ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

Τίτλος πτυχιακής Εργασίας	**Παιχνίδι ρόλων και ελληνική μυθολογία – άθλοι του Ηρακλή** **Unity based Role playing game for Greek mythology – Labors of Hercules**
Ονοματεπώνυμο φοιτητή	**Λιλλής Εμμανουήλ – Χρήστος**
Πατρώνυμο	**Ιωάννης**
Αριθμός μητρώου	**Π19091**
Επιβλέπων	**Θ. Παναγιωτόπουλος** **Καθηγητής**

Σεπτέμβριος 2024

Copyright ©

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Κ. Παναγιωτόπουλο για όλη την βοήθεια και τις γνώσεις που μου έδωσε με σκοπό να γίνει αυτή η πτυχιακή εργασία καθώς και όλους τους καθηγητές του τμήματος Πληροφορικής από τους οποίους έλαβα γνώση καθ’ όλη την 5ετια που παρακολουθούσα τα μαθήματα τους.

Περίληψη

Στην εργασία αυτή θα δούμε συνοπτικά τους κάποιους από τους άθλους του Ηρακλή.

Στόχος της Πτυχιακής εργασίας αυτής είναι να γίνει ένα εύκολο για τον χρήστη παιχνίδι 2d με σκοπό να παίξει και να δει κάποιους άθλους του Ηρακλή , με σκοπό να του κινήσει το ενδιαφέρον για την ιστορία του , αλλά και γενικά για όλη την ελληνική μυθολογία.

Αυτό επιτεύχθηκε μέσω του Unity και της c# που χρησιμοποιήθηκαν αντίστοιχα ως editor για τα γραφικά και ως γλώσσα κώδικα για τα scripts.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:

- Sprites
- Scripts
- Prefabs
- Materials
- Animations

Abstract

In this project we will briefly look at some of the labors of Hercules.

The aim of this thesis is to make a 2d game that is easy for the user to play and see some of the labors of Hercules, in order to arouse his interest in his history, but also in general for all Greek mythology.

This was achieved through Unity and c# used respectively as the editor for the graphics and as the code language for the scripts.

KEY WORD:

- Sprites
- Scripts
- Prefabs
- Materials

- Animations

Πίνακας περιεχομένων

Εισαγωγή

Όλοι μας ,μικροί και μεγάλοι, γνωρίζουμε τους άθλους του Ηρακλή , μια αν όχι η πιο γνωστή , από τις πιο γνωστές ιστορίες της ελληνικής μυθολογίας. Στο project αυτό θα δούμε και θα αναλύσουμε ένα παιχνίδι με θεματική κάποιους από τους άθλους του Ηρακλή. Η ιδέα ήτα να φτιάξω ένα παιχνίδι τύπου 2d top down δηλαδή ο παίχτης μας να κινείται σε 2 άξονες x και y και η κάμερα να είναι από πάνω του (Όπως βλέπουμε και στην παρακάτω εικόνα)και να υπάρχουν κάποια stages που το κάθε ένα θα περιέχει και κάποιο άθλο.

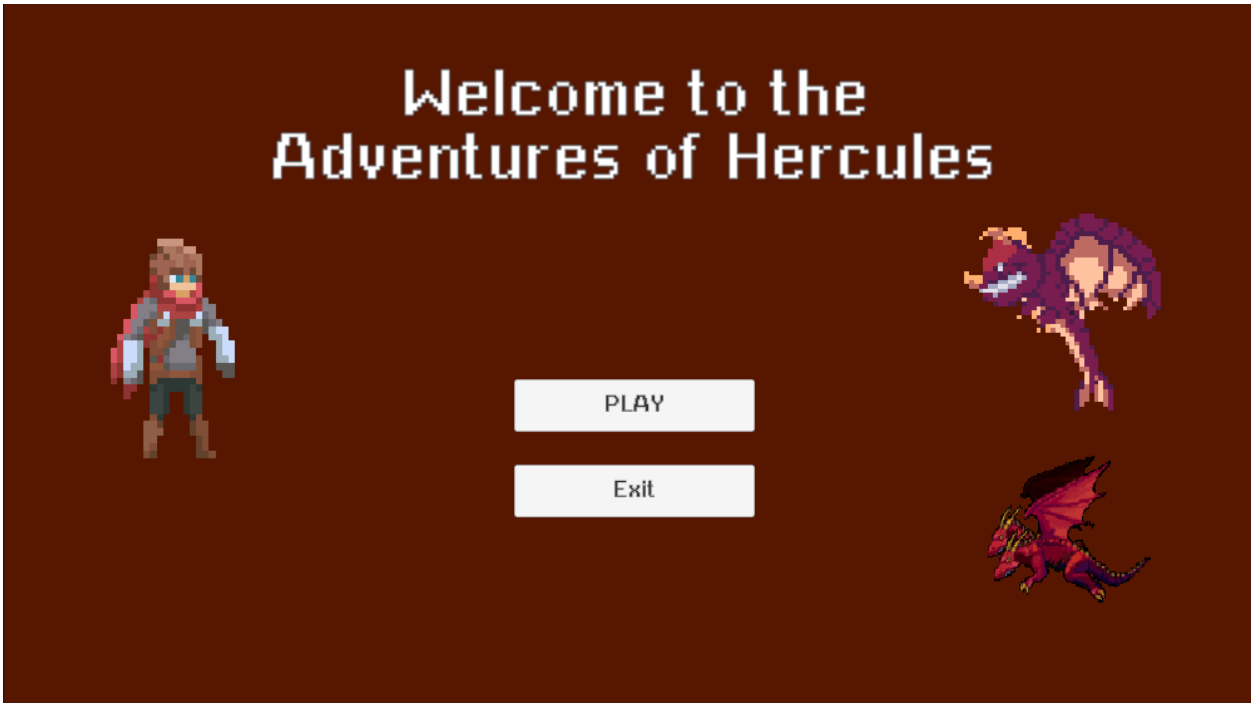
Εικόνα εισαγωγής

Παρακάτω θα αναλύσουμε πως έχει υλοποιηθεί η εργασία αρχικά στο κομμάτι του UI και των πιστών και ύστερα στο κομμάτι του κώδικα.

1. Επεξήγηση πιστών

1.1 Αρχή εφαρμογής

Στο main menu όπως μπορούμε να το πούμε , έχουμε κάτι απλό και άμεσο , δηλαδή τον τίτλο μας κάποιους από τους χαρακτήρες και 2 Button ένα Play και ένα exit



Εικόνα 1.1.1

Και πατώντας play πάμε στην κεντρική πιστά

Εικόνα 1.1.2

1.2 Πρώτη πίστα

Όπως αναφέρθηκε και πριν ξεκινάμε από την Κεντρική πιστά , η οποία στο project μας λέγεται MainScene

Σε αυτή την πιστά σκοπός είναι ο παίχτης μας να μπορεί να κουνηθεί ελευθέρα στον χάρτη για να μπορέσει να καταλάβει πως κουνιέται ο παίχτης και πως επιτίθεται

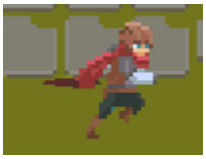
Εικόνα 1.2.1

Όπως βλέπουμε εδώ πχ που σημαδεύει με το τόξο το κερατάκι.

Να σημειωθεί ότι ο παίχτης κινείται με τα πλήκτρα W A S D

Κάνει attack με το left click

και κάνει dash με το space



Εικόνα 1.2.2

Επιπλέον , Σε αυτή την πρώτη πιστά μπορεί να σπάσει μερικά από τα κουτιά και πιθάρια (θα αναλύσουμε παρακάτω με λεπτομέρειες) και να του δώσει ως αντάλλαγμα λίγο gold , stamina η και ζωή

Εικόνα 1.2.3

Εικόνα 1.2.4

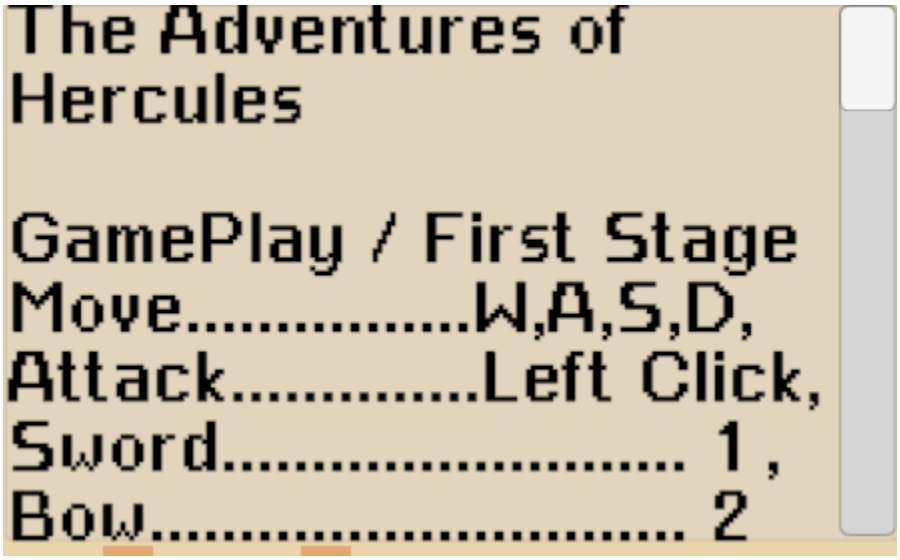
Αλλά και να σκοτώσει τα μικρά τερατάκια τα οποία και πάλι μπορούν να σε επιβραβεύσουν με ζωή , stamina και gold

Εικόνα 1.2.5

Επιπλέον έχει την δυνατότητα να δει κάποιες λεπτομέρειες σχετικά με την πίστα πατώντας τον πάπυρο στην πάνω δεξιά γωνιά της οθόνης

 *Εικόνα 1.2.6*

Οπού εδώ μας ανοίγει το παρακάτω σκηνικό



Εικόνα 1.2.7

Που μας λέει τις βασικές οδηγίες

Στο τέλος της πρώτης και βασικής πίστας μπορεί να πάρει το portal έτσι ώστε να πάει ουσιαστικά στον πρώτο άθλο.

Εικόνα 1.2.8

1.3 Δεύτερη πίστα

Και έπειτα μπαίνουμε στην 2η πίστα οπού εδώ έχουμε θεωρητικά τον άθλο με αποστολή να εξοντώσει τις Στυμφαλίδες όρνιθες

Εικόνα 1.3.1

Εδώ ο παίχτης πρέπει να σκοτώσει τις όρνιθες οι οποίες σε σου επιτίθονται και αυτές όπως βλέπουμε και στο σχετικό screenshot

Εικόνα 1.3.2

Στην περίπτωση που χάσουμε όλη μας την ζωή , ουσιαστικά ‘πεθαίνουμε ’

Εικόνα 1.3.3

Επίσης όπως και στην προηγούμενη πίστα πάλι εδώ άμα πατήσεις τον πάπυρο σου βγάζει λίγο από την ιστορία

Εικόνα 1.3.4

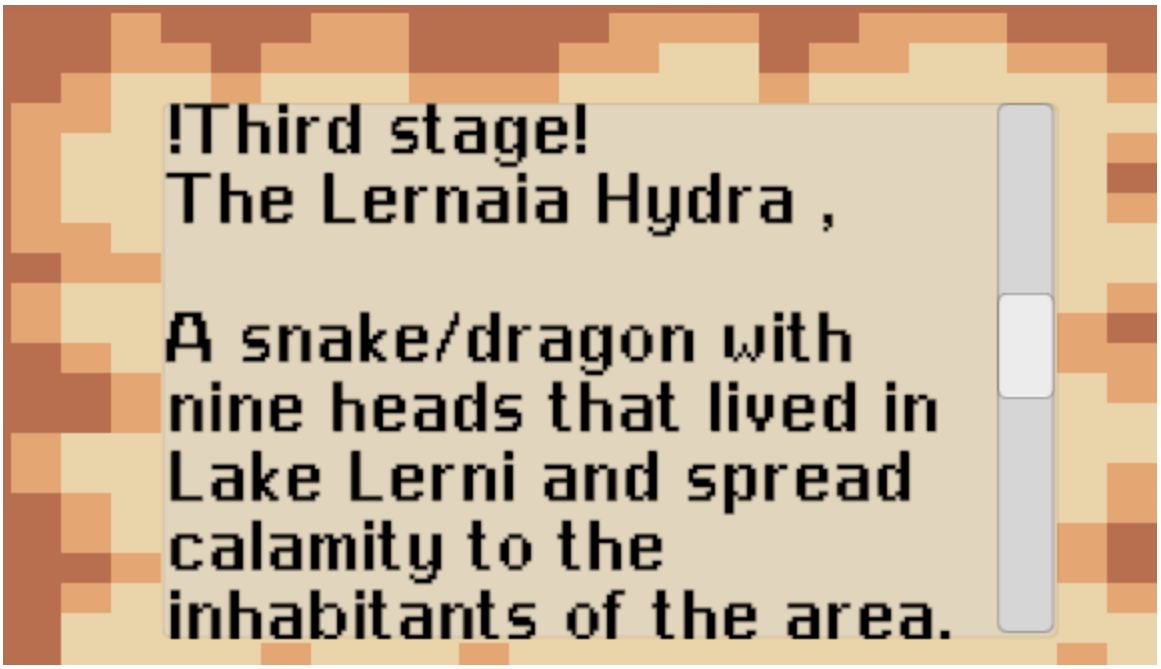
1.4 Τρίτη πίστα

Και έτσι πάμε στην τρίτη πίστα μέσω του επόμενου portal οπού εδώ έχουμε τον άθλο με την λερναία Ύδρα

Εικόνα 1.4.1

οπού εδώ σκοπός είναι να εξοντώσει την λερναία Ύδρα χωρίς να πεθάνει

και πάλι και σε αυτή την σκηνή πάλι έχει στον πάπυρο την ιστορία



Εικόνα 1.4.2

1.5 Τέταρτη πίστα

Και αφού την εξοντώσει πάει στον τελευταίο άθλο που έχει πραγματοποιηθεί σε αυτό το project οπού έχει να κάνει με το να καθαρίσει τους στάβλους του Αυγεία.

Εικόνα 1.5.1

Αυτή η σκηνή προσομοιάζει τον άθλο δηλαδή ο Ηρακλής να καθαρίσει την κοπριά από τον στάβλο και αφού τελειώσει και αυτόν τον άθλο παίρνει την τελευταία έξοδο και βλέπει αυτό το μήνυμα

Εικόνα 1.5.2

όπου αναφέρεται στο γεγονός ότι αυτό το Project θα έχει στο μέλλον και τους 12 άθλους.

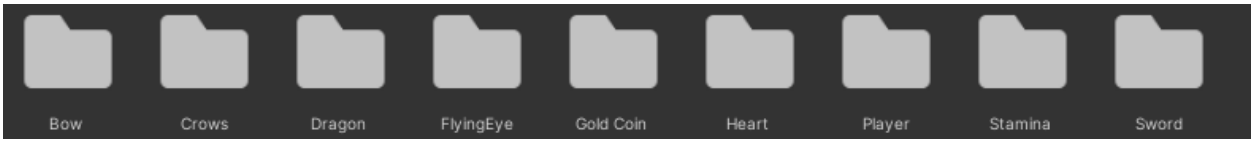
2. Επεξήγηση Assets

Παρακάτω θα δούμε με λεπτομέρειες όλα τα assets που υπάρχουν στο Project

2.1 Animations

Ας ξεκινήσουμε με τα animations

Όταν ανοίξουμε τον φάκελο θα δούμε ότι έχει μέσα διάφορους φακέλους σχετικά με τους χαρακτήρες στο παιχνίδι που κουνιόνται



Εικόνα 2.1.1

Εδώ διακρίνουμε ότι έχουμε διάφορες κατηγορίες τύπων , δηλαδή έχουμε τα animations Που αφορούν

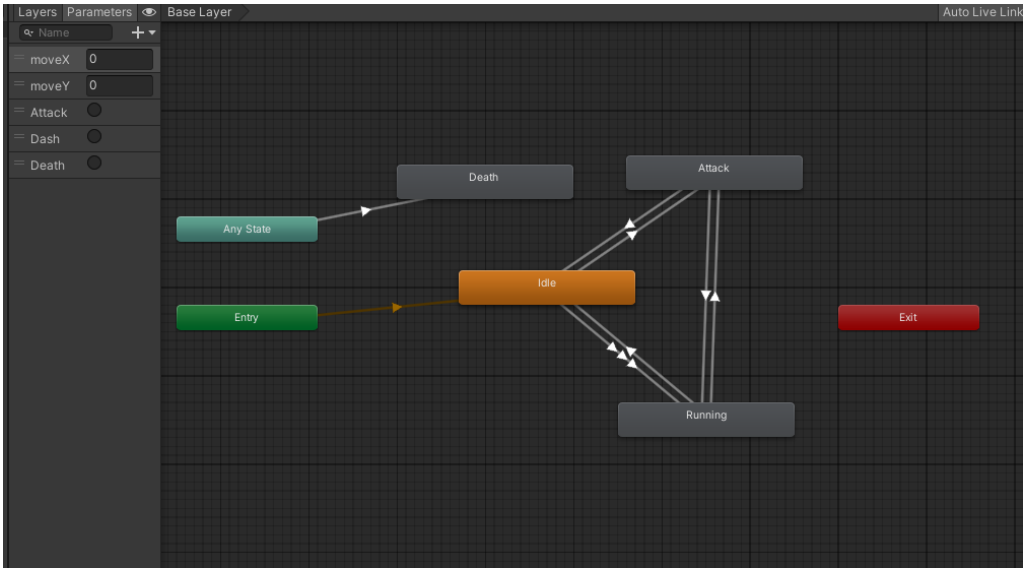
1. Τον χαρακτήρα και τα τέρατα (Player , Dragon , Crows, Flyingeye)
2. Τα όπλα (Bow και Sword)
3. Και τέλος τα αναλώσιμα(Gold coin , Heart και stamina)

2.1.1 Κατηγορία χαρακτήρα και Τεράτων

Όσο αφορά την πρώτη κατηγορία και οι 4 φάκελοι έχουν σχεδόν ίδιο μοτίβο , δηλαδή περιέχουν έναν Animator ως controller και τα animations που χρειάζεται ο κάθε ένας. Πχ. Ο Player έχει τα παρακάτω

Εικόνα 2.1.1.1

Πιο αναλυτικά , το animation Attack01 έχει να κάνει με το πως φαίνεται όταν ο παίχτης αποφασίζει να επιτεθεί , το animation Death όταν ο παίχτης μας πεθαίνει , το animation idle όταν ο παίχτης είναι σταθερός κα το Running όταν μετακινείται μέσα στο map. Και όλα αυτά συνδέονται με το παρακάτω διάγραμμα:



Εικόνα 2.1.1.2

Εδώ βλέπουμε ένα διάγραμμα στο animator Με τα διαφορά states. Προφανώς όταν ο παίχτης μας δεν κάνει τίποτα το state και animation idle τρέχει σε λούπα. Άμα ο παίχτης κάνει επίθεση τότε γίνεται trigger το Attack

Εικόνα 2.1.1.3

Και εκτελείται μια φορά το state Και animation Attack.

Αντίστοιχα λειτουργεί και το running.

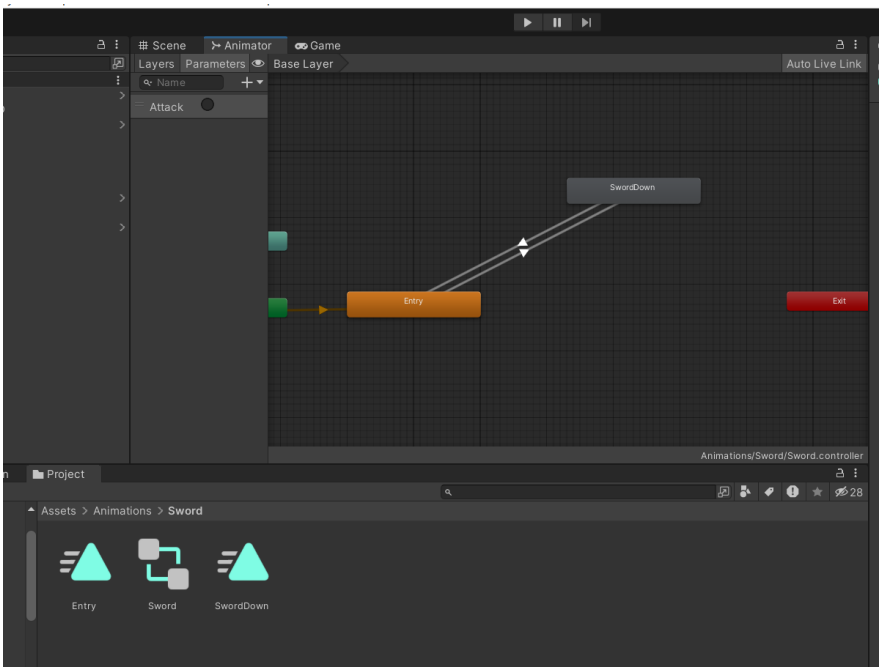
Όσο αφορά το state και animation Death επειδή ανά πάσα στιγμή ο παίχτης μπορεί να φάει αρκετό damage έτσι ώστε να πεθάνει το βάζουμε μόνο του στην άκρη και γίνεται trigger όταν το health του παίχτη φτάσει το 0.

αντίστοιχα λειτουργούν και τα αλλά 3 στην ιδιά κατηγορία δηλαδή και το dragon και το Crows και το Flyingeye.

2.1.2 Κατηγορία όπλων

Τώρα πάμε στην δεύτερή κατηγορία όσο αφορά τα animations δηλαδή ότι αφορά τα όπλα μας , δηλαδή το sword και το bow.

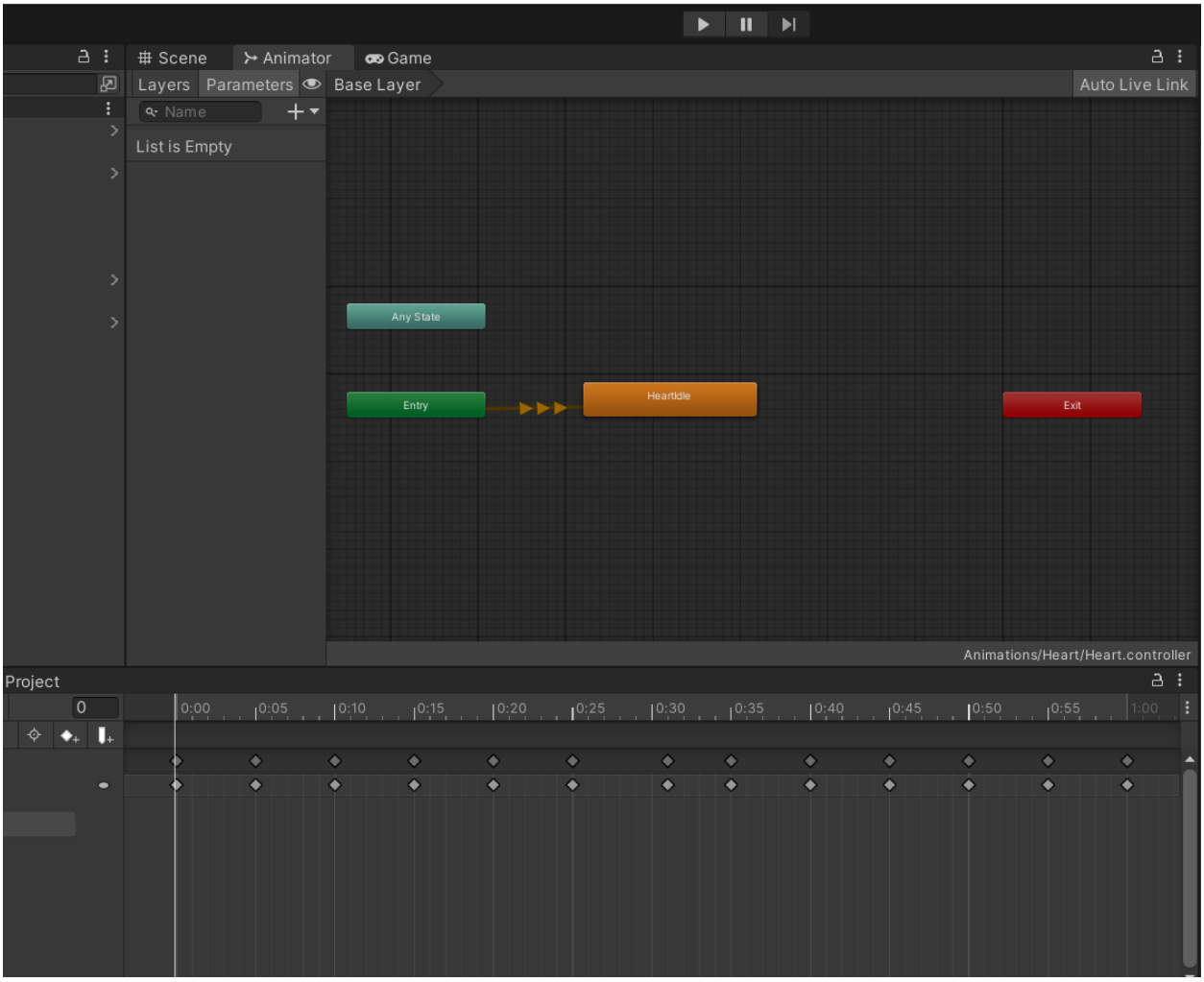
Και τα 2 έχουν ακριβώς την ιδιά αρχιτεκτονική δηλαδή ένα απλό animation όταν κάνουμε το attack που κουνιέται το όπλο.



Εικόνα 2.1.2.1

2.1.3 Κατηγορία αναλωσίμων

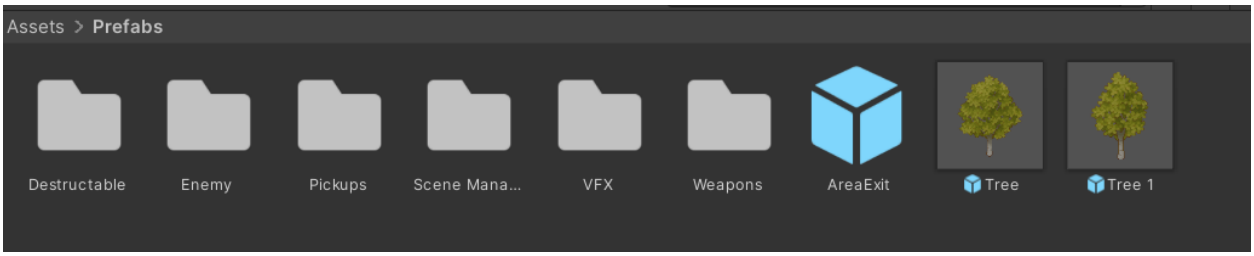
Και τέλος πάμε στην τρίτη κατηγορία animations που αφορά τα αναλώσιμα , δηλαδή τα icons των heart , coin και stamina , οπού εκεί έχουμε κάτι πολύ απλό , ένα idle animation που κάνει τα icons να γυρνάνε γύρω γύρω.



Εικόνα 2.1.3.1

2.2 Prefabs

Αφού τελειώσαμε με τα animations , πάμε στον επόμενο και πολύ σημαντικό φάκελο , τα **prefabs.**

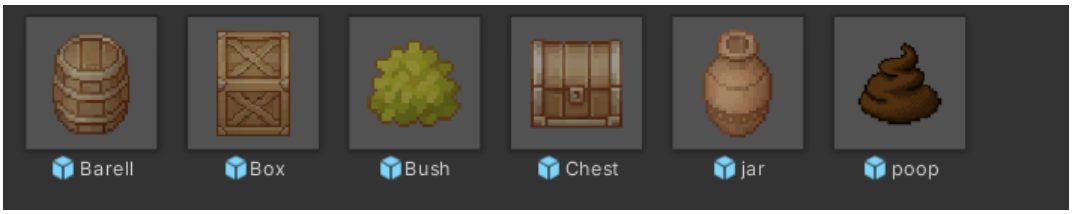


Η χρήση των **prefabs** γίνεται για την ευκολία που μπορεί να μας προσφέρει όσο αφορά την τροποποίηση ενός object σε μερικά ή και όλα τα scenes καθώς αλλάζοντας μια ρύθμιση σε ένα , αλλάζουν όλα.

Εδώ βλέπουμε τα prefabs για τα δέντρα αλλά και για τα portal που αλλάζει ο παίχτης τις πίστες

2.2.1 Φάκελος Destructable

Μέσα στον φάκελο Destructable έχουμε τα παρακάτω

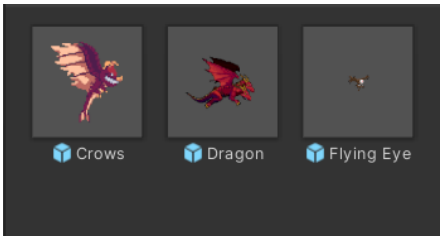


Εικόνα 2.2.1.1

Οπού έχουμε τα αντικείμενα που μπορείς να καταστρέψεις στην πίστα

2.2.2 Φάκελος Enemy

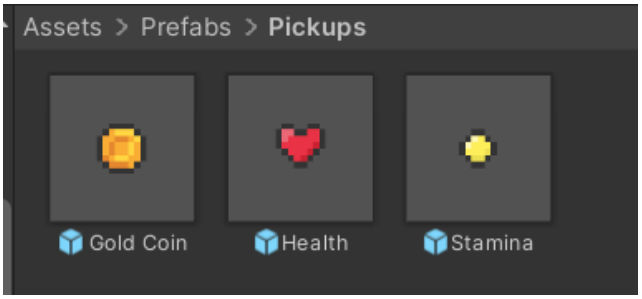
Έπειτα έχουμε το Enemy που περιέχει τους εχθρούς



Εικόνα 2.2.2.1

2.2.3 Φάκελος Prefabs

Τα αναλώσιμα



Εικόνα 2.2.3.1

2.2.4 Φάκελος VFX

Τον φάκελο VFX οπού μέσα έχει τα animation όταν καταστρέφουμε τα αντικείμενα

Εικόνα 2.2.4.1

2.2.5 Φάκελος Scene Manager

Τον scene manager που έχουμε το prefab του παίχτη μας καθώς και τα manager για την camera τον canvas(δηλαδή το ui)

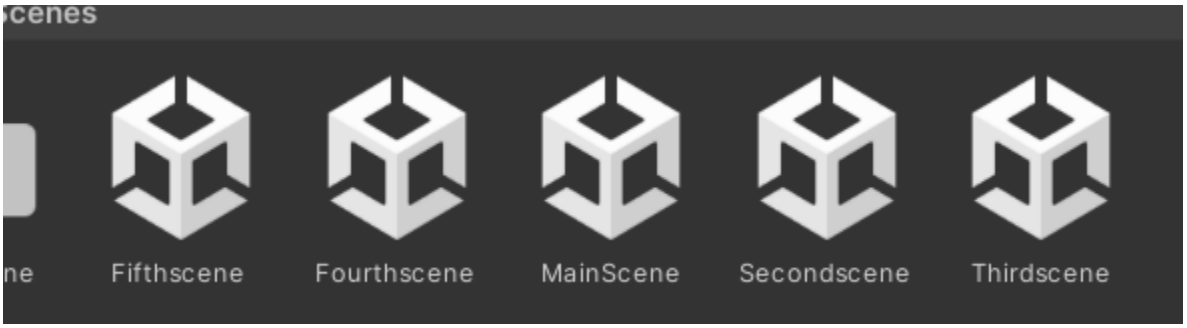
2.2.6 Φάκελος Weapon

Και τέλος έχουμε τον φάκελο weapon όπου περιέχει τα όπλα του παίχτη μας αλλά κα των enemy.

Εικόνα 2.2.6.1

2.3 Scenes

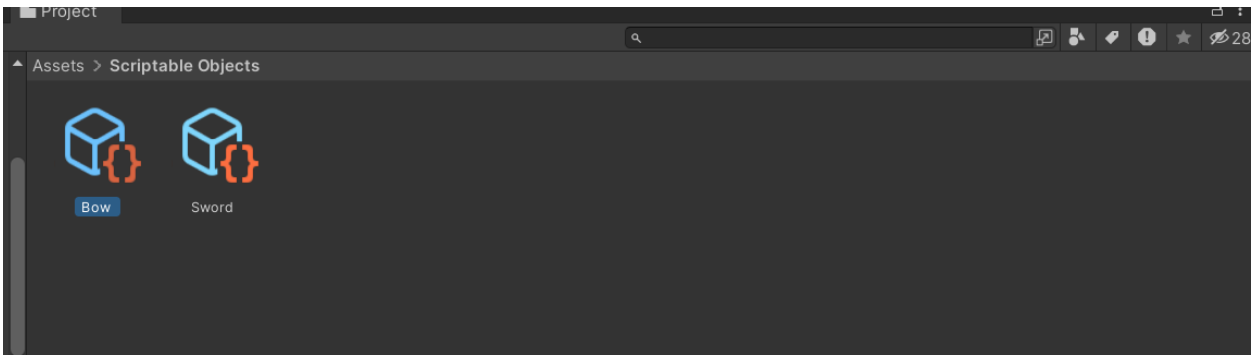
Ο επόμενος φάκελος που ακολουθεί είναι ο φάκελος με **τα scenes** του παιχνιδιού μας



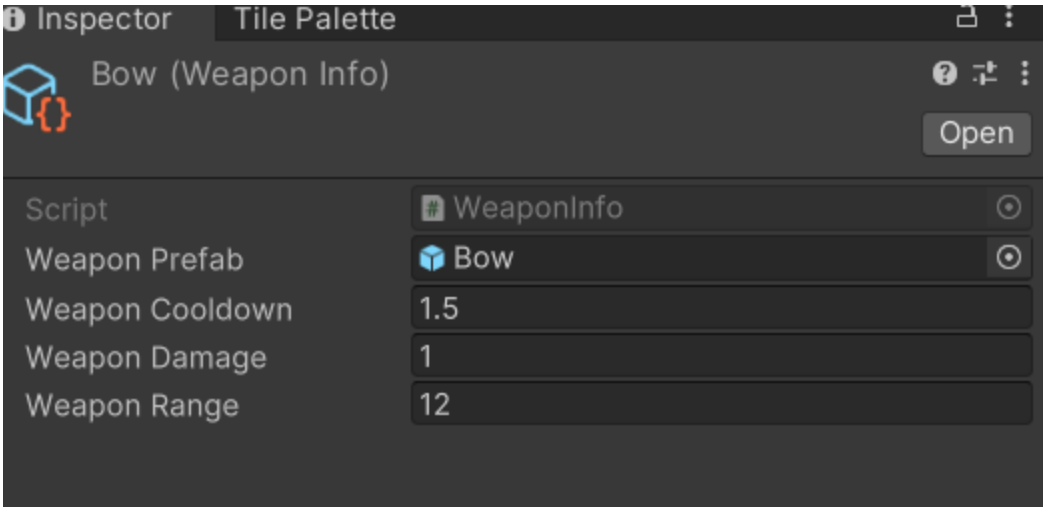
Εικόνα 2.3.1

2.4 Scriptable Objects

Έπειτα έχουμε τα **scriptable objects** οπού εκεί έχουμε τα 2 μας όπλα και με αυτά ρυθμίζουμε καλυτέρα τα βασικά χαρακτηριστικά των οπλών , όπως το cooldown , το damage και το range



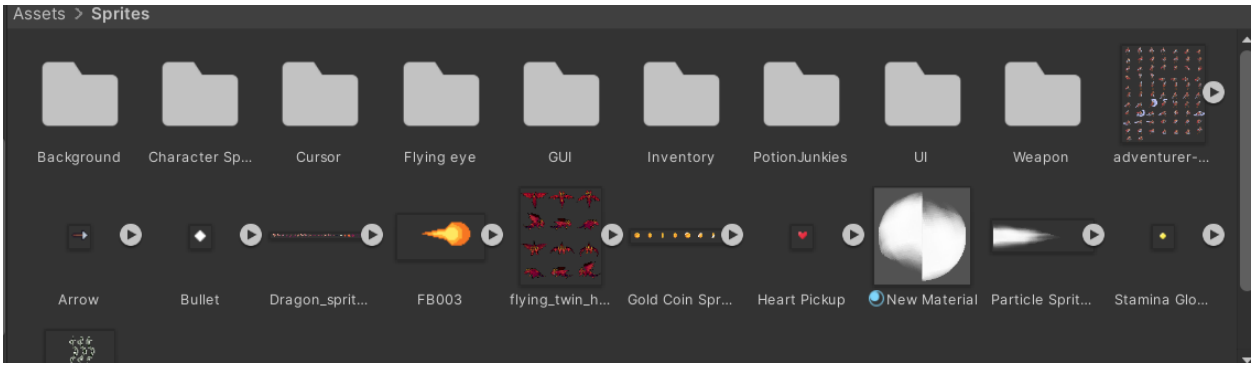
Εικόνα 2.4.1



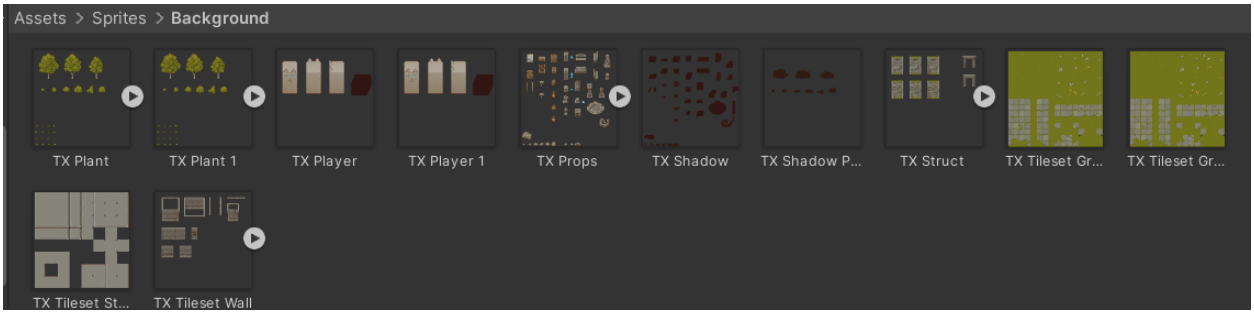
Εικόνα 2.4.2

2.5 Sprites

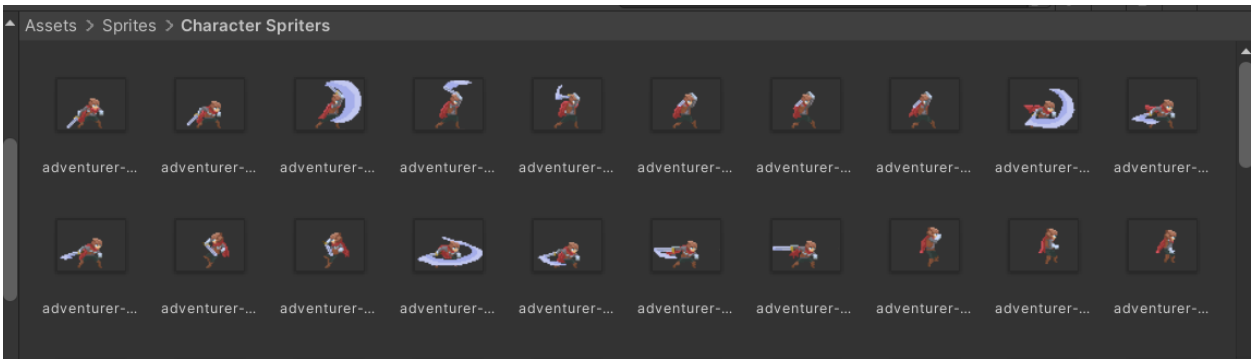
Στην συνέχεια έχουμε τον φάκελο με τα **sprites** οπού μέσα έχουνε όλα τα εικονίδια τα οποία χρησιμοποιήσαμε στο project



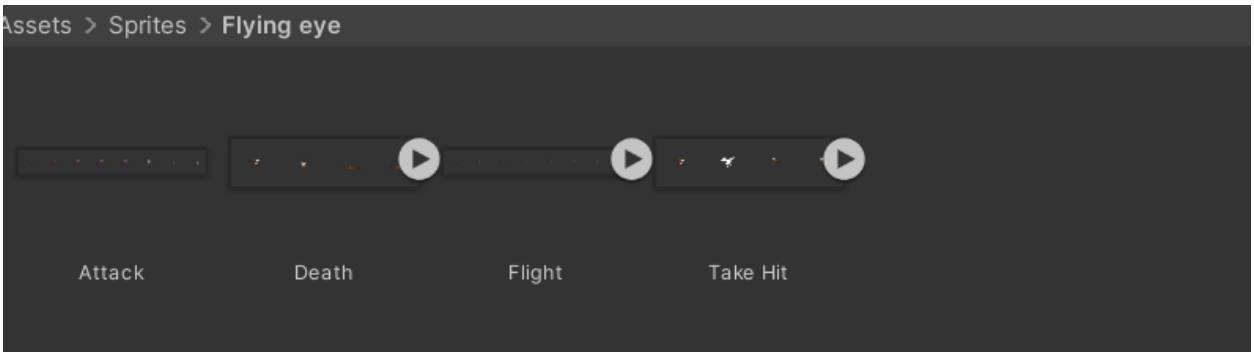
Εικόνα 2.5.1



Εικόνα 2.5.2

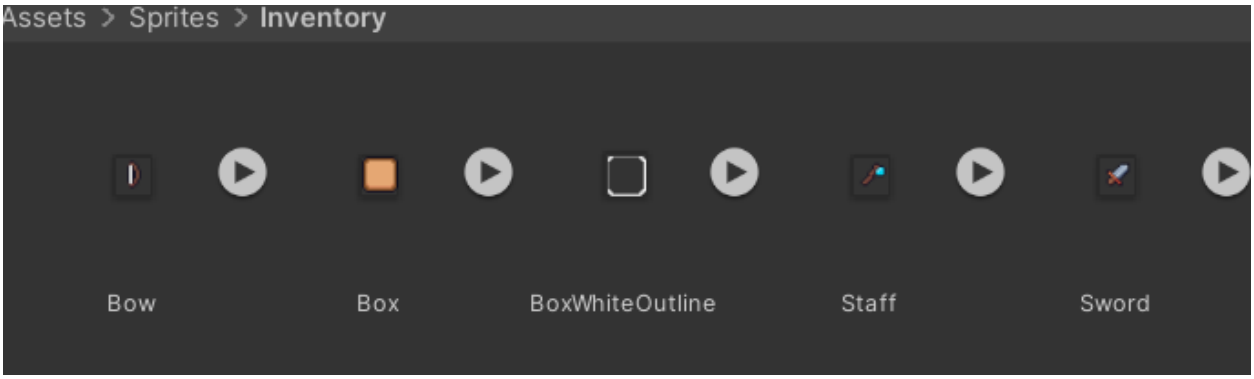


Εικόνα 2.5.3

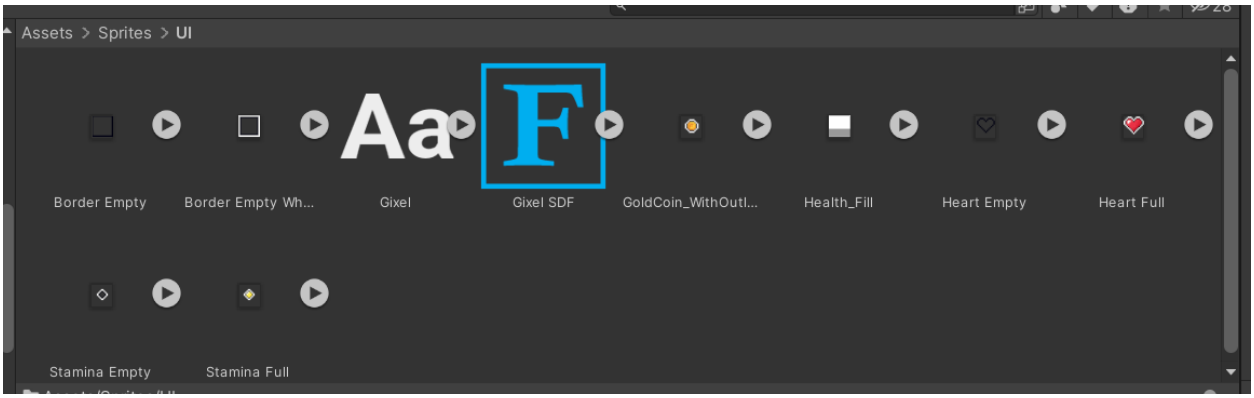


Εικόνα 2.5.4

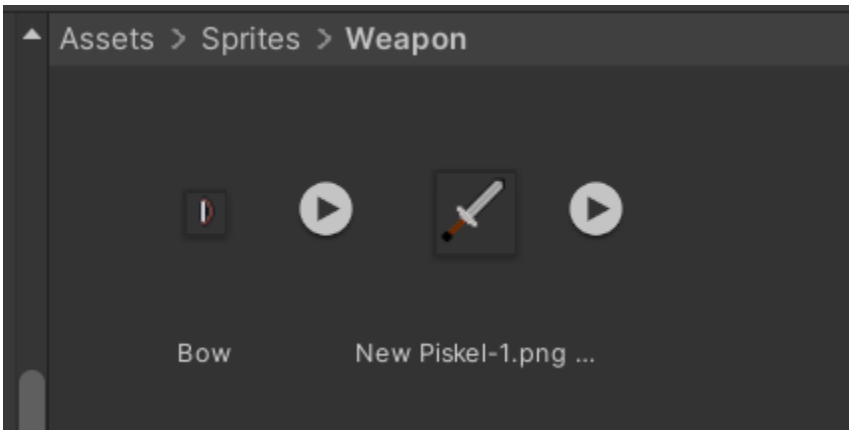
Εικόνα 2.5.5



Εικόνα 2.5.6



Εικόνα 2.5.7

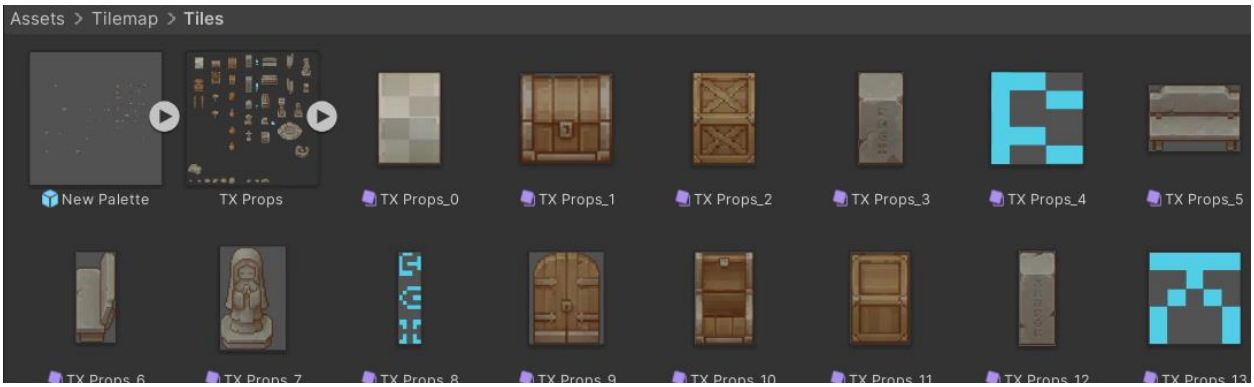


Εικόνα 2.5.8

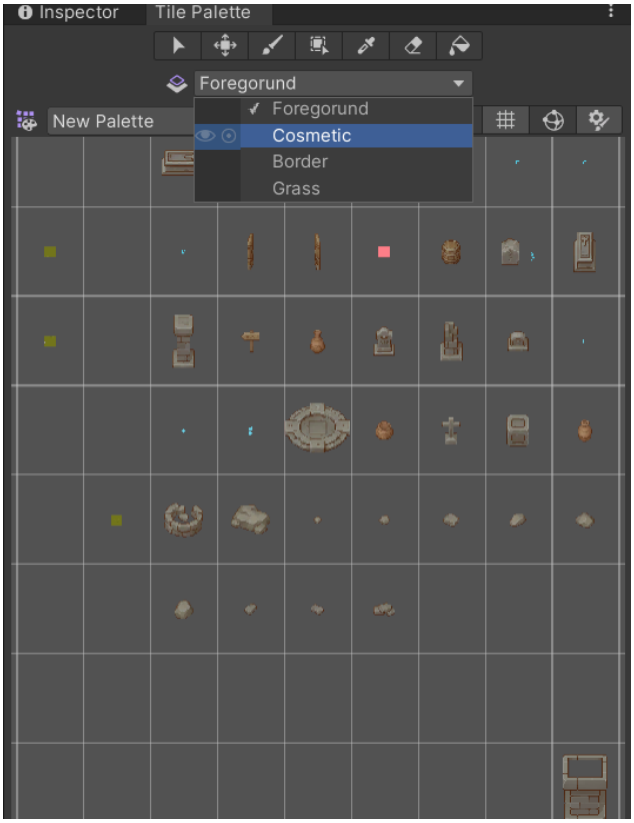
Να σημειωθεί ότι τα παραπάνω χρησιμοποιήθηκαν είτε χωρίς επεξεργασία στο κάθε scene είτε μέσω animations.

2.6 Tilemap

Και τελευταίος φάκελος είναι αυτός με το **tilemap** το οποίο ουσιαστικά μας επιτρέπει να ζωγραφήσουμε την σκηνή μας και σε διαφορά επίπεδα έτσι ώστε να έχουμε το χορτάρι τον δρόμοι τα διαφορά διακοσμητικά αλλά και τα barrier στα οποία ο παίχτης δεν μπορεί να τα προσπεράσει.



Εικόνα 2.6.1

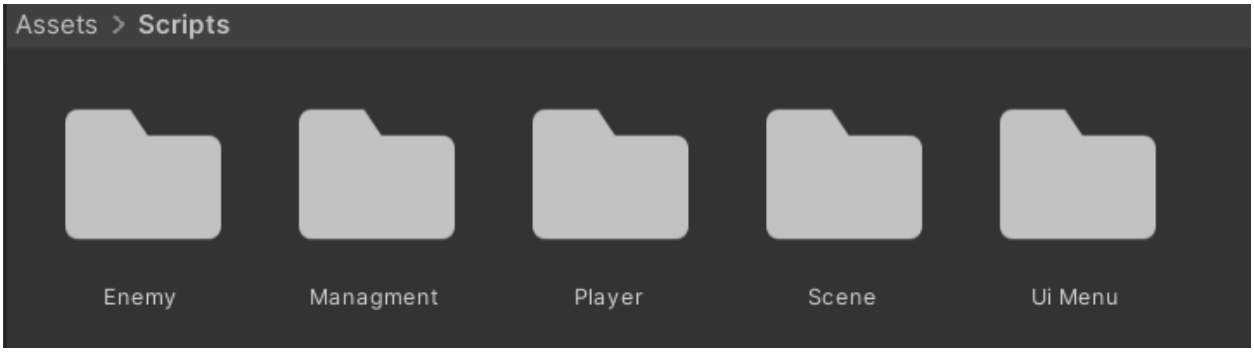


Εικόνα 2.6.2

3. Επεξήγηση Scripts

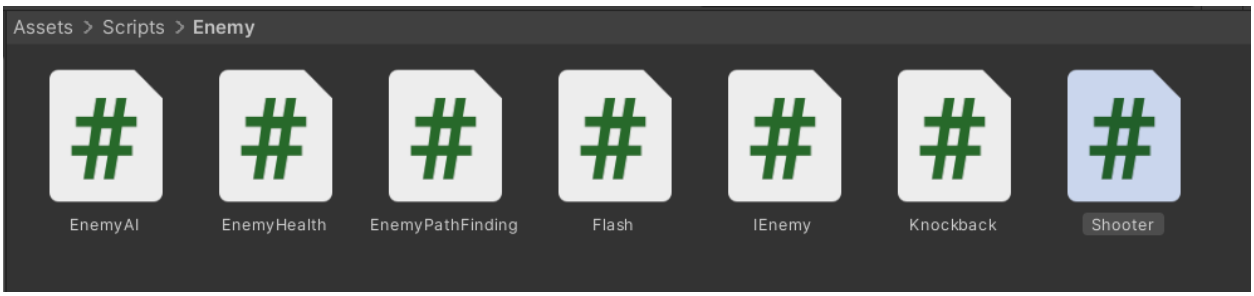
Τέλος είναι ο πολύ σημαντικός φάκελος ο οποίος περιέχει μέσα όλα τα scripts.

Μέσα εκεί έχουμε υποφάκελους με κώδικα που αφορά ο κάθε ένας και ένα κομμάτι του project .



3.1 Φάκελος Enemy

Ας ξεκινήσουμε με τον φάκελο Enemy.



3.1.1 EnemyAI

Θα ξεκινήσουμε με το πρώτο script που είναι το EnemyAI όπου κυρίως αφορά ολες τις λειτουργείες των εχθρών , όπως είναι το roaming δηλαδή το πως κουνιούνται μέσα στο map αλλά και το attack δηλαδή το πως επιτίθονται και πότε , ανάλογα με το cooldown που τους έχουμε βάλει.

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

public class EnemyAI : MonoBehaviour
{
    [SerializeField] private float roamChangeDirFloat = 2f;
    [SerializeField] private float attackRange = 5f;
    [SerializeField] private MonoBehaviour enemyType;
    [SerializeField] private float attackCD = 2f;
    [SerializeField] private bool stopMovingWhileAttacking = false;

    private bool canAttack = true;
    private enum State
    {
        Roaming,
        Attacking
    }

    private Vector2 roamPosition;
    private float timeRoaming = 0f;

    private State state;
    private EnemyPathFinding enemyPathFinding;

    private void Awake()
    {
        enemyPathFinding = GetComponent<EnemyPathFinding>();
        state = State.Roaming;
    }

    private void Start()
    {
        //StartCoroutine(RoamingRoutine());
        roamPosition = GetRoamingPosition();
    }

    private void Update()
    {
        MovementStateControl();
    }
```

Εικόνα 3.1.1.1

και έπειτα τις αρχικοποιήσεις έχουμε ένα switch για να ξέρουμε σε πιο state βρίσκεται.

```
private void MovementStateControl()
{
    switch (state)
    {
        default:
        case State.Roaming:
            Roaming();
            break;

        case State.Attacking:
            Attacking();
            break;
    }
}
```

Εικόνα 3.1.1.2

Άρα έχουμε το roaming

```
private void Roaming()
{
    timeRoaming += Time.deltaTime;

    enemyPathFinding.MoveTo(roamPosition);
    if(Vector2.Distance(transform.position , PlayerController.Instance.transform.position) < attackRange)
    {
        state = State.Attacking;
    }

    if (timeRoaming > roamChangeDirFloat)
    {
        roamPosition = GetRoamingPosition();
    }
}
```

Εικόνα 3.1.1.3

Και το attack

```
private void Attacking()
{
    if(Vector2.Distance(transform.position , PlayerController.Instance.transform.position)> attackRange)
    {
        state = State.Roaming;
    }

    if (attackRange != 0 && canAttack)
    {
        canAttack = false;
        (enemyType as IEnemy).Attack();

        if (stopMovingWhileAttacking)
        {
            enemyPathFinding.StopMoving();
        }
        else
        {
            enemyPathFinding.MoveTo(roamPosition);
        }

        StartCoroutine(AttackCDRoutine());
    }
}
```

Εικόνα 3.1.1.4

3.1.2 EnemyHealth

Επόμενο script είναι το EnemyHealth το οποίο είναι υπεύθυνο για την ζωή του αντιπάλου αλλα και για το death και το damage animation

```
public class EnemyHealth : MonoBehaviour
{
   [SerializeField] private int startingHealth  = 5;
    [SerializeField] private float knockBackThrust = 15f;

    private int currentHealth;
    private Knockback knockback;
    private Flash flash;

    private Animator animator;

    private void Awake()
    {
        animator = GetComponent<Animator>();
        flash = GetComponent<Flash>();
        knockback = GetComponent<Knockback>();
    }

    private void Start()
    {
        currentHealth = startingHealth;
    }
```

Εικόνα 3.1.2.1

Και μετά την αρχικοποίηση έχουμε την στιγμή που ο εχθρός χτυπιέται και ελέγχουμε αν έχει απομείνει ζωή και αν όχι τότε ενεργοποιείται το death animation και μετα εξαφανίζεται ο εχθρός.

```
public void TakeDamage(int damage)
{
    currentHealth -= damage;
    knockback.GetKnockedBack(PlayerController.Instance.transform, knockBackThrust);
    StartCoroutine(flash.FlashRoutine());
    StartCoroutine(CheckDetectDeathRoutine());
}

private IEnumerator CheckDetectDeathRoutine()
{
    yield return new WaitForSeconds(flash.GetRestoreMatTime());
    DetectDeath();
}

public void DetectDeath()
{
    if (currentHealth <= 0)
    {
        animator.SetTrigger("Death");
        GetComponent<PickUpSpawner>().DropItems();

        Destroy(gameObject , .35f);
    }
}
```

Εικόνα 3.1.2.2

3.1.3 EnemyPathRoaming

Έπειτά έχουμε το script EnemyPathRoaming το οποίο είναι βασικά ένα βοηθητικό script για το Movement του εχθρού από το roaming που είδαμε στο EnemyAI , δηλαδή πότε θα αλλάξει πλευρά.

```
}

private void FixedUpdate()
{
    if (knockback.GettingKnockedBack) { return; }

    rb.MovePosition(rb.position + moveDir * (moveSpeed * Time.fixedDeltaTime));

    if (moveDir.x < 0)
    {
        spriteRenderer.flipX = true;
    }
    else
    {
        spriteRenderer.flipX = false;
    }
}
```

Εικόνα 3.1.3.1

Πότε θα κουνηθεί και ποτέ θα σταματήσει.

```
public void MoveTo(Vector2 targetPosition)
{
    moveDir = targetPosition;
}

public void StopMoving()
{
    moveDir = Vector3.zero;
}
```

Εικόνα 3.1.3.2

3.1.4 Flash

Στην συνέχεια έχουμε το Flash που είναι ου είναι υπεύθυνο για το animation όταν χτυπάμε τον αντιπαλο.

```
public class Flash : MonoBehaviour
{
    [SerializeField] private Material whiteFlashMat;
    [SerializeField] private float restoreDefaultMatTime = .2f;

    private Material defaultMat;
    private SpriteRenderer spriteRenderer;


    private void Awake()
    {

        spriteRenderer = GetComponent<SpriteRenderer>();
        defaultMat = spriteRenderer.material;
    }
     public float GetRestoreMatTime()
    {
        return restoreDefaultMatTime;
    }

    public IEnumerator FlashRoutine()
    {
        spriteRenderer.material = whiteFlashMat;
        yield return new WaitForSeconds(restoreDefaultMatTime);
        spriteRenderer.material = defaultMat;

    }
}
```

Εικόνα 3.1.4.1

3.1.5 Knockback

Προτελευταίο script είναι αυτό του Knockback που μας βοηθάει όταν πετυχαίνουμε τον εχθρό να τον σπρώχνει προς τα πίσω

```
public class Knockback : MonoBehaviour
{
    public bool GettingKnockedBack { get; private set; }

    [SerializeField] private float knockBackTime = .2f;

    private Rigidbody2D rb;

    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    public void GetKnockedBack(Transform damageSource , float knockBackThrust)
    {
        GettingKnockedBack = true;
        Vector2 difference = (transform.position - damageSource.position).normalized * knockBackThrust * rb.mass;
        rb.AddForce(difference, ForceMode2D.Impulse);
        StartCoroutine(KnockRoutine());
    }

    private IEnumerator KnockRoutine()
    {
        yield return new WaitForSeconds(knockBackTime);
        rb.velocity = Vector2.zero;
        GettingKnockedBack = false;
    }
}
```

Εικόνα 3.1.5.1

3.1.6 Shooter

Και τελευταίο script στην κατηγορία Enemy είναι το Shooter. Σε αυτό το script έχουμε φτιάξει τον τρόπο να φεύγουν οι ‘φλωγιες’ από τους εχθρούς και να μπορούμε μέσα από το unity να πειράξουμε διάφορες παραμέτρους , όπως , γωνίες κλείσεις επίθεσης , ταχύτητα , πόσες φλωγιες , πόσες επαναλήψεις κλπ κλπ.

```
public class Shooter : MonoBehaviour , IEnemy
{
    [SerializeField] private GameObject bulletPrefab;
    [SerializeField] private float bulletMoveSpeed;
    [SerializeField] private int burstCount;
    [SerializeField] private float timeBetweenBurst;
    [SerializeField] private float restTime = 1f;
    [SerializeField] private int projectilesPerBurst;
    [SerializeField] [Range(0, 359)] private float angleSpread;
    [SerializeField] private float startingDistance = 0.1f;
    [SerializeField] private bool stagger;
    [SerializeField] private bool oscillate;

    private bool isShooting = false;
    private void OnValidate()
    {
        if (oscillate) { stagger = true; }
        if (!oscillate) { stagger = false; }
        if (projectilesPerBurst < 1) { projectilesPerBurst = 1; }
        if (burstCount < 1) { burstCount = 1; }
        if (timeBetweenBurst < 0.1f) { timeBetweenBurst = 0.1f; }
        if (restTime < 0.1f) { restTime = 0.1f; }
        if (startingDistance < 0.1f) { startingDistance = 0.1f; }
        if (angleSpread == 0) { projectilesPerBurst = 1; }
        if (bulletMoveSpeed <= 0) { bulletMoveSpeed = 0.1f; }
    }

    public void Attack()
    {

        if (!isShooting)
        {
            StartCoroutine(ShootRoutine());
        }

    }
```

Εικόνα 3.1.6.1

```
private IEnumerator ShootRoutine()
{
    isShooting = true;

    float startAngle, currentAngle, angleStep, endAngle;
    float timeBetweenProjectiles = 0f;

    TargetConeOfInfluence(out startAngle, out currentAngle, out angleStep, out endAngle);

    if (stagger) { timeBetweenProjectiles = timeBetweenBurst / projectilesPerBurst; }

    for (int i = 0; i < burstCount; i++)
    {
        if (!oscillate)
        {
            TargetConeOfInfluence(out startAngle, out currentAngle, out angleStep, out endAngle);
        }

        if (oscillate && i % 2 != 1)
        {
            TargetConeOfInfluence(out startAngle, out currentAngle, out angleStep, out endAngle);
        }
        else if (oscillate)
        {
            currentAngle = endAngle;
            endAngle = startAngle;
            startAngle = currentAngle;
            angleStep *= -1;
        }
```

Εικόνα 3.1.6.2

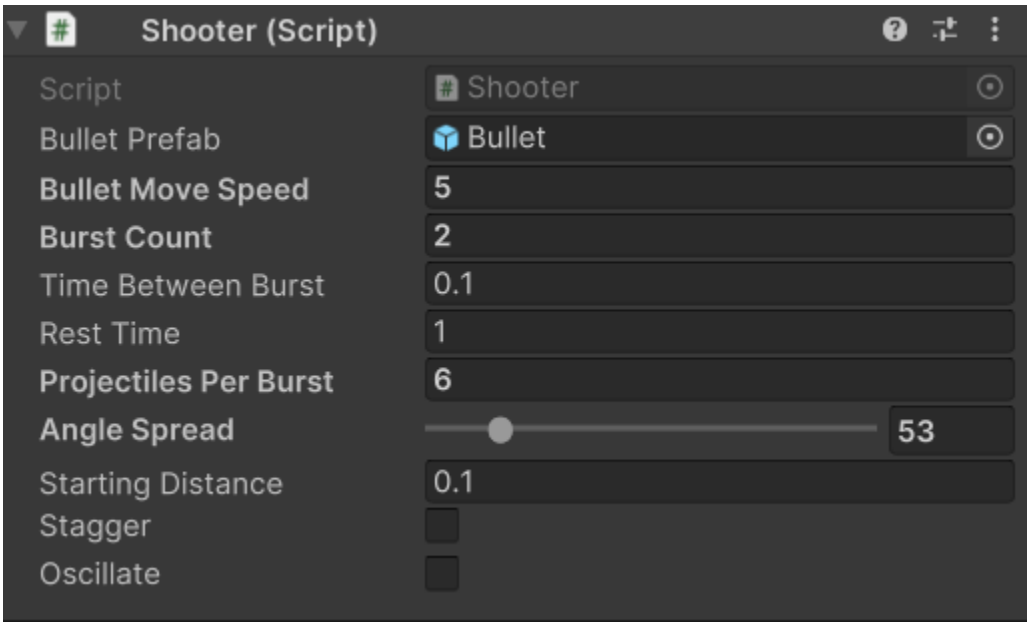
```
        for (int j = 0; j < projectilesPerBurst; j++)
        {
            Vector2 pos = FindBulletSpawnPos(currentAngle);

            GameObject newBullet = Instantiate(bulletPrefab, pos, Quaternion.identity);
            newBullet.transform.right = newBullet.transform.position - transform.position;


            if (newBullet.TryGetComponent(out Projectile projectile))
            {
                projectile.UpdateMoveSpeed(bulletMoveSpeed);
            }

            currentAngle += angleStep;

            if (stagger) { yield return new WaitForSeconds(timeBetweenProjectiles); }
        }

        currentAngle = startAngle;

        if (!stagger) { yield return new WaitForSeconds(timeBetweenBurst); }
    }

    yield return new WaitForSeconds(restTime);
    isShooting = false;
}
```

Εικόνα 3.1.6.3

```
private void TargetConeOfInfluence(out float startAngle, out float currentAngle, out float angleStep, out float endAngle)
{
    Vector2 targetDirection = PlayerController.Instance.transform.position - transform.position;
    float targetAngle = Mathf.Atan2(targetDirection.y, targetDirection.x) * Mathf.Rad2Deg;
    startAngle = targetAngle;
    endAngle = targetAngle;
    currentAngle = targetAngle;
    float halfAngleSpread = 0f;
    angleStep = 0;
    if (angleSpread != 0)
    {
        angleStep = angleSpread / (projectilesPerBurst - 1);
        halfAngleSpread = angleSpread / 2f;
        startAngle = targetAngle - halfAngleSpread;
        endAngle = targetAngle + halfAngleSpread;
        currentAngle = startAngle;
    }
}

private Vector2 FindBulletSpawnPos(float currentAngle)
{
    float x = transform.position.x + startingDistance * Mathf.Cos(currentAngle * Mathf.Deg2Rad);
    float y = transform.position.y + startingDistance * Mathf.Sin(currentAngle * Mathf.Deg2Rad);

    Vector2 pos = new Vector2(x, y);

    return pos;
}
```

Εικόνα 3.1.6.4

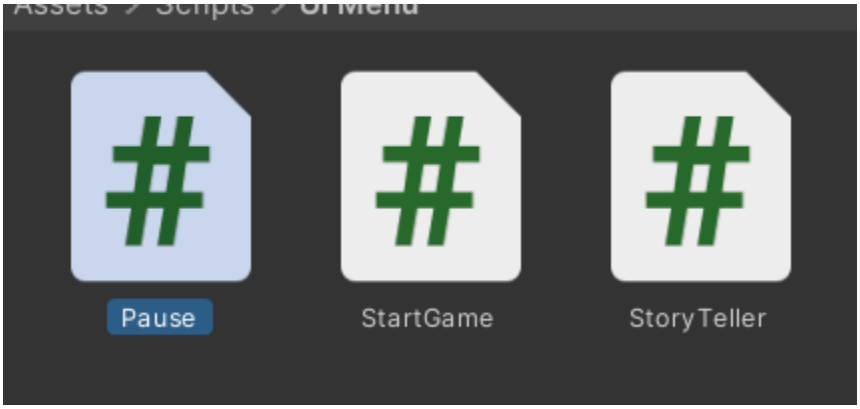
Και όπως βλέπουμε στο ui του Unity έχουμε αυτές τις επιλογές που διαφέρουν ανάλογα τον εχθρό.



Εικόνα 3.1.6.5

3.2 Φάκελος UI Menu

Επόμενος φάκελος που θα αναλύσουμε είναι αυτός του UI menu οπού μέσα έχει τα script που μας βοηθάνε όταν ανοίγουμε κάποιο menu , είτε αυτό είναι το main menu είτε το μενού με την ιστορία .



3.2.1 Pause

Με το script pause όταν ανοίγουμε το μενού ουσιαστικά σταματάμε οποιαδήποτε ροή στο παιχνίδι.

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

public class Pause : MonoBehaviour
{


    public void StopGame()
    {
        Time.timeScale = 0;
    }

    public void StartAgain()
    {
        Time.timeScale = 1;
    }

    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Εικόνα 3.2.1.1

3.2.2 Start Game

Με το start game όταν πατάμε play κλείνουμε μενού και ξεκινάμε το παιχνίδι.

```
public class StartGame : MonoBehaviour
{

    public GameObject startpanel;

    private void Start()
    {
        Time.timeScale = 0;
    }

    public void StartAgain()
    {
        Time.timeScale = 1;
        startpanel.SetActive(false);
    }

    public void Exit()
    {
        Application.Quit();
    }


}
```

Εικόνα 3.2.2.1

3.2.3 StoryTeller

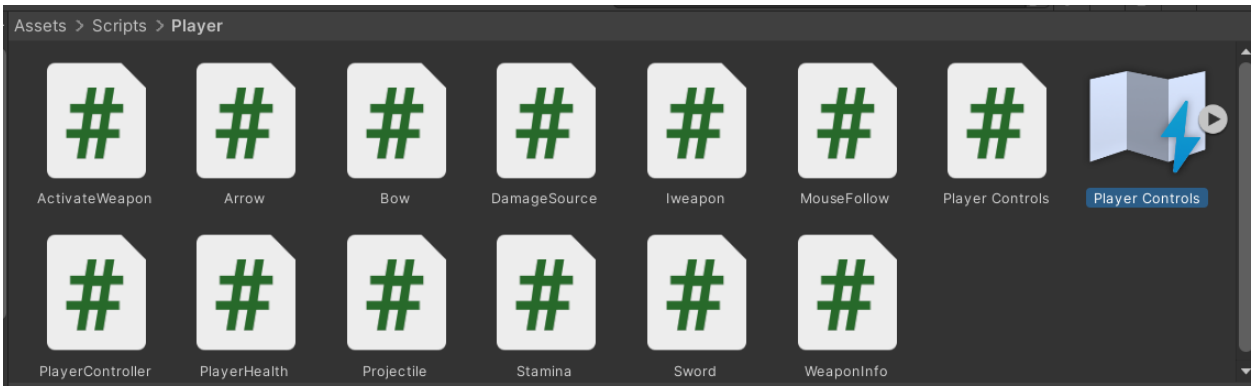
Και τέλος με το StoryTeller ανοίγουμε το panel με την ιστορία .

```
public class StoryTeller : MonoBehaviour
{
    public GameObject story;

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (!story.activeSelf)
            {
                story.SetActive(true);
            }
            else
            {
                story.SetActive(false);
            }

        }

    }

}
```

Εικόνα 3.2.3.1

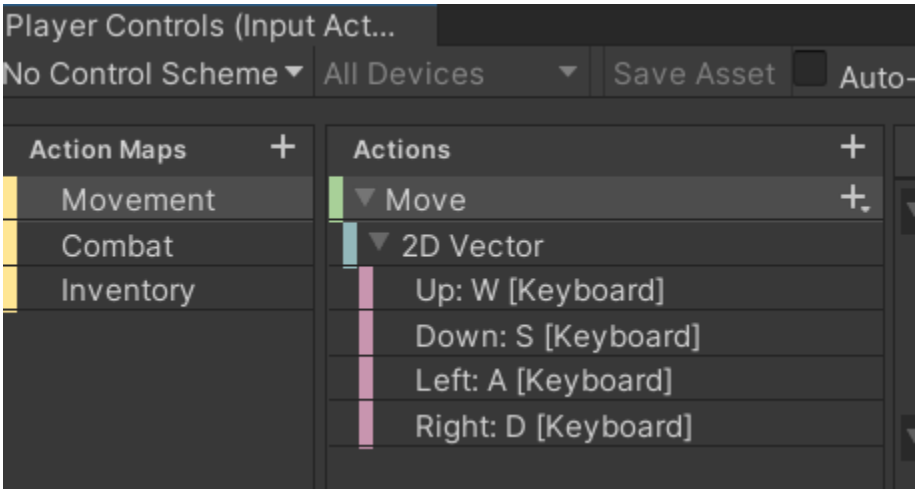
3.3 Φάκελος Player

Επόμενος φάκελος που θα αναλύσουμε είναι αυτός του player.



3.3.1 Player Cotrnol

Εδώ θα ξεκινήσουμε με το Player Control καθώς το script μπαίνει αυτόματα όταν βάλουμε τα controls τα οποία μας βοηθάνε να μπορούμε να χρησιμοποιήσουμε όλα τα πλήκτρα του πληκτρολογίου μας αλλά και το ποντίκι μας.



Εικόνα 3.3.1.1

3.3.2 WeaponInfo

Στην συνέχεια έχουμε το WeaponInfo το οποίο μας δείχνει πιο όπλο έχουμε και τις λεπτομερείς του.

```
[CreateAssetMenu(menuName ="New Weapon")]
public class WeaponInfo : ScriptableObject
{
    public GameObject weaponPrefab;
    public float weaponCooldown;
    public int weaponDamage;
    public float weaponRange;
}
```

Εικόνα 3.3.2.1

3.3.3 Sword

Έπειτα θα δούμε το script που αφορά το sword.

Αφού αρχικοποιήσουμε τα πάντα που χρειάζεται το σπαθί μας , cooldown , animation και collider

```
public class Sword : Singleton<Sword> , Iweapon
{
    [SerializeField] private float swordCD = .5f;
    [SerializeField] private WeaponInfo weaponInfo;

    private Animator myAnimator;
    private Transform weaponCollider;

    protected override void Awake()
    {
        base.Awake();
        myAnimator = GetComponent<Animator>();
    }

    private void Start()
    {
        weaponCollider = PlayerController.Instance.GetWeaponCollider();
    }

    private void Update()
    {
        MouseFolow();
    }

    public WeaponInfo GetWeaponInfo()
    {
        return weaponInfo;
    }
```

Εικόνα 3.3.3.1

Μετά έχουμε τον κώδικα της επίθεσης

```
public  void Attack()
{
    PlayerController.Instance.myAnimator.SetTrigger("Attack");
    myAnimator.SetTrigger("Attack");
    weaponCollider.gameObject.SetActive(true);
}

public void DoneAttackingEvent()
{
    weaponCollider.gameObject.SetActive(false);
}
```

Εικόνα 3.3.3.2

Οπού όταν πατάμε το κουμπί της επίθεσης ενεργοποιείται το animation και το collider σε περίπτωση που χτυπήσει κάτι.

3.3.4 Bow n Arrow

Επόμενος κώδικας που θα δούμε είναι του bow και του arrow δηλαδή του 2ου όπλου μας , το οποίο είναι χωρισμένο σε 2 script. Και τα 2 script λειτουργούν αντίστοιχά με το sword , μόνο που η διαφορά είναι ότι το collider του bow βρίσκεται στο arrow, όπου μέσω αυτού επιτυγχάνεται η επίθεση.

Κώδικας bow:

```
public class Bow : MonoBehaviour , Iweapon
{
    [SerializeField] private WeaponInfo weaponInfo;
    [SerializeField] private GameObject arrowPrefab;
    [SerializeField] private Transform arrowSpawnPoint;

    readonly int FIRE_HASH = Animator.StringToHash("Fire");

    private Animator myAnimator;

    private void Awake()
    {
        myAnimator = GetComponent<Animator>();
    }
    public void Attack()
    {
        myAnimator.SetTrigger(FIRE_HASH);
        GameObject newArrow = Instantiate(arrowPrefab, arrowSpawnPoint.position, ActivateWeapon.Instance.transform.rotation);
        newArrow.GetComponent<Arrow>().UpdateWeaponInfo(weaponInfo);
    }

    public WeaponInfo GetWeaponInfo()
    {
        return weaponInfo;
    }
```

Εικόνα 3.3.4.1

Και κώδικας Arrow:

```
public class Arrow : MonoBehaviour
{
    [SerializeField] private float moveSpeed = 22f;
    [SerializeField] private GameObject particleOnHitPrefabVFX;

    private WeaponInfo weaponInfo;
    private Vector3 startPosition;

    private void Start()
    {
        startPosition = transform.position;
    }

    private void Update()
    {
        MoveProjectile();
        DetectFireDistance();
    }

    public void UpdateWeaponInfo(WeaponInfo weaponInfo)
    {
        this.weaponInfo = weaponInfo;
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        EnemyHealth enemyHealth = other.gameObject.GetComponent<EnemyHealth>();
        Indestructuble indestructible = other.gameObject.GetComponent<Indestructuble>();

        if (!other.isTrigger && (enemyHealth || indestructible))
        {
            enemyHealth?.TakeDamage(1);
            Instantiate(particleOnHitPrefabVFX, transform.position, transform.rotation);
            Destroy(gameObject);
        }
    }
```

Εικόνα 3.3.4.2

```
private void DetectFireDistance()
{
    if (Vector3.Distance(transform.position, startPosition) > weaponInfo.weaponRange
    {
        Destroy(gameObject);
    }
}

private void MoveProjectile()
{
    transform.Translate(Vector3.right * Time.deltaTime * moveSpeed);
}
```

Εικόνα 3.3.4.3

3.3.5 ActivateWeapon n DamageSource

Τέλος ότι αφορά τα όπλα του παίχτη έχουμε επίσης τα script ActivateWeapon και DamageSource τα όποια ενεργοποιούνται όταν χρησιμοποιούμε τα όπλα και όταν πετυχαίνουμε κάποιον αντίπαλο αντίστοιχά

```
public class DamageSource : MonoBehaviour
{
    private int damageAmnt;

    private void Start()
    {
        MonoBehaviour currentActiveWeapon = ActivateWeapon.Instance.CurrentActiveWeapon;
        damageAmnt = (currentActiveWeapon as Iweapon).GetWeaponInfo().weaponDamage;
    }



    private void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.gameObject.GetComponent<EnemyHealth>())
        {
            EnemyHealth enemyHealth = collider.gameObject.GetComponent<EnemyHealth>();
            enemyHealth.TakeDamage(damageAmnt);
        }

    }
}
```

Εικόνα 3.3.5.1

```
public class ActivateWeapon : Singleton<ActivateWeapon>
{

    public MonoBehaviour CurrentActiveWeapon { get; private set; }

    private PlayerControls playerControls;
    private float timeBetweenAttacks;

    private bool attackButtonDown, isAttacking = false;

    protected override void Awake()
    {
        base.Awake();

        playerControls = new PlayerControls();

    }

    private void OnEnable()
    {
        playerControls.Enable();
    }

    private void Start()
    {
        playerControls.Combat.Attack.started += _ => StartAttacking();
        playerControls.Combat.Attack.canceled += _ => StopAttacking();

        AttackCooldown();

    }

    private void Update()
    {
        Attack();
    }
```

Εικόνα 3.3.5.2

```
public void NewWeapon(MonoBehaviour newWeapon)
{
    CurrentActiveWeapon = newWeapon;
    AttackCooldown();
    timeBetweenAttacks = (CurrentActiveWeapon as Iweapon).GetWeaponInfo().weaponCooldown;
}

public void WeaponNull()
{
    CurrentActiveWeapon = null;
}


private void AttackCooldown()
{
    isAttacking = true;
    StopAllCoroutines();
    StartCoroutine(TimeBetweenAttacksRoutine());
}

private IEnumerator TimeBetweenAttacksRoutine()
{
    yield return new WaitForSeconds(timeBetweenAttacks);
    isAttacking = false;
}

private void StartAttacking()
{
    attackButtonDown = true;
}

private void StopAttacking()
{
    attackButtonDown = false;
}
```

Εικόνα 3.3.5.3

```
private void Attack()
{
    S
    if(attackButtonDown && !isAttacking && CurrentActiveWeapon)
    {
        AttackCooldown();
        (CurrentActiveWeapon as Iweapon).Attack();

    }
}
```

Εικόνα 3.3.5.4

Και αφού τελειώσαμε ότι αφορά τα scripts για τα weapon ας δούμε και του παίχτη μας.

3.3.6 Stamina

Ας ξεκινήσουμε με το script που αφορα το stamina δηλαδη το πόσες φορές μπορούμε να κάνουμε dash , και με τα cool down και με το auto refresh.

```
public class Stamina : Singleton<Stamina>
{
    public int CurrentStamina { get; private set; }

    [SerializeField] private Sprite fullStaminaImage, emptyStaminaImage;
    [SerializeField] private int timeBetweenStaminaRefresh = 3;

    private Transform staminaContainer;
    private int startingStamina = 3;
    private int maxStamina;
    const string STAMINA_CONTAINER_TEXT = "Stamina Container";

    protected override void Awake()
    {
        base.Awake();

        maxStamina = startingStamina;
        CurrentStamina = startingStamina;
    }

    private void Start()
    {
        staminaContainer = GameObject.Find(STAMINA_CONTAINER_TEXT).transform;
    }

    public void UseStamina()
    {
        CurrentStamina--;
        UpdateStaminaImage();
    }

    public void RefreshStamina()
    {
        if(CurrentStamina < maxStamina)
        {
            CurrentStamina++;
        }
```

Εικόνα 3.3.6.1

```
    UpdateStaminaImage();
}

private IEnumerator RefreshStaminaRoutine()
{
    while (true)
    {
        yield return new WaitForSeconds(timeBetweenStaminaRefresh);
        RefreshStamina();
    }
}

private void UpdateStaminaImage()
{
    for (int i = 0; i < maxStamina; i++)
    {
        if(i <= CurrentStamina - 1)
        {
            staminaContainer.GetChild(i).GetComponent<Image>().sprite = fullStaminaImage;
        }
        else
        {
            staminaContainer.GetChild(i).GetComponent<Image>().sprite = emptyStaminaImage;
        }
    }

    if (CurrentStamina < maxStamina)
    {
        StopAllCoroutines();
        StartCoroutine(RefreshStaminaRoutine());
    }
}
```

Εικόνα 3.3.6.2

3.3.7 PlayerHealth

Στην συνέχεια πάμε στο script Που αφορά την ζωή του παίχτη μας . Δηλαδή το πόση ζωή έχει το να κάνει regen την ζωή του και το να πεθαίνει.

```
public class PlayerHealth : Singleton<PlayerHealth>
{
    public bool isDead { get; private set; }
    [SerializeField] private int maxHealth = 4;
    [SerializeField] private float knockThrustBackAmount = 10f;
    [SerializeField] private float damageRecoveryTime = 1f;

    private Slider healthSlider;
    private int currentHealth;
    private bool canTakeDamage = true;
    private Knockback knockback;
    private Flash flash;
    const string HEALTH_SLIDER_TEXT = "Heart Slider";
    const string TOWN_TEXT = "MainScene";
    readonly int DEATH_HASH = Animator.StringToHash("Death");

    protected override void Awake()
    {
        base.Awake();
        flash = GetComponent<Flash>();
        knockback = GetComponent<Knockback>();
    }

    private void Start()
    {
        isDead = false;
        currentHealth = maxHealth;

        UpdateHealthSlider();
    }
```

Εικόνα 3.3.7.1

```
private void OnCollisionStay2D(Collision2D collision)
{
    EnemyAI enemy = collision.gameObject.GetComponent<EnemyAI>();

    if (enemy)
    {
        TakeDamage(1, collision.transform);
    }


}

public void HealPlayer()
{
    if(currentHealth < maxHealth)
    {
        currentHealth += 1;
        UpdateHealthSlider();
    }
}

public void TakeDamage(int damage , Transform hitTransform)
{
    if (!canTakeDamage) { return; }

    ScreenShackingManage.Instance.ShakeScreen();
    knockback.GetKnockedBack(hitTransform, knockThrustBackAmount);
    StartCoroutine(flash.FlashRoutine());

    canTakeDamage = false;
    currentHealth -= damage;
    StartCoroutine(DamageRecoveryRoutine());

    UpdateHealthSlider();
    CheckIfPlayerDeath();
}
```

Εικόνα 3.3.7.2

```
private void CheckIfPlayerDeath()
{
    if (currentHealth <= 0 && !isDead)
    {
        isDead = true;
        Destroy(ActivateWeapon.Instance.gameObject);
        currentHealth = 0;
        GetComponent<Animator>().SetTrigger(DEATH_HASH);
        StartCoroutine(DeathLoadSceneRoutine());
    }
}

private IEnumerator DeathLoadSceneRoutine()
{
    yield return new WaitForSeconds(1.5f);
    Destroy(gameObject);
    SceneManager.LoadScene(TOWN_TEXT);

}

private IEnumerator DamageRecoveryRoutine()
{
    yield return new WaitForSeconds(damageRecoveryTime);
    canTakeDamage = true;
}

private void UpdateHealthSlider()
{
    if(healthSlider == null)
    {
        healthSlider = GameObject.Find(HEALTH_SLIDER_TEXT).GetComponent<Slider>();
    }

    healthSlider.maxValue = maxHealth;
    healthSlider.value = currentHealth;
}
```

Εικόνα 3.3.7.3

3.3.8 PlayerCotrnoller

Στην συνέχεια έχουμε το PlayerController Που ουσιαστικά ελέγχει συνέχεια όλες τις δραστηριότητες του παίχτη , δηλαδή τα animation του , το τι input θα κάνει ο χρήστης , το dash και γενικά οτιδήποτε μπορεί να κάνει ο ήρωας μας.

```
public class PlayerController : Singleton<PlayerController>
{
    public bool FacingLeft { get { return facingLeft;  } }

    [SerializeField] private float moveSpeed = 1f;
    [SerializeField] private float dashSpeed = 4f;
    [SerializeField] private TrailRenderer trailRenderer;
    [SerializeField] private Transform weaponCollider;


    //for move
    private PlayerControls playerControls;
    private Vector2 movement;
    private Rigidbody2D rb;
    //for animations
    public Animator myAnimator;
    private SpriteRenderer mySpriteRender;
    private Knockback knockback;
    private float startingMovementSpeed;

    private bool facingLeft = false;
    private bool isDashing = false;

    public string type;

    protected override void Awake()
    {
        base.Awake();
        playerControls = new PlayerControls();
        rb = GetComponent<Rigidbody2D>();
        myAnimator = GetComponent<Animator>();
        mySpriteRender = GetComponent<SpriteRenderer>();
        knockback = GetComponent<Knockback>();

    }
```

Εικόνα 3.3.8.1

```
private void Start()
{
    playerControls.Combat.Dash.performed += _ => Dash();
    startingMovementSpeed = moveSpeed;

    ActiveInventory.Instance.EquipStartingWeapon();

}

private void OnEnable()
{
    playerControls.Enable();
}
private void OnDisable()
{
    playerControls.Disable();
}

private void Update()
{
    PlayerInput();
}

private void FixedUpdate()
{
    AdjustPlayerDirection();
    Move();
}

public Transform GetWeaponCollider()
{
    return weaponCollider;
}
```

Εικόνα 3.3.8.2

```
private void PlayerInput()
{
    movement = playerControls.Movement.Move.ReadValue<Vector2>();
    myAnimator.SetFloat("moveX", movement.x);
    myAnimator.SetFloat("moveY", movement.y);

}

private void Move()
{
    if (knockback.GettingKnockedBack || PlayerHealth.Instance.isDead)
    {
        return;
    }
    rb.MovePosition(rb.position + movement * (moveSpeed * Time.fixedDeltaTime));
}

private void AdjustPlayerDirection()
{
    Vector3 mousePos = Input.mousePosition;
    Vector3 playerScreenPoint = Camera.main.WorldToScreenPoint(transform.position);

    if(mousePos.x < playerScreenPoint.x)
    {
        mySpriteRender.flipX = true;
        facingLeft = true;
    }
    else
    {
        mySpriteRender.flipX = false;
        facingLeft = false;
    }
}
```

Εικόνα 3.3.8.3

```
private void Dash()
{
    if (!isDashing && Stamina.Instance.CurrentStamina >0)
    {
        Stamina.Instance.UseStamina();
        isDashing = true;
        moveSpeed *= dashSpeed;
        trailRenderer.emitting = true;
        StartCoroutine(EndDashRoutine());
    }
}

private IEnumerator EndDashRoutine()
{
    float dashTime = .2f;
    float dashCD = .25f;
    yield return new WaitForSeconds(dashTime);
    moveSpeed = startingMovementSpeed;
    trailRenderer.emitting = false;
    yield return new WaitForSeconds(dashCD);
    isDashing = false;
}
```

Εικόνα 3.3.8.4

3.3.9 MouseFollow

Και τελευταίο script στην κατηγορία Player είναι το MouseFollow που απλά ο χαρακτήρας μας κοιτάει συνέχεια εκεί που έχουμε το ποντίκι

```
public class MouseFollow : MonoBehaviour
{
    private void Update()
    {
        FaceMouse();
    }

    private void FaceMouse()
    {
        Vector3 mousePosition = Input.mousePosition;
        mousePosition = Camera.main.ScreenToWorldPoint(mousePosition);

        Vector2 direction = transform.position - mousePosition;

        transform.right = -direction;
    }
}
```

Εικόνα 3.3.9.1

3.4 Φάκελος Scene

Επόμενος φάκελος που θα δούμε είναι αυτός του Scene που έχει μέσα script που αφορούν το περιβάλλον του παιχνιδιού.

3.4.1 Transparency

Δηλαδή το πρώτο script που θα δούμε είναι αυτό του Transparency που μέσω αυτού μπορούμε και φαινόμαστε πίσω από κάποια object πχ τα δέντρα.

```
public class Transparency : MonoBehaviour
{
    [Range(0, 1)]
    [SerializeField] private float transparencyAmount = 0.8f;
    [SerializeField] private float fadeTime = .4f;

    private SpriteRenderer spriteRenderer;

    private void Awake()
    {
        spriteRenderer = GetComponent<SpriteRenderer>();
    }


    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.GetComponent<PlayerController>())
        {
            if (spriteRenderer)
            {
                StartCoroutine(FadeRoutine(spriteRenderer, fadeTime, spriteRenderer.color.a, transparencyAmount));
            }
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.gameObject.GetComponent<PlayerController>())
        {
            if (spriteRenderer)
            {
                StartCoroutine(FadeRoutine(spriteRenderer, fadeTime, spriteRenderer.color.a, 1f));
            }
        }
    }

    private IEnumerator FadeRoutine(SpriteRenderer spriteRenderer , float fadeTime , float startValue , float targetTransparency)
    {
        float elapsedTime = 0;
        while (elapsedTime < fadeTime)
        {
            elapsedTime += Time.deltaTime;
            float newAlpha = Mathf.Lerp(startValue, targetTransparency, elapsedTime / fadeTime);
            spriteRenderer.color = new Color(spriteRenderer.color.r, spriteRenderer.color.g, spriteRenderer.color.b, newAlpha);
            yield return null;
        }
    }
}
```

Εικόνα 3.4.1.1

Εικόνα 3.4.1.2

3.4.2 ScreenShacikngManage

Επόμενο είναι το ScreenShackingManage που όταν μας χτυπάει ο εχθρός , κουνιέται η οθόνη

```
public class ScreenShackingManage : Singleton<ScreenShackingManage>
{
    private CinemachineImpulseSource source;

    protected override void Awake()
    {
        base.Awake();

        source = GetComponent<CinemachineImpulseSource>();
    }

    public void ShakeScreen()
    {
        source.GenerateImpulse();
    }
}
```

Εικόνα 3.4.2.1

3.4.3 Destructable

Μετά έχουμε το Destructable που μέσω αυτου καταστρέφουμε τα αντικείμενα όπως τα κουτιά.

```
public class Destuctable : MonoBehaviour
{
    [SerializeField] private GameObject destroyVFX;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.GetComponent<DamageSource>() || collision.gameObject.GetComponent<Projectile>() || collision.gameObject.GetComponent<Arrow>())
        {
            GetComponent<PickUpSpawner>().DropItems();
            Instantiate(destroyVFX, transform.position, Quaternion.identity);
            Destroy(gameObject);
        }
    }
}
```

Εικόνα 3.4.3.1

3.4.4 EconomyManager

Το EconomyManager που μας μετράει τα κέρματα

```
public class EconomyManager : Singleton<EconomyManager>
{
    private TMP_Text goldText;
    private int currentGold = 0;

    const string COIN_AMNT_TEXT = "Gold Amnt Text";

    public void UpdateCurrentGold()
    {
        currentGold += 1;

        if(goldText == null)
        {
            goldText = GameObject.Find(COIN_AMNT_TEXT).GetComponent<TMP_Text>();
        }

        goldText.text = currentGold.ToString("D3");
    }
}
```

Εικόνα 3.4.4.1

Και έπειτα έχουμε 2 ζευγάρια από script το πρώτο είναι το Pickup και το PickupSpawner.

3.4.5 Pickup

Το Pickup είναι το script το οποίο δημιουργεί τα αντικείμενα που μπορούμε να συλλέξουμε , δηλαδή την καρδιά , το stamina και τα κέρματα και όταν τα πλησιάζουμε αυτά έρχονται προς εμάς και μπορούμε να τα συλλέξουμε.

```
public class Pickup : MonoBehaviour
{
    private enum PickUpType {
        GoldCoin,
        Stamina,
        Health,

    }


    [SerializeField] private PickUpType pickUpType;
    [SerializeField] private float pickUpDistance = 5f;
    [SerializeField] private float accelarationRate = .2f;
    [SerializeField] private float moveSpeed = 3f;
    [SerializeField] private AnimationCurve animCruve;
    [SerializeField] private float heightY = 1.5f;
    [SerializeField] private float popDuration = 1f;
    private Vector3 moveDir;
    private Rigidbody2D rb;

    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    private void Start()
    {
        StartCoroutine(AnimCruveSpawnRoutine());
    }

    private void Update()
    {
        Vector3 playerPos = PlayerController.Instance.transform.position;

        if(Vector3.Distance(transform.position , playerPos)< pickUpDistance)
        {
            moveDir = (playerPos - transform.position).normalized;
            moveSpeed += accelarationRate;
        }
        else
        {
            moveDir = Vector3.zero;
            moveSpeed = 0;
        }
    }
```

Εικόνα 3.4.5.1

```
private void FixedUpdate()
{
    rb.velocity = moveDir * moveSpeed * Time.deltaTime;
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.GetComponent<PlayerController>())
    {
        DetectPickupType();
        Destroy(gameObject);
    }
}

private IEnumerator AnimCruveSpawnRoutine()
{
    Vector2 startPoint = transform.position;
    float randomX = transform.position.x + Random.Range(-2f, 2f);
    float randomY = transform.position.y + Random.Range(-1f, 1f);

    Vector2 endPoint = new Vector2(randomX, randomY);

    float timePassed = 0f;

    while (timePassed < popDuration)
    {
        timePassed += Time.deltaTime;
        float linearT = timePassed / popDuration;
        float heightT = animCruve.Evaluate(linearT);
        float height = Mathf.Lerp(0f, heightY, heightT);

        transform.position = Vector2.Lerp(startPoint, endPoint, linearT) + new Vector2(0f, height);
        yield return null;
    }
}
```

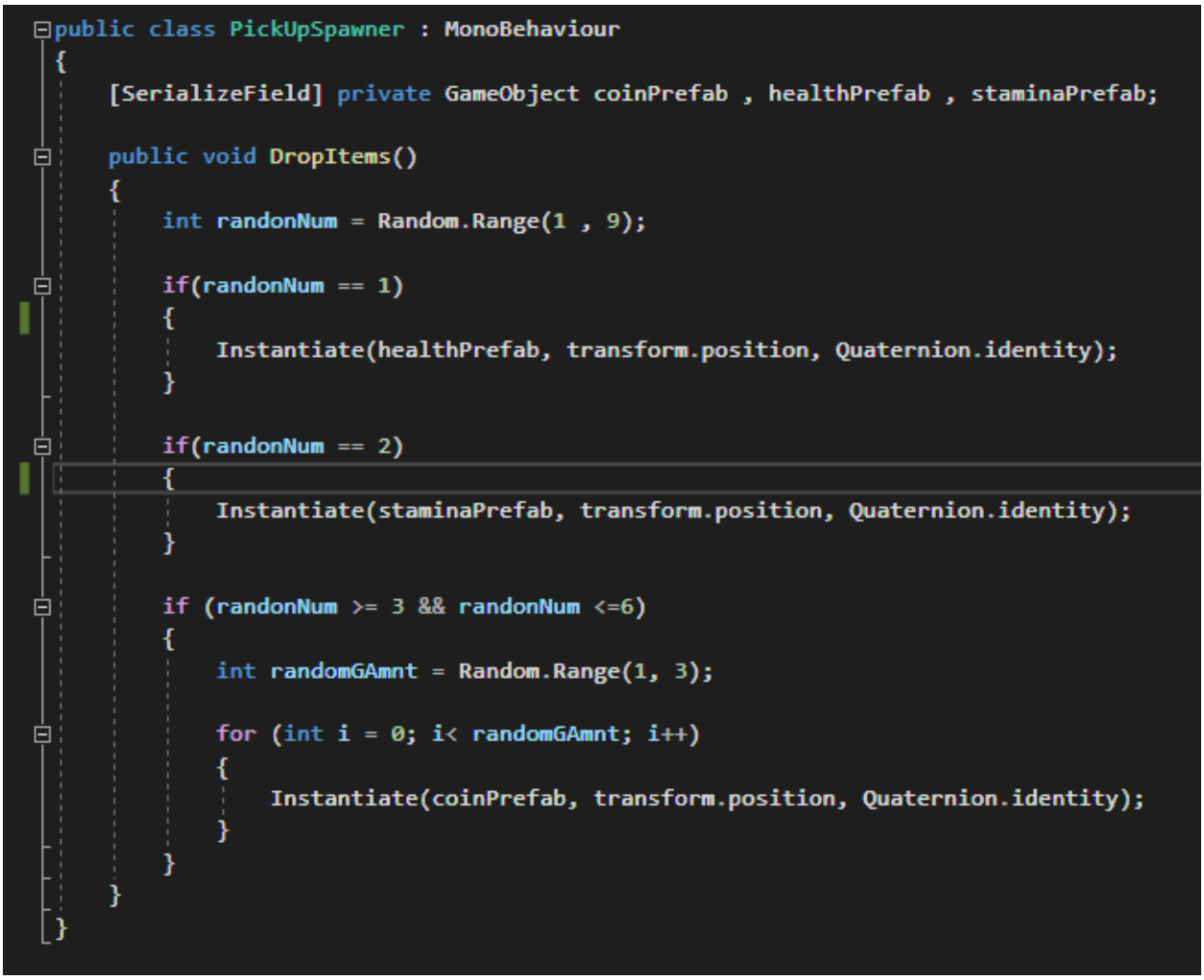
Εικόνα 3.4.5.2

```
private void DetectPickupType()
{
    switch (pickUpType)
    {
        case PickUpType.GoldCoin:
            EconomyManager.Instance.UpdateCurrentGold();

            break;
        case PickUpType.Health:
            PlayerHealth.Instance.HealPlayer();

            break;
        case PickUpType.Stamina:
            Stamina.Instance.RefreshStamina();
            break;
    }
}
```

Εικόνα 3.4.5.3

3.4.6 PickUpSpawner

Και το PickUpSpawner που αυτό βάση κάποιον πιθανοτήτων , μας δίνει ένα απτά 3 αντικείμενα που μπορούμε να συλλέξουμε , όταν σπάμε πχ ένα κουτί .



```
public class PickUpSpawner : MonoBehaviour
{
    [SerializeField] private GameObject coinPrefab , healthPrefab , staminaPrefab;

    public void DropItems()
    {
        int randonNum = Random.Range(1 , 9);

        if(randonNum == 1)
        {
            Instantiate(healthPrefab, transform.position, Quaternion.identity);
        }

        if(randonNum == 2)
        {
            Instantiate(staminaPrefab, transform.position, Quaternion.identity);
        }

        if (randonNum >= 3 && randonNum <=6)
        {
            int randomGAmnt = Random.Range(1, 3);

            for (int i = 0; i< randomGAmnt; i++)
            {
                Instantiate(coinPrefab, transform.position, Quaternion.identity);
            }
        }
    }
}
```

Εικόνα 3.4.6.1

3.4.7 AvitveInventory n InvetorySlot

Και τελευταίο ζευγάρι είναι το ActiveInventory και το InventorySlot τα οποια είναι αυτά που μας βοηθάνε να αλλάζουμε weapons και στην οθόνη μας

Εικόνα 3.4.7.1

```
public class ActiveInventory : Singleton<ActiveInventory>
{
    private int activeSlotIndexNum = 0;

    private PlayerControls playerControls;


    protected override void Awake()
    {
        base.Awake();
        playerControls = new PlayerControls();
    }

    private void Start()
    {
        playerControls.Inventory.ChangeInventory.performed += ctx => ToggleActiveSlot((int)ctx.ReadValue<float>());

    }

    private void OnEnable()
    {
        playerControls.Enable();
    }

    public void EquipStartingWeapon()
    {
        ToggleActiveHighlight(0);
    }

    private void ToggleActiveSlot(int numVlaue)
    {
        ToggleActiveHighlight(numVlaue - 1);
    }

    private void ToggleActiveHighlight (int indexNum)
    {
        activeSlotIndexNum = indexNum;

        foreach(Transform inventorySlot in this.transform)
        {
            inventorySlot.GetChild(0).gameObject.SetActive(false);
        }

        this.transform.GetChild(indexNum).GetChild(0).gameObject.SetActive(true);

        ChangeActiveWeapon();
    }
```

Εικόνα 3.4.7.2

```
private void ChangeActiveWeapon()
{

    if (ActivateWeapon.Instance.CurrentActiveWeapon != null)
    {
        Destroy(ActivateWeapon.Instance.CurrentActiveWeapon.gameObject);
    }
    Transform childTransform = transform.GetChild(activeSlotIndexNum);
    InventorySlot inventorySlot = childTransform.GetComponentInChildren<InventorySlot>();
    WeaponInfo weaponInfo = inventorySlot.GetWeaponInfo();
    GameObject weaponToSpawn = weaponInfo.weaponPrefab;

    if (weaponInfo == null)
    {
        ActivateWeapon.Instance.WeaponNull();
        return;
    }

    GameObject newWeapon = Instantiate(weaponToSpawn, ActivateWeapon.Instance.transform);

    ActivateWeapon.Instance.NewWeapon(newWeapon.GetComponent<MonoBehaviour>());
}
```

Εικόνα 3.4.7.3

```
public class InventorySlot : MonoBehaviour
{
    [SerializeField] private WeaponInfo weaponInfo;


    public WeaponInfo GetWeaponInfo()
    {
        return weaponInfo;
    }
}
```

Εικόνα 3.4.7.4

3.5 Φάκελος Management

3.5.1 AreaEntrance , AreaExit ,UIFade, Scenemanagment

Και τελευταίος φάκελος σε αυτό το project είναι το management που εδώ έχουμε τα scripts Που αλλάζουν πίστες , δηλαδή τα , AreaEntrance , AreaExit , UIFade, και SceneManagemnet , οπού τα πρώτα δυο αλλάζουμε την σκηνή με τα Portals που έχουμε σε συνδυασμό με το SceneManagment και το UIFade είναι αυτό που κάνει το fade από αλλαγή σε αλλαγή.

```
public class AreaExit : MonoBehaviour
{
    [SerializeField] private string sceneToLoad;
    [SerializeField] private string sceneTransitionName;

    private float waitToLoadTime = 1f;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.GetComponent<PlayerController>())
        {
            SceneManagement.Instance.SetTransitionName(sceneTransitionName);
            UIFade.Instance.FadeToBlack();
            StartCoroutine(LoadSceneRoutine());
        }
    }


    private IEnumerator LoadSceneRoutine()
    {
        while (waitToLoadTime >= 0)
        {
            waitToLoadTime -= Time.deltaTime;
            yield return null;
        }

        SceneManager.LoadScene(sceneToLoad);
    }
}
```

Εικόνα 3.5.1.1

```
public class AreaEntrance : MonoBehaviour
{
    [SerializeField] private string transitionName;

    private void Start()
    {
        if(transitionName == SceneManagement.Instance.SceneTransitionName)
        {
            PlayerController.Instance.transform.position = this.transform.position;
            CameraController.Instance.SetPlayerCameraFollow();
            UIFade.Instance.FadeToClear();
        }
    }
}
```

Εικόνα 3.5.1.2

```
public class SceneManagement : Singleton<SceneManagement>
{
    public string SceneTransitionName { get; private set; }

    public void SetTransitionName(string sceneTransitionName)
    {
        this.SceneTransitionName = sceneTransitionName;
    }
}
```

Εικόνα 3.5.1.3

```
public class UIFade : Singleton<UIFade>
{
    [SerializeField] private Image fadeScreen;
    [SerializeField] private float fadeSpeed = 1f;

    private IEnumerator fadeRoutine;

    public void FadeToBlack()
    {
        if(fadeRoutine != null)
        {
            StopCoroutine(fadeRoutine);

        }

        fadeRoutine = FadeRoutine(1);
        StartCoroutine(fadeRoutine);
    }

    public void FadeToClear()
    {
        if(fadeRoutine != null)
        {
            StopCoroutine(fadeRoutine);
        }

        fadeRoutine = FadeRoutine(0);
        StartCoroutine(fadeRoutine);
    }

    private IEnumerator FadeRoutine(float targetAlpha)
    {
        while (!Mathf.Approximately(fadeScreen.color.a , targetAlpha))
        {
            float alpha = Mathf.MoveTowards(fadeScreen.color.a, targetAlpha, fadeSpeed * Time.deltaTime);
            fadeScreen.color = new Color(fadeScreen.color.r, fadeScreen.color.g, fadeScreen.color.b, alpha);
            yield return null;
        }
    }
}
```

Εικόνα 3.5.1.4

3.5.2 CameraCotrnoller

Μετά έχουμε το cameraController που βάση αυτού η κάμερα ακολουθάει τον παίχτη

```
public class CameraController : Singleton<CameraController>
{
    private CinemachineVirtualCamera cinemachineVirtualCamera;

    private void Start()
    {
        SetPlayerCameraFollow();
    }

    public void SetPlayerCameraFollow()
    {
        cinemachineVirtualCamera = FindAnyObjectByType<CinemachineVirtualCamera>();
        cinemachineVirtualCamera.Follow = PlayerController.Instance.transform;
    }
}
```

Εικόνα 3.5.2.1

3.5.3 Singleton

Και τέλος το Singleton το οποίο είναι ένα υποστηρικτικό script για πάρα πολλά άλλα καθώς με αυτό μπορούμε να κρατάμε αντικείμενα από αλλαγή σκηνής σε σκηνή χωρίς να καταστρέφονται και όπως είδαμε σε αλλά script αντί να έχουν την MonoBehaviour στην αρχή τους έχουν σε singleton τον εαυτό τους.

```
public class Singleton<T> : MonoBehaviour where T : Singleton<T>
{
    private static T instance;
    public static T Instance { get { return instance; } }

    protected virtual void Awake()
    {
        if (instance != null && this.gameObject != null)
        {
            Destroy(this.gameObject);
        }
        else
        {
            instance = (T)this;
        }
        if (!gameObject.transform.parent)
        {
            DontDestroyOnLoad(gameObject);
        }

    }
```

Εικόνα 3.5.3.1

Συμπέρασμα

Εν τέλη συμπεραίνουμε ότι με την κατάλληλη χρήση των εργαλείων του Unity αλλά και με την σωστή χρήση κώδικα πάνω στα διαφορά Objects που έχουμε δημιουργήσει μπορούμε να φτιάξουμε ένα πλήρες λειτουργικό παιχνίδι με σκοπό πέρα την ευχαρίστηση του παίχτη και την εκμάθηση του .

Βιβλιογραφία

Για την συγκεκριμένη εργασία χρησιμοποιήθηκαν έτοιμα assets από το site

- https://itch.io/game-assets/tag-2d
- https://assetstore.unity.com/2d .