Αναφορά 2ης 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

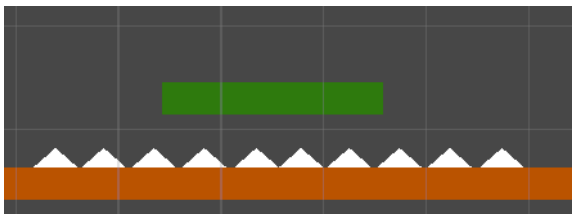
Ερώτημα 1 – Προσθήκη Κατάβασης από Πλατφόρμα

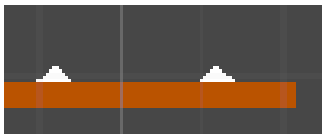
Η λειτουργία αυτή προστίθεται στο **PlatformerPlayer.cs** script. Όταν ο χρήστης πατήσει ‘S’ ή το κάτω βέλος και βρίσκεται πάνω σε πλατφόρμα, απενεργοποιείται προσωρινά το collision μεταξύ αυτής και του παίκτη, χρησιμοποιώντας το coroutine **IgnoreCollisions**. Το coroutine πραγματοποιεί κλήση του **Physics2D.IgnoreCollision** για 0.5 δευτερόλεπτα.

Ερώτημα 2 – Επέκταση του Επιπέδου

Για να εμπλουτιστεί η εμπειρία του παίκτη, προστέθηκαν διάφορα στοιχεία στον σχεδιασμό του επιπέδου:

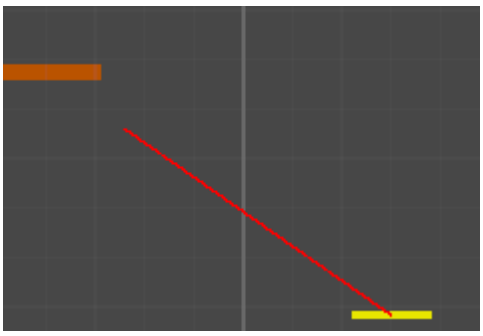
Αιχμές (Spike.cs)

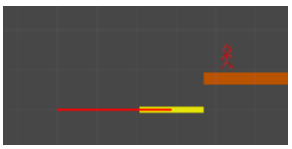




Οι αιχμές προκαλούν την επανεκκίνηση του επιπέδου όταν ο παίκτης έρχεται σε επαφή με αυτές. Το Spike.cs script συνδέεται με τα αντικείμενα που αντιπροσωπεύουν τις αιχμές στο επίπεδο. Όταν ο παίκτης (με το tag Player) μπαίνει στην περιοχή του trigger της αιχμής, η σκηνή του παιχνιδιού επαναφορτώνεται, ανανεώνοντας έτσι το επίπεδο (έχουμε χρησιμοποίηση polygon 2d collider για να έχουμε πολύ μεγάλη ακρίβεια).

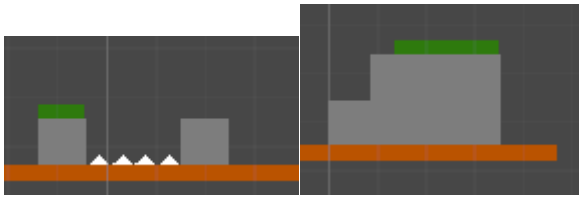
Κινούμενες Πλατφόρμες (MovingPlatform.cs)





Η κινούμενη πλατφόρμα προσθέτει δυναμικούς στόχους στο επίπεδο, οι οποίοι μετακινούνται μεταξύ δύο καθορισμένων σημείων. Η κίνηση της πλατφόρμας ελέγχεται από μια μεταβλητή ταχύτητας και ένα καθορισμένο σημείο εκκίνησης και τερματισμού. Η πλατφόρμα κινείται ομαλά πίσω και μπροστά μεταξύ αυτών των σημείων, με την κατεύθυνση να αναστρέφεται όταν φτάνει στα άκρα. Ο κώδικας χρησιμοποιεί την μεταβλητή trackPercent για να ελέγξει τη θέση της πλατφόρμας με βάση το χρόνο. Χρησιμοποιήθηκε η λογική και μέρη του κώδικα από το φροντιστήριο.

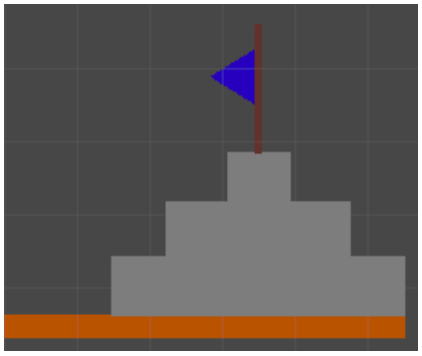
Τραμπολίνο (Trampoline.cs)



Το τραμπολίνο προσθέτει ένα διαδραστικό στοιχείο στο επίπεδο που εκτοξεύει τον παίκτη στον αέρα. Όταν ο παίκτης έρχεται σε επαφή με αυτό, η ταχύτητα του Rigidbody2D του παίκτη προσαρμόζεται,

δίνοντάς του μια δύναμη αναπήδησης. Η δύναμη αναπήδησης είναι παραμετροποιήσιμη μέσω της μεταβλητής bounceForce. Τα τραμπολινα στην πίστα έχουν χρώμα πράσινο.

Σημαία Νίκης (WinFlag.cs)



Η σημαία της νίκης σηματοδοτεί το τέλος του επιπέδου. Όταν ο παίκτης έρχεται σε επαφή με τη σημαία νίκης, εμφανίζεται ένα μήνυμα νίκης. Μετά την εμφάνιση του μηνύματος Win, το επίπεδο επανεκκινείται. Η σημαία νίκης ενεργοποιείται μέσω ενός Collider2D, και με την επαφή με τον παίκτη, καλείται η μέθοδος ShowWinMessage για να εμφανιστεί το μήνυμα νίκης. Το παιχνίδι επανεκκινείται μετά από 2 δευτερόλεπτα για να δώσει στον παίκτη λίγο χρόνο να γιορτάσει τη νίκη του πριν ξεκινήσει ξανά το επίπεδο.

Ανίχνευση Πτώσης (FallDetector.cs)

Ο ανιχνευτής πτώσης ανιχνεύει αν ο παίκτης πέφτει εκτός ορίων (π.χ. εκτός πλατφόρμας ή σε λάκκο). Όταν ο παίκτης έρχεται σε επαφή με τον Fall Detector Game Element, το επίπεδο επαναφορτώνεται και καταχωρείται μια αποτυχία. Αυτό επιτυγχάνεται με την προσθήκη ενός άδειου αντικειμένου GameObject με BoxCollider2D που έχει τεθεί σε trigger mode για την ανίχνευση της πτώσης του παίκτη.

Ερώτημα 3 – Προσθήκη Αντιπάλων με Διαφορετικά AI

Για τα δύο είδη εχθρών χρησιμοποιείται το ίδιο spreadsheet που έχει ο παίκτης, με το χρώμα αλλαγμένο σε κόκκινο. Συγκεκριμένα, για τον Fireball Enemy προστέθηκαν δύο frames ρίψης fireball.

Εχθρός που Κινείται Δεξιά – Αριστερά

Για την υλοποίηση αυτού του είδους εχθρού δημιουργήθηκε το **WanderingEnemy.cs** script. Ο εχθρός κινείται δεξιά – αριστερά σε ένα συγκεκριμένο εύρος, αλλάζοντας φορά στις άκρες. Χρησιμοποιείται ένα RigidBody2D component για την κίνηση, η ταχύτητα του οποίου ορίζεται βάση της επιλεγμένης δυσκολίας.

Το animation αυτού του εχθρού είναι πανομοιότυπο με αυτό του πάικτη.

Εχθρός με Σύστημα Εντοπισμού και Ρίψη Fireball

Ο συγκεκριμένος εχθρός παραμένει ακίνητος και διαθέτει ξεχωριστό animation. Για την υλοποίηση του δημιουργήθηκε ένα **Fireball prefab** με δικό του animation και τρία scripts, που εξηγούνται συνοπτικά παρακάτω.

FireballEnemy.cs:

Το script που προστίθεται στον εχθρό. Εντοπίζει την τοποθεσία του παίκτη και κάνει instantiate ένα Fireball prefab με velocity προς της διεύθυνση του παίκτη, όταν αυτός είναι εντός εύρους.

Παράλληλα μέσω του **fireball_instatiated** parameter, πραγματοποιείται η μετάβαση από το idle animation στο fireball-charging animation.

Fireball.cs:

Το script του Fireball prefab. Το fireball έχει συγκεκριμένη διάρκεια ζωής, για να μην καταναλώνει μνήμη όταν δεν πετύχει κάποιο εμπόδιο. Όταν χτυπήσει τον παίκτη προκαλεί επαναφόρτωση της σκηνής.

ThrowFireball.cs:

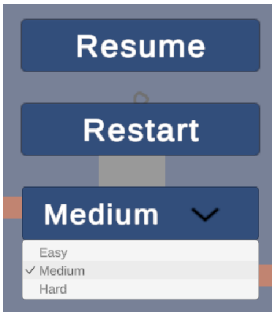
Αποτελεί behavior script της animation κατάστασης **Enemy_idle** του εχθρού. Εκτελείται όταν ολοκληρωθεί το **Charge_fireball** animation και καλεί την μέθοδο ThrowFireball του FireballEnemy.cs, για να πραγματοποιηθεί ρίψη fireball.

Ερώτημα 4 – Σύστημα Καταμέτρησης Αποτυχιών

Στο **PlayerStats.cs** script του player component, οι αποτυχίες του παίκτη καταμετρούνται αυξάνοντας την μεταβλητή failures κάθε φορά που καλείται η μέθοδος **AddFailure()**. Ο ενημερωμένος αριθμός των αποτυχιών αποθηκεύεται στα **PlayerPrefs**, για να διατηρηθεί μεταξύ sessions. Οι αποτυχίες απεικονίζονται σε πραγματικό χρόνο στο UI ενημερώνοντας το πεδίο failureText, που είναι τύπου TextMeshPro.

Ο αριθμός των αποτυχιών αυξάνεται κάθε φορά που ο παίκτης έρθει σε επαφή με fireball, spike, εχθρό ή όταν πέσει από το επίπεδο.

Ερώτημα 5 – Προσθήκη UI



Ο κύριος στόχος αυτής της λειτουργίας ήταν ο σχεδιασμός ενός μενού παύσης που επιτρέπει στον παίκτη να διακόπτει το παιχνίδι και να έχει πρόσβαση σε επιπλέον λειτουργίες, όπως η ρύθμιση της δυσκολίας και η επανεκκίνηση του παιχνιδιού. Η υλοποίηση πραγματοποιήθηκε ως εξής:

Το μενού παύσης ενεργοποιείται όταν ο παίκτης πατά το πλήκτρο Escape ή ‘P’ . Περιλαμβάνει τις εξής επιλογές: **Resume**: Κλείνει το μενού παύσης και συνεχίζει το παιχνίδι επαναφέροντας το Time.timeScale στο 1. **Restart**: Επαναφορτώνει την τρέχουσα σκηνή, επαναφέρει τις αποτυχίες του παίκτη και θέτει την δυσκολία σε Μεσαία. **Ρύθμιση Δυσκολίας**: Αλλάζει το επίπεδο δυσκολίας του παιχνιδιού, επηρεάζοντας την ταχύτητα εχθρών και αντικειμένων.

Για τις ρυθμίσεις δυσκολίας χρησιμοποιήθηκε ένα κουμπί τύπου dropdown, που παρέχει τρεις επιλογές: **Εύκολη**, **Μεσαία**, και **Δύσκολη**. Για το παραπάνω κουμπί δημιουργήθηκε ένα PauseManager Game Element, στο οποίο συνδέετε το script διαχειρίσεις του menu και εν συνέχεια αλληλοεπιδρά με το PauseMenuUI. Το επιλεγμένο επίπεδο δυσκολίας αποθηκεύεται στα PlayerPrefs για να διατηρηθεί μεταξύ sessions. Όταν ο χρήστης αλλάζει την δυσκολία, ενημερώνονται όλα τα instances τύπου Fireball Enemy και Wandering Enemy, αλλάζοντας την ταχύτητα του fireball και της κίνησης αντίστοιχα. Για να διασφαλιστεί μια συνεπής εμπειρία εκκίνησης, το επίπεδο δυσκολίας φορτώνεται από τα PlayerPrefs.

Σύνδεσμος του Project Folder

https://drive.google.com/file/d/1hCa12ZTs_uWMjXo0n7MdXu9MGeGWnxvb/view?usp=sharing