ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας:	Ανάπτυξη Android δυσδιάστατου παιχνιδιού με την Χρήση Firebase και Unity Android 2d Game Using Firebase and Unity
Ονοματεπώνυμο:	Σωτήρης Χατζηκυριάκου
Πατρώνυμο:	Θωμάς
Αριθμός Μητρώο:	Π20011
Επιβλέπων:	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία Παράδοσης: Σεπτέμβριος 2024

Copyright ©

Ευχαριστίες

Για την υλοποίηση της παρούσας εργασίας, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου, κ Ευθύμιο Αλέπη, για την εμπιστοσύνη του και την καθοδήγηση που μου έχει δείξει.

Επίσης, θέλω να ευχαριστήσω τους συμφοιτητές μου για την άψογη συνεργασία που είχαμε καθ' όλη τη διάρκεια του ΠΜΣ, για την επιτυχή ολοκλήρωση όλων των εξαμήνων.

Εν κατά κλειδί, Θέλω να ευχαριστήσω την οικογένεια μου για την υποστήριξη που μου έχει δώσει καθ' όλη την διάρκεια μου στο ΠΜΣ.

Περίληψη:

Η παρούσα πτυχιακή εργασία έχει ως στόχο την υλοποίηση μίας εφαρμογής Android και συγκεκριμένα ενός παιχνιδιού πλατφόρμας, σε πραγματικό χρόνο για έναν χρήστη – παίχτη με την χρήση του Firebase. Το παιχνίδι αυτό αφορά τον ανταγωνισμό μεταξύ πολλαπλών χρηστών μέσα από πολλά δύσκολα επίπεδα. Αρχικά ο χρήστης για να ζήσει αυτήν την περιπέτεια πρέπει να δημιουργήσει ένα λογαριασμό, αυτός ο λογαριασμός αποθηκεύετε αυτόματα στην Firebase. Όταν ο χρήστης συνδεθεί με τον λογαριασμό του μπορεί να απολαύσει τα δύσκολα μονοπάτια που τον περιμένουν. Για να ισοβαθμήσω την δυσκολία του παιχνιδιού έχω φτιάξει σημεία ελέγχου και έχω δώσει στον παίχτη 3 καρδίες ζωής. Σε κάθε ένα επίπεδο ο παίχτης κρίνετε να αντιμετωπίσει διαφορές δυσκολίες όπως φωτιές, παγίδες βέλους και εχθρούς για να φτάσει στον κρυμμένο θησαυρό.

Abstract:

This thesis aims to implement an Android application and specifically a real time game platform for a user using Firebase. This game involves competition between multiple users through many challenging levels. Initially the user to experience this adventure must create an account, this account is automatically saved in Firebase. Once the user logs in with his account he can enjoy the challenging paths that await him. To balance the difficulty of the game I have made a checkpoint system, and I gave the player 3 hearts of life. In each level the player is judged to face different difficulties such as fires, arrow traps and enemies to reach the hidden treasure.

Πίνακας περιεχομένων

Κεφάλαιο 1 – **Εργαλεία Ανάπτυξης**

1.1 Game Engines

Το Game Engine είναι ένα λογισμικό που χρησιμοποιείται για την ανάπτυξη βιντεοπαιχνιδιών. Εμφανίστηκε κατά την διάρκεια της δεκαετίας του 1970 με 1980 όταν η βιομηχανία των παιχνιδιών άρχισε να κάνει τα πρώτα βήματα στην αγορά. Εκείνο τον καιρό κάθε παιχνίδι αναπτυσσόταν με μοναδικό κώδικα, από το μηδέν κάτι που ήταν χρονοβόρο και ακριβό. Κατά την διάρκεια της περιόδου οι προγραμματιστές συνειδητοποίησαν την αναγκαιότητα για ένα κοινό εργαλείο που θα μπορούσε να επαναχρησιμοποιηθεί σε πολλαπλά παιχνίδια. Το Game Engine άρχισε να αναπτύσσεται ευρύτερα στα τέλη της δεκαετίας του 1990 και τις αρχές του 2000 με αποτέλεσμα σήμερα να υπάρχουν πλήρως εξοπλισμένα εργαλεία. Παραδείγματα γνωστών Game Engines περιλαμβάνουν το Unity, το Unreal Engine και το Godot.

1.1.2 Unity

Το Unity πρωτοεμφανίστηκε το 2005 από την εταιρεία Unity Technologies με στόχο να προσφέρει όσο το δυνατόν εύκολη χρήση για τους προγραμματιστές ανεξαρτήτων παιχνιδιών. Με τον καιρό το Unity έγινε ένα από τα δημοφιλέστερα Game Engines σε ολόκληρο τον κόσμο χρησιμοποιούμενο για την ανάπτυξη και εξαγωγή παιχνιδιών 2D και 3D σε διάφορες πλατφόρμες όπως το Android, iOS, Windows και κονσόλες. Με το Unity οι προγραμματιστές μπορούν να δημιουργήσουν γραφικά, να διαχειριστούν το περιβάλλον του παιχνιδιού, να προσθέσουν τεχνητή νοημοσύνη και να ενσωματώσουν ήχο και animations. Η ενσωμάτωση ισχυρών εργαλείων όπως το Unity Asset Store επιτρέπει στους προγραμματιστές να αγοράζουν ή να κατεβάζουν assets κάτι που κάνει την ανάπτυξη του παιχνιδιού ταχύτερη και ευκολότερη στην χρήση. Σήμερα το Unity χρησιμοποιείται από ανεξάρτητους δημιουργούς όσο και μεγάλες εταιρείες καθιστώντας το ιδανικό για παιχνίδια κάθε είδους και μήκους.

Όπως έχω προαναφέρει επέλεξα το Unity για την ανάπτυξη του παιχνιδιού μου λόγω της ευκολίας και της ευελιξίας που παρέχει στην υποστήριξη πολλαπλών πλατφορμών. Βάση του Unity έχω δημιουργήσει ένα 2D παιχνίδι που μπορεί να λειτουργήσει σε Android και iOS συσκευές. Το Unity με βοηθά να τρέξω το παιχνίδι μου σε άλλες συσκευές αφού παρέχει Android και iOS προσομοιωτές. Επίσης με την χρήση του Unity Assets Store είχα την δυνατότητα να κατεβάσω textures & environments για την δημιουργία του παιχνιδιού μου.

1.2 Database

Η βάση δεδομένων ξεκίνησε πριν την εμφανίσει των υπολογιστών. Παλαιότερα τα στοιχεία αποθηκεύονταν σε αρχεία, βιβλιοθήκες και ντουλάπια. Κατά την έναρξη της δεκαετίας του 1960 οι πρώτες υπολογιστικές βάσεις δεδομένων εμφανίστηκαν αφού ήταν οικονομικά αποδοτικές για ιδιωτικές οργανώσεις. Το 1970 πείρε την σκυτάλη ο E.F. Codd ο οποίος εισήγαγε το σχεσιακό μοντέλο το οποίο άλλαξε ριζικά τον τρόπο σκέψεις για τις βάσεις δεδομένων. Το μοντέλο αυτό αποσύνδεε το σχήμα της βάσης δεδομένων από την φυσική αποθήκευση των πληροφοριών. Έπειτα το 1980 η γλώσσα SQL έγινε η βάση για την αναζήτηση και την διαχείριση των δεδομένων καθώς αύξησε σε πωλήσεις την αγορά βάσεων δεδομένων. Κατά την διάρκεια του 2000 οι σχεσιακές βάσεις δεδομένων γνωστές ως NoSQL (π.χ. MongoDB και Firebase) έγιναν δημοφιλής χρησιμοποιώντας διαφορετικές γλώσσες αναζήτησης και μοντέλα δεδομένων. Σήμερα οι βάσεις δεδομένων συνεχίζουν να εξελίσσονται με την εμφάνιση νέων τεχνολογιών όπως οι βάσεις δεδομένων Cloud και οι κατανεμημένες βάσεις δεδομένων που επιτρέπουν την ευελιξία διαχείρισης μεγάλων ποσοστών δεδομένων.

1.2.1 Firebase

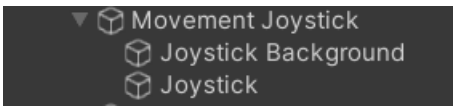
Η Firebase ξεκίνησε ως μια νέα επιχειρήσει με το όνομα Envolve που ιδρύθηκε το 2011 από τους James Tamplin και Andrew Lee. Αρχικά η Envolve παρείχε μια API για την ενσωμάτωση λειτουργιών σε διαδικτυακές συνομιλίες σε ιστοσελίδες. Ωστόσο ο James Tamplin και ο Andrew Lee ανακάλυψαν ότι το Envolve χρησιμοποιούταν για την μετάδοση δεδομένων εφαρμογών που δεν ήταν μηνύματα συνομιλίας. Αυτό οδήγησε τους ιδρυτές να δημιουργήσουν την Firebase ως ξεχωριστή εταιρεία το 2011. Η πρώτη δημόσια κυκλοφορία έγινε τον Απρίλιο του 2012 με το προϊόν Firebase Realtime Database που συγχρόνιζε δεδομένα εφαρμογών σε iOS, Android και Web συσκευές και τα αποθήκευε στο Cloud της Firebase. Έπειτα το 2014 η εταιρεία αγοραστικέ από την Google. Από τότε η Firebase έχει εξελιχθεί σε μια ολοκληρωμένη πλατφόρμα ανάπτυξης εφαρμογών προσφέροντας υπηρεσίες όπως Firebase Hosting, Firebase Authentication, Firebase Cloud Messaging και πολλά άλλα. Σήμερα η Firebase χρησιμοποιείται για την ανάπτυξη κινητών και Web εφαρμογών. Έχω επιλέξει την Firebase γιατί μπορεί εύκολα να ενσωματωθεί με το Unity καθιστώντας την ιδανική επιλογή για παιχνίδια που απαιτούν διαδικτυακή σύνδεση. Με την Firebase μπορώ εύκολα να διαχειριστώ τα δεδομένα των χρηστών καθώς και να τα αποθηκεύσω με ασφάλεια.

Κεφάλαιο 2 – Εγχειρίδιο Παιχνιδιού

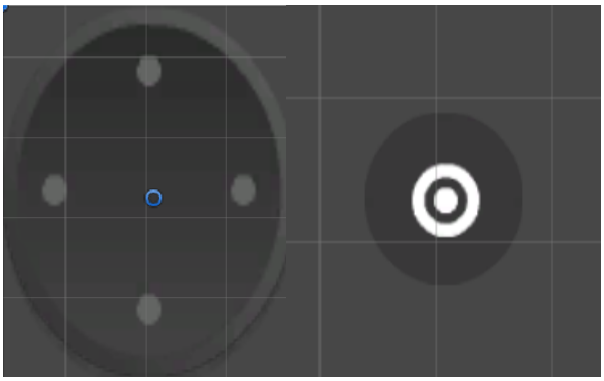
2.1 Joystick – Κίνηση Χαρακτήρα

Έχω φτιάξει ένα χειριστήριο για το κινητό ώστε ο παίκτης να μπορεί να ελέγξει τον χαρακτήρα του μέσα στο παιχνίδι.

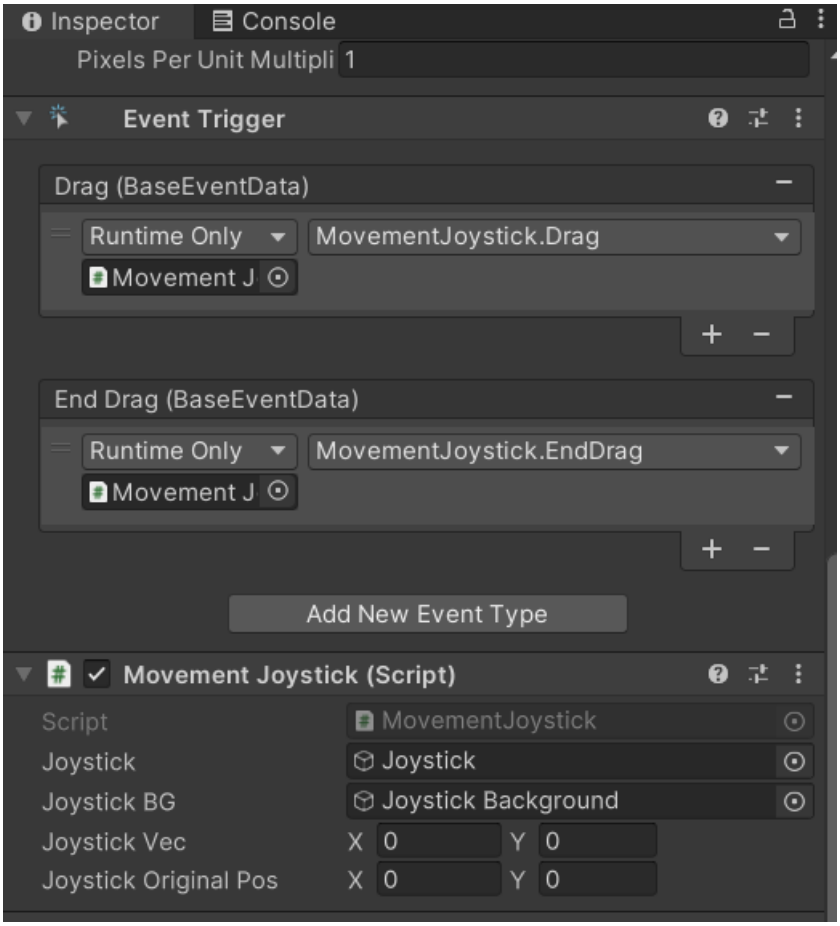
Για να φτιάξω αυτό το χειριστήριο έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity 3 UI Images.



Από τα 3 UI Images το πιο σημαντικό είναι το Movement Joystick διότι έχει όλες τις λειτουργίες. Τα άλλα δυο είναι απλά Background εικόνες και δεν έχουν κανένα ενσωματωμένο κώδικα.



Στο UI Image (Movement Joystick) υπάρχουν τα Event Trigger που χρησιμοποιούνται για να καθορίσουν λειτουργίες εκτέλεσης ανά κάθε συμβάν. Όπως βλέπετε πιο κάτω χρησιμοποιώ 2 Event Triggers όταν ο User (παίκτης) τραβάει το Joystick να καλεί την συνάρτηση Drag() και όταν τελειώνει το τράβηγμα να καλεί την συνάρτηση EndDrag().



Ακόμη όπως βλέπουμε πιο πάνω στην εικόνα υπάρχει ένας κώδικας ο οποίος είναι ο λόγος που ο παίκτης μπορεί να κινηθεί. Κατά την διάρκεια του παιχνιδιού όταν ο παίκτης ενεργοποιήσει το χειριστήριο τρέχει η Start() η οποία αποθηκεύει την αρχική θέση του Joystick Background και υπολογίζει την ακτίνα του χειριστηρίου. Στον υπολογισμό της ακτίνα του χειριστηρίου διαιρω με το 4 ώστε να περιορίσω την περιοχή του χειριστηρίου που μπορεί ο παίκτης να τραβήξει.

```
0 references
void Start()
{
    // Store the original position of the joystick background
    joystickOriginalPos = joystickBG.transform.position;

    // Calculate the joystick radius based on the size of the background's RectTransform
    joystickRadius = joystickBG.GetComponent<RectTransform>().sizeDelta.y / 4;
}
```

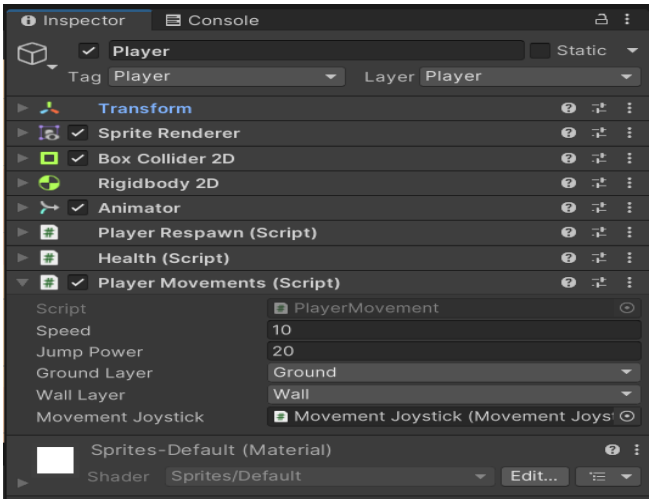
Πιο κάτω βλέπουμε την συνάρτηση Drag() η οποία εκτελείτε όταν τραβάτε το χειριστήριο. Αρχικά μετατρέπει τα δεδομένα της κίνησης του χειριστηρίου σε PointerEventData και αποθηκεύει την αρχική τοποθεσία του. Ύστερα υπολογίζει την κατεύθυνση του χειριστηρίου αφαιρώντας τις 2 τιμές. Έχω περιορίσει την κίνηση του παίχτη προς τα πάνω γιατί έχω φτιάξει άλλο κώδικα για το Jump() το οποίο θα μιλήσω αργότερα. Εν τέλη ελέγχω την κίνηση του χειριστήριού όταν το δάκτυλο του παίχτη είναι εντός και εκτός του ορίου ακτίνας ώστε να καλυτερέψω το UI.

```
0 references
public void Drag(BaseEventData baseEventData)
{
    // Cast the base event data to pointer event data to get drag position
    PointerEventData pointerEventData = baseEventData as PointerEventData;
    Vector2 dragPos = pointerEventData.position;

    // Calculate the joystick vector based on the drag position relative to the original joystick position
    joystickVec = (dragPos - joystickOriginalPos).normalized;

    // Restrict the joystick vector to horizontal and downward directions
    if (joystickVec.y > 0)
    {
        joystickVec.y = 0; // Prevent upward movement
    }

    // Calculate the distance between the drag position and the joystick's original position
    float joystickDist = Vector2.Distance(dragPos, joystickOriginalPos);

    // If the drag position is within the joystick's radius, move the joystick to the drag position
    if (joystickDist < joystickRadius)
    {
        joystick.transform.position = joystickOriginalPos + joystickVec * joystickDist;
    }
    // If the drag position exceeds the joystick's radius, move the joystick to the edge of the radius
    else
    {
        joystick.transform.position = joystickOriginalPos + joystickVec * joystickRadius;
    }

    isDragging = true; // Indicate that the joystick is being dragged
}
```

Πιο κάτω βλέπουμε την συνάρτηση EndDrag() η οποία εκτελείτε όταν σταματάει ο παίχτης να τραβάει το χειριστήριο. Αρχικά αποθηκεύω σε μια μεταβλητή bool ότι ο παίχτης έχει σταματήσει το τράβηγμα του χειριστηρίου και αρχίζω μια κορουτίνα με την συνάρτηση ReturnToOriginalPosition(). Χρησιμοποιώ την κορουτίνα για να εκτελέσω ενέργειες που διαρκούν για αρκετά frames έτσι ώστε να σταματήσω την παύσει στην εφαρμογή μου. Στην συνάρτηση ReturnToOriginalPosition() επαναφέρετε ομαλά το χειριστήριο στην αρχική του θέση σε διάρκεια 0.2f (2 δευτερολέπτων) κατά την αφαιρέσει του δακτύλου από το χειριστήριο.

```
0 references
public void EndDrag(BaseEventData baseEventData)
{
    // Set isDragging to false when dragging ends
    isDragging = false;
    // Start the coroutine to return the joystick to its original position
    StartCoroutine(ReturnToOriginalPosition());
}

1 reference
private IEnumerator ReturnToOriginalPosition()
{
    // Smoothly return the joystick to its original position
    Vector3 startPos = joystick.transform.position;
    Vector3 endPos = joystickOriginalPos;
    float elapsedTime = 0f;
    float duration = 0.2f; // Duration of the return effect in seconds

    while (elapsedTime < duration)
    {
        joystick.transform.position = Vector3.Lerp(startPos, endPos, elapsedTime / duration);
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    // Ensure the joystick is exactly at the original position at the end
    joystick.transform.position = endPos;
    joystickVec = Vector2.zero; // Reset joystick vector
}
```

Το χειριστήριο για να λειτουργήσει χρειάζεται να εφαρμοστεί στον χαρακτήρα και σε αυτήν την περίπτωση στο κώδικα PlayerMovement.



Κατά την διάρκεια του παιχνιδιού αποθηκεύετε η κίνηση του χειριστήριού που υπολογίσαμε στο Movement Joystick και ελέγχουμε την κατεύθυνση του παίχτη κατά πόσο είναι δεξιά ή αριστερά και μετακινούμε το παίχτη αντίστοιχα. Έπειτα ελέγχω εάν υπάρχει το Animator και στην συνέχεια ενεργοποιώ την παράμετρο Run όταν είναι μεγαλύτερη από το 0.01f (Λειτουργεία πάνω από 1msec).

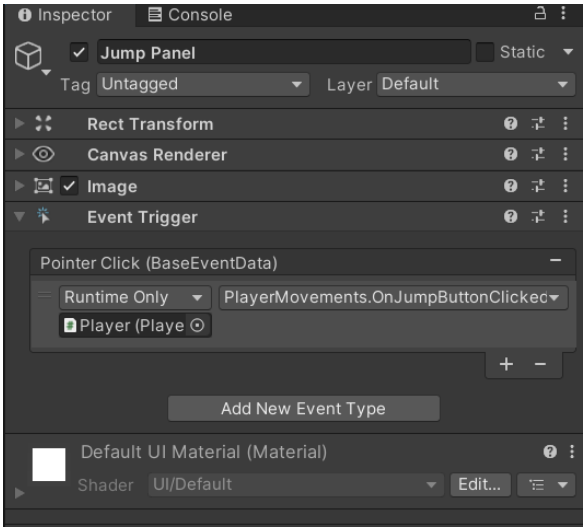
```
0 references
private void Update()
{
    // Get horizontal input from the joystick
    horizontalInput = movementJoystick.joystickVec.x;

    // Flip player when moving left and right
    if (horizontalInput > 0.01f)
    {
        transform.localScale = Vector3.one;
    }
    else if (horizontalInput < -0.01f)
    {
        transform.localScale = new Vector3(-1, 1, 1);
    }

    // Set animator parameters
    if (anim != null)  // Ensure Animator component exists before setting parameters
    {
        anim.SetBool("Run", Mathf.Abs(horizontalInput) > 0.01f);
        anim.SetBool("grounded", isGrounded());
    }
```

2.1 Jump Move – Κίνηση Χαρακτήρα

Έχω φτιάξει ένα κουμπί Jump για το κινητό ώστε ο παίκτης να μπορεί να ελέγξει τον χαρακτήρα του μέσα στο παιχνίδι.

Για να φτιάξω αυτό το κουμπί έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity 2 UI Images. Από τα 2 UI Images το πιο σημαντικό είναι το Jump Panel διότι έχει όλες τις λειτουργίες. Το άλλο είναι απλά Background εικόνας και δεν έχει κανένα ενσωματωμένο κώδικα.

Στο UI Image (Jump Panel) υπάρχει το Event Trigger Pointer Click το οποίο καλεί την συνάρτηση OnJumpButtonClicked() η οποία βρίσκετε στον κώδικα PlayerMovement() του Player.



Αρχικά όταν ενεργοποιηθεί το Event Trigger τότε καλείται η συνάρτηση Jump().

```
// Public method to be called by EventTrigger
0 references
public void OnJumpButtonClicked()
{
    Jump();
}
```

```
1 reference
public void Jump()
{
    if (isGrounded())
    {
        body.velocity = new Vector2(body.velocity.x, jumpPower);
        if (anim != null) anim.SetTrigger("Jump");  // Ensure Animator component exists before setting trigger
    }
    else if (onWall() && !isGrounded())
    {
        if (horizontalInput == 0)
        {
            body.velocity = new Vector2(-Mathf.Sign(transform.localScale.x) * 10, 0);
            transform.localScale = new Vector3(-Mathf.Sign(transform.localScale.x), transform.localScale.y, transform.localScale.z);
        }
        else
        {
            body.velocity = new Vector2(-Mathf.Sign(transform.localScale.x) * 3, 6);
        }
        wallJumpCooldown = 0;
    }
}
```

Αυτή η συνάρτηση ελέγχει εάν ο χαρακτήρας βρίσκεται στο έδαφος. Εάν ναι τότε εφαρμόζεται μια νέα ταχύτητα στο σώμα του χαρακτήρα ίση με την προκαθορισμένη μεταβλητή (20) για τον άξονα y , ενώ για τον άξονα x η ταχύτητα του σώματος παραμένει ίδια. Έπειτα ελέγχετε εάν υπάρχει Animator και ενεργοποιώ το animation Jump. Εάν όχι τότε ελέγχετε εάν ο χαρακτήρας βρίσκετε στον τοίχο και ταυτόχρονα δεν είναι στο έδαφος. Στην συνέχεια αλλάζω την πορεία του άλματος στην αντίθετη φορά

με τον τοίχο και επαναφέρω το WallJumpCooldown. Αυτό το κάνω ώστε ο παίχτης να μην κολλάει στον τοίχο.

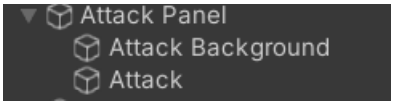
```
27      private void Update()
47          }
48
49          // Wall Jump logic
50          if (wallJumpCooldown > 0.02f)
51          {
52              body.velocity = new Vector2(horizontalInput * speed, body.velocity.y);
53
54              if (onWall() && !isGrounded())
55              {
56                  body.gravityScale = 0;
57                  body.velocity = Vector2.zero;
58              }
59              else
60              {
61                  body.gravityScale = 7;
62              }
63          }
64          else
65          {
66              wallJumpCooldown += Time.deltaTime;
67          }
68      }
```

Κατά την διάρκεια του παιχνιδιού ελέγχω εάν το WallJumpCooldown είναι μεγαλύτερο του 0.02f, εάν ναι τότε η κίνηση του σώματος του χαρακτήρα ενημερώνετε βάση της κίνησης του χειριστήριού και την προκαθορισμένη τιμή speed (10). Έπειτα ελέγχετε εάν ο χαρακτήρας βρίσκεται σε επαφή με τον τοίχο και είναι εχτός εδάφους, εάν ναι τότε η ταχύτητα και η βαρύτητα του χαρακτήρα μετατρέπετε σε μηδέν κάνοντάς τον να κολλάει στον τοίχο, εάν όχι τότε επαναφέρετε η βαρύτητα στην αρχική της μορφή. Αν η τιμή του WallJumpCooldown είναι μικρότερη του 0.02f τότε αυξάνετε σταδιακά με τα frames per second.

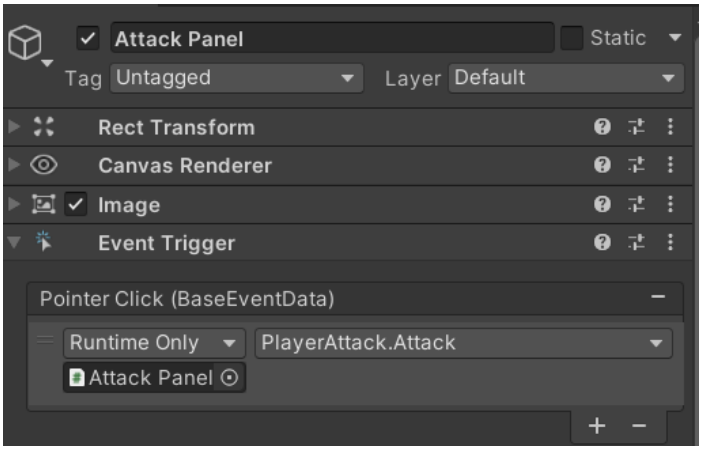
2.3 Attack Move – Σύστημα Επίθεσης

Έχω φτιάξει ένα κουμπί Attack (Επίθεσης) για το κινητό ώστε ο παίκτης να μπορεί να ελέγξει τον χαρακτήρα του μέσα στο παιχνίδι.

Για να φτιάξω αυτό το κουμπί έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity 3 UI Images. Από τα 3 UI Images το πιο σημαντικό είναι το Attack Panel διότι έχει όλες τις λειτουργίες. Τα άλλα είναι απλά Background εικόνας και δεν έχουν κανένα ενσωματωμένο κώδικα.



Στο UI Image (Attack Panel) υπάρχει το Event Trigger Pointer Click το οποίο καλεί την συνάρτηση Attack() η οποία βρίσκετε στον κώδικα PlayerAttack() του Attack Panel.



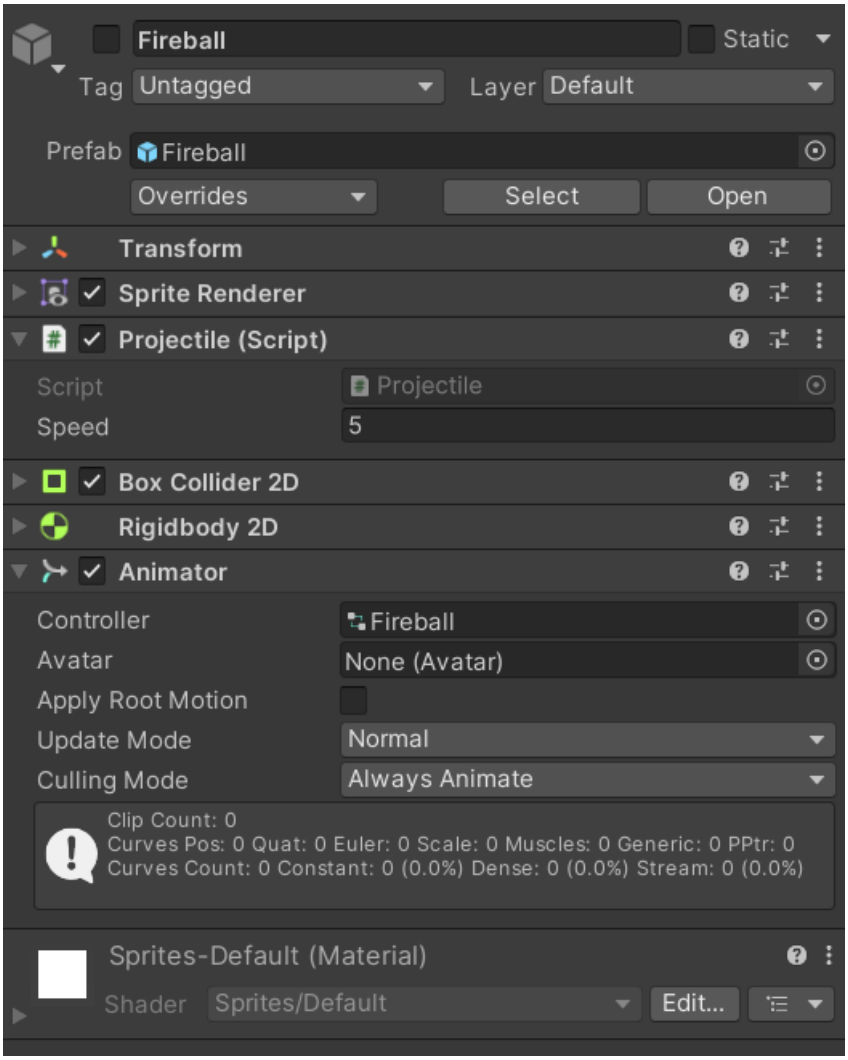

```
public class PlayerAttack : MonoBehaviour
    private void Update()
    {
        cooldownTimer += Time.deltaTime;
    }

    0 references
    public void Attack()
    {
        SoundManager.instance.PlaySound(fireballSound);
        anim.SetTrigger("attack");
        cooldownTimer = 0;

        fireballs[FindFireball()].transform.position = firePoint.position;
        fireballs[FindFireball()].GetComponent<Projectile>().SetDirection(Mathf.Sign(transform.localScale.x));
    }
```

Στον πιο πάνω κώδικα όταν το Event Trigger ενεργοποιηθεί τότε καλείτε η συνάρτηση Attack() η οποία ενεργοποιεί τον προκαθορισμένο ήχο και το Animation του. Το Cooldown Timer είναι η λειτουργά μέτρησης του χρόνου κατά την οποία το στοιχείο

Fireball βρίσκετε στον αέρα (αυτό αυξάνετε σταδιακά κατά την διάρκεια της εφαρμογής). Έπειτα βρίσκει την τοποθεσία του στοιχείου Fireball και την κατευθύνει προς την αντίστοιχη πορεία.

```
public class PlayerAttack : MonoBehaviour
    private int FindFireball()
    {
        for (int i = 0; i < fireballs.Length; i++)
        {
            if (!fireballs[i].activeInHierarchy)
                return i;
        }
        return 0;
    }
}
```

Έπειτα καλείτε η συνάρτηση FindFireball() η οποία έχει την λειτουργία να ψάχνει στην ιεραρχία του Unity και να βρίσκει την Fireball η οποία δεν είναι ακόμη ενεργεί.



Στην Ιεραρχία έχω φτιάξει ένα προκατασκευασμένο αντικείμενο με το όνομα Fireball το οποίο έχει τον κώδικα Projectile.cs .

```
public class Projectile : MonoBehaviour
    1 reference
    public void SetDirection(float _direction)
    {
        lifetime = 0;
        direction = _direction;
        gameObject.SetActive(true);
        hit = false;
        boxCollider.enabled = true;

        float localScaleX = transform.localScale.x;
        if (Mathf.Sign(localScaleX) != _direction)
            localScaleX = -localScaleX;

        transform.localScale = new Vector3(localScaleX, transform.localScale.y, transform.localScale.z);
    }
```

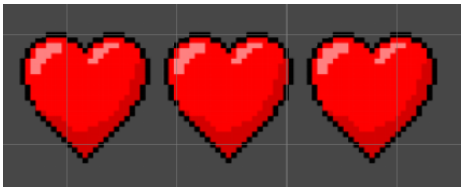
Αρχικά κατά την εκτέλεση του Attack καλείται η συνάρτηση SetDirection() η οποία ενεργοποιεί το αντικείμενο και το κατευθύνει στην σωστή φορά.

```
public class Projectile : MonoBehaviour
    private void Update()
    {
        if (hit) return;
        float movementSpeed = speed * Time.deltaTime * direction;
        transform.Translate(movementSpeed, 0, 0);

        lifetime += Time.deltaTime;
        if (lifetime > 5) gameObject.SetActive(false);
    }
```

Κατά την διάρκεια του παιχνιδιού ελέγχετε εάν το αντικείμενο έχει συγκρουστεί, εάν όχι τότε το αντικείμενο κινητέ κανονικά με την προκαθορισμένη ταχύτητα (5) και μετράτε σταδιακά ο χρόνος ανά frame. Όταν ο χρόνος κίνησης του αντικειμένου φτάσει να είναι μεγαλύτερος του 5 τότε απενεργοποιώ το αντικείμενο Fireball.

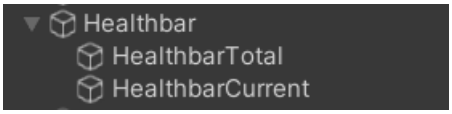
2.4 Health Bar – Μπάρα Ζωής

Έχω φτιάξει ένα Health Bar – Μπάρα Ζωής για το κινητό ώστε ο παίκτης να μπορεί να δει πόσες ζωές του έχουν απομείνει για να νικήσει το παιχνίδι.

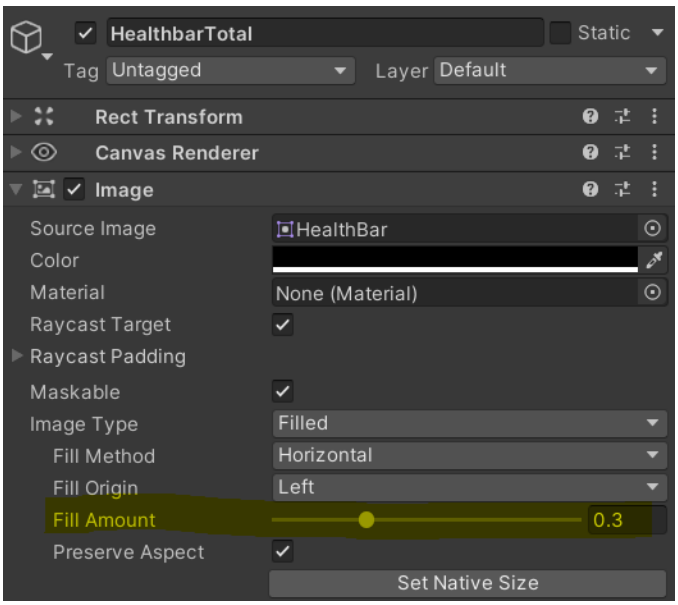


Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity 3 UI Images. Από τα 3 UI Images το πιο σημαντικό είναι το HealthBar διότι έχει όλες τις

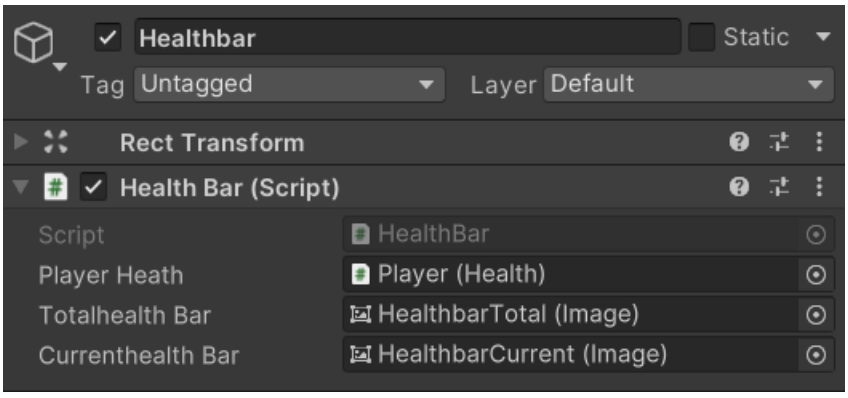
λειτουργίες. Τα άλλα είναι απλά Background εικόνας και δεν έχουν κανένα ενσωματωμένο κώδικα.



Είναι ανάγκη να προσθέσω ότι το HealthbarTotal & HealthbarCurrent έχουν στο Fill Amount 0.3, αυτό σημαίνει ότι ο χαρακτήρας έχει στην διάθεσή του 3 ζωές.



(Διευκρίνηση το Fill Amount μπορεί να δεχτεί τιμές από το 0 – 1)



Στο αντικείμενο Healthbar υπάρχει ο κώδικας HealthBar.cs ο οποίος είναι συνδεδεμένος με τον κώδικα Health.cs που βρίσκεται στο αντικείμενο Player και έχει ως καθορισμένα στοιχεία το 0.3 από τα 2 UI Images.

```
public class HealthBar : MonoBehaviour
    private void Start(){
        totalhealthBar.fillAmount = playerHeath.currentHealth / 10;
    }

    0 references
    private void Update(){

        currenthealthBar.fillAmount = playerHeath.currentHealth / 10;
    }
}
```

Στον παραπάνω κώδικα η μπάρα ζωής ενημερώνετε από την αρχή που τρέχει η εφαρμογή και κατά διαστήματα της.

Διευκρίνηση διαιρείτε με το 10 ώστε να μπορεί να εισαχθεί σωστά η τιμή της ζωής στο Fill Amount του UI Image.

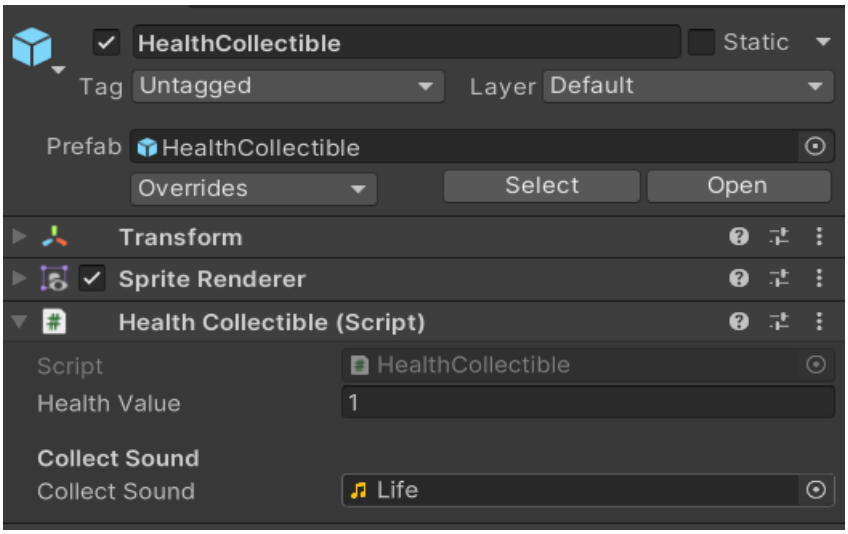
Ο κώδικας Health βρίσκετε στο αντικείμενο Player και λειτουργά αυτόματα κατά την έναρξη του παιχνιδιού και αποθηκεύει σε μια μεταβλητή το αρχικό ποσό των ζωών του παίχτη (3).

```
public class Health : MonoBehaviour

    0 references
    private void Awake()
    {
        currentHealth = startingHealth;
        anim = GetComponent<Animator>();
        spriteRend = GetComponent<SpriteRenderer>();
    }
```

2.5 Health Collectable – Είσπραξη Ζωής

Έχω φτιάξει ένα Health Collectable – Είσπραξη Ζωής για το κινητό ώστε ο παίκτης να μπορεί να πάρει έξτρα ζωές κατά την διάρκεια του παιχνιδιού ώστε να τελειώσει το παιχνίδι.

Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα Sprite με το όνομα HealthCollectible.



Στο αντικείμενο αυτό έχω φτιάξει ένα κώδικα που ελέγχει το Box Collider του και κάνει Trigger όταν ο παίχτης έρθει σε επαφή με αυτό. Στην συνέχεια καλή την συνάρτηση AddHealth() η οποία βρίσκετε στο Health.cs του αντικειμένου Player και προσθετή την ζωή και απενεργοποιώ το αντικείμενο (Health Collectable).

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player")
    {
        SoundManager.instance.PlaySound(collectSound);
        collision.GetComponent<Health>().AddHealth(healthValue);
        gameObject.SetActive(false);
    }
}
```

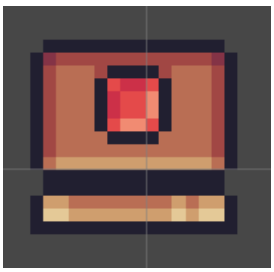
```
public class Health : MonoBehaviour
    2 references
    public void AddHealth(float _value)
    {
        currentHealth = Mathf.Clamp(currentHealth + _value, 0, startingHealth);
    }
```

Όταν καλείτε η συνάρτηση AddHealth() από την HealthCollectible ελέγχει εάν η ζωή που πήρε ο παίχτης είναι δεκτή για να προσθετή στην μπάρα ζωής του. Δηλαδή εάν ο παίχτης έχει 3 ζωές τότε η καρδία δεν προσθετέ στην μπάρα ζωής , ενώ εάν έχει 2 ή 1 ζωή τότε προσθετέ κανονικά στην μπάρα ζωής του.

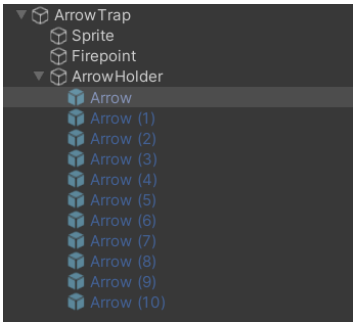
2.6 Traps

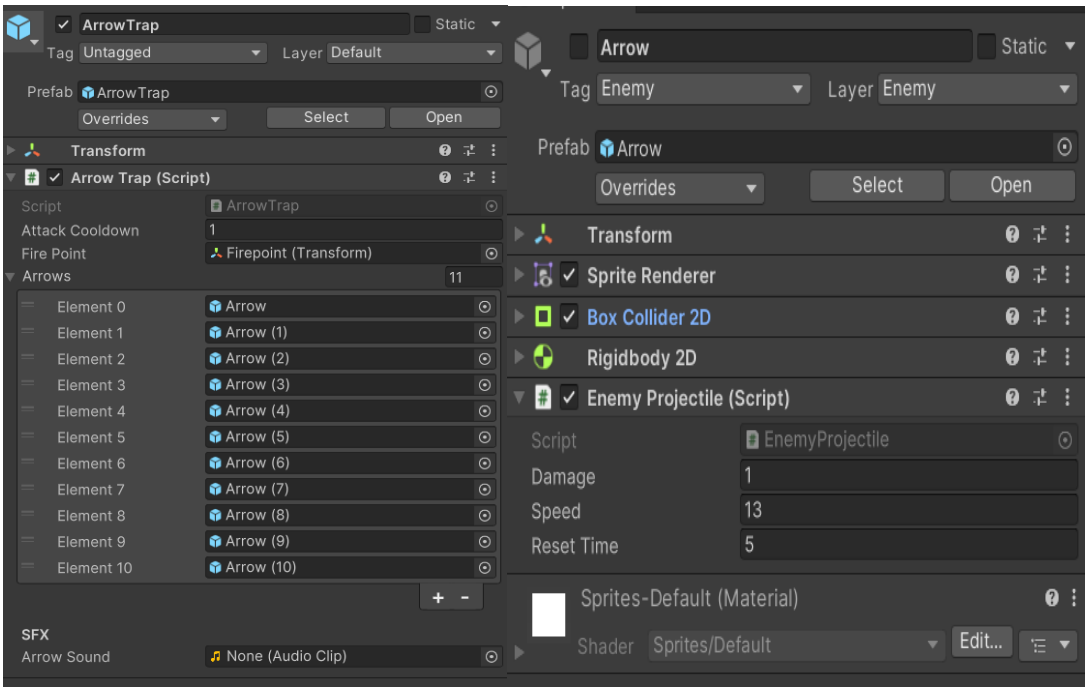
2.6.1 Arrow – Βέλη

Έχω φτιάξει ένα Trap Arrow ώστε να κάνω το παιχνίδι πιο ενδιαφέρον καθώς και να αυξήσω την δυσκολία του παιχνιδιού.



Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα ArrowTrap, που έχει παιδία το αντικείμενο Arrow.





Στο κάθε ένα αντικείμενο υπάρχει ξεχωριστός κώδικας που κάνει διαφορετικά πράγματα όπως θα μιλήσουμε αργότερα.

```
public class ArrowTrap : MonoBehaviour
    private void Update()
    {
        cooldownTimer += Time.deltaTime;

        if (cooldownTimer >= attackCooldown)
            Attack();
    }
}
```

Αρχικά κατά την διάρκεια του παιχνιδιού ο κώδικας ArrowTrap ελέγχει εάν το Cooldown Timer είναι μεγαλύτερο της προκαθορισμένης μεταβλητής (1) και καλή την συνάρτηση Attack().

```
public class ArrowTrap : MonoBehaviour
    1 reference
    private void Attack()
    {
        cooldownTimer = 0;

        SoundManager.instance.PlaySound(arrowSound);
        arrows[FindArrow()].transform.position = firePoint.position;
        arrows[FindArrow()].GetComponent<EnemyProjectile>().ActivateProjectile();

    }
    2 references
    private int FindArrow()
    {
        for (int i = 0; i < arrows.Length; i++)
        {
            if (!arrows[i].activeInHierarchy)
                return i;
        }
        return 0;
    }
```

Έπειτα η συνάρτηση Attack() ενεργοποιεί τον προκαθορισμένο ήχο και αυξάνει σταδιακά το Cooldown Timer κατά την διάρκεια της εφαρμογής. Έπειτα βρίσκει την τοποθεσία του στοιχείου Arrow και την κατευθύνει προς την αντίστοιχη πορεία.

Στην συνέχεια καλείτε η συνάρτηση FindArrow() η οποία έχει την λειτουργία να ψάχνει στην ιεραρχία του Unity και βρίσκει το Arrow το οποίο δεν είναι ακόμη ενεργό.

Στην Ιεραρχία έχω φτιάξει ένα προκατασκευασμένο αντικείμενο με το όνομα Arrow το οποίο έχει τον κώδικα EnemyProjectile.cs .

```
2 references
public void ActivateProjectile()
{
    lifetime = 0;
    gameObject.SetActive(true);
}
0 references
private void Update()
{
    float movementSpeed = speed * Time.deltaTime;
    transform.Translate(movementSpeed, 0, 0);

    lifetime += Time.deltaTime;
    if (lifetime > resetTime)
        gameObject.SetActive(false);
}

0 references
private new void OnTriggerEnter2D(Collider2D collision)
{
    base.OnTriggerEnter2D(collision); //Execute logic from parent script first
    gameObject.SetActive(false); //When this hits any object deactivate arrow
}
```

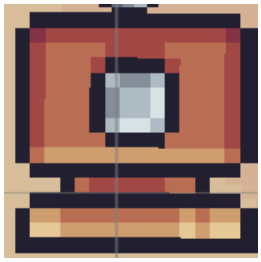
Κατά την διάρκεια του παιχνιδιού το αντικείμενο κινητέ κανονικά με την προκαθορισμένη ταχύτητα (13) και μετράτε σταδιακά ο χρόνος ανά frame. Όταν ο χρόνος κίνησης του αντικειμένου φτάσει να είναι μεγαλύτερος του 13 τότε απενεργοποιώ το αντικείμενο Arrow.

Επίσης όταν το Collision του αντικειμένου έρχεται σε επαφή με τον παίχτη τότε καλείται η συνάρτηση OnTriggerEnter2D() ο οποίος βρίσκεται στον κώδικα EnemyDamage.cs και αφαιρεί την προκαθορισμένη ζημία (1).

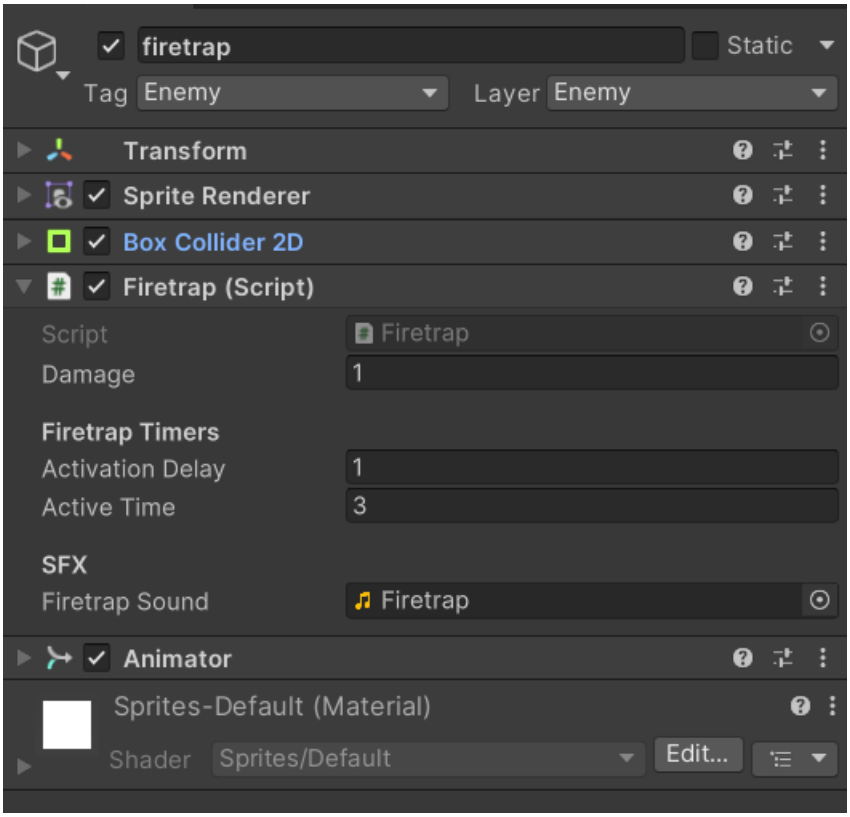
```
2 references
protected void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player")
        collision.GetComponent<Health>().TakeDamage(damage);
}
```

2.6.2 Fire – Φωτιά

Έχω φτιάξει ένα Fire Trap ώστε να κάνω το παιχνίδι πιο ενδιαφέρον καθώς και να αυξήσω την δυσκολία του παιχνιδιού.



Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα firetrap.



Μέσα σε αυτό το αντικείμενο υλοποίησα τον κώδικα Firetrap.cs ο οποίος έχει προκαθορισμένες τιμές για το Damage, για το Activation Delay (Reload) και για το Active Time.

```
public class Firetrap : MonoBehaviour

    0 references
    private void Update()
    {
        if (playerHealth != null && active)
            playerHealth.TakeDamage(damage);
    }
```

Αρχικά κατά την διάρκεια του παιχνιδιού ελέγχει εάν η μπάρα ζωής του παίχτη είναι ενεργεί και ταυτόχρονα δεν είναι ίση με μηδέν τότε ο παίχτης χάνει ένα πόντο ζωής.

```
public class Firetrap : MonoBehaviour
    0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            playerHealth = collision.GetComponent<Health>();

            if (!triggered)
                StartCoroutine(ActivateFiretrap());

            if (active)
                collision.GetComponent<Health>().TakeDamage(damage);
        }
    }
```

Ακόμη στο αντικείμενο firetrap έχω υλοποίηση ένα κώδικα που ενεργοποιείτε όταν ο παίχτης έρθει σε επαφή με το collision του firetrap. Στην περίπτωση αυτή το Health δίνει το current Health του παίχτη και αρχίζει σε υπορουτίνα την συνάρτηση ActivateFiretrap() την οποία θα μιλήσω αργότερα. Έπειτα ελέγχει εάν η παγίδα είναι ενεργοποιημένη και εάν ναι τότε καλεί την συνάρτηση TakeDamage() που βρίσκετε στο κώδικα Health.cs του Player.

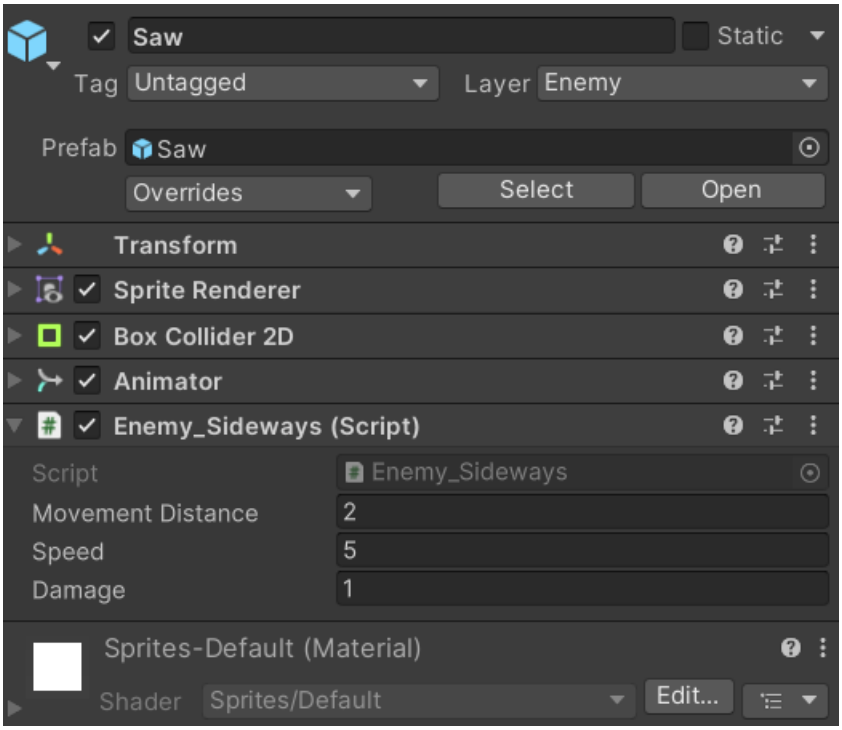
```
public class Firetrap : MonoBehaviour
    private IEnumerator ActivateFiretrap()
    {
        //turn the sprite red to notify the player and trigger the trap
        triggered = true;
        spriteRend.color = Color.red;

        //Wait for delay, activate trap, turn on animation, return color back to normal
        yield return new WaitForSeconds(activationDelay);
        SoundManager.instance.PlaySound(firetrapSound);
        spriteRend.color = Color.white; //turn the sprite back to its initial color
        active = true;
        anim.SetBool("activated", true);

        //Wait until X seconds, deactivate trap and reset all variables and animator
        yield return new WaitForSeconds(activeTime);
        active = false;
        triggered = false;
        anim.SetBool("activated", false);
    }
}
```

Όταν καλείτε η υπορουτίνα ActivateFiretrap() τότε μετατρέπει το sprite του firetrap σε κόκκινο ώστε να δείξει πως υπάρχει κίνδυνος. Στην συνέχεια περιμένει ένα χρονικό διάστημα και ενεργοποιεί τον ήχο και την παγίδα. Έπειτα μετατρέπει το sprite στην κανονική του μορφή και μετά από κάποιο χρονικό διάστημα επαναφέρει τις μεταβλητές στην αρχική τους μορφή. Τέλος απενεργοποιώ το animation και το trap.

2.6.3 Saw

Έχω φτιάξει ένα Saw Trap ώστε να κάνω το παιχνίδι πιο ενδιαφέρον καθώς και να αυξήσω την δυσκολία του παιχνιδιού.

Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα Saw.



Μέσα σε αυτό το αντικείμενο υλοποίησα τον κώδικα Enemy_Sideways.cs ο οποίος έχει προκαθορισμένες τιμές για το Movement Distance, για το Speed και για το Damage.

```
0 references
private void Awake()
{
    leftEdge = transform.position.x - movementDistance;
    rightEdge = transform.position.x + movementDistance;
}

0 references
private void Update()
{

    if (movingLeft)
    {
        if (transform.position.x > leftEdge)
        {
            transform.position = new Vector3(transform.position.x - speed * Time.deltaTime, transform.position.y, transform.position.z);
        }
        else
            movingLeft = false;
    }
    else
    {
        if (transform.position.x < rightEdge)
        {
            transform.position = new Vector3(transform.position.x + speed * Time.deltaTime, transform.position.y, transform.position.z);
        }
        else
            movingLeft = true;
    }
}
```

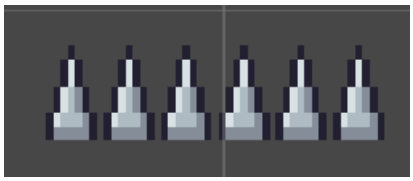
Αρχικά ορίζω τα όρια της παγίδας προς στα αριστερά και στα δεξιά βάση της προκαθορισμένης μεταβλητής που έχω δώσει. Στην συνέχεια κατά την διάρκεια της εφαρμογής ελέγχει την πορεία της παγίδας και αυξάνει την κίνηση με την προκαθορισμένη μεταβλητή της ταχύτητας.

```
public class Enemy_Sideways : MonoBehaviour
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            collision.GetComponent<Health>().TakeDamage(damage);
        }
    }
}
```

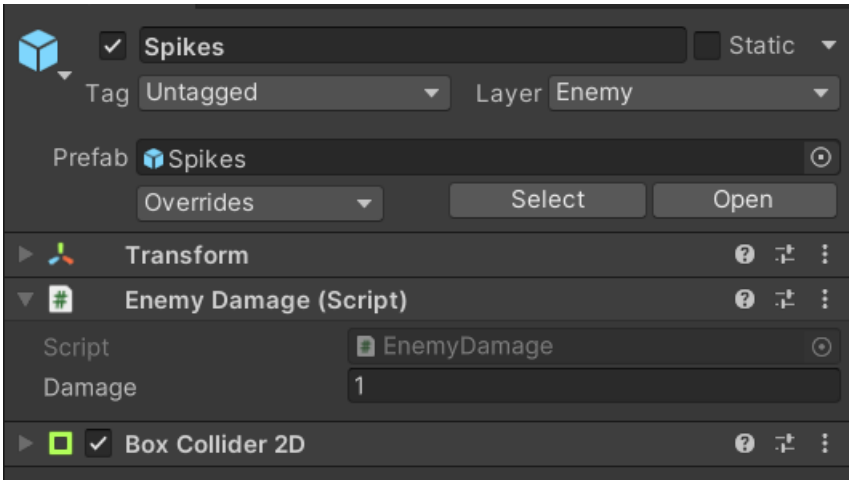
Όταν η παγίδα εισέλθει στο Collision του Player τότε καλείτε η συνάρτηση TakeDamage() του κώδικα Health.cs που βρίσκεται στον Player και χάνει έναν πόντο ζωής.

2.6.4 Spikes

Έχω φτιάξει ένα Spikes Trap ώστε να κάνω το παιχνίδι πιο ενδιαφέρον καθώς και να αυξήσω την δυσκολία του παιχνιδιού.



Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα Spikes.



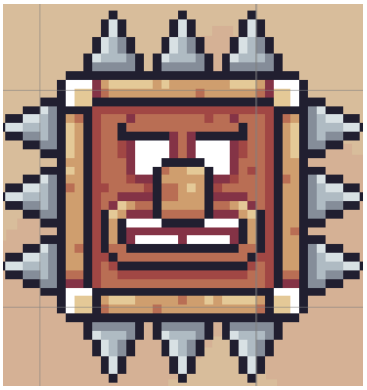
Μέσα σε αυτό το αντικείμενο υλοποίησα τον κώδικα EnemyDamage.cs ο οποίος έχει προκαθορισμένες τιμές για το Damage.

```
using UnityEngine;

2 references
public class EnemyDamage : MonoBehaviour
{
    1 reference
    [SerializeField] protected float damage;

    2 references
    protected void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
            collision.GetComponent<Health>().TakeDamage(damage);
    }
}
```

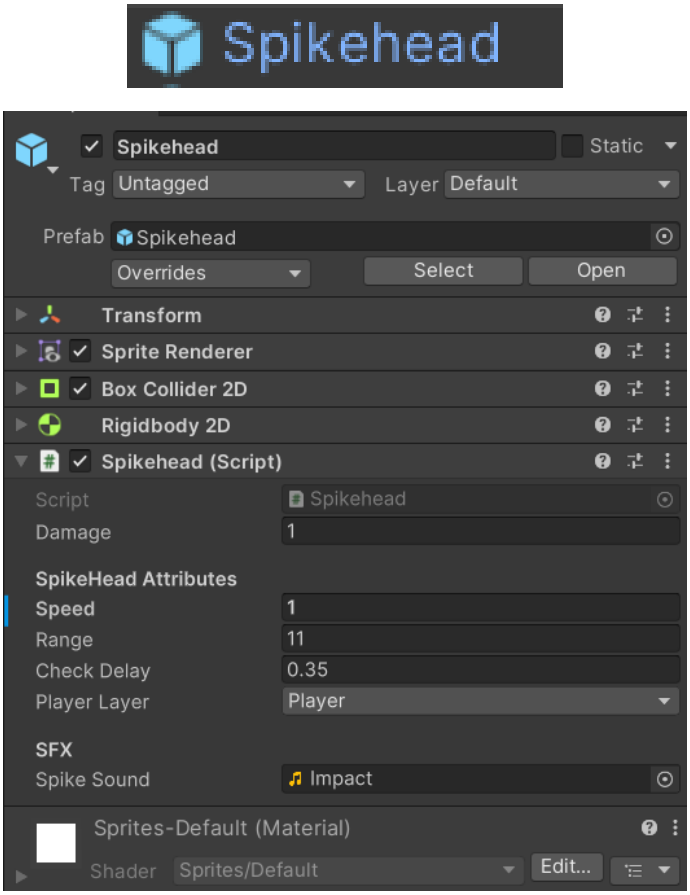
Όταν ο Player εισέλθει στο Collision της παγίδας τότε καλείτε η συνάρτηση TakeDamage() του κώδικα Health.cs που βρίσκεται στον Player και χάνει έναν πόντο ζωής.

2.6.5 Spike Head

Έχω φτιάξει ένα Spike Head Trap ώστε να κάνω το παιχνίδι πιο ενδιαφέρον καθώς και να αυξήσω την δυσκολία του παιχνιδιού.



Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα Spikehead.



Μέσα σε αυτό το αντικείμενο υλοποίησα τον κώδικα Spikehead.cs ο οποίος έχει προκαθορισμένες τιμές για το Damage, το Speed, το Range, το Check Delay και για το Player Layer.

```
public class Spikehead : EnemyDamage
    0 references
    private void OnEnable()
    {
        Stop();
    }
```

```
public class Spikehead : EnemyDamage
    private void Stop()
    {
        destination = transform.position; //Set destination as current position so it doesn't move
        attacking = false;
    }
```

Όταν αρχίσει το παιχνίδι τότε καλείτε η συνάρτηση Stop() η οποία σταματά την παγίδα από το να επιτίθεται και ορίζει το destination στο συγκεκριμένο σημείο.

```
public class Spikehead : EnemyDamage
    private void Update()
    {
        //Move spikehead to destination only if attacking
        if (attacking)
            transform.Translate(destination * Time.deltaTime * speed);
        else
        {
            checkTimer += Time.deltaTime;
            if (checkTimer > checkDelay)
                CheckForPlayer();
        }
    }
```

Κατά την διάρκεια του παιχνιδιού ελέγχετε εάν η παγίδα επιτίθεται, εάν ναι τότε η παγίδα επιτίθεται προς την κατεύθυνση του. Αλλιώς ελέγχει εάν ο χρόνος είναι μεγαλύτερος από το 0.35 τότε καλείτε η συνάρτηση CheckForPlayer().

```
public class Spikehead : EnemyDamage
    private void CheckForPlayer()
    {
        CalculateDirections();

        //Check if spikehead sees player in all 4 directions
        for (int i = 0; i < directions.Length; i++)
        {
            Debug.DrawRay(transform.position, directions[i], Color.red);
            RaycastHit2D hit = Physics2D.Raycast(transform.position, directions[i], range, playerLayer);

            if (hit.collider != null && !attacking)
            {
                attacking = true;
                destination = directions[i];
                checkTimer = 0;
            }
        }
    }
    1 reference
    private void CalculateDirections()
    {
        directions[0] = transform.right * range; //Right direction
        directions[1] = -transform.right * range; //Left direction
        directions[2] = transform.up * range; //Up direction
        directions[3] = -transform.up * range; //Down direction
    }
```

Κατά την εκτέλεση της συνάρτησης CheckForPlayer() υπολογίζονται οι κατευθύνσεις πάνω, κάτω, δεξιά και αριστερά. Έπειτα ελέγχετε εάν η παγίδα βλέπει τον παίχτη και του επιτίθεται.

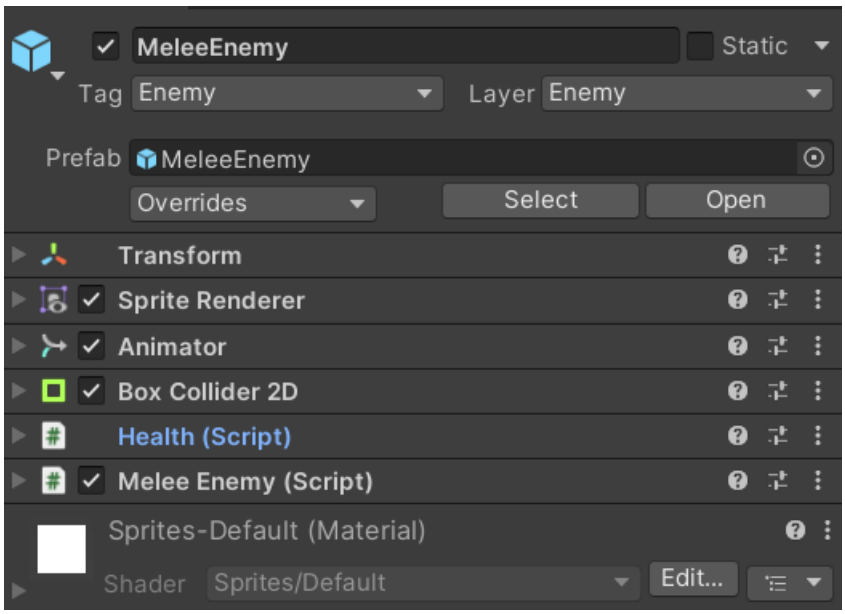
```
0 references
private new void OnTriggerEnter2D(Collider2D collision)
{
    SoundManager.instance.PlaySound(spikeSound);
    base.OnTriggerEnter2D(collision);
    Stop(); //Stop spikehead once he hits something
}
```

Στην περίπτωση που η παγίδα συγκρουστεί με τον παίχτη τότε αρχίζει ο ήχος και καλείτε η συνάρτηση OnTriggerEnter2D() του EnemyDamage η οποία με την σειρά της καλεί την συνάρτηση TakeDamage() του κώδικα Health.cs που βρίσκεται στον Player και χάνει έναν πόντο ζωής. Έπειτα καλείτε η συνάρτηση Stop() ώστε να σταματήσει το destination και η κίνηση της παγίδας.

2.7 Melee Enemy

Έχω φτιάξει ένα Melee Enemy ώστε να κάνω το παιχνίδι πιο ενδιαφέρον καθώς και να αυξήσω την δυσκολία του παιχνιδιού.

Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα MeleeEnemy.



Μέσα σε αυτό το αντικείμενο υλοποίησα τους κώδικες Health.cs και MeleeEnemy.cs.

```
public class MeleeEnemy : MonoBehaviour
    0 references
    private void Awake()
    {
        anim = GetComponent<Animator>();
        enemyPatrol = GetComponentInParent<EnemyPatrol>();
    }
```

Αρχικά ενεργοποιείτε η κίνηση του εχθρού προς στα αριστερά και έπειτα προς στα δεξιά βάση της ενεργοποίησης του enemyPatrol.cs αυτή η συνιστώσα αποθηκεύετε στην μεταβλητή enemyPatrol.

```
public class MeleeEnemy : MonoBehaviour
    private void Update()
    {
        cooldownTimer += Time.deltaTime;

        //Attack only when player in sight?
        if (PlayerInSight())
        {
            if (cooldownTimer >= attackCooldown && playerHealth.currentHealth > 0)
            {
                cooldownTimer = 0;
                anim.SetTrigger("meleeAttack");

                SoundManager.instance.PlaySound(attackSound);
            }
        }

        if (enemyPatrol != null)
            enemyPatrol.enabled = !PlayerInSight();
    }
```

Κατά την διάρκεια του παιχνιδιού αυξάνεται ο χρόνος και ελέγχετε εάν ο παίχτης είναι στην περιοχή επίθεσης του εχθρού. Εάν αυτό ισχύει τότε ελέγχετε εάν ο χρόνος είναι μεγαλύτερος ή ίσος με τον προκαθορισμένο χρόνο (2) και ο τρέχον πόντος ζωής είναι μεγαλύτερος του μηδέν τότε ενεργοποιείτε το Animation και ακούγετε ο ήχος. Στην συνέχεια ελέγχετε ο μηχανισμός περιπολίας και διακόπτεται εάν ο παίχτης είναι στο οπτικό του πεδίο.

2.8 Health (Take Damage)

```
public class Health : MonoBehaviour
    6 references
    public void TakeDamage(float _damage)
    {
        if (invulnerable) return;
        currentHealth = Mathf.Clamp(currentHealth - _damage, 0, startingHealth);

        if (currentHealth > 0)
        {
            anim.SetTrigger("hurt");
            StartCoroutine(Invunerability());

            SoundManager.instance.PlaySound(hurtSound);
        }
        else if (currentHealth < 0){
            anim.SetTrigger("die");

            //Deactivate all attached component classes
            foreach (Behaviour component in components)
                component.enabled = false;

            dead = true;

            SoundManager.instance.PlaySound(deathSound);
        }
```

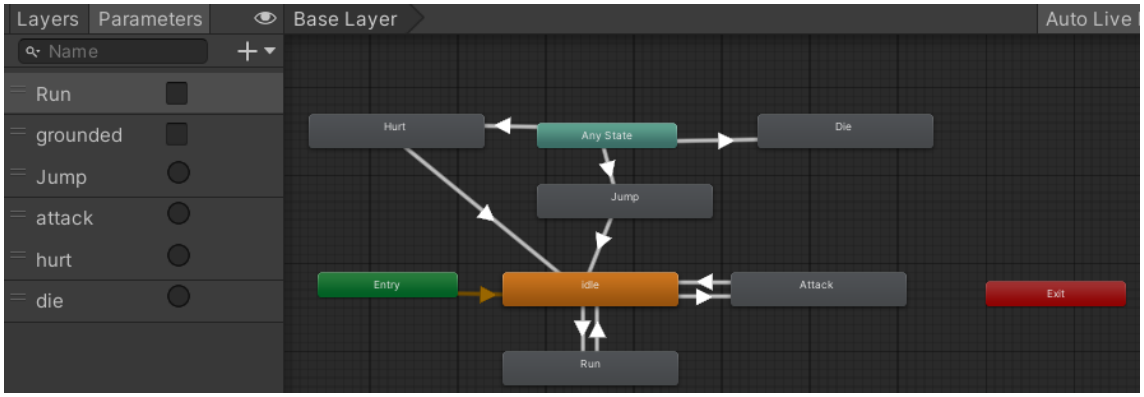
```
    else
    {
        print("The player is " + dead);
        if (!dead)
        {

            //Deactivate all attached component classes
            foreach (Behaviour component in components)
                component.enabled = false;

            anim.SetBool("grounded",true);
            anim.SetTrigger("die");

            dead = true;

            SoundManager.instance.PlaySound(deathSound);
        }
    }
}
```

Όταν καλείτε η συνάρτηση TakeDamage() τότε ελέγχετε εάν ο χαρακτήρας είναι αόρατος (Δεν μπορεί να δεχθεί επίθεση), εάν όχι τότε αφαιρείτε ένας πόντος ζωής από την μπάρα ζωής. Στην συνέχεια ελέγχετε εάν το Current Health είναι μεγαλύτερο του 0 (ελέγχει εάν ο χαρακτήρας είναι ακόμη ζωντανός) και αρχίζει το animation, τον ήχο και την υπορουτίνα Invunerability() η οποία κάνει τον χαρακτήρα να είναι αόρατος από κάθε ζημία για λίγο χρονικό διάστημα. Στην συνέχεια ελέγχει εάν το Current Health του παίχτη – χαρακτήρα είναι μικρότερο του μηδέν (αυτό γίνετε όταν ο παίχτης δεχτή πολλαπλές επιθέσεις από εχθρούς) και αρχίζει το animation, απενεργοποιεί όλες τις λειτουργίες (Joystick movement, Attack και Jump) και ορίζει μια μεταβλητή bool ως true. Με αυτήν την μεταβλητή ελέγχει εάν ο παίχτης – χαρακτήρας είναι ήδη νεκρός και απενεργοποιεί όλες τις λειτουργίες όπως ανάφερα πιο πάνω. Αυτό γίνεται για να αποτραπεί η loop του death animation, που ενεργοποιείται όταν ο παίχτης – χαρακτήρας δεχτεί ξανά επίθεση στην θέση αδράνειας (dead).

```
2 references
private IEnumerator Invunerability()
{
    invulnerable = true;
    Physics2D.IgnoreLayerCollision(10, 11, true);
    for (int i = 0; i < numberOfFlashes; i++)
    {
        spriteRend.color = new Color(1, 0, 0, 0.5f);
        yield return new WaitForSeconds(iFramesDuration / (numberOfFlashes * 2));
        spriteRend.color = Color.white;
        yield return new WaitForSeconds(iFramesDuration / (numberOfFlashes * 2));
    }
    Physics2D.IgnoreLayerCollision(10, 11, false);
    invulnerable = false;
}
```

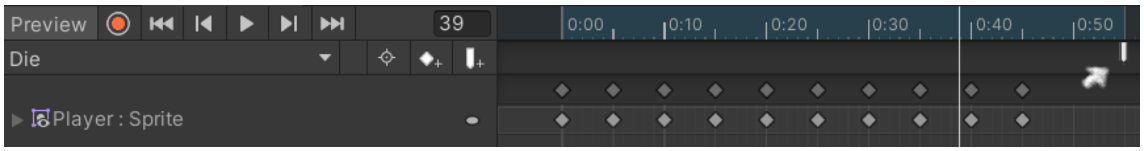
Κεφάλαιο 3 – Animations

3.1 Player



Ο χαρακτήρας Player έχει πολλά Animations όμως το πιο σημαντικό από όλα αυτά είναι το Die Animation.

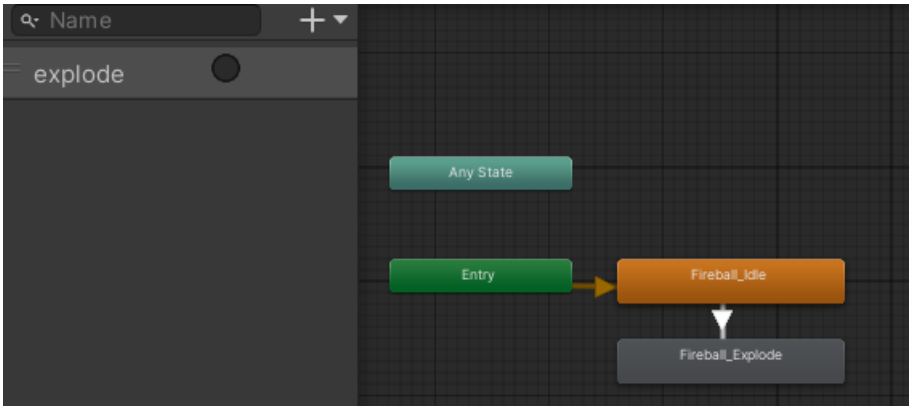
3.1.1 Die



Το Die Animation καλείται όταν ο χαρακτήρας έχει πεθάνει από οποία δίποδε κατάσταση. Όπως βλέπουμε από το άσπρο βελάκι υπάρχει ένα Event, το οποίο καλεί την συνάρτηση RespawnCheck() του χαρακτήρα Player.

Αυτή η συνάρτηση ελέγχει εάν ο παίχτης έχει ξεκλειδώσει checkpoints, εάν ναι τότε τον μεταφέρει στο πιο πρόσφατο με πλήρη ζωή. Εάν όχι τότε καλεί την συνάρτηση GameOver() την οποία θα μιλήσω αργότερα.

3.2 Fireball



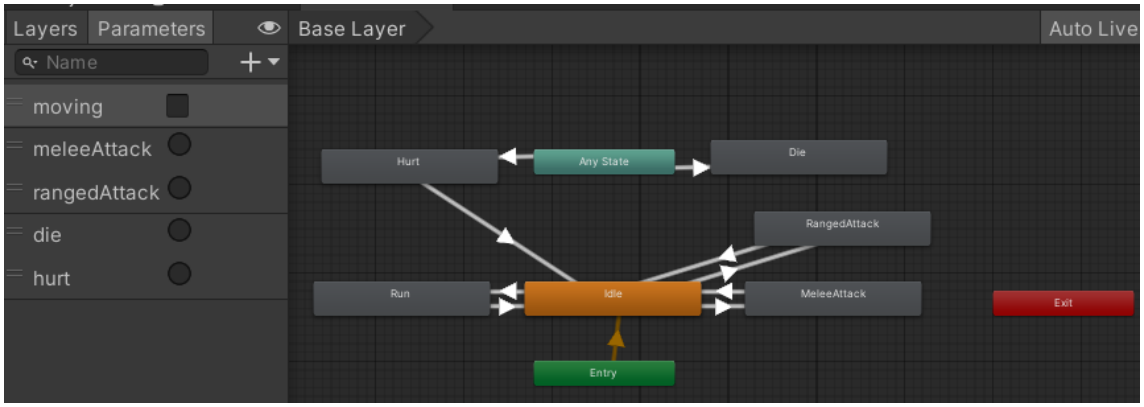
Στο Fireball υπάρχουν δυο καταστάσεις το Idle και το Explode. Το πιο σημαντικό από αυτά είναι το Explode.

3.2.1 Explode

Το Explode animation καλείτε όταν ο παίχτης – χαρακτήρας ενεργοποιεί την κίνηση Attack. Όπως βλέπουμε από το άσπρο βελάκι υπάρχει ένα Event, το οποίο καλεί την συνάρτηση Deactivated() του κώδικά Projectiles.cs .

Αυτή η συνάρτηση απενεργοποιεί το αντικείμενο του Fireball ώστε να μπορεί ο χαρακτήρας – παίχτης να πυροβολήσει πολλές Fireballs.

3.3 Melee Enemy

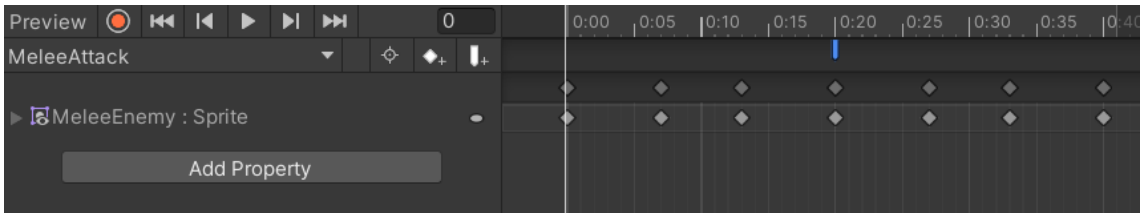


Ο χαρακτήρας Melee Enemy έχει πολλά Animations όμως τα πιο σημαντικό από όλα αυτά είναι το Die Animation και το MeleeAttack Animation.

3.3.1 Die

Το Die Animation καλείτε όταν ο Melee Enemy έχει πεθάνει. Αυτό με την σειρά του ενεργοποιεί το Event που καλεί την συνάρτηση Deactivated() του Health.cs το οποίο απενεργοποιεί τον εχθρό.

3.3.2 MeleeAttack

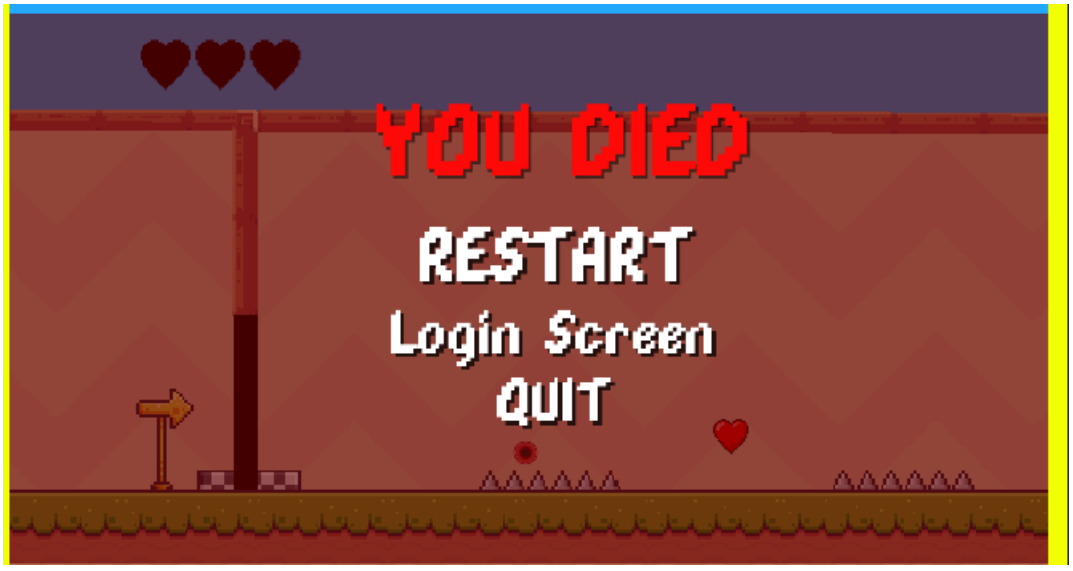


Το MeleeAttack Animation καλείται όταν ο παίχτης είναι κοντά στο Range του εχθρού, αυτό με την σειρά του ενεργοποιεί το Event το οποίο καλεί την συνάρτηση DamagePlayer() του κώδικα MeleeEnemy.cs.

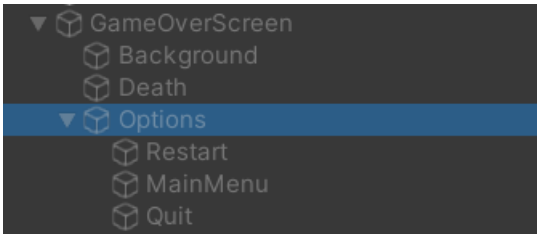
Αυτή η συνάρτηση ελέγχει εάν ο παίχτης – χαρακτήρας είναι στο Range και καλεί την συνάρτηση TakeDamage() για να αφαίρεση έναν πόντο ζωής.

Κεφάλαιο 4 – Συστήματα Παιχνιδιού

4.1 Death Menu



Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα GameOverScreen το οποίο έχει ως παιδία το Background, το Death Text (You Died) και ένα αντικείμενο με το όνομα options που έχει 3 παιδία (Restart, MainMenu και Quit).



Το Restart, το MainMenu και το Quit είναι TextMeshPro αντικείμενα που έχουν την ικανότητα ενός κουμπιού.

4.1.1 Restart

Αρχικά όταν πατηθεί το κουμπί της Restart καλείτε η συνάρτηση Restart() που βρίσκετε στον κώδικα UIManager.cs .

```
public void Restart()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
```

Βάση αυτής της συνάρτησης επαναφέρετε η τρέχουσα σκηνή που βρίσκεται ο παίχτης.

4.1.2 Login Screen

Αρχικά όταν πατηθεί το κουμπί του MainMenu καλείτε η συνάρτηση MainMenu() που βρίσκετε στον κώδικα UIManager.cs .

```
SceneManager.LoadScene(0);
```

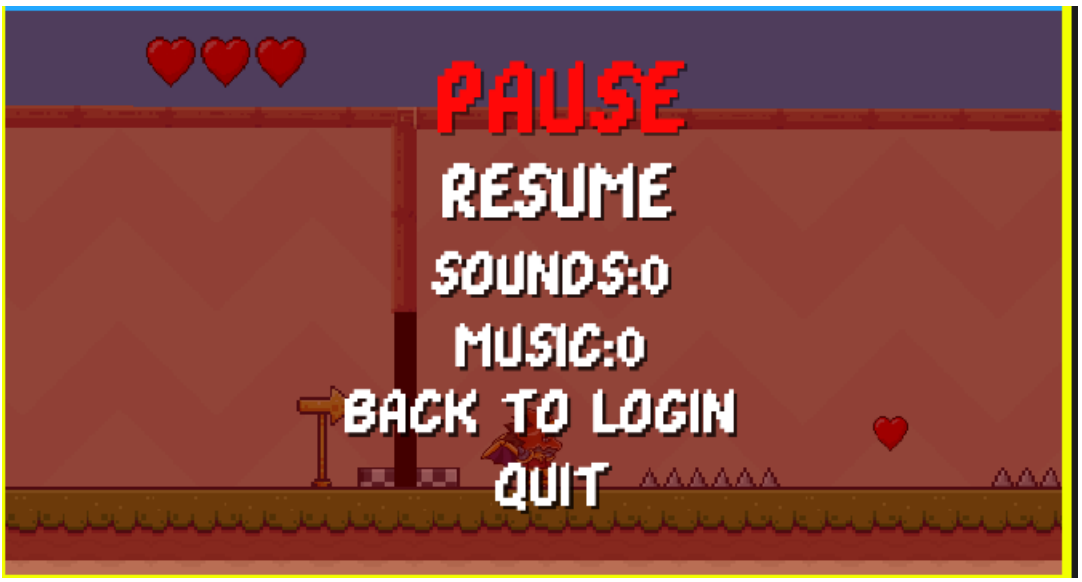
Βάση αυτής της λειτουργία μεταφέρετε ο χρήστης στην πρώτη σκήνη (Login Screen).

4.1.3 Quit

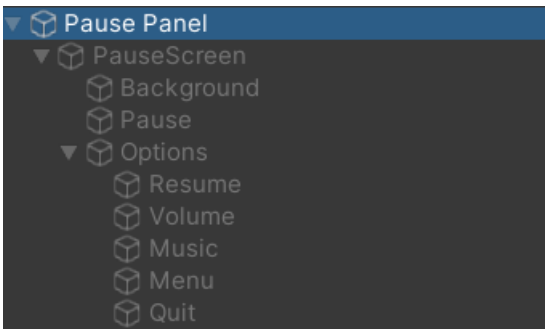
Αρχικά όταν πατηθεί το κουμπί του Quit καλείτε η συνάρτηση Quit() που βρίσκετε στον κώδικα UIManager.cs .Βάση αυτής της λειτουργίας το σύστημα της εφαρμογής τερματίζεται.

```
Application.Quit();
```

4.2 Pause Menu



Για να το υλοποιήσω έπρεπε να δημιουργήσω στην Ιεραρχία (hierarchy) του Unity ένα αντικείμενο με το όνομα Pause Panel το οποίο έχει ως παιδί PauseScreen. Μέσα σε αυτό το αντικείμενο υπάρχουν 3 παιδία. Το Background, το Pause και το αντικείμενο option που έχει 5 παιδία (Resume, Volume, Music, Menu και Quit).



Το Resume, το Volume, το Music, το Menu και το Quit είναι TextMeshPro αντικείμενα που έχουν την ικανότητα ενός κουμπιού.

Αρχικά όταν ο χρήστης πατήσει το σημαδάκι του Pause που βρίσκετε στην πάνω δεξιά πλευρά της οθόνης ενεργοποιείται η συνάρτηση TogglePause() η οποία απενεργοποιεί όλα τα εξαρτήματα που του έχω δηλώσει όπως είναι το Joystick , το Attack και το Jump.

4.2.1 Resume

Στην συνέχεια όταν πατηθεί το κουμπί του Resume καλείτε η συνάρτηση PauseGame() που βρίσκετε στον κώδικα UIManager.cs .

```
public void PauseGame(bool status)
{
    if (status == pauseScreen.activeInHierarchy)
    {
        return;
    }

    pauseScreen.SetActive(status);

    // Ensure Time.timeScale is set correctly
    Time.timeScale = status ? 0 : 1;
}
```

Βάση αυτής της λειτουργία ελέγχετε εάν η τρέχουσα κατάσταση είναι η ίδια με την κατάσταση αντικειμένου PauseScreen, εάν ναι τότε δεν κάνει τίποτα και εάν όχι τότε ενεργοποιεί το αντικείμενο PauseScreen και σταματά τον χρόνο του παιχνιδιού.

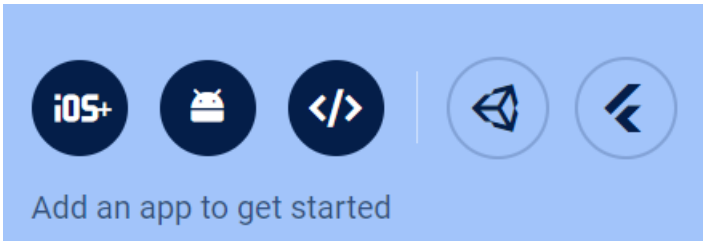
4.2.2 Volume – Music

Έπειτά όταν πατηθεί το κουμπί του Volume και του Music καλείτε η συνάρτηση SoundVolume() και MusicVolume() που βρίσκετε στον κώδικα UIManager.cs .Βάση αυτών των συναρτήσεων ρυθμίζεται η ένταση της μουσικής – ήχου.

Κεφάλαιο 5 – Ενσωμάτωση Βάσης

5.1 Firebase

Αρχικά για να συνδέσω την Firebase με το Unity έπρεπε να δημιουργήσω στην Console του Firebase ένα project, αυτό με την σειρά του πρέπει να προστεθεί στην εφαρμογή Unity.



Όταν επιλέξω το εικονίδιο του Unity μου εμφανίζεται η επιλογή για καταγραφή της εφαρμογής σε Android και iOS. Έπειτα για να ολοκληρωθεί η καταγραφή επιλέγω μία από τις 2 επιλογές και εισάγω το Package Name το οποίο μπορώ να το βρω στο Project Settings > Player > Identification > Bundle Identifier. Στην συνέχεια το Firebase φτιάχνει αυτόματα ένα Google Service Json και το αποθηκεύω στην εφαρμογή μου. Η Firebase για να συνδεθεί απόλυτα στο Unity χρειάζεται να κατεβάσω και να εισάγω το Google API που θέλω να χρησιμοποιήσω (Firebase Authentication).

Με την ένωση του Firebase στο Unity δημιούργησα ένα Login Screen για την είσοδο και εγγραφή του χρήστη (παίχτη) στο παιχνίδι.

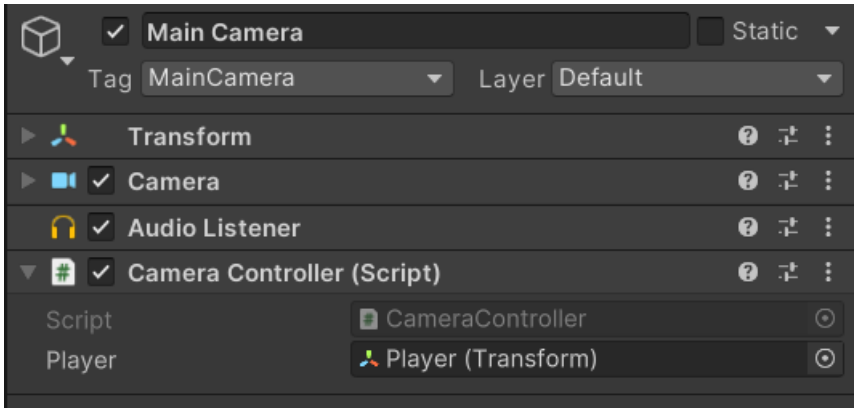
Το Login Button έχει την ιδιότητα να ελέγχει εάν ο χρήστης έχει λογαριασμό αποθηκευμένο στην βάση και εάν ναι τότε τον μεταφέρει στην σκήνη του παιχνιδιού. Στην περίπτωση που ο χρήστης καταχωρίσει λάθος στοιχεία του εμφανίζεται μήνυμα και του ζήτα να καταχωρίσει ορθά ή να εισάγει καινούργιο λογαριασμό.

Το Register Button έχει την ιδιότητα να καταγράφει και να αποθηκεύει τα στοιχεία του χρήστη στην Firebase.

Κεφάλαιο 6 – Camera Control

6.1 Camera

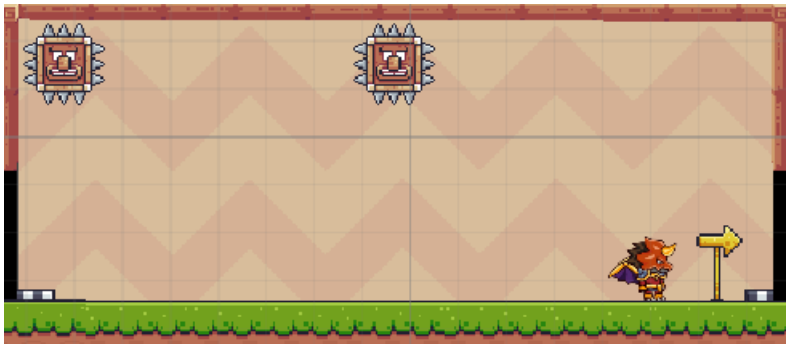
Για να φτιάξω την Camera του παίχτη έπρεπε να δημιουργήσω ένα αντικείμενο παιχνιδιού με το όνομα Main Camera και να προσθέσω επάνω σε αυτό την λειτουργία της Cameras.



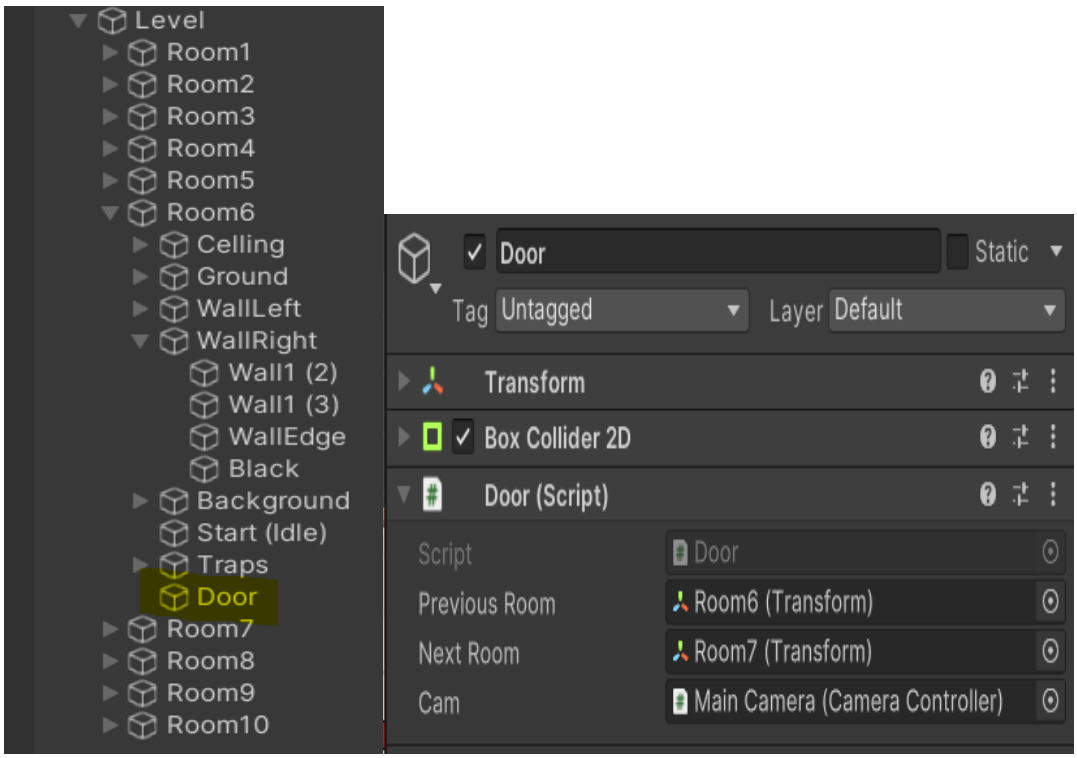
Έχω φτιαξει ένα κώδικα που τρέχει κατά την διάρκεια της εφαρμογής και ακολουθεί την τοποθεσία του παίχτη από frame σε frame.

```
0 references
private void Update()
{
    //Room Camera
    //transform.position = Vector3.SmoothDamp(transform.position, new Vector3(currentPosx, transform.position.
    // ref velocity, speed);

    //Follow player
    transform.position = new Vector3(player.position.x, transform.position.y, transform.position.z);
}
```

6.2 Door



Στο Room 6 έχω φτιάξει ένα αντικείμενο με το όνομα Door το οποίο με βοηθά να αποτρέψω την περίπτωση που τα Spike Heads ακολουθούν τον παίχτη στο επόμενο Δωμάτιο.

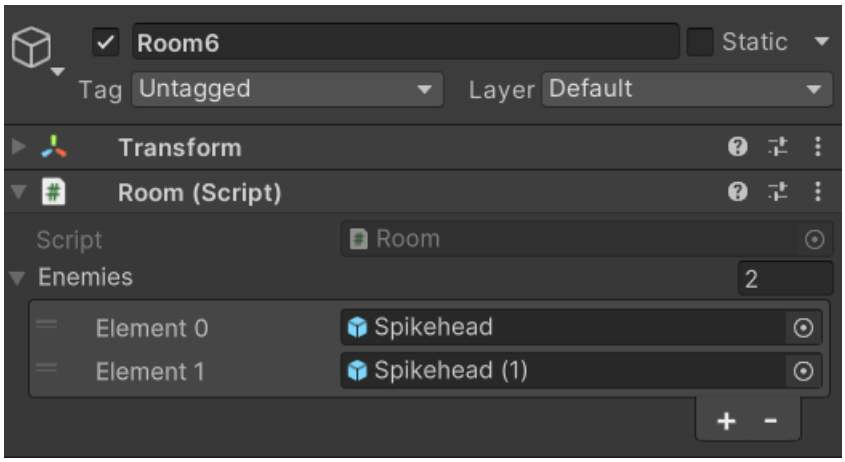


```
0 references
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player")
    {
        if (collision.transform.position.x < transform.position.x)
        {
            cam.MoveToNewRoom(nextRoom);
            nextRoom.GetComponent<Room>().ActivateRoom(true);
            previousRoom.GetComponent<Room>().ActivateRoom(false);
        }
        else
        {
            cam.MoveToNewRoom(previousRoom);
            previousRoom.GetComponent<Room>().ActivateRoom(true);
            nextRoom.GetComponent<Room>().ActivateRoom(false);
        }
    }
}
```

Έτσι λοιπόν έχω δημιουργήσει ένα κώδικα που ελέγχει εάν ο παίχτης έρθει σε επαφή με το Collision του Door και βάση αυτού ελέγχω τη φορά του παίχτη με την διαφορά του Collision. Έτσι λοιπόν ελέγχω εάν είναι αριστερά ή δεξιά και μεταφέρω την Camera στο αντίστοιχο δωμάτιο και απενεργοποιώ – ενεργοποιώ το αντίστοιχο δωμάτιο με την συνάρτηση ActiveRoom() την οποία θα μιλήσω σε βάθος αργότερα.

6.3 Rooms

Σε κάθε δωμάτιο έχω προσθέσει την λειτουργία Room.cs η οποία έχει όλα τα στοιχεία του δωματίου και με την βοήθεια του κώδικα Door.cs στην πορεία θα απενεργοποιηθούν.



Ο παρακάτω κώδικας έχει την λειτουργία να αποθηκεύει την αρχική τοποθεσία των στοιχείων και κατά την εκτέλεσης της συνάρτησης ActiveRoom να ενεργοποιεί – να απενεργοποιεί τα στοιχεία βάση του _status που έχει δοθεί από το Door.cs .

```
8 references
[SerializeField] private GameObject[] enemies;
3 references
private Vector3[] initialPosition;

0 references
private void Awake()
{
    //Save the initial positions of the enemies
    initialPosition = new Vector3[enemies.Length];
    for (int i = 0; i < enemies.Length; i++)
    {
        if(enemies[i] != null)
            initialPosition[i] = enemies[i].transform.position;
    }
}
4 references
public void ActivateRoom(bool _status)
{
    //Activate/deactivate enemies
    for (int i = 0; i < enemies.Length; i++)
    {
        if (enemies[i] != null)
        {
            enemies[i].SetActive(_status);
            enemies[i].transform.position = initialPosition[i];
        }
    }
}
```

Βιβλιογραφία

- Game Engine - Definition, Examples, History & More - Digital Art and Technology Glossary - jerwoodvisualarts.org
- Game engine - Wikipedia
- Unity (game engine) - Wikipedia
- The history of databases | ThinkAutomation
- A Timeline of Database History | Quickbase
- Database - Wikipedia
- Firebase - Wikipedia
- Unity - Scripting API: EventTrigger (unity3d.com)
- Unity - Scripting API: MonoBehaviour.StartCoroutine (unity3d.com)
- Unity - Scripting API: Collision2D (unity3d.com)
- Unity - Scripting API: Camera (unity3d.com)
- Unity - Scripting API: Time.timeScale (unity3d.com)

- Jack Brett, Alain Simons: Implementation of the Unity Engine for Developing 2D Mobile Games in Consideration of Start-Up/Student Developers. Edutainment 2017: 271-278

- Wooseong Jeong, Until Yun: Development of 2D Side-Scrolling Running Game Using the Unity 3D Game Engine. CSA/CUTE 2016: 132-136

- Luka Blaskovic, Alesandro Zuzic, Tihomir Orehovacki: Stranded Away: Implementation and User Experience Evaluation of an Indie Platformer Game Developed Using Unity Engine. MIPRO 2023: 92-97

- Fatima Sapundzhi, Anton Kitanov, Meglena D.Lazarova, Slavi G. Georgiev: Mobile Game Development Using Unity Engine. MIS4TEL (Workshops) 2023: 129-138

- Jun-Jie Hew, Voon-Hsien Lee, Soo-Ting T'ng, Garry Wei-Han Tan, Keng-Boon Ooi, Yogesh K. Dwivedi: Are Online Mobile Gamers Really Happy? On the Suppressor Role of Online Game Addiction. Inf .Syst. Frontiers 26(1): 217-249 (2024)

- Tselepatiotis Michail, Efthimios Alepis: Desing of Real-Time Multiplayer Word Game for the Android Platform Using Firebase and Fuzzy Logic. IISA 2023: 1-8

- Basu, Sourav (2017): 2D Platform Game: Developed using Unity Game Engine

- Patrick Felicia: A Beginner's Guide to 2D Platform Games with Unity: Create a Simple 2D Platform Game and Learn to Code in the Process

- Hannula, Toni (2021): Unity mobile application with a serverless Firebase backend

- Adam Sinicki (2017): Learn Unity for Android Game Development: A Guide to Game Design, Development, and Marketing

- P. Buttfield-Addison, J Manning, T Nugent – 2019: Unity game development cookbook: essentials for every game

- K Langley – 2015: Learning Unity IOS Game Development

Assets

- Knight Sprite Sheet (Free) | 2D Characters | Unity Asset Store

- Pixel Adventure 1 | 2D Characters | Unity Asset Store
- Joystick Pack | Input Management | Unity Asset Store