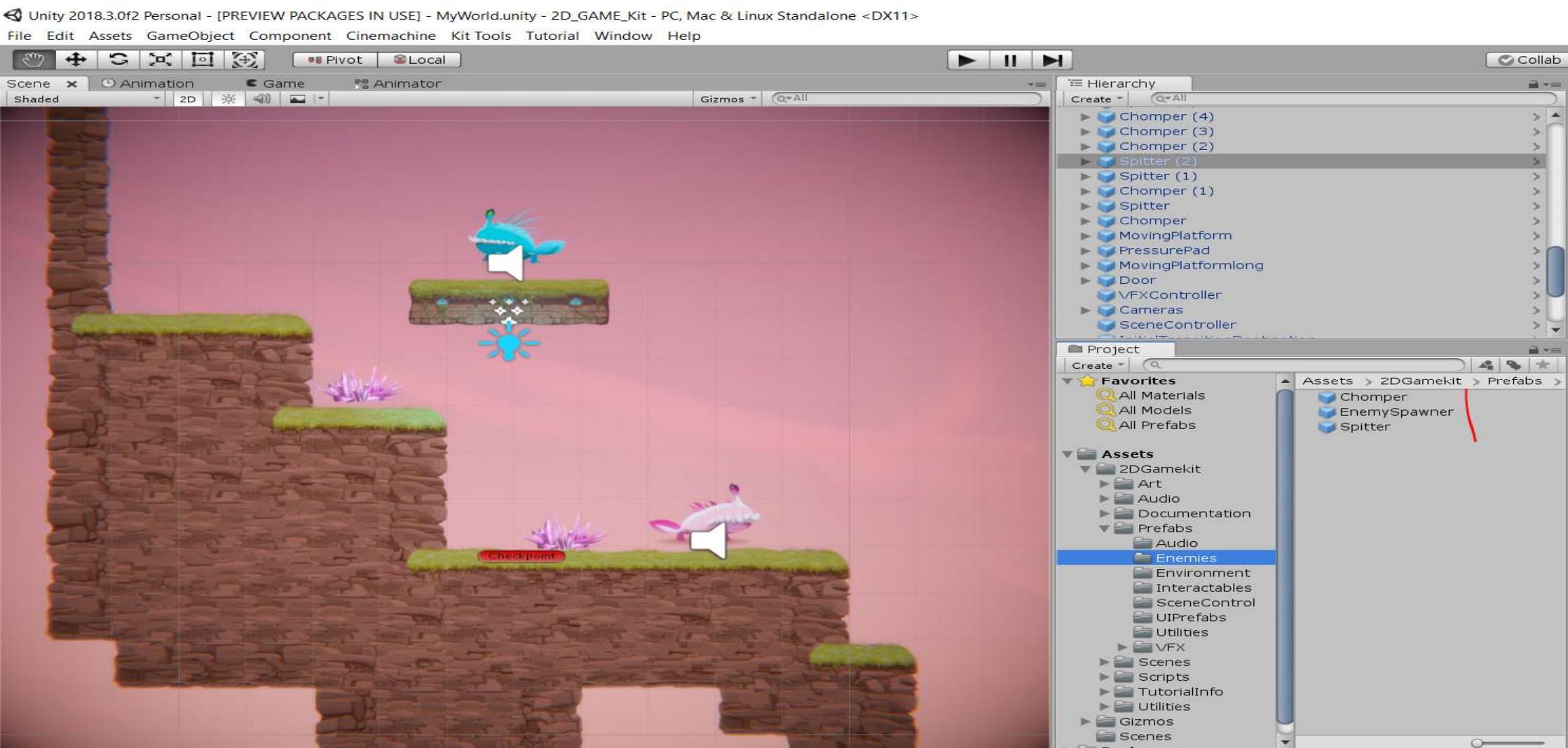
Δημιουργία εμποδίων μέσα στο παιχνίδι μας . Επιλέγουμε Assets->2DGamekit-> PressurePad και Door .Τα τοποθετούμε στο περιβάλλον μας το σημείο PrusserePad το οποίο ανοίγει την πόρτα μας.



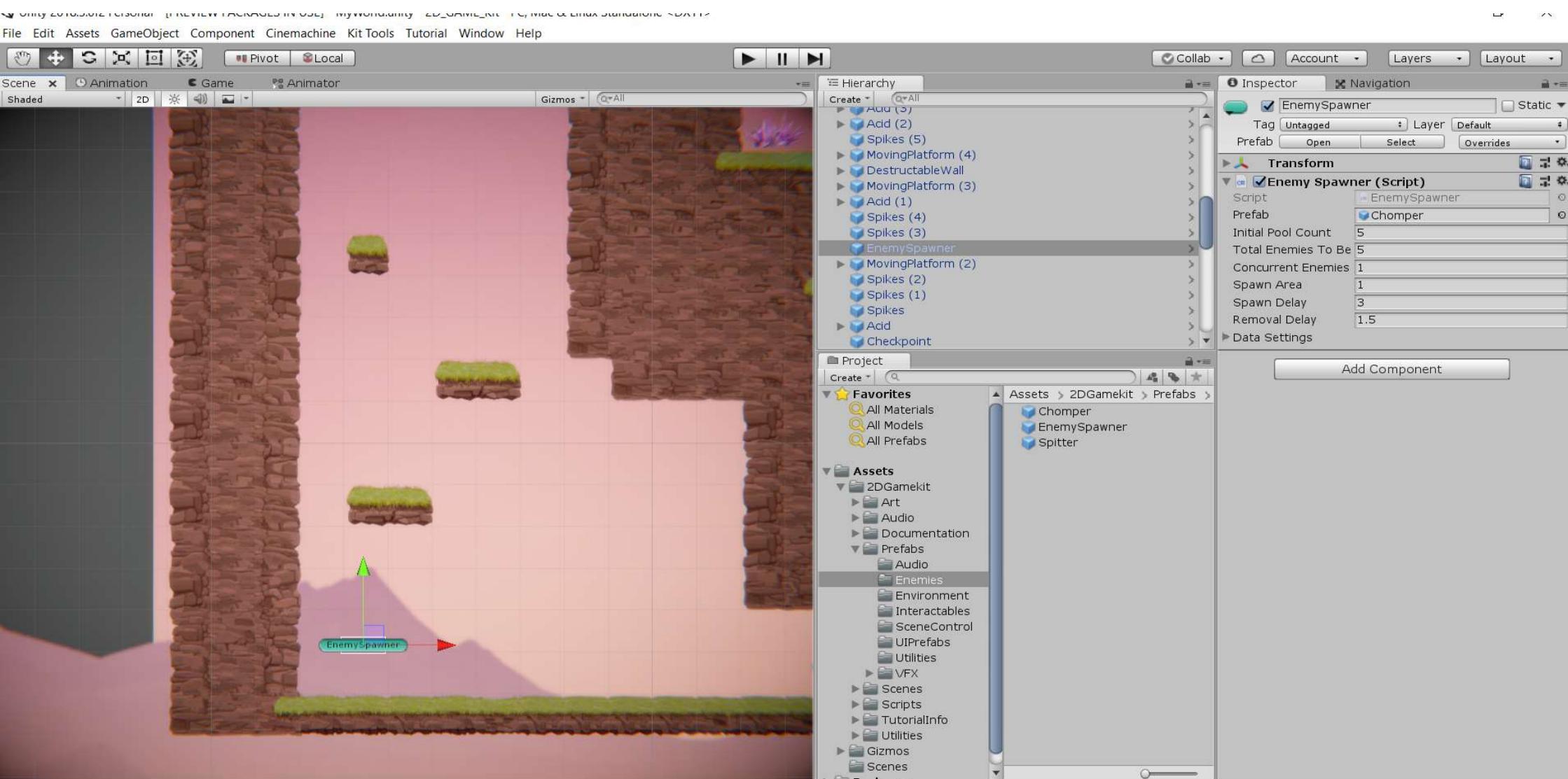
Ρυθμίζουμε τον τρόπο με τον οποίο θα ανοίγει η πόρτα μας πληκτρολογούμε την ονομασία της πόρτας μας και πληκτρολογούμε τι θέλουμε να κάνει **DoorOpening** να ανοίγει η πόρτα μας .



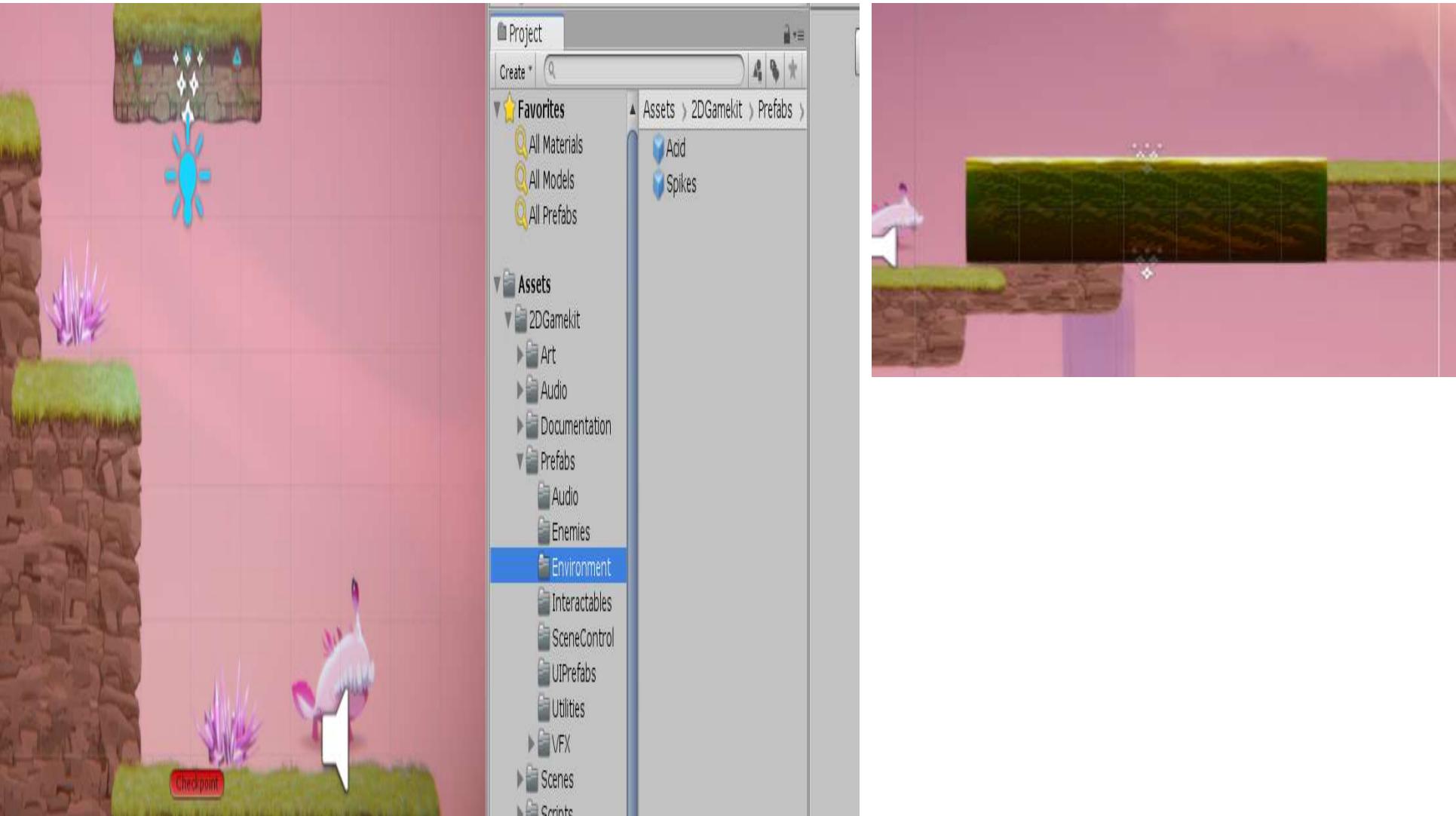
Δημιουργία εμποδίων μέσα στο παιχνίδι μας . Επιλέγουμε Assets->2DGamekit ->Prefabs Spitter, Chomper, EnemySpawner . Τα επιλέγουμε και τα τοποθετούμε στο περιβάλλον μας .



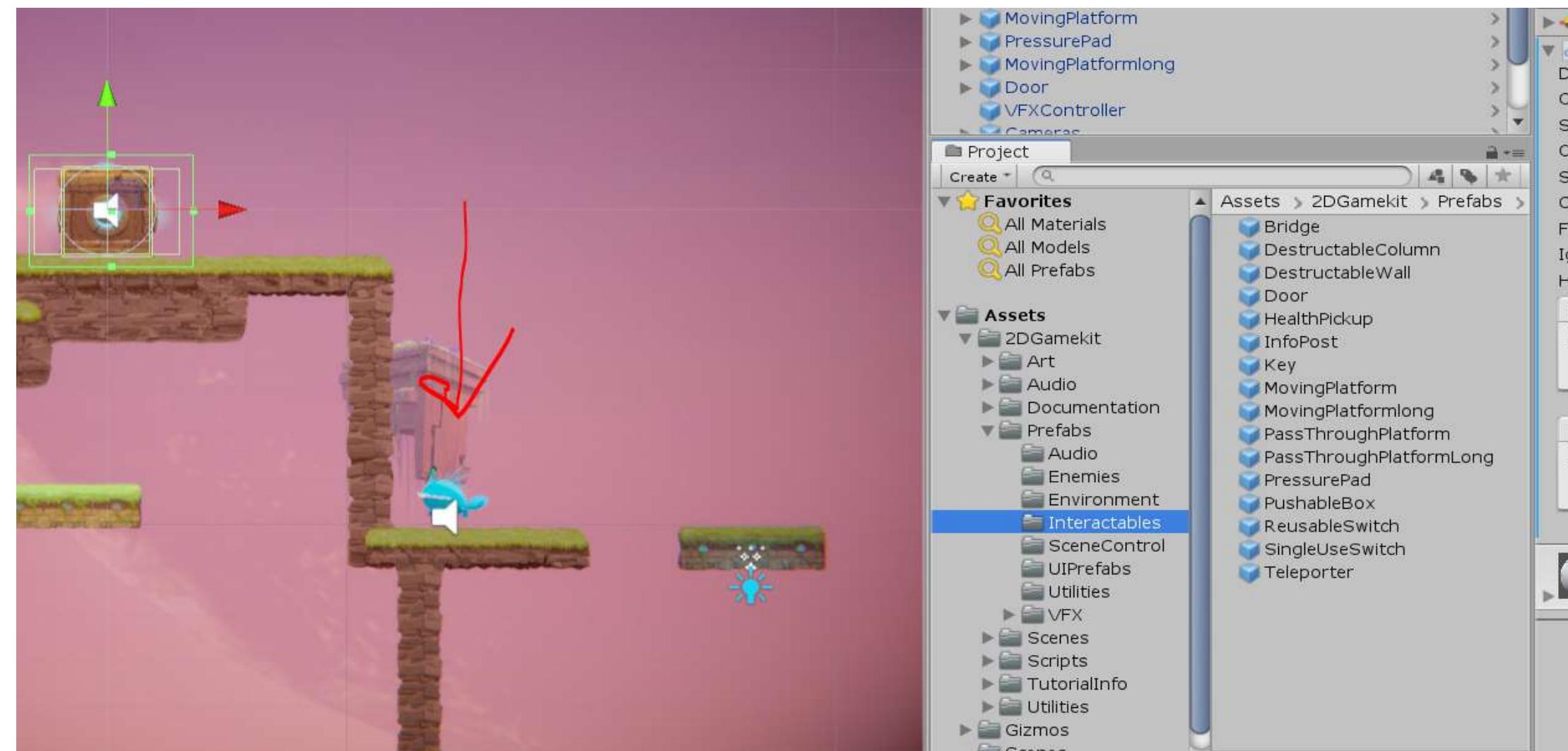
Για το EnemySpawner επιλέγουμε απλά ποιο από τις δύο επιλογές μας να δημιουργείτε Chomper ή Splitter.



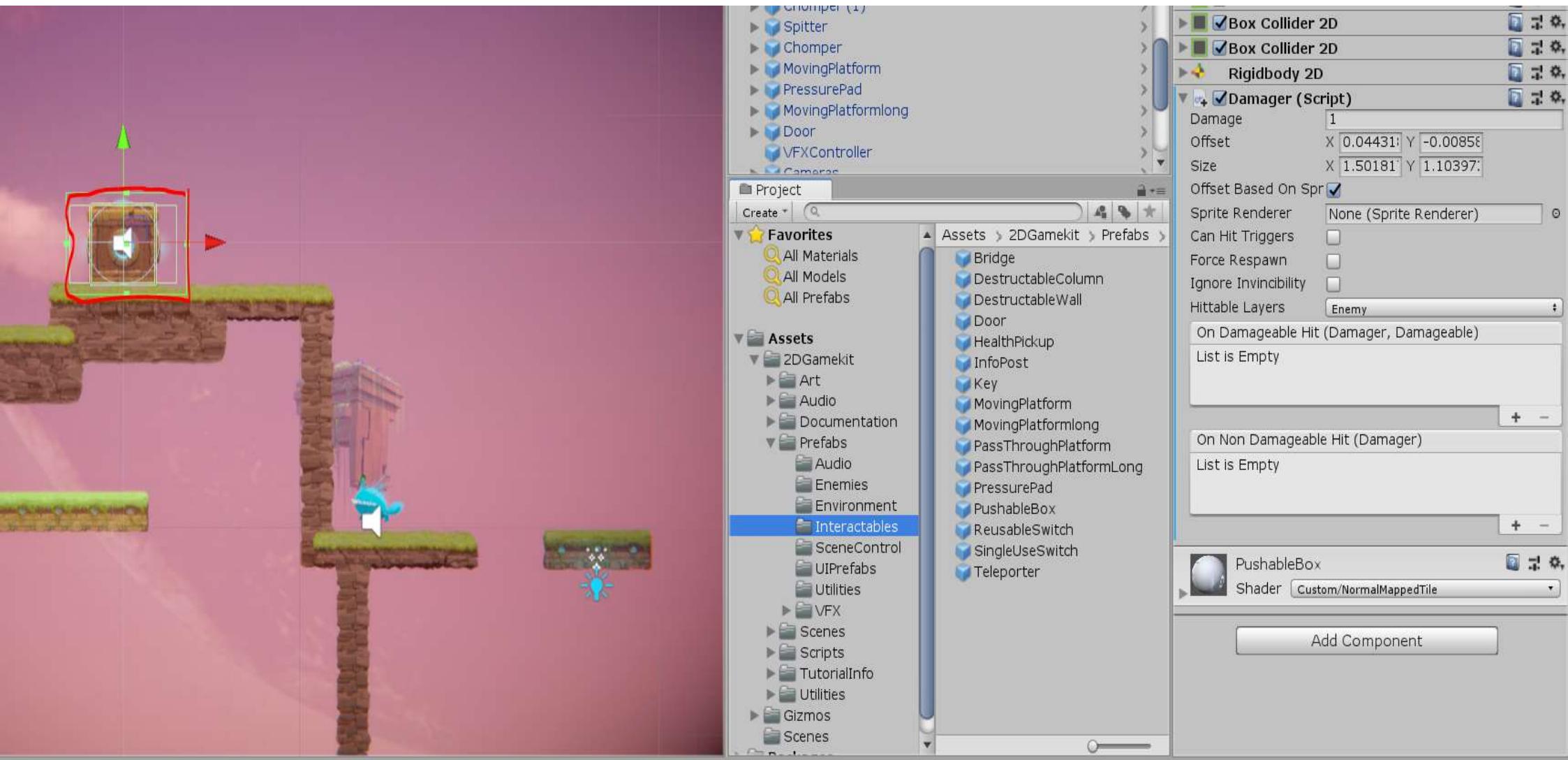
Επιλέγουμε Assets -> 2DGamekit->Prefabs ->Environment επιλέγουμε να τοποθετήσουμε όπου θέλουμε είτε το Acid είτε Spike . To spike είναι τα αγκάθια που βλέπουμε και για το Acid είναι ο βάλτος μας δεξιά .



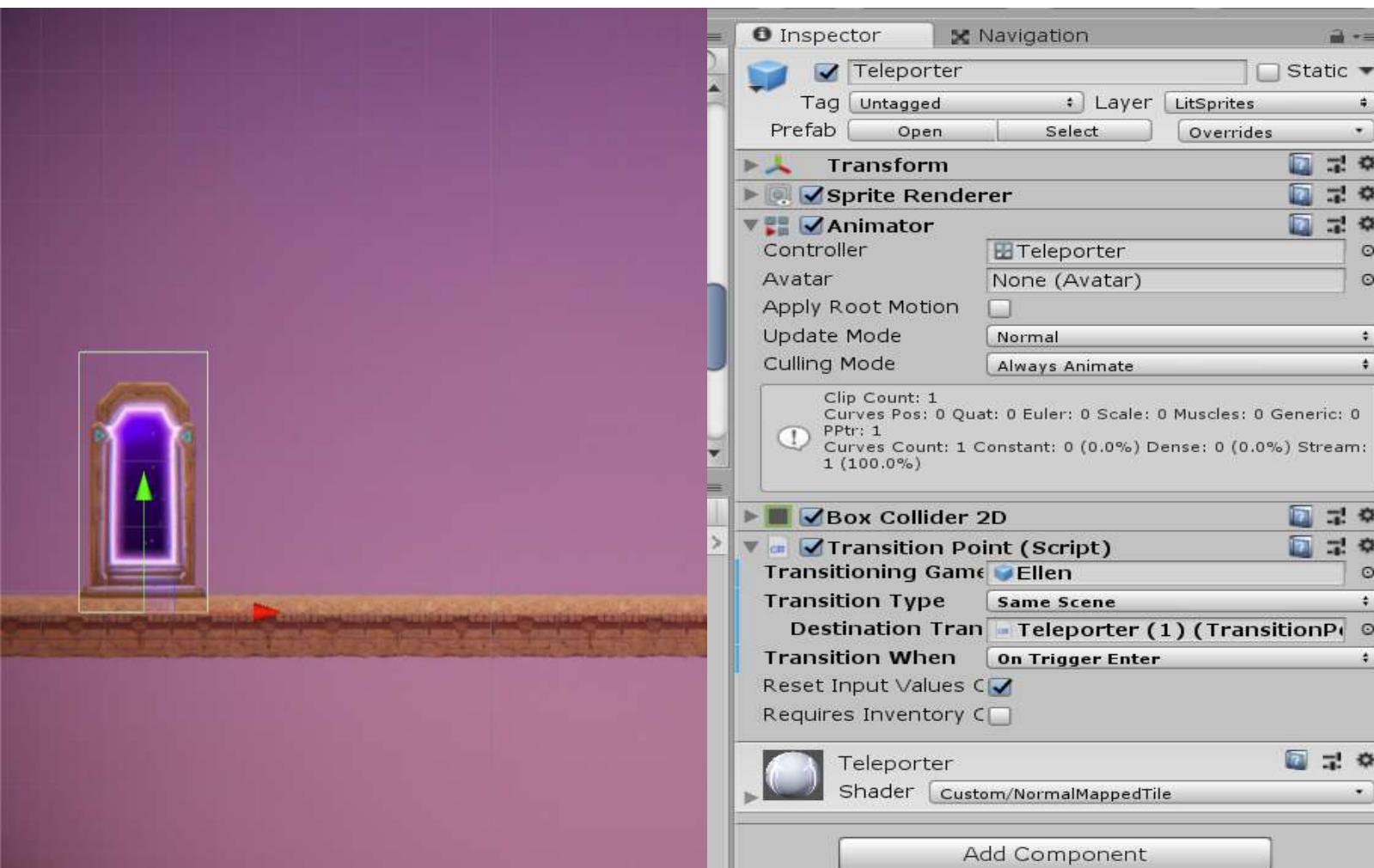
Επιλέγουμε Assets -> 2DGamekit->Prefabs ->Interactables επιλέγουμε PushableBox θέλουμε το κουτί μας να σκοτώνει τον αντίπαλο μας όταν το ρίχνουμε στον γκρεμό .



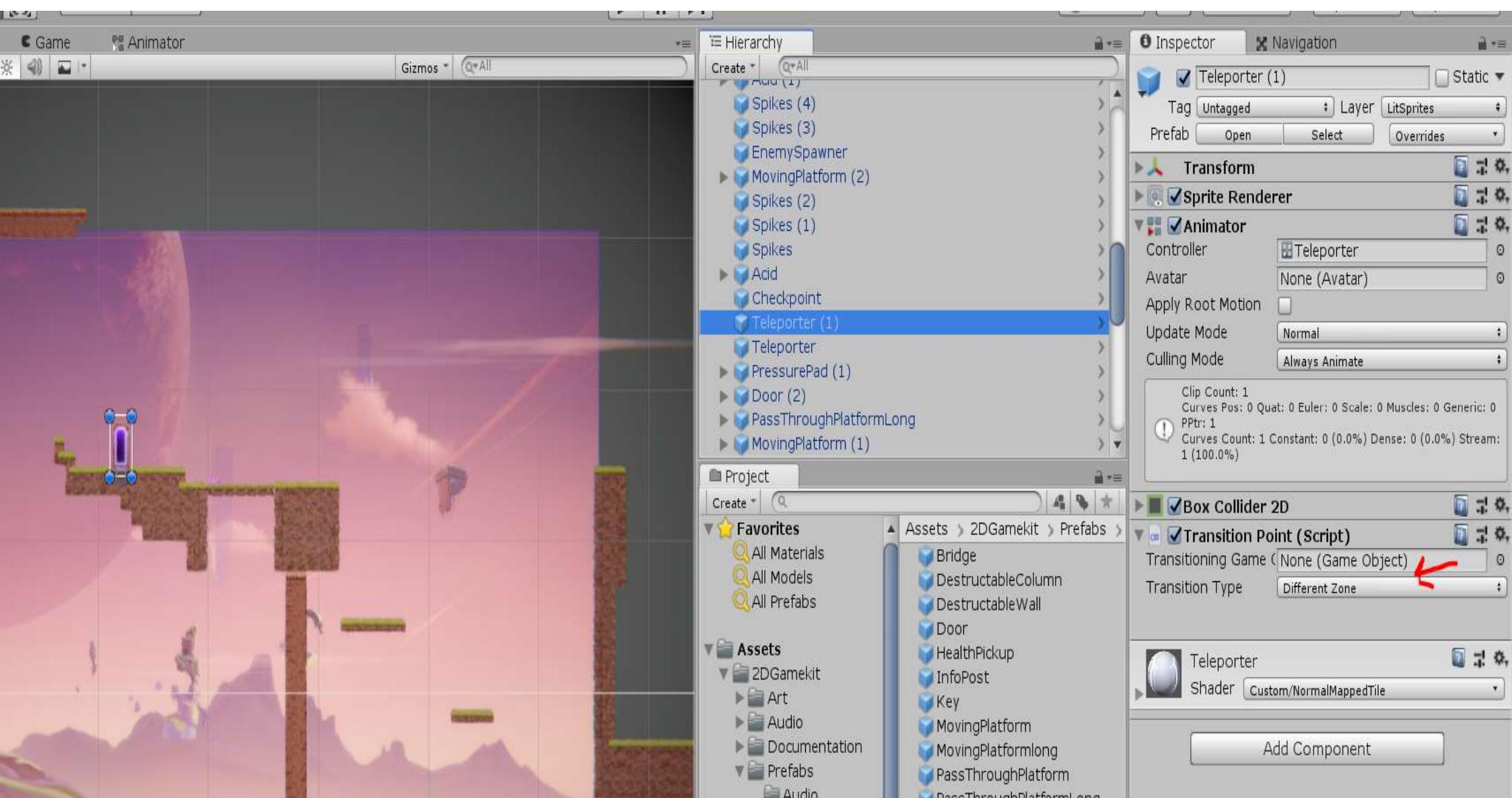
Το δεύτερο πλαίσιο που υπάρχει είναι για να ασκεί δύναμη στο Splitter μας.



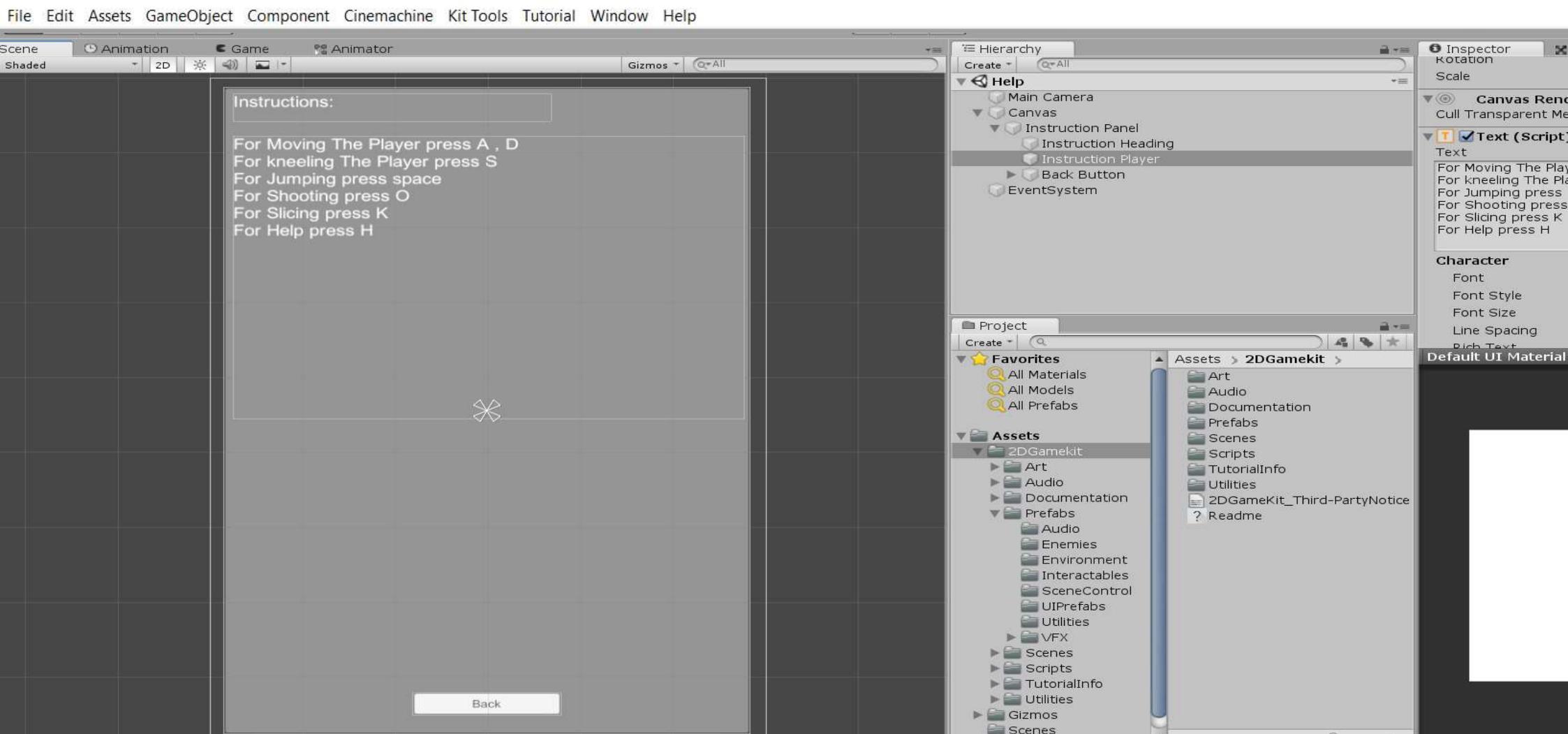
Επιλέγουμε Assets -> 2DGamekit->Prefabs ->Interactables επιλέγουμε Teleporter θα επιλέξουμε να τοποθετήσουμε σε δύο σημεία το teleporter μας όπου από το ένα θα εισέρχεται ο player και από το άλλο θα βγαίνει. Όπως βλέπουμε επιλέγουμε να περνάει από την πρώτη πύλη η Hellen και να πηγαίνει στην δεύτερη πύλη μας **Teleporter(1)** προορισμός.



Επειδή δεν επιθυμούμε να ξανά γυρνάει δεν βάζουμε τίποτα στην δεύτερη πύλη μας



Δημιουργία Help Screen και Restart για τον παίχτη μας . Δημιουργούμε ένα νέο σκηνικό από File ->new Scene και το κάνουμε save σαν Help. Στο νέο σκηνικό επιλέγουμε GameObject ->UI->Panel και στην συνέχεια επιλέγουμε να βάλουμε δύο text και ένα button για να επιστρέφουμε στο παιχνίδι μας .



# **Video Ηλεκτρονικού Παιχνιδιού**

## **2D\_GAME\_Kit**



Scene Animation Game Animator

Shaded 2D

Gizmos All

Hierarchy Create &gt; All

MyWorld

----- Gameplay -----  
Checkpoint (2)  
Acid (14)  
Acid (13)  
Acid (12)  
Acid (11)  
Acid (10)  
Acid (9)  
Acid (8)  
Acid (7)  
Acid (6)  
Teleporter (3)  
Acid (5)  
Teleporter (2)  
Checkpoint (1)

Project

Favorites

- All Materials
- All Models
- All Prefabs

Assets

2DGamekit  
Art  
Audio  
Documentation  
Prefabs  
Audio  
Enemies  
Environment  
Interactables  
SceneControl  
UIPrefabs  
Utilities  
VFX  
Scenes  
Scripts  
TutorialInfo  
Utilities  
Gizmos  
Scenes

Assets &gt; 2DGamekit &gt;

- Art
- Audio
- Documentation
- Prefabs
- Scenes
- Scripts
- TutorialInfo
- Utilities
- 2DGameKit\_Third-PartyNotice
- Readme

