2024

ΑΣΚΗΣΗ 1 - UNITY

ΓΡΥΛΛΙΑΣ ΓΕΡΑΣΙΜΟΣ 1084651

1. Προσθήκη Ζωής στους Εχθρούς:

Η πρώτη σκέψη στο ερώτημα αυτό είναι να τροποποιήσουμε κατάλληλα τη μέθοδο ReactToHit() η οποία καλείται όταν ένα αντικείμενο χτυπηθεί:

Έτσι, η μέθοδος ReactToHit() θα μειώνει την υγεία του εχθρού αντί να τον σκοτώνει.

Πρώτα ορίζουμε μια μεταβλητή int με όνομα health και τιμή 3.

Μπαίνοντας στην νέα μέθοδο ReactToHit(), (μετά από κάθε επιτυχημένο χτύπημα) το πρώτο πράγμα που γίνεται είναι να μειώνει το health κατά 1 μονάδα.

Αμέσως μετά, κάνουμε έλεγχο με ένα if statement για να ελέγξουμε αν έχει απομείνει ζωή στον εχθρό:

Α) Αν δεν του έχει απομείνει ζωή (έχει δεχτεί 3 χτυπήματα) τότε καλείται η μέθοδος Die() για να τον σκοτώσει.

Β) Αν του έχει απομείνει ζωή, η μέθοδος ReactToHit() ολοκληρώνεται και ενεργοποιείται ξανά όταν υπάρξει νέο επιτυχημένο χτύπημα.

2. Δυνατότητα Άλματος για τον Χαρακτήρα:

Πριν ξεκινήσουμε με τη διαδικασία του άλματος θα βεβαιωθούμε ότι ο παίχτης βρίσκεται στο έδαφος με τη μέθοδο isGrounded() :

Για το λόγο αυτό δημιουργώ ένα νέο C# script που το ονομάζω Jump στο οποίο δημιουργώ αρχικά μια μεταβλητή controller για να του αναθέσουμε το game object του χαρακτήρα μας με την εντολή

controller = GetComponent<CharacterController>();

Στο μεταξύ, ορίζουμε άλλες δύο public μεταβλητές gravity και jumpSpeed που θα τις χρειαστούμε στο Jump, καθώς και ένα Vector3 για να δημιουργήσουμε την κατακόρυφη κίνηση.

Αμέσως μετά, στη συνάρτηση Update που τρέχει σε κάθε frame, ελέγχουμε με ένα if statement αν ο παίχτης βρίσκεται στο έδαφος και παράλληλα αν έχει πατηθεί το κουμπί space:

if (controller.isGrounded && Input.GetButtonDown("Jump"))

Αν αυτή η συνθήκη δώσει τιμή true, δίνουμε στον y άξονα του Vector3 την τιμή jumpSpeed για να μετακινήσουμε το object κατακόρυφα.

moveVelocity.y = jumpSpeed;

Βγαίνοντας από το if, αλλάζουμε ξανά την τιμή του y του Vector3 που είχαμε ανανεώσει μέσα στο if για να δημιουργήσουμε την βαρύτητα του αντικειμένου:
Για να γίνει αυτό ομαλά και το αντικείμενο να πέφτει προς το έδαφος πιο φυσικά, θα μειώνουμε την τιμή του z σε κάθε frame με την εντολή moveVelocity.y += gravity * Time.deltaTime;

(παρά το += operator η τιμή του moveVelocity.y μειώνεται διότι το gravity είναι ορισμένο με αρνητική τιμή).

Τέλος, δίνουμε την κίνηση στον χαρακτήρα μας με την εντολή controller.Move(moveVelocity * Time.deltaTime);

Υλοποίηση Double Jump:

Για την ανάγκη του ερωτήματος αυτού, στο ίδιο script θα ορίσουμε μία νέα μεταβλητή private bool doubleJump που

θα είναι true μόνο όταν ο παίχτης έχει τη δυνατότητα να κάνει διπλό άλμα στη διάρκεια του παιχνιδιού.
Μέσα στο ήδη υπάρχον if που ελέγχουμε αν ο παίχτης βρίσκεται στο έδαφος, θέτουμε την μεταβλητή doubleJump

σε true ώστε ο χαρακτήρας να έχει τη δυνατότητα του double jump.

Έπειτα, δημιουργούμε ένα νέο if statement που τροφοδοτείται μόνο όταν ο παίχτης δεν βρίσκεται στο έδαφος, ενώ παράλληλα πατιέται το κουμπί space και το doubleJump είναι true (δηλαδή έχει κάνει ήδη το 1ο jump).

if (!controller.isGrounded && Input.GetButtonDown("Jump") && doubleJump)

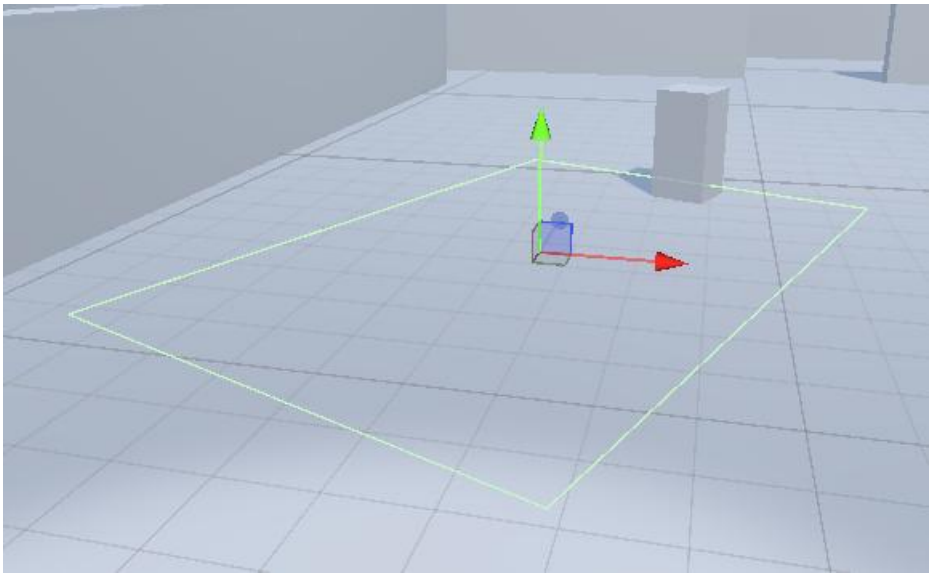
Μέσα σε αυτό το if τοποθετούμε την ίδια εντολή moveVelocity.y = jumpSpeed;

για να γίνει ένα ταυτόσημο jump με το αρχικό, ενώ έπειτα θέτουμε την τιμή του doubleJump σε false, ώστε να αποτραπεί επόμενο jump όσο βρίσκεται ακόμα στον αέρα.

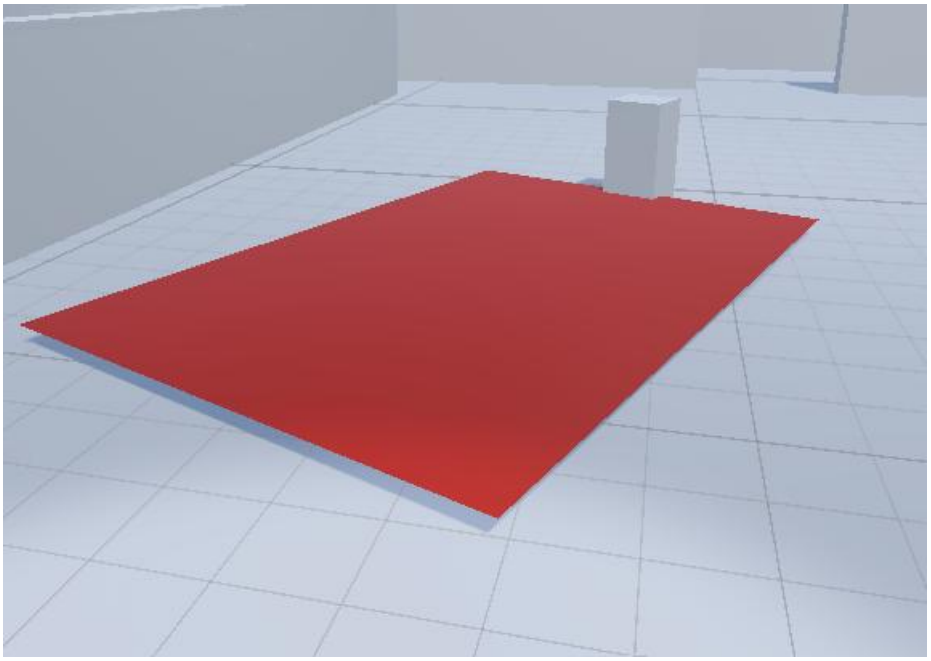
3. Βελτίωση AI Εχθρών:

Η πρώτη σκέψη σε αυτό το σημείο είναι να δημιουργήσουμε ένα 3d game object που θα παριστάνει την όραση του εχθρού:

Επιλέγω να χρησιμοποιήσω το object “plane” , καθώς υπάρχει έτοιμο στο Unity Editor και ρυθμίζοντάς το πάνω στον εχθρό μας, μπορεί να καλύψει ένα σημαντικό χώρο για την όραση του:



Στη συνέχεια, σε αυτό το νέο object “enemyVision” απενεργοποιώ το component Mesh Renderer που χρησιμοποιείται για να κάνει ορατό το μοντέλο (ή το αφήνω ενεργό βάζοντας και χρώμα για να φαίνεται η όραση του παίκτη), και από το component Box Collider ενεργοποιώ το Is Trigger για να επιτρέπουμε την ανίχνευση επαφής χωρίς να προκαλούνται φυσικές αλληλεπιδράσεις:



Τέλος, αποθηκεύω το νέο ολοκληρωμένο prefab ονομάζοντάς το “EnemywVision” σαν ένα νέο prefab.

Όσον αφορά την υλοποίηση, έχω δημιουργήσει ένα νέο C# script με όνομα fieldOfView το οποίο το κάνω attach στο child object enemyVision.

Εφόσον θέλουμε να ανιχνεύουμε πότε εισέρχεται σε αυτή την περιοχή ο **Player,** μέσα σε αυτό το script θα δημιουργήσουμε μια μεταβλητή τύπου GameObject και στη συνάρτηση start() θα την αναθέτουμε τον Player με την εντολή player = GameObject.Find("Player");

Δημιουργούμε ακόμα μια συνάρτηση OnTriggerStay που ενεργοποιείται μόνο όταν ο Player βρίσκεται μέσα στην περιοχή του Trigger.

Με την είσοδο του player στην περιοχή, παίρνουμε αρχικά το Vector3 της θέσης του Player: Vector3 position = other.transform.position;

Αμέσως μετά καλούμε τη συνάρτηση chasing με όρισμα το Vector3 του position του παίχτη την οποία την έχω υλοποιήσει στο script WanderingAI (και για αυτό το λόγο έχω δημιουργήσει μια μεταβλητή wanderingai τύπου WanderingAI για να έχουμε προσβαση στην συνάρτηση chasing).

Αυτή η συνάρτηση κανονικοποιεί το Vector, και αμέσως μετά με τη μέθοδο transform.Translate στέλνουμε τον εχθρό προς τον παίχτη.

Σημείωση:

Για να φαίνεται καλύτερα η υλοποίηση του ερωτήματος, έθεσα τον εχθρό να κινείται στον χώρο αρχικά με ταχύτητα walk = 1, και όταν εντοπίζει τον παίχτη να φαίνεται σαν να τρέχει κυνηγώντας τον, δηλαδή η ταχύτητα του να γίνεται speed = 3).