Ελληνικό Μεσογειακό Πανεπιστήμιο
Σχολή Μηχανικών

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πρόγραμμα σπουδών - Μηχανικών Πληροφορικής Τ.Ε.

Πτυχιακή Εργασία

Σχεδιασμός και υλοποίηση διαδραστικού μουσικού ακούσματος σε VR, με την Unity3D

Χαμηλός Αντώνιος – Α.Μ. 4224

Επιβλέπων Καθηγητής : Παχουλάκης Ιωάννης

ΗΡΑΚΛΕΙΟ 2020

Περίληψη

Το αντικείμενο της πτυχιακής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός παιχνιδιού όπου ο παίκτης πρέπει να αλληλεπιδράσει με ένα stream αντικειμένων που έρχονται καταπάνω του, στον ρυθμό ενός επιλεγμένου μουσικού κομματιού, με τη βοήθεια κατάλληλων χειριστηρίων (VR motion controllers). Τα διαθέσιμα "όπλα" μπορεί να διαφέρουν ανάλογα το mode του παιχνιδιού που έχει επιλεχτεί (lightsabers, rifles, ασπίδες).
Κάθε αντικείμενο θα είναι είτε ένας κύβος, με τον οποίο πρέπει να υπάρξει αλληλεπίδραση, είτε άλλα αντικείμενα όπως βόμβες - σφαιρίδια τα οποία ο παίκτης θα πρέπει να αποφύγει.
Η σειριακή δημιουργία των αντικειμένων, θα δημιουργείται από τους ίδιους τους παίκτες, με τη βοήθεια ενός ενσωματωμένου Game Editor, όπου θα διευθετήσουν το συγχρονισμό των αντικειμένων με το μουσικό κομμάτι.
Για την αποθήκευση και την φόρτωση του Level, θα χρησιμοποιηθούν JSON αρχεία, τα οποία θα χρησιμοποιεί η Unity για την σειριακή δημιουργία των
αντικειμένων.
Απαραίτητο για την δημιουργία των αντικειμένων είναι το BPM (aka Beats Per Minute) του επιλεγμένου μουσικού κομματιού, καθώς και η διάρκεια του.
Το παιχνίδι θα αναπτυχθεί στο Unity Game Engine και για την υποστήριξη εικονικής πραγματικότητας (VR) θα χρησιμοποιηθεί το εργαλείο Steam VR Plugin από το Asset Store της Unity.
Απώτερος σκοπός είναι να προσφέρεται "Immersion" στον παίκτη, μέσω της χρήσης του VR Headset και των VR controllers.

Abstract

The subject of the thesis is the design and implementation of a game, so that the player interacts with a stream of objects that come towards him, to the rhythm of a selected music track, using appropriate controls (VR motion controllers).
The available "weapons" may differ depending on the selected game mode. (lightsabers, rifles, shields). Each item will be either a cube with which you interact, or other items such as bombs - spheres that the player must avoid.
The stream of objects will be created by the players themselves, using the built-in Game Editor, where they will arrange the synchronization of the objects with the music track.
To save and load the Level, JSON files will be used, which will be imported in Unity to create the stream of objects.
Necessary to create the stream of objects, is the BPM (aka Beats Per Minute) of the selected music track, as well as its duration.
The game will be developed using Unity Game Engine. For Virtual Reality support,
the Steam VR Plugin tool from Unity's Asset Store will be used.
The ultimate goal is to offer "Immersion" to the player, through the use of VR Headset and VR controllers.

Περιεχόμενα

Κατάλογος Εικόνων

Κεφάλαιο 1 : Εισαγωγή

Την τελευταία δεκαετία, η τεχνολογία της εικονικής πραγματικότητας αναπτύσσεται με ραγδαίους ρυθμούς. Η τεχνολογία αυτή χρησιμοποιεί ηλεκτρονικά μέσα , ώστε να γίνει προσομοίωση εικονικών περιβαλλόντων. Ο σκοπός ειναι να δημιουργείται ψευδαίσθηση στον χρήστη, ότι αποτελεί μέρος αυτού του εικονικού περιβάλλοντος, ενώ παράλληλα μπορεί να περιπλανηθεί και να αλληλεπιδράσει σε αυτό. Κατι που θα έκανε σε εάν μη εικονικό περιβάλλον.

Πολυ σημαντικό, για την κατάλληλη εμβύθιση (Immersion) του χρήστη στον κόσμο της εικονικής πραγματικότητας, ειναι η απομόνωση των αισθήσεων του, απο τον πραγματικό κόσμο, και η αντικατάσταση αυτών με αντίστοιχες εικονικές – ψευδείς αισθήσεις. Αυτές οι ψευδαισθήσεις, δημιουργούνται μέσω των ηλεκτρονικών συστημάτων, που προσομοιάζουν μια εικονική πραγματικότητα.

Οι πιο σημαντικές αισθήσεις του ανθρώπου ειναι η όραση και η ακοή. Μετέπειτα ειναι και η αφή. Αυτές ειναι οι βασικές αισθήσεις που χρησιμοποιεί ένα σύστημα εικονικής πραγματικότητας, προκειμένου να δημιουργήσει την καλύτερη δυνατή εμπειρία στον χρήστη. Έτσι ένα τέτοιο σύστημα, πρεπει να χρησιμοποιεί στερεοσκοπική όραση, το οποιο επιτυγχάνεται με την χρήση δυο οθονών (μια για κάθε μάτι), που η κάθε μια εχει διαφορετική γωνία θέασης. Με αυτή την τεχνική δημιουργείται η ψευδαίσθηση του βάθους στον χρήστη. Επίσης σημαντική ειναι η χρήση στερεοσκοπικού τρισδιάστατου ήχου, καθώς ειναι απαραίτητο για την εμβύθιση στον εικονικό κόσμο. Μέσω του στερεοσκοπικού ήχου ο χρήστης μπορεί να αντιληφθεί την ύπαρξη κάποιας ηχητικής πηγής στον εικονικό κόσμο, καθώς και την τοποθεσία της στο περιβάλλον αυτό.

Έτσι αν γίνει ο συνδυασμός όλων των παραπάνω, μαζι με την σωστή ανίχνευση του VR Headset και των VR χειριστηρίων, θα δημιουργείται στον χρήστη, μια ρεαλιστική προσομοίωση ενός εικονικού κόσμου. (Wikipedia, 2020)

Για την ανάπτυξη εικονικών περιβαλλόντων, προσομοιώσεων και παιχνιδιών για εκπαιδευτικούς σκοπούς (Serious Games), γίνεται χρήση σύγχρονων μηχανών ανάπτυξης βιντεοπαιχνιδιών (Game Engines), που αποτελούν την κατάλληλη επιλογή για αυτές τις δημιουργίες, καθώς τηρούν όλες τις τεχνικές προϋποθέσεις. (Μουστάκας Κ., 2015)

Η χρήση της εικονικής πραγματικότητας, εκτός απο τον εκπαιδευτικό τομέα, εφαρμόζεται και σε άλλους τομείς. Μερικοί απο αυτούς ειναι η εικονική προσομοίωση διάφορων μοντέλων, όπως εικονικά σπίτια ή ακόμα και διάφορες εμπειρίες εμβύθισης (Immersion), όπως η εικονική επίσκεψη τουριστικών μνημείων ή ακόμα και μουσείων.

Επιπλέον ένας ακόμη τομέας, στον οποιο θα εμβαθύνουμε ειναι η ψυχαγωγία μέσω της εικονικής πραγματικότητας. Και πιο συγκεκριμένα μέσω των εμπειριών των εικονικών βιντεοπαιχνιδιών. Χρησιμοποιώντας αυτή την τεχνολογία θα αναπτυχθεί και θα υλοποιηθεί ένα ρυθμικό μουσικό παιχνίδι εικονικής πραγματικότητας, με σκοπό την εμβύθιση και ψυχαγωγία του παίκτη.

1.1 Υλικό

Υπάρχουν διάφορες συσκευές στην χρήση της εικονικής πραγματικότητας. Οι βασικές συσκευές είναι το VR Headset (γυαλιά ε.π.) και τα VR controllers (χειριστήρια ε.π.). Στην συγκεκριμένη υλοποίηση, χρησιμοποιήθηκε το kit της Lenovo, με την ονομασία Explorer. Αυτό το kit χρησιμοποιεί τεχνολογικά Standard της Microsoft, της σειράς εικονικών συσκευών, με την ονομασία Windows Mixed Reality.

Εικόνα 1 - Lenovo Explorer VR Headset

Το συγκεκριμένο VR Headset είναι εξοπλισμένο με μια οθόνη για κάθε μάτι, με συνολική ανάλυση 2880 x 1440 pixel, FOV (Περιφερειακή όραση) στις 110 μοίρες και ρυθμό ανανέωσης καρέ, στα 90hz. Γενικά, προτείνεται να έχουν μεγάλη ανάλυση και μεγάλο ρυθμό ανανέωσης καρέ, έτσι ώστε μην δημιουργεί στον χρήστη δυσφορία (ζαλάδα – ναυτία), και να του προσφέρει εμβύθιση, στον κόσμο της εικονικής πραγματικότητας.

Επίσης, εχει αισθητήρα εγγύτητας (απόστασης) που χρησιμεύει στην ανίχνευση των χειριστηρίων μέσω των δυο καμερών μπροστά από το Headset. Γυροσκόπιο, που βοηθάει στην ανίχνευση περιστροφής του VR headset, σε τρισδιάστατο άξονα (x, y, z). Επιταχυνσιόμετρο, το οποιο εντοπίζει την επιτάχυνση και την κίνηση του VR Headset στον χώρο. Καθώς και μαγνητόμετρο, όπου μπορεί να μετρήσει το μαγνητικό πεδίο, λειτουργώντας ως πυξίδα, για να ελέγχει προς ποια κατεύθυνση «κοιτάει» (είναι σε ευθεία).

Για την μεταφορά της εικόνας από τον υπολογιστή, χρησιμοποιείται ένα HDMI καλώδιο που συνδέεται πάνω στην κάρτα γραφικών. Για την μεταφορά πληροφοριών προς τον υπολογιστή από τους αισθητήρες και γενικά τροφοδοσίας χρησιμοποιείται ένα USB 3.0 καλώδιο.

Για τον χειρισμό ενός VR παιχνιδιού η μιας VR διεπαφής, απαραίτητα είναι τα VR Controllers με τα οποία ο χρήστης εκτελεί οποιαδήποτε άμεση αλληλεπίδραση με την εφαρμογή. Αυτά, επικοινωνούν με τον υπολογιστή μέσω Bluetooth 4.0 και έχουν σχεδόν όλους τους αισθητήρες που εχει και το Headset. Έχουν, μαγνητόμετρο, επιταχυνσιόμετρο και γυροσκόπιο, οι οποίοι λειτουργούν ακριβώς όπως αυτοί του Headset. Επίσης, για μια πιο εύκολη αλληλεπίδραση, τα χειριστήρια έχουν διεπαφές εισόδου. Μερικές από αυτές είναι κουμπιά, αφή, σκανδάλες, ή ακόμα και μοχλούς.

Έτσι οι πληροφορίες που στέλνονται προς τον υπολογιστή (και κατά συνέπεια στην εφαρμογή), από όλους τους αισθητήρες, χρησιμοποιούνται για να μεταφράσουν σε τρισδιάστατο χώρο, την τοποθεσία αυτών των συσκευών, μέσα στο εικονικό περιβάλλον (εφαρμογή - παιχνίδι), σε πραγματικό χρόνο.

Για καλύτερη εμπειρία, ιδανικό θα ηταν ο υπολογιστής που χρησιμοποιείται να είναι VR Ready. Απαιτείται να είναι ένα σχετικά ισχυρό επεξεργαστικά σύστημα, καθώς πρεπει να ανανεώνονται τα καρέ σε συχνότητα πάνω απο 90hz, όπου το κάθε καρέ είναι ανάλυσης 1440p.

1.2 Λογισμικό

Για την ανάπτυξη μιας VR εφαρμογής μπορούν να χρησιμοποιηθούν διάφορα λογισμικά, αναλόγως την υλοποίηση που απαιτείται. Στην συγκεκριμένη υλοποίηση στόχος είναι η ανάπτυξη ενός βιντεοπαιχνιδιού, οποτε μεταξύ διάφορων μηχανών ανάπτυξης παιχνιδιών, η Unity μεταξύ άλλων, θεωρείται άξια μηχανή με τα καταλληλά εργαλεία για την ανάπτυξη τέτοιων εικονικής πραγματικότητας εφαρμογών. Οι εφαρμογές τις Unity μπορούν να «τρέξουν» σχεδόν σε όλα τα δημοφιλή λειτουργικά συστήματα (καθώς και σε κονσόλες). Αυτό κανει την ανάπτυξη των εφαρμογών πολύ πιο γρήγορη. Βέβαια η κάθε εφαρμογή χρειάζεται την αντίστοιχη προσαρμογή για να μην υπαρχουν προβλήματα στο κάθε λειτουργικό σύστημα. Επιπλέον, η Unity χρησιμοποιεί μια γλώσσα υψηλού επιπέδου την C# η οποία εχει όλες τις απαραίτητες βιβλιοθήκες, ενώ παράλληλα παρέχει και ασφάλεια στην σωστή λειτουργία της εφαρμογής.

Η Unity από μόνη της δεν παρέχει εργαλεία για την ανάπτυξη εφαρμογών εικονικής πραγματικότητας. Οπότε αυτό το πετυχαίνει μέσω επεκτάσεων (third-party plugins) που υποστηρίζει. Ακόμα και οι επεκτάσεις για την υποστήριξη VR είναι αρκετές. Για μια καλύτερη υποστήριξη έγινε η χρήση της επέκτασης SteamVR. Μιας επέκτασης που χρησιμοποιείται ευρέως για τις περισσότερες VR εφαρμογές. Το SteamVR υποστηρίζει όλες τις VR συσκευές, που συνδέονται με τον υπολογιστή, όπως συσκευές της Valve (Valve Index VR), της Oculus (Oculus Rift), και όλες τις συσκευές που έχουν πρότυπο της Microsoft (Windows Mixed Reality).

Επίσης, στην ανάπτυξη του παιχνιδιού χρησιμοποιήθηκαν και εξωτερικά προγράμματα που ενσωματωθήκαν σε αυτό, καθώς ηταν απαραίτητα για κάποιες συγκεκριμένες λειτουργίες. Έτσι εγινε χρήση, ενός πληκτρολογίου εικονικής πραγματικότητας, ενός προγράμματος περιήγησης αρχείων, λογισμικό οπτικού ελέγχου (VR Eye Raycaster), λογισμικό για τις εικονικές υποδείξεις (VRTK Tooltip), μιας βιβλιοθήκης που βοηθαει στον διαχωρισμό αντικειμένων (Mesh Split), και ενός εργαλείου για την καλύτερη προσαρμογή τρισδιάστατου κειμένου μέσα στο παιχνίδι (Text Mesh Pro). Επιπλέον εγινε χρήση λειτουργειών no-sql βάσης, τύπου JSON, από τις βιβλιοθήκες της Unity, για την αποθήκευση και ανάκτηση αρχείων που ηταν απαραίτητα για την συγκεκριμένη υλοποίηση. Για αυτά τα προγράμματα υπάρχει λεπτομερής ανάπτυξη στα παρακάτω κεφάλαια.

Κεφάλαιο 2 : Σχεδίαση Παιχνιδιού

Στόχος στην σχεδίαση του παιχνιδιού, είναι να κανει τον παίκτη να διασκεδάσει έχοντας μια ευχάριστη και ενθουσιώδη εμπειρία. Στην συγκεκριμένη υλοποίηση, χρησιμοποιήθηκαν απλοί βασικοί κανόνες και μηχανικές παιχνιδιών.

Έτσι μέσα από αυτούς τους βασικούς κανόνες, σκοπός του παίκτη είναι να μπορεί να μάθει ευκολά και γρηγορα να χειρίζεται το παιχνίδι, αλλα ταυτόχρονα να δυσκολεύεται να γίνει κάλος σε αυτό, με αποτέλεσμα αυτό που θα του κινεί το ενδιαφερον για να συνεχίζει να παίζει , θα είναι η πρόκληση στην δυσκολία που εχει το παιχνίδι.
Βέβαια αυτό μπορεί να διαφέρει σε κάθε περίπτωση καθώς μπορεί να υπάρχει η επιλογή επιπέδων δυσκολίας.

Το συγκεκριμένο παιχνίδι χωρίζεται σε δυο μέρη. Στο επεξεργαστικό μέρος και στο ρυθμικό μέρος. Στο επεξεργαστικό μέρος του παιχνιδιού ο παίκτης θα μπορεί να δημιουργεί την δικιά του πίστα θα εχει σαν ανταμοιβή από αυτή την υλοποίηση την χαρα της δημιουργίας. Ενώ στο ρυθμικό μέρος του παιχνιδιού θα εχει σαν ανταμοιβή πόντους που κάθε φορά σκοπός του θα είναι να συλλέγει όσο περισσοτέρους μπορεί.

2.1 Κανόνες και η χρήση του παιχνιδιού

Στο επεξεργαστικό μέρος του παιχνιδιού ο παίκτης εχει την δυνατότητα να δημιουργήσει μια πίστα από σειριακά μπλε και κόκκινα κουτιά και σφαιρίδια.

Πριν ξεκινήσει την δημιουργία της πίστας ζητείται από τον παίκτη να δώσει κάποιες πληροφορίες χρήσιμες για την πίστα. Έτσι πρεπει να συμπληρώσει τα πεδία με το όνομα της πίστας - μουσικού κομματιού, τον ρυθμό του συγκεκριμένου μουσικού κομματιού (BPM) και το ίδιο το μουσικό κομμάτι, δηλαδή το mp3 ή ogg αρχείο που θα παίζει.

Αφού συμπληρώσει τα απαραίτητα πεδία, ξεκινάει την δημιουργία της πίστας, χρησιμοποιώντας ένα πάνελ από τις παρακάτω επιλογές:

1) Add Red Box – Προσθήκη κόκκινου κουτιού
2) Add Blue Box - Προσθήκη μπλε κουτιού
3) Add Sphere - Προσθήκη κόκκινου κουτιού
4) Delete Object – Διαγραφή αντικειμένου
5) Clear All – Καθαρισμός όλων των αντικειμένων
6) Save – Αποθήκευση της τωρινής πίστας

Έτσι ο παίκτης θα κανει κάθε φορά μια από τις παραπάνω επιλογές και θα τοποθετεί, όπως θεωρεί εκείνος κατάλληλα τα αυτά αντικείμενα, ώστε να με βάση την χρονική στιγμή του μουσικού κομματιού, να ταιριάζει με τον ρυθμό του. Αφού ολοκληρώσει αυτή την σειριακή δημιουργία αντικειμένων ο παίκτης θα πρεπει να την αποθηκεύσει, για να μπορεί να χρησιμοποιηθεί αργότερα στο ρυθμικό μέρος του παιχνιδιού.

Επιπλέον ο παίκτης εχει την δυνατότητα να ξανα επεξεργαστεί την πίστα που δημιούργησε, για να μπορεί να διορθώσει τυχών λάθη. Για αυτή την επεξεργασία δίνονται παλι οι ίδιες επιλογές που αναφέρθηκαν πιο πάνω, καθώς επίσης θα μπορεί να επεξεργαστεί το όνομα του τραγουδιού και το BPM (Beats per Minute - Ρυθμός), αυτής της πίστας.

Στο ρυθμικό μέρος του παιχνιδιού, ο παίκτης εχει την δυνατότητα να φορτώσει οποιαδήποτε πίστα είχε φτιαχτεί προηγουμένως από το επεξεργαστικό μέρος του παιχνιδιού, και να παίξει επιλέγοντας μεταξύ 3 διαφορετικών τρόπων παιξίματος (modes). Του Lightsaber Mode, του Saber Mode και του Rifle Mode.

Αφού γίνει η επιλογή τρόπου παιξίματος, και ο παίκτης πατήσει εκκίνηση, τότε ξεκινάνε η μουσική και τα αντικείμενα ξεκινάνε να έρχονται προς το μέρος του παίκτη με ταχύτητα ανάλογη αυτή του ρυθμού (BPM), του τραγουδιού. Αυτά τα αντικείμενα είναι όπως και στο επεξεργαστικό μέρος, είτε μπλε κουτιά, είτε κόκκινα, είτε σφαιρίδια. Στόχος του παίκτη ειναι να μαζέψει όσο πιο πολλούς πόντους μπορεί.

Σε όλα τα modes ο γενικός κανόνας είναι πως ο παίκτης θα εχει στο δεξί του χέρι το μπλε «όπλο» και στο αριστερό του το κόκκινο. Με σκοπό να καταστρέφει αντικείμενα που έχουν το ίδιο χρώμα με αυτό του όπλου του. Επίσης τα σφαιρίδια πρεπει να αποφεύγονται από τον παίκτη.

Στο Lightsaber Mode ο παίκτης εχει την δυνατότητα να αλληλεπιδράσει με τα αντικείμενα με δυο «όπλα», με τους φωτιζόμενους ράβδους (στο δεξί μπλε και στο αριστερό κόκκινο) ή με ασπίδες (και στα δυο χέρια). Ο παίκτης εχει την δυνατότητα οποιαδήποτε στιγμή που παίζει να μπορεί να εναλλάσσει τα όπλα του. Από φωτιζόμενους ράβδους σε ασπίδες και το ανάποδο. Έτσι αν ο παίκτης καταστρέψει με τα όπλα του, λάθος χρώμα κουτί από αυτό που εχει το όπλο του, τότε χάνει x πόντους, ενώ αν είναι το σωστό χρώμα κερδίζει x πόντους. Μόνο στο συγκεκριμένο mode, ο παίκτης μπορεί να αποφύγει τα σφαιρίδια με την χρήση των ασπίδων. Αν τα αποφύγει με τις ασπίδες, τότε θα κερδίσει 2x πόντους. Αν δεν τα χρησιμοποιήσει απλά δεν θα κερδίσει αυτούς τους πόντους.

Στο Saber Mode ο παίκτης εχει για όπλα σπαθιά, και ακολουθεί τον γενικό κανόνα του παιχνιδιού, δηλαδή να καταστρέψει τα αντικείμενα που έχουν ίδιο χρώμα με αυτό του αντίστοιχου όπλου, και να αποφευχθούν τα σφαιρίδια. Οι ίδιοι κανόνες για τους πόντους ισχύουν και σε αυτό το mode όπως στο Lightsaber Mode, με εξαίρεση αυτών, για τις ασπίδες.

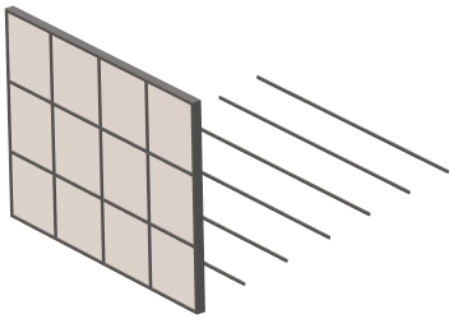
Τέλος στο Rifle Mode ο παίκτης εχει για όπλα πολυβόλα, και ξανα, ακολουθεί τον γενικό κανόνα του παιχνιδιού όπως και στα παραπάνω modes, καθώς το ίδιο ισχύει και για τους πόντους. Η μόνη διαφορά σε αυτό το mode είναι ότι ο παίκτης εχει μπροστά του μια επιφάνεια που θα πρεπει να καταστρέψει τα αντικείμενα την χρονική στιγμή που αυτά εφάπτονται σε αυτή την επιφάνεια. Μόλις περάσουν αυτή την επιφάνεια και δεν έχουν καταστραφεί τότε αυτόματα εξαφανίζονται.

2.2 Σχεδίαση του ρυθμικού μέρους του παιχνιδιού

Η βασική σχεδίαση, του ρυθμικού μέρους του παιχνιδιού, είναι σχετικά απλή. Για να είναι όλα γεωμετρικά στοιχισμένα, εγινε χρήση πλέγματος (GRID) στο 3D περιβάλλον του παιχνιδιού, έτσι ώστε τα αντικείμενα (Streaming Assets) να έχουν τις κατάλληλες αποστάσεις και να μην υπάρχουν αποκλίσεις. Τα αντικείμενα αυτά είναι τριών ειδών. Μπλε κουτί, κόκκινο κουτί και σφαιρίδιο. Έτσι προκειμένου αυτά τα αντικείμενα να φαίνονται σε φυσιολογικό

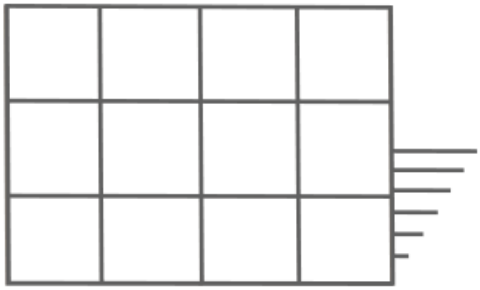
μέγεθος σε σχέση με τον παίκτη, τα πρωτότυπα που δημιουργήθηκαν έχουν διαστάσεις 0.4m πλάτος, 0.4 ύψος και 0.4 βάθος.

Εικόνα 2 - Επιφάνεια Πλέγματος / 3D



Οι αποστάσεις που έχουν αυτά τα αντικείμενα μεταξύ τους στον x και y και z άξονα είναι 0.5m από το κέντρο του κάθε αντικειμένου. Άρα μεταξύ αυτών των αντικειμένων υπάρχει ένα κενό σε όλους τους άξονες που είναι 0.1m. Το επιτρεπόμενο όριο δημιουργίας αυτών τον αντικειμένων είναι σε ένα δισδιάστατο πλέγμα 4x3 στους άξονες x και y αντίστοιχα. Άρα μπορούν να δημιουργηθούν συνολικά μεχρι 12 αντικείμενα, σε ένα σημείο στον άξονα z κάθε φορά.

Εικόνα 3 - Επιφάνεια Πλέγματος / 2D

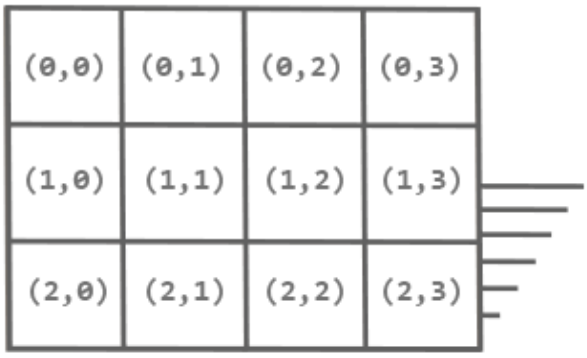


Παρακάτω φαίνονται τα σημεία που μπορούν να δημιουργηθούν σε αυτό το 4x3 πλέγμα.

1. Θέση (0,0) με x : -0.5 και με y : 1.0
2. Θέση (0,1) με x : 0.0 και με y : 1.0
3. Θέση (0,2) με x : 0.5 και με y : 1.0
4. Θέση (0,3) με x : 1.0 και με y : 1.0
5. Θέση (1,0) με x : -0.5 και με y : 0.5
6. Θέση (1,1) με x : 0.0 και με y : 0.5
7. Θέση (1,2) με x : 0.5 και με y : 0.5
8. Θέση (1,3) με x : 1.0 και με y : 0.5
9. Θέση (2,0) με x : -0.5 και με y : 0.0
10. Θέση (2,1) με x : 0.0 και με y : 0.0
11. Θέση (2,2) με x : 0.5 και με y : 0.0

12. Θέση (2,3) με x : 1.0 και με y : 0.0

Εικόνα 4 - Αριθμημένη επιφάνεια πλέγματος



Επιπλέον προκειμένου αυτά τα αντικείμενα να μην είναι πολύ χαμηλά σε σχέση με τον παίκτη (καθώς το y : 0.0 είναι στο ίδιο ύψος με το πάτωμα που πατάει ο παίκτης), έγινε η τοποθέτηση του περιβάλλοντος και του παίκτη y: -1.0m από το κέντρο. Έτσι ο παίκτης θα μπορεί με άνεση να φτάνει τα αντικείμενα που θα χρειαστεί για να καταστρέψει.

Ο άξονας z χρησιμοποιείται για να αντιπροσωπεύει τον χρονο του μουσικού κομματιού που παίζει την συγκεκριμένη χρονική στιγμή, στο οποιο υπάρχει παλι η απόσταση 0.5m το ένα αντικείμενο από το άλλο.

Το σημαντικό για την εύρεση αυτών των αποστάσεων στον z άξονα, που πρεπει να υπαρχουν σε όλη την διάρκεια του μουσικού κομματιού, είναι να το υπολογίσουμε με βάση το BPM (Beats Per Minute), ή ρυθμό του τραγουδιού και την διάρκεια του. Η παρακάτω συνάρτηση υπολογίζει τα συνολικά beats-αποστάσεις που πρεπει να δημιουργηθούν.

```
TotalBeats = (BeatsPerMinute / 60) * SongTimeSeconds;
```

Εκτός από τις αποστάσεις, βασικός παράγοντας του ρυθμικού παιχνιδιού είναι η ταχύτητα με την οποία θα έρχονται αυτά τα αντικείμενα.

```
float distance = 0.5f;
float velocity = distance * BeatsPerMinute / 60;
float ObjectsTimePosition = Time.deltaTime * velocity;
```

Οι παραπάνω υπολογισμοί μας δίνουν την θέση των αντικειμένων για κάθε χρονική στιγμή (ObjectsTimePosition). Η ταχύτητα υπολογίζεται από το velocity. Ενώ το distance είναι η απόσταση μεταξύ των αντικειμένων στον άξονα z.

Όσο για την σχεδίαση του gameplay του ρυθμικού μέρους, στο Lightsaber και Saber Mode, χρησιμοποιήθηκε μια τεχνική για την καταστροφή των αντικειμένων. Ουσιαστικά ο παίκτης με τα όπλα του (ξίφος και φωτόσπαθο), προσπαθεί να διαχωρίσει αυτά τα αντικείμενα σε δυο κομμάτια. Έτσι εγινε χρήση τεχνικής Mesh Splitting στην Unity για να επιτευχθεί κατι τέτοιο.

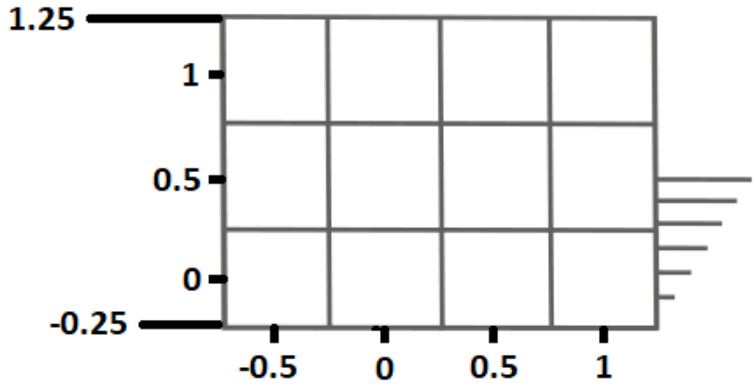
Ενώ στην σχεδίαση του Rifle Mode, τα αντικείμενα καταστρέφονται με το στιγμιαίο πάτημα της σκανδάλης, που με την χρήση των Raycaster μπορεί να αναγνωριστεί αν το Ray του όπλου, χτύπησε (ή αλλιώς ακούμπησε), κάποιο αντικείμενο.

Οι ασπίδες στο mode (τρόπος παιχνιδιού) του Lightsaber καταγράφουν κάποιο χτύπημα ενός αντικειμένου, πολύ απλά με Collision Physics που έχουν τα αντικείμενα. Δηλαδή αυτό ελέγχει πότε δυο αντικείμενα με ιδιότητες φυσικής έρχονται σε επαφή.

Για όλα τα παραπάνω modes γίνεται ενημέρωση του score, που θα φαίνεται σε μια ταμπέλα, οπού ο παίκτης θα μπορεί συνέχεια να ενημερώνεται για το score του.

Τέλος, ο παίκτης (το αντικείμενο VR του παίκτη), τοποθετείται στην συγκεκριμένη σκηνή του ρυθμού μέρους, στο σημείο με συντεταγμένες x: 0.25 y: -0.5 και z: -1. Προκειμένου όλα τα αντικείμενα να φαίνονται στην μέση του παίκτη, θα πρεπει να μετακινήσουμε τον παίκτη προς τα δεξιά κατά 0.25 που είναι το μισό της διάστασης ενός αντικειμένου που δημιουργείται ακριβώς στο κέντρο της σκηνής (δηλαδή με συντεταγμένες xyz = 0.0).

Εικόνα 5 - Θέσεις των αντικειμένων, πάνω στην επιφάνεια πλέγματος



Επίσης, ο παίκτης τοποθετήθηκε σε ύψος y: -0.5 ώστε τα αντικείμενα να είναι στο σωστό ύψος του παίκτη, και στο z: -1.0 για να υπάρχει ένα κενό αναμεσά στο σώμα και τα όπλα του παίκτη, ώστε να προλαβαίνει στον σωστό ρυθμό να χτυπάει τα αντικείμενα που έρχονται κατά πάνω του.

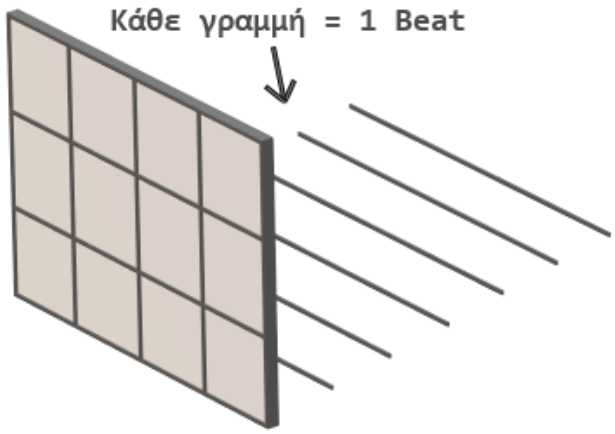
2.3 Σχεδίαση του επεξεργαστικού μέρους, του παιχνιδιού

Για την σχεδίαση του επεξεργαστικού μέρους του παιχνιδιού ακολουθήθηκε παρόμοια σχεδίαση με αυτή του ρυθμικού μέρους, έτσι ώστε να υπάρχει αντιστοίχιση των αντικειμένων που θα δημιουργεί ή θα επεξεργάζεται ο χρήστης. Δηλαδή, χρησιμοποιήθηκαν οι ιδίες διαστάσεις, οι ιδίες αποστάσεις για την δημιουργία των αντικειμένων (Streaming Assets) και τα ιδιά κενά μεταξύ αυτών.

Το βασικό για την δημιουργία των αντικειμένων είναι η επιφάνεια όπου ο παίκτης με την χρήση του αριστερού VR χειριστηρίου στοχεύει πάνω στο πλέγμα (4x3) των δώδεκα θέσεων για να επιλέξει που να τοποθετήσει κάποιο αντικείμενο. Αυτή η επιφάνεια μετακινείται κατά μήκος του z άξονα ώστε να υποδείξει σε ποιο beat – ρυθμικό σημείο του μουσικού

κομματιού βρίσκεται. Έτσι ο παίκτης μπορεί να μετακινήσει αυτή την επιφάνεια με την χρήση του αριστερού χειριστηρίου, για να μετακινηθεί σε οποιο σημείο του τραγουδιού επιθυμεί. Αυτή η μετακίνηση μπορεί να είναι 1 beat την φορά ή πολλαπλά beats την φορά.

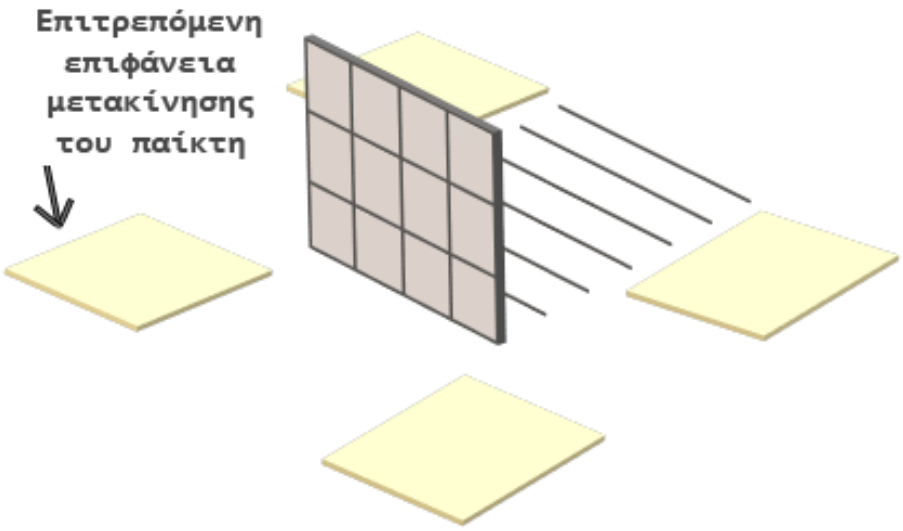
Εικόνα 6 - Επιφάνεια πλέγματος / Υπόδειξη Beat



Έτσι όταν ο παίκτης θελήσει να μετακινηθεί πολλά beats, μπορεί να του δημιουργήσει ναυτία με την ταχύτητα γρήγορης μετακίνησης του. Έτσι, προκειμένου να αποφευχθεί αυτό, ο παίκτης, το περιβάλλον του και η επιφάνεια τοποθέτησης αντικειμένων θα μένουν στατικά – σταθερά, και θα γίνεται μετακίνηση όλων των δημιουργημένων αντικειμένων. Με αυτό τον τροπο επιτυγχάνεται το ίδιο αποτέλεσμα, χωρίς να δημιουργεί δυσφορία στον παίκτη.

Επιπλέον, ο παίκτης εχει περιορισμούς, στην μετακίνηση του. Γύρω από την επιφάνεια τοποθέτησης αντικειμένων υπάρχουν περιμετρικά τέσσερις οριζόντιες τετράγωνες επιφανείς, στις οποίες ο παίκτης με την χρήση του δεξιού χειριστηρίου θα μπορεί να στοχεύσει σε ποια από αυτές τις επιφάνειες θέλει να μετακινηθεί. Αυτή η δυνατότητα μετακίνησης, είναι αρκετά χρήσιμη στην περίπτωση που ο παίκτης δεν μπορεί να στοχεύσει πάνω στην επιφάνεια τοποθέτησης αντικειμένων, και να τοποθετήσει κάποιο αντικείμενο. Δηλαδή κάποια αντικείμενα από την μια πλευρά μπορεί να κρύβονται πίσω από αλλα.

Εικόνα 7 - Επιτρεπόμενη επιφάνεια μετακίνησής παίκτη



Έτσι ο χρήστης εχει πρόσβαση και από τη μπροστά αλλα και από την πίσω πλευρά αυτής της επιφάνειας. Αυτές οι περιοχές (επιφάνειες) μετακίνησης, μένουν και αυτές στάσιμες μαζι με τον παίκτη και το περιβάλλον του.

Τέλος ο παίκτης μπορεί με το αριστερό χειριστήριο να πατήσει play για να παίζει το τραγούδι και να βλέπει τα αντικείμενα που δημιούργησε να έρχονται με τον ρυθμό σε πραγματικό χρονο καθώς και Pause για την παύση αυτών. Η ταχύτητα (Velocity) με την οποία μετακινούνται τα αντικείμενα (streaming Assets), είναι ίδια με αυτή που αναφέρθηκε στο προηγούμενο υποκεφάλαιο.

Κεφάλαιο 3 : Υλοποίηση του παιχνιδιού στην Unity

Για την υλοποίηση του παιχνιδιού χρησιμοποιήθηκε η Unity 2017 μαζι με το SteamVR για την ενσωμάτωση της εικονικής πραγματικότητας. Στην Unity το παιχνίδι εχει χωριστεί σε τρεις σκηνές (scenes). Την mainLobbyScene όπου είναι η πρώτη σκηνή που φορτώνεται όταν ξεκινάει το παιχνίδι, ώστε να μπορεί ο παίκτης να επιλέξει σε ποιο μέρος του παιχνιδιού θέλει να προχωρήσει. Σε αυτή την σκηνή χρησιμοποιείται ένας Scene Manager της Unity για να γίνεται η μετάβαση μεταξύ σκηνών. Η gameScene είναι η σκηνή που αντιπροσωπεύει το ρυθμικό μέρος του παιχνιδιού. Και η σκηνή editorScene, η οποία αντιπροσωπεύει το επεξεργαστικό μέρος του παιχνιδιού. Και στις τρεις σκηνές γίνεται χρήση του Sound Manager και του SteamVR , αλλα με διαφορετικές ρυθμίσεις κατάλληλες για κάθε σκηνή, αναλόγως.

3.1 Υλοποίηση του ρυθμικού μέρους, του παιχνιδιού

Στην σκηνή του ρυθμικού μέρους βασικό για τις γενικές λειτουργίες της, είναι ο Game Manager. Στον Game Manager υπάρχει ένα έξτρα Audio Source, μέσα απο το οποιο κανει αναπαραγωγή το βασικό μουσικό κομμάτι. Τα βασικά prefabs (αντικείμενα της Unity) που είναι σε αυτή την σκηνή είναι το sound manager που είναι υπεύθυνο για τα ηχητικά εφέ της σκηνής, JSON Operations που είναι υπεύθυνο για την ανάκτηση των JSON αρχείων στην Unity, το Menu, που είναι ουσιαστικά ο καμβάς του μενού, της διεπαφής του παιχνιδιού, το Graphic Objects που είναι τα αντικείμενα που υπαρχουν στο περιβάλλον της σκηνής, και το VR Settings που είναι όλα τα prefabs που ενσωματώνουν στο παιχνίδι, την εικονική πραγματικότητα.

Στην αρχή της σκηνής ο παίκτης αντικρίζει μια διεπαφή. Αφού επιλέξει το Mode που θέλει να παίξει, φορτώνει το μουσικό κομμάτι της επιλογής του με την χρήση ενός File Browser και αυτό με την σειρά του διαβάζει το JSON αρχείο του κομματιού και ενημερώνει τον Game Manager για τις ιδιότητες του. Δηλαδή την τοποθεσία στα αρχεία του παιχνιδιού, τον τίτλο του, την διάρκεια του σε δευτερόλεπτα και τον ρυθμό του (BPM).

Εικόνα 8 - Μεταβλητές του Game Manager του Ρυθμικού Μέρους

```
//Variables for Calculations
public string CurrentSongNamePath = "";
public string SongTitle = "";
public float SongTimeSeconds;
public float BeatsPerMinute;
```

Μετα που θα ενημερωθεί το Game Manager, ο παίκτης θα μπορεί να πατήσει το κουμπί Play και να ξεκινήσει το παιχνίδι με το επιλεγμένο mode. Όταν γίνει αυτό γίνονται κάποιες αρχικοποιήσεις από τον Game Manager ανάλογα το mode που επιλέχτηκε.

Εικόνα 9 - Αρχικοποίηση του SaberMode

```
public class SaberMode : MonoBehaviour
{
    void Start()
    {
        RightLightSaber.SetActive(false);
        LeftLightSaber.SetActive(false);
        RightShield.SetActive(false);
        LeftShield.SetActive(false);
        RightHandRifle.SetActive(false);
        LeftHandRifle.SetActive(false);
        RightHandSaber.SetActive(true);
        LeftHandSaber.SetActive(true);
        GameObject.FindGameObjectWithTag("VRSettingsTag")
            .transform.position = new Vector3(0.25f, -0.5f, -1);
        GameObject.FindGameObjectWithTag("MenuTag")
            .transform.position = new Vector3(0.25f, 0, 0);
    }
```

Εικόνα 10 - Αρχικοποίηση του RifleMode

```
public class RifleMode : MonoBehaviour
{
    void Start()
    {
        RightLightSaber.SetActive(false);
        LeftLightSaber.SetActive(false);
        RightShield.SetActive(false);
        LeftShield.SetActive(false);
        RightHandSaber.SetActive(false);
        LeftHandSaber.SetActive(false);
        RightHandRifle.SetActive(true);
        LeftHandRifle.SetActive(true);
        GameObject.FindGameObjectWithTag("VRSettingsTag")
            .transform.position = new Vector3(0.25f, -0.5f, -5);
        GameObject.FindGameObjectWithTag("MenuTag")
            .transform.position = new Vector3(0.25f, 0, -4);
    }
```

Εικόνα 11 - Αρχικοποίηση του LightSaberMode

```
public class LightsaberMode : MonoBehaviour
{
    void Start()
    {
        RightHandRifle.SetActive(false);
        LeftHandRifle.SetActive(false);
        RightHandSaber.SetActive(false);
        LeftHandSaber.SetActive(false);
        RightLightSaber.SetActive(true);
        LeftLightSaber.SetActive(true);
        RightShield.SetActive(false);
        LeftShield.SetActive(false);
        GameObject.FindGameObjectWithTag("VRSettingsTag")
            .transform.position = new Vector3(0.25f, -0.5f, -1);
        GameObject.FindGameObjectWithTag("MenuTag")
            .transform.position = new Vector3(0.25f, 0, 0);
        GameObject.FindGameObjectWithTag("ScoreLabelTag")
            .transform.position = new Vector3(0.25f, 2f, -1f);

        GameObject scorePanel = GameObject.Find("ScorePanel");
        scorePanel.GetComponent<Image>().enabled = true;
        scorePanel.transform.GetChild(0)
            .GetComponent<MeshRenderer>().enabled = true;
        scorePanel.transform.GetChild(1)
            .GetComponent<MeshRenderer>().enabled = true;
    }
```

Η κάθε συνάρτηση αντίστοιχα κανει ανενεργά τα αντικείμενα που πιθανόν να δημιουργήθηκαν προηγουμένως από άλλο mode και να ενεργοποιήσει τα δικά της όπλα, και αντικείμενα στην σκηνή.

Μετα από αυτή την αρχικοποίηση του mode γίνεται δημιουργία του prefab StandardObjects όπου θα εχει σαν παιδιά, τα αντικείμενα (Streaming Assets), μηδενίζει το Total Score, φορτώνει τα αντικείμενα από το JSON αρχείο με την βοήθεια της κλάσης JSON Operations, ξεκινάει να παίζει το μουσικό κομμάτι από τον SoundManager του Game Manager, και τέλος δημιουργεί τα όπλα και τα αντικείμενα του επιλεγμένου mode. Παρακάτω φαίνεται ο κώδικας.

Εικόνα 12 - Συνάρτηση του Game Manager

```
public void StartMainGame()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
    Invoke("ExecuteAfterTimeJsonLoad", 0.5f);
}
```

Εικόνα 13 - Συνάρτηση του Game Manager

```
public void ExecuteAfterTimeJsonLoad()
{
    Instantiate(StandardObjects);
    TotalScore = 0;
    GameObject.Find("JSON_OperationsMainGame")
        .GetComponent<JsonOperationsMainGame>().CreateLoadedObjects();
    GetComponent<AudioSource>().Stop();
    GetComponent<AudioSource>().Play();
    ActivateGameMode();
}
```

Όταν ο παίκτης τερματίσει το παιχνίδι, σταματάει η μουσική και απενεργοποιούνται όλα τα αντικείμενα γύρω και τα όπλα του, εκτός από το μενού και το τελικό score του.

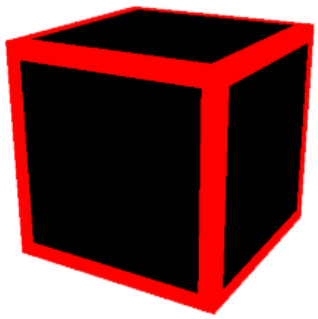
Εικόνα 14 - Update συνάρτηση του Game Manager

```
// Update is called once per frame
void Update()
{
    //Checks if clip has reached the end and activates the menu
    if (!GetComponent<AudioSource>().isPlaying &&
        GetComponent<AudioSource>().clip.length
        .Equals(GetComponent<AudioSource>().time))
    {
        Eyeraycast.SetActive(true);
        UIEndTrackMenu.gameObject.SetActive(true);
        ScoreEndText.GetComponent<TextMeshPro>().text =
            TotalScore.ToString();

        //Disables Mesh from ScorePanel
        GameObject scorePanel = GameObject.Find("ScorePanel");
        scorePanel.GetComponent<Image>().enabled = false;
        scorePanel.transform.GetChild(0)
            .GetComponent<MeshRenderer>().enabled = false;
        scorePanel.transform.GetChild(1)
            .GetComponent<MeshRenderer>().enabled = false;

        //Change between ControllerModel and Weapons
        WeaponsLeft.SetActive(false);
        WeaponsRight.SetActive(false);
        ControllerModelLeft.SetActive(true);
        ControllerModelRight.SetActive(true);
    }
}
```

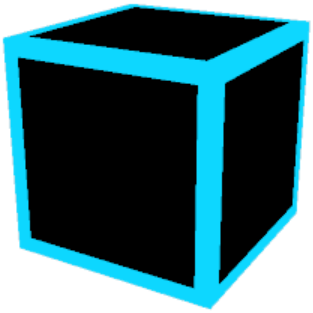
3.1.1 Αντικείμενα και η χρήση τους

Και στα δυο μέρη του παιχνιδιού χρησιμοποιήθηκαν τα ιδιά αντικείμενα (Streaming Assets), τα οποία δημιουργεί και αλληλοεπιδρά ο παίκτης. Όπως αναφέρθηκε στα παραπάνω κεφάλαια, τα αντικείμενα είναι τριών ειδών. Είναι το μπλε κουτί, το κόκκινο κουτί και το σφαιρίδιο.

Εικόνα 15 - Αντικείμενο Κόκκινο Κουτί



Εικόνα 16 - Αντικείμενο Μπλέ Κουτί



Εικόνα 17 - Αντικείμενο Σφαιρίδιο

Στο ρυθμικό μέρος αυτά τα αντικείμενα είναι τα βασικά αντικείμενα που θα πρεπει να αλληλεπιδράσει (καταστρέψει), ο παίκτης χρησιμοποιώντας τα όπλα του, και να γίνει η καταγραφή του score.

Αυτά τα αντικείμενα δημιουργούνται σειριακά με κάποιες ιδιότητες. Οι βασικές ιδιότητες είναι ότι έχουν ένα Mesh Collider, που είναι ουσιαστικά το εξάρτημα το οποιο μπορεί να καταγράψει στην Unity με ποια αλλα αντικείμενα εχει αλληλεπιδράσει. Και ένα Rigid body που ουσιαστικά δίνει στο αντικείμενο ιδιότητες φυσικής, όπως βάρος, τριβή, μάζα και αλλα. Το Rigid body είναι αναγκαίο για να γίνουν κάποιες λειτουργίες μαζι με το Mesh Collider.

Εκτός από αυτά τα βασικά, εχει προστεθεί στις ιδιότητες των αντικειμένων ένα Script, το Splittable που τα καθιστά να μπορούν να διαχωρίζονται στα δυο, με την βοήθεια του Mesh Splitter.

3.1.2 Όπλα και η χρήση τους

Για την χρήση της εξωτερικής βιβλιοθήκης Mesh Splitter, δημιουργήθηκε μια κλάση η WeaponSplitController. Αυτή την χρησιμοποιούμε σε κάθε όπλο (ε) για να γίνεται η καταστροφή των αντικειμένων, όταν αυτά έρχονται σε επαφή με τα Streaming Assets. Επιπλέον, τα Streaming Assets έχουν την κλάση Splittable για να γνωρίζει το σύστημα ποια αντικείμενα προορίζονται να καταστραφούν και ποια οχι .

Έτσι όταν ένα όπλο ή η γραμμή (Line Renderer) που δείχνει το όπλο (στην περίπτωση των Rifles), ακουμπήσει (ή κανει Collide), ένα Splittable αντικείμενο, εκτελείται η παρακάτω συνάρτηση ως Event, η οποία ελέγχει με βάση την σχεδίαση του παιχνιδιού μας αν πρεπει να προσθέσει ή να αφαιρέσει πόντους, ή να κανει αναπαράσταση σωματιδίων (particles) και ηχητικών εφέ (Εφέ κοψίματος αντικειμένου). Αυτή η συνάρτηση χρησιμοποιείται και για τα τρία modes οποτε βλέπουμε μέσα της υλοποίηση διαφορετικών περιπτώσεων.

Εικόνα 18 - Συνάρτηση της WeaponSplitController

```
public void OnTriggerEnter(Collider other)
{
    //Make objects have gravity once they are split.
    if (other.gameObject.tag.Equals("SpawnObject") &&
        !other.gameObject.name.Equals("SpherePrefab(Clone)") &&
        other.gameObject.GetComponent<Splitable>() != null)
    {
        //Play Particle System
        other.transform.GetChild(0).GetChild(0).
            GetComponent<ParticleSystem>().Play();
        other.transform.GetChild(0).GetChild(1).
            GetComponent<ParticleSystem>().Play();

        if ((WeaponColor.Equals("Red") &&
            other.gameObject.name.Equals("RedCubePrefab(Clone)")) ||
            (WeaponColor.Equals("Blue")
            && other.gameObject.name.Equals("BlueCubePrefab(Clone)")))
        {
            AddScorePoints(10);
            PlaySfxOnMode(true, other);
        }
        else
        {
            RemoveScorePoints(10);
            PlaySfxOnMode(false, other);
        }
    }
```

Εικόνα 19 - Συνάρτηση της WeaponSplitController

```
    else if (other.gameObject.tag.Equals("SpawnObject") &&
        other.gameObject.name.Equals("SpherePrefab(Clone)") &&
        other.gameObject.GetComponent<Splitable>() != null)
    {
        //Play Particle System
        other.transform.GetChild(0).GetChild(0)
            .GetComponent<ParticleSystem>().Play();
        other.transform.GetChild(0).GetChild(1)
            .GetComponent<ParticleSystem>().Play();

        RemoveScorePoints(20);
        SoundManager.instance.PlaySingle(
            other.GetComponent<AudioClipSFX>().audioClip[0]
        );
    }

    collisionEnterPos = transform.position;
}
```

- **Rifles – Πολυβόλα**

Εικόνα 20 - Πολυβόλο δεξιού χεριού

Εικόνα 21 - Πολυβόλο αριστερού χεριού

Τα Rifles ενεργοποιούνται στο Rifle Mode και το κάθε όπλο εχει ένα script με τις ιδιότητες του. Αυτά τα script ελέγχουν σε κάθε καρέ αν ο παίκτης πάτησε την σκανδάλη («Squeeze») κάποιου χειριστηρίου. Αν πατηθεί, θα δημιουργηθεί ένα Line Renderer (δηλαδή μια ευθεία γραμμή) και μαζι με ένα Raycaster να ελέγξει αν αυτή η γραμμή χτύπησε (έκανε Collide) κάποιο αντικείμενο (που εχει Mesh Collider). Μετά διαγράφει το Splittable Component από το αντικείμενο που «κατέστρεψε», ώστε να μην ξανα γίνει Collide, και στέλνει πληροφορίες για το αντικείμενο αυτό στην κλάση Weapon Split Controller για περεταίρω επεξεργασία.

Εικόνα 22 - Κλάση RightRifleProperties

```
public class RightRifleProperties : MonoBehaviour {

    private LineRenderer lr;

    // Use this for initialization
    void Start()
    {
        lr = gameObject.transform.GetChild(2).GetComponent<LineRenderer>();
        lr.enabled = false;
    }
```

Εικόνα 23 - Update της κλασης RightRifleProperties

```
// Update is called once per frame
void Update()
{
    if (SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.RightHand))
    {
        lr.enabled = true;
        RaycastHit hit;
        if (Physics.Raycast(transform.position, transform.forward, out hit))
        {
            if (hit.collider)
            {
                Debug.Log("COLLIDED");
                Destroy(hit.collider.gameObject.GetComponent<Splitable>());
                WeaponSplitController.instance.WeaponColor = "Blue";
                WeaponSplitController.instance.OnTriggerEnter(hit.collider);
            }
        }
    }
    else if (SteamVR_Input.GetStateUp(("Squeeze"), SteamVR_Input_Sources.RightHand))
    {
        lr.enabled = false;
    }
}
```

Εικόνα 24 - Κλάση LeftRifleProperties

```
public class LeftRifleProperties : MonoBehaviour
{
    private LineRenderer lr;

    // Use this for initialization
    void Start()
    {
        lr = gameObject.transform.GetChild(2).GetComponent<LineRenderer>();
        lr.enabled = false;
    }
```

Εικόνα 25 - Update της κλάσης LeftRifleProperties

```
// Update is called once per frame
void Update()
{
    if (SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.LeftHand))
    {
        lr.enabled = true;
        RaycastHit hit;
        if (Physics.Raycast(transform.position, transform.forward, out hit))
        {
            if (hit.collider)
            {
                Debug.Log("COLLIDED");
                Destroy(hit.collider.gameObject.GetComponent<Splitable>());
                WeaponSplitController.instance.WeaponColor = "Red";
                WeaponSplitController.instance.OnTriggerEnter(hit.collider);
            }
        }
    }
    else if (SteamVR_Input.GetStateUp(("Squeeze"), SteamVR_Input_Sources.LeftHand))
    {
        lr.enabled = false;
    }
}
```

Στις παραπάνω εικόνες φαίνονται οι κλάσεις που έχουν δημιουργηθεί για το δεξί και για το αριστερό όπλο αντίστοιχα. Χρησιμοποιούνται διαφορετικές κλάσεις σε κάθε όπλο καθώς αργότερα μπορεί να χρειαστεί η προσθήκη κάποιας διαφορετικής λειτουργίας στο ένα όπλο σε σχέση με το άλλο.

Η κλάση Rifle Laser που φαίνεται στην παρακάτω εικόνα χρησιμοποιείται για την υλοποίηση αυτής της ευθείας γραμμής (Line Renderer) του δημιουργείται στο όπλο, με το πάτημα της σκανδάλης του χειριστηρίου. Με αυτή την κλάση ορίζουμε τις ιδιότητες αυτής της γραμμής.
Επίσης η απεικόνιση αυτής της γραμμής, δεν πρεπει να διαπερνάει τα αντικείμενα που κανει Collide, οπότε ορίζεται το τέλος αυτής της γραμμής να είναι στο ίδιο σημείο με το σημείο που έκανε Collide με κάποιο αντικείμενο.

Εικόνα 26 - Κλάση RifleLaser

```
public class RifleLaser : MonoBehaviour
{
    private LineRenderer lr;

    // Use this for initialization
    void Start()
    {
        lr = GetComponent<LineRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
        lr.SetPosition(0, transform.position);
        RaycastHit hit;
        if (Physics.Raycast(transform.position, transform.forward, out hit))
        {
            if (hit.collider)
            {
                lr.SetPosition(1, hit.point);
            }
        }
        else lr.SetPosition(1, transform.forward * 5000);
    }
}
```

- **Sabers και Lightsabers – Ξίφη και Φωτόσπαθα**

Τα Light Sabers χρησιμοποιούνται στο Light Saber Mode, ενώ τα Sabers χρησιμοποιούνται στο Saber Mode. Και στις δυο περιπτώσεις η υλοποίηση εγινε με τον ίδιο τροπο. Έτσι αυτά, χρησιμοποιούν την κλάση Weapon Split Controller για τον έλεγχο των συμβάντων (Events) αυτών των όπλων. Τα φωτόσπαθα και τα σπαθιά, χρησιμοποιούν μια παραπάνω συνάρτηση από αυτή των Rifles, που υλοποιήθηκε σε αυτή την κλάση, την CreateCutPlane, η οποία δημιουργεί ένα αόρατο αντικείμενο plane στην σκηνή, και είναι ακριβώς πάνω στην θέση που είναι και το ξίφος - φωτόσπαθο. Ουσιαστικά ο ρόλος αυτού του plane είναι να χωρίσει ένα αντικείμενο σε δυο μέρη. Αυτή η συνάρτηση εκτελείται στην μέσα στην συνάρτηση OnTriggerExit. Η OnTriggerExit είναι ένα Event που εκτελείται μόλις το όπλο σταματήσει να κανει Collide με ένα Streaming Asset. Στις παρακάτω εικόνες φαίνονται αυτές οι συναρτήσεις, και τα όπλα.

Εικόνα 27 - Συνάρτηση της κλάσης WeaponSplitController

```
void OnTriggerExit(Collider other)
{
    collisionExitPos = transform.position;
    CreateCutPlane(collisionEnterPos, collisionExitPos, bladeCollider.transform.up);
}
```

Εικόνα 28 - Συνάρτηση της κλάσης WeaponSplitController

```
private void CreateCutPlane(Vector3 startPos, Vector3 endPos, Vector3 forward)
{
    Vector3 center = Vector3.Lerp(startPos, endPos, .5f);
    Vector3 cut = (endPos - startPos).normalized;
    Vector3 fwd = forward.normalized;
    Vector3 normal = Vector3.Cross(fwd, cut).normalized;

    GameObject goCutPlane = new GameObject(
        "CutPlane",
        typeof(BoxCollider),
        typeof(Rigidbody),
        typeof(SplitterSingleCut));

    goCutPlane.GetComponent<Collider>().isTrigger = true;
    Rigidbody bodyCutPlane = goCutPlane.GetComponent<Rigidbody>();
    bodyCutPlane.useGravity = false;
    bodyCutPlane.isKinematic = true;

    Transform transformCutPlane = goCutPlane.transform;
    transformCutPlane.position = center;
    transformCutPlane.localScale = new Vector3(CutPlaneSize, .01f, CutPlaneSize);
    transformCutPlane.up = normal;
    float angleFwd = Vector3.Angle(transformCutPlane.forward, fwd);
    transformCutPlane.RotateAround(center, normal, normal.y < 0f ? -angleFwd : angleFwd);
}
```

Εικόνα 29 - Όπλο / Μπλέ φωτόσπαθο δεξιού χεριού

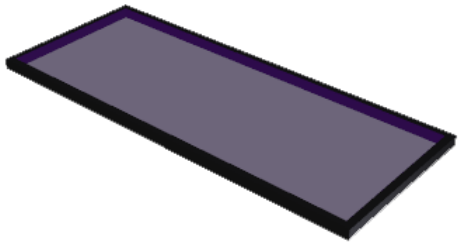
Εικόνα 30 - Όπλο / Κόκκινο φωτόσπαθο αριστερού χεριού

Εικόνα 31 - Όπλο / Μπλέ ξίφος δεξιού χεριού

Εικόνα 32 - Όπλο / Κόκκινο ξίφος αριστερού χεριού

- **Shield – Ασπίδα**

Εικόνα 33 - Όπλο / Ασπίδα



Η ασπίδα χρησιμοποιείται στο Lightsaber Mode και χρησιμοποιεί την κλάση ShieldController για τον έλεγχο των αλληλεπιδράσεων με τα Streaming Assets.

Εικόνα 34 - Κλάση ShieldController

```
public class ShieldController : MonoBehaviour {

    public void OnTriggerEnter(Collider other)
    {
        Debug.Log("Collided Shield");
        //Make objects have gravity once they are split.
        if (other.gameObject.tag.Equals("SpawnObject") &&
            other.gameObject.name.Equals("SpherePrefab(Clone)")
            && other.gameObject.GetComponent<Splitable>() != null)
        {
            //Activate Gravity on Rigidbody
            other.GetComponent<Rigidbody>().isKinematic = false;
            other.GetComponent<Rigidbody>().useGravity = true;

            //Play Particle System
            other.transform.GetChild(0).GetChild(0)
                .GetComponent<ParticleSystem>().Play();
            other.transform.GetChild(0).GetChild(1)
                .GetComponent<ParticleSystem>().Play();

            //Adding Thrust Force to GameObjects
            other.gameObject.GetComponent<Rigidbody>()
                .AddForce(0, 1, 1, ForceMode.Impulse);
            //Adding Rotation to Gameobjects
            other.gameObject.GetComponent<Rigidbody>()
                .AddTorque(new Vector3(0, 1, 0), ForceMode.Impulse);


            AddScorePoints(20);
            PlaySfxOnMode(true, other);
            Destroy(other.gameObject.GetComponent<Splitable>());
        }
    }
```

Έτσι ελέγχεται αν ο παίκτης χτύπησε (έκανε collide), με την ασπίδα του κάποιο σφαιρίδιο, και εφόσον το χτυπήσει θα καταστραφούν τα σφαιρίδια και θα προστεθούν οι πόντοι στον Game Manager.

Επίσης στο Lightsaber Mode ο παίκτης εχει την δυνατότητα να ενεργοποιήσει ή να απενεργοποιήσει οποιαδήποτε στιγμή την ασπίδα (κάνοντας αλλαγή από Light Sabers σε ασπίδα) με ένα πλαϊνό κουμπί του χειριστηρίου που του εχει οριστεί το χαρακτηριστικό «GrabGrip». Στην παρακάτω εικόνα φαίνεται πως ελέγχεται και στα δυο χειριστήρια αν πατήθηκε αυτό το κουμπί, ώστε να γίνει η εναλλαγή των όπλων.

Εικόνα 35 - Συνάρτηση Update της κλάσης ShieldController

```
private void Update()
{
    //Left Hand Shield on Grip Button
    if (SteamVR_Input.GetStateDown(("GrabGrip"), SteamVR_Input_Sources.LeftHand))
    {
        LeftLightSaber.SetActive(false);
        LeftShield.SetActive(true);
    }
    else if (SteamVR_Input.GetStateUp(("GrabGrip"), SteamVR_Input_Sources.LeftHand))
    {
        LeftShield.SetActive(false);
        LeftLightSaber.SetActive(true);
    }

    //Right Hand Shield on Grip Button
    if (SteamVR_Input.GetStateDown(("GrabGrip"), SteamVR_Input_Sources.RightHand))
    {
        RightLightSaber.SetActive(false);
        RightShield.SetActive(true);
    }
    else if (SteamVR_Input.GetStateUp(("GrabGrip"), SteamVR_Input_Sources.RightHand))
    {
        RightShield.SetActive(false);
        RightLightSaber.SetActive(true);
    }
}
```

3.2 Υλοποίηση του επεξεργαστικού μέρους, του παιχνιδιού

Όπως και στην σκηνή του ρυθμικού μέρους, έτσι και στο επεξεργαστικό μέρος το βασικό είναι ο Game Manager, που διαχειρίζεται τις κύριες λειτουργίες και ιδιότητες της σκηνής. Ο Game Manager του επεξεργαστικού μέρους εχει τις ίδιες και κάποιες ακόμα ιδιότητες με αυτές του ρυθμικού. Αλλα εχει διαφορετικές συναρτήσεις που εκτελούν διαφορετικές λειτουργίες.

Εικόνα 36 - Μεταβλητές Game Manager Επεξεργαστικού Μέρους

```
//Variables for Calculations
public string CurrentSongNamePath = "";
public string SongTitle = "";
public float SongTimeSeconds;
public float BeatsPerMinute;
public float TotalBeats;
```

Ο παίκτης διαχειρίζεται μια διεπαφή που βρίσκεται στο αριστερό του χειριστήριο για να κανει τις επιλογές που του είναι διαθέσιμες. Μερικές από τις επιλογές είναι αν θέλει να δημιουργήσει μια καινούρια πίστα ή αν θέλει να επεξεργαστεί μια ήδη δημιουργημένη. Σε κάθε περίπτωση απαραίτητο για να ξεκινήσει την δημιουργία της πίστας είναι, η συμπλήρωση των πεδίων, όπως το όνομα του μουσικού κομματιού, το BPM (ρυθμό του κομματιού) και το ίδιο το μουσικό αρχείο (.mp3 - .ogg).

Έτσι ελέγχεται από ένα Script αν οι τιμές των μεταβλητών στο Game Manager υπαρχουν και δεν είναι κενές, ενεργοποιώντας το κουμπί στην διεπαφή, που δίνει την δυνατότητα στον παίκτη να ξεκινήσει την επεξεργασία.

Αφού πατήσει ο παίκτης αυτό το κουμπί και βρίσκεται στην δημιουργία πίστας εκτελούνται οι παρακάτω συναρτήσεις από τον Game Manager.

Εικόνα 37 - Συνάρτηση του Game Manager

```
public void StartEditor()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
    Invoke("ExecuteAfterTime", 0.5f);
}
```

Εικόνα 38- Συνάρτηση του Game Manager

```
public void ExecuteAfterTime()
{
    Instantiate(StandardObjects);

    UICanvasEditSongTitle.
        GetComponent<InputField>().text = SongTitle;
    UICanvasEditSongBPM.
        GetComponent<InputField>().text = BeatsPerMinute.ToString();

    //Updating Properties
    InitAllObjectsProperties();
}
```

Εικόνα 39 - Συνάρτηση του Game Manager

```
public void InitAllObjectsProperties()
{
    Properties tempProperties = new Properties();
    tempProperties.SongTitle = SongTitle;
    tempProperties.SongTimeSeconds = SongTimeSeconds;
    tempProperties.BeatsPerMinute = BeatsPerMinute;
    allObjects.properties.Add(tempProperties);
}
```

Αυτό που κανει η συνάρτηση Start Editor είναι να καταστρέψει τα Standard Object αν είχαν δημιουργηθεί πιο πριν, και να εκτελέσει την συνάρτηση Execute After Time μετά από μισό δευτερόλεπτο, ώστε να μην δημιουργούνται προβλήματα στην Unity με την αρχικοποίηση των Standard Objects.

Η συνάρτηση Execute After Time κάνει αρχικοποίηση των Standard Objects, ορίζει τις τιμές για το όνομα και τον ρυθμό του κομματιού στην διεπαφή και ορίζει μέσω της συνάρτησης InitAllObjectsProperties, τα Properties – ιδιότητες που εχει το συγκεκριμένο μουσικό κομμάτι που μόλις δημιούργησε ο παίκτης. Αυτά τα Properties που αντιστοιχούν σε κάθε μουσικό κομμάτι, χρησιμοποιούνται και για την αποθήκευση τους σε JSON βάση.

Εικόνα 40 - Συνάρτηση του Game Manager

```
public void StartEditorAndLoad()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
    Invoke("ExecuteAfterTimeJsonLoad", 0.5f);
}
```

Εικόνα 41 - Συνάρτηση του Game Manager

```
public void ExecuteAfterTimeJsonLoad()
{
    Instantiate(StandardObjects);

    UICanvasEditSongTitleProperties.
        GetComponent<InputField>().text = SongTitle;
    UICanvasEditSongBPMProperties.
        GetComponent<InputField>().text = BeatsPerMinute.ToString();

    GameObject.Find("JSON_Operations").
        GetComponent<JsonOperations>().CreateLoadedObjects();
}
```

Ενώ αν ο παίκτης πατήσει το αντίστοιχο κουμπί για να ξεκινήσει την επεξεργασία της πίστας, η οποία εχει ήδη δημιουργηθεί προηγουμένως , τότε εκτελούνται οι συναρτήσεις που απεικονίζονται παραπάνω.

Η εκτέλεση των συναρτήσεων γίνεται με τον ίδιο τροπο, όπως με τις προηγούμενες, με την μονή διαφορά ότι δεν εκτελείται πλέον η συνάρτηση InitAllObjectProperties, καθώς έχουν ήδη αρχικοποιήσεις αυτές οι τιμές προηγουμένως. Αλλά, εκτελείται μια συνάρτηση από το την κλάση JSON_Operations η οποία δημιουργεί όλα τα αντικείμενα (ουσιαστικά την πίστα), από την JSON βάση, που είχαν δημιουργηθεί από τον παίκτη.

Όλες αυτές οι συναρτήσεις είναι υλοποιημένες στην κλάση του Game Manager.

3.2.1 Αντικείμενα και η χρήση τους

Τα αντικείμενα ή αλλιώς τα Streaming Assets είναι τα ίδια με αυτά που αναφέρθηκαν στο ρυθμικό μέρος του παιχνιδιού, με τις ίδιες ιδιότητες. Η μόνη διαφορά είναι στην χρήση τους. Στο επεξεργαστικό μέρος του παιχνιδιού, χρησιμοποιούνται για να φτιάξει ο παίκτης την πίστα. Αυτό γίνεται με την βοήθεια των VR χειριστηρίων που εχει ο παίκτης.

Αφού επιλέξει ο παίκτης από το μενού που βρίσκεται στο αριστερό χειριστήριο, την επιλογή αντικειμένου που θέλει να δημιουργήσει πάνω στο πλέγμα, και σημαδεύει (υποδεικνύει) με το δεξί χειριστήριο που εχει ένα Laser Pointer πάνω του, σε ποιο σημείο του πλέγματος θέλει να το τοποθετήσει. Καθώς ο παίκτης περνάει το Laser Pointer πάνω από το πλέγμα γίνεται προεπισκόπηση του αντικειμένου πάνω από το σημείο που σημαδεύει. Το αντικείμενο θα δημιουργηθεί όταν ο παίκτης πατήσει την σκανδάλη του χειριστηρίου.

Κάθε αντικείμενο που δημιουργείται. προστίθεται οι ιδιότητες του σε μια λίστα μαζι με όλα τα αντικείμενα.Η μεταβλητή της κλάσης AllObjects είναι αρχικοποιημένη στον Game Manager. Στην παρακάτω εικόνα φαίνονται αυτές οι ιδιότητες και η λίστα.

Εικόνα 42 - Κλάση AllObjects

```
[Serializable]
public class AllObjects
{
    public List<Properties> properties = new List<Properties>();
    public List<Objects> objects = new List<Objects>();
}

[Serializable]
public class Objects
{
    public int objectType;
    public float xPosition;
    public float yPosition;
    public float zTime;
}
```

Όταν ο παίκτης κανει μια επιλογή από την διεπαφή του αριστερού χειριστηρίου, η κάθε λειτουργία διαχειρίζεται από την κλάση UI Input Manager. Η διεπαφή εχει τέσσερις επιλογές αλληλεπίδρασης και χρήσης, με τα αντικείμενα. Έτσι στην κλάση UI Input Manager έχουν υλοποιηθεί τέσσερις συναρτήσεις για αυτό τον σκοπό.

- **DeleteObject**

Η Delete Object είναι η συνάρτηση υπεύθυνη για την διαγραφή των δημιουργημένων αντικειμένων. Η οποία για να βρει ποιο αντικείμενο να διαγράψει παίρνει τις συντεταγμένες x,y,z του αντικειμένου που επέλεξε προς διαγραφή και το συγκρίνει με όλα τα αντικείμενα στην λίστα που έχουν δημιουργηθεί. Όταν βρεθεί το αντικείμενο, διαγράφονται αυτές οι συντεταμένες από την λίστα και διαγράφεται και το δημιουργημένο αντικείμενο από την σκηνή.

Εικόνα 43 - Συνάρτηση της UIInputManager

```
//Delete Object Option
public static void DeleteObject(PointerEventArgs hoverEvent)
{
    //Getting Moving Objects z position, because objects..
    //..are moved in the negative direction so the plane stays still
    float negativePosition = hoverEvent.target.parent.parent.transform.position.z;

    if (hoverEvent.target.tag.Equals("SpawnObject"))
    {
        //Searching for the selected object, that is about to get deleted
        Objects searchedObject = new Objects();
        foreach (Objects item in GameManager.allObjects.objects)
        {
            //Debug.Log("Item " + item.zTime);
            if (item.xPosition.Equals(hoverEvent.target.transform.position.x) &&
                item.yPosition.Equals(hoverEvent.target.transform.position.y) &&
                item.zTime.Equals(hoverEvent.target.transform.position.z
                - negativePosition))
            {
                searchedObject = item;
            }
        }

        //Removing from Objects
        GameManager.allObjects.objects.Remove(searchedObject);
        //Destroying the gameObject from the scene
        Destroy(hoverEvent.target.gameObject);
    }
}
```

- **AddRedBox – AddBlueBox**

Οι συναρτήσεις AddRedBox και AddBlueBox εκτελούν την ίδια λειτουργία αλλα για η κάθε μια για διαφορετικό χρώμα κουτί. Η AddRedBox είναι για την δημιουργία του κόκκινου κουτιού, ενώ η AddBlueBox είναι για την δημιουργία του μπλε κουτιού.

Η κάθε μια συνάρτηση ξεχωριστά, πρώτα θα βρει το prefab (προ δημιουργημένα αντικείμενα), του αντικειμένου από την σκηνή, και θα δημιουργήσει ένα αντίγραφο αυτού του prefab στην σκηνή με τις συντεταγμένες x,y,z που το δόθηκαν από το σημείο που ο παίκτης επέλεξε να το τοποθετήσει πάνω στο πλέγμα.

Έπειτα δημιουργείται μια προσωρινή μεταβλητή τύπου Objects (από την κλάση Objects), και περνάει σε κάθε πεδίο αυτής της μεταβλητής τις x,y,z συντεταγμένες του αντικειμένου που πρόκειται να δημιουργηθεί καθώς και τον τύπο του αντικειμένου (Για το κόκκινο κουτί το objectType είναι 0, και για το μπλε είναι 1). Αυτή την μεταβλητή με αυτές τις πληροφορίες, περνιούνται στην λίστα objects που βρίσκεται στην κλάση AllObjects, μέσω της μεταβλητής allObjects του Game Manager.

Στις παρακάτω εικόνες φαίνονται οι συναρτήσεις με τις λειτουργίες που αναφέρθηκαν.

Εικόνα 44 - Συνάρτηση της UIInputManager

```
//Add Red Box Option
public static void AddRedBox(GameObject obj, Transform currentTransform)
{
    Debug.Log("Add Red Box");

    //Getting Prefab and Setting Variables
    obj = GameObject.Find("EditorRedCube");

    //Create New Cube Object
    GameObject game = Instantiate(obj, new Vector3(
            currentTransform.position.x,
            currentTransform.position.y,
            currentTransform.position.z),
        Quaternion.identity,
        GameObject.Find("CreatedObjects").transform);

    //Adding Object info to Serializable object
    Objects tempObject = new Objects();
    tempObject.objectType = 0;
    tempObject.xPosition = (float)game.transform.position.x;
    tempObject.yPosition = (float)game.transform.position.y;
    tempObject.zTime = Mathf.Abs((float)GameObject.Find("CreatedObjects").
                                            transform.parent.position.z);

    //Adding tempObject to AllObjects
    GameManager.allObjects.objects.Add(tempObject);
}
```

Εικόνα 45 - Συνάρτηση της UIInputManager

```
//Add Blue Box Option
public static void AddBlueBox(GameObject obj, Transform currentTransform)
{
    Debug.Log("Add Blue Box");

    //Getting Prefab and Setting Variables
    obj = GameObject.Find("EditorBlueCube");

    //Create New Cube Object
    GameObject game = Instantiate(obj, new Vector3(
            currentTransform.position.x,
            currentTransform.position.y,
            currentTransform.position.z),
        Quaternion.identity,
        GameObject.Find("CreatedObjects").transform);

    //Adding Object info to Serializable object
    Objects tempObject = new Objects();
    tempObject.objectType = 1;
    tempObject.xPosition = (float)game.transform.position.x;
    tempObject.yPosition = (float)game.transform.position.y;
    tempObject.zTime = Mathf.Abs((float)GameObject.Find("CreatedObjects")
                                            .transform.parent.position.z);
    //Adding tempObject to AllObjects
    GameManager.allObjects.objects.Add(tempObject);
}
```

- **AddSphere**

Η συνάρτηση AddSphere είναι υπεύθυνη για την δημιουργία του Sphere (σφαιριδίου), αντικειμένου. Όπως και στην δημιουργία των κουτιών, έτσι και σε αυτή την περίπτωση η συνάρτηση πρώτα βρίσκει το prefab του σφαιριδίου και το περνάει σε μια προσωρινή μεταβλητή. Στην συνέχεια δημιουργείται ένα αντίγραφο αυτού το prefab στην σκηνή, στις συντεταγμένες x,y,z που δόθηκαν από τον παίκτη.

Επίσης δημιουργείται η προσωρινή μεταβλητή τύπου Objects που περνιούνται σε αυτήν οι συντεταμένες και ο τύπος του αντικειμένου (για το σφαιρίδιο το objectType είναι 2), με σκοπό να περαστεί αυτή η μεταβλητή στην λίστα allObjects με όλα τα δημιουργημένα αντικείμενα, μέσω του Game Manager.

Εικόνα 46 - Συνάρτηση της UIInputManager

```
//Add Sphere Option
public static void AddSphere(GameObject obj, Transform currentTransform)
{
    Debug.Log("Add Sphere");

    //Getting Prefab and Setting Variables
    obj = GameObject.Find("EditorSpherePrefab");

    //Create New Cube Object
    GameObject game = Instantiate(obj, new Vector3(
            currentTransform.position.x,
            currentTransform.position.y,
            currentTransform.position.z),
        Quaternion.identity,
        GameObject.Find("CreatedObjects").transform);

    //Adding Object info to Serializable object
    Objects tempObject = new Objects();
    tempObject.objectType = 2;
    tempObject.xPosition = (float)game.transform.position.x;
    tempObject.yPosition = (float)game.transform.position.y;
    tempObject.zTime = Mathf.Abs((float)GameObject.Find("CreatedObjects")
                                                .transform.parent.position.z);
    //Adding tempObject to AllObjects
    GameManager.allObjects.objects.Add(tempObject);
}
```

- **ClearAllObjects**

Αυτή η συνάρτηση όταν εκτελείται από την διεπαφή του παίκτη, διαγράφει όλα τα αντικείμενα της λίστας των αντικειμένων (Streaming Assets) καθώς και τα ίδια τα δημιουργημένα αντικείμενα της σκηνής.

Εικόνα 47 - Συνάρτηση της UIInputManager

```
//Deleting all Objects Created
public void ClearAllObjects()
{
    GameObject createdObjects = GameObject.Find("CreatedObjects");
    //Deleting Physical Objects
    foreach (Transform child in createdObjects.transform)
    {
        GameObject.Destroy(child.gameObject);
    }

    //Destroy Objects as data from allObjects
    GameManager.allObjects.objects.Clear();
}
```

3.2.2 Πλέγμα επίπεδης επιφάνειας και η χρήση του

Όπως στην σχεδίαση έτσι και στην υλοποίηση του επεξεργαστικού μέρους, εγινε χρήση μιας επίπεδης επιφάνειας χωρισμένη σε πλέγμα (Grid) με διαστάσεις 4x3 ώστε ο παίκτης να τοποθετεί πάνω του τα αντικείμενα (Streaming Assets).

Στην κλάση Grid εχει δημιουργηθεί μια συνάρτηση η GetNearestPointOnGrid η οποία μετατρέπει ένα σημείο στο ποιο κοντινό σημείο πάνω σε αυτό το πλέγμα. Ουσιαστικά μετατρέπει απο τις συντεταγμένες αυτού του σημείου στις απόλυτες τιμές τους και επιστρέφει το αποτέλεσμα (που είναι και αυτό σημείο – Vector3), όπως φαίνεται στην παρακάτω εικόνα.

Εικόνα 48 - Συνάρτηση της κλάσης Grid

```
//Returns nearest point on grid
public Vector3 GetNearestPointOnGrid(Vector3 position)
{
    position -= transform.position;

    int xCount = Mathf.RoundToInt(position.x / GridDisplacement);
    int yCount = Mathf.RoundToInt(position.y / GridDisplacement);
    int zCount = Mathf.RoundToInt(position.z / GridDisplacement);

    Vector3 result = new Vector3(
        xCount * GridDisplacement,
        yCount * GridDisplacement,
        zCount * GridDisplacement
     );

    result += transform.position;
    return result;
}
```

Η κλάση Cube Grid Snap εχει λειτουργίες που έχουν αν κάνουν με την αλληλεπίδραση των αντικειμένων πάνω στη επίπεδη επιφάνεια του πλέγματος. Στην συνάρτηση Update ελέγχει σε κάθε καρέ αν πατήθηκε η σκανδάλη από το δεξί χειριστήριο και αν το Laser Pointer ηταν μέσα στα όρια της επιφάνειας, ώστε να εκτελέσει μια λειτουργία για την δημιουργία η διαγραφή αντικειμένων πάνω στο πλέγμα. Η παρακάτω εικόνα, δείχνει αυτή την συνάρτηση.

Εικόνα 49 - Συνάρτηση Update της κλάσης CubeGridSnap

```
// Update is called once per frame
void Update()
{
    //If trigger is presses on GridPlane
    if (hoverEvent.target != null && (hoverEvent.target.tag.Equals("TempCubeTag")
        || hoverEvent.target.tag.Equals("SpawnObject")) &&
         SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.RightHand))
    {
        //Checks Fuction Selected from UI and activates it
        UIInputManager.DoSelectedFunction(hoverEvent, obj, CurrentGameObjectTransofrm);
    }
}
```

Στην ίδια κλάση υπάρχουν οι συναρτήσεις που λειτουργούν με Event από την βιβλιοθήκη του SteamVR, η PointerInside και η PointerOutside. Στην PointerInside όταν το Laser Pointer που υπάρχει στο δεξί χειριστήριο, ακουμπήσει – δείχνει σε κάποιο αντικείμενο αυτή η συνάρτηση εκτελείται. Στην συγκεκριμένη περίπτωση ελέγχεται αν το αντικείμενο αυτό είναι το πλέγμα επίπεδης επιφάνειας. Αν είναι τότε δημιουργείται ένα προσωρινό αντικείμενο (αναλόγως τι αντικείμενο εχει επιλέξει να τοποθετήσει ο παίκτης), το οποιο εμφανίζεται στο πλέγμα με διαφάνεια, δημιουργώντας μια προεπισκόπηση του αντικειμένου που μπορεί να τοποθετήσει ο παίκτης, πατώντας την σκανδάλη.

Εικόνα 50 - Συνάρτηση της κλάσης CubeGridSnap

```
//VR Laser Pointer Handlers - Pointer Inside
public void PointerInside(object sender, PointerEventArgs e)
{

    if (e.target.tag.Equals("PlaneGridTag") || e.target.tag.Equals("TempCubeTag"))
    {
        Ray raycast = new Ray(laserPointer.pointer.transform.position,
            laserPointer.transform.forward);

        if (Physics.Raycast(raycast, out hitInfo))
        {
            grid = FindObjectOfType<Grid>();
            finalPosition = grid.GetNearestPointOnGrid(hitInfo.point);

            if (UIInputManager.CurrentSelected.Equals(3)) //Red Cube
            {
                redCube.transform.position = finalPosition;
                redCube.GetComponent<Renderer>().enabled = true;
                blueCube.GetComponent<Renderer>().enabled = false;
                spherePrefab.GetComponent<Renderer>().enabled = false;
                CurrentGameObjectTransofrm = redCube.transform;
            }
            else if (UIInputManager.CurrentSelected.Equals(4)) //Blue Cube
            {
                blueCube.transform.position = finalPosition;
                blueCube.GetComponent<Renderer>().enabled = true;
                redCube.GetComponent<Renderer>().enabled = false;
                spherePrefab.GetComponent<Renderer>().enabled = false;
                CurrentGameObjectTransofrm = blueCube.transform;
            }
            else if (UIInputManager.CurrentSelected.Equals(5)) //Sphere Cube
            {
                spherePrefab.transform.position = finalPosition;
                spherePrefab.GetComponent<Renderer>().enabled = true;
                redCube.GetComponent<Renderer>().enabled = false;
                blueCube.GetComponent<Renderer>().enabled = false;
                CurrentGameObjectTransofrm = spherePrefab.transform;
            }
        }
    }
}
```

Αντίστοιχα η PointerOutside εκτελείται οταν το Lase Pointer σταματήσει να εφάπτεται με το αντικείμενο που είχε εκτελέσει προηγουμένως την PointerInside. Έτσι αν το αντικείμενο που ακούμπησε καθώς εξήλθε από ένα προηγούμενο, δεν είναι η επιφάνεια του πλέγματος τότε σταματάει να απεικονίζει – εμφανίζει τα διαφανή αντικείμενα που δημιουργήθηκαν με σκοπό την προεπισκόπηση.

Εικόνα 51 - Συνάρτηση της κλάσης CubeGridSnap

```
//VR Laser Pointer Handlers - Pointer Outside
public void PointerOutside(object sender, PointerEventArgs e)
{
    if (e.target.tag != "PlaneGridTag")
    {
        if (UIInputManager.CurrentSelected.Equals(3)) //Red Cube
        {
            GameObject redCube = GameObject.Find("HoverRedCube");
            redCube.GetComponent<Renderer>()
                .GetComponent<Renderer>().enabled = false;
        }
        else if (UIInputManager.CurrentSelected.Equals(4)) //Blue Cube
        {
            GameObject blueCube = GameObject.Find("HoverBlueCube");
            blueCube.GetComponent<Renderer>()
                .GetComponent<Renderer>().enabled = false;
        }
        else if (UIInputManager.CurrentSelected.Equals(5)) //Sphere Cube
        {
            GameObject spherePrefab = GameObject.Find("HoverSpherePrefab");
            spherePrefab.GetComponent<Renderer>()
                .GetComponent<Renderer>().enabled = false;
        }
    }
}
```

- **Μετακίνηση της επιφάνειας του πλέγματος**

Στο επεξεργαστικό μέρος ο παίκτης εχει την δυνατότητα να μετακινεί την επιφάνεια του πλέγματος, ώστε να μπορεί να επεξεργάζεται τα αντικείμενα πάνω σε αυτή την επιφάνεια, η οποία υποδεικνύει σε ποιο beat του μουσικού κομματιού βρίσκεται σε μια χρονική στιγμή.

Έτσι η παρακάτω συνάρτηση ειναι η Update της κλάσης MovingPlaneObj η οποία είναι υπεύθυνη για τις λειτουργίες που εκτελεί αυτή η επιφάνεια.

Εικόνα 52 - Συνάρτηση Update της κλάσης MovingPlaneObj

```
// Update is called once per frame
void Update()
{
    //Pausing Unpausing with Spacebar
    if (SteamVR_Input.GetStateDown(("GrabGrip"),
        SteamVR_Input_Sources.LeftHand) & isPlaying)
    {
        AudioSource.Pause();
        isPlaying = false;
    }

    else if (SteamVR_Input.GetStateDown(("GrabGrip"),
        SteamVR_Input_Sources.LeftHand) & !isPlaying)
    {
        isPlaying = true;
        AudioSource.UnPause();
    }

 //Moving Plane on grid when Pause or Unpause
 if (isPlaying)
 {
     PlanePosition = Time.deltaTime * velocity;
     spherePosition = new Vector3(
         MovingObjects.transform.position.x,
         MovingObjects.transform.position.y,
         MovingObjects.transform.position.z - PlanePosition);
     MovingObjects.transform.position = spherePosition;
 }
else
{
    //Update Joystick position every frame
    JoystickPositionY = Mathf.RoundToInt(
        joystickPosition.GetAxis(SteamVR_Input_Sources.LeftHand).y * 3);

    //Moving Plane with Scroll
    if (SteamVR_Input.GetStateDown(("ForwardMovePlane"),
        SteamVR_Input_Sources.LeftHand))
    {
        PlanePositionScroll += GridDisplacement;
    }
    else if (SteamVR_Input.GetStateDown(("BackwordMovePlane"),
        SteamVR_Input_Sources.LeftHand))
    {
        PlanePositionScroll -= GridDisplacement;
    }
```

```
    //Check the Plane is between the song limits
    spherePosition = new Vector3(
        MovingObjects.transform.position.x,
        MovingObjects.transform.position.y,
    (float)System.Math.Round(MovingObjects.transform.position.z * 2,
    MidpointRounding.AwayFromZero) / 2 - PlanePositionScroll - JoystickPositionY);

    if (System.Math.Round(spherePosition.z, MidpointRounding.AwayFromZero) <= 0f
        && System.Math.Round(Mathf.Abs(spherePosition.z),
        MidpointRounding.AwayFromZero) <=
        GameManager.GetComponent<GameManager>().TotalBeats)
    {
        MovingObjects.transform.position = spherePosition;
        //Make Audio Time equal to Plane Position
        AudioSource.time = (60 * Mathf.Abs(spherePosition.z)) /
            GameManager.GetComponent<GameManager>().BeatsPerMinute;
    }

    //Reset Scroll Variable
    PlanePositionScroll = 0;
}

//Make Plane Stop when song has ended
if ((int)GameManager.GetComponent<GameManager>()
    .TotalBeats <= Mathf.Abs(spherePosition.z))
{
    AudioSource.Pause();
    //Debug.Log("Audio Finished");
    isPlaying = false;
    spherePosition = new Vector3(
        MovingObjects.transform.position.x,
        MovingObjects.transform.position.y,
        MovingObjects.transform.position.z);
    MovingObjects.transform.position = spherePosition;
}
```

Όλες οι παραπάνω εικόνες δείχνουν την Update συνάρτηση. Αυτή η συνάρτηση εκτελείται κάθε καρέ για να ελέγχει τις κινήσεις του παίκτη μέσα από τα χειριστήρια. Ελέγχεται αν ο παίκτης πάτησε το πλήκτρο GrabGrip που είναι υπεύθυνο για το Play / Pause της μουσικής. Όταν το μουσικό κομμάτι παίζει τότε η επιφάνεια (Plane) τότε γίνεται μετατόπιση του κάθε δευτερόλεπτο με ταχύτητα (Velocity), ανάλογη του αποτελέσματος από τους υπολογισμούς που έγιναν στην συνάρτηση BPMCalculations (γίνεται ανάλυση σε επόμενο κεφάλαιο).

Όταν το μουσικό κομμάτι είναι σε Pause τότε ο παίκτης μπορεί να κανει μια παραπάνω λειτουργία. Με το κουμπί του χειριστηρίου που αντιστοιχεί το «ForwardMovePlane», και το κουμπί που αντιστοιχεί στο «BackwordMovePlane», ο παίκτης μπορεί να μετακινηθεί ένα beat μπροστά η ένα beat προς τα πίσω. Επιπλέον με το Joystick στον Y άξονα του χειριστηρίου μπορεί να μεταβεί πολλά beat την φορά.

Μετά που θα κάνει ο παίκτης την μετακίνηση, ελέγχεται σε ποιο beat εχει σταματήσει την επιφάνεια για να γίνει ο αντίστοιχος υπολογισμός από beat σε δευτερόλεπτα, ώστε όταν πατήσει το Play να ξεκινήσει και το μουσικό κομμάτι από αυτό το δευτερόλεπτο.

Άλλος ένας έλεγχος είναι η να μην μεταβεί η επιφάνεια πέρα από κάποιο beat. Ελέγχεται ουσιαστικά να μην υπερβαίνει τον μέγιστο αριθμό beats που υπαρχουν καθώς και να μην μεταβεί σε beat μικρότερο του μηδενός. Αυτό ελέγχεται και όταν παίζει το μουσικό κομμάτι για να σταματάει η επιφάνεια όταν φτάνει στο τελευταίο beat (άρα και στο τέλος του μουσικού κομματιού).

Με βάση τον σχεδιασμό, η υλοποίηση αυτής της μετακίνησης της επιφάνειας στην σκηνή, γίνεται με τροπο ώστε να μην κουράζει τον παίκτη στην VR εμπειρία του. Δηλαδή ο παίκτης και τα αντικείμενα στο περιβάλλον του και η επιφάνεια του πλέγματος είναι σε σταθερό σημείο (δεν κινούνται). Έτσι αυτό που γίνεται οταν μετακινείται η επιφάνεια μεταξύ των beats, ειναι να μετακινούνται τα αντικείμενα (Standard Objects) και οι γραμμές που υποδεικνύουν στην σκηνή τα beats. Αυτό δίνει την ψευδαίσθηση στον παίκτη ότι μετακινεί την επιφάνεια.

3.3 Φυσική του παιχνιδιού – Physics

Στο επεξεργαστικό μέρος δεν εγινε τροποποίηση της φυσικής του παιχνιδιού, καθώς οι λειτουργίες που χρειαζόντουσαν ηταν βασικές ως προς την φυσική. Η μόνη παρέμβαση είναι ότι τα αντικείμενα που δημιουργεί ο παίκτης έχουν Rigid Body, που σημαίνει ότι χρησιμοποιούνται λειτουργίες φυσικής από την Unity, αλλα τροποποιούνται ώστε να μένουν σταθερά εκει που δημιουργούνται. Η τροποποίηση αυτή είναι να κανουμε kinematic το RigidBody από τα αντικείμενα.

Στο ρυθμικό μέρος του παιχνιδιού υπάρχουν τροποποιήσεις στην φυσική των αντικειμένων (Streaming Assets). Κάθε τέτοιο αντικείμενο εχει την ιδιότητα και κλάση Splittable που είναι από την εξωτερική βιβλιοθήκη MeshSplitter. Στην παρακάτω εικόνα φαίνονται οι τροποποιήσεις που έγιναν στην κλάση Splittable.

Εικόνα 53 - Εντολές προσαρμογής, της φυσικής των αντικειμένων

```
newGOs[0].gameObject.GetComponent<Rigidbody>().isKinematic = false;
newGOs[0].gameObject.GetComponent<Rigidbody>().useGravity = true;
newGOs[1].gameObject.GetComponent<Rigidbody>().isKinematic = false;
newGOs[1].gameObject.GetComponent<Rigidbody>().useGravity = true;

//Adding Thrust Force to GameObjects
newGOs[0].gameObject.GetComponent<Rigidbody>()
    .AddForce(ForceThrust, ForceThrust, 0, ForceMode.Impulse);
newGOs[1].gameObject.GetComponent<Rigidbody>()
    .AddForce(-ForceThrust, ForceThrust, 0, ForceMode.Impulse);
//Adding Rotation to Gameobjects
newGOs[0].gameObject.GetComponent<Rigidbody>()
    .AddTorque(new Vector3(TorqueThrust, 0, 0), ForceMode.Impulse);
newGOs[1].gameObject.GetComponent<Rigidbody>()
    .AddTorque(new Vector3(-TorqueThrust, 0, 0), ForceMode.Impulse);
```

Αυτό που κανει ουσιαστικά η Splittable είναι να χωρίζει ένα αντικείμενο σε δυο μέρη. Έτσι δημιουργούνται δυο νέα αντικείμενα στην θέση του. Προκειμένου να φαίνεται όμορφα αυτή η κοπή εγινε τροποποίηση της φυσικής αυτών των δυο μετέπειτα δημιουργημένων αντικειμένων. Το newGOs[0] είναι το ένα από τα δυο αντικείμενα και το newGOs[1] είναι το άλλο. Έτσι μόλις γίνει η κοπή του αντικειμένου θα απενεργοποιηθεί το kinematic και θα ενεργοποιηθεί το gravity (βαρύτητα), για κάθε ένα αντικείμενο. Αφού γίνει αυτό προστίθεται μια δύναμη στον άξονα x και y του κάθε αντικειμένου, ώστε να πεταχτούν ελαφρώς προς τα πάνω και προς τα πλάγια αντίστοιχα το κάθε ένα. Ταυτόχρονα δημιουργείται και μια ροπή σε κάθε ένα αντικείμενο το ένα δεξιόστροφα και το άλλο αριστερόστροφα ως προς τον x άξονα.

Αυτή η τροποποίηση φυσικής στα αντικείμενα εφαρμόζεται μόνο στο Lightsaber Mode και στο Saber Mode που χρησιμοποιούνται ξίφη και φωτόσπαθα για να καταστρέψουν («κόψουν» στα δυο) τα αντικείμενα (Streaming Assets). Στο Rifle Mode απλά γίνεται καταστροφή των αντικειμένων χωρίς την χρήση φυσικής της Unity.

3.4 Υπολογισμοί για τον συγχρονισμό ενός ρυθμικού κομματιού

Σε ένα ρυθμικό παιχνίδι βασικό για την υλοποίηση είναι ο συνολικός αριθμός των beats που εχει ένα μουσικό κομμάτι. Τα συνολικά beats χρειάζονται για να μπορούν τα αντικείμενα να είναι συγχρονισμένα με τον ρυθμό του μουσικού κομματιού. Για τον υπολογισμό αυτών των beats χρησιμοποιείται μια συνάρτηση. Αυτή η συνάρτηση είναι:

Συνολικός αριθμός Beats = (Beats το λεπτό / 60) x (Τα συνολικά δευτερόλεπτα του τραγουδιού)

Η υλοποίηση φαίνεται στην παρακάτω εικόνα.

Εικόνα 54 - Εντολες υπολογιμού των beats / UIGrid

```
// Use this for initialization
void Start()
{
    GManager = GameObject.Find("GameManager");
    GridDisplacement = GManager.GetComponent<GameManager>().GridDisplacement;

    //Calculation to find beats on a song
    GManager.GetComponent<GameManager>().TotalBeats =
        (GManager.GetComponent<GameManager>().BeatsPerMinute / 60) *
         GManager.GetComponent<GameManager>().SongTimeSeconds;
    GridNumber = GManager.GetComponent<GameManager>().TotalBeats;

    CreateDynamicObjectsGrid();
}
```

Επίσης για να μπορεί ο παίκτης να βλέπει σε ποιο beat βρίσκεται κάθε φορά, εγινε η δημιουργία γραμμών στην σκηνή που υποδεικνύουν τα beats. Η δημιουργία αυτών των γραμμών – αντικειμένων γίνεται με την παρακάτω συνάρτηση η οποία εκτελείται με την έναρξη επεξεργασίας κάποιου μουσικού κομματιού.
Έτσι για κάθε beat δημιουργείται μια γραμμή με απόσταση όσο αυτή που εχουμε ορίσει με το GridDisplacement (σε αυτή την υλοποίηση το GridDisplacement είναι ίσο με 1).

Εικόνα 55 - Εντολες δημιουργίας των beats στην σκηνή / UIGrid

```
private void CreateDynamicObjectsGrid()
{
    for (float z = 0; z < (int)GridNumber + 1; z += GridDisplacement)
    {
        BeatsNumberText.GetComponent<TextMeshPro>().text = z.ToString();
        Instantiate(BeatsNumberText,
            new Vector3(-1.1f, -0.249f, z - 0.5f),
            Quaternion.Euler(90, 0, 0),
            GameObject.Find("CreatedBeatsNumberText").transform);
        Instantiate(BeatsPlaneLine,
            new Vector3(0.25f, -0.25f, z),
            Quaternion.Euler(90, 0, 0),
            GameObject.Find("CreatedBeatsNumberText").transform);
    }
}
```

Εκτός από την δημιουργία των συνολικών beats, επίσης βασικό είναι η ταχύτητα (velocity) με την οποία η επιφάνεια του πλέγματος (plane), πρεπει να μετατοπίζεται, όσο αφορά το επεξεργαστικό μέρος. Στο ρυθμικό μέρος είναι εξίσου σημαντική η ταχύτητα καθώς την χρησιμοποιούν τα αντικείμενα - Streaming Assets για να μετατοπίζονται προς το παίκτη με τον κατάλληλο ρυθμό που εχει το μουσικό κομμάτι. Η ταχύτητα αυτή υπολογίζεται με βάση την παρακάτω συνάρτηση:

Ταχύτητα = (Απόσταση μεταξύ των beats) x (Beats το λεπτό) / 60

Εικόνα 56 - Συνάρτηση υπολογισμού της ταχύτητας του BPM

```
void BPMCalculation()
{
    //Velocity = Distance * BeatsPerMinute / 60;
    distance = 1f; //Distance for each beat in physical world
    velocity = distance *
        GameManager.GetComponent<GameManager>()
        .BeatsPerMinute / 60;
}
```

Στην εικόνα φαίνεται η υλοποίηση αυτής της συνάρτησης που χρησιμοποιείται και στα δυο μέρη του παιχνιδιού.

3.5 Αποθήκευση και Ανάκτηση

Για να υπάρχει η δυνατότητα ο παίκτης να φορτώνει κάποια πίστα της επιλογής του, θα πρεπει να αποθηκευτεί αυτή η πίστα στον υπολογιστή ώστε να μπορεί να ξανα παίξει τη ίδια πίστα ακόμα και αν κλείσει η εφαρμογή - παιχνίδι.

Έτσι εγινε χρήση JSON αρχείων για την αποθήκευση των πληροφοριών της κάθε πίστας (η οποία αντιστοιχεί σε ένα μόνο μουσικό κομμάτι). Αυτές οι πληροφορίες χωρίζονται σε δυο JSON αρχεία. Στο ένα αποθηκεύονται τα σειριακά αντικείμενα (Streaming Assets) και στο άλλο JSON αρχείο αποθηκεύονται οι ιδιότητες του μουσικού κομματιού.

Η ανάκτηση αρχείων που λειτουργεί και στις δυο σκηνές, φορτώνει τρία αρχεία για να χρησιμοποιηθούν στην αντίστοιχη σκηνή. Αυτά είναι, το JSON με την βάση των αντικειμένων, το JSON με τις ιδιότητες του μουσικού κομματιού που αντιστοιχεί και τέλος το ίδιο το αρχείο με το μουσικό κομμάτι.

Στο ρυθμικό μέρος του παιχνιδιού γίνεται μόνο ανάκτηση των πληροφοριών από τα JSON αρχεία. Καθώς σε αυτό το μέρος του παιχνιδιού δεν αποθηκεύονται πληροφορίες. Αντιθέτως στο επεξεργαστικό κομμάτι χρειάζονται και οι δυο λειτουργίες. Και η ανάκτηση και η αποθήκευση των JSON αρχείων, διότι στην επιλογή επεξεργασίας ήδη δημιουργημένης πίστας χρειάζεται πρώτα να φορτωθούν και να εμφανιστούν τα αντικείμενα στην σκηνή και αργότερα να αποθηκευτούν οι αλλαγές των αντικειμένων που έκανε ο παίκτης στην πίστα.

3.5.1 Αποθήκευση και ανάκτηση JSON βάσης

- **Αποθήκευση**

Στην παρακάτω εικόνα φαίνεται η συνάρτηση SaveJSON, της κλάσης JSONOperations, που είναι υπεύθυνη για την αποθήκευση των αντικειμένων (των ιδιοτήτων τους) σε JSON βάση (αρχεία). Τα αντικείμενα που έχουν δημιουργηθεί από τον παίκτη είναι προσωρινά

αποθηκευμένα στην λίστα objects και οι ιδιότητες του μουσικού κομματιού είναι προσωρινά αποθηκευμένες στην μεταβλητή properties της κλάσης AllObjects.
Αργότερα με την βοήθεια την κλάσης JSON Helper μετατρέπονται αυτές οι πληροφορίες σε μια δομημένη συμβολοσειρά και εν τέλη με την βοήθεια της File κλάσης αποθηκεύεται αυτή η συμβολοσειρά σε JSON αρχείο.

Εικόνα 57 - Συνάρτηση της κλάσης JSONOperations

```
//Saving Function
public void SaveJSON()
{
    GameObject gameManager = GameObject.Find("GameManager");
    string currentSongPathName =
        gameManager.GetComponent<GameManager>().CurrentSongNamePath;
    string songPathJson =
        "\\Resources\\" + currentSongPathName + ".json";
    string songPathJsonProperties =
        "\\Resources\\" + currentSongPathName + "Properties.json";
    //Saving Serializable Object to JSON
    JSONSave1 = JsonHelper.ToJson<Objects>(
        GameManager.allObjects.objects.ToArray());
    JSONSave2 = JsonHelper.ToJson<Properties>(
        GameManager.allObjects.properties.ToArray());

    //Save to to JSON File
    File.WriteAllText(System.IO.Directory.GetCurrentDirectory()
        + "\\Assets" + songPathJson, JSONSave1);
    File.WriteAllText(System.IO.Directory.GetCurrentDirectory()
        + "\\Assets" + songPathJsonProperties, JSONSave2);

    PrintAllObjects();
}
```

- **Ανάκτηση**

Για την ανάκτηση των πληροφοριών από τα JSON αρχεία που έχουν δημιουργηθεί, είναι υπεύθυνη η LoadJSON από την κλάση JSONOperations. Στην συνάρτηση αυτή ελέγχεται αν το JSON αρχείο που χρειάζεται να φορτωθεί, υπάρχει. Εφόσον υπάρχει, γίνεται η αντίθετη διαδικασία από αυτή της αποθήκευσης. Δηλαδή, με την χρήση της File φορτώνει το JSON αρχείο και το μετατρέπει σε μια συμβολοσειρά (string), και αποθηκεύεται σε μια μεταβλητή.
Αργότερα, με την βοήθεια της JSON Helper γίνεται μετατροπή από συμβολοσειρά, σε πληροφορίες που μπορούν να αποθηκευτούν στις προσωρινές λίστες – μεταβλητές Objects και Properties, οι οποίες χρησιμοποιούνται από το παιχνίδι.
Στην παρακάτω εικόνα φαίνεται η συνάρτηση LoadJSON που εκτελεί αυτές τις λειτουργίες.

Εικόνα 58 - Συνάρτηση της κλάσης JSONOperations

```
//Loading JSON
public void LoadJSON()
{
    GameObject gameManager = GameObject.Find("GameManager");
    string currentSongPathName =
        gameManager.GetComponent<GameManager>().CurrentSongNamePath;
    string songPathJson =
        "\\Resources\\" + currentSongPathName + ".json";
    string songPathJsonProperties =
        "\\Resources\\" + currentSongPathName + "Properties.json";

    //If JSON file exists Load file
    if (File.Exists(System.IO.Directory.GetCurrentDirectory()
        + "\\Assets" + songPathJson))
    {
        string JsonSavedString1 = File.ReadAllText(
            System.IO.Directory.GetCurrentDirectory()
            + "\\Assets" + songPathJson);

        Objects[] allObjects = JsonHelper
            .FromJson<Objects>(JsonSavedString1);
        GameManager.allObjects.objects =
            new List<Objects>(allObjects);

        string JsonSavedString2 = File.ReadAllText(
            System.IO.Directory.GetCurrentDirectory() +
            "\\Assets" + songPathJsonProperties);

        Properties[] properties = JsonHelper
            .FromJson<Properties>(JsonSavedString2);
        GameManager.allObjects.properties =
            new List<Properties>(properties);
    }
}
```

Στην παρακάτω εικόνα φαίνεται η συνάρτηση JsonHelper που η λειτουργίες της είναι να μετατρέπει μια συμβολοσειρά σε αντικείμενα – μεταβλητές της Unity ώστε να μπορούν να χρησιμοποιηθούν αργότερα από την Unity, και το ανάποδο. Δηλαδή τα αντικείμενα της Unity να μετατρέπονται σε μια δομημένη συμβολοσειρά.

Εικόνα 59 - Κλάση JsonHelper

```
public static class JsonHelper
{

    public static T[] FromJson<T>(string json)
    {
        Wrapper<T> wrapper = UnityEngine.JsonUtility.FromJson<Wrapper<T>>(json);
        return wrapper.Items;
    }

    public static string ToJson<T>(T[] array)
    {
        Wrapper<T> wrapper = new Wrapper<T>();
        wrapper.Items = array;
        return UnityEngine.JsonUtility.ToJson(wrapper);
    }

    [Serializable]
    private class Wrapper<T>
    {
        public T[] Items;
    }
}
```

Ένα παράδειγμα στη αποθήκευση των JSON αρχείων φαίνεται στις παρακάτω εικόνες. Στην μια εικόνα απεικονίζονται οι ιδιότητες – Properties του μουσικού κομματιού.
Ενώ στην άλλη εικόνα απεικονίζονται οι ιδιότητες κάθε ενός αντικειμένου που είχε δημιουργηθεί στην σκηνή του επεξεργαστικού μέρους, από τον παίκτη.

Εικόνα 60 - Δείγμα αποθηκευμένου Json αρχείου / AllObjects

```
"Items": [
    {
        "objectType": 1,
        "xPosition": 2.0,
        "yPosition": 1.0,
        "zTime": 1.0
    },
    {
        "objectType": 2,
        "xPosition": 0.0,
        "yPosition": 0.0,
        "zTime": 1.0
    }
]
```

Εικόνα 61 - Δείγμα αποθηκευμένου Json αρχείου / Properties

```
"Items": [
    {
        "SongTitle": "DemoSong",
        "SongTimeSeconds": 67.57877349853516,
        "BeatsPerMinute": 95.0
    },
    {
        "SongTitle": "Demo Song",
        "SongTimeSeconds": 67.57877349853516,
        "BeatsPerMinute": 95.0
    }
]
```

- **CreateLoadedObjects**

Η συνάρτηση CreateLoadedObjects εχει ως λειτουργία να δημιουργήσει και να εμφανίσει στην σκηνή, όλα τα αντικείμενα που είχαν φορτωθεί στην λίστα Objects από το JSON αρχείο. Αυτό που κανει ουσιαστικά είναι να ελέγξει μια-μια όλες τις εγγραφές της λίστας και αναλόγως το τι τύπου αντικείμενο (objectType) είναι να δημιουργήσει το αντίστοιχο στην σκηνή. Το objectType “0” είναι για το κόκκινο κουτί, το “1” είναι για το μπλε κουτί και το “2” για το σφαιρίδιο.

Εικόνα 62 - Συνάρτηση της κλάσης JSONOperations

```
//Loading all objects from JSON to the game
public void CreateLoadedObjects()
{
    //Getting Prefab and Setting Variables
    foreach (Objects objec in GameManagerMainGame.allObjects.objects)
    {
        //Red Cube Prefab
        if (objec.objectType.Equals(0))
        {
            obj = GameObject.Find("RedCubePrefab");
            //Instantiating Object
            GameObject game = Instantiate(obj, new Vector3(
            objec.xPosition,
            objec.yPosition,
            objec.zTime),
            Quaternion.identity,
            GameObject.Find("CreatedObjects").transform);
        }
```

```
        //Blue Cube Prefab
        else if (objec.objectType.Equals(1))
        {
            obj = GameObject.Find("BlueCubePrefab");
            //Instantiating Object
            GameObject game = Instantiate(obj, new Vector3(
            objec.xPosition,
            objec.yPosition,
            objec.zTime),
            Quaternion.identity,
            GameObject.Find("CreatedObjects").transform);
        }
        //Sphere Prefab
        else if (objec.objectType.Equals(2))
        {
            obj = GameObject.Find("SpherePrefab");
            //Instantiating Object
            GameObject game = Instantiate(obj, new Vector3(
            objec.xPosition,
            objec.yPosition,
            objec.zTime),
            Quaternion.identity,
            GameObject.Find("CreatedObjects").transform);
        }
    }
}
```

3.5.2 Ανάκτηση μουσικού κομματιού

Η ανάκτηση του μουσικού κομματιού χρησιμοποιείται και στα δυο μέρη του παιχνιδιού. Απλά γίνεται έλεγχος κάθε φορά ποια σκηνή είναι την συγκεκριμένη χρονική στιγμή ανοικτή ώστε να μην υπαρχουν προβλήματα στην φόρτωση του μουσικού κομματιού.

Η βασική λειτουργία είναι ότι με την χρήση της Resource.Load της Unity φορτώνεται το μουσικό κομμάτι ως AudioClip μόνο με την χρήση του συνδέσμου της τοποθεσίας του, στον δίσκο του υπολογιστή (Path). Αφού γίνει αυτό ορίζεται στον Audio Manager, του Game Manager της αντίστοιχης σκηνής, το AudioClip με το μουσικό κομμάτι.
Επιπλέον φορτώνονται στις μεταβλητές του Game Manager όλες οι ιδιότητες του μουσικού κομματιού απαραίτητες για το παιχνίδι.

Εικόνα 63 - Εντολές φόρτωσης αρχείων, μέσω της UIFileBrowser

```
GameObject gameManager = GameObject.Find("GameManager");
AudioSource audioSrc = gameManager.GetComponent<AudioSource>() as AudioSource;

//Assigning Audio Clip for GameManager AudioSource
AudioClip audioClip = Resources.Load<AudioClip>(songNamePath);
audioSrc.clip = audioClip;

//Assinging variables to GameManager
gameManager.GetComponent<GameManager>().SongTimeSeconds = audioClip.length;
gameManager.GetComponent<GameManager>().CurrentSongNamePath = songNamePath;

//Assinging variables to GameManager
gameManager.GetComponent<GameManager>().SongTitle =
    GameManager.allObjects.properties[0].SongTitle;

gameManager.GetComponent<GameManager>().BeatsPerMinute =
   GameManager.allObjects.properties[0].BeatsPerMinute;
```

Κεφάλαιο 4 : Ενσωμάτωση VR Τεχνολογίας στο παιχνίδι

4.1 VR Τεχνολογία και περιορισμοί

Οι περισσότεροι προγραμματιστές σήμερα επικεντρώνονται σχεδόν αποκλειστικά σε Unity ή Unreal Engine για εφαρμογές εικονικής πραγματικότητας. Αυτό για μια βιομηχανία που γίνεται πιο ανταγωνιστική μέρα με τη μέρα, είναι αρκετά δύσκολο. Τις περισσότερες φορές, αυτή η έλλειψη ευελιξίας ευθύνεται λογο των προκλήσεων που αντιμετωπίζουν οι εταιρείες παιχνιδιών VR, καθώς υπάρχουν περιορισμοί στην τωρινή VR τεχνολογία.
Αυτοί οι τεχνολογικοί περιορισμοί είναι:

- **Latency - Καθυστέρηση**

Το Latency είναι ουσιαστικά η χρονική καθυστέρηση μεταξύ της εκτέλεσης μιας συγκεκριμένης λειτουργίας. Δηλαδή το χρονικό διάστημα που υπάρχει μεταξύ του ερεθίσματος και της απόκρισης του. Αυτός είναι ο λόγος που οι χρήστες έχουν ναυτία όταν παίζουν ένα VR παιχνίδι. Όσο πιο γρήγορα λειτουργεί μια εφαρμογή και όσο μικρότερη είναι αυτή η χρονική καθυστέρηση (latency), τόσο καλύτερη είναι η ροή της οπτικής πληροφορίας. Είναι καθήκον ενός έμπειρου προγραμματιστή να αναγνωρίζει την αιτία που προκαλούν χρονική καθυστέρηση (latency), και να τα διορθώνει από την εφαρμογή για να παρέχει μια ομαλή εμπειρία στον χρήστη.

- **Hardware integration – Ενσωμάτωση υλικού**

Εκτός από την διαφορετικότητα που υπάρχει σε κάθε VR λογισμικό, όλα τα παιχνίδια και οι εφαρμογές VR εκτελούνται σε διαφορετικές υλοποιήσεις υλικών συσκευών (π.χ. VR Headsets με διαφορετικά πρότυπα από κάθε εταιρία). Ένας προγραμματιστής - developer για να στοχεύσει σε ένα μεγαλύτερο κοινό και μια επιτυχημένη εφαρμογή, πρέπει να διασφαλίσει ότι η εφαρμογή λειτουργεί ομαλά όχι μόνο σε μία, αλλά και σε πολλές και διαφορετικές συσκευές υλικού εικονικής πραγματικότητας.

- **Interfaces – Διεπαφές**

Η διεπαφή, το περιβάλλον, τα γραφικά, τα μενού, ο χειρισμός αντικειμένων και η συνολική αλληλεπίδραση εικονικής πραγματικότητας μέσα σε μια VR εφαρμογή επιτυγχάνεται με διάφορους τρόπους. Έτσι, η προσπάθεια για την επιλογή της σωστής διεπαφής, είναι χρονοβόρα και πολύπλοκη, τόσο ώστε να απαιτεί ένα καλό τεχνικό υπόβαθρο.
(Eisenberg, 2020)

4.2 Καλές πρακτικές για την ανάπτυξη παιχνιδιού σε VR

Προκειμένου το παιχνίδι ή η εφαρμογή να προσφέρει την καλύτερη εμπειρία στον παίκτη χωρίς να του δημιουργεί προβλήματα, πρέπει να εφαρμόζονται κάποιες καλές πρακτικές ως προς την υλοποίηση. Αναλόγως την υλοποίηση, κάποιες καλές πρακτικές είναι πιο αναγκαίες σε σχετικές με κάποιες άλλες. Έτσι στην ανάπτυξη ενός παιχνιδιού, πρεπει να γίνεται εκτίμηση, ποιες βασικές καλές πρακτικές πρεπει να εφαρμοστούν.

Μερικές καλές πρακτικές για την ανάπτυξη ενός παιχνιδιού σε VR τεχνολογία είναι:

Σχεδιασμός Διεπαφής

Κάθε διαδραστικό αντικείμενο πρέπει να ανταποκρίνεται σε οποιαδήποτε περιστασιακή κίνηση. Για παράδειγμα, σε ένα κουμπί, κάθε αλληλεπίδραση θα πρέπει να προκαλεί κίνηση, ακόμη και αν αυτή η κίνηση δεν έχει ως αποτέλεσμα το πλήκτρο να πιεστεί πλήρως. Όταν συμβαίνει αυτό, η απόκριση του αντικειμένου παρακινεί τον παίκτη να αλληλεπιδράσει με αυτό.

- **Περιγραφές χειρονομίας (Gestures)**

Τα μνήματα υπόδειξης, βασισμένα σε κείμενο ή ήχο μπορούν επίσης να είναι απαραίτητα για τους χρήστες την πρώτη φορά. Πρεπει να περιγράφονται με σαφήνεια οι προβλεπόμενες αλληλεπιδράσεις, καθώς αυτό θα επηρεάσει σημαντικά τον τρόπο με τον οποίο ο χρήστης αλληλεπίδρα.

- **Ευκρίνεια κειμένου και εικόνας**

Ενώ η VR τεχνολογία είναι καταπληκτική σε πολλά πράγματα, μερικές φορές υπάρχουν προβλήματα που σχετίζονται με την απεικόνιση κειμένου. Λόγω περιορισμών ανάλυσης, μόνο το κείμενο στο κέντρο του FOV μπορεί να φαίνεται καθαρό. Το κείμενο κατά μήκος της περιφέρειας μπορεί να φαίνεται θολό, εκτός εάν οι χρήστες στρέψουν να το δουν απευθείας. Έτσι είναι καλύτερη πρακτική να μην γίνεται χρήση κείμενων με πολλές γραμμές.

(LeapMotion, 2015)

- **Εστίαση στην κίνηση του παίκτη**

Σημαντική είναι η χρήση ομαλών κινήσεων. Η ομαλή κίνηση συνήθως συνδυάζεται είτε με ομαλή περιστροφή είτε με απότομη περιστροφή, ώστε να μπορείτε να γυρίσετε ολόκληρο το σώμα του χαρακτήρα σας σε προς κάποια άλλη κατεύθυνση. Για μερικούς ανθρώπους, η ομαλή κίνηση μπορεί να τους προκαλέσει ναυτία, καθώς τα μάτια σας βλέπουν μια κίνηση που το σώμα σας δεν ακολουθεί στον φυσικό κόσμο. Η άλλη επιλογή κίνησης σε ένα παιχνίδι VR, είναι η τηλεμεταφορά. Όπου ο παίκτης στοχεύει μέσα στο παιχνίδι, το σημείο που θέλει να μεταφερθεί.

- **Σταθερότητα στις μετρήσεις και στην κλίμακα του παιχνιδιού**

Η διασφάλιση της σωστής κλίμακας του παιχνιδιού, είναι από τα πιο σημαντικά πραγματα ώστε να εξασφαλιστεί στον παίκτη η καλύτερη δυνατή εμπειρία στο VR. Η λανθασμένη κλίμακα μπορεί να οδηγήσει σε διάφορα προβλήματα αισθητικής αντίληψης για τους χρήστες. Τα αντικείμενα φαίνονται καλυτερα στον εικονικό κόσμο, όταν βρίσκονται σε εύρος 0,75 έως 3,5 μέτρα από την κάμερα της συσκευής αναπαραγωγής (VR Headset).

(UnrealEngine, 2020)

- **Χρήση ήχου τοπικής θέσεως - Positional Audio**

Για την καλύτερη εμπειρία του παίκτη, σημαντικό είναι να γίνεται η χρήση ήχου τοπικής θέσεως. Έχοντας κάθε ήχο που προέρχεται από μια πραγματική πηγή εχει μεγάλη σημασία στην εικονική πραγματικότητα, για να βελτιωθεί η εμβύθιση (Immersion) του παίκτη.

(LeapMotion, 2015)

- **Βελτιστοποίηση ρυθμού καρέ (Frame Rate)**

Πρέπει να υπάρχει σταθερότητα στον ρυθμό αλλαγής των εικόνων - καρέ (frame rate) και, ιδανικά, να υπάρχει λίγο buffer έτσι ώστε το frame rate του παιχνιδιού να ειναι πάντα πάνω από το framerate που παρέχει το HMD-VR Headset. Ο χαμηλός ρυθμός εναλλαγής καρέ, είναι ένας από τους λογούς που μπορούν να προκαλέσουν ναυτία στον χρήστη.

(UnrealEngine, 2020)

4.3 Χρήση του SteamVR

Στο συγκεκριμένο μουσικό παιχνίδι, η ενσωμάτωση της VR τεχνολογίας εγινε μέσω της χρήσης της εργαλειοθήκης της Valve, το SteamVR. Αυτή η εργαλειοθήκη είναι συμβατή με την Unity καθώς και με άλλες μηχανές ανάπτυξης παιχνιδιών. Το SteamVR προσφέρει πολλές λειτουργίες και εργαλεία, έτσι ώστε να μπορεί να αναπτυχθεί οποιαδήποτε εφαρμογή στην εικονική πραγματικότητα. Επίσης, υποστηρίζει όλων των ειδών τις συσκευές εικονικής πραγματικότητας, έτσι ώστε να μην υπαρχουν θέματα ασυμβατότητας.

4.3.1 Ιδιότητες του SteamVR για το VR Headset HMD

Όπως αναφέρθηκε στον σχεδιασμό του παιχνιδιού, η θέση του παίκτη είναι διαφορετική σε κάθε σκηνή. Το VR Headset αντιπροσωπεύει το κεφάλι του παίκτη μέσα στο παιχνίδι. Έτσι ανάλογα που θα τοποθετηθεί ο παίκτης (άρα και η κάμερα), μέσα στο παιχνίδι, όταν γίνει εκκίνηση του παιχνιδιού, θα ξεκινήσει να βλέπει από αυτό το σημείο.

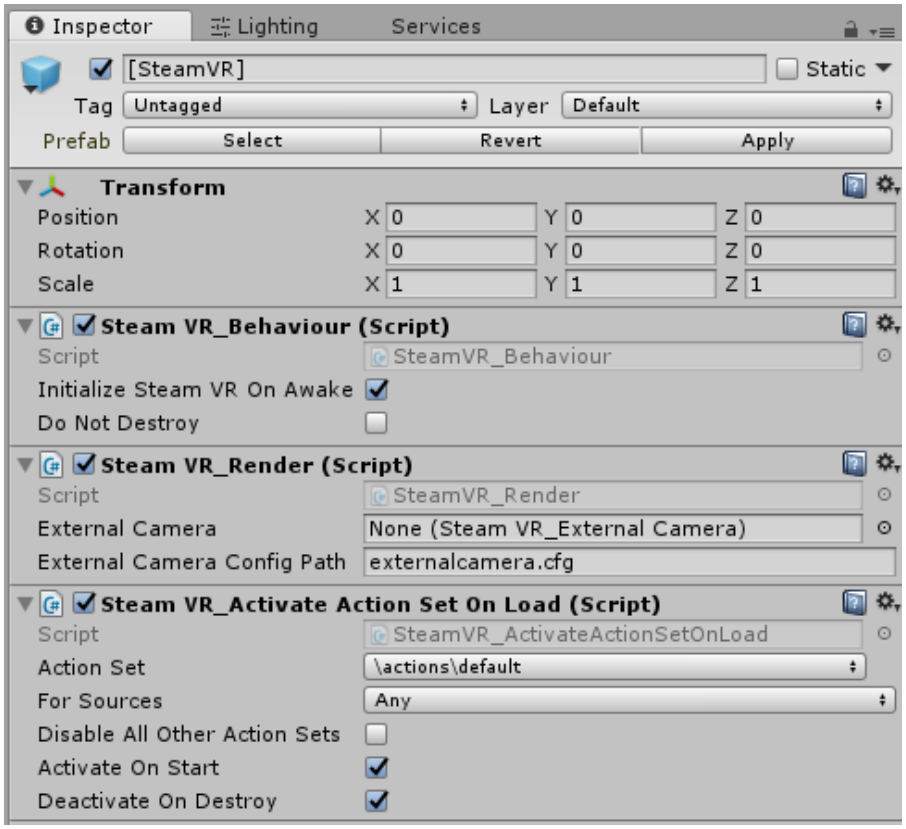
Παρόλα αυτά δεν είναι απαραίτητο ότι θα μείνει σε αυτό το σημείο καθώς επιτρέπεται στον παίκτη μέσω των VR προδιαγραφών να ορίσει ο ίδιος τα όρια στον φυσικό χώρο, και να κινηθεί με το HMD μέσα σε αυτά. Αυτή η μετακινεί του παίκτη στον φυσικό χώρο, μπορεί να μεταφραστεί – αντιστοιχιστεί στον εικονικό χώρο, με την χρήση των αισθητήρων του Headset.

Επιπλέον το ίδιο ισχύει και για το ύψος του παίκτη μέσα στο παιχνίδι. Αναλόγως το ύψος που εχει ο παίκτης στον φυσικό χώρο, το ίδιο ανάλογο ύψος θα εχει και στον εικονικό. Για την αντιστοίχιση φυσικού σε εικονικού χώρου, είναι υπεύθυνη η εργαλειοθήκη SteamVR που την κανει αυτόματα, χωρίς να χρειαστεί αναγκαστικά παρέμβαση από τον προγραμματιστή.

- **SteamVR – Ελεγκτής για όλες τις VR συμπεριφορές**

Το αντικείμενο [SteamVR] συμπεριλαμβάνεται στην βιβλιοθήκη της επέκτασης SteamVR, και είναι υπεύθυνη για όλες τις λειτουργίες των VR συμπεριφορών μέσω των Script που χρησιμοποιεί.

Εικόνα 64 - Ιδιότητες του SteamVR στο Inspector



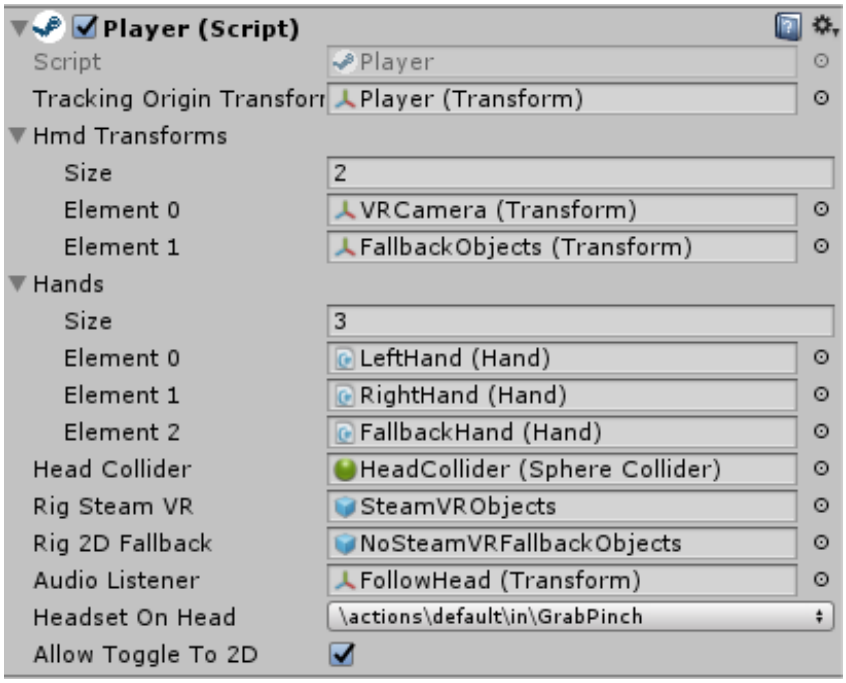
Όπως φαίνεται στην παραπάνω εικόνα το SteamVR Behavior ελέγχει τα βασικά Events (αλλαγές) που δημιουργούν τα VR αντικείμενα (π.χ. κίνηση του headset).

Το VR_Renderer είναι υπεύθυνο για την σωστή απεικόνιση από την κάμερα του παιχνιδιού, στους φακούς κάθε ματιού, του VR Headset ώστε να δημιουργείται η ψευδαίσθηση του βάθους.

Το SteamVR_ActivateActionSetOnLoad ειναι υπεύθυνο για να φορτώνει από το SteamVR Input και να ενεργοποιεί / απενεργοποιεί τα action set που εχει επιλέξει ο χρήστης. Αυτά τα action set τα προσαρμόζει ο χρήστης.

- **Player**

Εικόνα 65 - Ιδιότητες του Player στο Inspector



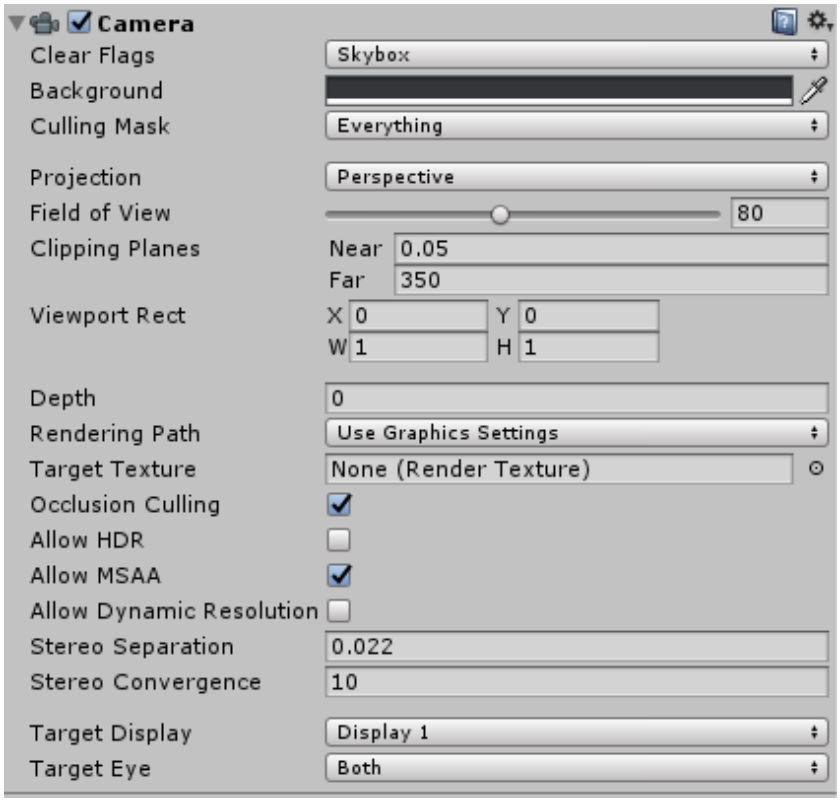
Το αντικείμενο Player της SteamVR, είναι ουσιαστικά όλες οι ιδιότητες του παίκτη όσον αφορά τα VR χαρακτηριστικά του. Από το κεφάλι με την κάμερα μεχρι και τα χέρια με την χρήση των χειριστηρίων. Το player είναι σημαντικό για να μπορούν να λειτουργήσουν τα VR μέρη σαν μια ενιαία οντότητα μέσα στο παιχνίδι, που αντιπροσωπεύει τον παίκτη.

Έτσι στην παραπάνω εικόνα φαίνονται οι αντιστοιχίες των αντικειμένων της σκηνής στο Script του player. Τα βασικά αντικείμενα που χρησιμοποιεί είναι η VR Camera. Τα script που είναι υπεύθυνα για τις λειτουργίες των χεριών. Το headCollider που είναι η περιοχή γύρω από την VR Camera και ελέγχει με ποια αντικείμενα ήρθε σε επαφή. Τα SteamVRObjects που περιέχουν τα αντικείμενα των χεριών και των VR χειριστηρίων. Καθώς και το Audio Listener που είναι υπεύθυνο για τον ήχο και την θέση αυτού, που δέχεται ο παίκτης από το παιχνίδι. Αυτό το AudioListener ακολουθεί μέσω του αντικειμένου FollowHead, το VR Headset, όπου και να μετακινηθεί.

- **VR Camera**

Η VR Camera είναι ουσιαστικά ένα απλό αντικείμενο Camera της Unity, το οποιο το χρησιμοποιεί η SteamVR μέσω του script VR Render, για να φαίνεται σωστά στο Headset. Αυτή η κάμερα ακολουθεί συνεχόμενα το VR Headset. Γενικά γίνεται προσαρμογή των ιδιοτήτων της κάμερας σαν να ηταν μια απλή κάμερα στην Unity. Όπως φαίνεται στην παρακάτω εικόνα, οι σημαντικές ιδιότητες που προσαρμοστήκαν στις σκηνές του παιχνιδιού είναι: ο τύπος απεικόνισης (Projection), το οπτικό πεδίο (Field Of View), το Occlusion Culling που είναι υπεύθυνο για να μην κανει render αντικείμενα που η κάμερα δεν «βλέπει», και το Clipping Planes που είναι υπεύθυνο για τα όρια των αποστάσεων που θα γίνονται render τα αντικείμενα.

Εικόνα 66 - Ιδιότητες της VR Camera στο Inspector



4.3.2 Ιδιότητες και λειτουργίες των VR Χειριστηρίων

- **Επεξεργαστικό μέρος του παιχνιδιού**

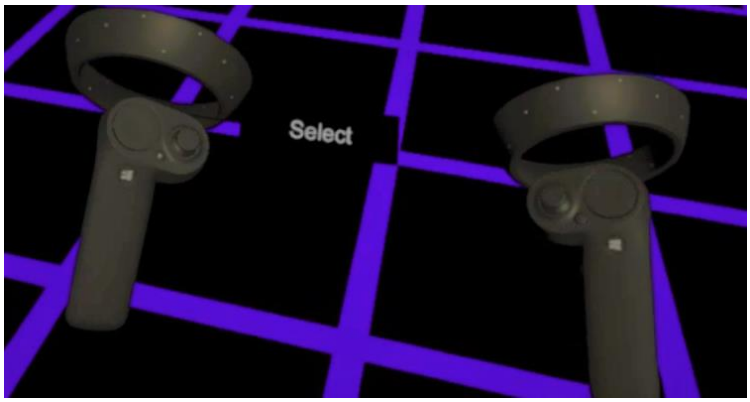
Εικόνα 67 - Αντικείμενο αριστερού χεριού στην ιεραρχία της σκηνής

Εικόνα 68 - Αντικείμενο δεξιού χεριού στην ιεραρχία της σκηνής

Στις παραπάνω εικόνες βλέπουμε στην ιεραρχία της σκηνής του επεξεργαστικού μέρους του παιχνιδιού, τα αντικείμενα που υπάρχουν σαν «παιδιά» κάτω από κάθε χέρι - αντικείμενο. Αυτή η δομή εγινε με αυτόν τον τροπο έτσι ώστε τα αντικείμενα κάτω από τα χέρια να ακολουθούν με τις ίδιες συντεταγμένες, την ίδια πορεία που θα κάνουν τα χέρια μέσω των χειριστηρίων του παίκτη.

Η βιβλιοθήκη SteamVR εχει κάποια έτοιμα αντικείμενα προς χρήση (prefab), όπως αυτό των χεριών, που έχουν έτοιμες λειτουργίες. Σε αυτή την υλοποίηση παρόλα αυτά δεν χρειάστηκε κάποια από αυτές τις λειτουργίες για γίνει αυτή η χρήση χεριών. Έτσι εγινε απλά η χρήση του Controller Model μαζι με τα Action Set από το Input του SteamVR. Το Controller Model χρησιμοποιεί το Script SteamVR_RenderModel το οποίο κάνει Render - απεικόνιση στην σκηνή τα αντίστοιχα χειριστήρια, με αυτά που εχει ο παίκτης στον φυσικό κόσμο. Έτσι δημιουργείται η ψευδαίσθηση στον παίκτη ότι κρατάει τα πραγματικά - φυσικά χειριστήρια στο παιχνίδι ή γενικά στον εικονικό κόσμο.

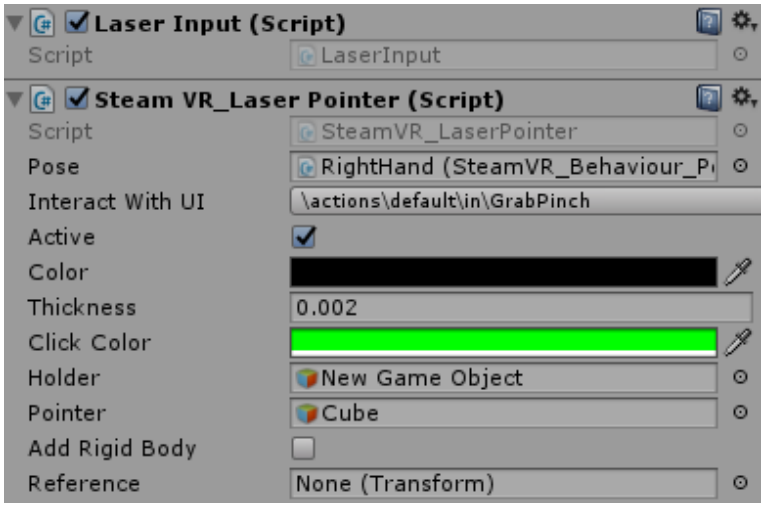
Εικόνα 69 - Μοντέλα χειριστηρίων στον εικονικό κόσμο



Έτσι στο αριστερό χέρι υπάρχει όλη η διεπαφή με τις επιλογές δημιουργίας - επεξεργασίας καθώς και οι υποδείξεις των κουμπιών των χειριστηρίων.

Στο δεξί χέρι υπαρχουν μόνο οι υποδείξεις κουμπιών και ο δείκτης Laser (Laser Pointer). Ο Laser Pointer είναι ένας δείκτης για να κανει ο παίκτης επιλογές μόνο πάνω στην επιφάνεια του πλέγματος (Grid). Αυτός ο δείκτης είναι μαύρος, και όταν ο παίκτης πατήσει την σκανδάλη (που εχει οριστεί ως GrabPinch), γίνεται πράσινος. Η παρακάτω εικόνα δείχνει αυτές τις ιδιότητες του Laser Pointer , που υπάρχουν στο αντικείμενο του δεξιού χεριού.

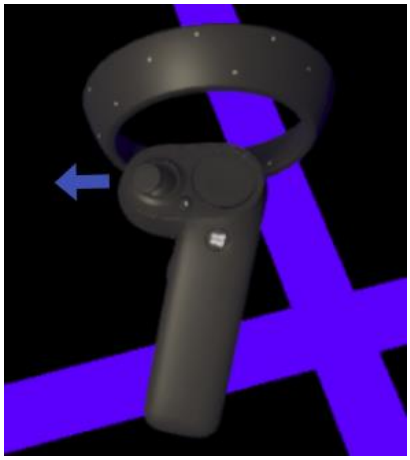
Εικόνα 70 - Ιδιότητες του δεξιού χειριστηρίου στο Inspector



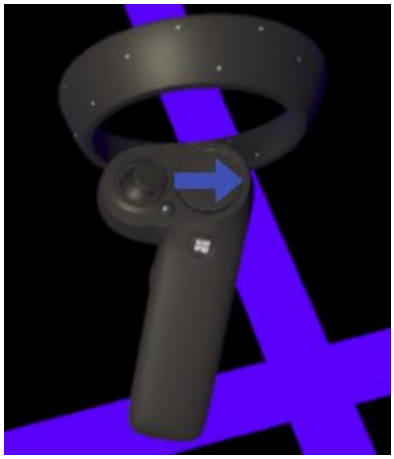
- **SnapTurn**

Το SteamVR περιέχει διάφορα εργαλεία και βιβλιοθήκες, ένα από αυτά είναι το εργαλείο Snap Turn (απότομο γύρισμα). Η λειτουργία του είναι να γυρνάει απότομα την κάμερα του παίκτη κατά κάποιες μοίρες, την φορά. Η κλάση – Script που φαίνεται παρακάτω είναι προδημιουργημένη απο την επέκταση SteamVR, ώστε να μπορεί να υλοποιηθεί αυτή η ιδιότητα εύκολα σε ένα παιχνίδι ή μια εφαρμογή. Αυτή η λειτουργία χρησιμοποιήθηκε μόνο στο επεξεργαστικό μέρος του παιχνιδιού καθώς σε πολλές περιπτώσεις ο παίκτης χρειάζεται να γυρίσει ολόκληρο το σώμα του προκειμένου να εχει στο οπτικό του πεδίο, την επιφάνεια με το πλέγμα που τοποθετούνται τα Streaming Assets. Στην εικόνα αυτή φαίνονται οι ιδιότητες που μπορούν να προσαρμοστούν. Το Script αυτό εχει κάποιες προεπιλεγμένες τιμές. Οι αλλαγές που έγιναν σε αυτές είναι η γωνιά στριψίματος σε 45μοιρες. Η εφαρμογή των action, ώστε να εκτελείται το action «SnapLeft» με την μετακίνηση του μοχλού (Thumbstick - Χειριστηρίου) προς τα αριστερά και το «SnapRight» με την μετακίνηση του μοχλού προς τα δεξιά. Τα SnapLeft και SnapRight actions είναι υπεύθυνα για αυτή την γωνιακή μετακίνηση της κάμερας. Αυτή η λειτουργία χρησιμοποιείται μόνο στο δεξί χειριστήριο και μόνο για το επεξεργαστικό μέρος του παιχνιδιού.

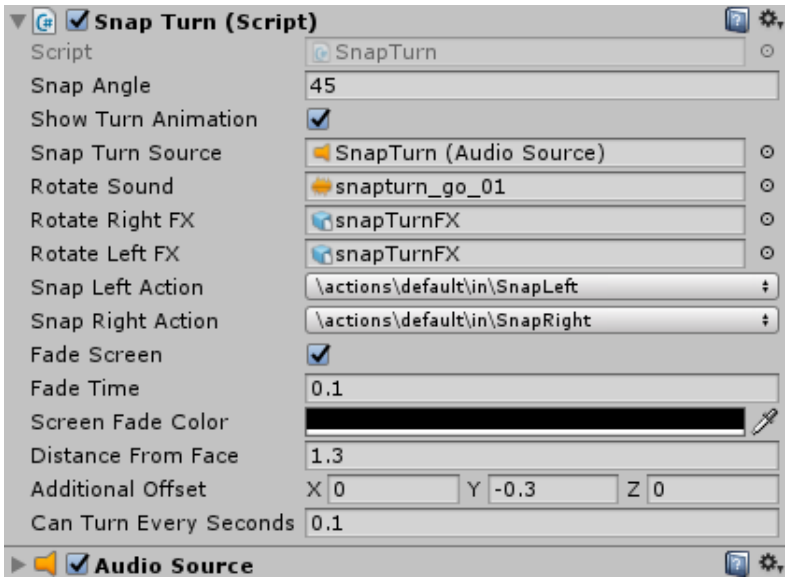
Εικόνα 71 - Υπόδειξη SnapTurn στο δεξί χειριστήριο



Εικόνα 72 - Υπόδειξη SnapTurn στο αριστερό χειριστήριο



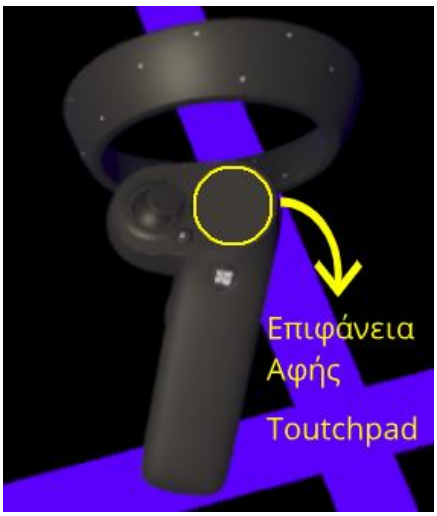
Εικόνα 73 - Ιδιότητες του SnapTurn στο Inspector



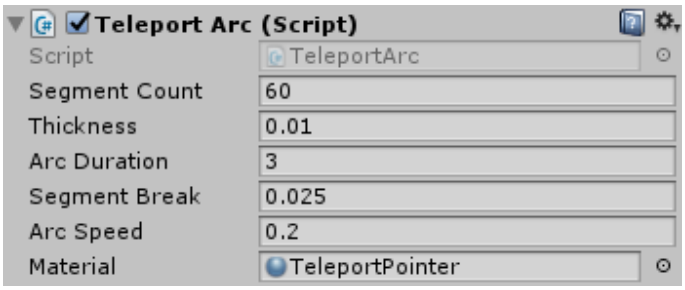
- **Teleport**

Το Teleport είναι άλλη μια έτοιμη υλοποίηση που συμπεριλαμβάνεται στην επέκταση SteamVR της οποίας η λειτουργία είναι να μεταφέρει τον παίκτη από την θέση που βρίσκεται στην θέση την οποία εχει διαλέξει - υποδείξει με το χειριστήριο του. Ο τρόπος που εχει οριστεί, ώστε ο παίκτης να μπορεί να τηλεμεταφερθεί, γίνεται με την χρήση των κουμπιών αφής του χειριστηρίου. Αυτή η λειτουργία χρησιμοποιείται επίσης αποκλειστικά στο δεξί χειριστήριο και μόνο στο επεξεργαστικό μέρος του παιχνιδιού.

Εικόνα 74 - Υπόδειξη της επιφάνειας αφής στο δεξί χειριστήριο

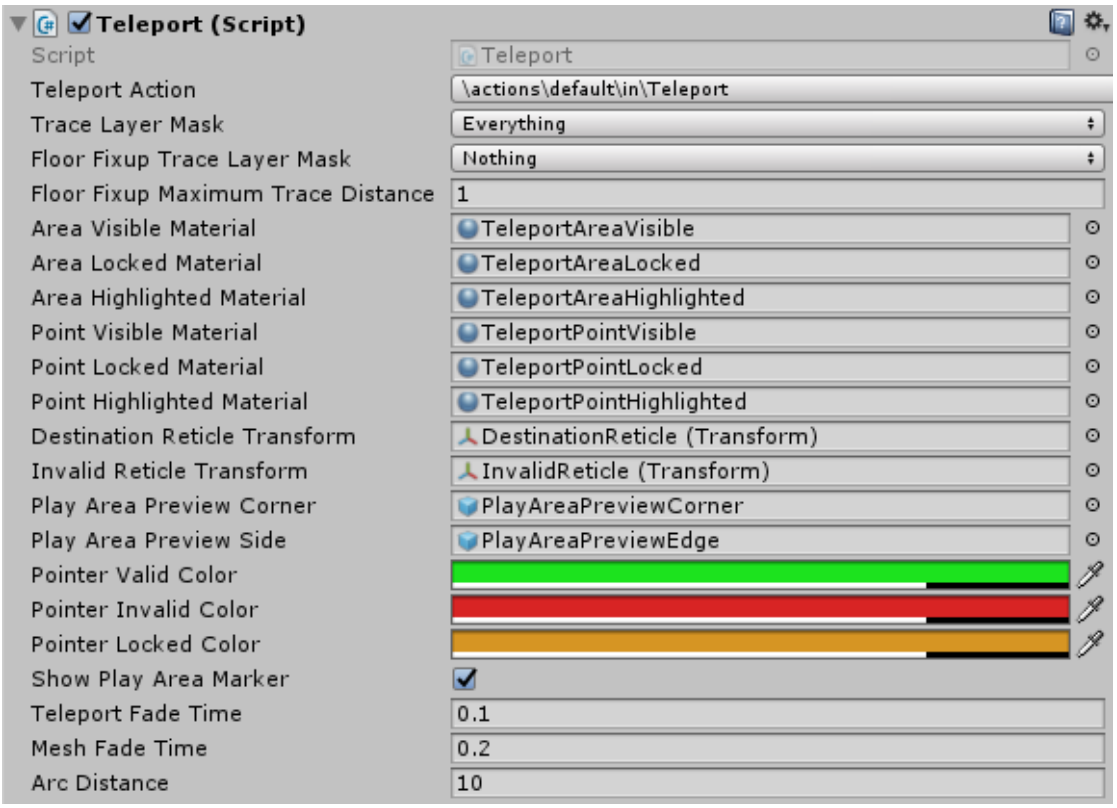


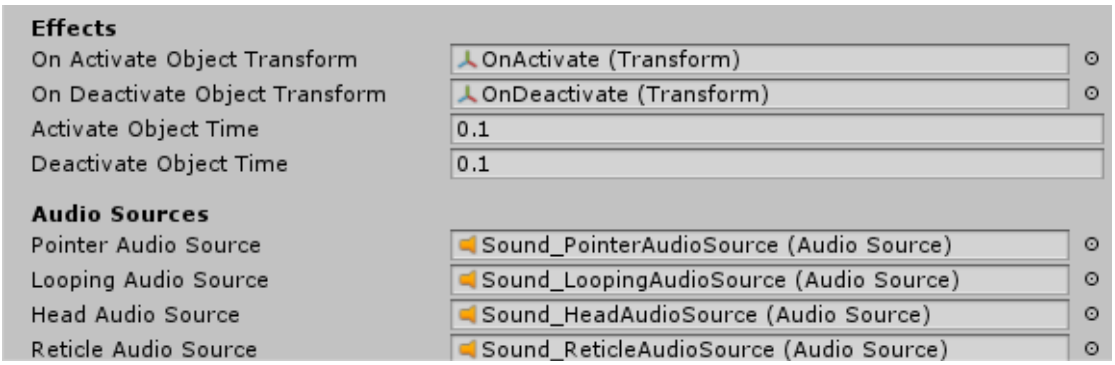
Εικόνα 75 - Ιδιότητες του Teleport Arc στο Inspector



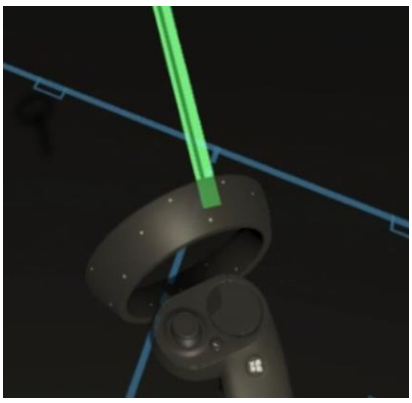
Στο TeleportArc φαίνονται οι προεπιλεγμένες ιδιότητες της γραφικής καμπύλης που βλέπει ο παίκτης για να ορίσει σημείο μετακίνησης. Και σε αυτή την περίπτωση, όπως το Snap Turn, οι ιδιότητες είναι προεπιλεγμένες. Έτσι η μοναδική αλλαγή που εγινε στο Script Teleport, είναι ο ορισμός του action σε Teleport action.

Εικόνα 76 - Ιδιότητες του Teleport στο Inspector

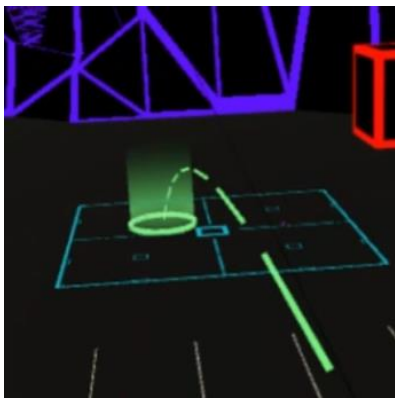




Εικόνα 77 - Υπόδειξη του Teleport στο χειριστήριο



Εικόνα 78 - Υπόδειξη επιτρεπόμενης μετακίνησης



Εικόνα 79 - Υπόδειξη μη επιτρεπόμενης μετακίνησης

Για να μπορεί ο παίκτης να μετακινηθεί με την χρήση της λειτουργίας Teleport, απαραίτητος είναι και ο ορισμός κάποιας επιφάνειας στην οποία ο παίκτης επιθυμεί να μετακινηθεί. Στο επεξεργαστικό μέρος έχουν τοποθετηθεί τέσσερις επιφάνειες (δυο σε κάθε πλευρά της επιφάνειας),στις οποίες εχει προστεθεί η λειτουργία-Script Teleport Area, έτσι ώστε το Teleport Script να αναγνωρίζει σε ποια σημεία και επιφάνειες, επιτρέπεται, να κινηθεί ο παίκτης, και σε ποια όχι.

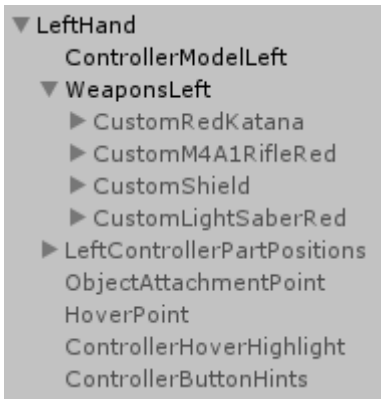
Εικόνα 80 - Ιδιότητες του Teleport Area στο Inspector

Οι επιφάνειες - αντικείμενα που χρησιμοποιούν το Teleport Area, μπορούν να τηλεμεταφερθούν. Στις υπόλοιπες επιφάνειες που δεν χρησιμοποιείται αυτό το Script, δεν επιτρέπει το Teleport Area να γίνει κάποια μετακίνηση. Η υπόδειξη που επιτρέπει την μετακίνηση γίνεται με την εφαρμογή πράσινου χρώματος πάνω στην καμπύλη τηλεμεταφορά (Teleport Arc), ενώ η υπόδειξη μη επιτρεπόμενης μετακίνησης γίνεται με την εφαρμογή κόκκινου χρώματος πάνω στην αυτή την καμπύλη.

Εικόνα 81 - Πάνοψη κάμερας στο επεξεργαστικό μέρος

- **Ρυθμικο μερος**

Εικόνα 82 - Αντικείμενο αριστερού χεριού στην ιεραρχία της σκηνής



Εικόνα 83 - Αντικείμενο δεξιού χεριού στην ιεραρχία της σκηνής

Στην σκηνή του ρυθμικού μέρους και στα δυο χέρια - αντικείμενα, τα «παιδιά» τους στην ιεραρχία της σκηνής είναι τα όλα όπλα που μπορεί να χρησιμοποιήσει ο παίκτης απο όλα τα modes. Επιπλέον υπάρχουν και οι Controller Model που κάνουν render τα χειριστήρια μέσα στο παιχνίδι. Καθώς και οι υποδείξεις (hints) για τα κουμπιά των χειριστηρίων.

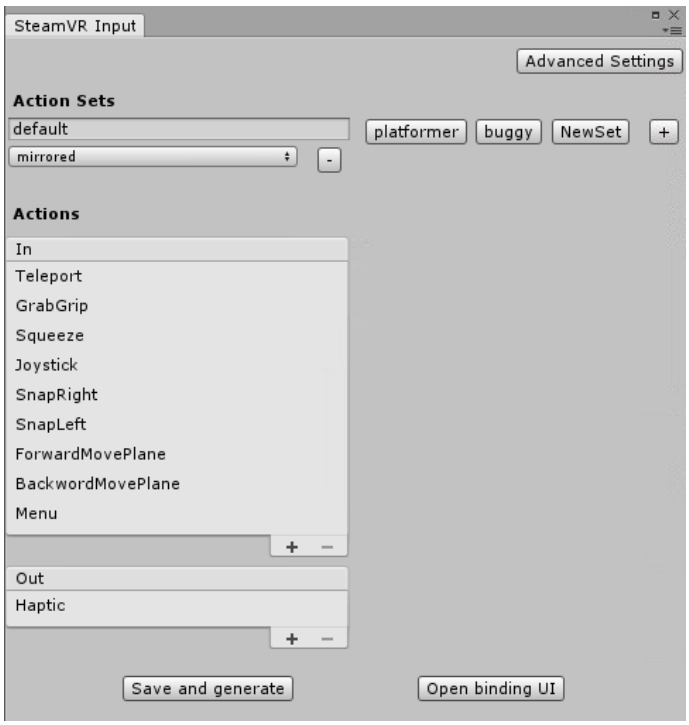
Έτσι αναλόγως το mode και την τις επιλογές του παίκτη, γίνεται εναλλαγή των όπλων και τον αντικειμένων που θα απεικονίζονται την εκάστοτε στιγμή στο παιχνίδι. Εχει δομηθεί έτσι ώστε να μπορεί να απεικονιστεί μόνο ένα παιδί - αντικείμενο (π.χ. όπλο) σε κάθε χέρι την φορά.

4.3.3 Steam VR Input

Το SteamVR Input λειτουργεί ως διαμεσολαβητής μεταξύ των πραγματικών κουμπιών και των event που κατανόει η Unity Engine. Έτσι πρεπει να οριστούν κάποιες λειτουργίες (ή αλλιώς Actions) και ο τρόπος με τον οποίο δουλεύουν, αναλόγως την λειτουργία που εκτελεί το κάθε Action. Παρακάτω φαίνεται στην εικόνα το Action Set με όνομα Default, και τα ονόματα που ορίστηκαν για τα Actions. Αυτά τα actions χρησιμοποιούνται για διάφορες λειτουργίες (διαφορά Script) του παιχνιδιού που συσχετίζονται με αυτά τα Event των χειριστηρίων.

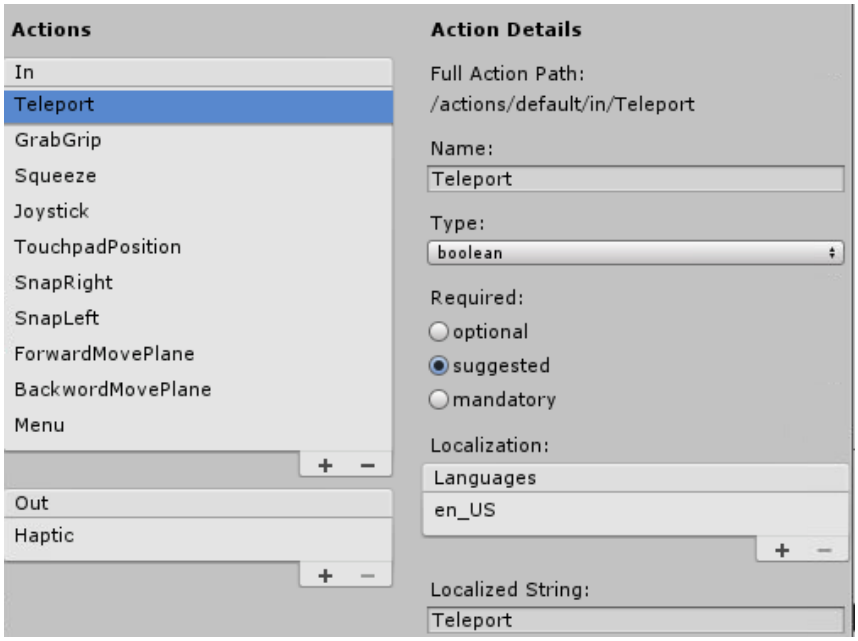
Action Set – Όλα τα Actions

Εικόνα 84 - Παράθυρο SteamVR Input



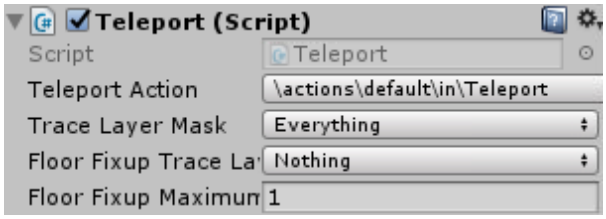
- **Teleport**

Εικόνα 85 - Teleport action και οι ιδιότητες του



Το Teleport Action ορίζεται ως τύπος Boolean, καθώς ενεργοποιείται όταν πατιέται το κουμπί που υπάρχει στην επιφάνεια αφής του δεξιού χειριστηρίου.

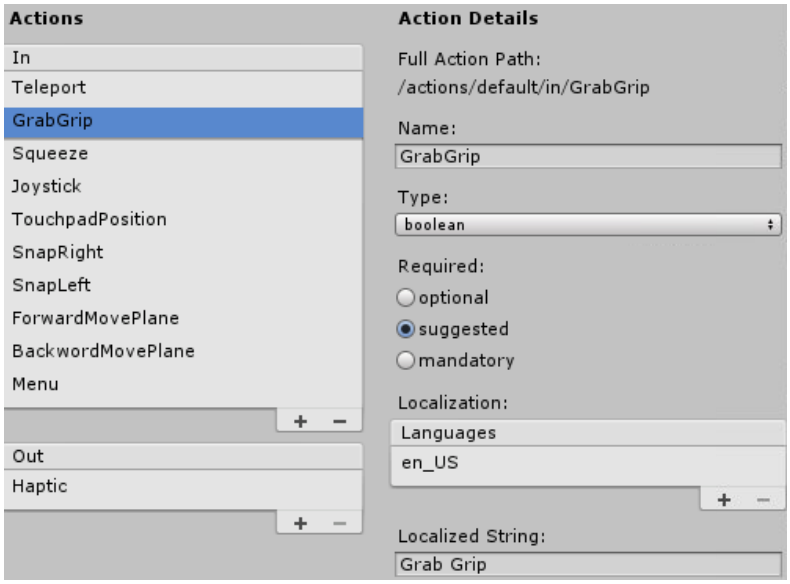
Εικόνα 86 - Ιδιότητες του Teleport Script



Στην παραπάνω εικόνα φαίνεται ότι εχει οριστεί το Teleport ως Action για το Teleport Script. Ουσιαστικά γίνεται αντιστοίχιση του Action από το Action Set του SteamVR Input που πρεπει να αναφέρεται για να μπορεί να το χρησιμοποιήσει το Teleport Script.

- **GrabGrip**

Εικόνα 87 - GrabGrip action και οι ιδιότητες του



Το GrabGrip Action ειναι υπεύθυνο για το Play και το Pause στο επεξεργαστικό μέρος του παιχνιδιού. Και αυτό το action είναι τύπου Boolean. Αυτό το action το χρησιμοποιεί η κλάση MovingPlaneObj και κανει την εναλλαγή μεταξύ Play και Pause.

Εικόνα 88 - Κλάση MovingPlaneObj στο επεξεργαστικό μέρος

```
// Update is called once per frame
void Update()
{
    //Pausing Unpausing with Spacebar
    if (SteamVR_Input.GetStateDown(("GrabGrip"),
        SteamVR_Input_Sources.LeftHand) & isPlaying)
    {
        AudioSource.Pause();
        isPlaying = false;
    }
    else if (SteamVR_Input.GetStateDown(("GrabGrip"),
        SteamVR_Input_Sources.LeftHand) & !isPlaying)
    {
        isPlaying = true;
        AudioSource.UnPause();
    }
```

Επίσης χρησιμοποιείται και στο ρυθμικό μέρος από την κλάση LightSaberMode και είναι υπεύθυνη για την εναλλαγή μεταξύ όπλων. Όταν πατιέται το κουμπί που είναι που ορισμένο το GrabGrip action εμφανίζεται η ασπίδα ενώ όταν δεν πατιέται συνεχίζει να εμφανίζεται το LightSaber. Αυτό γίνεται αντίστοιχα και στα δυο χειριστήρια όπως φαίνεται στην παρακάτω εικόνα.

Εικόνα 89 - Συνάρτηση Update της κλάσης LightSaberMode

```
private void Update()
{
    //Left Hand Shield on Grip Button
    if (SteamVR_Input.GetStateDown(("GrabGrip"),
        SteamVR_Input_Sources.LeftHand))
    {
        LeftLightSaber.SetActive(false);
        LeftShield.SetActive(true);
    }
    else if (SteamVR_Input.GetStateUp(("GrabGrip"),
        SteamVR_Input_Sources.LeftHand))
    {
        LeftShield.SetActive(false);
        LeftLightSaber.SetActive(true);
    }

//Right Hand Shield on Grip Button
if (SteamVR_Input.GetStateDown(("GrabGrip"),
    SteamVR_Input_Sources.RightHand))
{
    RightLightSaber.SetActive(false);
    RightShield.SetActive(true);
}
else if (SteamVR_Input.GetStateUp(("GrabGrip"),
    SteamVR_Input_Sources.RightHand))
{
    RightShield.SetActive(false);
    RightLightSaber.SetActive(true);
}
```

Στην βιβλιοθήκη της SteamVR υπάρχει η κλάση SteamVR_Input που περιέχει έτοιμες συναρτήσεις προς χρήση από τον προγραμματιστή. Αυτό χρησιμεύει στην διασύνδεση των μεταξύ γεγονότων από τα χειριστήρια και των εντολών της Unity. Μερικές από τις βασικές συναρτήσεις που χρησιμοποιεί αυτή η κλάση είναι οι παρακάτω:

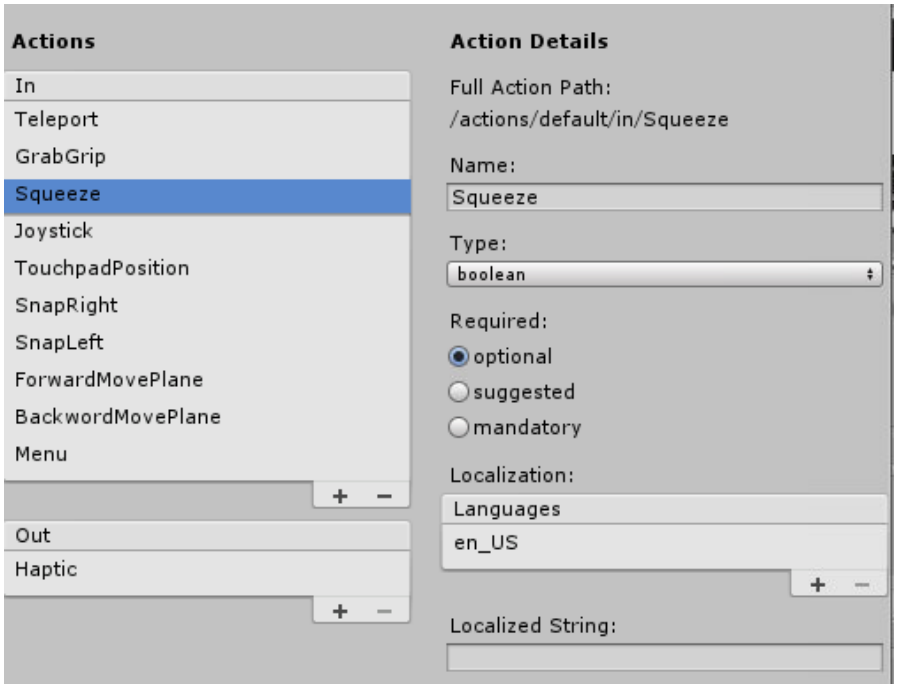
SteamVR_Input.GetStateDown();

SteamVR_Input.GetStateUp();

Η κάθε συνάρτηση παίρνει δυο ορίσματα. Το όνομα του Action που υπάρχει στην λίστα SteamVR Input και το χέρι που αντιπροσωπεύει σε κάθε περίπτωση. Το GetStateDown ελέγχει πότε πιέστηκε (έγινε pressed) κάποιο κουμπί, ενώ το GetStateUp ελέγχει πότε αυτό το κουμπί αφέθηκε (έγινε released).

- **Squeeze**

Εικόνα 90 - Squeeze action και οι ιδιότητες του



Το Squeeze ενεργοποιείται με την σκανδάλη των χειριστηρίων και εχει οριστεί να είναι τύπου Boolean, καθώς δεν γίνεται κάποια χρήση των αναλογικών τιμών, στο παιχνίδι.

Εικόνα 91 - Συνάρτηση Update της κλάσης CubeGridSnap

```
// Update is called once per frame
void Update()
{
    //If trigger is presses on GridPlane
    if (hoverEvent.target != null && (hoverEvent.target.tag.Equals("TempCubeTag")
        || hoverEvent.target.tag.Equals("SpawnObject")) &&
         SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.RightHand))
    {
        //Checks Fuction Selected from UI and activates it
        UIInputManager.DoSelectedFunction(hoverEvent, obj, CurrentGameObjectTransofrm);
    }
}
```

Η κλάση CubeGridSnap χρησιμοποιείται στο επεξεργαστικό μέρος του παιχνιδιού και ελέγχει αν ο παίκτης εχει πατήσει την σκανδάλη στο δεξί χειριστήριο ώστε να τοποθετηθεί ένα αντικείμενο πάνω στην επιφάνεια πλέγματος.

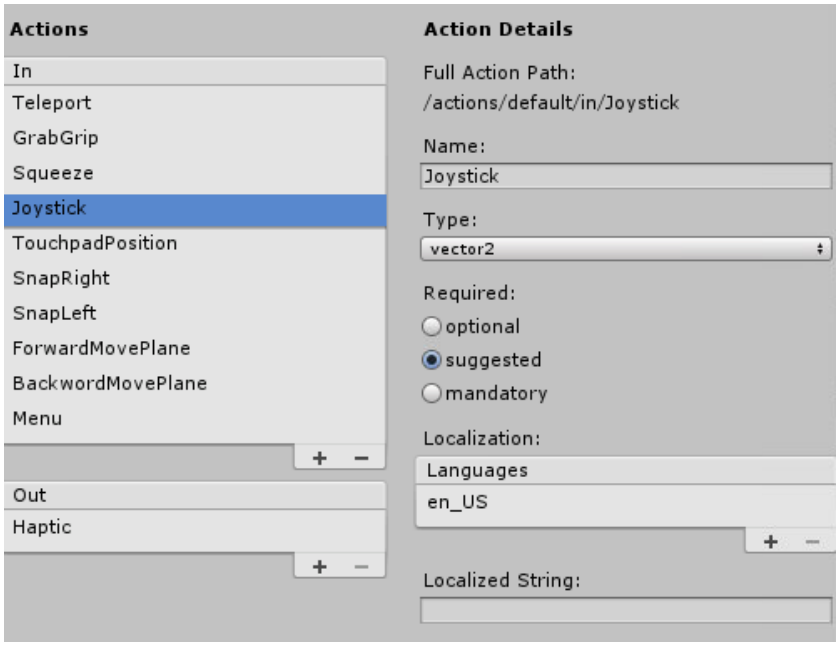
Εικόνα 92 - Κλάση EyeRaycaster και στο επεξεργαστικό και στο ρυθμικό μέρος

```
//Select after controller click
if (SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.LeftHand))
{
    Debug.Log("Clicked");
    Select(newSelectable);
    m_currentRaycastResult = result;
    ButtonSFX.Play();
```

Για να μπορεί ο παίκτης να αλληλεπιδρά με τα αντικείμενα διεπαφής της Unity, χρησιμοποιήθηκε μια εξωτερική επέκταση την EyeRaycaster. Στην κλάση EyeRaycaster αυτής της επέκτασης ελέγχεται αν πάτησε ο παίκτης την σκανδάλη ώστε να επιλεγεί η συγκεκριμένη διεπαφή (π.χ. κουμπί διεπαφής στην Unity) που κοιτάει με το headset.

- **Joystick**

Εικόνα 93 - Joystick action και οι ιδιότητες του



Το Joystick Action είναι τύπου Vector2 και δίνει αναλογικές float τιμές για δυο άξονες. Τον άξονα x και τον άξονα y. Στην συγκεκριμένη υλοποίηση του παιχνιδιού χρησιμοποιήθηκε μόνο άξονας y. Αυτό εγινε διότι γίνεται χρήση του Joystick με κατεύθυνση προς τα πάνω για την μετακίνηση της επιφάνειας του πλέγματος προς τα μπροστά και με κατεύθυνση προς τα κάτω για μετακίνηση της επιφάνειας του πλέγματος προς τα πίσω. Με την χρήση του Joystick αυτή η επιφάνεια κινείται πολλά beats την φορά, που σημαίνει ότι χρησιμεύει στην γρήγορη μετακίνηση της επιφάνειας σε κάποιο σημείο (beat), του μουσικού κομματιού.

Εικόνα 94 - Κλάση MovingPlaneObj στο επεξεργαστικό μέρος

```
public class MovingPlaneObj : MonoBehaviour
{
    //VR Variables
    public SteamVR_Action_Vector2 joystickPosition;
    public GameObject MovingObjects;
```

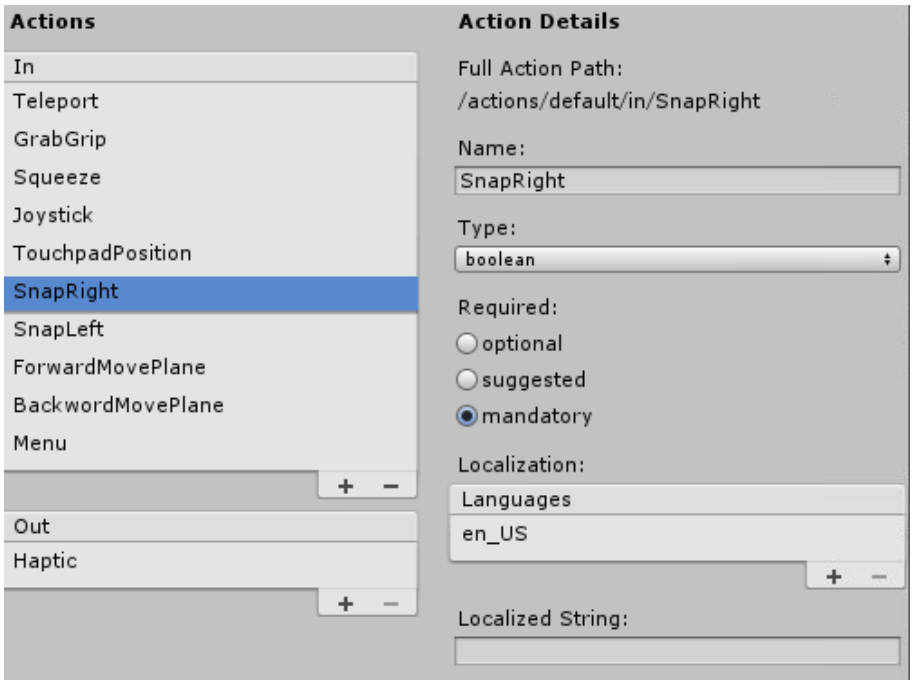
Εικόνα 95 - Κλάση MovingPlaneObj στο επεξεργαστικό μέρος

```
//Update Joystick position every frame
JoystickPositionY = Mathf.RoundToInt(
    joystickPosition.GetAxis(SteamVR_Input_Sources.LeftHand).y * 3);
```

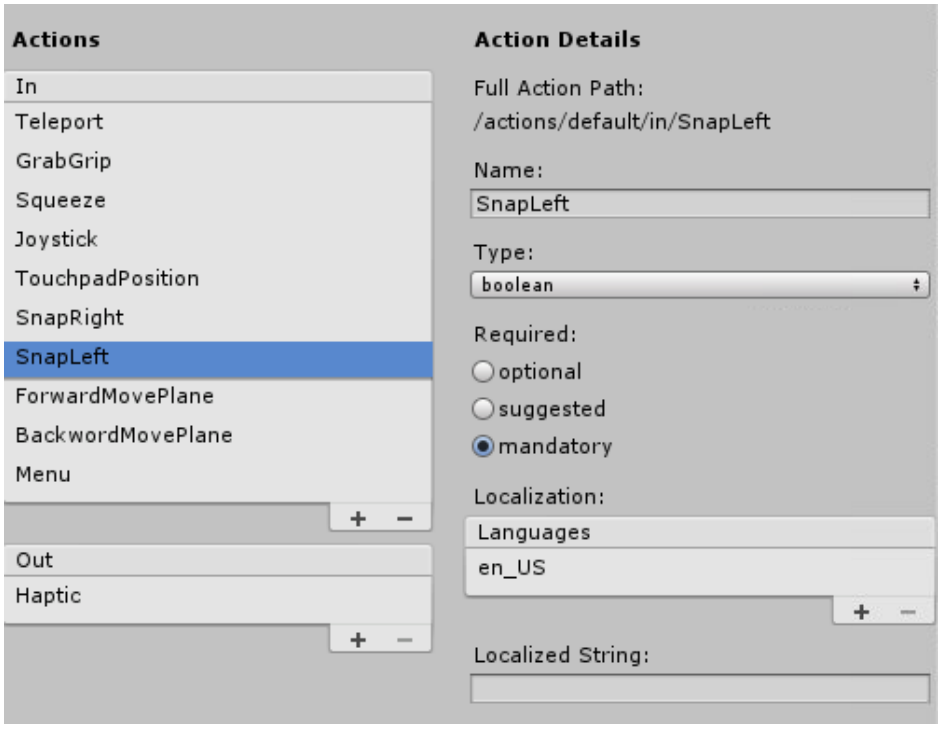
Όπως φαίνεται παραπάνω χρησιμοποιείται αυτή η τιμή με την συνάρτηση της Vector2 μεταβλητής, GetAxis. Δηλώνεται πως πρόκειται για τον y άξονα του αριστερού χειριστηρίου. Τέλος γίνεται στρογγυλοποίηση αυτής της τιμής για να χρησιμοποιηθεί στην συνέχεια. Επίσης το Joystick χρησιμοποιείται μόνο στο επεξεργαστικό μέρος του παιχνιδιού.

- **SnapRight και SnapLeft**

Εικόνα 96 - SnapRight action και οι ιδιότητες του

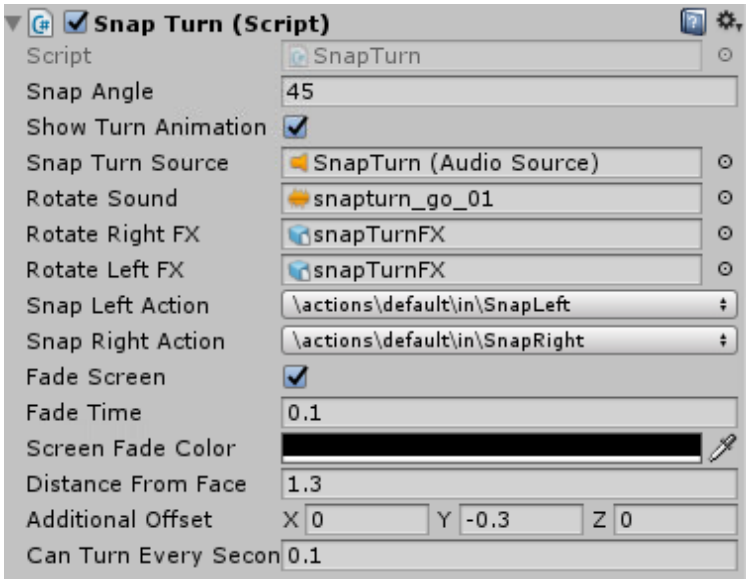


Εικόνα 97 - SnapLeft action και οι ιδιότητες του



Το SnapRight και το SnapLeft Action είναι τύπου boolean και είναι υπεύθυνα, για την καταγραφή ενεργειών που γίνονται από το Thumbstick-Joystick του αριστερού χειριστηρίου στο επεξεργαστικό μέρος του παιχνιδιού.

Εικόνα 98 - Ιδιότητες του SnapTurn Script

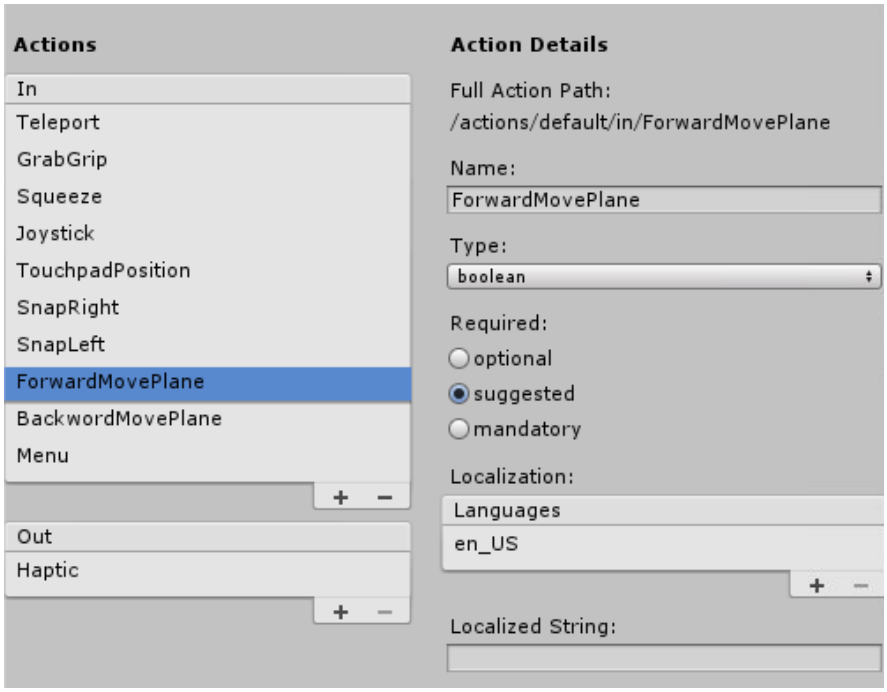


Αυτά τα action είναι απαραίτητα για να χρησιμοποιηθούν στο script SnapTurn. Έτσι γίνεται ανάθεση του Snap Left Action με το SnapLeft και το Snap Right Action με το SnapRight. Με αυτόν τον τροπο γίνεται αντιστοίχιση των Action του χειριστηρίου με τα Action που απαιτεί

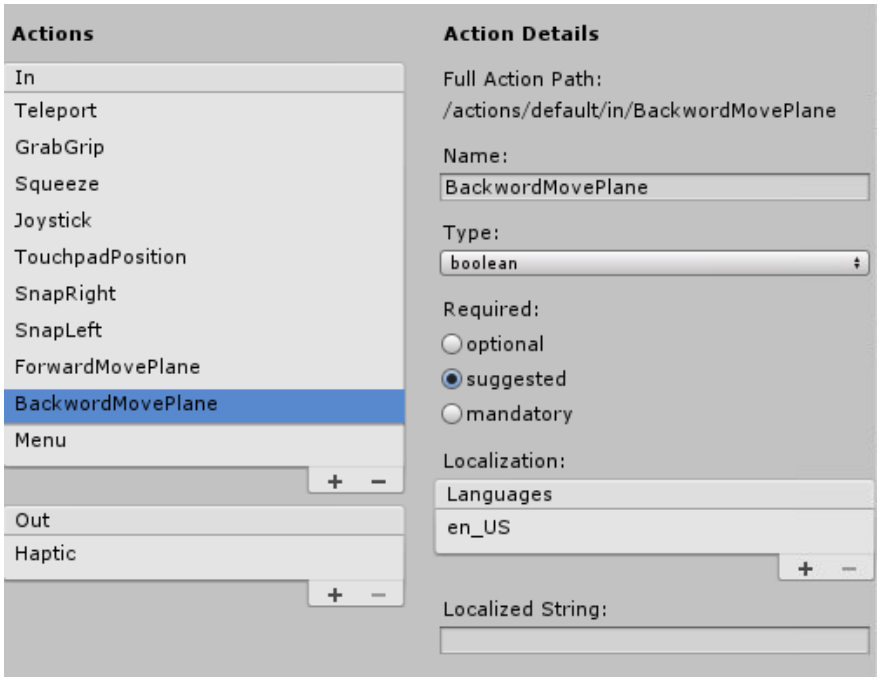
το script ώστε να μπορούν αργότερα να χρησιμοποιηθούν από αυτό. Οι υπόλοιπες ιδιότητες είναι προεπιλεγμένες, και στην συγκεκριμένη υλοποίηση δεν χρειάστηκε κάποια αλλαγή.

- **ForwardMovePlane και BackwardMovePlane**

Εικόνα 99 - ForwardMovePlane action και οι ιδιότητες του



Εικόνα 100 - BackwardsMovePlane action και οι ιδιότητες του



Τα action ForwardMovePlane και BackwardsMovePlane ειναι και αυτά τύπου Boolean και ενεργοποιούνται από τα κουμπιά της επιφάνειας αφής του αριστερού χειριστηρίου. Η χρήση τους γίνεται μόνο στο επεξεργαστικό μέρος του παιχνιδιού.

Αυτά τα Actions είναι υπεύθυνα για την μετακίνηση της επιφάνειας του πλέγματος, όπως και το Joystick. Απλώς σε αυτή την περίπτωση η μετακίνηση της, γίνεται μόνο κατά ένα beat την φορά, προς τα εμπρός ή προς τα πίσω. Το Action που εχει οριστεί για να μετακινηθεί η επιφάνεια προς τα μπροστά είναι το ForwardMovePlane και το Action που εχει οριστεί για την μετακίνηση προς τα πίσω είναι το BackwardMovePlane.

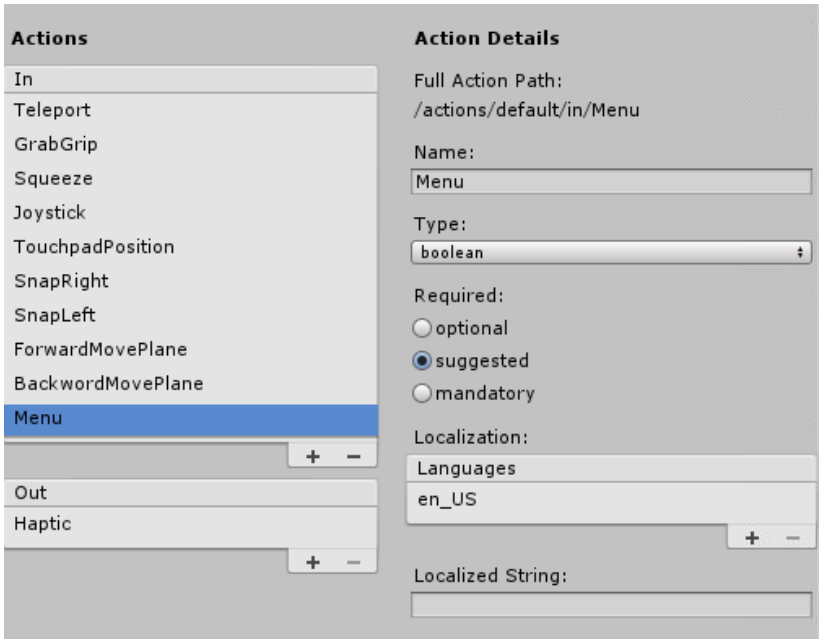
Εικόνα 101 - Εντολές στην κλάση MovingPlaneObj

```
//Moving Plane with Scroll
if (SteamVR_Input.GetStateDown(("ForwardMovePlane"), SteamVR_Input_Sources.LeftHand))
{
    PlanePositionScroll += GridDisplacement;
}
else if (SteamVR_Input.GetStateDown(("BackwordMovePlane"), SteamVR_Input_Sources.LeftHand))
{
    PlanePositionScroll -= GridDisplacement;
}
```

Η χρήση των δυο αυτών action γίνεται από την κλάση MovingPlaneObj, η οποία είναι υπεύθυνη για την μετακίνηση της επιφάνειας του πλέγματος στο παιχνίδι. Έτσι στην Update συνάρτηση της κλάσης ελέγχεται κάθε καρέ, αν ενεργοποιήθηκε κάποιο από αυτά τα action. Έτσι αναλόγως το action, που ενεργοποιήθηκε, το PlanePositionScroll αυξάνεται ή μειώνεται κατά το GridDisplacement, που στην συγκεκριμένη υλοποίηση εχει οριστεί ίσο με 1. Η παραπάνω εικόνα απεικονίζει τις εντολές που εκτελούν αυτές τις λειτουργίες.

- **Menu**

Εικόνα 102 - Menu action και οι ιδιότητες του



Το Menu action ενεργοποιείται με το πάτημα του κουμπιού που είναι κοινώς ορισμένο ως κουμπί «Menu» από τα Standard των Windows Mixed Reality υλοποιήσεων της Microsoft.Παρόλα αυτά, αυτό το κουμπί μπορεί να χρησιμοποιηθεί από τους προγραμματιστές και για άλλες χρήσεις. Στην συγκεκριμένη υλοποίηση έγινε χρήση του κουμπιού και από δυο χειριστήρια.

Εικόνα 103 - Κλάση VRToolTipsHelp επεξεργαστικό και ρυθμικό μέρος

```
// Update is called once per frame
void Update () {
    if (SteamVR_Input.GetStateDown(("Menu"), SteamVR_Input_Sources.LeftHand))
    {
        MenuTooltipObject.SetActive(false);
        AllTooltipObjectsRightHand.SetActive(true);
        AllTooltipObjectsLeftHand.SetActive(true);
    }
    else if (SteamVR_Input.GetStateUp(("Menu"), SteamVR_Input_Sources.LeftHand))
    {
        MenuTooltipObject.SetActive(true);
        AllTooltipObjectsRightHand.SetActive(false);
        AllTooltipObjectsLeftHand.SetActive(false);
    }
}
```

Στο επεξεργαστικό μέρος του παιχνιδιού το Menu Action χρησιμοποιήθηκε για να την εμφάνιση των Tooltips (Εικονικές Υποδείξεις) που υποδεικνύουν τι λειτουργία εκτελεί το κάθε κουμπί πάνω στα χειριστήρια.

Έτσι στην παραπάνω εικόνα φαίνεται η συνάρτηση Update της κλάσης VRToolTipsHelp, η οποία ελέγχει σε κάθε καρέ, αν πατήθηκε ή αφέθηκε το Menu κουμπί από το αριστερό χειριστήριο. Αν πατηθεί τότε εμφανίζονται τα Tooltips, αλλιώς αν αφεθεί τότε γίνεται απόκρυψη αυτών.

Εικόνα 104 - Update της κλάσης PauseTime στο ρυθμικό μέρος

```
// Update is called once per frame
void Update()
{
    if (SteamVR_Input.GetStateDown(("Menu"), SteamVR_Input_Sources.Any)
        && !PauseMenu.active && gameObject.GetComponent<AudioSource>().isPlaying)
    {
        Pause();
        PauseMenu.SetActive(true);
    }
    else if (SteamVR_Input.GetStateDown(("Menu"), SteamVR_Input_Sources.Any)
        && PauseMenu.active && !gameObject.GetComponent<AudioSource>().isPlaying)
    {
        Resume();
        PauseMenu.SetActive(false);
    }
}
```

Εικόνα 105 - Συνάρτηση Pause της κλάσης PauseTime

```
public void Pause()
{
    WeaponsLeft.SetActive(false);
    WeaponsRight.SetActive(false);
    ControllerModelLeft.SetActive(true);
    ControllerModelRight.SetActive(true);

    GetComponent<GameManagerMainGame>().Eyeraycast.SetActive(true);
    GameObject scorePanel = GameObject.Find("ScorePanel");
    scorePanel.GetComponent<Image>().enabled = false;
    scorePanel.transform.GetChild(0).GetComponent<MeshRenderer>().enabled = false;
    scorePanel.transform.GetChild(1).GetComponent<MeshRenderer>().enabled = false;

    GameObject[] planeObjects = GameObject.FindGameObjectsWithTag("PlaneGridTag");
    foreach (GameObject obj in planeObjects)
    {
        obj.GetComponent<MeshRenderer>().enabled = false;
    }

    gameObject.GetComponent<AudioSource>().Pause();
    GameObject[] gameObjects = GameObject.FindGameObjectsWithTag("SpawnObject");
    foreach (GameObject obj in gameObjects)
    {
        obj.GetComponent<MeshRenderer>().enabled = false;
    }
}
```

Εικόνα 106 - Συνάρτηση Resume της κλάσης PauseTime

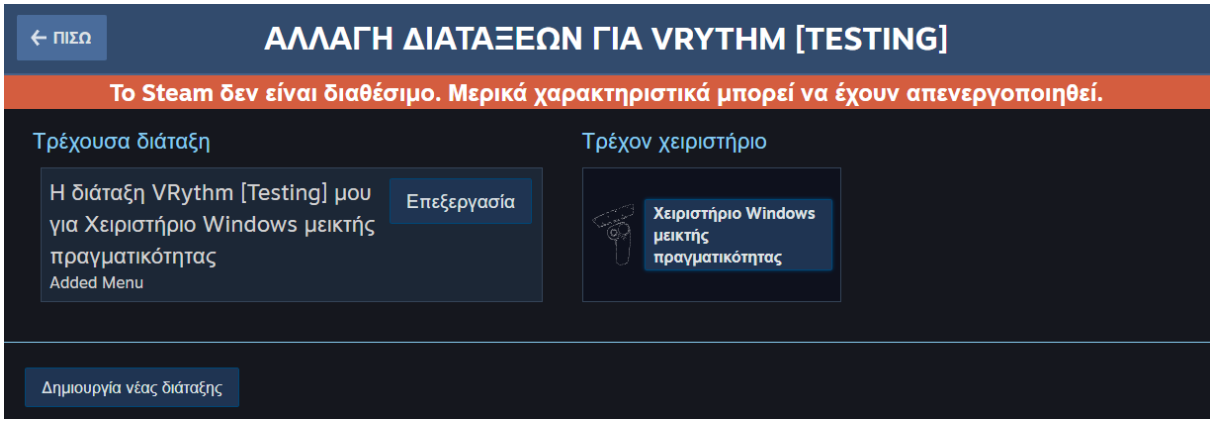
```
public void Resume()
{
    WeaponsLeft.SetActive(true);
    WeaponsRight.SetActive(true);
    ControllerModelRight.SetActive(false);
    ControllerModelLeft.SetActive(false);

    GetComponent<GameManagerMainGame>().Eyeraycast.SetActive(false);
    GameObject scorePanel = GameObject.Find("ScorePanel");
    scorePanel.GetComponent<Image>().enabled = true;
    scorePanel.transform.GetChild(0).GetComponent<MeshRenderer>().enabled = true;
    scorePanel.transform.GetChild(1).GetComponent<MeshRenderer>().enabled = true;

    GameObject[] planeObjects = GameObject.FindGameObjectsWithTag("PlaneGridTag");
    foreach (GameObject obj in planeObjects)
    {
        obj.GetComponent<MeshRenderer>().enabled = true;
    }

    gameObject.GetComponent<AudioSource>().UnPause();
    GameObject[] gameObjects = GameObject.FindGameObjectsWithTag("SpawnObject");
    foreach (GameObject obj in gameObjects)
    {
        obj.GetComponent<MeshRenderer>().enabled = true;
    }
}
```

Στο ρυθμικό μέρος, το Menu Action χρησιμοποιείται διαφορετικά. Όπως φαίνεται στην παραπάνω εικόνα, το Menu action είναι υπεύθυνο για την προσωρινή παύση και την συνέχιση του παιχνιδιού. Αυτό γίνεται με την χρήση της κλάσης PauseTime.

Στην περίπτωση αυτή ελέγχεται αν πατήθηκε το Menu κουμπί από οποιοδήποτε χειριστήριο και αν την συγκεκριμένη χρονική στιγμή είναι σε παύση το παιχνίδι. Αν εχει τεθεί σε παύση τότε με το πάτημα του κουμπιού, το παιχνίδι θα συνεχιστεί, σε διαφορετική περίπτωση το παιχνίδι θα τεθεί σε παύση.

4.3.4 SteamVR Input - Binding UI

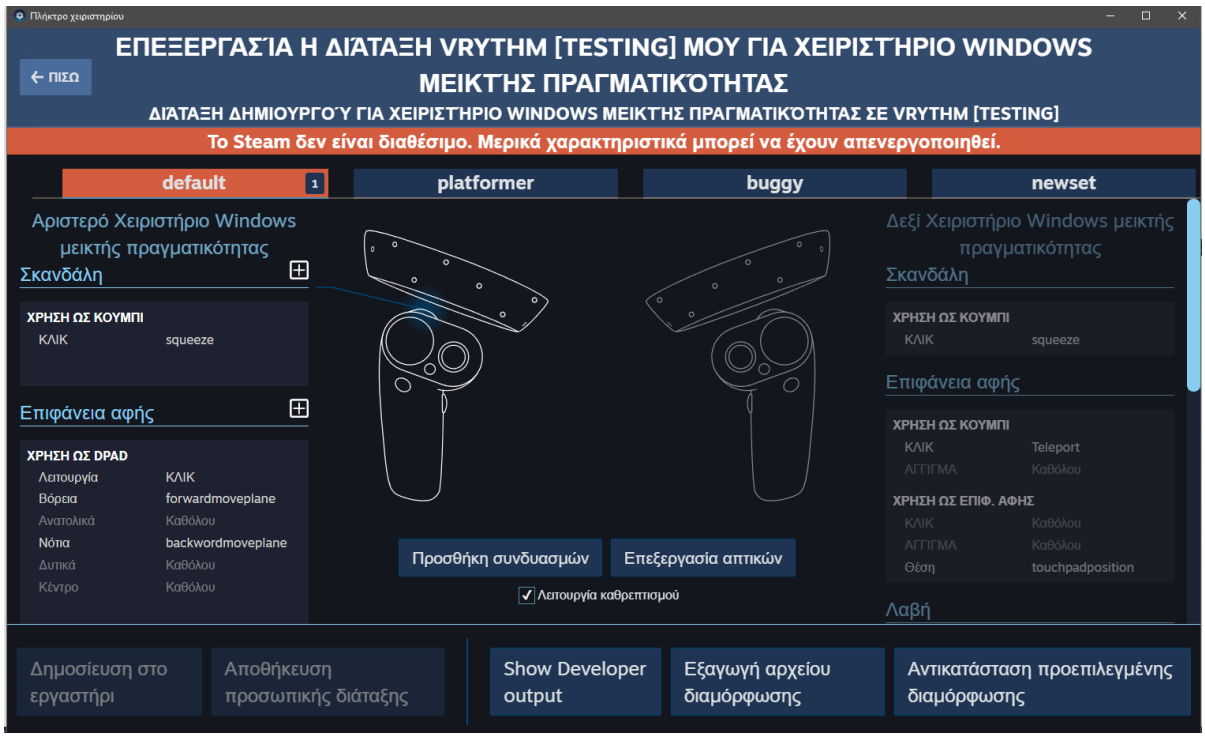
Εικόνα 107 - Αρχικό παράθυρο Binding UI



Η επέκταση SteamVR χρησιμοποιεί μια διεπαφή που μέσω του Binding Window όπου ο προγραμματιστής πρεπει να κανει την διασύνδεση – αντιστοίχιση του κάθε action που δημιούργησε στο SteamVR Input, με τα αντίστοιχα φυσικά κουμπιά των χειριστηρίων.

Σε κάποιες περιπτώσεις υπάρχουν τα ίδια actions με το ίδιο όνομα και στα δυο χειριστήρια. Παρόλα αυτά, αργότερα μέσα από το παιχνίδι ελέγχεται κάθε φορά από ποιο χειριστήριο ενεργοποιήθηκε το κάθε action. Έτσι δεν δημιουργούνται προβλήματα, με ανεπιθύμητα action, σε κάποια σκηνή.

Εικόνα 108 - Παράθυρο επεξεργασίας του Binding UI

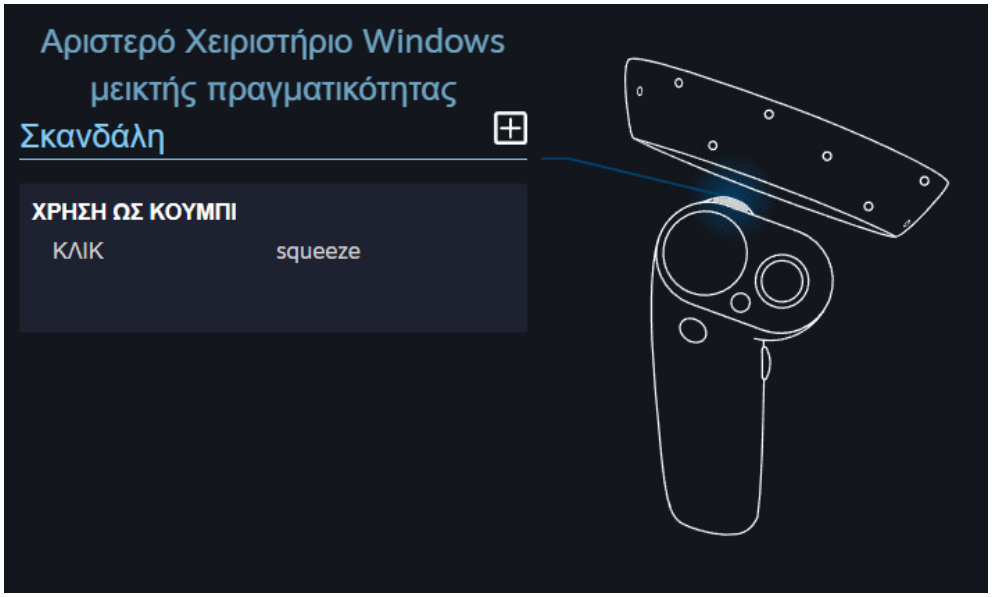


Αριστερό Χειριστήριο

Παρακάτω φαίνονται οι αντιστοιχήσεις των κουμπιών του αριστερού χειριστηρίου.

- **Σκανδάλη**

Εικόνα 109 - Λειτουργία σκανδάλης στο αριστερό χειριστήριο



Για την σκανδάλη ενεργοποιείται το Squeeze Action ως κουμπί (δηλαδή με δυο τιμές true - false).

- **Επιφάνεια αφής - Λαβή**

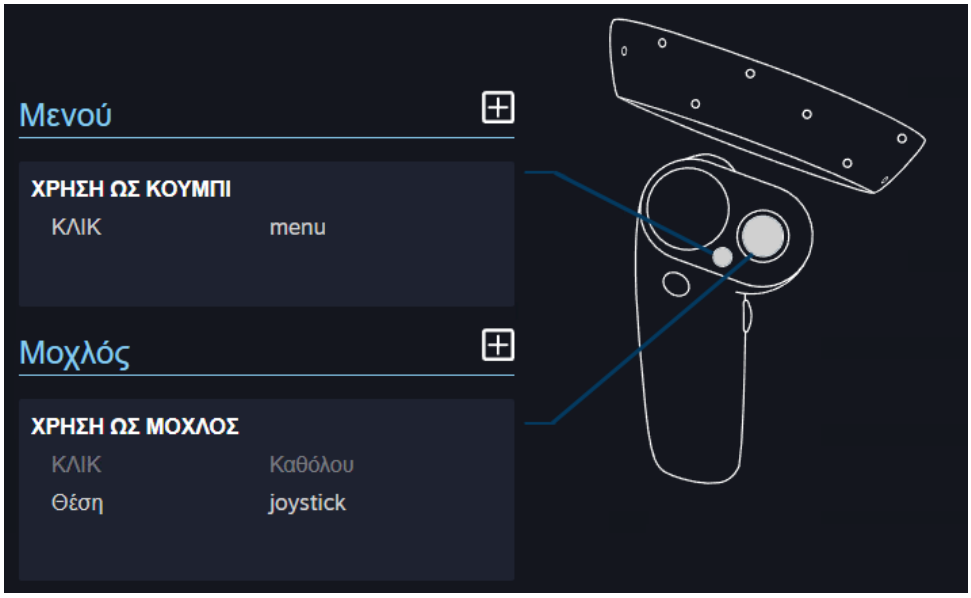
Εικόνα 110 - Λειτουργία επιφάνειας αφής στο αριστερό χειριστήριο

Η επιφάνεια αφής ενεργοποιεί τα ForwardMovePlane και BackwardMovePlane actions. Εχει γίνει ανάθεση στην βόρεια (πάνω) πλευρά της επιφάνειας αφής, όταν πατιέται (γίνεται κλικ) να ενεργοποιείται το ForwardMovePlane action. Ενώ αντίστοιχα όταν πατιέται η νότια (κάτω) πλευρά της επιφάνειας αφής,, ενεργοποιείται το BackwardMovePlane action.

Η λαβή είναι το πλαϊνό κουμπί του χειριστηρίου και όταν πατιέται ενεργοποιείται το GrabGrip action.

- **Μενού - Μοχλός**

Εικόνα 111 - Λειτουργία μενού και μοχλού στο αριστερό χειριστήριο



Το κουμπί μενου είναι τοποθετημενο αναμεσα στην επιφανεια αφης και τον μοχλο - Joystick του χειρηστηριου. Όταν πατιεται ενεργοποιειται το Menu action.

Ο μοχλός ή αλλιώς Joystick είναι υπεύθυνος για την καταγραφή της θέσης (ως Vector2) του και να ενημερώσει το Joystick action από το SteamVR Input. Έτσι με κάθε κίνηση του μοχλού γίνεται ενημέρωση των x και y αξόνων του στο Joystick action.

Δεξί Χειριστήριο

Παρακάτω φαίνονται οι αντιστοιχήσεις των κουμπιών του δεξιού χειριστηρίου.

- **Σκανδάλη**

Εικόνα 112 - Λειτουργία σκανδάλης στο δεξί χειριστήριο

Για την σκανδάλη ενεργοποιείται το Squeeze Action ως κουμπί (δηλαδή με δυο τιμές true - false). Αντίστοιχα όπως και στο αριστερό χειριστήριο.

- **Επιφάνεια Αφής**

Εικόνα 113 - Λειτουργία επιφάνειας αφής στο δεξί χειριστήριο

Η επιφάνεια αφής στο δεξί χειριστήριο ενεργοποιεί το Teleport action. Αυτή η επιφάνεια αφής εχει οριστεί να λειτουργεί ως κουμπί. Δηλαδή μόλις πατηθεί προς τα μέσα η επιφάνεια και κανει κλικ, τότε ενεργοποιείται το Teleport Action.

- **Λαβή - Μενού**

Εικόνα 114 - Λειτουργία λαβής και μενού στο δεξί χειριστήριο

Η λαβή είναι το πλαϊνό κουμπί του χειριστηρίου και όταν πατιέται ενεργοποιείται το GrabGrip action και σε αυτό το χειριστήριο.

Όπως και στο αριστερό χειριστήριο, έτσι και στο δεξί, το κουμπί μενού είναι τοποθετημένο ανάμεσα στην επιφάνεια αφής και τον μοχλό - Joystick του χειριστηρίου και οταν πατιέται ενεργοποιείται το Menu action.

- **Μοχλός**

Εικόνα 115 - Λειτουργία μοχλού στο δεξί χειριστήριο

Ο μοχλός στο δεξί χειριστήριο χρησιμοποιείται διαφορετικά από το αριστερό χειριστήριο. Σε αυτή την περίπτωση, όταν ο μοχλός μετακινείται προς τα ανατολικά (δεξιά) ενεργοποιείται το action SnapRight, ενώ όταν μετακινείται προς τα δυτικά (αριστερά) τότε ενεργοποιείται το SnapLeft action. Επίσης, επειδή η λειτουργία εχει οριστεί ως «Άγγιγμα» κάθε

φορά που ενεργοποιείται κάποιο από αυτά τα actions, η πληροφορία μεταφέρεται ως τύπος Boolean , όπως φαίνεται και στο SteamVR Input.

Κεφάλαιο 5 : Διεπαφή του Παιχνιδιού

Γραφικό περιβάλλον ή γραφική διεπαφή χρήστη, ειναι το σύνολο των εικονικών αντικειμένων, που απεικονίζονται σαν γραφικά στοιχεία, σε μια ψηφιακή συσκευή. Αυτά τα γραφικά στοιχεία, στοχεύουν στην διασύνδεση και αλληλεπίδραση ενός χρήστη και μιας ψηφιακής συσκευής. Ο σχεδιασμός ενός σωστού γραφικού περιβάλλοντος, πρεπει να τηρεί κάποιες προϋποθέσεις, προκειμένου να μπορεί να χρησιμοποιηθεί σε κάποια εφαρμογή ή παιχνίδι. Όπως ειναι η λειτουργικότητα, η ευχρηστία, ακόμα και η ομορφιά σε ένα γραφικό περιβάλλον εργασίας.

(Wikipedia, Γραφικό_περιβάλλον_χρήστη, 2018)

Έτσι στο συγκεκριμένο παιχνίδι, προκειμένου να μπορεί ο παίκτης να αλληλεπιδράσει με αυτό, εγινε υλοποίηση διάφορων γραφικών διεπαφών μέσα σε αυτό. Με την χρήση αυτών των διεπαφών ο παίκτης μπορεί να επιλέξει οποιαδήποτε στιγμή να προχωρήσει σε ένα μέρος του παιχνιδιού ή να οπισθοδρομήσει από ένα σημείο του παιχνιδιού. Δηλαδή να προηγηθεί στο παιχνίδι. Επίσης, μπορεί να κάνει επιλογές που διαμορφώνουν το παιχνίδι και την εμπειρία του. Αυτό μπορεί να επιτευχθεί με τον χειρισμό διαφόρων αντικειμένων από τον παίκτη.

5.1 Τεχνικές χρήσης UI σε τεχνολογία VR

Η δημιουργία μιας γραφικής διεπαφής σε τεχνολογία εικονικής πραγματικότητας, εχει κάποιες διαφορές σε σχέση με τις διεπαφές που δημιουργούνται για άλλες συμβατικές εφαρμογές ή παιχνίδια. Αυτές οι διαφορές είναι απαραίτητες ώστε να δημιουργηθεί μια εύχρηστη και ανώδυνη εμπειρία στον χρήστη.

Υπάρχουν κάποιες βασικές τεχνικές ώστε να επιτευχθεί αυτή η εμπειρία σε ένα engine όπως η Unity.

- **World Space**

Ένα από τα βασικότερα είναι η δημιουργία της διεπαφής σε World Space (τρισδιάστατο χώρο). Δηλαδή όταν δημιουργείται ένας καμβάς (Canvas) στην Unity, όπου σε αυτόν πάνω τοποθετούνται τα στοιχεία διεπαφής (π.χ. όπως κουμπιά), να γίνεται αλλαγή στον τρόπο απεικόνισης από δυο διαστάσεις που χρησιμοποιείται σε συμβατικά (μη VR) παιχνίδια, σε τρείς διαστάσεις μέσα στον εικονικό χώρο του περιβάλλοντος.

- **World Space Raycaster**

Άλλη μια βασική τεχνική είναι η αλλαγή του τρόπου αλληλεπίδρασης με τις διεπαφές. Σε ένα κανονικό (μη VR) παιχνίδι της Unity, γίνεται χρήση ενός mouse ή ενός πληκτρολογίου. Σε αυτή την περίπτωση που το παιχνίδι είναι σε VR δεν προτείνεται να χρησιμοποιηθούν αυτές οι εξωτερικές συσκευές καθώς κάνουν την εμπειρία του χρήστη πιο οδυνηρή. Έτσι έχοντας μόνο

τα χειριστήρια σαν μέσω αλληλεπίδρασης, πρεπει να γίνεται αυτή η αλληλεπίδραση με άλλη προσέγγιση.

Η καλύτερη προσέγγιση είναι να γίνονται οι αλληλεπιδράσεις με world space Raycasters. Δηλαδή με μια τρισδιάστατη ακτίνα - ευθεία που εχει μια πορεία και μπορεί να δείξει σε οποιοδήποτε σημείο στο τρισδιάστατο περιβάλλον του παιχνιδιού. Ο δείκτης του ποντικιού λειτουργεί ως μια δισδιάστατη ακτίνα (Raycaster) που χρησιμοποιείται από την Unity από προεπιλογή για την αλληλεπίδραση των δισδιάστατων διεπαφών της. Έτσι αντίστοιχα μπορεί να τοποθετηθεί ένα τρισδιάστατο Raycaster πάνω στα χειριστήρια ή ένα δισδιάστατο στην μέση του VR Headset ώστε να γίνεται αλληλεπίδραση με κάποια διεπαφή στον εικονικό κόσμο, χωρίς την χρήση άλλων εξωτερικών συσκευών. Τα Raycasters, ουσιαστικά είναι υπεύθυνα να αλληλοεπιδρούν από προεπιλογή, μέσω του Event System της Unity, με οποιαδήποτε διεπαφή της Unity εχει δημιουργηθεί στο παιχνίδι.

- **Physical Interaction**

Επίσης μια καλή τεχνική που βοηθαει πολύ στην εμπειρία του παίκτη μέσα στο παιχνίδι, είναι η αλληλεπίδραση με την χρήση «φυσικών» (ουσιαστικά τρισδιάστατων), αντικειμένων. Ένα παράδειγμα μπορεί να είναι η αλληλεπίδραση των εικονικών χεριών του παίκτη με ένα τρισδιάστατο κουμπί. Δηλαδή να εκτελείται μια ενέργεια, όταν ο παίκτης πιέσει με το χέρι του αυτό το κουμπί στον εικονικό κόσμο. Μια άλλη παρόμοια υλοποίηση μπορεί να είναι, με ένα τρισδιάστατο εικονικό πληκτρολόγιο με το οποιο ο παίκτης θα αλληλεπίδρα πατώντας (ακουμπώντας) με το δάκτυλο του εικονικού χεριού του κάποιο πλήκτρο αυτού του πληκτρολογίου. Στο συγκεκριμένο παιχνίδι δεν εγινε κάποια παρόμοια υλοποίηση διεπαφής, όπου γίνεται αλληλεπίδραση με «φυσικά» αντικείμενα.

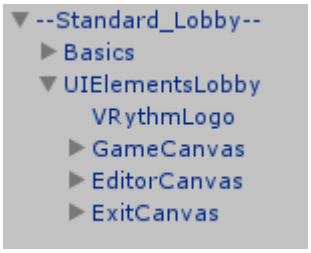
5.2 Διεπαφή αρχικής σκηνής

Στην παρακάτω εικόνα φαίνεται η διεπαφή της αρχικής σκηνής του παιχνιδιού. Αυτή η σκηνή είναι η πρώτη που φορτώνεται όταν ξεκινάει το παιχνίδι.
Οι επιλογές που δίνει στον παίκτη είναι, το άνοιγμα του επεξεργαστικού μέρους του παιχνιδιού (Enter Track Editor), το άνοιγμα του ρυθμικού - κύριου μέρους του παιχνιδιού (Start Game) και η επιλογή εξόδου (Exit) από το παιχνίδι.

Εικόνα 116 - Εικόνα απο το μενού του παιχνιδιού

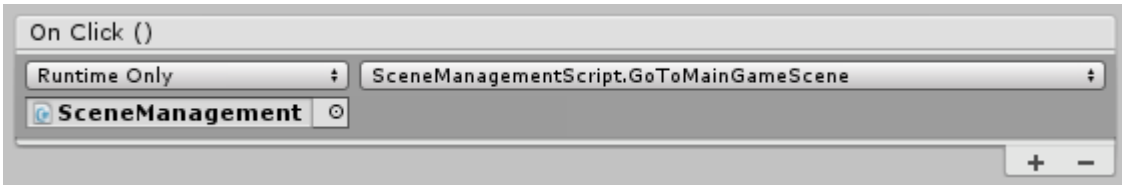
Στην ιεραρχία αυτής της αρχικής σκηνής φαίνονται τα αντικείμενα με τα στοιχεία διεπαφής (UIElementsLobby), τα οποία είναι καμβάδες (Canvas) που περιέχουν τα κουμπιά.

Εικόνα 117 - Αντικείμενα στην ιεραρχία της αρχικής σκηνής



- **GameCanvas**

Εικόνα 118 - Ιδιότητες του Game Button



Στο GameCanvas υπάρχει ένα κουμπί που μέσω του editor της Unity γίνεται ορισμός των ενεργειών που εκτελούνται όταν πατηθεί αυτό το κουμπί. Αυτό γίνεται με την On Click() και εκτελείται η συνάρτηση GoToMainGameScene() της κλάσης Scene Management.

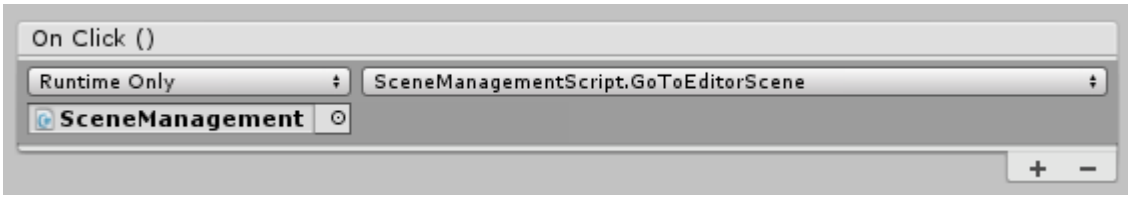
Εικόνα 119 - Συνάρτηση της κλάσης SceneManagement

```
public void GoToMainGameScene()
{
    SceneManager.LoadScene("gameScene", LoadSceneMode.Single);
}
```

Αυτή η συνάρτηση όταν εκτελεστεί φορτώνει την σκηνή με όνομα «gameScene» η οποία είναι η σκηνή του ρυθμικού μέρους του παιχνιδιού.

- **EditorCanvas**

Εικόνα 120 - Ιδιότητες του Editor Button



Με τον ίδιο τρόπο, στο EditorCanvas υπάρχει το κουμπί που εκτελεί την συνάρτηση GoToEditorScene() της κλάσης Scene Management.

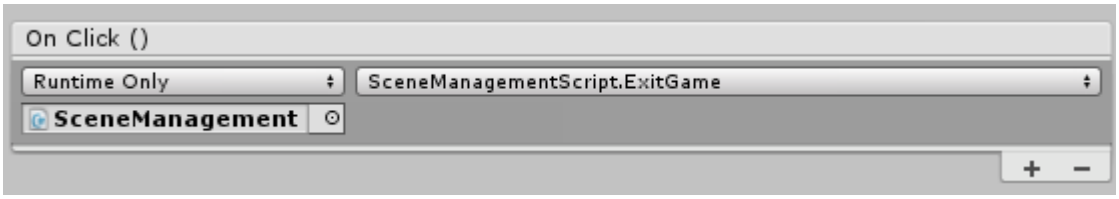
Εικόνα 121 - Συνάρτηση της κλάσης SceneManagement

```
public void GoToEditorScene()
{
    SceneManager.LoadScene("editorScene", LoadSceneMode.Single);
}
```

Αυτή η συνάρτηση όταν εκτελεστεί φορτώνει την σκηνή με όνομα «editorScene» η οποία είναι η σκηνή του επεξεργαστικού μέρους του παιχνιδιού.

- **ExitCanvas**

Εικόνα 122 - Ιδιότητες του Exit Button



Το κουμπί που υπάρχει στο ExitCanvas είναι υπεύθυνο να εκτελέσει την συνάρτηση ExitGame της κλάσης Game Management, όταν πατηθεί.

Εικόνα 123 - Συνάρτηση της κλάσης SceneManagement

```
public void ExitGame()
{
    Application.Quit();
}
```

Αυτή η συνάρτηση είναι υπεύθυνη να κλείσει το παιχνίδι, όταν εκτελεστεί.

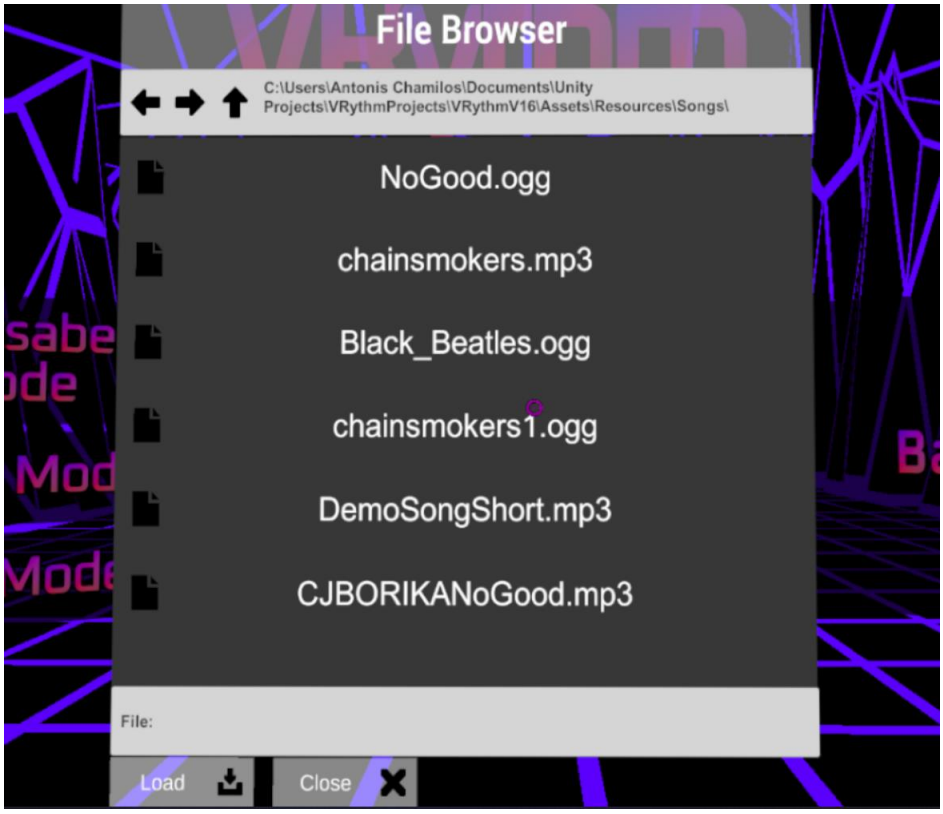
5.3 Διεπαφή ρυθμικού μέρους του παιχνιδιού

Στην παρακάτω εικόνα φαίνεται η διεπαφή της σκηνής του ρυθμικού μέρους του παιχνιδιού. Σε αυτή την σκηνή ο παίκτης κάνει επιλογές για τον τρόπο παιξίματος (Game Modes).
Για αρχή επιλέγει ένα από τα τρία game modes. Το Lightsaber Mode, το Saber Mode και το Rifle Mode. Αν δεν επιλέξει κάποιο Mode τότε τα κουμπιά Load Track και Play παραμένουν απενεργοποιημένα μεχρι ο παίκτης να κανει κάποια επιλογή.

Εικόνα 124 - Εικόνα απο το μενού του ρυθμικού μέρους του παιχνιδιού

Αφού επιλέξει κάποιο mode, τότε ενεργοποιείται το κουμπί Load Track ώστε να το πατήσει ο παίκτης και να επιλέξει και να φορτώσει από το File Browser ένα μουσικό κομμάτι. Το κουμπί Play παραμένει απενεργοποιημένο μεχρι να φορτωθεί κάποιο μουσικό κομμάτι.

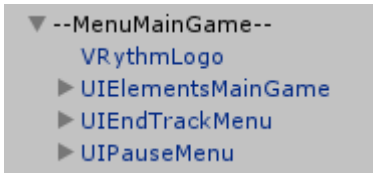
Εικόνα 125 - Εικόνα απο την περιήγηση αρχείων του παιχνιδιού



Μετα που ο παίκτης επιλέξει το μουσικό κομμάτι από τον περιηγητή αρχείων (File Browser) πατάει στο Load και το μουσικό κομμάτι φορτώνεται στο παιχνίδι και το κουμπί Play γίνεται ενεργό, ώστε να μπορεί ο παίκτης να ξεκινήσει να παίζει.

i. MenuMainGame

Εικόνα 126 - Αντικείμενα στην ιεραρχία της σκηνής, του ρυθμικού μέρους



Στο αντικείμενο MenuMainGame υπάρχουν όλα τα αντικείμενα διεπαφής που υπαρχουν στο ρυθμικό μέρος του παιχνιδιού. Αυτά χωρίζονται σε τρεις κατηγορίες. Τα αντικείμενα UIElementsMainGame που είναι η αρχική διεπαφή όπου ο παίκτης κάνει επιλογές για να ξεκινήσει το παιχνίδι. Τα αντικείμενα UIEndTrackMenu που είναι η διεπαφή που εμφανίζεται όταν τελειώσει το τραγούδι στο ρυθμικό μέρος. Και τα αντικείμενα UIPauseMenu που εμφανίζονται όταν ο παίκτης πατήσει το κουμπί “Menu” από το χειριστήριο, την ώρα του ρυθμικού παιχνιδιού.

ii. UIElementsMainGame

Το UIElementsMainGame αποτελείται από έξι καμβάδες (Canvas) που περιέχουν κουμπιά. Το PlayCanvas, το LoadCanvas, το LightSaberModeCanvas, το SaberModeCanvas, το RifleModeCanvas και τέλος το BackCanvas.

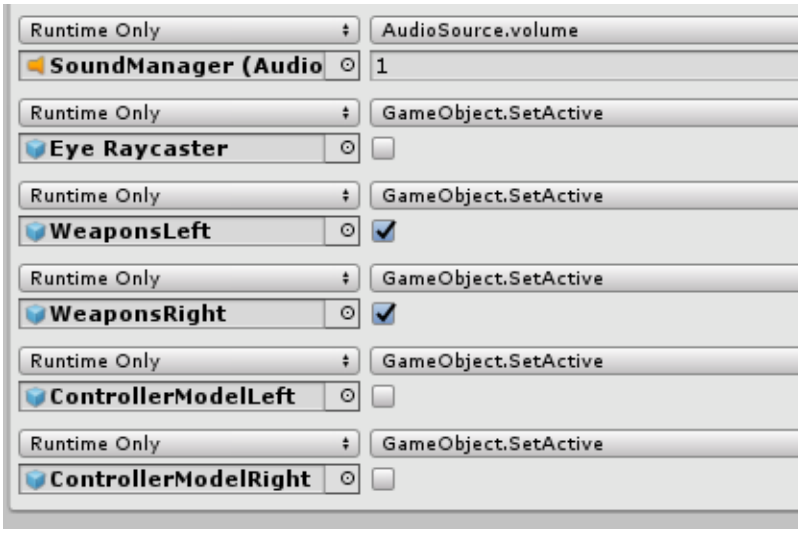
Εικόνα 127 - Αντικείμενα στην ιεραρχία της σκηνής, του ρυθμικού μέρους

- **PlayCanvas**

Όταν πατιέται το κουμπί Play του PlayCanvas γίνονται οι παρακάτω ενέργειες στην OnClick(). Οι πιο πολλές ενέργειες είναι αρχικοποιήσεις άλλων αντικειμένων (δηλαδή εμφάνιση/απόκρυψη ή ενεργοποίηση/απενεργοποίηση αντικειμένων). Όπως η απόκρυψη του λογότυπου και των στοιχείων της αρχικής διεπαφής (Menu - UIElementsMainGame), και η αλλαγή ιδιοτήτων των Audio Manager. Επίσης γίνεται εναλλαγή των εικονικών μοντέλων των χειριστηρίων, σε όπλα που εχει το κάθε χέρι αναλόγως το mode. Ουσιαστικά, γίνεται απόκρυψη των μοντέλων των χειριστηρίων και εμφάνιση των όπλων.

Εικόνα 128 - Ιδιότητες του Play Button

Εικόνα 129 - Ιδιότητες του Play Button (συνέχεια)



Η βασική λειτουργία αυτού του κουμπιού είναι η εκτέλεση της συνάρτησης StartMainGame() της κλάσης GameManagerMainGame, όπου δημιουργεί τα StandardObjects (αντικείμενα του ρυθμικού παιχνιδιού), μηδενίζει το σκορ, φορτώνει και δημιουργεί τα Streaming Assets (τα κουτιά και τα σφαιρίδια) μέσω των JSON αρχείων, που είχαν δημιουργηθεί στο επεξεργαστικό μέρος και τέλος αρχικοποιεί και ενεργοποιεί το game mode που είχε προηγουμένως επιλέξει ο παίκτης.

Παρακάτω φαίνονται αυτές οι συναρτήσεις και οι εντολές τους.

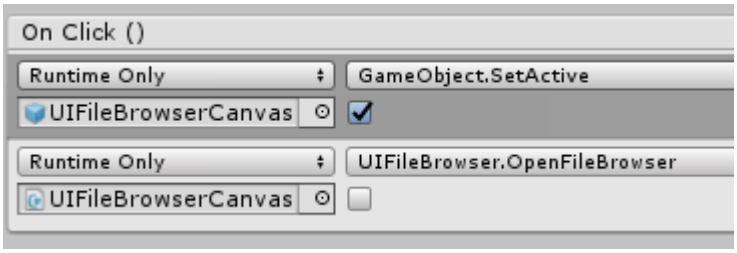
Εικόνα 130 - Συνάρτηση της κλάσης GameManagerMainGame

```
public void StartMainGame()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
    Invoke("ExecuteAfterTimeJsonLoad", 0.5f);
}

public void ExecuteAfterTimeJsonLoad()
{
    Instantiate(StandardObjects);
    TotalScore = 0;
    GameObject.Find("JSON_OperationsMainGame").
        GetComponent<JsonOperationsMainGame>().CreateLoadedObjects();
    GetComponent<AudioSource>().Stop();
    GetComponent<AudioSource>().Play();
    ActivateGameMode();
}
```

- **LoadCanvas**

Το LoadCanvas εχει το κουμπί Load Track το οποίο ανοίγει το UIFileBrowserCanvas το οποιο είναι ο περιηγητής αρχείων (File Browser), ώστε να επιλέξει ο παίκτης το μουσικό κομμάτι που θέλει να φορτώσει.

Εικόνα 131 - Ιδιότητες του Load Button



Η κλάση UIFileBrowserCanvas χρησιμοποιεί την συνάρτηση OpenFileBrowser για να εμφανίσει αναλόγως είτε το FileBrowser που ειναι για Save είτε για Load. Στην OnClick() αν το checkbox του OpenFileBowser δεν ειναι ενεργό τότε ανοίγει το Load Dialog, ενώ αν ειναι ενεργό ανοίγει το Save Dialog. Στην συγκεκριμένη περίπτωση που χρειάζεται να φορτωθεί εάν μουσικό αρχείο, γίνεται χρήση του Load Dialog.

.

Εικόνα 132 - Συνάρτηση της κλάσης File Browser

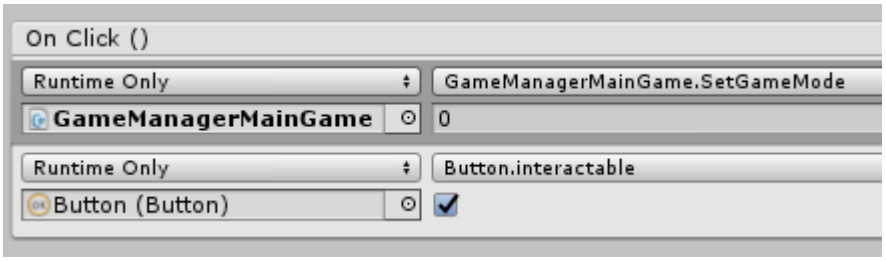
```
// Open a file browser to save and load files
protected void OpenFileBrowser(FileBrowserMode fileBrowserMode)
{
    // Create the file browser and name it
    GameObject fileBrowserObject = Instantiate(FileBrowserPrefab, transform);
    fileBrowserObject.name = "FileBrowser";
    // Set the mode to save or load
    FileBrowser fileBrowserScript = fileBrowserObject.GetComponent<FileBrowser>();
    fileBrowserScript.SetupFileBrowser(
        PortraitMode ? ViewMode.Portrait : ViewMode.Landscape);

    if (fileBrowserMode == FileBrowserMode.Save)
    {
        fileBrowserScript.SaveFilePanel("DemoText", FileExtensions);
        // Subscribe to OnFileSelect event (call SaveFileUsingPath using path)
        fileBrowserScript.OnFileSelect += SaveFileUsingPath;
    }
    else
    {
        fileBrowserScript.OpenFilePanel(FileExtensions);
        // Subscribe to OnFileSelect event (call LoadFileUsingPath using path)
        fileBrowserScript.OnFileSelect += LoadFileUsingPath;
    }
}
```

- **LightSaberModeCanvas, SaberModeCanvas και RifleModeCanvas**

Αυτές οι διεπαφές ειναι κουμπιά που χρησιμοποιεί ο παίκτης για να επιλέξει με ποιο mode θα παίξει το παιχνίδι. Έτσι στις παρακάτω εικόνες φαίνονται οι λειτουργίες που εκτελούν στην OnClick() το κάθε κουμπί. Η μια λειτουργία ειναι να ενημερώσει τον Game Manager το game mode. Στο Game Manager αναγνωρίζεται το game mode με τους αριθμούς 0 - 1 – 2. Το νούμερο 0 ειναι το Saber Mode, το νούμερο 1 ειναι το Rifle Mode και το νούμερο 2 ειναι το LightSaber Mode. Η άλλη λειτουργία ειναι σε κάθε ένα απο αυτά τα κουμπιά οταν πατηθούν να κάνουν το κουμπί Load Track ενεργό για αλληλεπίδραση.

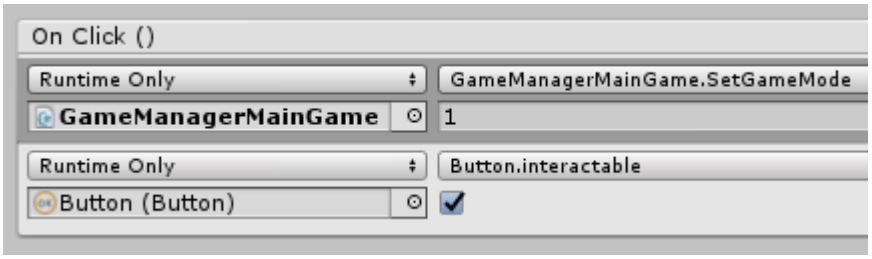
- **SaberModeCanvas**

Εικόνα 133 - Ιδιότητες του Saber Mode Button



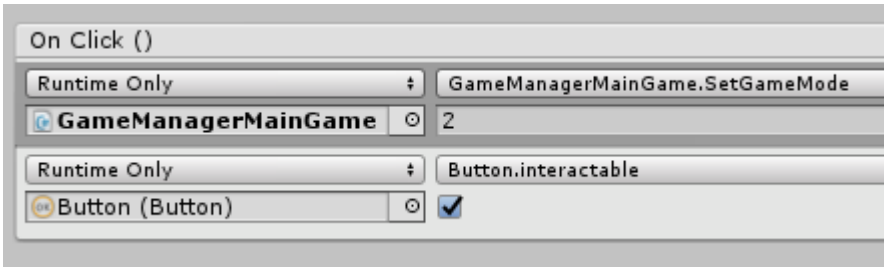
- **RifleModeCanvas**

Εικόνα 134 - Ιδιότητες του Rifle Mode Button



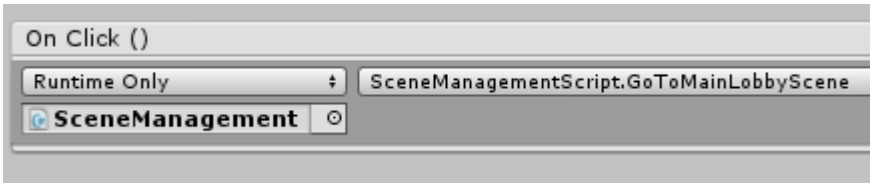
- **LightSaberModeCanvas**

Εικόνα 135 - Ιδιότητες του Lightsaber Mode Button



- **BackCanvas**

Εικόνα 136 - Ιδιότητες του Back Button



Το BackCanvas εχει ένα κουμπί που οταν ο παίκτης το πατήσει εκτελείται η συνάρτηση GoToMainLobbyScene.

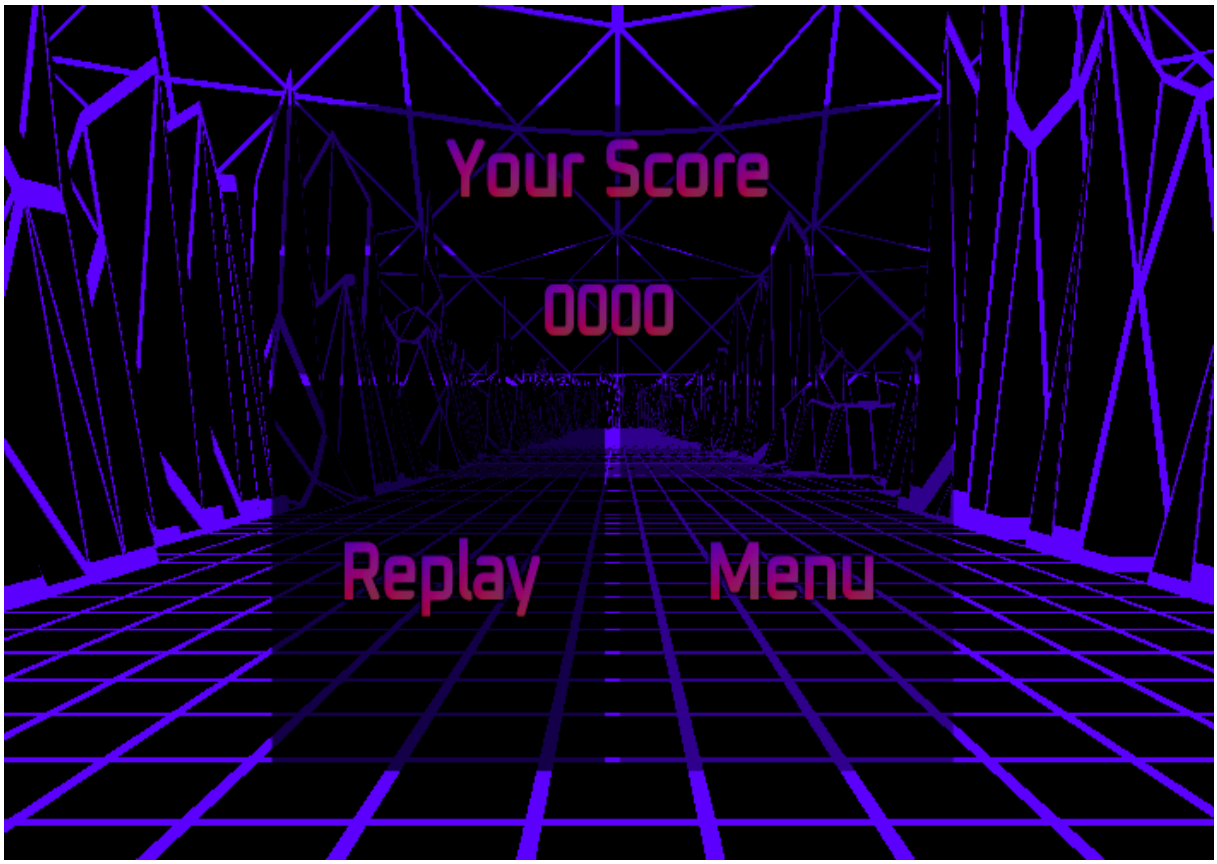
Εικόνα 137 - Συνάρτηση της κλάσης SceneManagement

```
public void GoToMainLobbyScene()
{
    SceneManager.LoadScene("mainLobbyScene", LoadSceneMode.Single);
}
```

Αυτό που κάνει η συνάρτηση αυτή ειναι να φορτώσει την πρώτη σκηνή «mainLobbyScene» με την χρήση του Scene Manager.

iii. UIEndTrackMenu

Εικόνα 138 - Εικόνα απο το μενού του ρυθμικού μέρους του παιχνιδιού, όταν τελειώνει



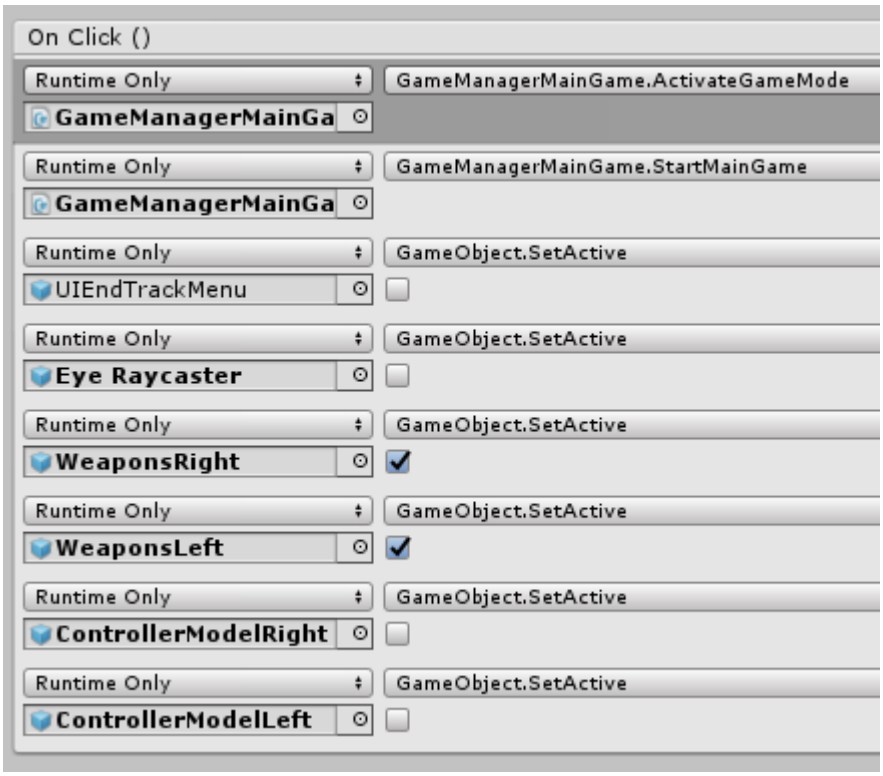
Το αντικείμενο UIEndTrackMenu ειναι το γκρουπ διεπαφών, το οποίο εμφανίζεται οταν τελειώσει το ρυθμικό παιχνίδι – ρυθμικό μουσικό κομμάτι, ώστε να μπορεί ο χρήστης να πάρει μια απόφαση για το πως να προχωρήσει μέσα στο παιχνίδι απο αυτό το σημείο. Επίσης ενημερώνει τον παίκτη για το σκορ που έκανε σε αυτό το μουσικό κομμάτι.

Εικόνα 139 - Αντικείμενα στην ιεραρχία της σκηνής, του ρυθμικού μέρους

Το UIEndTrackMenu εχει 4 αντικείμενα διεπαφών, 2 εκ των οποίων ειναι κουμπιά. Το ReplayCanvas και το MenuCanvas εμπεριέχουν κουμπιά, ενώ το ScoreCanvasTitle και το ScoreCanvas ειναι διεπαφές κειμένου.

- **Replay Canvas**

Εικόνα 140 - Ιδιότητες του Replay Button



Το κουμπί Replay εχει σχεδόν την ίδια λειτουργία με αυτή του play. Ουσιαστικά κάνει εναλλαγή απο τα μοντέλα των χειριστηρίων στα μοντέλα των όπλων. Δηλαδή απόκρυψη των χειριστηρίων και εμφάνιση των όπλων, όπως φαίνεται στην παραπάνω εικόνα με τα Checkboxes. Επίσης απενεργοποιεί την κουκίδα του EyeRaycaster που υπάρχει στην μέση της οθόνης, που χρησιμεύει στην επιλογή διεπαφών, καθώς και τα αντικείμενα αυτής της διεπαφής, UIEndTrackMenu.

Οι βασικές λειτουργίες γίνονται μέσω του Game Manager. Έτσι με το πάτημα του replay εκτελούνται οι συναρτήσεις, ActivateGameMode η οποία ειναι υπεύθυνη για την αρχικοποίηση του game mode και μετα η StartMainGame που ειναι υπεύθυνη για το ξεκίνημα του ρυθμικού παιχνιδιού, όπως ηταν και για το κουμπί play.

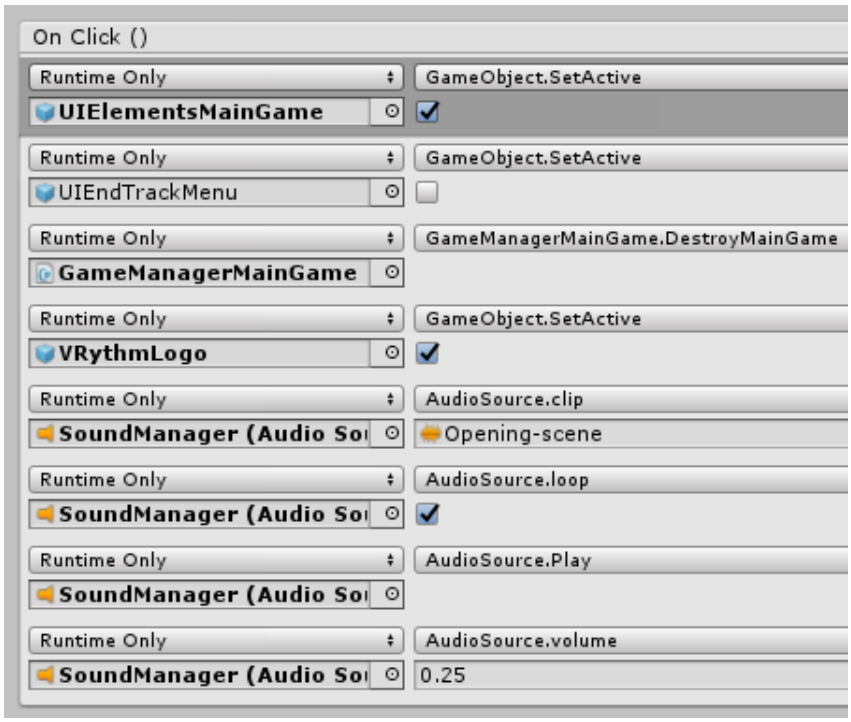
Εικόνα 141 - Συνάρτηση της κλάσης GameManagerMainGame

```
public void ActivateGameMode()
{
    if (GameModeNum.Equals(0))
    {
        InitializeSaberMode();
    }
    else if (GameModeNum.Equals(1))
    {
        InitializeRifleMode();
    }
    else if (GameModeNum.Equals(2))
    {
        InitializeLightSaberMode();
    }
}
```

Στην παραπάνω εικόνα φαίνεται η συνάρτηση ActivateGameMode() που ανάλογα με τον αριθμό που είχε οριστεί προηγουμένως μέσω του SetGameMode() θα κανει και την αντίστοιχη αρχικοποίηση που εχει οριστεί για κάθε mode.

- **MenuCanvas**

Εικόνα 142 - Ιδιότητες του Menu Button



Και στην περίπτωση του MenuCanvas γίνονται αρχικοποιήσεις για να ειναι η σκηνή όπως ηταν στην αρχή στο μενού. Δηλαδή ενεργοποιείται το λογότυπο του μενού, αποκρύπτεται η διεπαφή UIEndTrackMenu και εμφανίζεται η αρχική διεπαφή του μενού, UIElementsMainGame. Επιπλέον γίνονται αλλαγές στον SoundManager για να παίζει η μουσική του παρασκήνιου στο μενού.

Εικόνα 143 - Συνάρτηση της κλάσης GameManagerMainGame

```
public void DestroyMainGame()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
}
```

Τέλος εκτελείται η συνάρτηση DestroyMainGame() απο το GameManagerMainGame η οποία ειναι υπεύθυνη για να καταστρέψει όλα τα αντικείμενα που έχουν tag “StandardObjectsTag”. Αυτά τα αντικείμενα ειναι τα Streaming Assets (κουτιά και σφαιρίδια).

- **ScoreCanvas**

Εικόνα 144 - Συνάρτηση Update της κλάσης GameManagerMainGame

```
// Update is called once per frame
void Update()
{
    //Checks if clip has reached the end and activates the menu
    if (!GetComponent<AudioSource>().isPlaying &&
        GetComponent<AudioSource>().clip.length
        .Equals(GetComponent<AudioSource>().time))
    {
        Eyeraycast.SetActive(true);
        UIEndTrackMenu.gameObject.SetActive(true);
        ScoreEndText.GetComponent<TextMeshPro>().text =
            TotalScore.ToString();

        //Disables Mesh from ScorePanel
        GameObject scorePanel = GameObject.Find("ScorePanel");
        scorePanel.GetComponent<Image>().enabled = false;
        scorePanel.transform.GetChild(0)
            .GetComponent<MeshRenderer>().enabled = false;
        scorePanel.transform.GetChild(1)
            .GetComponent<MeshRenderer>().enabled = false;

        //Change between ControllerModel and Weapons
        WeaponsLeft.SetActive(false);
        WeaponsRight.SetActive(false);
        ControllerModelLeft.SetActive(true);
        ControllerModelRight.SetActive(true);
    }
}
```

Το ScoreCanvas περιέχει ένα panel με κείμενο το οποίο εμφανίζει το τελικό σκορ του παίκτη μόλις τελειώσει το ρυθμικό μουσικό κομμάτι. Όπως φαίνεται στην συνάρτηση Update() του GameManagerMainGame, ελέγχεται κάθε καρέ αν εχει τελειώσει το κομμάτι, και οταν τελειώσει, γίνεται ενημέρωση του σκορ στην διεπαφή ScorePanel.

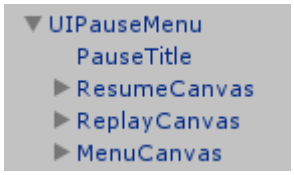
iv. UIPauseMenu

Το αντικείμενο UIPauseMenu ειναι το γκρουπ διεπαφών, το οποίο εμφανίζεται οταν ο παίκτης πατήσει το κουμπί «Μενού» που βρίσκεται πάνω στο χειριστήριο του.

Εικόνα 145 - Εικόνα απο το μενού του ρυθμικού μέρους του παιχνιδιού, όταν γίνεται παύση

Το Pause Menu δίνει στον παίκτη τρεις επιλογές για να προχωρήσει στο παιχνίδι. Η μια επιλογή ειναι με το Resume κουμπί, η άλλη ειναι το Replay κουμπί και η τελευταία το Menu κουμπί.

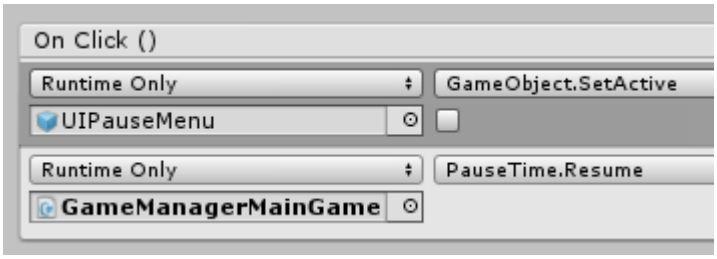
Εικόνα 146 - Αντικείμενα στην ιεραρχία της σκηνής, του ρυθμικού μέρους



Το UIPauseMenu περιέχει τέσσερα στοιχεία διεπαφής, απο τα οποία τα τρία ειναι κουμπιά.

- **ResumeCanvas**

Εικόνα 147 - Ιδιότητες του Resume Button



Στο ResumeCanvas υπάρχει ένα κουμπί που εκτελεί δυο λειτουργίες. Η μια ειναι να απενεργοποιήσει την διεπαφή του «UIPauseMenu» και η άλλη ειναι να εκτελέσει την συνάρτηση Resume() απο την κλάση PauseTime.

Εικόνα 148 - Συνάρτηση της κλάσης PauseTime

```
public void Resume()
{
    WeaponsLeft.SetActive(true);
    WeaponsRight.SetActive(true);
    ControllerModelRight.SetActive(false);
    ControllerModelLeft.SetActive(false);

    GetComponent<GameManagerMainGame>().Eyeraycast.SetActive(false);
    GameObject scorePanel = GameObject.Find("ScorePanel");
    scorePanel.GetComponent<Image>().enabled = true;
    scorePanel.transform.GetChild(0).GetComponent<MeshRenderer>().enabled = true;
    scorePanel.transform.GetChild(1).GetComponent<MeshRenderer>().enabled = true;


    GameObject[] planeObjects = GameObject.FindGameObjectsWithTag("PlaneGridTag");
    foreach (GameObject obj in planeObjects)
    {
        obj.GetComponent<MeshRenderer>().enabled = true;
    }

    gameObject.GetComponent<AudioSource>().UnPause();
    GameObject[] gameObjects = GameObject.FindGameObjectsWithTag("SpawnObject");
    foreach (GameObject obj in gameObjects)
    {
        obj.GetComponent<MeshRenderer>().enabled = true;
    }
}
```

Αυτή η συνάρτηση, κανει κάποιες αρχικοποιήσεις. Για αρχή κανει εναλλαγή των μοντέλων απο χειριστήρια σε όπλα. Έπειτα απενεργοποιεί την κουκίδα (pointer) του EyeRaycaster. Τέλος, εμφανίζει το Score Panel και μετα εμφανίζει και όλα τα αντικείμενα της σκηνής που έχουν tags “PlaneGridTag” και “SpawnObject”. Τα αντικείμενα με tag SpawnObject ειναι τα Streaming Assets του παιχνιδιού και τα αντικείμενα με tag PlaneGridTag ειναι τα αντικείμενα που συνθέτουν το πλαίσιο που εμφανίζεται μόνο στο Rifle Mode.

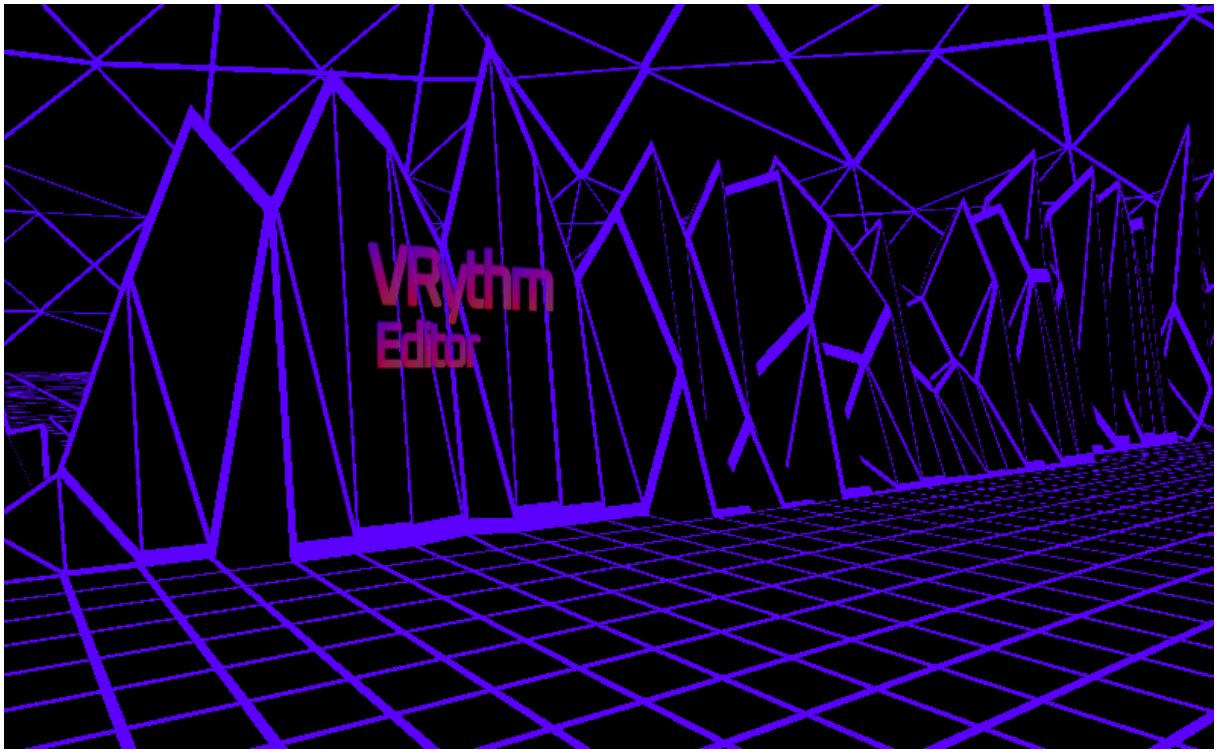
- **ReplayCanvas και MenuCanvas**

Τα κουμπιά των, ReplayCanvas και MenuCanvas, εκτελούν ακριβώς τις ιδίες διαδικασίες και εντολές, όπως στο Replay και το Menu του UIEndTrackMenu.

5.4 Διεπαφή επεξεργαστικού μέρους του παιχνιδιού

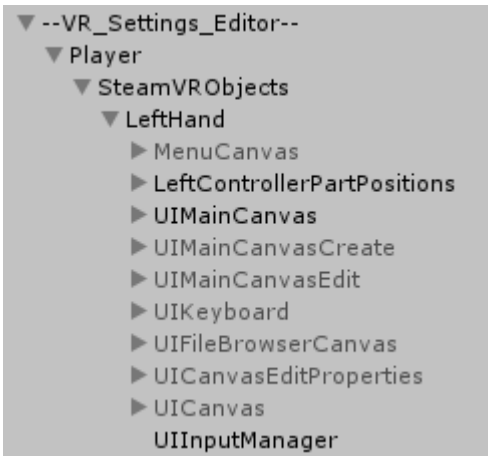
Στην παρακάτω εικόνα φαίνεται το περιβάλλον της σκηνής του επεξεργαστικού μέρους του παιχνιδιού. Σε αυτή την σκηνή ο παίκτης κάνει επιλογές που αφορούν την επεξεργασία του μουσικού κομματιού. Σε αυτή την σκηνή οι διεπαφές δεν ειναι σε μακρινή απόσταση όπως αυτές του ρυθμικού μέρους, αλλα πάνω στο χειριστήριο του. Έτσι οταν ο παίκτης θέλει να αλληλεπιδράσει με κάποια διεπαφή, θα πρεπει να κρατάει το χειριστήριο σταθερό σε μια απόσταση, ώστε να κανει μια επιλογή με την χρήση του δείκτη (EyeRaycaster), που αντιπροσωπεύει το κέντρο των ματιών του παίκτη.

Εικόνα 149 - Εικόνα της σκηνής του επεξεργαστικού μέρους του παιχνιδιού



Στην ιεραρχία αυτής της σκηνής φαίνονται τα αντικείμενα με τα στοιχεία διεπαφής που ειναι όλα τοποθετημένα σαν «παιδιά», κάτω απο το αριστερό χειριστήριο (VR Controller).

Εικόνα 150 – Αντικείμενα του VR, στην ιεραρχία της σκηνής, του επεξεργαστικού μέρους

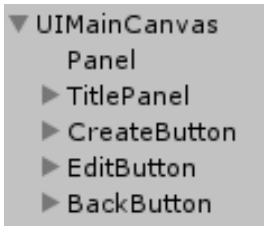


i. UIMainCanvas

Εικόνα 151 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού

Το UIMainCanvas ειναι η πρώτη διεπαφή που εμφανίζεται οταν εισέρχεται ο παίκτης σε αυτή την σκηνή. Ο παίκτης εχει δυο επιλογές σε αυτή την διεπαφή, είτε να δημιουργήσει εκ νέου μια πίστα με αντικείμενα του μουσικού κομματιού με το κουμπί ¨Create New Level¨, είτε να επεξεργαστεί μια ήδη δημιουργημένη, με το κουμπί ¨Edit Existing Level¨.

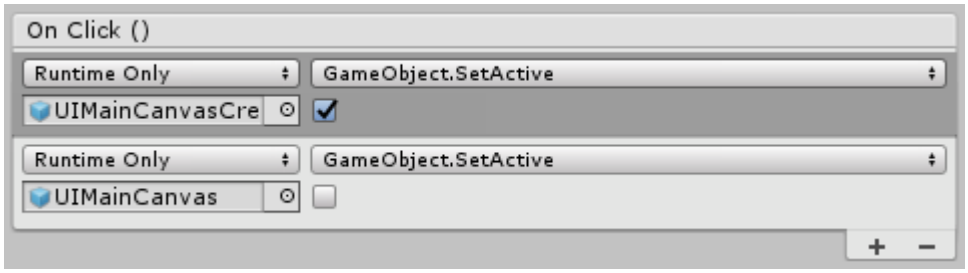
Εικόνα 152 - Αντικείμενα στην ιεραρχία της σκηνής, του επεξεργαστικού μέρους



Το UIMainCanvas περιέχει πέντε διεπαφές, όπου οι τρεις ειναι κουμπιά και οι άλλες δυο ειναι ένα πάνελ του παρασκήνιου και ένα πάνελ με το κείμενο του τίτλου.

- **CreateButton**

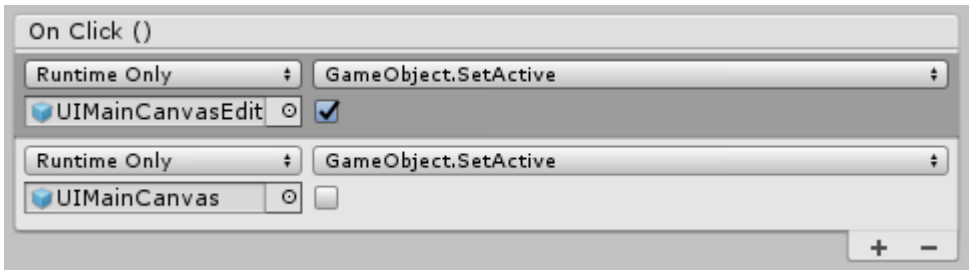
Εικόνα 153 - Ιδιότητες του Create Button



Το CreateButton ειναι υπεύθυνο για να προχωρήσει τον παίκτη στην επόμενη διεπαφή που χρειάζεται για να δημιουργήσει μια πίστα. Έτσι απλά κανει εναλλαγή διεπαφών, απο την τωρινή (UIMainCanvas), στην επόμενη (UIMainCanvasCreate), ενεργοποιώντας την μια και απενεργοποιώντας την άλλη.

- **EditButton**

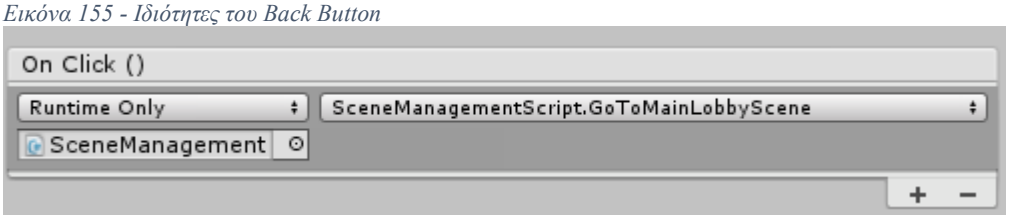
Εικόνα 154 - Ιδιότητες του Edit Button



Το EditButton εκτελεί και αυτή η διεπαφή την ίδια λειτουργία. Δηλαδή θα κανει εναλλαγή απο την τωρινή (UIMainCanvas) διεπαφή, στην επόμενη (UIMainCanvasEdit), ενεργοποιώντας την μια και απενεργοποιώντας την άλλη.

- **BackButton – Main Menu**

Εικόνα 155 - Ιδιότητες του Back Button



Το BackButton στην συγκεκριμένη διεπαφή ειναι υπεύθυνο για να κανει εναλλαγή μεταξύ των σκηνών. Έτσι πατώντας αυτό το κουμπί, μεταφέρει τον παίκτη στην προηγουμένη σκηνή που ειναι η σκηνή που εμφανίζεται στο ξεκίνημα του παιχνιδιού.

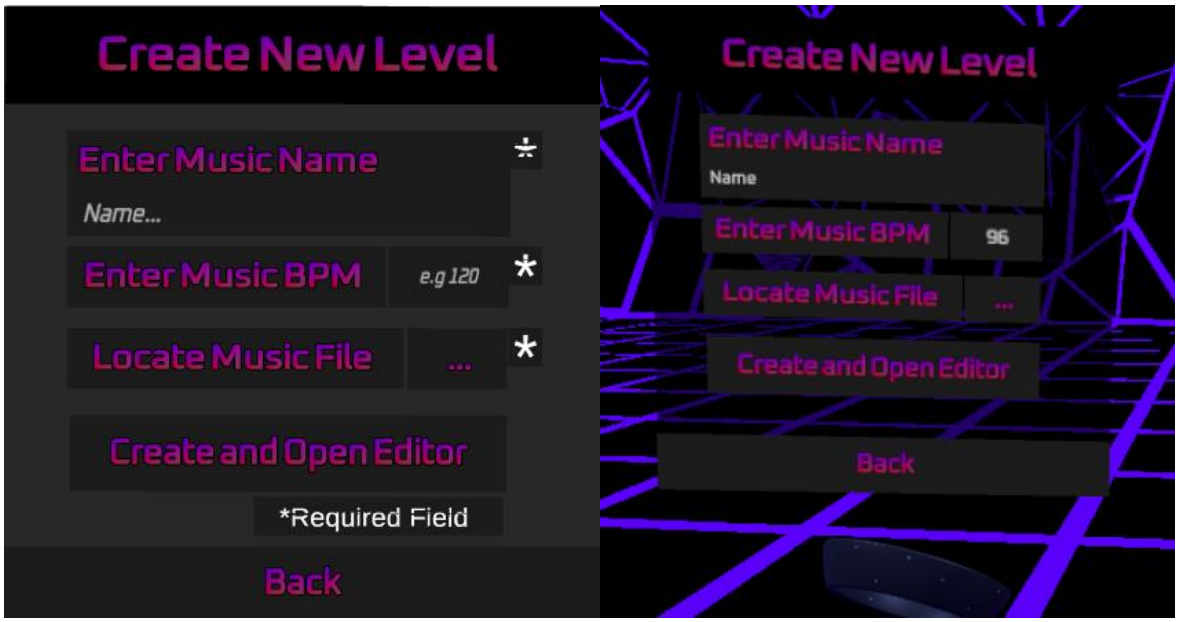
Εικόνα 156 - Συνάρτηση της κλάσης SceneManagementScript

```
public class SceneManagementScript : MonoBehaviour {

    public void GoToMainLobbyScene()
    {
        SceneManager.LoadScene("mainLobbyScene", LoadSceneMode.Single);
    }
```

Όπως φαίνεται στην παραπάνω εικόνα, αυτή την λειτουργία εκτελεί η συνάρτηση GoToMainLobbyScene της κλάσης SceneManagementScript. Έτσι με την χρήση της SceneManager φορτώνεται η αρχική σκηνή με όνομα «mainLobbyScene».

ii. UIMainCanvasCreate

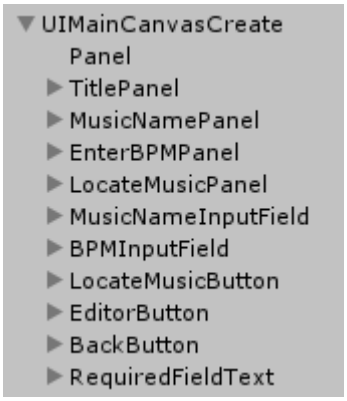
Εικόνα 157 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού



Το UIMainCanvasCreate ειναι η διεπαφή που λειτουργεί ως φόρμα ώστε να δώσει ο παίκτης τα απαραίτητα στοιχεία για το μουσικό κομμάτι και την πίστα. Αφού συμπληρώσει ο

παίκτης αυτά τα στοιχεία ενεργοποιείται το κουμπί “Create and Open Editor”, ώστε να ξεκινήσει ο παίκτης να την δημιουργία της πίστας.

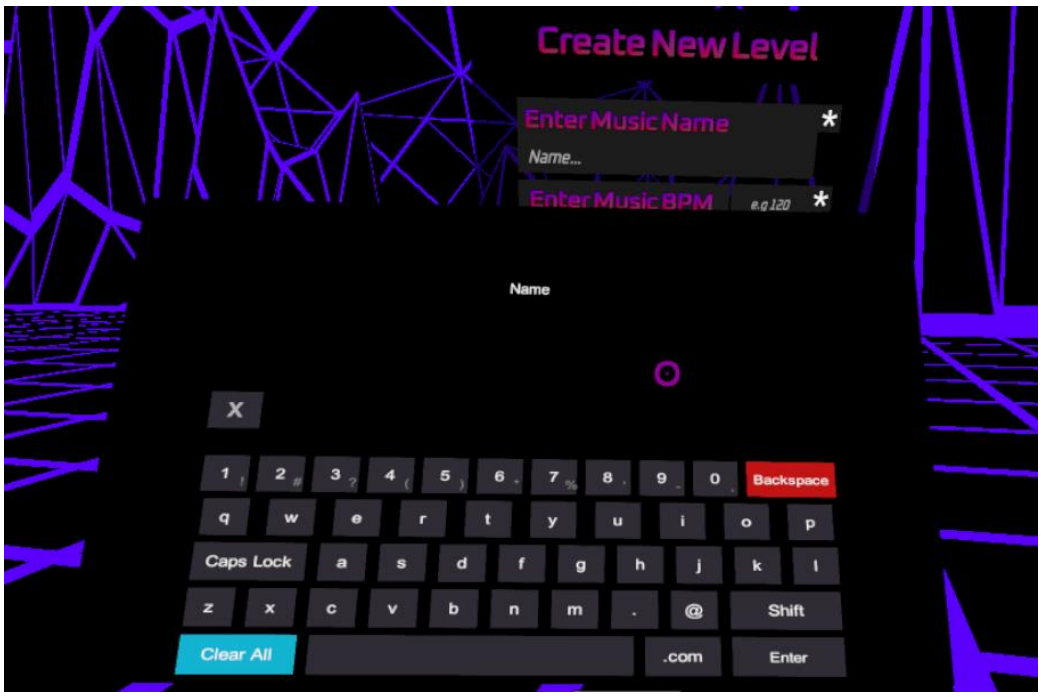
Εικόνα 158 - Αντικείμενα στην ιεραρχία της σκηνής, του επεξεργαστικού μέρους



Το UIMainCanvasCreate περιέχει έντεκα διεπαφές. Ένα πάνελ ως παρασκήνιο (Panel), ένα πάνελ που περιέχει τον τίτλο της διεπαφής (TitlePanel), τρία πάνελ κειμένου που περιέχουν τίτλους (MusicNamePanel – EnterBPMPanel - LocateMusicPanel), δυο πάνελ για την εισαγωγή κειμένου (MusicNameInputField - BPMInputField), τρία κουμπιά (LocateMusicButton – EditorButton - BackButton) και διεπαφές κειμένου (RequiredFieldText) για την προειδοποίηση εισαγωγής στοιχείων.

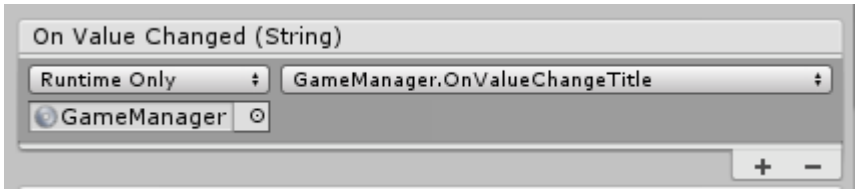
- **MusicNameInputField – Input Field**

Εικόνα 159 - Εικόνα μέσα απο το παιχνίδι, στην χρήση του VRKeyboard



Το MusicNameInputField ειναι ένα στοιχείο διεπαφής τύπου Input Field. Δηλαδή εισαγωγής κειμένου, απο τον χρήστη μέσω του πληκτρολογίου. Αυτό χρησιμοποιείται για την εισαγωγή του ονόματος του μουσικού κομματιού απο τον παίκτη. Σε αυτή την περίπτωση, λογο της αποκλειστικής χρήσης VR συσκευών, γίνεται χρήση μιας εξωτερικής επέκτασης το VR Keyboard, το οποιο όπως φαίνεται στην παραπάνω εικόνα λειτουργεί ως εικονικό πληκτρολόγιο μέσα στο παιχνίδι. Έτσι οταν ο παίκτης θέλει να γράψει κάποιο κείμενο σε αυτό το Input Field, εμφανίζεται το VR Keyboard και δίνει την δυνατότητα στον παίκτη να συμπληρώσει ό,τι κείμενο θελήσει.

Εικόνα 160 - Ιδιότητες του MusicNameInputField



Απο προεπιλογή η Unity χρησιμοποιεί την συνάρτηση της OnValueChange() που εκτελείται κάθε φορά που γίνεται αλλαγή στο κείμενο (είτε στην εισαγωγή καινούργιου γράμματος, είτε στην διαγραφή αυτού).

Εικόνα 161 - Συνάρτηση της κλάσης GameManager

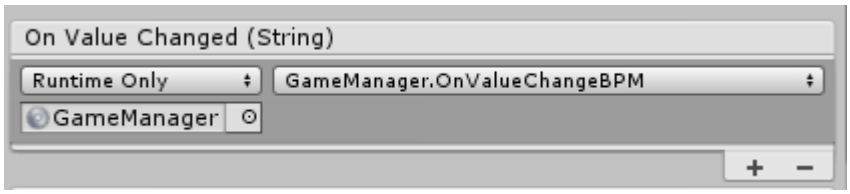
```
public void OnValueChangeTitle()
{
    SongTitle = UICanvasCreateSongTitle.GetComponent<InputField>().text;
}
```

Έτσι κάθε φορά που γίνεται κάποια αλλαγή, μεχρι να τελειώσει ο παίκτης την συμπλήρωση του επιθυμητού αυτού κειμένου, γίνεται αποθήκευση του, στην μεταβλητή SongTitle του Game Manager μέσω της συνάρτησης του, OnValueChangeTitle().

- **BPMInputField – Input Field**

Το BPMInputField ειναι το Input Field υπεύθυνο για την εισαγωγή του ρυθμού αναπαραγωγής του μουσικού κομματιού (BPM – Beats Per Minute). Όπως και στο Name Input Field έτσι και σε αυτό το Input Field γίνεται χρήση του VR Keyboard για την εισαγωγή κειμένου. Το συγκεκριμένο Input Field επιτρέπει μόνο την εισαγωγή αριθμών (καθώς το BPM ειναι συνήθως ένας διψήφιος ή τριψήφιος αριθμός).

Εικόνα 162 - Ιδιότητες του BPMInputField



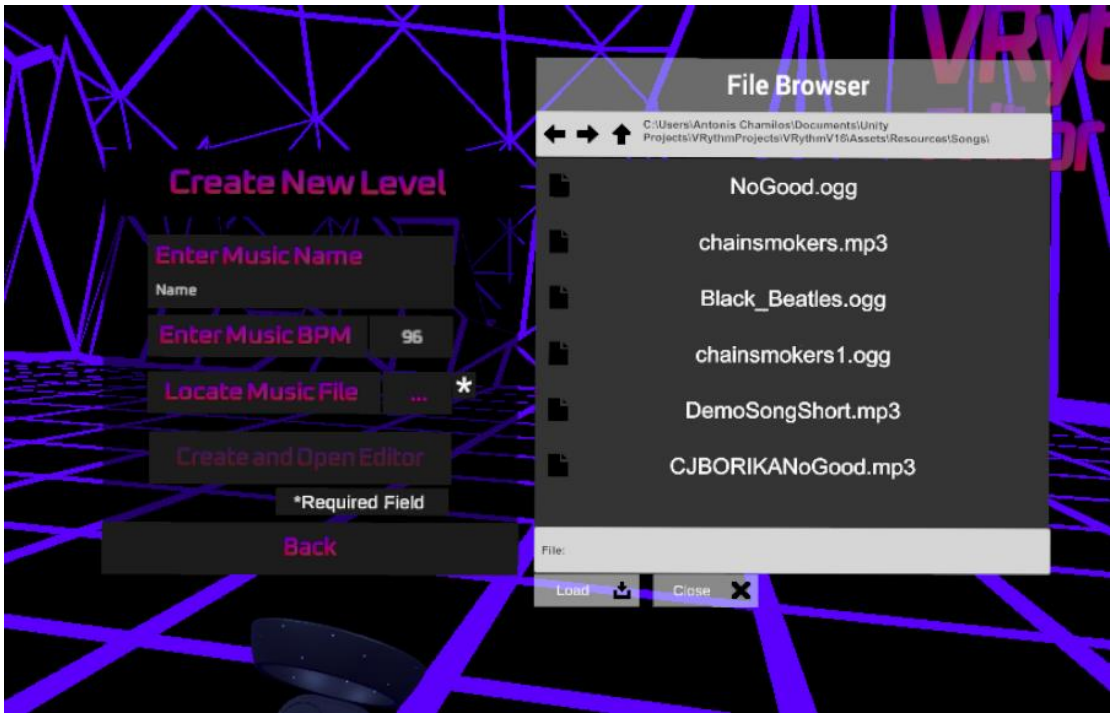
Εικόνα 163 - Συνάρτηση της κλάσης GameManager

```
public void OnValueChangeBPM()
{
    BeatsPerMinute = float.Parse(UICanvasCreateSongBPM.GetComponent<InputField>().text);
}
```

Και αυτό το Input Field λειτουργεί με το ίδιο τροπο. Δηλαδή σε κάθε αλλαγή κειμένου – αριθμού γίνεται αποθήκευση αυτού, στον Game Manager του παιχνιδιού. Ουσιαστικά εκτελείται η συνάρτηση OnValueChangeBPM() του Game Manager και αποθηκεύει στην μεταβλητή BeatsPerMinute τον αριθμό που εισήγαγε ο παίκτης, ως μια float τιμής.

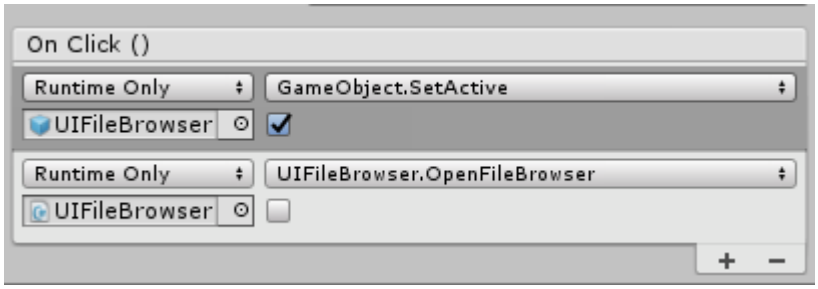
- **LocateMusicButton**

Εικόνα 164 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού



Το LocateMusicButton ειναι το κουμπί με τις τελείες (...) που ειναι υπεύθυνο για να εμφανίσει το File Browser ώστε να επιλέξει ο παίκτης ένα μουσικό κομμάτι.

Εικόνα 165 - Ιδιότητες του Locate Music Button



Έτσι οταν πατηθεί το κουμπί, εκτελείται η OnClick() και ενεργοποιείται το File Browser και εκτελείται η συνάρτηση OpenFileBrowser() όπου ορίζεται ο τύπος του File Browser που θα εμφανιστεί. Οταν το checkbox δεν ειναι ενεργοποιημένο σημαίνει ότι η λειτουργία του ειναι για να φορτώσει ένα αρχείο. Σε διαφορετική περίπτωση θα άνοιγε το File Browser που ειναι υπεύθυνο για την αποθήκευση κάποιου αρχείου. Στο συγκεκριμένο παιχνίδι χρησιμοποιείται το File Browser μόνο για την φόρτωση αρχείων.

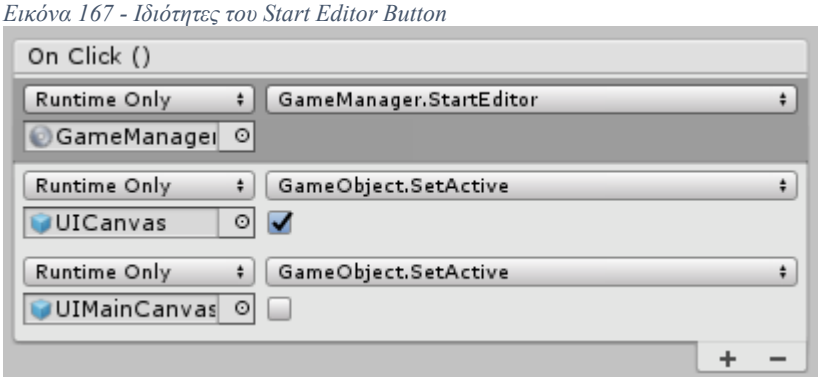
Εικόνα 166 - Συνάρτηση της κλάσης UIFileBrowser

```
// Open a file browser to save and load files
protected void OpenFileBrowser(FileBrowserMode fileBrowserMode)
{
    // Create the file browser and name it
    GameObject fileBrowserObject = Instantiate(FileBrowserPrefab, transform);
    fileBrowserObject.name = "FileBrowser";
    // Set the mode to save or load
    FileBrowser fileBrowserScript = fileBrowserObject.GetComponent<FileBrowser>();
    fileBrowserScript.SetupFileBrowser(
        PortraitMode ? ViewMode.Portrait : ViewMode.Landscape);

    if (fileBrowserMode == FileBrowserMode.Save)
    {
        fileBrowserScript.SaveFilePanel("DemoText", FileExtensions);
        // Subscribe to OnFileSelect event (call SaveFileUsingPath using path)
        fileBrowserScript.OnFileSelect += SaveFileUsingPath;
    }
    else
    {
        fileBrowserScript.OpenFilePanel(FileExtensions);
        // Subscribe to OnFileSelect event (call LoadFileUsingPath using path)
        fileBrowserScript.OnFileSelect += LoadFileUsingPath;
    }
}
```

- **EditorButton**

Το EditorButton ειναι ένα κουμπί υπεύθυνο για την εκκίνηση του επεξεργαστικού περιβάλλοντος, που ειναι απαραίτητο για να μπορεί ο παίκτης να δημιουργήσει μια πίστα για το μουσικό κομμάτι που επέλεξε.

Εικόνα 167 - Ιδιότητες του Start Editor Button



Οταν πατηθεί το κουμπί και εκτελεστεί η συνάρτηση OnClick() γίνονται κάποιες αλλαγές και αρχικοποιήσεις. Ουσιαστικά απενεργοποιείται η παρόν διεπαφή (UIMainCanvasCreate), ενεργοποιείται η διεπαφή UICanvas και γίνεται και η εκτέλεση της συνάρτησης StartEditor() του Game Manager.

Εικόνα 168 - Συνάρτηση της κλάσης GameManager

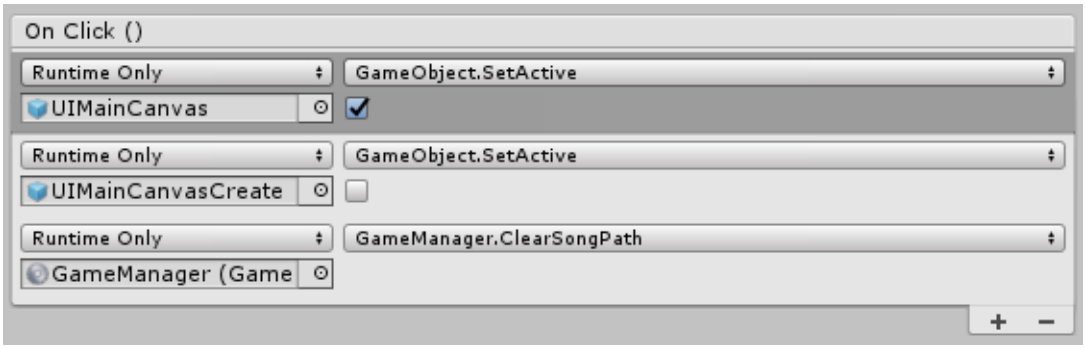
```
public void StartEditor()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
    Invoke("ExecuteAfterTime", 0.5f);
}

public void ExecuteAfterTime()
{
    Instantiate(StandardObjects);

    UICanvasEditSongTitle.GetComponent<InputField>().text =
        SongTitle;
    UICanvasEditSongBPM.GetComponent<InputField>().text =
        BeatsPerMinute.ToString();

    //Updating Properties
    InitAllObjectsProperties();
}
```

Η StartEditor() εκτελεί κάποιες αρχικοποιήσεις. Για αρχή καταστρέφει τυχόν αντικείμενα με tag «StandardObjectsTag», που ίσως είχαν δημιουργηθεί προηγουμένως. Στην συνέχεια αρχικοποιεί τα StandardObjects που ειναι τα αντικείμενα υπεύθυνα για το περιβάλλον της σκηνής του επεξεργαστικού μέρους. Ενημερώνει τις μεταβλητές του Game Manager για τον τίτλο του μουσικού κομματιού και τον ρυθμό αναπαραγωγής του (bpm) και τέλος ενημερώνει τις ιδιότητες όλων των αντικειμένων σε περίπτωση που είχαν διαφορετικές ιδιότητες απο προηγούμενη συνέδρια.

- **BackButton**

Το BackButton ευθύνεται για την εναλλαγή των διεπαφών, και να γίνει μετάβαση στην προηγούμενη διεπαφή. Οταν πατηθεί η OnClick() γίνεται απενεργοποίηση της παρούσας διεπαφής (UIMainCanvasCreate) και ενεργοποιείται η προηγούμενη διεπαφή (UIMainCanvas).

Εικόνα 169 - Ιδιότητες του Back Button



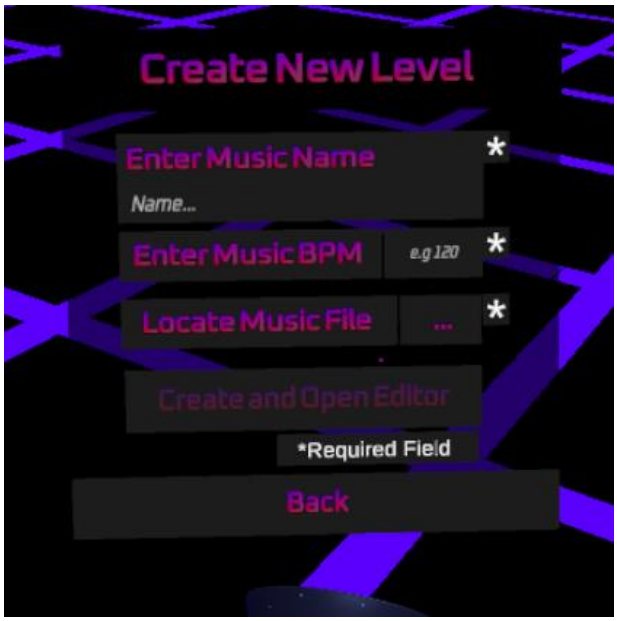
Επίσης εκτελείται η συνάρτηση ClearSongNamePath() του Game Manager η οποία ειναι υπεύθυνη να καθαρίσει το κείμενο της τοποθεσίας του μουσικού κομματιού, στην περίπτωση που ο παίκτης είχε ήδη φορτώσει κάποιο. Αυτό γίνεται για να μην δημιουργούνται προβλήματα με το παιχνίδι αργότερα.

Εικόνα 170 - Συνάρτηση της κλάσης GameManager

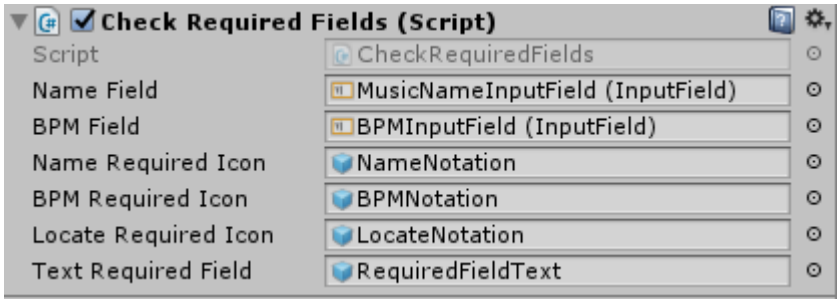
```
public void ClearSongPath()
{
    CurrentSongNamePath = "";
}
```

- **Required Fields**

Εικόνα 171- Εικόνα απο το μενού, για την δημιουργία πίστας



Εικόνα 172 - Ιδιότητες και πεδία της κλάσης CheckRequiredFields



Η κλάση CheckRequiredField δημιουργήθηκε για να ελέγχονται οι περιορισμοί της διεπαφής που ειναι απαραίτητοι για την σωστή λειτουργία των διεπαφών και γενικά του παιχνιδιού. Τα βασικά πεδία που πρεπει να ελεγχτούν ειναι το MusicNameInputField και το BPMInputField. Αυτοί οι έλεγχοι γίνονται με βάση το παρακάτω Script.

Εικόνα 173 - Συνάρτηση της κλάσης CheckRequiredFields

```
//Check Name Field
if (!NameField.text.Equals(""))
{
    NameRequiredIcon.SetActive(false);
}
else
{
    NameRequiredIcon.SetActive(true);
}
```

Η παραπάνω συνθήκη ελέγχει αν το πεδίο του ονόματος του μουσικού κομματιού (NameField) ειναι κενό. Αν ειναι κενό τότε εμφανίζεται ένα εικονίδιο – αστεράκι διπλά στο πεδίο αυτό.

Εικόνα 174 - Συνάρτηση της κλάσης CheckRequiredFields

```
//Check BPM Field
if (!BPMField.text.Equals(""))
{
    BPMRequiredIcon.SetActive(false);
}
else
{
    BPMRequiredIcon.SetActive(true);
}
```

Αυτή η συνθήκη ελέγχει αντίστοιχα αν το πεδίο του ρυθμού αναπαραγωγής του μουσικού κομματιού ειναι κενό. Στην περίπτωση που ειναι κενό, εμφανίζεται ένα εικονίδιο – αστεράκι διπλά στο πεδίο αυτό.

Εικόνα 175 - Συνάρτηση της κλάσης CheckRequiredFields

```
//Check Locate File Field
if (!GameObject.Find("GameManager").GetComponent<GameManager>()
    .CurrentSongNamePath.Equals(""))
{
    LocateRequiredIcon.SetActive(false);
}
else
{
    LocateRequiredIcon.SetActive(true);
}
```

Η παραπάνω συνθήκη ελέγχει αν το κείμενο – μονοπάτι της τοποθεσίας του μουσικού κομματιού ειναι κενό. Αν ειναι κενό σημαίνει ότι ο παίκτης δεν επέλεξε να φορτώσει κάποιο μουσικό κομμάτι απο το File Browser. Οπότε αν δεν εχει φορτωθεί κάποιο μουσικό κομμάτι εμφανίζεται και σε αυτή την περίπτωση ένα εικονίδιο - αστεράκι διπλά απο το κουμπί που ειναι υπεύθυνο για την εμφάνιση του File Browser.

Εικόνα 176 - Συνάρτηση της κλάσης CheckRequiredFields

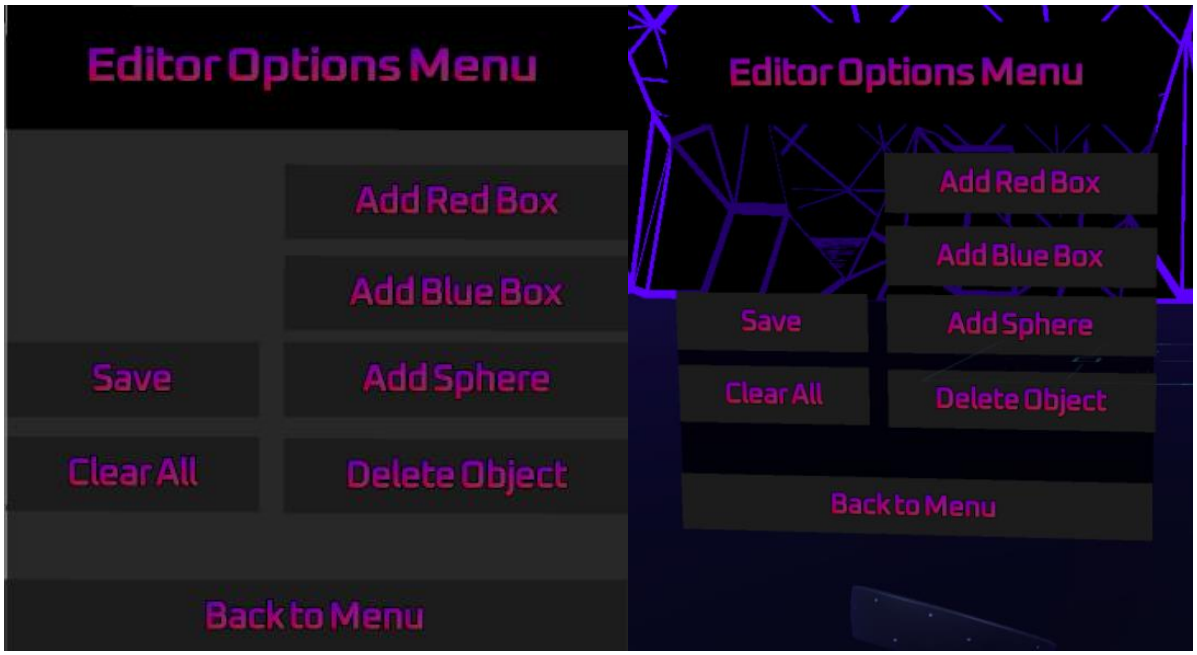
```
//Check Any Field
if (
!BPMField.text.Equals("") &&
!NameField.text.Equals("") &&
!GameObject.Find("GameManager")
    .GetComponent<GameManager>()
    .CurrentSongNamePath.Equals("")
){
    TextRequiredField.SetActive(false);
    GetComponent<Button>().interactable = true;
}
else
{
    TextRequiredField.SetActive(true);
    GetComponent<Button>().interactable = false;
}
```

Αυτή η συνθήκη ελέγχει ταυτόχρονα όλες τις παραπάνω συνθήκες αν αληθεύουν, ώστε στην περίπτωση που έστω και μια να μην ισχύει (δηλαδή να ειναι κενό το πεδίο της), να εμφανίζεται το κείμενο που υποδεικνύει τι σημαίνει το εικονίδιο αστεράκι που γραφεί «*Required Fields». Επίσης αν υπάρχει κενό πεδίο το κουμπί EditorButton με όνομα Create and Open Editor απενεργοποιείται, έτσι ώστε να μην μπορεί ο παίκτης να επεξεργαστεί κάποια πίστα χωρίς να τηρούνται οι προϋποθέσεις της διεπαφής.

iii. UICanvas

Το UICanvas ειναι η διεπαφή που εμφανίζεται οταν ο παίκτης εισέρχεται στο μέρος της επεξεργασίας (ή ξεκάνει η επεξεργασία) της πίστας του μουσικού κομματιού, σε αυτή την σκηνή. Ο παίκτης εχει επτά επιλογές σε αυτή την διεπαφή.

Εικόνα 177 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού



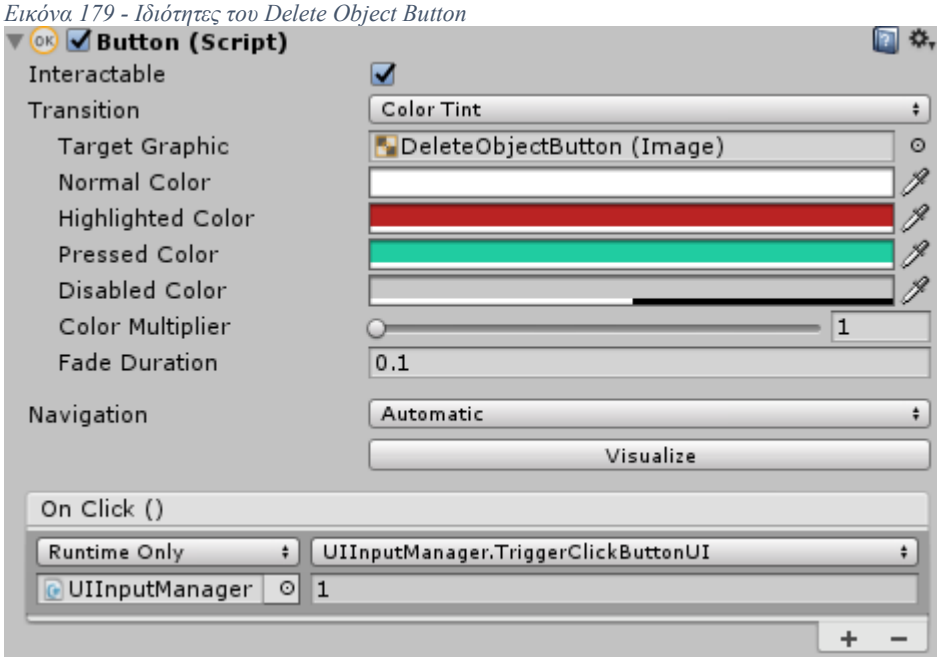
Οι βασικές επιλογές χωρίζονται σε δυο είδη, στα κουμπιά που ειναι υπεύθυνα για την αλληλεπίδραση του παίκτη με τα αντικείμενα Streaming Assets στο περιβάλλον της σκηνής και σε αυτά που εκτελούν λειτουργίες για τις πληροφορίες των δημιουργημένων αντικείμενων. Η πρώτη κατηγορία αποτελείται απο τα κουμπιά Add Red Box, Add Blue Box, Add Sphere και Delete Object. Ενώ η δεύτερη αποτελείται απο δυο κουμπιά το Save και το Clear All.

Εικόνα 178 - Αντικείμενα στην ιεραρχία της σκηνής, του επεξεργαστικού μέρους

Στην ιεραρχία της διεπαφής, όλα τα αντικείμενα διεπαφής ειναι κουμπιά, εκτός απο το Title Panel που περιέχει το κείμενο του Τίτλου, και το Panel που ειναι το παρασκήνιο (Background) του UICanvas.

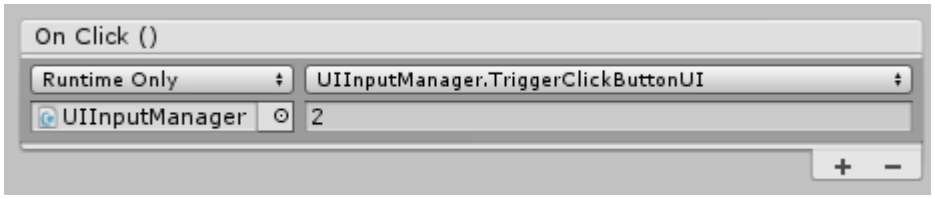
- **DeleteObjectButton**

Εικόνα 179 - Ιδιότητες του Delete Object Button



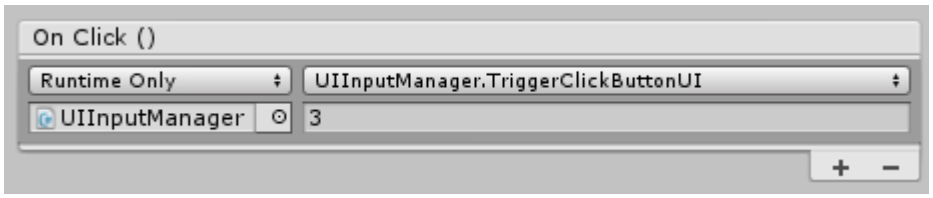
- **RedBoxButton**

Εικόνα 180 - Ιδιότητες του Red Box Button



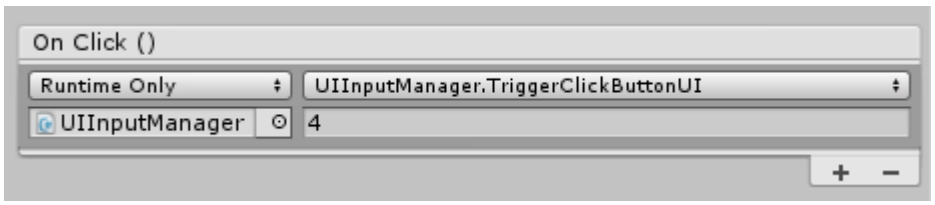
- **BlueBoxButton**

Εικόνα 181 - Ιδιότητες του Blue Box Button



- **SphereButton**

Εικόνα 182 - Ιδιότητες του Sphere Button



Εικόνα 183 - Συνάρτηση της κλάσης UIInputManager

```
//Method Events
public void TriggerClickButtonUI(int sCurrentSelected)
{
    CurrentSelected = sCurrentSelected;
}
```

Όλα τα παραπάνω κουμπιά οταν πατηθούν εκτελούν την συνάρτηση TriggerClickButtonUI() της κλάσης UIInputManager, όπου γίνεται ανάθεση ενός ακέραιου στην μεταβλητή CurrentSelected που αντιπροσωπεύει την κάθε λειτουργία που πρόκειται να εκτελεστεί αργότερα απο άλλη συνάρτηση.
Έτσι οταν πατηθεί το DeleteObjectButton το CurrentSelected θα γίνει ίσο με τον αριθμό 1.

- Αν πατηθεί το RedBoxButton θα γίνει ίσο με τον αριθμό 2.
- Αν πατηθεί το BlueBoxButton θα γίνει ίσο με τον αριθμό 3.
- Και τέλος αν πατηθεί το SphereButton το CurrentSelected θα γίνει ίσο με τον αριθμό 4.

Εικόνα 184 - Συνάρτηση της κλάσης UIInputManager

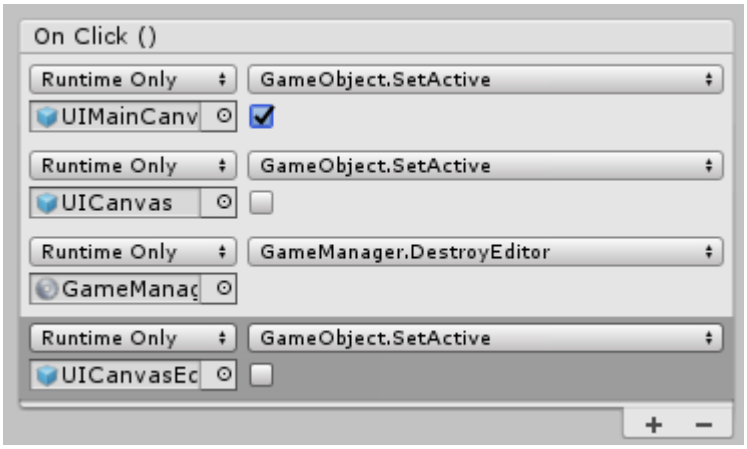
```
public static void DoSelectedFunction(
    PointerEventArgs hoverEvent,
    GameObject obj,
    Transform currentHoverObjectTransform) {

    if (UIInputManager.CurrentSelected.Equals(1))
    {
        UIInputManager.DeleteObject(hoverEvent);
    }
    else if (UIInputManager.CurrentSelected.Equals(2))
    {
        UIInputManager.AddRedBox(obj, currentHoverObjectTransform);
    }
    else if (UIInputManager.CurrentSelected.Equals(3))
    {
        UIInputManager.AddBlueBox(obj, currentHoverObjectTransform);
    }
    else if (UIInputManager.CurrentSelected.Equals(4))
    {
        UIInputManager.AddSphere(obj, currentHoverObjectTransform);
    }
}
```

Αφού εχει γίνει ήδη ο ορισμός της μεταβλητής CurrentSelected με ένα απο τους παραπάνω αριθμούς, οταν ο παίκτης πατήσει την σκανδάλη απο το χειριστήριο και στοχεύει με τον Ray του χειριστηρίου πάνω στην επιφάνεια αφής εκτελείται μεταξύ άλλων εντολών και η συνάρτηση DoSelectedFunction(). Η DoSelectedFunction εκτελεί λειτουργίες αναλόγως με το κουμπί που είχε πατηθεί απο τον παίκτη προηγουμένως. Έτσι αν προηγουμένως ο παίκτης είχε πατήσει απο την διεπαφή το DeleteObjectButton που εχει αριθμό 1, θα εκτελεστεί η συνάρτηση

UIInputManager.DeleteObject(). Το ίδιο συμβαίνει αντίστοιχα για όλα τα παραπάνω κουμπιά, όπως φαίνεται στην παραπάνω συνάρτηση.

- **ExitButton**

Εικόνα 185 - Ιδιότητες του Exit Button



Το ExitButton οταν πατιέται εκτελείται η OnClick() και εκτελεί τις λειτουργίες που φαίνονται στην παραπάνω εικόνα. Ουσιαστικά κανει εναλλαγή στις διεπαφές και μεταφέρει τον παίκτη σε προηγούμενη διεπαφή, την UIMainCanvas. Ενεργοποιεί την UIMainCanvas και απενεργοποιεί τις διεπαφές UICanvas και UICanvasEditProperties. Επίσης εκτελεί και την συνάρτηση DestroyEditor του Game Manager.

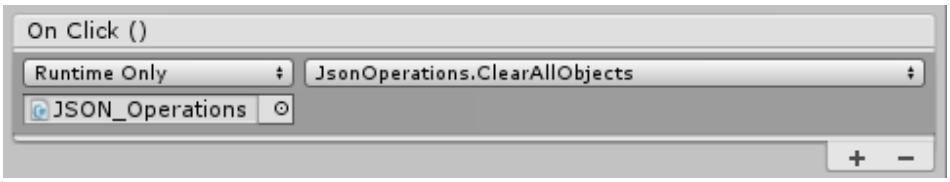
Εικόνα 186 - Συνάρτηση της κλάσης GameManager

```
public void DestroyEditor()
{
    GetComponent<AudioSource>().Stop(); //Stop the music before destroying
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
}
```

Η συνάρτηση DestroyEditor() ουσιαστικά σταματάει την αναπαραγωγή του μουσικού κομματιού και καταστρέφει τα αντικείμενα με ετικέτα «StandardObjectsTag», που δημιουργήθηκαν για την επεξεργασία ή δημιουργία μιας πίστας.

- **ClearAll**

Εικόνα 187 - Ιδιότητες του Clear All Button



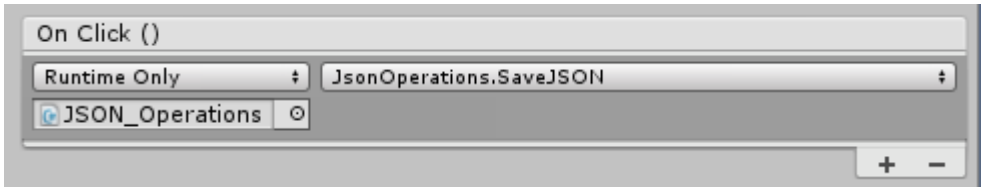
Το κουμπί ClearAll εκτελεί την συνάρτηση ClearAllObjects() της κλάσης JSON_Operations. Οι λειτουργίες που εκτελεί αυτή η συνάρτηση, ειναι η εύρεση του «πατέρα» (στην ιεραρχία της σκηνής) των δημιουργημένων αντικειμένων (Steaming Assets), και η διαγραφή των παιδιών του (CreatedObjects), απο την σκηνή. Και τέλος η διαγραφή των πληροφοριών - ιδιοτήτων αυτών των αντικειμένων απο την λίστα αντικειμένων που υπάρχει στην κλάση AllObjects.

Εικόνα 188 - Συνάρτηση της κλάσης UIInputManager

```
//Deleting all Objects Created
public void ClearAllObjects()
{
    GameObject createdObjects = GameObject.Find("CreatedObjects");
    //Deleting Physical Objects
    foreach (Transform child in createdObjects.transform)
    {
        GameObject.Destroy(child.gameObject);
    }

    //Destroy Objects as data from allObjects
    GameManager.allObjects.objects.Clear();
}
```

- **Save**

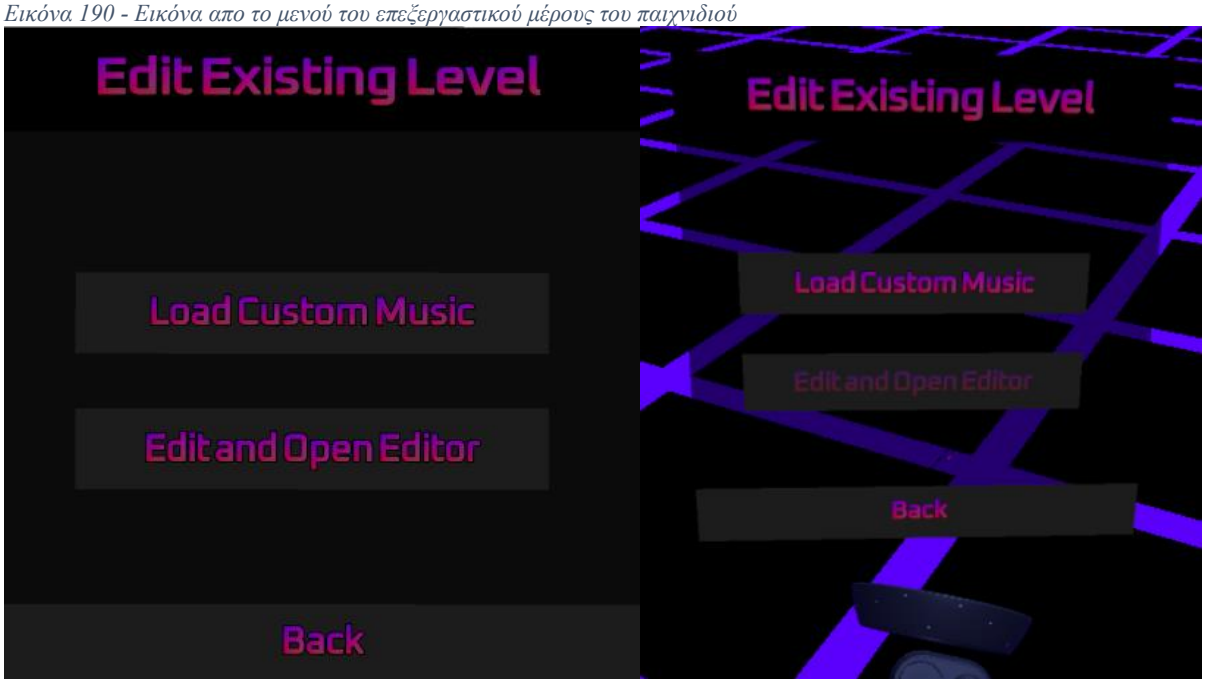
Εικόνα 189 - Ιδιότητες του Save Button



Το κουμπί Save εκτελεί την συνάρτηση SaveJSON της κλάσης JSON_Operations. Η συνάρτηση SaveJSON() ειναι υπεύθυνη για την αποθήκευση των JSON αρχείων που περιέχουν ιδιότητες και πληροφορίες για το μουσικό κομμάτι και τα αντικείμενα – Streaming Assets που δημιουργήθηκαν για αυτό.

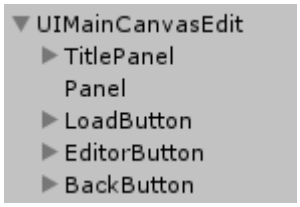
iv. UIMainCanvasEdit

Εικόνα 190 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού



Το UIMainCanvasEdit ειναι η διεπαφή που εμφανίζεται οταν ο παίκτης επιλέξει απο προηγούμενη διεπαφή (UIMainCanvas) το κουμπί Edit Existing Level. Αυτή η διεπαφή ειναι αναγκαία για να μπορέσει ο παίκτης να επιλέξει πιο μουσικό κομμάτι θέλει να επεξεργαστεί, πριν ξεκινήσει η ίδια η διαδικασία της επεξεργασίας. Έτσι ο παίκτης μπορεί να κανει τρεις επιλογές σε αυτή την διεπαφή.

Εικόνα 191 - Αντικείμενα στην ιεραρχία της σκηνής, του επεξεργαστικού μέρους

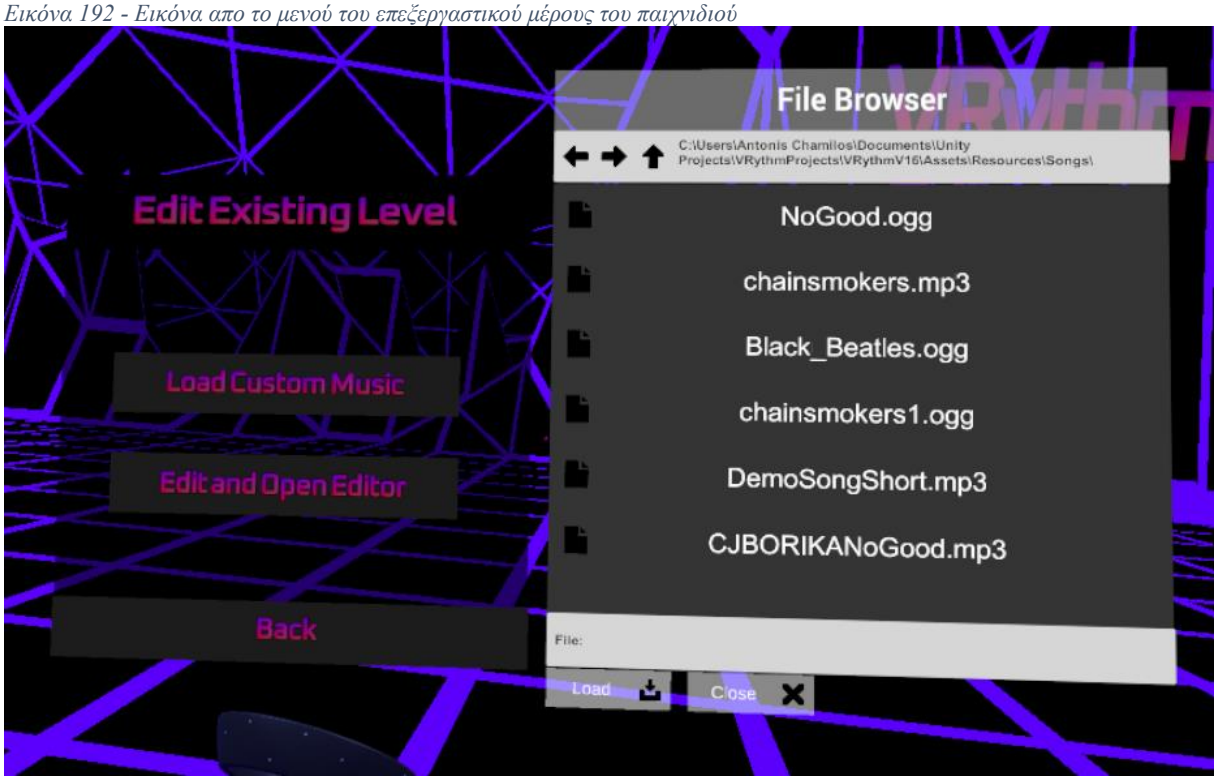


Όπως φαίνεται στην παραπάνω εικόνα, στην ιεραρχία της διεπαφής υπάρχουν πέντε αντικείμενα διεπαφών. Τα δύο ειναι, το πάνελ παρασκήνιου της διεπαφής, και το πάνελ για τον τίτλο της διεπαφής. Τα υπόλοιπα στοιχεία ειναι κουμπιά, που το κάθε ένα εκτελεί κάποιες λειτουργίες.

- **LoadButton**

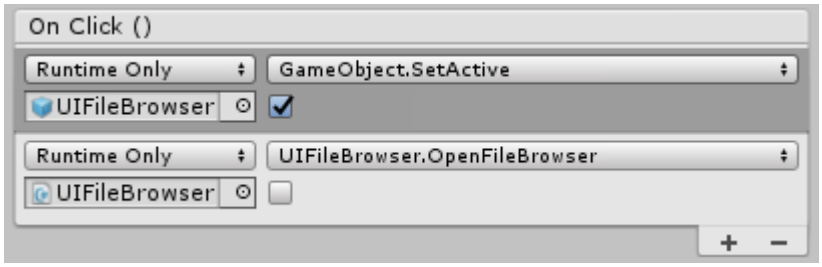
Το LoadButton ειναι το κουμπί με όνομα LoadCustomMusic, και ειναι υπεύθυνο για την εμφάνιση της διεπαφής File Browser ώστε να επιλέξει ο παίκτης , μουσικό κομμάτι για να φορτώσει.

Εικόνα 192 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού



Έτσι οταν πατηθεί το LoadButton θα εκτελεστεί η OnClick(), που εκτελεί δυο λειτουργίες. Ενεργοποιεί το UIFileBrowser και εκτελεί την συνάρτηση OpenFileBrowser() θέτοντας την λειτουργία Load για την φόρτωση αρχείων.

Εικόνα 193 - Ιδιότητες του LoadCustom Music Button

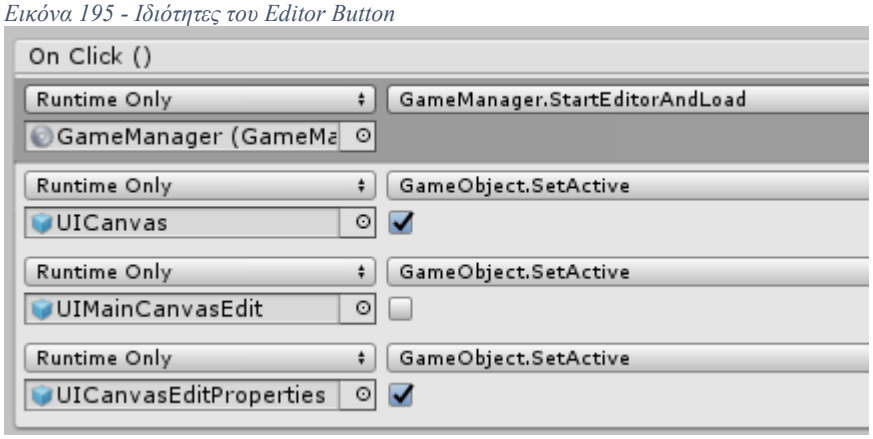


Εικόνα 194 - Συνάρτηση της κλάσης FileBrowser

```
// Open the file browser using boolean parameter so it can be called in GUI
public void OpenFileBrowser(bool saving)
{
    OpenFileBrowser(saving ? FileBrowserMode.Save : FileBrowserMode.Load);
}
```

- **Editor Button**

Εικόνα 195 - Ιδιότητες του Editor Button



Το EditorButton ειναι το κουμπί με όνομα Edit and Open Editor, το οποίο εκτελεί κάποιες λειτουργίες με την OnClick(). Κανει εναλλαγή στις διεπαφές. Απενεργοποιεί την τωρινή διεπαφή (UIMainCanvasEdit) και ενεργοποιεί τις επόμενες που ειναι η UICanvas και η UICanvasEditProperties. Η βασική λειτουργία που εκτελεί, ειναι η εκκίνηση του επεξεργαστικού μέρους του παιχνιδιού, όπου δίνεται η δυνατότητα στον παίκτη να δημιουργήσει μια πίστα για κάποιο μουσικό κομμάτι της επιλογής του.

Εικόνα 196 - Συνάρτηση της κλάσης GameManager

```
public void StartEditorAndLoad()
{
    Destroy(GameObject.FindGameObjectWithTag("StandardObjectsTag"));
    Invoke("ExecuteAfterTimeJsonLoad", 0.5f);
}

public void ExecuteAfterTimeJsonLoad()
{
    Instantiate(StandardObjects);

    UICanvasEditSongTitleProperties.GetComponent<InputField>().text
        = SongTitle;
    UICanvasEditSongBPMProperties.GetComponent<InputField>().text
        = BeatsPerMinute.ToString();

    GameObject.Find("JSON_Operations").GetComponent<JsonOperations>()
        .CreateLoadedObjects();
}
```

Έτσι εκτελείται η συνάρτηση StartEditorAndLoad() του Game Manager η οποία αρχικοποιεί - δημιουργεί τα StandardObjects που ειναι απαραίτητα για την σκηνή.
Παίρνει τις ιδιότητες του μουσικού κομματιού (Όνομα και ρυθμό αναπαραγωγής), που ειναι αποθηκευμένα στην Game Manager και τις μεταφέρει στα Input Field της επόμενης διεπαφής (UICanvasEditProperties), ώστε να μπορεί να δει ο παίκτης αυτές τις ιδιότητες και να τις αλλάξει αν χρειαστούν διόρθωση.

Τέλος εκτελεί την συνάρτηση CreateLoadedObjects() της κλάσης JsonOperations, ώστε να δημιουργήσουν όλα τα Streaming Assets στην σκηνή, που είχαν δημιουργηθεί απο

προηγούμενη συνεδρία του παίκτη απο την Create New Level, που ειναι για την εκ νέου δημιουργία πίστας.

- **CheckLoadedEditor**

Εικόνα 197 - Κλάση CheckLoadedTrackEditor

Η κλάση CheckLoadedTrackEditor ειναι υπεύθυνη για να ελέγχει συνεχόμενα (στην Update), τις απαραίτητες ιδιότητες του παιχνιδιού και να ενεργοποιεί η να απενεργοποιεί το κουμπί που ξεκινάει την επεξεργασία της πίστας, προκειμένου να μην υπαρχουν προβλήματα στο παιχνίδι.

Εικόνα 198 - Συνάρτηση της κλάσης CheckLoadedTrackEditor

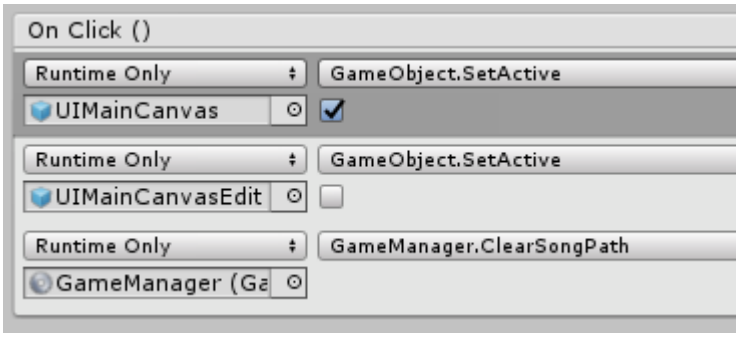
```
// Update is called once per frame
void Update()
{
    if (!gameManager.GetComponent<GameManager>().CurrentSongNamePath.Equals("")
        && !gameManager.GetComponent<GameManager>().SongTitle.Equals("")
        && !gameManager.GetComponent<GameManager>().BeatsPerMinute.Equals(0))
    {
        GetComponent<Button>().interactable = true;
    }
    else
    {
        GetComponent<Button>().interactable = false;
    }
}
```

Έτσι ελέγχονται οι παραπάνω συνθήκες. Ελέγχεται αν το path (μονοπάτι τοποθεσίας) του μουσικού κομματιού ειναι κενό, που σημαίνει ότι δεν εχει φορτωθεί κάποιο μουσικό κομμάτι. Ελέγχεται αν ο τίτλος του μουσικού κομματιού ειναι κενός καθώς και αν ο ρυθμός αναπαραγωγής του ειναι ίσος με το μηδέν. Αν μια απο τις παραπάνω συνθήκες ισχύει, τότε το κουμπί EditorButton παραμένει απενεργοποιημένο. Και αν καμία απο τις συνθήκες δεν ισχύει τότε το EditorButton ενεργοποιείται ώστε να ξεκινήσει την επεξεργασία ο παίκτης.

- **BackButton**

Το BackButton ειναι υπεύθυνο για να μεταφέρει τον παίκτη σε προηγούμενη διεπαφή απο την τωρινή. Έτσι ενεργοποιεί την UIMainCanvas και απενεργοποιεί την τωρινή που ειναι η UIMainCanvasEdit. Επίσης εκτελεί την συνάρτηση ClearSongPath() του Game Manager.

Εικόνα 199 - Ιδιότητες του Back Button



Εικόνα 200 - Συνάρτηση της κλάσης GameManager

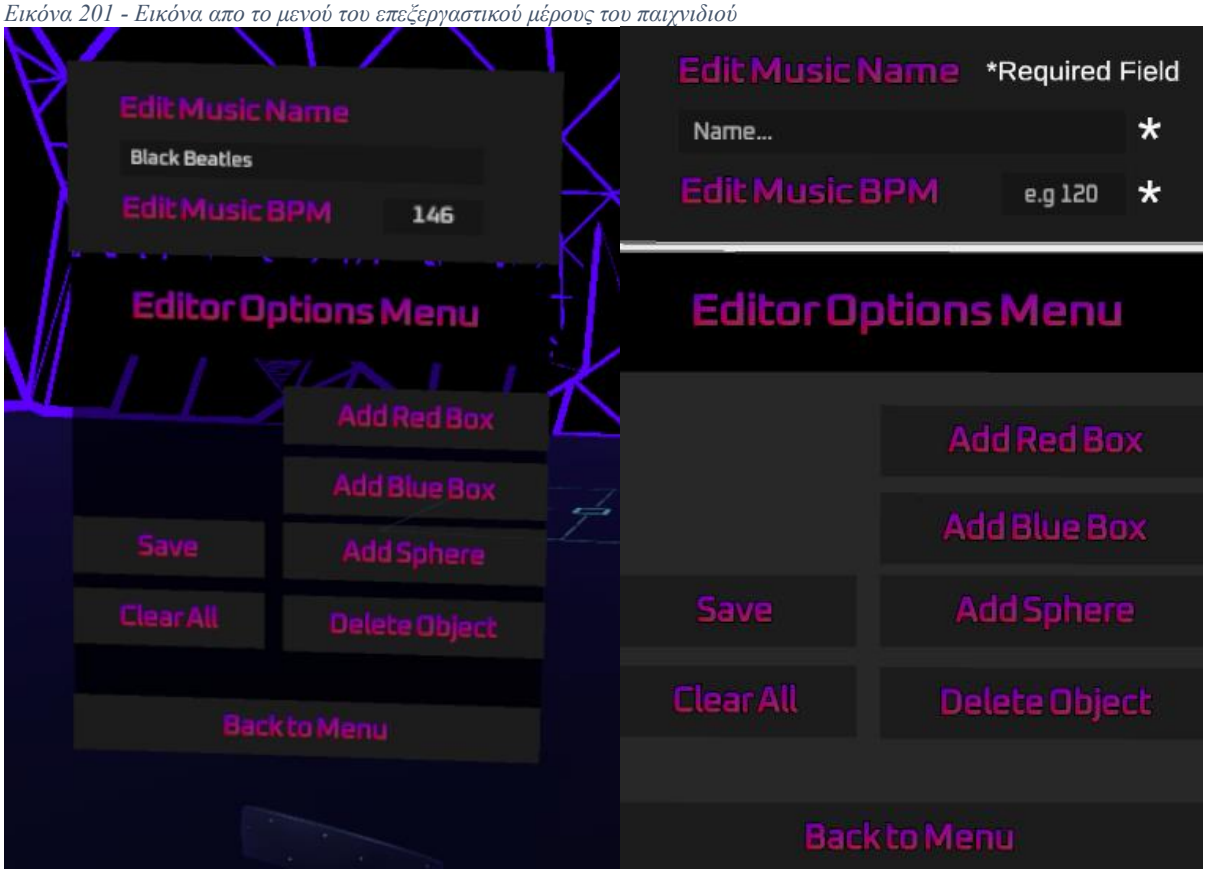
```
public void ClearSongPath()
{
    CurrentSongNamePath = "";
}
```

Αυτή η συνάρτηση ουσιαστικά «ξεφορτώνει» το μουσικό κομμάτι διαγράφοντας (θέτοντας κενό) το μονοπάτι της τοποθεσίας του μουσικού κομματιού που ίσως είχε φορτωθεί προηγουμένως απο τον παίκτη.
Αυτό ειναι απαραίτητο για να μην δημιουργούνται προβλήματα στις επιλογές του παίκτη, καθώς και στην ομαλή λειτουργία του παιχνιδιού.

v. UICanvasEditProperties

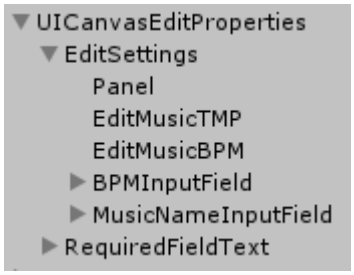
Το UICanvasEditProperties ειναι η διεπαφή που εμφανίζεται οταν ο παίκτης επιλέξει – πατήσει το κουμπί Edit and Open Editor. Έτσι, ξεκινάει η επεξεργασία της πίστας ενός ήδη δημιουργημένου μουσικού κομματιού. Η διεπαφή UICanvasEditProperties εμφανίζεται μαζι με την διεπαφή UICanvas που ειναι απαραίτητη για την δημιουργία της πίστας.

Εικόνα 201 - Εικόνα απο το μενού του επεξεργαστικού μέρους του παιχνιδιού



Αυτή η διεπαφή στην ιεραρχία της σκηνής περιέχει έξι διεπαφές. Ένα πάνελ παρασκήνιου της διεπαφής, δυο διεπαφές κειμένου για τους τίτλους, δυο Input Fields και μια διεπαφή RerquiredFieldText που ειναι υπεύθυνη για την εμφάνιση των προειδοποιητικών αστερίσκων διπλά απο τα Input Fields.

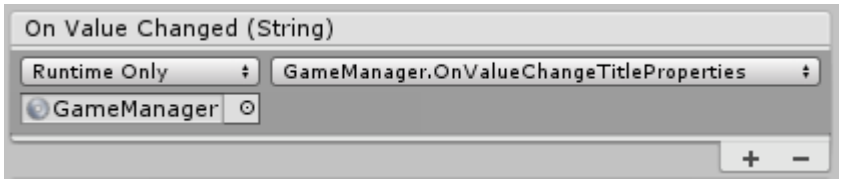
Εικόνα 202 - Αντικείμενα στην ιεραρχία της σκηνής, του επεξεργαστικού μέρους



- **MusicNameInputField – Input Field**

Το MusicNameInputField ειναι ένα στοιχείο διεπαφής τύπου Input Field. Δηλαδή εισαγωγής κειμένου, απο τον χρήστη μέσω του πληκτρολογίου. Αυτή η διεπαφή εκτελεί και λειτουργεί ακριβώς με τον ίδιο τροπο όπως και το MusicNameInputField που αναφέρθηκε στην διεπαφή UIMainCanvasCreate.

Εικόνα 203 - Ιδιότητες του MusicNameInputField



Έτσι με κάθε αλλαγή κειμένου απο τον παίκτη εκτελείται η συνάρτηση OnValueChangeTitleProperties() του Game Manager.

Εικόνα 204 - Συνάρτηση της κλάσης GameManager

```
public void OnValueChangeTitleProperties()
{
    SongTitle = UICanvasEditSongTitleProperties
        .GetComponent<InputField>().text;
    UpdateAllObjectProperties();
}
```

Εικόνα 205 - Συνάρτηση της κλάσης GameManager

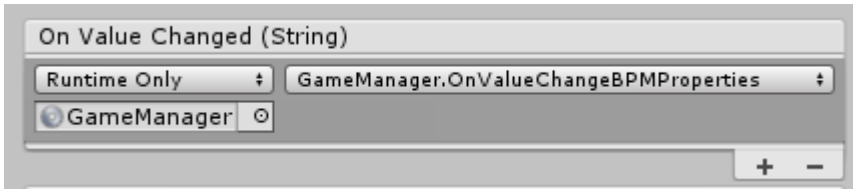
```
public void UpdateAllObjectProperties()
{
    allObjects.properties[0].SongTitle = SongTitle;
    allObjects.properties[0].SongTimeSeconds = SongTimeSeconds;
    allObjects.properties[0].BeatsPerMinute = BeatsPerMinute;
}
```

Με την χρήση αυτών των συναρτήσεων γίνεται ενημέρωση του τίτλου του μουσικού κομματιού, στον Game Manager, αλλα και ενημέρωση όλων των ιδιοτήτων της πίστας, στην μεταβλητής allObjects που χρησιμοποιείται για την αποθήκευση αυτών των πληροφοριών σε JSON αρχεία.

- **BPMInputField – Input Field**

Το BPMInputField ειναι το Input Field υπεύθυνο για την εισαγωγή του ρυθμού αναπαραγωγής του μουσικού κομματιού (BPM – Beats Per Minute). Όπως και στο Name Input Field έτσι και αυτή η διεπαφή εκτελεί και λειτουργεί ακριβώς με τον ίδιο τροπο όπως και το BPMInputField που εχει αναφερθεί προηγουμένως στην διεπαφή UIMainCanvasCreate.

Εικόνα 206 - Ιδιότητες του BPMInputField



Έτσι με κάθε αλλαγή κειμένου απο τον παίκτη εκτελείται η συνάρτηση OnValueChangeBPMProperties() του Game Manager.

Εικόνα 207 - Συνάρτηση της κλάσης GameManager

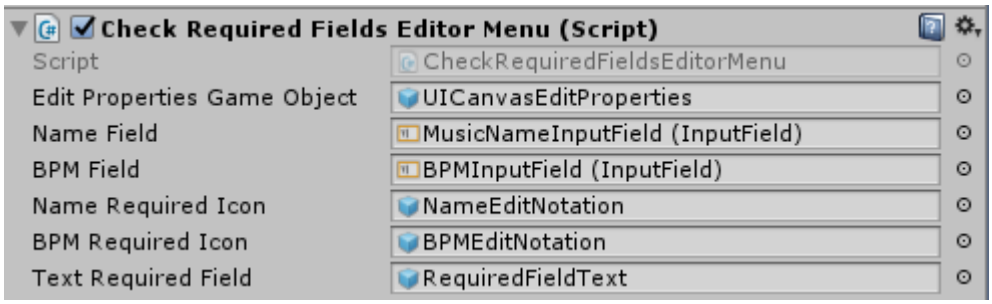
```
public void OnValueChangeBPMProperties()
{
    BeatsPerMinute = float.Parse(UICanvasEditSongBPMProperties
        .GetComponent<InputField>().text);
    UpdateAllObjectProperties();
}
```

Με την χρήση αυτών των συναρτήσεων γίνεται ενημέρωση του ρυθμού αναπαραγωγής του μουσικού κομματιού, στον Game Manager. Επίσης γίνεται ξανα ενημέρωση όλων των ιδιοτήτων της πίστας, στην μεταβλητής allObjects που χρησιμοποιείται για την αποθήκευση αυτών των πληροφοριών σε JSON αρχεία.

- **CheckRequiredFieldsEditorMenu**

Η κλάση CheckRequiredFieldsEditorMenu δημιουργήθηκε για να ελέγχονται οι περιορισμοί της διεπαφής που ειναι απαραίτητοι για την σωστή λειτουργία των διεπαφών και γενικά του παιχνιδιού.
Τα βασικά πεδία που πρεπει να ελεγχτούν ειναι το MusicNameInputField και το BPMInputField. Αυτοί οι έλεγχοι γίνονται με βάση το παρακάτω Script.

Εικόνα 208 - Ιδιότητες και πεδία της κλάσης CheckRequiredFieldsEditorMenu



Εικόνα 209 - Συνάρτηση Update της κλάσης CheckRequiredFieldsEditorMenu

```
// Update is called once per frame
void Update()
{
    if (EditPropertiesGameObject.activeSelf)
    {
        //Check Name Field
        if (!NameField.text.Equals(""))
        {
            NameRequiredIcon.SetActive(false);
        }
        else
        {
            NameRequiredIcon.SetActive(true);
        }
```

Η παραπάνω συνθήκη ελέγχει αν το πεδίο του ονόματος του μουσικού κομματιού (NameField) ειναι κενό. Αν ειναι κενό τότε εμφανίζεται ένα εικονίδιο – αστεράκι διπλά στο πεδίο αυτό.

Εικόνα 210 - Συνάρτηση Update της κλάσης CheckRequiredFieldsEditorMenu

```
//Check BPM Field
if (!BPMField.text.Equals(""))
{
    BPMRequiredIcon.SetActive(false);
}
else
{
    BPMRequiredIcon.SetActive(true);
}
```

Αυτή η συνθήκη ελέγχει αντίστοιχα αν το πεδίο του ρυθμού αναπαραγωγής του μουσικού κομματιού ειναι κενό. Στην περίπτωση που ειναι κενό, εμφανίζεται ένα εικονίδιο – αστεράκι διπλά στο πεδίο αυτό.

Εικόνα 211 - Συνάρτηση Update της κλάσης CheckRequiredFieldsEditorMenu

```
//Check Any Field
if (!BPMField.text.Equals("") && !NameField.text.Equals(""))
{
    TextRequiredField.SetActive(false);
    GetComponent<Button>().interactable = true;
}
else
{
    TextRequiredField.SetActive(true);
    GetComponent<Button>().interactable = false;
}
```

Αυτή η συνθήκη ελέγχει ταυτόχρονα τις δύο παραπάνω συνθήκες αν αληθεύουν, ώστε στην περίπτωση που έστω και μια να μην ισχύει (δηλαδή να ειναι κενό το πεδίο της), να εμφανίζεται το κείμενο που υποδεικνύει τι σημαίνει το εικονίδιο αστεράκι που γραφεί «*Required Fields». Επίσης αν υπάρχει κενό πεδίο το κουμπί Save της διεπαφής UICanvas απενεργοποιείται, έτσι ώστε να μην μπορεί ο παίκτης να αποθηκεύσει τις ιδιότητες του μουσικού κομματιού καθώς θα αποθηκευτούν οι πληροφορίες ως κενά. Ουσιαστικά, θα αντικαταστήσουν – διαγράψουν τις προηγούμενες αποθηκευμένες πληροφορίες απο τα JSON αρχεία.

Κεφάλαιο 6 : Διαχείριση ήχου στις σκηνές του παιχνιδιού

Και στις τρεις σκηνές του παιχνιδιού γίνεται χρήση ενός Sound Manager αντικειμένου, όπου στην συγκεκριμένη υλοποίηση του παιχνιδιού χρησιμοποιείται για την αναπαραγωγή και τον τροπο αναπαραγωγής των ηχητικών εφέ.

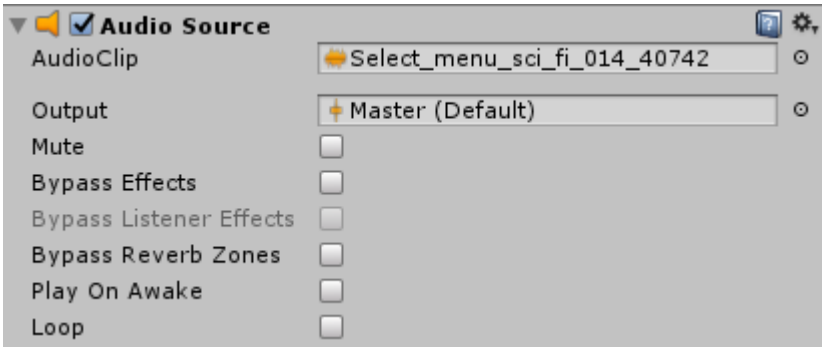
Εικόνα 212 - Αντικείμενο SoundManager στην ιεραρχία της κάθε σκηνής

Εικόνα 213 - Ιδιότητες της κλάσης Sound Manager

Στον SoundManager γίνεται ρύθμιση – επεξεργασία των ιδιοτήτων του Audio Source στο ButtonSFX που ειναι υπεύθυνο για τα ηχητικά εφέ των κουμπιών. Έτσι γίνεται αλλαγή στις τιμές του εύρους των υψηλών και χαμηλών τόνων του Audio Source, για να υπάρχει ένα σωστό αποτέλεσμα ήχου.

Το ButtonSFX ειναι ένα αντικείμενο που περιέχει ένα Audio Source. Σε αυτό το Audio Source εχει οριστεί μονιμά ένα Audio Clip με ηχητικό εφέ που αντιπροσωπεύει μια επιλογή (π.χ. πάτημα ενός κουμπιού). Σε αυτό το Audio Source εχει οριστεί η έξοδος ήχου να ειναι απο το Master Output (δηλαδή ο κύριος ήχος του υπολογιστή). Επίσης δεν εχει γίνει κάποια αλλαγή στις επιλογές του Audio Source απο τις προεπιλεγμένες.

Εικόνα 214 - Ιδιότητες του Audio Source



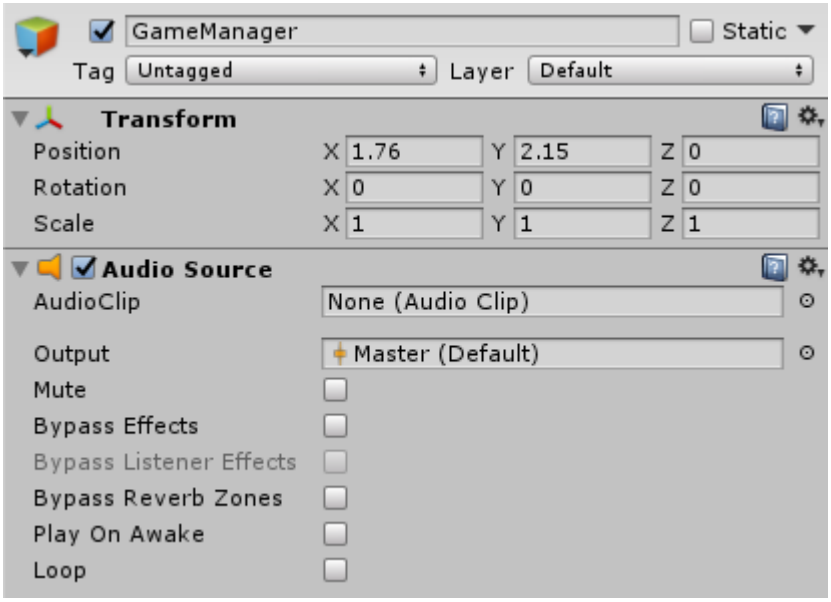
Έτσι οταν ο παίκτης τοποθετήσει την κουκίδα – δείκτη του EyeRaycaster πάνω απο κάποια διεπαφή αλληλεπίδρασης (όπως κουμπί), και πατήσει την σκανδάλη του αριστερού χειριστηρίου, τότε γίνεται αναπαραγωγή αυτού του ηχητικού εφέ (Select_Menu_sci_fi).

Εικόνα 215 - Συνθήκη στην Update της κλάσης EyeRaycaster

```
//Select after controller click
if (SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.LeftHand))
{
    ButtonSFX.Play();
```

Αυτή η αναπαραγωγή γίνεται μέσω της κλάσης EyeRaycaster και εκτελεί την ButtonSFX.Play(), όπου το ButtonSFX ειναι το Audio Source κάτω απο τον SoundManager στην ιεραρχία της σκηνής. Άρα αυτό που εκτελεί αυτή η εντολή ειναι η αναπαραγωγή (.Play()), του Clip (τοποθετημένο ηχητικό εφέ) που εχει αυτό το Audio Source (ButtonSFX).

Εικόνα 216 - Ιδιότητες του Audio Source του Game Manager



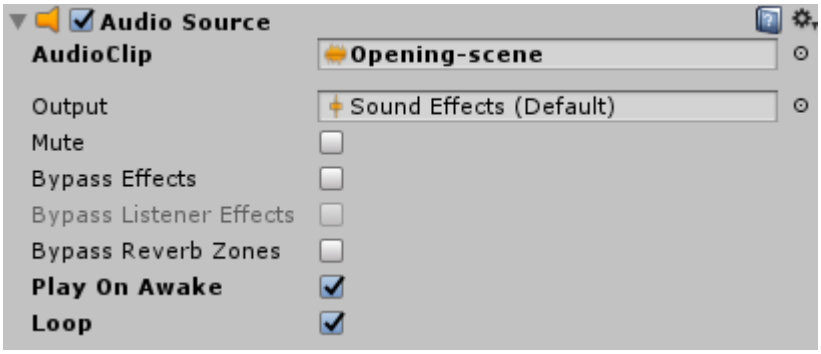
Στην σκηνή μόνο του ρυθμικού και του επεξεργαστικού μέρους του παιχνιδιού, γίνεται χρήση του Audio Source που ειναι τοποθετημένο στο αντικείμενο του Game Manager.
Αυτό το Audio Source χρησιμοποιείται και στις δυο σκηνές μόνο για την αναπαραγωγή / παύση του μουσικού κομματιού, που επέλεξε και φόρτωσε ο παίκτης, για να παίξει ή να επεξεργαστεί. Και σε αυτό το Audio Source η έξοδος ήχου ειναι απο το Master Output.

6.1 Διαχείριση ήχου στην αρχική σκηνή

Στο Audio Source εχει οριστεί για AudioClip η μουσική (Opening-scene), που ειναι ένα μουσικό κομμάτι που παίζει σε χαμηλή ένταση στο παρασκήνιο της σκηνής και λειτουργεί ως μουσική - ήχος για το μενού.

Οι ιδιότητες του Audio Source έχουν αλλάξει, έτσι ώστε αυτός ο ήχος παρασκήνιου να ξενίσει να αναπαράγεται με το που ξεκινήσει η σκηνή. Επίσης έχουν γίνει αλλαγές για να γίνεται η αναπαραγωγή σε επανάληψη, ώστε να υπάρχει συνεχόμενα ήχος ανεξάρτητα τον χρονο που μπορεί να βρίσκεται ο παίκτης σε αυτή την σκηνή.

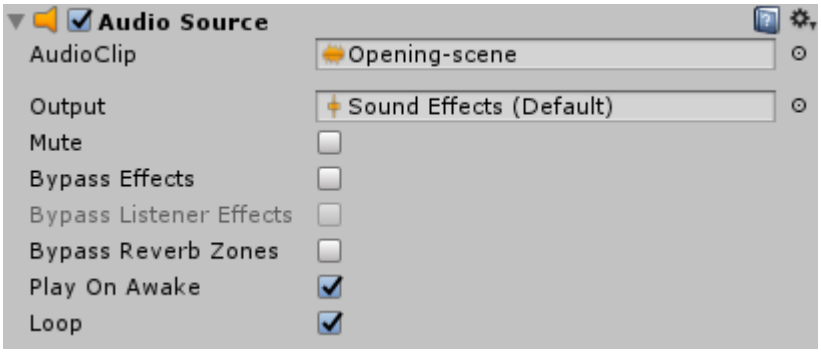
Εικόνα 217 - Ιδιότητες του Audio Source



6.2 Διαχείριση ήχου στην σκηνή του ρυθμικού μέρους

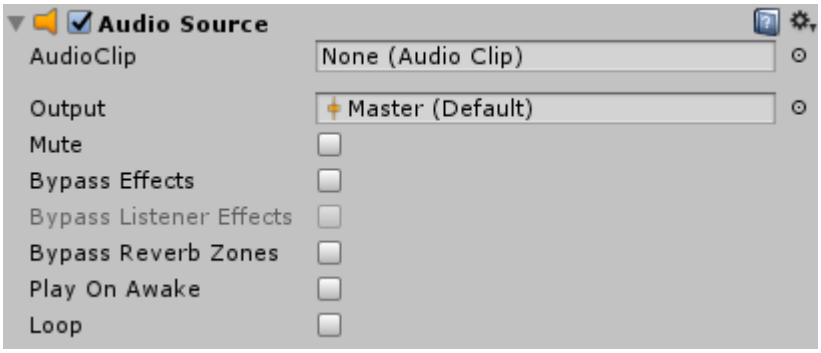
Όπως και στην αρχική σκηνή, έτσι και σε αυτή, γίνεται χρήση του Audio Clip (Opening-Scene), μέσω του Audio Source που εχει τις ίδιες ιδιότητες, ώστε να γίνεται αναπαραγωγή της μουσικής στο παρασκήνιο, όσο ο παίκτης κανει επιλογές στο μενού του ρυθμικού μέρους.

Εικόνα 218 - Ιδιότητες του Audio Source



Audio Source Game Manager

Εικόνα 219 - Ιδιότητες του Audio Source του GameManager



Όταν ο παίκτης αλληλεπιδράσει με τα αντικείμενα (Streaming Assets) με την χρήση των «οπλών» που υπάρχουν στα χειριστήρια, τότε γίνεται αναπαραγωγή ηχητικών εφέ που υποδηλώνουν αυτή την αλληλεπίδραση μεταξύ αντικειμένων και οπλών.
Αυτά τα ηχητικά εφέ ελέγχονται μέσω του WeaponSplitController, που ειναι υπεύθυνο γενικά για τις λειτουργίες που πρέπει να εκτελούν τα όπλα.

- **WeaponSplitController**

Εικόνα 220 - Συνθήκες στην κλάση WeaponSplitController

```
if ((WeaponColor.Equals("Red") &&
    other.gameObject.name.Equals("RedCubePrefab(Clone)")) ||
    (WeaponColor.Equals("Blue") &&
    other.gameObject.name.Equals("BlueCubePrefab(Clone)")))
{
    AddScorePoints(10);
    PlaySfxOnMode(true, other);
}
else
{
    RemoveScorePoints(10);
    PlaySfxOnMode(false, other);
}
else if (other.gameObject.tag.Equals("SpawnObject") &&
    other.gameObject.name.Equals("SpherePrefab(Clone)") &&
    other.gameObject.GetComponent<Splitable>() != null)
{
    //Play Particle System
    other.transform.GetChild(0).GetChild(0).GetComponent<ParticleSystem>().Play();
    other.transform.GetChild(0).GetChild(1).GetComponent<ParticleSystem>().Play();

    RemoveScorePoints(20);
    SoundManager.instance.PlaySingle(other.GetComponent<AudioClipSFX>().audioClip[0]);
}
```

Έτσι οταν γίνει αλληλεπίδραση με κάποιο κουτί, θα εκτελεστεί η συνάρτηση PlaySfxOnMode(), είτε γίνει αλληλεπίδραση με το ίδιο χρώμα όπλου και αντικειμένου, είτε ειναι διαφορετικό. Αν γίνει αλληλεπίδραση με σφαιρίδιο τότε γίνεται αναπαραγωγή του πρώτου ηχητικού εφέ στην λίστα των Audio Clip που υπάρχει στο σφαιρίδιο αυτό, που ειναι το ηχητικό εφέ ErrorSFX.

Η λειτουργία της συνάρτησης PlaySfxOnMode() ειναι να ελέγχει σε ποιο mode επέλεξε ο παίκτης να παίξει. Επίσης ελέγχει, αν εγινε σωστή ή λάθος αλληλεπίδραση και αναλόγως να αναπαράγει το αντίστοιχο ηχητικό κλιπ που βρίσκεται στο κάθε αντικείμενο.

Εικόνα 221 - Συνάρτηση της κλάσης WeaponSplitController

```
private void PlaySfxOnMode(bool CorrectObject, Collider other)
{
    //Make Pitch Random
    other.GetComponent<AudioSource>().pitch =
        SoundManager.instance.RandomPitch();

    if (GameObject.Find("GameManagerMainGame").
        GetComponent<GameManagerMainGame>().GameModeNum.Equals(1))
    {
        if (CorrectObject)
        {
            other.GetComponent<AudioSource>().clip =
                other.GetComponent<AudioClipSFX>().audioClip[1];
            other.GetComponent<AudioSource>().Play();
        }
        else
        {
            other.GetComponent<AudioSource>().clip =
                other.GetComponent<AudioClipSFX>().audioClip[2];
            other.GetComponent<AudioSource>().Play();
        }
    }
```

Στην περίπτωση που ο παίκτης εχει επιλέξει Rifle Mode, αναπαράγονται διαφορετικά ηχητικά εφέ σε σχέση με τα αλλα δυο modes του παιχνιδιού. Οπότε αν το όπλο εχει το ίδιο χρώμα με το αντικείμενο θα γίνει αναπαραγωγή του ηχητικού εφέ που βρίσκεται στην δεύτερη θέση στην λίστα των Audio Clips (δηλαδή το audioClip[1]). Εάν ειναι αντίθετο χρώμα τότε γίνεται αναπαραγωγή του ηχητικού εφέ που βρίσκεται στην τρίτη θέση στην λίστα των Audio Clips (δηλαδή το audioClip[2]).

Εικόνα 222 - Συνάρτηση της κλάσης WeaponSplitController

```
else
{
    if (CorrectObject)
    {
        other.GetComponent<AudioSource>().clip =
            other.GetComponent<AudioClipSFX>().audioClip[0];
        other.GetComponent<AudioSource>().Play();
    }
    else
    {
        other.GetComponent<AudioSource>().clip =
            other.GetComponent<AudioClipSFX>().audioClip[2];
        other.GetComponent<AudioSource>().Play();
    }
}
```

Στα υπόλοιπα δυο mode, αν το όπλο εχει το ίδιο χρώμα με το αντικείμενο θα γίνει αναπαραγωγή του ηχητικού εφέ που βρίσκεται στην πρώτη θέση στην λίστα των Audio Clips (δηλαδή το audioClip[0]) ενώ εάν ειναι αντίθετο χρώμα τότε γίνεται αναπαραγωγή του ηχητικού εφέ που βρίσκεται στην τρίτη θέση στην λίστα των Audio Clips (δηλαδή το audioClip[2]).

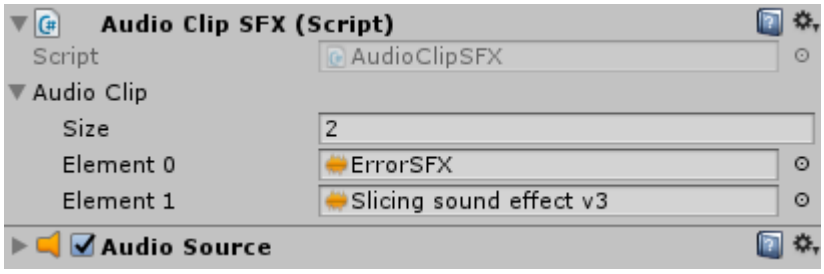
- **BlueCubePrefab και RedCubePrefab**

Εικόνα 223 - Ιδιότητες της κλάσης AudioClipSFX

Το Element 0 αντιστοιχεί στο audioClip[0] το οποίο ειναι ένα ηχητικό εφέ κοπής, το 1 αντιστοιχεί στο audioClip[1] που ειναι ένα ηχητικό εφέ σπασίματος γυαλιού (χρησιμοποιείται στο Rifle Mode), και το 2 αντιστοιχεί στο audioClip[2] που ειναι ένα ηχητικό εφέ υπόδειξης λάθους. Αυτά τα audioClips καλούνται απο την συνάρτηση PlaySfxOnMode().

- **SpherePrefab**

Εικόνα 224 - Ιδιότητες της κλάσης AudioClipSFX



Τα ηχητικά εφέ που υπαρχουν στο σφαιρίδιο ειναι δύο. Αυτό που υποδεικνύει κάποιο λάθος που βρίσκεται στο Element 0 και αντιστοιχεί στο audioClip[0] και αυτό της κοπής που βρίσκεται στο Element 1 και αντιστοιχεί στο audioClip[1].

6.3 Διαχείριση ήχου στην σκηνή του επεξεργαστικού μέρους

Στο επεξεργαστικό μέρος του παιχνιδιού γίνεται χρήση του Audio Source του Game Manager για την αναπαραγωγή του μουσικού κομματιού που εχει επιλέξει ο παίκτης για επεξεργασία η δημιουργία πίστας. Με την χρήση του GrabGrip κουμπιού στο χειριστήριο, δίνεται η δυνατότητα στον παίκτη να κανει αναπαραγωγή και παύση του μουσικού κομματιού.

Εικόνα 225 - Εντολή στην κλάση GameManager

```
gameObject.GetComponent<AudioSource>().Pause();
```

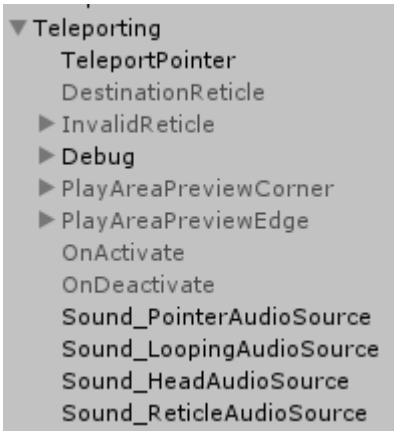
Εικόνα 226 - Εντολή στην κλάση GameManager

```
gameObject.GetComponent<AudioSource>().UnPause();
```

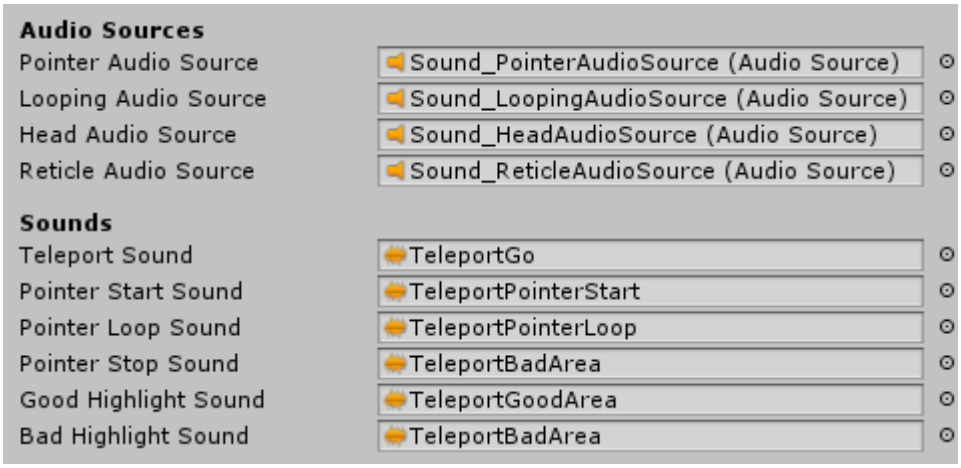
Οι παραπάνω εντολές εκτελούν αυτές τις λειτουργίες. Έτσι γίνεται Pause() ή UnPause() στο Audio Source που εχει φορτωθεί το κλιπ του μουσικού κομματιού.

- **Teleport**

Εικόνα 227 - Αντικείμενο Teleport στο επεξεργαστικό μέρος



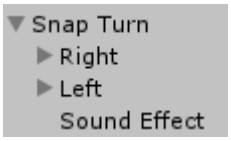
Εικόνα 228 - Πεδία Audio Sources και Sounds του Teleport



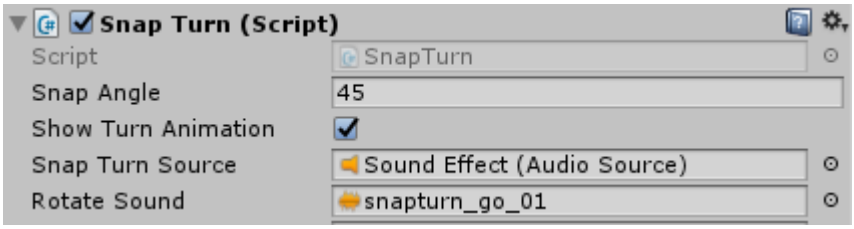
Επιπλέον οι εξτρά λειτουργίες που προσφέρει η επέκταση SteamVR όπως το teleport, χρησιμοποιούν ηχητικά εφέ για μια καλύτερη εμπειρία. Όπως φαίνεται στις παραπάνω εικόνες το αντικείμενο Teleporting εχει τέσσερα Audio Sources όπου το κάθε ένα ειναι τοποθετημένο σε διαφορετικό σημείο οταν ενεργοποιείται στην σκηνή. Επίσης εχει πέντε audio clips, που γίνονται αναπαραγωγή αναλόγως την περίπτωση.
Όλες οι ιδιότητες, Audio Sources και Audio Clips ειναι στις προεπιλεγμένες τιμές που υπήρχαν στην επέκταση.

- **Snap Turn**

Εικόνα 229 - Αντικείμενο SnapTurn στην ιεραρχία της σκηνής του επεξεργαστικού μέρους



Εικόνα 230 - Ιδιότητες της κλάσης SnapTurn



Έτσι και στο Snap Turn της επέκτασης SteamVR γίνεται χρήση ηχητικών εφέ για μια καλύτερη εμπειρία στον χρήστη. Σε αυτή την περίπτωση το Snap Turn χρησιμοποιεί μόνο ένα Audio Source και μόνο ένα ηχητικό κλιπ (το snapturn_go) για εφέ, που χρησιμοποιείται στο Rotate Sound. Στο Snap Turn θα γίνει αναπαραγωγή του rotate sound κάθε φορά που ο παίκτης θα κανει το απότομο γύρισμα, είτε δεξιά είτε αριστερά, με την χρήση του Joystick του δεξιού χειριστηρίου.

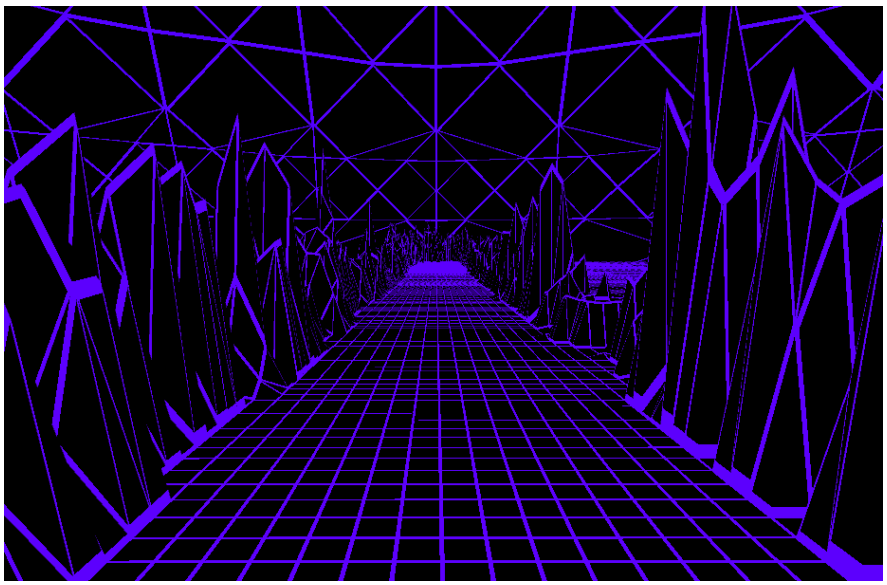
Κεφάλαιο 7 : Γραφικά στοιχεία του παιχνιδιού

7.1. Περιβάλλον

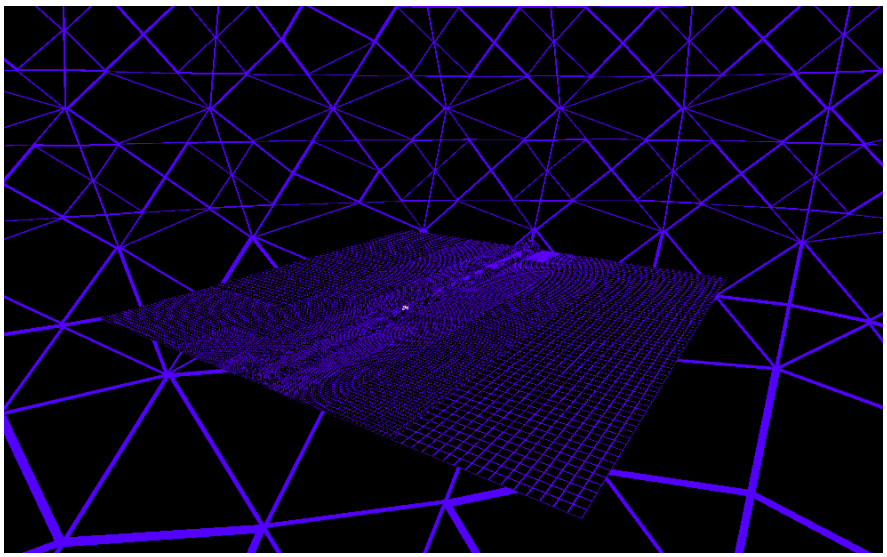
Εικόνα 231 - Αντικείμενα γραφικών στην ιεραρχία της αρχικής σκηνής

Το περιβάλλον (Level - Environment) σε κάθε σκηνή του παιχνιδιού ειναι το ίδιο. Τα βασικά γραφικά στοιχεία που αποτελούν το περιβάλλον ειναι τρία. Ο φωτισμός (Directional Light), ένα Terrain (Level) που συμπεριλαμβάνει και άλλα γραφικά αντικείμενα, και το Skybox που ειναι ουσιαστικά τα γραφικά που αποτελούν τον ουρανό του εικονικού κόσμου.

Εικόνα 232 - Γραφικό περιβάλλον των σκηνών

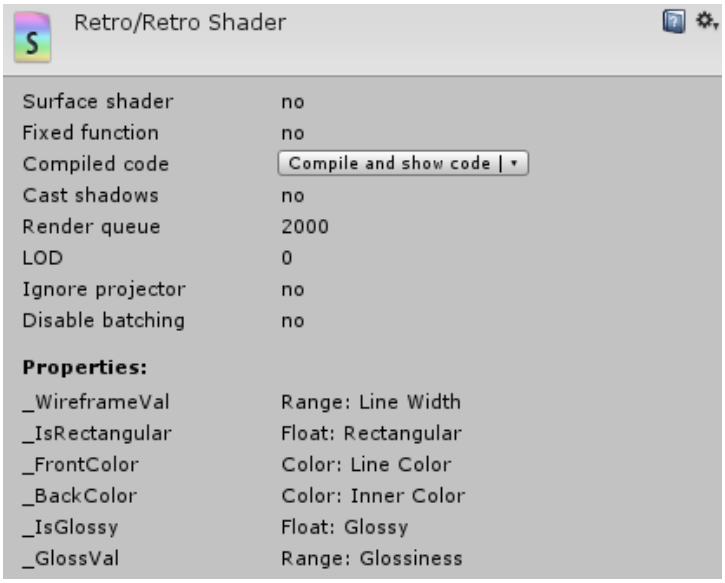


Εικόνα 233 - Γραφικό επίπεδο (Terrain) των σκηνών



Στις παραπάνω εικόνες φαίνεται το Terrain καθώς και το Skybox που χρησιμοποιήθηκαν σε κάθε σκηνή του παιχνιδιού.

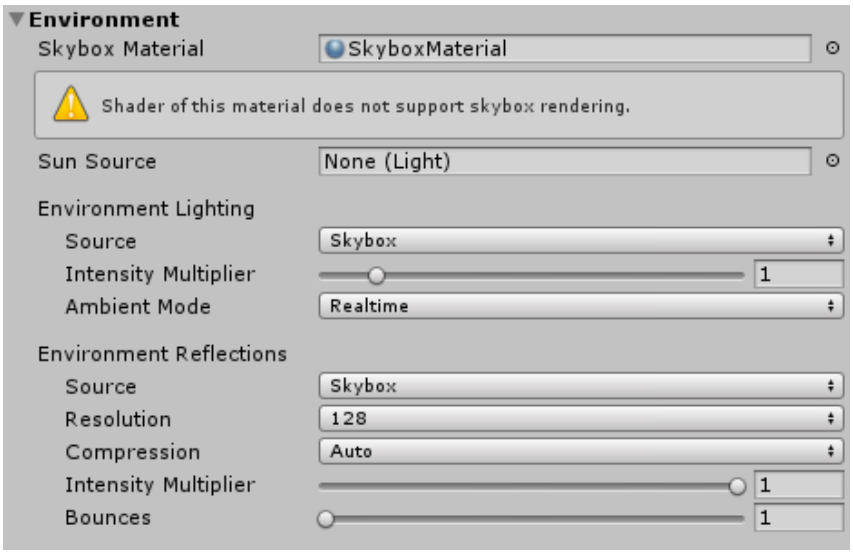
Εικόνα 234 - Ιδιότητες του Retro Shader



Και στο Terrain αλλα και στο Skybox εγινε χρήση ενός shader απο το Asset Store με το όνομα Retro Shader. Αυτό το shader ουσιαστικά δημιουργεί τα wireframes των αντικειμένων. Δηλαδή τις γραμμές (η και τρίγωνα) που συνθέτουν ένα αντικείμενο. Με αυτόν τον τρόπο γίνονται εμφανή τα wireframes των αντικειμένων μέσα στο παιχνίδι, με αποτέλεσμα να δημιουργούν ένα πλέγμα απο τετράγωνα ή και τρίγωνα.

- **Skybox**

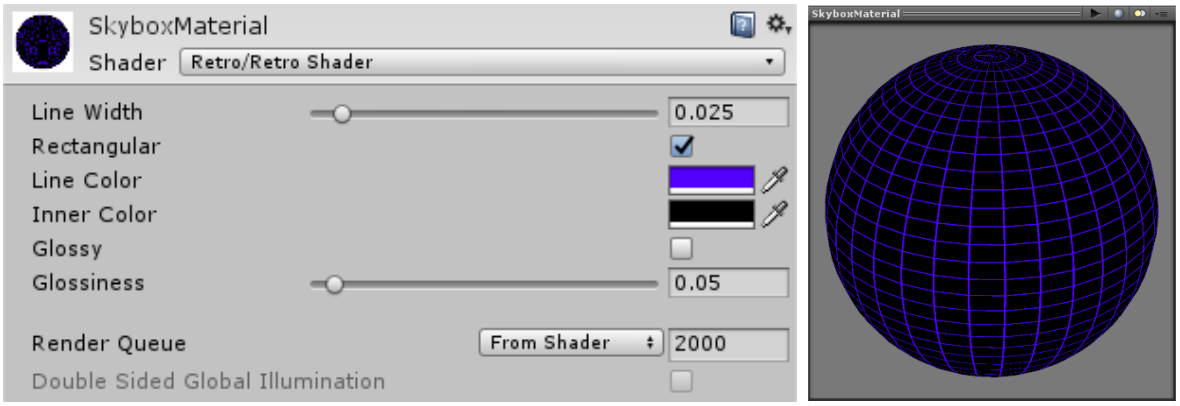
Εικόνα 235 - Ιδιότητες του Skybox



Στην παραπάνω εικόνα φαίνονται οι ιδιότητες του Skybox στο περιβάλλον. Το Skybox εχει τις προεπιλεγμένες ιδιότητες της Unity, εκτός απο το Skybox Material και το Sun Source. Στο παιχνίδι δεν χρειάζεται να υπάρχει κάποια πηγή φωτός απο το Skybox καθώς οι σκηνές χρησιμοποιούν το Directional Light.

Επιπλέον για το Skybox έγινε χρήση του SkyboxMaterial το οποιο χρησιμοποιεί το ίδιο shader με αυτό του Terrain, δηλαδή το Retro Shader. Στην παρακάτω εικόνα φαίνονται οι ιδιότητες του shader που χρησιμοποιήθηκαν για αυτό το material.

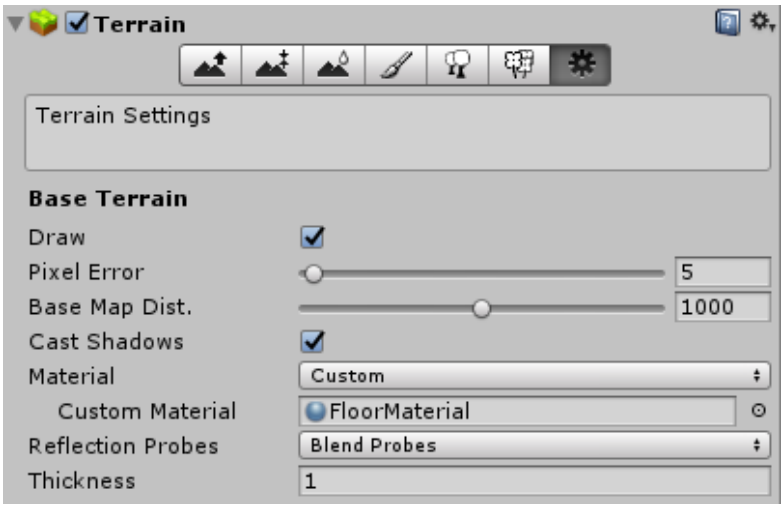
Εικόνα 236 - Ιδιότητες του υλικού του Skybox



- **Terrain**

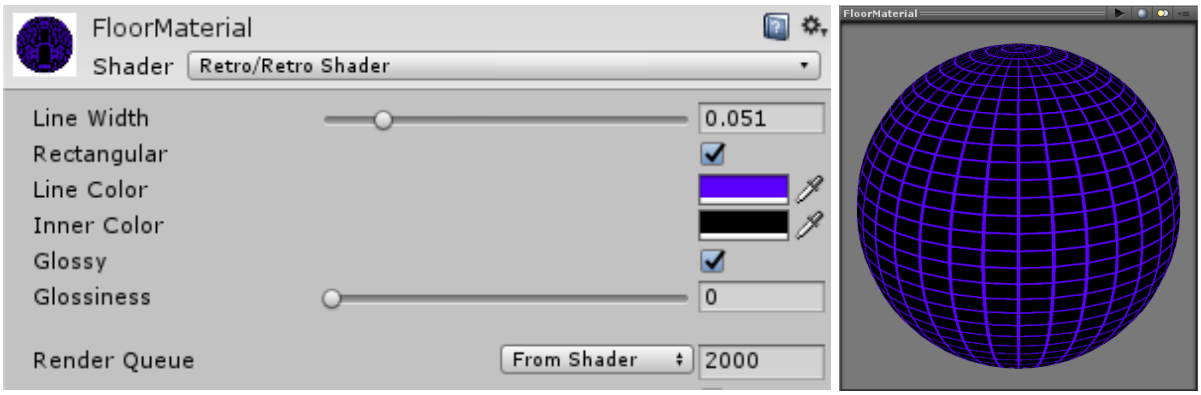
Όπως φαίνεται στην παρακάτω εικόνα οι ιδιότητες που εχει ένα terrain, ειναι στις προεπιλεγμένες τιμές της Unity. Η μόνη αλλαγή ειναι το custom material που εχει οριστεί το Floor Material.

Εικόνα 237 - Ιδιότητες του επιπέδου (Terrain)



Το FloorMaterial όπως φαίνεται παρακάτω χρησιμοποιεί και αυτό το shader με όνομα Retro Shader, ώστε να δημιουργηθεί αυτό το εφέ πλέγματος στο terrain. Οι ιδιότητες ειναι παρόμοιες με αυτές του SkyboxMaterial.

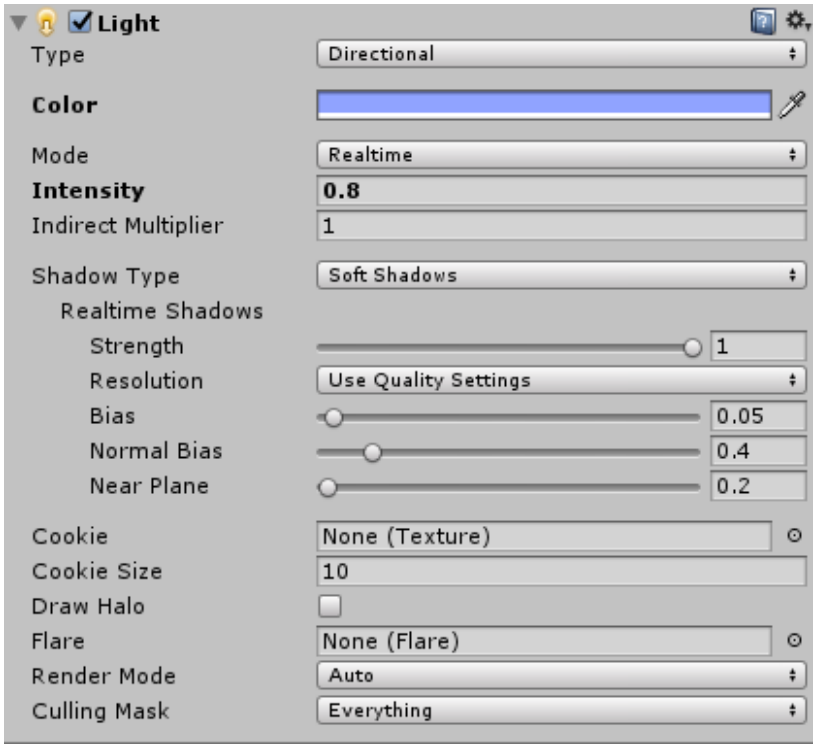
Εικόνα 238 - Ιδιότητες του υλικού του Πατώματος (Terrain - Floor)



- **Φωτισμός – Directional Light**

Σε κάθε σκηνή χρησιμοποιείται ένα Directional Light ώστε να φωτίσει τα αντικείμενα που υπαρχουν ή πρόκειται να δημιουργηθούν σε αυτή. Αυτός ο φωτισμός λειτουργεί σαν πηγή φωτός μέσα στην σκηνή και μπορεί να αλλάξει η κατεύθυνση του (ουσιαστικά η κλίση).

Εικόνα 239 - Ιδιότητες του φωτός



Αυτός ο φωτισμός χρησιμοποιείται απο την Unity στις σκηνές για να λειτουργεί σε πραγματικό χρονο (Realtime). Δηλαδή σε κάθε καρέ να γίνεται υπόκοσμος και ενημέρωση του φωτισμού και τον αντικειμένων που φωτίζονται απο αυτό ώστε να δημιουργούνται ρεαλιστικές σκιές.

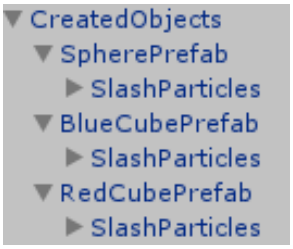
Εικόνα 240 - Επιλογή γενικού έμμεσου φωτισμού

Επίσης παράλληλα γίνεται χρήση του Realtime Global Illumination, που ειναι υπεύθυνο για να δημιουργεί έμμεσο φωτισμό στα στατικά και δυναμικά αντικείμενα.

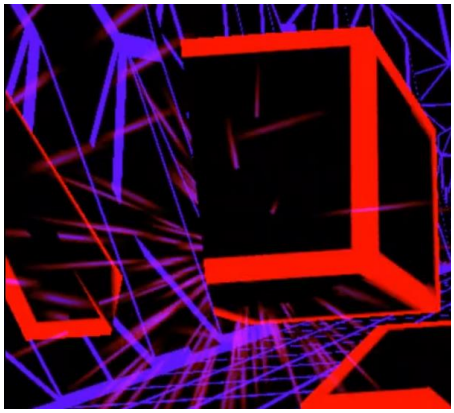
7.2 Εφέ Γραφικών Σωματιδίων

Στο ρυθμικό μέρος του παιχνιδιού οταν ο παίκτης κόβει (κανει slash) στα δυο κάποιο αντικείμενο με τα όπλα του τότε ενεργοποιείται ένα εφέ γραφικών σωματιδίων, στο σημείο που γίνεται η κοπή. Με αυτό το slash φαίνεται οπτικά καλύτερη η κοπή, ώστε να βοηθάει στην εμπειρία του παιχνιδιού.

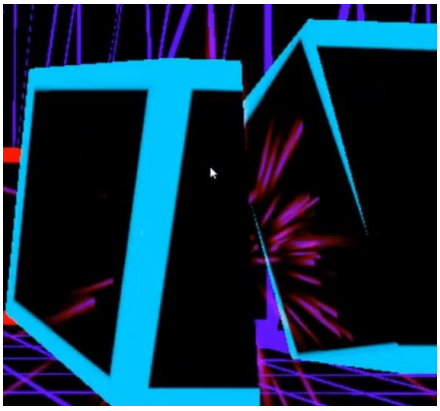
Εικόνα 241 - Αντικείμενα στη ιεραρχία του ρυθμικού μέρους



Εικόνα 242 - Υπόδειξη των ρόζ σώματιδιων

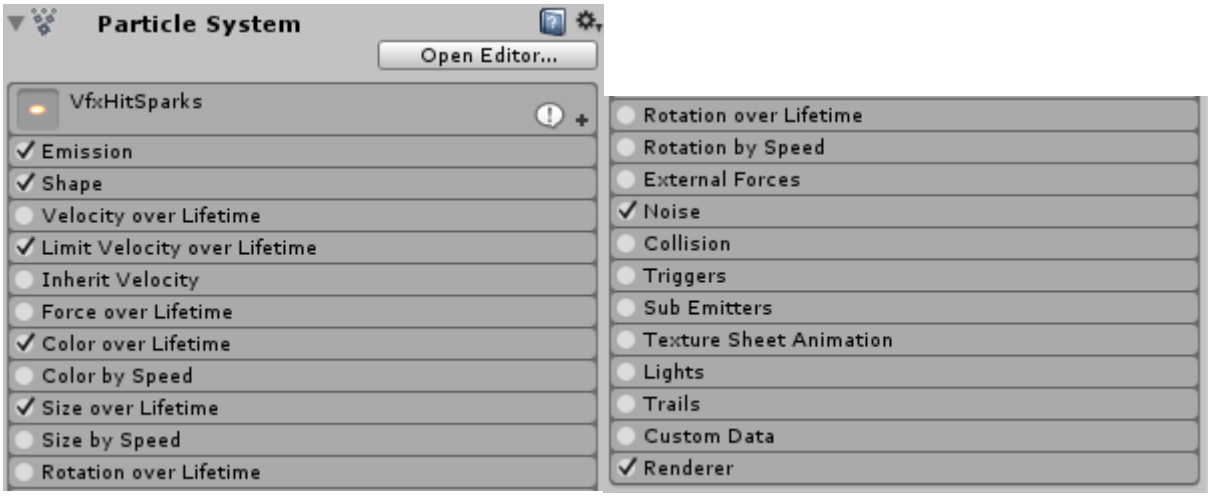


Εικόνα 243 - Υπόδειξη των ρόζ σώματιδιων



Έτσι σε κάθε Streaming Asset (Created Objects) που δημιουργείται περιέχει και το αντικείμενο SplashParticles. Αυτά τα particles δημιουργούνται μέσα απο την unity με το εργαλείο της, particle system. Έτσι για τα συγκεκριμένα particles (σωματίδια), φαίνονται στην παρακάτω εικόνα οι ιδιότητες και προσαρμογές που έγιναν για φαίνεται αυτό το αποτέλεσμα στο παιχνίδι.

Εικόνα 244 - Ιδιότητες του συστήματος σωματιδίων

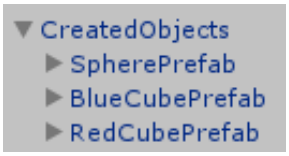


7.3 Γραφικό υλικό αντικειμένων και τα εφέ τους

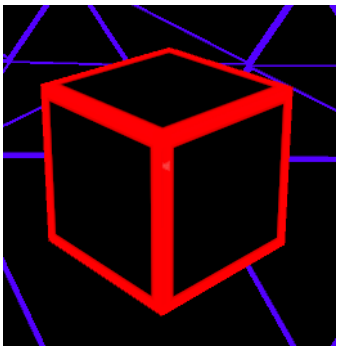
Ρυθμικό μέρος

Στο ρυθμικό μέρος του παιχνιδιού γίνεται χρήση τριών αντικειμένων. Του σφαιριδίου (SpherePrefab), του μπλε κουτιού (BlueCubePrefab) και του κόκκινου κουτιού (RedCubePrefab).

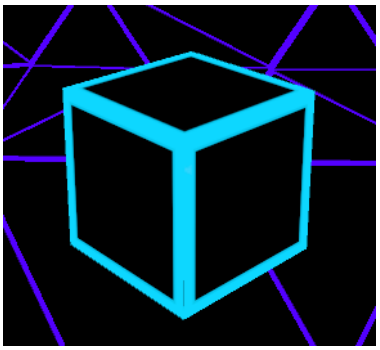
Εικόνα 245 - Αντικείμενα στην ιεραρχία του ρυθμικού μέρους



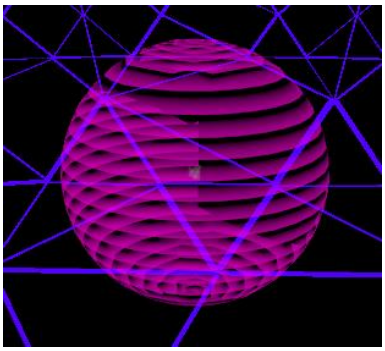
Εικόνα 246 - Κόκκινο κουτί μέσα στο παιχνίδι



Εικόνα 247 - Μπλέ κουτί μέσα στο παιχνίδι

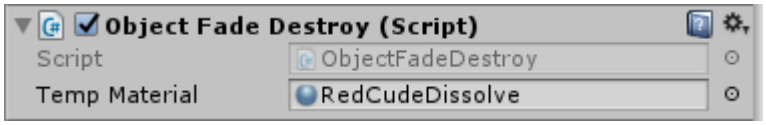


Εικόνα 248 - Σφαιρίδιο μέσα στο παιχνίδι



Αυτά τα αντικείμενα οταν χτυπηθούν (γίνουν collide), απο κάποιο όπλο, χωρίζονται σε δυο μέρη (δυο αλλα αντικείμενα), τα οποία μετα απο λιγο πρεπει να καταστραφούν για να μην στοιβαχτούν μπροστά στον παίκτη και χαλάσει η εμπειρία του παιχνιδιού. Έτσι εκτελείται η συνάρτηση FadeObject() της κλάσης ObjectFadeDestroy, η οποία χρησιμοποιεί τις μεταβλητές του shader που εχει το material του κάθε αντικειμένου. Αυτό που θα κανει ειναι να αλλάξει την μεταβλητή _Progress του Shader και να μικραίνει την τιμή με το πέρασμα του χρόνου με αποτέλεσμα να γίνεται σταδιακά και πιο διάφανες το αντικείμενο, με βάση το εφέ «dissolve texture», ως ότου το _Progress γίνει ίσο με το μηδέν και να εξαφανιστεί.

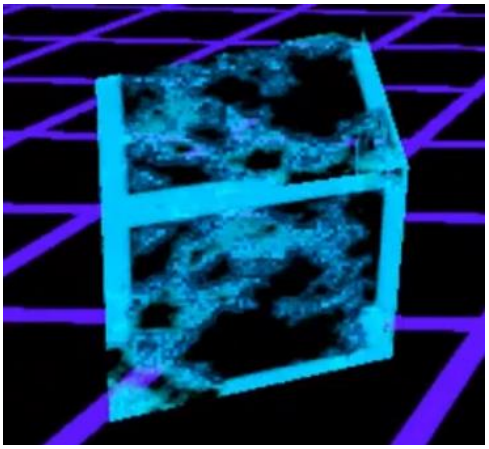
Εικόνα 249 - Ιδιότητες της κλάσης ObjectFadeDestroy



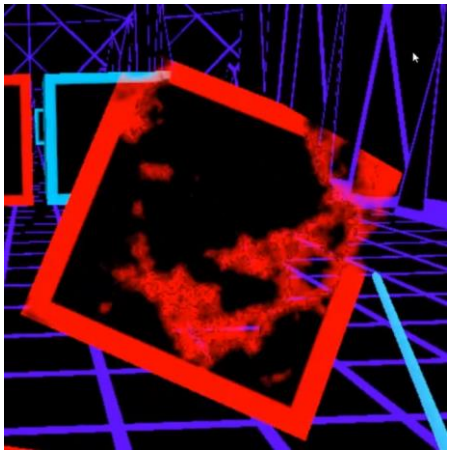
Εικόνα 250 - Συνάρτηση της κλάσης ObjectFadeDestroy

```
void FadeObject()
{
    timer += Time.deltaTime;
    float dissovleValue = 1f - timer;
    m_PropertyBlock.SetFloat("_Progress", dissovleValue);
    myRenderer.SetPropertyBlock(m_PropertyBlock);
}
```

Εικόνα 251 - Εφέ Dissolve στο μπλέ κουτί

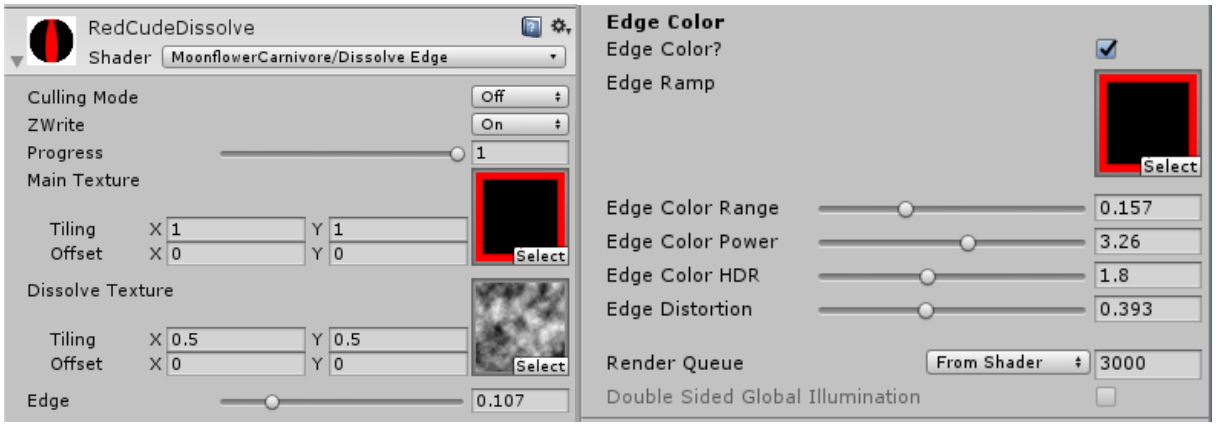


Εικόνα 252 - Εφέ Dissolve στο κόκκινο κουτί



Στις παραπάνω εικόνες φαίνεται το εφέ “Dissolve Texture” στα αντικείμενα που χρησιμοποιείται στο shader Dissolve Edge.

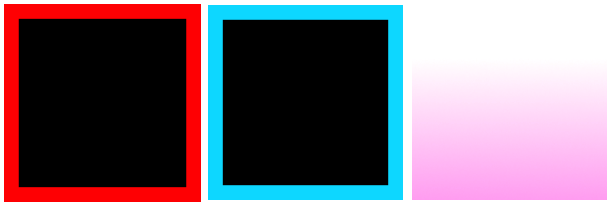
Εικόνα 253 - Ιδιότητες του Dissolve εφέ



Έτσι σε κάθε αντικείμενο γίνεται χρήση του shader με όνομα Dissolve Edge το οποιο παίρνει μια εικόνα και την προσαρμόζει πάνω στο αντικείμενο και αργότερα, αναλόγως το Progress εφαρμόζεται το Dissolve Texture πάνω στην εικόνα, ώστε να φανεί φθαρμένο το αντικείμενο καθώς ταυτόχρονα γίνονται διαφανή κάποια σημεία του αντικειμένου, με τον χρονο.

Οι παραπάνω εικόνες δείχνουν τις ιδιότητες του RedCubeDissolve Material που χρησιμοποιείται για το αντικείμενο του κόκκινου κουτιού. Έτσι με ακριβώς τις ίδιες ιδιότητες χρησιμοποιείται και στα υπόλοιπα αντικείμενα. Με την μόνη διαφορά να ειναι το Main Texture, που ειναι διαφορετικό για κάθε αντικείμενο.

Εικόνα 254 - Textures αντικειμένων που χρησιμοποιεί το Dissolve εφέ



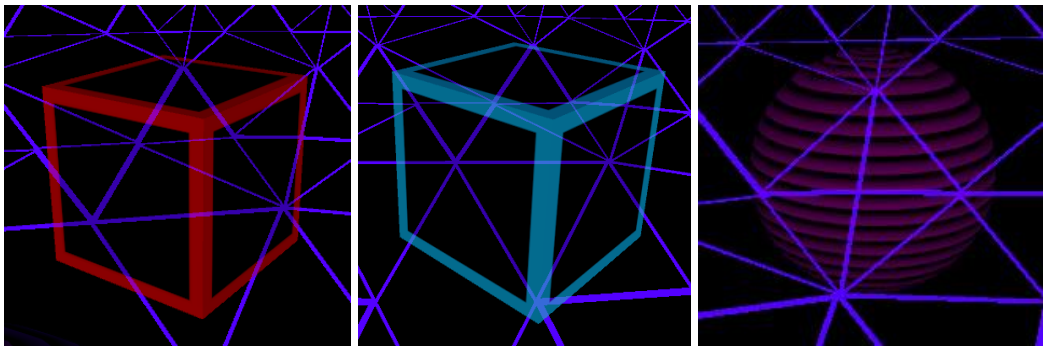
Οι παραπάνω εικόνες ειναι τα Main Textures που χρησιμοποιεί το κάθε αντικείμενο αντίστοιχα, για να δημιουργηθεί με την χρήση του dissolve material.

Επεξεργαστικό μέρος

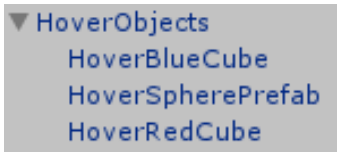
Στο επεξεργαστικό μέρος γίνεται χρήση δυο ειδών αντικειμένων. Αυτών που ειναι υπεύθυνα για την υπόδειξη της θέσης τους πάνω στην επιφάνεια πλέγματος, τα οποία ειναι ημιδιαφανή. Και αυτών των αντικειμένων που μετα την υπόδειξη, ο παίκτης θα τα δημιουργήσει πάνω στην επιφάνεια του πλέγματος

- **HoverObjects**

Εικόνα 255 - Διαφανή αντικείμενα με την χρήση του Diffuse εφέ

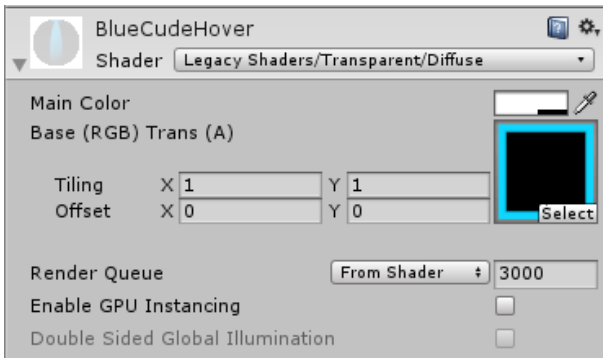


Εικόνα 256 - Αντικείμενα στην ιεραρχία του επεξεργαστικού μέρους

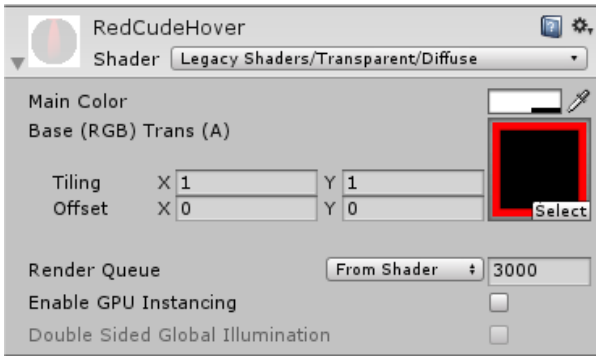


Τα Hover Objects δημιουργούνται πάνω στην επιφάνεια του πλέγματος οταν ο παίκτης τοποθετήσει τον δείκτη του χειριστηρίου, πάνω σε αυτήν. Έτσι, όπως φαίνεται και στις παραπάνω εικόνες, τα αντικείμενα ειναι ημιδιαφανή ώστε να υποδείξουν στον παίκτη ότι σε αυτό το σημείο πρόκειται να τοποθετηθεί το αντικείμενο που θέλει να δημιουργήσει.

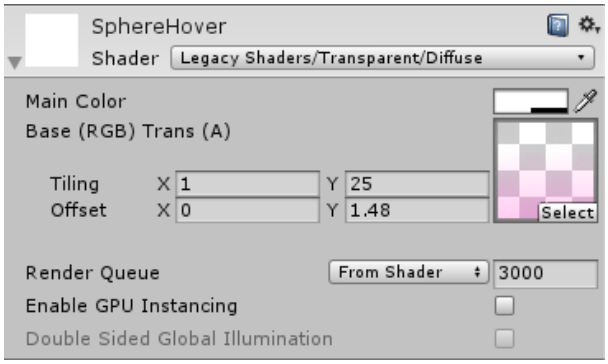
Εικόνα 257 - Ιδιότητες του Diffuse εφέ με διαφάνεια για το μπλέ κουτί



Εικόνα 258 - Ιδιότητες του Diffuse εφέ με διαφάνεια για το κόκκινο κουτί



Εικόνα 259 - Ιδιότητες του Diffuse εφέ με διαφάνεια για το σφαιρίδιο

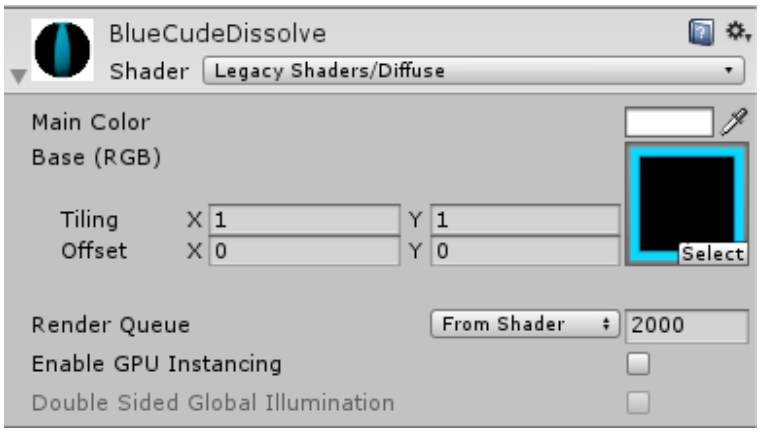


Οι παραπάνω εικόνες δείχνουν τις ιδιότητες του material που εχει το κάθε αντικείμενο. Για όλα αντικείμενα γίνεται χρήση του Diffuse shader και η διαφάνεια στο κύριο χρώμα τους ειναι στο μισό. Και σε αυτό το shader γίνεται χρήση των texture αντίστοιχα για κάθε αντικείμενο όπως εγινε και στο Dissolve Edge shader.

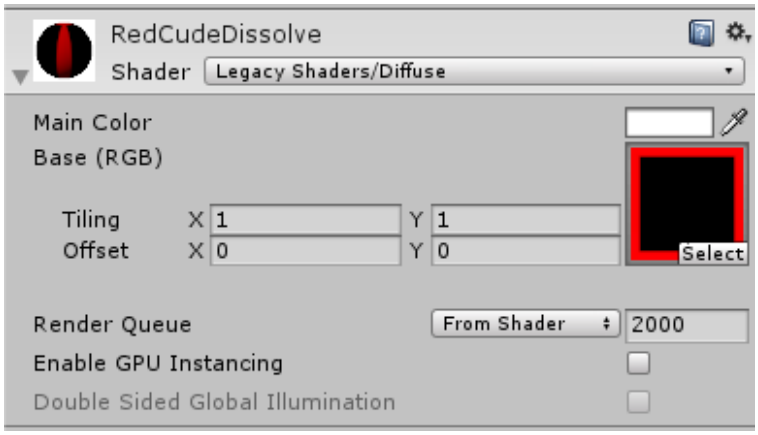
- **Created Objects**

Τα Created Objects ειναι τα αντικείμενα που θα δημιουργηθούν αφού επιλέξει ο παίκτης θέση πάνω στην επιφάνεια πλέγματος με το χειριστήριο του.

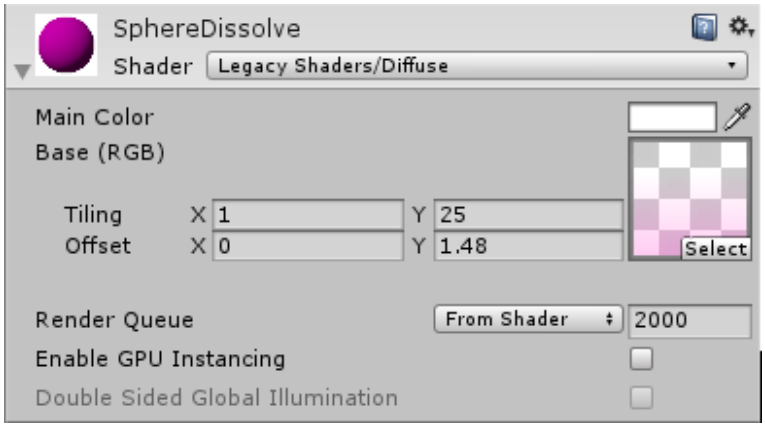
Εικόνα 260 - Ιδιότητες του Diffuse εφέ χωρίς διαφάνεια για το μπλέ κουτί



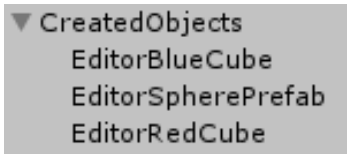
Εικόνα 261 - Ιδιότητες του Diffuse εφέ χωρίς διαφάνεια για το κόκκινο κουτί



Εικόνα 262 - Ιδιότητες του Diffuse εφέ χωρίς διαφάνεια για το σφαιρίδιο



Εικόνα 263 - Αντικείμενα στην ιεραρχία του επεξεργαστικού μέρους



Όπως και στα Hover Objects έτσι και τα Created Objects χρησιμοποιούν το ίδιο shader (το Diffuse) καθώς και τα ίδια αντίστοιχα textures για τα αντικείμενα. Η μόνη διαφορά σε αυτή την περίπτωση ειναι η αδιαφάνεια του Main Color, που ειναι ολόκληρη. Έτσι τα αντικείμενα φαίνονται πλήρως οταν δημιουργούνται πάνω στην επιφάνεια πλέγματος.

Κεφάλαιο 8 : Εξωτερικά προγράμματα και βιβλιοθήκες

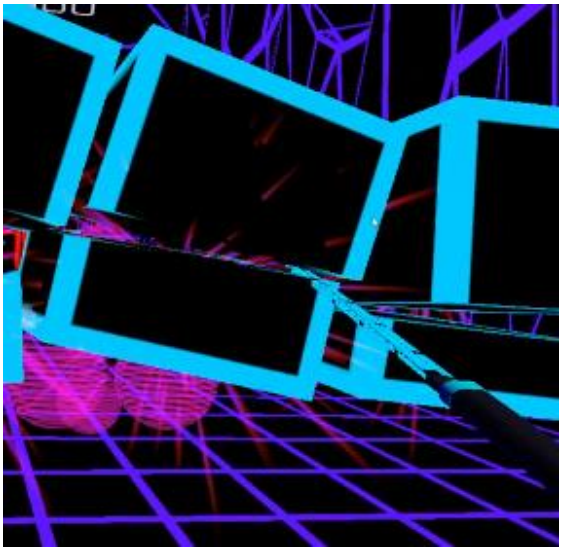
Για να μπορούν να εκτελεστούν διάφορες λειτουργίες μέσα στο παιχνίδι, που η κάθε μια ειναι μια ξεχωριστή και μεγάλη υλοποίηση απο μονή της, χρησιμοποιήθηκαν επεκτάσεις που είχαν ήδη υλοποιηθεί απο άλλους δημιουργούς της κοινότητας. Οι περισσότερες επεκτάσεις που χρησιμοποιήθηκαν υπάρχουν στο ηλεκτρονικό κατάστημα της Unity, το Asset Store και άλλες στο GitHub.

8.1 Mesh Splitting

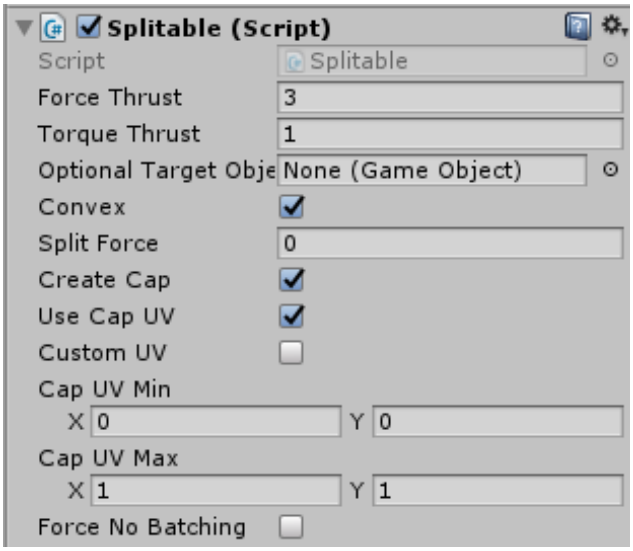
Το Mesh Splitting ειναι μια επέκταση που περιέχει όλες τις απαραίτητες λειτουργίες για να δημιουργήσει το εφέ κοπής ενός αντικειμένου, μέσα στην Unity. Η συγκεκριμένη επέκταση υπάρχει μόνο στο GitHub.

Η επέκταση περιέχει αρκετές λειτουργίες για να χρησιμοποιηθούν. Στο συγκεκριμένο παιχνίδι έγινε χρήση μόνο των βασικών λειτουργιών. Έτσι για την κοπή κάποιου αντικειμένου (Mesh Split), χρειάζονται δυο πραγματα. Το αντικείμενο που πρόκειται να κοπεί και το όπλο (Lightsaber - Saber) που θα χρησιμοποιηθεί απο τον παίκτη για να εκτελέσει αυτή την κοπή.

Εικόνα 264 - Υπόδειξη κοπής μέσα στο παιχνίδι



Εικόνα 265 - Ιδιότητες της κλάσης Splitable



Έτσι σε κάθε αντικείμενο (Streaming Asset) πρεπει να περιέχεται το Splittable Script ώστε να εφαρμόζονται σε κάθε τέτοιο αντικείμενο οι λειτουργίες της Mesh Splitting επέκτασης.

Εικόνα 266 - Ιδιότητες της κλάσης WeaponSplitController

Επίσης τα αντικείμενα των όπλων πρεπει να περιέχουν το Script WeaponSplitController, έτσι ώστε να αναγνωρίζει η επέκταση ποιο αντικείμενο - όπλο χρησιμοποιείται για να εφαρμόσει την κοπή (να κανει τον διαχωρισμό - split), σε κάποιο Splittable αντικείμενο.

Εικόνα 267 - Εντολές της κλάσης Splitable

```
GameObject[] newGOs = new GameObject[2];
if (OptionalTargetObject == null)
{
    newGOs[0] = Instantiate(gameObject) as GameObject;
    newGOs[0].name = gameObject.name;
    newGOs[1] = gameObject;

    // My Changes ////////////////////////////////////////////////
    newGOs[0].transform.parent = null;
    newGOs[1].transform.parent = null;

    newGOs[0].gameObject.GetComponent<Rigidbody>().isKinematic = false;
    newGOs[0].gameObject.GetComponent<Rigidbody>().useGravity = true;
    newGOs[1].gameObject.GetComponent<Rigidbody>().isKinematic = false;
    newGOs[1].gameObject.GetComponent<Rigidbody>().useGravity = true;

    //Adding Thrust Force to GameObjects
    newGOs[0].gameObject.GetComponent<Rigidbody>()
        .AddForce(ForceThrust, ForceThrust, 0, ForceMode.Impulse);
    newGOs[1].gameObject.GetComponent<Rigidbody>()
        .AddForce(-ForceThrust, ForceThrust, 0, ForceMode.Impulse);
    //Adding Rotation to Gameobjects
    newGOs[0].gameObject.GetComponent<Rigidbody>()
        .AddTorque(new Vector3(TorqueThrust, 0, 0), ForceMode.Impulse);
    newGOs[1].gameObject.GetComponent<Rigidbody>()
        .AddTorque(new Vector3(-TorqueThrust, 0, 0), ForceMode.Impulse);

    // My Changes ////////////////////////////////////////////////////

    DestroyImmediate(newGOs[0].GetComponent<Splitable>(), true);
    DestroyImmediate(newGOs[1].GetComponent<Splitable>(), true);
```

Η συνθήκη που φαίνεται στην παραπάνω εικόνα της συνάρτησης CreateNewObjects() εχει προσαρμοστεί ώστε να εφαρμοστούν κάποια στοιχεία φασικής στα αντικείμενα που δημιουργούνται.

Εικόνα 268 - Συνθήκη της Splitable

```
else
{
    newGOs[0] = Instantiate(OptionalTargetObject) as GameObject;
    newGOs[1] = Instantiate(OptionalTargetObject) as GameObject;
}
```

Αυτό που κανει αυτή η συνθήκη ειναι να ελέγξει, αν το αντικείμενο που πρόκειται να διαχωριστεί, ειναι δημιουργημένο απο την Splittable κλάση (δηλαδή αν ειναι ένα απο τα ήδη διαχωρισμένα αντικείμενα).

Αν ειναι δημιουργημένο απο αυτήν τότε εφαρμόζονται στα δυο αντικείμενα προσαρμοσμένες ιδιότητες φυσικής και αφαιρείται η ιδιότητα Splittable απο αυτά τα δυο αντικείμενα, ώστε να μην μπορούν να διαχωριστούν τα ήδη διαχωρισμένα αντικείμενα.

Αν δεν ειναι δημιουργημένο απο αυτήν, σημαίνει ότι το όπλο έκανε Collide (ακούμπησε) με ένα Splittable αντικείμενο που δεν εχει διαχωριστεί και το διαχωρίζει κάνοντας instantiate δυο καινούρια αντικείμενα.

Εικόνα 269 - Συνάρτηση - Event της κλάσης WeaponSplitController

```
public void OnTriggerEnter(Collider other)
{
    //Make objects have gravity once they are split.
    if (other.gameObject.tag.Equals("SpawnObject") &&
        !other.gameObject.name.Equals("SpherePrefab(Clone)") &&
        other.gameObject.GetComponent<Splitable>() != null)
    {
        //Play Particle System
        other.transform.GetChild(0).GetChild(0).
            GetComponent<ParticleSystem>().Play();
        other.transform.GetChild(0).GetChild(1).
            GetComponent<ParticleSystem>().Play();

        if ((WeaponColor.Equals("Red") &&
            other.gameObject.name.Equals("RedCubePrefab(Clone)")) ||
            (WeaponColor.Equals("Blue")
            && other.gameObject.name.Equals("BlueCubePrefab(Clone)")))
        {
            AddScorePoints(10);
            PlaySfxOnMode(true, other);
        }
        else
        {
            RemoveScorePoints(10);
            PlaySfxOnMode(false, other);
        }
    }
```

```
    else if (other.gameObject.tag.Equals("SpawnObject") &&
        other.gameObject.name.Equals("SpherePrefab(Clone)") &&
        other.gameObject.GetComponent<Splitable>() != null)
    {
        //Play Particle System
        other.transform.GetChild(0).GetChild(0)
            .GetComponent<ParticleSystem>().Play();
        other.transform.GetChild(0).GetChild(1)
            .GetComponent<ParticleSystem>().Play();

        RemoveScorePoints(20);
        SoundManager.instance.PlaySingle(
            other.GetComponent<AudioClipSFX>().audioClip[0]
        );
    }

    collisionEnterPos = transform.position;
}
```

Το OnTriggerEnter() ειναι ένα event που εκτελείται οταν ένα όπλο έρθει σε επαφή (ακουμπήσει) με ένα Splittable αντικείμενο. Σε αυτή την συνάρτηση ελέγχονται όλες οι συνθήκες που μπορεί να υπάρχουν σε κάθε mode του ρυθμικού μέρους του παιχνιδιού, και εκτελούνται οι ανάλογες λειτουργίες. Αυτές οι λειτουργίες έχουν αναφερθεί σε προηγούμενα κεφάλαια.

Το βασικό ειναι ότι αποθηκεύεται η θέση (Position) , που εγινε Collide (ακούμπησε) το όπλο με το αντικείμενο, ώστε να χρησιμοποιηθεί στην OnTriggerExit().

Εικόνα 270 - Συνάρτηση - Event της κλάσης WeaponSplitController

```
void OnTriggerExit(Collider other)
{
    collisionExitPos = transform.position;
    CreateCutPlane(collisionEnterPos,
                   collisionExitPos,
                   bladeCollider.transform.up);
}
```

Η OnTriggerExit() ειναι και αυτή ένα event, που εκτελείται οταν το όπλο σταματήσει να εφάπτεται με το αντικείμενο, δηλαδή εξέλθει απο το αντικείμενο (δεν γίνεται Collide).

Οταν εκτελεστεί αυτό το event δημιουργείται μια αόρατη επιφάνεια (ένα Plane), πάνω στο όπλο, η οποία ειναι υπεύθυνη να χωρίσει το αντικείμενο στα δυο με την χρήση της συνάρτησης CreateCutPlane() που υπάρχει στην WeaponSplitController.

Εικόνα 271 - Συνάρτηση της κλάσης WeaponSplitController

```
private void CreateCutPlane(Vector3 startPos, Vector3 endPos, Vector3 forward)
{
    Vector3 center = Vector3.Lerp(startPos, endPos, .5f);
    Vector3 cut = (endPos - startPos).normalized;
    Vector3 fwd = forward.normalized;
    Vector3 normal = Vector3.Cross(fwd, cut).normalized;

    GameObject goCutPlane = new GameObject("CutPlane",
        typeof(BoxCollider), typeof(Rigidbody), typeof(SplitterSingleCut));

    goCutPlane.GetComponent<Collider>().isTrigger = true;
    Rigidbody bodyCutPlane = goCutPlane.GetComponent<Rigidbody>();
    bodyCutPlane.useGravity = false;
    bodyCutPlane.isKinematic = true;

    Transform transformCutPlane = goCutPlane.transform;
    transformCutPlane.position = center;
    transformCutPlane.localScale = new Vector3(CutPlaneSize, .01f, CutPlaneSize);
    transformCutPlane.up = normal;
    float angleFwd = Vector3.Angle(transformCutPlane.forward, fwd);
    transformCutPlane.RotateAround(center, normal, normal.y < 0f ? -angleFwd : angleFwd);
}
```

Αυτή η συνάρτηση παίρνει τις ιδιότητες του διαχωρισμού, όπως το σημείο που εισήλθε το όπλο στο αντικείμενο (Vector3 startPos), καθώς και το σημείο που εξήλθε (Vector3 endPos) και υπολογίζει μέσω άλλων συναρτήσεων της επέκτασης, τον τροπο που θα δημιουργήσει τα δυο (διαχωρισμένα) αντικείμενα ώστε να ειναι συμμετρικά.

(DanniSchou, 2012)

8.2 Οπτικός ελεγκτής εικονικής πραγματικότητας

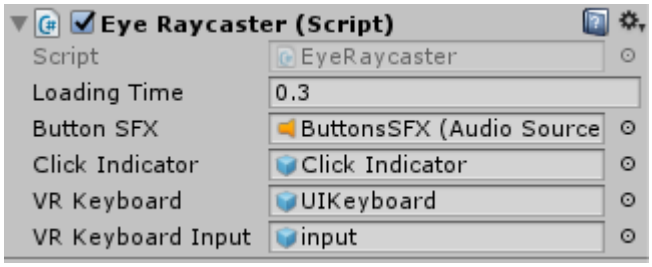
Η επέκταση που χρησιμοποιήθηκε ονομάζεται Gaze UI και ειναι απαραίτητη σε αρκετές υλοποιήσεις εφαρμογών εικονικής πραγματικότητας, καθώς ειναι μια υλοποίηση που λειτουργεί ως δείκτης επίλογων (κατι αντίστοιχο με την λειτουργία της συσκευής mouse). Έτσι εγινε χρήση αυτής της επέκτασης για να μπορεί ο χρήστης να αλληλεπιδράσει αποκλειστικά με τις διεπαφές που χρησιμοποιήθηκαν μέσα στο παιχνίδι.

Εικόνα 272 - Εικονίδιο που αντιπροσωπεύει το ray

Εικόνα 273 - Αντικείμενα στην ιεραρχία κάθε σκηνής

Στις παραπάνω εικόνες φαίνεται η διεπαφή αυτού του δείκτη και το αντικείμενο στην ιεραρχία της σκηνής, που αποτελείται απο τον κύκλο (Indicator) και την κουκίδα στην μέση (Reticle). Το EyeRaycaster χρησιμοποιείται σε όλες τις σκηνές του παιχνιδιού, καθώς όλες οι σκηνές περιέχουν διεπαφές που ειναι απαραίτητες για το παιχνίδι.

Εικόνα 274 - Ιδιότητες και τα πεδία της κλάσης EyeRaycaster



Σε αυτή την εικόνα φαίνονται οι ιδιότητες της κλάσης, του EyeRaycaster, οι οποίες έχουν προσαρμοστεί για την ιδιαίτερη χρήση αυτής της επέκτασης μέσα στο παιχνίδι.

- **EyeRaycaster**

Εικόνα 275 - Εντολή της κλάσης EyeRaycaster

```
// Set pointer position
m_pointerEvent.position = new Vector3(
    XRSettings.eyeTextureWidth / 2,
    XRSettings.eyeTextureHeight / 2,
    m_pointerEvent.pointerCurrentRaycast.distance);
```

Αρχικά το εικονίδιο που λειτουργεί ως δείκτης, τοποθετείται στην μέση των οθονών του VR Headset ώστε να μπορεί να το βλέπει ο παίκτης κανονικά μέσα στο παιχνίδι. Αυτό γίνεται με την βοήθεια της XRSettings που βρίσκει και χρησιμοποιεί την ανάλυση που εχει το συγκεκριμένο Headset.

Εικόνα 276 - Συνθήκη στην κλάση EyeRaycaster

```
// Detect selectable
if (raycastResults.Count > 0)
{
    foreach (var result in raycastResults)
    {
        //Changes Canvase's distance based on the object distance
        GetComponent<Canvas>().planeDistance = result.distance;
        newSelectable =
            result.gameObject.GetComponentInParent<Selectable>();

        if (newSelectable)...
    }
}
```

Στην Update() της κλάσης ελέγχεται κάθε καρέ για τα αντικείμενα που «βλέπει» το Ray του EyeRaycaster. Έτσι ελέγχεται αν το αποτέλεσμα του Ray (της ακτίνας εχει το εικονίδιο του δείκτη) ηταν κάποιο καινούριο αντικείμενο. Επίσης για κάθε καινούριο αντικείμενο διεπαφής, ο καμβάς (Canvas) με το εικονίδιο του δείκτη (Indicator), αλλάζει ώστε να προσαρμόζεται και να ειναι διακριτό, ανεξαρτήτως της απόστασης της διεπαφής.

Εικόνα 277 - Συνθήκη στην κλάση EyeRaycaster

```
if (newSelectable)
{
    ClickIndicator.SetActive(true); //Activate ClickIndicator

    if (SteamVR_Input.GetStateDown(("Squeeze"), SteamVR_Input_Sources.LeftHand))
    {
        Select(newSelectable);
        m_currentRaycastResult = result;
        ButtonSFX.Play();

        // Activate VRKeyboard when player ckicks on the Input Field
        if (newSelectable.gameObject.GetComponent<InputField>())...
        else if (!newSelectable.gameObject.GetComponent<InputField>() &&
                 !newSelectable.gameObject.GetComponent<Button>() &&
                 !newSelectable.gameObject.tag.Equals("VRGazeInteractable"))
        {
            VRKeyboard.SetActive(false);
            isInputField = false;
        }
    }
    break;
}
```

Αν υπάρξει καινούριο αντικείμενο διεπαφής, ελέγχεται κάθε φορά αν ο παίκτης πάτησε την σκανδάλη του αριστερού χειριστηρίου, ώστε να εκτελεστούν ενέργειες για κάθε διεπαφή. Αυτό αντιπροσωπεύει το αντίστοιχο κλικ του ποντικιού ενός υπολογιστή.

Εικόνα 278 - Συνθήκη στην κλάση EyeRaycaster

```
// Activate VRKeyboard when player ckicks on the Input Field
if (newSelectable.gameObject.GetComponent<InputField>())
{
    tempInputField = newSelectable.gameObject.GetComponent<InputField>();
    VRKeyboard.SetActive(true);

    VRKeyboard.gameObject.transform.position =
        newSelectable.gameObject.transform.position;
    RectTransform rectTrans = (RectTransform)VRKeyboard.gameObject.transform;

    VRKeyboard.gameObject.transform.position = new Vector3(
        newSelectable.gameObject.transform.position.x,
        newSelectable.gameObject.transform.position.y - 0.05f,
        newSelectable.gameObject.transform.position.z - 0.2f);
}
```

Αν το αντικείμενο ειναι newSelectable και ο παίκτης πατήσει την σκανδάλη του χειριστηρίου, γίνεται άλλος ένας έλεγχος αν το newSelectable αντικείμενο ειναι διεπαφή τύπου Input Field. Αν ειναι Input Field τότε ενεργοποιείται το VRKeyboard (εικονικό πληκτρολόγιο) και αλλάζει την θέση που βρίσκεται, και τοποθετείται στην θέση που βρίσκεται η διεπαφή Input Field.

Εικόνα 279 - Συνάρτηση της κλάσης EyeRaycaster

```
void Select(Selectable s, Selectable exclude = null)
{
    m_excluded = exclude;

    if (m_currentSelectable)
        m_currentSelectable.OnPointerExit(m_pointerEvent);

    m_currentSelectable = s;

    if (m_currentSelectable)
    {
        m_currentSelectable.OnPointerEnter(m_pointerEvent);
        m_clickHandler = m_currentSelectable
            .GetComponent<IPointerClickHandler>();
        m_dragHandler = m_currentSelectable
            .GetComponent<IDragHandler>();
    }

    m_elapsedTime = 0;
}
```

Στην περίπτωση που δεν ειναι Input Field απλά απενεργοποιεί το VRKeyboard, αν ειναι ενεργοποιημένο και καλεί την συνάρτηση Select() που εκτελεί ένα event ,που κανει «κλικ» μέσω του click Handler οταν το Selectable αντικείμενο ειναι διεπαφή τύπου κουμπί.

Εικόνα 280 - Συνάρτηση στην κλάση VRKeyboardInput

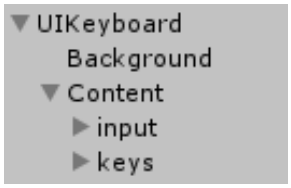
```
//When ENTER is pressed on the VRKeyboard
//it passes the text to the temporary input field
public void OnEnter()
{
    tempInputField.text =
        VRKeyboardInput.GetComponent<Text>().text;
}
```

Η παραπάνω συνάρτηση OnEnter() εκτελείται οταν το κουμπί που εχει πατήσει ο παίκτης με το EyeRaycaster ειναι το κουμπί Enter απο το πληκτρολόγιο VRKeyboard. Οταν πατηθεί το Enter θα περαστεί το κείμενο που έγραψε ο παίκτης με το VRKeyboard, στο Input Field που είχε επιλέξει αυτή την χρονική στιγμή στο παιχνίδι.

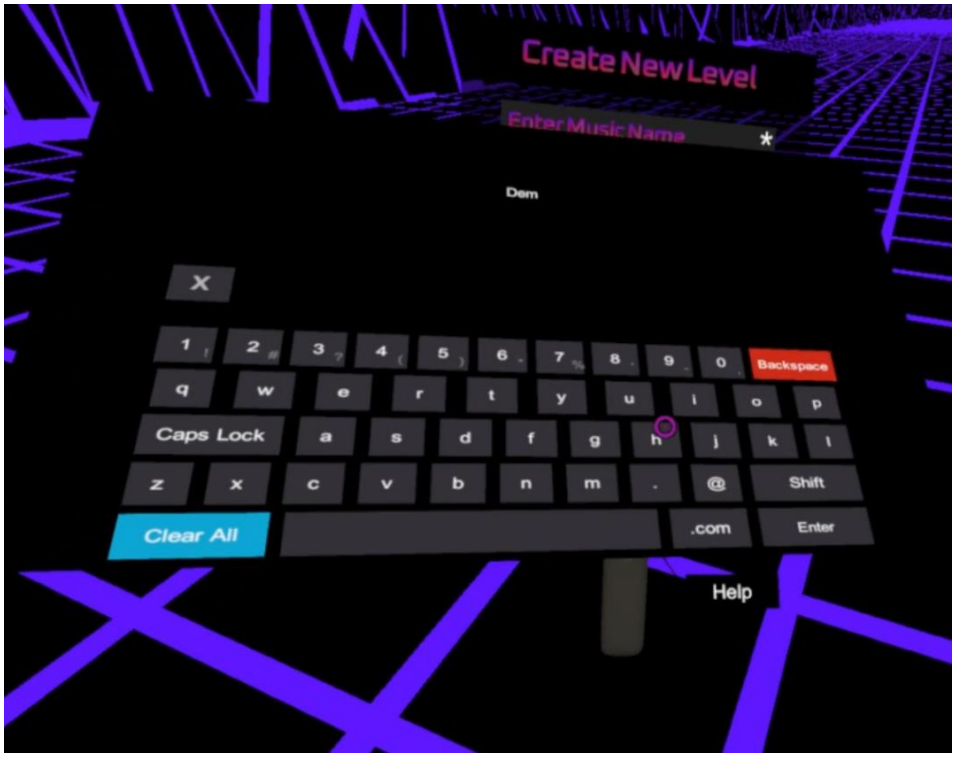
(Graillot, 2019)

8.3 Πληκτρολόγιο Εικονικής Πραγματικότητας

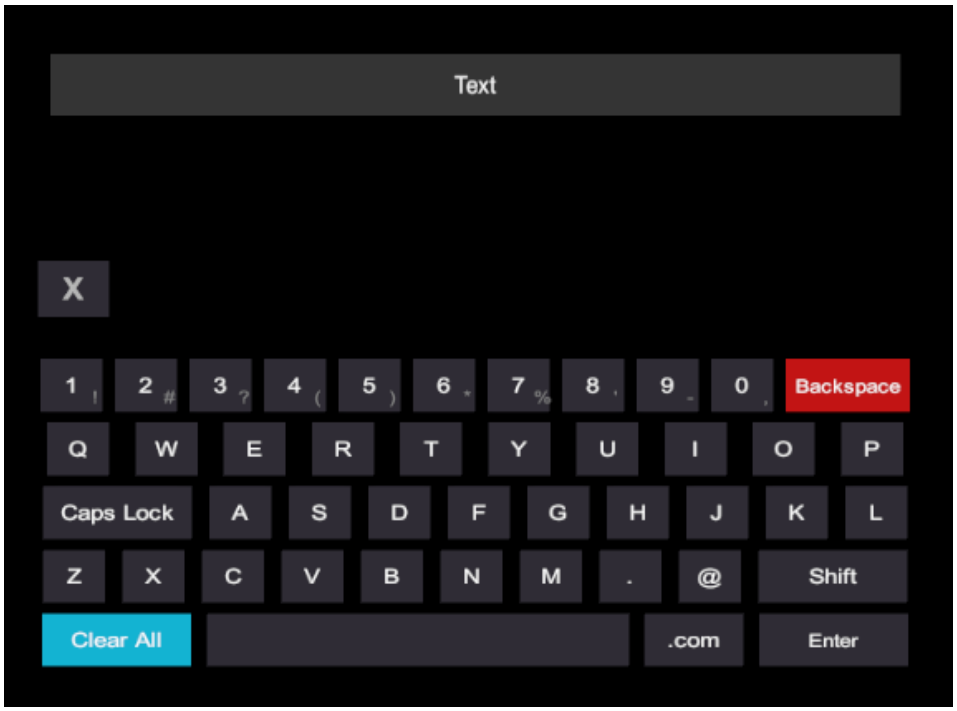
Εικόνα 281 - Αντικείμενα στην ιεραρχία κάθε σκηνής



Εικόνα 282 - Υπόδειξη του VRKeyboard μέσα στο παιχνίδι



Εικόνα 283 - Υπόδειξη του VRKeyboard

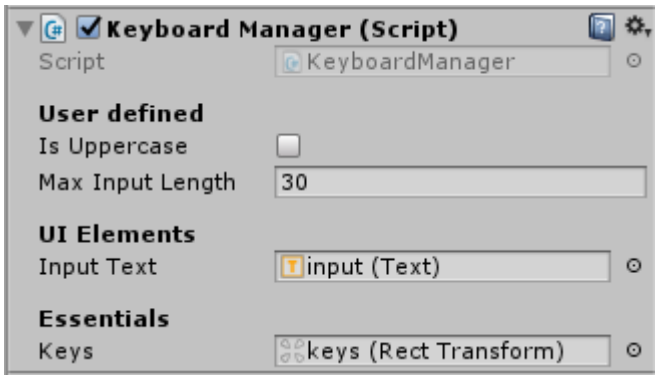


Για την εισαγωγή κειμένου στον εικονικό κόσμο χωρίς κάποια εξτρά συσκευή (όπως το πληκτρολόγιο), ειναι απαραίτητη η χρήση κάποιας υλοποιημένης διεπαφής, εικονικού

πληκτρολογίου. Σε αυτό το παιχνίδι, εγινε χρήση της επέκτασης VRKeyboard απο το AssetSotre της Unity.

Στις παραπάνω εικόνες φαίνεται η διεπαφή VRKeyboard και η χρήση της, καθώς και το περιεχόμενο της στην ιεραρχία της σκηνής.

Εικόνα 284 - Ιδιότητες της κλάσης KeyboardManager

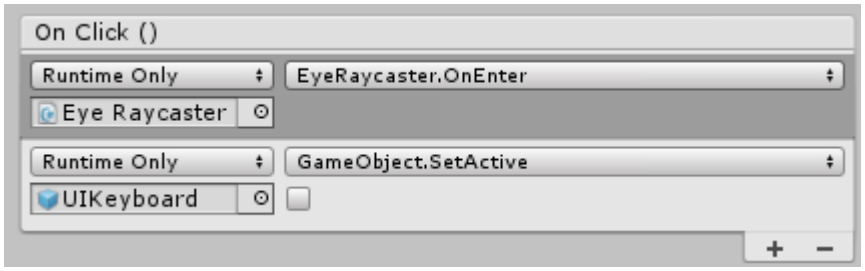


Στην παραπάνω εικόνα φαίνεται το Script (KeyboardManager) και οι ιδιότητες, που χρησιμοποιεί η διεπαφή VRKeyboard, ώστε τα κουμπιά αυτής της διεπαφής να ειναι λειτουργικά.

Οι μόνες προσαρμογές που έγιναν στο VRKeyboard ηταν στο κουμπί Enter και το κουμπί X (Exit Button) που ειναι για το κλείσιμο (απόκρυψη) αυτής της διεπαφής.

- **Enter**

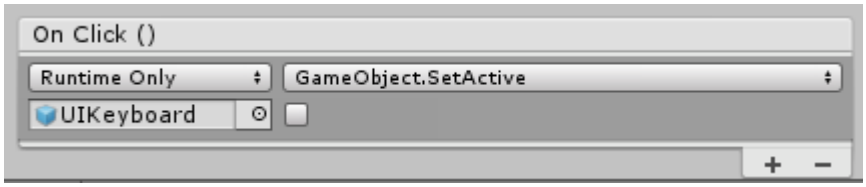
Εικόνα 285 - Λειτουργίες του Enter Button



Το κουμπί Enter οταν πατηθεί εκτελεί την συνάρτηση OnEnter() του EyeRaycaster ώστε να μεταφέρει το κείμενο απο το input του UIKeyboard σε κάποιο Input Field. Αφού μεταφερθεί το κείμενο, το VRKeyboard απενεργοποιείται (αποκρύπτεται).

- **Exit**

Εικόνα 286 - Λειτουργίες του Exit Button



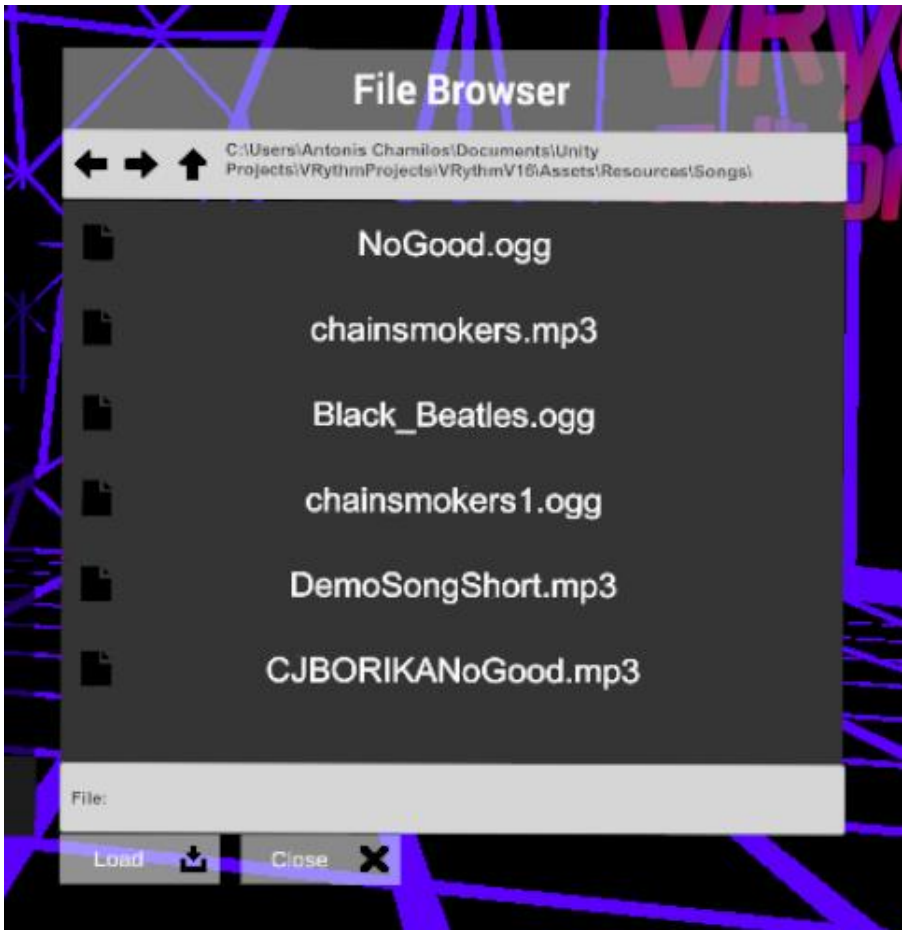
Ενώ το κουμπί X που ειναι το Exit Button, οταν πατηθεί, απλά θα κανει απόκρυψη του VRKeyboard.

(Li, 2018)

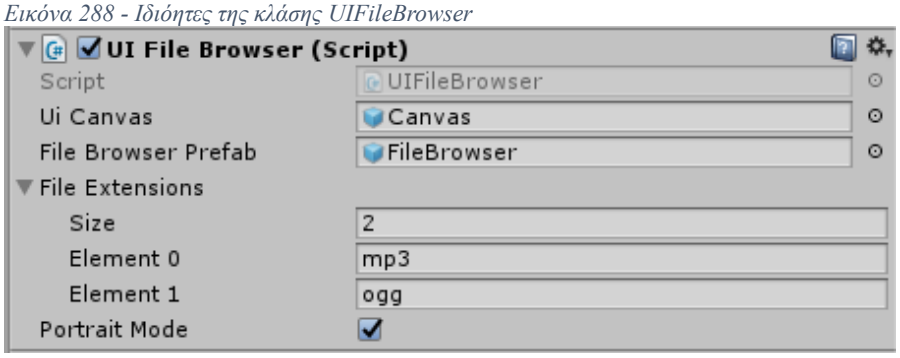
8.4 Πρόγραμμα Περιήγησης Αρχείων

Σε αυτό το παιχνίδι, ειναι απαραίτητη η χρήση ενός περιηγητή αρχείων, καθώς πρεπει να γίνει φόρτωση μουσικών και JSON αρχείων. Έτσι μόνο με αυτόν τον τροπο δίνεται η δυνατότητα στον παίκτη να επιλέξει κάθε φορά το μουσικό κομμάτι θέλει να φορτώσει είτε για επεξεργασία είτε για να παίξει στο ρυθμικό μέρος με αυτό. Η παρακάτω εικόνα δείχνει το File Browser που χρησιμοποιείται και στο ρυθμικό και στο επεξεργαστικό μέρος του παιχνιδιού.

Εικόνα 287 - Υπόδειξη του περιηγητή αρχείων μέσα στο παιχνίδι

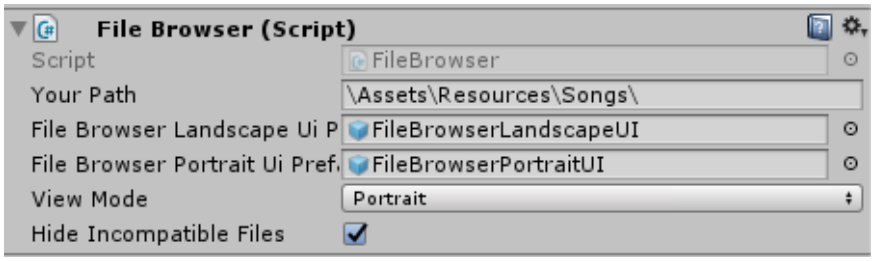


Εικόνα 288 - Ιδιότητες της κλάσης UIFileBrowser



Το UIFileBrowser ειναι το script που ρυθμίζει τον τροπο λειτουργίας της διεπαφής του file browser. Έτσι εχει επιλεχτεί να λειτουργεί με ιδιότητες πορτρέτου (Portrait Mode). Επίσης εχει οριστεί να φορτώνει μόνο .mp3 και .ogg αρχεία, καθώς στο παιχνίδι χρειάζεται να φορτώνονται μόνο μουσικά αρχεία. Σε περίπτωση που τα αρχεία στον φάκελο δεν ειναι mp3 ή ogg τότε δεν θα τα εμφανίσει στην διεπαφή.

Εικόνα 289 - Ιδιότητες της κλάσης File Browser



Το UIFileBrowser προσαρμόζει και εκτελεί το File Browser, με βάση τα ορίσματα που του δόθηκαν. Το File Browser Script παίρνει και αυτό ορίσματα. Τα βασικά ειναι το path (μονοπάτι τοποθεσίας), του φακέλου που περιέχει τα μουσικά αρχεία, και ο τρόπος εμφάνισης του File Browser που εχει οριστεί ως πορτραίτο.

- **UIFileBrowser Script**

Εικόνα 290 - Συνάρτηση της κλάσης UIFileBrowser

```
// Open the file browser using boolean parameter so it can be called in GUI
public void OpenFileBrowser(bool saving)
{
    OpenFileBrowser(saving ? FileBrowserMode.Save : FileBrowserMode.Load);
}
```

Εικόνα 291 - Συνάρτηση της κλάσης UIFileBrowser

```
// Open a file browser to save and load files
protected void OpenFileBrowser(FileBrowserMode fileBrowserMode)
{
    // Create the file browser and name it
    GameObject fileBrowserObject = Instantiate(FileBrowserPrefab, transform);
    fileBrowserObject.name = "FileBrowser";
    // Set the mode to save or load
    FileBrowser fileBrowserScript = fileBrowserObject.GetComponent<FileBrowser>();
    fileBrowserScript.SetupFileBrowser(
        PortraitMode ? ViewMode.Portrait : ViewMode.Landscape);

    if (fileBrowserMode == FileBrowserMode.Save)
    {
        fileBrowserScript.SaveFilePanel("DemoText", FileExtensions);
        // Subscribe to OnFileSelect event (call SaveFileUsingPath using path)
        fileBrowserScript.OnFileSelect += SaveFileUsingPath;
    }
    else
    {
        fileBrowserScript.OpenFilePanel(FileExtensions);
        // Subscribe to OnFileSelect event (call LoadFileUsingPath using path)
        fileBrowserScript.OnFileSelect += LoadFileUsingPath;
    }
}
```

Η πρώτη εντολή που εκτελείται ώστε να γίνει η δημιουργία του File Browser στην ιεραρχία της κάθε σκηνής ειναι το OpenFileBrowser(), που καθορίζει την λειτουργία και την εμφάνιση του File Browser. Υπαρχουν δυο λειτουργίες, το Save και το Load. Όπως αναφέρθηκε σε προηγουμένη ενότητα στο συγκεκριμένο παιχνίδι χρησιμοποιείται μόνο η λειτουργία Load για το File Browser.

- **LoadFileUsingPath**

Η συνάρτηση LoadSongCoroutine(), εκτελείται μέσα απο την συνάρτηση LoadFileUsingPath(), και ειναι υπεύθυνη για την φόρτωση των μουσικών αρχείων και των JSON αρχείων στο παιχνίδι.

Εικόνα 292 - Συνάρτηση της κλάσης UIFileBrowser

```
IEnumerator LoadSongCoroutine(string path)
{
    using (WWW www = new WWW(path))
    {
        //Refactoring Path String - get useful parts
        string strReplace = path.Replace("/", "\\");
        string usefullStr = strReplace.Replace(
            System.IO.Directory.GetCurrentDirectory() + "\\Assets", "");
        string[] x = usefullStr.Split('\\');
        string songNameOnly = x[3].Substring(0, x[3].Length - 4);
        string songNamePath = x[2] + "\\" + songNameOnly;
```

Όπως φαίνεται στην παραπάνω εικόνα γίνονται προσαρμογές στο string (κείμενο) του path (τοποθεσίας), ώστε να αποσπαστούν κάποια κομμάτια αυτής της πληροφορίας και να χρησιμοποιηθούν σε άλλες λειτουργίες του παιχνιδιού. Οι string μεταβλητές που χρησιμοποιούνται για να ενημερωνόσουν το Game Manager της σκηνής ειναι δυο. Η «songNameOnly», που κρατάει μόνο το όνομα του μουσικού κομματιού μετα απο αποκοπή του υπολοίπου path και της κατάληξης του τύπου αρχείου. Αλλα και η «songNamePath» που κρατάει μόνο το τελευταίο κομμάτι απο το path και προσδεθεί και το songNameOnly. Αυτές οι μεταβλητές χρησιμοποιούνται μέσω του Game Manager στις λειτουργίες της ιδίας της συνάρτησης, καθώς και σε άλλες λειτουργίες του παιχνιδιού.

Εικόνα 293 - Εντολές μεσα στην συνάρτηση LoadSongCoroutine

```
GameObject gameManager = GameObject.Find("GameManager");
AudioSource audioSrc = gameManager
    .GetComponent<AudioSource>() as AudioSource;

//Assigning Audio Clip for GameManager AudioSource
AudioClip audioClip = Resources.Load<AudioClip>(songNamePath);
audioSrc.clip = audioClip;

//Assinging variables to GameManager
gameManager.GetComponent<GameManager>()
    .SongTimeSeconds = audioClip.length;
gameManager.GetComponent<GameManager>()
    .CurrentSongNamePath = songNamePath;

//LoadJSON
GameObject.Find("JSON_Operations")
    .GetComponent<JsonOperations>().LoadJSON();

//Assinging variables to GameManager
gameManager.GetComponent<GameManager>().SongTitle =
    GameManager.allObjects.properties[0].SongTitle;

gameManager.GetComponent<GameManager>().BeatsPerMinute =
   GameManager.allObjects.properties[0].BeatsPerMinute;
```

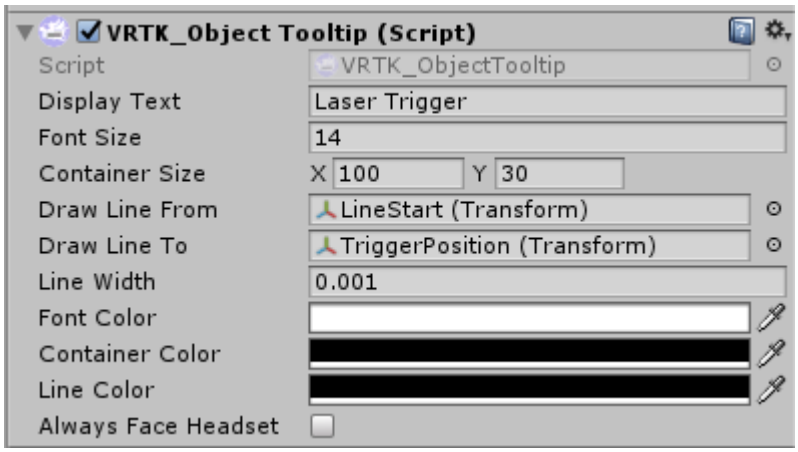
Μετά που θα γίνει η προσαρμογή του path, η συνάρτηση εκτελεί και άλλες λειτουργίες. Αρχικά φορτώνει το μουσικό κομμάτι με την χρήση του μονοπατιού songNamePath και θα το περάσει στο Audio Source του Game Manager. Περνάει στον Game Manger της σκηνής, την διάρκεια του μουσικού κομματιού (SongTimeSeconds) και το string με το μονοπάτι του (songNamePath). Φορτώνει τα JSON αρχεία στο παιχνίδι με την συνάρτηση LoadJSON() της κλάσης JsonOperations και ενημερώνει τα πεδία του μουσικού κομματιού (allObjects.properties) στο παιχνίδι. Αφού ενημερωθούν τα πεδία στο allObjects.properties, γίνεται ενημέρωση – αντιγραφή αυτών των πεδίων, στις μεταβλητές του Game Manger. Αυτά τα πεδία ειναι ο τίτλος του μουσικού κομματιού (SongTitle) και ο ρυθμός αναπαραγωγής του (BeatsPerMinute).

(Graces Games, 2018)

8.5 Εικονικές Υποδείξεις με το λογισμικό VRTK

Οι εικονικές υποδείξεις (Tooltips) της επέκτασης VRTK, ειναι αρκετά σημαντικές μέσα στο παιχνίδι, καθώς βοηθάνε τον παίκτη στην εκμάθηση, της χρήσης του παιχνιδιού. Η χρήση τους γίνεται πάνω στα εικονικά χειριστήρια του παίκτη. Αν δεν υπήρχαν οι υποδείξεις ίσως ηταν δύσκολο στον παίκτη να προηγηθεί σε αυτό. Έτσι διπλά απο κάθε κουμπί του χειριστηρίου υπάρχει και ένα tooltip που δείχνει την λειτουργία του.

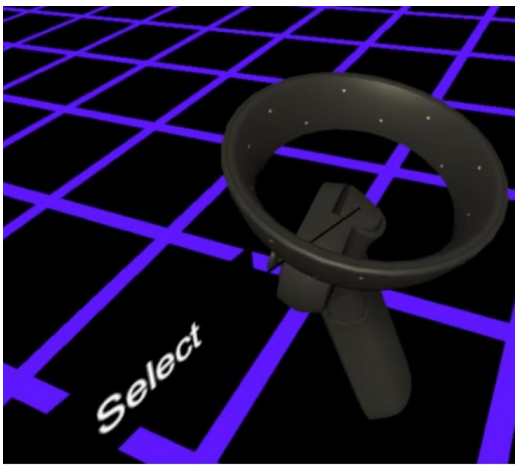
Εικόνα 294 - Ιδιότητες της κλάσης VRTK_ObjectTooltip



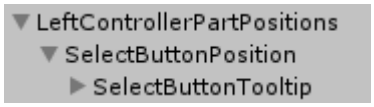
Στην παραπάνω εικόνα φαίνονται οι ιδιότητες του Script που χρησιμοποιούν όλα τα Tooltips. Οι ιδιότητες που αλλάζουν σε κάθε tooltip εκτός απο το κείμενο, ειναι οι γραμμές που συνδέουν το κάθε κουμπί του χειριστηρίου με την διεπαφή που περιέχει το κείμενο υπόδειξης. Έτσι το script χρησιμοποίει 2 σημεία (Transform για την Unity), το σημείο που ξεκινάει η γραμμή (Draw Line From) και το σημείο που καταλήγει (Draw Line To).

- **Αρχική σκηνή και σκηνή ρυθμικού μέρους**

Εικόνα 295 - Υπόδειξη του Tooltip στο αριστερό χειριστήριο της αρχικής σκηνής



Εικόνα 296 - Αντικείμενα του αριστερού χειριστηρίου στην ιεραρχία της σκηνής



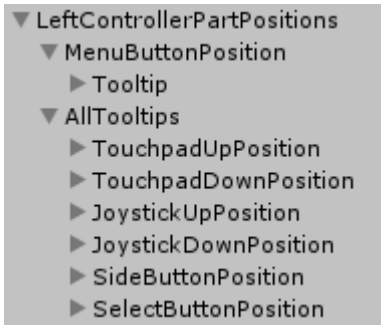
Στις παραπάνω εικόνες φαίνεται η χρήση των tooltips στην αρχική σκηνή καθώς και στην σκηνή του ρυθμικού μέρους. Σε αυτές τις σκηνές γίνεται χρήση μόνο ενός tooltip, που υπάρχει στην σκανδάλη του αριστερού χειριστηρίου, που υποδεικνύει την λειτουργία «επιλογή» (Select).

- **Σκηνή επεξεργαστικού μέρους.**

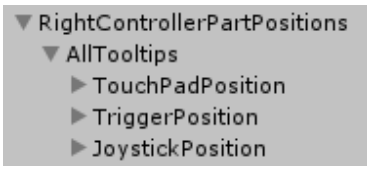
Οι πιο πολλές λειτουργίες που γίνονται με την χρήση των χειριστηρίων, ειναι στην σκηνή του επεξεργαστικού μέρους. Σε αυτή την σκηνή σχεδόν όλα τα κουμπιά των χειριστηρίων έχουν κάποια λειτουργικότητα. Στις παρακάτω εικόνες φαίνονται τα αντικείμενα tooltip στην ιεραρχία της σκηνής, για κάθε χειριστήριο.

Εικόνα 297 - Υπόδειξη του Tooltip στο αριστερό χειριστήριο, στο επεξεργαστικό μέρος

Εικόνα 298 - Αντικείμενα του αριστερού χειριστηρίου στην ιεραρχία της σκηνής



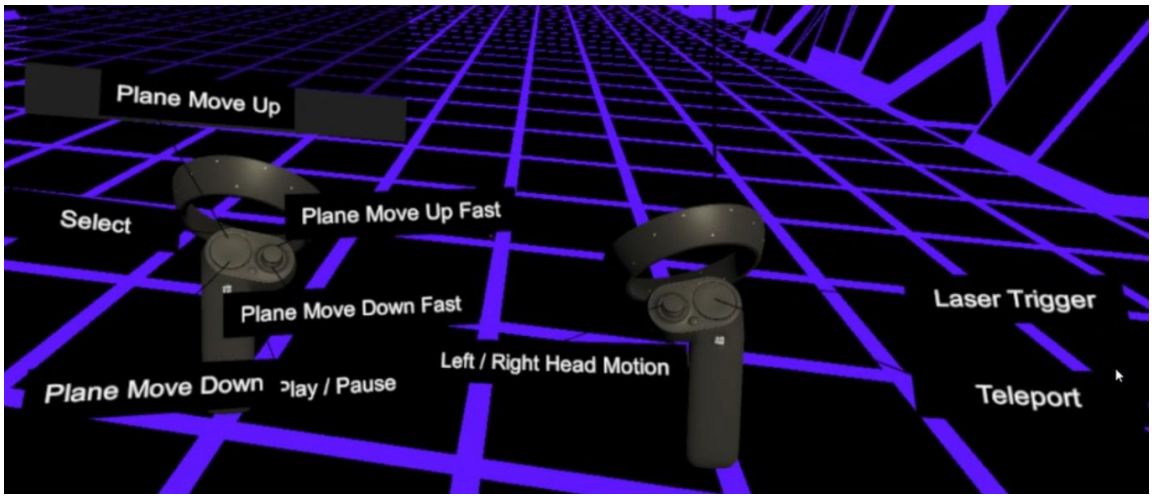
Εικόνα 299 - Αντικείμενα του δεξιού χειριστηρίου στην ιεραρχία της σκηνής



Στο αριστερό χειριστήριο υπάρχει συνεχόμενα ενεργό το tooltip «Help», που συνδέεται στο κουμπί «Menu», του χειριστηρίου. Και χρησιμοποιείται σε περίπτωση που ο παίκτης δεν γνωρίζει τις λειτουργίες των χειριστηρίων.

Μόλις πατηθεί το Menu κουμπί, τότε απενεργοποιείται το «Help» Tooltip και ενεργοποιούνται όλα τα υπόλοιπα, τα οποία δείχνουν σε όλα τα λειτουργικά κουμπιά.

Εικόνα 300 - Υπόδειξη των Tooltips στο επεξεργαστικό μέρος



Έτσι για το αριστερό χειριστήριο η αντιστοίχιση των tooltip με τα κουμπιά ειναι η εξής:

- Select – Trigger Button (Σκανδάλη)
- Play/Pause – GrabGrip Button (Πλαϊνό κουμπί)
- Plane Move Up – Trackpad North (Επιφάνεια αφής)
- Plane Move Down – Trackpad South (Επιφάνεια αφής)
- Plane Move Up Fast – Joystick Up (Μοχλός)
- Plane Move Down Fast - Joystick Down (Μοχλός)

Ενώ για το δεξί χειριστήριο η αντιστοίχιση των tooltip με τα κουμπιά ειναι η εξής:

- Left / Right Head Motion – Joystick Left / Right (Μοχλός)
- Laser Trigger - Trigger Button (Σκανδάλη)
- Teleport – Trackpad Button (Κουμπί επιφάνειας αφής)

(Sysdia Solutions Ltd, 2019)

Επίλογος

Συμπεράσματα

Αυτή η εργασια είχε σκοπό να παρουσιάσει τα στάδια της ανάπτυξης ενός παιχνιδιού που υποστηρίζει τεχνολογία εικονικής πραγματικότητας, με την χρήση της Unity και της επέκτασης SteamVR. Τα πρώτα στάδια, που ηταν η ενσωμάτωση της επέκτασης SteamVR με το παιχνίδι, αποδείχτηκαν δύσκολα και χρονοβόρα, καθώς υπαρχουν ασυμβατότητες και πολλές αλλαγές στις καινούριες εκδόσεις της επέκτασης. Αυτές οι αλλαγές δεν ηταν τεκμηριωμένες και δεν υπήρχαν πολλά μέσα εκμάθησης για την συγκεκριμένη επέκταση στην Unity. Αυτή θα ηταν μια αλλαγή του προτζεκτ, ώστε να χρησιμοποιηθεί μια πιο εύκολη ως προς τον δημιουργό επέκταση, που δουλεύει χωρίς προβλήματα στην unity και που εχει δοκιμαστεί απο περισσοτέρους δημιουργούς. Μετα το στάδιο της ενσωμάτωσης του SteamVR, δεν υπήρξαν αλλα προβλήματα στην ανάπτυξη του παιχνιδιού.

Γενικά ειναι πολυ σημαντικό, στην βιομηχανία των ηλεκτρονικών παιχνιδιών, αλλα και εφαρμογών, να έχουν υπόψιν τους αυτή την τεχνολογία, καθώς με την συνεχόμενη και ραγδαία ανάπτυξη της, δίνεται η δυνατότητα στους παίκτες, να βιώνουν διαφορετικές εμπειρίες. Έτσι αργότερα που η εικονική πραγματικότητα πιθανώς να ειναι ένα βασικό μέσο για την χρήση παιχνιδιών και άλλων πολυμέσων, η βιομηχανία θα στραφεί προς αυτό τον τομέα. Προς το παρόν όπως φαίνεται υπαρχουν ελάχιστοι μεγάλοι τίτλοι παιχνιδιών για VR, καθώς δεν υπάρχει μεγάλο κοινό που μπορεί να τα χρησιμοποιήσει. Αυτό το πρόβλημα δημιουργείται διότι υπαρχουν λίγες υλοποιήσεις συσκευών εικονικής πραγματικότητας και ειναι πολυ ακριβές για τον μέσο καταναλωτή.

Μελλοντικές επεκτάσεις

Το συγκεκριμένο παιχνίδι στοχεύει μόνο στην εμβύθιση και την ψυχαγωγία του παίκτη, αλλα μπορεί να επεκταθεί, και ταυτόχρονα να εχει πρακτική χρήση του. Μια τέτοια επέκταση, θα μπορούσε να ειναι η χρήση των κινήσεων του παίκτη μέσα στο παιχνίδι και να γίνεται υπολογισμός, των θερμίδων που καταναλώνει. Έτσι με αυτό τον τροπο το παιχνίδι στοχεύει και στην εκγύμναση του παίκτη. Εκτός απο την επέκταση του παιχνιδιού, που στοχεύει σε μια διαφορετική χρησιμότητα, μπορούν να γίνουν κάποιες προσθήκες ώστε να γίνει το παιχνίδι πιο προκλητικό στον παίκτη, και κάποιες προσθήκες που αλλάζουν το παιχνίδι και το κάνουν πιο διασκεδαστικό.

Έτσι μερικές απο αυτές τις προσθήκες και αλλαγές ειναι η χρήση παγκοσμίου διαδικτυακού πίνακα που κρατάει το σκορ για κάθε μουσικό κομμάτι, ώστε να δίνεται η επιλογή στον παίκτη να προσπαθήσει να νικήσει αυτό το σκορ. Μια επιπλέον αλλαγή, θα μπορούσε να ειναι των ιδιοτήτων των αντικειμένων ώστε να δημιουργούνται έχοντας κάποια κλίση καθώς και να ορίζεται μια απο τις τέσσερις πλευρές των κουτιών που θα πρεπει αποκλειστικά να χτυπήσει – αλληλεπιδράσει ο παίκτης. Μια ακόμη αλλαγή, θα μπορούσε να ειναι στην φόρτωση μουσικών κομματιών. Αντί να γίνεται φόρτωση τοπικά απο τον αποθηκευτικό χώρο του υπολογιστή, να γίνεται απευθείας stream απο το διαδίκτυο το μουσικό κομμάτι και να κατεβαίνουν εκείνη την στιγμή τα JSON αρχεία της πίστας. Δηλαδή οταν δημιουργείται μια πιστά στο επεξεργαστικό μέρος, απο τον παίκτη να επιλέγει για πιο

διαδικτυακό μουσικό κομμάτι θέλει να φτιάξει μια πίστα, και μόλις την δημιουργήσει αυτή την πίστα αυτή θα ανεβαίνει στο διαδίκτυο μαζι με άλλες πίστες άλλων παικτών. Στο ρυθμικό μέρος οταν ο παίκτης επιλέξει ένα διαδικτυακό μουσικό κομμάτι, να επιλέγει ποια πίστα, που δημιουργήθηκε απο αυτόν η άλλους παίκτες, θέλει να παίξει.

Αυτό το παιχνίδι μπορεί να επεκταθεί και να βελτιστοποιηθεί, με οποιαδήποτε τροπο, αρκετά ευκολά, καθώς μπορεί να υλοποιηθεί και να προσομοιωθεί σχεδόν οτιδήποτε στην Unity.

Βιβλιογραφία

DanniSchou. (2012). *Github*. Ανάκτηση από github.com: https://github.com/DanniSchou/MeshSplitting

Dealessandri, M. (2020, April 2). *GamesIndustry.* Ανάκτηση από https://www.gamesindustry.biz: https://www.gamesindustry.biz/articles/2020-04-01-the-best-practices-and-design-principles-of-vr-development

Eisenberg, A. (2020). *appreal-vr.* Ανάκτηση από https://appreal-vr.com/blog/: https://appreal-vr.com/blog/vr-software-and-the-art-of-integration/

Graces Games. (2018). *Assetstore Unity*. Ανάκτηση από assetstore.unity.com: https://assetstore.unity.com/packages/tools/input-management/simple-file-browser-98451

Graillot, Y. (2019). *Assetstore Unity*. Ανάκτηση από assetstore.unity.com: https://assetstore.unity.com/packages/tools/gui/gaze-ui-for-canvas-70881

LeapMotion. (2015, August 29). *medium.* Ανάκτηση από https://medium.com/: https://medium.com/@LeapMotion/vr-design-best-practices-bb889c2dc70

Li, Y. (2018). *Assetstore Unity*. Ανάκτηση από assetstore.unity.com: https://assetstore.unity.com/packages/tools/input-management/vr-keyboard-95936

Sysdia Solutions Ltd. (2019). *Assetstore Unity*. Ανάκτηση από assetstore.unity.com: https://assetstore.unity.com/packages/tools/integration/vrtk-virtual-reality-toolkit-vr-toolkit-64131

UnrealEngine. (2020). *unrealengine.* Ανάκτηση από https://docs.unrealengine.com: https://docs.unrealengine.com/en-US/Platforms/VR/DevelopVR/ContentSetup/index.html

Wikipedia. (2018, Οκτώβρης 19). *Γραφικό_περιβάλλον_χρήστη.* Ανάκτηση από el.wikipedia.org: https://el.wikipedia.org/wiki/Γραφικό_περιβάλλον_χρήστη

Wikipedia. (2020, 4 18). *Εικονική Πραγματικότητα.* Ανάκτηση από el.wikipedia.org: https://el.wikipedia.org/wiki/Εικονική_πραγματικότητα

Μουστάκας Κ., Π. Ι. (2015). *Γραφικά και Εικονική Πραγματικότητα.* Ανάκτηση από repfiles kallipos: http://repfiles.kallipos.gr/html_books/50/Chapter_9/index.html