Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΙΑ ΠΡΟΤΥΠΟΥ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΤΗΝ ΧΡΗΣΗ ΠΛΑΤΦΟΡΜΑΣ UNITY ΚΑΙ BLENDER.

ΚΩΝΣΤΑΝΤΊΝΟΣ ΠΑΝΑΓΟΠΟΥΛΟΣ 3080

ΚΡΙΣΤΙ ΚΕΛΕΜΕΝΙ 3015

ΕΙΣΗΓΗΤΗΣ ΚΑΘΗΓΗΤΗΣ: ΓΕΩΡΓΙΟΣ ΑΣΗΜΑΚΟΠΟΥΛΟΣ

ΠΑΤΡΑ 2022

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Στην εργασία αυτή θα μελετηθεί η ιστορική εξέλιξη στις πλατφόρμες παιχνιδιών καθώς και ο τρόπος παραγωγής των διαφόρων μοντέλων που τα παιχνίδια χρησιμοποιούν. Παράλληλα θα εξερευνηθεί αναλυτικά η μεθοδολογία δημιουργίας παιχνιδιών με τη χρήση της πλατφόρμας Unity, καθώς και η δημιουργία μοντέλων με την χρήση του προγράμματος Blender. Στα πλαίσια της εργασίας θα αναπτυχθεί πρότυπο παιχνίδι με την χρήση της πλατφόρμας Unity.

Για την εκπόνηση της εργασίας, θα εστιάσουμε στα τεχνικά χαρακτηριστικά αυτής. Θα αναφέρουμε τυχόν προβλήματα που αντιμετωπίσαμε και την επίλυση τους. Χρησιμοποιούμε την δομή της εργασίας έτσι ώστε να μπορεί να ασχοληθεί κάποιος για πρώτη φορά και να καταλάβει τα βασικά χαρακτηριστικά του θέματος, καθώς και οι δύο συμμετέχοντες της εργασίας αυτής, ασχοληθήκαμε πρώτη φορά με το θέμα αυτό.

Η εργασία αυτή περιλαμβάνει από έναν οδηγό για το Blender και το Unity.Στο Blender παρουσιάζουμε τις αρχές λειτουργίας της εφαρμογής, μια γενική εικόνα για το πως μπορούμε να δημιουργήσουμε ένα μοντέλο και τον τρόπο δημιουργίας animation του μοντέλου. Στην εφαρμογή Unity,παρουσιάζουμε τα ποιο βασικά χαρακτηριστικά της, καθώς και τα πολύ βασικά εξαρτήματα που θα χρειαστούν για την ανάπτυξη του παιχνιδιού. Αυτοί οι οδηγοί είναι στα πλαίσια του ατόμου που ασχολείται πρώτη φορά.

Στο τελευταίο ασχοληθήκαμε με πράγματα όσον αφορά την δημιουργία του παιχνιδιού. Η μετακίνηση του χαρακτήρα μας είναι ένα εισαγωγικό κομμάτι στον κώδικα που χρειάζεται για να αναπτυχθεί Χρησιμοποιήσαμε κώδικα επίσης, για την αλληλεπίδραση αντικειμένων. Έπειτα θα δημιουργήσουμε τα UI του παιχνιδιού και τον τρόπο που λειτουργούν in game.Τέλος, με τα εργαλεία UI στο Unity.θα δημιουργήσουμε ένα αρχικό menu παιχνιδιού, ένα menu παύσης και ένα inventory όπου θα αποθηκεύει αντικείμενα.

ABSTRACT

On this project the historical evolution of video game platforms will be studied as well as the way of production of the various models that the games use. At the same time, the creation of video games methodology will be explored in detail using the unity platform, as well as creating models with the help of blender. As part of this project, we will develop a game, based on the unity platform.

For the preparation of the project, we will focus on its technical characteristics. We will report any issues we encountered and how they were resolved. We use the structure of the project so that someone can deal with it for the first time and understand the main features of the topic, cause before this project we were inexperienced with these platforms.

This work includes a guide for Blender and Unity. In Blender we present the principles of operation of the application, an overview of how we can create a model and how to animate the model. In the Unity application, we present its most basic features, as well as the very basic components that will be needed to develop the game. These guides are for users that are in experienced with the platform.

In the last one we dealt with things regarding the creation of the game. Moving our character is an introductory piece of code needed to develop. We also used code to interact with objects. Then we will create the game UIs and how they work in game. Finally, with the UI tools in Unity, we will create an initial game menu, a pause menu and an inventory where it will store items.

1 GAME DEVELOPMENT

1.1 ΕΙΣΑΓΩΓΗ ΣΤΟ VIDEO GAME DEVELOPMENT

Video game development είναι η διαδικασία μέσω της οποίας αναπτύσσεται ένα παιχνίδι. Η ανάπτυξη ενός project παιχνιδιού επιτυγχάνεται από τουλάχιστον ένα άτομο και μπορεί να φτάσει μέχρι ένα σύνολο ατόμων συμβάλλοντας στην προσπάθεια ανάπτυξης του. Η διαδικασία της ανάπτυξης ενός παιχνιδιού χρηματοδοτείται από έναν εκδότη. Ανάλογα με το εύρος των απαιτήσεων από τον εκδότη το project αυτό μπορεί να είναι πολύ χρονοβόρο. Η βιομηχανία παιχνιδιών τα τελευταία χρόνια έχει γνωρίσει τεράστια ανάπτυξη και αυτό οφείλεται σε κάποιες πλατφόρμες ανάπτυξης παιχνιδιών όπως είναι η Unreal Engine και η Unity platform.

Εικόνα 1.1.1: Λογότυπο της εταιρείας Unity.

Το Unity είναι μια cross-platform game engine, αναπτύχθηκε από την Unity Technologies, ανακοινώθηκε και έγινε το release της, τον Ιούνιο του 2005 σε μια εκδήλωση της Apple, ως μιας αποκλειστικής πλατφόρμας για τα Mac. Αργότερα αυτό άλλαξε και επεκτάθηκε ώστε να παρέχει υποστήριξη σε υπολογιστές , κινητά και κονσόλες καθώς και πλατφόρμες εικονικής πραγματικότητας. Η πλατφόρμα unity είναι πολύ διάσημη στον χώρο του mobile game development και χρησιμοποιήθηκε σε παιχνίδια όπως το Pokémon go, Monument Valley , Call of duty mobile κ.α.. Η πλατφόρμα θεωρείται ότι είναι εύκολο στην χρήση σε αρχάριους προγραμματιστές .

Εικόνα 1.1.2: Λογότυπο πλατφόρμας Unreal Engine.

Εικόνα 1.1.3: Game development.

1.2 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Η ιστορία των βιντεοπαιχνιδιών άρχισε μεταξύ 1950 και 1960 καθώς επιστήμονες αρχίσαν να δημιουργούν απλά παιχνίδια και προσομοιώσεις στους μίνι υπολογιστές καθώς και στα μεγάλα συστήματα. Το Spacewar αναπτύχθηκε από έναν φοιτητή του MIT το 1962 , καθώς ήταν ένα από τα πρώτα παιχνίδια που αναπαράγονταν σε μια οθόνη. Η πρώτη κονσόλα παιχνιδιού ευρείας χρήσης λανσαρίστηκε στις αρχές του 1970 και ονομάζονταν «Magnavox Odyssey», και τα πρώτα μηχανήματα με κερματοδέκτη είναι το Computer Space και Pong.

Εικόνα 1.2.1: Μίνι υπολογιστής.

Εικόνα 1.2.2: Μεγάλο σύστημα.

Εικόνα 1.2.3: Κονσόλα Magnavox Odyssey.

Εικόνα 1.2.4: Pong και Computer Space.

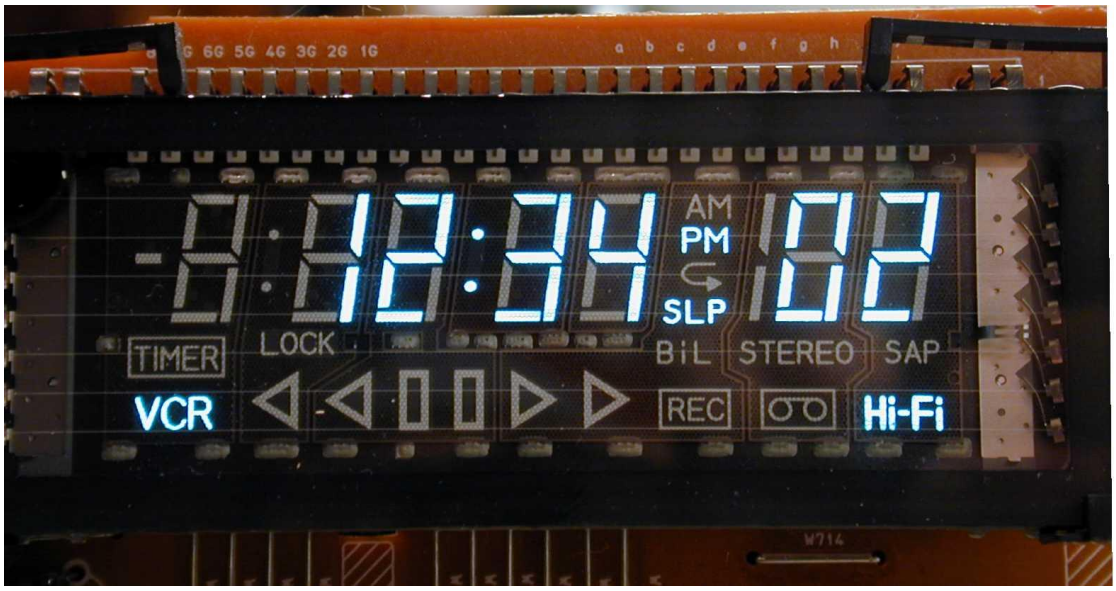
Το 1972 δημιουργήθηκε η πρώτη εταιρία βιντεοπαιχνιδιών από τον Nolan Bushnell και ονομάστηκε ATARI, ενώ αργότερα το 1979 δημιουργήθηκε η πρώτη ανεξάρτητη εταιρία έκδοσης και ανάπτυξης βιντεοπαιχνιδιών για τις κονσόλες ονόματι ACTIVISION. Ιδρύθηκε από τον Jim Levy και διάφορους πρώην προγραμματιστές της εταιρείας ATARI.

Εικόνα 1.2.5: Λογότυπο της εταιρείας ATARI.

Εικόνα 1.2.6: Λογότυπο της εταιρείας ACTIVISION.

Τα ηλεκτρονικά παιχνίδια χειρός χρησιμοποιούσαν όλα τα υπολογιστικά μέρη, καθώς και Leds(Light-emitting diode) ή VFD(Vacuum fluorescent display) τα οποία πρωτοεμφανίστηκαν στις αρχές του 1970. Οι οθόνες LCD έγιναν φθηνότερες σε προϊόντα κατανάλωσης στα μέσα του 1970 και αντικατέστησαν τα LED και VFD σε τέτοια παιχνίδια , χάρις στο μικρότερο κόστος κατανάλωσης ηλεκτρικής ενέργειας και στο μικρότερο μέγεθος. Κάποιες εταιρίες όπως οι Mattel Electronics και Bandai δημιούργησαν αμέτρητα παιχνίδια από το 1970 έως και τις αρχές του 1980.

Εικόνα 1.2.7: LED.



Εικόνα 1.2.8: VFD.

Εικόνα 1.2.9: Παιχνίδι από την εταιρεία Mattel electronics.

Εικόνα 1.2.10: Λογότυπο της εταιρείας BANDAI.

Κατά την μετάβαση στα οπτικά μέσα , η βιομηχανία εξ' ολοκλήρου είχε στραφεί στα real-time 3d γραφικά υπολογιστών σε όλα τα παιχνίδια κατά την διάρκεια του 1990. Υπήρχε ένα πλήθος από παιχνίδια arcade τα οποία χρησιμοποιούσαν wireframe vector graphics για την αναπαραγωγή 3D , όπως είναι το Battlezone,Tempest και το Star Wars. Ωστόσο το μεγάλο πρόβλημα στα γραφικά 3D ήταν ότι η απόδοση σε πραγματικό χρόνο απαιτούσε υπολογισμούς κινητής υποδιαστολής όπου μέχρι το 1990 , το μεγαλύτερο μέρος του υλικού των βιντεοπαιχνιδιών δεν ήταν κατάλληλο. Αντί αυτού πολλά παιχνίδια προσομοίωναν 3D εφέ με τη χρήση parallax rendering από διάφορα στρώματα φόντου η με άλλες διάφορες μεθόδους ώστε να επιτευχθεί η απεικόνιση γραφικών 3D μέσο του συστήματος 2D γνωστά ως και (2.5D Graphics).

Στις αρχές του 1990, η βελτίωση που έγινε στην τεχνολογία των μικροεπεξεργαστών , είχε ως αποτέλεσμα να φέρει 2 τεράστιες τεχνολογικές εξελίξεις, οι οποίες περιλάμβαναν την επαφή του κοινού με τα οπτικά μέσα, με την βοήθεια των CD-ROMs και ενός 3D polygonal graphic rendering. Αυτές οι δύο πτυχές , ενσωματώθηκαν εύκολα στους προσωπικούς υπολογιστές , δημιουργώντας μια αγορά για τις γραφικές κάρτες, συμπεριλαμβάνοντας της κονσόλα της Sony(PlayStation) στις ήδη υπάρχοντες κονσόλες. Στα τέλη του 1990 το ίντερνετ απέκτησε ευρεία καταναλωτική χρήση και αυτό είχε σαν αποτέλεσμα τα βιντεοπαιχνίδια να ενσωματώσουν στοιχεία του διαδικτύου.

Εικόνα 1.2.11: «Πρόγονος» των οπτικών μέσων, η δισκέτα. Έχει αντικατασταθεί από τα οπτικά μέσα (CD,DVD,USB και άλλα).

Εικόνα 1.2.12: CD. Η πλευρά που έκανε ανάγνωση και εγγραφή το σύστημα.

2 ΣΥΣΤΑΤΙΚΑ ΣΤΟΙΧΕΙΑ ΛΟΓΙΜΙΚΩΝ

2.1 UNITY GAME ENGINE

Μια Game Engine όπως είναι η Unity , είναι μια πλατφόρμα βασισμένη σε ένα ανεπτυγμένο πλαίσιο λογισμικού , σχεδιασμένη για την ανάπτυξη των video games. Οι πλατφόρμες αυτές συνήθως περιλαμβάνουν διάφορες βιβλιοθήκες καθώς και διαφορά βοηθητικά προγράμματα.

Οι προγραμματιστές χρησιμοποιούν τα Game Engines για να δημιουργήσουν κάποιο παιχνίδι για τις κονσόλες και για διάφορα είδη υπολογιστών. Συνήθως αυτά που προσφέρει μια τέτοια πλατφόρμα είναι κάποια rendering engine , γραφικά 2DS και γραφικά 3D, μηχανή για τα physics, ήχους, scripting, διάφορα animation, τεχνητή νοημοσύνη. Έτσι οι πλατφόρμες αυτές έχουν λύσει τα χέρια των προγραμματιστών και έχουν κάνει το κόστος για τις εταιρίες φθηνότερο καθώς χρησιμοποιούνται για διάφορα παιχνίδια ξανά και ξανά καθώς προσαρμόζονται εύκολα σε διάφορες απαιτήσεις

2.1.1 ΣΚΟΠΟΣ ΤΩΝ GAME ENGINE

Στην ουσία τα Game Engine παρέχουν μια γκάμα από εργαλεία οπτικής ανάπτυξης καθώς και επαναχρησιμοποιήσιμα στοιχεία λογισμικού. Αυτά τα εργαλεία παρέχονται σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης προκειμένου η ανάπτυξη των παιχνιδιών να γίνει όσο πιο απλή γίνεται, με μεγαλύτερη ταχύτητα, καθώς και με βάση την σωστή ανάλυση των δεδομένων που γίνεται από τις εταιρείες να προσφερθεί καλύτερη εμπειρία στο χρήστη. Ένα άλλο σημαντικό πλεονέκτημα είναι ότι αυτές οι πλατφόρμες παρέχουν την δημιουργία ενός παιχνιδιού σε διαφορετικά είδη κονσόλων.

2.2 BLENDER

Το Blender είναι ένα ελεύθερο λογισμικό που αρχικά αναπτύχθηκε από τα studio **“NeoGeo”** και **“Not a Number Technologies”**.Η ιδέα άρχισε να υλοποιείται από τον **Ton Roosendaal** και διαμοιραζόταν μέχρι το 2002 όπου η **“Not a Number Technologies”** χρεωκόπησε. Οι πιστωτές συμφώνησαν για την έκδοση άδειας το λογισμικού με αντίτιμο 100.000€. Ο **Ton Roosendaal** μάζεψε το ποσό από κάποιες δωρεές και από τον Σεπτέμβριο του 2002 το Blender διανέμεται με την άδεια GNU GPL(Γενική άδεια δημόσιας χρήσης GNU).

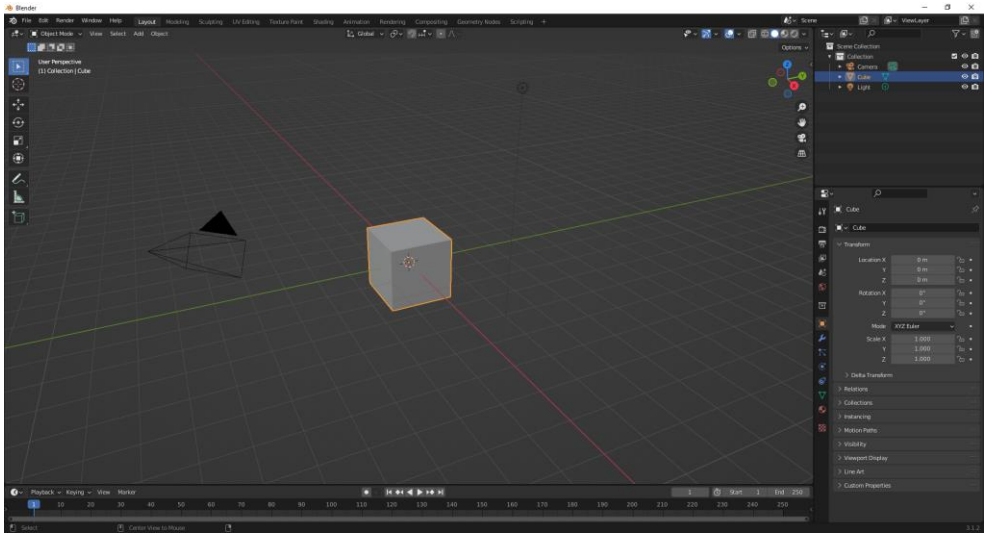
2.2.1 ΣΚΟΠΟΣ ΤΟΥ BLENDER

Σκοπός του Blender είναι να μας διευκολύνει να δημιουργήσουμε μοντέλα τα οποία χρειαζόμαστε, όπως είναι ένας χαρακτήρας ή ένα διακοσμητικό στοιχείο και να μπορούμε να το ενσωματώσουμε στο βιντεοπαιχνίδι μας.

3 ΓΝΩΡΙΜΙΑ ΜΕ ΤΟ BLENDER

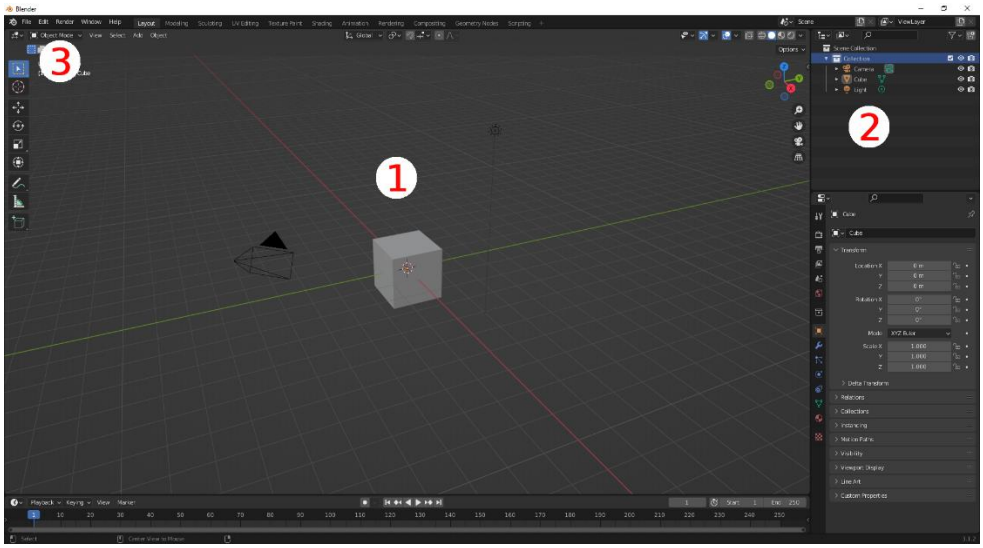
3.1 ΑΡΧΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΤΟΥ BLENDER

Προκειμένου να δημιουργήσουμε και να επεξεργαστούμε τα μοντέλα του παιχνιδιού εκτελούμε την εφαρμογή Blender.Ας γνωρίσουμε το περιβάλλον αυτής της εφαρμογής και τις δυνατότητες της.



Εικόνα 3.1.1: Αρχικό περιβάλλον της εφαρμογής Blender.

Παρατηρούμε ότι έχει πολλές επιλογές και θα προσπαθήσουμε να καλύψουμε τις βασικότερες επιλογές του.



Εικόνα 3.1.2: Ανάλυση ορισμένων panel και των modes.

1. **Scene**

Ουσιαστικά δεν είναι η τελική σκηνή όταν το κάνουμε render αλλά από αυτό το panel μπορούμε να δούμε τα αντικείμενα-μοντέλα που επεξεργαζόμαστε. Αυτό που γίνεται render είναι ότι βλέπει η κάμερα που έχουμε βάλει στο hierarchy panel.

2. **Layer hierarchy**

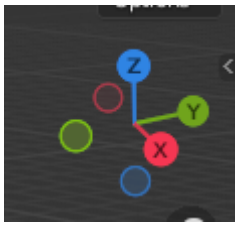
Αυτό το panel περιέχει την ιεραρχία των μοντέλων. Παρατηρούμε πως αρχικά το Blender δημιουργεί 3 αντικείμενα(Camera,Light,Cube) και τα ομαδοποιεί σε μια συλλογή(Collection).Η Camera χρησιμοποιείται σε περίπτωση που θέλουμε να κάνουμε render.Το Light είναι μια λάμπα όπου μπορούμε να την κατευθύνουμε στην rendered σκηνή μας για καλύτερο φωτισμό. Και τέλος το Cube είναι το μοντέλο μας όπου μπορούμε να το επεξεργαστούμε.

3. **Modes**

Σε αυτήν την επιλογή αφότου έχουμε επιλέξει το σωστό αντικείμενο(πχ το Cube) μπορούμε να μεταβούμε σε κάποια από τα modes που μας προσφέρει η εφαρμογή έτσι ώστε να το επεξεργαστούμε κατάλληλα. Κυρίως θα μας απασχολήσει το Object mode και το Edit Mode.

3.2 ΟΙ ΑΞΟΝΕΣ ΣΤΟ BLENDER

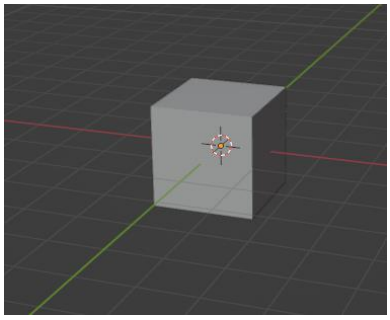
Εν συνεχεία στην αρχική μας οθόνη θα παρατηρήσουμε πως υπάρχουν κάποια επιπλέον στοιχεία. Αρχικά βλέπουμε στο επάνω δεξιό μέρος της σκηνής μας τους τρείς άξονες(X,Y,Z) που αντιπροσωπεύουν έναν τρισδιάστατο χώρο.



Εικόνα 3.2.1: Άξονες του Blender.

Ο άξονας X αντιπροσωπεύει το δεξιό και αριστερό μέρος του μοντέλου μας, ο άξονας Y το εμπρόσθιο και πίσω μέρος του μοντέλου μας και ο άξονας Z το επάνω και το κάτω μέρος του μοντέλου μας. Με αριστερό κλίκ πάνω στους άξονες στην παραπάνω εικόνα μεταβαίνετε σε οπτική γωνία κατά μήκος του άξονα που έχετε επιλέξει. Υπάρχουν και κάποιες συντομεύσεις από το πληκτρολόγιο(Numpad 3 για X, Numpad 1 για Y, Numpad 7 για Z), εάν

θέλουμε την αρνητική πλευρά του άξονα χρησιμοποιούμε το πλήκτρο Ctrl και το αντίστοιχο πλήκτρο στο Numpad.

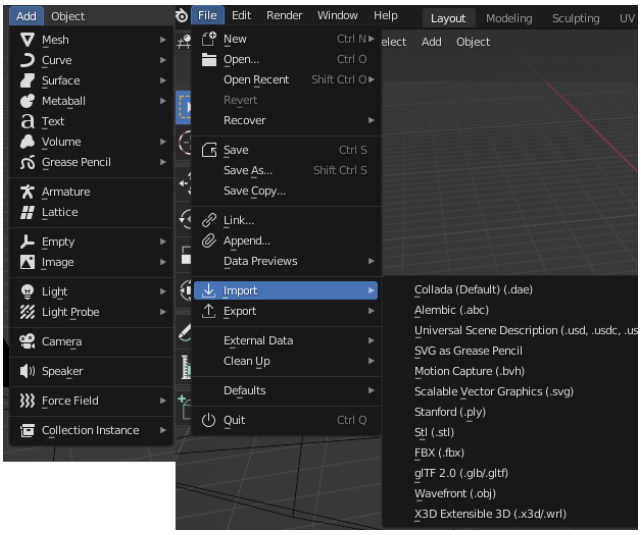


Εικόνα 3.2.2: Άξονες στο μοντέλο μας.

Τέλος το Blender φροντίζει οπτικά για να ξεχωρίσουμε τις πλευρές του μοντέλου μας. Αυτό το επιτυγχάνει με τις χρωματικές αλλαγές στους άξονες κόκκινο για X,πράσινο για Y και μπλε για Z όπως είδαμε και στην εικόνα 17.

3.3 ΠΡΟΣΘΗΚΗ ΜΟΝΤΕΛΟΥ

Υπάρχουν δύο τρόποι για να προστεθεί ένα μοντέλο στο project.Αν θέλουμε να κάνουμε το μοντέλο μας από την αρχή, μπορούμε να πατήσουμε το Add που βρίσκεται δεξιά από τα modes και να επιλέξουμε το αρχικό σχήμα(mesh) που επιθυμούμε καθώς και πολλά άλλα αντικείμενα όπως για παράδειγμα ένα armature(σκελετός) στο μοντέλο μας. Ο δεύτερος τρόπος βρίσκεται στην επιλογή File>Import, στην συνέχεια επιλέγουμε τον τύπο αρχείου και βρίσκουμε το υπάρχων αρχείο που έχουμε δημιουργήσει στο παρελθόν ή που έχουμε κατεβάσει από κάποιο site με δωρεάν μοντέλα ή επί πληρωμή.



Εικόνα 3.3.1: Add menu, File menu.

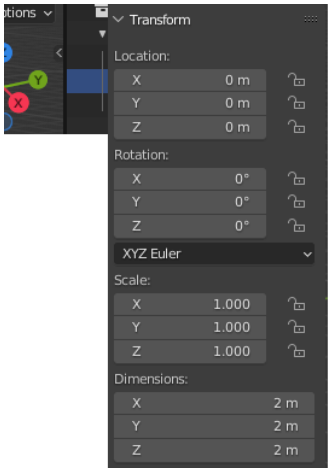
3.4 TOOLBAR

Επιπλέον στο αριστερό μέρος, βλέπουμε μια λίστα από εργαλεία που μας εμφανίζει όπου μπορούμε να χρησιμοποιήσουμε έτσι ώστε να επιλέξουμε ένα αντικείμενο, να το μετακινήσουμε, να το περιστρέψουμε, να προσαρμόσουμε το μέγεθος του και άλλα. Επίσης το toolbar έχει συντομεύσεις πληκτρολογίου όπου θα αναφέρουμε κάποιες βασικές. Για τη επιλογή υπάρχει το πλήκτρο B και για το mode(tweak,box,circle,lasso) της επιλογής μπορούμε να το αλλάξουμε με το W.Για την μετακίνηση(move) χρησιμοποιούμε το πλήκτρο G. Για την περιστροφή(rotate) χρησιμοποιούμε το πλήκτρο R.Για την αναπροσαρμογή μεγέθους(scale)το πλήκτρο S.Τέλος εάν θέλουμε να κάνουμε move,rotate,scale σε έναν συγκεκριμένο άξονα μπορούμε να πατήσουμε για παράδειγμα G+Z,για την μετακίνηση του ως προς Z.

Εικόνα 3.4.1: Toolbar.

3.5 TRANSFORM MENU

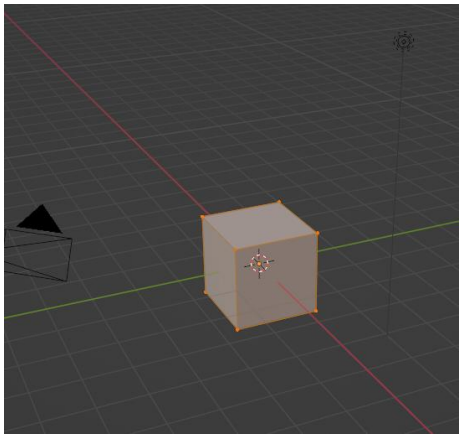
Στο transform menu μπορούμε να δούμε και να αλλάξουμε τις ακριβείς τιμές ως προς το location στο χώρο, το rotate, το scale και τις διαστάσεις του. Το menu αυτό μπορούμε να το δούμε επιλέγοντας το μοντέλο μας και κάνοντας αριστερό κλίκ στο πολύ μικρό βελάκι που έχει δεξιά από τους άξονες.



Εικόνα 3.5.1: Transform menu.

3.6 EDIT MODE

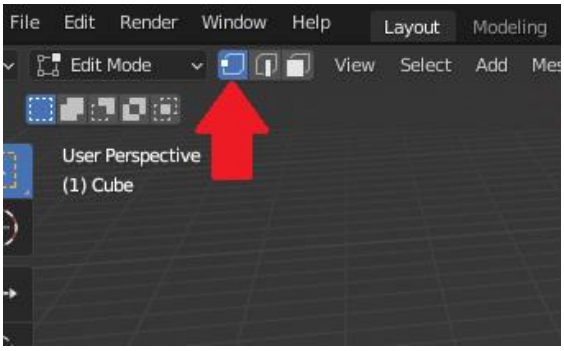
Σε αυτό το mode μπορούμε να επεξεργαστούμε το μοντέλο μας. Τα μοντέλα χωρίζονται σε vertexes, edges και faces για καλύτερη διαχείρισή τους όπου θα περιγράψουμε παρακάτω. Όπως είδαμε στο κεφάλαιο 3.1 , για να μεταβούμε στο edit mode ,αφότου έχουμε επιλέξει το μοντέλο μας(πχ το Cube), πατάμε στον αριθμό 3 που έχουμε προαναφέρει στην εικόνα 3.1.2 ή πατάμε τη συντόμευση TAB.



Εικόνα 3.6.1:Εισαγωγή στο Edit mode.

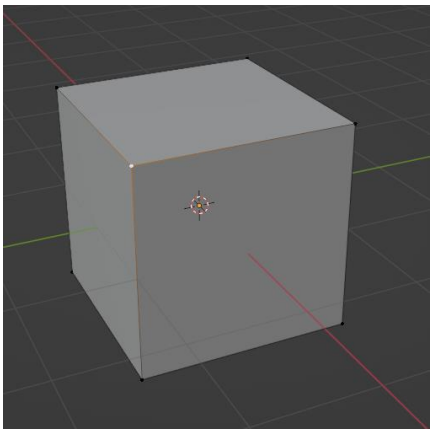
1. **Vertex**

Ένα vertex είναι ένα σημείο ή μια τοποθεσία σε ένα 3D διάστημα. Συγκεκριμένα χρησιμοποιείται στα 3D μοντέλα μας ως «ενώσεις» από τα κομμάτια που αποτελείται.



Εικόνα 3.6.2:Επιλογή Vertex mode στο BLENDER.

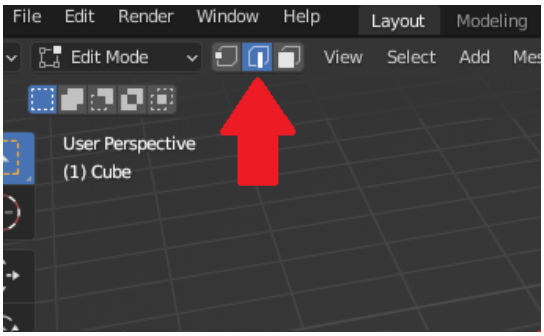
Αφού επιλέξουμε το mode στην παραπάνω εικόνα μπορούμε να επιλέξουμε το vertex που θέλουμε να επεξεργαστούμε.



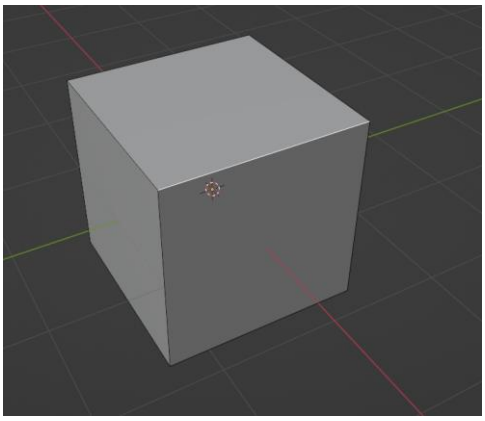
Εικόνα 3.6.3:Επιλεγμένο Vertex στο Blender

2. **Edge**

Το edge είναι μια ευθεία γραμμή σε ένα 3D διάστημα. Στα 3D μοντέλα είναι οι ευθείες γραμμές όπου ενώνουν τα vertexes μεταξύ τους.



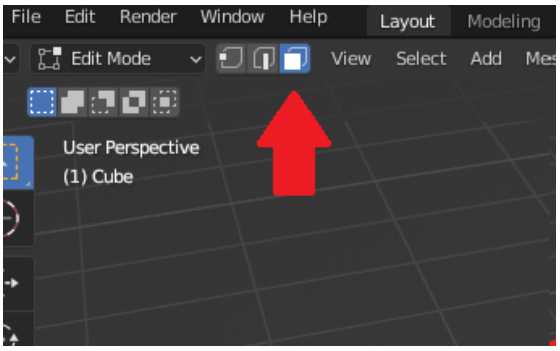
Εικόνα 3.6.4:Επιλογή Edge mode στο BLENDER.



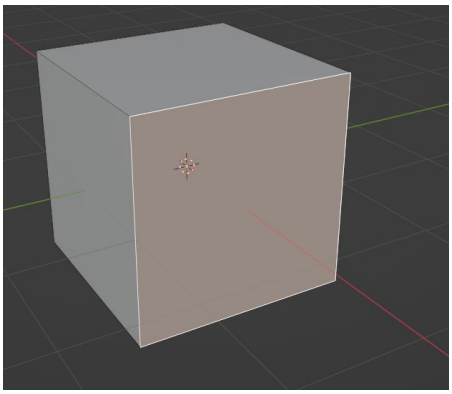
Εικόνα 3.6.5:Επιλεγμένο Edge στο BLENDER.

3. Face

Το face περιέχει vertexes και edges.Είναι η επιφάνεια όπου εάν υπάρχει κενός χώρος ανάμεσα στα vertexes και τα edges,τον «γεμίζει», με αποτέλεσμα να δημιουργείται ένα «πρόσωπο» αυτού του χώρου.



Εικόνα 3.6.6: Επιλογή Face mode στο BLENDER.



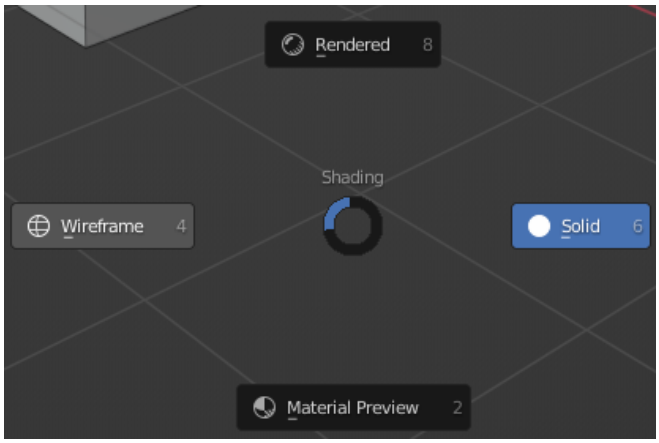
Εικόνα 3.6.7: Επιλεγμένο Face στο BLENDER.

3.6.1 DISPLAY MODES

Για διευκόλυνση μας το Blender έχει κάποια display modes που θα μας βοηθήσουν στην επεξεργασία μας. Τα display modes μπορούμε να τα χρησιμοποιήσουμε στο edit mode αλλά και στο object mode.Υπάρχουν 4 display modes:

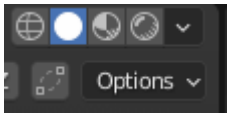
1. **Solid**
   Το solid mode εμφανίζει το μοντέλο μας ως ένα στερεό σώμα.
2. **Wireframe**
   Το wireframe mode εμφανίζει τα περιγράμματα του μοντέλου μας.
3. **Material Preview**
   Το material preview mode εμφανίζει το μοντέλο μας με το υλικό (θα αναλύσουμε παρακάτω τι είναι ένα material).
4. **Rendered**
   Το rendered mode εμφανίζει το μοντέλο μας με το υλικό του και τον φωτισμό. Σε αυτό το mode μπορούμε να δούμε το αποτέλεσμα μιας σκηνής.

Μπορούμε να αλλάξουμε τα display modes με δύο τρόπους, με την συντόμευση (πλήκτρο Z) ή με τα εικονίδια από το περιβάλλον(πάνω δεξιά).



Εικόνα 3.6.8: Αλλαγή display modes με την συντόμευση Z.

Και μετά μπορούμε να κουνήσουμε το ποντίκι προς το mode που θέλουμε ή να πατήσουμε τον αντίστοιχο αριθμό.



Εικόνα 3.6.9: Εικονίδια για αλλαγή display mode(wireframe, solid, material preview, rendered)με τη σειρά.

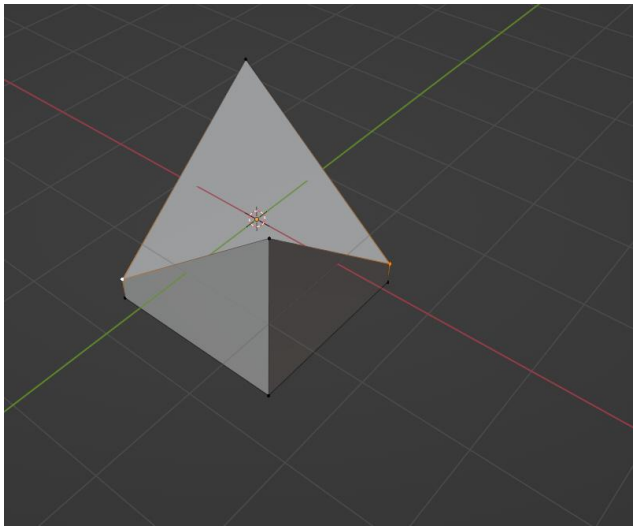
Επιπλέον μαζί με αυτά τα displays μπορούμε να συνδυάσουμε και το X-Ray mode για να μπορούμε να δούμε μέσα από αντικείμενα. Το ενεργοποιούμε/απενεργοποιούμε με την συντόμευση (Alt+Z) ή το παρακάτω εικονίδιο(πάνω δεξιά στο περιβάλλον).

Εικόνα 3.6.10: X-Ray button.

3.6.2 ΤΡΟΠΟΠΟΙΗΣΗ ΜΟΡΦΗΣ ΜΟΝΤΕΛΟΥ

Σε αυτό το σημείο θα τροποποιήσουμε την μορφή του μοντέλου(Cube) για να δείξουμε κάποιες λειτουργίες που έχει το edit mode όσων αφορά την επεξεργασία. Θα χρησιμοποιήσουμε κυρίως το solid display mode και το wireframe display mode.Το edit mode έχει μια πληθώρα από εργαλεία που μπορούμε να χρησιμοποιήσουμε έτσι ώστε να επεξεργαστούμε το μοντέλο μας (χρησιμοποιώντας τα vertexes, edges και faces του μοντέλου) και να το κάνουμε να πάρει το σχήμα που θέλουμε εμείς. Όπως και στο object mode που έχει toolbar,έτσι και στο edit mode έχει ένα δικό του toolbar(αριστερά) με κάποια από τα εργαλεία του.

Ο πιο απλός τρόπος που μπορούμε να αλλάξουμε την μορφή ενός μοντέλου είναι να επιλέξουμε vertexes,edges και faces και να τα κάνουμε move ή rotate ή scale.



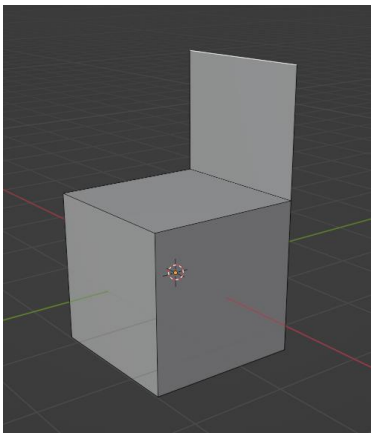
Εικόνα 3.6.11: Παράδειγμα τροποποίησης μορφής με χρήση Vertexes και move στον άξονα Z.

Υπάρχουν όμως και άλλοι τρόποι που μπορούμε να χρησιμοποιήσουμε:

- **EXTRUDE**
  Το extrude στην ουσία επεκτείνει την επιλογή μας. Ενεργοποιείται με τη συντόμευση(πλήκτρο E)ή από το εικονίδιο στο toolbar.
  

Εικόνα 3.6.12: Εικονίδιο extrude.

Εάν επιλέξουμε ένα edge και το κάνουμε extrude στον άξονα Z το αποτέλεσμα θα είναι κάπως έτσι:

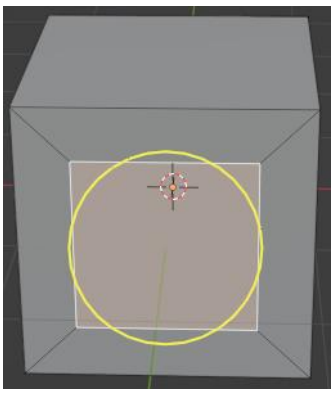


Εικόνα 3.6.13: Παράδειγμα Extrude.

Παρατηρούμε ότι δημιουργείται ένα επιπλέον edge που μπορούμε να διαχειριστούμε άρα και vertexes και face.

- **INSET FACES**
  Το inset faces δημιουργεί ένα face μέσα στο face που έχουμε επιλέξει. Ενεργοποιείται με τη συντόμευση(πλήκτρο I)ή από το εικονίδιο στο toolbar.

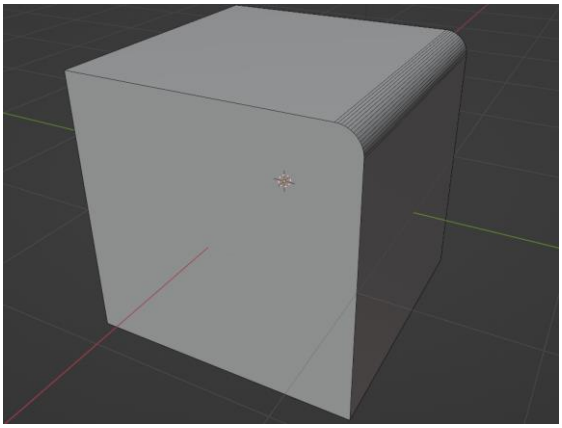
Εικόνα 3.6.14: Εικονίδιο inset faces.



Εικόνα 3.6.15: Παράδειγμα inset faces.

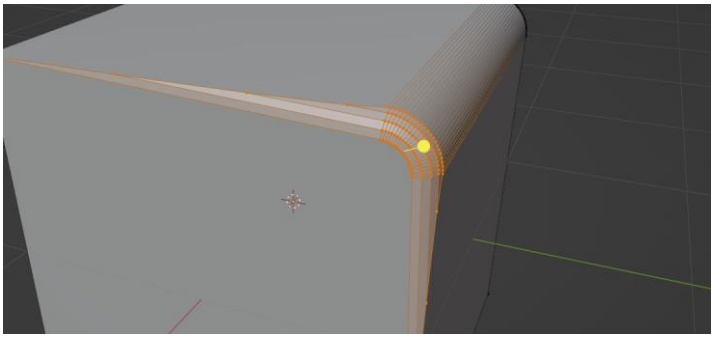
- **BEVEL**
  Το bevel μπορούμε να το χρησιμοποιήσουμε για να κάνουμε μια γωνία πιο στρογγυλή. Ενεργοποιείται με τη συντόμευση(πλήκτρα CTRL+B)ή από το εικονίδιο στο toolbar.

Εικόνα 3.6.16:Εικονίδιο bevel.



Εικόνα 3.6.17: Παράδειγμα bevel edge.

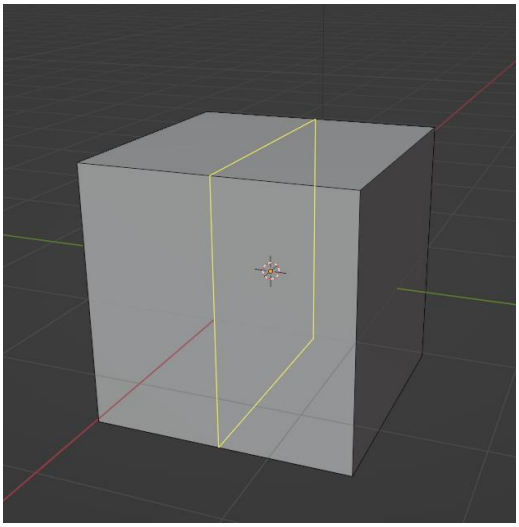
Για το αποτέλεσμα αυτό όπως στην εικόνα ενεργοποιούμε το bevel και μόλις επιλέγουμε το edge εμφανίζει έναν κίτρινο κύκλο όπου τον μετακινούμε για να ρυθμίσουμε την λοξή τομή και με την ροδέλα του ποντικιού το υποδιαιρούμε στα πολλαπλά edges όπου φαίνεται σαν να γίνεται στρογγυλό.



Εικόνα 3.6.18: Παράδειγμα bevel vertex.

- **LOOP CUT**
  Το loop cut κόβει το μοντέλο μας γύρω από μια περιοχή. Ενεργοποιείται με τη συντόμευση(πλήκτρα CTRL+R)ή από το εικονίδιο στο toolbar.

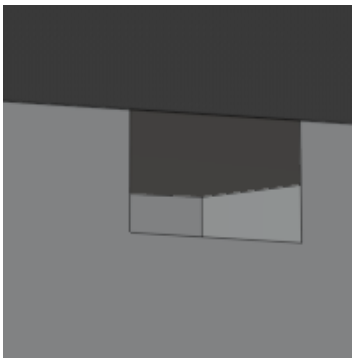
Εικόνα 3.6.19: Εικονίδιο loop cut.



Εικόνα 3.6.20: Παράδειγμα loop cut.

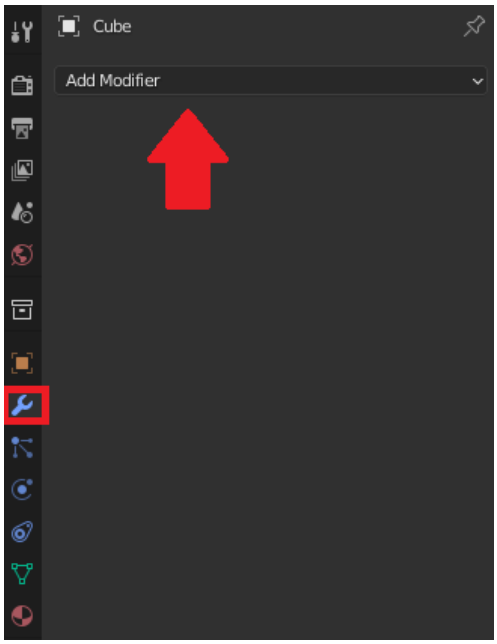
- **KNIFE**
  Με το knife μπορούμε να κόψουμε όποιο κομμάτι θέλουμε από το μοντέλο μας. Ενεργοποιείται με τη συντόμευση(πλήκτρο K)ή από το εικονίδιο στο toolbar.
  

Εικόνα 3.6.21: Εικονίδιο knife.



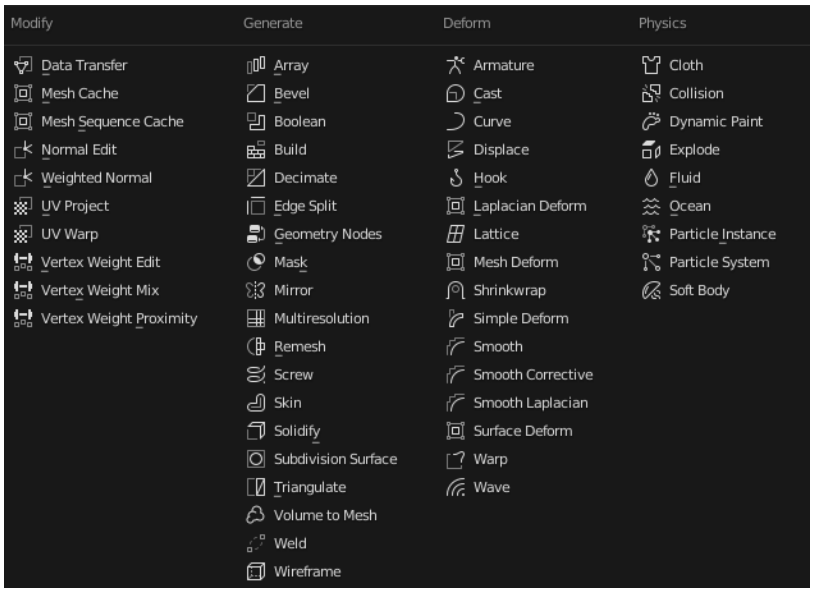
Εικόνα 3.6.22:Παράδειγμα knife.

Επιπλέον εάν θέλουμε να τροποποιήσουμε τη μορφή του μοντέλου μας συνολικά και όχι μεμονωμένα κομμάτια του, μπορούμε να χρησιμοποιήσουμε τα modifiers που μας προσφέρει το Blender.Επιλέγοντας το μοντέλο μας, θα βρούμε την επιλογή των modifiers στο panel που βρίσκεται κάτω από το hierarchy.



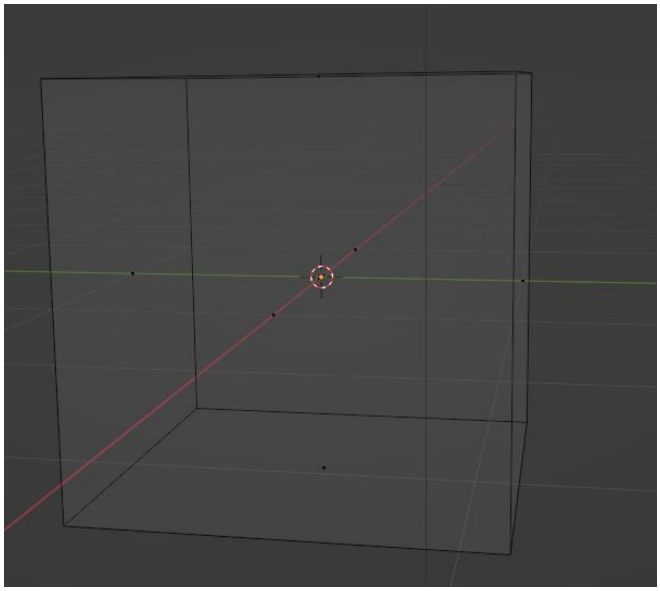
Εικόνα 3.6.23: Μετάβαση στα modifiers.

Έπειτα πατάμε το κουμπί «Add Modifier» και εμφανίζεται η λίστα με τα modifiers που μπορούμε να χρησιμοποιήσουμε.



Εικόνα 3.6.24: Λίστα modifiers.

Τέλος θα δούμε πως το X-Ray με το wireframe display mode μας διευκολύνει στην επεξεργασία. Για παράδειγμα θέλουμε από το Cube να επιλέξουμε το εμπρόσθιο και το πίσω face ταυτόχρονα χωρίς να χρειαστεί να αλλάξουμε το view.

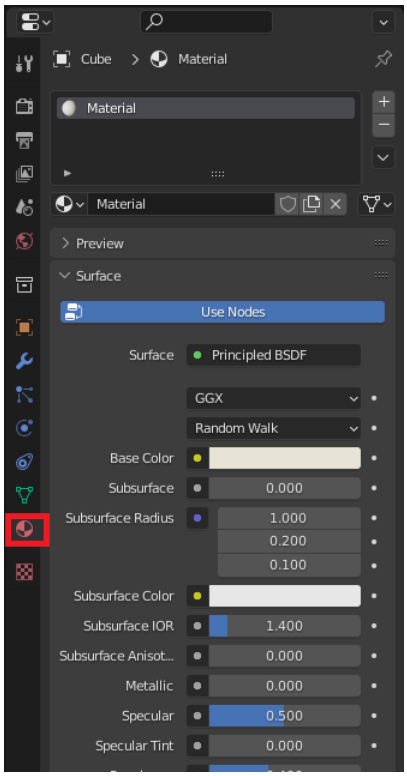


Εικόνα 3.6.25: Wireframe mode και X-Ray.

Οι τελείες μας δείχνουν το κέντρο του εκάστοτε face,εάν υπάρχει.

3.7 MATERIALS

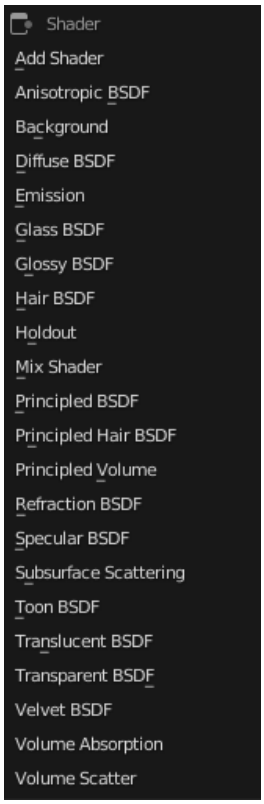
Τα materials είναι βασικό συστατικό στην δημιουργία ενός μοντέλου όπου θα χρησιμοποιηθεί σε ένα παιχνίδι. Το material είναι το «υλικό» που έχει ένα μοντέλο, πιο συγκεκριμένα το χρώμα του υλικού. Μεταβαίνουμε σε material preview display mode για να βλέπουμε τις αλλαγές που κάνουμε στο material και έπειτα στο Material properties(κάτω από το hierarchy panel).



Εικόνα 3.7.1:Επιλογή Material properties.

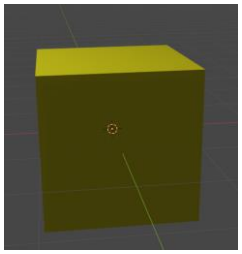
Αρχικά βλέπουμε ότι το Cube έχει ένα material που το έχει ονομάσει by default «Material».Εάν θέλουμε να αφαιρέσουμε αυτό το material,όταν το έχουμε επιλεγμένο πατάμε το κουμπί -,ενώ αν θέλουμε να προσθέσουμε πατάμε το κουμπί +.Επίσης μπορούμε να του αλλάξουμε όνομα ακριβώς από κάτω.

Ένας shader βοηθάει στον τρόπο της εμφάνισης ενός material.Υπάρχουν πολλοί shaders που μπορούμε να χρησιμοποιήσουμε ανάλογα με το πώς θέλουμε να φαίνεται το χρώμα μας. Ο Principled BSDF είναι ο default shader που επιλέγει το Blender σε ένα νέο material.Πατώντας επάνω στον shader εμφανίζεται η λίστα με όλους τους shaders.

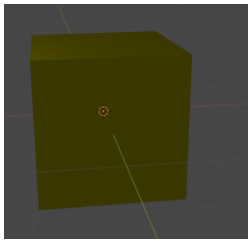


Εικόνα 3.7.2: Λίστα των shaders.

Ο πιο απλός τρόπος να αλλάξουμε το χρώμα ενός material είναι να αλλάξουμε απλά το base color.Για παράδειγμα να το κάνουμε κίτρινο και να δούμε την διαφορά χρώματος Principled BSDF και Glossy BSDF.



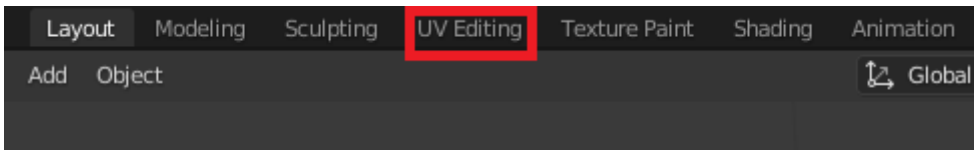
Εικόνα 3.7.3: Χρώμα με Principled BSDF shader.



Εικόνα 3.7.4:Χρώμα με Glossy BSDF shader.

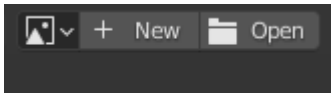
Ο Glossy BSDF shader έχει την ιδιότητα να γυαλίζει καθώς πέφτει ο φωτισμός.

Υπάρχουν και άλλοι τρόποι για να προσθέσουμε ένα χρώμα σε ένα material. Από το UV Editing του Blender μπορούμε να χρησιμοποιήσουμε μια εικόνα ως σημείο αναφοράς στα χρώματα της. Μεταβαίνουμε στο UV Editing για να δώσουμε ένα παράδειγμα.



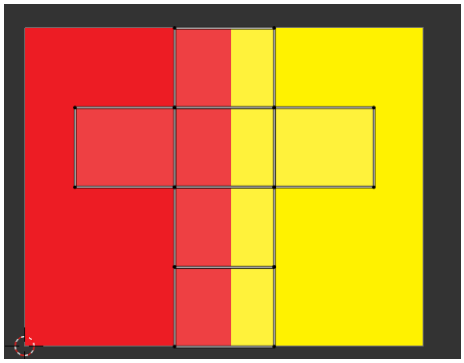
Εικόνα 3.7.5:Επιλογή UV Editing.

Στο αριστερό μέρος, επάνω έχει κάποιες επιλογές, για να δημιουργήσουμε μια εικόνα πατάμε New και για να προσθέσουμε μια εικόνα που έχουμε αποθηκεύσει τοπικά πατάμε Open.Θα χρησιμοποιήσουμε μια εικόνα που έχουμε τοπικά στο παράδειγμα.



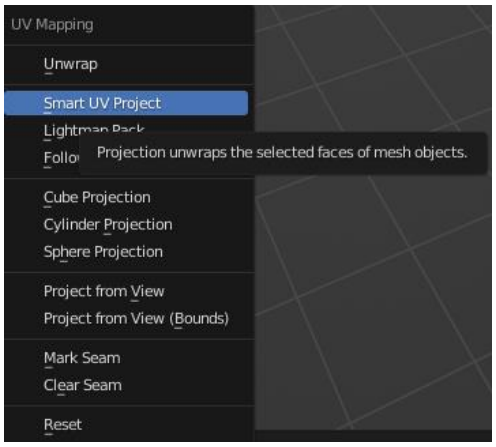
Εικόνα 3.7.6: New/Open image UV Editing.

Επιλέγουμε όλο το μοντέλο μας έτσι ώστε να κάνουμε unwrap,στην ουσία ξεδιπλώνουμε το μοντέλο στην εικόνα.



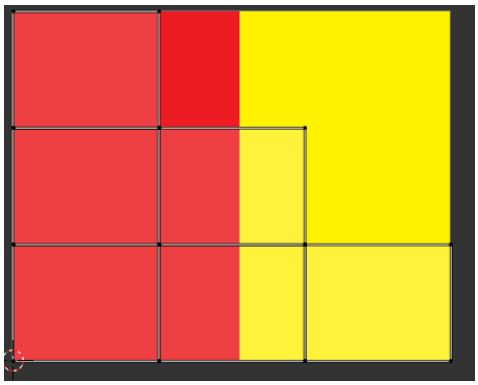
Εικόνα 3.7.7:Default Unwrap.

Πατάμε το πλήκτρο U για να κάνουμε Unwrap για παράδειγμα σε Smart UV project.



Εικόνα 3.7.8:Επιλογή Smart UV Project.

Με τις default μοίρες του Smart UV Project το αποτέλεσμα είναι αυτό:

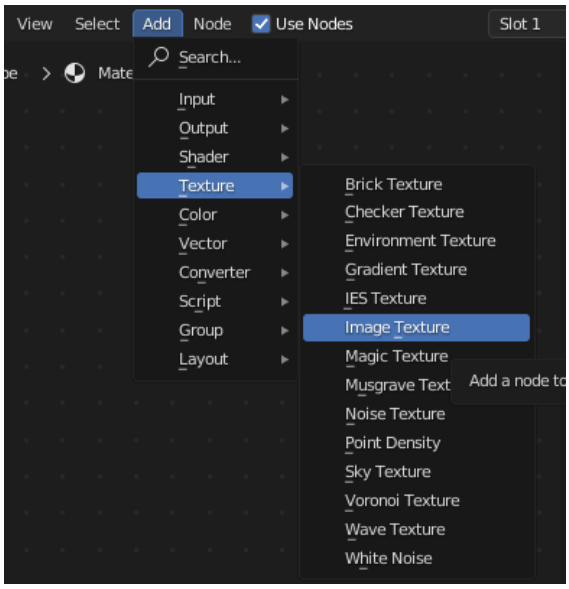


Εικόνα 3.7.9:Αποτέλεσμα Smart UV Project.

Το αποτέλεσμα στο μοντέλο μας δεν είναι ακόμα ορατό, γιατί στον shader έχει κρατήσει το απλό χρώμα. Για να βάλουμε σε έναν shader χρώμα από εικόνα μεταβαίνουμε στο Shading.

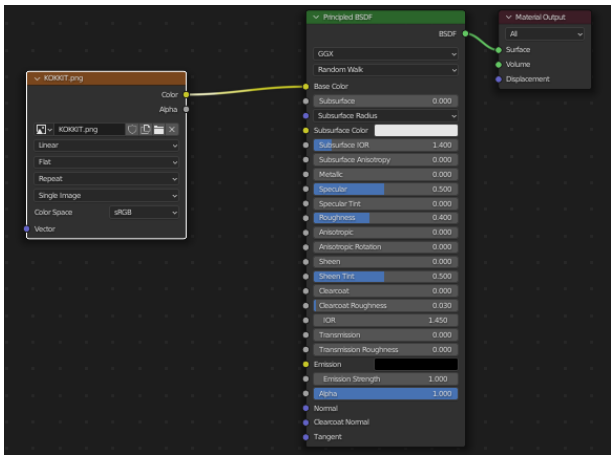
Εικόνα 3.7.10:Επιλογή Shading.

Στο shading μπορούμε να κάνουμε πολύ εξειδικευμένα πράγματα στο χρώμα μας. Στο παιχνίδι χρησιμοποιήσαμε πολύ το shading έτσι ώστε να αναμίξουμε χρώματα, να κάνουμε πιο σκούρες τις εικόνες, αναμίξαμε ακόμα και διαφορετικά shaders μεταξύ τους. Στο κάτω μέρος βρίσκεται το panel με τα nodes και η διασύνδεση αυτών, μέχρι το Material Output.Για να βάλουμε την εικόνα texture στο base color του shader πατάμε το κουμπί «Add» στο node panel ή την συντόμευση (SHIFT+A),έπειτα επιλέγουμε Texture->Image Texture.



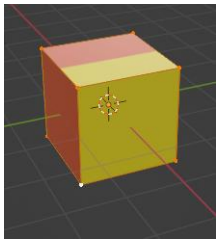
Εικόνα 3.7.11:Επιλογή Image Texture.

Αφού επιλέξουμε Image Texture εμφανίζεται το αντίστοιχο node.Πατάμε Open, βρίσκουμε την εικόνα μας και έπειτα συνδέουμε το Color του Image Texture στο Base Color του shader.



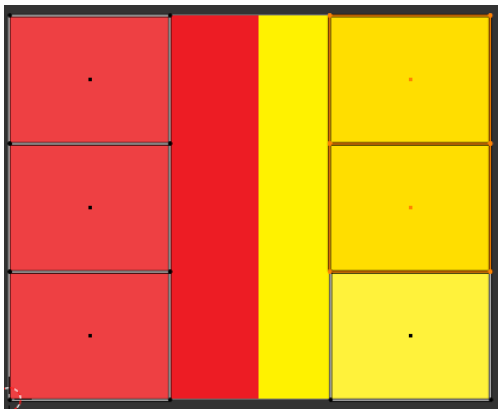
Εικόνα 3.7.12: Συνολική διασύνδεση nodes.

Γυρίζουμε στο UV Editing και το αποτέλεσμα είναι το εξής:

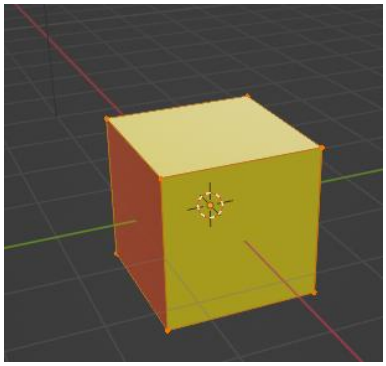


Εικόνα 3.7.13: Αποτέλεσμα Image Texture με Smart UV Project.

Για να προσαρμόσουμε τα χρώματα στο Cube σωστά ας υποθέσουμε ότι θέλουμε να έχουμε τρείς πλευρές κόκκινες και τρείς πλευρές κίτρινες. Στο αριστερό μέρος όπου βλέπουμε το unwrap κάθε face αντιστοιχεί σε ένα face του Cube. Περαιτέρω μπορούμε να επιλέξουμε τα μεσαία faces και να τα μετακινήσουμε στο κίτρινο κομμάτι του texture.



Εικόνα 3.7.14: Unwrap μετά την μετακίνηση των faces.



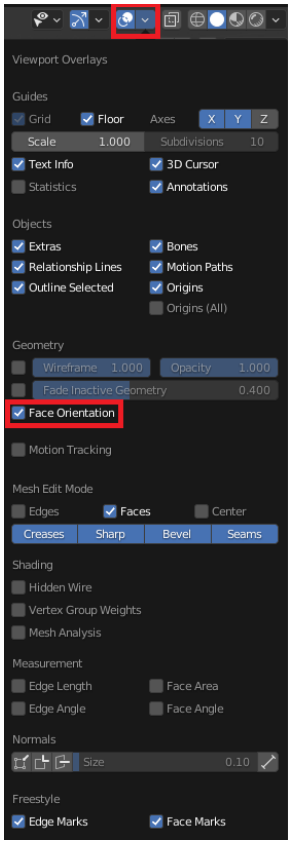
Εικόνα 3.7.15:Αποτέλεσμα μετά την μετακίνηση των faces.

Έτσι πέτυχαμε το αποτέλεσμα που θέλαμε. Με τον ίδιο τρόπο εργασθήκαμε σε πολλά materials μέσα στο πρότυπο παιχνίδι που δημιουργήσαμε και τροποποιήσαμε το χρώμα τους μέσω του Shading και των shaders.

3.7.1 FACE ORIENTATION

Το face orientation θα μπορούσαμε να πούμε με απλά λόγια ότι είναι η κατεύθυνση που «βλέπει» ένα face.Καθώς δημιουργούσαμε το παιχνίδι στο Unity αντιμετωπίσαμε το face orientation σαν εμπόδιο, διότι το face orientation επηρεάζει την εμφάνιση των materials,δεν εμφάνιζε το χρώμα ή ακόμα και το στερεό κομμάτι του μοντέλου. Αυτό σε πολύπλοκα μοντέλα μπορεί να είναι ένα πολύ χρονοβόρο πρόβλημα.

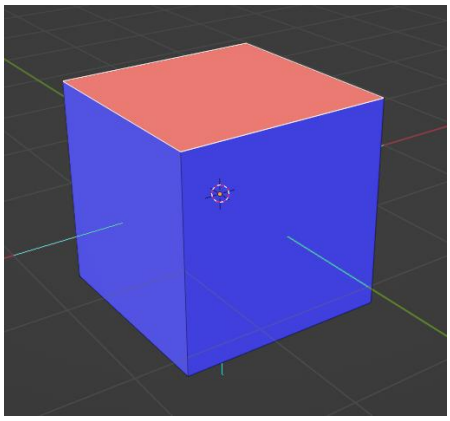
Για να λύσουμε το πρόβλημα του face orientation στο Blender,αφού βρισκόμαστε στο edit mode,για το δούμε επιλέγουμε το βελάκι δίπλα από το «Show Overlays»(πάνω δεξιά) και έπειτα επιλέγουμε το checkmark στο face orientation κάτω από το πεδίο Geometry.



Εικόνα 3.7.16: Επιλογή Face Orientation.

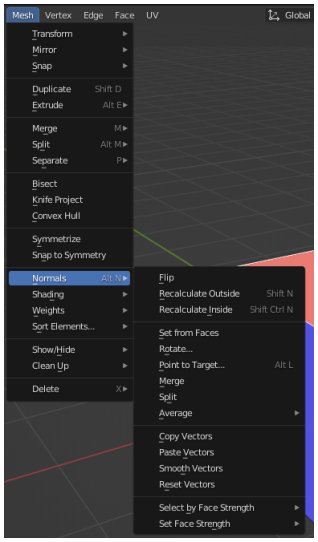
Κάτω από το πεδίο Normals επιλέγουμε το face και προσαρμόζουμε το Size έτσι ώστε να είναι πιο ορατό. Αυτή η ρύθμιση εμφανίζει μια ευθεία γραμμή που μας δείχνει την κατεύθυνση που «βλέπει» το face.

Ας υποθέσουμε ότι το Cube έχει την μία του πλευρά με κατεύθυνση προς το εσωτερικό του.



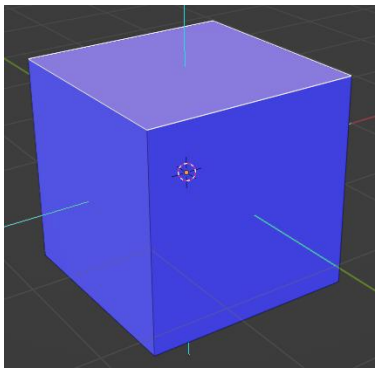
Εικόνα 3.7.17: Παράδειγμα Face Orientation.

Οι μπλε πλευρές σημαίνουν ότι η πλευρά που βλέπουμε είναι το μπροστινό μέρος του face,ενώ οι κόκκινες το πίσω μέρος του face.Αυτό σημαίνει ότι το material θα εμφανίζεται σωστά μόνο στις μπλε πλευρές. Για να το λύσουμε αυτό και να γυρίσουμε την πλευρά στην επιθυμητή κατεύθυνση ανάλογα με το μοντέλο, υπάρχουν δυο αυτόματοι τρόποι(recalculate inside,recalculate outside)και ένας χειροκίνητος(flip).Το recalculate inside κατευθύνει τα όλα faces στο εσωτερικό του μοντέλου. Το recalculate outside κατευθύνει όλα τα faces στο εξωτερικό του μοντέλου, στο 3D διάστημα.Το flip,αφού έχουμε επιλέξει τα faces χειροκίνητα, κατευθύνουν τα faces προς στην αντίθετη κατεύθυνση από αυτήν που έχουν την δεδομένη στιγμή. Για να χρησιμοποιήσουμε αυτές τις λειτουργίες επιλέγουμε Mesh->Normals(στο επάνω μέρος) ή την συντόμευση (Alt+N).



Εικόνα 3.7.18: Επιλογή Normals.

Για το παράδειγμα μας επιλέγουμε το κόκκινο face και μετά απλά πατάμε flip.

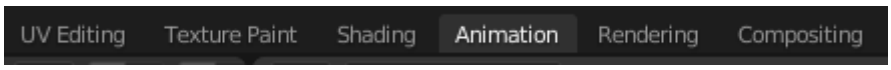


Εικόνα 3.7.19: Αποτέλεσμα flip normal για το face orientation.

3.8 ANIMATION

Με το animation μπορούμε να δημιουργήσουμε κίνηση σε ένα μοντέλο με το πέρασμα του χρόνου. Ένα animation σε ένα μοντέλο ενός χαρακτήρα, προϋποθέτει να εφαρμοσθεί armature(σκελετός) στο μοντέλο. Για να δημιουργήσουμε ένα animation σε ένα μοντέλο-χαρακτήρα, χρησιμοποιούνται τα bones(κόκκαλα) του armature(σκελετού).Επιπλέον μπορούμε να δημιουργήσουμε animation σε ένα μοντέλο για παράδειγμα μιας πόρτας, όπου δεν είναι απαραίτητη η χρήση του armature,έτσι ώστε να ανοίγει ή να κλείνει.

Εν συνεχεία θα δώσουμε ένα παράδειγμα με ένα animation μιας πόρτας που ανοίγει. Για να ξεκινήσουμε την διαδικασία μεταβαίνουμε στο Animation στο Blender.

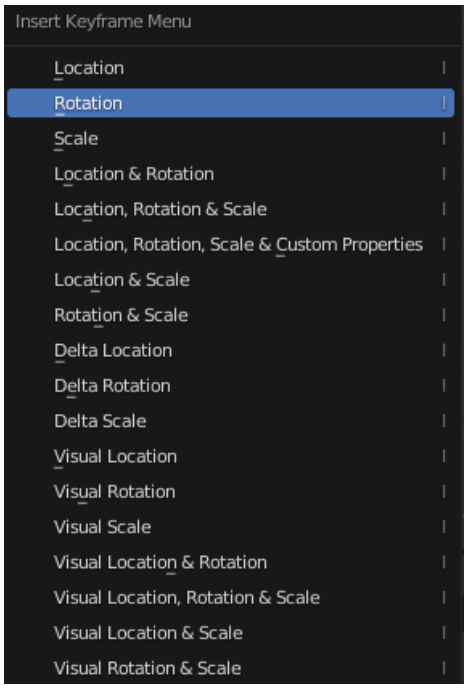


Εικόνα 3.8.1:Μετάβαση στην καρτέλα «Animation”.

Στο κάτω μέρος βλέπουμε το χρονοδιάγραμμα του animation σε frames.Τα 30 frames υλοποιούνται σε 1 δευτερόλεπτο. Για να ορίσουμε το συνολικό χρονικό διάστημα του animation στο κάτω δεξιό μέρος του χρονοδιαγράμματος, αλλάζουμε τις τιμές του Start και του End σε frames.

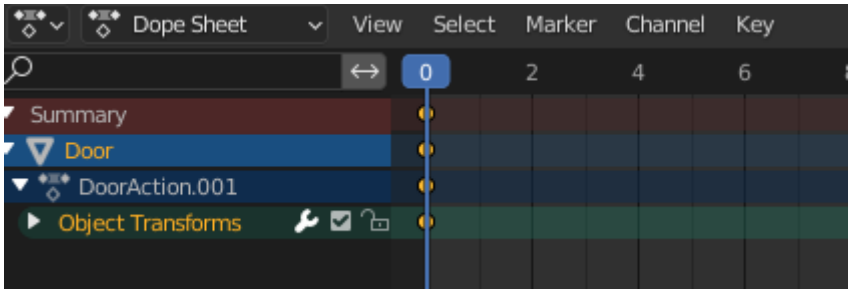
Εικόνα 3.8.2: Ορισμός χρονικού διαστήματος animation σε frames για 1 δευτερόλεπτο.

Για το παράδειγμα μας θα χρησιμοποιήσουμε ένα έτοιμο μοντέλο μιας πόρτας. Για την εγγραφή του animation στο χρονοδιάγραμμα, αποθηκεύουμε το location,rotation και scale,στο εκάστοτε frame.Για να ξεκινήσουμε την πρώτη μας εγγραφή, επιλέγουμε το μοντέλο και έπειτα πατάμε την συντόμευση(πλήκτρο I).Όταν η συντόμευση πατηθεί, εμφανίζεται το Insert Keyframe Menu.



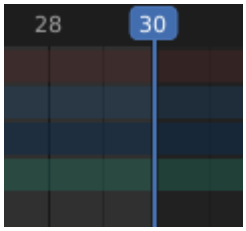
Εικόνα 3.8.3: Insert Keyframe Menu.

Στην περίπτωση μας θα χρησιμοποιήσουμε το Rotation και στο χρονοδιάγραμμα εμφανίζεται η εγγραφή μας.



Εικόνα 3.8.4: Πρώτη εγγραφή Keyframe.

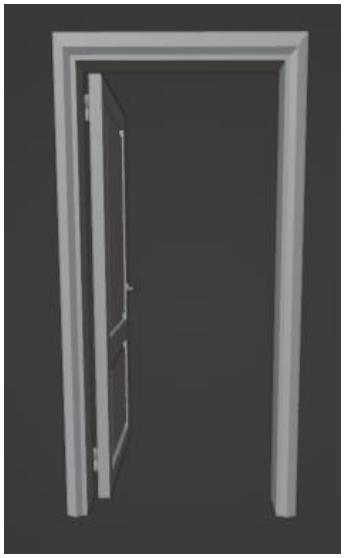
Έπειτα μετακινούμε τον κέρσορα από το frame 0(Start),στο frame 30(End).



Εικόνα 3.8.5:Μετακίνηση κέρσορα στο frame 30.

Στην συνέχεια θα κάνουμε την εγγραφή μας στο frame 30.Στα δεξιά, υπάρχουν τα properties όπου μπορούμε να αλλάξουμε για την νέα εγγραφή .Στο πεδίο Rotation Z αλλάζουμε την τιμή από 0^o σε 90^o,ώστε να επιτύχουμε το άνοιγμα της πόρτας κατά 90^o σε 1 δευτερόλεπτο. Στην συνέχεια, κάνουμε εγγραφή το Rotation.Για να εκτελεστεί το animation πατάμε το πλήκτρο SPACE.

Εικόνα 3.8.6:Η πόρτα στο frame 0.



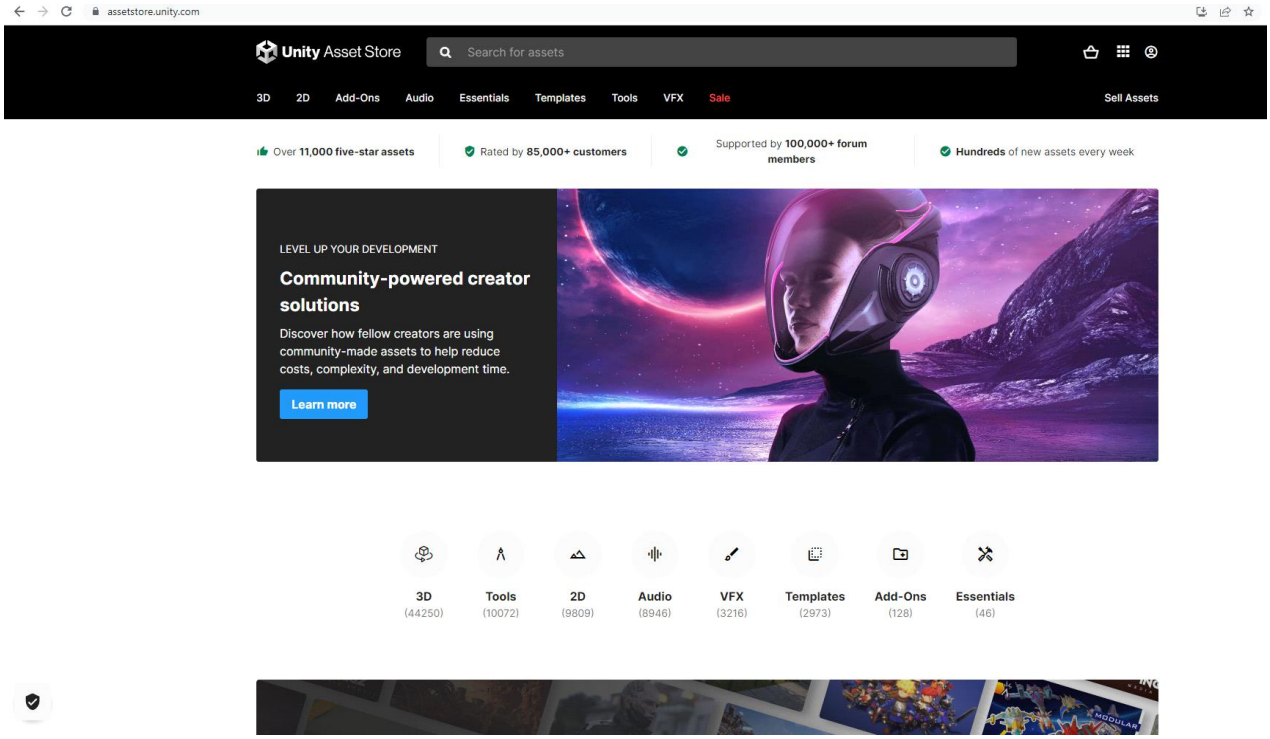
Εικόνα 3.8.7:Η πόρτα στο frame 30.

Στα ενδιάμεσα frames το Blender αυξάνει το Rotation Z από 0 στο 90 σταδιακά. Με αυτόν τον τρόπο δημιουργείται το animation για το άνοιγμα της πόρτας σε 1 δευτερόλεπτο. Εάν αυξήσουμε ή μειώσουμε το συνολικό χρονικό διάστημα, η πόρτα θα ανοίγει αργά ή γρήγορα αντίστοιχα.

4 ΓΝΩΡΙΜΙΑ ΜΕ ΤΟ UNITY

4.1 UNITY ASSET STORE

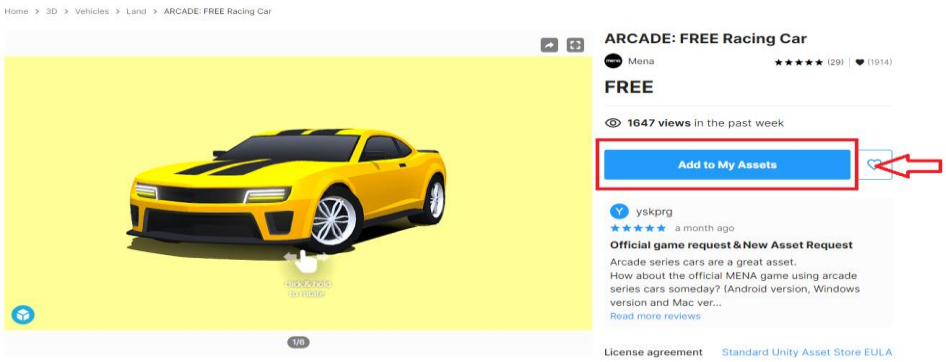
Ένα από τα βασικά κομμάτια του Unity είναι το Unity Asset store. Πρόκειται για μια αναπτυσσόμενη βιβλιοθήκη από διάφορα assets. Τόσο η ίδια η Unity όσο και τα μέλη της κοινότητας δημιουργούν αυτά τα στοιχεία (assets), τα οποία οι ίδιοι τα δημοσιεύουν στο κατάστημα είτε δωρεάν, είτε επί πληρωμή. Στο κατάστημα αυτό υπάρχουν διάφοροι τύποι στοιχείων που κυμαίνονται από textures, κινούμενα σχέδια και μοντέλα μέχρι και ολόκληρα παραδείγματα από projects , σεμινάρια, καθώς και κάποια editor extensions.



Εικόνα 4.1.1: Unity Asset Store.

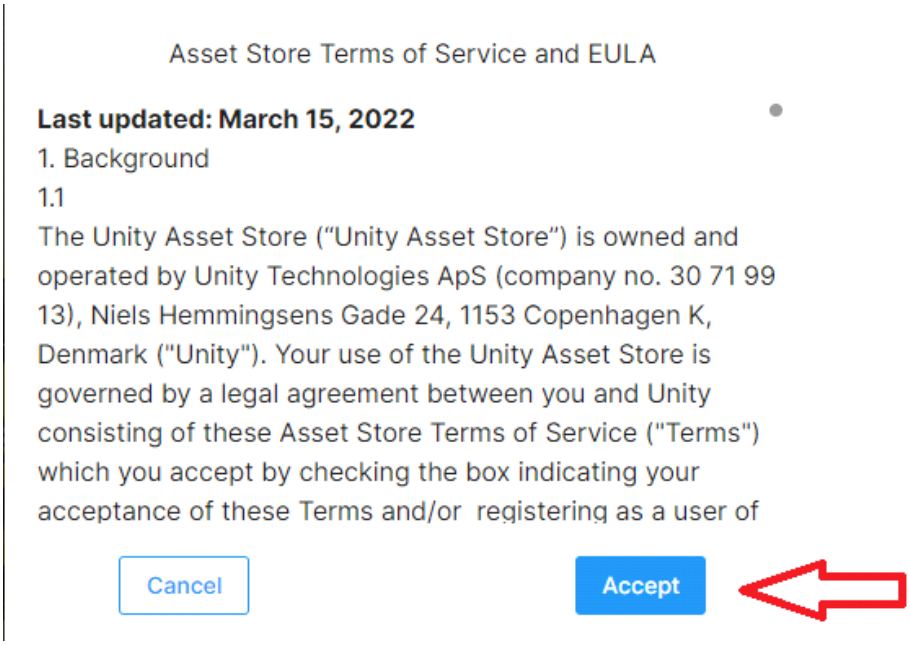
Όπως βλέπουμε και στην εικόνα 4.1.1, μπορούμε να κατεβάσουμε διάφορα μοντέλα / textures η και έτοιμα project. Αυτό που έχουμε να κάνουμε είναι αρχικά να δημιουργήσουμε ένα λογαριασμό στο unity store , στην συνέχεια να κάνουμε log in , έπειτα αφού επιλέξουμε το asset το οποίο θέλουμε να δοκιμάσουμε.

Για να προσθέσουμε ένα asset στον λογαριασμό μας και μετέπειτα να το ανοίξουμε στο Unity,βρίσκουμε το επιθυμητό asset και στην συνέχεια πατάμε το «Add to My Assets».



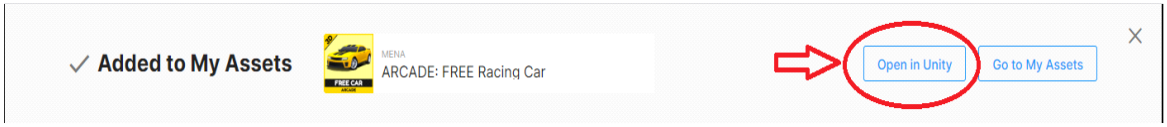
Εικόνα 4.1.2:Επιλογή Add to my Assets.

Στην συνέχεια επιλέγουμε «Accept».



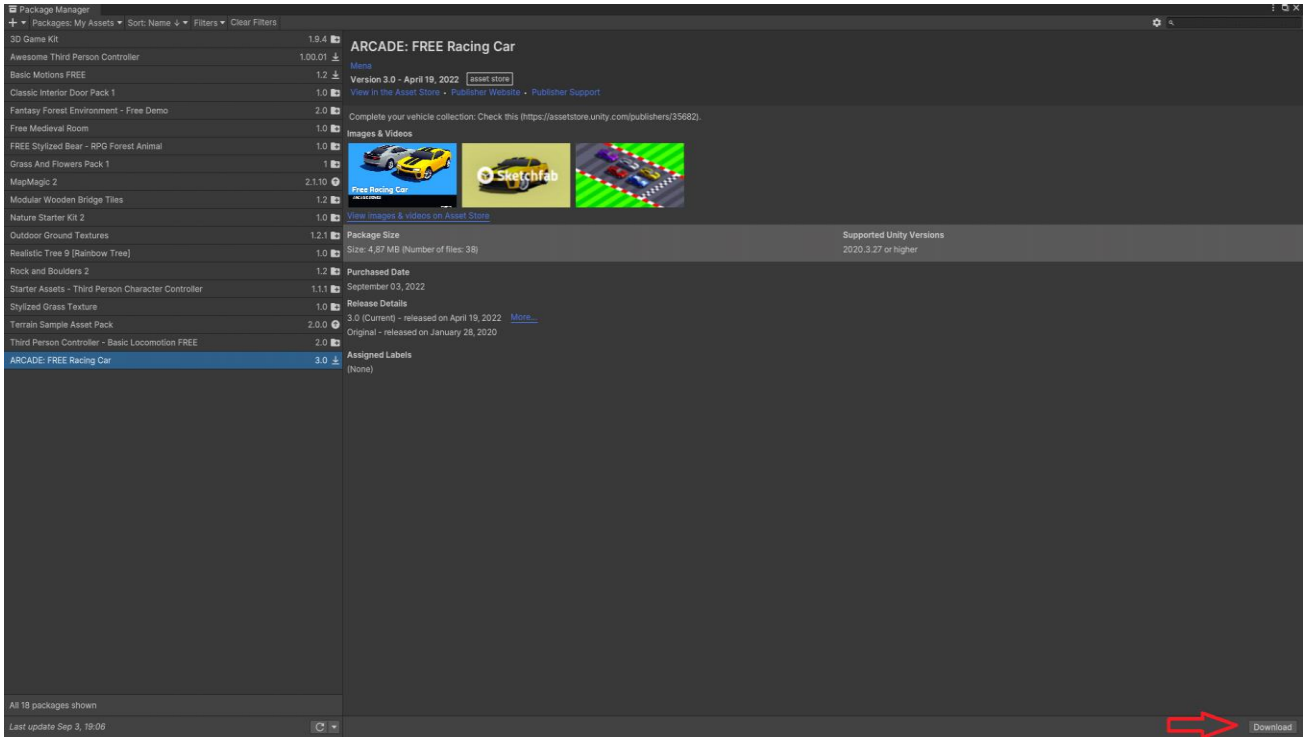
Εικόνα 4.1.3: Accept.

Στην συνέχεια θα εμφανιστεί η παρακάτω ειδοποίηση:



Εικόνα 4.1.4: Ειδοποίηση «Open in Unity» ή «Go to My Assets».

Πατάμε «Open in Unity».Αυτό θα μας κατευθύνει αυτόματα στο Unity όπου και θα ανοίξει αυτόματα το Package manager όπου και θα επιλέξουμε download το asset μας κάτω δεξιά.



Εικόνα 4.1.5:Download asset στο Package manager.

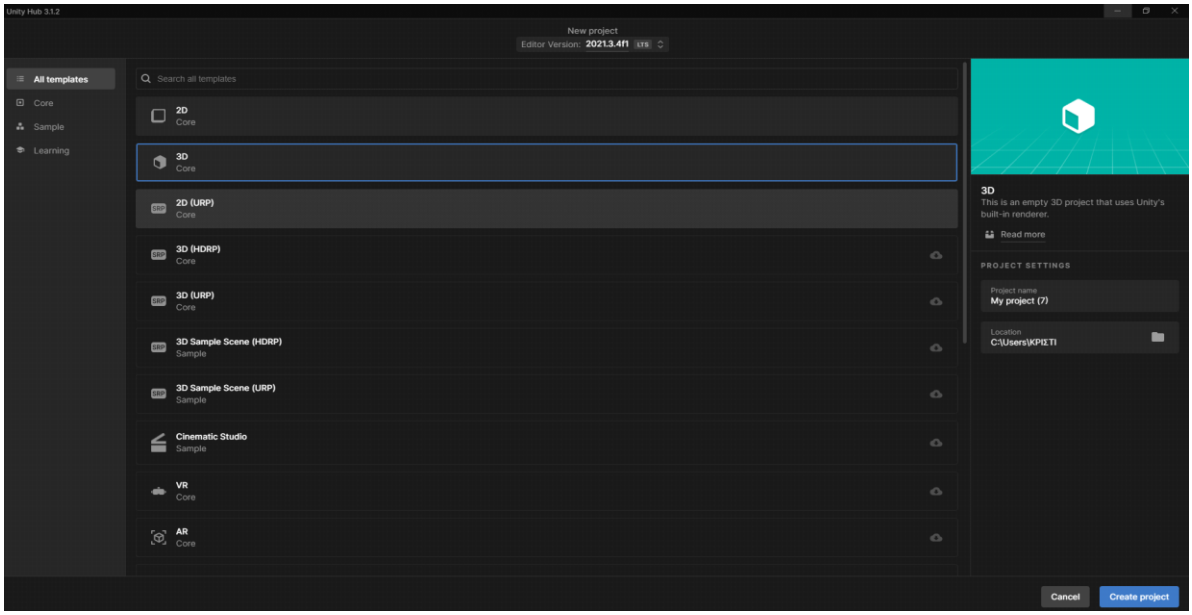
Τέλος αφού ολοκληρωθεί το download θα κάνουμε ένα import το Asset μας στο project το οποίο θέλουμε να το βάλουμε.

4.2 ΔΗΜΙΟΥΡΓΊΑ PROJECT ΣΤΟ UNITY

Για να δημιουργήσουμε ένα project στο Unity αρχικά θα ανοίξουμε το Unity Hub, το οποίο είναι μια αυτόνομη εφαρμογή όπου συμβάλει στην βελτιστοποίηση του τρόπου με τον οποίο βρίσκουμε κατεβάζουμε και διαχειριζόμαστε τα project μας.

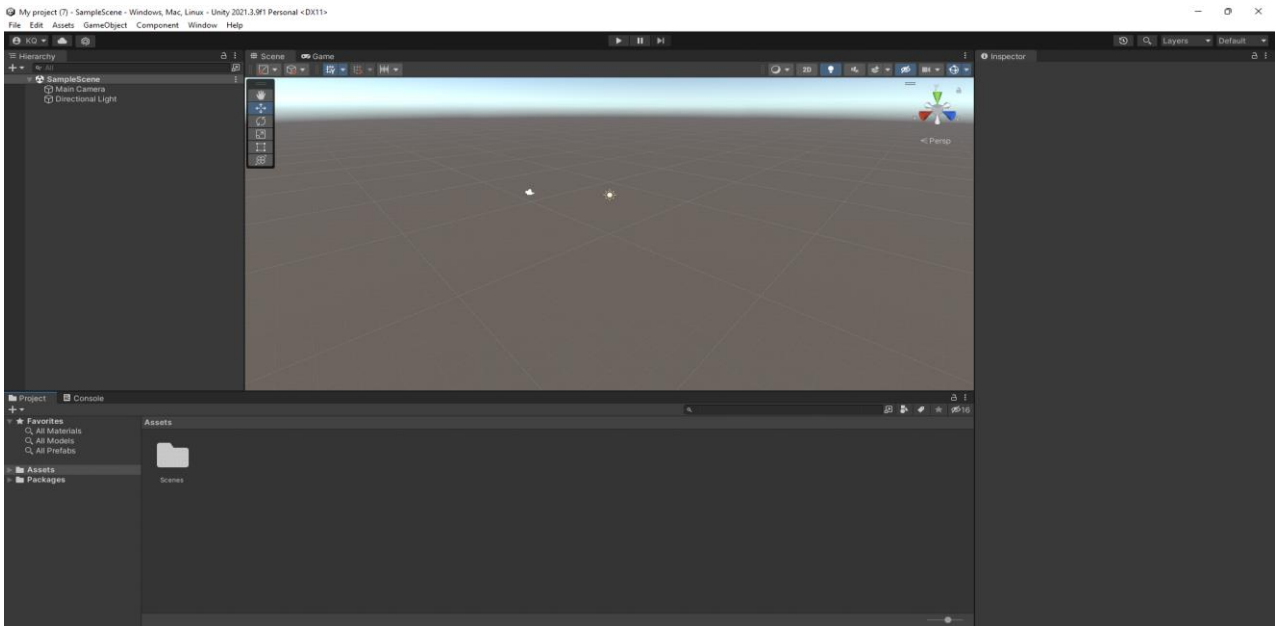
Εικόνα 4.2.1: Εικονίδιο Unity Hub.

Άρα αφού ανοίξουμε την εφαρμογή Unity Hub , θα πατήσουμε το New project , όπου εκεί θα επιλέξουμε έναν από τους διάφορους τύπους που έχει όπως, 3d, 2d , fps microgame και άλλα πολλά. Τέλος, θα πατήσουμε create project και εκεί θα περιμένουμε λίγα λεπτά μέχρι το unity να μας ετοιμάσει το περιβάλλον το οποίο θέλουμε.



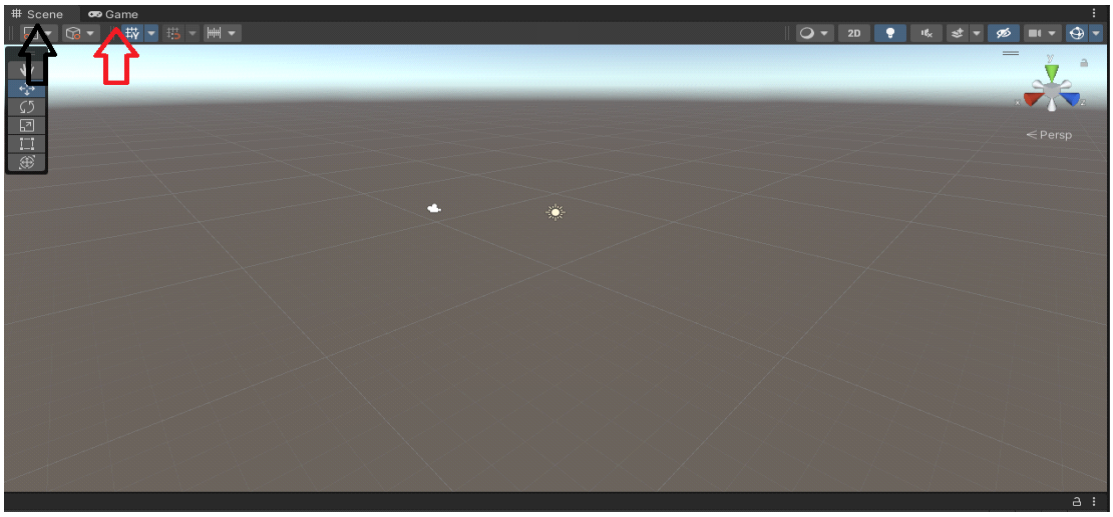
Εικόνα 4.2.2: Διάφοροι τύποι Project που μπορούμε να δημιουργήσουμε.

4.3 ΑΡΧΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΣΤΟ UNITY



Εικόνα 4.3.1:Πρώτη επαφή με το περιβάλλον.

Αυτό είναι το περιβάλλον του Unity πάνω στο οποίο δουλεύουμε, κάθε φορά που αρχίζουμε και δημιουργούμε ένα project. Αυτό που βλέπουμε είναι ότι by default έχουμε ένα sample scene όπου μέσα έχουμε μια main camera όπου στην ουσία αυτό που γίνεται με την main camera είναι , ότι με βάση που θα τοποθετηθεί στο scene , όταν κάνουμε έναρξη το παιχνίδι, εμείς ως παίκτες θα βλέπουμε αυτό που απεικονίζει (κοιτάει) η κάμερα.

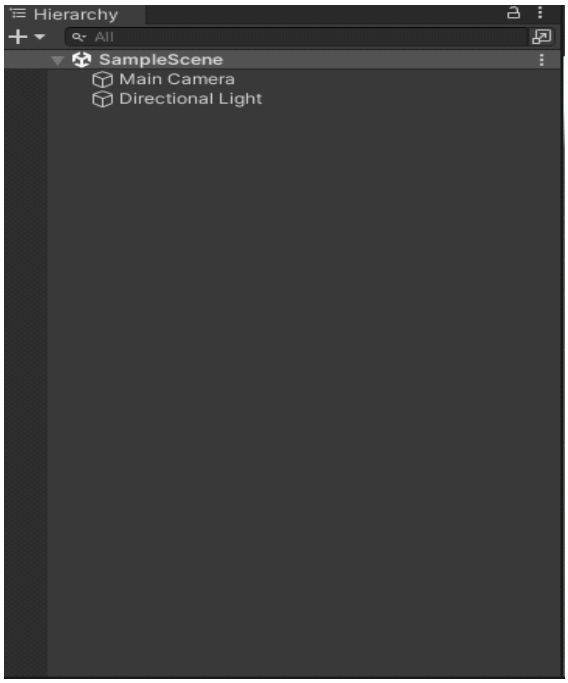


Εικόνα 4.3.2:Μεσαίο panel.

Στο μεσαίο panel θα παρατηρήσουμε ότι έχουμε δύο παράθυρα, το scene και το game. Το scene(μαύρο βελάκι) είναι ουσιαστικά αυτό πάνω στο οποίο δουλεύουμε το παιχνίδι μας και

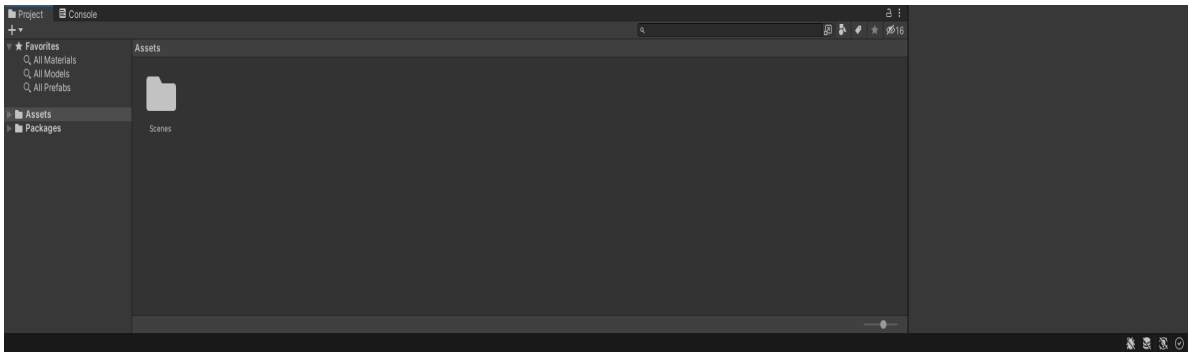
εκεί θα μπορούμε να δούμε οτιδήποτε έχουμε βάλει μέσα στο project μας . Ενώ αντίστοιχα το game (αυτό με το κόκκινο βελάκι) είναι αυτό το οποίο βλέπει η κάμερα μας η αντίστοιχα ο χαρακτήρας μέσα στο παιχνίδι μας.

Έπειτα στα αριστερά της οθόνης μας έχουμε ένα κουτάκι το οποίο ονομάζεται Hierarchy(ιεραρχία).Στην ουσία είναι ένας κατάλογος με τα αντικείμενα τα οποία εμείς έχουμε τοποθετήσει μέσα στο scene , τα οποία είναι συνδεδεμένα μεταξύ τους. Όταν επιλέξεις ένα αντικείμενο μέσα στο Hierarchy θα το επιλέξει και μέσα στο scene μας αυτόματα. Μέσα στην λίστα λοιπόν μπορείς να κάνεις drag and drop κάποιο αρχείο και έτσι μπορείς να τους αλλάξεις σειρά μεταξύ τους όσον αφορά την ιεραρχία τους.



Εικόνα 4.3.3:Hierarchy panel.

Στο κάτω μέρος του Unity υπάρχει το project panel.

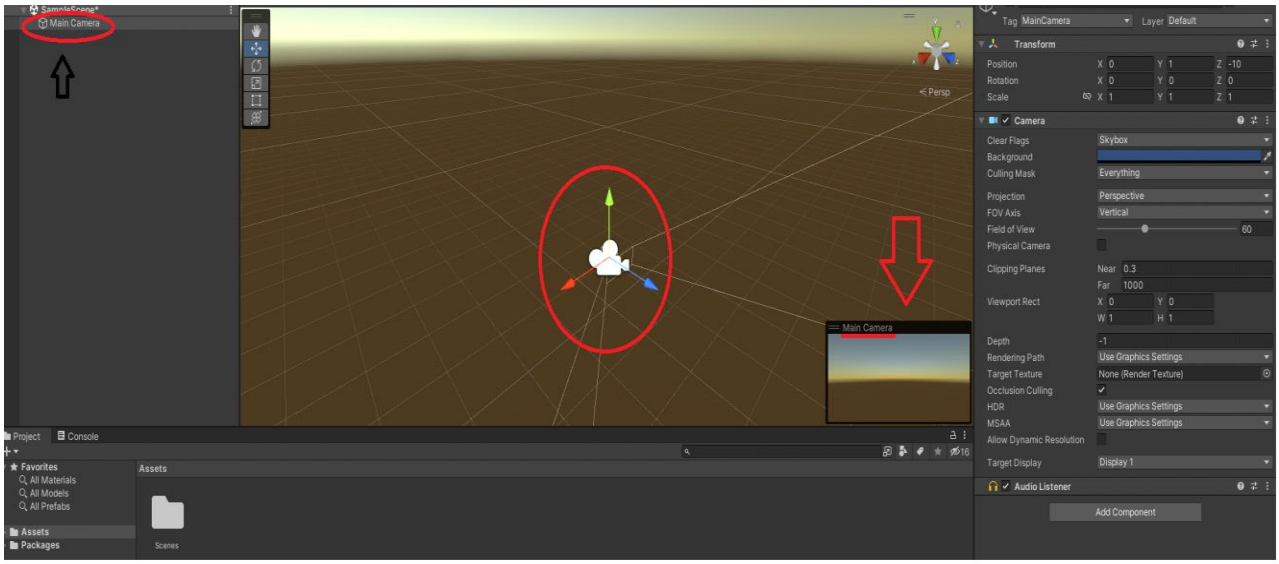


Εικόνα 4.3.4: Project panel.

Αυτό by default είναι χωρισμένο σε δυο μέρη , στα αριστερά έχουμε τα αρχεία μας ενώ στα δεξιά έχουμε κάποια εικονίδια. Στην ουσία το project panel είναι σαν μια αποθήκη οπού κρατάμε διάφορα πράγματα όπως πχ (μοντέλα και textures) τα οποία θα τα χρησιμοποιήσουμε κάποια στιγμή στο scene η σε κάποιο άλλο επίπεδο.

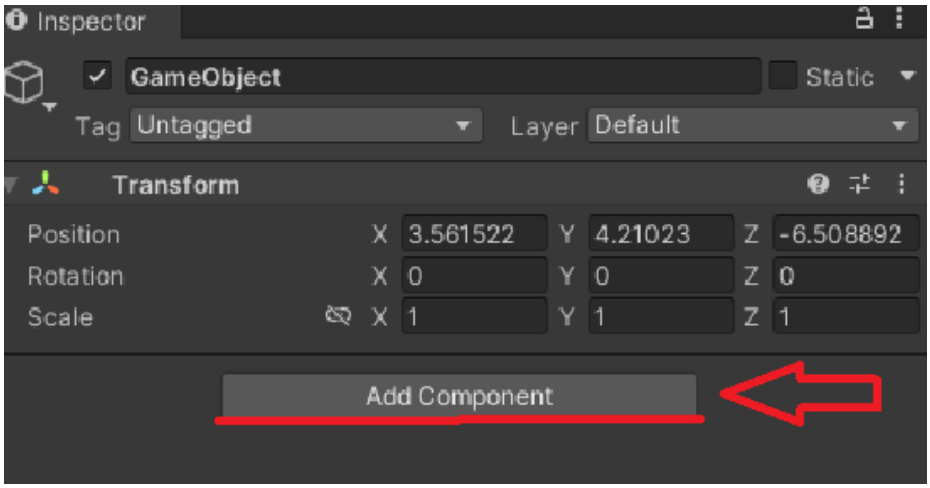
4.4 ΠΩΣ ΛΕΙΤΟΥΡΓΕΙ ΤΟ UNITY?

Στο unity λοιπόν, όλο το gameplay ενός παιχνιδιού, λαμβάνει χώρα στα **scenes.** Στην ουσία τα scenes είναι επίπεδα όπου όλα τα χαρακτηριστικά του παιχνιδιού μας, όπως τα game levels, ο τίτλος στην οθόνη η και το main menu περιέχονται. Ένα scene από προεπιλογή περιέχει μια camera object όπου θα ονομάζεται main camera. Είναι εφικτό να τοποθετήσουμε πάνω από μια κάμερα στο scenes μας. Το χαρακτηριστικό της main camera είναι ότι αποδίδει ότι βλέπει (ότι κάνει capture) σε μια συγκεκριμένη περιοχή, αυτό είναι το λεγόμενο viewpoint, σε αυτό το σημείο λοιπόν ότι έρθει σε επαφή , έχει σαν αποτέλεσμα να γίνεται ορατό στον παίχτη.



Εικόνα 4.4.1:Main camera.

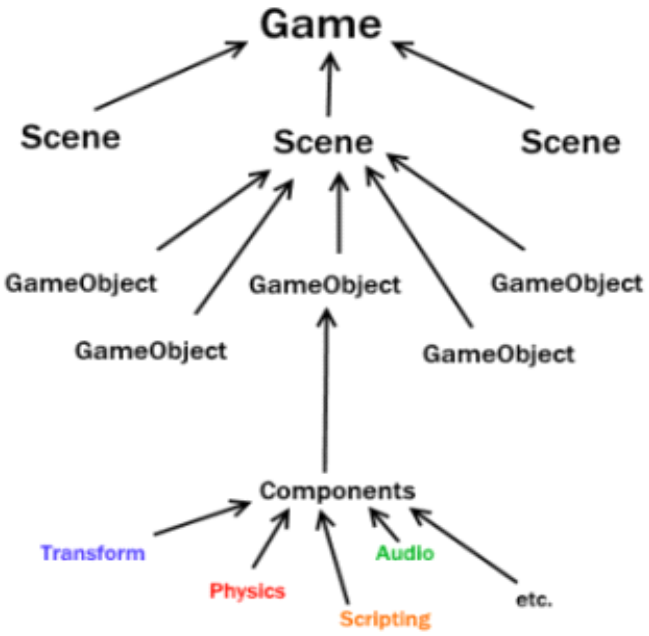
Ένα Scene αποτελείται από αντικείμενα, τα οποία ονομάζονται GameObjects. Τα gameobject μπορεί να είναι οτιδήποτε από το μοντέλο ενός παίχτη μέχρι και το GUI στην οθόνη, ή και τα κουμπιά. Τα gameobject λοιπόν έχουν ένα σετ από εξαρτήματα (components), ενωμένα πάνω τους, όπου περιγράφουν πως συμπεριφέρεται ένα scene. Το πιο σημαντικό εξάρτημα για κάθε Gameobject είναι το transform. Οποιοδήποτε αντικείμενο το οποίο υπάρχει σε ένα scene θα περιέχει ένα transform , το οποίο καθορίζει το position, το rotation, καθώς και το scale. Τα επιπλέον component μπορούν να ενωθούν σε οποιοδήποτε object, αν γίνει κλικ στο κουμπί Add Component, όπως βλέπουμε και στην εικόνα 4.4.2 και έπειτα επιλέξουμε το επιθυμητό component το οποίο θέλουμε.



Εικόνα 4.4.2: Κουμπί Add Component.

Παραδείγματα components:

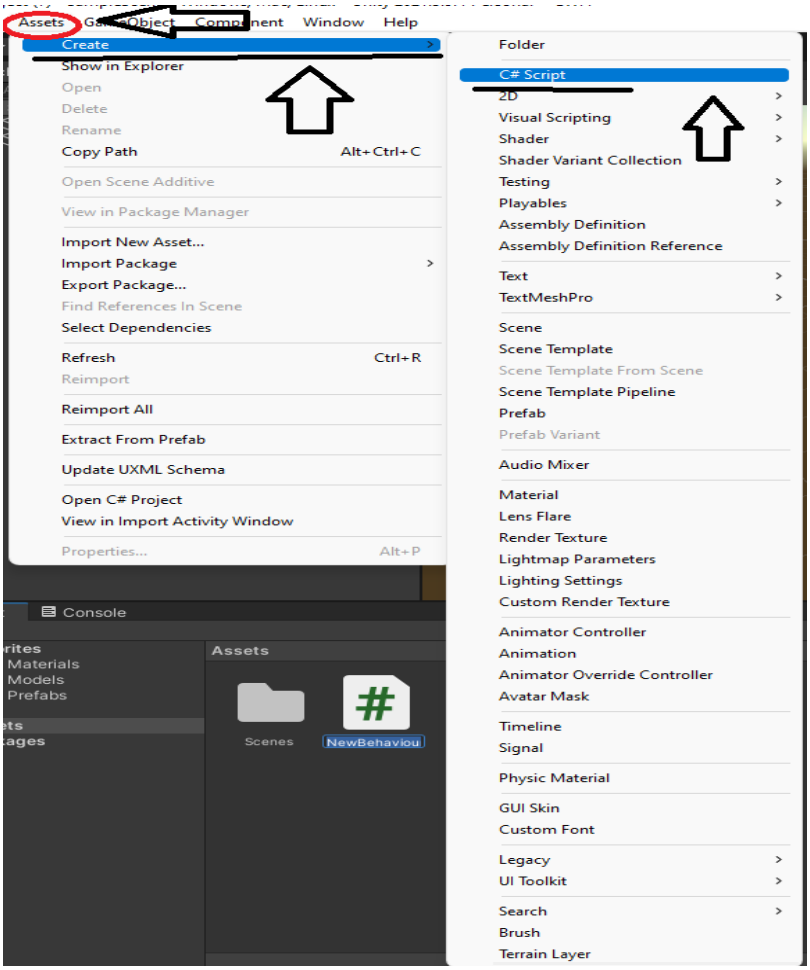
- **Renderer:** είναι υπεύθυνο για να κάνει rendering καθώς και για να κάνει ορατά τα objects.
- **Collider:** Καθορίζει τα φυσικά όρια σύγκρουσης για αντικείμενα, είναι πολύ σημαντικό καθώς χωρίς αυτό ο παίχτης μας μπορεί για παράδειγμα να περπατάει μέσα από τους τοίχους καθώς και μέσα από κάθε αντικείμενο.
- **Rigidbody:** Με αυτό το component το αντικείμενο μας αποκτάει φυσικές ιδιότητες σε πραγματικό χρόνο, όπως αυτό είναι το βάρος και η βαρύτητα.
- **Audio Source:** Δίνει στο αντικείμενο ιδιότητες αναπαραγωγής και αποθήκευσης μουσικής.
- **Audio Listener:** Έχει την ικανότητα να "ακούσει" το ήχο και να το μεταφέρει στα ηχεία του παίχτη.
- **Animator:** Δίνει την πρόσβαση στο animation system σε ένα αντικείμενο.
- **Light:** Κάνει το αντικείμενο να συμπεριφέρεται σαν μία πηγή φωτός, με ποικίλα από διαφορετικά εφέ.



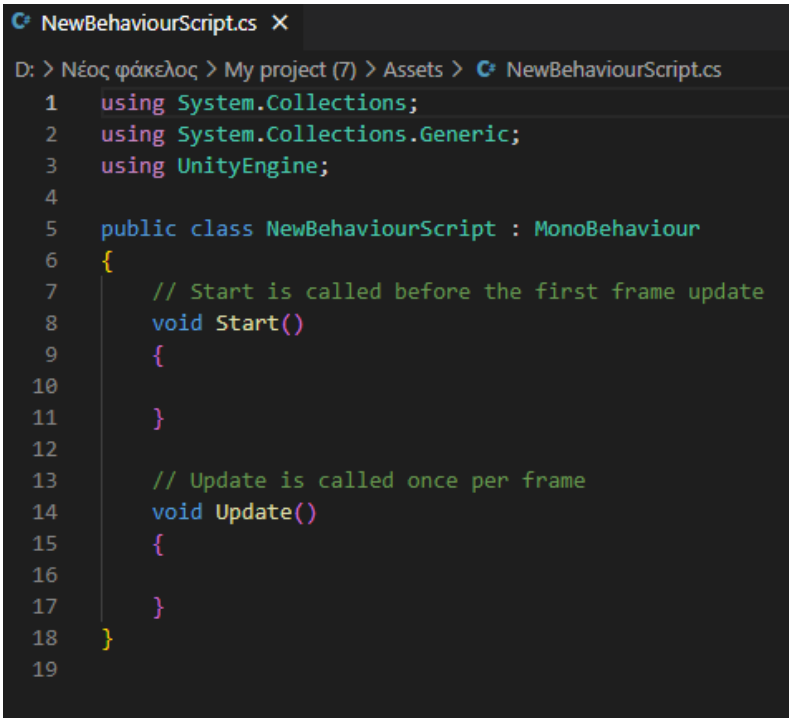
Εικόνα 4.4.3: Ιεραρχία παιχνιδιού, όπου τα components βρίσκονται στο κατώτερο επίπεδο.

Unity Internal Assets:

- **Materials:** Ορίζουν το πως ο φωτισμός θα επηρεάσει την εμφάνιση ενός αντικειμένου.
- **Scripts:** Ο κώδικας που θα γραφτεί για το GameObject.



Εικόνα 4.4.4:Παράδειγμα εισαγωγής script(κώδικα) C# στο Unity.



Εικόνα 4.4.5: Περιεχόμενα νέου αρχείου script.

Όταν δημιουργούμε λοιπόν ένα script,τον κώδικα στην εικόνα 4.4..5,τον περιέχει μέσα από προεπιλογή. Έτσι βλέπουμε μια κλάση ακολουθούμενη από το «MonoBehaviour». Το «MonoBehaviour» λοιπόν, είναι μια τεράστια βιβλιοθήκη κλάσεων και μεθόδων. Καθώς συνεχίζουμε, έχουμε δύο blocks κώδικα, τα οποία δεν έχουν κάποια μεταβλητή η οποία να γυρνάει, δηλαδή η μέθοδος Start, καθώς και η Update. Η μέθοδος Start τρέχει μια φορά για το πρώτο πλαίσιο για το gameobject όπου χρησιμοποιείται και είναι ενεργό στην σκηνή. Η μέθοδος Update, εκτελείται κάθε καρέ (frame) του παιχνιδιού μετά την μέθοδο Start. Τα παιχνίδια στο Unity εκτελούνται με ταχύτητα 60 FPS η καρέ ανά δευτερόλεπτο, πράγμα που σημαίνει ότι η μέθοδος Update καλείται 60 φορές το δευτερόλεπτο όσο το αντικείμενο είναι ακόμα ενεργό.

GAME VIEW:

Το παράθυρο αυτό στην ουσία μας δείχνει τι βλέπει η main camera μας(στην ουσία ότι υπάρχει , όταν πατήσουμε το κουμπί «Play», επίσης για παράδειγμα εάν υπάρχει ένα object στο scene μας το οποίο του έχουμε δώσει κάποια χαρακτηριστικά μέσω script και μέσω κάποιον animator, τότε όταν πατηθεί το κουμπί Play και μεταβούμε στο GameView , θα παρατηρήσουμε το object μας να μετακινείται ξαφνικά.

Εικόνα 4.4.6: Επιλογή μετάβασης στο Game View.

- **PLAY:** Όταν μεταβούμε στο Game View, τότε θα μας εμφανίσει αυτό του κουμπί. Όταν πατηθεί αυτό το κουμπί, τότε θα τρέξει το παιχνίδι μας.

Εικόνα 4.4.7: Κουμπί «Play».

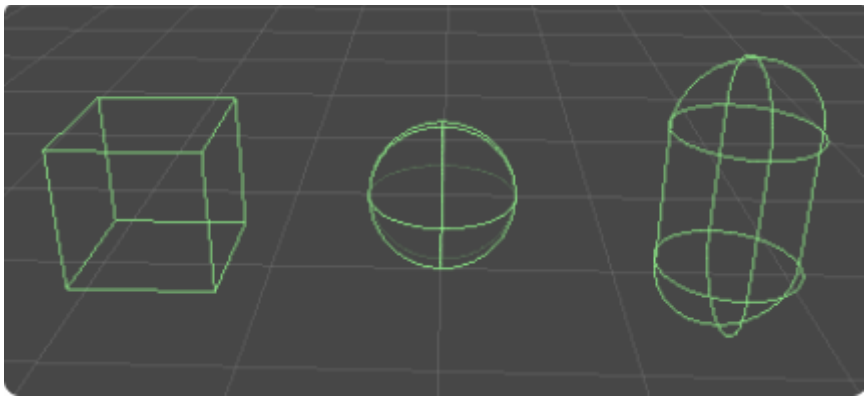
- **PAUSE:** Όταν πατηθεί, τότε το παιχνίδι μας θα παγώσει σε αυτό το συγκεκριμένο frame, αυτό βοηθάει κυρίως όταν θέλουμε να παρατηρήσουμε κάτι συγκεκριμένο στο παιχνίδι μας, η για παράδειγμα όταν θέλουμε να αλλάξουμε κάποιες τιμές στο παίχτη μας, πχ (speed).

Εικόνα 4.4.8: Κουμπί «Pause».

- **NEXT FRAME:** Όταν πατηθεί το κουμπί αυτό τότε μπορούμε να επεξεργαστούμε το κάθε frame του παιχνιδιού.

Εικόνα 4.4.9: Κουμπί «Next Frame».

4.5 UNITY COLLIDERS KAI TRIGGERS



Εικόνα 4.5.1: Τριών ειδών colliders.

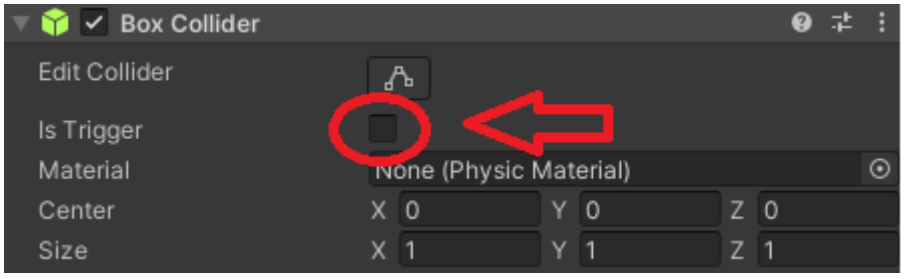
Ένα από τα βασικά και πιο σημαντικά πράγματα που συμβαίνουν σε ένα παιχνίδι, είναι τα events. Για παράδειγμα όταν ο χαρακτήρας μας (avatar/player), εισαχθεί σε ένα συγκεκριμένο χώρο/περιοχή, τότε ξαφνικά να αρχίσουν να γίνονται κάποια events αυτοματοποιημένα. Τέτοια events μπορούν να χρησιμοποιηθούν σε πολλές και διαφορετικές εφαρμογές όπως μια απλή ιδέα είναι, μετά την είσοδο του παίχτη σε έναν χώρο να δεχθεί ξαφνικά επίθεση από διάφορους χαρακτήρες με αυτοματοποιημένες ενέργειες, τα λεγόμενα NPCS.

Εικόνα 4.5.2: Παράδειγμα διαλόγου με NPC(event),αφού ο παίκτης εισαχθεί στην περιοχή που γίνεται trigger.

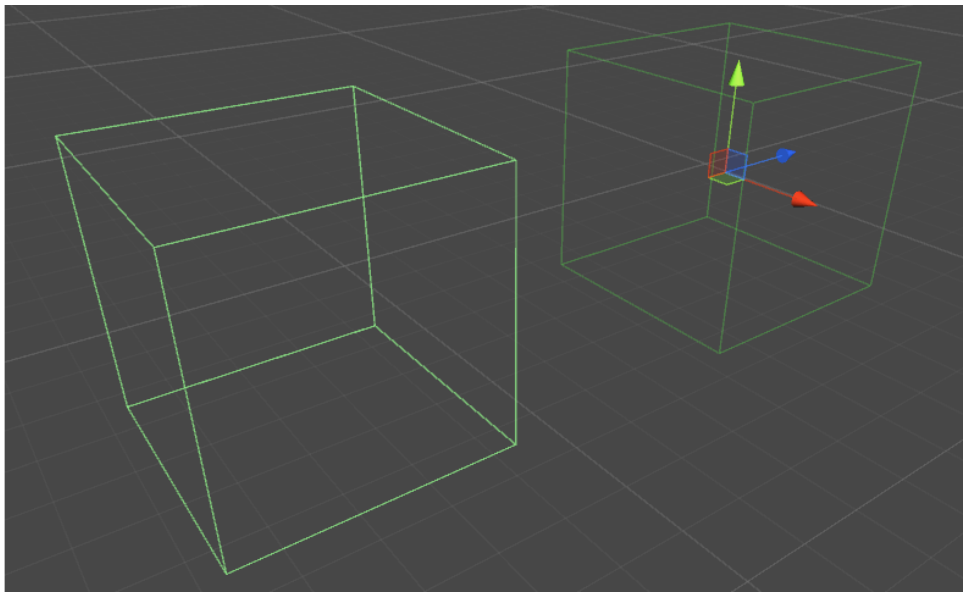
Τα colliders και τα triggers στην ουσία αποτελούν περιοχές στο χώρο του Unity(3D η 2D), τα οποία συνήθως εξυπηρετούν διάφορους σκοπούς σε ένα παιχνίδι. Η βασική διαφορά μεταξύ των δύο είναι ότι, τα collider αυτό που κάνουν είναι να εμποδίζουν στοιχεία με άλλα διαφορετικά collider να προσπεράσουν μία περιοχή, με άλλα λόγια πρόκειται για τους λεγόμενους αόρατους τοίχους στο unity. Από την άλλη αντίθετα τα triggers είναι αόρατες μικρές, είτε μεγάλες περιοχές πάνω στο περιβάλλον του unity, είναι αόρατες και για παράδειγμα όταν ο Player μας είναι μέσα σε αυτή την αόρατη περιοχή να έχει την δυνατότητα να κάνει κάποιες ενέργειες. Ως συμπέρασμα από αυτά βγάζουμε ότι, οι colliders υπάρχουν με σκοπό την προσομοίωση της σύγκρουσης 2 αντικειμένων, ενώ αντίθετα τα triggers έχουν να κάνουν με την μετάδοση της πληροφορίας των αντικειμένων, όταν αυτά εισέρχονται, μένουν η εξέρχονται από το συγκεκριμένο σημείο που έχουμε ορίσει ως trigger event. Αξίζει να σημειωθεί πως για να αναγνωρίσουν τα triggers ένα συγκεκριμένο αντικείμενο, αυτό το αντικείμενο θα πρέπει να έχει κάποιο collider πάνω του.

- **BOX COLLIDER:**

Πρόκειται για ορθογώνια κυβοειδή και είναι περισσότερο χρήσιμα για αντικείμενα όπως κιβώτια η σεντούκια. Για να γίνει εισαγωγή ενός collider πρέπει να υπάρχει το κατάλληλο component. Όπως βλέπουμε και στην παραπάνω εικόνα προκειμένου το συγκεκριμένο component να λειτουργεί ως trigger, θα πρέπει να γίνει checked το κουτάκι που γράφει «Is Trigger». Από την εικόνα βλέπουμε ότι υπάρχουν πεδία τα οποία ορίζουν τι περιοχή που θα καλύπτει ο συγκεκριμένος collider και υπάρχει και ένα πεδίο για τα physic materials. Τα physic Materials στην ουσία είναι στοιχεία που περιέχουν φυσικές ιδιότητες για έναν collider. Αντίστοιχα για 2D αντικείμενα, υπάρχουν αντίστοιχοι 2D colliders/triggers.



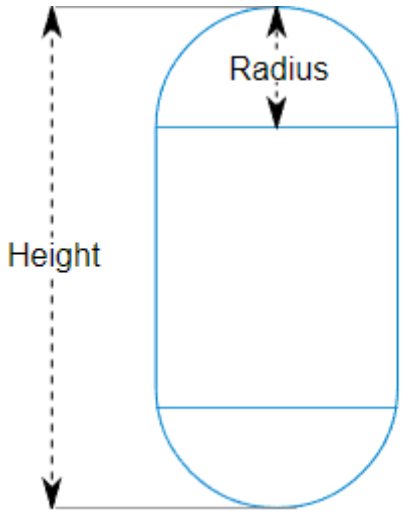
Εικόνα 4.5.3: Box Collider component με το κουμπί Boolean «Is Trigger».



Εικόνα 4.5.4 Box Collider.

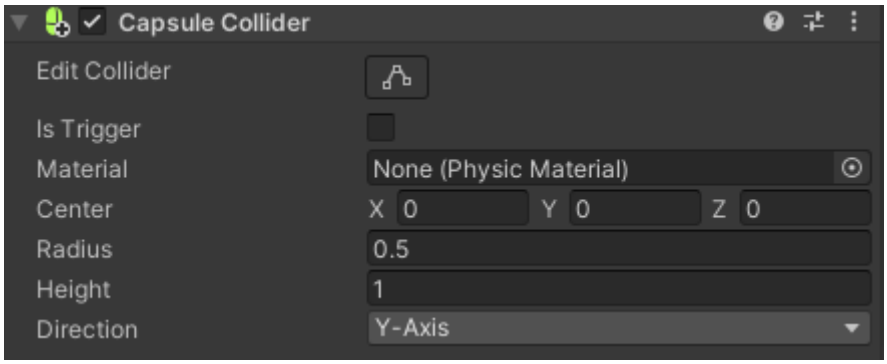
- **CAPSULE COLLIDER:**

Αποτελείται από δύο ημισφαίρια που ενώνονται μεταξύ τους με έναν κύλινδρο.

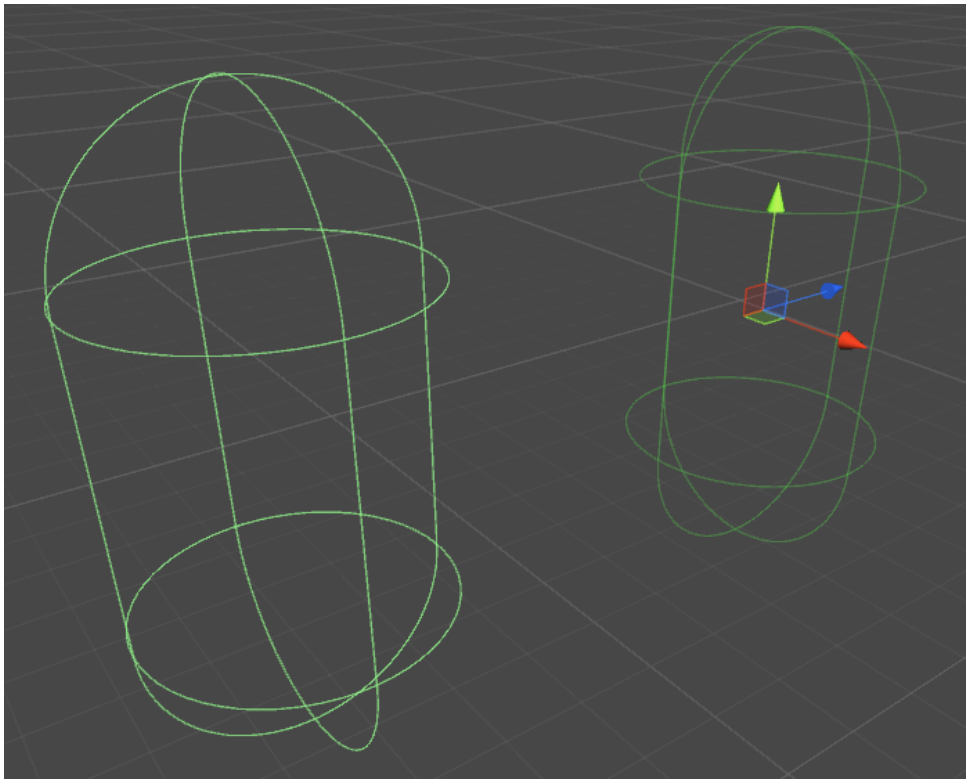


Εικόνα 4.5.5: Πως χωρίζεται το Capsule Collider.

Μπορούμε να ρυθμίσουμε την ακτίνα και το ύψος ενός capsule collider ανεξάρτητα το ένα από το άλλο. χρησιμοποιείται κυρίως στο character controller και λειτουργεί καλά για πόλους, καθώς και μπορεί να συνδυαστεί με άλλους colliders για ασυνήθιστα σχήματα.



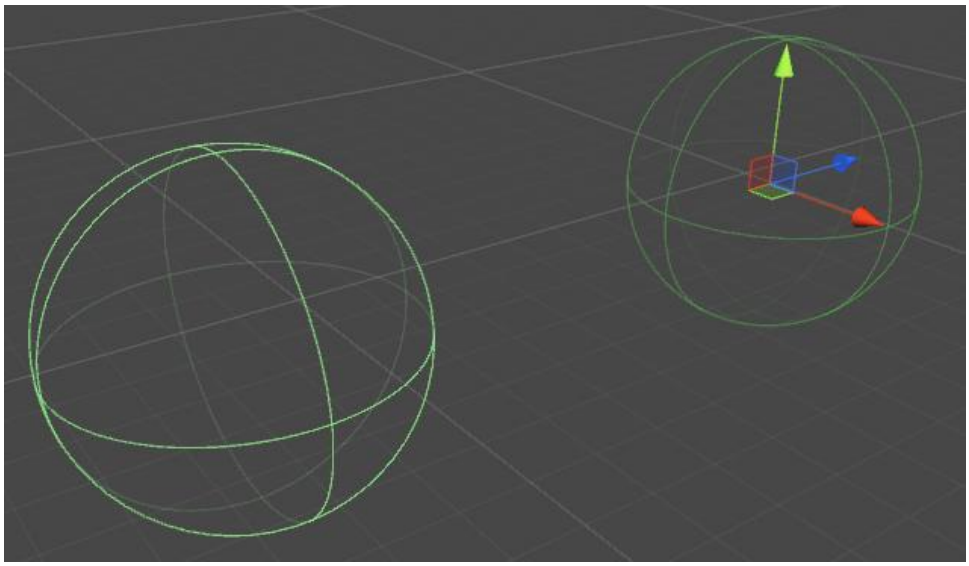
Εικόνα 4.5.6: Capsule Collider component.



Εικόνα 4.5.7: Capsule Collider.

- **SPHERE COLLIDER:**

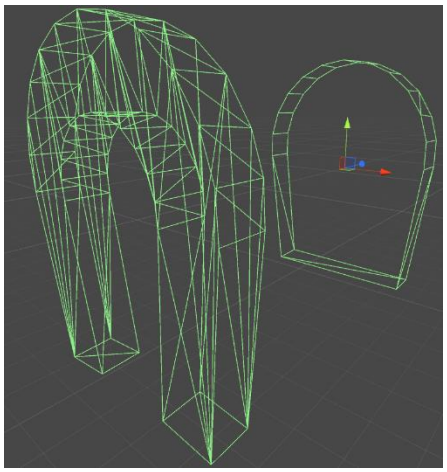
Το sphere collider πρόκειται για ένα σχήμα σφαίρας. Χρησιμοποιείται κυρίως για σφαιρικά αντικείμενα όπως μπάλες τένις, παρόλα αυτά λειτουργεί επίσης εξίσου καλά και για πέτρες που πέφτουν, η γενικά για αντικείμενα που πρέπει να πέσουν και να κυλήσουν.



Εικόνα 4.5.8:Sphere Collider.

- **MESH COLLIDER:**

Το mesh collider παίρνει ένα mesh asset και δημιουργεί τον collider του βασισμένο σε αυτό το mesh το οποίο είναι συνδεδεμένο στο game object, έτσι ώστε διαβάζει τις ιδιότητες του αντικειμένου που είναι κολλημένο προκειμένου να γίνει transform και να ρυθμίσει σωστά τη θέση και τη κλίμακα. Ένα από τα μεγάλα πλεονεκτήματα του είναι ότι μπορεί να κάνει το σχήμα του collider ακριβώς το ίδιο με το σχήμα του ορατού Mesh για το GameObject, με σκοπό την δημιουργία μεγαλύτερης ακρίβειας και καλύτερων συγκρούσεων, ωστόσο αυτή η ακρίβεια συνοδεύεται από υψηλότερο κόστος επεξεργασίας. Αυτό που πρέπει να κρατήσουμε είναι ότι, ανιχνεύει με πολύ μεγαλύτερη ακρίβεια μια σύγκρουση.



Εικόνα 4.5.9: Mesh Collider.

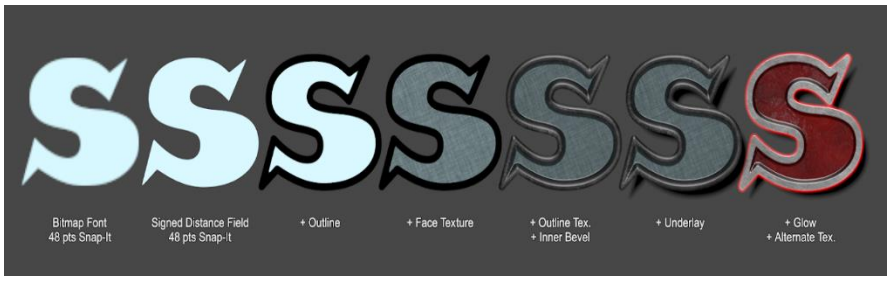
4.6 ΧΡΗΣΙΜΑ ΕΡΓΑΛΕΙΑ-ΕΦΑΡΜΟΓΕΣ ΓΙΑ ΤΟ UNITY

Σημαντικές εφαρμογές που χρειάζονται για ένα μεγάλο Project.

- **Text Mesh pro**

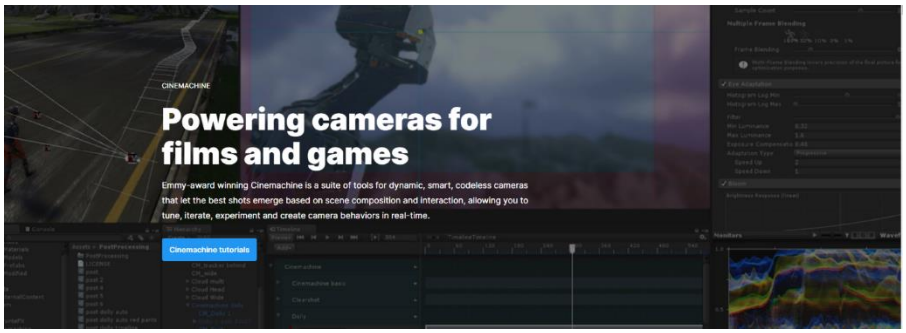
Μια από τις βασικές εφαρμογές που χρειάζονται και θα βοηθήσει πολλά άτομα είναι το text mesh pro. Το text mesh pro στην ουσία αυτό που κάνει είναι να χρησιμοποιεί προηγμένες τεχνικές απόδοσης κειμένου μαζί με ένα σύνολο προσαρμοσμένων shaders. Έτσι στην ουσία παρέχει ουσιαστικές βελτιώσεις στην οπτική ποιότητα παρέχοντας στους χρήστες απίστευτη ευελιξία όσον αφορά το στυλ κειμένου και την υφή.

Εικόνα 4.6.1: Logo Text Mesh Pro.



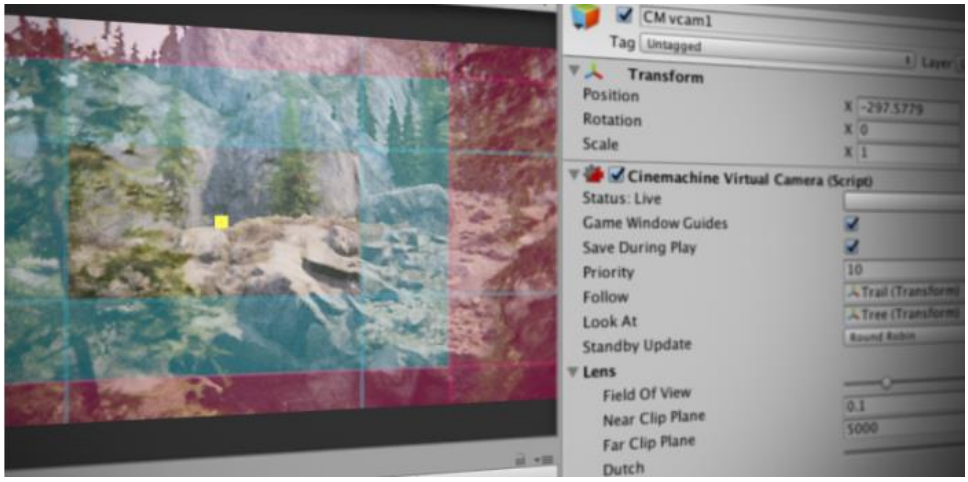
Εικόνα 4.6.2: Εικόνα ιδιότητας του εργαλείου.

- **Cinemachine**



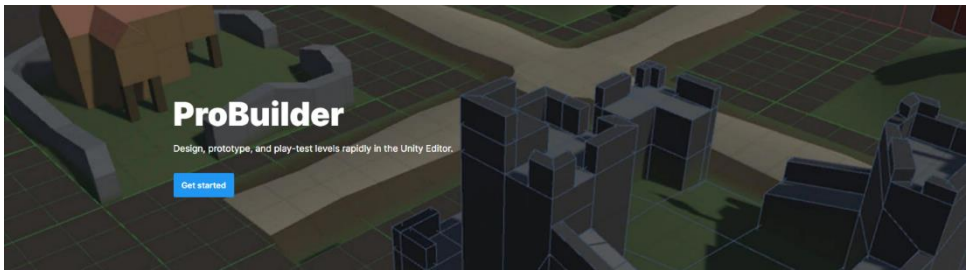
Εικόνα 4.6.3: Ιστοσελίδα CINEMACHINE.

Το cinemachine είναι μια σουίτα προτύπων για την λειτουργία της κάμερας unity. Αυτό που κάνει στην ουσία είναι να λύνει πολύπλοκα μαθηματικά, καθώς και της λογικής παρακολούθησης στόχων, της σύνθεσης, της ανάμειξης και της κοπής μεταξύ των λήψεων. Στην ουσία έχει σχεδιαστεί για να μειώνει σημαντικά τον αριθμό των χρονοβόρων εργασιών που καλούμαστε να κάνουμε πάνω σε ένα Project.

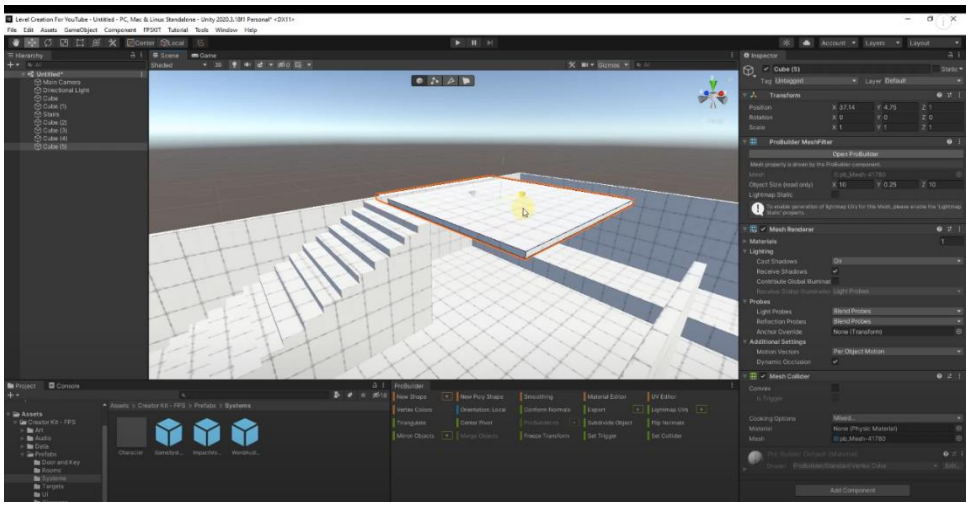


Εικόνα 4.6.4:UI από το Cinemachine

- Probuilder



Με την βοήθεια του probuilder είμαστε σε θέση να κάνουμε build, να επεξεργαστούμε και να δημιουργήσουμε υφή προσαρμοσμένης γεωμετρίας με τις ενέργειες και τα εργαλεία που είναι διαθέσιμα στο πακέτο probuilder. Επίσης μπορεί να βοηθήσει στην σχεδίαση σε επίπεδο σκηνής, τη δημιουργία πρωτοτύπων, τα πλέγματα σύγκρουσης και τη δοκιμή αναπαραγωγής.



Εικόνα 4.6.5: Λειτουργίες Pro Builder.

- **URP and HDRP**

Εικόνα 4.6.6: Διάφοροι Renderers.

Το unity μας αφήνει να δημιουργήσουμε URP(Universal Render Pipeline) η HDRP(High Definition Render) Project προκειμένου να εστιάσουμε είτε στην απόδοση είτε στην ποιότητα του rendering.

5 ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ ΣΤΟ UNITY

5.1 ΠΕΡΙΛΗΨΗ ΠΑΙΧΝΙΔΙΟΥ

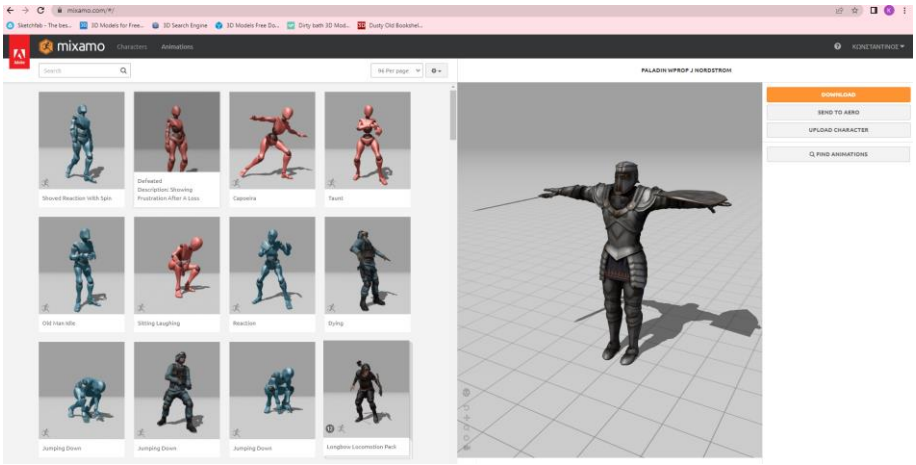
Αναπτύξαμε ένα 3D puzzle game, το οποίο είναι Third-Person,με βασικό χαρακτήρα ένα έφηβο αγόρι, όπου βρίσκεται σε ένα σπίτι. Ο χαρακτήρας μας έχει σκοπό να βρει αντικείμενα, τα οποία είναι στοιχεία για να μπορέσει να περάσει από το κάθε δωμάτιο προκειμένου να βρει την έξοδο του.

5.2 ΕΞΑΡΤΗΜΑΤΑ ΓΙΑ ΤΗΝ ΜΕΤΑΚΙΝΗΣΗ ΤΟΥ ΧΑΡΑΚΤΗΡΑ

Σε αυτό το κεφάλαιο θα εφαρμόσουμε κατάλληλα components έτσι ώστε να μετακινηθεί ο χαρακτήρας μας. Καθώς πρόκειται για ένα 3D παιχνίδι ,οι άξονες όπου χρησιμοποιούνται είναι τρείς. Με την μετακίνηση του χαρακτήρα ακολουθεί η κάμερα(main camera),σε θέση ώστε το παιχνίδι μας να είναι Third-Person. Η μετακίνηση του χαρακτήρα γίνεται μέσω script,ενώ ταυτόχρονα συνεργάζεται με έναν animator,όπου διαχειρίζεται τα διάφορα animations ανάλογα με τα events που συμβαίνουν, για να επιτύχουμε την φυσική κίνηση ενός ανθρώπου(του χαρακτήρα μας).Έπειτα θα αναφέρουμε τα colliders,για το πως επηρεάζουν την μετακίνηση του χαρακτήρα, όταν συγκρούεται με ένα αντικείμενο και πως το αντιμετωπίσαμε ως πρόβλημα.

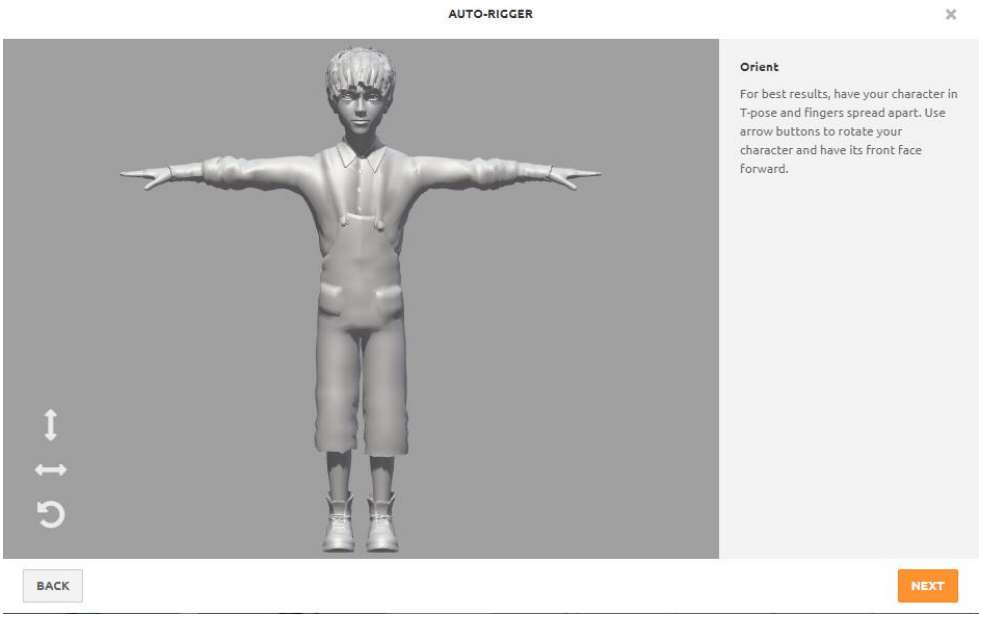
Κάθε χαρακτήρας, για τη κίνηση του(animations),χρειάζεται να εφαρμόσουμε έναν σκελετό, έτσι ώστε να εισαχθεί στο Unity.Υπάρχουν διάφοροι τρόποι για να το κάνουμε αυτό, είτε χειροκίνητοι, αλλά υπάρχουν και εφαρμογές που δημιουργούν τον σκελετό αυτόματα. Μια γνωστή ιστοσελίδα που μπορούμε να «πακετάρουμε» animations με τον αυτόματο σκελετό του χαρακτήρα, είναι η mixamo, όπου και εμείς προσαρμόσαμε σκελετό και animations από αυτήν. Κατά την εισαγωγή του χαρακτήρα στο Unity,με βάση τον σκελετό, δημιουργεί ένα avatar, όπου χρησιμοποιείται από τα animations.

Εικόνα 5.2.1: Λογότυπο mixamo.

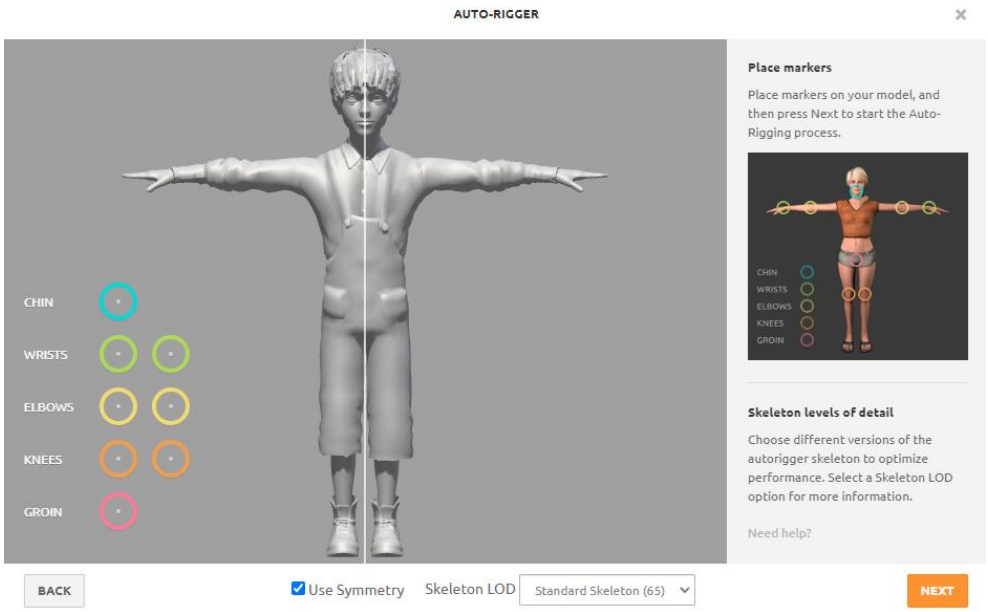


Εικόνα 5.2.2: Ιστοσελίδα mixamo.com.

Για την εισαγωγή του δικού μας χαρακτήρα κάνουμε «UPLOAD CHARACTER» όπως βλέπουμε στα δεξιά της εικόνας 5.2.2 και έπειτα μας εμφανίζει το παράθυρο «AUTO RIGGER»(εικόνα 5.2.3).

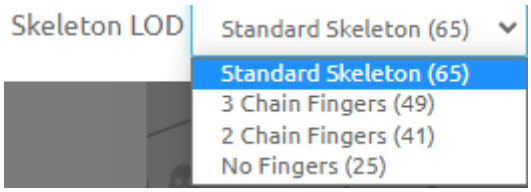


Εικόνα 5.2.3: Πρώτο παράθυρο «AUTO RIGGER» όπου μπορούμε να κάνουμε rotate στους 3 άξονες.



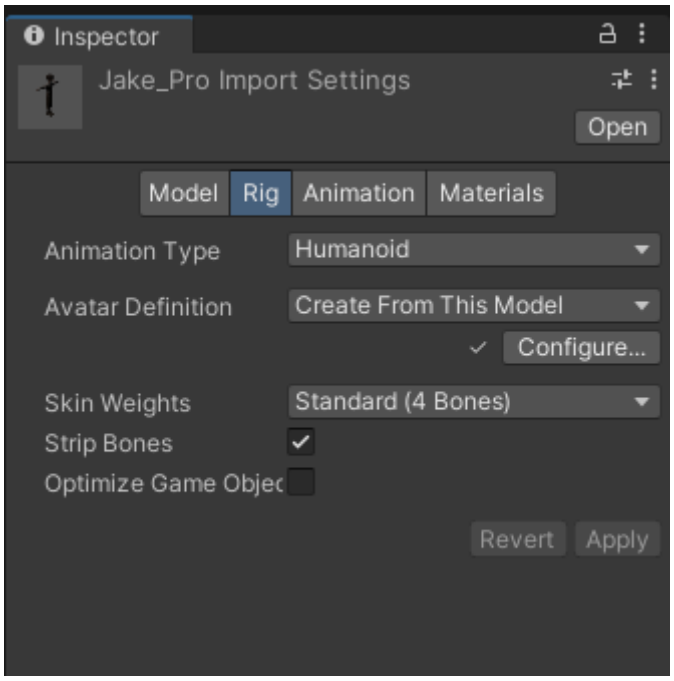
Εικόνα 5.2.4: Δεύτερο παράθυρο «AUTO RIGGER» οδηγός για τον σκελετό.

Στην εικόνα 5.2.4 στα δεξιά βλέπουμε έναν οδηγό για το πως πρέπει να είναι τα σημεία αναφοράς στο σώμα του χαρακτήρα. Στο κάτω μέρος αν μας διευκολύνει μπορούμε να επιλέξουμε το «Use Symmetry» ,έτσι ώστε αν μετακινήσουμε ένα σημείο αναφοράς πχ στο δεξί καρπό, με την βοήθεια της συμμετρίας, θα βάλει το αριστερό σημείο αναφοράς στον αριστερό καρπό. Και τέλος, στο «Skeleton LOD» επιλέγουμε ένα πρότυπο, από πόσα κόκκαλα θα αποτελείται ο σκελετός μας.



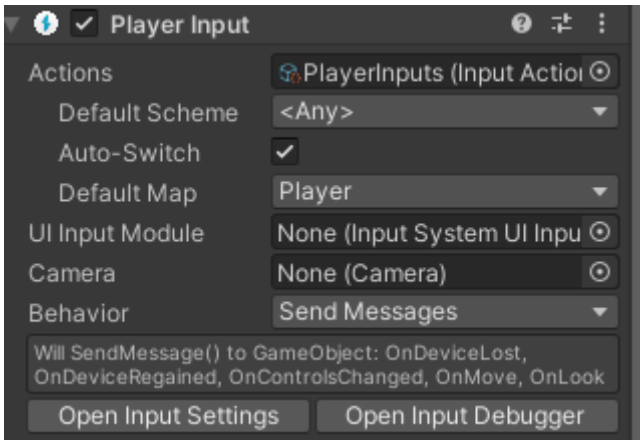
Εικόνα 5.2.5: Skeleton LOD επιλογές.

Όταν ο χαρακτήρας μας εισάγεται στο Unity,πρέπει να εισαχθεί ο σκελετός και να αλλάξουμε το είδος του animation σε humanoid.Αφού επιλέξουμε το asset του χαρακτήρα μας, στο panel «inspector»(δεξιά),μας εμφανίζει διάφορες επιλογές – ρυθμίσεις και καρτέλες που μπορούμε να προηγηθούμε. Μεταβαίνουμε στην καρτέλα «Rig»,έπειτα αλλάζουμε το animation type σε humanoid, πατάμε «apply» και δημιουργείται το avatar.



Εικόνα 5.2.6: Inspector panel ,RIG - Animation Type -> Humanoid.

Επίσης για πάρουμε την είσοδο από τον χρήστη, στο παιχνίδι μας χρησιμοποιήσαμε το νέο Input System.Αρχικά για να μπορέσουμε να το χρησιμοποιήσουμε πηγαίνουμε στο package mananger και αναζητούμε «Input System» και το εγκαθιστούμε στο Unity.Αυτόματα με αυτόν τον τρόπο έχουμε την επιλογή να προσθέσουμε ένα νέο component,το «Player Input».

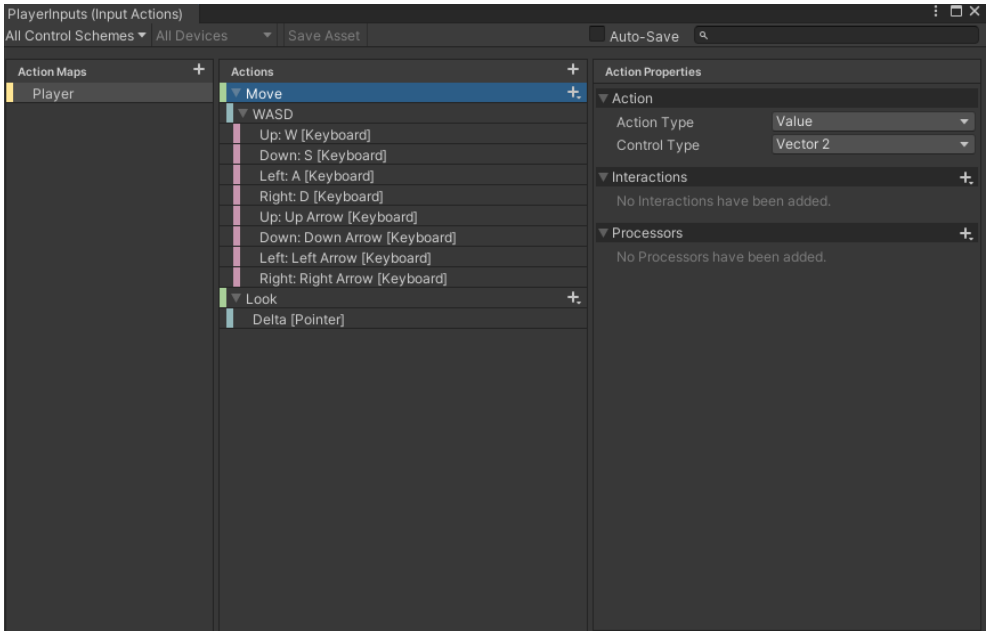


Εικόνα 5.2.7:Player Input component.

Εν συνεχεία για να το χρησιμοποιήσουμε στο πεδίο Actions θα πρέπει να βάλουμε ένα Input Actions Asset.Για να δημιουργήσουμε αυτό το asset επιλέγουμε Assets->Create->Input Actions και του ορίζουμε ένα όνομα.

Εικόνα 5.2.8: Input Actions Asset.

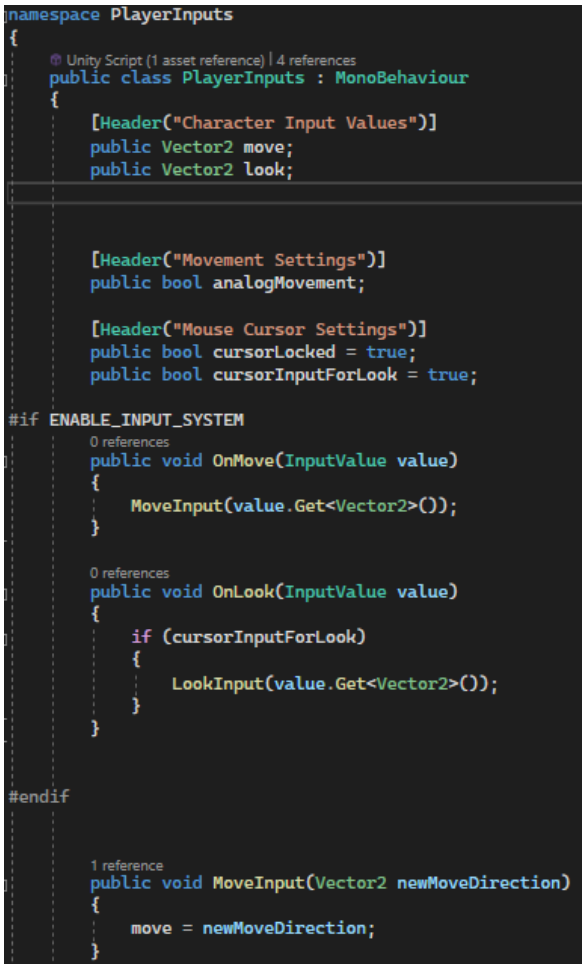
Όταν πατήσουμε διπλό κλικ επάνω του, θα μας εμφανίσει ένα παράθυρο, όπου εκεί μέσα μπορούμε να ορίσουμε τα Inputs.



Εικόνα 5.2.9 : Input Actions properties.

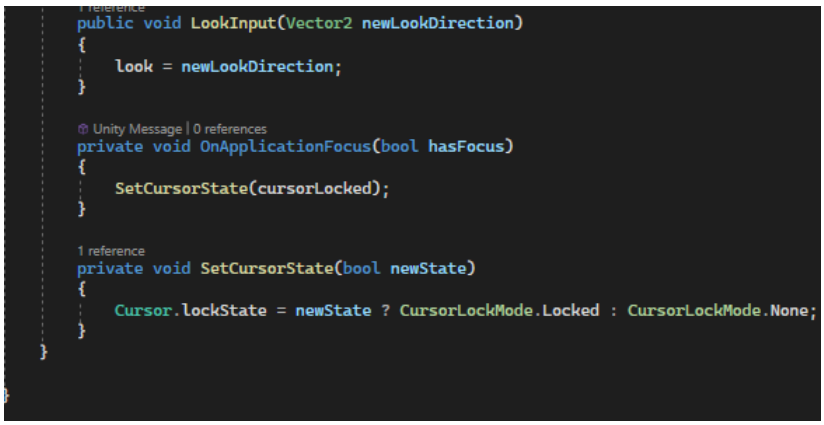
Σε αυτό το παράθυρο μπορούμε να αντιστοιχίσουμε τα Inputs με διάφορες λειτουργίες. Αρχικά μπορούμε να ομαδοποιήσουμε τα actions στα Action Maps(αριστερό panel),όπως βλέπουμε εμείς έχουμε το Action Map «Player».Έπειτα στο μεσαίο panel μπορούμε δημιουργήσουμε τα actions.Τέλος, στο δεξί panel ορίζουμε τις ιδιότητες ενός action.

Για να διαβάσουμε την είσοδο από τον χρήστη, χρησιμοποιούμε ένα συνοδευτικό script για το κάθε action,όπου μπορούμε να προσπελάσουμε από άλλα scripts για να πάρουμε την τιμή τους. Αρχικά στο συνοδευτικό script πρέπει να συμπεριλάβουμε στις βιβλιοθήκες το «UnityEngine.InputSystem».Ένα παράδειγμα από το συνοδευτικό script θα δούμε παρακάτω στις εικόνες 5.2.10 και 5.2.11.



```
namespace PlayerInputs
{
    Unity Script (1 asset reference) | 4 references
    public class PlayerInputs : MonoBehaviour
    {
        [Header("Character Input Values")]
        public Vector2 move;
        public Vector2 look;



        [Header("Movement Settings")]
        public bool analogMovement;

        [Header("Mouse Cursor Settings")]
        public bool cursorLocked = true;
        public bool cursorInputForLook = true;

#if ENABLE_INPUT_SYSTEM
        0 references
        public void OnMove(InputValue value)
        {
            MoveInput(value.Get<Vector2>());
        }

        0 references
        public void OnLook(InputValue value)
        {
            if (cursorInputForLook)
            {
                LookInput(value.Get<Vector2>());
            }
        }


#endif



        1 reference
        public void MoveInput(Vector2 newMoveDirection)
        {
            move = newMoveDirection;
        }
```

Εικόνα 5.2.10: PlayerInputs script μέρος 1.

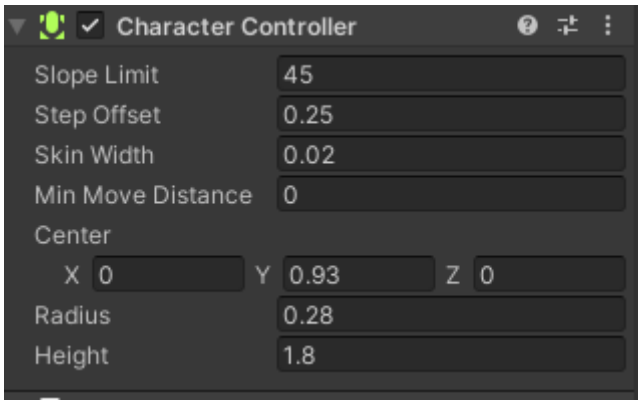


```
        public void LookInput(Vector2 newLookDirection)
        {
            look = newLookDirection;
        }

        Unity Message | 0 references
        private void OnApplicationFocus(bool hasFocus)
        {
            SetCursorState(cursorLocked);
        }

        1 reference
        private void SetCursorState(bool newState)
        {
            Cursor.lockState = newState ? CursorLockMode.Locked : CursorLockMode.None;
        }
    }
}
```

Εικόνα 5.2.11: PlayerInputs script μέρος 2.

Στην ουσία πρόκειται για ένα αλυσιδωτό script όπου διαβάζει την είσοδο και την αποθηκεύει σε μια μεταβλητή όπου θα την χρησιμοποιήσουμε αργότερα για να «διαβάσουμε» την τιμή της.

Στην συνέχεια θα χρειαστούμε ένα επιπλέον component το οποίο λέγεται «Character Controller».Το Character Controller στην δικιά μας περίπτωση είναι υπεύθυνο για την μετακίνηση του χαρακτήρα μας και μέσω του script θα το προσπελάσουμε έτσι ώστε να το επιτύχουμε. Στην εικόνα 5.2.12 θα δούμε το Character Controller και τις ρυθμίσεις που μπορούμε να κάνουμε σε αυτό, όσον αφορά το μέγεθος του έτσι ώστε να είναι στο μέγεθος του κορμού του χαρακτήρα μας.



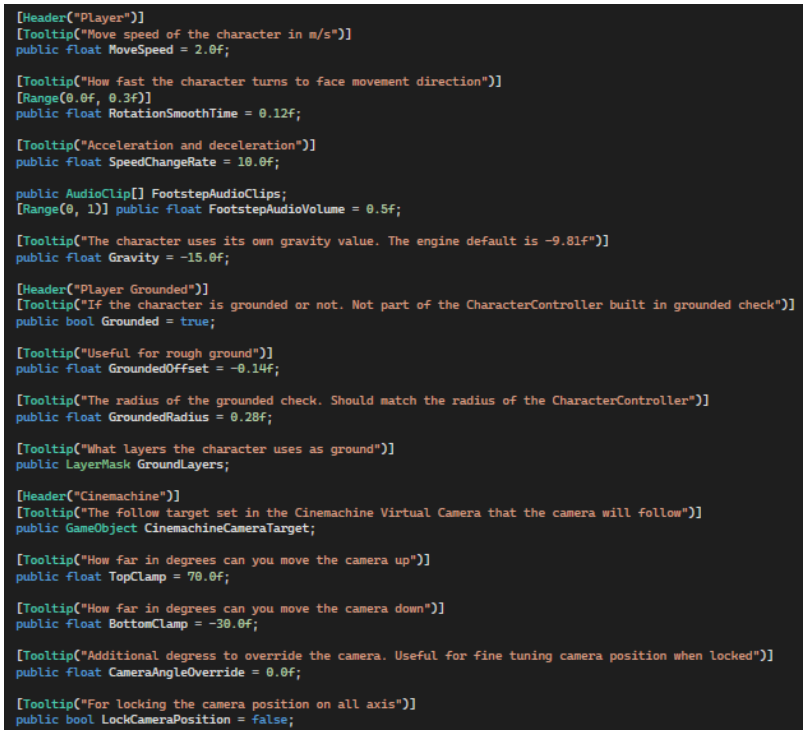
Εικόνα 5.2.12: Character Controller component.

5.3 ΜΕΤΑΚΙΝΗΣΗ ΤΟΥ ΧΑΡΑΚΤΗΡΑ

Σε αυτό το κεφάλαιο θα δούμε πώς τα components που αναφέραμε θα συνδυαστούν μεταξύ τους για να επιτύχουμε το αποτέλεσμα. Θα εξετάσουμε την δομή του βασικού script οπού είναι υπεύθυνο για την μετακίνηση.

Εικόνα 5.3.1:Script Component.

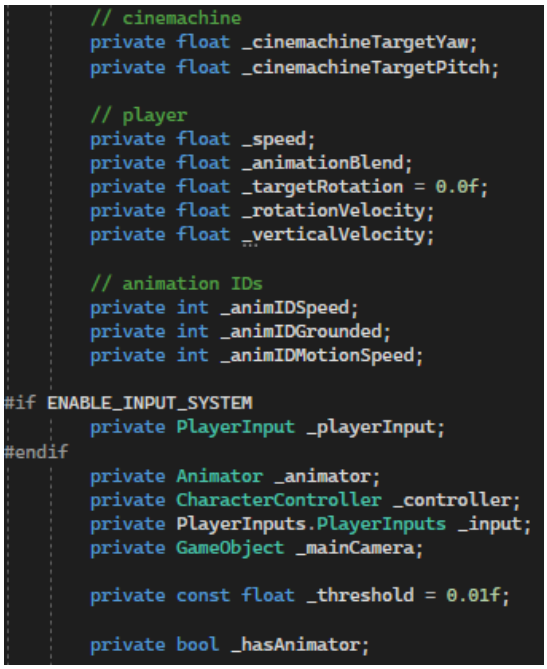
Αρχικά στο script αυτό θα ορίσουμε τις μεταβλητές που θα χρησιμοποιήσουμε. Στην εικόνα 5.3.1 βλέπουμε διάφορες επιλογές οι οποίες αντιστοιχούν σε μεταβλητές μέσα στον κώδικα.



```
[Header("Player")]
[Tooltip("Move speed of the character in m/s")]
public float MoveSpeed = 2.0f;

[Tooltip("How fast the character turns to face movement direction")]
[Range(0.0f, 0.3f)]
public float RotationSmoothTime = 0.12f;

[Tooltip("Acceleration and deceleration")]
public float SpeedChangeRate = 10.0f;

public AudioClip[] FootstepAudioClips;
[Range(0, 1)] public float FootstepAudioVolume = 0.5f;

[Tooltip("The character uses its own gravity value. The engine default is -9.81f")]
public float Gravity = -15.0f;

[Header("Player Grounded")]
[Tooltip("If the character is grounded or not. Not part of the CharacterController built in grounded check")]
public bool Grounded = true;

[Tooltip("Useful for rough ground")]
public float GroundedOffset = -0.14f;

[Tooltip("The radius of the grounded check. Should match the radius of the CharacterController")]
public float GroundedRadius = 0.28f;

[Tooltip("What layers the character uses as ground")]
public LayerMask GroundLayers;

[Header("Cinemachine")]
[Tooltip("The follow target set in the Cinemachine Virtual Camera that the camera will follow")]
public GameObject CinemachineCameraTarget;

[Tooltip("How far in degrees can you move the camera up")]
public float TopClamp = 70.0f;

[Tooltip("How far in degrees can you move the camera down")]
public float BottomClamp = -30.0f;

[Tooltip("Additional degress to override the camera. Useful for fine tuning camera position when locked")]
public float CameraAngleOverride = 0.0f;

[Tooltip("For locking the camera position on all axis")]
public bool LockCameraPosition = false;
```

Εικόνα 5.3.2: Εξωτερικές (ορατές, public) μεταβλητές στο Inspector panel.

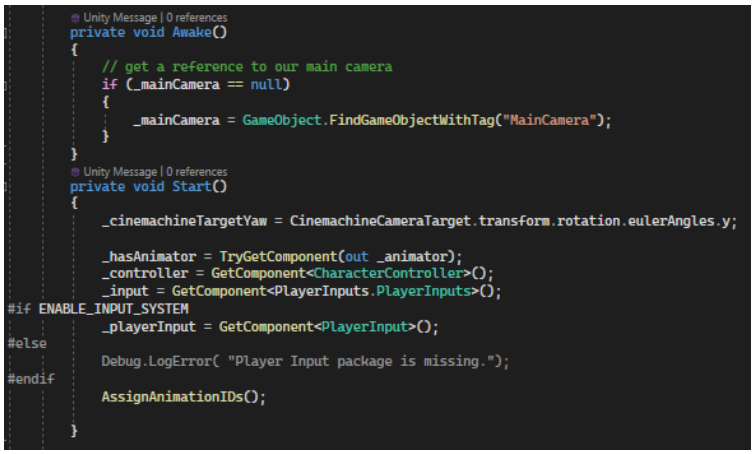
Έπειτα ορίζουμε τις εσωτερικές(μη ορατές,private) μεταβλητές του κώδικα.



```
        // cinemachine
        private float _cinemachineTargetYaw;
        private float _cinemachineTargetPitch;

        // player
        private float _speed;
        private float _animationBlend;
        private float _targetRotation = 0.0f;
        private float _rotationVelocity;
        private float _verticalVelocity;

        // animation IDs
        private int _animIDSpeed;
        private int _animIDGrounded;
        private int _animIDMotionSpeed;

#if ENABLE_INPUT_SYSTEM
        private PlayerInput _playerInput;
#endif
        private Animator _animator;
        private CharacterController _controller;
        private PlayerInputs.PlayerInputs _input;
        private GameObject _mainCamera;

        private const float _threshold = 0.01f;

        private bool _hasAnimator;
```

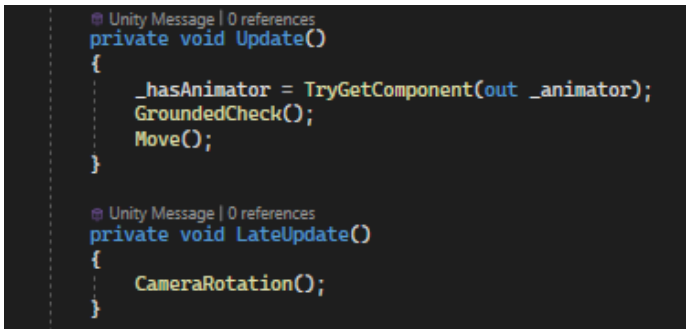
Εικόνα 5.3.3: Εσωτερικές(μη ορατές, private) μεταβλητές του κώδικα.

Στην συνέχεια στην μέθοδο Awake ψάχνουμε το game object όπου έχει το tag «MainCamera».Στην μέθοδο Start χρειάζεται να πάρουμε τα components όπου είναι στον χαρακτήρα μας και να τα αποθηκεύσουμε σε μια μεταβλητή. Επίσης η μέθοδος Start καλεί μια άλλη μέθοδο όπου χρειάζεται για το Animator component.

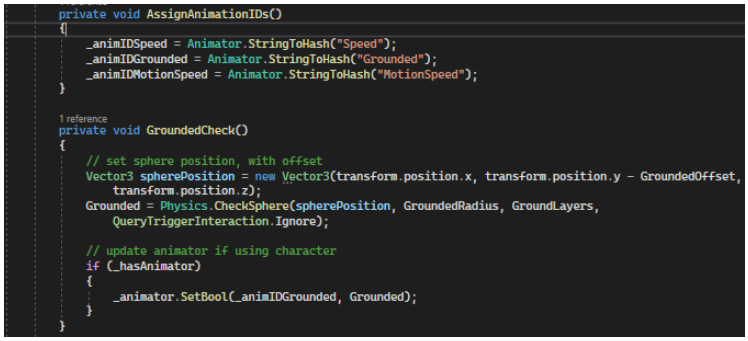


Εικόνα 5.3.4: Awake και Start.

Στην μέθοδο update παίρνει την έξοδο του animator και καλεί δυο μεθόδους GroundCheck και Move.Στην μέθοδο LateUpdate καλεί την CameraRotation.



Εικόνα 5.3.5: Update και LateUpdate.



Εικόνα 5.3.6: AssignAnimationIDs ,GroundCheck.

Στο σημείο αυτό θα δούμε το βασικότερο κομμάτι του Script για την μετακίνηση. Η μέθοδος move θα δούμε ότι χρησιμοποιεί μεταβλητές που ανήκουν σε όλα τα components που αναφέραμε.

```
private void Move()
{
    // set target speed based on move speed, sprint speed and if sprint is pressed
    float targetSpeed = MoveSpeed;

    // a simplistic acceleration and deceleration designed to be easy to remove, replace, or iterate upon

    // note: Vector2's == operator uses approximation so is not floating point error prone, and is cheaper than magnitude
    // if there is no input, set the target speed to 0
    if (_input.move == Vector2.zero) {
        targetSpeed = 0.0f;
    }

    // a reference to the players current horizontal velocity
    float currentHorizontalSpeed = new Vector3(_controller.velocity.x, 0.0f, _controller.velocity.z).magnitude;

    float speedOffset = 0.1f;
    float inputMagnitude = _input.analogMovement ? _input.move.magnitude : 1f;

    // accelerate or decelerate to target speed
    if (currentHorizontalSpeed < targetSpeed - speedOffset ||
        currentHorizontalSpeed > targetSpeed + speedOffset)
    {
        // creates curved result rather than a linear one giving a more organic speed change
        // note T in Lerp is clamped, so we don't need to clamp our speed
        _speed = Mathf.Lerp(currentHorizontalSpeed, targetSpeed * inputMagnitude,
            Time.deltaTime * SpeedChangeRate);

        // round speed to 3 decimal places
        _speed = Mathf.Round(_speed * 1000f) / 1000f;
    }
    else
    {
        _speed = targetSpeed;
    }
```

Εικόνα 5.3.7: Μέθοδος Move μέρος 1.

```
    _animationBlend = Mathf.Lerp(_animationBlend, targetSpeed, Time.deltaTime * SpeedChangeRate);
    if (_animationBlend < 0.01f) _animationBlend = 0f;

    // normalise input direction
    Vector3 inputDirection = new Vector3(_input.move.x, 0.0f, _input.move.y).normalized;

    // note: Vector2's != operator uses approximation so is not floating point error prone, and is cheaper than magnitude
    // if there is a move input rotate player when the player is moving
    if (_input.move != Vector2.zero)
    {


        _targetRotation = Mathf.Atan2(inputDirection.x, inputDirection.z) * Mathf.Rad2Deg +
                          _mainCamera.transform.eulerAngles.y;
        float rotation = Mathf.SmoothDampAngle(transform.eulerAngles.y, _targetRotation, ref _rotationVelocity,
            RotationSmoothTime);

        // rotate to face input direction relative to camera position
        transform.rotation = Quaternion.Euler(0.0f, rotation, 0.0f);
    }


    Vector3 targetDirection = Quaternion.Euler(0.0f, _targetRotation, 0.0f) * Vector3.forward;

    // move the player
    _controller.Move(targetDirection.normalized * (_speed * Time.deltaTime) +
                     new Vector3(0.0f, _verticalVelocity, 0.0f) * Time.deltaTime);

    // update animator if using character
    if (_hasAnimator)
    {
        _animator.SetFloat(_animIDSpeed, _animationBlend);
        _animator.SetFloat(_animIDMotionSpeed, inputMagnitude);
    }
}
```

Εικόνα 5.3.8:Μέθοδος Move μέρος 2.

Για την περιστροφή της κάμερας χρησιμοποιείται ένα game object όπου βρίσκεται στον χαρακτήρα μας σαν «παιδί» του στο Hierarchy panel.Αυτό το game object είναι το σημείο από το οποίο η κάμερα περιστρέφεται. Στο παρακάτω script βλέπουμε πως επιτυγχάνεται η περιστροφή της μέσω του ποντικιού.

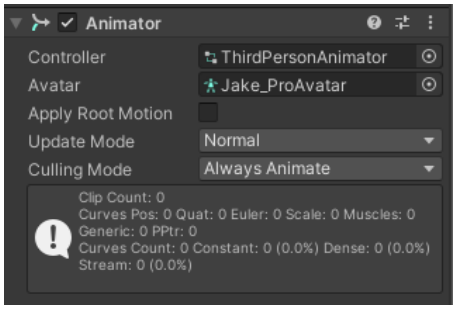
```
1 reference
private void CameraRotation()
{
    // if there is an input and camera position is not fixed
    if (_input.look.sqrMagnitude >= _threshold && !LockCameraPosition)
    {
        //Don't multiply mouse input by Time.deltaTime;
        float deltaTimeMultiplier = IsCurrentDeviceMouse ? 1.0f : Time.deltaTime;

        _cinemachineTargetYaw += _input.look.x * deltaTimeMultiplier;
        _cinemachineTargetPitch += _input.look.y * deltaTimeMultiplier;
    }

    // clamp our rotations so our values are limited 360 degrees
    _cinemachineTargetYaw = ClampAngle(_cinemachineTargetYaw, float.MinValue, float.MaxValue);
    _cinemachineTargetPitch = ClampAngle(_cinemachineTargetPitch, BottomClamp, TopClamp);

    // Cinemachine will follow this target
    CinemachineCameraTarget.transform.rotation = Quaternion.Euler(_cinemachineTargetPitch + CameraAngleOverride,
        _cinemachineTargetYaw, 0.0f);
}
```

Εικόνα 5.3.9: Camera Rotation μέθοδος.

```
2 references
private static float ClampAngle(float lfAngle, float lfMin, float lfMax)
{
    if (lfAngle < -360f) lfAngle += 360f;
    if (lfAngle > 360f) lfAngle -= 360f;
    return Mathf.Clamp(lfAngle, lfMin, lfMax);
}
```

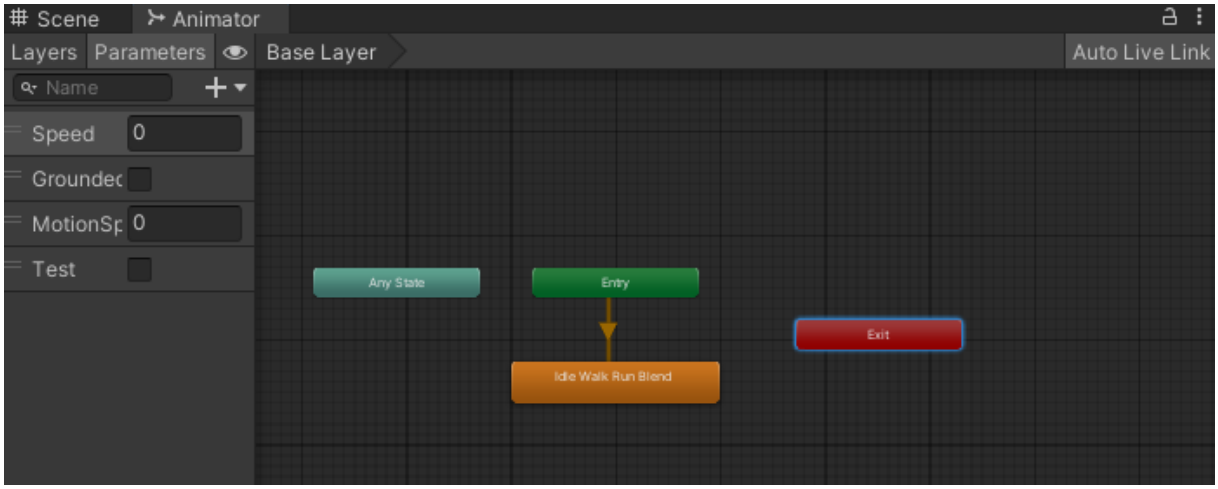
Εικόνα 5.3.10: ClampAngle.

Στην συνέχεια θα δούμε πως χρησιμοποιείται ο Animator για να animations.Στην εικόνα 5.3.11 θα δούμε ότι χρησιμοποιείται το avatar όπου δημιουργήθηκε στην αρχή του κεφαλαίου 5.2 και ο Controller για τα animations.



Εικόνα 5.3.11:Animator Component.

Τον Controller μπορούμε να τον δημιουργήσουμε από τα Assets και όταν τον ανοίξουμε μπορούμε να διαχειριστούμε το πότε θα παίζουν τα animations,ανάλογα με κάποια events,όπως είναι η αλλαγή μιας μεταβλητής σε μία τιμή.



Εικόνα 5.3.12: Animator Controller.

Στα αριστερά βλέπουμε τις μεταβλητές όπου ορίστηκαν και στον κώδικα, έτσι ώστε να μπορούν να προσπεραστούν. Με βάση τις τιμές αυτές το Blend Tree που βλέπουμε αλλάζει κατάσταση ώστε να αναπαραχθεί το κατάλληλο animation.

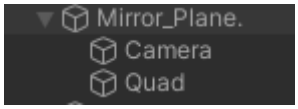
Τέλος, όταν ολοκληρώσαμε την μετακίνηση του χαρακτήρα διαπιστώσαμε ένα μεγάλο πρόβλημα, όπου ήταν σχετικό με τους Colliders.Οι Colliders είναι βασικό κομμάτι σε πολλά σημεία του παιχνιδιού. Το πρόβλημα που αντιμετωπίσαμε ήταν ότι ο χαρακτήρας μας περνούσε μέσα από αντικείμενα. Έπειτα καταλάβαμε ότι δεν μπορούσε να αλληλοεπιδράσει με αντικείμενα. Αυτό μας οδήγησε στην καλύτερη εκμάθηση των Colliders ,διότι όπως αναφέραμε σε προηγούμενο κεφάλαιο είναι πολύ σημαντικά.

Εικόνα 5.3.13:Παράδειγμα καρέκλας χωρίς Collider.

5.4 ΕΠΙΠΛΕΟΝ ΚΑΜΕΡΑ ΣΤΟ ΠΑΙΧΝΙΔΙ ΜΑΣ

Όπως αναφέραμε σε προηγούμενο κεφάλαιο μπορούμε να χρησιμοποιήσουμε επιπλέον κάμερα, ανάλογα με τις προτιμήσεις μας. Εμείς στο παιχνίδι μας χρησιμοποιήσαμε επιπλέον κάμερα σε έναν καθρέπτη. Ο καθρέπτης συνοδεύεται από την κάμερα, ένα gameobject τύπου Quad,ένα game empty object(γονέας) και τρία scripts.

Εικόνα 5.4.1: Καθρέπτης στο παιχνίδι.



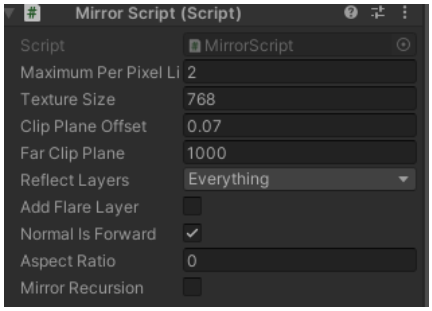
Εικόνα 5.4.2: Ιεραρχία καθρέπτη.

Εικόνα 5.4.3:Επιλεγμένος καθρέπτης στο Scene View.

Το empty game object(«Mirror_Plane.»), συνοδεύεται από το πρώτο script που περιέχει μεταβλητές για ρυθμίσεις της αντανάκλασης, όπου μπορούμε να ρυθμίσουμε μέσω του inspector panel του.

```
public class MirrorScript : MonoBehaviour
{
    [Tooltip("Maximum number of per pixel lights that will show in the mirrored image")]
    public int MaximumPerPixelLights = 2;

    [Tooltip("Texture size for the mirror, depending on how close the player can get to the mirror, this will need to be larger")]
    public int TextureSize = 768;

    [Tooltip("Subtracted from the near plane of the mirror")]
    public float ClipPlaneOffset = 0.07f;

    [Tooltip("Far clip plane for mirro camera")]
    public float FarClipPlane = 1000.0f;

    [Tooltip("What layers will be reflected?")]
    public LayerMask ReflectLayers = -1;

    [Tooltip("Add a flare layer to the reflection camera?")]
    public bool AddFlareLayer = false;

    [Tooltip("For quads, the normal points forward (true). For planes, the normal points up (false)")]
    public bool NormalIsForward = true;

    [Tooltip("Aspect ratio (width / height). Set to 0 to use default.")]
    public float AspectRatio = 0.0f;

    [Tooltip("Set to true if you have multiple mirrors facing each other to get an infinite effect, otherwise leave as false for a more realistic mirror effect.")]
    public bool MirrorRecursion;
}
```

Εικόνα 5.4.4: MirrorScript στο empty game object.



Εικόνα 5.4.5: Empty object inspector panel.

Το Quad στην ουσία είναι η επιφάνεια που «αντανακλά» το περιβάλλον, το οποίο Quad συνοδεύεται από ένα script.

```
public class MirrorReflectionScript : MonoBehaviour
{
    private MirrorCameraScript childScript;

    Unity Message | 0 references
    private void Start()
    {
        childScript = gameObject.transform.parent.gameObject.GetComponentInChildren<MirrorCameraScript>();

        if (childScript == null)
        {
            Debug.LogError("Child script (MirrorCameraScript) should be in sibling object");
        }
    }

    Unity Message | 0 references
    private void OnWillRenderObject()
    {
        childScript.RenderMirror();
    }
}
```

Εικόνα 5.4.6: MirrorReflectionScript στο Quad game object.

Το script αυτό ψάχνει ένα άλλο script σε κάποιο παιδί του γονέα του(στην ιεραρχία), εάν υπάρχει σαν component, το MirrorCameraScript.Έπειτα η μέθοδος OnWillRenderObject

καλείται για κάθε ορατή κάμερα στο object.Τέλος, αν το MirrorCameraScript βρεθεί, εκτελεί την βασική μέθοδό του,RenderMirror.

Στην συνέχεια έχουμε βάλει μια κάμερα, η οποία είναι ακριβώς πάνω στο Quad,όπου κοιτάει προς την ίδια κατεύθυνση, ώστε να αποδίδει ότι «βλέπει». Στην κάμερα αυτήν χρησιμοποιείται ένα script, όπου είναι ο πυρήνας του καθρέπτη. Το Script αυτό παίρνει τις ρυθμίσεις από το MirrorScript(το οποίο βρίσκεται στο empty game object) και αναπαριστά το camera view στο quad.Στην παρακάτω εικόνα θα δούμε την βασική μέθοδο RenderMirror,όπου καλείται από το MirrorReflectionScript του Quad.

```
internal void RenderMirror()
{
    Camera cameraLookingAtThisMirror;

    // bail if we don't have a camera or renderer
    if (renderingMirror || !enabled || (cameraLookingAtThisMirror = Camera.current) == null ||
        mirrorRenderer == null || mirrorMaterial == null || !mirrorRenderer.enabled)
    {
        return;
    }

    renderingMirror = true;

    int oldPixelLightCount = QualitySettings.pixelLightCount;
    if (QualitySettings.pixelLightCount != mirrorScript.MaximumPerPixelLights)
    {
        QualitySettings.pixelLightCount = mirrorScript.MaximumPerPixelLights;
    }

    try
    {
        UpdateCameraProperties(cameraLookingAtThisMirror, cameraObject);

        if (mirrorScript.MirrorRecursion)
        {
            mirrorMaterial.EnableKeyword("MIRROR_RECURSION");
            cameraObject.ResetWorldToCameraMatrix();
            cameraObject.ResetProjectionMatrix();
            cameraObject.projectionMatrix = cameraObject.projectionMatrix * Matrix4x4.Scale(new Vector3(-1, 1, 1));
            cameraObject.cullingMask = ~(1 << 4) & mirrorScript.ReflectLayers.value;
            GL.invertCulling = true;
            cameraObject.Render();
            GL.invertCulling = false;
        }
        else
        {
            mirrorMaterial.DisableKeyword("MIRROR_RECURSION");
            Vector3 pos = transform.position;
            Vector3 normal = (mirrorScript.NormalIsForward ? transform.forward : transform.up);
```

Εικόνα 5.4.7: RenderMirror μέθοδος μέρος 1.

```
        // Reflect camera around reflection plane
        float d = -Vector3.Dot(normal, pos) - mirrorScript.ClipPlaneOffset;
        Vector4 reflectionPlane = new Vector4(normal.x, normal.y, normal.z, d);
        CalculateReflectionMatrix(ref reflectionPlane);
        Vector3 oldpos = cameraObject.transform.position;
        float oldclip = cameraObject.farClipPlane;
        Vector3 newpos = reflectionMatrix.MultiplyPoint(oldpos);

        Matrix4x4 worldToCameraMatrix = cameraLookingAtThisMirror.worldToCameraMatrix;

        if (VRMode)
        {
            if(cameraLookingAtThisMirror.stereoActiveEye == Camera.MonoOrStereoscopicEye.Left)
            {
                worldToCameraMatrix[12] += 0.011f;
            }
            else if (cameraLookingAtThisMirror.stereoActiveEye == Camera.MonoOrStereoscopicEye.Right)
            {
                worldToCameraMatrix[12] -= 0.011f;
            }
        }

        worldToCameraMatrix *= reflectionMatrix;
        cameraObject.worldToCameraMatrix = worldToCameraMatrix;

        // Clip out background
        Vector4 clipPlane = CameraSpacePlane(ref worldToCameraMatrix, ref pos, ref normal, 1.0f);
        cameraObject.projectionMatrix = cameraLookingAtThisMirror.CalculateObliqueMatrix(clipPlane);
        GL.invertCulling = true;
        cameraObject.transform.position = newpos;
        cameraObject.farClipPlane = mirrorScript.FarClipPlane;
        cameraObject.cullingMask = ~(1 << 4) & mirrorScript.ReflectLayers.value;
        cameraObject.Render();
        cameraObject.transform.position = oldpos;
        cameraObject.farClipPlane = oldclip;
        GL.invertCulling = false;


    }
```

Εικόνα 5.4.8: RenderMirror μέθοδος μέρος 2.

5.5 ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΜΕ ΑΝΤΙΚΕΙΜΕΝΑ(INTERACTS)

Αφού μιλάμε για ένα παιχνίδι φυσικά θα υπάρχει η αλληλεπίδραση μεταξύ αντικειμένων. Σε αυτό το κεφάλαιο θα καταλάβουμε πώς λειτουργούν τα interacts,όπως θα δούμε ο χαρακτήρας μας μπορεί να πάρει αντικείμενα να ανοίξει πόρτες και όλα αυτά να συνεργαστούν μεταξύ τους μέσω του κώδικα.

Αν πρόκειται για αντικείμενο που μπορούμε να πάρουμε(πχ ένα κλειδί), θα χρειαστούμε μια βάση ενός κώδικα όπου ορίζει ένα αντικείμενο με συγκεκριμένες ιδιότητες(πχ Όνομα: Key,Icon: Εικόνα αντικειμένου), όπου χρησιμοποιείται δημιουργώντας ένα Item Asset,έτσι ώστε να το χρησιμοποιήσουμε σε άλλον κώδικα για τα interacts.Εάν πρόκειται για ένα interact με μία πόρτα απλά χρησιμοποιούμε τον κώδικα για τα interacts.

```
[CreateAssetMenu(fileName="New Item",menuName ="Inventory/Item")]
public class Item : ScriptableObject
{
    new public string name= "New Item";
    public Sprite icon = null;
    public bool isDefaultItem = false;
}
```

Εικόνα 5.5.1:Item Script.

Αρχικά για τα interacts χρησιμοποιούμε ένα empty game object στην σκηνή μας για να παίρνουμε το πλήκτρο του χρήστη μέσω ενός Asset που δημιουργούμε.

```
[CreateAssetMenu(fileName = "InteractionInputData", menuName = "InteractionSystem/InputData")]
public class InteractionInputData : ScriptableObject
{
    private bool m_interactedClicked;
    private bool m_interactedRelease;

    public bool InteractedClicked
    {
        get => m_interactedClicked;
        set => m_interactedClicked = value;
    }
    public bool InteractedRelease
    {
        get => m_interactedRelease;
        set => m_interactedRelease = value;
    }
    public void ResetInput()
    {
        m_interactedClicked = false;
        m_interactedRelease = false;
    }
}
```

Εικόνα 5.5.2:InteractionInputData Script.

```
public class InputHandler : MonoBehaviour
{
    #region Data
    [Header("InputData")]
    public InteractionInputData interactionInputData;
    #endregion
    // Start is called before the first frame update
    void Start()
    {
        interactionInputData.ResetInput();
    }

    // Update is called once per frame
    void Update()
    {
        GetInteractionInputData();
    }
    void GetInteractionInputData()
    {
        interactionInputData.InteractedClicked = Keyboard.current.eKey.wasPressedThisFrame;
        interactionInputData.InteractedRelease = Keyboard.current.eKey.wasReleasedThisFrame;
    }
}
```

Εικόνα 5.5.3:InputHandler Script.

Στις παρακάτω εικόνες θα δούμε ότι χρειαζόμαστε τρεις κώδικες για να ορίσουμε τα interactable αντικείμενα, τις ιδιότητες τους και έναν κώδικα που δημιουργεί ένα Asset όπου θα παίρνουμε πληροφορίες έτσι ώστε να ξεκινήσει η διαδικασία.

```
public interface IInteractable
{
    float HoldDuration { get; }

    bool HoldInteract { get; }
    bool MultipleUse { get; }
    bool IsInteractable { get; }

    string TooltipMessage { get; }

    string InteractionPromptMessage { get; }

    void OnInteract();
}
```

Εικόνα 5.5.4: IInteractable interface script.

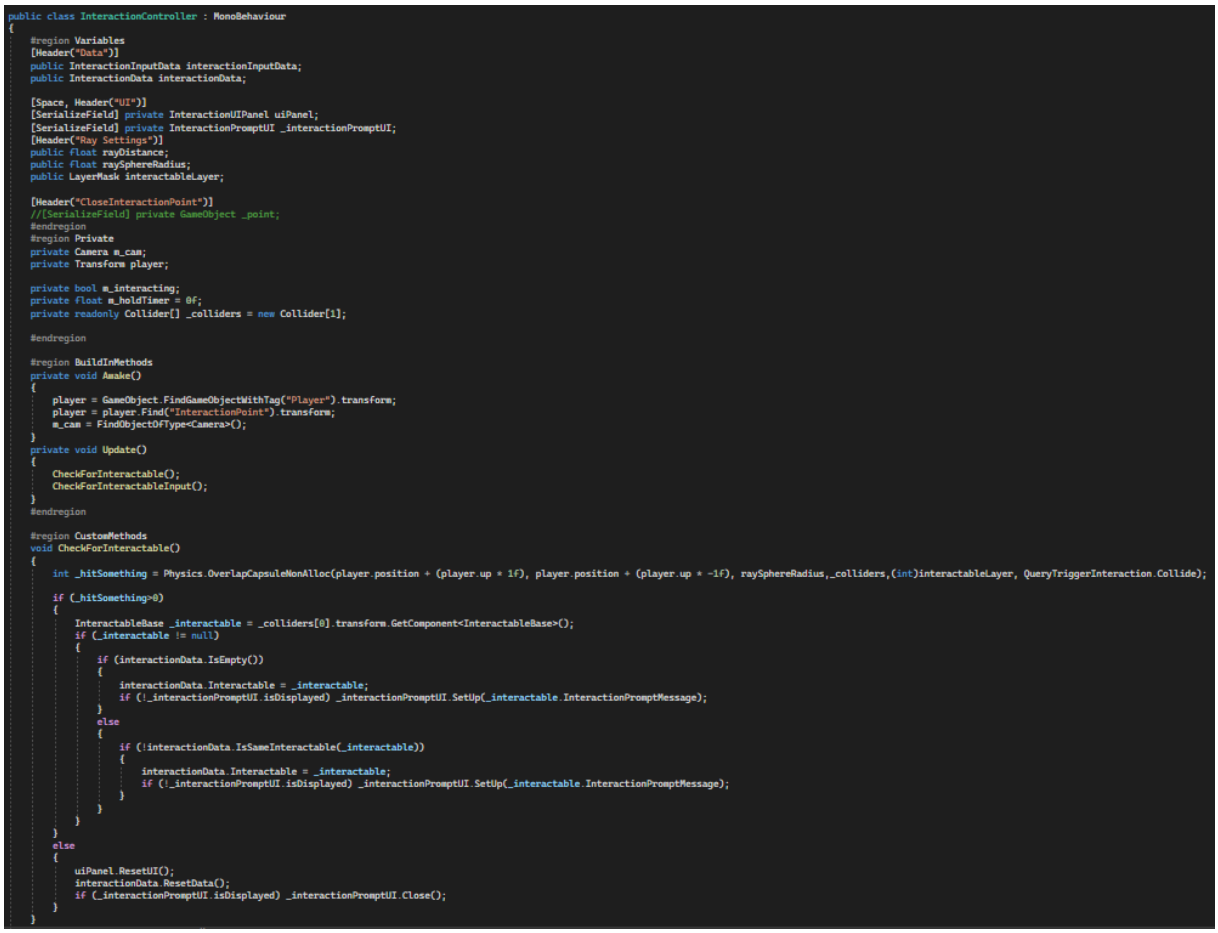
```
public class InteractableBase : MonoBehaviour, IInteractable
{
    #region Variables
    [Header("Interactable Settings")]
    public float holdDuration;
    [Space]
    public bool holdInteract;
    public bool multipleUse;
    public bool isInteractable;
    [SerializeField] private string tooltipMessage="Interact";
    [SerializeField] private string interactionPromptMessage = "Interact";
    #endregion

    #region Properties
    public float HoldDuration => holdDuration;

    public bool HoldInteract => holdInteract;

    public bool MultipleUse => multipleUse;

    public bool IsInteractable => isInteractable;

    public string TooltipMessage => tooltipMessage;

    public string InteractionPromptMessage => interactionPromptMessage;
    #endregion
    #region Methods
    public virtual void OnInteract()
    {
        Debug.Log("INTERACTED " + gameObject.name);
    }
    #endregion
}
```

Εικόνα 5.5.5: InteractableBase Script.

```
[CreateAssetMenu(fileName ="Interaction Data",menuName ="InteractionSystem/InteractionData")]
public class InteractionData : ScriptableObject
{
    private InteractableBase m_interactable;

    public InteractableBase Interactable
    {
        get => m_interactable;
        set => m_interactable = value;
    }

    public void Interact()
    {
        m_interactable.OnInteract();
        ResetData();
    }
    public bool IsSameInteractable(InteractableBase _newInteractable) => m_interactable == _newInteractable;
    public bool IsEmpty() => m_interactable == null;
    public void ResetData()=> m_interactable = null;
}
```

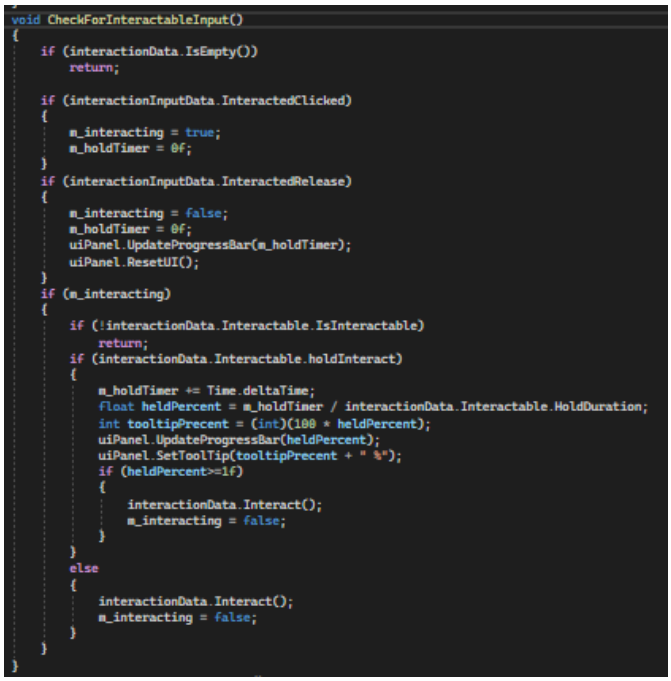
Εικόνα 5.5.6: InteractionData Script.

Τέλος, για να πραγματοποιηθεί το interaction θα χρειαστούμε έναν κώδικα όπου θα βρίσκει τα interactable αντικείμενα(χρειάζεται να δημιουργήσουμε ένα νέο Layer με όνομα «interactable») και θα συνδέει όλους τους παραπάνω κώδικες σε ένα interaction controller.Αυτός ο κώδικας βρίσκεται σαν component στον χαρακτήρα μας και έχουμε βάλει ένα game object σαν σημείο αλληλεπίδρασης. Στον κώδικα θα δούμε και μερικές μεταβλητές όπου χρησιμοποιούνται στα UI σε επόμενο κεφάλαιο.



```
public class InteractionController : MonoBehaviour
{
    #region Variables
    [Header("Data")]
    public InteractionInputData interactionInputData;
    public InteractionData interactionData;

    [Space, Header("UI")]
    [SerializeField] private InteractionUIPanel uiPanel;
    [SerializeField] private InteractionPromptUI _interactionPromptUI;
    [Header("Ray Settings")]
    public float rayDistance;
    public float raySphereRadius;
    public LayerMask interactableLayer;

    [Header("CloseInteractionPoint")]
    //[SerializeField] private GameObject _point;
    #endregion
    #region Private
    private Camera m_cam;
    private Transform player;

    private bool m_interacting;
    private float m_holdTimer = 0f;
    private readonly Collider[] _colliders = new Collider[1];

    #endregion

    #region BuildInMethods
    private void Awake()
    {
        player = GameObject.FindGameObjectWithTag("Player").transform;
        player = player.Find("InteractionPoint").transform;
        m_cam = FindObjectOfType<Camera>();
    }
    private void Update()
    {
        CheckForInteractable();
        CheckForInteractableInput();
    }
    #endregion

    #region CustomMethods
    void CheckForInteractable()
    {
        int _hitSomething = Physics.OverlapCapsuleNonAlloc(player.position + (player.up * 1f), player.position + (player.up * -1f), raySphereRadius,_colliders,(int)interactableLayer, QueryTriggerInteraction.Collide);

        if (_hitSomething>0)
        {
            InteractableBase _interactable = _colliders[0].transform.GetComponent<InteractableBase>();
            if (_interactable != null)
            {
                if (interactionData.IsEmpty())
                {
                    interactionData.Interactable = _interactable;
                    if (!_interactionPromptUI.isDisplayed) _interactionPromptUI.SetUp(_interactable.InteractionPromptMessage);
                }
                else
                {
                    if (!interactionData.IsSameInteractable(_interactable))
                    {
                        interactionData.Interactable = _interactable;
                        if (!_interactionPromptUI.isDisplayed) _interactionPromptUI.SetUp(_interactable.InteractionPromptMessage);
                    }
                }
            }
        }
        else
        {
            uiPanel.ResetUI();
            interactionData.ResetData();
            if (_interactionPromptUI.isDisplayed) _interactionPromptUI.Close();
        }
    }
```

Εικόνα 5.5.7:InteractionController Script μέρος 1(Μέθοδος CheckForInteractable).



```
void CheckForInteractableInput()
{
    if (interactionData.IsEmpty())
        return;

    if (interactionInputData.InteractedClicked)
    {
        m_interacting = true;
        m_holdTimer = 0f;
    }
    if (interactionInputData.InteractedRelease)
    {
        m_interacting = false;
        m_holdTimer = 0f;
        uiPanel.UpdateProgressBar(m_holdTimer);
        uiPanel.ResetUI();
    }
    if (m_interacting)
    {
        if (!interactionData.Interactable.IsInteractable)
            return;
        if (interactionData.Interactable.holdInteract)
        {
            m_holdTimer += Time.deltaTime;
            float heldPercent = m_holdTimer / interactionData.Interactable.HoldDuration;
            int tooltipPrecent = (int)(100 * heldPercent);
            uiPanel.UpdateProgressBar(heldPercent);
            uiPanel.SetToolTip(tooltipPrecent + " %");
            if (heldPercent>=1f)
            {
                interactionData.Interact();
                m_interacting = false;
            }
        }
        else
        {
            interactionData.Interact();
            m_interacting = false;
        }
    }
}
```

Εικόνα 5.5.8:InteractionController Script μέρος 2(Μέθοδος CheckForInteractableInput).

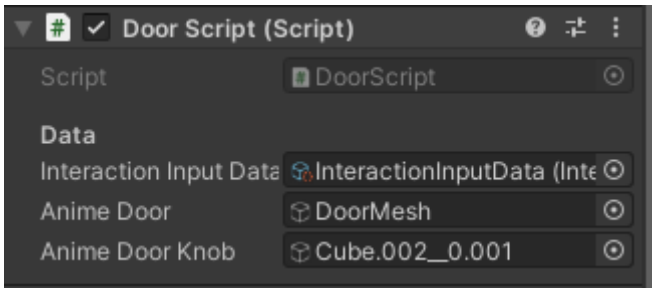
Σε αυτό το σημείο θα δούμε πως δουλεύει το interaction system.Θα δώσουμε δύο παραδείγματα, το άνοιγμα μιας πόρτας και να πάρει ένα αντικείμενο στο inventory(θα το δούμε παρακάτω). Πρέπει και τα δύο αντικείμενα πρέπει να έχουν το layer «Interactable».

Για την πόρτα χρησιμοποιήσαμε δύο animators,το InteractionInputData script και το InteractableBase script.Το animator component (DoorKnob) χρησιμοποιείται στο πόμολο της πόρτας και όταν τελειώνει το animation,με τη σειρά του, μπαίνει το δεύτερο animation component(DoorOPEN) για το animation της συνολικής πόρτας. Με σκοπό να διαχειριστούμε τα animation δημιουργήσαμε ένα script(DoorScript) που δημιουργεί καθυστέρηση ανάμεσα στα animations.Άρα όλη η συνεργασία ξεκινάει με τη σειρά ως εξής:

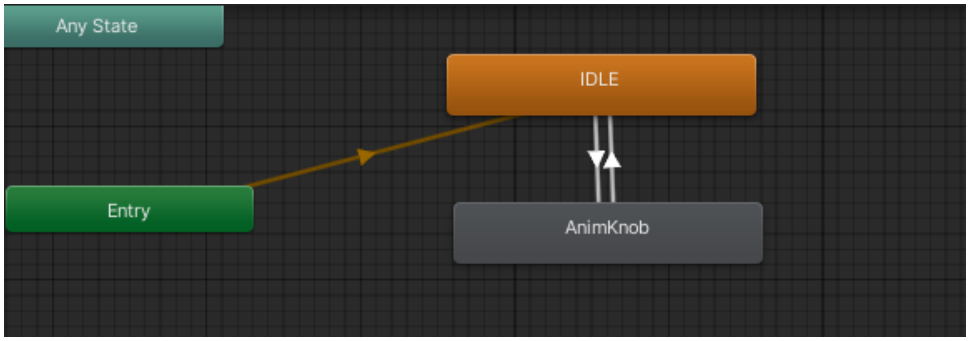
1. InteractionController script.
2. InteractableBase script.
3. InteractionInputData script.
4. DoorScript script.
5. Animator DoorKnob
6. Animator DoorOPEN

```
public class DoorScript : MonoBehaviour
{
    [Header("Data")]
    public InteractionInputData interactionInputData;
    public GameObject AnimeDoor;
    public GameObject AnimeDoorKnob;
    private bool isOpened = false;
    private AnimationClip[] clip;


    Unity Message | 0 references
    async void Update()
    {
        if (interactionInputData.InteractedClicked)
        {
            clip = AnimeDoorKnob.GetComponent<Animator>().runtimeAnimatorController.animationClips;
            if (!isOpened)
            {
                AnimeDoorKnob.GetComponent<Animator>().SetBool("DoorKnob", true);
                await Task.Delay(clip.Length * 1000);
                TriggerDoor(isOpened);
                AnimeDoorKnob.GetComponent<Animator>().SetBool("DoorKnob", false);
            }else
            {
                AnimeDoorKnob.GetComponent<Animator>().SetBool("DoorKnob", true);
                await Task.Delay(clip.Length * 1000);
                TriggerDoor(isOpened);
                AnimeDoorKnob.GetComponent<Animator>().SetBool("DoorKnob", false);
            }
        }
        else if(interactionInputData.InteractedRelease)
        {

            Debug.Log("E button released");
        }
    }
    2 references
    async void TriggerDoor(bool isOpen)
    {
        clip = AnimeDoorKnob.GetComponent<Animator>().runtimeAnimatorController.animationClips;
        if (isOpen)
        {
            AnimeDoor.GetComponent<Animator>().SetBool("OpenDoor", false);
            AnimeDoor.GetComponent<Animator>().SetBool("CloseDoor", true);
            await Task.Delay(clip.Length * 1000);
            isOpened = false;
        }
        else if (!isOpen)
        {
            AnimeDoor.GetComponent<Animator>().SetBool("OpenDoor", true);
            AnimeDoor.GetComponent<Animator>().SetBool("CloseDoor", false);
            await Task.Delay(clip.Length * 1000);
            isOpened = true;
        }
        AnimeDoor.GetComponent<Animator>().SetBool("OpenDoor", false);
        AnimeDoor.GetComponent<Animator>().SetBool("CloseDoor", false);
    }
}
```

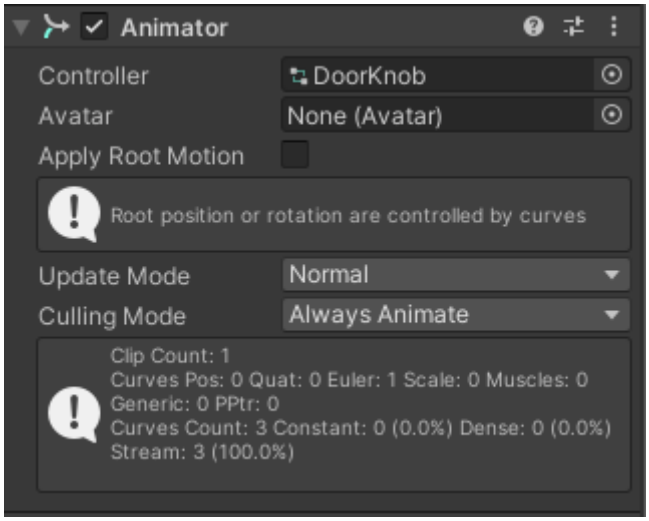
Εικόνα 5.5.9:DoorScript script.



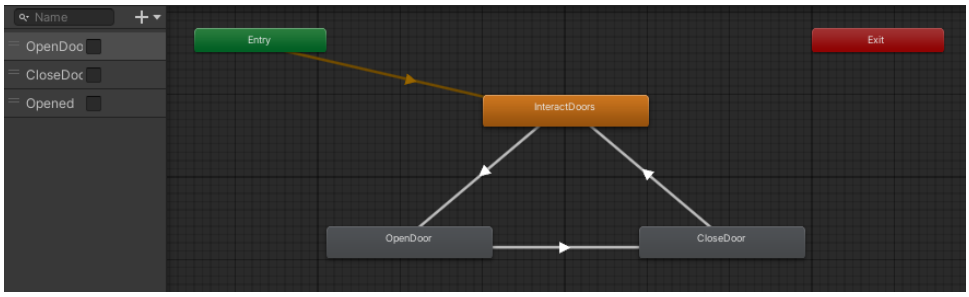
Εικόνα 5.5.10: DoorScript Component.



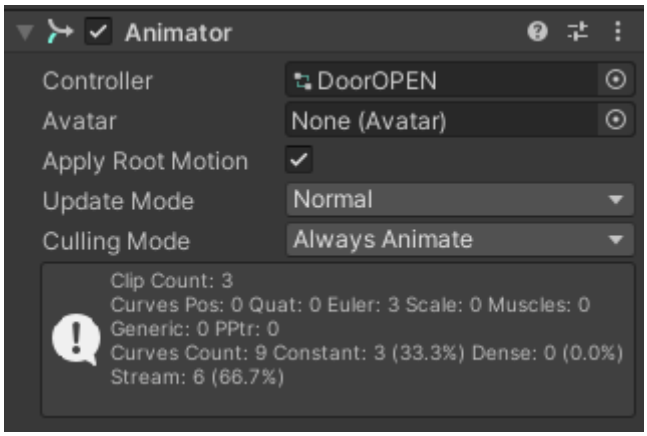
Εικόνα 5.5.11:Animator DoorKnob.



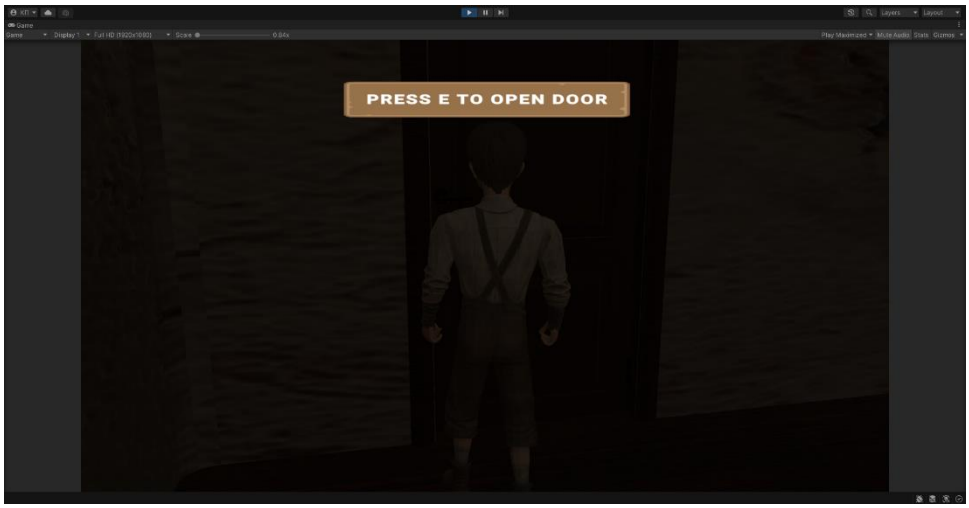
Εικόνα 5.5.12: Animator DoorKnob component.



Εικόνα 5.5.13: Animator DoorOPEN.



Εικόνα 5.5.14: Animator DoorOPEN component.



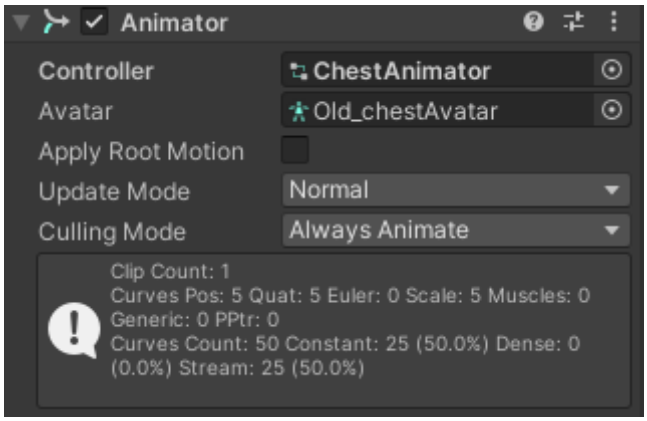
Εικόνα 5.5.15:Πλησιάζοντας την πόρτα.

Εικόνα 5.5.16:Αποτέλεσμα ,ανοιχτή πόρτα.

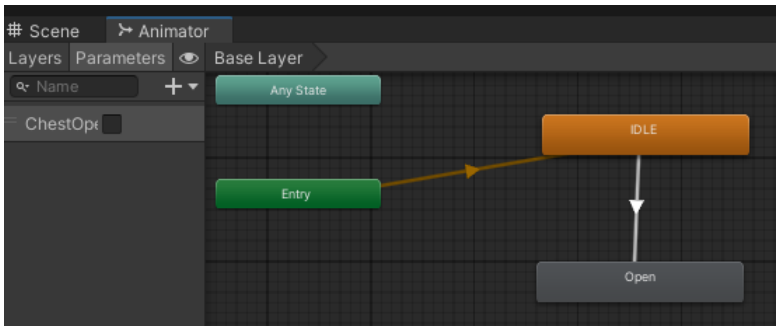
Για να πάρουμε ένα αντικείμενο στο inventory θα δώσουμε δύο παραδείγματα, άνοιγμα ενός σεντουκιού το οποίο περιέχει ένα κλειδί και μιας φωτογραφίας polaroid.Τα δύο αυτά αντικείμενα θα δείξουμε πως φαίνονται στο UI του inventory.

- **Σεντούκι**

Για το σεντούκι θα χρειαστούμε αρχικά να του βάλουμε το Layer Interactable καθώς έχουμε δει στο InteractionController ότι έτσι βρίσκει τα αντικείμενα. Το σεντούκι χρησιμοποιεί ένα Animator component για να κάνει το animation που ανοίγει. Συνοδεύεται από το script InteractableChest και χρησιμοποιεί αρχικά το InteractableBase script και ένα item asset.



Εικόνα 5.5.17:Animator component.



Εικόνα 5.5.18: Animator.

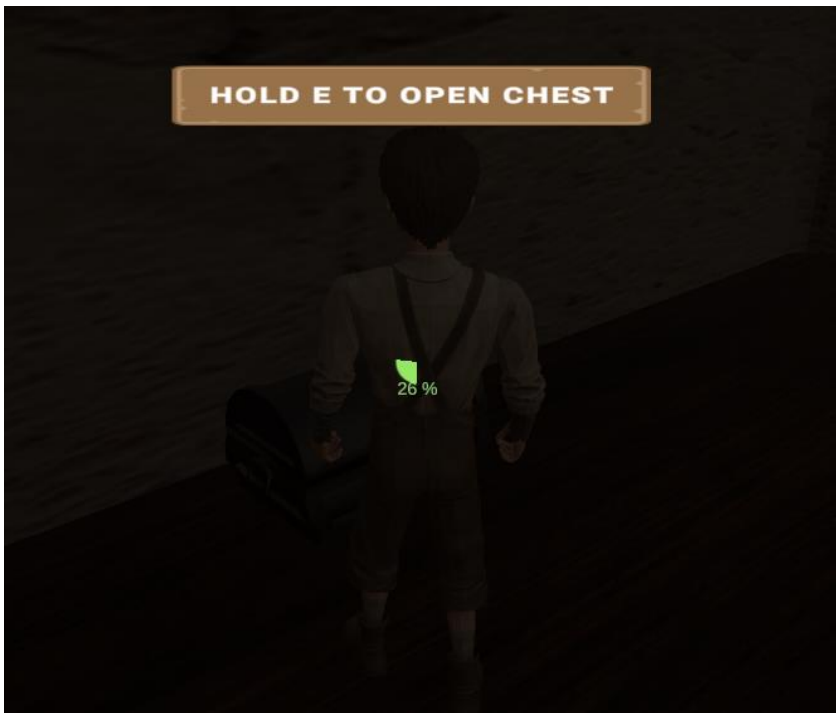
```
public class InteractableChest : InteractableBase
{
    private Animator chestAnimator;
    private AnimationClip[] clip;
    private bool chestOpened=false;
    public Item item;


    5 references
    public override void OnInteract()
    {
        base.OnInteract();
        OpenChest();
    }

    // Update is called once per frame
    1 reference
    async void OpenChest()
    {
        chestAnimator = this.GetComponent<Animator>();
        clip = chestAnimator.GetComponent<Animator>().runtimeAnimatorController.animationClips;
        if (!chestOpened)
        {
            chestAnimator.GetComponent<Animator>().SetBool("ChestOpened", true);
            await Task.Delay(clip.Length*1000);
            chestOpened = true;
            KeyObtained();
        }
        else
        {
            gameObject.layer = 0;
        }
    }
    1 reference
    void KeyObtained()
    {
        InventoryScript.instance.Add(item);
        Debug.Log("OBTAIN A KEY! PLEASE CHECK INVENTORY!");
        gameObject.layer = 0;
    }
}
```

Εικόνα 5.5.19:InteractableChest script.

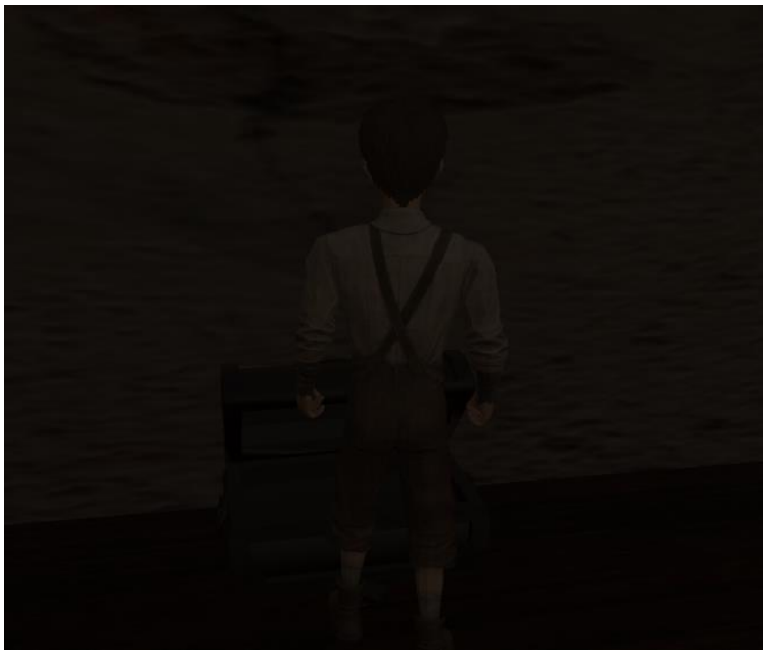
Ο κώδικας ενεργοποιείται για μόνο μια φορά αφού το σεντούκι μας θέλουμε μια φορά να το ανοίξουμε. Για να ανοίξουμε το σεντούκι πρέπει να κρατήσουμε πατημένο το πλήκτρο για 2 δευτερόλεπτα. Για να χρησιμοποιούμε την boolean για να ενεργοποιήσουμε το animation,μόλις

τελειώσει μεταφέρει το αντικείμενο στην λίστα του inventory.Τέλος ,μας εμφανίζει μήνυμα στο Console μόνο εάν είναι επιτυχείς το interact.

Εικόνα 5.5.20: Κοντά στο σεντούκι.

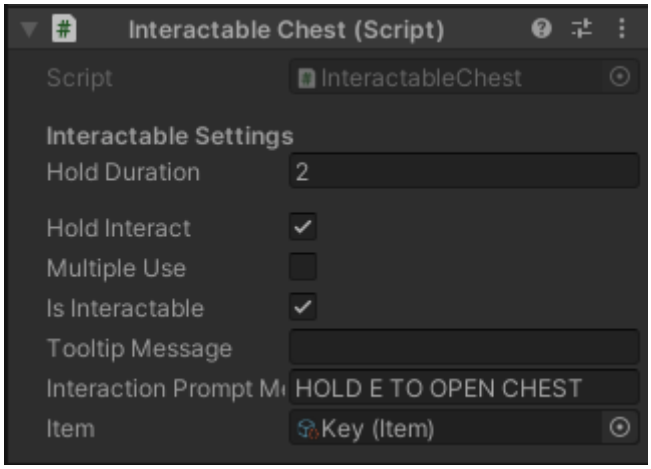


Εικόνα 5.5.21:Πατημένο κουμπί, αρχίζει και γεμίζει ο χρόνος.



Εικόνα 5.5.22 Ανοιχτό σεντούκι λείπει το UI.

Εικόνα 5.5.23:Μήνυμα στο Console.



Εικόνα 5.5.24:InteractableChest script,ρυθμίσεις για το σεντούκι.

- **Polaroid Photo pick up**

Στην ουσία για να βάλουμε το polaroid photo στο inventory απλά αποθηκεύουμε το item asset και μετά διαγράφουμε το game object.Φυσικά χρησιμοποιούμε και πάλι InteractableBase αφού είναι η βάση μας.

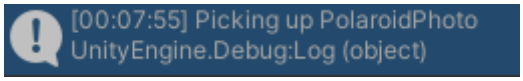
```
public class TestPickUp : InteractableBase
{
    public Item item;
    5 references
    public override void OnInteract()
    {
        base.OnInteract();
        PickUp();
    }
    1 reference
    void PickUp()
    {
            bool wasPickedUp = InventoryScript.instance.Add(item);
            Debug.Log("You picked up " + item.name);
            if (wasPickedUp) Destroy(gameObject);
    }
}
```

Εικόνα 5.5.25: TestPickUp script.

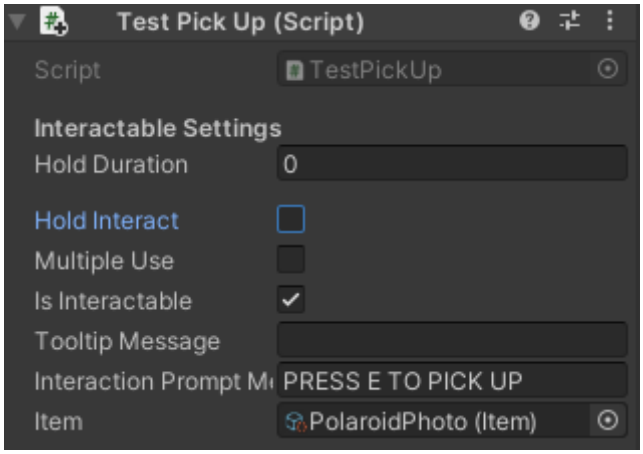
Εικόνα 5.5.26:Κοντά στο polaroid photo.

Όταν πατηθεί το πλήκτρο Ε παίρνει την polaroid photo,και εξαφανίζεται από την σκηνή μας και εμφανίζει μήνυμα στο Console.

Εικόνα 5.5.27:Polaroid photo Interacted.



Εικόνα 5.5.28:Μήνυμα στο console.



Εικόνα 5.5.29:TestPickUp script,ρυθμίσεις polaroid.

Εικόνα 5.5.30:Αντικείμενα στο Inventory.

5.6 USER INTERFACE(UI)

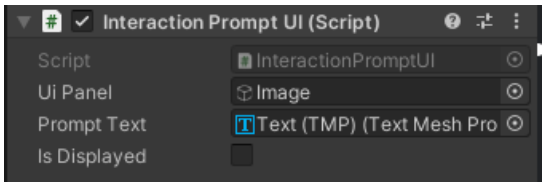
Σε αυτό το κεφάλαιο θα αναφέρουμε τα UI που χρησιμοποιήσαμε στο παιχνίδι μας, με ποιον τρόπο τα δημιουργήσαμε και τον τρόπο λειτουργίας τους. Θα δούμε αρχικά ένα UI (**InteractionPromptUI**) όπου δίνει πληροφορίες στον παίκτη σχετικά με το πλήκτρο που πρέπει να πατήσει σε ένα interactable αντικείμενο. Στην συνέχεια εάν αυτό το interactable αντικείμενο αλληλοεπιδράει με το holdInteract(πού είδαμε στο InteractionController) ενεργοποιείται το UI του(**InteractionUIPanel**). Έπειτα θα δούμε το UI(**InventoryUI**) όπου είναι υπεύθυνο για το inventory system.Επίσης θα δούμε το UI(**PauseMenu**) όπου πετυχαίνουμε ακινητοποίηση του παιχνιδιού. Τέλος θα δούμε το UI(**MainMenu**) που είναι υπεύθυνο για το εισαγωγικό μενού του παιχνιδιού μας.

- **InteractionPromptUI**

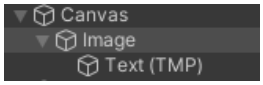
  Το UI αυτό αποτελείται από τρία game objects,το Canvas,το Image και το Text,όπου είναι διαθέσιμα από το Asset menu του Unity στην υποκατηγορία UI.Το Canvas(parent object) είναι αυτό που καθορίζει τον χώρο εμφάνισης του UI.Στο Canvas υπάρχει ένα script όπου ορίζει τις βασικές public μεθόδους έτσι ώστε να μπορούμε να τις χρησιμοποιήσουμε σε άλλα scripts και στον κώδικα βρίσκεται το rotate της κάμερας έτσι ώστε να φαίνεται ακόμα και αν γυρίσουμε την κάμερα όπως θέλουμε.

```
public class InteractionPromptUI : MonoBehaviour
{
    private Camera _mainCam;
    [SerializeField] private GameObject _uiPanel;
    [SerializeField] private TextMeshProUGUI _promptText;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        _mainCam = Camera.main;
        _uiPanel.SetActive(false);
    }

    // Update is called once per frame
    Unity Message | 0 references
    void LateUpdate()
    {
        var rotation = _mainCam.transform.rotation;
        transform.LookAt(transform.position + rotation * Vector3.forward, rotation * Vector3.up);
    }
    public bool isDisplayed=false;
    3 references
    public void SetUp(string promptText)
    {
        _promptText.text = promptText;
        _uiPanel.SetActive(true);
        isDisplayed = true;
    }
    2 references
    public void Close()
    {
        _uiPanel.SetActive(false);
        isDisplayed = false;
    }
}
```

Εικόνα 5.6.1:InteractionPromptUI script.



Εικόνα 5.6.2: InteractionPromptUI script component.



Εικόνα 5.6.3: InteractionPromptUI hierarchy.

Εικόνα 5.6.4: InteractionPromptUI.

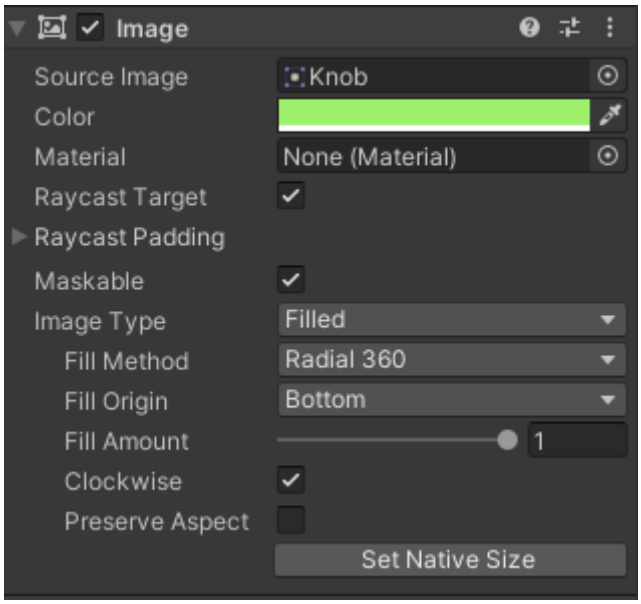
- **InteractionUIPanel**

Όπως είδαμε στο InteractionController script μέσα στην μέθοδο CheckForInteractableInput χρησιμοποιεί το holdInteract και καλεί δύο μεθόδους από το InteractionUIPanel για να γεμίσει την εικόνα του UI.Με τις κατάλληλες ρυθμίσεις στην εικόνα επιτυγχάνουμε το γέμισμα σε έναν κύκλο. Το InteractionUIPanel αποτελείται από το ProgressBar(Image),το TooltipText όπου δείχνει το ποσοστό επί τοις εκατό και το βασικό του script.

```
public class InteractionUIPanel : MonoBehaviour
{
    [SerializeField] private Image progressBar;
    [SerializeField] private TextMeshProUGUI tooltipText;

    public void SetToolTip(string tooltip)
    {
        tooltipText.SetText(tooltip);
    }
    public void UpdateProgressBar(float fillAmount)
    {
        progressBar.fillAmount = fillAmount;
    }
    public void ResetUI()
    {
        progressBar.fillAmount = 0f;
        tooltipText.SetText("");
    }
}
```

Εικόνα 5.6.5: InteractionUIPanel script.

Εικόνα 5.6.6: InteractionUI hierarchy.



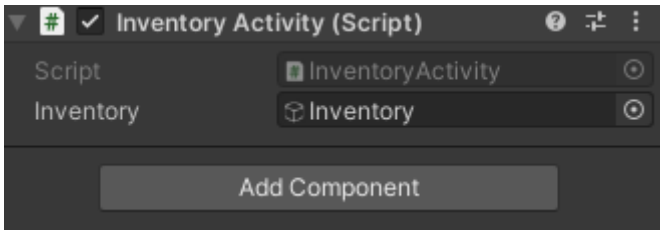
Εικόνα 5.6.7: Ρυθμίσεις ProgressBar image.

Εικόνα 5.6.8: InteractionUIPanel.

- **InventoryUI**
  Το UI αυτό μας απεικονίζει τα αντικείμενα που έχει στον «σάκο» του ο χαρακτήρας μας. Αυτό το UI συνοδεύεται από τέσσερα scripts.
  1. **InventoryActivity script.**
     Αυτό το script είναι υπεύθυνο για το πλήκτρο που ενεργοποιεί το UI και επίσης ακινητοποιεί το παιχνίδι μέχρι ξαναπατήσουμε το πλήκτρο και να επιστρέψουμε στο play mode.To script βρίσκεται σε ένα empty object στην σκηνή μας. Το πλήκτρο που χρησιμοποιείται είναι το I.

```
public class InventoryActivity : MonoBehaviour
{
    public GameObject inventory;
    GameObject cameraFollow;
    private bool keyPressed;
    private bool isActive;
    // Start is called before the first frame update
    void Start()
    {
        cameraFollow = GameObject.FindGameObjectWithTag("PlayerFollowCamera");
        inventory.SetActive(false);
        isActive=false;
    }

    // Update is called once per frame
    void Update()
    {
        if (Keyboard.current.iKey.wasPressedThisFrame)
        {
            if (!isActive)
            {
                Activate();
            }
            else
            {
                Deactivate();
            }
        }
    }
    void Activate()
    {
        Cursor.lockState = CursorLockMode.None;
        inventory.SetActive(true);
        cameraFollow.SetActive(false);
        isActive =true;
        Time.timeScale = 0f;
    }
    void Deactivate()
    {
        Cursor.lockState = CursorLockMode.Locked;
        inventory.SetActive(false);
        cameraFollow.SetActive(true);
        isActive = false;
        Time.timeScale = 1f;
    }
}
```

Εικόνα 5.6.9: InventoryActivity script.



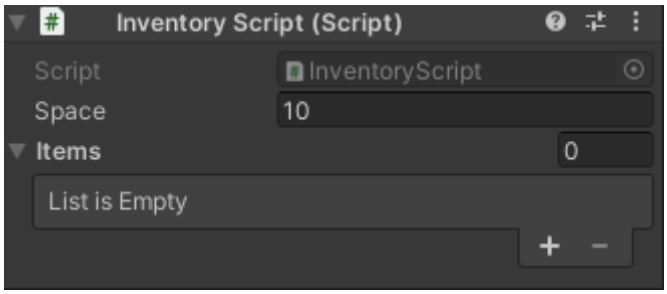
Εικόνα 5.6.10: InventoryActivity script component.

2. **InventoryScript**

Αυτό το script βρίσκεται σε empty object στην σκηνή μας και στην ουσία αυτό που κάνει είναι να ορίζει το συνολικό πλήθος αντικειμένων που μπορεί να χωρέσει, δημιουργεί μια λίστα από items,γεμίζει την λίστα με την μέθοδο Add,αν δεν υπάρχει χώρος σταματάει και εμφανίζει μήνυμα. Επίσης περιέχει την μέθοδο Remove η οποία αφαιρεί items από την λίστα.

```
public class InventoryScript : MonoBehaviour
{
    #region Singleton
    public static InventoryScript instance;

    Unity Message | 0 references
    private void Awake()
    {
        if (instance != null)
        {
            Debug.LogWarning("More than one instance of inventory found");
            return;
        }
        instance = this;
    }
    #endregion
    public delegate void OnItemChanged();
    public OnItemChanged onItemChangedCallback;
    public int space = 10;
    public List<Item> items = new List<Item>();
    1 reference
    public bool Add(Item item)
    {
        if (items.Count >= space)
        {
            Debug.Log("Not enough room");
            return false;
        }
        items.Add(item);
        if (onItemChangedCallback != null) onItemChangedCallback.Invoke();
        return true;
    }
    0 references
    public void Removed(Item item)
    {
        items.Remove(item);
        if (onItemChangedCallback != null) onItemChangedCallback.Invoke();
    }
}
```

Εικόνα 5.6.11:InventoryScript script.



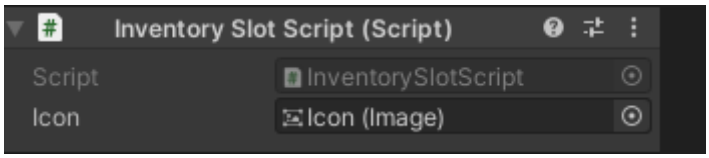
Εικόνα 5.6.12: InventoryScript script component.

3. **InventorySlotScript**

Αυτό το script περιέχει δύο μεθόδους να προσθέτει τα items στα slots(AddItem) ή να τα αφαιρεί(ClearSlot).Επίσης πρέπει να ορίσουμε το game object τύπου Image στο component έτσι ώστε να φαίνεται εκεί το εικονίδιο του αντικειμένου.

```
public class InventorySlotScript : MonoBehaviour
{
    public Image icon;
    Item item;

    1 reference
    public void AddItem(Item newItem)
    {
        item = newItem;
        icon.sprite = item.icon;
        icon.enabled = true;
    }
    1 reference
    public void ClearSlot()
    {
        item = null;
        icon.sprite = null;
        icon.enabled = false;
    }
}
```

Εικόνα 5.6.13:InventorySlotScript script.



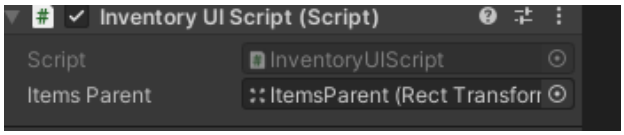
Εικόνα 5.6.14: InventorySlotScript script.

4. **InventoryUIScript**

Στην ουσία αυτό το script είναι υπεύθυνο για να ενημερώσει τα slots του InventoryUI με τα items. Η μόνη παράμετρος που παίρνει για να τρέξει είναι το parent game object των slots.

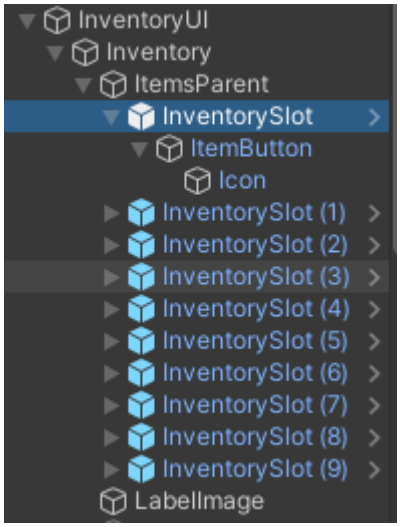
```
public class InventoryUIScript : MonoBehaviour
{
    public Transform itemsParent;
    InventoryScript inventory;
    InventorySlotScript[] slots;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        inventory = InventoryScript.instance;
        inventory.onItemChangedCallback += UpdateUI;
        slots = itemsParent.GetComponentsInChildren<InventorySlotScript>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
    }
    1 reference
    void UpdateUI()
    {
        for (int i = 0; i < slots.Length; i++)
        {
            if (i < inventory.items.Count)
            {
                slots[i].AddItem(inventory.items[i]);

            }
            else
            {
                slots[i].ClearSlot();
            }

        }
        //Debug.Log("UPDATING UI");
    }
}
```

Εικόνα 5.6.15 InventoryUIScript script.



Εικόνα 5.6.16: InventoryUIScript script component.

Έτσι, σε αυτό το σημείο μας μένει να δούμε την ιεραρχία του InventoryUI και το συνολικό αποτέλεσμα.



Εικόνα 5.6.17: InventoryUI hierarchy.

Εικόνα 5.6.18: InventoryUI.

5.7 MENUS

Αφού μιλήσαμε για τα UI,τώρα θα δούμε δύο πολύ βασικά menu σε ένα παιχνίδι, το Pause Menu και το Main Menu.

- **Pause Menu**

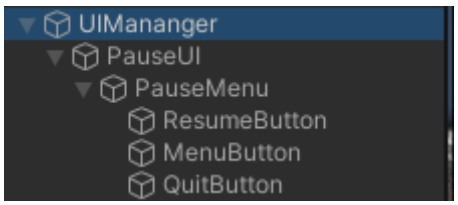
Το Pause menu αποτελείται από τρείς επιλογές(buttons), το resume, το menu και το quit game. Το έχουμε χωρίσει σε δυο κομμάτια κώδικα. Έναν βασικό(PauseMenu script) όπου μπορούμε να πατήσουμε το κουμπί Escape και να ενεργοποιηθεί το UI,υλοποιούνται οι βασικοί μέθοδοι σε αυτόν(Resume-Συνέχιση παιχνιδιού, Pause-πάγωμα παιχνιδιού)και ο δεύτερος κώδικάς χρησιμοποιείται για επιλογή με κλίκ στα buttons.

```
public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject playerToPause;
    public GameObject pauseMenuUI;
    public GameObject cameraFollow;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        playerToPause = GameObject.FindGameObjectWithTag("Player");
        cameraFollow = GameObject.FindGameObjectWithTag("PlayerFollowCamera");
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Keyboard.current.escapeKey.wasPressedThisFrame)
        {
            if (GameIsPaused)
            {
                Resume();
            } else {
                Pause();
            }
        }

    }
    2 references
    public void Resume()
    {
        Cursor.lockState = CursorLockMode.None;
        cameraFollow.SetActive(true);
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
        Cursor.lockState = CursorLockMode.Locked;
    }
    1 reference
    public void Pause()
    {
        Cursor.lockState = CursorLockMode.None;
        cameraFollow.SetActive(false);
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }
}
```

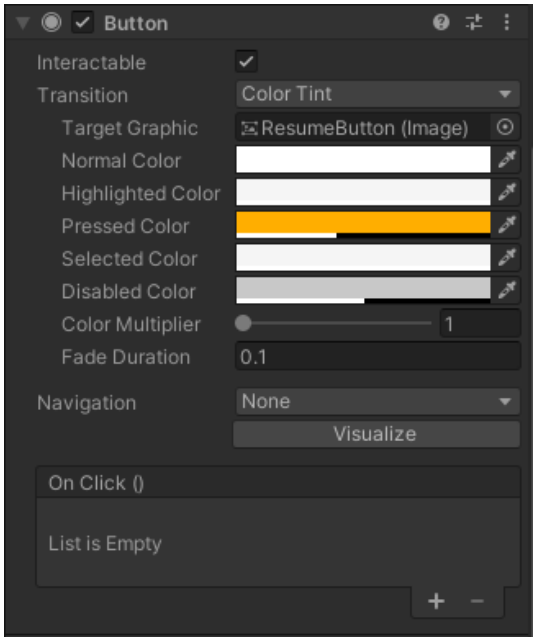
Εικόνα 5.7.1:PauseMenu script.

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

public class PauseButtons : MonoBehaviour
{
    private PauseMenu pauseMenu;

    private void Start()
    {
        pauseMenu = this.gameObject.GetComponent<PauseMenu>();
    }

    public void OnClickResume()
    {
        pauseMenu.Resume();
        Debug.Log("Resume");
    }

    public void OnClickMenu()
    {
        SceneManager.LoadScene("MainMenuScene");
        Debug.Log("Main Menu");
    }

    public void QuitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }
}
```

Εικόνα 5.7.2: PauseButton script.

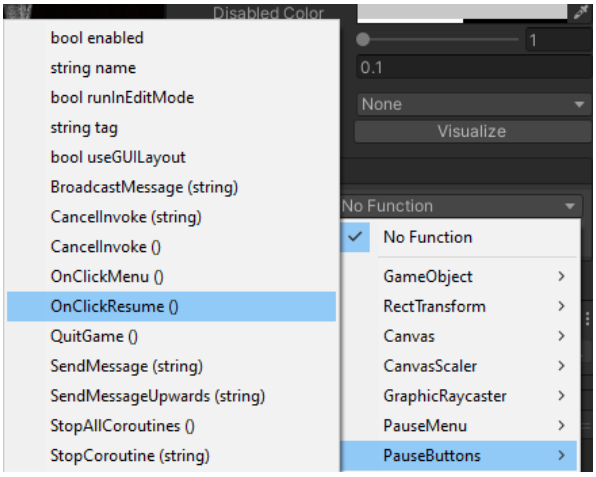


Εικόνα 5.7.3:PauseMenu hierarchy,

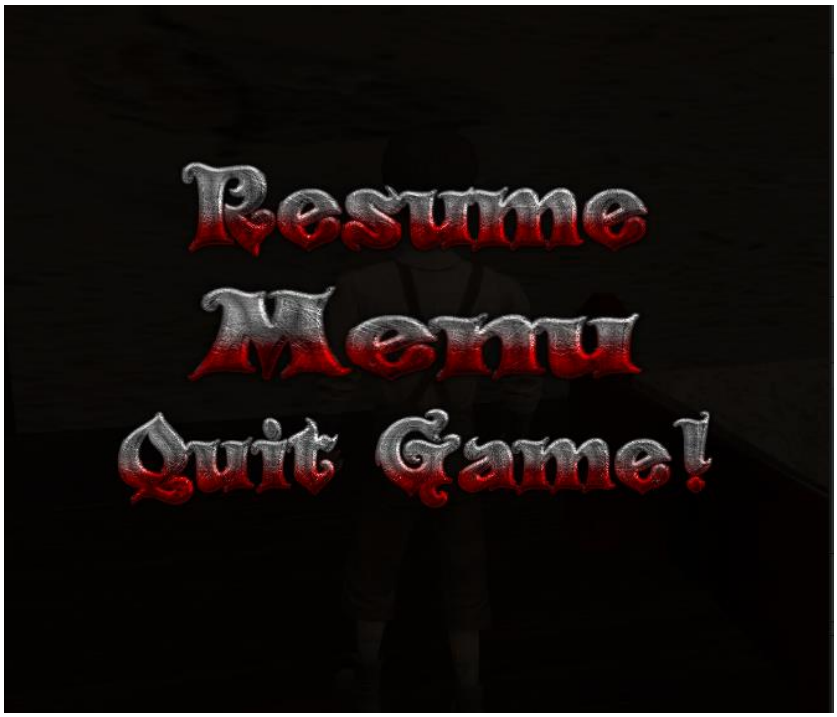


Εικόνα 5.7.4:Ιδιότητες κουμπιού ResumeButton, παρατηρήστε Pressed Color και On Click().

Στην παραπάνω εικόνα που βλέπουμε, θα πρέπει στο On Click() να μπει η παρακάτω επιλογή για να λειτουργήσει σωστά, βάση τον δικό μας κώδικα.



Εικόνα 5.7.5: Εισαγωγή μεθόδου σε button.



Εικόνα 5.7.6:Αποτέλεσμα Pause Menu.

Τέλος, στην εικόνα 5.7.4 παρατηρήσαμε ότι το Pressed Color έχει ένα χρώμα κίτρινο. Αυτό το κάναμε για να δημιουργήσουμε το παρακάτω εφέ, όταν για παράδειγμα επιλέγουμε το Quit Game με το ποντίκι μας να αλλάζει το shade του text μας και να παίρνει κίτρινο χρώμα.

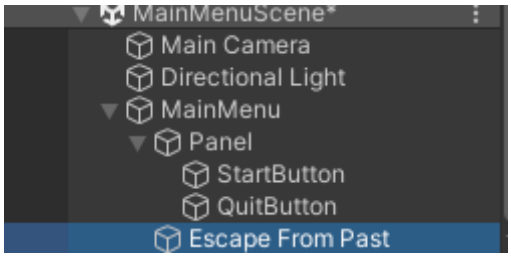
Εικόνα 5.7.7:Pause Menu πατημένο Quit Game!

- **Main Menu**

Για το Main Menu χρησιμοποιήσαμε ένα script μόνο για τα buttons.Υπάρχουν οι επιλογές Start Game και Quit Game!

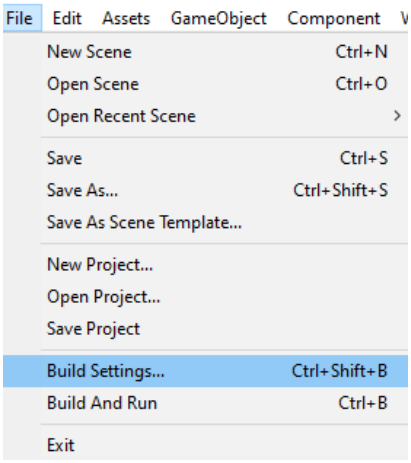
```
public class MainMenu : MonoBehaviour
{
    0 references
    public void StartGame()
    {
        SceneManager.LoadScene("GameScene");
    }
    0 references
    public void QuitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }
}
```

Εικόνα 5.7.8: Main Menu script.

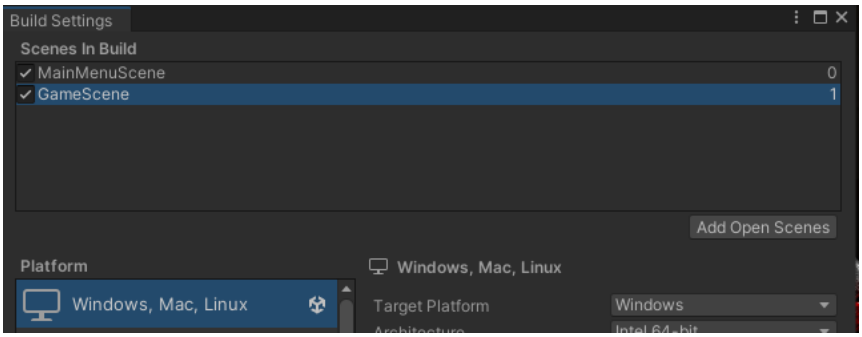


Εικόνα 5.7.9: Main Menu hierarchy.

Στην ουσία αυτό που χρειάζεται εδώ για να δουλέψει, είναι να αντιστοιχίσουμε τις μεθόδους στα δύο buttons που είδαμε πιο πάνω και να συμπεριλάβουμε τις δύο διαφορετικές σκηνές μας στο Build Settings του Unity.



Εικόνα 5.7.10:Επιλογή Build Settings.



Εικόνα 5.7.11: Build Settings Scenes.

Εικόνα 5.7.12: Αποτέλεσμα MainMenu.

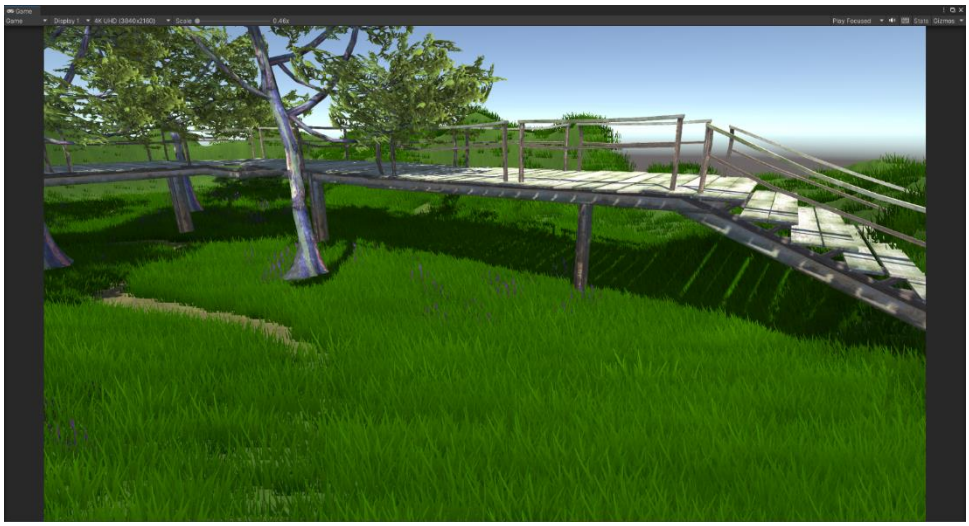
5.8 ΕΠΙΠΛΕΟΝ ΕΙΚΟΝΕΣ ΠΑΙΧΝΙΔΙΟΥ

Σε αυτό το σημείο καθώς φτάσαμε στο τέλος την εργασίας μας θα σας επιδείξουμε μερικές εικόνες ακόμα σχετικά με τον χαρακτήρα μας,Jake.Ο χαρακτήρας μας επισκέφθηκε και άλλες τοποθεσίες.

Εικόνα 5.8.1:Λειτουργία play mode 1080.

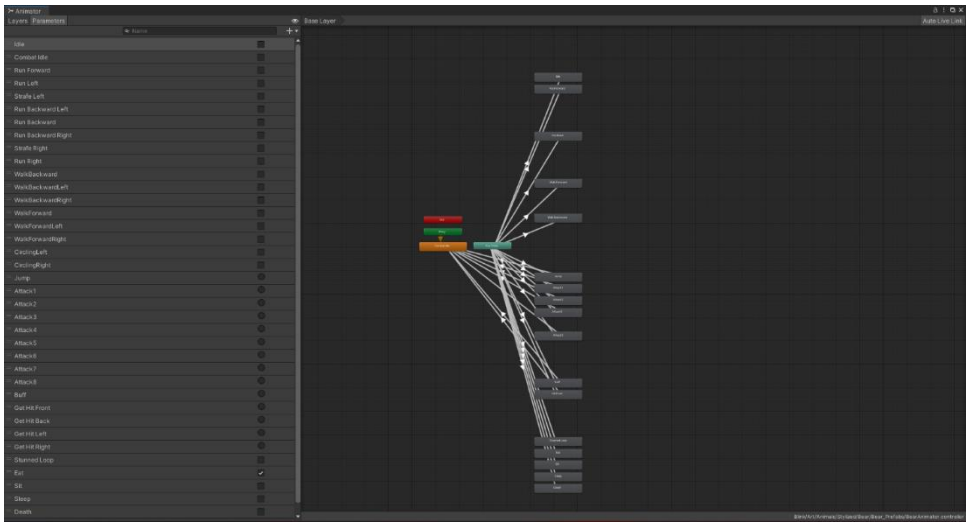
Εικόνες από το εξωτερικό περιβάλλον του παιχνιδιού.

Εικόνα 5.8.2:Γέφυρα.



Εικόνα 5.8.3: Περιβάλλον.

Εικόνα 5.8.4: Η αρκούδα του παιχνιδιού.



Εικόνα 5.8.5:Animator transition για την αρκούδα.

Στην εικόνα 5.8.5 βλέπουμε τα animations της αρκούδας. Μερικά από τα animations τα οποία μπορεί να κάνει είναι τα ακόλουθα: να τρέξει ευθεία, δεξιά, αριστερά, να πηδήξει, να κάνει επιθέσεις, να κοιμάται, να τρώει καθώς και κάποια άλλα.

6 ΣΥΜΠΕΡΑΣΜΑΤΑ

Μέσα από την ανάπτυξη και υλοποίηση του παιχνιδιού, ήμασταν σε θέση να πάρουμε αρκετές γνώσεις με την βοήθεια οδηγών από το YouTube,καθώς και από προσωπική εμπειρία όσο ασχολούμασταν με τις πλατφόρμες αυτές(Blender, Unity).Οι πλατφόρμες αυτές έχουν απεριόριστες δυνατότητες, αρκεί ο χρήστης που τις χρησιμοποιεί, να καταβάλλει αρκετή προσπάθεια και χρόνο, προκειμένου να εξοικειωθεί στη χρήση τους. Ένας νέος χρήστης, θα δυσκολευτεί αρκετά στην αρχή, καθώς θα διαπιστώσει ότι χρειάζεται να κατανοήσει πολλές άγνωστες ορολογίες (Rendering, Sculping,UHD unpacking,UV unwraping κ.α). Εν κατακλείδι αυτό που διαπιστώσαμε από την προσωπική μας εμπειρία, προκειμένου να είσαι σε θέση να κατανοήσεις, τις πλατφόρμες και τον τρόπο λειτουργίας τους χρειάζεται μόνο να καταβάλλεις αρκετή προσπάθεια, να έχεις όρεξη ώστε να ξοδέψεις αρκετό από τον προσωπικό σου χρόνο, προκειμένου να γίνεις ολοένα και καλύτερος και περισσότερο εξοικειωμένος, καθώς πρέπει να αντιληφθούμε ότι μια πλατφόρμα όπως η Unity, όπου έχει αναπτυχθεί και είχε πουληθεί με μεγάλη επιτυχία, ένα παιχνίδι σαν το Assasin’s Creed όπου ο καθένας που θα δοκιμάσει να παίξει αυτό το παιχνίδι, θα είναι σε θέση να διαπιστώσει τις απεριόριστες δυνατότητες που μπορεί να προσφέρει μια πλατφόρμα σαν αυτή. Θα βοηθήσει πολύ έναν νέο χρήστη αν μπει στο Asset Store,το οποίο είναι το επίσημο site που μας παρέχει η ίδια η εταιρία, από όπου και μπορούμε να κατεβάσουμε έτοιμα project άλλων χρηστών, προκειμένου να πάρει ιδέες, καθώς και να κατανοήσει την δομή στην ανάπτυξη ενός project. Από προσωπική μας εμπειρία η ασχολία με τέτοια project είναι αρκετά ενδιαφέρουσα καθώς είμαστε σε θέση να δημιουργήσουμε, να κάνουμε πράξη τις ιδέες μας και να δούμε τις προσπάθειες μας να λαμβάνουν μια τελική μορφή όταν ολοκληρώνεται ένα project. Τέλος, παρόλες τις δυσκολίες που αναφέραμε θα προσεγγίζαμε έναν νέο χρήστη στο αντικείμενο, διότι η εκμάθηση των πολλαπλών στοιχείων που παρέχουν οι πλατφόρμες, μπορεί στο μέλλον να τον βοηθήσει σε άπειρα projects, καθώς στην ανάπτυξη του παιχνιδιού περιλαμβάνονται στοιχεία από πολλούς εργασιακούς κλάδους(πχ μηχανικός λογισμικού,graphic designer, game developer).

7 ΠΗΓΕΣ

1. https://en.wikipedia.org/wiki/Minicomputer#/media/File:PDP-8.jpg
2. https://www.tomshardware.com/picturestory/508-mainframe-computer-history.html
3. https://dospace.org/blog/arcade-classic-computer-space/
4. https://en.wikipedia.org/wiki/Magnavox_Odyssey
5. https://en.wikipedia.org/wiki/Atari#/media/File:Atari_Official_2012_Logo_horizontal.svg
6. https://en.wikipedia.org/wiki/Activision#/media/File:Activision.svg
7. https://en.wikipedia.org/wiki/CD-ROM#/media/File:CD-ROM.png
8. https://el.wikipedia.org/wiki/%CE%94%CE%B9%CF%83%CE%BA%CE%AD%CF%84%CE%B1_%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%AE#/media/%CE%91%CF%81%CF%87%CE%B5%CE%AF%CE%BF:Floppy_disk_2009_G1.jpg
9. https://el.wikipedia.org/wiki/PlayStation_(%CE%BA%CE%BF%CE%BD%CF%83%CF%8C%CE%BB%CE%B1)#/media/%CE%91%CF%81%CF%87%CE%B5%CE%AF%CE%BF:PSX-Console-wController.jpg
10. https://en.wikipedia.org/wiki/History_of_video_games#/media/File:Entex_Baseball_3,_Model_6007,_Made_In_Taiwan,_Copyright_1980_(Electronic_Handheld_Game).jpg
11. https://en.wikipedia.org/wiki/Light-emitting_diode#/media/File:RBG-LED.jpg
12. https://en.wikipedia.org/wiki/Vacuum_fluorescent_display#/media/File:Vacuum_fluorescent_1.jpg
13. https://en.wikipedia.org/wiki/Intellivision#/media/File:Intellivision_logo.gif
14. https://en.wikipedia.org/wiki/Bandai#/media/File:Logo_Bandai.svg
15. https://en.wikipedia.org/wiki/3D_computer_graphics#/media/File:Dunkerque_3d.jpeg
16. https://en.wikipedia.org/wiki/Game_engine#/media/File:Quake_-_family_tree_2_Simplified.svg
17. https://www.esoui.com/downloads/info422-MuteNPCDialogue.html
18. https://en.wikipedia.org/wiki/Video_game_development
19. https://en.m.wikipedia.org/wiki/Unity_(game_engine)