ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΥΛΟΠΟΙΗΣΗ ΔΙΣΔΙΑΣΤΑΤΟΥ ΠΑΙΧΝΙΔΙΟΥ ΣΤΗΝ ΜΗΧΑΝΗ ΠΑΙΧΝΙΔΙΩΝ UNITY 3D»

Μαραγγιανός Ιωάννης

Επιβλέπων: Δρ. Χριστοδούλου Σωτήριος, Επίκουρος Καθηγητής

Πάτρα 2023

Περίληψη

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη ενός συναρπαστικού διδακτικού παιχνιδιού δύο διαστάσεων, υλοποιημένου στην πλατφόρμα Unity 3D. Ο λόγος που επιλέξαμε αυτό το θέμα είναι η αξιοσημείωτη ανάπτυξη της αγοράς των βιντεοπαιχνιδιών, η οποία έχει ξεπεράσει ακόμα και τα έσοδα του κινηματογραφικού Hollywood.

Το παιχνίδι που δημιουργούμε είναι ένα κλασικό platformer, εμπνευσμένο από το Mario Bros, αλλά με ένα πρωτότυπο στοιχείο: το επίπεδο δυσκολίας αυξάνεται διαρκώς, φθάνοντας σε σημείο που μπορεί να αποτελέσει πρόκληση για τον παίκτη. Σκοπός της εργασίας μας είναι να αποδείξουμε ότι ο σχεδιασμός των μηχανισμών του παιχνιδιού και η δομή των επιπέδων μπορούν να δημιουργήσουν την εντύπωση στον παίκτη ότι, παρά την αυξημένη δυσκολία, η αποτυχία οφείλεται στις ικανότητές του, δίνοντας έτσι μια ευχάριστη πρόκληση στον παίκτη. Αυτή η εμπειρία ανακαλεί παιχνίδια όπως το διάσημο Flappy Bird.

Ελπίζουμε ότι η εργασία μας θα συμβάλει στην κατανόηση του σχεδιασμού παιχνιδιών και της επίδρασής του στην εμπειρία του παίκτη, καθώς και στην ανάδειξη της σημασίας των σωστών μηχανισμών και επιπέδων στην δημιουργία εθιστικών παιχνιδιών που κρατούν το ενδιαφέρον του κοινού.

Λέξεις Κλειδιά

Unity, Unity3D, platformer, Mario Bros, βιντεοπαιχνίδια, ανάπτυξη εφαρμογών, παιχνίδια 2D, παιχνίδια δύο διαστάσεων, τεχνολογία, συμπεριφορά παίκτη

Abstract

The present work focuses on the development of an engaging educational two-dimensional game implemented on the Unity 3D platform. The reason for choosing this topic lies in the remarkable growth of the video game market, surpassing even the revenues of the Hollywood film industry.

The game we are creating is a classic platformer, inspired by Mario Bros, but with an original twist: the difficulty level continually increases, reaching a point that can pose a challenge to the player. The aim of our work is to demonstrate that the game's mechanics and level design can create the impression for the player that, despite the heightened difficulty, failure is due to their own skills, providing an enjoyable challenge. This experience harkens back to games like the famous Flappy Bird.

We hope that our work will contribute to understanding game design and its impact on the player's experience, as well as highlighting the importance of proper game mechanics and level design in creating addictive games that capture the audience's interest.

Keywords

Unity, Unity3D, platformer, Mario Bros, video games, software development, 2D games, two dimensional games, technology, player behavior

Περιεχόμενα

Περιεχόμενα

Περιεχόμενα Εικόνων 5

Περιεχόμενα Πινάκων.....6

Εισαγωγή 7

Μηχανές Ανάπτυξης Παιχνιδιών 10

Unity.....10

Unreal Engine 11

CryEngine.....12

Godot Engine 13

JMonkey 14

GameMaker 15

3D Ogre 16

Unity και Unreal Engine 16

Επιλογή της Unity.....20

Σχεδιασμός Παιχνιδιού.....21

Εργαλεία Υλοποίησης.....22

Rendering Pipelines 25

Περιγραφή Παιχνιδιού 27

Χειρισμός Παιχνιδιού.....27

Υλοποίηση Παιχνιδιού.....28

Εκκίνηση του Παιχνιδιού 43

Συμπέρασμα.....47

Βιβλιογραφία 49

Περιεχόμενα Εικόνων

Εικόνα 1.Unity……...10

Εικόνα 2.Unreal Engine…….11

Εικόνα 3.Cry Engine……..12

Εικόνα 4.Godot Engine……..…13

Εικόνα 5.JMonkey…….14

Εικόνα 6.GameMaker……15

Εικόνα 7.3D Ogre……...16

Εικόνα 8. Κενός editor του Unity……...29

Εικόνα 9. Μια πρώτη διάταξη sprites για τη δημιουργία ενός background του παιχνιδιού…….30

Εικόνα 10. Το γραφικό που χρησιμοποιήθηκε για το idle animation του παίκτη……...31

Εικόνα 11. Το Sprite Renderer component πάνω στο game object “Sprite”, που ανήκει στον παίκτη…….32

Εικόνα 12. Ο Animator Controller του παίκτη……...33

Εικόνα 13. Κομμάτι του script Player Animator……..…35

Εικόνα 14. Το αρχικό μενού……..….43

Εικόνα 15. Επίπεδο του παιχνιδιού…….44

Εικόνα 16. Μενού παύσης……..…45

Εικόνα 17. Τέλος του παιχνιδιού……46

Περιεχόμενα Πινάκων

Πίνακας 1. Σύγκριση Unity και Unreal Engine……..….17

Πίνακας 2. Σύγκριση εργαλείων υλοποίησης IDE…….24

Εισαγωγή

Η ιστορία των βιντεοπαιχνιδιών, των κονσολών και των ηλεκτρονικών υπολογιστών είναι μια συναρπαστική αναδρομή στον κόσμο της ψυχαγωγίας και της τεχνολογίας. Από τα πρώτα βήματα στη δημιουργία των πρώτων παιχνιδιών μέχρι την εποχή των υπερσύγχρονων παιχνιδομηχανών και των εξελιγμένων υπολογιστών, η εξέλιξη αυτή έχει συνδυάσει τη δημιουργικότητα, την τεχνολογία και τη διασκέδαση.

Η αρχή των βιντεοπαιχνιδιών χρονολογείται στη δεκαετία του 1950, όταν οι πρώτοι υπολογιστές άρχισαν να είναι διαθέσιμοι. Ο Willy Higinbotham δημιούργησε το "Tennis for Two," ένα από τα πρώτα βιντεοπαιχνίδια, το 1958. Ωστόσο, το πρώτο εμπορικά επιτυχημένο βιντεοπαιχνίδι ήταν το "Pong" της Atari το 1972.

Η δεκαετία του '80 φέρνει την έκρηξη των βιντεοπαιχνιδιών με τον εμβληματικό χαρακτήρα Super Mario της Nintendo. Ακολούθησαν οι πρώτες φορητές κονσόλες, όπως το Game Boy. Η δεκαετία του '90 είδε την εμφάνιση των τρισδιάστατων γραφικών και τον ανταγωνισμό μεταξύ των Sega και Nintendo.

Οι κονσόλες όπως το PlayStation της Sony και το Xbox της Microsoft έκαναν την εμφάνισή τους στις αρχές της δεκαετίας του 2000 και ανοίγουν νέους ορίζοντες για το gaming. Τα online παιχνίδια και οι πολυπαίκτες λειτουργίες καθιερώθηκαν.

Οι πρώτες κονσόλες ήταν πρωτότυπα, όπως το Magnavox Odyssey το 1972. Η εξέλιξη οδήγησε στην κονσόλα Atari 2600, η οποία έγινε αρχέτυπο για την βιομηχανία. Οι πολεμικές κονσόλες όπως το Nintendo Entertainment System (NES) και το Sega Genesis κυριάρχησαν τη δεκαετία του '80.

Στα '90s, η Sony άνοιξε νέα μονοπάτια με το PlayStation, ενώ η Sega Saturn και η Nintendo 64 έκαναν τον ανταγωνισμό ακόμη πιο ενδιαφέρον. Τα επόμενα χρόνια είδαμε τον ανταγωνισμό μεταξύ των Sony PlayStation και Microsoft Xbox.

Σήμερα, οι κονσόλες είναι υψηλής τεχνολογίας με απίστευτη γραφική ποιότητα και ποικιλία παιχνιδιών.

Η ιστορία των ηλεκτρονικών υπολογιστών ξεκινά με τον πρώτο ηλεκτρονικό υπολογιστή ENIAC το 1945. Ακολούθησαν οι υπολογιστές της δεκαετίας του '50 και '60, που χρησιμοποιούνταν κυρίως για στρατιωτικούς και επιστημονικούς υπολογισμούς.

Οι προσωπικοί υπολογιστές έγιναν προσιτοί στη δεκαετία του '70 με τον εμβληματικό υπολογιστή Altair 8800. Ακολούθησαν οι Apple II και Commodore PET. Η δεκαετία του '80 είδε την εμφάνιση του IBM PC και του λειτουργικού συστήματος MS-DOS.

Τα '90s ήταν η εποχή των Windows και του Internet. Οι υπολογιστές έγιναν αναπόσπαστο μέρος της καθημερινής ζωής, και η τεχνολογία εξελίσσεται με ανεξάντλητους ρυθμούς.

Η ιστορία των βιντεοπαιχνιδιών, των κονσολών και των ηλεκτρονικών υπολογιστών αντικατοπτρίζει την εξέλιξη της τεχνολογίας και την ανθρώπινη δημιουργικότητα. Από τη δημιουργία του πρώτου παιχνιδιού Pong μέχρι την εποχή των υπερσύγχρονων gaming κονσολών και υπολογιστών, ο κόσμος του gaming έχει διανύσει μια εντυπωσιακή πορεία.

Οι διάφορες έρευνες που χρησιμοποιούν τα βιντεοπαιχνίδια ως μέσο για την εξερεύνηση των θεμάτων τους, έχουν αναδείξει ενδιαφέρουσες πτυχές σχετικές με την ψυχολογία, την εκπαίδευση, την υγεία και άλλα. Ανάμεσα στις αξιοσημείωτες έρευνες συμπεριλαμβάνονται:

1. Βελτίωση των Γνωστικών Ικανοτήτων: Πολλές έρευνες έχουν δείξει ότι τα βιντεοπαιχνίδια μπορούν να βελτιώσουν τις γνωστικές ικανότητες όπως η προσοχή, η μνήμη, και η επίλυση προβλημάτων. Για παράδειγμα, μια έρευνα δείχνει ότι το παιχνίδι "Tetris" βελτιώνει την επίδοση σε προσόντα εργασίες.

2. Θεραπεία και Ανακούφιση από τον Πόνο: Σε κάποιες έρευνες, τα βιντεοπαιχνίδια έχουν χρησιμοποιηθεί ως μέσο ανακούφισης από τον πόνο. Συγκεκριμένα, τα παιχνίδια είναι γνωστά ότι μπορούν να μειώσουν τον αναισθητικό πόνο και το άγχος σε ορισμένες περιπτώσεις.

3. Εκπαιδευτική Χρησιμοποίηση: Πολλές έρευνες εξετάζουν τη χρήση των βιντεοπαιχνιδιών στην εκπαίδευση. Εφαρμόσεις παιχνιδιών όπως το "Minecraft" έχουν αποδειχθεί χρήσιμες για την ανάπτυξη δεξιοτήτων όπως η δημιουργικότητα και η προγραμματισμένη σκέψη.

4. Διαχείριση του Στρες: Κάποιες έρευνες έχουν εξετάσει τη δυνατότητα των βιντεοπαιχνιδιών να βοηθήσουν στη διαχείριση του στρες. Παιχνίδια όπως το "Stardew Valley" μπορούν να προσφέρουν χαλάρωση και ψυχαγωγία.

Συνολικά, οι έρευνες με τη χρήση των βιντεοπαιχνιδιών έχουν αποκαλύψει μια ποικιλία τρόπων με τους οποίους τα παιχνίδια μπορούν να επηρεάσουν την ανθρώπινη συμπεριφορά και την εμπειρία. Αυτές οι έρευνες συμβάλλουν στην κατανόηση του πώς τα βιντεοπαιχνίδια μπορούν να

χρησιμοποιηθούν για το καλό της ανθρωπότητας και την ανάπτυξη διαφόρων πτυχών της ζωής μας.

Μηχανές Ανάπτυξης Παιχνιδιών

Οι μηχανές ανάπτυξης παιχνιδιών είναι λογισμικά εργαλεία που χρησιμοποιούνται για τη δημιουργία και τον προγραμματισμό βιντεοπαιχνιδιών. Κάθε μηχανή έχει τα δικά της χαρακτηριστικά και προτεραιότητες, ενώ διαφέρουν σημαντικά μεταξύ τους. Ας δούμε μερικές από τις δημοφιλέστερες μηχανές ανάπτυξης παιχνιδιών και παραδείγματα παιχνιδιών που έχουν δημιουργηθεί με την κάθε μηχανή:

Unity

Εικόνα 1: Unity

Η Unity είναι μία από τις πιο δημοφιλείς μηχανές και χρησιμοποιείται για τη δημιουργία παιχνιδιών σε πολλά είδη και πλατφόρμες. Διαθέτει ένα ισχυρό γραφικό περιβάλλον και είναι κατάλληλη για ανεξάρτητους προγραμματιστές. Παιχνίδια όπως το "Hollow Knight," "Cuphead," και "Among Us" δημιουργήθηκαν με τη χρήση της Unity.

- **Γλώσσα Προγραμματισμού:** C# είναι η κύρια γλώσσα προγραμματισμού που χρησιμοποιείται στην Unity.
- **Υποστηριζόμενες Πλατφόρμες:** Unity υποστηρίζει πολλές πλατφόρμες, συμπεριλαμβανομένων των Windows, macOS, Linux, Android, iOS, PlayStation, Xbox, Nintendo Switch, και πολλές άλλες.
- **Δυσκολία:** Η Unity είναι φιλική προς τους αρχάριους, αλλά παρέχει και προηγμένες δυνατότητες για προγραμματιστές με εμπειρία. Επίσης, η Unity δεν έχει συγκεκριμένες απαιτήσεις σε υλικό.

Unreal Engine

Η Unreal Engine διακρίνεται για τα υψηλής ποιότητας γραφικά και τη δυνατότητα δημιουργίας προηγμένων 3D παιχνιδιών. Χρησιμοποιείται συχνά από μεγάλες εταιρείες ανάπτυξης. Παιχνίδια όπως το "Fortnite," "Gears of War," " Final Fantasy 7 Remake Intergrade" και "Unreal Tournament" είναι παραδείγματα που χρησιμοποίησαν την Unreal Engine.

- **Γλώσσα Προγραμματισμού:** Η Unreal Engine χρησιμοποιεί τη γλώσσα προγραμματισμού C++ και διαθέτει επίσης το Blueprints, ένα γραφικό σύστημα προγραμματισμού.
- **Υποστηριζόμενες Πλατφόρμες:** Υποστηρίζει παρόμοιες πλατφόρμες με την Unity, συμπεριλαμβανομένων όλων των μεγάλων gaming κονσολών και υπολογιστών.

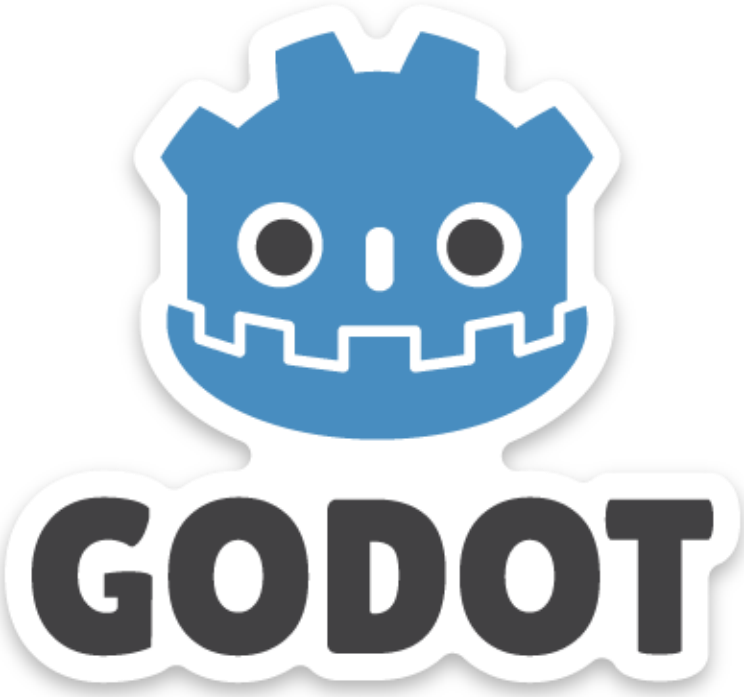
- **Δυσκολία:** Η Unreal Engine είναι πιο προηγμένη και πολύ ισχυρή, αλλά επομένως μπορεί να είναι πιο δύσκολη για αρχάριους. Επίσης απαιτεί συγκεκριμένες προδιαγραφές υλικού πράγμα το οποίο την κάνει ακόμη πιο δύσκολη.

CryEngine

Η CryEngine είναι γνωστή για τα εντυπωσιακά γραφικά και τον ρεαλισμό των περιβαλλόντων. Χρησιμοποιείται συχνά σε παιχνίδια περιπέτειας και επιβίωσης. Το "Far Cry" και το "Crysis" ανήκουν στα γνωστά παιχνίδια που χρησιμοποίησαν την CryEngine.

- **Γλώσσα Προγραμματισμού:** Κύρια γλώσσα προγραμματισμού είναι η C++, αλλά χρησιμοποιείται και η Lua για σκριπτάρισμα.
- **Υποστηριζόμενες Πλατφόρμες:** Στο παρελθόν είχε ειδική έκδοση για Crytek κονσόλες, αλλά πλέον υποστηρίζει κυρίως Windows, PlayStation και Xbox.
- **Δυσκολία:** Η CryEngine είναι αρκετά προηγμένη και μπορεί να είναι δύσκολη για αρχάριους. Δύσχρηστη ειδικά για την υλοποίηση 2D παιχνιδιών, έχει μεγάλες απαιτήσεις υλικού καθώς επίσης απευθύνεται σε μια πιο κοινότητα χρηστών.

Godot Engine



Το Godot είναι ανοικτού κώδικα και δωρεάν μηχανή ανάπτυξης παιχνιδιών. Είναι εξαιρετικά προσαρμόσιμο και κατάλληλο για ανεξάρτητους προγραμματιστές. Παιχνίδια όπως το "Hollow Knight: Silk song" και το "Hyper Light Drifter" δημιουργήθηκαν με την Godot Engine.

- **Γλώσσα Προγραμματισμού:** Υποστηρίζει διάφορες γλώσσες προγραμματισμού, με την GDScript να είναι παρόμοια με την Python.
- **Υποστηριζόμενες Πλατφόρμες:** Υποστηρίζει πολλές πλατφόρμες, συμπεριλαμβανομένων των Windows, macOS, Linux, Android, iOS, PlayStation, Xbox, και άλλες.
- **Δυσκολία:** Η Godot Engine είναι φιλική προς τους αρχάριους και η GDScript είναι εύκολη για την εκμάθηση, καθιστώντας την μια καλή επιλογή για νέους προγραμματιστές. Ωστόσο παρουσιάζει αδυναμίες στη φυσική των 2D παιχνιδιών σε σχέση με τις άλλες μηχανές ανάπτυξης.

Κάθε μηχανή προσφέρει μοναδικές δυνατότητες και επιτρέπει στους δημιουργούς να εκφραστούν με τρόπους που ανταποκρίνονται στις ανάγκες και τις φιλοδοξίες τους. Οι περισσότερες από αυτές παρέχουν δωρεάν εκδόσεις ή μοντέλα συνδρομής, προσφέροντας έτσι επιλογές για όλους, ανεξαρτήτως επιπέδου εμπειρίας.

Αξίζει να σημειωθεί ότι υπάρχουν πολλές ακόμα μηχανές ανάπτυξης παιχνιδιών όπως:

JMonkey

Το JMonkey είναι μια μηχανή ανάπτυξης παιχνιδιών ανοιχτού κώδικα. Είναι μια δωρεάν πλατφόρμα ανοιχτού κώδικα και η οποία έκανε για πρώτη φορά την εμφάνιση της το 2003 και είναι κατάλληλη για την ανάπτυξη και υλοποίηση τόσο 2D (δυσδιάστατων) όσο και 3D (τρισδιάστατων) παιχνιδιών και η οποία χρησιμοποιεί ως γλώσσα προγραμματισμού την Java. Οι υποστηριζόμενες πλατφόρμες της Monkey είναι τα Windows, Linux, macOS, Raspberry Pi, Android, και iOS. Επίσης μερικά από τα παιχνίδια της JMonkey είναι το “Nord”, το “Drohtin” και το “Chaos”

GameMaker

Η μηχανή παιχνιδιών GameMaker ή Animo όπως ήταν το όνομα της όταν κυκλοφόρησε το 1991 και φτιάχτηκε από τον Mark Overmars. Βασίζεται στη γλώσσα προγραμματισμού C++ και είναι κατάλληλη κυρίως για αρχάριους προγραμματιστές καθώς δεν απαιτούσε γνώσεις προγραμματισμού. Είναι μια μηχανή η οποία αρχικά χρησιμοποιήθηκε για την ανάπτυξη 2D παιχνιδιών και στη συνέχεια και 3D. Η GameMaker υποστηρίζεται από όλες τις υπάρχουσες πλατφόρμες: Microsoft Windows, macOS, Ubuntu, HTML5, Android, iOS, Amazon Fire TV, Android TV, Raspberry Pi, Microsoft UWP, PlayStation 4, Nintendo Switch and Xbox One, PlayStation 5 και Xbox Series X|S.

3D Ogre

Η μηχανή παιχνιδιών 3D Ogre είναι μια δωρεάν μηχανή παιχνιδιών ανοιχτού κώδικα. Η κυκλοφορία ξεκίνησε τον Νοέμβριο του 2013 και έκτοτε έχει και αυτή τη θέση της ανάμεσα στις υπόλοιπες μηχανές ανάπτυξης παιχνιδιών καθώς υποστηρίζει πολλές από τις υπάρχουσες πλατφόρμες όπως Windows, iOS, και Android. Από τα πιο γνωστά παιχνίδια της 3D Ogre είναι το “Shadows: Heretic Kingdom” και το “Running with Rifles”.

Unity και Unreal Engine

Για αυτή την πτυχιακή εργασία επέλεξα το Unity και την Unreal Engine για περαιτέρω ανάλυση καθώς είναι οι δυο πιο γνωστές και πιο χρησιμοποιήσιμες μηχανές ανάπτυξης παιχνιδιών.

Ευκολία χρήσης

Σε αυτή την κατηγόρια κερδίζει το Unity καθώς πολλοί χρήστες βρήκαν ότι το Unity είναι ελαφρώς πιο εύχρηστο, χάρη στην εγγενή γλώσσα προγραμματισμού C# η οποία είναι σχετικά

πιο γνωστή για όλους τους προγραμματιστές και συνολικά έχει καλύτερο περιβάλλον διεπαφής. Είναι μια πιο εύκολη πλατφόρμα για πιο αρχάριους και νέους game developers για να ξεκινήσουν ενώ από την άλλη η Unreal Engine, έχει μια ελαφρώς πιο απότομη καμπύλη εκμάθησης.

Οπτικά Εφέ και Ποιότητα

Σε αυτή την κατηγόρια κερδίζει η Unreal Engine. Ενώ και οι δύο πλατφόρμες παράγουν υψηλής ποιότητας οπτικά εφέ, οι περισσότεροι χρήστες διαπίστωσαν ότι το Unreal Engine έχει ένα μικρό πλεονέκτημα έναντι του Unity στην ποιότητα των οπτικών εφέ του. Μπορεί να δημιουργήσει φωτορεαλιστικές απεικονίσεις που βυθίζουν τους παίκτες και τους επιτρέπουν να ταξιδεύουν ελεύθερα σε έναν εκπληκτικό νέο κόσμο και να ενσωματώνουν στοιχεία υψηλής ποιότητας από διάφορες πηγές. Αν για παράδειγμα κάποιος θέλει μια λάμψη φακού που ή θέλει να δημιουργήσει σύνθετες υφές σε έναν εξωγήινο πλανήτη, η Unreal Engine είναι αυτή που θα το καταφέρει.

Απόδοση

Και σε αυτή την κατηγόρια νικητής είναι η Unreal Engine. Παρόμοια με το ελαφρύ πλεονέκτημά του με την ποιότητα γραφικών, η ποιότητα και η ταχύτητα απόδοσης στο Unreal Engine ξεπερνούν ελαφρώς το Unity σε αυτήν την κατηγορία. Παράγει κορυφαίες στον κλάδο ταχύτητες απόδοσης και οι χρήστες είναι σταθερά ευχαριστημένοι με την ποιότητα των γραφικών που αποδίδονται.

Animation

Εδώ νικητής είναι η Unreal Engine. Σε αυτή την κατηγόρια η Unreal Engine πραγματικά ξεχωρίζει. Με τις ισχυρές του δυνατότητες απόδοσης και τα κορυφαία οπτικά εφέ, η ποιότητα κινούμενων εικόνων (animation) δεν τίθενται ως θέμα σύγκρισης. Το συντριπτικό ποσοστό μεταξύ των χρηστών στην ποιότητα των εργαλείων κινουμένων σχεδίων και στις αποδόσεις κινούμενων εικόνων επέλεξαν την Unreal Engine.

Ομαδική Συνεργασία

Σε αυτή την κατηγόρια υπερτερεί η Unity. Η Unity κατέχει μεγαλύτερο μερίδιο αγοράς 63% έναντι 13% της Unreal Engine. Αυτό σημαίνει απλώς ότι περισσότεροι άνθρωποι χρησιμοποιούν το Unity από το Unreal Engine, πράγμα που σημαίνει ότι που ενδιαφέρεται να ασχοληθεί ενεργά

με το game developing θα έχει περισσότερους συνεργάτες στο Unity από ότι συνεργάτες στο Unreal Engine, δίνοντας στο Unity ένα μικρό πλεονέκτημα σε αυτήν την κατηγορία.

Συγγραφή

Σε αυτή την κατηγόρια υπάρχει ισοπαλία. Και οι δύο πλατφόρμες διαθέτουν εξαιρετικά εργαλεία συγγραφής που θα επιτρέπουν να γράψετε το σενάριο του παιχνιδιού. Όσον αφορά τη συγγραφή, οποιαδήποτε πλατφόρμα θα σας παρέχει όλη τη λειτουργικότητα που χρειάζεστε για να γράψετε γρήγορα και απρόσκοπτα το παιχνίδι.

Ποιότητα υποστήριξης

Ξανά ισοπαλία καθώς και οι δύο πλατφόρμες διαθέτουν υπηρεσίες υποστήριξης 24 ώρες το 24ωρο, 7 ημέρες την εβδομάδα, που μπορούν να βοηθήσουν τον μέσο χρήστη σε όλα τα ζητήματα που μπορεί να αντιμετωπίσει. Ανεξάρτητα από την πλατφόρμα που θα επιλέξει, θα έχει μια ομάδα που θα τον υποστηρίζει για να βεβαιωθεί ότι τα πράγματα συνεχίζονται ομαλά.

Βαθμολογία G2

“Η βαθμολογία G2 υπολογίζεται ως ο μέσος όρος των βαθμολογιών Ικανοποίησης και Παρουσίας στην αγορά. Για παράδειγμα, εάν ένα προϊόν με βαθμολογία ικανοποίησης 70 και βαθμολογία Παρουσίας στην αγορά 60 καταλήγει να έχει βαθμολογία G2 65.”

Ισοπαλία και πάλι. Το G2, μια από τις πιο αξιόπιστες πηγές για κριτικές λογισμικού, βαθμολόγησε τόσο το Unity όσο και το Unreal Engine ως σταθερά 4,5/5 αστέρια, υπογραμμίζοντας περαιτέρω το πόσο κοντά είναι αυτές οι πλατφόρμες όσον αφορά τη δημοτικότητα και την ποιότητα των χρηστών.

Τιμολόγηση

Unity: Το Unity έχει μια δωρεάν έκδοση, αλλά για να ξεκλειδώσετε όλες τις λειτουργίες του, θα χρειαστεί να κάνετε αναβάθμιση στην έκδοση Pro, η οποία είναι διαθέσιμη με μηνιαία συνδρομή 75 $ ανά μήνα.

Unreal Engine: Η Unreal Engine έχει ένα μοναδικό μοντέλο τιμολόγησης. Το ίδιο το λογισμικό είναι δωρεάν, αλλά μετά την κυκλοφορία του παιχνιδιού ενός χρήστη, η Unreal Engine δικαιούται ένα τέλος 5% για όλες τις πωλήσεις παιχνιδιών.

	Unity	**Unreal Engine**
Ευκολία χρήσης	Φιλική προς νέους προγραμματιστές	Έμπειροι προγραμματιστές
Ανοιχτός κώδικας	Όχι	Ναι
Απαιτήσεις Υλικού	Όχι	Ναι
Γλώσσα Προγραμματισμού	C#	C++
Υποστηριζόμενες Πλατφόρμες	Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft HoloLens, Nintendo 3DS family, Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS, WebGL	Windows PC, PlayStation 5, PlayStation 4, Xbox Series X, Xbox Series S, Xbox One, Nintendo Switch, macOS, iOS, Android, ARKit, ARCore, OpenXR, SteamVR, Oculus, Linux, and SteamDeck.
Δικτύωση	Client-Server, Multiplayer	Client-Server, Multiplayer
Εργαλεία Προγραμματιστών	Windows Phone SDK	Windows Phone SDK, iOS SDK, Android SDK, Android GDK

Πίνακας 1: Σύγκριση Unity και Unreal Engine

Συμπεράσματα

Το Unity είναι μια εξαιρετική πλατφόρμα για τους νέους game developers που μόλις ξεκινούν. Με τη γλώσσα προγραμματισμού C# και μια τεράστια κοινότητα άλλων προγραμματιστών και σχεδιαστών, το Unity είναι μια εξαιρετική πλατφόρμα για νέους game developers που θέλουν να

ξεκινήσουν να δημιουργούν αμέσως και δεν θέλουν να οφείλουν στην πλατφόρμα δικαιώματα από τα παιχνίδια τους. Από την άλλη το Unreal Engine έχει να κάνει με τα βελτιωμένα γραφικά και τις αστραπιαίες ταχύτητες απόδοσης, καθιστώντας το ιδανικό για έμπειρους προγραμματιστές παιχνιδιών σε επίπεδο επιχείρησης ή προγραμματιστές που θέλουν αυτή την εξαιρετική ποιότητα στα παιχνίδια τους και δεν τους πειράζει να οφείλουν τα δικαιώματα στην εκάστοτε εταιρία. Η καμπύλη εκμάθησης είναι συγκριτικά πιο απότομη και χρειάζεται αρκετό χρόνο και πειραματισμό για την εξοικείωση και την αποτελεσματική χρήση για την συγγραφή, ανάπτυξη και υλοποίηση ενός παιχνιδιού.

Επιλογή της Unity

Λαμβάνοντας υπόψιν τις παραπάνω συγκρίσεις ανάμεσα σε Unity και Unreal Engine και πιο συγκριμένα των χαρακτηριστικών που διαθέτει η κάθε μηχανή ανάπτυξης αλλά και τα πλεονεκτήματα και μειονεκτήματα , επέλεξα το Unity για την δημιουργία του δικού μου 2D παιχνιδιού, καθώς οι λειτουργίες, τα χαρακτηριστικά και οι ανάγκες του project μου καλύπτονται σε μεγαλύτερο βαθμό από αυτή. Ως νέος στον χώρο game developing το περιβάλλον και η διεπαφή χρήστη του Unity ήταν αρκετά ευκολότερο όπως επίσης οι απαιτήσεις λογισμικού είναι μικρότερες. Πράγματα που επίσης βοήθησαν στην επιλογή μου αυτή ήταν η γλώσσα προγραμματισμού C# στην οποία διαθέτω μεγαλύτερη οικειότητα. Επιπλέον, η μεγάλη ποικιλία από δωρεάν assets στο Asset Store της πλατφόρμας αλλά και από assetσ που είναι με πληρωμή, το μεγάλο πλήθος από παραδείγματα, tutorials, και βίντεο σχετικά με το ξεκίνημα και τα πρώτα βήματα πάνω στην ανάπτυξη βιντεοπαιχνιδιών έπαιξαν πολύ καθοριστικό ρόλο στην επιλογή μου καθιστώντας το Unity ως τη καταλληλότερη πλατφόρμα για την δημιουργία του πρώτου μου βιντεοπαιχνιδιού.

Σχεδιασμός Παιχνιδιού

Παρακάτω θα αναλυθεί η διαδικασία που ακολουθήσαμε για το παιχνίδι που αναπτύξαμε. Το πρώτο βήμα ήταν ο σχεδιασμός του παιχνιδιού.

Στοχεύοντας την μελέτη στην ψυχολογία του παίκτη, ήταν σημαντικό να διαλέξουμε μία κατηγορία παιχνιδιών χωρίς πολύπλοκους μηχανισμούς, προκειμένου να είναι εύκολο στον οποιοδήποτε χρήστη να το μάθει και να το παίξει. Έτσι, καταφύγαμε στα παλιά παιχνίδια που έβγαζαν αντίστοιχα συναισθήματα στον χρήστη, ότι δηλαδή χάνει λόγω δικής του έλλειψης ικανοτήτων και όχι λόγω του κακού σχεδιασμού του παιχνιδιού. Καταλήξαμε να χρησιμοποιήσουμε ως έμπνευση την σειρά παιχνιδιών Super Mario Bros.

Ύστερα, αρχίσαμε τον σχεδιασμό του παιχνιδιού, κρατώντας κατά νου την έμπνευσή μας. Αποφασίσαμε να βγάλουμε την λειτουργία scrolling, δηλαδή ότι όσο ο παίκτης κινείται, τον ακολουθεί η κάμερα, και να περιορίσουμε το παιχνίδι σε μία οθόνη. Θέλοντας να φτιάξουμε ένα παιχνίδι platformer ικανοτήτων, αποφασίσαμε να βρούμε ένα σύστημα φυσικής το οποίο συνεργάζεται με το σύστημα φυσικής της μηχανής παιχνιδιών που θα επιλέξουμε να αναπτύξουμε το παιχνίδι.

Επιλέξαμε την Unity για να αναπτύξουμε το παιχνίδι, καθώς βρήκαμε αρκετό υλικό είτε σε tutorials που υπάρχουν στο internet για να μας καθοδηγήσει να κάνουμε την ανάπτυξη συστημάτων που χρειαζόμαστε, καθώς επίσης η Unity μας παρείχε μεγαλύτερο πλήθος από δωρεάν assets. Αρχικά υλοποιήσαμε την κίνηση του παίκτη, και την αλληλεπίδραση του τόσο με τον τερματισμό του παιχνιδιού, όσο και με τους τοίχους, πάτωμα αλλά και τις παγίδες που υπάρχουν στις πίστες. Ύστερα αναπτύξαμε την λογική που ήταν απαραίτητη για την σωστή λειτουργία και εναλλαγή μεταξύ των animations του παίκτη. Στη συνέχεια βάλαμε μερικούς βασικούς ήχους που συνδέονται με την κίνηση του παίκτη (άλμα, προσγείωση και θάνατος) αλλά

και γραφικά απαραίτητα για τον σχεδιασμό των επιπέδων του παιχνιδιού. Σχεδιάσαμε και αναπτύξαμε ένα σύστημα αποθήκευσης της προόδου του παίκτη, το οποίο έχει και εναλλακτική μέθοδο αποθήκευσης σε περίπτωση που η πρώτη αποτύχει, και τρέχει αυτόματα στην αρχή του παιχνιδιού και κατά την ολοκλήρωση κάθε πίστας.

Εργαλεία Υλοποίησης

Για την υλοποίηση του παιχνιδιού χρησιμοποιήσαμε την μηχανή παιχνιδιών Unity. Η Unity είναι μία μηχανή παιχνιδιών, δωρεάν στην χρήση της για προσωπική και φοιτητική χρήση, αλλά και για επαγγελματική χρήση όσο η εταιρεία έχει έσοδα λιγότερα από ένα συγκεκριμένο ποσό ετησίως. Υπάρχουν διάφοροι τρόποι για προγραμματισμό στην Unity, οι δύο πιο δημοφιλείς είναι γράφοντας κώδικα με την γλώσσα C#, και γράφοντας λογική με κόμβους, γνωστό ως Visual Scripting. Για το δικό μου παιχνίδι, έγραψα κώδικα C#. Επιπλέον, υπάρχουν πολλά πακέτα που παρέχονται από δημιουργούς και προγραμματιστές δωρεάν στο Asset Store της Unity, κάτι που εκμεταλλεύτηκα για να γλιτώσω πολύτιμο χρόνο στην δημιουργία γραφικών και να μπορέσουμε να συγκεντρωθούμε στον προγραμματισμό του παιχνιδιού αλλά και στον σχεδιασμό του ώστε να μείνει λιτό και κατανοητό και ταυτόχρονα να προσφέρει την εμπειρία που ψάχνουμε να δημιουργήσουμε. Ωστόσο ένα στοιχείο μπορεί να προέρχεται από ένα αρχείο που δημιουργήθηκε εκτός του Unity, όπως ένα τρισδιάστατο μοντέλο, ένα αρχείο ήχου, μια εικόνα ή οποιονδήποτε από τους άλλους τύπους αρχείων που υποστηρίζει το Unity. Υπάρχουν επίσης ορισμένοι τύποι στοιχείων που μπορούν να δημιουργηθούν στο Unity, όπως ένας Ελεγκτής Animator, ένας Μείκτης ήχου ή μια υφή απόδοσης.

Για την συγγραφή και διόρθωση του κώδικα χρησιμοποιήσαμε το Visual Studio Community 2022. Το Visual Studio Community είναι ένα Ολοκληρωμένο Περιβάλλον Ανάπτυξης (από εδώ και στο εξής αναφέρεται ως ‘IDE’) που παρέχει η Microsoft δωρεάν. Ένα IDE είναι ένα πολύτιμο εργαλείο για τους προγραμματιστές, τόσο για προγραμματισμό στην Unity αλλά και γενικότερο προγραμματισμό. Ένα IDE προσφέρει ένα ολοκληρωμένο περιβάλλον ανάπτυξης όπου ο προγραμματιστής μπορεί να γράψει, να δοκιμάσει και να διαχειριστεί τον κώδικά του. Αυτό

συμπεριλαμβάνει χαρακτηριστικά όπως την επεξεργασία κειμένου, την αυτόματη συμπλήρωση κώδικα, την αποσφαλμάτωση (debugging) και τη δυνατότητα ενσωμάτωσης τρίτων βιβλιοθηκών.

Ο κώδικας C# που χρησιμοποιείται για την ανάπτυξη παιχνιδιών στην Unity είναι πολύπλοκος και απαιτεί προσεκτικό προγραμματισμό. Ένα IDE παρέχει εργαλεία που επιτρέπουν τον αποτελεσματικό προγραμματισμό, βοηθώντας τους προγραμματιστές να γράψουν σωστό κώδικα και να παρακολουθούν την εξέλιξη του. Ένα IDE παρέχει εργαλεία για την παρακολούθηση της εξέλιξης του κώδικα, την διαχείριση αλλαγών και τη συνεργασία με άλλους προγραμματιστές. Αυτό είναι κρίσιμο όταν εργάζεστε σε ένα μεγάλο παιχνίδι με ομάδα προγραμματιστών. Επίσης, το IDE παρέχει εργαλεία για την εκτέλεση και την αποσφαλμάτωση του κώδικα, επιτρέποντας στον προγραμματιστή να δοκιμάσει τον κώδικα του σε διάφορες περιπτώσεις και συνθήκες. Αυτό βοηθά στον εντοπισμό και την επίλυση προβλημάτων πριν την τελική ανάπτυξη. Σε ένα IDE, ο προγραμματιστής μπορεί να διαχειριστεί τον κώδικα του παιχνιδιού, τις βιβλιοθήκες, τις εξαρτήσεις και τα έργα του, όλα από ένα ενοποιημένο περιβάλλον. Αυτό έχει ως αποτέλεσμα την ευκολότερη ανάπτυξη, διαχείριση και συντήρηση του παιχνιδιού, γλιτώνοντας πολύτιμο χρόνο.

Συνοψίζοντας, ένα IDE είναι απαραίτητο για την ανάπτυξη παιχνιδιών στην Unity με χρήση της γλώσσας προγραμματισμού C#. Παρέχει την τεχνολογική υποδομή και τα εργαλεία που απαιτούνται για να γράψετε, να δοκιμάσετε και να διαχειριστείτε αποτελεσματικά τον κώδικά σας, κάνοντας τον ανάπτυξη παιχνιδιών μια σχετικά εύκολη και αποτελεσματική διαδικασία.

Το Visual Studio Community δεν είναι η μοναδική επιλογή παρόλα αυτά για IDE. Οι βασικές διαφορές μεταξύ του Visual Studio 2022 Community, του Visual Studio 2022 Professional και του JetBrains Rider όσον αφορά τη χρήση τους για τον κώδικα C# αναλύονται παρακάτω:

1. **Visual Studio 2022 Community:**
    - **Δωρεάν Εκδοχή:** Το Visual Studio 2022 Community είναι μια δωρεάν έκδοση που παρέχεται από τη Microsoft για προγραμματιστές και φοιτητές με μικρές ή μη εμπορικές ομάδες.
    - **Περιορισμοί:** Παρόλο που προσφέρει πολλές από τις βασικές λειτουργίες του Visual Studio, έχει ορισμένους περιορισμούς όπως προορίζεται για μη εμπορική χρήση και μικρές ομάδες έως 5 άτομα.
2. **Visual Studio 2022 Professional:**
    - **Επαγγελματική Έκδοση:** Το Visual Studio 2022 Professional είναι μια πλήρως επαγγελματική έκδοση της IDE που παρέχεται από τη Microsoft.
    - **Ευρεία Υποστήριξη:** Προσφέρει πλήρη υποστήριξη για όλες τις λειτουργίες ανάπτυξης και διαχείρισης έργων, συμπεριλαμβανομένης της ανάπτυξης C# και της διαχείρισης των έργων της ομάδας.
3. **JetBrains Rider:**
    - **Πολυλειτουργικό:** Το JetBrains Rider είναι μια πολυλειτουργική IDE που διαθέτει υποστήριξη για πολλές γλώσσες προγραμματισμού, συμπεριλαμβανομένης της C#.
    - **Επαγγελματική Χρήση:** Το Rider είναι κατάλληλο για επαγγελματική χρήση και προσφέρει προηγμένες λειτουργίες για προγραμματιστές C#. Είναι ιδανικό για ανάπτυξη σε πλατφόρμες όπως το .NET και το ASP.NET.
    - **Διαχείριση Έργων:** Παρέχει προηγμένες δυνατότητες διαχείρισης έργων και ομάδας.

	Visual Studio 2022 Community	**Visual Studio 2022 Professional**	**JetBrains Rider**
Δωρεάν Εκδοχή	Ναι	Όχι	Ναι
Περιορισμοί	Ναι	Όχι	Όχι
Επαγγελματική Έκδοση	Όχι	Ναι	Ναι
Υποστήριξη	Ναι	Ναι	Ναι
Πολυλειτουργικό	Όχι	Ναι	Ναι
Διαχείριση Έργων	Ναι	Ναι	Ναι

Πίνακας 2: Σύγκριση εργαλείων υλοποίησης IDE

Οι διαφορές μεταξύ αυτών των εκδόσεων κυμαίνονται από την τιμή και τους περιορισμούς στην περίπτωση του Visual Studio Community, μέχρι την πληρότητα των λειτουργιών και την πολυλειτουργικότητα του Rider.

Rendering Pipelines

Τα Rendering Pipelines στη Unity είναι κρίσιμα στοιχεία του γραφικού συστήματος που καθορίζουν πώς το παιχνίδι αποτυπώνεται γραφικά στην οθόνη του παίκτη. Αντιπροσωπεύουν τη διαδικασία που ακολουθεί η Unity για να αποδώσει τα γραφικά, από την επεξεργασία των γραφικών πόρων (textures, models, animations) μέχρι την παρουσίαση τους στην οθόνη του παίκτη. Υπάρχουν τρεις προκαθορισμένοι τύποι rendering pipelines: Το Built-In, το Universal, και το High Definition. Επιπλέον οι χρήστες έχουν την δυνατότητα να κάνουν ένα Custom Rendering Pipeline ακριβώς στα μέτρα του πρότζεκτ τους και στις ανάγκες τους.

Το Built-in Rendering Pipeline είναι το παραδοσιακό rendering pipeline της Unity και χρησιμοποιείται ευρέως για παιχνίδια 3D και 2D. Χρησιμοποιεί φωτισμό και σκιές προσέγγισης, όπου τα υλικά χρησιμοποιούν τις λειτουργίες διάχυσης και ανάκλασης. Διαθέτει λειτουργία forward rendering για λίγες πηγές φωτός και deferred rendering για πολλές πηγές φωτός. Είναι εύκολο στη ρύθμιση και κατάλληλο για παιχνίδια με μικρότερες απαιτήσεις σε γραφικά. Το "Hollow Knight", ένα 2D Metroidvania παιχνίδι, χρησιμοποιεί το Built-in Rendering Pipeline για την απεικόνιση του σκοτεινού κόσμου του. Παρόλο που τα γραφικά του είναι υπέροχα, το παιχνίδι δεν χρειάζεται την πολυπλοκότητα των πιο προηγμένων pipelines.

Το Universal Rendering Pipeline είναι ένα προηγμένο rendering pipeline που προορίζεται για διακριτικά γραφικά και υψηλή απόδοση. Είναι σχεδιασμένο για να είναι πολυλειτουργικό, υποστηρίζοντας τόσο 2D όσο και 3D παιχνίδια. Χρησιμοποιεί Shader Graph για τη δημιουργία προηγμένων υλικών και προσφέρει προσεγγιστικό φωτισμό και σκιές που διατηρούν την απόδοση.

Είναι ιδανικό για παιχνίδια που απαιτούν υψηλή απόδοση σε διάφορες πλατφόρμες. Το "Hades" είναι ένα παιχνίδι δράσης-ρογκλάικ με υψηλή ποιότητα γραφικών. Χρησιμοποιεί το Universal Rendering Pipeline για να διατηρήσει την απόδοση σε υψηλά επίπεδα ενώ παράλληλα προσφέρει εντυπωσιακά γραφικά σε πολλές πλατφόρμες.

Το High Definition Rendering Pipeline (HDRP) είναι ένα πολύ προηγμένο rendering pipeline για υψηλή ποιότητα γραφικών και ρεαλιστικό φωτισμό. Υποστηρίζει προηγμένους υλικούς χαρακτηριστικούς φωτισμό και σκιές και προσφέρει προηγμένη υποστήριξη για post-processing εφέ. Χρησιμοποιεί σύγχρονες τεχνικές όπως ray tracing για πραγματιστικό φωτισμό. Είναι κατάλληλο για παιχνίδια που επιδιώκουν υψηλή αισθητική και λεπτομέρεια. Το "Cyberpunk 2077" είναι ένα από τα πιο περίπλοκα και γραφικά εντυπωσιακά παιχνίδια που έχουν κυκλοφορήσει. Χρησιμοποιεί το HDRP για να προσφέρει ρεαλιστική απόδοση φωτισμού, σκιών και υλικών.

Τέλος, υπάρχουν τα Custom Rendering Pipelines**.** Ένα Custom Rendering Pipeline επιτρέπει στους προγραμματιστές να δημιουργήσουν το δικό τους, προσαρμοσμένο rendering pipeline. Παρέχει ακόμη μεγαλύτερο έλεγχο για τη διαδικασία αποτύπωσης γραφικών. Συνήθως χρησιμοποιείται για παιχνίδια που απαιτούν μοναδικά γραφικά και εφέ. Το "Inside" είναι ένα αριστούργημα τέχνης και γραφικής σχεδίασης. Το παιχνίδι χρησιμοποιεί ένα προσαρμοσμένο rendering pipeline για να δημιουργήσει μια μοναδική αισθητική και εφέ που ταιριάζουν με την ατμόσφαιρα του παιχνιδιού.

Συνοψίζοντας, τα Rendering Pipelines στη Unity δίνουν στους προγραμματιστές την δυνατότητα να επιλέξουν το κατάλληλο pipeline ανάλογα με τις ανάγκες του παιχνιδιού τους. Κάθε ένα από αυτά έχει τα δικά του πλεονεκτήματα και περιορισμούς, επιτρέποντας την δημιουργία παιχνιδιών με διαφορετικά επίπεδα γραφικής απόδοσης και αισθητικής.

Στο δικό μας project, χρησιμοποιήσαμε το Built-In Rendering Pipeline καθώς ο σκοπός του πρότζεκτ συγκεντρώνεται στην δημιουργία ενός συγκεκριμένου συναισθήματος στον παίκτη μέσω του σχεδιασμού των μηχανισμών του παιχνιδιού αλλά και των επιπέδων του παιχνιδιού, και όχι μέσω περίπλοκων γραφικών ή οπτικών εφέ.

Περιγραφή Παιχνιδιού

Το παιχνίδι χαρακτηρίζεται κυρίως ως παιχνίδι δεξιοτήτων και περιλαμβάνει 5 επίπεδα στο καθένα από τα οποία έχουν παγίδες αλλά και ένα τρόπαιο στο οποίο πρέπει να φτάσει ο παίκτης ώστε να κερδίσει το εκάστοτε επίπεδο και να πάει στο επόμενο. Η λογική του παιχνιδιού θυμίζει και άλλα παιχνίδια του ιδίου τύπου όπως το "Super Mario Bros" αλλά και ένα από τα δημοφιλέστερα και δυσκολότερα παιχνίδια των τελευταίων ετών "World's Hardest game". Ωστόσο κάτι που πρέπει να σημειωθεί είναι πως σε αυτό το παιχνίδι δεν υπάρχει κάποιος περιορισμός όπως ζωές ή χρόνος έτσι ώστε ο χρήστης να μπορεί να παίξει για όσο αυτός επιθυμεί. Για τους λόγους της πτυχιακής τα επίπεδα του παιχνιδιού είναι περιορισμένα καθώς είναι μόνο ένα δείγμα προς παρουσίαση. Σε άλλη περίπτωση ο βαθμός δυσκολίας θα ανεβαίνει όλο και περισσότερο με το πέρας κάθε επιπέδου βάζοντας μέσα ακόμα περισσότερα εμπόδια, παγίδες, εχθρούς, ακόμα και αντίστροφη χρονομέτρηση για την επίτευξη του στόχου.

Χειρισμός Παιχνιδιού

Ο χειρισμός του βασικού μενού και του μενού της παύσης του παιχνιδιού από τον χρήστη γίνεται αποκλειστικά με το ποντίκι ώστε να κάνει την επιλογή του. Από την άλλη πλευρά κατά την διάρκεια όμως του παιχνιδιού χρησιμοποιείται αποκλειστικά το πληκτρολόγιο.

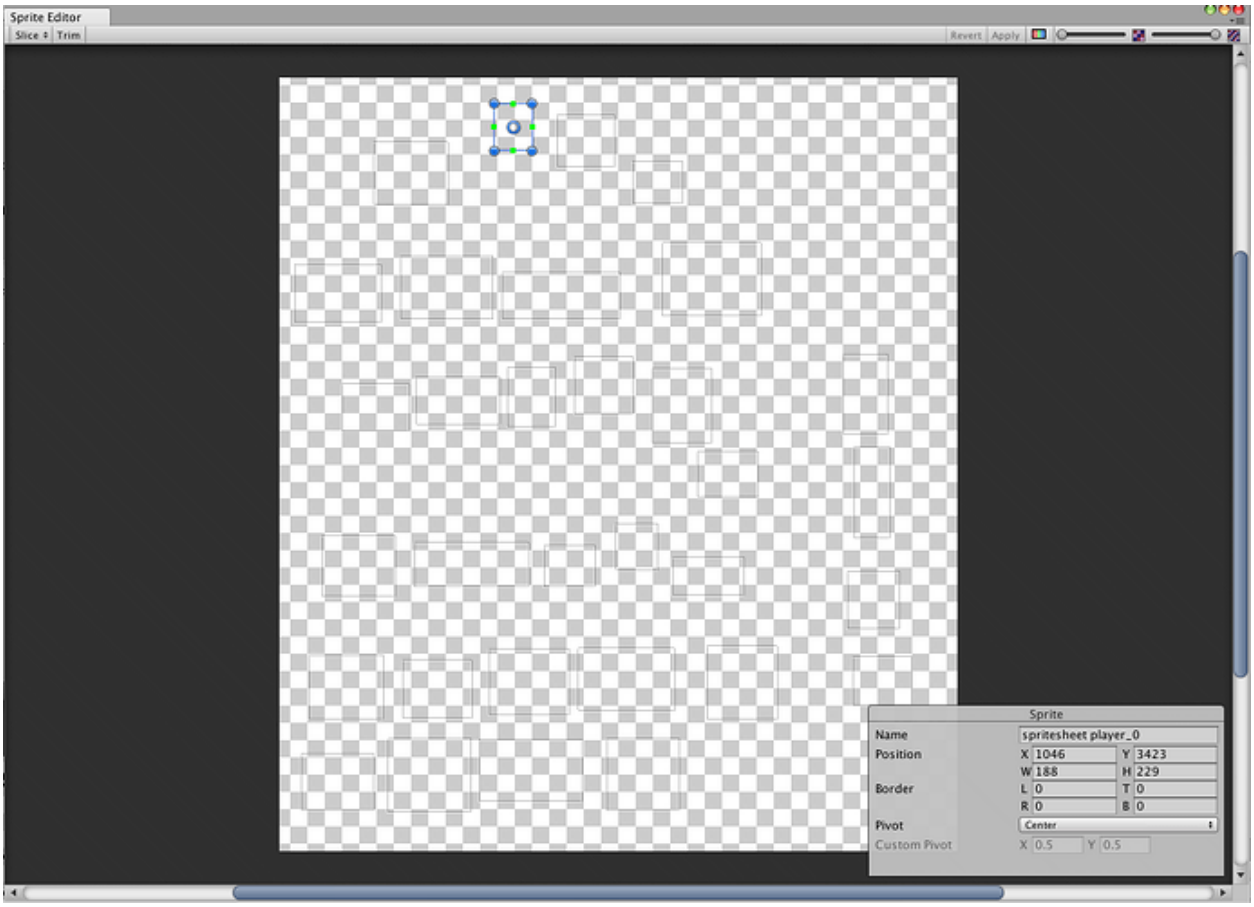
Τα πλήκτρα χειρισμού αποτελούνται από τα παρακάτω:

- Spacebar → Άλμα
- Δεξί βελάκι → Κίνηση δεξιά
- Αριστερό βελάκι → Κίνηση αριστερά
- Πλήκτρο R → Για εκκίνηση του παίχτη από την αρχική θέση του επιπέδου
- Esc → Σταμάτημα του παιχνιδιού (pause game)

Υλοποίηση Παιχνιδιού

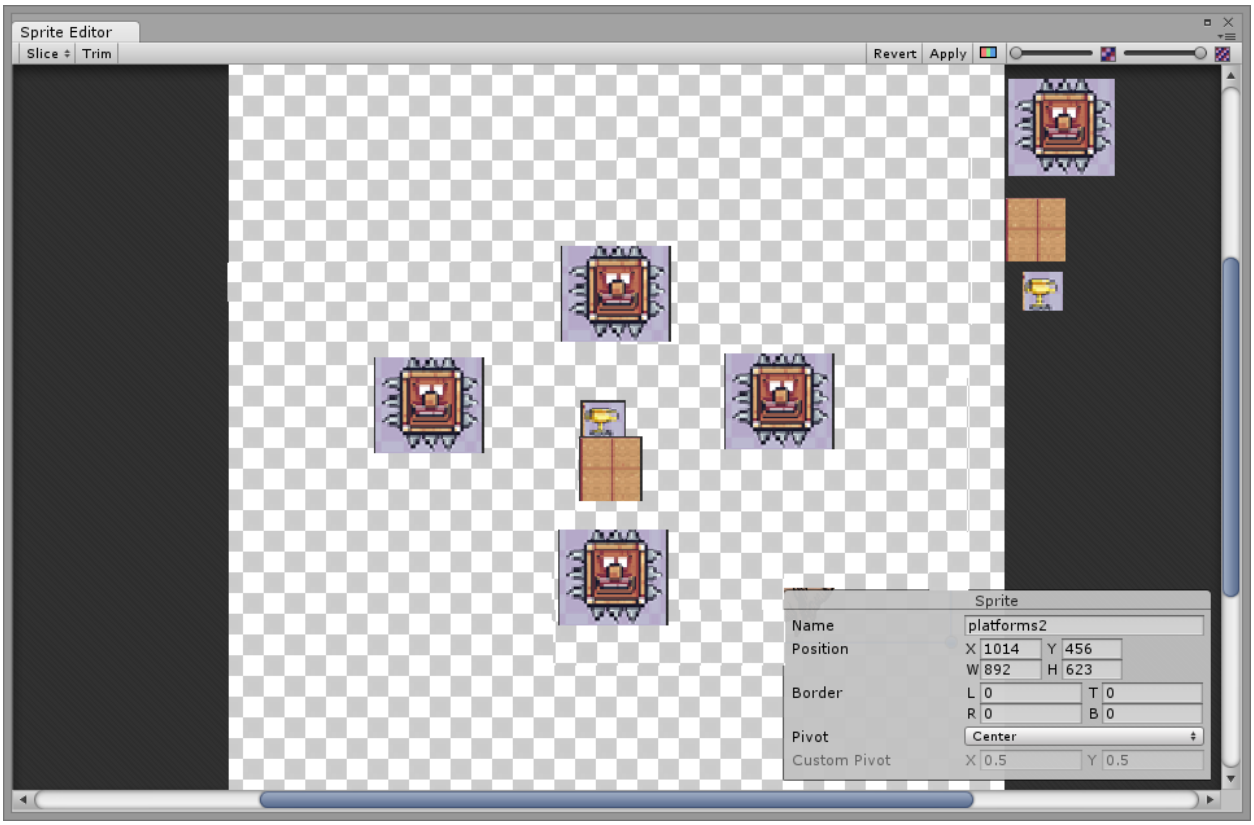
Δημιουργία Background

Το Unity έχει ένα πολύ ευνοϊκό περιβάλλον για την δημιουργία 2D backgrounds καθώς μπορούμε να παίξουμε και να πειραματιστούμε με τα assets που σου δίνει και να τα συνδυάσουμε με όποιον τρόπο εμείς θέλουμε ώστε να πάρει μια τελική μορφή.



Εικόνα 8: Κενός editor του Unity

Αυτός είναι ο editor του Unity μέσα στον οποίο μπορούμε να δημιουργήσουμε το γραφικό περιβάλλον καθώς και το background του παιχνιδιού.



Εικόνα 9: Μια πρώτη διάταξη sprites για τη δημιουργία ενός background του παιχνιδιού.

Ο σχεδιασμός ενός 2D background είναι σχετικά εύκολος όταν αυτός περιορίζεται σε ένα συγκεκριμένο χώρο. Βέβαια μπορεί να γίνει πολύ μεγαλύτερος ανάλογα με τις προδιάγραφες του κάθε παιχνιδιού. Στη συγκεκριμένη εργασία για την εξοικονόμηση χρόνο έγινε σε μικρή κλίμακα Ο σχεδιασμός έγινε με βάση τη φαντασία και το πως θα δυσκολέψουμε τον χρήστη στην επίτευξη του στόχου. Με απλές drag and drop πράξεις δημιουργήθηκαν τα διαφορά επίπεδα του παιχνιδιού τοποθετώντας spites παγίδων και spites πατώματος σε θέσεις οι οποίες δίνουν ένα ολοκληρωμένο σύνολο οπού ο χρήστης μπορεί να παίξει χωρίς πρόβλημα. Σε άλλη περίπτωση θα μπορούσαμε να προσθέσουμε ακόμα περισσότερα αντικείμενα και εφέ στο background ωστόσο περιοριστήκαμε στα απολύτως βασικά.

Παρακάτω θα αναλύσουμε τον τρόπο υλοποίησης που ακολουθήσαμε για να υλοποιήσουμε το παιχνίδι, θα παρουσιάσουμε τον κώδικα που γράψαμε αλλά και τον κώδικα που βρήκαμε και τροποποιήσαμε, καθώς και θα μελετήσουμε τις λειτουργίες του παιχνιδιού αναλυτικά, συγκεκριμένα θα αναλύσουμε τα παρακάτω:

- Γραφικά Assets που χρησιμοποιήσαμε
- Assets ήχου που χρησιμοποιήσαμε
- Λειτουργία χειρισμού του παίκτη
- Λειτουργία φυσικής του παίκτη
- Λειτουργία φυσικής των αντικειμένων στα επίπεδα
- Λειτουργία αρχικοποίησης (Bootstrapper)
- Αντικείμενα αρχικοποίησης (Bootstrappables)
- Λειτουργία φόρτωσης σκηνών (LoadingManager)
- Λειτουργία αποθήκευσης και επαναφοράς (SaveManager)
- Λειτουργία φυσικής σκανδάλης (TriggerAction)
- Λειτουργία παγίδων μέσω επέκτασης της φυσικής σκανδάλης
- Λειτουργία τερματισμού μέσω επέκτασης της φυσικής σκανδάλης
- Σχεδιασμός επιπέδων

Για το παιχνίδι χρησιμοποιήσαμε μονάχα τρία πακέτα τρίτων. Το πρώτο, είναι το Pixel Adventure 1. Το Pixel Adventure 1 περιέχει τα γραφικά που χρησιμοποιήσαμε τόσο για τον παίκτη, για την παγίδα, για τον τερματισμό αλλά και για τα γραφικά της πίστας.

Εικόνα 10: Το γραφικό που χρησιμοποιήθηκε για το idle animation του παίκτη.

Το γραφικό του παίκτη ζωγραφίζεται στην οθόνη του χρήστη με την βοήθεια του component Sprite Renderer της Unity. Το Sprite Renderer είναι ένα από τα βασικά στοιχεία της Unity που χρησιμοποιείται για την απεικόνιση 2D γραφικών στα παιχνίδια και τις εφαρμογές τους. Το Sprite Renderer είναι υπεύθυνο για την απεικόνιση των 2D σπράιτ (sprite) στη σκηνή του παιχνιδιού. Ένα σπράιτ αντιπροσωπεύει μια 2D εικόνα ή αντικείμενο που μπορεί να κινείται και να αλλάζει κατάσταση. Κάθε Sprite Renderer είναι ένα στοιχείο της σκηνής (scene) του παιχνιδιού και

συνήθως προσαρτάται σε ένα κεντρικό αντικείμενο (game object) όπως έναν χαρακτήρα ή ένα αντικείμενο στην περιβάλλουσα περιοχή του παιχνιδιού.

Το Sprite Renderer component έχει διάφορες επιλογές τροποποίησης. Συγκεκριμένα:

- **Sprite**: Ορίζει το σπράιτ που θα απεικονιστεί. Μπορεί να επιλεγεί από μια λίστα με τα διαθέσιμα σπράιτ του παιχνιδιού ή να γίνει drag and drop το επιθυμητό σπράιτ.

- **Color**: Ορίζει τον χρωματικό φίλτρο του σπράιτ , επιτρέποντας την αλλαγή του χρώματος και της διαφάνειας του σπράιτ .

- **Sorting Layer**: Ορίζει το επίπεδο της ταξινόμησης (sorting layer) στο οποίο ανήκει το σπράιτ . Αυτό επηρεάζει τη σειρά εμφάνισης των σπράιτ στη σκηνή.

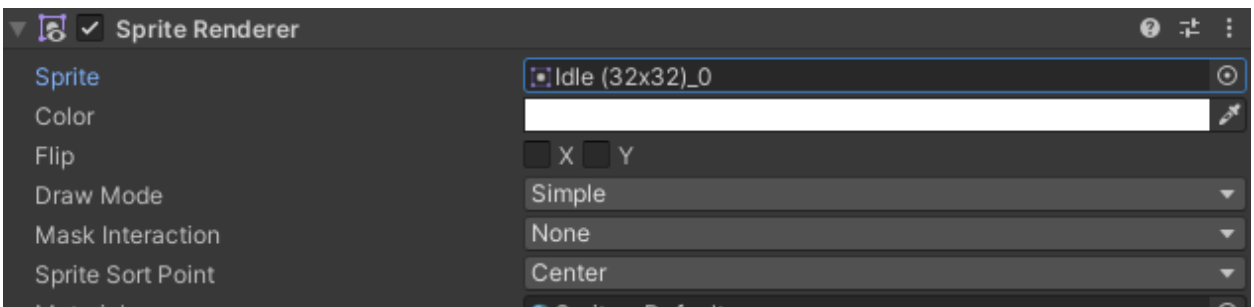
- **Order in Layer**: Καθορίζει τη σειρά εμφάνισης του σπράιτ εντός του sorting layer. Τα σπράιτ με μεγαλύτερη τιμή εμφανίζονται πάνω από τα σπράιτ με μικρότερη τιμή.

- **Flip**: Προσφέρει τη δυνατότητα αναστροφής του σπράιτ κατά μήκος του οριζόντιου (X) ή κατακόρυφου (Y) άξονα.

- **Size**: Ορίζει το μέγεθος του σπράιτ στη σκηνή.

- **Material**: Επιτρέπει την ανάθεση ενός υλικού (material) στο σπράιτ για προηγμένες γραφικές εφέ και αλλαγές στην υφή.

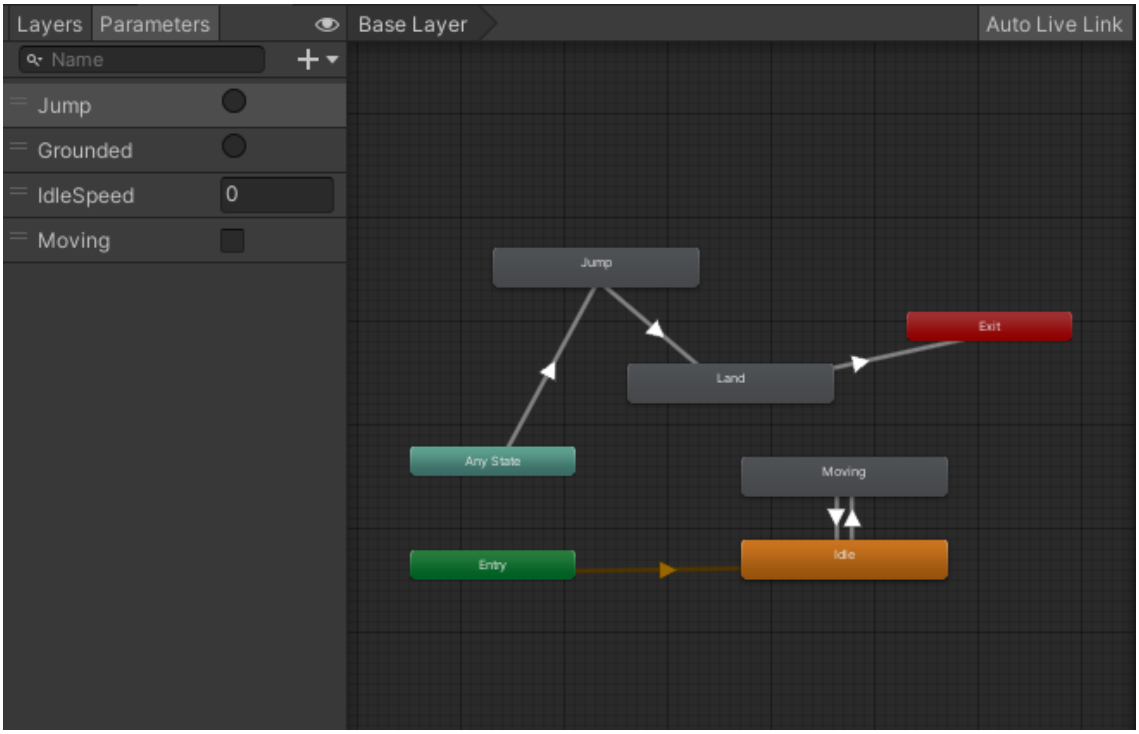
Όταν ορίζεται ένα Sprite Renderer σε ένα αντικείμενο στη σκηνή, αντιστοιχίζεται ένα σπράιτ (εικόνα) στο Sprite πεδίο. Ο Sprite Renderer θα πάρει αυτό το σπράιτ και θα το απεικονίσει στη σκηνή σύμφωνα με τις ρυθμίσεις χρώματος, μεγέθους και θέσης που έχουν οριστεί. Το sorting layer και η τιμή της ταξινόμησης (order in layer) καθορίζουν τη σειρά εμφάνισης του σπράιτ στη σκηνή, σε σχέση με τα υπόλοιπα δισδιάστατα γραφικά (τόσο σπράιτς όσο και γραφικά αλληλεπίδρασης του χρήστη (User Interface). Συνολικά, το Sprite Renderer είναι ένα απαραίτητο στοιχείο για την απεικόνιση 2D γραφικών στη Unity και προσφέρει ευελιξία στη ρύθμιση της εμφάνισης των σπράιτ στο παιχνίδι.



Εικόνα 11: Το Sprite Renderer component πάνω στο game object “Sprite”, που ανήκει στον παίκτη.

Αξίζει να σημειωθεί, ότι αναλόγως του project και των ρυθμίσεών του, και συγκεκριμένα το Rendering Pipeline που χρησιμοποιεί, ο τρόπος που ζωγραφίζεται η οθόνη του παίκτη μπορεί να αλλάξει σημαντικά. Συγκεκριμένα, πλέον υπάρχουν τρεις διαφορετικές “κατηγορίες” project: Built-In Rendering Pipeline, που είναι ο τύπος project αρχικά αλλά μπορεί να αλλάξει είτε στην αρχή είτε και αργότερα εάν φανεί ανάγκη. Universal Rendering Pipeline, που είναι ο τύπος project που έχει αρχίσει ο κόσμος να προτιμά για τα project του λόγω διάφορων βελτιστοποιήσεων που θα δούμε παρακάτω. Τέλος, το High Definition Rendering Pipeline, το οποίο χρησιμοποιείται λιγότερο σε παιχνίδια, και τα περισσότερα από αυτά είναι παιχνίδια AAA, με μεγάλες ομάδες και πολύ υψηλά μπάτζετ. Όπως έχει προαναφερθεί, το δικό μας project χρησιμοποιεί το Built-In Rendering Pipeline.

Ο παίκτης έχει κάποια απλά animations τα οποία βοηθούν στο να ζωντανέψει ελαφρώς ο δισδιάστατος χαρακτήρας μας.



Εικόνα 12: Ο Animator Controller του παίκτη.

Χρησιμοποιώντας τις παραμέτρους του Animator Controller του παίκτη, και με τις τροποποιημένες τιμές στις μεταβάσεις μεταξύ των Animation States, και σε συνδυασμό με μερικά απλά ηχητικά εφέ, ο παίκτης φαντάζει ζωντανός μέσα στην μία οθόνη που παίζει.

Το Animator component στη Unity είναι ένα κρίσιμο εργαλείο για τη δημιουργία και διαχείριση animations στα παιχνίδια. Χρησιμεύει για την δημιουργία πολύπλοκων και δυναμικών κινήσεων για τους χαρακτήρες, τα αντικείμενα και άλλα στοιχεία του παιχνιδιού. Οι κύριοι λόγοι χρήσης του Animator component είναι οι εξής:

1. **Δημιουργία κινήσεων χαρακτήρων:** Μπορούν να δημιουργηθούν animations για τους χαρακτήρες, συμπεριλαμβανομένων των περιπατητικών κινήσεων, τρεξιμάτων, αλμάτων, επιθέσεων και πολλών άλλων.
2. **Διαχείριση καταστάσεων:** Ο Animator επιτρέπει τη διαχείριση των διάφορων καταστάσεων (states) των χαρακτήρων και αντικειμένων. Κάθε κατάσταση μπορεί να αντιστοιχεί σε μια διαφορετική κινητική ενέργεια ή συμπεριφορά.
3. **Μεταβάσεις μεταξύ καταστάσεων:** Μπορείτε να καθορίσετε τις συνθήκες υποκείμενες στις οποίες θα πραγματοποιούνται αυτόματες μεταβάσεις από μια κατάσταση σε μια άλλη. Για παράδειγμα, ένας χαρακτήρας μπορεί να μεταβαίνει από την κατάσταση "περπατά" στην κατάσταση "τρέχει" όταν πατηθεί το κουμπί για τρέξιμο.
4. **Blending και transitions:** Ο Animator επιτρέπει τη λεία blending (ανάμιξη) μεταξύ διαφορετικών animations, προσφέροντας μια φυσική ροή κινήσεων. Επίσης, γίνεται να καθοριστεί ο χρόνος μετάβασης από μια κατάσταση σε μια άλλη.
5. **Εκδηλώσεις (events) και κλήσεις κωδικού:** Γίνεται να προστεθούν εκδηλώσεις σε συγκεκριμένες χρονικές στιγμές ενός animation και να καλέσετε συναρτήσεις κώδικα όταν αυτές τις στιγμές επιτευχθούν. Αυτό επιτρέπει τη συγχρονισμένη λειτουργία των animations με τη λογική του παιχνιδιού.

Το Animator component είναι καθοριστικό για τη δημιουργία παιχνιδιών με δυναμικές κινήσεις και εφέ, επιτρέποντας στους προγραμματιστές να δώσουν ζωή στον χαρακτήρα και τον κόσμο τους.

Φυσικά, ο Animator δεν δουλεύει από μόνος του, πρέπει να λαμβάνει εντολές και αλλαγές στις μεταβλητές του μέσω κώδικα. Στο παιχνίδι μας, αυτό γίνεται μέσω του script PlayerAnimator.

Το PlayerAnimator script αναλαμβάνει τα πάντα σε σχέση με το animation του παίκτη. Διαβάζοντας το input του παίκτη, και την κατάστασή του κάθε καρέ, ανανεώνει όλες τις τιμές του Animator του παίκτη προκειμένου να ταιριάζει η συμπεριφορά του παίκτη με την κατάστασή του. Επιπλέον, παρότι δεν είναι ιδανικό, το script επηρεάζει ελαφρώς και το rotation του σπράιτ του παίκτη. Αυτό γίνεται για να φαίνεται καλύτερα όταν ο παίκτης κινείται και όταν δεν κινείται. Η δομή των αντικειμένων που αποτελούν τον παίκτη συνολικά, επιτρέπει την μεταχείριση του γραφικού του παίκτη ξεχωριστά από τον παίκτη, καθιστώντας την μικρή τροποποίηση στο rotation του ασφαλή από την δημιουργία προβλημάτων κατά την κίνηση του παίκτη.

```
public class PlayerAnimator : MonoBehaviour {
    [SerializeField] private Animator _anim;
    [SerializeField] private AudioSource _source;
    [SerializeField] private LayerMask _groundMask;
    [SerializeField] private ParticleSystem _jumpParticles, _launchParticles;
    [SerializeField] private ParticleSystem _moveParticles, _landParticles;
    [SerializeField] private AudioClip[] _footsteps;
    [SerializeField] private float _maxTilt = .1f;
    [SerializeField] private float _tiltSpeed = 1;
    [SerializeField, Range(1f, 3f)] private float _maxIdleSpeed = 2;
    [SerializeField] private float _maxParticleFallSpeed = -40;

    private IPlayerController _player;
    private bool _playerGrounded;
    private ParticleSystem.MinMaxGradient _currentGradient;
    private Vector2 _movement;

    void Awake() => _player = GetComponentInParent<IPlayerController>();

    void Update() {
        if (_player == null) return;

        if (_player.Input.X != 0)
        {
            transform.localScale = new Vector3(_player.Input.X > 0 ? 1 : -1, 1, 1);
            _anim.SetBool("Moving", true);
        }
        else _anim.SetBool("Moving", false);

        var targetRotVector = new Vector3(0, 0, Mathf.Lerp(-_maxTilt, _maxTilt, Mathf.InverseLerp(-1, 1, _player.Input.X)));
        _anim.transform.rotation = Quaternion.RotateTowards(_anim.transform.rotation, Quaternion.Euler(targetRotVector), _tiltSpeed * Time.deltaTime);

        _anim.SetFloat(IdleSpeedKey, Mathf.Lerp(1, _maxIdleSpeed, Mathf.Abs(_player.Input.X)));

        if (_player.LandingThisFrame) {
            _anim.SetTrigger(GroundedKey);
            _source.PlayOneShot(_footsteps[Random.Range(0, _footsteps.Length)]);
        }

        if (_player.JumpingThisFrame) {
            _anim.SetTrigger(JumpKey);
            _anim.ResetTrigger(GroundedKey);
            if (_player.Grounded) {
                SetColor(_jumpParticles);
                SetColor(_launchParticles);
                _jumpParticles.Play();
            }
        }

        if (!_playerGrounded && _player.Grounded) {
            _playerGrounded = true;
            _moveParticles.Play();
            _landParticles.transform.localScale = Vector3.one * Mathf.InverseLerp(0, _maxParticleFallSpeed, _movement.y);
            SetColor(_landParticles);
            _landParticles.Play();
        }
        else if (_playerGrounded && !_player.Grounded) {
            _playerGrounded = false;
            _moveParticles.Stop();
        }

        var groundHit = Physics2D.Raycast(transform.position, Vector3.down, 2, _groundMask);
        if (groundHit && groundHit.transform.TryGetComponent(out SpriteRenderer r)) {
            _currentGradient = new ParticleSystem.MinMaxGradient(r.color * 0.9f, r.color * 1.2f);
            SetColor(_moveParticles);
        }
```

Εικόνα 13: Κομμάτι του script PlayerAnimator.

Ο κώδικας του script PlayerAnimator.

```
public class PlayerAnimator : MonoBehaviour {

        [SerializeField] private Animator _anim;

        [SerializeField] private AudioSource _source;

        [SerializeField] private LayerMask _groundMask;

        [SerializeField] private ParticleSystem _jumpParticles, _launchParticles;

        [SerializeField] private ParticleSystem _moveParticles,_launchParticles;

        [SerializeField] private AudioClip[] _footsteps;

        [SerializeField] private float _maxTilt = .if;

        [SerializeField] private float _tiltSpeed = 1;

        [SerializeField, Range(1f, 3f) private float _maxIdSpeed = 2;

        [SerializeField] private float _maxParticleFallSpeed = -40;


        private IPlayerController _player;

        private bool _playerGrounded;

        private ParticleSystem.MinMaxGradient _currentGradient;

        private Vector _movement;


        void Awake() => _player = GetComponentInParent<IPlayerController>();
```

```
void Update() {
        if (_player == null) return;

        if (_player.Input.X != 0)
        {
            transform.localScale = new Vector3(_player.Input.X > 0 ? 1 : -1, 1, 1);
            _anim.SetBool("Moving", true);
        }
        else _anim.SetBool("Moving", false);

        var targetRotVector = new Vector3(0, 0, Mathf.Lerp(-_maxTilt, _maxTilt, Mathf.InverseLerp(-1, 1, _player.Input.X)));
    _anim.transform.rotation = Quaternion.RotateTowards(_anim.transform.rotation, Quaternion.Euler(targetRotVector), _tiltSpeed * Time.deltaTime);

        _anime.SetFloat(IdleSpeedKey, Mathf.Lerp(1, _maxIdleSpeed, Mathf>Abs(_player.Input.X)));

        if (_player.LandingThisFrame){
            _anim.SetTrigger(GroundedKey);
            _source.PlayOneShot(_footsteps[Random.Range(0, _footsteps.Length)]);
```

```
            }


            if (_player.JumpingThisFrame){
                _animSetTrigger(JumpKey);
                _anim.ResetTrigger(GroundedKey);
            if (_player.Grounded){
                SetColor(_jumpParticles);
                SetColor(_launchParticles);
                _jumpParticles.Play();
            }
        }


        if (!_playerGrounded && _playerGrounded){
            _playerGrounded = true;
            _moveParticles.Play();
            _landParticles.transform.localScale = Vector3.one * Mathf.InverseLerp(0,
_maxParticleFallSpeed, _movement.y);
            SetColor(_landParticles);
            _landParticles.Play();
        }
```

```
        else if (_playerGrounded && !_playerGrounded){
            _playerGrounded = false;
            _moveParticles.Stop();
        }

        var groundHit = Physics2D.Raycast(transform.position,Vector3.down, 2, _groundMask);
        if(groundHit && groundHit.transfor.TryGetComponent(out SpriteRender r)){
            _currentGradient = new ParticleSystem.MinMaxGradient(r.color * 0.9f, r.color * 1.2f);
            SetColor(_moveParticles);
```

Όπως είναι προφανές, η κίνηση του παίκτη και το animation του είναι δύο διαφορετικά στοιχεία του τα οποία όμως εξαρτώνται το ένα από το άλλο. Συγκεκριμένα, το PlayerAnimator script έχει εξάρτηση στο PlayerController script.

Το PlayerController script είναι υπεύθυνο για όλες τις λειτουργίες του παίκτη. Από την κίνηση του, στην βαρύτητα του, στα άλματά του και σε όλα τα πράγματα αναμεταξύ, στην φυσική που επηρεάζει τον παίκτη συνολικά. Ο παίκτης δημιουργήθηκε με φυσική ξεχωριστή από την φυσική της Unity, και για αυτό τον λόγο ο παίκτης δεν φέρει κανενός είδος Collider στο αντικείμενό του, αλλά και κανένα Rigidbody. Η απόφαση αυτή έγινε μετά από έρευνα και πειραματισμό, καθώς έτσι είχαμε περισσότερο έλεγχο στο πως θα νιώθει ο παίκτης όταν χειρίζεται τον χαρακτήρα. Η ταχύτητα του παίκτη υπολογίζεται ακόμα κάθε καρέ, απλά διαβάζοντας την θέση του στο τρέχον καρέ, αφαιρώντας από αυτήν την θέση του στο προηγούμενο καρέ και ύστερα διαιρώντας με τον χρόνο που πέρασε από το προηγούμενο καρέ μέχρι το τρέχον, γνωστό ως delta time.

Παρόλα αυτά η ταχύτητα αυτή χρησιμοποιείται μόνο για μερικούς υπολογισμούς σχετικούς με το άλμα του (και την πτώση του αντίστοιχα). Η κίνηση του παίκτη γίνεται με την χρήση συγκεκριμένων μεταβλητών για την οριζόντια και την κάθετη ταχύτητά του. Όταν ο χρήστης επιχειρεί να κουνήσει τον χαρακτήρα, γίνεται ένας έλεγχος πρώτα για την μελλοντική θέση του παίκτη. Εφόσον δεν υπάρχουν εμπόδια, γίνεται η κίνηση. Διαφορετικά εάν υπάρχουν εμπόδια, γίνονται έλεγχοι στην νέα θέση προκειμένου να προσδιοριστεί η καινούργια θέση του παίκτη, ή η κοντινότερη θέση που μπορεί ο παίκτης να πάει, προς την νέα θέση που έχει υπολογιστεί.

Τα όρια του παίκτη, λόγω της έλλειψης Collider, υπολογίζονται με τιμές που περνάνε κατευθείαν μέσα στο script. Οι τιμές αυτές είναι σε μία μεταβλητή τύπου Bounds, που σε συνδυασμό με την θέση του παίκτη, προκύπτουν κάθε καρέ τα όρια του παίκτη. Ύστερα, κάθε καρέ γίνονται έλεγχοι για συγκρούσεις (collisions) και δράσεων σκανδάλης (trigger actions). Οι συγκρούσεις και οι δράσεις σκανδάλης έχουν κάποια κοινά (π.χ. Όρια φυσικής) αλλά και κάποιες διαφορές.

Ένα Collider είναι ένα φυσικό περίγραμμα ή σχήμα που περιβάλλει ένα αντικείμενο στην σκηνή της Unity. Χρησιμοποιείται για να ανιχνεύσει συγκρούσεις και να διαχειρίζεται τη φυσική αλληλεπίδραση μεταξύ αντικειμένων, όπως σύγκρουση και αντίσταση. Ένα Collider μπορεί να χρησιμοποιηθεί για να εμποδίσει ένα αντικείμενο να διαπεράσει άλλα αντικείμενα, όπως το έδαφος ή τοίχοι. Αντίθετα, ένα Trigger είναι ένα Collider που έχει την επιλογή "Is Trigger" ενεργοποιημένη. Όταν ένα αντικείμενο με Trigger Collider εισέρχεται στο περιβάλλον του, δεν προκαλεί συγκρούσεις ή αντίδραση φυσικής, αλλά αντίθετα εκπέμπει σήματα για να ενημερώσει το σενάριο του παιχνιδιού ότι κάτι έχει συμβεί. Τα Trigger Colliders χρησιμοποιούνται συνήθως για την ενεργοποίηση ειδικών συμβάντων, όπως η συλλογή αντικειμένων, η ενεργοποίηση πόρτας, και άλλες μη φυσικές αλληλεπιδράσεις.

Παρότι ο παίκτης δεν έχει ούτε Collider της Unity, ούτε Rigidbody, τα υπόλοιπα αντικείμενα στα επίπεδα του παιχνιδιού έχουν, απλά ο παίκτης αλληλοεπιδρά με αυτά με διαφορετικό τρόπο, έξω από την ανανέωση φυσικής (Fixed Update/Physics Update) της Unity. Στο παιχνίδι μας, Collider βρίσκονται στις πλατφόρμες, στο πάτωμα και στους τοίχους, ενώ Triggers βρίσκονται στην παγίδα, στο όριο θανάτου (εκτός πίστας), και στο κύπελλο νίκης. Προκειμένου τα Triggers να δουλεύουν σωστά με την φυσική του παίκτη, δημιουργήσαμε ένα script για να “θυμάται” την τελευταία κατάσταση του παίκτη σε σχέση με το εκάστοτε Trigger.

Το script TriggerAction, που απαιτεί την ύπαρξη ενός Collider2D στο ίδιο αντικείμενο για να δουλέψει, είναι ένα abstract class το οποίο έχει μία πολύ απλή λογική. Κατά την αρχικοποίηση στην μέθοδο Awake της Unity, διαβάζει τον Collider2D που βρίσκεται στο ίδιο αντικείμενο. Ύστερα, όταν ο παίκτης βρει εντός των ορίων του ένα TriggerAction, καλεί την μέθοδο του TriggerAction με όνομα CheckTriggerAction, περνώντας το instance του script του, PlayerController, ως παράμετρο. Μέσα στην CheckTriggerAction, ελέγχεται εάν τα όρια του παίκτη τέμνονται με τα όρια του collider. Στην συνέχεια, με την βοήθεια της μεταβλητής playerInTrigger που έχει αρχική τιμή false, γίνεται η διαπίστωση του TriggerAction που έγινε. Υπάρχουν τέσσερα πιθανά TriggerActions. Το πρώτο είναι ένα κενό TriggerAction. Αυτό συμβαίνει όταν ο παίκτης είναι αρκετά κοντά μεν σε ένα Trigger για να το εντοπίσει με το raycast που πραγματοποιεί τους ελέγχους του, αλλά όχι αρκετά κοντά για να είναι και μέσα στα όριά του με τα δικά του όρια. Το δεύτερο είναι το Enter. Αυτό γίνεται όταν ο παίκτης στο τωρινό καρέ είναι μέσα στο Trigger με τα όριά του, αλλά στο προηγούμενο καρέ δεν ήταν. Το τρίτο είναι το Stay. Αυτό συμβαίνει κάθε καρέ που ο παίκτης βρίσκεται μέσα στα όρια του trigger, και που το προηγούμενο καρέ ήταν επίσης μέσα στα όρια. Το τέταρτο και τελευταίο, είναι το Exit. Αυτό συμβαίνει όταν το τρέχον καρέ ο παίκτης είναι εκτός των ορίων του trigger, αλλά στο προηγούμενο ήταν εντός αυτών. Με εξαίρεση το TriggerEvent.None που δεν γίνεται καμία περαιτέρω κίνηση στον κώδικα, οι υπόλοιπες τρείς αλληλεπιδράσεις έχουν και τις δικές τους abstract μεθόδους, μέσα στο TriggerAction script. Αυτές καλούνται και μετά μέσω inheritance και override, μπορεί το κάθε script να κάνει ότι χρειάζεται.

Τα script SpikeHead και DeathTrigger, μέσα στο TriggerEventAction, καλούν την μέθοδο Die του παίκτη. Αυτό με την σειρά του θα εξαφανίσει το γραφικό του παίκτη, θα εμφανίσει ένα σύστημα σωματιδίων (particle system) που αναπαριστά τον θάνατο του χαρακτήρα, και αυτόματα μετά θα φορτώσει το επίπεδο από την αρχή.

Το script Trophy στο TriggerEnterAction καλεί την μέθοδο ProgressAndSave του SaveManager script, και ύστερα την μέθοδο LoadLevel του LoadingManager script, για να φορτώσει το επόμενο επίπεδο. Όταν ο παίκτης ολοκληρώσει το πέμπτο επίπεδο, συνεχίζει να το παίζει μέχρι να αποφασίσει να κλείσει το παιχνίδι.

Το script SaveManager είναι ένα Singleton το οποίο χρησιμοποιείται για να αποθηκεύει την πρόοδο του παίκτη στο παιχνίδι, χρησιμοποιώντας το class Save. Η προσθήκη παραπάνω δεδομένων για αποθήκευση είναι εύκολη, αρκεί η τροποποίηση της κλάσης Save. Η αποθήκευση γίνεται σε αρχείο json και στην περίπτωση σφάλματος κατά την αποθήκευση, γίνεται στα PlayerPrefs ως εναλλακτική.

Το script LoadingManager είναι ένα Singleton το οποίο χρησιμοποιείται για να φορτώνει το παιχνίδι διάφορες σκηνές, και να παρουσιάζει ένα Loading Screen αντί να 'κρεμάει' το παιχνίδι μέχρι να φορτώσει η επόμενη σκηνή. Επιπλέον, αυτό δίνει παραπάνω έλεγχο στην αποφόρτωση της προηγούμενης σκηνής, και βοηθά στην ομαλή μετάβαση στην νέα σκηνή.

Το LoadingManager αλλά και το SaveManager script, κάνουν inherit από το abstract class Bootstrappable το οποίο έχει μία abstract μέθοδο που επιστρέφει IEnumerator, για να μπορεί να τρέχει σε coroutine, και δεν παίρνει παραμέτρους. Επίσης έχει έναν ακέραιο αριθμό ως μεταβλητή με το όνομα BootstrapPriority, το οποίο μπορεί να δώσει προτεραιότητα σε κάποιο script να αρχικοποιηθεί από το Bootstrapper script πριν από τα άλλα (όσο υψηλότερος ο αριθμός, τόσο μεγαλύτερη η προτεραιότητα αρχικοποίησης).

Το script Bootstrapper, που βρίσκεται στην πρώτη σκηνή 'Bootstrapper' βοηθά στην αρχικοποίηση των Bootstrappable, και βεβαιώνει πως όλα τα έχουν αρχικοποιηθεί σωστά, πριν φορτώσει το Main Menu. Με την βοήθεια ενός Editor script, και το attribute 'RuntimeInitializeOnLoadMethod' καθώς και μίας στατικής μεταβλητής τύπου boolean στο Bootstrapper script, η εκκίνηση του παιχνιδιού στον Editor γίνεται πάντα μέσω της σκηνής Bootstrapper, προκειμένου να είναι ομαλή, και χωρίς να διακόπτει την ομαλή ανάπτυξη του παιχνιδιού, αναγκάζοντας τον προγραμματιστή να πηγαίνει πίσω στην σκηνή Bootstrapper πριν πατήσει το κουμπί Play.

Εκκίνηση του Παιχνιδιού

Κατά την εκκίνηση του παιχνιδιού παρουσιάζεται μπροστά στην οθόνη του χρήστη το αρχικό μενού το οποίο περιλαμβάνει τα εξής πεδία:

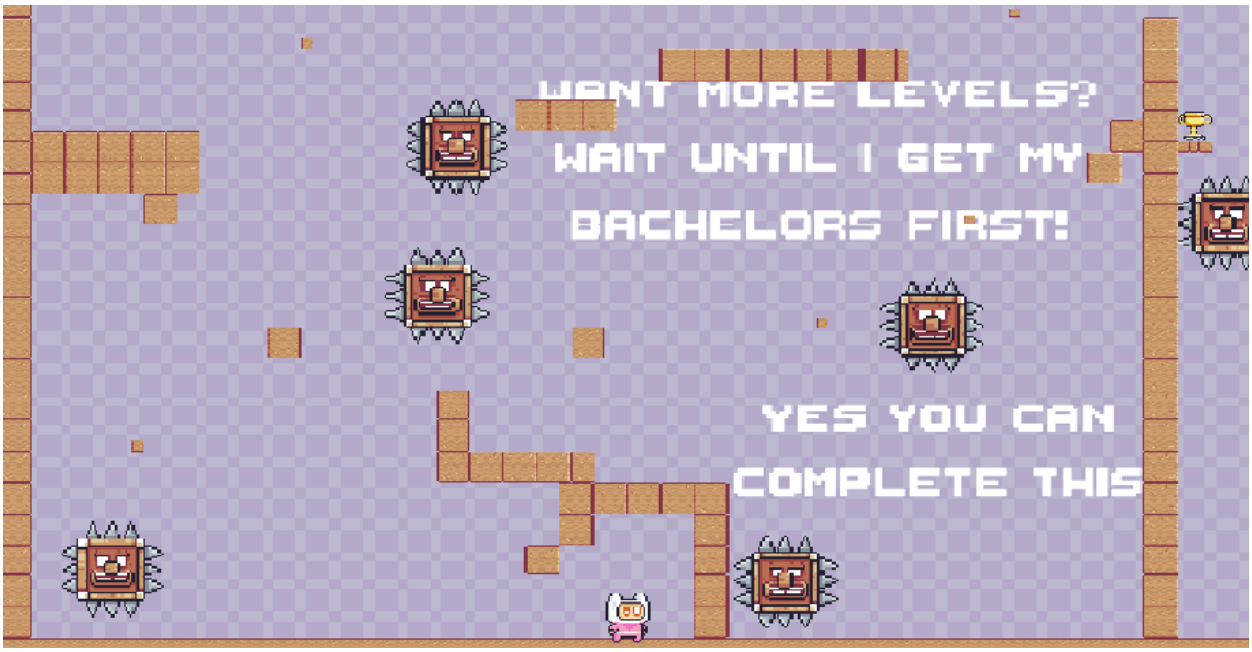
Start: Το οποίο είναι η έναρξη και του παιχνιδιού.

Settings: Οι ρυθμίσεις του παιχνιδιού.

QUIT: Έξοδος από το παιχνίδι.

Εικόνα 14: Το αρχικό μενού.

Στη συνέχεια σειρά παίρνουν τα επίπεδα του παιχνιδιού τα οποία ο χρήστης καλείται να περάσει ώστε να φτάσει στις επόμενες αλλά και στον τερματισμό του παιχνιδιού.

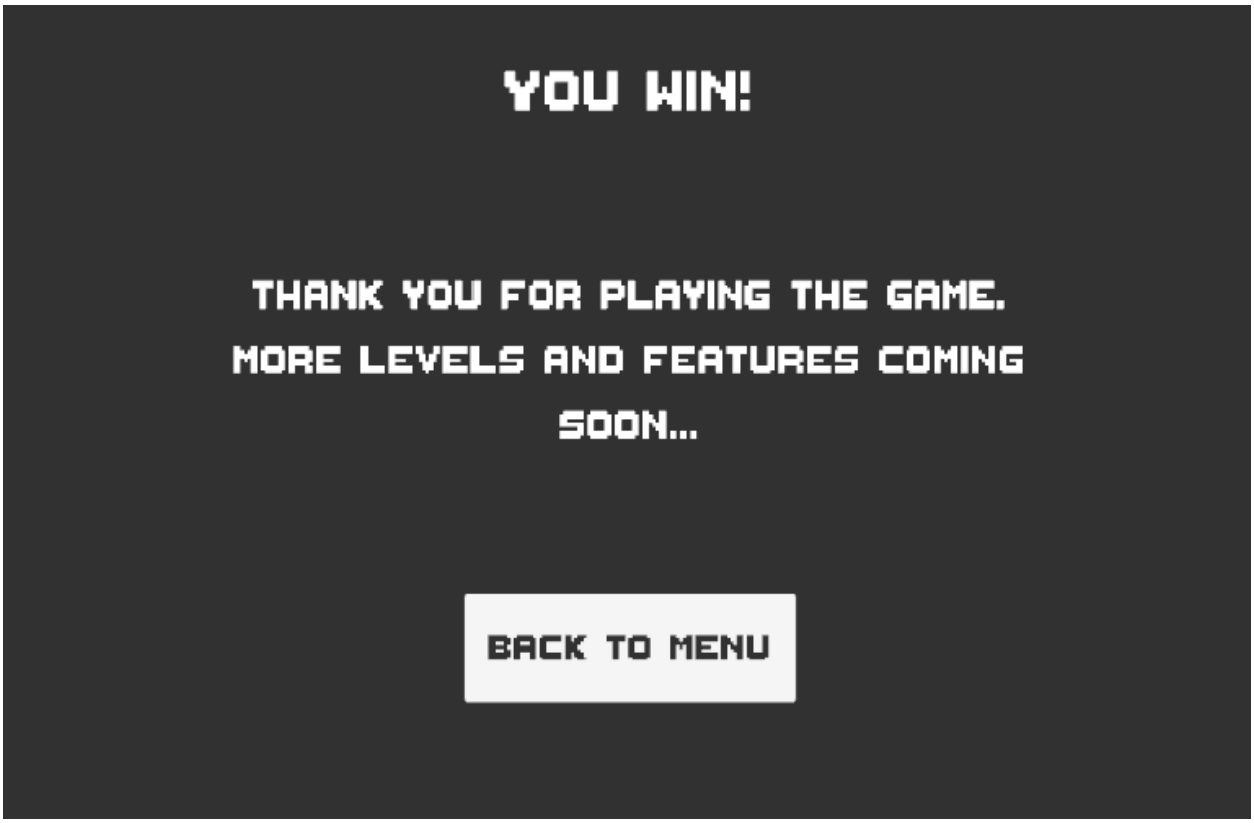


Εικόνα 15: Επίπεδο του παιχνιδιού.

Σε κάθε επίπεδο του παιχνιδιού υπάρχουν στο background και διάφορες οδηγίες ή λογοπαίγνια ώστε να βοηθήσουν ή και να πειράξουν με χιούμορ το χρήστη. Ο χρήστης μπορεί ανά πάσα στιγμή να κάνει παύση του παιχνιδιού εάν το επιθυμεί πατώντας το κουμπί "Esc".

Εικόνα 16: Μενού παύσης

Στο μενού παύσης ο χρήστης έχει τις επιλογές του να συνεχίσει το παιχνίδι ή να πάει στο αρχικό μενού. Ωστόσο ακόμα και αν αποχωρήσει εντελώς από το παιχνίδι μπορεί να όταν ξαναπαίξει να βρίσκεται στο ίδιο ακριβώς επίπεδο όπου είχε μείνει καθώς υπάρχει η δυνατότητα αυτόματης αποθήκευσης προόδου (autosave) έτσι ώστε να μην χάνεται η πρόοδος που έχει κάνει.



Εικόνα 17: Τέλος του παιχνιδιού.

Μετά την επιτυχή λήξη και του τελευταίου επιπέδου παρουσιάζεται το τέλος και η ανακοίνωση της νίκης, ένα ευχαριστήριο μήνυμα καθώς και μια υπόσχεση για προσεχή επίπεδα με μεγαλύτερη δυσκολία.

Το παιχνίδι αυτό σχεδιάστηκε για να παίζεται σε υπολογιστή (Desktop) αλλά αυτό δεν σημαίνει ότι θα λειτουργεί καλά σε όλες τις πλατφόρμες αυτόματα. Χρειάζεστε ακόμα προσοχή στον τρόπο που σχεδιάζετε το παιχνίδι σας έτσι ώστε να μπορεί να ανιχνεύει την αναλογία διαστάσεων της οθόνης και να προσαρμόζει ανάλογα τη διάταξη διεπαφής χρήστη ή να υποστηρίζει διαφορετικά παραδείγματα εισόδου μεταξύ κινητού, υπολογιστή, κονσόλας και Apple TV. Υπάρχουν επίσης ζητήματα απόδοσης που πρέπει να εξεταστούν. Εάν κατασκευάζετε το παιχνίδι σας για

υπολογιστές υψηλής ποιότητας, θα μπορούσατε να το εξαγάγετε για ένα κινητό τηλέφωνο, αλλά μην περιμένετε να έχει ρυθμούς καρέ με δυνατότητα αναπαραγωγής.

Δεν υποστηρίζονται όλες οι λειτουργίες σε όλες τις πλατφόρμες, μπορείτε να αναπαράγετε βίντεο σε textures σε υπολογιστή και κονσόλα, αλλά όχι για κινητό ή WebGL για παράδειγμα. Ειδικά με το ζήτημα του κατακερματισμού του Android, πρέπει απλώς να γνωρίζετε το εύρος των διαφορετικών αναλογιών διαστάσεων στις οποίες μπορεί να εμφανίζεται το παιχνίδι σας. Εάν η περιοχή αναπαραγωγής του παιχνιδιού σας είναι εντάξει για να έχει ελαφρώς διαφορετικές αναλογίες, τότε είναι συνήθως μια δουλειά χρήσης του UI αγκύρωση έτσι ώστε η διεπαφή χρήστη να χειρίζεται αυτόματα την αλλαγή αναλογίας διαστάσεων. Εάν το παιχνίδι σας έχει ένα σταθερό μέγεθος πεδίου παιχνιδιού (όπως ένα παιχνίδι παζλ), τότε απλώς σχεδιάζουμε για τη στενότερη πτυχή που υποστηρίζετε και δημιουργούμε τέχνη padding γύρω από τον πίνακα για μεγαλύτερες οθόνες.

Συμπέρασμα

Περνώντας από την διαδικασία ανάπτυξης του παιχνιδιού στην μηχανή παιχνιδιών Unity, μας έγινε προφανές το γεγονός ότι η μηχανή, παρά την απλότητά της σε σχέση με τις άλλες μηχανές που προαναφέραμε και συγκρίναμε πριν διαλέξουμε την Unity, εξακολουθεί να είναι αρκετά περίπλοκη όσο περισσότερο ασχολείται κανείς. Πιο συγκεκριμένα, χρειάζονται ολοένα και περισσότερες γνώσεις σε τομείς διαφορετικούς από τον προγραμματισμό, όπως για παράδειγμα το Animation, τα γραφικά και ο ήχος. Επιπλέον, οι γνώσεις μαθηματικών που χρειάζονται για ένα παιχνίδι μόλις δύο διαστάσεων, είναι σημαντικά πολλές, και είναι ξεκάθαρο ότι εάν είχαμε επιλέξει να αναπτύξουμε παιχνίδι τριών διαστάσεων αντί για δύο, τα μαθηματικά θα ήταν ακόμα πιο σύνθετα.

Όσον αφορά την έρευνα πάνω στην δημιουργία συγκεκριμένων συναισθημάτων στον παίκτη, και συγκεκριμένα το αίσθημα έλλειψης ικανοτήτων στον παίκτη κατά την αποτυχία του στο παιχνίδι, φαίνεται να είναι πιθανό, όπως έχει φανεί και σε άλλα παιχνίδια (Dark Souls, Super Mario Bros, Flappy Bird) και δεν είναι τυχαίο όταν συμβαίνει, είναι μέσω σχεδιασμού. Προκειμένου να επιτευχθεί κάτι τέτοιο, απαιτείται ξεκάθαρος σχεδιασμός στις μηχανικές του παιχνιδιού, και το αίσθημα πλήρη ελέγχου στον παίκτη.

Συμπεράναμε ότι ο τομέας ανάπτυξης παιχνιδιών είναι ένας πολύ συναρπαστικός τομέας, που θα θέλαμε στο μέλλον, εφόσον μας δοθεί η ευκαιρία, να ξαναδουλέψουμε πάνω σε αυτόν, τόσο σε δικά μας πρότζεκτ, μικρού μεγέθους αλλά και μεγαλύτερα, όσο και σε παιχνίδια με μεγαλύτερες ομάδες σε ένα επαγγελματικό επίπεδο.

Βιβλιογραφία

[1] “Unreal Engine” https://www.unrealengine.com/

[2] “Unity3d” https://unity3d.com/

[3] “CryEngine” https://www.cryengine.com/

[4] “Godot Engine” https://godotengine.org/

[5] “JMonkey” https://jmonkeyengine.org/

[6] “GameMaker” https://gamemaker.io/en

[7] “3D Ogre” https://www.ogre3d.org/

[8] Sarcar, V. (2020). Singleton Pattern. In: Design Patterns in C#. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6062-3_1

[9] P. Ljung, C. Winskog, A. Persson, C. Lundstrom and A. Ynnerman, "Full Body Virtual Autopsies using a State-of-the-art Volume Rendering Pipeline," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 869-876, Sept.-Oct. 2006, doi: 10.1109/TVCG.2006.146.

[10] Jonkers, H., Stroucken, M., Vdovjak, R., & Campus, H. T. (2006). Bootstrapping domain-specific model-driven software development within philips. In *6th OOPSLA Workshop on Domain Specific Modeling (DSM 2006)* (p. 10).