Τμήμα Πληροφορικής
Πανεπιστήμιο Δυτικής Μακεδονίας

ΑΝΑΠΤΥΞΗ ΚΑΙ ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΜΗΧΑΝΗ UNITY

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΚΟΤΣΑΡΗΣ ΠΑΝΑΓΙΩΤΗΣ

ΑΜ : 2600

ΕΙΣΗΓΗΤΗΣ ΚΑΘΗΓΗΤΗΣ : ΜΙΧΑΗΛ ΔΟΣΗΣ

ΚΑΣΤΟΡΙΑ ΙΟΥΛΙΟΣ 2024

ΑΝΑΠΤΥΞΗ ΚΑΙ ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΜΗΧΑΝΗ UNITY

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΚΟΤΣΑΡΗΣ ΠΑΝΑΓΙΩΤΗΣ

ΑΜ : 2600

ΕΙΣΗΓΗΤΗΣ ΚΑΘΗΓΗΤΗΣ : ΜΙΧΑΗΛ ΔΟΣΗΣ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Ιουλίου 2024

.....	.....	.....
Ον/μο Μέλος Ε.Π	Ον/μο Μέλος Ε.Π	Ον/μο Μέλος Ε.Π
Ιδίοτητα Μέλους Ε.Π	Ιδίοτητα Μέλους Ε.Π	Ιδίοτητα Μέλους Ε.Π

ΚΑΣΤΟΡΙΑ ΙΟΥΛΙΟΣ 2024

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον κ. Μιχαήλ Δόση, καθηγητή του Τμήματος Πληροφορικής του Πανεπιστημίου Δυτικής Μακεδονίας, για την πολύτιμη βοήθεια και καθοδήγησή του ως επιβλέπων καθηγητής αυτής της πτυχιακής εργασίας. Επιπλέον, εκφράζω τις ευχαριστίες μου στους καθηγητές του Τμήματος Πληροφορικής του Πανεπιστημίου Δυτικής Μακεδονίας, για τις γνώσεις και τις δεξιότητες που μου μετέδωσαν, και για την υποστήριξή τους καθ' όλη τη διάρκεια των σπουδών μου.Τέλος, ευχαριστώ από καρδιάς την οικογένειά μου, που μου παρείχε την ευκαιρία να ολοκληρώσω τις σπουδές μου και που στάθηκε δίπλα μου καθ' όλη τη διάρκεια αυτού του ταξιδιού.

ΠΕΡΙΛΗΨΗ

Σε αυτή τη πτυχιακή θα αναπτύξουμε και θα σχεδιάσουμε ένα παιχνίδι 3d με την πλατφόρμα της Unity Engine 3D. Τα πρώτα κεφάλαια ξεκινούν με μια ιστορική αναδρομή στην ιστορία των βιντεοπαιχνιδιών, αλλά και πως αυτά ξεκίνησαν να μπάινουν στη ζωή μας. Επίσης, θα αναλύσουμε τα βασικά προγράμματα για την δημιουργία Video Games. Έπειτα, θα μεταβούμε απ' τη θεωρία στη πράξη και θα εγκαταστήσουμε τη μηχανή Unity 3D. Αμέσως μετά, θα περάσουμε στην δημιουργία του project, όπου θα αναλύσουμε και θα περιγράψουμε με λεπτομέρεια τα βήματα και όποιες δυσκολίες για την δημιουργία ενός 3D Video Game. Τέλος, έχουμε την παρουσίαση του Jellyfish Hunter.

ABSTRACT

In this thesis, we will develop and design a 3D video game using the Unity Engine 3D. The initial chapters begin with a historical overview of the history of video games and how they started becoming a part of our lives. Additionally, we will analyze the fundamental software for creating video games. Following that, we will move from theory to practice and install the Unity 3D engine. Right after, we will proceed to create the project, where we will analyze and describe in detail the steps and any challenges involved in creating a 3D video game. Finally, we have the presentation of Jellyfish Hunter.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ

1 ΚΕΦΑΛΑΙΟ 1 VIDEO GAMES

1.1 ΟΡΙΣΜΟΣ ΗΛΕΚΤΡΟΝΙΚΟΥ ΠΑΙΧΝΙΔΙΟΥ

Οι άνθρωποι αντιδρούν διαφορετικά στο άκουσμα του όρου video game, ανάλογα με το αν έχουν ασχοληθεί με αυτά ή όχι. Τα πρώτα video games έκανα την εμφάνιση τους τέλη της δεκαετίας του ‘70. Από τότε και μέχρι σήμερα, οι νέες γενιές εκτίθονται σε αυτά όλο και περισσότερο. Αυτά τα παιχίδια λοιπόν, λειτουργούν με βάση την τεχνολογία και συνήθως ο χρήστης μπορεί να παίξει είτε σε Η/Υ, είτε σε κάποια κονσόλα. Για να παραχθεί διασκέδαση ακόμα και κάποιες φορές ανταμοιβή στον παίχτη, τα παιχνίδια χρησιμοποιούν γραφικά, ήχο, και άλλα πολυμεσικά στοιχεία. Υπάρχει μια ευρεία γκάμα ειδών, από παιχνίδια περιπέτειας και δράσης μέχρι παιχνίδια στρατηγικής, αθλητικά παιχνίδια και παιχνίδια προσομοίωσης για όλες τις ηλικιακές ομάδες [1]. Τα video games για υπολογιστή ή κονσόλες παίζονται σε οθόνη, υπολογιστή, τηλεόραση ή και σε φορητή κονσόλα. Η αλληλεπίδραση του χρήστη με το παιχνίδι γίνεται χρησιμοποιώντας διάφορες συσκευές εισόδου, όπως πληκτρολόγια, joysticks, ποντίκια, οθόνες αφής ακόμα και συστήματα VR (virtual reality) ανάλογα με τις απαιτήσεις του παιχνιδιού. Αυτές οι συσκευές εισόδου επιτρέπουν στους παίκτες να αλληλεπιδρούν με το εικονικό περιβάλλον του παιχνιδιού. Με αυτόν τον τρόπο, οι παίκτες βιώνουν το παιχνίδι και αντιδρούν στα γεγονότα του μέσα από την οθόνη. Η εξέλιξη της τεχνολογίας έχει επιτρέψει τη δημιουργία παιχνιδιών με εντυπωσιακά γραφικά, ρεαλιστικούς ήχους και συναρπαστική gameplay εμπειρία. Τα περισσότερα video games που κυκλοφορούν στην αγορά μπορούν να θεωρηθούν προσομοίωσης. Θα μπορούσαν να χωριστούν σε δύο ομάδες αναλόγως με τον ρεαλισμό τους. Στην πρώτη ομάδα κατατάσσονται τα παιχνίδια με τον μεγαλύτερο ρεαλισμό, κυρίως τα σύγχρονα παιχνίδια μαχών, αθλητισμού, πολιτισμού και προσομοίωσης επιχειρήσεων. Τέτοια παιχνίδια είναι για παράδειγμα το FIFA , το Gran Turismo 7, το UFC και διαφορά αλλά.

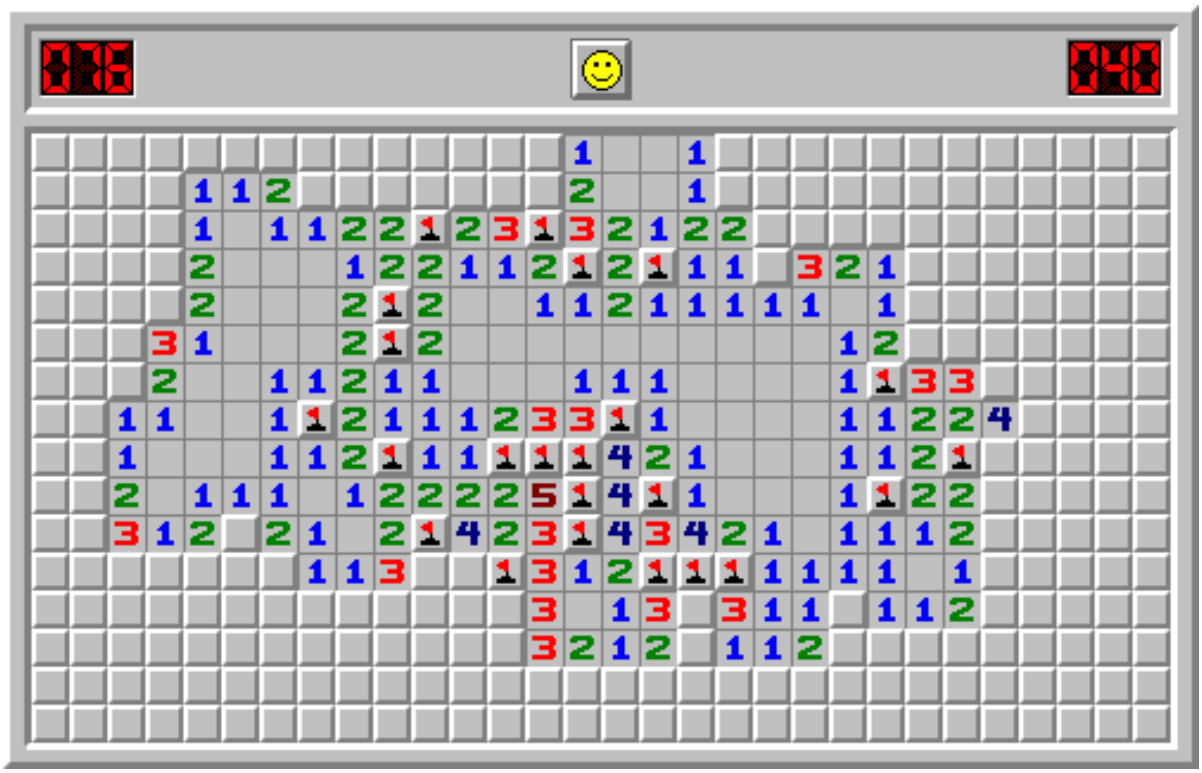
No table of figures entries found.

Εικόνα 1.1.1 : Παιχνίδι μεγάλου ρεαλισμού. Gran Turismo 7 (PS5, Polyphony Digital).

Στην δεύτερη ομάδα τοποθετούνται τα video games που παρουσιάζουν λιγότερο ρεαλισμό, παιχνίδια κατηγορίας φαντασίας, platform, περιπέτειας όπως για παράδειγμα το Sackboy: A Big Adventure ή το Stray.

Εικόνα 1.1.2 : Παιχνίδι μικρού ρεαλισμού. Sackboy: A Big Adventure (PS5/PC, Sumo Digital).

Ακόμα, υπάρχουν παιχνίδια προσομοίωσης με γρίφους (puzzle games) αλλά και πιο παραδοσιακά όπως πασιέντζα , ναρκαλιευτής και άλλα παιχνίδια τράπουλας.



Εικόνα 1.1.3 : Παιχνίδι λογικής και γρίφου. (Ναρκαλιευτής).

1.2 ΑΡΧΗ ΚΑΙ ΕΞΕΛΙΞΗ

Ταξιδεύοντας πίσω στον χρόνο, τα video games ήταν κάτι επαναστατικό και διαφορετικό. Τα παραδοσιακά παιχνίδια που υπήρχαν απ' την αρχαιότητα είχαν αρχίσει να "φθείρονται" και να εκλείπουν. Η γέννηση των πρώτων video games, πίσω στην δεκαετία του '60 ήταν η σπίθα για να ξεκινήσουν σιγά σιγά στα τέλη της δεκαετίας του '70 να κάνουν την εμφάνιση τους σε ειδικά καταστήματα (ουφάδικα) και έπειτα στα ράφια και στα σπίτια των καταναλωτών. Η ραγδαία εξέλιξη της τεχνολογίας ήταν εκείνη που βοήθησε τον κλάδο να επεκταθεί αλλά και η προβολή των ηλεκτρονικών παιχνιδιών στα Μ.Μ.Ε . Μεγάλο ρόλο έπαιξε επίσης, ο αυξανόμενος φόβος των γονέων, οι οποίοι κατέκλυζαν τους δρόμους μεγαλουπόλεων διαμαρτυρόμενοι για την ασφάλεια των παιδιών τους. Στις αρχές της δεκαετίας του

‘70, το παιχνίδι **Pong** κάνει την εμφάνιση του και είναι εκείνο που προαναγγέλλει την μεγάλη έλευση των video games. Η εποχή των Computer Control Games (παιχνίδια ελεγχόμενα απ’ τον υπολογιστή) έχει ξεκινήσει. Το Pong ελέγχεται απ΄τον υπολογιστή εξολοκλήρου χωρίς να έχει κινούμενα μέρη εκτός απ’ τα βαριά και μεγάλα στην χρήση Joysticks. Όπου και αν περπατούσε κανείς, το συναντούσε σε κάθε μέρος μιας και είχε γνωρίσει μεγάλη κοινωνική αποδοχή. Η χαμηλή του τιμή και το μικρό του μέγεθος έπαιξε μεγάλο ρόλο και κάποιες εκδόσεις του έγιναν ατομικές και μπορούσε κανείς να το προμηθευτεί για το σπίτι του [2]. Το Pong ήταν τόσο απλό και καλόγουστο σε σχέση με τα προηγούμενα arcade games που γνώρισε αυτή την μεγάλη κοινωνική αποδοχή σπάνια για την εποχή.

Εικόνα 1.2.1: Time-Out arcade (Ουφάδικο της δεκαετίας του ‘70)

Τα δεδομένα από τότε έχουν αλλάξει πάρα πολύ, μιας και οι παλιότερες γενιές που μεγάλωσαν στις γειτονιές παίζοντας κουτσό, κρυφτό και κυνηγητό θα θεωρούν την σημερινή κατάσταση εξωπραγματική. Η έννοια του παιχνιδιού έχει μεταφερθεί

απ' τους δρόμους στις κονσόλες και στους υπολογιστές. Τα video games της δεκαετίας του '70 και του '80 έχουν "σβήσει", καθώς η πρόοδος της τεχνολογίας τα τελευταία χρόνια είναι ραγδαία και αυτό έχει ως αποτέλεσμα τα νεότερα βιντεοπαιχνίδια να προσφέρουν καλύτερα γραφικά, μεγαλύτερη ποικιλία και πιο ενδιαφέρον σενάρια και πλοκές. Το 1981, περίπου 500 νέοι τίτλοι ηλεκτρονικών παιχνιδιών παρήχθησαν και το πρώτο εξάμηνο του 1982, άλλα 375 παιχνίδια έκαναν την εμφάνιση τους στην αγορά. Το 1989 μια νέα κονσόλα έκανε την εμφάνιση της. Η Sega κυκλοφόρησε το Sega Genesis, μια παιχνιδομηχανή βασισμένη στην μεγάλη χωρητικότητα μνήμης. Ο μεγάλος ανταγωνιστής της Sega εκείνη την εποχή η Nintendo, κυκλοφόρησε την φορητή κονσόλα GameBoy. Το μέγεθος του και η ευκολία στην μεταφορά του, το έκαναν ανάρπαστο κερδίζοντας έτσι μεγάλο προβάδισμα απέναντι στους ανταγωνιστές του [2]. Έτσι, ο καθένας μπορούσε να το πάρει μαζί του χωρίς να χρειαζόταν επιπλέον Joystick, οθόνη ή κάποιο καλώδιο τροφοδοσίας.

Εικόνα 1.2.2 : Οι κονσόλες Nintendo Gameboy & Sega Genesis (1989).

Το 1991, την εμφάνιση του στα ράφια της αγοράς έκανε το Sega Game Gear. Η φορητή κονσόλα της Sega πρόσθεσε έγχρωμα γραφικά, μια καινοτόμα κίνηση για την εποχή σε αντίθεση με τον ανταγωνιστή του Game Boy. Η φορητή κονσόλα της Sega δεν πούλησε τα αναμενόμενα και έτσι θεωρήθηκε ως μια εμπορική αποτυχία για την

εταιρία. Την ίδια χρόνια κυκλοφόρησε και το SuperNES της Nintendo. Βρισκόμενοι πλέον στις αρχές της δεκαετία του '90, τα σενάρια και οι χαρακτήρες των παιχνιδιών είναι πιο αληθοφανή απ' ότι στο παρελθόν. Η εμπειρία των χρηστών έχει αναβαθμιστεί καθώς παρέχονται στους παίχτες νέες και περισσότερες δυνατότητες για τον έλεγχο των χαρακτήρων τους στα video games. Στα παιχνίδια πολεμικών τεχνών για παράδειγμα (arcade), οι χρήστες έχουν την δυνατότητα να πραγματοποιήσουν περισσότερα combos και κινήσεις μιας και οι κονσόλες έχουν γίνει πιο ισχυρές. Αυτό έχει ως αποτέλεσμα ο χρήστης να αισθάνεται με την πάροδο των γενιών των παιχνιδομηχανών, ότι τα παιχνίδια γίνονται όλο και πιο ρεαλιστικά.

Εικόνα 1.2.3 : Οι κονσόλες Sega Game Gear & Nintendo SNES (1991)

Το 1995 η Sega κυκλοφόρησε το Sega Saturn, το οποίο ήταν και η πρώτη παιχνιδομηχανή που υποστήριζε τρισδιάστατα γραφικά, ενώ έχει αρχίσει πλέον η αντικατάσταση των κασετών (cartridges) με CD. Τα τρισδιάστατα γραφικά στην αρχή ήταν πολυγωνικά, αλλά για την εποχή αυτή ήταν μια σπουδαία βελτίωση όσον αφορά τα γραφικά και την απεικόνιση χαρακτήρων του παιχνιδιού, μετατρέποντας τις

κινήσεις τους πιο ρεαλιστικές συγκριτικά με τα 2d παιχνίδια που υπήρχαν παλαιότερα. Έτσι, οι χαρακτήρες άρχισαν να κινούνται σε περισσότερες κατευθύνσεις.

Εικόνα 1.2.4 : Η κονσόλα Sega Saturn (1995).

Η Nintendo, το 1996 κυκλοφορεί το Nintendo 64 και παίρνει το όνομα του από τον καινούριο επεξεργαστή 64bit που μόλις έχει εισάγει στην αγορά η εταιρία. Το λάθος της εταιρίας όμως να συνεχίσει με κασέτα (cartridge) δεν έδωσε περιθώρια στους developers να αξιοποιήσουν τις δυνατότητες της κονσόλας. Ήταν η τελευταία κονσόλα της Nintendo που λειτουργούσε με κασέτες.

Εικόνα 1.2.5 : Η κονσόλα Nintendo 64 (1996).

Το 1995 η Sony κυκλοφόρησε το PlayStation. Η πιο εξελιγμένη κονσόλα της εποχής είναι γεγονός , καθώς η Sony κατάφερε να βάλει την παιχνιδομηχανή σε κάθε σαλόνι. Ο μεγαλύτερος ρεαλισμός στα 3d γραφικά και η φυσική κίνηση των χαρακτήρων που μοιάζει με ανθρώπινη αλλά και η τεράστια γκάμα παιχνιδιών φέρνουν το PlayStation στην κορυφή. Τρία χρόνια αργότερα, η Sega λανσάρει την νέα της κονσόλα Sega Dreamcast αλλά χωρίς ιδιαίτερη επιτυχία, καθώς το PlayStation τα έχει σαρώσει όλα.

Εικόνα 1.2.6 : Η κονσόλα Sony PlayStation (1995) & Sega Dreamcast (1998) αντίστοιχα.

Το 2007 ο επεξεργαστής PhysX βγαίνει σε κυκλοφορία. Βοήθησε τους developers στο να μπορούν εύκολα να προσομοιώνουν φυσικές ιδιότητες των αντικειμένων και τα επίπεδα ρεαλισμού των παιχνιδιών με ευκολία και μεγάλη ακρίβεια σε σύγκριση με τους υπόλοιπους επεξεργαστές. Την αγορά των video games αποτελούν τα εξής λειτουργικά συστήματα:

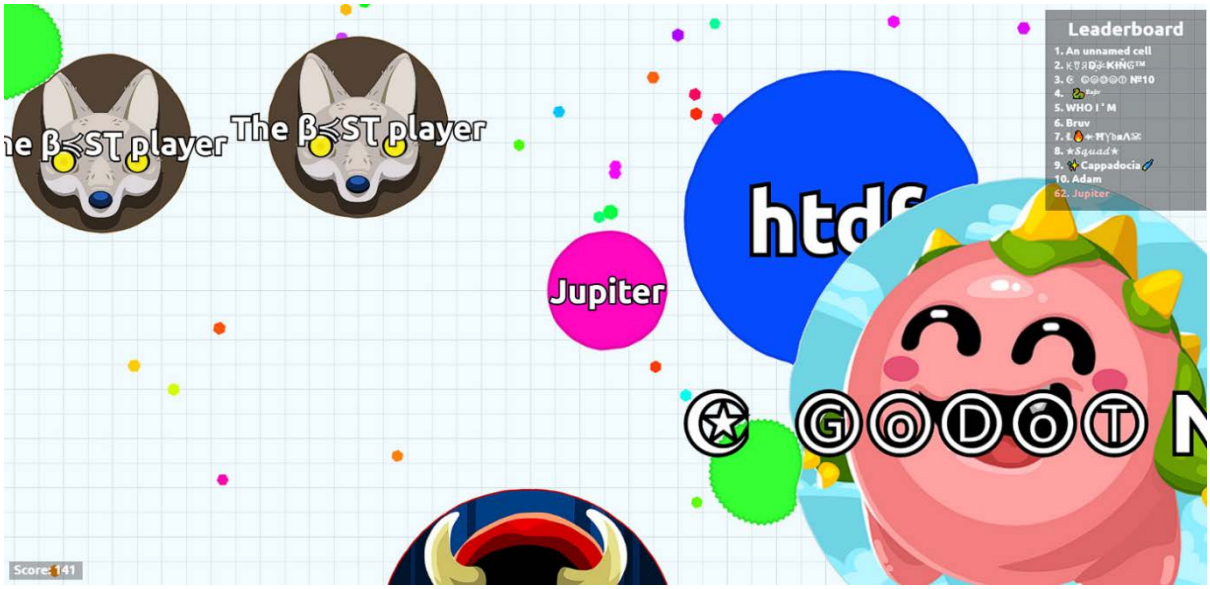
- Οικιακές κονσόλες (PlayStation, Xbox)
- Κονσόλες χειρός (PS Vita, PSP, Gameboy)
- Ηλεκτρονικοί Υπολογιστές (Desktop, Laptop)

Και παλαιότερα:

- Παιχνιδομηχανές Arcade (Bubble, Pong, Mortal Kombat)

Την τελευταία δεκαετία μεγάλη αναγνωρισιμότητα έχουν γνωρίσει τα online βιντεοπαιχνίδια τα οποία χωρίζονται στις εξής κατηγορίες:

- Online browser games: Είναι τα παιχνίδια τα οποία τις περισσότερες φορές παρέχονται δωρεάν σε ιστοσελίδες με διαφημίσεις. Συνήθως είναι αρκετά απλά παιχνίδια, απλού χαρακτήρα χωρίς κάποιο ιδιαίτερο complexity. Τέτοια παιχνίδια για παράδειγμα είναι το Agar.io, Farmerama, Zynga Poker.

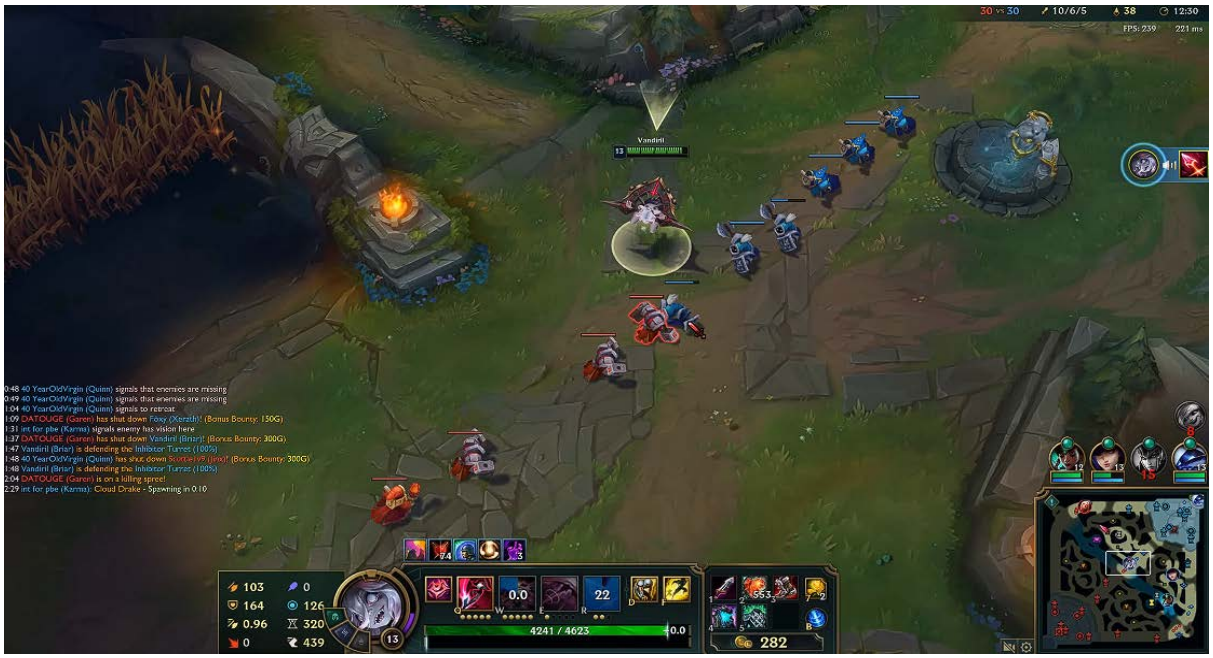


Εικόνα 1.2.7 : Online browser game Agar.io .

- Παιχνίδια με διαφημιστικό σκοπό : είναι τα video games που σχεδιάστηκαν για να προωθήσουν μία υπηρεσία ή ένα προϊόν. Για παράδειγμα, τέτοια παιχνίδια είναι το Zool (Chupa Chups), το CoolSpot (7Up).

Εικόνα 1.2.8 : Online browser game με διαφημιστικό σκοπό CoolSpot (7Up) .

- Video Game διτκύου: είναι τα παιχνίδια που ο χρήστης χρειάζεται να τα εγκαταστήσει στον ηλεκτρονικό του υπολογιστή, αλλά για να παίξει η σύνδεση στο διαδίκτυο είναι απαραίτητη. Τέτοια παιχνίδια είναι το League of Legends (Riot Games), Fortnite (Epic Games).



Εικόνα 1.2.9 : Παιχνίδι δικτύου League of Legends (Riot Games) .

- Παιχνίδια MMORPG: Είναι τα παιχνίδια τα οποία είναι παιχνίδια δικτύου, αλλά ο χρήστης μπορεί να συνεχίσει να προοδεύει με τον χαρακτήρα του ακόμα και εκτός σύνδεσης. Σε αυτά τα παιχνίδια συνήθως είναι εγγεγραμμένοι πολλές χιλιάδες παίκτες παγκοσμίως. Παράδειγμα τέτοιων παιχνιδιών είναι το Guild of Wars 2 (Arena Net), World of Warcraft (Blizzard Entertainment).

Εικόνα 1.2.10 : Παιχνίδι MMORPG World of Warcraft (Blizzard Entertainment) .

2 ΚΕΦΑΛΑΙΟ 2 : ΠΡΟΓΡΑΜΜΑΤΑ ΔΗΜΙΟΥΡΓΙΑΣ VIDEO GAME

2.1 ΠΡΟΛΟΓΟΣ

Ο δημιουργός, για να καταφέρει να υλοποιήσει ένα βιντεοπαιχνίδι ή μια εφαρμογή, χρειάζεται να χρησιμοποιήσει αρκετά εργαλεία ή πρόγραμμα αν θέλει να παρουσιάσει ένα ικανοποιητικό αποτέλεσμα. Κάποια απ' αυτά τα προγράμματα είναι το Unity για την δημιουργία του παιχνιδιού, το Blender για την δημιουργία των 3d μοντέλων και το Photoshop για την επεξεργασία των εικόνων και των textures. Αυτός είναι ο λόγος που οι μεγάλες εταιρίες παραγωγής ηλεκτρονικών βιντεοπαιχνιδιών απασχολούν πολύ κόσμο, μιας και πρέπει να καλυφθούν θέσεις όπως το τμήμα ήχου, γραφικών, προγρμματιστών, videoediting κλπ. Συνήθως αυτά τα τμήματα αποτελούνται από ομάδες ανθρώπων και ο καθένας έχει τον ρόλο του.

2.2 UNITY 3D

Είναι απ' τις πιο γνωστές μηχανές στον χώρο κατασκευής βιντεοπαιχνιδιών και μπορεί να δημιουργήσει παιχνίδια για έναν πολύ μεγάλο αριθμό πλατφόρμων. Το Unity συμπεριλαμβάνει ένα ενοποιημένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE). Μέσω της πλαρφόρμας της Unity Engine έχουν αναπτυχθέι και κυκλοφορήσει πολλά παιχνίδια για

- Ηλεκτρονικούς Υπολογιστές
- Smartphones
- Tablets
- Κονσόλες
- Online ιστότοπους

Η πλατφόρμα δημιουργίας παιχνιδιών της Unity είχε παρουσιαστεί στην συνέντευξη που είχε διωργανώσει η Apple "Worldwide Developers Conference" και λειτουργούσε μόνο σε συστήματα Mac. Στη συνέχεια η εταιρία θέλησε να επεκταθεί σε περισσότερες πλατφόρμες και τα κατάφερε. Η Unity επιτρέπει στον κάθε developer την δυνατότητα να δημοσιεύσει το παιχνίδι που έχει δημιουργήσει σε όποια πλατφόρμα εκείνος επιθυμεί. Επίσης η Unity δίνει την δυνατότητα στον developer να μετατρέπει και να συμπιέζει τις εικόνες και την ανάλυση που χρησιμοποιεί για να μπορέσει να τα κάνει πλήρως συμβατά για την πλατφόρμα που θέλει να δημισιεύσει το παιχνίδι του. Η Unity υλοποιήθηκε με βάση το "*Mono*" [3] , η οποία είναι εφαρμογή ανοιχτού κώδικα της NET Framework. Ο Developer παλαιότερα μπορούσε να γράφει τον κώδικα του σε C#, Javascript και Boo αλλά πλέον δίνεται η δυνατότητα συγγραφής κώδικα μόνο με C#.

2.3 GAMEMAKER

Μία ακόμα αναγνωρισμένη εφαρμογή δημιουργίας 2d video game για υπολογιστές και όχι μόνο είναι το Gamemaker. Ο δημιουργός κι εκείνος που το εξέδοσε είναι ο Mark Overmars, λέκτορας στο τμήμα πληροφοριών και επιστημών πληροφορικής στο πανεπιστήμιο της Ουτρέχτης. Το Gamemaker έχει σχεδιαστεί με τέτοιο τρόπο, ούτως ώστε να δίνει την δυνατότητα στον developer να δημιουργήσει ένα 2d video game μαθαίνοντας μόνο κάποιες βασικές αρχές και λειτουργίες του προγράμματος [4]. Κάπως έτσι το Gamemaker δίνει την δυνατότα και στον πιο αρχάριο χρήστη ηλεκτρονικού υπολογιστή να δημιουργήσει το δικό του video game. Η πλατφόρμα του Gamemaker έχει σχεδιαστεί για να μπορεί να δουλέψει σχεδόν σε όλα τα λειτουργικά συστήματα που κυκλοφορούν στην αγορά και πιο συγκεκριμένα απ' τα Windows 98 και έπειτα.

2.4 CRY ENGINE

Η μηχανή γραφικών της CryEngine, είναι μια προηγμένη μηχανή γραφικών που αναπτύχθηκε από την γερμανική εταιρεία Crytek. Κυκλοφόρησε το 2002 και έχει χρησιμοποιηθεί για τη δημιουργία παιχνιδιών με υψηλή ποιότητα γραφικών. Η μηχανή είναι γνωστή μέχρι και σήμερα για την εξαιρετική ποιότητα γραφικών , τα

φωτορεαλιστικά περιβάλλοντα και τις προηγμένες δυνατότητες φυσικής. Η CryEngine παρέχει εργαλεία για τη δημιουργία εκπληκτικών γραφικών, συμπεριλαμβανομένης της δυνατότητας ραγδαίας αναπαραγωγής γραφικών σε πραγματικό χρόνο. Αυτό βοηθά τους developers να δημιουργούν εντυπωσιακά περιβάλλοντα και εφέ, προσφέροντας ένα ευέλικτο περιβάλλον ανάπτυξης για τη δημιουργία παιχνιδιών. Η μηχανή υποστηρίζει διάφορες πλατφόρμες, συμπεριλαμβανομένων των Ηλεκτρονικών Υπολογιστών, κονσολών και VR συσκευών. Έχει χρησιμοποιηθεί σε πολλά γνωστά παιχνίδια, όπως τα Crysis, Far Cry, και Hunt: Showdown [5]. Η μηχανή της CryEngine προσφέρει επίσης εκτεταμένη υποστήριξη για τη δημιουργία ποικίλων ειδών παιχνιδιών, από first-person shooters μέχρι και παιχνίδια ρόλων (RPGs). Με τη συνεχή εξέλιξη της τεχνολογίας, η CryEngine συνεχίζει να είναι μια ισχυρή επιλογή για τους developers που αναζητούν υψηλής ποιότητας γραφικά και ευελιξία στην ανάπτυξη των παιχνιδιών τους.

2.5 UNREAL ENGINE

Μια άλλη μηχανή γραφικών είναι η Unreal Engine. Η ισχυρή μηχανή παιχνιδιών αναπτύχθηκε από την Epic Games. Κυκλοφόρησε για πρώτη φορά το 1998 και παρουσιάστηκε στο κοινό με το παιχνίδι Unreal το οποίο ήταν first-person και είχε κυκλοφορήσει για την πλατφόρμα των Windows. Η Unreal Engine (UE) είναι ευρέως χρησιμοποιούμενη στη βιομηχανία των βιντεοπαιχνιδιών, αλλά χρησιμοποιείται επίσης σε εφαρμογές εικονικής πραγματικότητας (VR), επαυξημένης πραγματικότητας (AR) και πολλών άλλων πεδίων. Επίσης, προσφέρει εξαιρετική ποιότητα γραφικών και φωτορεαλιστικά εφέ. Το Blueprint Visual Scripting System της UE επιτρέπει στους developers να δημιουργούν βιντεοπαιχνίδια χωρίς να γνωρίζουν την συγγραφή κώδικα, κάτι που καθιστά τη διαδικασία ανάπτυξης πιο προσιτή [5]. Η μηχανή της Unreal Engine υποστηρίζει πολλές πλατφόρμες, συμπεριλαμβανομένων των PC, κονσολών, κινητών συσκευών και VR συσκευών. Ευθύνεται για την δημιουργία δημοφιλών παιχνιδιών όπως το Fortnite, το Gears of War, το Unreal Tournament και πολλά άλλα. Ένα από τα πλεονεκτήματα της Unreal Engine είναι η κοινότητα των χρηστών της και η ποικιλία των διαθέσιμων εργαλείων. Για να

μπορέσει κάποιος να δημιουργήσει ένα βιντεοπαιχνίδι με την Unreal Engine, εκτός απο έναν πολύ ισχυρό ηλεκτρονικό υπολογιστή, θα χρειαστεί να γνωρίζει C++ .

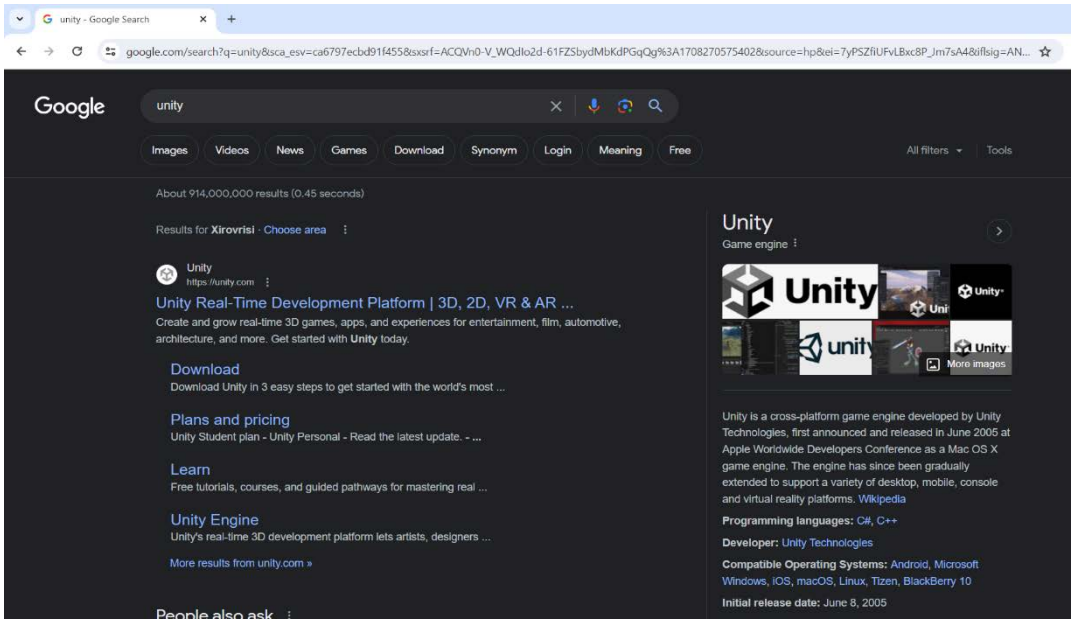
3 ΚΕΦΑΛΑΙΟ 3 : ΕΥΡΕΣΗ ΚΑΙ ΕΓΚΑΤΑΣΤΑΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ ΔΗΜΙΟΥΡΓΙΑΣ VIDEO GAME ΚΑΙ ΓΡΑΦΙΚΩΝ

3.1 ΠΡΟΛΟΓΟΣ

Η επιλογή του Unity 3D για τη δημιουργία του παιχνιδιού και του Blender για τη δημιουργία των γραφικών δεν ήταν τυχαία. Και τα δύο προγράμματα έχουν επιλεγεί λόγω της ευχρηστίας τους και της ευρείας γκάμας δυνατοτήτων που προσφέρουν. Το Unity 3D είναι μια δημοφιλής μηχανή παιχνιδιών που προσφέρει ένα ολοκληρωμένο περιβάλλον ανάπτυξης για δημιουργία παιχνιδιών σε πολλές πλατφόρμες. Είναι γνωστό για την ευκολία χρήσης του και τη δυνατότητα ανάπτυξης παιχνιδιών χωρίς την ανάγκη για πολύπλοκο προγραμματισμό, χάρη στο σύστημα του Visual Scripting. Το Blender, από την άλλη πλευρά, είναι ένα προηγμένο πρόγραμμα δημιουργίας τρισδιάστατων γραφικών και animation. Οι δύο αυτές επιλογές συνδυάζουν τη δυνατότητα δημιουργίας υψηλής ποιότητας παιχνιδιών με την ευκολία χρήσης, καθιστώντας τα ιδανικά για νέους developers που θέλουν να εξερευνήσουν τον κόσμο της ανάπτυξης παιχνιδιών και των τρισδιάστατων γραφικών. Και τα δύο εργαλεία που επιλέχθηκαν είναι δωρεάν.

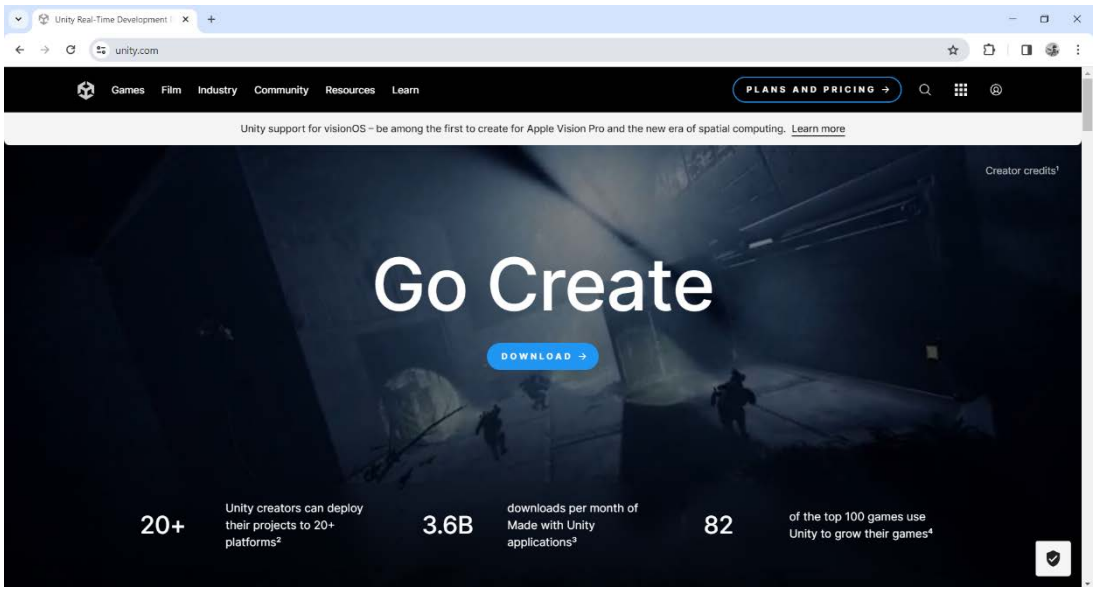
3.2 ΕΒΡΕΣΗ ΚΑΙ ΕΓΚΑΤΑΣΤΑΣΗ UNITY 3D

Για την υλοποίηση του παιχνιδιού The Jellyfish Hunter επιλέχθηκε το πρόγραμμα δημιουργίας παιχνιδιών της Unity 3d.



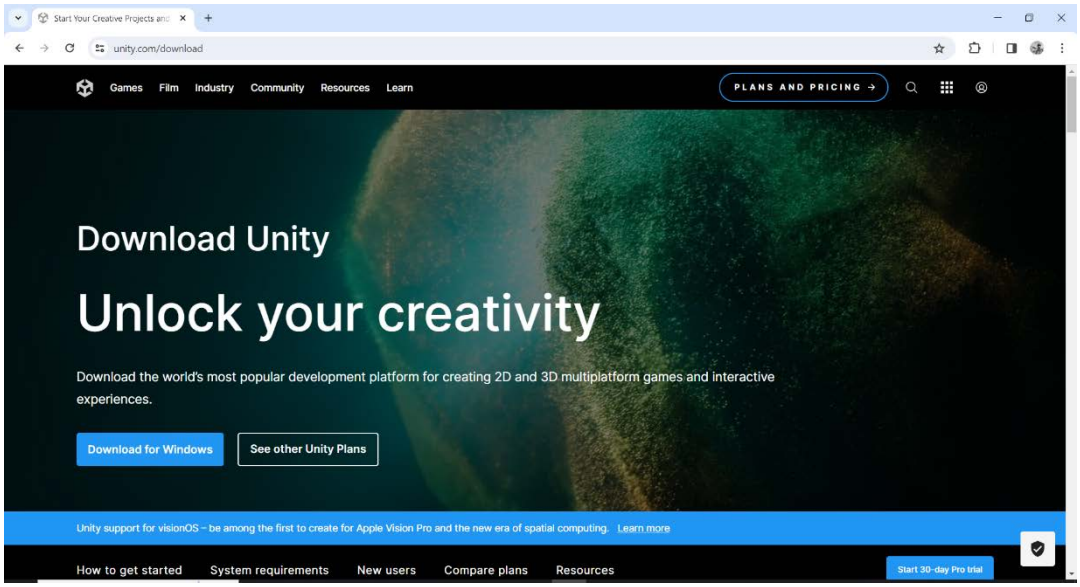
Εικόνα 3.2.1 : Εύρεση Unity στο διαδύκτιο

Μόλις ο χρήστης εισέλθει στην σελίδα της Unity, θα βρει εκεί την επιλογή download για να μπορέσει να κατεβάσει δωρεάν την έκδοση που επιθυμεί.



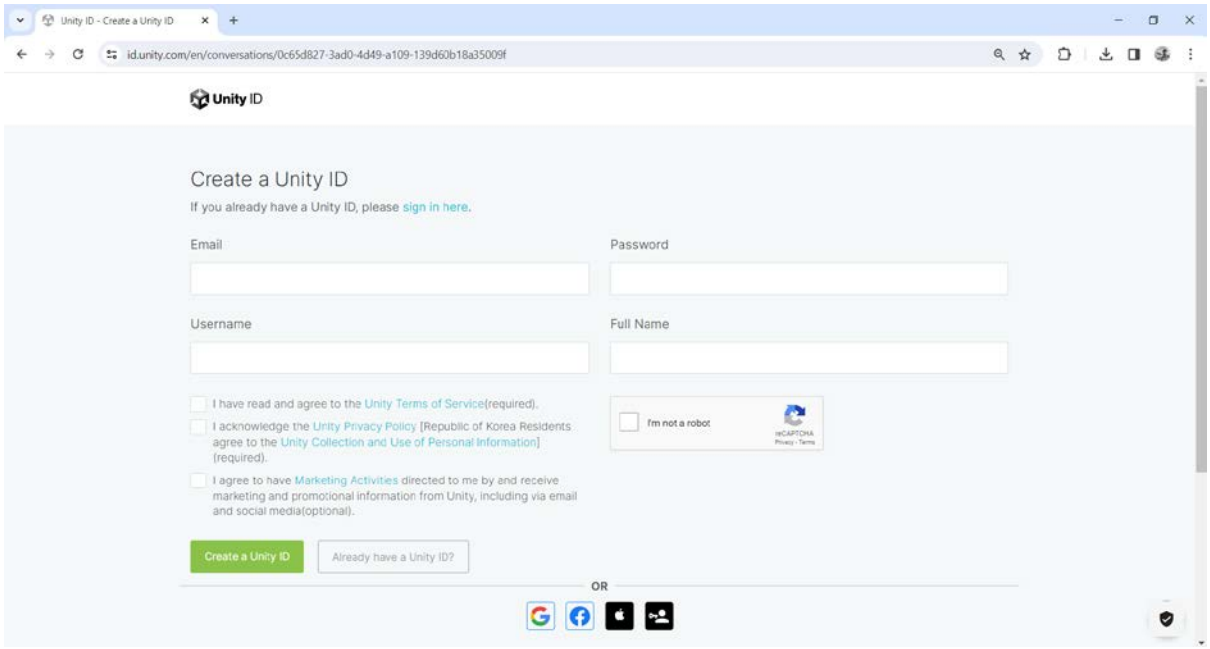
Εικόνα 3.2.2 : Εύρεση επιλογής λήψης της Unity.

Μόλις ο χρήστης κάνει κλικ στην επιλογή “download”, θα μεταφερθεί στην επόμενη καρτέλα για να κατεβάσει την δωρεάν έκδοση της μηχανής. Αν επιθυμεί μία εκ των δύο επί πληρωμής έκδοση θα χρειαστεί να δει και να επιλέξει κάποιο άλλο πλάνο.



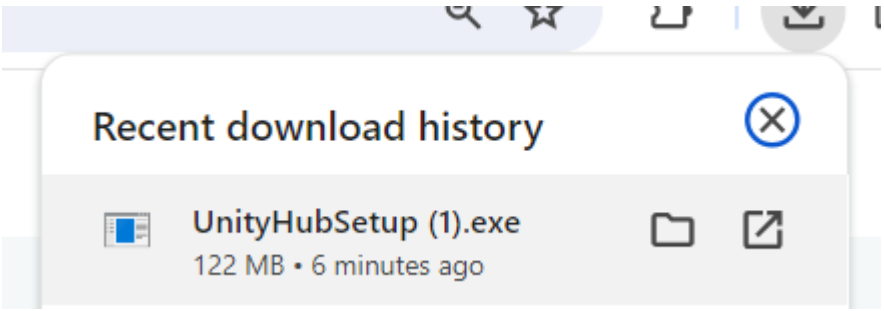
Εικόνα 3.2.3 : Λήψη δωρεάν πλάνου Unity για Windows.

Για να χρησιμοποιήσουμε το Unity θα χρειαστεί ο χρήστης να δημιουργήσει έναν λογαριασμό εντελώς δωρεάν.



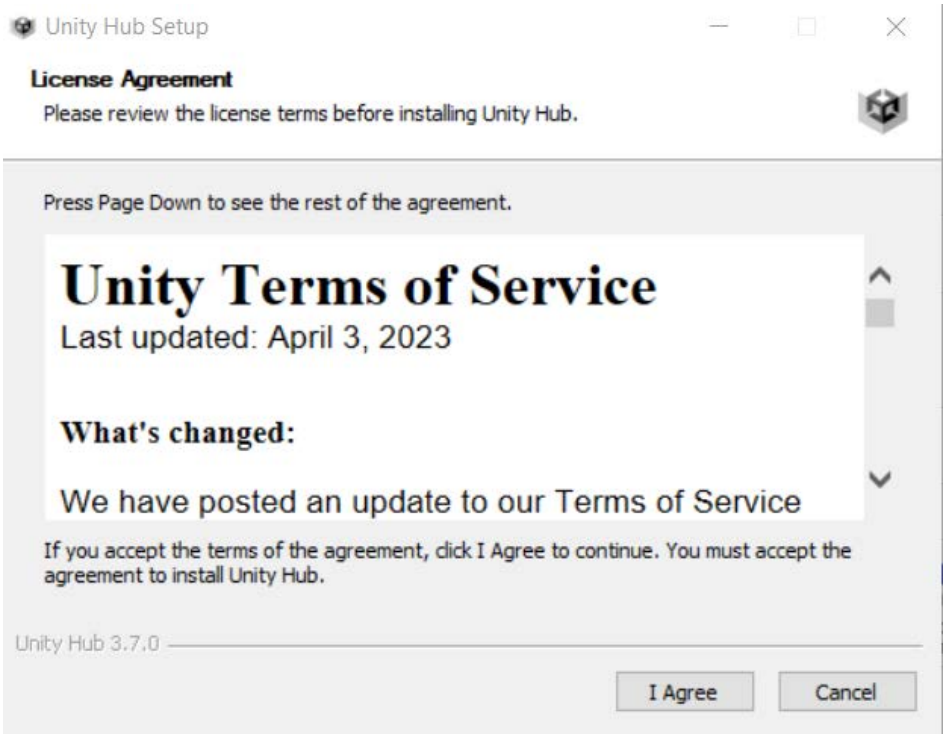
Εικόνα 3.2.4 : Δημιουργία λογαριασμού Unity.

Μόλις κατέβει το πρόγραμμα στον υπολογιστή, ξεκινάει η εγκατάσταση.



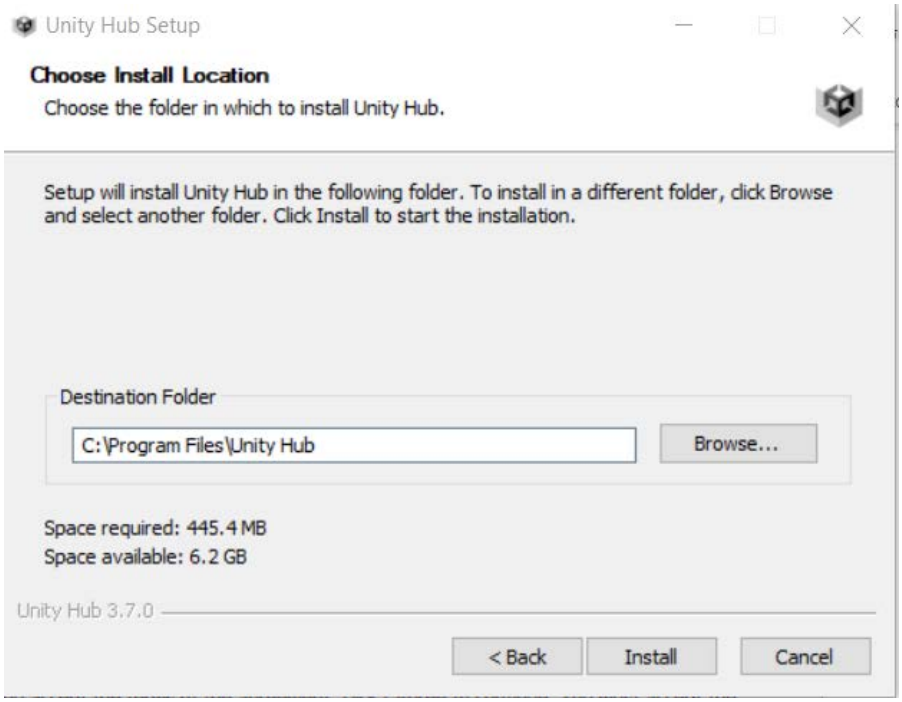
Εικόνα 3.2.5 : Αρχείο εγκατάστασης.

Στο πρώτο παράθυρο, ο χρήστης επιλέγει την επιλογή "I agree".



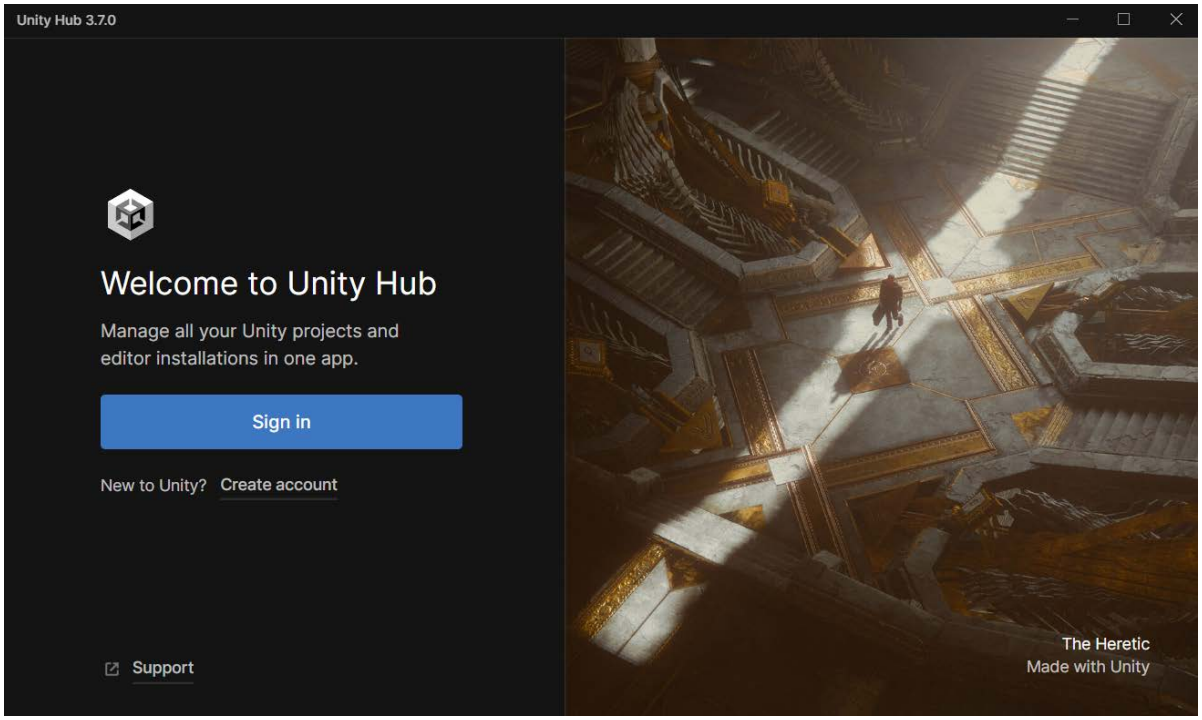
Εικόνα 3.2.6 : Εγκατάσταση Unity Hub.

Έπειτα, στην αμέσως επόμενη καρτέλα ο χρήστης θα πρέπει να επιλέξει τον φάκελο που θέλει να εγκατασταθεί το Unity Hub.



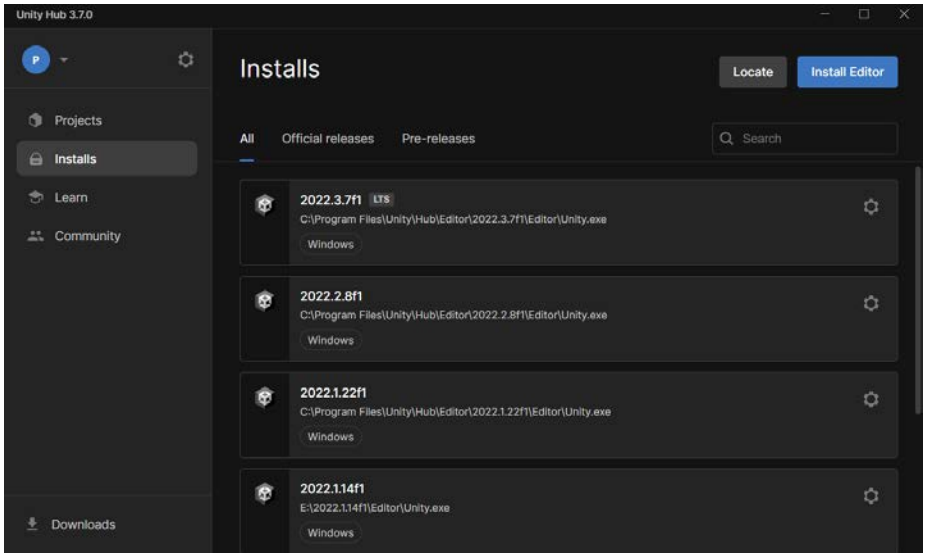
Εικόνα 3.2.7 : Επιλογή διαδρομής φακέλου αποθήκευσης.

Μόλις η εγκατάσταση έχει ολοκληρωθεί, το Unity Hub μπορεί να ανοίξει και να γίνει σύνδεση στον λογαριασμό Unity. Μόλις η σύνδεση ολοκληρωθεί, ο χρήστης θα κατεβάσει όποια έκδοση της Unity χρειάζεται.



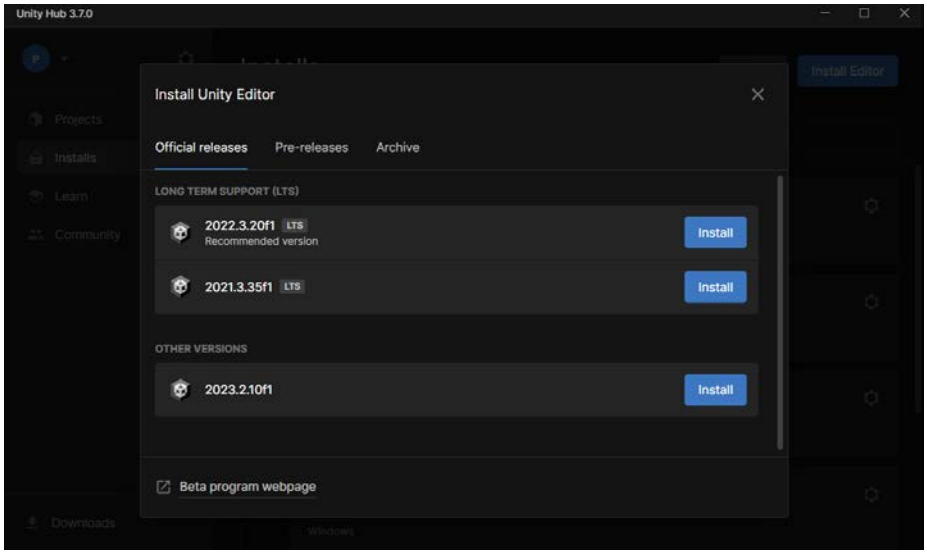
Εικόνα 3.2.8 : Σύνδεση στον λογαριασμό Unity.

Η μορφή του Unity Hub θα είναι κάπως έτσι. Για να κατεβάσει ο χρήστης την έκδοση που θέλει θα χρειαστεί να μεταβεί στον φάκελο “Installs” , ο οποίος βρίσκεται αριστερά, ακριβώς κάτω απ’ τον φάκελο Projects.



Εικόνα 3.2.9 : Η μορφή του Unity Hub στον φάκελο ”Installs”.

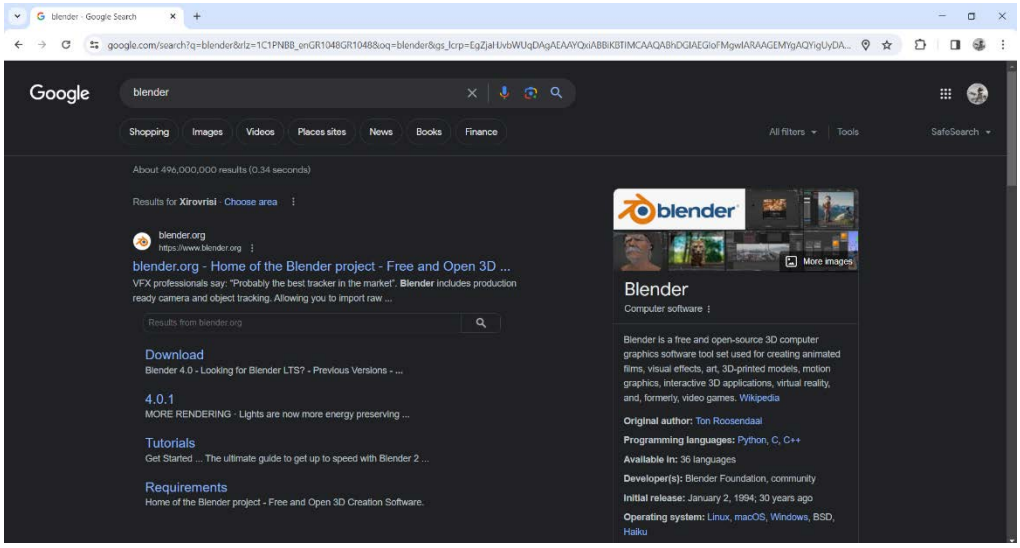
Αμέσως μετά ο χρήστης θα επιλέξει το μπλε κουμπί το οποίο αναγράφει “Install Editor” που βρίσκεται δεξιά πάνω, για να επιλέξει την έκδοση που επιθυμεί για εγκατάσταση.



Εικόνα 3.2.10 : Επιλογή έκδοσης Unity για εγκατάσταση.

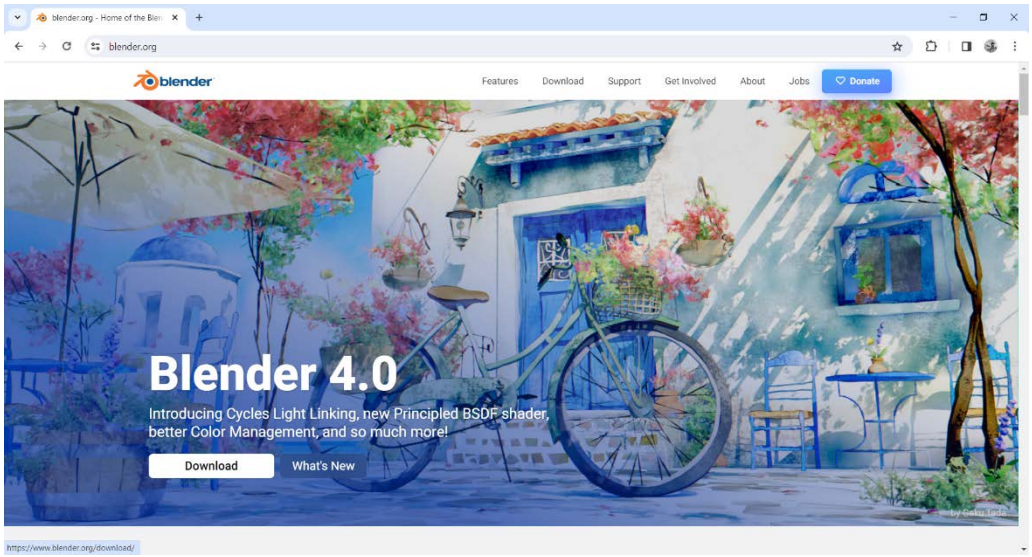
3.3 ΕΥΡΕΣΗ ΚΑΙ ΕΓΚΑΤΑΣΤΑΣΗ BLENDER

Το Blender αποτελεί έναν από τους κορυφαίους εκπροσώπους στον χώρο της δημιουργίας τρισδιάστατων μοντέλων, προσφέροντας ευρεία γκάμα λειτουργιών και εργαλείων στους χρήστες. Με μια σύντομη αναζήτηση στο διαδίκτυο, μπορεί να διαπιστώσει ο καθένας ότι το Blender είναι ίσως το καλύτερο δωρεάν πρόγραμμα δημιουργίας 3d μοντέλων online.



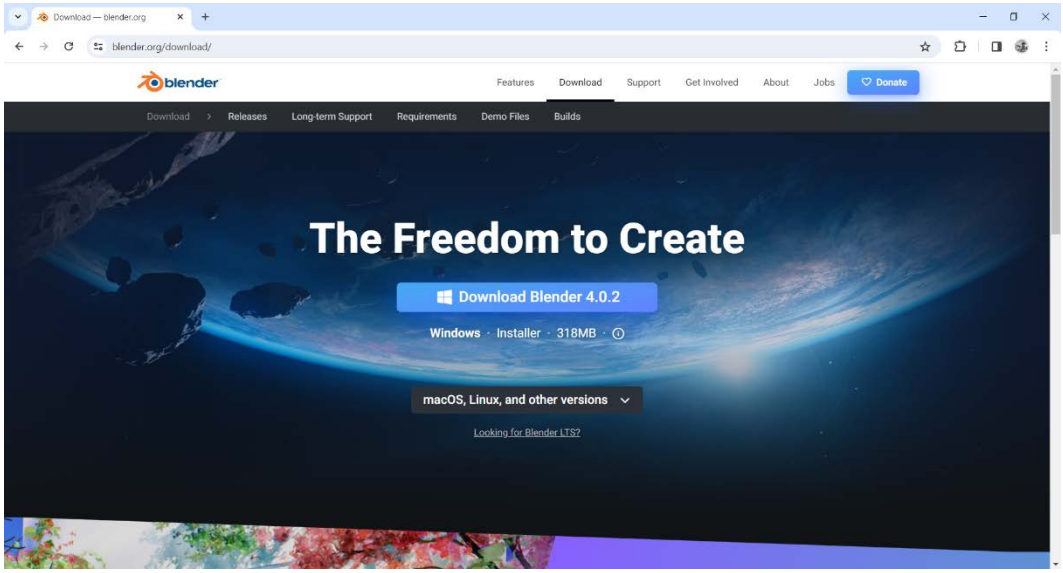
Εικόνα 3.3.1 : Εύρεση σελίδας Blender.

Αφού βρεθεί η σελίδα και γίνει είσοδος σε αυτή, ο χρήστης θα μπορέσει να κατεβάσει το πρόγραμμα Blender από εκεί.



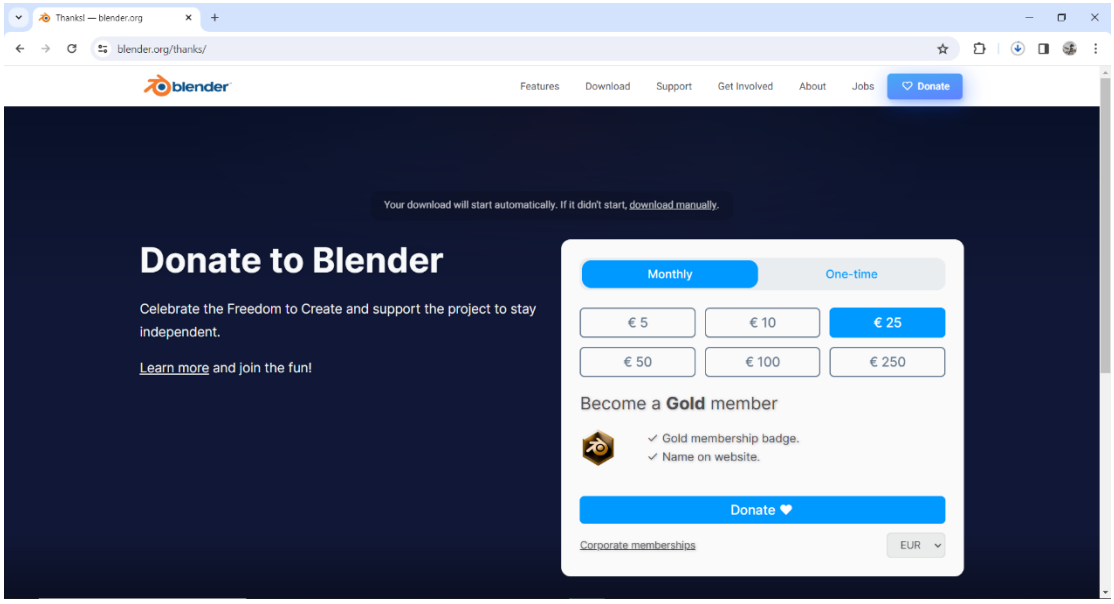
Εικόνα 3.3.2 : Επιλογή λήψης του προγράμματος Blender.

Έπειτα, δίνονται οι επιλογές για το εκάστοτε λειτουργικό σύστημα του χρήστη. Εκεί ο χρήστης θα κάνει κλικ στο πεδίο “Download Blender”, για να ξεκινήσει η λήψη.



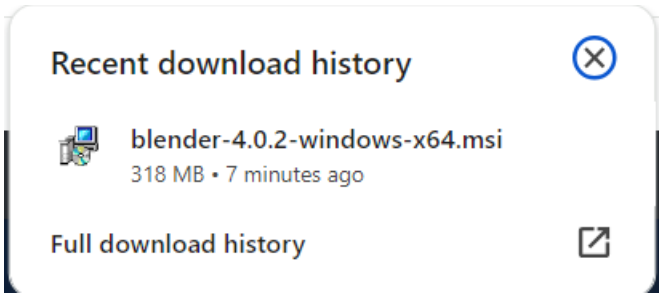
Εικόνα 3.3.3 : Επιλογή λειτουργικού συστήματος και λήψη Blender.

Αφού ο χρήστης κάνει κλικ στο πεδίο λήψης, θα μεταφερθεί σε μια νέα καρτέλα η οποία του αναφέρει ότι μπορεί να στηρίξει με μια δωρεά έτσι ώστε το Blender να παραμείνει ανεξάρτητο και δωρεάν. Είναι προαιρετικό.



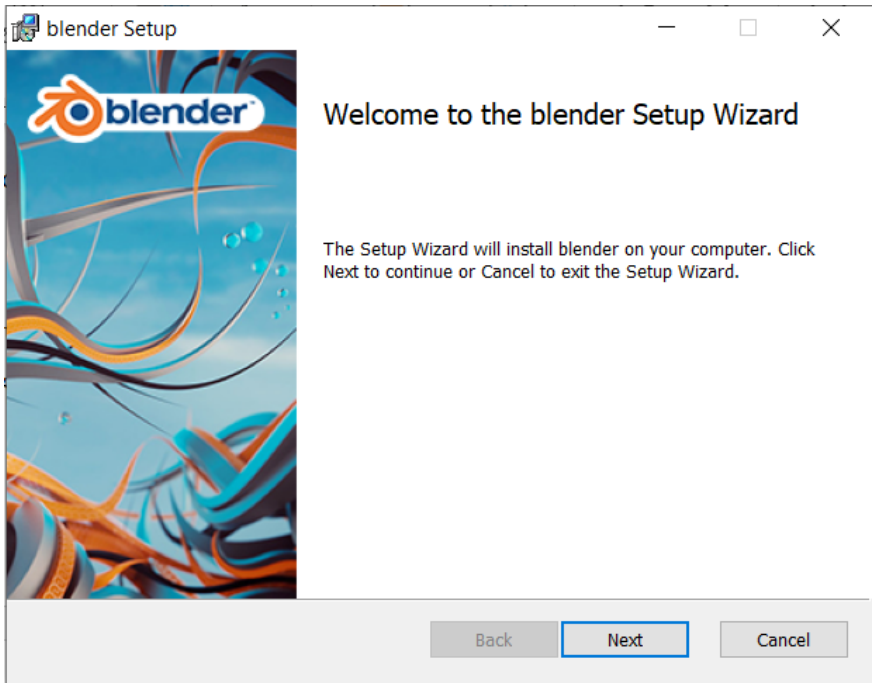
Εικόνα 3.3.4 : Καρτέλα δωρεάς στο Blender.

Αφού η λήψη έχει ολοκληρωθεί, θα ξεκινήσει η διαδικασία εγκατάστασης.



Εικόνα 3.3.5 : Αρχείο λήψης Blender.

Στη συνέχεια και αφού ο χρήστης κάνει κλικ στο αρχείο που μόλις κατέβασε, έχει ξεκινήσει η διαδικασία εγκατάστασης. Στην πρώτη καρτέλα στο πρόγραμμα εγκατάστασης, επιλέγει Next.

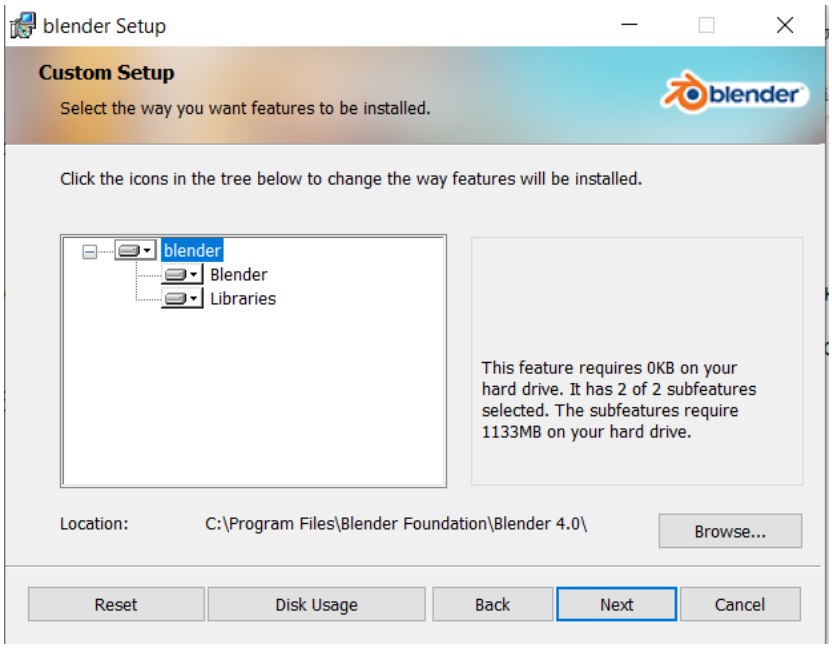


Εικόνα 3.3.6 : Πρόγραμμα εγκατάστασης Blender.

Στην επόμενη καρτέλα, ο χρήστης αφού διαβάσει τους όρους χρήσης, κάνει κλίκ στο checkbox αποδοχής τους και έπειτα επιλέγει το πεδίο Next.

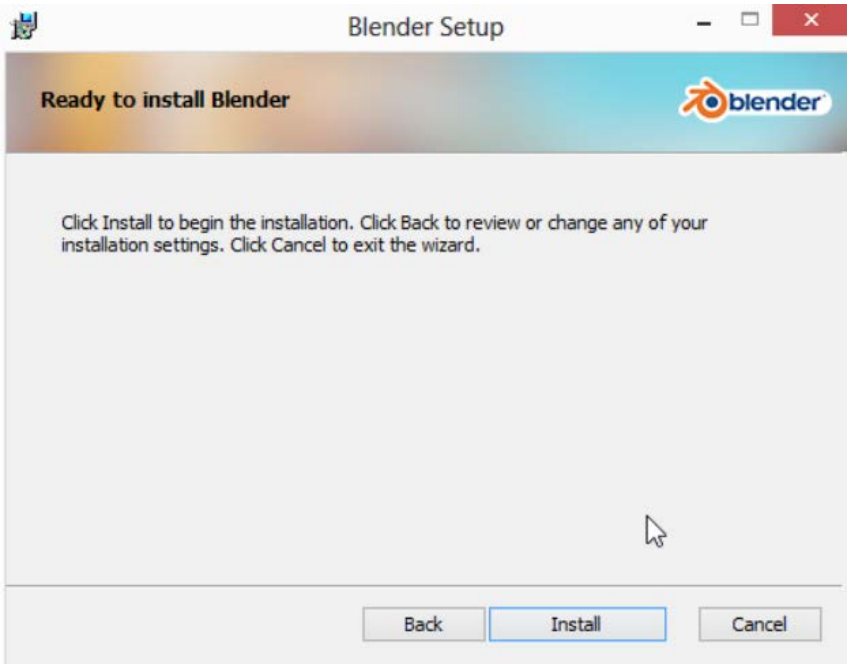
Εικόνα 3.3.7 : Όροι χρήσης Blender.

Στην αμέσως επόμενη καρτέλα, ο χρήστης καλείται να δηλώσει τον φάκελο που θέλει να γίνει εγκατάσταση το πρόγραμμα Blender. Αφού το κάνει αυτό, θα επιλέξει το πεδίο Next έτσι ώστε να συνεχίσει την εγκατάσταση.



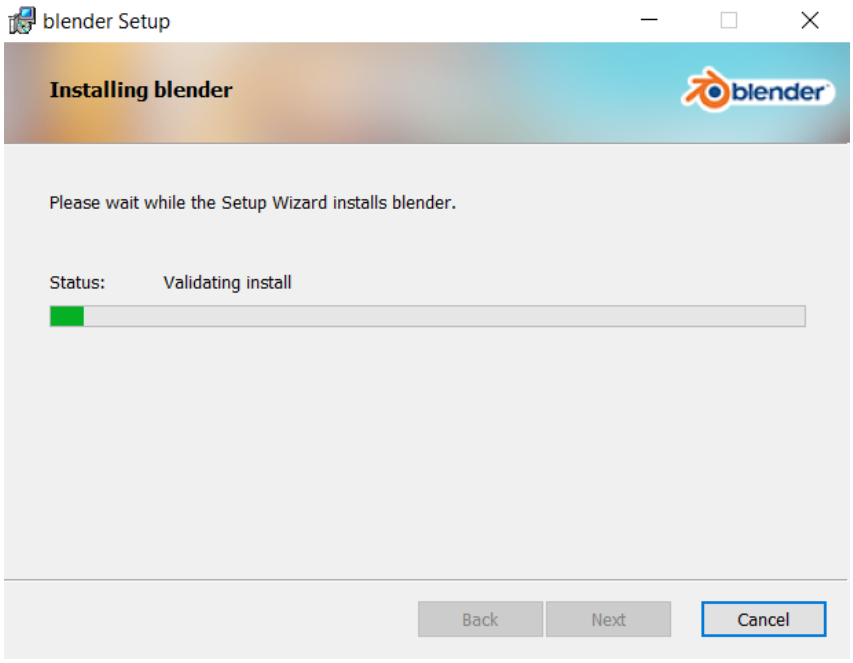
Εικόνα 3.3.8 : Επιλογή φακέλου προορισμού εγκατάστασης και συνέχεια της.

Στην επόμενη καρτέλα, θα οριστικοποιηθεί και θα ξεκινήσει η διαδικασία εγκατάστασης κάνοντας κλικ στο πεδίο Install.



Εικόνα 3.3.9 : Αρχή διαδικασίας εγκατάστασης.

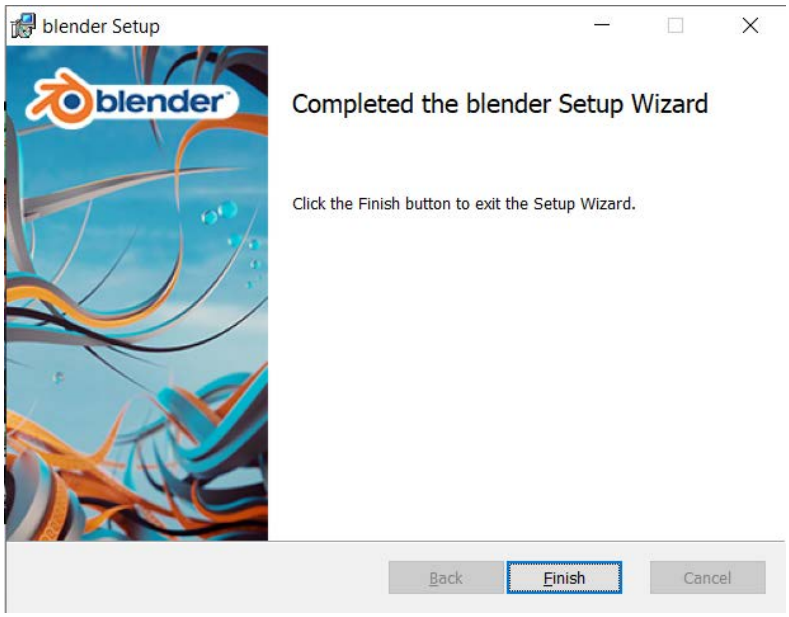
Στη συνέχεια, η πρόοδος της διαδικασίας θα εμφανιστεί.



Εικόνα 3.3.10 : Διαδικασία προόδου εγκατάστασης.

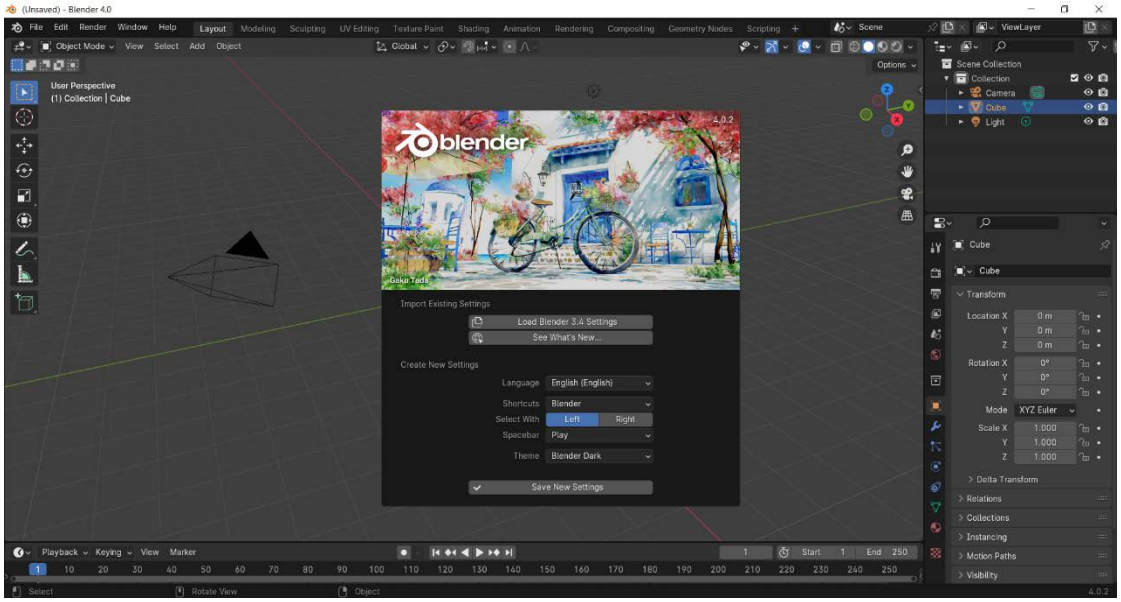
Μόλις η εγκατάσταση του προγράμματος έχει ολοκληρωθεί, εμφανίζεται μια νέα καρτέλα στην οποία αναφέρεται το τέλος της εγκατάστασης. Επιλέγοντας το πεδίο

Finish, ο χρήστης τερματίζει το πρόγραμμα εγκατάστασης και μπορεί πλέον να ανοίξει το πρόγραμμα Blender.



Εικόνα 3.3.11 : Ολοκλήρωση εγκατάστασης Blender.

Πλέον, αφού το πρόγραμμα έχει εγκατασταθεί σωστά, μπορεί να ανοίξει.

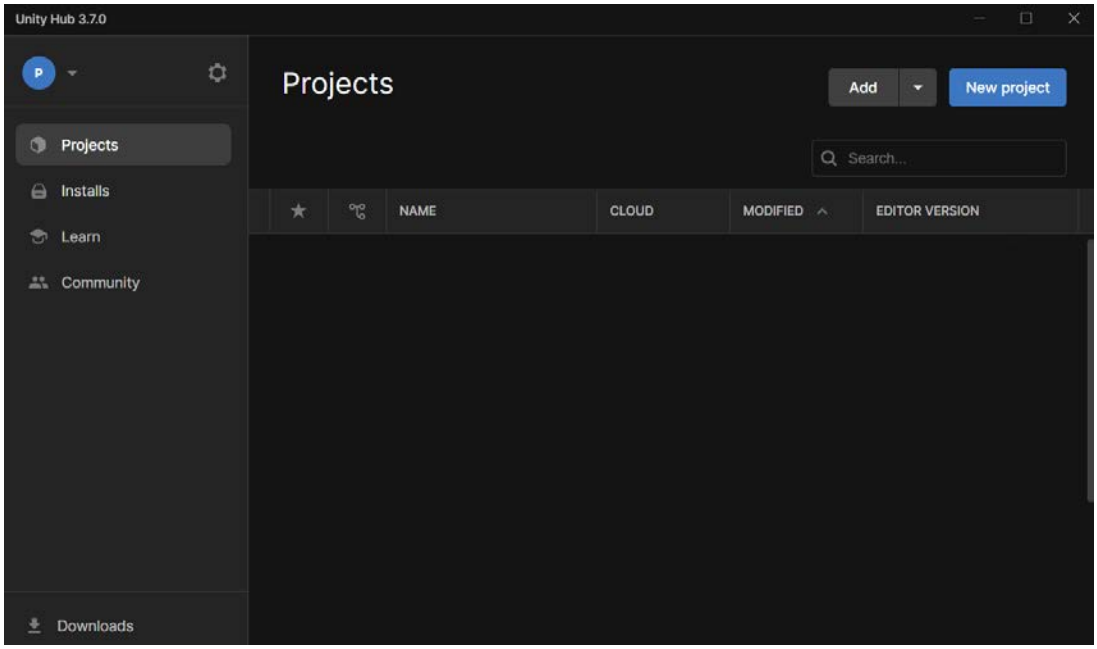


Εικόνα 3.3.12 : Η αρχική οθόνη του Blender.

4 ΚΕΦΑΛΑΙΟ 4 : ΤΟ DEVELOPMENT ΤΟΥ VIDEO GAME

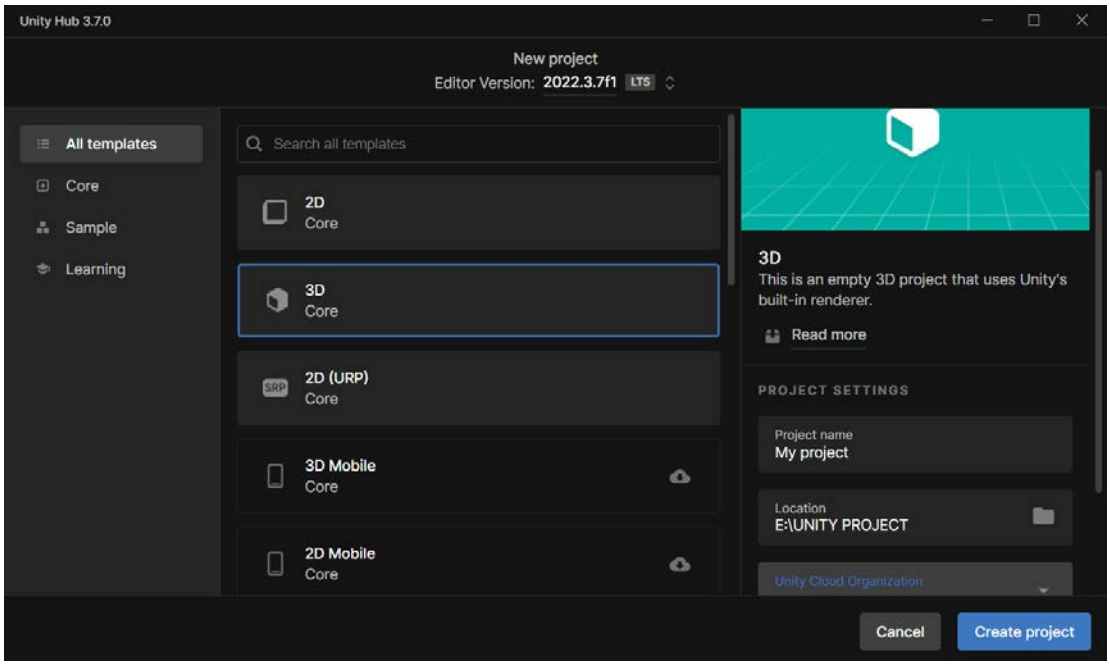
4.1 Η ΠΡΩΤΗ ΕΠΑΦΗ ΜΕ UNITY

Ξεκινώντας, για να μπορέσει το video game να δημιουργηθεί, θα χρειαστεί ο χρήστης να κάνει κάποιες ενέργειες. Αφού το Unity Hub έχει πλέον ανοιχτεί, επιλέγει το πεδίο New Project.



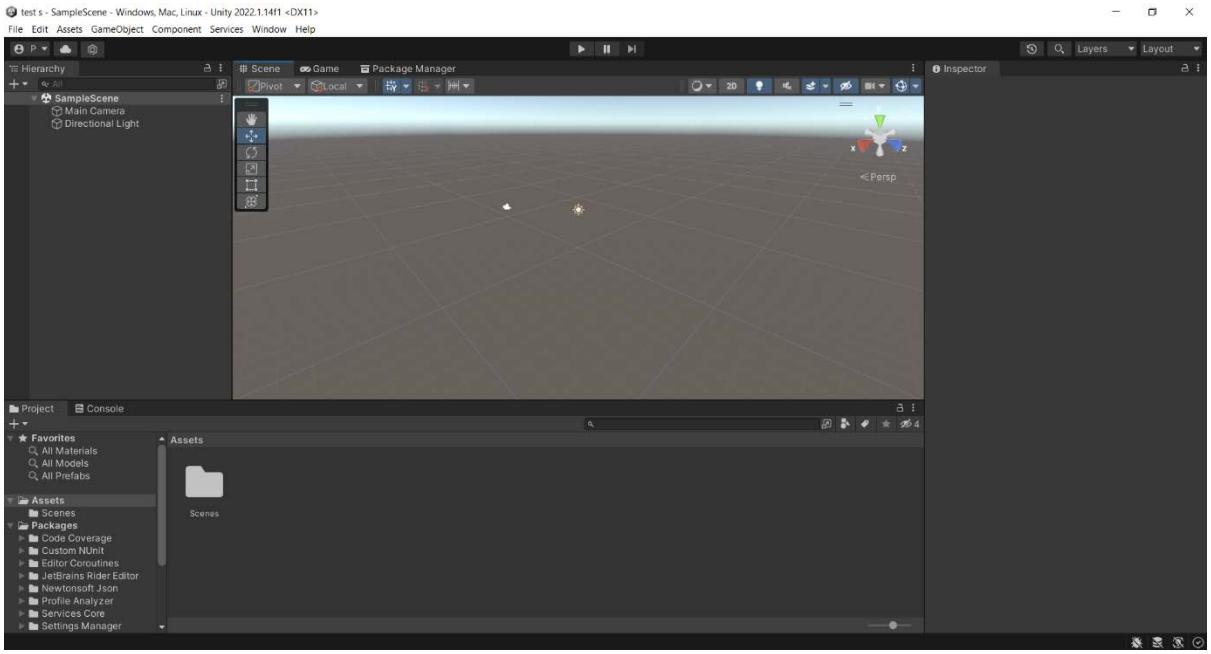
Εικόνα 4.1.1 : Επιλογή New Project.

Μόλις γίνει κλικ στην επιλογή του νέου project, μια νέα καρτέλα θα ανοίξει κι εκεί ο χρήστης θα πρέπει να δώσει μια ονομασία στο project αλλά και να επιλέξει επίσης τον φάκελο που θέλει να αποθηκεύσει τα αρχεία του. Ακόμα, το Unity θα του ζητήσει να επιλέξει τις διαστάσεις του κόσμου του παιχνιδιού (2D ή 3D).



Εικόνα 4.1.2 : Επιλογή και ονομασία του νέου Project.

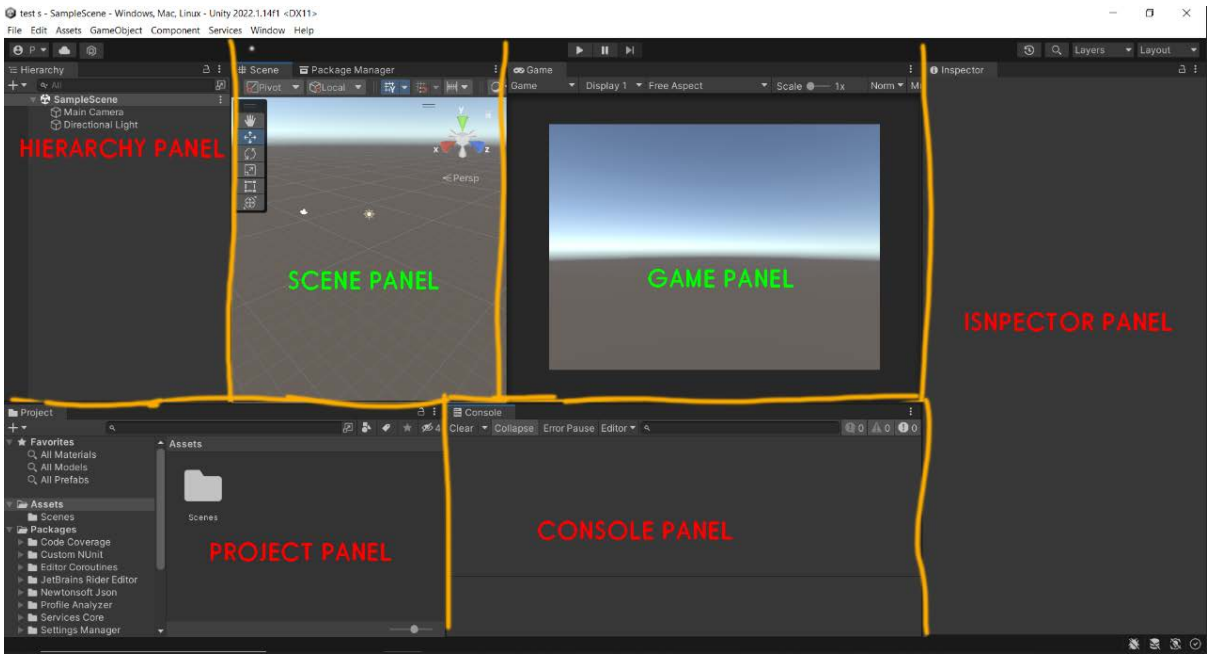
Μόλις το νέο Project δημιουργήθηκε, ανοίγει η αρχική οθόνη του Unity. Ξεκινάει πάντα στην αρχική σκηνή, πάνω στην οποία θα δημιουργηθεί το video game.



Εικόνα 4.1.3 : Unity Interface και αρχική σκηνή.

Στην παραπάνω σκηνή, υπάρχουν κάποια panels τα οποία ο καθένας τα διαμορφώνει με σκοπό να τον βολεύει να διαμορφώσει και να δημιουργήσει το Project καλύτερα. Επάνω δεξιά υπάρχει η επιλογή Layout, στην οποία πατώντας την ο χρήστης θα βρει αρκετές επιλογές διάταξης της σκηνής. Σε περίπτωση που δεν του αρέσει κάποιο απ' τα ήδη υπάρχων, μπορεί να διαμορφώσει τα panel χειροκίνητα. Τα πιο απαραίτητα και βασικά panel είναι τα παρακάτω:

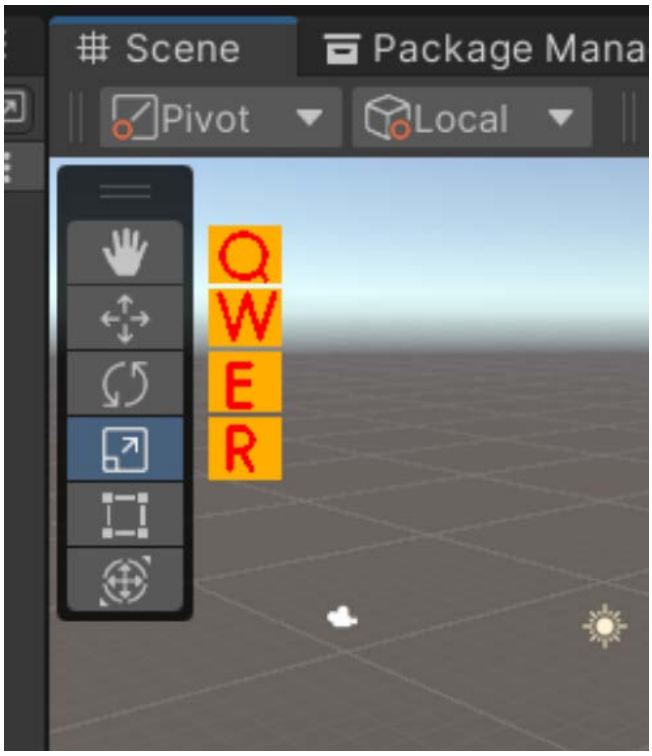
- **Scene Panel**
- **Game Panel**
- **Project Panel**
- **Hierarchy Panel**
- **Inspector Panel**
- **Console Panel**



Εικόνα 4.1.4 : Layout σκηνής στο Unity.

4.1.1 Scene View

Το scene view είναι ο χώρος δημιουργίας του video game και ο χώρος που μπορείς να επεξεργαστείς την σκηνή του παιχνιδιού. Υπάρχουν οι συντομεύσεις πλήκτρων, που κάνουν την εναλλαγή των κύριων εργαλείων πιο γρήγορη και το Unity πιο γρήγορο και εύκολο στη χρήση. Τα πλήκτρα αυτά είναι τα Q, W, E, R και αντιστοιχούν στα εργαλεία που θα δείτε στην εικόνα 5 παρακάτω.



Εικόνα 4.1.5 : Συντομεύσεις πλήκτρων κύριων εργαλείων.

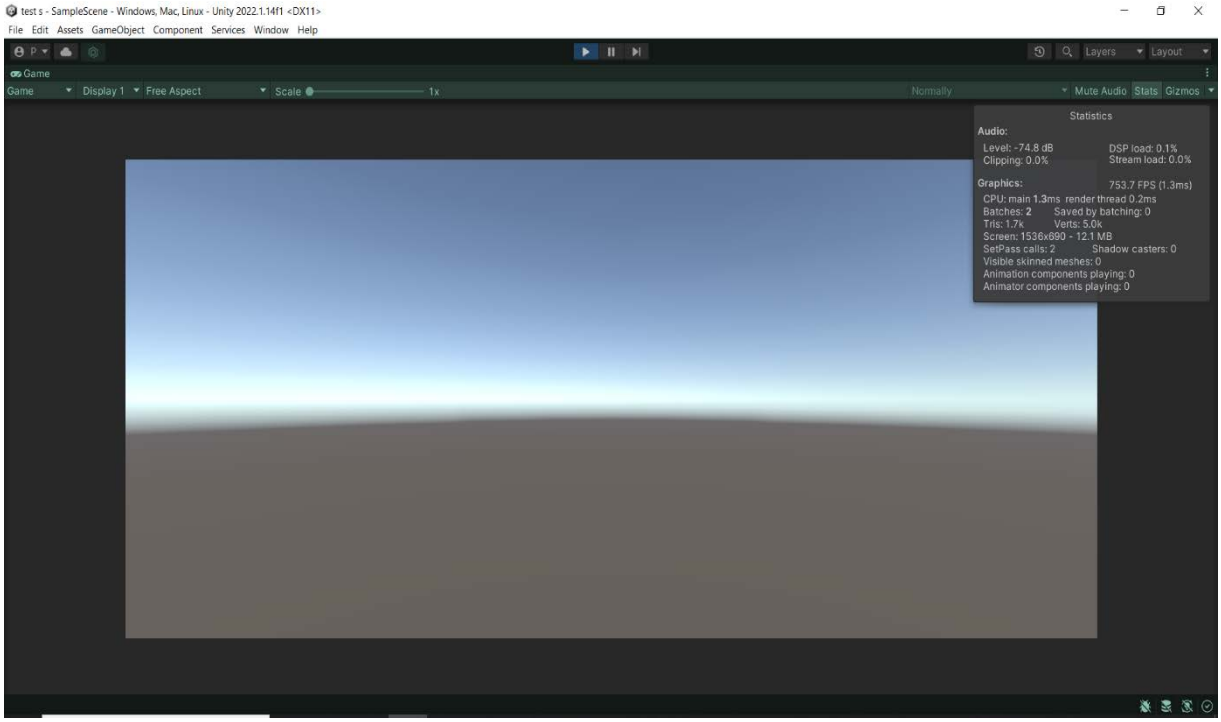
Αυτά τα εργαλεία λοιπόν είναι τα εξής:

- **Hand Tool (Q) :** Με αυτό το πλήκτρο και πατώντας το αριστερό κλικ, ο χρήστης μπορεί να κάνει ελεύθερη περιήγηση στη σκηνή του project και να δει από όποια γωνία θέλει την σκηνή που δημιουργεί. Με το δεξί κλικ πατημένο, ο χρήστης μπορεί να μετακινήσει την κάμερα μετακινώντας μόνο το ποντίκι του.
- **Move Tool (W) :** Με το συγκεκριμένο πλήκτρο, ο χρήστης μετακινεί επάνω στους άξονες X, Y και Z όποιο αντικείμενο επιλέξει απλά επιλέγοντας το, μετακινώντας το με το ποντίκι του.
- **Rotate Tool (E) :** Ο χρήστης, μπορεί να περιστρέψει στους τρεις άξονες που αναφέραμε παραπάνω το αντικείμενο που έχει επιλέξει.

- **Scale Tool (R) :** Με αυτό το πλήκτρο ο χρήστης, αφού έχει επιλέξει ένα αντικείμενο, μπορεί να αλλάξει τις διαστάσεις του σύμφωνα πάλι με τις τρεις διαστάσεις.

4.1.2 Game View

Μέσω του Game View, ο χρήστης θα μπορεί να ελέγχει πως θα φαίνεται το παιχνίδι που κατασκευάζει, βλέποντας την κάμερα την οποία έχει ενσωματώσει. Υπάρχουν τρεις επιλογές και είναι οι εξής, Play, Pause και Step. Ο χρήστης θα πρέπει να προσέχει καθώς όποια αλλαγή κάνει στην σκηνή του ενώ έχει πατημένο το Play ή το Pause δεν θα αποθηκευτούν με αποτέλεσμα όταν βγει απ' το Game Mode να χάσει την πρόοδο του. Το Unity δίνει την δυνατότητα στους χρήστες να μπορούν να αλλάξουν το χρώμα στο background οταν το Play Button είναι πατημένο με αποτέλεσμα να μην μπερδεύονται. Ακόμα υπάρχει και το κουμπί MaximizeOnPlay, το οποίο όταν είναι ενεργοποιημένο και το πλήκτρο Play πατηθεί, τότε το παιχνίδι ανοίγει σε λειτουργία πλήρης οθόνης. Επίσης, ακόμα ένα βασικό εργαλείο είναι και το κουμπί Stats, το οποίο δείχνει τα στατιστικά του παιχνιδιού όπως τους πόρους που καταναλώνει, τα fps κ.α .



Εικόνα 4.1.6 : Το Game View όταν το Play είναι ενεργό (το background με ελαφρύ πράσινο χρώμα).

Play : Το πλήκτρο Play, μπορεί να ενεργοποιηθεί οποιαδήποτε στιγμή το θελήσει ο χρήστης. Με το πάτημα του, το παιχνίδι που δημιουργεί ξεκινάει και ο χρήστης μεταφέρεται σε αυτό βλέποντας την κάμερα ή τις κάμερες που έχει τοποθετήσει.

Pause : Το Pause απ' την άλλη, παγώνει το παιχνίδι στη περίπτωση που ο χρήστης θέλει να αλλάξει κάποιες τιμές όπως τον φωτισμό, την ταχύτητα του παίχτη ή για να παρατηρήσει κάποια λεπτομέρια.

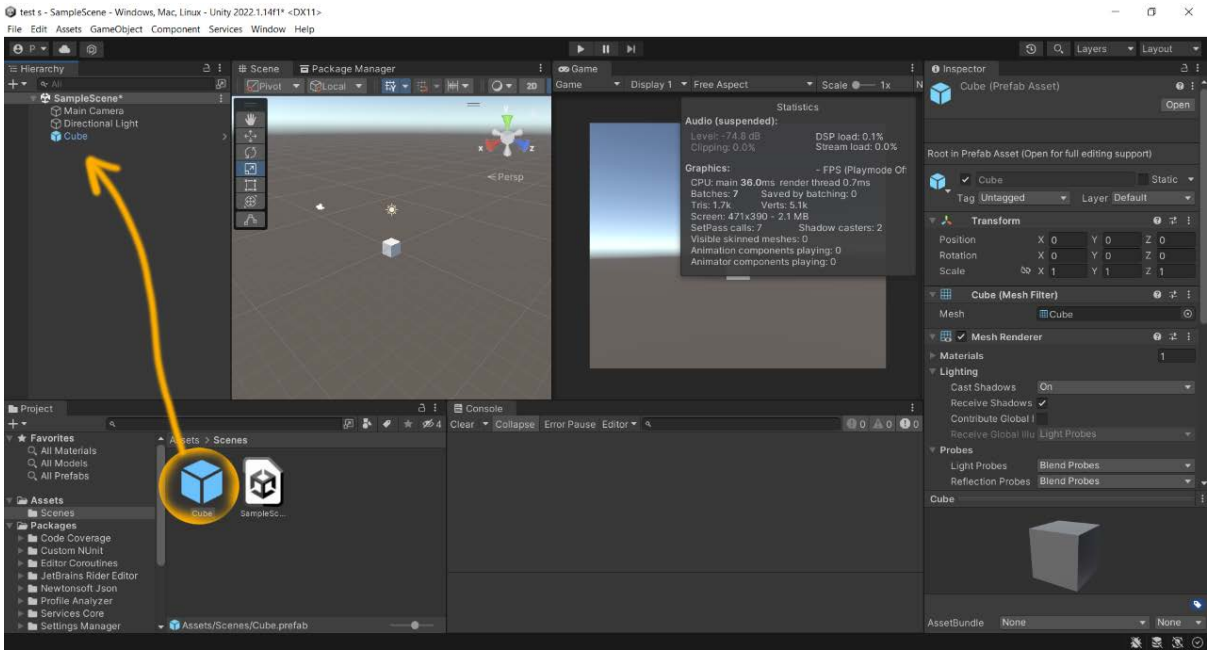
Step: Πατώντας αυτό το κουμπί, ο χρήστης μπορεί να μεταφερθεί στο επόμενο frame για να κάνει τις αλλαγές που επιθυμεί.

4.1.3 Hierarchy

Το Hierarchy Panel, είναι το παράθυρο το οποίο περιέχει τα αντικείμενα που βρίσκονται στη σκηνή (scene panel). Ο χρήστης μπορεί να αλλάξει τις τιμές, τα ονόματα ή ό,τι άλλο κρίνει εκείνος απαραίτητο στο παιχνίδι. Όταν το πλήκτρο Play πατηθεί για να ξεκινήσει το παιχνίδι, τότε οι πληροφορίες που χρειάζονται αντλούνται απ' το Hierarchy Panel.

4.1.4 Project Panel

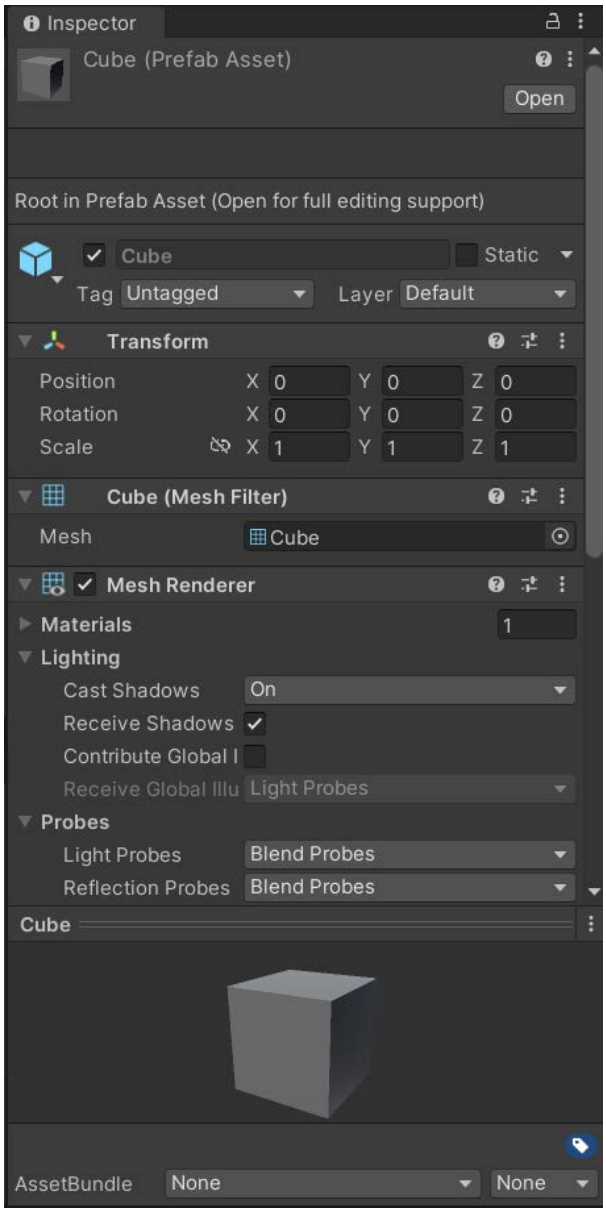
Εκεί βρίσκονται όλα τα Assets του παιχνιδιού. Ακόμα ο χρήστης μπορεί να αποθηκεύσει τα scripts του, animations, 3d μοντέλα, αρχεία ήχου κ.α και να τα χρησιμοποιήσει αν και όποτε εκείνος θέλει. Μπορεί με την μέθοδο drag and drop να τα σύρει απ' το Project Panel στο Hierarchy Panel ή στο Scene Panel αν είναι 3d μοντέλο.



Εικόνα 4.1.7 : Μεταφορά 3D μοντέλου απ' το Project Panel στο Hierarchy Panel.

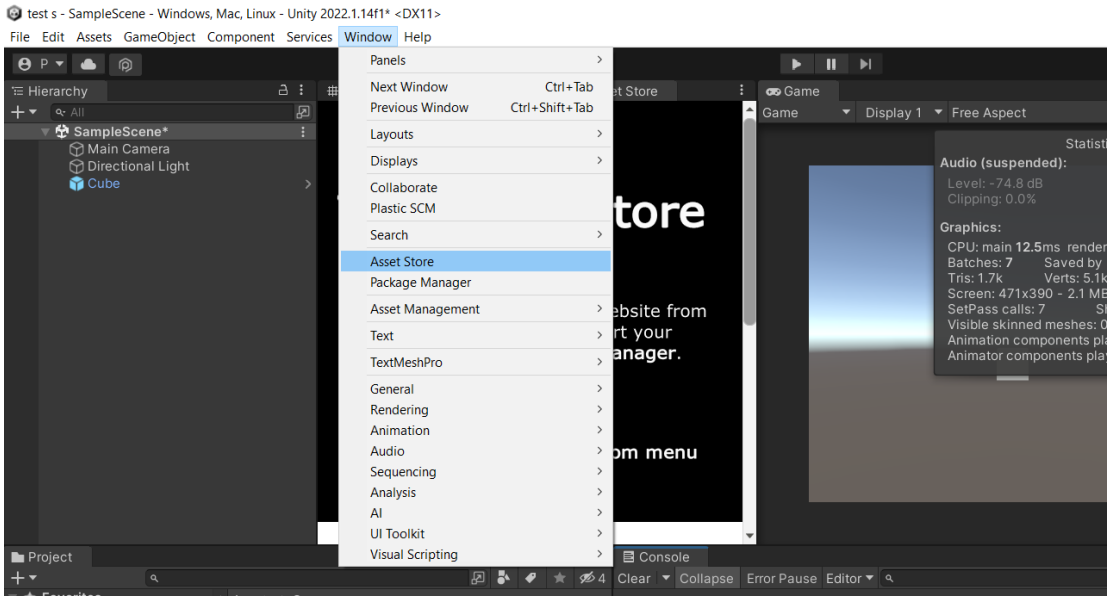
4.1.5 Inspector Panel

Σε αυτό το πάνελ, ο χρήστης βλέπει τα χαρακτηριστικά του αντικειμένου που έχει επιλέξει όπου και αν βρίσκεται αυτό (είτε στο Project είτε στο Hierarchy Panel). Ακόμα μπορεί να επεξεργαστεί διάφορα δεδομένα του αντικειμένου, να προσθέσει scripts κ.α .

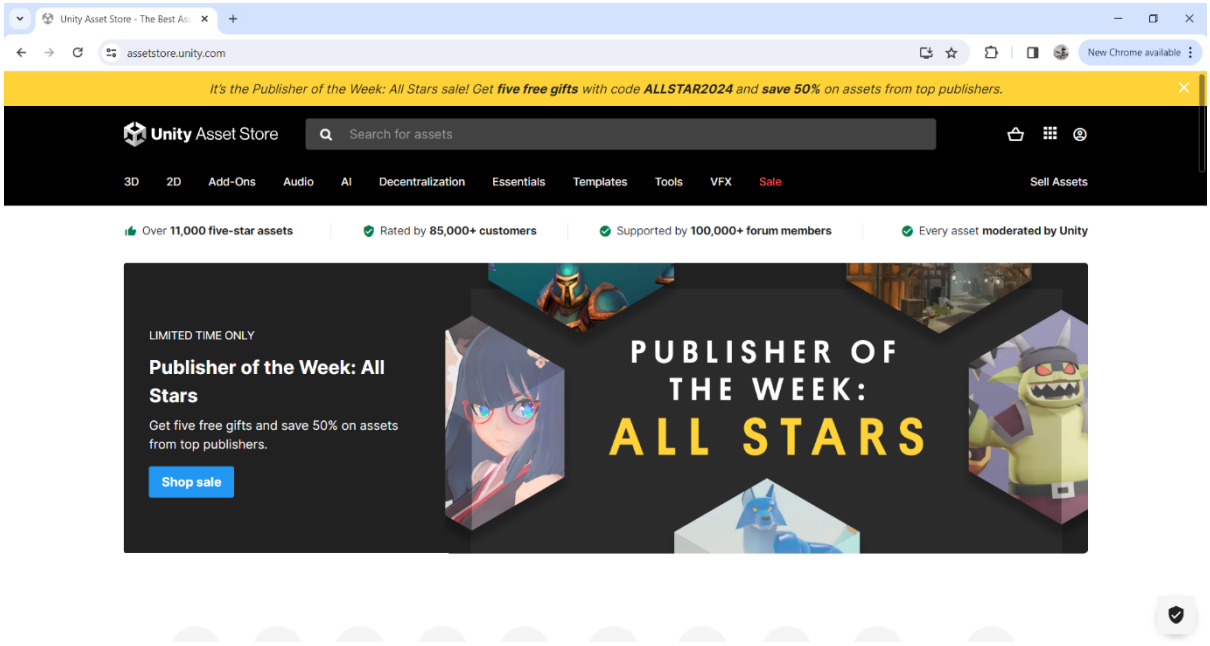


Εικόνα 4.1.8 : Το Inspector Panel του αντικειμένου Cube.

Το Unity όπως και τα περισσότερα προγράμματα δημιουργίας βιντεοπαιχνιδιών, δίνουν την δυνατότητα στο χρήστη να κατεβάσει 3D μοντέλα, σκριπτς, ήχους κ.α τα οποία τα έχουν δημιουργήσει άλλοι χρήστες μέσα απ' το Asset Store. Μπορεί να έχει πρόσβαση σε αυτό είτε μέσω του προγράμματος του Unity (έκδοση 2020.1 και παλαιότερες μόνο), είτε μέσω του browser του στον σύνδεσμο https://assetstore.unity.com/ . Τα περισσότερα Assets είναι έναντι χρηματικού αντιτίμου, αλλά υπάρχουν και κάποια δωρεάν χαμηλότερης ποιότητας.



Εικόνα 4.1.9 : Πρόσβαση στο Asset Store μέσω του Unity.

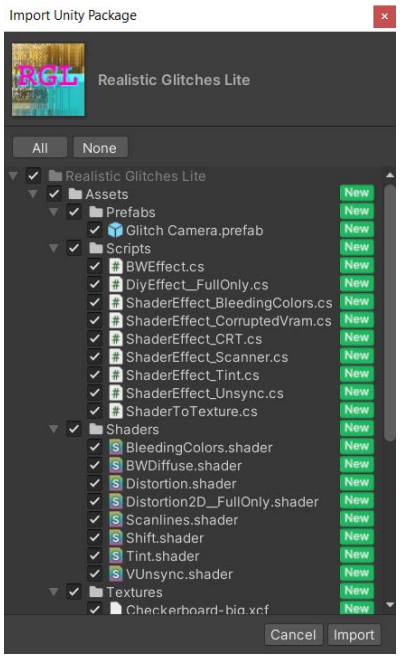


Εικόνα 4.1.10 : Το Asset Store μέσω του browser (https://assetstore.unity.com/) .

Το Asset Store, θα μπορούσε να φανεί χρήσιμο κυρίως σε αρχάριους χρήστες του Unity, θα ήταν ένα πολύ εύχρηστο εργαλείο και θα έλυνε τα χέρια τους αφού περιέχει πάρα πολλές επιλογές για κατέβασμα. Ακόμα πιο εύκολο το κάνει και η απλή μεταφορά των Assets απ' την σελίδα στο scene του χρήστη, είτε βρίσκεται στην σελίδα μέσω του προγράμματος, είτε μέσω του browser του. Για να εγκαταστήσει λοιπόν το Asset, ο χρήστης το βρίσκει και έπειτα το κατεβάζει. Αμέσως μετά

πατώντας Import μπορεί να το φορτώσει στο scene του με ένα κλικ. Πολύ χρήσιμη είναι η δυνατότητα επιλογής συγκεκριμένων Assets απ' αυτά που υπάρχουν στο πακέτο. Με αυτή την επιλογή ο χρήστης δεν φορτώνει στη σκηνή του τα άχρηστα Asset και δεν βαραίνει το project του με αυτά.

Εικόνα 4.1.11 : Εύρεση Asset στο Asset Store.



Εικόνα 4.1.12 : Εισαγωγή του Asset στο Unity με μόλις ένα κλικ.

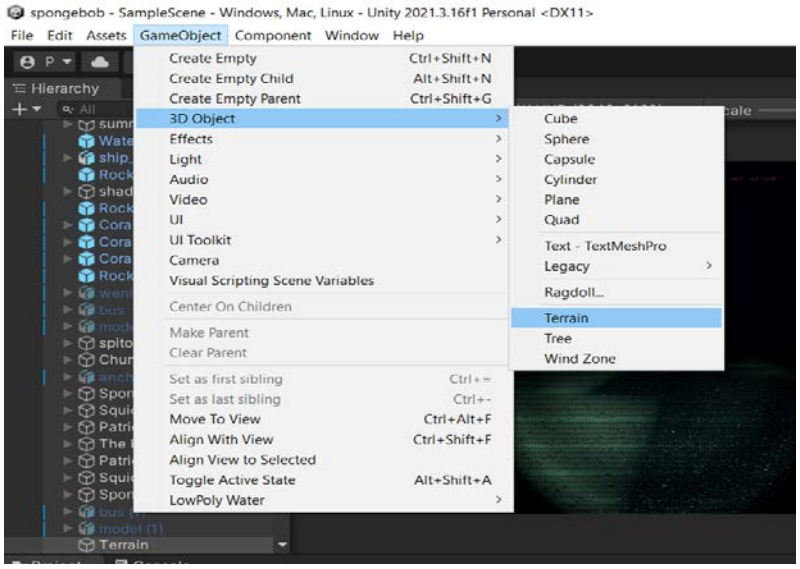
4.2 ΠΡΟΛΟΓΟΣ ΠΑΙΧΝΙΔΙΟΥ

Αφού λοιπόν έχει ανοίξει το Unity και το αρχικό Project έχει δημιουργηθεί, έχει φτάσει η ώρα να προστεθούν τα κατάλληλα Assets για να αρχίσει να δημιουργείται το παιχνίδι. Το όνομα του παιχνιδιού είναι The Jellyfish Hunter. Είναι ένα single player first person παιχνίδι και αποτελείται από 2 σκηνές, την σκηνή του main menu και την βασική σκηνή του παιχνιδιού. Ο σκοπός του παιχνιδιού είναι ο παίχτης να βρεί και τις 7

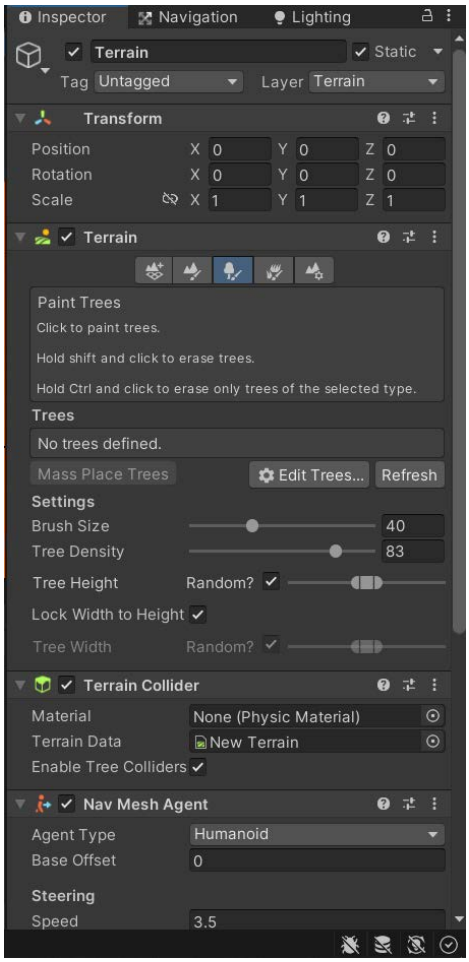
μέδουσες (Jellyfishes) και να κερδίσει το παιχνίδι αλλά δεν είναι μόνος. Ο εχθρός τον κυνηγάει παρεμποδίζοντας τον. Αν ο παίχτης χάσει ξεκινάει απ' την αρχή.

4.3 ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ TERRAIN

Ξεκινώντας την δημιουργία του παιχνιδιού, θα χρειαστεί να δημιουργηθεί το έδαφος. Για την δημιουργία του, χρησιμοποιήθηκε ένα έτοιμο και βασικό αντικείμενο του Unity, το Terrain. Το Unity δίνει την δυνατότητα στον χρήστη να χρησιμοποιεί έτοιμα αντικείμενα όπως το Terrain το οποίο είναι ένα έτοιμο 3d αντικείμενο και χρησιμοποιείται σαν έδαφος. Έχει κάποιες έτοιμες ιδιότητες όπως δημιουργία διαφορετικών Textures, επεξεργασία της μορφολογίας του εδάφους κ.α, και όλα αυτά με ένα brush. Για να έχει πρόσβαση σε αυτά ο χρήστης το μόνο που χρειάζεται να κάνει είναι ένα κλικ στο Terrain αντικείμενο που δημιούργησε και έπειτα μέσω του Inspector Panel να κάνει τις αλλαγές που εκείνος θέλει.

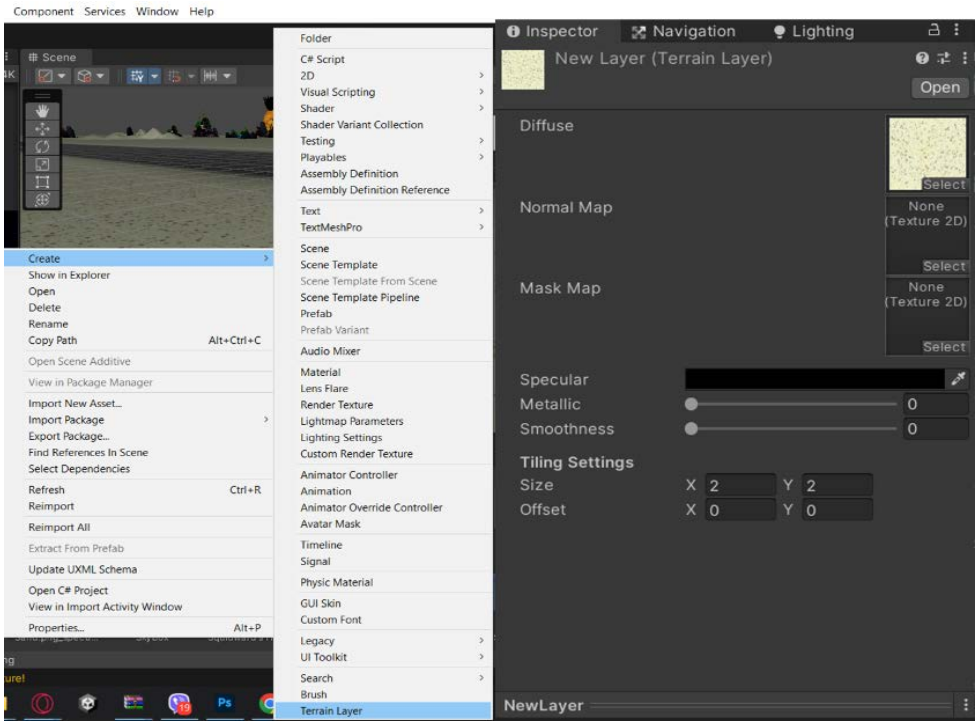


Εικόνα 4.3.1 : Δημιουργία Terrain.



Εικόνα 4.3.2 : Επεξεργασία στοιχείων Terrain.

Κάτι άλλο που μπορεί να επεξεργαστεί είναι και το χρώμα του εδάφους. Για να μπορέσει να συμβεί αυτό, θα πρέπει να υπάρχει και το κατάλληλο Terrain Layer. Ο χρήστης μπορεί να βρει έτοιμα Textures στο Unity Asset Store ή να δημιουργήσει τα δικά του. Για να ξεκινήσει την δημιουργία πρέπει να κάνει δεξί κλικ κάπου στο Project Panel και στην συνέχεια Create και Terrain Layer. Δημιουργώντας το Terrain Layer ο χρήστης θα πρέπει να επιλέξει το Texture που θα θέλει να εμφανίζεται μέσω του Inspector Panel.

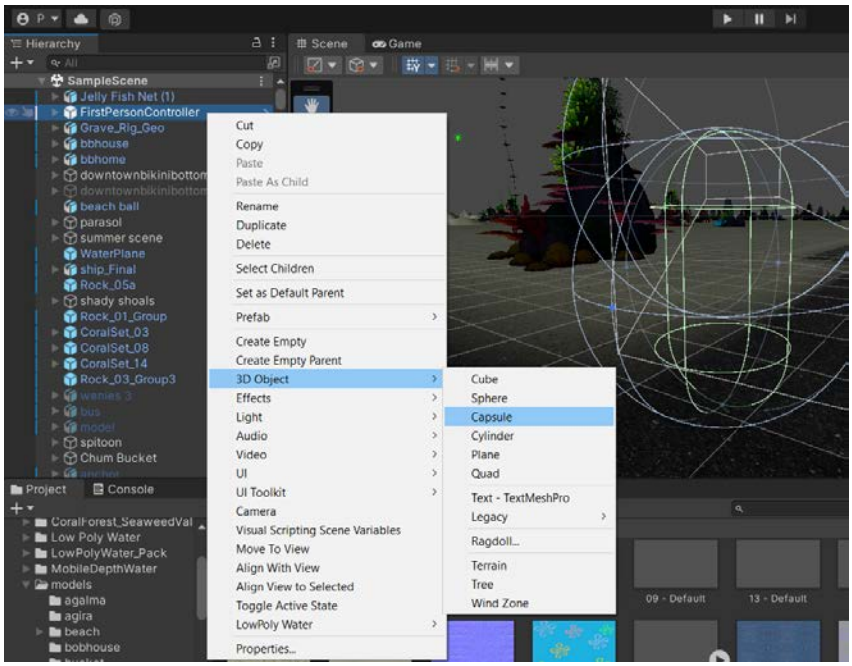


Εικόνα 4.3.3 : Δημιουργία και επεξεργασία Terrain Layer.

Έπειτα, για να μπορέσει να το χρησιμοποιήσει θα πρέπει να επιλέξει το Terrain που έχει δημιουργήσει και απ' τις ρυθμίσεις στο Inspector Panel να προσθέσει το Terrain Layer που δημιούργησε. Μετά, το Unity του δίνει την δυνατότητα να ζωγραφίσει το Terrain με ένα Brush το οποίο είναι και αυτό επεξεργάσιμο και έτσι ο χρήστης μπορεί να δημιουργήσει δρόμους και να βάψει το έδαφος του όπως εκείνος επιθυμεί.

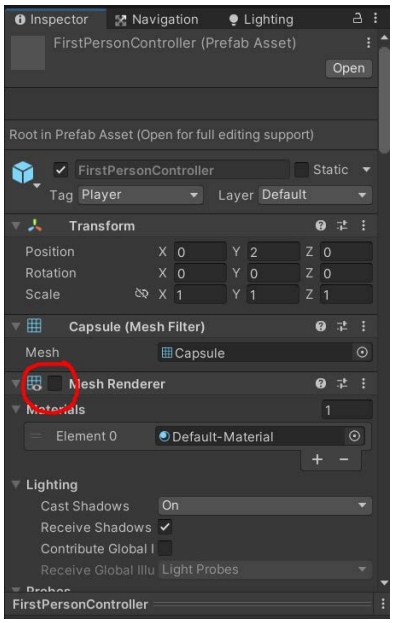
4.4 ΔΗΜΙΟΥΡΓΙΑ FIRST PERSON ΠΑΙΧΤΗ - PLAYER

Η διαδικασία που ακολουθεί για την δημιουργία του παίχτη, μοιάζει πάρα πολύ με αυτή που για την δημιουργία εδάφους - Terrain. Η διαφορά είναι ότι θα δημιουργηθεί μία κάψουλα (capsule) αντί για Terrain. Όπως είχε αναφερθεί και πριν, και η κάψουλα και το Terrain είναι έτοιμα αντικείμενα του Unity που μπορεί ο χρήστης να δημιουργήσει με σχετική ταχύτητα και ευκολία ή ακόμα μπορεί να εγκαταστήσει έναν απλό παίχτη απ' το Asset Store. Οπότε, ένας First Person παίχτης - player, αποτελείται από ένα 3d αντικείμενο, ένα script κίνησης και μία κάμερα.



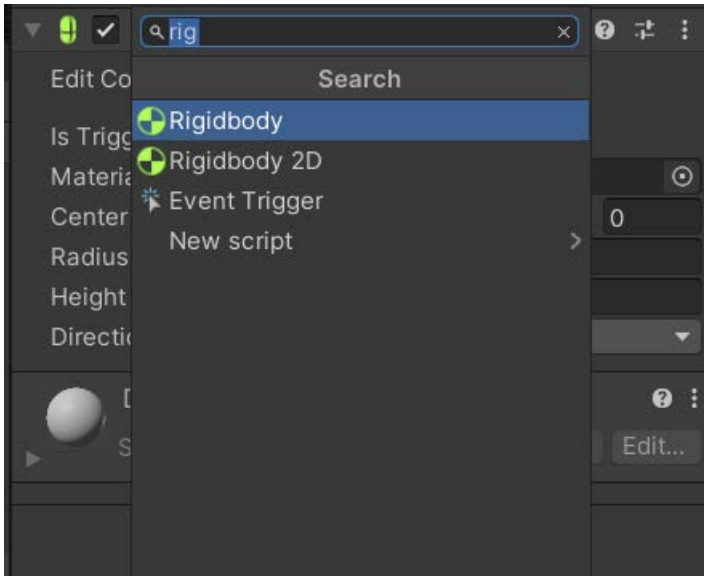
Εικόνα 4.4.1 :Δημιουργία κάψουλας παίχτη.

Εφόσον το παιχνίδι είναι πρώτου προσώπου και ο παίχτης δεν θα έχει την δυνατότητα να δει τον χαρακτήρα του, καλό θα ήταν ο χρήστης να σβήσει το Mesh Renderer της κάψουλας. Είναι αυτό που δίνει την τρισδιάστατη εικόνα της κάψουλας και του κάθε μοντέλου γενικά, οπότε αν απενεργοποιηθεί απ' το αντικείμενο τότε αυτό θα πάψει να φαίνεται στο μάτι αλλά θα συνεχίσει να υπάρχει σαν αντικείμενο μαζί με τις ιδιότητες του στο παιχνίδι.



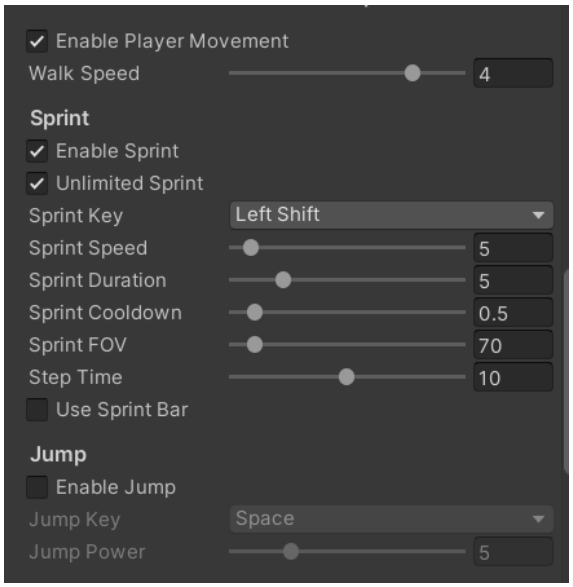
Εικόνα 4.4.2 : Απενεργοποίηση του Mesh Renderer της κάψουλας - Player.

Ακόμα κι έτσι, ο παίχτης δεν διαφέρει σε τίποτα από ένα τρισδιάστατο αντικείμενο της Unity. Ακόμα και αν πατηθεί το πλήκτρο Play ο παίχτης δεν θα μπορέσει να κινηθεί και θα παραμένει στο σημείο που βρισκόταν εξ αρχής, χωρίς να το επηρεάζει κάποια φυσική δύναμη. Για να μπορέσει ο χρήστης να ασκήσει κάποια δύναμη και να κινήσει τον παίχτη, θα χρειαστεί να περάσει το RigidBody Component το οποίο θα μπορέσει να το βρει στο Inspector Panel επιλέγοντας την κάψουλα, Add Component και έπειτα RigidBody.



Εικόνα 4.4.3 : Προσθήκη RigidBody Component στον παίχτη.

Πλέον, πατώντας ο χρήστης το Play, το αντικείμενο παίχτης θα δέχεται δύναμη απ΄τη βαρύτητα και θα πέφτει στο έδαφος - terrain που δημιουργήσαμε. Για να μπορέσειο χρήστης να μετακινήσει τον παίχτη - player, θα χρειαστεί να δημιουργήσει ένα script κίνησης. Έτσι, ο player θα μετακινείται με τα κουμπιά W,A,S,D .



Εικόνα 4.4.4 : Προσθήκη και επεξεργασία script κίνησης.

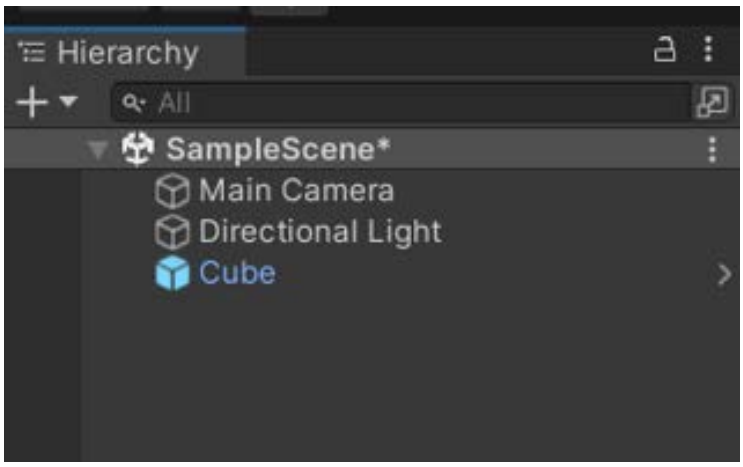
Το συγκεκριμένο script κίνησης, ζητάει απ' τον χρήστη να ορίσει τις τιμές για την ταχύτητα με την οποία θα κινείται ο χαρακτήρας - player, τον χρόνο για τον οποίο θα μπορεί να κάνει sprint, το cooldown του. Απ' όσο βλέπετε στην εικόνα 4 παραπάνω, ο χαρακτήρας μπορεί να κάνει sprint για απεριόριστο χρόνο αφού το Unlimited Sprint checkbox είναι ενεργοποιημένο.

```
void FixedUpdate()
{
    #region Movement

    if (playerCanMove)
    {
        // Calculate how fast we should be moving
        Vector3 targetVelocity = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));

        // Checks if player is walking and isGrounded
        // Will allow head bob
        if (targetVelocity.x != 0 || targetVelocity.z != 0 && isGrounded)
        {
            isWalking = true;
        }
        else
        {
            isWalking = false;
        }

        // All movement calculations shile sprint is active
        if (enableSprint && Input.GetKey(sprintKey) && sprintRemaining > 0f && !isSprintCooldown)
        {
            targetVelocity = transform.TransformDirection(targetVelocity) * sprintSpeed;

            // Apply a force that attempts to reach our target velocity
            Vector3 velocity = rb.velocity;
            Vector3 velocityChange = (targetVelocity - velocity);
            velocityChange.x = Mathf.Clamp(velocityChange.x, -maxVelocityChange, maxVelocityChange);
            velocityChange.z = Mathf.Clamp(velocityChange.z, -maxVelocityChange, maxVelocityChange);
            velocityChange.y = 0;
```

Εικόνα 4.4.5 : Τμήμα κώδικα απ΄το script κίνησης του παίχτη.

4.5 ΠΡΟΣΘΗΚΗ ΚΑΜΕΡΑΣ

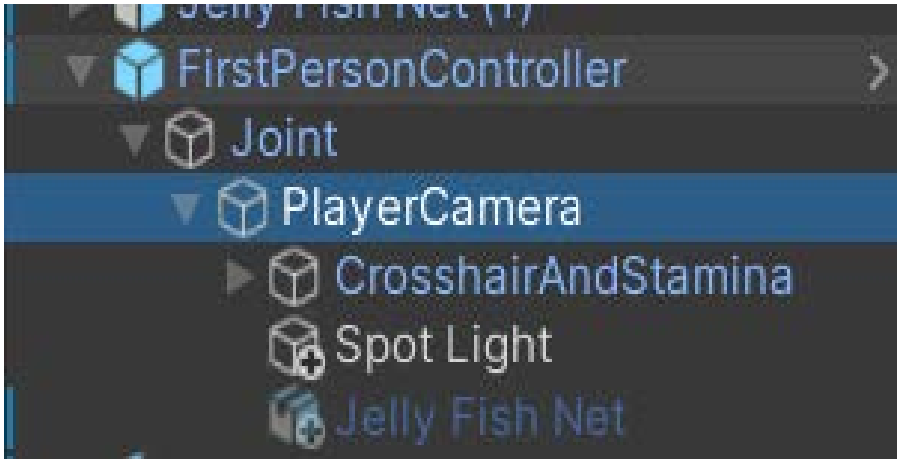
Ένα απ' τα πιο βασικά εργαλεία για να δημιουργηθεί ένα first person video game, είναι η κάμερα καθώς χωρίς αυτή το παιχνίδι δεν είναι λειτουργικό. Αυτός είναι και ο λόγος που κάθε φορά που ο χρήστης δημιουργεί ένα νέο project, το Unity δημιουργεί αυτομάτως μία Main Camera.



Εικόνα 4.5.1 : Η Main Camera σε νέο project.

Οι ρυθμίσεις της κάμερας είναι διαφορετικές σε κάθε είδος παιχνιδιού που σκοπεύει να δημιουργήσει ο κάθε χρήστης. Η κάμερα μπορεί να διαμορφωθεί με πολλούς τρόπους, μπορεί με κάποιο script ακολουθώντας την τοποθεσία του παίχτη, αλλά

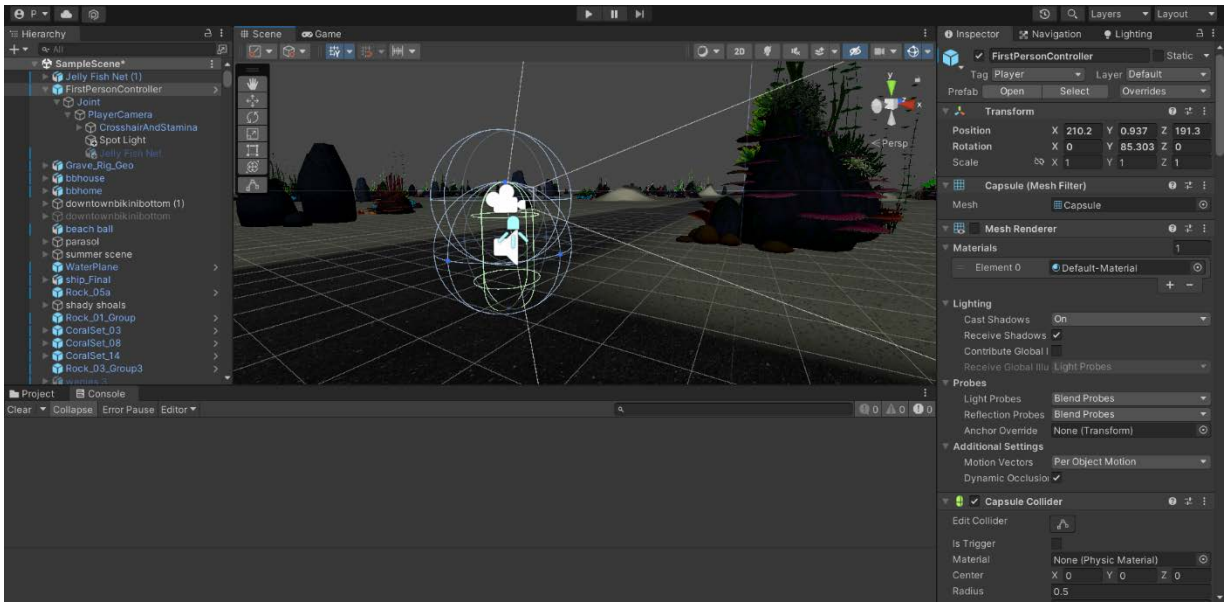
ακόμα μπορεί να επιτευχθεί ο συγκεκριμένος σκοπός απλά εφαρμόζοντας την κάμερα επάνω στο αντικείμενο του player. Η συγκεκριμένη μέθοδος εφαρμόζεται με την βοήθεια του Parenting, δηλαδή η κάμερα θα γίνει "παιδί" του player και είναι μια απ' τις βασικές έννοιες του Unity. Ακόμα μπορεί να βρίσκεται κολλημένη σε ένα σημείο στο χάρτη, αλλά υπάρχουν και διάφοροι άλλοι τρόποι. Η κάμερα στο The Jellyfish Hunter βρίσκεται σαν παιδί μέσα σε ένα emty object το οποίο βρίσκεται μέσα στον παίχτη. Έχει μέσα της 2 script που έχουν να κάνουν με διάφορα εφέ που έχουν χρησιμοποιηθεί, τα οποία θα αναλυθούν αργότερα. Η δημιουργία ενός empty object γίνεται πολύ απλά με ένα δεξί κλικ στο Hierarchy Panel, θα εμφανιστεί η επιλογή Create Emty. Αμέσως μετά, με drag and drop μέθοδο θα οδηγήσει την κάμερα στο άδειο αντικείμενο κι έτσι θα έχει δημιουργηθεί δεσμός Parenting.



Εικόνα 4.5.2 : Δημιουργία Joint(άδειο αντικείμενο) και Parenting με PlayerCamera.

Ένα αντικείμενο EmptyObject, είναι αυτό που δεν είναι εμφανές στο παιχνίδι απ΄τον χρήστη, δεν έχει διαστάσεις και δεν μπορεί να τοποθετηθεί πουθενά στη σκηνή. Το μόνο που μπορεί να κάνει ο χρήστης σε αυτό είναι να προσθέσει κάποιο Component, όπως ένα Script ή κάποιο αρχείο ήχου. Το συγκεκριμένο αντικείμενο έχει ονομαστεί Joint και περιέχει μέσα του με τη μέθοδο Parenting την κάμερα, ένα αρχείο φωτός που χρησιμοποιείται σαν φακός και την απόχη που χρειάζεται ο παίχτης για να μαζέψει τις μέδουσες. Είναι τοποθετημένο μέσα στο αρχείο του παίχτη, έτσι ώστε η κάμερα να τον ακολουθεί χωρίς την βοήθεια κάποιου Script με αποτέλεσμα το παιχνίδι να είναι ελαφρύτερο. Το script που ελέγχει την κάμερα βρίσκεται στο αρχείο του χαρακτήρα με ονομασία FirstPersonController και επιτρέπει στον χρήστη να κάνει

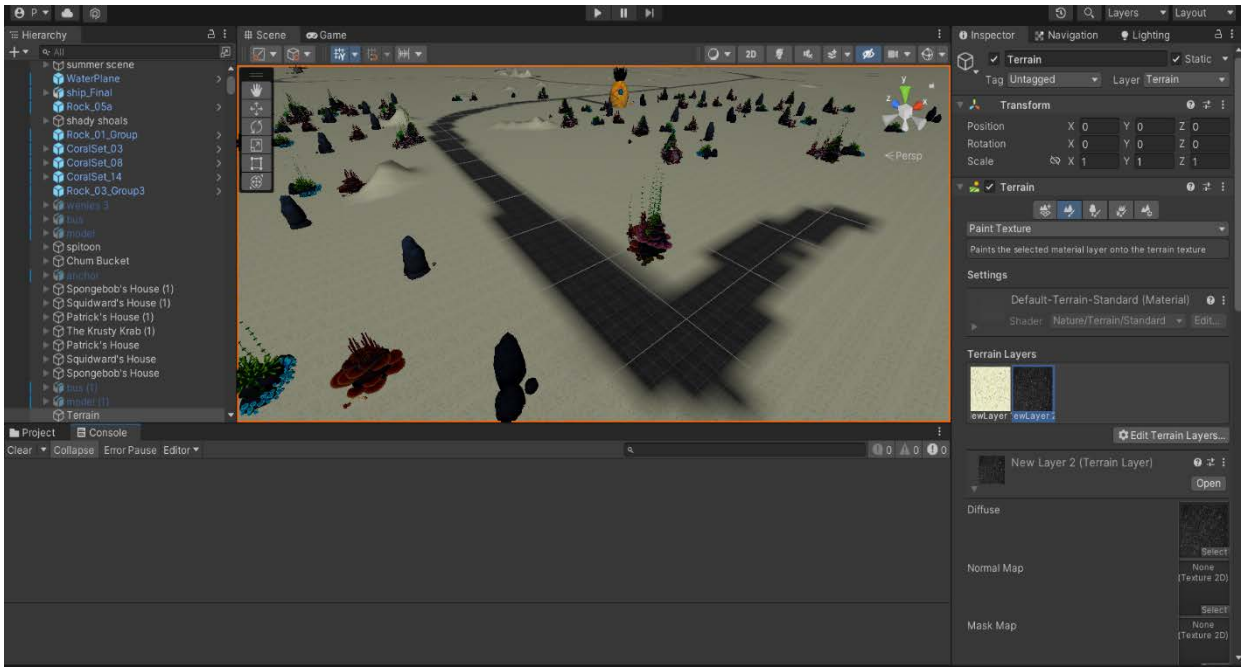
Rotate την κάμερα στον άξονα της. Επίσης ο χρήστης μπορεί να επεξεργαστεί και το Field of View.



Εικόνα 4.5.3 : Το τελικό αποτέλεσμα του FirstPersonController με την κάμερα.

4.6 ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ LEVEL

Αφού ο χρήστης έχει δημιουργήσει τον βασικό χαρακτήρα του παιχνιδιού (Player), έχει δημιουργήσει το έδαφος, την κάμερα και όλες τις λοιπές βασικές λειτουργίες για το παιχνίδι, ήρθε η στιγμή να δημιουργήσει και τη Level του. Η δημιουργία του Level είναι αρκετά χρονοβόρα διαδικασία, γιατί ο χρήστης πρέπει πρώτα να σχεδιάσει την σκηνή και μετά να υλοποιήσει τα κατάλληλα Objects και Scripts για να λειτουργήσει σωστά. Ένας απ΄τους γρηγορότερους τρόπους για να δημιουργηθεί η σκηνή, είναι η δημιουργία των απαραίτητων Objects. Εφόσον ο χρήστης δημιουργήσει τα αντικείμενα (Objects) που χρειάζεται, μπορεί να τα κάνει αντιγραφή και επικόλληση στο HierrarchyPanel και να τα τοποθετεί γρήγορα στη σκηνή του όποτε εκείνος το θέλει. Για να μπορέσει το παιχνίδι να λειτουργεί ομαλά, κάποια αντικείμενα θα συνοδεύονται απ' τα απαραίτητα Scripts.



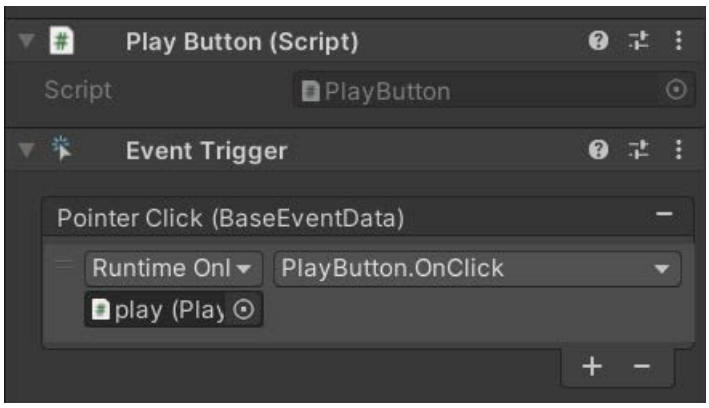
Εικόνα 4.6.1 : Μέρος της σκηνής του παιχνιδιού.

4.7 ΔΗΜΙΟΥΡΓΙΑ MAIN MENU

Στα βιντεοπαιχνίδια, το μενού αναφέρεται στην διεπαφή του χρήστη που επιτρέπει στους παίχτες να αλληλοεπιδρούν με διάφορες επιλογές και ρυθμίσεις του παιχνιδιού. Συνήθως περιλαμβάνει κουμπί για την έναρξη του παιχνιδιού, την αλλαγή ρυθμίσεων όπως την ανάλυση οθόνης του παιχνιδιού, την έξοδο απ' αυτό και άλλες λοιπές λειτουργίες. Το μενού είναι κρίσιμο για την παροχή μιας ομαλής και ευχάριστης εμπειρίας στον χρήστη, καθώς του επιτρέπει να προσαρμόσει το παιχνίδι σύμφωνα με τις προτιμήσεις και ανάγκες του.

Εικόνα 4.7.1 : Το κύριο μενού του παιχνιδιού.

Για να φτάσει όμως σε αυτό το αποτέλεσμα ο χρήστης, χρειάζεται να προβεί σε κάποιες απαραίτητες ενέργειες. Αρχικά θα χρειαστεί να δημιουργήσει μια νέα άδεια σκηνή στην οποία αφού εισάγει τα 3D μοντέλα του και ρυθμίσει τον φωτισμό της σκηνής θα προσθέσει μια κάμερα. Αμέσως μετά θα μεταβεί στο GameObject > UI > Canvas για να δημιουργήσει έναν νέο καμβά. Στη συνέχεια θα χρειαστεί να δημιουργήσει τα κουμπιά ή να δημιουργήσει Text που θα κάνει την δουλειά τους και θα τα ονομάσει εκείνος ανάλογα με τις επιλογές που θέλει να δώσει στον παίχτη. Αφού ολοκληρώσει και αυτή τη διαδικασία ο χρήστης, θα χρειαστεί να κάνει αυτά τα κουμπιά λειτουργικά. Για να το καταφέρει αυτό, θα προσθέσει ένα OnClick event στο κουμπί που ξεκινά το παιχνίδι και θα το προσθέσει με τη μέθοδο που ξεκινά αυτό. Την ίδια διαδικασία θα ακολουθήσει για κάθε κουμπί που έχει δημιουργήσει. Τέλος για να ολοκληρωθούν οι ενέργειες που απαιτεί ο χρήστης, θα χρειαστεί να γράψει και τα ανάλογα script ετσι ώστε να εκτελούνται μόλις πατήσει το κάθε κουμπί.



Εικόνα 4.7.2 : Προσθήκη OnClick event στο κουμπί έναρξης του παιχνιδιού.

4.8 ΔΗΜΙΟΥΡΓΙΑ ΕΧΘΡΙΚΟΥ AI

Το εχθρικό AI στα video games αποτελεί πολλές φορές την καρδιά των εχθρικών χαρακτήρων. Μέσω αυτού του σχεδιασμένου κώδικα, καθορίζετε η συμπεριφορά τους μετατρέποντας τους από απλούς κινούμενους στόχους σε πραγματικούς αντιπάλους και ο σκοπός τους είναι να θέσουν προκλήσεις στον παίχτη και να τον κάνουν να σκεφτεί στρατηγικές για να περάσουν το εμπόδιο τους. Σκοπός αυτών των AI είναι να δοκιμάσουν τις ικανότητες του παίχτη δημιουργώντας του εμπόδια κατά τη διάρκεια του παιχνιδιού. Ένα εχθρικό AI μπορεί να περιλαμβάνει μια πληθώρα συμπεριφορών όπως αλλαγές ταχύτητας, επιθέσεις στον παίχτη καθώς και άλλες ενέργειες που το κάνουν να φαίνεται έξυπνο και ρεαλιστικό. Κάθε εχθρικό AI ενεργεί με τρόπο που αναλογεί στον χαρακτήρα του και στο περιβάλλον του εκάστοτε παιχνιδιού, συμβάλλοντας έτσι στη δημιουργία μιας δυναμικής εμπειρίας για τον παίχτη. Το script που έχει δημιουργηθεί για το εχθρικό AI στο παιχίδι The Jellyfish Hunter, είναι βασισμένο σε εκείνο του Slenderman, κάνοντας τον εχθρό απρόβλεπτο και επιθετικό ανάλογα με την πρόοδο του παίχτη στο παιχνίδι. Ο κώδικας επιτρέπει στον εχθρό να παρακολουθεί και να ακολουθεί τον παίχτη, ενώ παράλληλα έχει την δυνατότητα να τηλεμεταφέρεται σε προκαθορισμένα σημεία στον χάρτη γύρω απ' τον παίχτη. Ακόμα, το script περιλαμβάνει μεταβλητές για την καθυστέρηση και την αναμονή τηλεμεταφοράς του εχθρού κάνοντας τον απρόβλεπτο. Η θέση του παίχτη ελέγχεται συνέχεια απ' το εχθρικό AI προσαρμόζοντας έτσι την δική του θέση και κατεύθυνση του ανάλογα. Όταν ισχύουν όλες οι προϋποθέσεις για τηλεμεταφορά, ο εχθρός επιλέγει ένα τυχαίο σημείο κοντά στον παίχτη και μετακινείται εκεί. Όσο περισσότερο φτάνει προς το τέλος του παιχνιδιού ο παίχτης, τόσο αυξάνονται οι τηλεμεταφορές και η απόσταση του εχθρού απ τον παίχτη. Έχει χρησιμοποιηθεί NavMeshAgent για την ομαλή κίνηση του εχθρού.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class SlendermanAI : MonoBehaviour
{
    public float baseTeleportDistance = 50f;
    public float baseMinTeleportDistance = 10f;
    public float teleportDelay = 5f;
    public float teleportCooldown = 5f;
    public Transform[] teleportPoints;

    private Transform player;
    private NavMeshAgent navMeshAgent;
    private float teleportTimer = 0f;
    private bool canTeleport = true;
    private NavMeshAgent agent;

    public static SlendermanAI Instance { get; private set; }

    private void Awake()
    {
        if (Instance != null && Instance != this)
        {
            gameObject.SetActive(false);
        }
        else
        {
            Instance = this;
        }
    }

    private void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").transform;
        navMeshAgent = GetComponent<NavMeshAgent>();
        agent = GetComponent<NavMeshAgent>();
    }

    private void Update()
    {
        Debug.DrawRay(agent.transform.position, Vector3.forward * 10f, Color.red);
        Debug.Log("canTeleport flag value: " + canTeleport);
        agent.nextPosition = transform.position;
        Vector3 targetPos = player.transform.position;
        targetPos.y = transform.position.y;
        transform.LookAt(targetPos);

```

Εικόνα 4.8.1 : Πρώτο μέρος του κώδικα του εχθρικού AI.

```
        teleportTimer += Time.deltaTime;

        if (teleportTimer >= teleportDelay && canTeleport)
        {
            Teleport();
            teleportTimer = 0f;
            canTeleport = false;
            StartCoroutine(StartTeleportCooldown());
        }


        Debug.Log("Slenderman position: " + transform.position);
    }

    private IEnumerator StartTeleportCooldown()
    {
        yield return new WaitForSeconds(teleportCooldown);
        canTeleport = true;
    }

    private void Teleport()
    {
        Vector3 teleportPosition;
        int attempts = 0;
        const int maxAttempts = 10;

        float adjustedMinTeleportDistance = baseMinTeleportDistance;
        float adjustedMaxTeleportDistance = baseTeleportDistance;

        do
        {
            int randomIndex = Random.Range(0, teleportPoints.Length);
            teleportPosition = teleportPoints[randomIndex].position;

            float distanceToPlayer = Vector3.Distance(player.position, teleportPosition);

            attempts++;

            if (distanceToPlayer >= adjustedMinTeleportDistance && distanceToPlayer <= adjustedMaxTeleportDistance)
            {
                break;
            }
        } while (attempts < maxAttempts);

        Debug.Log("Teleport position: " + teleportPosition);

        NavMeshHit hit;
        if (NavMesh.SamplePosition(teleportPosition, out hit, 1.0f, NavMesh.AllAreas))
        {
            transform.position = hit.position;
        }
    }
}
```

Εικόνα 4.8.2 : Δεύτερο μέρος του κώδικα του εχθρικού AI

4.9 ΜΗΧΑΝΙΣΜΟΙ (MECHANICS)

Οι μηχανισμοί στα βιντεοπαιχνίδια είναι οι κανόνες και τα συστήματα που καθορίζουν πως παίζεται το παιχνίδι. Περιλαμβάνουν όλα όσα μπορεί να κάνει ο παίχτης, όπως η κίνηση του χαρακτήρα, η μάχη με κάποιον εχθρό, η επίλυση γρίφων κ.α. Οι μηχανισμοί αυτοί κάνουν το παιχνίδι διασκεδαστικό και ενδιαφέρον, δίνοντας στόχους και προκλήσεις στον παίχτη. Αν είναι καλά σχεδιασμένοι, κάνουν το παιχνίδι ευχάριστο, ενώ αν είναι κακοί, μπορεί να κάνουν το παιχνίδι βαρετό ή ακόμα και απογοητευτικό. Στο The Jellyfish Hunter, έχουν χρησιμοποιηθεί ο μηχανισμός κίνησης

του παίχτη, η επιλογή να μπορεί να μαζέψει αντικείμενα και η αντιμετώπιση του εχθρού.

ΧΡΗΣΙΜΑ ΣΥΜΠΕΡΑΣΜΑΤΑ

Η διαδικασία δημιουργίας του παιχνιδιού για αυτή την πτυχιακή εργασία ήταν εξαιρετικά χρήσιμη, προσφέροντας πολλές νέες γνώσεις. Η μεγάλη κοινότητα χρηστών των εργαλείων που χρησιμοποιήθηκαν (Blender & Unity) ήταν πολύτιμη, καθώς συχνά παρείχαν λύσεις σε διάφορα προβλήματα μέσω των σχολίων τους. Επιπλέον, πολλά βίντεο χρησιμοποιήθηκαν για την επίλυση ζητημάτων κατά την ανάπτυξη του παιχνιδιού.

Οι γνώσεις προγραμματισμού που υπήρχαν συνδυάστηκαν με νέες πληροφορίες, οδηγώντας σε φρέσκες ιδέες και δεξιότητες. Κατά τη διάρκεια της εργασίας, μέσω της βιβλιογραφικής έρευνας, αποκτήθηκαν επιπλέον γνώσεις και ιδέες για μελλοντικές καινοτομίες.

ΠΑΡΑΡΤΗΜΑ 1

Στο παράρτημα αυτό θα παρουσιαστούν κάποια Script τα οποία χρησιμοποιήθηκαν για την υλοποίηση του Video Game. Είναι γραμμένα σε γλώσσα C# στο Notepad++.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class FishCounter : MonoBehaviour
{
    public int totalFish = 5;
    private int fishCaught = 0;
    public Text countText;
    public GameObject jellyfishImage;
    public GameObject slenderman;
    private bool isSlendermanActivated = false;

    private void Start()
    {
        UpdateFishCount();
    }

    private void UpdateFishCount()
    {
        countText.text = "Fish caught: " + fishCaught + "/" + totalFish;

        if (fishCaught == totalFish)
        {
            // All fish have been caught
            Debug.Log("You caught all the fish!");
            Time.timeScale = 0; // pause the game
            jellyfishImage.SetActive(true); // show the image
            Invoke("GoToMenuScene", 3f); // go to menu scene after 3 seconds
        }
    }

    public void FishCaught()
    {
        fishCaught++;
        UpdateFishCount();

        if (fishCaught == 1)
        {
            // Player picked up first jellyfish, activate Slenderman game object
            slenderman.SetActive(true);
            isSlendermanActivated = true;
        }
    }

    public int GetFishCaught()
    {
        return fishCaught;
    }

    private void GoToMenuScene()
    {
        Time.timeScale = 1; // unpause the game
        SceneManager.LoadScene("MENU"); // replace "MainMenu" with the name of your menu scene
    }
}
```

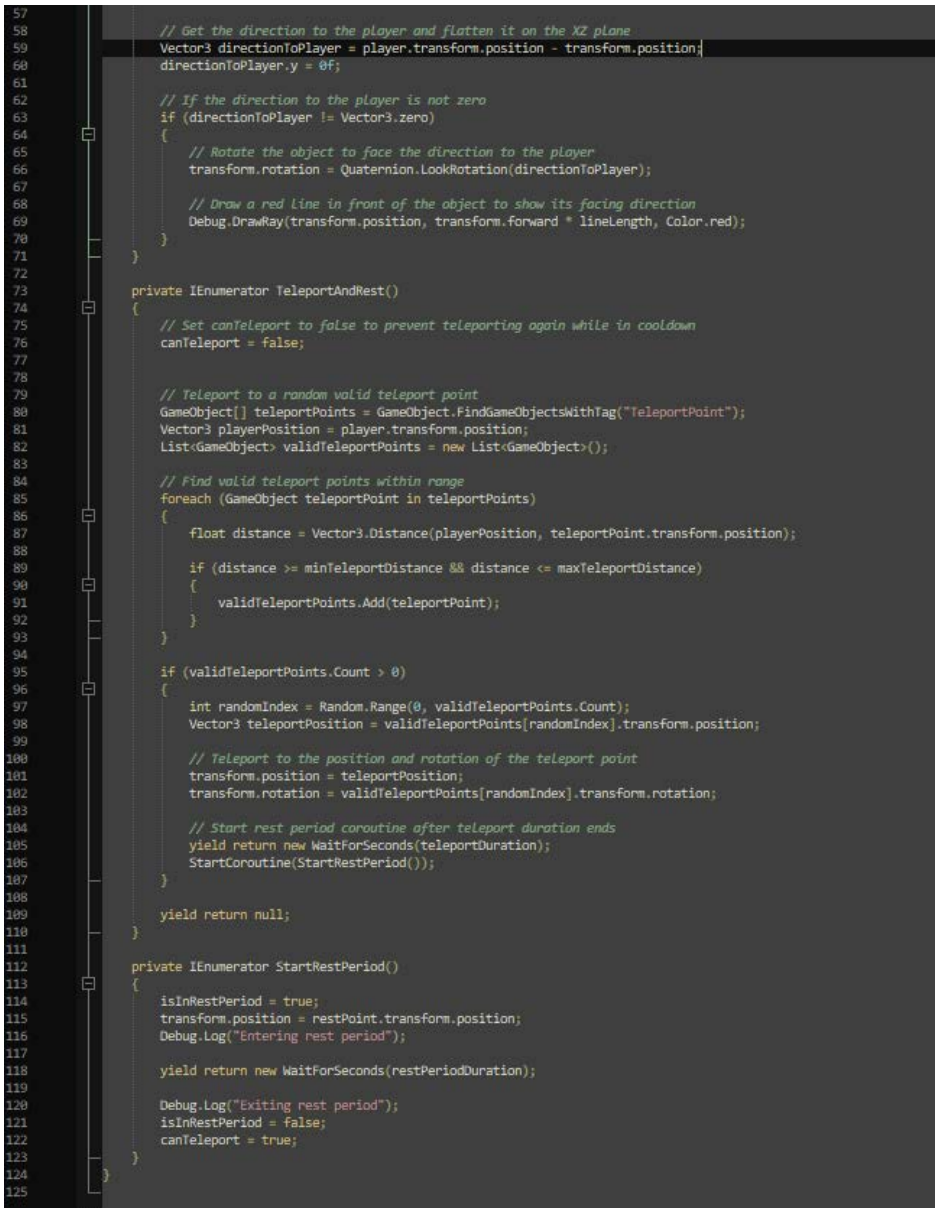
Εικόνα 0.1 : Script καταμέτρησης μεδουσών.

```
using UnityEngine;
using UnityEngine.UI;

public class ItemPickup : MonoBehaviour
{
    public float pickupDistance = 2f; // Distance from which the player can pick up the item
    public GameObject replacementItem; // GameObject to enable when this item is destroyed
    public GameObject[] droppedItems; // Array of GameObjects to drop when this item is picked up

    public Text pickupText; // The UI Text object for the pick up message

    private bool hasPickedUp = false;
    private bool isDisplayingMessage = false;

    private void Update()
    {
        if (!hasPickedUp)
        {
            // Check if the player presses the "E" key
            if (Input.GetKeyDown(KeyCode.E))
            {
                // Check if the player is looking at the item
                RaycastHit hit;
                if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, pickupDistance))
                {
                    if (hit.collider.gameObject == gameObject)
                    {
                        // Enable the replacement item
                        if (replacementItem != null)
                        {
                            replacementItem.SetActive(true);
                        }

                        // Drop the other items
                        foreach (GameObject item in droppedItems)
                        {
                            item.SetActive(true);
                        }

                        // Set the flag to prevent multiple pickups
                        hasPickedUp = true;

                        // Disable this item
                        gameObject.SetActive(false);
                    }
                }
            }

            // Check if the player is looking at the item and show the pick up message
            RaycastHit hitInfo;
            if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hitInfo, pickupDistance))
            {
                if (hitInfo.collider.gameObject == gameObject && !isDisplayingMessage)
                {
                    pickupText.gameObject.SetActive(true);
                    isDisplayingMessage = true;
                }
            }
            else if (isDisplayingMessage)
            {
                pickupText.gameObject.SetActive(false);
                isDisplayingMessage = false;
            }
        }
    }
}
```

Εικόνα 0.2 : Script PickUp αντικειμένων.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class slenderainew : MonoBehaviour
{
    [SerializeField] private float maxTeleportDistance = 150f;
    [SerializeField] private float minTeleportDistance = 10f;
    [SerializeField] private float teleportCooldown = 10f;
    [SerializeField] private float teleportDuration = 6f;
    [SerializeField] private float restPeriodDuration = 10f;
    [SerializeField] private GameObject restPoint;
    [SerializeField] private float lineLength = 5f;

    private bool canTeleport = true;
    private bool isInRestPeriod = false;
    private GameObject player;
    private FishCounter fishCounter;

        private void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        fishCounter = FindObjectOfType<FishCounter>();
    }

    private void Update()
    {
        if (canTeleport)
        {
            StartCoroutine(TeleportAndRest());
        }
            if (fishCounter.GetFishCaught() == 2)
        {
            maxTeleportDistance = 120f;
            minTeleportDistance = 40f;
        }
        else if (fishCounter.GetFishCaught() == 3)
        {
            maxTeleportDistance = 110f;
            minTeleportDistance = 35f;
        }
        else if (fishCounter.GetFishCaught() == 4)
        {
            maxTeleportDistance = 100f;
            minTeleportDistance = 30f;
        }
        else if (fishCounter.GetFishCaught() == 5)
        {
            maxTeleportDistance = 90f;
            minTeleportDistance = 25f;
        }
        else if (fishCounter.GetFishCaught() >= 6)
        {
            maxTeleportDistance = 80f;
            minTeleportDistance = 10f;
        }
```

Εικόνα 0.3 : Script εχθρού (1).

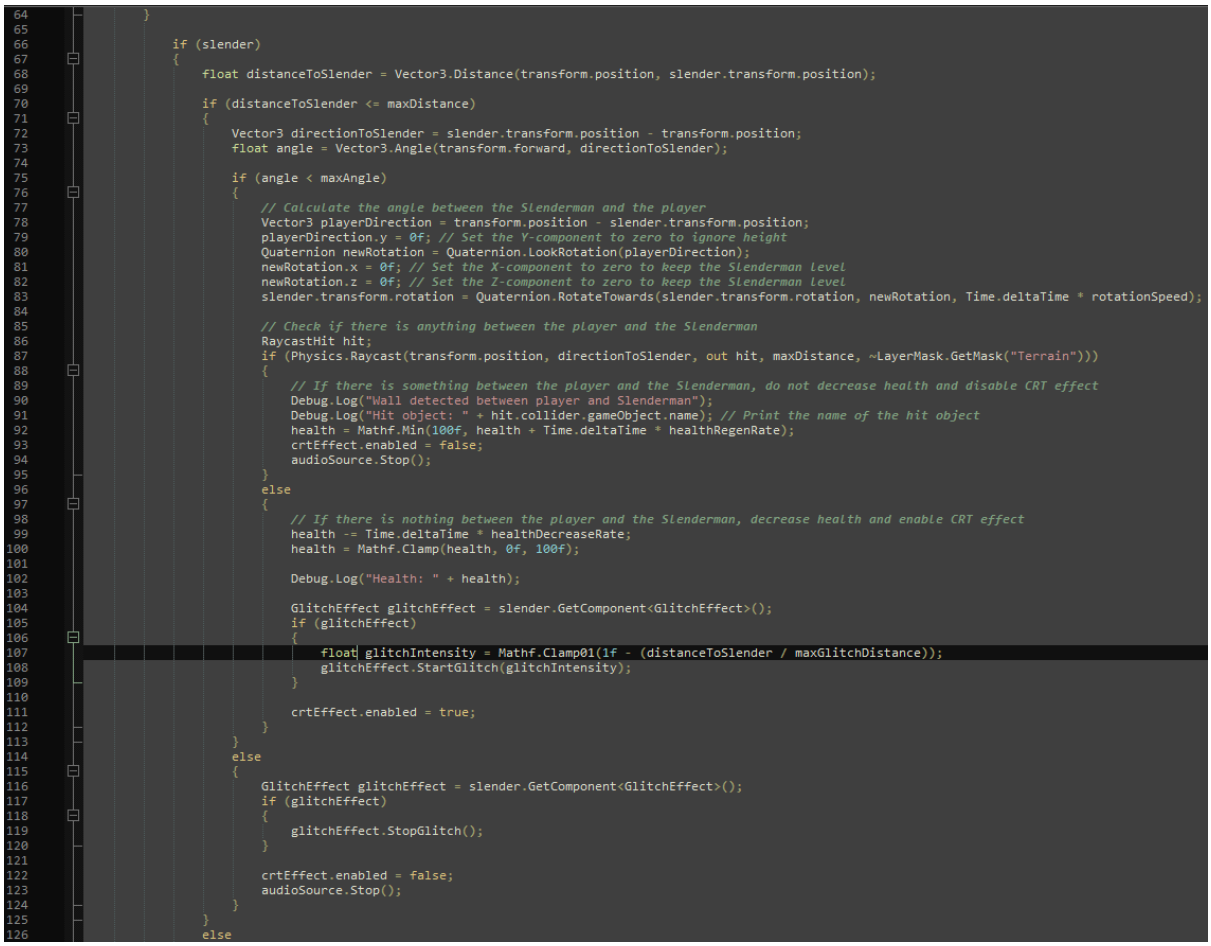


```
        // Get the direction to the player and flatten it on the XZ plane
        Vector3 directionToPlayer = player.transform.position - transform.position;
        directionToPlayer.y = 0f;

        // If the direction to the player is not zero
        if (directionToPlayer != Vector3.zero)
        {
            // Rotate the object to face the direction to the player
            transform.rotation = Quaternion.LookRotation(directionToPlayer);

            // Draw a red line in front of the object to show its facing direction
            Debug.DrawRay(transform.position, transform.forward * lineLength, Color.red);
        }
    }

    private IEnumerator TeleportAndRest()
    {
        // Set canTeleport to false to prevent teleporting again while in cooldown
        canTeleport = false;


        // Teleport to a random valid teleport point
        GameObject[] teleportPoints = GameObject.FindGameObjectsWithTag("TeleportPoint");
        Vector3 playerPosition = player.transform.position;
        List<GameObject> validTeleportPoints = new List<GameObject>();

        // Find valid teleport points within range
        foreach (GameObject teleportPoint in teleportPoints)
        {
            float distance = Vector3.Distance(playerPosition, teleportPoint.transform.position);

            if (distance >= minTeleportDistance && distance <= maxTeleportDistance)
            {
                validTeleportPoints.Add(teleportPoint);
            }
        }

        if (validTeleportPoints.Count > 0)
        {
            int randomIndex = Random.Range(0, validTeleportPoints.Count);
            Vector3 teleportPosition = validTeleportPoints[randomIndex].transform.position;

            // Teleport to the position and rotation of the teleport point
            transform.position = teleportPosition;
            transform.rotation = validTeleportPoints[randomIndex].transform.rotation;

            // Start rest period coroutine after teleport duration ends
            yield return new WaitForSeconds(teleportDuration);
            StartCoroutine(StartRestPeriod());
        }

        yield return null;
    }

    private IEnumerator StartRestPeriod()
    {
        isInRestPeriod = true;
        transform.position = restPoint.transform.position;
        Debug.Log("Entering rest period");

        yield return new WaitForSeconds(restPeriodDuration);

        Debug.Log("Exiting rest period");
        isInRestPeriod = false;
        canTeleport = true;
    }
}
```

Εικόνα 0.4 : Script εχθρού (2).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Player : MonoBehaviour
{
    [SerializeField] public float maxGlitchDistance = 80f;
    [SerializeField] public float health = 100f;
    [SerializeField] public float healthRegenRate = 10f;
    [SerializeField] public float healthDecreaseRate = 20f;
    [SerializeField] private float rotationSpeed = 5f;
    [SerializeField] private float maxDistance = 80f;
    [SerializeField] private float maxAngle = 20f;
    //DOULEVEI!!!

    private bool isLookingAtSlender = false;

    private GameObject slender;
    public Camera mainCamera;
    public Camera secondCamera;

    public ShaderEffect_CRT crtEffect;
    public ShaderEffect_Scanner scannerEffect;
    public AudioSource audioSource; // Reference to the AudioSource component
    public AudioClip hurtSound;
    public float audioFadeOutTime = 0.1f;
    private float audioFadeOutTimer = 0f;

    public float GetPlayerHealth()
    {
        return health;
    }

    private void Update()
    {
        Debug.Log("Update function called.");
        // Check health and enable/disable CRT and Scanner effects accordingly
        if (health < 100f && crtEffect)
        {
            crtEffect.enabled = true;
            if (!audioSource.isPlaying) {
                audioSource.clip = hurtSound;
                audioSource.Play();
            }
        }
        else if (health >= 100f && crtEffect)
        {
            crtEffect.enabled = false;
            audioSource.Stop();
        }
        if (health < 100f && scannerEffect)
        {
            scannerEffect.enabled = true;
        }
        else if (health >= 100f && scannerEffect)
        {
            scannerEffect.enabled = false;
        }

        if (!slender)
        {
            slender = GameObject.FindGameObjectWithTag("Slenderman");
```

Εικόνα 0.5 : Script διαχείρησης ζωής Player (1).



```
64          }
65
66          if (slender)
67          {
68              float distanceToSlender = Vector3.Distance(transform.position, slender.transform.position);
69
70              if (distanceToSlender <= maxDistance)
71              {
72                  Vector3 directionToSlender = slender.transform.position - transform.position;
73                  float angle = Vector3.Angle(transform.forward, directionToSlender);
74
75                  if (angle < maxAngle)
76                  {
77                      // Calculate the angle between the Slenderman and the player
78                      Vector3 playerDirection = transform.position - slender.transform.position;
79                      playerDirection.y = 0f; // Set the Y-component to zero to ignore height
80                      Quaternion newRotation = Quaternion.LookRotation(playerDirection);
81                      newRotation.x = 0f; // Set the X-component to zero to keep the Slenderman level
82                      newRotation.z = 0f; // Set the Z-component to zero to keep the Slenderman level
83                      slender.transform.rotation = Quaternion.RotateTowards(slender.transform.rotation, newRotation, Time.deltaTime * rotationSpeed);
84
85                      // Check if there is anything between the player and the Slenderman
86                      RaycastHit hit;
87                      if (Physics.Raycast(transform.position, directionToSlender, out hit, maxDistance, ~LayerMask.GetMask("Terrain")))
88                      {
89                          // If there is something between the player and the Slenderman, do not decrease health and disable CRT effect
90                          Debug.Log("Wall detected between player and Slenderman");
91                          Debug.Log("Hit object: " + hit.collider.gameObject.name); // Print the name of the hit object
92                          health = Mathf.Min(100f, health + Time.deltaTime * healthRegenRate);
93                          crtEffect.enabled = false;
94                          audioSource.Stop();
95                      }
96                      else
97                      {
98                          // If there is nothing between the player and the Slenderman, decrease health and enable CRT effect
99                          health -= Time.deltaTime * healthDecreaseRate;
100                         health = Mathf.Clamp(health, 0f, 100f);
101
102                         Debug.Log("Health: " + health);
103
104                         GlitchEffect glitchEffect = slender.GetComponent<GlitchEffect>();
105                         if (glitchEffect)
106                         {
107                             float glitchIntensity = Mathf.Clamp01(1f - (distanceToSlender / maxGlitchDistance));
108                             glitchEffect.StartGlitch(glitchIntensity);
109                         }
110
111                         crtEffect.enabled = true;
112                     }
113                 }
114                 else
115                 {
116                     GlitchEffect glitchEffect = slender.GetComponent<GlitchEffect>();
117                     if (glitchEffect)
118                     {
119                         glitchEffect.StopGlitch();
120                     }
121
122                     crtEffect.enabled = false;
123                     audioSource.Stop();
124                 }
125             }
126             else
```

Εικόνα 0.6 : Script διαχείρησης ζωής Player (2).

```
                else
                {
                    GlitchEffect glitchEffect = slender.GetComponent<GlitchEffect>();
                    if (glitchEffect)
                    {
                        glitchEffect.StopGlitch();
                    }

                    crtEffect.enabled = false;
                    audioSource.Stop();
                }
            }

        // Check if player is looking at Slenderman and decrease/regenerate health accordingly
        if (isLookingAtSlender)
        {
            health -= Time.deltaTime * healthDecreaseRate;
            health = Mathf.Clamp(health, 0f, 100f);
        }
        else
        {
            health = Mathf.Min(100f, health + Time.deltaTime * healthRegenRate);
        }

        // Check if player is dead
        if (health <= 0.1f)
        {
            mainCamera.gameObject.SetActive(false);
            secondCamera.gameObject.SetActive(true);
            Invoke("LoadMainMenuScene", 6f);
            Debug.Log("You died!");
            Time.timeScale = 0; // Pause the game

            // TODO: Add code to display

        }
    }

    private void LoadMainMenuScene()
    {
        SceneManager.LoadScene("MENU");
    }


}

```

Εικόνα 0.7 : Script διαχείρησης ζωής Player (3).

```
using UnityEngine;

public class MiniMapFollow : MonoBehaviour
{
    public Transform playerTransform; //player transform
    public float height = 50f; // fixed height above player

    void LateUpdate()
    {
        //follow the player position but keep the Y position fixed
        Vector3 newPosition = playerTransform.position;
        newPosition.y = height;
        transform.position = newPosition;

        //rotate the camera to players rotation
        transform.rotation = Quaternion.Euler(90, playerTransform.eulerAngles.y, 0);
    }
}

```

Εικόνα 0.8 : Script του MiniMap το οποίο ακολουθεί τον παίχτη & rotates.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] P. Owen, «What Is A Video Game? A Short Explainer,» 9th March 2016. [Ηλεκτρονικό]. Available: https://www.thewrap.com/what-is-a-video-game-a-short-explainer/.

[2] M. S. M. M. a. C. Z. A Onion, «Video Game History,» 17th October 2022. [Ηλεκτρονικό]. Available: https://www.history.com/topics/inventions/history-of-video-games.

[3] T. Juszczak, «What is Unity 3D and why you should use it!,» 28th November 2023. [Ηλεκτρονικό]. Available: https://prographers.com/blog/what-is-unity-3d-and-why-you-should-use-it.

[4] A. C. Marie Dealessandri, «What is the best game engine: is GameMaker right for you?,» 28th February 2024. [Ηλεκτρονικό]. Available: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-gamemaker-the-right-game-engine-for-you.

[5] Thinkwik, «CryEngine vs Unreal vs Unity: Select the Best Game Engine,» 20 April 2018. [Ηλεκτρονικό]. Available: https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e.

ΧΡΗΣΙΜΑ LINKS

https://discussions.unity.com/questions/

https://stackoverflow.com

https://www.youtube.com/@unity/featured

https://forum.unity.com

https://assetstore.unity.com

https://sketchfab.com

https://www.cgtrader.com