Αναφορά 1ης Άσκηση Unity
2024-2025

Ανάπτυξη Βιντεοπαιχνιδιών

(CEID_NE565)

Στοιχεία Φοιτητών

Υπεύθυνος Καθηγητής:

Κωνσταντίνος Τσίχλας

Μέλος 1

Ακαδημαϊκό Έτος : 9^{Ο} Εξάμηνο 2023-2024

Ονοματεπώνυμο: Ορέστης Αντώνιος Μακρής **Α-Μ** 1084516

Μέλος 2

Ακαδημαϊκό Έτος : 9^{Ο} Εξάμηνο 2023-2024

Ονοματεπώνυμο: Γρηγόρης Δελημπαλταδάκης **Α-Μ** 1084647

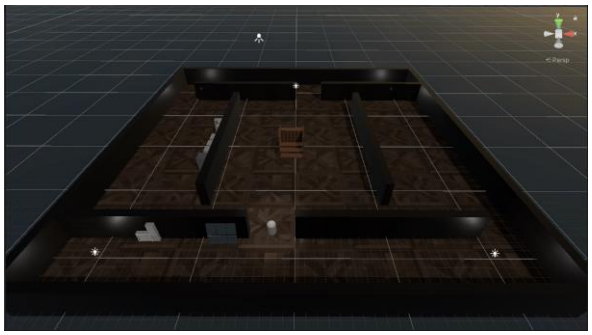
Πανεπιστήμιο Πατρών Τμήμα Μηχανικών Η/Υ και Πληροφορικής

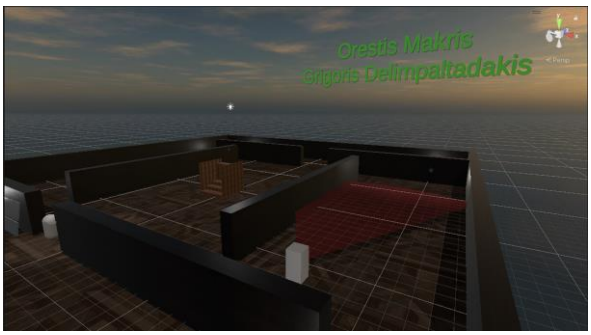
Πάτρα 2024

Contents

Τροποποιημένος Χάρτης

Τροποποιήθηκε ο κόσμος του Εργαστηρίου 3, προσθέτοντας μια πιο σκοτεινή και τρομακτική ατμόσφαιρα, που αρμόζει σε μια stealth σκηνή.





Ερώτημα 1 – Προσθήκη Ζωής στους Εχθρούς

Για την προσθήκη ζωής στον εχθρό χρησιμοποιείται το script EnemyHealth.cs στο EnemyPrefab.

Η ζωή του εχθρού ελέγχεται από την μεταβλητή health που αρχικοποιείται με την τιμή 3. Ο εχθρός λαμβάνει damage με την μέθοδο HurtEnemy η οποία μειώνει την ζωή του κατά ένα.

Στο script RayShooter.cs, όταν ο παίκτης κάνει click πάνω στον εχθρό καλείται η μέθοδος ReactToHit ενός αντικειμένου ReactiveTarget. Σε αυτήν την μέθοδο καλείται η συνάρτηση HurtEnemy, προκαλώντας damage στον εχθρό.

Όταν η τιμή της μεταβλητής health φτάσει στο μηδέν, η μεταβλητή isAlive του εχθρού τίθεται σε false, για να μην συνεχίσει να κινείται και να εκτοξεύει fireballs. Στην συνέχεια καλείται το Coroutine Die, όπου περιστρέφεται κατά -75 μοίρες στον άξονα X και μετά από 1.5 δευτερόλεπτα καταστρέφεται.

Ερώτημα 2 – Δυνατότητα Άλματος για τον Χαρακτήρα

Για την προσθήκη δυνατότητας άλματος στον player χρησιμοποιείται το script FPSInput.cs.

Ανάλυση Κώδικα:

Start() μέθοδος:

Στο ξεκίνημα, το script αποκτάει αναφορά στο component CharacterController, το οποίο είναι υπεύθυνο για τις φυσικές αλληλεπιδράσεις του χαρακτήρα με το περιβάλλον (όπως συγκρούσεις και κίνηση).

Λογική Άλματος

Έλεγχος Εδάφους:

Το script ελέγχει αν ο χαρακτήρας είναι στο έδαφος χρησιμοποιώντας την μέθοδο charController.isGrounded. Αν είναι αληθές, ο παίκτης επιτρέπεται να πηδήξει.

```
        if (charController.isGrounded)
```

Όταν ο χαρακτήρας είναι στο έδαφος, η ταχύτητα στον άξονα Y (verticalSpeed) μηδενίζεται για να αποτραπεί η συσσώρευση κάθετης ταχύτητας.

Πρώτο Άλμα:

Ο παίκτης μπορεί να κάνει άλμα πατώντας το πλήκτρο άλματος space στην δικιά μας περίπτωση και η ταχύτητα στον άξονα Y ορίζεται στην τιμή του jumpSpeed για να εκτοξεύσει τον χαρακτήρα προς τα πάνω.

Η σημαία canDoubleJump ορίζεται σε true, επιτρέποντας την δυνατότητα για διπλό άλμα όταν ο χαρακτήρας βρίσκεται στον αέρα.

Βαρύτητα:

Εάν ο παίκτης δεν βρίσκεται στο έδαφος, η βαρύτητα εφαρμόζεται συνεχώς αυξάνοντας την τιμή της verticalSpeed. Αυτό εξασφαλίζει ότι ο χαρακτήρας θα πέσει όταν δεν πηδάει.

Διπλό Άλμα:

Ο παίκτης μπορεί να κάνει ένα δεύτερο άλμα στον αέρα αν η σημαία canDoubleJump είναι true και το πλήκτρο άλματος πατηθεί. Μετά το δεύτερο άλμα, η σημαία canDoubleJump ορίζεται σε false για να αποτραπεί περαιτέρω άλμα.

```
        if (canDoubleJump && Input.GetButtonDown("Jump"))
```

Extra παράμετροι και Χρήση:

jumpSpeed (4.7f):

Η αρχική ταχύτητα στον άξονα Y όταν ο παίκτης κάνει άλμα. Η τιμή 4.7f επιλέχθηκε για να δώσει μια λογική ύψος άλματος χωρίς να είναι υπερβολικά υψηλό ή χαμηλό.

verticalSpeed (0.0f):

Αυτή η παράμετρος παρακολουθεί την τρέχουσα ταχύτητα στον άξονα Y, περιλαμβάνοντας τόσο την άνοδο όσο και την πτώση. Αρχικοποιείται σε 0.0f για να μην υπάρχει αρχική κάθετη κίνηση όταν ξεκινά το παιχνίδι.

canDoubleJump (bool):

Αυτή η σημαία επιτρέπει στον παίκτη να πραγματοποιήσει ένα δεύτερο άλμα στον αέρα. Ορίζεται σε true μετά το πρώτο άλμα και σε false μετά το δεύτερο για να περιορίσει τα άλματα σε δύο.

Υλοποίηση Bonus (Διπλό Άλμα):

Το διπλό άλμα υλοποιήθηκε επιτρέποντας την επαναφορά της verticalSpeed όταν ο παίκτης πατάει το πλήκτρο άλματος για δεύτερη φορά στον αέρα (εάν η σημαία canDoubleJump είναι true). Αυτό δίνει στον παίκτη μια επιπλέον δυνατότητα άλματος.

Ερώτημα 3 – Βελτίωση AI Εχθρών

Για την υλοποίηση του ερωτήματος 2 προστέθηκε το script EnemyAI.cs, το οποίο ελέγχει την δυνατότητα επίθεσης του εχθρού και την ελεύθερη κίνηση του στον χώρο ή την ικανότητα να ακολουθεί τον παίκτη όταν βρίσκεται στο πεδίο όρασης του. Παράλληλα για την οπτικοποίηση του πεδίου όρασης του εχθρό προστέθηκε το script EnemyFOV.cs.

EnemyAI.cs

Η λειτουργία ελεύθερης κίνησης και αποφυγή εμποδίων, καθώς επίσης και η επίθεση με χρήση fireballs, παραμένουν ίδιες με την υλοποίηση του Εργαστηρίου 3.

Για τον εντοπισμό και tracking του παίκτη, αρχικά υπολογίζεται το διάνυσμα κατεύθυνσης του παίκτη σε σχέση με τον εχθρό.

```
Vector3 playerDirection = playerPos - transform.position;
```

Στην συνέχεια υπολογίζεται η γωνία μεταξύ του forward direction διανύσματος του εχθρού και του διανύσματος τοποθεσίας του παίκτη, χρησιμοποιώντας την συνάρτηση Vector3.SignedAngle.

```
angleToPlayer = Vector3.SignedAngle(playerDirection, transform.forward, Vector3.up);
```

magnitude του διανύσματος playerDirection.

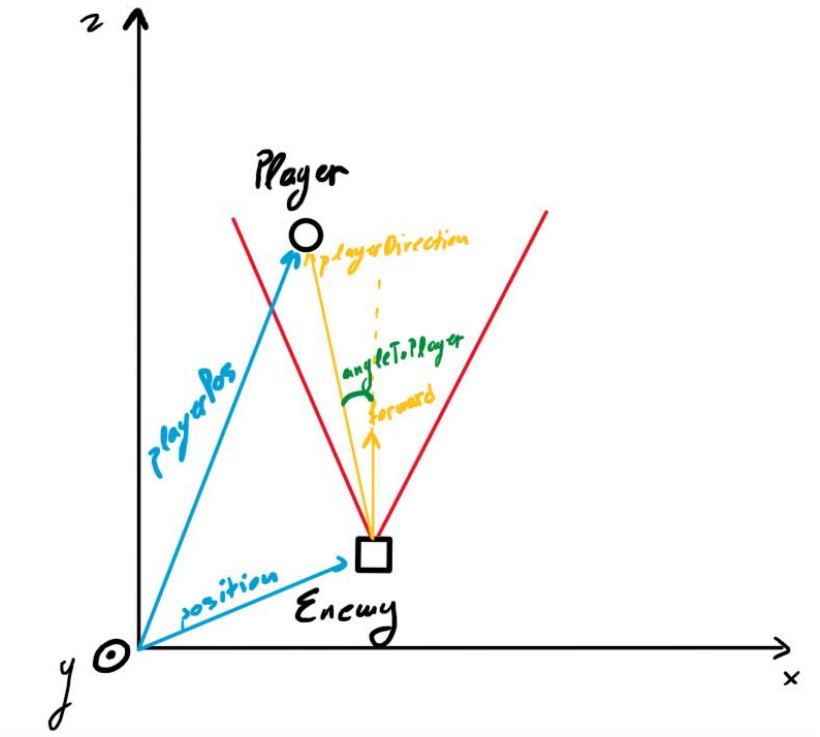
```
distanceToPlayer = playerDirection.magnitude;
```

Η γωνία και το μήκος της όρασης του εχθρού ορίζονται από τις μεταβλητές aiFieldOfView και aiSightRange.

```
public float aiFieldOfViewAngle = 60.0f;
public float aiSightRange = 20.0f;
```

Στην μέθοδο Update ελέγχεται αν η απόλυτη τιμή του angleToPlayer είναι μικρότερη από το aiFieldOfViewAngle/2 και αν η απόσταση του εχθρού από τον παίκτη είναι μικρότερη από το aiSightRange. Αν ισχύουν οι δύο παραπάνω συνθήκες, καλείται η συνάρτηση MoveToPlayer.

Σε αυτή την συνάρτηση ο εχθρός περιστρέφεται με συγκεκριμένη ταχύτητα προς τον παίκτη μέχρι να φτάσει η γωνία με το παίκτη κοντά στο μηδέν, ενώ παράλληλα προχωράει ευθεία.



Στην περίπτωση που φαίνεται παραπάνω, το angleToPlayer θα έχει θετική τιμή λόγω του κανόνα του δεξιού χεριού.

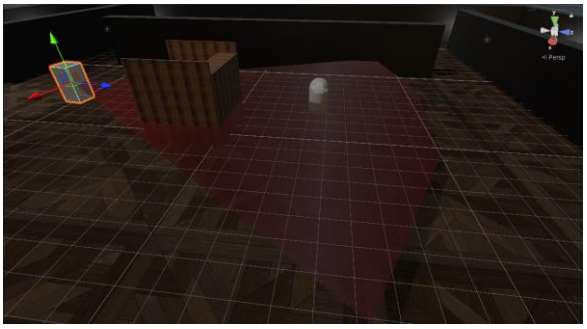
EnemyFOV.cs

Για την οπτικοποίηση του πεδίου όρασης του του εχθρού κατά την διάρκεια εκτέλεσης του παιχνιδιού παράγεται ένα σύνολο από τριγωνικά segments, που εξομοιώνουν ένα τμήμα κύκλου με γωνία aiFieldOfViewAngle και ακτίνα aiSightRange.

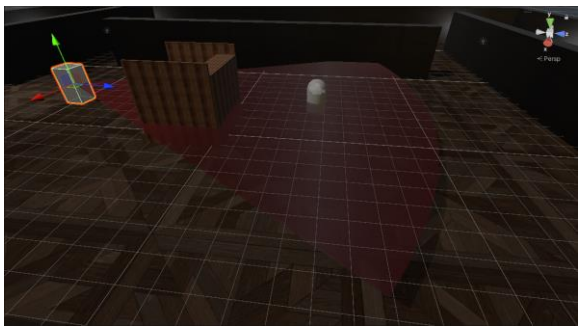
Το τριγωνικό mesh είναι rendered χρησιμοποιώντας τα components MeshFilter και MeshRenderer. Για κάθε τρίγωνο υπολογίζεται ή κατεύθυνση των δύο ακρών και η τοποθεσία των vertices βάση του aiSightRange.

```
        float angleStep = aiFieldOfViewAngle / triangleSegments;
        float startAngle = -aiFieldOfViewAngle / 2;

        for(int i = 0; i <= triangleSegments; i++)
        {
            float angle = startAngle + angleStep * i;
            Vector3 direction = Quaternion.Euler(0, angle, 0) * transform.forward;
            Vector3 vertex = enemyPos + direction * aiSightRange;
            mesh.Vertices.Add(vertex);

            if (i < triangleSegments)
            {
                mesh.Indices.Add(0);
                mesh.Indices.Add(i + 1);
                mesh.Indices.Add(i + 2);
            }
        }
```

Η επιλογή της συγκεκριμένης υλοποίησης έγινε για την παροχή της δυνατότητας ελέγχου της ανάλυσης του πεδίου όρασης.



Segments = 3



Segments = 20

Σύνδεσμος του Project Folder

https://drive.google.com/file/d/18fDif8Lb55rJGC1J502sZ3chINpNKHqt/view?usp=sharing