Μεταπτυχιακό Πρόγραμμα 2014-2015, Προηγμένα Συστήματα Πληροφορικής

Μεταπτυχιακή Διατριβή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

Θέμα:

Υλοποίηση Rhythm Shooter Game με Ανίχνευση Ρυθμού Μουσικής σε Πραγματικό Χρόνο στην Πλατφόρμα Unity 3D

(Rhythm Shooter Game Development with Real Time Beat Detection in Unity 3D)

Επιβλέπων Καθηγητής: κος Τσιχριντζής Γεώργιος

Εκπονητής Εργασίας: Σιώτα Νικολέττα – Ελευθερία, ΜΠΣΠ 14080

ΠΕΡΙΛΗΨΗ

Η παρούσα μεταπτυχιακή διατριβή έχει ως σκοπό την παρουσίαση των βημάτων υλοποίησης ενός Rhythm Shooter Video Game με Real Time Beat Detection και VR Compatibility για Oculus Rift Headset, σε γλώσσα προγραμματισμού C#, στην πλατφόρμα Unity 3D, με χρήση αυστηρά μη μισθωμένων μοντέλων και με επίκεντρο την διαχείριση μέσω script και όχι animations ή prefabs. Σκοπός μας είναι να εντρυφήσουμε σταδιακά στις δυνατότητες της πλατφόρμας προκειμένου να δημιουργήσουμε ένα τελικό προϊόν που να συνδυάζει υψηλό performance χωρίς lag, σταθερά υψηλά fps, μινιμαλιστική αισθητική κατάλληλη για VR και φυσικά αδιάκοπη λειτουργικότητα με seamless αποτέλεσμα που να δοκιμάζει τις δυνατότητες του παίχτη. Με την ολοκλήρωση των βημάτων της παρούσας εργασίας, θα έχουμε επιτύχει ένα καινοτόμο αποτέλεσμα που είναι εύκολα προσαρμόσιμο και εξελίξιμο για εμπορική διάθεση καθώς και θα έχουμε εξοικειωθεί σε μεγάλο βαθμό με τις μεθόδους ανάπτυξης παιχνιδιών στην πλατφόρμα Unity.

Λέξεις κλειδιά: Rhythm Game, Beat Detection, C#, Unity 3D, Oculus Rift, VR

~ ΠΕΡΙΕΧΟΜΕΝΑ ΕΡΓΑΣΙΑΣ ~

Δήλωση Πνευματικής Ιδιοκτησίας

Η παρούσα εργασία αποτελεί προϊόν αποκλειστικά δικής μου προσπάθειας. Όλες οι πηγές που χρησιμοποιήθηκαν περιλαμβάνονται στη βιβλιογραφία και γίνεται ρητή αναφορά σε αυτές μέσα στο κείμενο όπου έχουν χρησιμοποιηθεί.

Σιώτα Νικολέττα – Ελευθερία

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους γονείς για την αμέριστη στήριξη και υπομονή τους σε όλη τη διάρκεια της προσπάθειάς μου αυτής, και στα καλά αλλά προπάντων στα δύσκολα. Θα ήθελα επίσης να ευχαριστήσω τον επιβλέπων καθηγητή μου κο Τσιχριντζή Γεώργιο για την εμπιστοσύνη που μου έδειξε αναλαμβάνοντας με σε αυτή την εργασία καθώς και για την στήριξη του όταν το είχα πραγματικά ανάγκη. Θα ήθελα επίσης να ευχαριστήσω τους συναδέλφους μου που δέχτηκαν να γίνουν δοκιμαστές αυτής η εφαρμογή από την αρχή μέχρι το τέλος της. Τέλος, ένα μεγάλο ευχαριστήσω στον κο Καρακίτσιο Ξενοφών για την ασταμάτητη στήριξη του και την “γκρίνια” του για να πιέσω τον εαυτό μου να δοκιμάσει και να δοκιμαστεί μέσα από αυτό το πόνημα.

ΜΕΡΟΣ Ι: Εισαγωγή

1.1 Rhythm Games – Τα Underdog της Gaming Βιομηχανίας που Κατέκτησαν τον Κόσμο

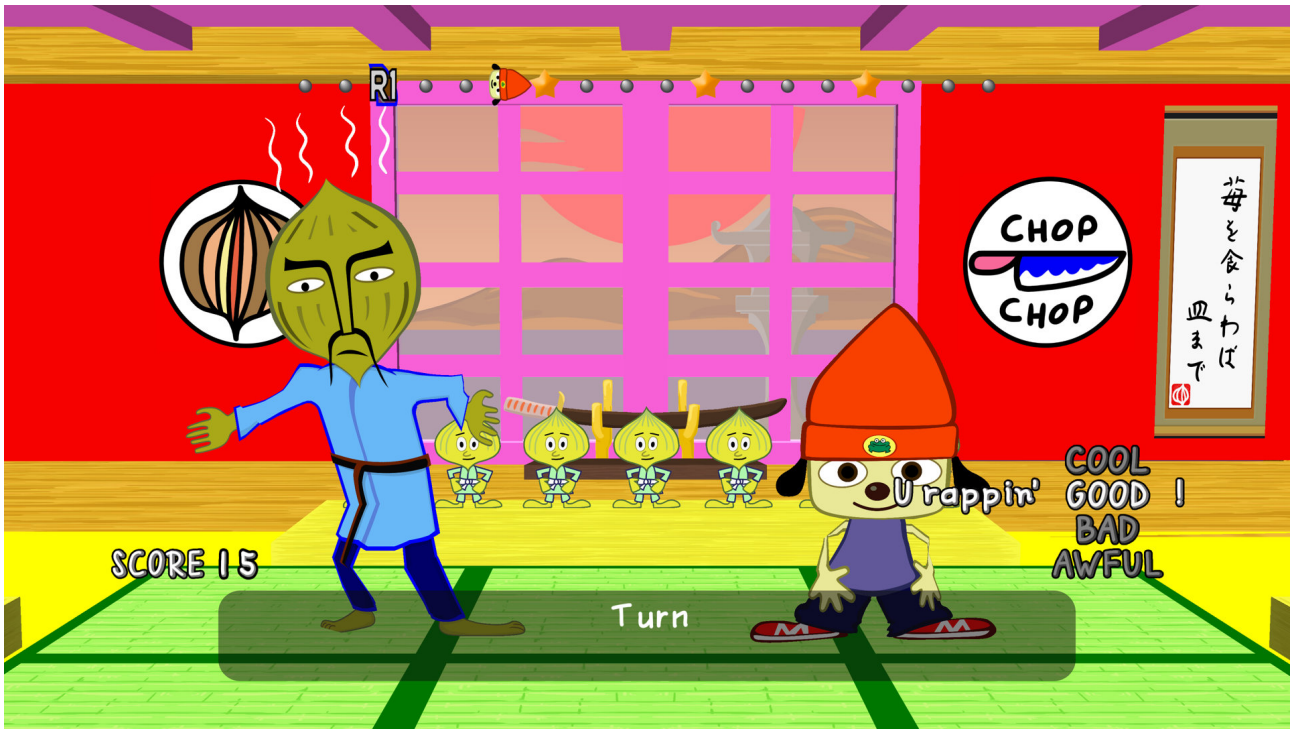
Rhythm games (παιχνίδια ρυθμού) είναι μια υποκατηγορία βιντεοπαιχνιδιών δράσης τα οποία χρησιμοποιούν είτε φυσική κίνηση είτε ακολουθίες πλήκτρων (αυξανόμενης δυσκολίας) για να δοκιμάσουν τις ικανότητες του παίχτη στο εκάστοτε τραγούδι.
Ο παίχτης πρέπει να εστιάσει στον ρυθμό της μουσικής και να δίνει το αντίστοιχο input που απαιτεί το παιχνίδι σε κάθε περίπτωση. Οι σωστές και ακριβείς ακολουθίες ενεργειών ανταμείβονται με πόντους ενώ αστοχίες μειώνουν το σκορ του παίχτη ή σταματούν τυχόν ενεργά bonus. Συνήθως, λόγω της ακρίβειας που απαιτείται για να ολοκληρωθούν συγκεκριμένες πίστες, οι παίχτες αναγκάζονται να τις επαναλάβουν πολλές φορές μέχρι να μπορούν να ακολουθήσουν απόλυτα τον ρυθμό.
Για να αντισταθμιστεί το επίπεδο δυσκολίας, τα rhythm games συνήθως είναι μικρής διάρκειας και χρησιμοποιούν προκαθορισμένες ακολουθίες με διάρκεια εισαγωγής του κάθε input ανάλογη της μουσικής.

Εικόνα 1.1.i: Στιγμιότυπο από το Gameplay του παγκοσμίως γνωστού παιχνιδιού της Sega Hatsune Miku: Project DIVA Future Tone με τους χαρακτήρες του Voice Synthesising Software Vocaloid.

Αντίθετα με τα συμβατικά παιχνίδια, το input των rhythm games περιορίζεται μόνο στις επιθυμίες του δημιουργού με επιλογές από απλά χειριστήρια, πληκτρολόγιο ή ένα ποντίκι σε ρέπλικες μουσικών οργάνων, control pads και arcade dance stages.
Άλλη μια εξίσου βασική ιδιομορφία των rhythm games είναι η μη απαραίτητη ύπαρξη από Avatar ή Player Controller χαρακτήρα. Τα avatar που μας τοποθετούν στην θέση του παίχτη στις πίστες των παιχνιδιών πλέον εδώ συμβάλουν μόνο στο να ακολουθούν τις κινήσεις που πραγματοποιεί ο παίχτης ή για την θέαση από μη συμμετέχοντες.
Αντιπροσωπευτικά δείγματα αυτής της συναρπαστικής κατηγορίας παιχνιδιών σε διαφορετικές πλατφόρμες λειτουργίας, από arcade μέχρι κονσόλες και smartphones, είναι τα Beatmania, Dance Dance Revolution, Guitar Hero, Rock Band, Just Dance, Singstar, Patapon, The Idolmaster, Hatsune Miku: Project Diva, Love Live! School Idol Festival και φυσικά το κλασσικό PaRappa the Rapper.

1.2 Η Σύντομη Ιστορία των Rhythm Games – Παίζοντας στον Ρυθμό της Μουσικής από το 1969

Το πρώτο παιχνίδι που βασίστηκε στον ρυθμό της μουσικής δημιουργήθηκε για Ιαπωνικά Arcade Stores στις αρχές του 1970 από τους Kenzou Furukawa και Kenji Nagata. Η ιδέα προήλθε από μια διαφήμιση της εποχής που ενέπνευσε τον Furukawa να δημιουργήσει ένα βιντεοπαιχνίδι που να συγκαταλέγει κίνηση/χορό με μουσική. Στην πραγματικότητα όμως, οι πρώιμες μορφές των rhythm games μέχρι την δεκαετία του 90 δεν έμοιαζαν με την εικόνα που έχουμε για το είδος σήμερα αλλά πιο πολύ με παιχνίδια μίμησης ή παραγωγής ήχων συμβατών με τη μουσική του παιχνιδιού.
Το παιχνίδι όμως που αποτέλεσε την βάση για τα μουσικά βιντεοπαιχνίδια και το ουσιαστικά πρώτο rhythm game με mechanics που χρησιμοποιούνται με επιτυχία ακόμα και σε φετινές κυκλοφορίες, είναι το PaRappa the Rapper που υλοποιήθηκε από το studio NanaOn-Sha και κυκλοφόρησε στην Ιαπωνία στις 6 Δεκεμβίου του 1996 από την Sony Computer Entertainment για το Playstation.



Εικόνα 1.2.i: Στιγμιότυπο από το remastered PaRappa The Rapper που διατέθηκε για το Playstation 4 στην εικοστή επέτειο κυκλοφορίας του τίτλου.

Το PaRappa the Rapper απέκτησε μεγάλο κοινό οπαδών καθώς ο χειρισμός του βασίστηκε για πρώτη φορά στην ικανότητα του παίχτη να πατάει τα εικονιζόμενα πλήκτρα και να μένει στον ρυθμό του κομματιού χωρίς να πέφτει η απόδοση του. Όταν ο παίχτης διατηρούσε το ρυθμό με επιτυχία, ο δείκτης “U rappin’” αυξάνονταν σταδιακά με μέγιστη τιμή το Cool, ενώ η αποτυχία οδηγούσε σε πτώση του δείκτη. Τα μοναδικά καρτουνίστικα γραφικά που αντικατόπτριζαν την αισθητική της δεκαετίας σε συνδυασμό με το καινοτόμο gameplay, έφεραν μια νέα εποχή για τα παιχνίδια μουσικής.
Η Konami ήταν επίσης ένας καταλυτικός παράγοντας για την εξέλιξη των rhythm games από το 1997 έως το 1999. Παρόλο που μας έφεραν τα καινοτόμα Beatmania, Dance Dance Revolution, GuitarFreaks και DrumMania, τα παιχνίδια αυτά περιορίστηκαν δυστυχώς για ακόμα μια φορά στα Arcade Stores. Τεχνολογίες που χρησιμοποιούνταν στο GuitarFreaks όπως το χειριστήριο κιθάρα καθώς και χειριστήριο drums για το DrumMania, εντάχθηκαν σε τεράστιες δυτικές εμπορικές επιτυχίες όπως το Guitar Hero και το Rock Band πολύ αργότερα, το 2005 και 2007 αντίστοιχα.
Στον δυτικό κόσμο, ο κόσμος των παιχνιδιών μουσικής αποθεώθηκε μέσω παιχνιδιών όπως το Guitar Hero, το DJ Hero, το Rock Band, το Singstar και το Just Dance. Ταυτόχρονα όμως, η Ιαπωνία άλλαζε για ακόμα μια φορά πλεύση στα στερεότυπα των rhythm games.

Εικόνα 1.2.ii: Στιγμιότυπο από το δημοφιλές παιχνίδι Guitar Hero το οποίο λειτουργεί με χειριστήριο σε σχήμα κιθάρας για να προσομοιώνει το παίξιμο έγχορδων σε γνωστά ροκ κομμάτια.

Το 2004, η Ιαπωνική βιομηχανία των rhythm game στράφηκε προς το παιδικό/γυναικείο κοινό. Οι νέοι τίτλοι επικεντρώθηκαν σε kawaii χαρακτήρες (εξαιρετικά χαριτωμένοι) και σε συλλογή καρτών/τρόπαια με πόζες των χαρακτήρων σε κάθε επιτυχία του παίχτη. Από εταιρίες όπως Sega, Taito, Bandai, Tomy παρήχθησαν χαρακτηριστικά παιχνίδια όπως το PriPara και το Pretty Cure. Η απήχηση αυτών των τίτλων όμως δεν μπορούσε να συγκριθεί με την παγκόσμια επιτυχία που ήρθε το 2008/2009 με την εισαγωγή των Virtual Idol στα rhythm games από τα παιχνίδια IdolMaster και Hatsune Miku:Project Diva. Ειδικά στην περίπτωση του Project Diva, τα videoclip που χρησιμοποιήθηκαν ήταν δημοφιλή τραγούδια που παρήχθησαν από ανεξάρτητους δημιουργούς μέσω του λογισμικού Voice Synthesising Vocaloid και προβάλλονταν στην Ιαπωνική Online Πλατφόρμα Video Nico Nico Douga. Ήταν μια καινοτόμα ιδέα που συνδύαζε την τεχνογνωσία της Sega στα rhythm games και την αδιαμφισβήτητη δημοφιλία του χαρακτήρα Hatsune Miku που από απλή μασκότ της φωνής του λογισμικού της Crypton Future Media έγινε παγκόσμιο virtual μουσικό είδωλο με συναυλίες που είχαν μουσικούς επί σκηνής και Realtime Hologram του χαρακτήρα.
Σήμερα, τα μουσικά παιχνίδια έχουν βρίσκουν νέους τρόπους να ξαφνιάζουν ευχάριστα το κοινό τους μέσω εφαρμογών για Android και iOS. Μικρές indie εταιρίες παράγουν συνεχώς νέο υλικό που διατίθεται μέσω του Steam, PSNetwork, Xbox Game Store, Play Store κλπ. Επίσης, το 2017 είναι η χρονιά που η διάθεση ολοκληρωμένων rhythm game τα οποία είναι είτε VR Exclusive είτε VR Compatible ξεκινά δυναμικά κυρίως για Oculus Rift και PlayStation VR.

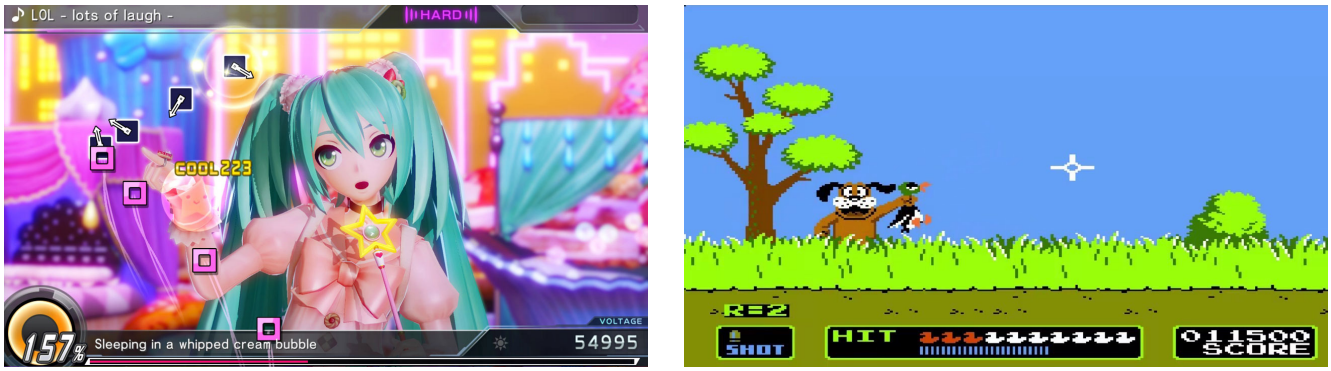
1.3 Shooter Games: Ένα Genre, Χιλιάδες Τίτλοι, Εκατομμύρια Θαυμαστές Παγκοσμίως

Shooter games είναι επίσης μια υποκατηγορία βιντεοπαιχνιδιών δράσης που δημιουργήθηκε για να δοκιμάζει τα αντανακλαστικά και τις γρήγορες αντιδράσεις των παιχτών. Είναι παιχνίδια που χρησιμοποιούν ως βασικά συστατικά χαρακτήρες πρώτου ή τρίτου προσώπου και ένα όπλο, για melee ή long range μάχες. Στόχος σε αυτά τα παιχνίδια είναι η μέγιστη εξουδετέρωση αντιπάλων καθώς και η επιβίωση του παίχτη μέσα στα διάφορα stages.
Χαρακτηριστικά δείγματα των πολλών shooter games που έχουν ενθουσιάσει γενιές παιχτών είναι τα Space Invaders, Duck Hunt, Doom, Call of Duty, Overwatch κλπ.

ΜΕΡΟΣ ΙΙ: Υλοποιώντας το Παιχνίδι

2.1 Rhythm Shooter Game – Κατανοώντας την Λογική του Project

Έχοντας μιλήσει στην εισαγωγή για το τι είναι ένα Rhythm Game καθώς και για την ιστορία τους, ας μιλήσουμε για το πως μπορεί αυτό το μοναδικό είδος να συνδυαστεί με την λογική των Shooter Games. Και τα δύο είδη ανήκουν στα παιχνίδια δράσης, δοκιμάζοντας συνεχώς την ικανότητα των παιχτών να πραγματοποιούν σύνθετες ενέργειες μέσα σε λίγα δευτερόλεπτα. Τι γίνεται όμως όταν παράγουμε ταχύτατα στόχους, σε ρυθμό χορευτικής μουσικής, και αναγκάζουμε τον παίχτη να δοκιμάσει όχι μόνο τα αντανακλαστικά του αλλά και την ικανότητα του να μένει στον ρυθμό για να κερδίσει; Αυτό είναι που θα υλοποιήσουμε και σε αυτό το project.
Πηγή έμπνευσης για την δομή του παιχνιδιού ήταν δύο πολύ αγαπημένα παιχνίδια, το κλασσικό αλλά πρωτοποριακό Shooter Game Duck Hunt [που κυκλοφόρησε τον Απρίλιο του 1984 για το NES (Nintendo Entertainment System) από την Nintendo] και το δημοφιλέστατο Rhythm Game Project Diva: Fututre Tone [το οποίο κυκλοφόρησε πρώτη φορά τον Ιανουάριο του 2010 και διατέθηκε δωρεάν στο PSN (PlayStation Network) τον Ιανουάριο του 2017 από την Sega]. Το Project Diva απαιτεί εκτέλεση περίπλοκων ακολουθιών πλήκτρων ακριβώς στο ρυθμό της μουσικής ενώ το Duck Hunt απαιτούσε να χτυπάς συγκεκριμένο αριθμό στόχων σε κάθε επίπεδο για να κερδίζεις. Για να συνδυάσουμε τα δύο διαφορετικά concept σε ένα παιχνίδι σκεφτήκαμε την παραγωγή ενός τυχαίου αριθμού στόχων σε κάθε χτύπημα του ρυθμού και να ζητάμε από τον παίχτη να προλαβαίνει να χτυπάει αρκετούς έτσι ώστε να καλύπτει ένα ελάχιστο όριο για κάθε τραγούδι.

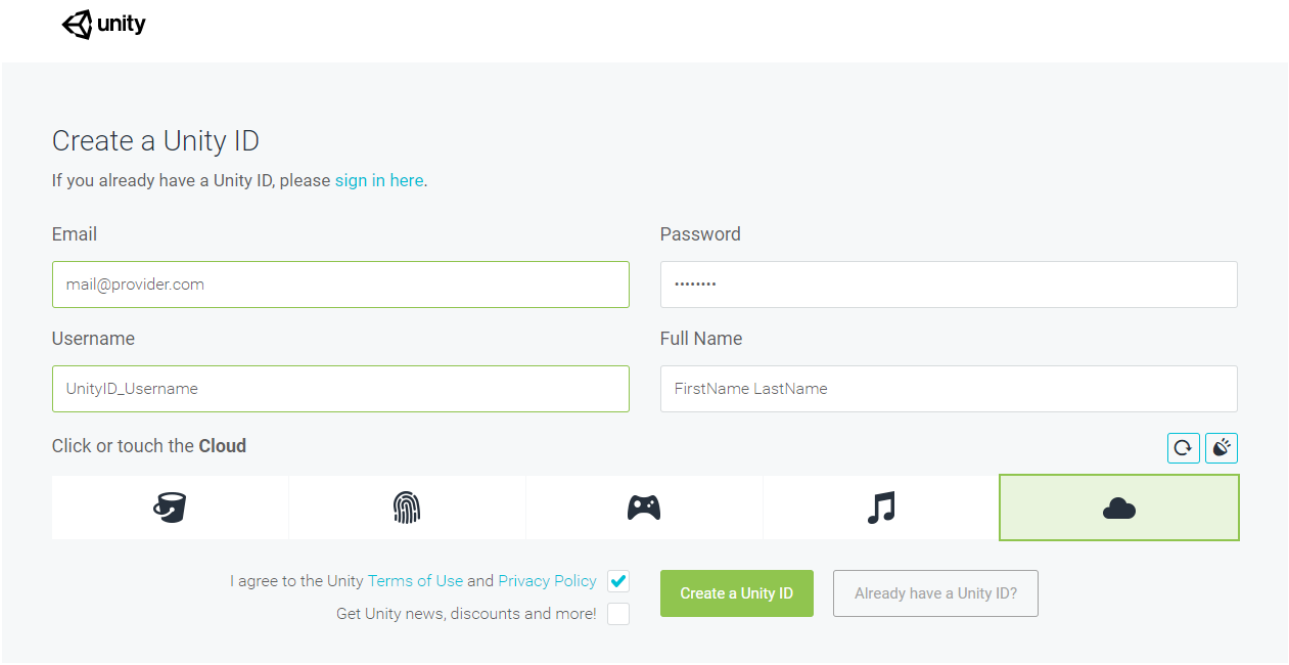


Εικόνα 2.1.i: Project Diva και Duck Hunt, δύο διαφορετικά παιχνίδια, ένα ξεχωριστό αποτέλεσμα

Όλα τα Rhythm Games μέχρι σήμερα όμως, χρησιμοποιούσαν προκαθορισμένα spawn, χρονισμένα ακριβώς σε κάθε ρυθμικό χτύπημα χωρίς κανένα δυναμικό στοιχείο. Τα αργά κομμάτια αποτελούσαν τα εύκολα επίπεδα ενώ τα γρήγορα και ρυθμικά τα δύσκολα. Η ιδέα που προέκυψε για να αλλάξει αυτή τη νόρμα και να δώσει μια νέα πνοή στο είδος ήταν η χρήση δυναμικής ανίχνευσης του ρυθμού της μουσικής με sampling, την ώρα που παίζει το τραγούδι στο επίπεδο, με χρήση Fourier. Δυστυχώς η Unity δεν διαθέτει έτοιμες βιβλιοθήκες που επιτρέπουν την χρήση τέτοιων μεθόδων εύκολα (π.χ. επεξεργασία σήματος σε Matlab) οπότε θα πρέπει να αναπτύξουμε τον κώδικα εξ’ ολοκλήρου μόνοι μας και βάση τις ιδιομορφίες της πλατφόρμας.
Τέλος, είναι ενδιαφέρον ότι τα πρώτα Rhythm Games που κυκλοφόρησαν για μάσκες εικονικής πραγματικότητας ήταν ελάχιστα και μόλις φέτος, το 2017. Υπάρχει ένα πολύ μεγάλο εμπορικό κενό στις κυκλοφορίες τίτλων συμβατών για VR Headsets γενικότερα, με τα Horror Games να κυριαρχούν στην αγορά. Αυτός ήταν ένας από τους βασικούς λόγους που επιθυμούμε αυτό το παιχνίδι να είναι συμβατό με μάσκες VR προκειμένου να δοκιμάσουμε τις δυνατότητες ενός τόσο τελευταίας τεχνολογίας εξοπλισμού με ένα τόσο ιδιόμορφο είδος βιντεοπαιχνιδιού.
Ελπίζουμε ότι στο τέλος αυτής της εργασίας όχι μόνο να επιτύχουμε τα παραπάνω αλλά να συνδυάσουμε υψηλό performance, υψηλά fps, seamless αποτέλεσμα και καθόλου lag.

2.2 Δημιουργία Λογαριασμού κ Οδηγίες Εγκατάστασης της Πλατφόρμας Unity

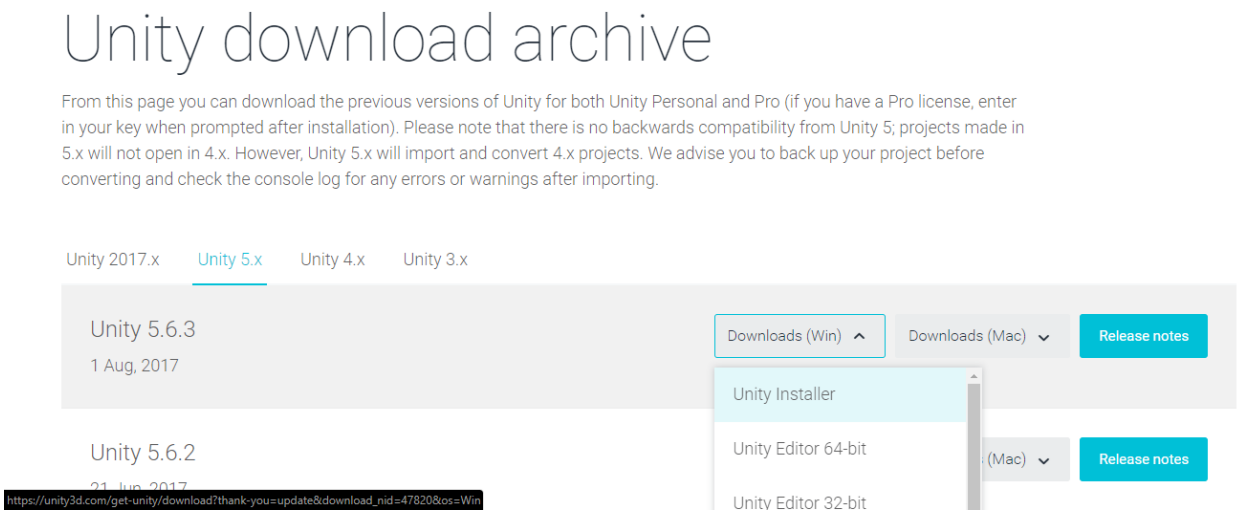
Πριν ξεκινήσουμε την υλοποίηση του παιχνιδιού μας, βασικό είναι να προχωρήσουμε στην εγκατάσταση της πλατφόρμας της επιλογής μας. Για την συγκεκριμένη εργασία επιλέξαμε την πλατφόρμα δωρεάν διάθεσης Unity 3D, Version 5.6.3f1. Υπήρχαν και άλλες διαθέσιμες επιλογές όπως η Unreal Engine 4, η Godot Engine, η Construct 2 (απευθύνεται σε άτομα με περιορισμένες γνώσεις προγραμματισμού) αλλά η Unity επιλέχθηκε λόγω προηγούμενης ατομικής εμπειρίας με το περιβάλλον καθώς και από επιθυμία να κατανοήσω καλύτερα την δυνατότητα προσαρμογής της στην εποχή του VR. Το Unity διατίθεται για Windows και MacOS.
Το πρώτο βήμα στα στάδια εγκατάστασης της εφαρμογής είναι και το βασικότερο. Πριν επιλέξουμε την έκδοση που θα έχουμε στον υπολογιστή μας, πρέπει να εγγραφούμε ως μέλος της Unity για να μπορούμε να χρησιμοποιούμε όλες τις παροχές της όπως το Asset Store. Στον ιστότοπο unity3d.com πηγαίνουμε στο εικονίδιο του χρήστη και επιλέγουμε Create Account όπου και εισάγουμε τα στοιχεία μας καθώς και δηλώνουμε ότι το προϊόν θα χρησιμοποιηθεί καθαρά για ακαδημαϊκή χρήση.



Εικόνα 2.2.i: Δημιουργώντας το προφίλ χρήστη μας για την πλατφόρμα Unity 3D.

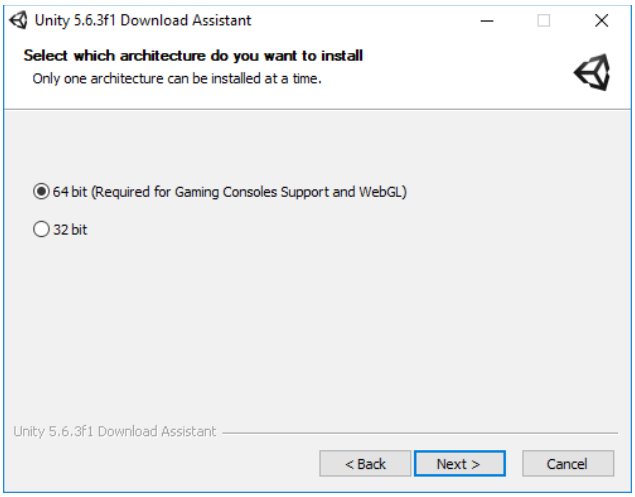
Έχοντας ολοκληρώσει την εγγραφή μας ως χρήστης, θα προχωρήσουμε στην εγκατάσταση της έκδοσης της επιλογής μας. Το project που υλοποιείται σε αυτή την εργασία, όπως αναφέραμε και στην αρχή του κεφαλαίου, είναι στην έκδοση 5.6.3. Φυσικά και είναι απόλυτα θεμιτό να χρησιμοποιηθεί και κάποια νεότερη έκδοση όπως το Unity 2017.1 αλλά θα πρέπει να τονιστεί ότι κάποια πεδία ίσως να μην βρίσκονται στα ίδια path και να απαιτείται έλεγχος του κώδικα μέσω του documentation της πλατφόρμας για Obsolete εκφράσεις. Αντενδείκνυται η χρήση πολύ παλαιότερων εκδόσεων της πλατφόρμας καθώς οι τεχνοτροπίες που παρουσιάζονται στις διάφορες μεθόδους υλοποίησης του παιχνιδιού δεν υποστηρίζονταν πριν την έκδοση 5.
Σε περίπτωση που δεν επιθυμούμε να κατεβάσουμε την πιο πρόσφατη έκδοση που διατίθεται, στην σελίδα store.unity.com επιλέγουμε την κατηγορία Personal (Free) και στην συνέχεια επιλέγουμε τον υπερσύνδεσμο Older Versions of Unity. Ο σύνδεσμος αυτός μας κατευθύνει στο Unity Download Archive όπου έχουμε πρόσβαση στα download links για όλες τις εκδόσεις τις πλατφόρμας.

Σημείωση: εάν επιθυμείτε να χρησιμοποιήσετε τα ίδια γραφικά στοιχεία με αυτό το project, προτείνουμε να μην επιλέξετε κάτι πριν από την έκδοση 5.3.6.

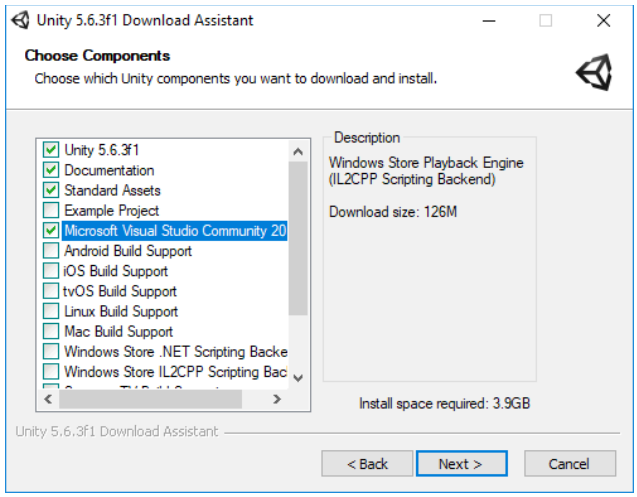


Εικόνα 2.2.ii: Το αρχείο όλων των εκδόσεων της πλατφόρμας.

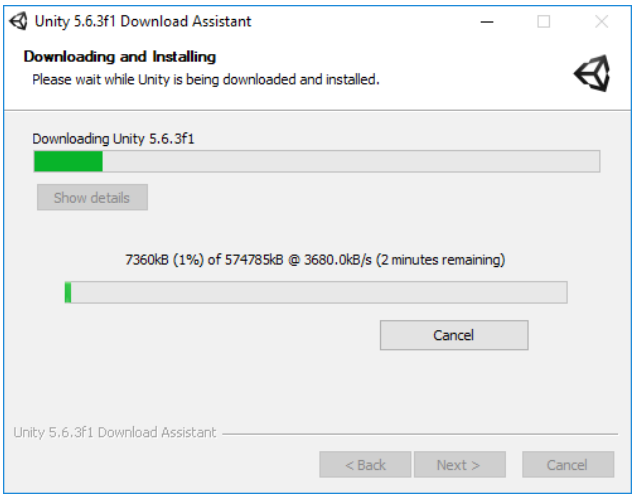
Το βέλτιστο είναι να επιλέξουμε σε κάθε περίπτωση τον Unity Installer προκειμένου να διευκολύνουμε την διαδικασία και να ελαχιστοποιήσουμε τα βήματα εγκατάστασης χωρίς να ανησυχούμε για λάθη. Επίσης, προτείνεται η ένταξη και του Microsoft Visual Studio στην διαδικασία εγκατάστασης χωρίς όμως να είναι δεσμευτική. Παραθέτονται στην συνέχεια τα βασικά στάδια εγκατάστασης σε μορφή εικόνων.



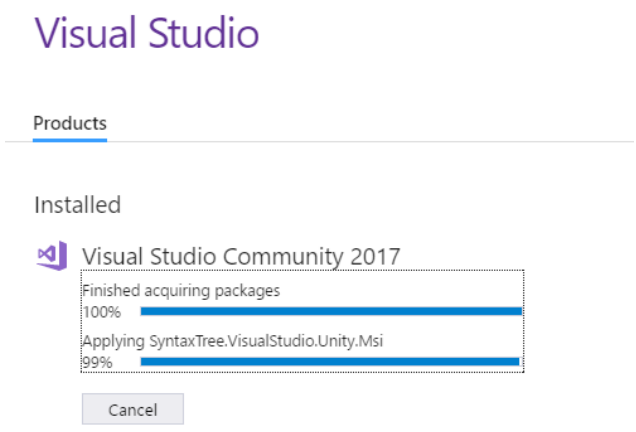
1) Επιλογή Αρχιτεκτονικής Η/Υ



2) Επιλογή Component & Visual Studio

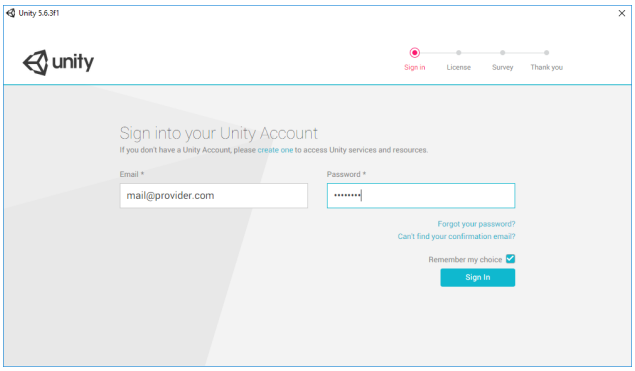


3) Λήψη & Εγκατάσταση Unity

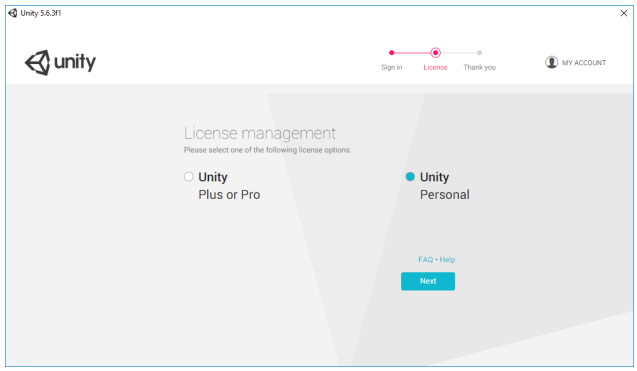


4) Εγκατάσταση Visual Studio

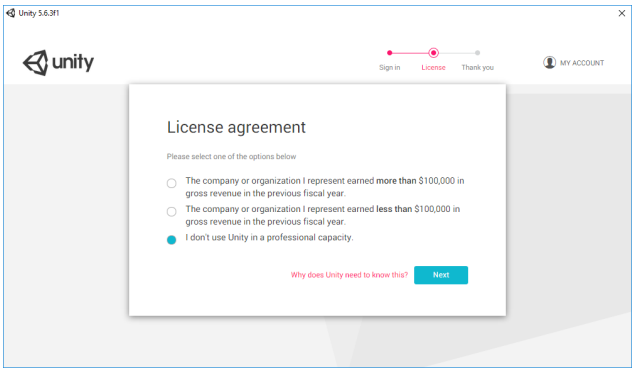
Έχοντας ολοκληρώσει την εγκατάσταση της πλατφόρμας, πρέπει κατά την πρώτη εκκίνηση της να κάνουμε τοπική σύνδεση του λογαριασμού Unity που δημιουργήσαμε στην αρχή του κεφαλαίου και να δηλώσουμε ότι πρόκειται να γίνει χρήση των παροχών της αποκλειστικά για μη επαγγελματική χρήση. Είναι σημαντικό να γίνει δήλωση του ανωτέρω για να μην υπάρχει conflict μεταξύ του ακαδημαϊκού licence που έχουμε στο account μας καθώς η παραγωγή εμπορικών εφαρμογών δεν είναι στις δωρεάν παροχές της πλατφόρμας. Για διευκόλυνση κατανόησης των βημάτων και αποφυγή λαθών, θα χρησιμοποιήσουμε για ακόμα μια φορά τον σύντομο οδηγό μέσω εικόνων για την παρουσίαση της διαδικασίας.



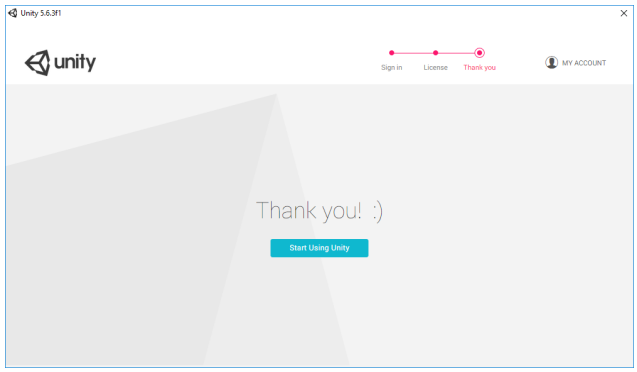
1) Log In με το Unity Account που έχουμε



2) Χρήση Licence για Προσωπική Χρήση



3) Δήλωση μη επαγγελματικής χρήσης (κέρδος).



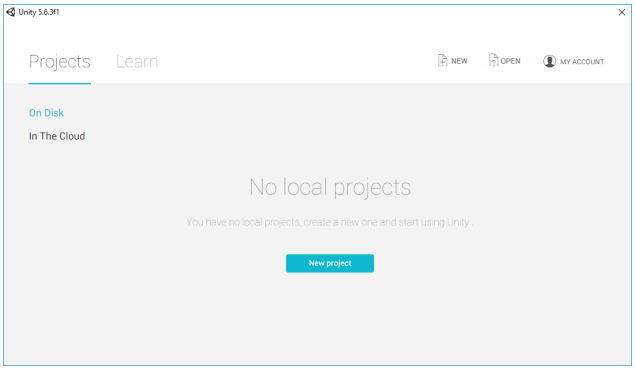
4) Ολοκλήρωση Διαδικασίας.

Έχοντας ολοκληρώσει και αυτό το βήμα, έχουμε πλέον ολοκληρώσει τα στάδια εγκατάστασης της πλατφόρμας και είμαστε έτοιμοι να προχωρήσουμε στην υλοποίηση του παιχνιδιού μας.

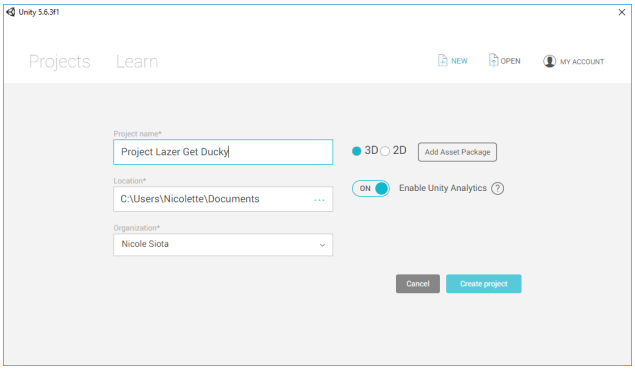
Σημείωση: εάν σας εξυπηρετεί, μπορείτε φυσικά να κατεβάσετε και κάποιον Editor για να τροποποιείτε εκτενέστερα τα Script μας. Σε περίπτωση που μια τέτοια λύση σας φαίνεται ενδιαφέρουσα, προτείνουμε το Notepad ++ το οποίο υπερκαλύπτει όλες τις ανάγκες μας.

2.3 Δημιουργία του Project και Γνωριμία με το Unity Asset Store

Έχοντας πλέον έτοιμη την πλατφόρμα μας, βλέπουμε για πρώτη φορά το μενού υποδοχής και θα προχωρήσουμε στην δημιουργία ενός νέου Project. Για χάριν ευκολίας και επειδή θα κάνουμε αυτό το Project ατομικά, επιλέγουμε να δουλέψουμε αποθηκεύοντας τα αρχεία τοπικά στο δίσκο και όχι στο cloud. Επιλέγουμε New Project και στην φόρμα που εμφανίζεται εισάγουμε το όνομα του Project, το path του τοπικά, τον δημιουργό, το format του παιχνιδιού (επιλέγουμε 3D διότι έχουμε τρισδιάστατο παιχνίδι με χωρικούς άξονες x, y, z). Εφόσον είναι η πρώτη φορά που χρησιμοποιούμε την πλατφόρμα δεν έχουμε ήδη υπάρχοντα Asset (δηλαδή πακέτα με μοντέλα, κώδικα, υφές κλπ) οπότε δεν θα προχωρήσουμε και σε εισαγωγή κάποιου. Όταν είμαστε έτοιμοι, πατάμε Create Project για να μπορέσουμε πλέον να εργαστούμε εντός της πλατφόρμας.

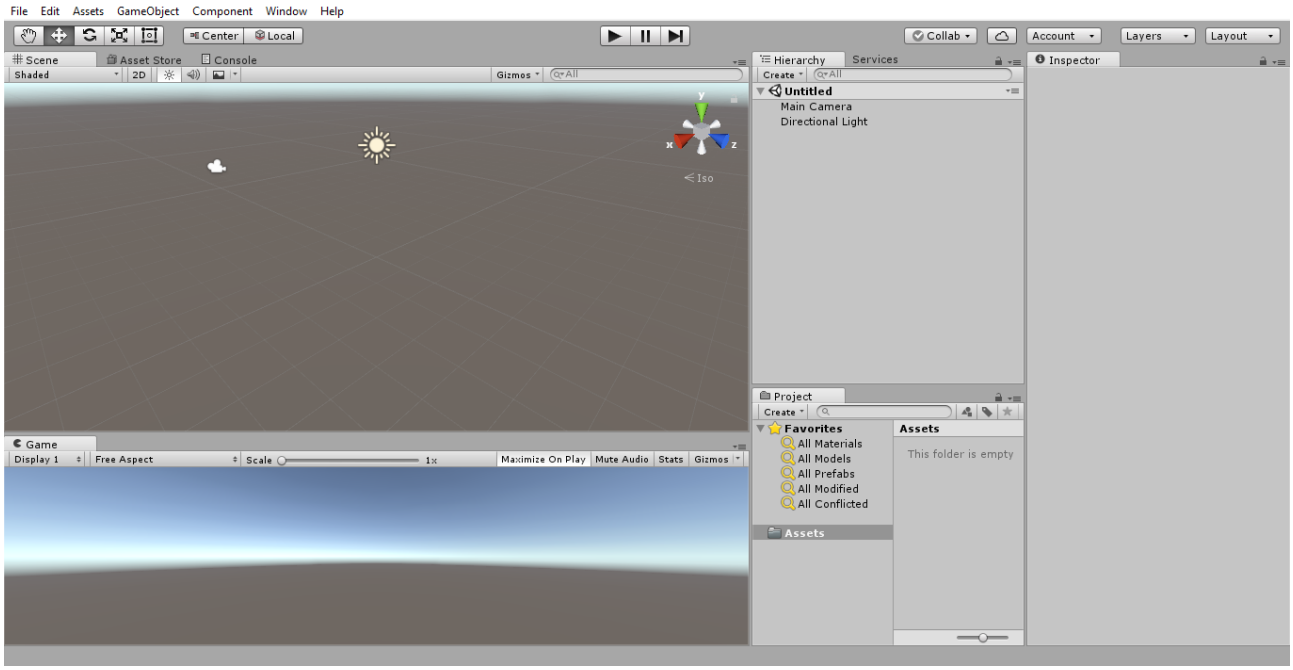


1) Κεντρικό Μενού Unity



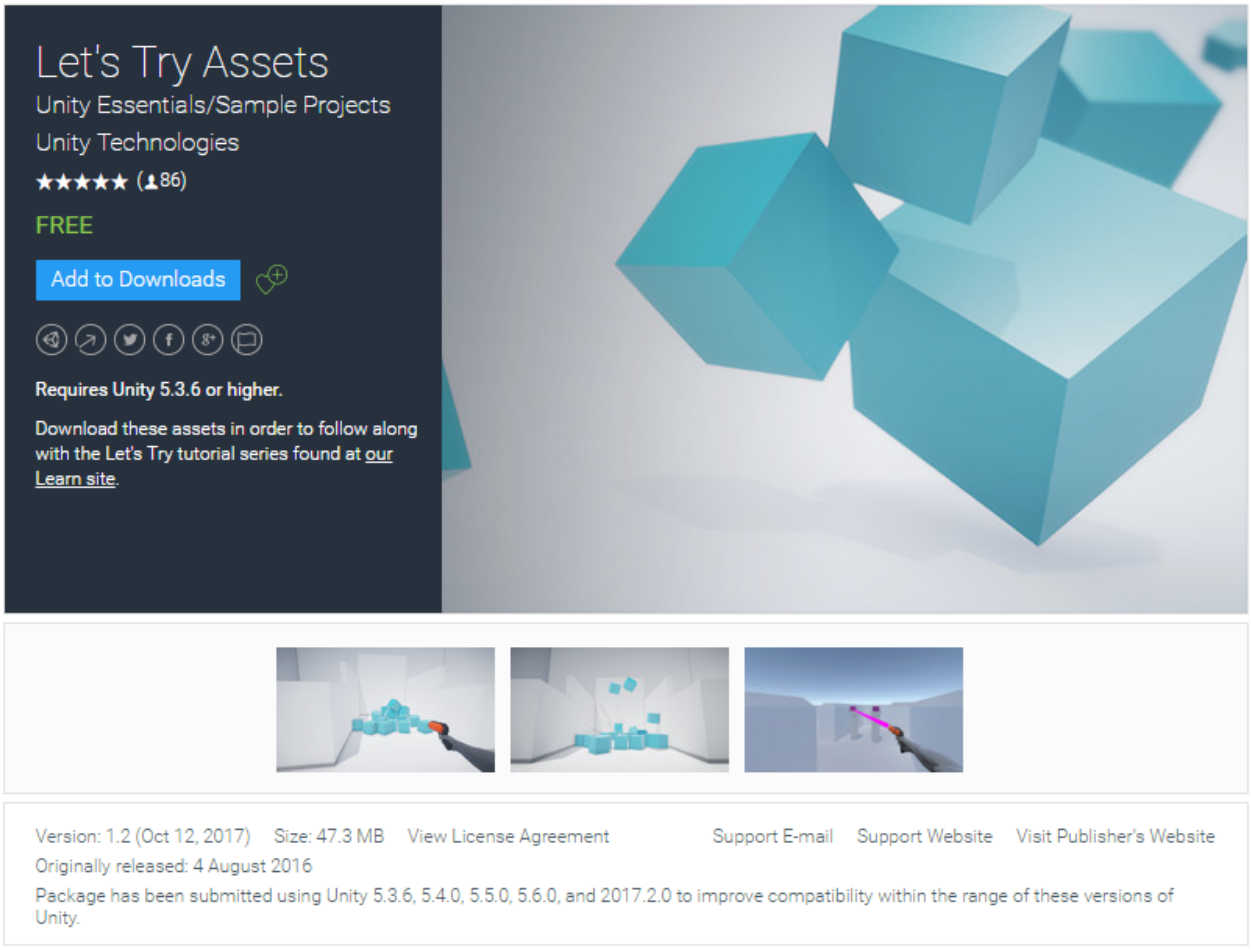
2) Δημιουργία Project

Μπαίνοντας για πρώτη φορά στην πλατφόρμα, είναι καίριο να κάνουμε μια ορθή οργάνωση του περιβάλλοντος προκειμένου να έχουμε μια δομημένη οπτική επί της ιεραρχίας των αντικειμένων του project, των φακέλων με τα Asset μας καθώς και να έχουμε ταυτόχρονη εικόνα της πίστας από την οπτική του δημιουργού αλλά και του παίκτη συγχρόνως. Οπότε, πριν προβούμε σε οποιαδήποτε άλλη ενέργεια, ας οργανώσουμε την πλατφόρμα όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 2.3.i: Ορθή δομή του περιβάλλοντος της πλατφόρμας.

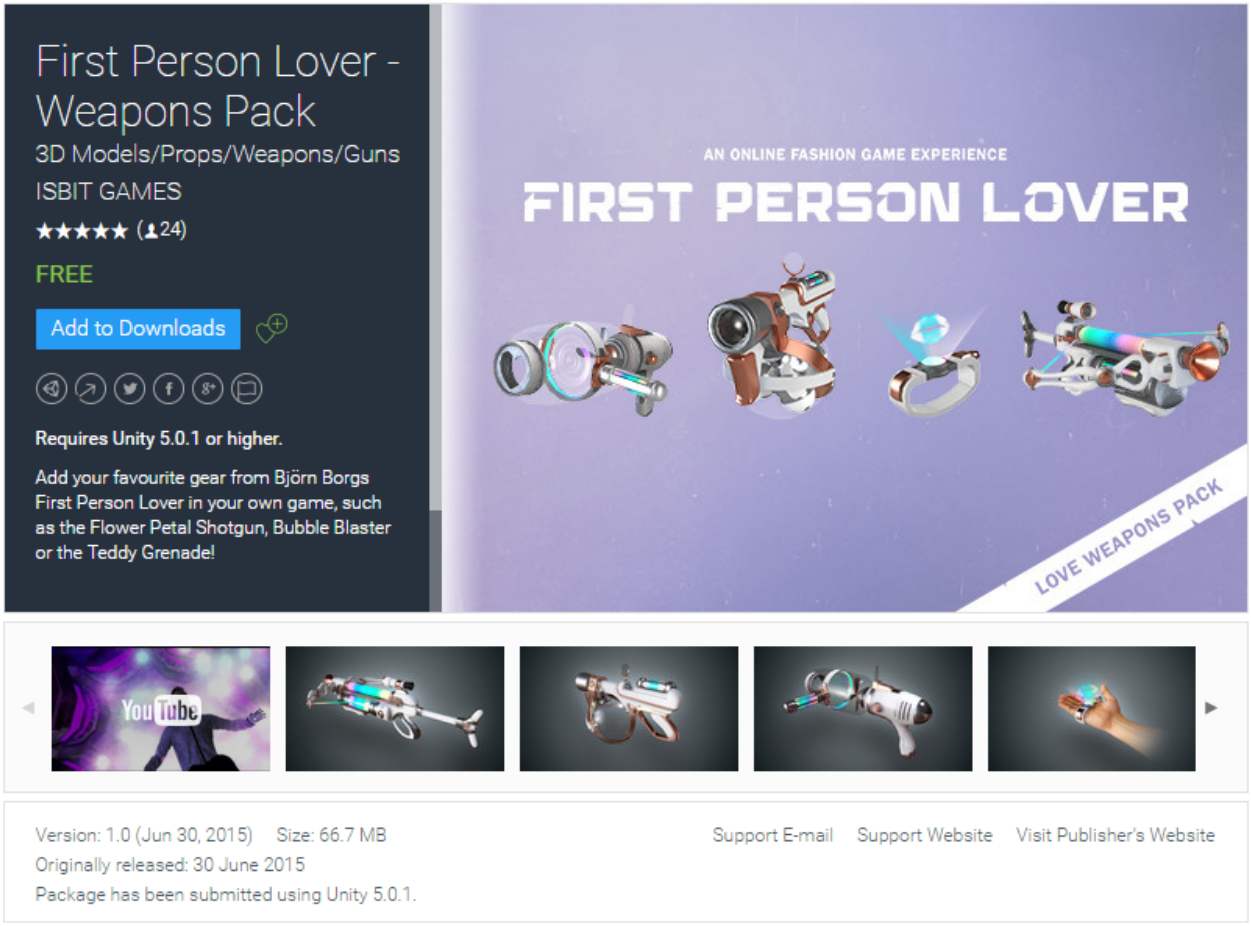
Για να έχουμε όλα τα στοιχεία μας υπό έλεγχο, πρέπει η ενεργή σκηνή που εργαζόμαστε και το τελικό αποτέλεσμα αυτής να είναι και τα δύο ορατά το ένα κάτω από το άλλο. (Το Asset Store και το Console είναι βέλτιστο να βρίσκονται μαζί με την ενεργή σκηνή καθώς με αυτό τον τρόπο μπορούμε να βλέπουμε το Debug καθώς τρέχουμε το παιχνίδι μας στο Game.) Επίσης, πρέπει να έχουμε στην ίδια στήλη τα αντικείμενα της σκηνής καθώς και τα διαθέσιμα δομικά στοιχεία και script για να τα επεξεργαζόμαστε όσο πιο εύκολα γίνεται. Τέλος, είναι απαραίτητο ο Inspector να είναι μια στήλη μόνος του καθώς είναι το πεδίο όπου μπορούμε να τροποποιήσουμε όλες τις πληροφορίες του εκάστοτε αντικειμένου καθώς και να ρυθμίσουμε τα script που εντάσσουμε.
Ολοκληρώνοντας και αυτό το βήμα, ας οργανώσουμε τα δομικά μας υλικά για να μπορέσουμε να κατασκευάσουμε σταδιακά το παιχνίδι. Για να συλλέξουμε δομικά στοιχεία υπάρχουν δύο μέθοδοι, είτε να κατασκευάσουμε εμείς δικά μας μοντέλα και υφές είτε να χρησιμοποιήσουμε ήδη υπάρχοντα από το Asset Store του Unity. Καθώς δεν διαθέτουμε την καλλιτεχνική ικανότητα να παράγουμε ικανοποιητικά αποτελέσματα προς χρήση, επιλέξαμε να χρησιμοποιήσουμε τα έτοιμα μοντέλα του Asset Store. Όμως, προκειμένου να είμαστε όσο το δυνατόν πιο κοντά στην ιδέα ενός DIY Project που θα έφτιαχνε ένας προγραμματιστής χωρίς σχεδιαστικές ικανότητες, επιλέξαμε να φτιάξουμε το παιχνίδι μας με μοντέλα τα οποία διατίθενται ΔΩΡΕΑΝ και μόνο. Δεν υπάρχει λόγος για αγορά μοντέλων όταν ο στόχος μας είναι η υλοποίηση ενός παιχνιδιού που δεν θα έχει εμπορική διάθεση αλλά αποτελεί ένα ακαδημαϊκό πόνημα.
Στην καρτέλα Asset Store που υπάρχει εντός της πλατφόρμας, θα αναζητήσουμε το Let's Try Assets της Unity Technologies και θα το κατεβάσουμε τοπικά. Στη συνέχεια, θα κάνουμε Import των Asset και θα επιτρέψουμε στην πλατφόρμα να αναδιαμορφώσει το κενό μας project σε μια σκηνή κατάλληλη για εργασία.



Εικόνα 2.3.ii: Το πακέτο Let’s Try πάνω στο οποίο θα φτιάξουμε το δικό μας παιχνίδι.

Σημείωση: κάθε φορά που κατεβάζουμε ένα πακέτο Asset, είναι σημαντικό να ελέγχουμε την συμβατότητα του με την έκδοση Unity που λειτουργούμε.

Όταν ολοκληρωθεί το import των βασικών μας δομικών στοιχείων γι' αυτό το project, θα δούμε πλέον ότι στην καρτέλα Asset υπάρχουν πλέον πολλοί νέοι φάκελοι με καινούργια στοιχεία. Προς το παρόν δεν θα ερευνήσουμε εκτενέστερα αλλά θα προχωρήσουμε στο να λάβουμε το δεύτερο βασικό δομικό στοιχείο για το παιχνίδι, δηλαδή τα όπλα για τον χαρακτήρα μας. Το πακέτο όπλων που επιλέχθηκε καθαρά με κριτήρια αισθητικής ικανοποίησης είναι αυτά του First Person Lover - Weapons Pack της ISBIT GAMES αλλά φυσικά οποιοδήποτε μοντέλο όπλου είναι εξίσου αποδεκτό. Πρέπει λοιπόν να ψάξουμε για ακόμα μια φορά στο Asset Store για το πακέτο και να κάνουμε Download τοπικά και στη συνέχεια Import στο Project για να προστεθούν στον φάκελο FPL_ISBIT.



Εικόνα 2.3.iii: Το πακέτο FPL Weapon Pack για να οπλίσουμε τον παίκτη μας.

Έχοντας όλα τα πρώτα στοιχεία που χρειαζόμαστε για να ξεκινήσουμε να φτιάχνουμε τον χαρακτήρα μας, το σύστημα βολών καθώς και τους στόχους, στα κεφάλαια που ακολουθούν, θα δούμε μεθόδους υλοποίησης των προαναφερθέντων καθώς και πώς κινούμαστε εντός της πλατφόρμας. Επίσης, θα δούμε βασικές τεχνικές κατασκευής ενός Main Menu για να έχουμε έτοιμη την εναρκτήρια πίστα του παιχνιδιού μας.

2.4 Βοηθητικό Index Αντικειμένων/ Script για τις Σκηνές του Παιχνιδιού

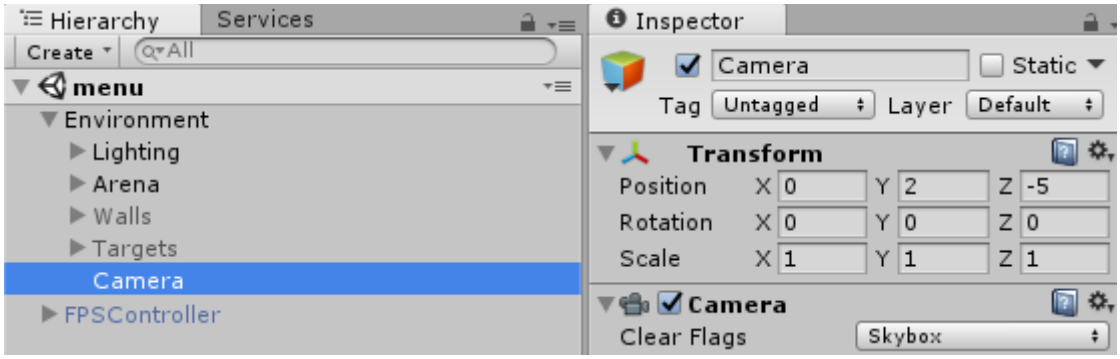
<table>
<tr><td colspan="2">menu.unity</td></tr>
<tr><td>Hierarchy</td><td>Scripts</td></tr>
<tr><td>Environment
◦ Lighting
▪ Directional Lighting
◦ Arena
▪ Arena Walls
▪ Arena Floor
◦ Camera</td><td>Basic Form/ No Scripts Added</td></tr>
<tr><td>Canvas
◦ Text Title
◦ Text Subtitle
◦ Text Song
◦ Dropdown Song
▪ Label
▪ Arrow
◦ Text Difficulty
◦ Dropdown Difficulty
▪ Label
▪ Arrow
◦ Button Play
▪ Text
◦ Button Exit
▪ Text</td><td>Canvas: MenuSelections.cs</td></tr>
<tr><td>EventSystem</td><td>Basic Form/ No Scripts Added</td></tr>
<tr><td colspan="2">main_level.unity
(η δομή είναι ακριβώς ίδια και στο main_level_hard.unity με τη μόνη διαφορά ότι έχουμε κάποιους επιπλέον Bad_Cubes)</td></tr>
<tr><td>Hierarchy</td><td>Scripts</td></tr>
<tr><td>Environment
◦ Lighting
▪ Directional Lighting
◦ Arena
▪ Arena Walls
▪ Arena Floor</td><td>Basic Form/ No Scripts Added</td></tr>
<tr><td>Canvas
◦ Target Reticle Image</td><td>Basic Form/ No Scripts Added</td></tr>
<tr><td>FPSController
◦ FirstPersonController
▪ Gun
• PetalShotGun
◦ AmmoTube
◦ GunBody
◦ PetalBalls
◦ GunEnd</td><td>Gun: RayCastShootComplete.cs/
RayViewerComplete.cs</td></tr>
</table>

Hierarchy	Scripts
VideoWalls ◦ VideoWallShape ◦ VideoWallBeliever ◦ VideoWallLight ◦ VideoWallRun ◦ VideoWallLean	VideoWalls: set_videowall.cs VideoWallShape/Believer/Light/Run/Lean: AudioProcessor.cs/ random_spawn.cs
Good Cubes ◦ Line 1 ▪ Good Cube 1-1 ▪ Good Cube 1-3 ▪ Good Cube 1-4 (not in hard level) ▪ Good Cube 1-5 ▪ Good Cube 1-7 ◦ Line 2 ▪ Good Cube 2-1 (not in hard level) ▪ Good Cube 2-2 ▪ Good Cube 2-4 ▪ Good Cube 2-6 ▪ Good Cube 2-7 ◦ Line 3 ▪ Good Cube 3-1 ▪ Good Cube 3-2 (not in hard level) ▪ Good Cube 3-3 ▪ Good Cube 3-5 ▪ Good Cube 3-6 ◦ Line 4 ▪ Good Cube 4-2 ▪ Good Cube 4-3 (not in hard level) ▪ Good Cube 4-4 ▪ Good Cube 4-6 ▪ Good Cube 4-7 ◦ Line 5 ▪ Good Cube 5-1 ▪ Good Cube 5-2 (not in hard level) ▪ Good Cube 5-3 ▪ Good Cube 5-5 ▪ Good Cube 5-7	Good Cube X-Y: ShootableBox.cs/ Rotator.cs
Bad Cubes ◦ Line 1 ▪ Bad Cube 1-2 ▪ Bad Cube 1-4 (only in hard level) ▪ Bad Cube 1-6 ◦ Line 2 ▪ Bad Cube 2-1 (only in hard level) ▪ Bad Cube 2-3 ▪ Bad Cube 2-5 ◦ Line 3 ▪ Bad Cube 3-2 (only in hard level) ▪ Bad Cube 3-4 ▪ Bad Cube 3-7	Bad Cube X-Y: ShootableBox.cs/ Rotator.cs

Hierarchy	Scripts
◦ Line 4 ▪ Bad Cube 4-1 ▪ Bad Cube 4-3 (only in hard level) ▪ Bad Cube 4-5 ◦ Line 5 ▪ Bad Cube 5-2 (only in hard level) ▪ Bad Cube 5-4 ▪ Bad Cube 5-6	Bad Cube X-Y: ShootableBox.cs/ Rotator.cs
Score 3D Text ◦ Score Text ◦ MinScore Text	Basic Form/ No Scripts Added
EndGame Components ◦ WinLose Text ◦ Reload Cube ◦ Menu Cube ◦ Exit Cube	Reload/Menu/Exit Cube: Shootable.cs
EventSystem	Basic Form/ No Scripts Added

2.5 Σχεδιάζοντας το Κεντρικό Μενού του Παιχνιδιού

Σαν πρώτη επαφή με την υλοποίηση ενός παιχνιδιού στην πλατφόρμα μας, η κατασκευή ενός κεντρικού μενού είναι μια εύκολη και δομημένη σκηνή που θα εξομαλύνει την είσοδο μας στο Game Development.
Ας δούμε για αρχή τα περιεχόμενα των φακέλων των πακέτων που κατεβάσαμε στο προηγούμενο κεφάλαιο. Είναι σημαντικό να γνωρίζουμε ανά πάσα στιγμή τα διαθέσιμα εργαλεία που έχουμε στο δυναμικό μας. Εξερευνήστε τους φακέλους και ελέγξτε τα textures (υφές αντικειμένων), τα prefabs (μοντέλα), ηχητικά κλιπ, τις έτοιμες σκηνές, ότι έχετε διαθέσιμο.
Στη συνέχεια, από το πακέτο Let's Try, θα ανοίξουμε από τον φάκελο Scenes την πίστα με το όνομα ShootingWithRaycasts. Σε αυτό το κεφάλαιο δεν μας ενδιαφέρει τόσο να εξετάσουμε την ήδη υπάρχουσα δομή της πίστας/δείγμα αλλά να στήσουμε τα δικά μας στοιχεία πάνω στο απλό γκρι πλέγμα που περιστοιχίζει την πίστα. Οπότε, σαν πρώτη κίνηση, θα κάνουμε κλικ στο στοιχείο FPSController που βρίσκεται στην καρτέλα Hierarchy και θα ελέγξουμε τον Inspector. Δεν είναι ανάγκη να διαγράψουμε ένα αντικείμενο για να μην είναι εμφανές στην πίστα μας, μόνο να το απενεργοποιήσουμε. Εάν ξεμαρκάρουμε το check box που βρίσκεται στα αριστερά του ονόματος του αντικειμένου, στην κορυφή του Inspector, το αντικείμενο πλέον δεν εμφανίζεται στην πίστα και επίσης γίνεται γκρίζο/ανενεργό στην ιεραρχία. Θα δούμε ότι στην καρτέλα Game όμως ότι έχει "σκοτεινιάσει" η πίστα. Αυτό είναι επειδή πλέον δεν έχουμε ενεργή κάμερα να μας προβάλει τα περιεχόμενα της σκηνής. Αυτό θα το διευθετήσουμε στα επόμενα βήματα. Ανοίγοντας το αντικείμενο Environment, απενεργοποιούμε επίσης τα αντικείμενα Walls και Targets αφήνοντας ενεργά μόνο το Lightining και Arena.
Στη συνέχεια, όντας ακόμα στην καρτέλα Hierarchy και πατώντας το κενό, όχι πάνω σε κάποια από τα αντικείμενα, κάνουμε Create → Camera. Θα δούμε ότι στην ιεραρχία έχει δημιουργηθεί στο τέλος ένα νέο αντικείμενο με το όνομα Camera. Αυτό θα το συγκαταλέξουμε στα στοιχεία περιβάλλοντος και θα το σύρουμε μέσα στο Environment αλλά με προσοχή μην μπει κάτω στην ιεραρχία από κάποια από τα ήδη υπάρχοντα αντικείμενα. Στη συνέχεια, θα κάνουμε Transform της θέσης του στο χώρο για να μας δίνει την οπτική του περιβάλλοντος που είναι βέλτιστη για την σκηνή. Η θέση του αντικειμένου Camera στην ιεραρχία και το transform της πρέπει να είναι όπως φαίνεται στην κάτωθι εικόνα.

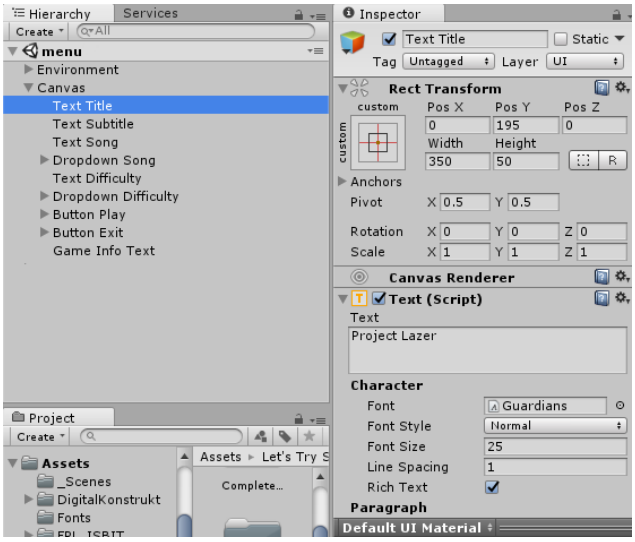


Εικόνα 2.5.i: Σωστή θέση κάμερας στην Ιεραρχία και βέλτιστη θέση στο χώρο

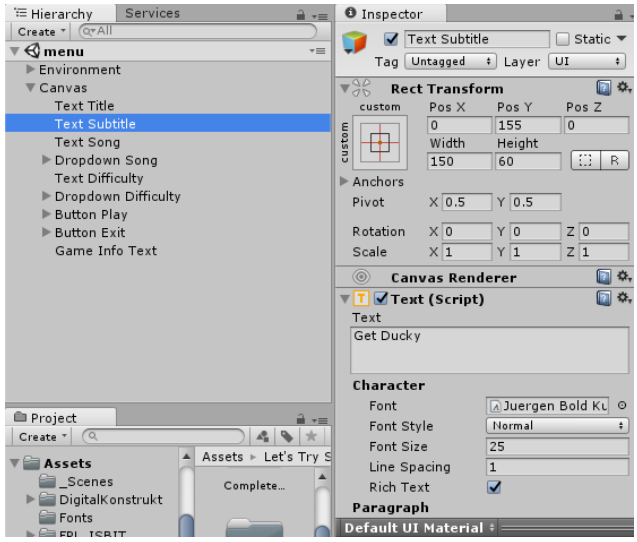
Έχοντας ολοκληρώσει την ρύθμιση της κάμερας, ας προχωρήσουμε με την υλοποίηση των γραφικών στοιχείων του μενού. Τα default δομικά στοιχεία ενός μενού ανήκουν στην κατηγορία UI και όλα είναι υποαντικείμενα του Canvas. Η θέση των αντικειμένων που υπάγονται στο Canvas είναι σε συνάρτηση της εκάστοτε κάμερας που υπάρχει στη σκηνή. Εάν έχουμε στατική κάμερα, όπως στην συγκεκριμένη σκηνή, η προβολή των στοιχείων Canvas είναι σε σταθερή θέση. Εάν είχαμε έναν FPS χαρακτήρα όμως, όπως θα δούμε και στα ακόλουθα κεφάλαια, η θέση των Canvas αντικειμένων ακολουθεί το πεδίο όρασης της κάμερας. Το μενού που θα υλοποιήσουμε θέλουμε να έχει τον τίτλο του παιχνιδιού, τις επιλογές για το επίπεδο δυσκολίας καθώς και το τραγούδι της πίστας, το play button, το exit button και φυσικά τις οδηγίες του παιχνιδιού.

Εικόνα 2.5.ii: Οπτική παρουσίαση του μενού όπως θα πρέπει να φαίνεται στην καρτέλα σκηνής καθώς το κατασκευάζουμε.

Ας ξεκινήσουμε λοιπόν με την κατασκευή του τίτλου και του υπότιτλου του παιχνιδιού. Όπως και με την προσθήκη της κάμερας, πατάμε στο κενό τμήμα του Hierarchy και κάνουμε Create → UI → Text. Θα δούμε ότι δημιουργείται ένα γενικό αντικείμενο Canvas και κάτω από αυτό έχουμε πλέον ένα αντικείμενο Text. Το μετονομάζουμε σε Text Title και κάνουμε δεξί κλικ πάνω του και επιλέγουμε Duplicate. Το αντικείμενο κλώνος που δημιουργήθηκε το μετονομάζουμε σε Text Subtitle. Κάνοντας κλικ πάνω στο κάθε αντικείμενο αλλάζουμε το Transform όπως φαίνεται στις κάτωθι εικόνες.



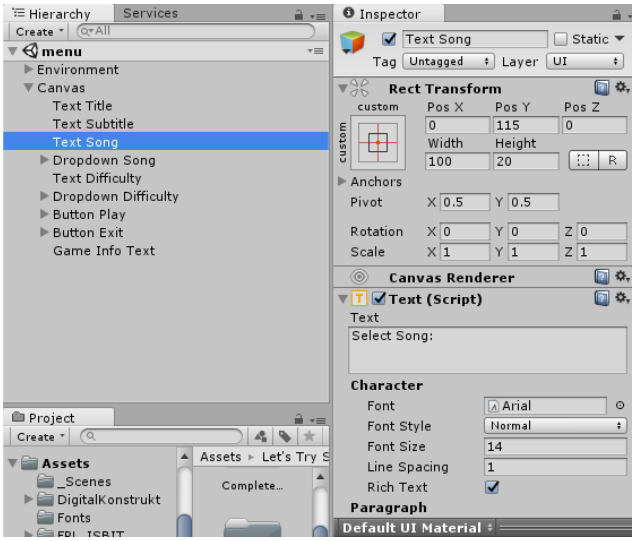
1) Title Settings



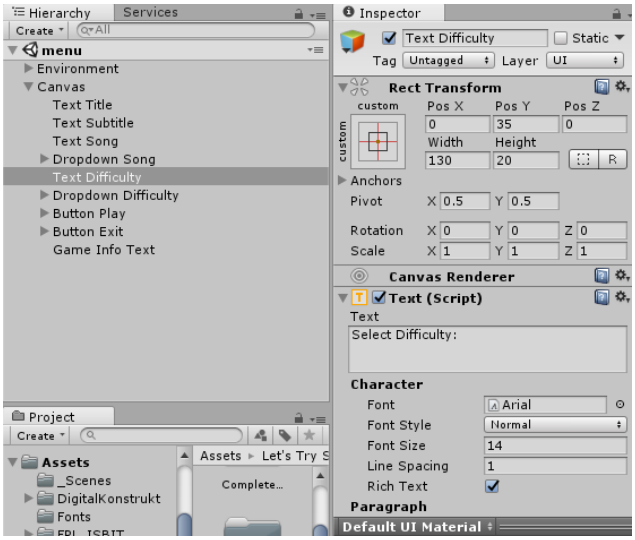
2) Subtitle Settings

Στα πεδία Text του Inspector θα βάλουμε για τον τίτλο Project Lazer ενώ για τον υπότιτλο Get Ducky. Φυσικά μπορούσε να είναι οποιοδήποτε άλλος τίτλος και υπότιτλος αλλά για το συγκεκριμένο Project είναι ένα χιουμοριστικό συνονθύλευμα των στοιχείων έμπνευσης του παιχνιδιού (Project: Project Diva/ Lazer: Major Lazer/ Get Ducky: Get Lucky – Daft Punk και Duck Hunt). Για να μην έχουν το default font που είναι Arial μπορούμε να βάλουμε τα δικά μας όπως φαίνεται και στην οπτική παρουσίαση του Μενού. Για να προσθέσουμε Font, μπορούμε απλά να προσθέσουμε ένα φάκελο στα Assets κάνοντας κλικ στο κενό τμήμα της καρτέλας Assets και να πατήσουμε Create → Folder. Το μετονομάζουμε σε

Fonts, τον ανοίγουμε και στην δεξιά πλευρά της καρτέλας που βλέπουμε το κενό περιεχόμενο του φακέλου κάνουμε δεξί κλικ και επιλέγουμε Import New Asset. Για να αλλάξουμε το Font του κάθε Text, πάμε απλά στον Inspector και στα πεδία Character, πατώντας στον κύκλο δίπλα από το Arial στο Font ανοίγει το παράθυρο επιλογής στοιχείου από τα Asset ή τη σκηνή. Από τα Asset, επιλέγουμε το Font που επιθυμούμε και βλέπουμε να αλλάζει και στη σκηνή επιτόπου.
Για να φτιάξουμε στη συνέχεια τα βοηθητικά κείμενα πάνω από κάθε Dropdown Menu, πατάμε πάλι στο κενό τμήμα του Hierarchy και κάνουμε Create → UI → Text. Το μετονομάζουμε σε Text Song, το κάνουμε Duplicate, μετονομάζουμε τον κλώνο σε Text Difficulty και αλλάζουμε στον Inspector τα Transform όπως φαίνεται στις εικόνες που ακολουθούν.

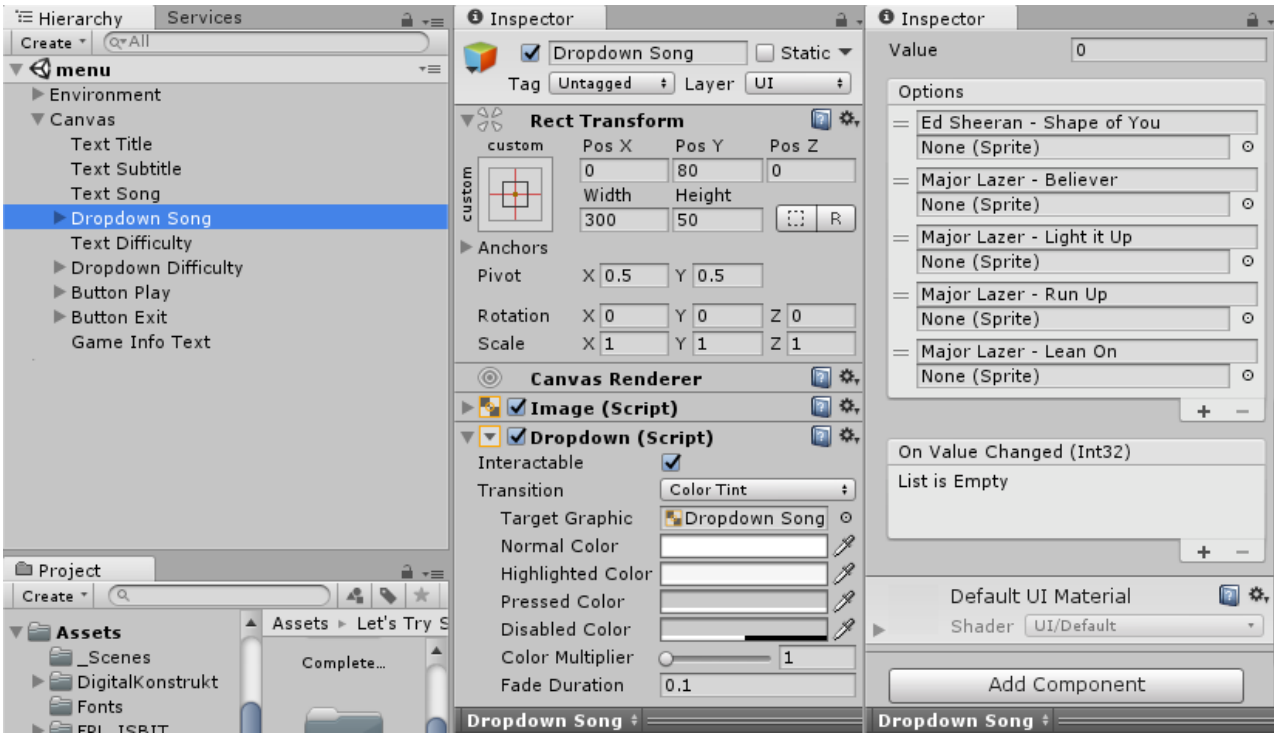


3) Select Song Text

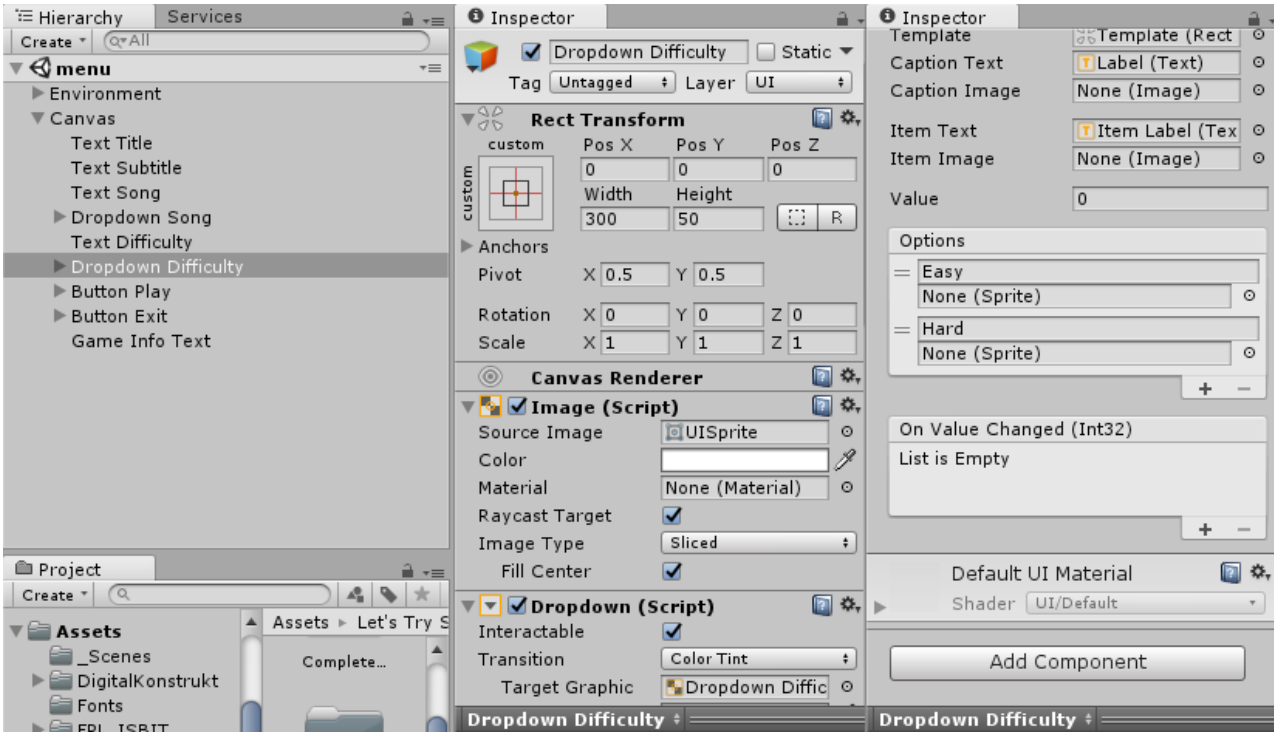


4) Select Difficulty Text

Αντίστοιχα με τη δημιουργία των νέων αντικειμένων Text, για να φτιάξουμε τα Dropdown Menu με τις επιλογές των παιχτών κάνουμε Create → UI → Dropdown. Το δημιουργηθέν αντικείμενο έχει κάποια υποαντικείμενα σε αυτό που είναι το Label και το Arrow, τα βασικά οπτικά στοιχεία που φαίνονται στον παίχτη όταν δεν είναι expanded και δεν είναι ορατές όλες οι επιλογές του. Το Label θα εμφανίζει αρχικά την πρώτη επιλογή στη λίστα των διαθέσιμων και σε περίπτωση που ο παίχτης διαλέξει κάποια άλλη τότε θα εμφανίζεται αυτή. Μετονομάζουμε το Dropdown αντικείμενο σε Dropdown Song. Θα δημιουργήσουμε και εδώ έναν κλώνο του αντικειμένου αυτού αλλά προσέχουμε να κάνουμε κλικ πάνω στο ανώτερο αντικείμενο της ιεραρχίας, δηλαδή το Dropdown Song, και πατάμε δεξί κλικ Duplicate. Μετονομάζουμε και τον κλώνο σε Dropdown Difficulty και πλέον θα ασχοληθούμε με τον ορισμό των διαθέσιμων επιλογών σε κάθε Dropdown. Για να αλλάξουμε τις ρυθμίσεις του κάθε αντικειμένου, θα πρέπει να κάνουμε τις αλλαγές στον Inspector, στο Dropdown Script, στα πεδία Options. Να τονίζουμε ότι δεν πρόκειται για script που χρειάζεται να επεξεργαστούμε με κώδικα εμείς αλλά μόνο μέσω του Inspector καθώς είναι βασικός κώδικας που διατίθεται από την ίδια την πλατφόρμα. Θα αλλάξουμε τη θέση και το μέγεθος των αντικειμένων όπως φαίνεται στις εικόνες που ακολουθούν καθώς και το κείμενο που πρέπει να ενταχθεί σε κάθε πεδίο του Options. Για να προσθέσουμε ή να αφαιρέσουμε πεδία, απλά πατάμε τα σύμβολα + και – κάτω δεξιά.



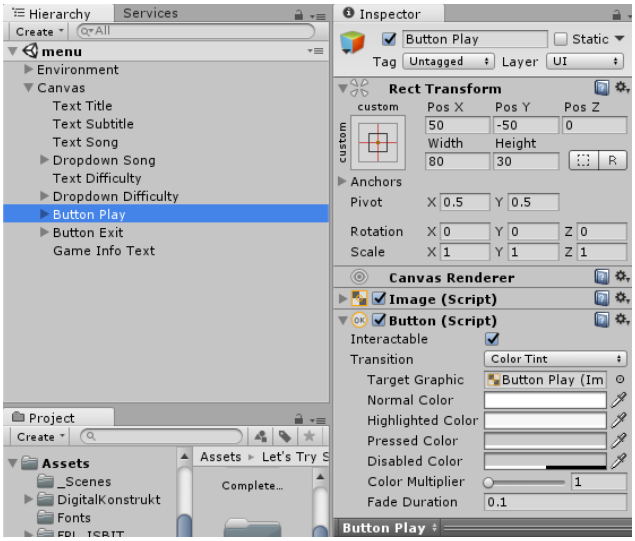
5) Song Dropdown Menu



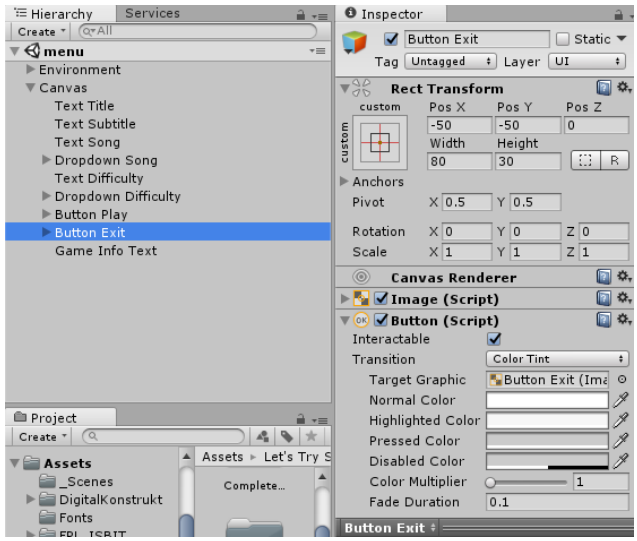
6) Difficulty Dropdown Menu

Σημείωση: Οι επιλογές που αναγράφονται στα Options φυσικά μπορούν να προσαρμοστούν σε οποιαδήποτε επιλογή και να γίνει για τα τραγούδια που θα μπορούν να παίζουν οι παίχτες ή και να αλλάξουν τα επίπεδα δυσκολίας. Όλα είναι θέμα επιλογών του δημιουργού.

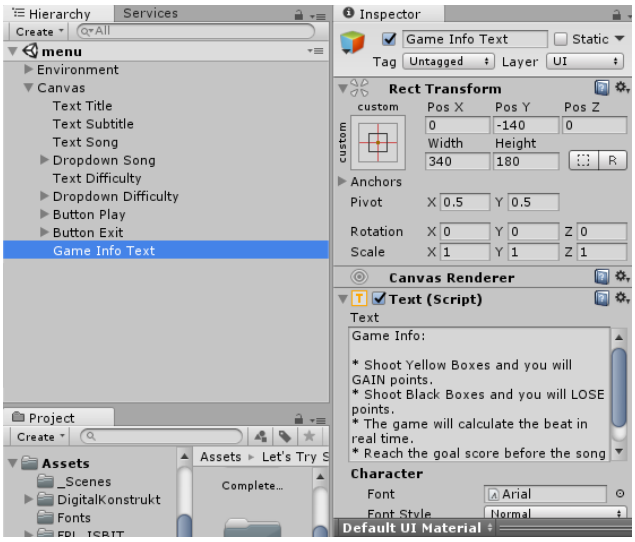
Πλέον, μένει μόνο να προσθέσουμε τα αντικείμενα για τα κουμπιά και τις οδηγίες του παιχνιδιού. Καθώς είδαμε πως δημιουργούνται τα UI αντικείμενα και στα προηγούμενα βήματα, κάνουμε Create → UI → Button για τα δυο κουμπιά και Create → UI → Text για τις οδηγίες. Μετονομάζουμε τα αντικείμενα, αλλάζουμε τα Transform και προσθέτουμε τις οδηγίες ακριβώς όπως φαίνεται και στις επόμενες εικόνες.



7) Play Button Settings



8) Exit Button Settings



9) Game Info Settings

Text for Game Info Text

Game Info:

* Shoot Yellow Boxes and you will GAIN points.
* Shoot Black Boxes and you will LOSE points.
* The game will calculate the beat in real time.
* Reach the goal score before the song is over to WIN!
* Oculus Rift Compatible!

Σε αυτό το σημείο, έχουμε πλέον ολοκληρώσει με αυτό το βήμα όλη την κατασκευή του Menu. Ας δημιουργήσουμε λοιπόν και το script που θα διαχειρίζεται τα στοιχεία που έχουμε. Θέλουμε ένα script που να μπορεί να κρατάει τις επιλογές που κάνει ο παίχτης στο Dropdown χωρίς να χάνονται με την φόρτωση της κεντρικής πίστας. Υπάρχουν τέσσερις διαφορετικοί τρόποι για να αποθηκεύσουμε πληροφορία χωρίς να καταστρέφεται με το κάθε loading διαφορετικού επιπέδου. Οι μέθοδοι είναι οι ακόλουθοι:

- χρήση της δομής PlayerPrefs για να αποθηκεύσουμε τις επιλογές (βέλτιστη μέθοδος/ λειτουργεί μόνο με βασικούς τύπους μεταβλητών όπως int)
- χρήση static variable
- δημιουργία ενός βοηθητικού gameObject (αντικειμένου) και να γίνει κλήση της DontDestroyOnLoad(…)
- χρήση βοηθητικού αρχείου (μακράν η χειρότερη λύση για καταγραφή λίγων αριθμών/ να γίνεται χρήση μόνο για κείμενα)

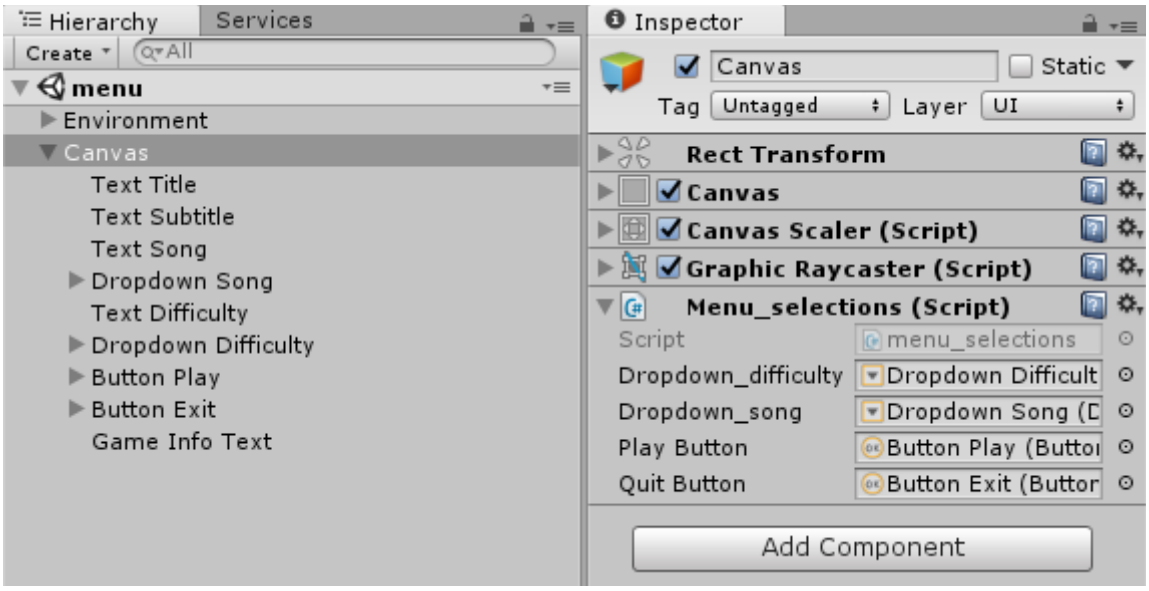
Προτείνεται η δημιουργία ενός νέου φακέλου στα Assets που να ονομάζεται Scripts_Videos για να είναι πιο εύκολα προσβάσιμοι οι κώδικες μας και να είναι διαχωρισμένοι από τους υπόλοιπους. Στον νέο φάκελο λοιπόν κάνουμε δεξί κλικ και Create → C# Script, το μετονομάζουμε σε menu_selections.cs και το ανοίγουμε για επεξεργασία. Ο κώδικας που πρέπει να συντάξουμε είναι ο ακόλουθος:

menu_selections.cs

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

public class menu_selections : MonoBehaviour
{
      //dropdown_difficulty: the dropdown menu for the easy/hard selection
  public Dropdown dropdown_difficulty;
       //dropdown_song: the dropdown menu for selecting the song that the player wants to play with
      public Dropdown dropdown_song;
      //playButton: the button that initiates the main level scene
  public Button playButton;
      //quitButton: the button that exits the game
      public Button quitButton;

  void Awake()
  {
            //when the game begins, we add listeners for the buttons so that once the player hits them, they each call their respective function
    playButton.onClick.AddListener(new UnityAction(SaveNumberToPlayerPrefs));
            quitButton.onClick.AddListener(new UnityAction(QuitGame));
  }

     //SaveNumberToPlayerPrefs is activated once the player hits the play button
  private void SaveNumberToPlayerPrefs()
  {
             //selectedDifficulty: keeps the current selection of the difficulty dropdown menu
    int selectedDifficulty = dropdown_difficulty.value;
            //selectedSong: keeps the current selection of the song dropdown menu
            int selectedSong = dropdown_song.value;
             //we chose to utilise the PlayerPrefs struct to keep certain variables unaffected throughout the levels of the game
             //in this script, we will set the values for the player's selected difficulty and song
            PlayerPrefs.SetInt("SelectedDifficulty", selectedDifficulty);
            PlayerPrefs.SetInt("SelectSong", selectedSong);
             //we show the values in the Debug Log to be certain they are properly stored
            Debug.Log ("Song = " + selectedSong);
            Debug.Log ("Difficulty = " + selectedDifficulty);
             //the dropdown menus start from the value 0, so if the player selects easy (0) we load the easy scene or if the player selects hard (1) we load the hard one
            if (selectedDifficulty == 0){
                 SceneManager.LoadScene("main_level");
                 //Application.LoadLevel("main_level");             //obsolete expression, not valid in Unity 5
```

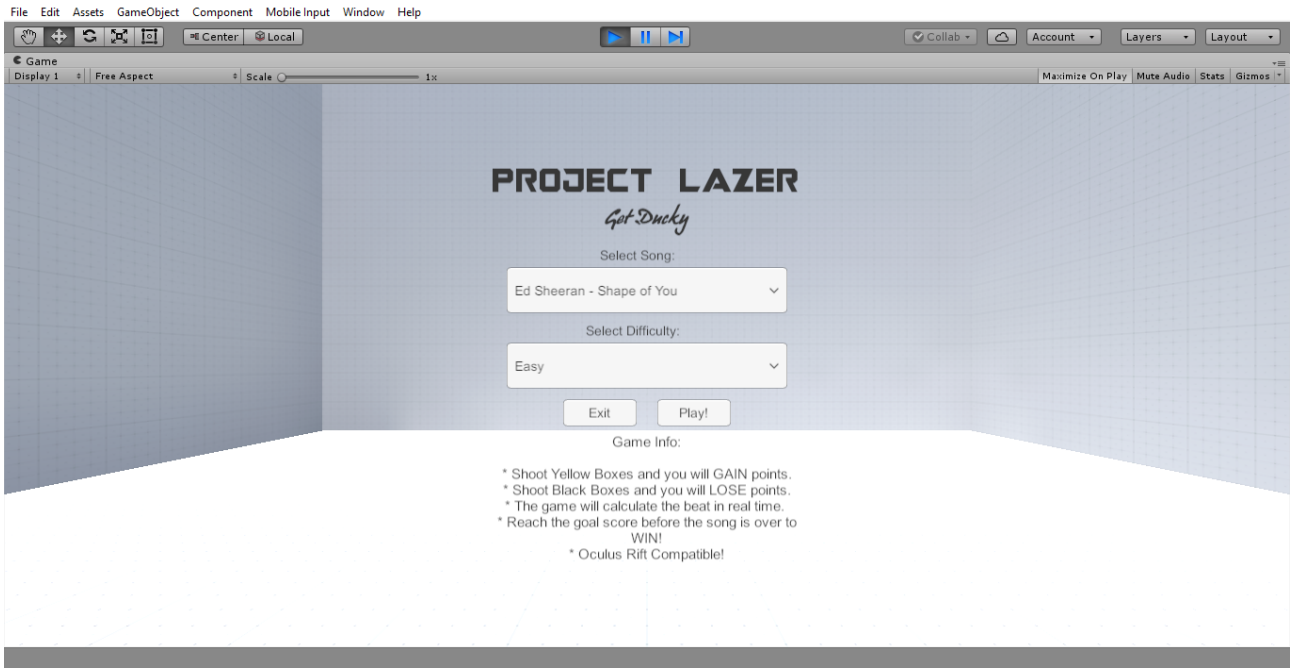
```
            }
            else if (selectedDifficulty == 1){
                  SceneManager.LoadScene("main_level_hard");
            }
   }

      //QuitGame is activated once the player hits the exit button
      public void QuitGame()
      {
            //when we test run the game in Unity we can not actually exit the
game so in order to check the functionality of our script we need a test
Debug message
            Debug.Log ("Game is exiting...");
            Application.Quit();
      }
}
```

Έχοντας ολοκληρώσει και το script μας, ήρθε η ώρα να το εντάξουμε και σε ένα αντικείμενο για μπορεί να καλεστεί από αυτό. Το αντικείμενο που επιλέγουμε είναι το ανώτατο στην ιεραρχία των γραφικών στοιχείων, δηλαδή το Canvas. Για να βάλουμε το script στο αντικείμενο, το πιάνουμε από την καρτέλα των Asset και το κάνουμε Drag and Drop πάνω στο αντικείμενο Canvas και βλέπουμε ότι εντάσσεται ως στοιχείο του στο Inspector. Προσοχή να μην το βάλουμε κατά λάθος σε άλλο αντικείμενο! Για να μπορεί να επικοινωνεί το script με τα αντικείμενα της σκηνής, πρέπει να τα αντιστοιχίσουμε στο Inspector. Τα αντικείμενα που θα επικοινωνούν είναι όπως είδαμε και στον κώδικα, τα Dropdown και τα Buttons. Οι αντιστοιχίσεις γίνονται επιλέγοντας τον κύκλο στα δεξιά κάθε πεδίου και επιλέγοντας από το αναδυόμενο παράθυρο το εκάστοτε αντικείμενο από την καρτέλα Scene. Οι αντιστοιχίσεις εν τέλη θα πρέπει να είναι όπως φαίνονται στην εικόνα.



Εικόνα 2.5.iii: Αντιστοίχιση αντικειμένων και του Script για ολοκλήρωση του μενού.

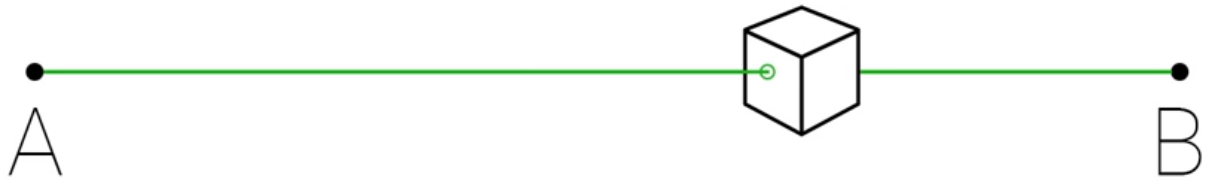
Τώρα μένει μόνο να ορίσουμε ότι αυτή η σκηνή είναι το μενού μας. Στον φάκελο Assets κάνουμε Create → Folder και ονομάζουμε τον φάκελο _Scenes για να είναι πάντα στην κορυφή των Asset. Στο κεντρικό μενού της πλατφόρμας, επιλέγουμε File → Save Scene As και στο παράθυρο πηγαίνουμε στον φάκελο _Scenes και την αποθηκεύουμε ως menu.unity. Έχοντας κάνει και αυτό το τελευταίο βήμα, μπορούμε πλέον να τρέξουμε τη σκηνή μας για να δούμε το τελικό αποτέλεσμα. Φυσικά εάν πατήσουμε το Play Button δεν θα έχουμε μεταφορά σε κάποια πίστα γιατί είναι η μόνη κατασκευασμένη αλλά θα φροντίσουμε γι αυτό στο κεφάλαιο που ακολουθεί.



Εικόνα 2.5.iv: Το μενού του παιχνιδιού μας στην τελική δοκιμή.

2.6 Κατασκευάζοντας το Raycasting

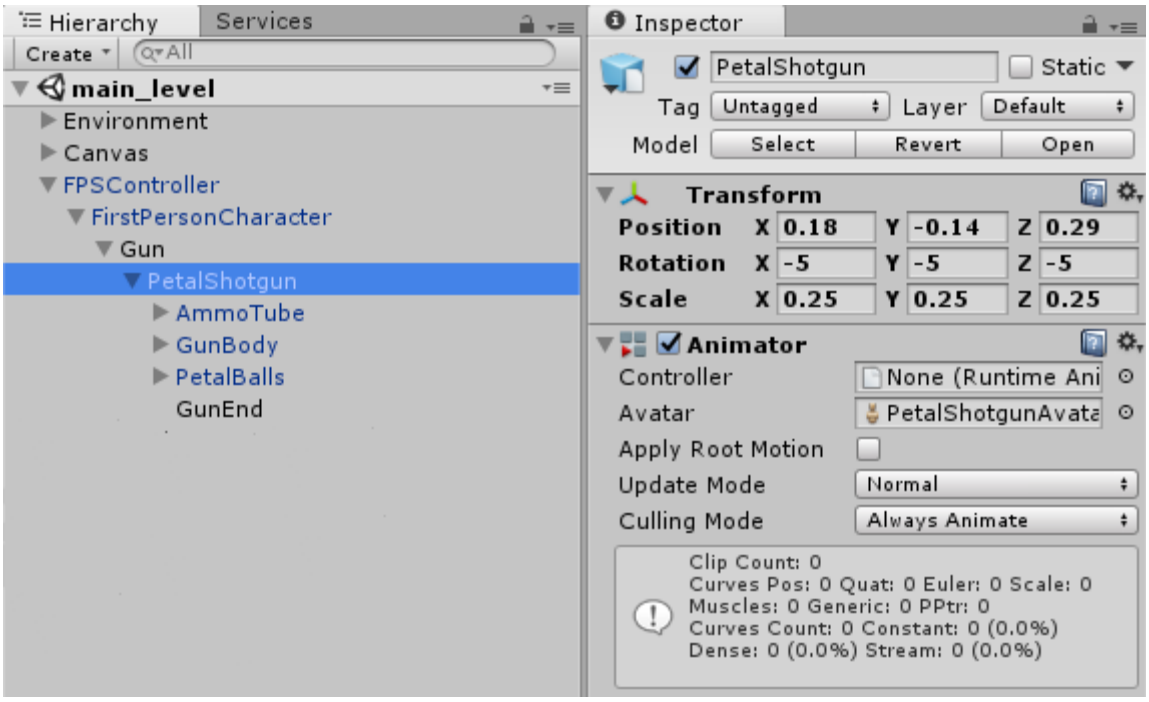
Σε αυτό το κεφάλαιο θα ασχοληθούμε με τα στάδια που απαιτούνται προκειμένου ένας χαρακτήρας πρώτου προσώπου να μπορεί να πυροβολεί με το όπλο της επιλογής μας μέσω Raycast. Η τεχνοτροπία του Raycast είναι πολύ αγαπητή στα Space Shooters καθώς επίσης και σε κάποιους κλασσικούς FPS τίτλους όπως το Doom ή το Quake3D (απλά με διαφορετικό rendering της ακτίνας βάση των Lighting/Shading/Object Rendering δυνατοτήτων του εκάστοτε Engine).
Ας ξεκινήσουμε όμως εξηγώντας πως λειτουργεί το Raycast και γιατί επιλέχθηκε έναντι των κλασσικών Bullet Ballistics που χρησιμοποιούνται σε μεγάλα FPS Games όπως το Counter Strike. Κάθε βολή με Raycast είναι μια ακτίνα που έχει ως σημείο εκκίνησης την κάνη του όπλου και σχηματίζει ευθύγραμμη πορεία μέχρι το τελικό σημείο που μπορεί να φτάσει στον χώρο βάση απόστασης που ορίζουμε εμείς (ισχύς του όπλου). Όπως σε κάθε FPS Game, παίζοντας σε Η/Υ στοχεύουμε με χρήση της κάμερας ενώ στο VR με χρήση του Gyroscope και την κίνηση του κεφαλιού μας. Η κάθε βολή Raycast θα καταλήξει στο σημείο όπου “κοιτάζει“ ο παίκτης κάθε φορά, χωρίς αλλοιώσεις. Η κύρια διαφορά μεταξύ των δυο είναι ότι κάθε βολή που πραγματοποιείται με Raycast κινείται ευθύγραμμα, για προκαθορισμένη απόσταση ενώ στα Ballistics, κάθε σφαίρα είναι ένα σώμα με μάζα που ακολουθεί πορεία αντίστοιχη με τα physics του κόσμου. Δηλαδή, σε έναν κόσμο με φυσική αντίστοιχη με του δικού μας, μια σφαίρα από ένα πιστόλι θα κινούταν σε μικρότερη απόσταση με σταδιακή πτώση προς το έδαφος (καμπυλόγραμμα) σε σχέση με ένα όπλο ακτίνας που η κάθε βολή του δεν θα επηρεαζόταν καθόλου από εξωτερικούς παράγοντες και θα διατηρούσε όλα τα preset του. Η κάθε βολή που πραγματοποιείται με Raycast έχει τη δυνατότητα να φέρει πληροφορία για το χωρικό σημείο όπου υπήρχε επαφή (Collision) με άλλα αντικείμενα (συμπεριλαμβανομένου τη θέση του αντικειμένου), το μήκος της ακτίνας τη στιγμή της επαφής καθώς και πληθώρα άλλων πληροφοριών.



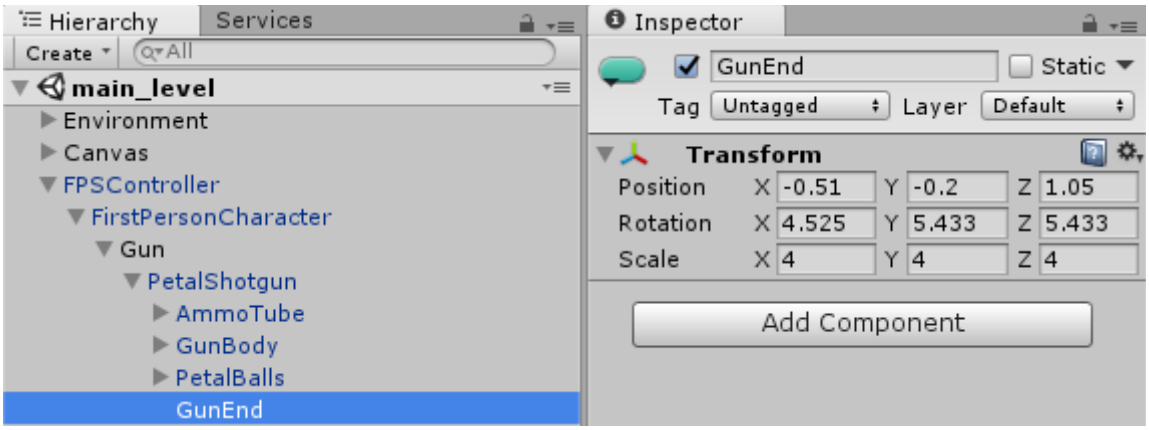
Εικόνα 2.6.i: Οπτική απεικόνιση της λειτουργίας του Raycast

Έχοντας κατεβάσει το πακέτο Let’s Try στο προηγούμενο κεφάλαιο, θα ξεκινήσουμε να χτίζουμε τον FPS χαρακτήρα. Στην καρτέλα των Asset θα ανοίξουμε τον φάκελο Scenes και θα κάνουμε διπλό κλικ στη σκηνή ShootingWithRaycasts. Στην κεντρική καρτέλα Scene θα δούμε την πίστα μας όπως έχει κατασκευαστεί. Υπάρχουν ήδη κάποια ενεργά στοιχεία όπως τοίχοι, στόχοι, βάθρα και ο FPS χαρακτήρας μας. Στην καρτέλα Hierarchy υπάρχουν δυο βασικά αντικείμενα που έχουν εμφωλευμένα άλλα αντικείμενα χαμηλότερα στην ιεραρχία. Σκοπός μας στο Unity είναι να πηγαίνουν οι ιεραρχίες αντικειμένων από το γενικό προς το ειδικό. Χαρακτηριστικά, το αντικείμενο Environment είναι το συλλογικό στοιχείο των αντικειμένων που συνθέτουν το Stage μας (Lightning, Arena, Walls, Targets). Σε αυτό το σημείο, ας αδράξουμε την ευκαιρία να απενεργοποιήσουμε τα αντικείμενα Walls και Targets από τον Inspector τους για να έχουμε έτοιμη τη σκηνή μας για τα επόμενα βήματα.
Ανοίγοντας την ιεραρχία του αντικειμένου FPSController βρίσκουμε να υπάρχει ήδη το αντικείμενο FirstPersonCharacter και εμφωλευμένο σε αυτό το αντικείμενο Gun. Στην καρτέλα Assets, θα ανοίξουμε το path FPL_ISBIT/WeaponsPack/3D/PetalShotgun και θα σύρουμε το μοντέλο PetalShotgun μέσα στο αντικείμενο Gun στο Hierarchy. Όπως βλέπουμε το PetalShotgun δεν εμφανίζεται ως απλό αντικείμενο στην ιεραρχία αλλά ως ένα σύνολο από υπομοντέλα (AmmoTube, GunBody, PetalBalls) τα οποία απαρτίζονται και αυτά από άλλα υπομοντέλα. Δεν θα πειράξουμε κάποιο από αυτά τα δομικά αντικείμενα, αλλά κάτω από το PetalShotgun θα κάνουμε Create → Create Empty και θα το μετονομάσουμε σε GunEnd. Για να τοποθετήσουμε σωστά τα νέα αντικείμενα που προσθέσαμε στο

χώρο, πρέπει να τα επιλέξουμε στην ιεραρχία και να αλλάξουμε τις τιμές του Transform στα πεδία που φαίνονται στο Inspector. Οι τιμές πρέπει να είναι όπως φαίνονται στις ακόλουθες εικόνες.

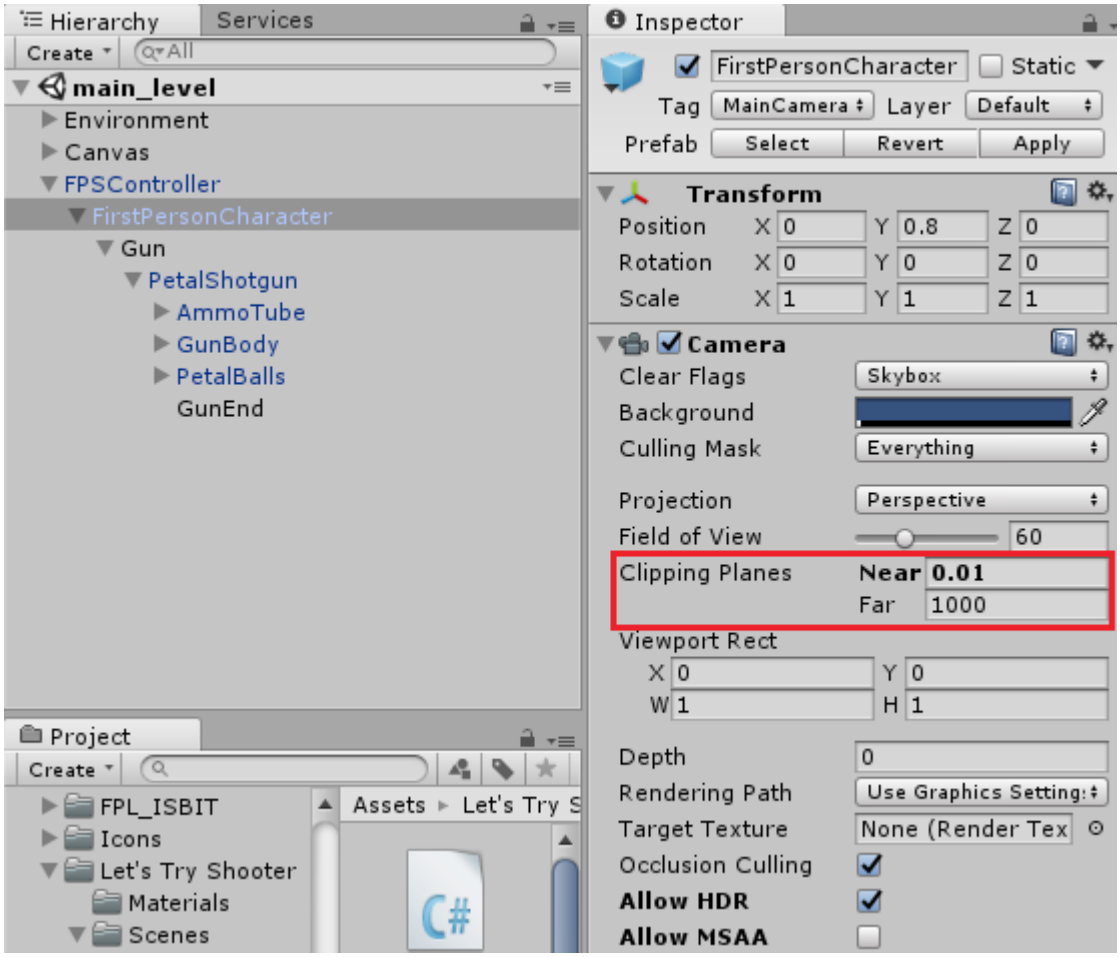


1) Transform PetalShotgun



2) Transform GunEnd

Στη συνέχεια θα παρατηρήσουμε ότι παρόλο και ας έχουμε ορίσει σωστά τα tranform μας ότι μπορούμε να δούμε μέσα από το όπλο μας. Το συγκεκριμένο είναι ένα οπτικό τρικ που κάνει η κάμερα του FPS χαρακτήρα προκειμένου να προσομοιώνει πιο ρεαλιστικά την ανθρώπινη όραση σε ένα μέσο παιχνίδι. Στην πραγματικότητα όμως είναι κάτι εντελώς άχρηστο σε ένα FPS Game όπου θέλουμε να έχουμε πλήρη ορατότητα του πεδίου. Στο αντικείμενο FirstPersonCharacter υπάρχει η κάμερα, δηλαδή τα ¨μάτια¨ του χαρακτήρα μας. Κοιτάζοντας τον Inspector στο πεδίο Camera θα δούμε μια παράμετρο με την ονομασία Clipping Planes η οποία όπως αναφέραμε περιορίζει το οπτικό πεδίο για να προσδίδει αληθοφάνεια. Αλλάζοντας την τιμή του πεδίου στην μικρότερη δυνατή, τότε θα παρατηρήσουμε ότι πλέον δεν βλέπουμε μέσα από το όπλο αλλά λες και το κρατάμε κανονικά.



Εικόνα 2.6.ii: Δημιουργώντας την ορθή αληθοφανή όραση για ένα FPS Game χαρακτήρα.

Έχοντας δημιουργήσει όλα τα βασικά αντικείμενα μας, θα προχωρήσουμε στην υλοποίηση του script που θα πραγματοποιεί το Raycast. Το project διαθέτει ήδη ένα φάκελο στα Asset του με το όνομα Scripts. Καλό θα ήταν να αφαιρέσουμε τα ήδη υπάρχοντα για να δημιουργήσουμε εκ νέου τα δικά μας. Στον κενό πλέον φάκελο, κάνουμε Create → C# Script. Μετονομάζουμε το script σε RayCastShootComplete και κάνουμε διπλό κλικ πάνω του για να το επεξεργαστούμε στο MonoDevelop του Unity. Το script που πρέπει να αναπτύξουμε και περιγράφει πλήρως την λειτουργικότητα των Raycast είναι το ακόλουθο:

RayCastShootComplete.cs

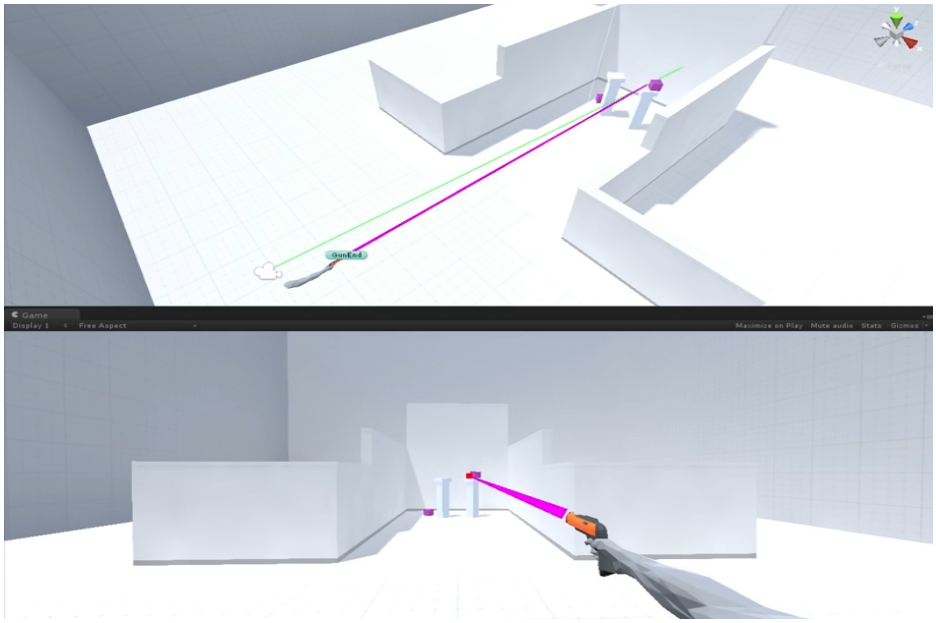
```
using UnityEngine;
using System.Collections;

public class RayCastShootComplete : MonoBehaviour {

	// gunDamege: Set the number of hitpoints that this gun will take away from shot objects with a health script
	public int gunDamage = 1;
	// fireRate: Number in seconds which controls how often the player can fire
	public float fireRate = 0.25f;
	// weaponRange: Distance in Unity units over which the player can fire
	public float weaponRange = 50f;
	// hitForce: Amount of force which will be added to objects with a rigidbody shot by the player
```

```
	public float hitForce = 100f;
	// gunEnd: Holds a reference to the gun end object, marking the muzzle location of the gun
	public Transform gunEnd;


	// fpsCam: Holds a reference to the first person camera
	private Camera fpsCam;
	// shotDuration: WaitForSeconds object used by our ShotEffect coroutine, determines time laser line will remain visible
	private WaitForSeconds shotDuration = new WaitForSeconds(0.07f);
	// gunAudio: Reference to the audio source which will play our shooting sound effect
	private AudioSource gunAudio;
	// laserLine: Reference to the LineRenderer component which will display our laserline
	private LineRenderer laserLine;
	// nextFire: Float to store the time the player will be allowed to fire again, after firing
	private float nextFire;



	void Start ()
	{
		// Get and store a reference to our LineRenderer component
		laserLine = GetComponent<LineRenderer>();

		// Get and store a reference to our AudioSource component
		gunAudio = GetComponent<AudioSource>();

		// Get and store a reference to our Camera by searching this GameObject and its parents
		fpsCam = GetComponentInParent<Camera>();

		//Use the PlayerPrefs to initialize the Score to zero
		PlayerPrefs.SetInt("Score", 0);
	}


	void Update ()
	{
		// Check if the player has pressed the fire button and if enough time has elapsed since they last fired
		if (Input.GetButtonDown("Fire1") && Time.time > nextFire)
		{
			// Update the time when our player can fire next
			nextFire = Time.time + fireRate;

			// Start our ShotEffect coroutine to turn our laser line on and off
	StartCoroutine (ShotEffect());

	// Create a vector at the center of our camera's viewport
```

```
        Vector3 rayOrigin = fpsCam.ViewportToWorldPoint (new Vector3(0.5f, 0.5f, 0.0f));

      // Declare a raycast hit to store information about what our raycast has hit
      RaycastHit hit;

                  // Set the start position for our visual effect for our laser to the position of gunEnd
                  laserLine.SetPosition (0, gunEnd.position);

                  // Check if our raycast has hit anything
                  if (Physics.Raycast (rayOrigin, fpsCam.transform.forward, out hit, weaponRange))
                  {
                        // Set the end position for our laser line
                        laserLine.SetPosition (1, hit.point);

                        // Get a reference to a health script attached to the collider we hit
                        ShootableBox health = hit.collider.GetComponent<ShootableBox>();

                        // If there was a health script attached
                        if (health != null)
                        {
                              // Call the damage function of that script, passing in our gunDamage variable
                              health.Damage (gunDamage);
                        }

                        // Check if the object we hit has a rigidbody attached
                        if (hit.rigidbody != null)
                        {
                              // Add force to the rigidbody we hit, in the direction from which it was hit
                              hit.rigidbody.AddForce (-hit.normal * hitForce);
                        }
                  }
                  else
                  {
                        // If we did not hit anything, set the end of the line to a position directly in front of the camera at the distance of weaponRange
                  laserLine.SetPosition (1, rayOrigin + (fpsCam.transform.forward * weaponRange));
                  }
            }
      }



      private IEnumerator ShotEffect()
      {
            // Play the shooting sound effect
            gunAudio.Play ();
```

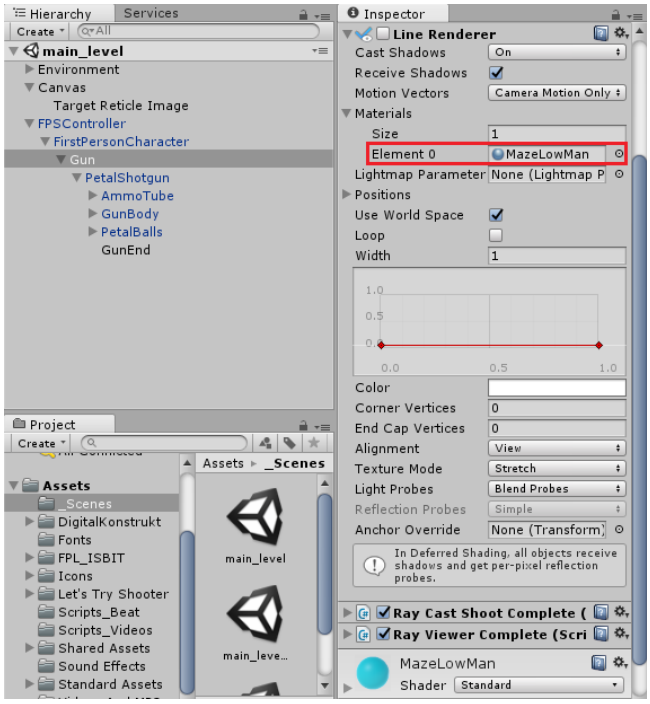
```
            // Turn on our line renderer
            laserLine.enabled = true;

            //Wait for .07 seconds
            yield return shotDuration;

            // Deactivate our line renderer after waiting
            laserLine.enabled = false;
        }
}
```

Το συγκεκριμένο script δεν θα μπορέσει να λειτουργήσει απευθείας καθώς όπως κάθε shooting script έχει αλληλεπίδραση με ένα στόχο καθώς και με την οπτικοποίηση της βολής/ακτίνας. Όπως βλέπουμε και από τα σχόλια του κώδικα, σε αυτό το script ορίζουμε τον χρόνο μεταξύ των ριπών, το σημείο έναρξης της ακτίνας (δηλαδή το GunEnd), ορισμός του πεδίου όπου και θα στοχεύει ο παίκτης (δηλαδή το κέντρο της κάμερας) και πληροφορίες για το αντικείμενο που έρχεται σε επαφή η ακτίνα (απενεργοποίηση του σε περίπτωση επαφής όπως θα δούμε και στη συνέχεια). Προστέθηκε επιπλέον και κάτι που εν τέλει δεν βρήκε πρακτική χρήση στο project αλλά διατηρήθηκε για πιθανή μελλοντική χρήση. Είναι το τμήμα κώδικα που ορίζει ότι, σε περίπτωση που το αντικείμενο που έρθει σε επαφή με την ακτίνα είναι τύπου Rigidbody (δηλαδή έχει μάζα και δέχεται την επίδραση κανόνων φυσικής) να δέχεται και το ανάλογο impact στο transform του κατά το χτύπημα. Τέλος, εάν κάποιος είναι αρκετά παρατηρητικός, θα δει την ιδιόμορφη χρήση της δομής PlayerPrefs για την αρχικοποίηση του Score System με την τιμή μηδέν. Επειδή αυτό το initialisation της τιμής δεν θα επηρέαζε την λειτουργικότητα του κώδικα του Raycasting, το προσθέτουμε ακόμα και σε αυτό το αρχικό στάδιο χωρίς προβλήματα. Όμως, καλό θα ήταν όταν θα υλοποιήσουμε το Scoring System στη συνέχεια να έχετε κατά νου ότι έχει ήδη γίνει η αρχικοποίηση της τιμής Score.
Ας συνεχίσουμε όμως με την υλοποίηση της φυσικής “ακτίνας” που είναι και η ουσία του Raycasting. Το Raycast μπορούμε να πούμε ότι στην πραγματικότητα δεν απαρτίζεται μόνο από μια ακτίνα αλλά από δύο, την νοητή γραμμή από το “μάτι” του παίχτη μέχρι τον στόχο και ορατή βολή από την κάνη του όπλου μέχρι τον στόχο. Το σημείο στο οποίο τέμνονται αυτές οι δύο ευθείες είναι και το σημείο που θα χτυπήσει η βολή μας. Αυτό είναι και το σημείο που πρέπει να φαίνεται στην οθόνη μας με έναν στόχο για να βοηθάει τον παίχτη να γνωρίζει που στοχεύει κάθε φορά.
Στην εικόνα που ακολουθεί έχουμε μια πολύ καλή αναπαράσταση αυτών των δύο ακτίνων που αναφέραμε. Η πράσινη ακτίνα, η οποία δεν είναι ορατή εν ώρα του παιχνιδιού αλλά χρησιμοποιείται μόνο για οπτικοποίηση της ιδέας σε αυτό το σημείο, είναι η κατεύθυνση στην οποία κοιτάζει και στοχεύει ο παίχτης. Η νοητή ευθεία από το μάτι του μέχρι τον στόχο. Αυτό δεν είναι όμως το Raycast, δηλαδή η βολή. Είναι το σημείο που θα χτυπήσουμε εάν κάνουμε μια βολή. Το σημείο αυτό είναι και ο κέντρο της κάμερας και ορίζεται με τον κόκκινο στόχο. Η ροζ γραμμή όμως είναι το ορατό λέιζερ που βλέπουμε μέσα στο παιχνίδι, δηλαδή η βολή, που βγαίνει από την κάνη του όπλου του παίχτη και χτυπάει όποιο αντικείμενο βρίσκεται στον στόχο.

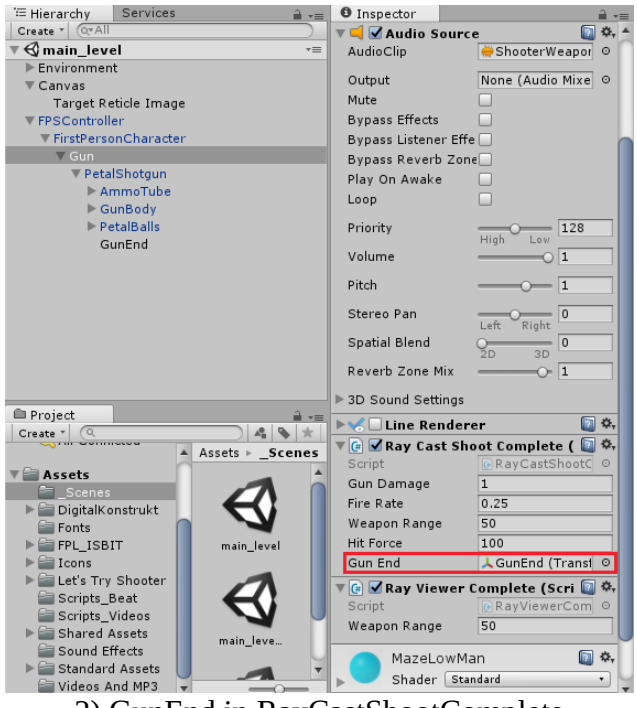


Εικόνα 2.6.iii: Θεωρητική απεικόνιση των δύο ακτίνων που απαρτίζουν το Raycast

Ας φτιάξουμε λοιπόν το ορατό λέιζερ για τις βολές μας. Για να το πετύχουμε αυτό πρέπει να εντάξουμε ένα component τύπου Line Renderer στο αντικείμενο Gun. Για να το πετύχουμε αυτό επιλέγουμε το αντικείμενο και στη συνέχεια πατάμε Add Component → Effects → Line Renderer. Στο Inspector θα δούμε πλέον το νέο πεδίο με το οποίο θα διαχειριζόμαστε την ακτίνα βολών. Το default χρώμα της ακτίνας είναι ροζ αλλά για αισθητικούς λόγους θα το αλλάξουμε. Για να το κάνουμε αυτό, παίρνουμε ένα Material της επιλογής μας και το κάνουμε drag and drop στο πεδίο Element 0. Επίσης, καλό θα ήταν να ορίσουμε το μέγεθος της ακτίνας σε 0.05 στο Start Width και στο End Width για καλύτερο αισθητικό αποτέλεσμα. Έχοντας κάνει και αυτό το βήμα, απενεργοποιούμε το Line Renderer για να μην είναι συνέχεια ορατό, μόνο όταν γίνεται μια βολή, και κάνουμε drag and drop του RayCastShootComplete.cs στο αντικείμενο Gun φροντίζοντας να ορίσουμε ότι η κάνη του όπλου είναι το GunEnd. Ενδεικτικά, ελέγξτε τις ακόλουθες εικόνες για να βεβαιωθείτε ότι έχουν γίνει όλες οι απαραίτητες ενέργειες.

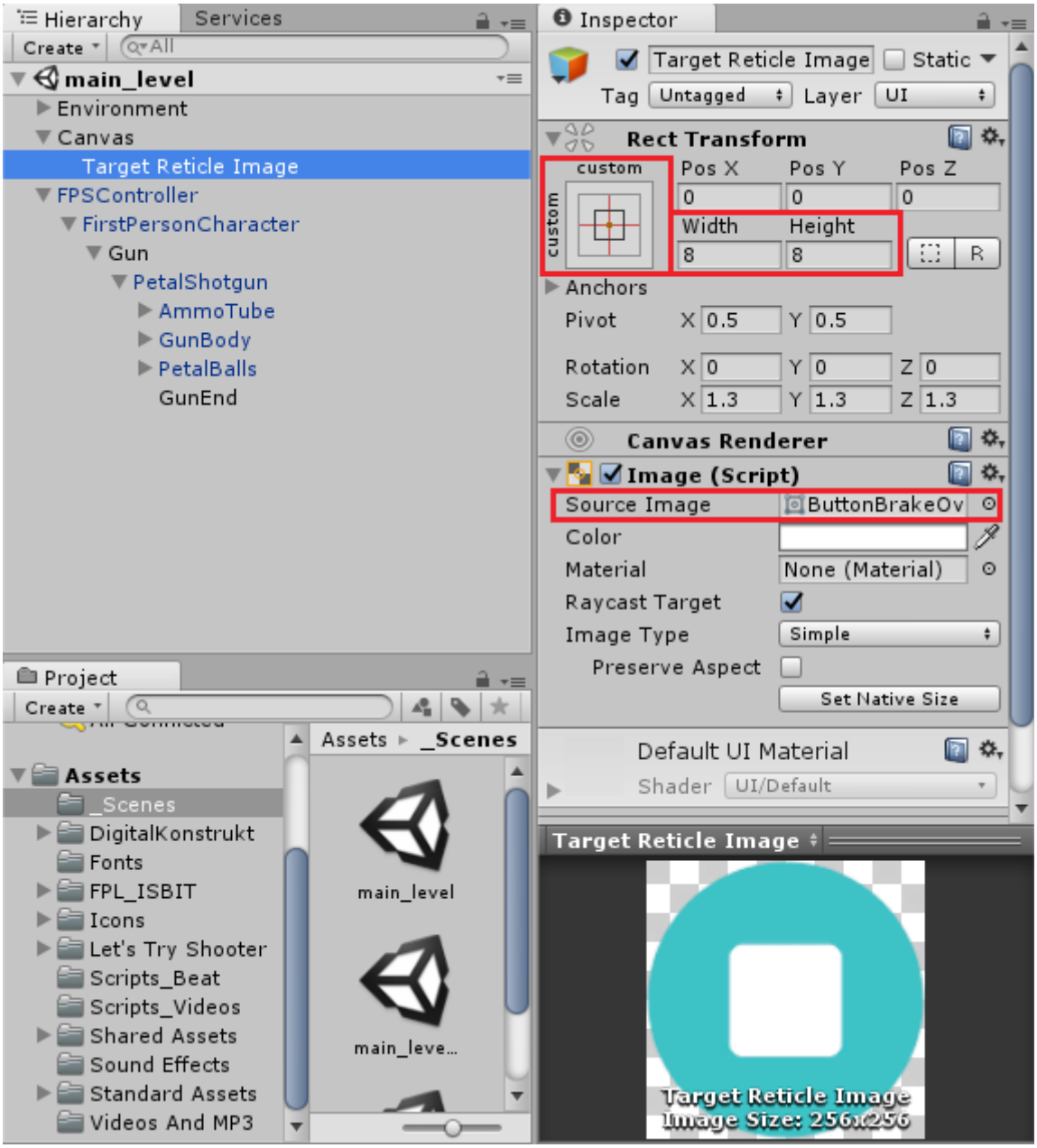


1) Line Renderer



2) GunEnd in RayCastShootComplete

Για να κλείσουμε το κομμάτι του Raycasting, μένει να φτιάξουμε τον στόχο για να βοηθήσουμε τον παίχτη να γνωρίζει που στοχεύει με το όπλο του κάθε φορά. Η παρούσα υλοποίηση είναι πολύ απλή και βασίζεται στην δυνατότητα των UI αντικείμενων να ακολουθούν την κίνηση της κάμερας. Εάν βάλουμε μια εικόνα τύπου UI στο κέντρο του οπτικού πεδίου της κάμερας θα έχουμε ακριβώς το σημείο που στοχεύουμε με το όπλο μας σε κάθε βολή ανεξαρτήτως το πού “κοιτάζουμε” κάθε φορά.
Επιλέγοντας GameObject → UI → Image θα δούμε ότι στην ιεραρχία θα δημιουργηθεί ένα αντικείμενο με την ονομασία Canvas που θα περιέχει το UI αντικείμενο Image. Το μετονομάζουμε σε Target Reticle Image και προχωράμε να το ρυθμίσουμε κατάλληλα μέσω του Inspector. Το βασικό είναι να ορίσουμε σωστά το Anchor του, δηλαδή τη θέση του βάση της κάμερας. Πρέπει να επιλέξουμε το απόλυτο κέντρο. Για να το κάνουμε πιο ορατό στον παίχτη μπορούμε να του δώσουμε ένα χρώμα ή ακόμα και μία εικόνα. Η πιο κλασσική επιλογή είναι ένα στόχαστρο. Εμείς επιλέξαμε μια εναλλακτική εκδοχή του τυπικού reticle που ταιριάζει με την μπλε ακτίνα που έχουμε στο όπλο μας και ορίζει μινιμαλιστικά τη θέση του στόχου στον χώρο.



Εικόνα 2.6.iv: Βασικές ρυθμίσεις του στόχου του παίχτη

Επίσης, μπορούμε να προσθέσουμε στο παιχνίδι μας το script RayViewerComplete εάν θέλουμε να δούμε και πρακτικά την ευθεία γραμμή που ορίζει το σημείο που στοχεύει κάθε φορά ο παίχτης (πράσινη γραμμή που αναφέραμε πριν) καθώς και τη μέγιστη απόσταση στο χώρο που μπορεί να καλύψει η βολή. Το ακόλουθο script δεν επηρεάζει την λειτουργικότητα του RayCastShootComplete απλά διατίθεται καθαρά για να έχουμε την πλήρη οπτική του Raycast καθώς σχεδιάζουμε το παιχνίδι. Για να το δοκιμάσουμε απλά χρειάζεται να το βάλουμε στο αντικείμενο Gun και είμαστε έτοιμοι.

RayViewerComplete.cs

```
using UnityEngine;
using System.Collections;

public class RayViewerComplete : MonoBehaviour {

      // weaponRange: Distance in Unity units over which the Debug.DrawRay
will be drawn
  public float weaponRange = 50f;
      // fpsCam: Holds a reference to the first person camera
  private Camera fpsCam;


      void Start ()
  {
     // Get and store a reference to our Camera by searching this GameObject
and its parents
    fpsCam = GetComponentInParent<Camera>();
      }


      void Update ()
  {
    // Create a vector at the center of our camera's viewport
       Vector3 lineOrigin = fpsCam.ViewportToWorldPoint(new Vector3(0.5f, 0.5f,
0.0f));

       // Draw a line in the Scene View  from the point lineOrigin in the
direction of fpsCam.transform.forward * weaponRange, using the color green
           Debug.DrawRay(lineOrigin, fpsCam.transform.forward * weaponRange,
Color.green);
      }
}
```

Τέλος, μένει να μιλήσουμε για τον κώδικα που θα διαχειρίζεται τους στόχους όταν δέχονται μια βολή από το Raycast. Μπορούμε δοκιμαστικά να ενεργοποιήσουμε τα αντικείμενα Target που βρίσκονται στο Environment και να εντάξουμε ένα βασικό script που θα τα απενεργοποιεί εάν δέχονται μια βολή. Σε περίπτωση που θέλετε να κάνετε χρήση και του τμήματος του κώδικα του RayCastShootComplete για την άσκηση physical force με κάθε hit στους στόχους απλά φροντίστε να είναι Rigidbody (δηλαδή να έχουν μάζα) και το currentHealth να είναι πάνω από 1. Απλά είναι σημαντικό να αναφέρουμε ότι στα κεφάλαια που ακολουθούν θα υπάρχει αναλυτική προσέγγιση στον τομέα σχεδίασης στόχων οπότε δεν χρειάζεται να το αναλύσουμε σε μεγαλύτερο βαθμό στο σημείο που βρισκόμαστε. Ακολουθεί η πρώιμη εκδοχή του script ShootableBox.cs που θα είναι το βασικό script όλων των στόχων που θα έχουμε στο παιχνίδι.

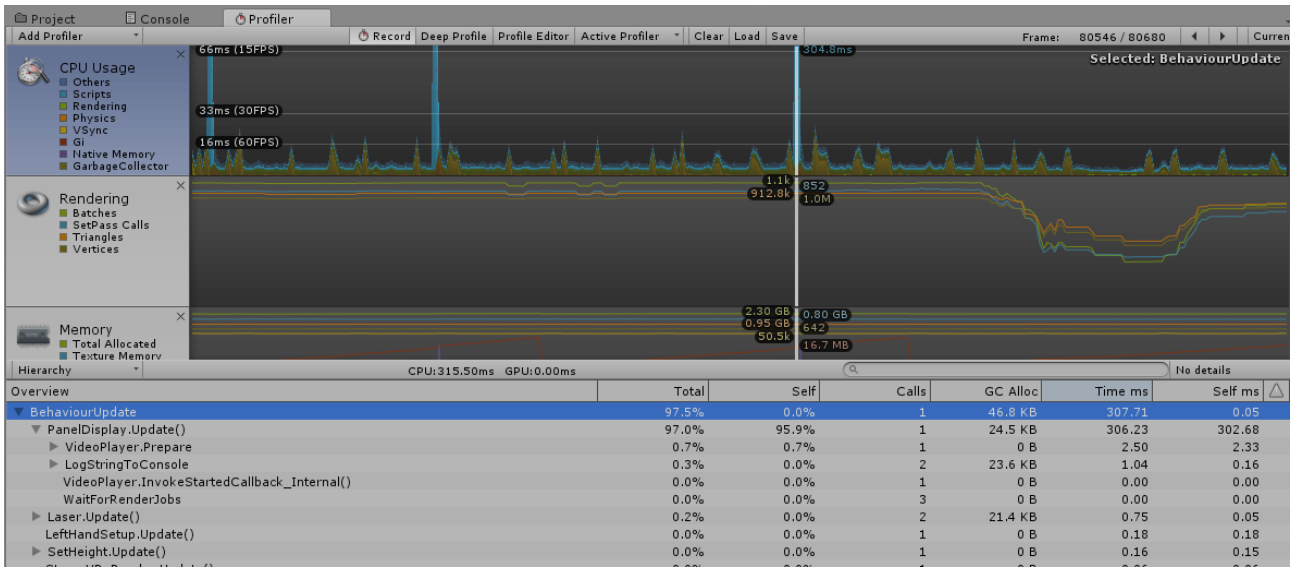
ShootableBox.cs

```
using UnityEngine;
using System.Collections;

public class ShootableBox : MonoBehaviour {

	//The box's current health point total
	public int currentHealth = 1;

	public void Damage(int damageAmount)
	{
		//subtract damage amount when Damage function is called
		currentHealth -= damageAmount;

		//Check if health has fallen below zero
		if (currentHealth <= 0)
		{
			//if health has fallen below zero, deactivate it
			gameObject.SetActive (false);
		}
	}
}
```

Σημείωση: Το αρχείο ShootableBox.cs θα έχει αρκετές προσθήκες κατά την υλοποίηση του Scoring System οπότε θα δούμε την Updated εκδοχή του στο κεφάλαιο Ολοκληρώνοντας το Παιχνίδι με Spawns, Point System και Pause Feature.

2.7 Σχεδιάζοντας τα Videowalls

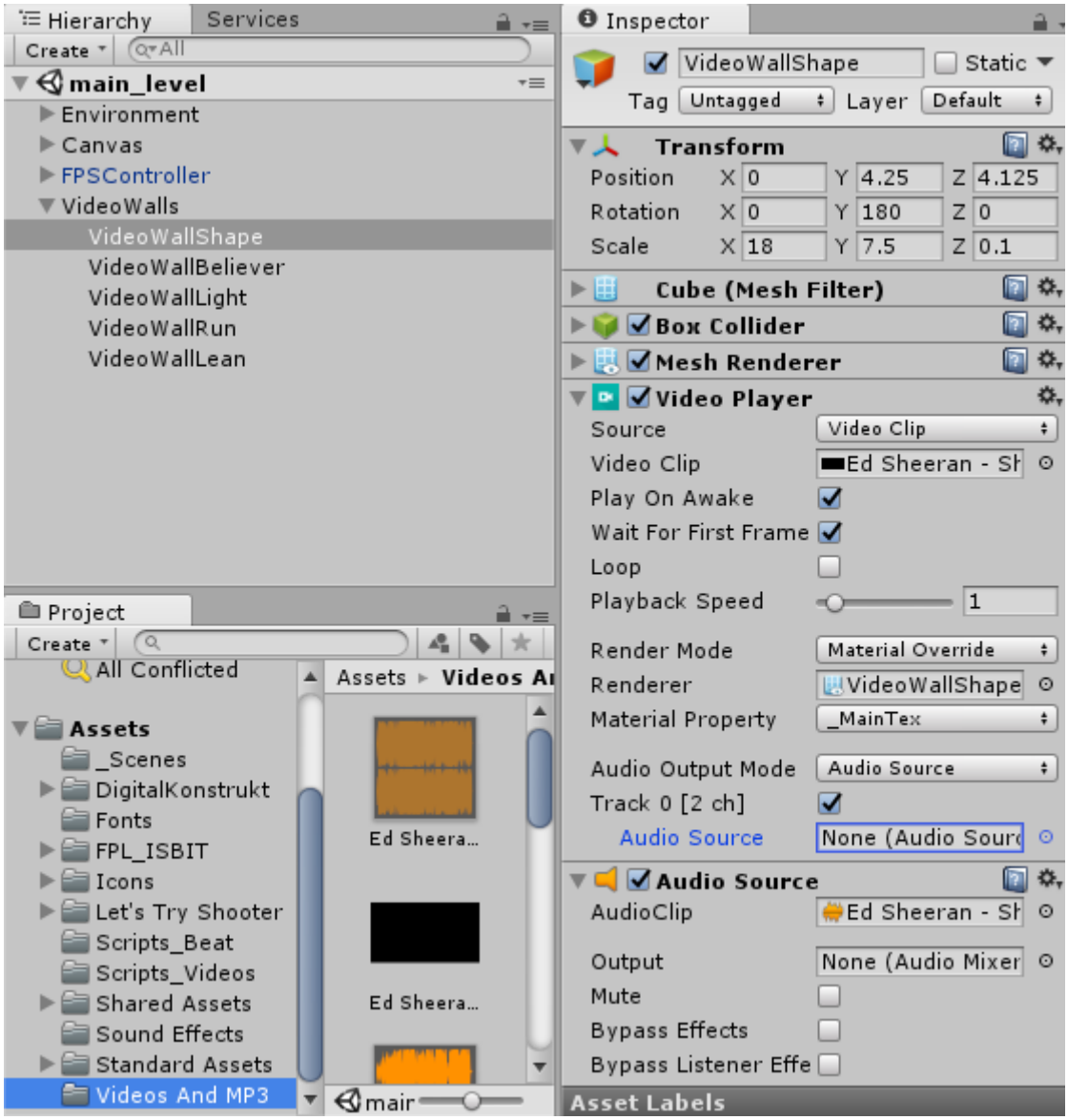
Αφού έχουμε ολοκληρώσει και το στάδιο του Raycasting, ας προχωρήσουμε σε μια πιο απλή διαδικασία, την υλοποίηση των Videowall που θα προβάλλουν μπροστά από τον παίχτη μας το video clip της επιλογής του και θα παίζουν τη μουσική.
Στην πραγματικότητα, η υλοποίηση πολλαπλών αντικειμένων Videowall (ένα για κάθε μουσικό κομμάτι που μπορεί να επιλέξει ο παίκτης) δεν είναι η optimal λύση για το project μας αλλά δυστυχώς έπρεπε να καταλήξουμε σε αυτή λόγω τεχνικού προβλήματος της πλατφόρμας που θα αναλύσουμε διεξοδικά.
Ο αρχικός κώδικας που υλοποιήθηκε χρησιμοποιούσε ένα και μόνο Videowall στο οποίο αλλάζαμε τα τραγούδια βάση την επιλογή του παίχτη στο αρχικό μενού. Αυτή η λύση ήταν λειτουργική όσο η εφαρμογή ήταν απλά desktop και λειτουργούσε με ποντίκι/πληκτρολόγιο/οθόνη. Όμως, το παιχνίδι αυτό κατασκευάστηκε έχοντας κατά νου την συμβατότητα με VR (Oculus Rift αλλά θα δείξουμε και για HTC Vive). Όταν το παιχνίδι αυτό δοκιμάστηκε για πρώτη φορά στο Oculus Rift, υπήρχε μεγάλη καθυστέρηση από το Videowall αλλά παρατηρήσαμε κιόλας ξαφνική, έντονη δραστηριότητα του ανεμιστήρα του Η/Υ. Τα fps (frames per second) έπεσαν δραματικά και είχαμε διακοπτόμενες εικόνες στο feed της μάσκας. Ελέγχθηκε αμέσως η δραστηριότητα στη CPU όπου και υπήρχε χρήση 100% από το παιχνίδι. Παρόλο και ας οι δυνατότητες του Η/Υ που υλοποιήθηκε αυτό το παιχνίδι είναι περιορισμένες, δεν θα έπρεπε να υπάρχει αυτή η συμπεριφορά στο VR εφόσον ο κώδικας ήταν λειτουργικός στην οθόνη. Το πρόβλημα ελέγχθηκε στο Forum των Developers της Unity όπου και άλλοι χρήστες δήλωναν ότι το πρόβλημα υπάρχει μέχρι και την έκδοση 2017.1.0p4 χωρίς λύση από το Support της πλατφόρμας. Επίσης, οι υπόλοιποι developers που αντιμετώπιζαν αντίστοιχα ζητήματα, ανέφεραν αδυναμία εύρεσης βέλτιστης λύσης και για επίτευξη seamless αποτελέσματος στην προβολή των video ότι έπρεπε να γίνει χρήση ενός Videowall ανά μουσική επιλογή. Βάση αναφορών των χρηστών της πλατφόρμας, αυτό συμβαίνει λόγω συλλογής garbage κατά την αλλαγή ή τερματισμό του video στο Videowall το οποίο κανονικά δεν θα έπρεπε να συμβαίνει διότι είναι πληροφορία που θα έπρεπε να καθαρίζεται από την ίδια την πλατφόρμα σε κάθε load.



Εικόνα 2.7.i: Η απότομη αύξηση στο 100% της χρήσης της CPU από το παιχνίδι σε VR Mode

Παρόλα αυτά, ο αρχικός απορριφθέν κώδικας θα προστεθεί στο τέλος του κεφαλαίου ως δείγμα της βέλτιστης εκδοχής του παιχνιδιού που θα γινόταν χρήση του μόνο εφόσον το πρόβλημα της Unity είχε διορθωθεί μέχρι την παρουσίαση αυτού του παιχνιδιού.
Έχοντας αναλύσει πλήρως τα προβλήματα της πλατφόρμας, ας προχωρήσουμε στην ανάλυση της τρέχουσας λειτουργικής σχεδίασης των Videowalls.
Αρχικά, εφόσον θα έχουμε πλέον να διαχειριστούμε πολλαπλά αντικείμενα, θα ήταν ιδανικό να δημιουργήσουμε ένα γενικό αντικείμενο πάνω στο οποίο θα υπάγονται τα λοιπά Videowalls. Για τον λόγο

αυτό θα πατήσουμε στον κενό χώρο του Hierarchy και θα κάνουμε Create → Empty και θα το μετονομάσουμε σε VideoWalls. Στον Inspector του αντικειμένου, στην επιλογή Transform, επιλέγουμε το γρανάζι και πατάμε Reset για να φέρουμε το αντικείμενο στη θέση (0, 0, 0). Αυτό είναι αναγκαίο να γίνεται σε κάθε ανώτατο αντικείμενο ιεραρχίας δημιουργούμε για να μην αλλοιώνουμε το Transform των αντικειμένων κάτω από αυτό και να έχουμε ξεκάθαρη εικόνα της θέσης τους στον χώρο.
Εν συνεχεία, θα κάνουμε Create → 3D Object → Cube και θα δημιουργήσουμε ένα κύβο τον οποίο θα μετασχηματίσουμε για να μοιάζει με μια μεγάλη οθόνη προβολής των video μας. Μετονομάζουμε τον κύβο και ορίζουμε την θέση του και τις διαστάσεις όπως φαίνεται και στην εικόνα που ακολουθεί. Σέρνοντας το αντικείμενο και βάζοντας το κάτω από το VideoWalls, πλέον έχουμε φτιάξει το πρώτο μας Videowall.



Εικόνα 2.7.ii: Ορθή εικόνα της ιεραρχίας των Videowall και του Inspector για κάθε ένα από αυτά

Σε αυτό το σημείο πρέπει να αναφέρουμε ότι ένα παραλληλόγραμμο σχήμα είναι όντως η βέλτιστη επιλογή για τη μορφή του αντικειμένου. Κατά τη διάρκεια μελέτης των VideoPlayers, καταλήξαμε στο συμπέρασμα ότι το αντικείμενο που θα έχει ως υφή ένα βίντεο πρέπει η δομή του να είναι ακριβώς ίδια με αυτή του βίντεο. Επί παραδείγματι, εάν θέλουμε να προβάλουμε ένα βίντεο 360 μοιρών, μπορούμε να το κάνουμε μόνο με κάμερα μέσα σε μία σφαίρα. Δεν θα πετύχει σε κάτι άλλο. Χαρακτηριστικά, σε αυτό

το project, δοκιμάστηκε ένα καμπυλωτό μοντέλο που έμοιαζε με curved οθόνη νέας γενειάς αλλά δυστυχώς με μεγάλη αλλοίωση κατά την προβολή.
Συνεχίζοντας όμως, για να δώσουμε ήχο και εικόνα στο Videowall πρέπει να κάνουμε Import στα Asset μας τα μουσικά κομμάτια και βίντεο της επιλογής μας. Στα Assets, κάνουμε Create → Folder, τον ονομάζουμε Videos and MP3 και κάνουμε Import New Asset. Σημαντικό είναι να αναφερθεί ότι το Unity υποστηρίζει συγκεκριμένους τύπους αρχείων multimedia, όπως ορίζονται και στο documentation, και εμείς κάναμε χρήση .mp4 για τα video σε ανάλυση 1080p για HD feed στο VR καθώς και .mp3 για τη μουσική. Κάνοντας drag and drop των εκάστοτε video και μουσικού clip πάνω στον κύβο/Videowall, δημιουργείται στον Inspector ένα πεδίο με το όνομα Video Player και ένα με το όνομα Audio Source.
Επειδή βρισκόμαστε ακόμα στα αρχικά στάδια της υλοποίησης της εφαρμογής, προτείνεται τα υπόλοιπα τέσσερα αντικείμενα να είναι dummy κλώνοι του πρώτου καθώς θα γίνουν αρκετές αλλαγές επί αυτών στα κεφάλαια που ακολουθούν. Οπότε για αυτό το κεφάλαιο, αρκεί απλά να γίνει Duplicate και μετονομασία των λοιπών αντικείμενων προβολής video.
Ας προχωρήσουμε λοιπόν στη δημιουργία του script που θα διαβάζει την επιλογή που έκανε ο παίχτης στο αρχικό μενού και να ενεργοποιεί το αντίστοιχο Videowall. Στον φάκελο που είχαμε βάλει και το script για την διαχείριση των επιλογών στο μενού μας, θα κάνουμε Create → C# Script και θα το ονομάσουμε set_videowall.cs.

set_videowall.cs

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

public class set_videowall : MonoBehaviour {

	//whichWall: the variable that keeps which song was selected via the
PlayerPrefs getter
	public int whichWall;
	//videowallShape: the VideoWall of the video/song Shape of You
	public GameObject videowallShape;
	//videowallBeliever: the VideoWall of the video/song Believer
	public GameObject videowallBeliever;
	//videowallLight: the VideoWall of the video/song Light it Up
	public GameObject videowallLight;
	//videowallRun: the VideoWall of the video/song Run Up
	public GameObject videowallRun;
	//videowallLean: the VideoWall of the video/song Lean On
	public GameObject videowallLean;

	// use this for initialization
	void Start () {

		//before we start we need to make sure no VideoWall is active or
the music will start playing and cause issues to the Beat Detector
		videowallShape.SetActive(false);
		videowallBeliever.SetActive(false);
		videowallLight.SetActive(false);
		videowallRun.SetActive(false);
		videowallLean.SetActive(false);

		//in the variable whichWall, we keep the number of the song
dropdown selection and proceed to activate the proper VideoWall
		if(PlayerPrefs.HasKey("SelectSong")){
			whichWall = PlayerPrefs.GetInt("SelectSong");
			//0 is the number of the first value in the dropdown menu
```

```
and is the song Shape of You
                if (whichWall == 0){
                      //activate the GameObject videowallShape
                      videowallShape.SetActive(true);
                }
                //1 is for the second selection aka Believer
                else if (whichWall == 1){
                      //activate the GameObject videowallBeliever
                      videowallBeliever.SetActive(true);
                }
                //2 is for the third choice aka Light it Up
                else if (whichWall == 2){
                      //activate the GameObject videowallLight
                      videowallLight.SetActive(true);
                }
                //3 is for the fourth value aka Run Up
                else if (whichWall == 3){
                      //activate the GameObject videowallRun
                      videowallRun.SetActive(true);
                }
                //4 is for the final option aka Lean On
                else if (whichWall == 4){
                      //activate the GameObject videowallLean
                      videowallLean.SetActive(true);
                }
          }
          else{
                //Use the Debug Log to see if there is no song selected/ not
needed because the dropdown menus give the default value of 0 when the player
just presses play
                Debug.Log("Non Existing Song Selection");
          }
     }

     // Update is called once per frame
     void Update () {
          //do nothing
     }
}
```

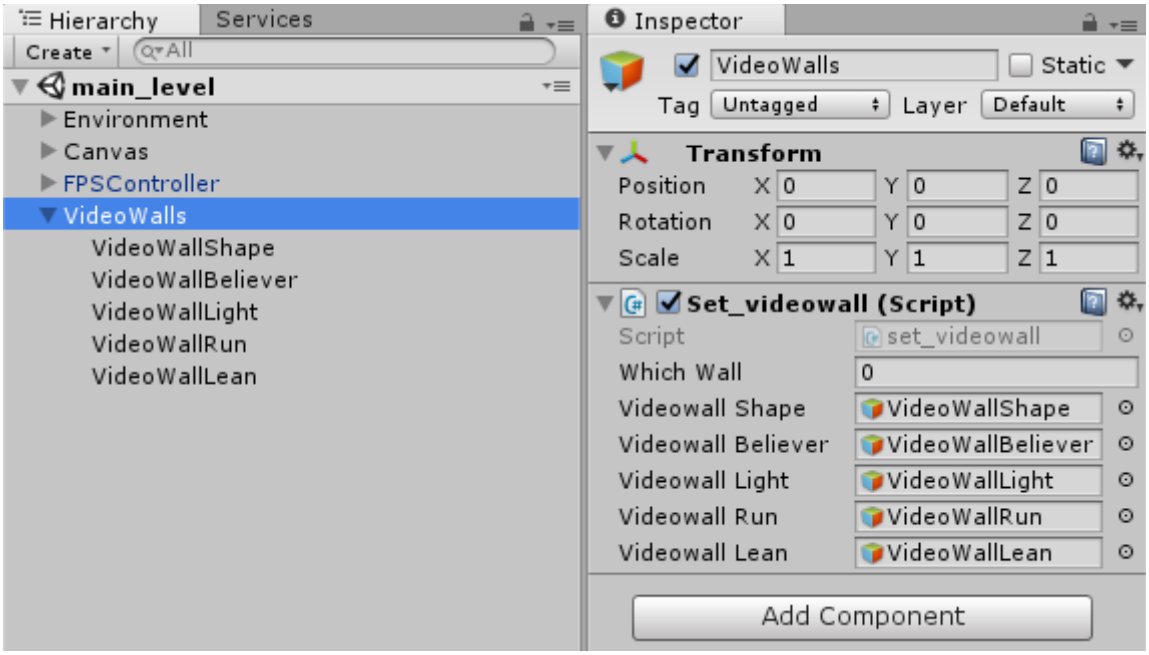
Επίσης, πρέπει να δημιουργήσουμε ένα απλό script το οποίο θα αναλαμβάνει να ενεργοποιεί νέο το Texture με το video που θα επικαλύπτει το default του αντικειμένου πάνω στο οποίο προβάλλεται. Για να το επιτύχουμε αυτό πρέπει να κάνουμε, στον ίδιο φάκελο στο Unity, Create → C# Script και να το ονομάσουμε start_video.cs. Δεν είναι απαραίτητο να είναι προσαρτημένο σε κάποιο αντικείμενο αλλά πρέπει να περιέχει τον ακόλουθο κώδικα:

start_video.cs

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

public class start_video : MonoBehaviour {

     // Use this for initialization
     void Start () {
          // this line of code will make the Movie Texture begin playing
```

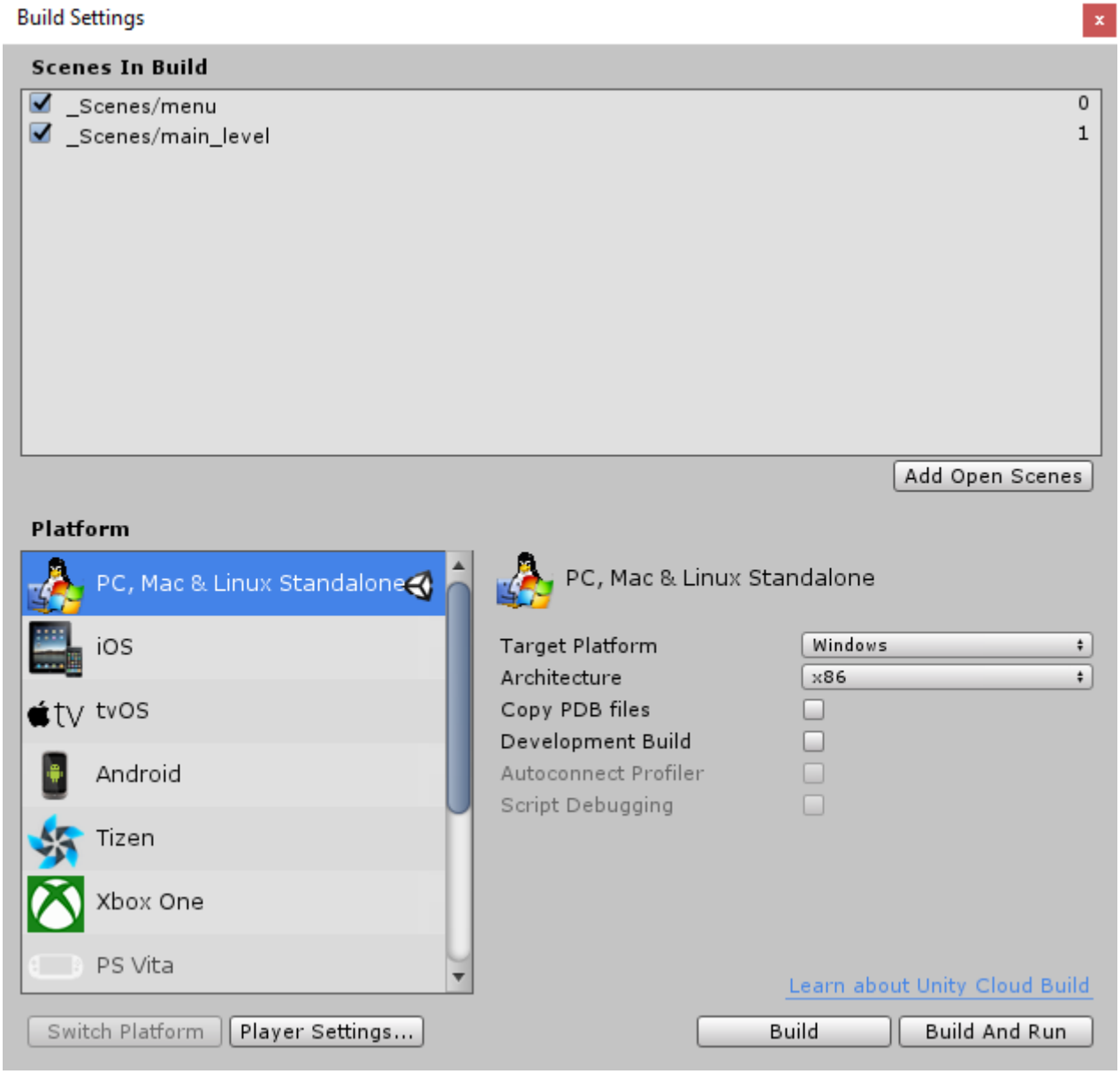
```
((MovieTexture)GetComponent<Renderer>().material.mainTexture).Play();
	}



	// Update is called once per frame
	void Update () {
		//do nothing
	}
}
```

Ολοκληρώνοντας και τα script μας, μπορούμε πλέον να τα χρησιμοποιήσουμε στην σκηνή μας. Κάνοντας drag and drop το script set_videowall στο ανώτατο αντικείμενο της ιεραρχίας των Videowall μπορούμε να τα ελέγχουμε πλήρως βάση των επιλογών το κεντρικό μενού. Το μόνο που μένει είναι να αντιστοιχίσουμε τα αντικείμενα που δηλώνονται στο script με τα φυσικά αντικείμενα της σκηνής και είμαστε έτοιμοι. Η εικόνα που πρέπει να έχουμε στην πλατφόρμα μας έχοντας κάνει όλα τα βήματα πρέπει να είναι η εξής:



Εικόνα 2.7.iii: Ένταξη του script set_videowall στο αντικείμενο VideoWalls του επιπέδου.

Για να μπορούμε να δοκιμάζουμε πλήρως το παιχνίδι μας, είναι σημαντικό σε αυτό το στάδιο να συνδέσουμε τις σκηνές του παιχνιδιού. Πατώντας File → Build Settings… μπορούμε να βάλουμε με ποια σειρά πρέπει να εμφανίζονται οι σκηνές μας. Κάνοντας κλικ στο Add Open Scenes και επιλέγοντας από τον φάκελο _Scenes πρώτα την σκηνή menu.unity και στη συνέχεια το main_level.unity μπορούμε να δούμε πρακτικά πως λειτουργεί το μενού επιλογών μας. Μονάχα εάν επιλέξουμε το Hard επίπεδο δυσκολίας δεν θα ανοίξει κάποια σκηνή γιατί δεν την έχουμε υλοποιήσει ακόμα αλλά θα το κάνουμε στα κεφάλαια που ακολουθούν.



Εικόνα 2.7.iv: Ρύθμιση της σειράς λειτουργίας των σκηνών του παιχνιδιού

Ακολουθεί η παράθεση του απορριφθέντος script set_videowall.cs λόγω δυσλειτουργίας της Unity:

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

public class set_videowall : MonoBehaviour {

      //whichWall: the variable that keeps which song was selected via the
PlayerPrefs getter
      public int whichWall;
      //videowall: the VideoWall of the video/song that the player has chosen
      public GameObject videowall;
      //audiosource: the component that is responsible for playing the music
for the VideoWall
```

```
	public AudioSource audioSource;
	//videoPlayer: the component that is responsible for playing the video
for the VideoWall
	public VideoPlayer videoPlayer;
	//clip: the music clip of the VideoWall
	public AudioClip clip;
	//video: the video clip of the VideoWall
	public VideoClip video;

	// Use this for initialization
	void Start () {

		//in this version of the code we do not need to deactivate the
VideoWall because the only thing that changes is the video/song played
		//we connect the variables with the GameObject's audioSource and
videoPlayer
		audioSource = GetComponent<AudioSource>();
		videoPlayer = GetComponent<VideoPlayer> ();
		//in the variable whichWall, we keep the number of the song
dropdown selection and proceed to activate the proper VideoWall
		if(PlayerPrefs.HasKey("SelectSong")){
			whichWall = PlayerPrefs.GetInt("SelectSong");
			//0 is the number of the first value in the dropdown menu
and is the song Shape of You
			if (whichWall == 0){
				//show on Debug the number of the song
				Debug.Log(whichWall);
				//load from the Assets the needed song and video for
the VideoWall
				clip = Resources.Load("Resources/songvideo/shapesong")
as AudioClip;
				video =
Resources.Load("Resources/songvideo/shapevideo") as VideoClip;
			}
			else if (whichWall == 1){
				Debug.Log(whichWall);
				clip =
Resources.Load("Resources/songvideo/believersong") as AudioClip;
				video =
Resources.Load("Resources/songvideo/believervideo") as VideoClip;

			}
			else if (whichWall == 2){
				Debug.Log(whichWall);
				clip = Resources.Load("Resources/songvideo/lightsong")
as AudioClip;
				video =
Resources.Load("Resources/songvideo/lightvideo") as VideoClip;

			}
			else if (whichWall == 3){
				Debug.Log(whichWall);
				clip = Resources.Load("Resources/songvideo/runsong") as
AudioClip;
				video = Resources.Load("Resources/songvideo/runvideo")
as VideoClip;
			}
			else if (whichWall == 4){
```

```
                        Debug.Log(whichWall);
                        clip = Resources.Load("Resources/songvideo/leansong")
as AudioClip;
                        video = Resources.Load("Resources/songvideo/leanvideo")
as VideoClip;
                  }
            }
                  //the VideoPlayer is set with the selected video
                  videoPlayer.clip = video;
                  //and it starts to play
                  videoPlayer.Play();
                  //with the music clip starting simultaneously
                  audioSource.PlayOneShot(clip);
      }

      // Update is called once per frame
      void Update () {
            //do nothing
      }
}
```

2.8 Ανιχνεύοντας το Ρυθμό της Μουσικής σε Πραγματικό Χρόνο

Σε αυτό το κεφάλαιο θα ασχοληθούμε με μια καινοτόμα ιδέα για τον κόσμο των παιχνιδιών μουσικής. Σε όλες τις κυκλοφορίες των παιχνιδιών ρυθμού μέχρι σήμερα, τα επίπεδα δυσκολίας και οι πίστες ορίζονται από το tempo του κάθε κομματιού, αργά μουσικά κομμάτια για τις εύκολες πίστες και γρήγορα χορευτικά κομμάτια για τις δύσκολες. Τα κομμάτια προ-επεξεργάζονταν πριν την υλοποίηση κάθε επιπέδου για να μετράται με ακρίβεια ο ρυθμός και το επίπεδο υλοποιούταν γύρω από αυτόν. Κανένα παιχνίδι όμως δεν έχει επιχειρήσει να προσαρμόζει σε πραγματικό χρόνο το επίπεδο στον ρυθμό της μουσικής ανιχνεύοντας τα peak της καθώς παίζει από την πηγή ήχου στον χώρο. Αυτό θα πραγματοποιήσουμε με επιτυχία σε αυτό το στάδιο της εργασίας.
Η ιδέα πίσω από αυτή την υλοποίηση ήταν πολύ απλή αλλά δυστυχώς με μεγάλο παράγοντα λάθους. Σκοπός ήταν να λαμβάνουμε τις τιμές της συχνότητας του σήματος σε πραγματικό χρόνο από τη πηγή ήχου και να κρατάμε με δειγματοληψία τις τιμές του φάσματος. Η μέγιστη τιμή κάθε δείγματος για το χρονικό διάστημα που ελέγχουμε το φάσμα, είναι και ο ρυθμός. Όμως συχνά αυτό σαν μέτρηση δεν αρκεί. Πρέπει να εξετάζουμε εάν υπήρχαν και άλλα αντίστοιχα peak εντός του δείγματος για να δούμε εάν πρόκειται απλά για υψηλούς συχνοτικά ήχους ή για τον ρυθμό. Σε περίπτωση που υπάρχουν όντως πάνω από δύο peak και πάνω στο ίδιο χρονικό διάστημα, δεν είναι πραγματικά ο ρυθμός του κομματιού. Η ιδέα αυτή ήταν ικανή να μας δώσει ένα ικανοποιητικό αποτέλεσμα που λειτουργεί με πάρα πολύ μικρό ποσοστό λάθους ακόμα και σε dance κομμάτια. Ενδεικτικά σε αυτή την εργασία, έγινε χρήση κυρίως τέτοιων τραγουδιών για να φανεί πλήρως η δυναμική του κώδικα.
Το κλειδί πίσω από την υλοποίηση της ιδέας, όπως καταλαβαίνουμε και από την προηγούμενη παράγραφο, δεν είναι τίποτε άλλο από τον μετασχηματισμό Fourier και ακόμα πιο συγκεκριμένα από Discrete Fourier Transform (DFT) καθώς έχουμε για δειγματοληψία μουσικά κομμάτια, δηλαδή ηχητικά σήματα με πεπερασμένη ακολουθία συχνοτήτων. Εάν υλοποιούσαμε αυτόν τον κώδικα σε Matlab θα κάναμε χρήση της Fast Fourier Transform (FFT) αλλά δυστυχώς δεν είναι διαθέσιμη στη υποδομή του Unity οπότε θα προσπαθήσουμε να αναπαράγουμε την λογική αυτής στον ακόλουθο κώδικα.
Έχοντας αναλύσει τα παραπάνω, πάμε να εντάξουμε το script στο project μας. Στα Assets, θα κάνουμε Create → Folder για να φτιάξουμε ένα νέο βοηθητικό φάκελο με τον όνομα Scripts_Beat. Σε αυτόν θα κάνουμε Create → C# Script και θα το ονομάσουμε AudioProcessor.cs. Στο script πρέπει να συντάξουμε τον ακόλουθο κώδικα.

AudioProcessor.cs

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

[RequireComponent (typeof(AudioSource))]
public class AudioProcessor : MonoBehaviour
{
	//previousStepSpectrum[]: the spectrum of the previous step
	float[] previousStepSpectrum;

	//fftSize: buffer size
	public int fftSize = 1024;
	//fftRate: sampling frequency
	private int fftRate = 44100;

	//The Autocorrelation structure
	//maximumFrameLag: largest lag to track (counted in frames)
	int maximumFrameLag = 100;
	//decay: smoothing constant for running average
	float decay = 0.997f;
	Autocorrelator autocor;
```

```
	public AudioSource audioSource;

	//list of utility variables
	private long lastTime;
	private long nowTime;
	private long difference;
	private long entries;
	private long sum;

	//We will require arrays to store our buffer info:
	private int maxcolumn = 120;
	float[] spectrum;
	float[] averages;
	float[] acVals;
	float[] onsets;
	float[] scorefun;
	float[] dobeat;
	//currentTimeMark: time index for circular buffer within above
	int currentTimeMark = 0;

	//The variables used to log-frequency averaging controls:
	//numBands: number of bands
	private int numBands = 12;
	//globalThreshold: sensitivity
	public float globalThreshold = 0.1f;
	//doubleBeatSuppressor: counts since the previous beat detection to
suppress double-beats (may occur on spectrum peaks)
	private int doubleBeatSuppressor = 0;
	private float framePeriod;
	int beatSignalDelayLenght = 16;
	int[] beatSignalDelay;

	//alph: trade-off constant between tempo deviation penalty and onset
strength
	private float alph;

	[Header ("Events")]
	public OnBeatEventHandler onBeat;
	public OnSpectrumEventHandler onSpectrum;


	//------------------------------------------------------------------
--------

	private void initializeArrays ()
	{
		beatSignalDelay = new int[beatSignalDelayLenght];
		onsets = new float[maxcolumn];
		scorefun = new float[maxcolumn];
		dobeat = new float[maxcolumn];
		spectrum = new float[fftSize];
		averages = new float[12];
		acVals = new float[maximumFrameLag];
		alph = 100 * globalThreshold;
	}

	private long getCurrentMilliseconds ()
```

```
	{
		long milliseconds = System.DateTime.Now.Ticks /
System.TimeSpan.TicksPerMillisecond;
		return milliseconds;
	}


	// Use this for initialization
	void Start ()
	{
		initializeArrays ();

		audioSource = GetComponent<AudioSource> ();
		fftRate = audioSource.clip.frequency;

		framePeriod = (float)fftSize / (float)fftRate;

		//initialize record of previous spectrum
		previousStepSpectrum = new float[numBands];
		for (int i = 0; i < numBands; ++i)
			previousStepSpectrum [i] = 100.0f;

		autocor = new Autocorrelator (maximumFrameLag, decay, framePeriod,
getBandWidth ());

		lastTime = getCurrentMilliseconds ();
	}

	public void tappingTempo ()
	{
		nowTime = getCurrentMilliseconds ();
		difference = nowTime - lastTime;
		lastTime = nowTime;
		sum = sum + difference;
		entries++;

		int average = (int)(sum / entries);

		Debug.Log ("average = " + average);
	}

	double[] toDoubleArray (float[] arr)
	{
		if (arr == null)
			return null;
		int n = arr.Length;
		double[] ret = new double[n];
		for (int i = 0; i < n; i++) {
			ret [i] = (float)arr [i];
		}
		return ret;
	}



	//Update is called once per frame
	void Update ()
```

```
	{
		if (audioSource.isPlaying) {
			audioSource.GetSpectrumData (spectrum, 0,
FFTWindow.BlackmanHarris);
			calculateAverages (spectrum);
			onSpectrum.Invoke (averages);

			//we need to calculate the value of the onset function in
current frame
			float onset = 0;
			for (int i = 0; i < numBands; i++) {
				//specVal: dB value of this band
				float specVal = (float)System.Math.Max (-100.0f, 20.0f
* (float)System.Math.Log10 (averages [i]) + 160);
				specVal *= 0.025f;
				//dbIncrementrement: dB increment since last frame
				float dbIncrement = specVal - previousStepSpectrum [i];

				//record this frame to use next time
				previousStepSpectrum [i] = specVal;
				//and keep the onset which is the sum of dB increments
				onset += dbIncrement;
			}

			onsets [currentTimeMark] = onset;

			//we should make an update on the autocorrelator and find
peak lag aka the current tempo
			autocor.newValue (onset);
			//record largest value in (weighted) autocorrelation as it
will be the tempo
			float aMax = 0.0f;
			int tempopd = 0;
			//float[] acVals = new float[maximumFrameLag];
			for (int i = 0; i < maximumFrameLag; ++i) {
				float acVal = (float)System.Math.Sqrt
(autocor.autocorrelate (i));
				if (acVal > aMax) {
					aMax = acVal;
					tempopd = i;
				}
				//store in array backwards, so it displays right-to-
left, in line with traces
				acVals [maximumFrameLag - 1 - i] = acVal;
			}

			//we must now calculate DP-ish function to update the best-
score function
			float smax = -999999;
			int smaxix = 0;
			//the weight can be varied dynamically with the mouse
			alph = 100 * globalThreshold;
			//we should consider all possible preceding beat times from
0.5 to 2.0 x current tempo period
			for (int i = tempopd / 2; i < System.Math.Min (maxcolumn, 2
* tempopd); ++i) {
				//objective function - this beat's cost + score to last
beat + transition penalty
```

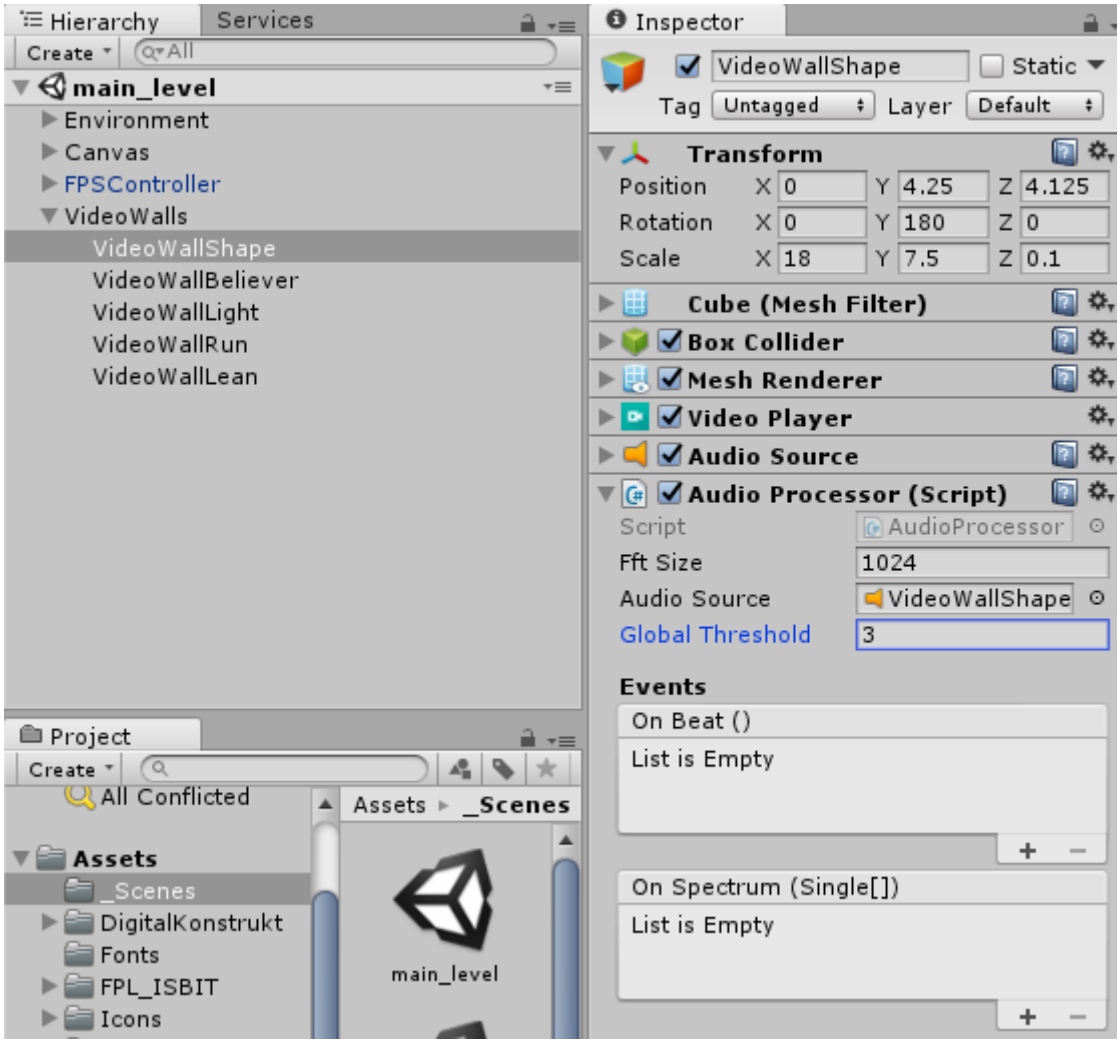
```
                        float score = onset + scorefun [(currentTimeMark - i +
maxcolumn) % maxcolumn] - alph * (float)System.Math.Pow (System.Math.Log
((float)i / (float)tempopd), 2);
                        //and keep track of the best-scoring predecesor
                        if (score > smax) {
                              smax = score;
                              smaxix = i;
                        }
                  }

                  scorefun [currentTimeMark] = smax;
                  //it is best if we keep the smallest value in the score fn
window as zero
                  //we can easily do that by subtracing the min val
                  float smin = scorefun [0];
                  for (int i = 0; i < maxcolumn; ++i)
                        if (scorefun [i] < smin)
                              smin = scorefun [i];
                  for (int i = 0; i < maxcolumn; ++i)
                        scorefun [i] -= smin;

                  //we need to find the largest value in the score fn window,
to decide if we emit a beat signal
                  smax = scorefun [0];
                  smaxix = 0;
                  for (int i = 0; i < maxcolumn; ++i) {
                        if (scorefun [i] > smax) {
                              smax = scorefun [i];
                              smaxix = i;
                        }
                  }

                  //dobeat array records where we actally place beats
                  //we initially assume there is no beat this frame
                  dobeat [currentTimeMark] = 0;
                  ++doubleBeatSuppressor;
                  //if current value is largest in the array, probably means
we have found a beat
                  if (smaxix == currentTimeMark) {
                        //make sure the most recent beat isn't close to another
one
                        if (doubleBeatSuppressor > tempopd / 4) {
                              onBeat.Invoke ();
                              beatSignalDelay [0] = 1;
                              //record that we did actually mark a beat in this
frame
                              dobeat [currentTimeMark] = 1;
                              //reset counter of frames since last beat
                              doubleBeatSuppressor = 0;
                        }
                  }

                  //update the column index for the ring buffer
                  if (++currentTimeMark == maxcolumn)
                        currentTimeMark = 0;
            }
      }
```

```
	public float getBandWidth ()
	{
		return (2f / (float)fftSize) * (fftRate / 2f);
	}

	public int frequencyToIndex (int freq)
	{
		//if the frequency is lower than the bandwidth of spectrum[0]
		if (freq < getBandWidth () / 2)
			return 0;
		//if the frequency is within the bandwidth of spectrum[512]
		if (freq > fftRate / 2 - getBandWidth () / 2)
			return (fftSize / 2);
		//all other possible cases
		float fraction = (float)freq / (float)fftRate;
		int i = (int)System.Math.Round (fftSize * fraction);
		return i;
	}

	public void calculateAverages (float[] data)
	{
		for (int i = 0; i < 12; i++) {
			float avg = 0;
			int lowFrequency;
			if (i == 0)
				lowFrequency = 0;
			else
				lowFrequency = (int)((fftRate / 2) /
(float)System.Math.Pow (2, 12 - i));
			int hiFrequency = (int)((fftRate / 2) /
(float)System.Math.Pow (2, 11 - i));
			int lowBoundary = frequencyToIndex (lowFrequency);
			int hiBoundary = frequencyToIndex (hiFrequency);
			for (int j = lowBoundary; j <= hiBoundary; j++) {
				avg += data [j];
			}
			avg /= (hiBoundary - lowBoundary + 1); //not working as
expected with avg /= (hiBoundary - lowBoundary);
			averages [i] = avg;
		}
	}

	float mapping (float s, float a1, float a2, float b1, float b2)
	{
		return b1 + (s - a1) * (b2 - b1) / (a2 - a1);
	}

	public float addConstrain (float value, float inclusiveMinimum, float
inlusiveMaximum)
	{
		if (value >= inclusiveMinimum) {
			if (value <= inlusiveMaximum) {
				return value;
			}

			return inlusiveMaximum;
		}
```

```
            return inclusiveMinimum;
      }


      [System.Serializable]
      public class OnBeatEventHandler : UnityEngine.Events.UnityEvent
      {

      }

      [System.Serializable]
      public class OnSpectrumEventHandler :
UnityEngine.Events.UnityEvent<float []>
      {

      }


      //Autocorrelator: used to compute an array of online autocorrelators
      private class Autocorrelator
      {
            private int indx;
            private int delayLength;
            private float[] delays;
            private float[] outputs;
            private float decay;

            private float octaveWidth;
            private float[] beatPerMillisecond;
            private float[] rweight;
            private float wmidbpm = 120f;

            public Autocorrelator (int len, float alpha, float framePeriod,
float bandwidth)
            {
                  octaveWidth = bandwidth;
                  decay = alpha;
                  delayLength = len;
                  delays = new float[delayLength];
                  outputs = new float[delayLength];
                  indx = 0;

                  //calculate a log-lag gaussian weighting function, tempi
should be around 120 bpm
                  beatPerMillisecond = new float[delayLength];
                  //weighting is Gaussian on log-BPM axis, centered at wmidbpm
                  rweight = new float[delayLength];
                  for (int i = 0; i < delayLength; ++i) {
                        beatPerMillisecond [i] = 60.0f / (framePeriod *
(float)i);
                        rweight [i] = (float)System.Math.Exp (-0.5f *
System.Math.Pow (System.Math.Log (beatPerMillisecond [i] / wmidbpm) /
System.Math.Log (2.0f) / octaveWidth, 2.0f));
                  }
            }

            public void newValue (float val)
            {
```

```
                delays [indx] = val;

                //update running autocorrelator values
                for (int i = 0; i < delayLength; ++i) {
                      int delayIndex = (indx - i + delayLength)
% delayLength;
                      outputs [i] += (1 - decay) * (delays [indx] * delays
[delayIndex] - outputs [i]);
                }

                if (++indx == delayLength)
                      indx = 0;
          }

          //read back the current autocorrelator value at a particular lag
          public float autocorrelate (int del)
          {
                float ac = rweight [del] * outputs [del];
                return ac;
          }

          public float averageBeatPerMillisecond ()
          {
                float sum = 0;
                for (int i = 0; i < beatPerMillisecond.Length; ++i) {
                      sum += beatPerMillisecond [i];
                }
                return sum / delayLength;
          }
     }
}
```

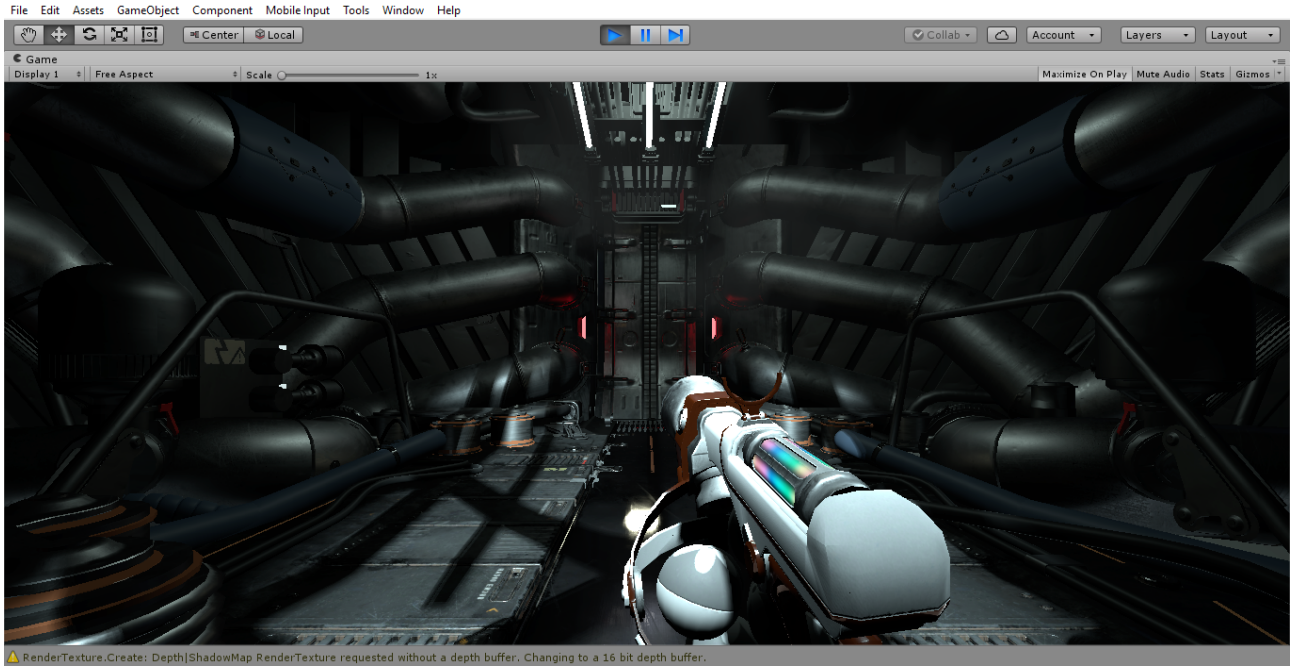
Στην συνέχεια, έχοντας ολοκληρώσει και το script για την ανίχνευση του ρυθμού, θα το εντάξουμε και αυτό σε αντικείμενο. Επειδή η πηγή ήχου θα είναι κάθε φορά ένα διαφορετικό VideoWall λόγω του ζητήματος της πλατφόρμας, θα πρέπει να ενσωματώσουμε το script σε κάθε υποαντικείμενο που διαθέτει audiosource με μουσικό κομμάτι. Αντίστοιχα με το προηγούμενο κεφάλαιο, προτείνεται η ένταξη του κώδικα μόνο στο VideoWallShape διότι στο επόμενο κεφάλαιο θα γίνουν πολύ σημαντικές προσθήκες και θα κάνουμε νέους κλώνους βάση του αντικείμενου αυτού.
Όταν προσθέσουμε τον κώδικα στο αντικείμενο πρέπει να προσέξουμε το πεδίο Global Threshold. Μέσω του πεδίου αυτού, μπορούμε να αλλάζουμε την τιμή στο κατώφλι της ευαισθησίας. Μετά από δοκιμές, η τιμή που θεωρήθηκε ως βέλτιστη ήταν ο αριθμός 3.
Πρέπει να αναφερθεί πως μόνο η προσάρτηση του κώδικα σε ένα αντικείμενο δεν αρκεί για να δοκιμάσουμε την λειτουργία του. Στο επόμενο κεφάλαιο, θα εντάξουμε σε άλλο script αντικείμενο τύπου AudioProcessor έτσι ώστε να λαμβάνει την πληροφορία κάθε φορά που έχουμε ανίχνευση ρυθμού από την πηγή ήχου (κλήση της συνάρτησης onOnbeatDetected).



Εικόνα 2.8.i: Προσθήκη του ανιχνευτή ρυθμού στα Videowall της σκηνής

2.9 Ολοκληρώνοντας το Παιχνίδι με Spawns, Point System και Pause Feature

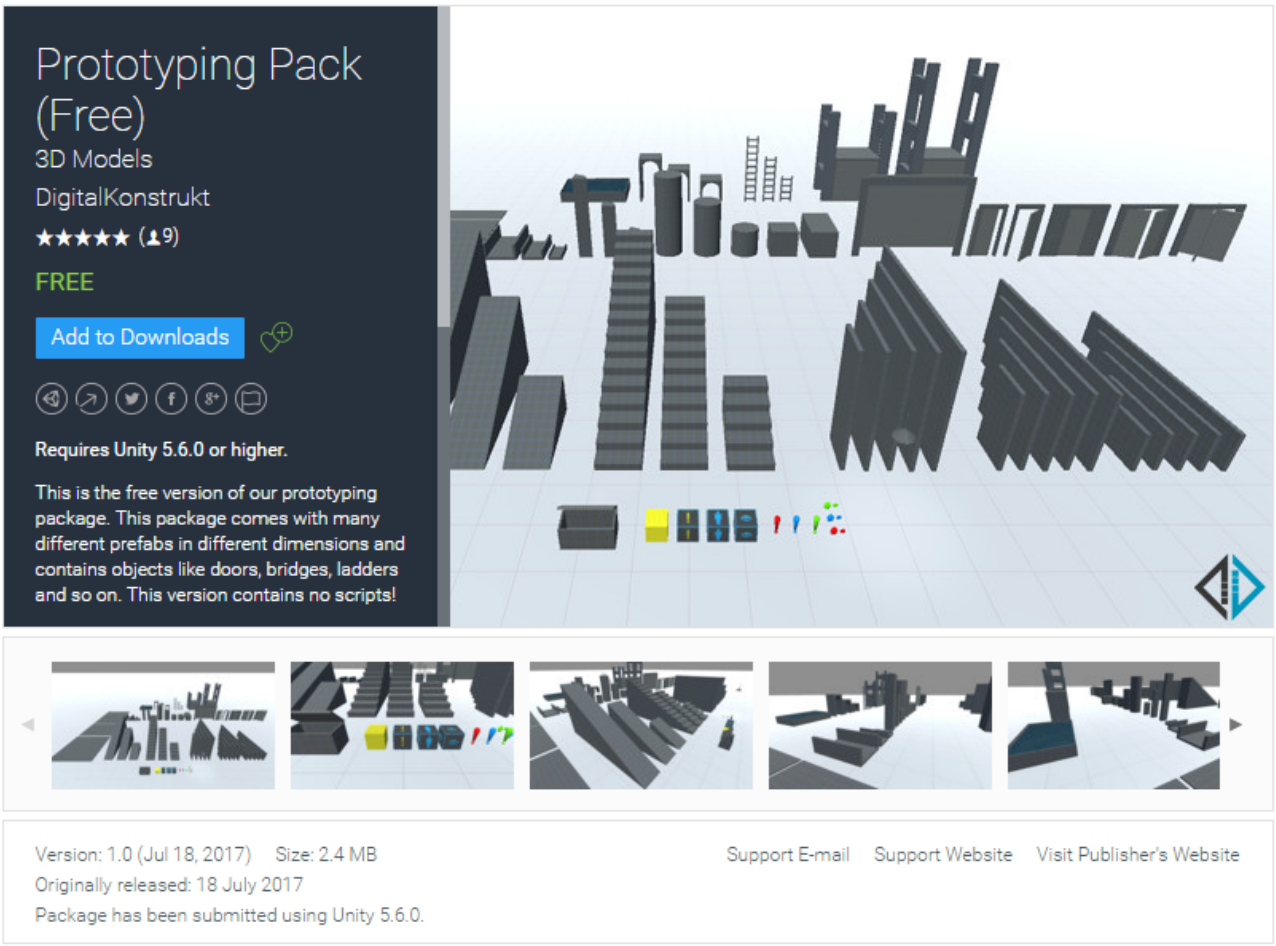
Στο προηγούμενο κεφάλαιο είδαμε πως γίνεται να επεξεργαστούμε σε πραγματικό χρόνο ένα ηχητικό σήμα για να ανιχνεύσουμε τον ρυθμό. Σε ένα τυπικό rhythm game, ο παίχτης πρέπει να πραγματοποιεί μια ακολουθία ενεργειών σε κάθε ρυθμικό χτύπημα. Εφόσον μέρος της έμπνευσης για την υλοποίηση του παιχνιδιού ήταν το κλασσικό shooter Duck Hunt, πρέπει να σχεδιάσουμε την σκηνή κατάλληλα έτσι ώστε ο παίχτης να πυροβολεί στόχους (καλούς και κακούς) σε κάθε ανίχνευση ρυθμού.
Η αρχική ιδέα για την υλοποίηση των στόχων ήταν η δημιουργία μόνο δύο αντικειμένων parent, ένα καλό και ένα κακό, τα οποία θα χρησιμοποιούταν για χρονισμένα object/child instances που θα παράγονταν με ψευδοτυχαίο ποσοστό ανάλογα με το επίπεδο δυσκολίας που επέλεξε ο παίχτης. Ο κώδικας αυτός χρησιμοποιείται συχνά στα ballistic συστήματα βολών για την κατασκευή bullet. Κάθε φορά που ο παίχτης πατάει το Fire Button έχουμε και δημιουργία ενός instance για το βλήμα βάση του αντικειμένου αναφοράς bullet. Η τεχνική αυτή αποδίδει πολύ καλά αποτελέσματα σε παιχνίδια με υψηλές απαιτήσεις για hardware καθώς και δημιουργία μικρών αντικειμένων που δεν μεταφέρουν πολύ πληροφορία. Εάν επιχειρήσουμε να χρησιμοποιήσουμε αυτή τη μεθοδολογία σε ένα online game για παραγωγή Turret επί παραδείγματι, θα έχουμε πάρα πολύ μεγάλο lag και πτώση των fps (frame per second) διότι πρόκειται για περίπλοκο μοντέλο, με μεγάλη μεταφορά πληροφορίας και online παιχνίδι σε πραγματικό χρόνο.
Αντίστοιχο ζήτημα υπήρξε δυστυχώς και σε αυτή την εργασία. Καλώντας συνεχώς constructor και destructor για δημιουργία instances των parent αντικειμένων, σε κάθε χτύπημα του ρυθμού, είχε ως αποτέλεσμα πολλαπλά threads με ακριβώς την ίδια πληροφορία που απομυζούσαν πόρους του τερματικού και οδηγούσαν σε διακεκομμένο feed και πτώση fps. Πιθανότατα, εάν πραγματοποιούσαμε το project αυτό σε υπολογιστή με μεγαλύτερες δυνατότητες να είχαμε ένα seamless αποτέλεσμα χωρίς προβλήματα. Χαρακτηριστικά, στα αρχικά στάδια δοκιμών, έγινε χρήση του εξαιρετικής ποιότητας Asset εσωτερικού χώρου για το cinematic Adam της Unity το οποίο λόγω της υψηλής ποιότητας του, όταν περιείχε έναν FPS χαρακτήρα να κινείται μέσα σε αυτόν, είχε ζητήματα στο depth buffer, πτώση σε 16bit rendering και choppy feed με πολύ χαμηλά fps.



Εικόνα 2.9.i: Απορριφθείσα εκδοχή της σκηνής του παιχνιδιού λόγω υψηλών απαιτήσεων από το τερματικό

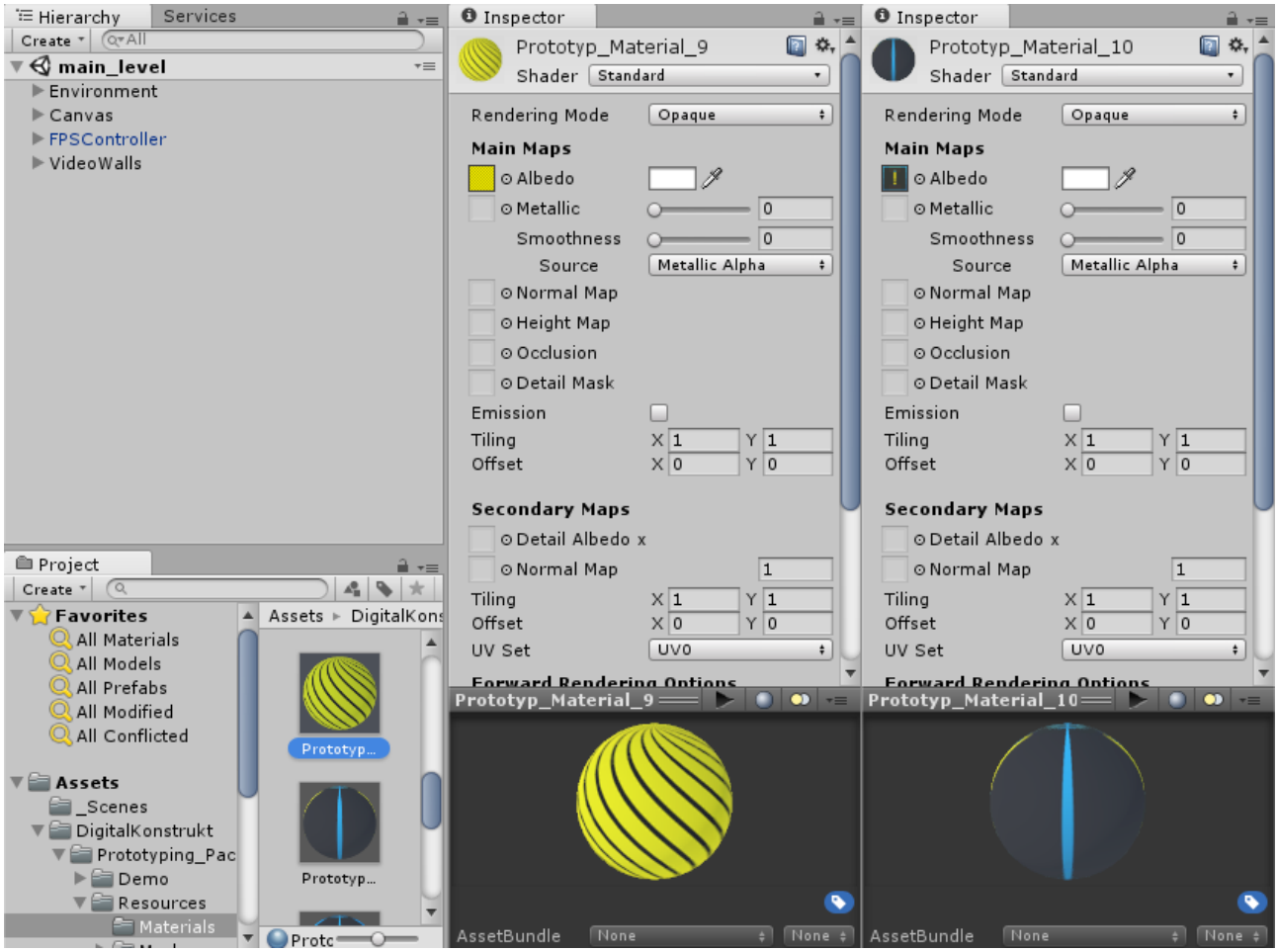
Και σε αυτό το κεφάλαιο, θα παρατεθεί στο τέλος του ενδεικτικά η αρχική μορφή του κώδικα σε περίπτωση που μπορεί να χρησιμοποιηθεί σε μελλοντική εκδοχή του παιχνιδιού ή ακόμα και σε κάποιο άλλο project.

Για να ξεπεράσουμε το ζήτημα, επιλέχθηκε η stable αλλά όχι optimal λύση της δημιουργίας ενός προκαθορισμένου πλέγματος με στόχους. Σε αυτή τη λύση δημιουργούμε ένα πεπερασμένο αριθμό αντικειμένων με το φόρτωμα της σκηνής και απλά τα ενεργοποιούμε και απενεργοποιούμε κατά βούληση σε κάθε ρυθμικό χτύπημα. Η τεχνική αυτή δεν κατανάλωνε επιπλέον πόρους και επέφερε σταθερό οπτικό αποτέλεσμα ακόμα και στο VR οπότε και επιλέχθηκε ως η τελική υλοποίηση.
Ας ξεκινήσουμε όμως με την σχεδίαση του πλέγματος στόχων. Σαν πρώτη κίνηση πρέπει να κατεβάσουμε και να φορτώσουμε ένα νέο πακέτο με Asset για τους στόχους μας. Ανοίγουμε λοιπόν για μια ακόμη φορά το Asset Store και αναζητούμε το Prototyping Pack (Free) της DigitalKonstruct.



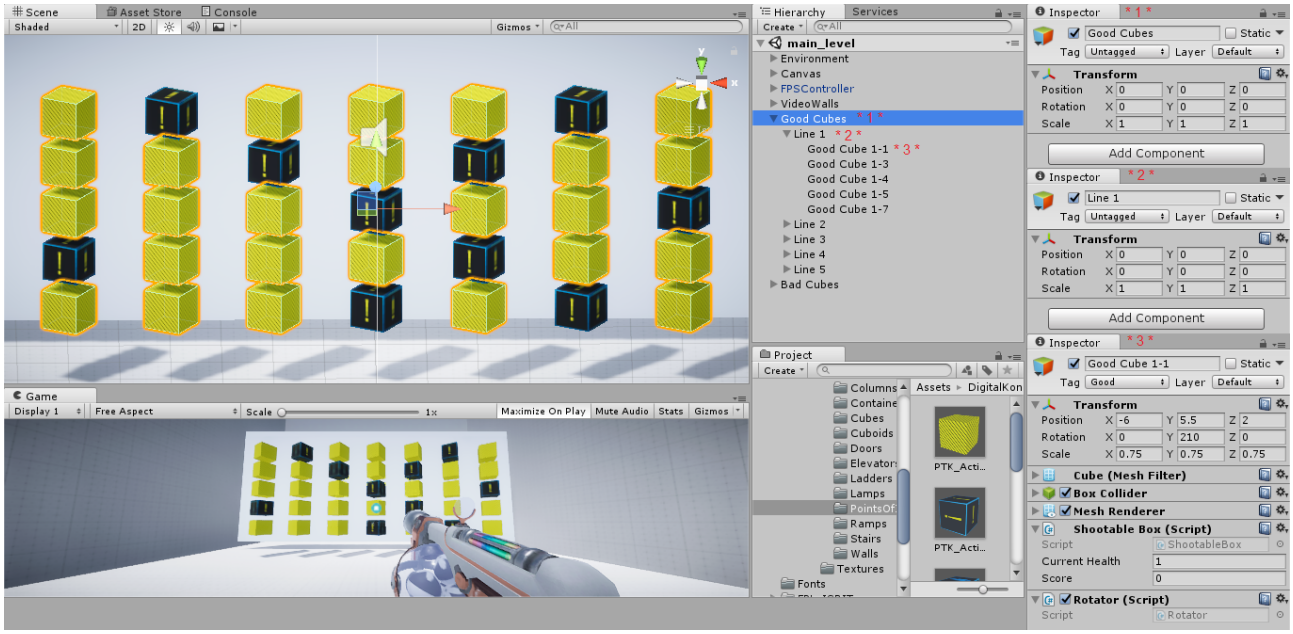
Εικόνα 2.9.ii: Το πακέτο Prototyping Pack (Free) για την δημιουργία των στόχων του παιχνιδιού

Όταν έχουμε κάνει και το Import του πακέτου, θα δούμε ότι στα Asset μας υπάρχει πλέον ο φάκελος DigitalKonstruct. Αυτό το πακέτο επιλέχθηκε κυρίως διότι είναι ένα από τα πιο πολυχρηστικά Asset δωρεάν διάθεσης με τα οποία μπορεί να γίνει πολύ εύκολα σχεδίαση ολόκληρων επιπέδων. Διαθέτει όλα τα βασικά γεωμετρικά στερεά που συναντούμε σε δομικές κατασκευές καθώς και πιο σύνθετα μοντέλα όπως γέφυρες, σκάλες ή game collectibles. Θα μπορούσαμε κάλλιστα να χρησιμοποιήσουμε απλές υφές που υπάρχουν στο διαδίκτυο για τους στόχους μας, αλλά αυτή ήταν πραγματικά μια ευκαιρία να δείξουμε ένα τόσο εύχρηστο δωρεάν πακέτο μοντέλων και υφών. Από το πακέτο αυτό θα κάνουμε χρήση μόνο των υφών που υπάρχουν στον αντίστοιχο υποφάκελο Materials αλλά μην χάσετε την ευκαιρία να δοκιμάσετε τις δυνατότητες του σε άλλα παιχνίδια.



Εικόνα 2.9.iii: Τα Materials που θα χρησιμοποιηθούν για τους καλούς και τους κακούς στόχους

Ας αρχίσουμε να δουλεύουμε πάνω στο πλέγμα στόχων όμως. Για να έχουμε μια εικόνα για το αποτέλεσμα που θέλουμε να πετύχουμε, ας ελέγξουμε την ακόλουθη εικόνα.



Εικόνα 2.9.iv: Πλέγμα καλών και κακών στόχων

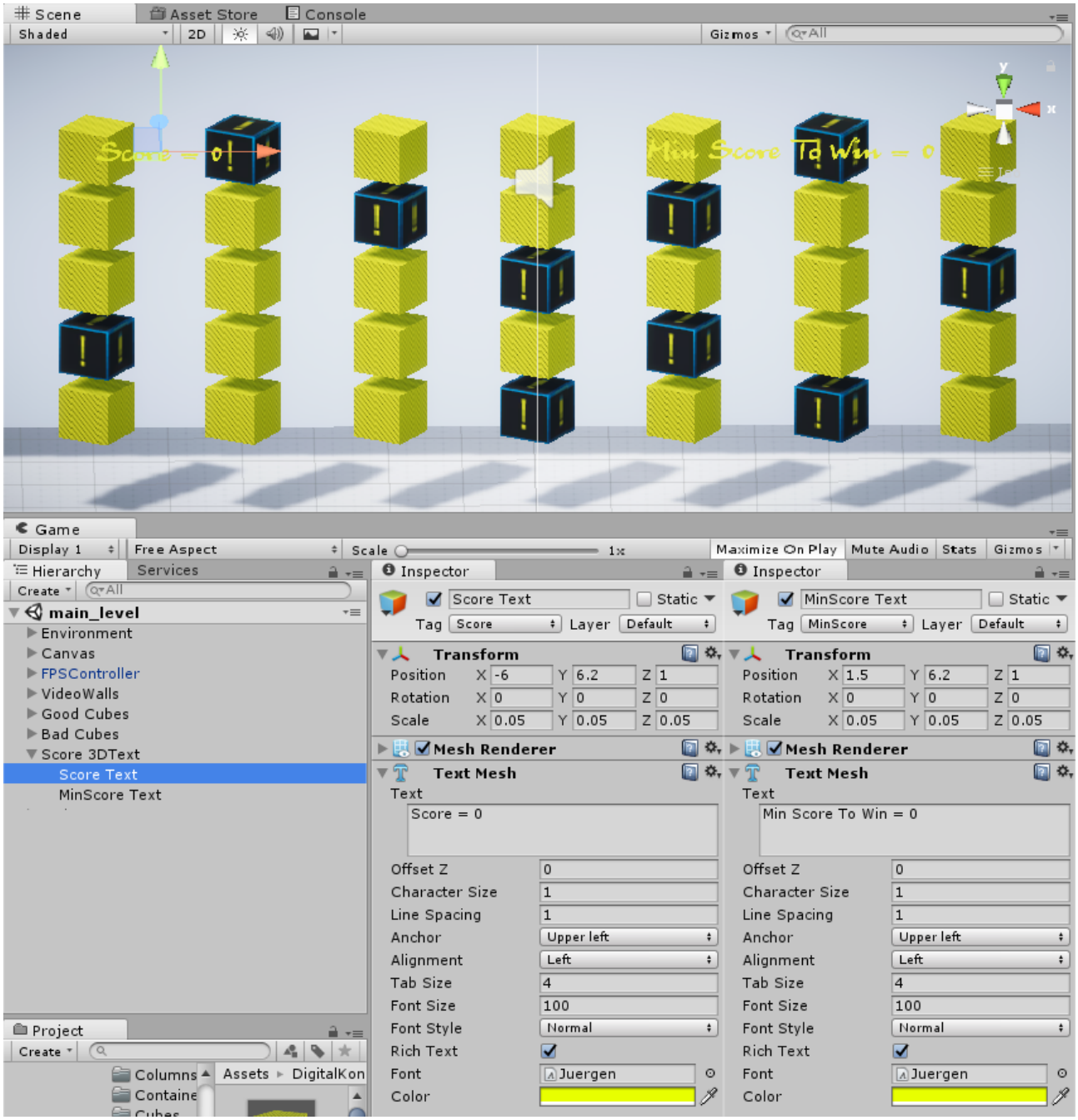
Καλό θα ήταν αρχικά να φτιάξουμε ένα σύστημα οργάνωσης των αντικειμένων από το να τα προσθέσουμε απλά στην ιεραρχία. Θα ξεκινήσουμε με τους καλούς κύβους. Αρχικά θα κάνουμε Create → Empty και θα φτιάξουμε ένα γενικό αντικείμενο με όνομα Good Cubes. Από το Inspector το κάνουμε Reset για να πάρει την default θέση στον χώρο και να μην επηρεάζει τα υποαντικείμενα του. Βασιζόμενοι στις διαστάσεις του Videowall πάνω στο οποίο θα εμφανίζονται οι στόχοι, υπολογίζουμε ότι χωράνε πέντε γραμμές ευκρινών στόχων οπότε και θα πρέπει να δημιουργήσουμε και κάνουμε reset πέντε υποαντικείμενα, ένα για κάθε γραμμή στόχων. Το κάθε αντικείμενο ονομάζεται Line X, όπου X ο αριθμός της εκάστοτε γραμμής. Μετά, κάνουμε έναν κλώνο αυτής της υλοποίησης και ονομάζουμε το ανώτατο αντικείμενο Bad Cubes.
Σε κάθε γραμμή χωράνε επτά στόχοι αλλά δεν είναι απαραίτητο να είναι όλοι καλοί. Πρέπει να έχουμε και κακούς στόχους ώστε ο παίχτης να χάνει πόντους όταν τους χτυπήσει. Αρχικά θα φτιάξουμε ένα καλό στόχο κάνοντας Create → 3D Object → Cube και τον εντάσσουμε στην ιεραρχία των κύβων/στόχων. Το μετονομάζουμε όπως φαίνεται στην παραπάνω εικόνα σε Good Cube 1-1 και ορίζουμε το Transform του. Στον Inspector του βλέπουμε ότι το αντικείμενο δεν περιέχει κάτι στην κατηγορία Tag. Μια εύκολη μέθοδος για να γνωρίζουμε τι είναι ο κάθε στόχος που πυροβολούμε, είναι η χρήση ετικετών. Κάνοντας κλικ πάνω στην επιλογή Untagged → Add Tag θα προσθέσουμε δύο νέες ετικέτες, Good και Bad. Στον κύβο που ήδη έχουμε θα βάλουμε ετικέτα Good και σε έναν κλώνο αυτού Bad. Επίσης προσθέτουμε με drag and drop το κίτρινο material στο καλό κύβο και το μαύρο στον κακό. Στη συνέχεια προχωράμε να κάνουμε κλώνους των αντικειμένων και να τα τοποθετήσουμε τόσο κατάλληλα στην ιεραρχία όσο και στην σκηνή. Για αναλυτικό σχεδιάγραμμα της ιεραρχίας των αντικειμένων, ελέγξτε το κεφάλαιο με τίτλο Βοηθητικό Index Αντικειμένων/ Script για τις Σκηνές του Παιχνιδιού.
Μπορούμε να ακολουθήσουμε το πρότυπο όπως εμφανίζεται στην παραπάνω εικόνα ή να το τροποποιήσουμε όπως επιθυμούμε. Οι θέσεις των κύβων (Transform) στο πλέγμα είναι οι παρακάτω:

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
Line 1	(-6, 5.5, 2)	(-4, 5.5, 2)	(-2, 5.5, 2)	(0, 5.5, 2)	(2, 5.5, 2)	(4, 5.5, 2)	(6, 5.5, 2)
Line 2	(-6, 4.5, 2)	(-4, 4.5, 2)	(-2, 4.5, 2)	(0, 4.5, 2)	(2, 4.5, 2)	(4, 4.5, 2)	(6, 4.5, 2)
Line 3	(-6, 3.5, 2)	(-4, 3.5, 2)	(-2, 3.5, 2)	(0, 3.5, 2)	(2, 3.5, 2)	(4, 3.5, 2)	(6, 3.5, 2)
Line 4	(-6, 2.5, 2)	(-4, 2.5, 2)	(-2, 2.5, 2)	(0, 2.5, 2)	(2, 2.5, 2)	(4, 2.5, 2)	(6, 2.5, 2)
Line 5	(-6, 1.5, 2)	(-4, 1.5, 2)	(-2, 1.5, 2)	(0, 1.5, 2)	(2, 1.5, 2)	(4, 1.5, 2)	(6, 1.5, 2)

Σημείωση: Το Rotation είναι σταθερά (0, 210, 0) και το Scale (0.75, 0.75, 0.75) για όλα τα αντικείμενα.

Έχοντας ολοκληρώσει και το πλέγμα των στόχων, ας φτιάξουμε και το σύστημα υπολογισμού του score του παίχτη και των ελάχιστου ορίου πόντων που πρέπει να καλύψει για να κερδίσει το παιχνίδι. Επειδή τα παιχνίδια ρυθμού είναι εξ ορισμού δύσκολα διότι στοχεύουν σε περίπλοκες ακολουθίες ενεργειών και τα shooter game στα αντανακλαστικά και στις γρήγορες κινήσεις, επιλέξαμε σε κάθε ρυθμικό χτύπο να εμφανίζουμε ένα ψευδοτυχαίο πλήθος στόχων για να πυροβολεί ο παίχτης. Κάθε καλός στόχος δίνει δέκα πόντους ενώ κάθε κακός αφαιρεί 10. Στο Easy επίπεδο, κάθε ρυθμικός χτύπος θα αυξάνει το ελάχιστο κατώφλι για να κερδίσεις κατά 2 μονάδες, ενώ στο Hard κατά 4.
Όπως είδαμε και στα κεφάλαια κατασκευής του μενού και του Raycast, όταν κάνουμε χρήση χαρακτήρων FPS για προβολή και κίνηση στο χώρο, δεν μπορούμε να χρησιμοποιήσουμε τα UI στοιχεία για να βάλουμε το κείμενο του score στην οθόνη. Θα μετακινούνται όπου και το σημείο όρασης του παίχτη που είναι πολύ ενοχλητικό και λάθος γι’ αυτή την υλοποίηση. Χρειαζόμαστε ένα φυσικό αντικείμενο με τρισδιάστατη υπόσταση. Οπότε, για το score του παίχτη και για το ελάχιστο κατώφλι πόντων, θα χρησιμοποιήσουμε αντικείμενα τύπου 3D Text.
Όπως και στις υπόλοιπες υλοποιήσεις μας, θα φτιάξουμε ένα κενό αντικείμενο με όνομα Score 3DText και θα το κάνουμε Reset στον χώρο. Στη συνέχεια, θα κάνουμε Create → 3D Object → 3D Text και θα το τοποθετήσουμε ως υποαντικείμενο στην ιεραρχία. Το μετονομάζουμε σε Score Text και δημιουργούμε

κλώνο αυτού με το όνομα MinScore Text. Μπορούμε να τα μορφοποιήσουμε όπως επιθυμούμε και στο συγκεκριμένο παιχνίδι οι ρυθμίσεις των αντικειμένων είναι οι ακόλουθες:

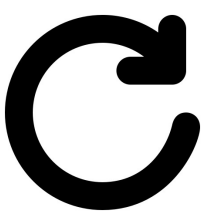


Εικόνα 2.9.v: Κατασκευή και παραμετροποίηση των 3D Text για το score.

Σημείωση: εάν φαίνεται θολό το κείμενο αρχικά, μην το θεωρήσετε ως πρόβλημα. Αλλάζοντας το μέγεθος της γραμματοσειράς σε μεγαλύτερο το πρόβλημα διορθώνεται αμέσως. Μην ξεχνάμε ότι πρόκειται για GameObject οπότε το scale του μπορεί να αλλάξει ανεξαρτήτως Font Size.

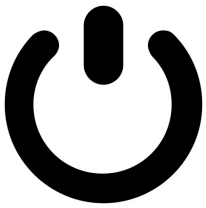
Έχοντας φτιάξει και τα αντικείμενα για το scoreboard, ας υλοποιήσουμε και τα τελευταία αντικείμενα αυτού το παιχνιδιού, δηλαδή το Pause/End Game Menu. Μια τυπική εικόνα ενός τέτοιου μενού είναι το κείμενο που υποδηλώνει την τρέχουσα κατάσταση (Pause/ Win/ Lose), ένα reload stage button, ένα return to main menu button και ένα exit game button. Για να μπορέσουμε να φτιάξουμε το μενού δεν μπορούμε να βάλουμε UI στοιχεία για τα κουμπιά, οπότε μια απλή και διασκεδαστική λύση είναι να φτιάξουμε ένα μενού από στόχους. Αρχικά φτιάχνουμε ένα γενικό αντικείμενο με την ονομασία EndGame Components

και το κάνουμε Reset. Στη συνέχεια χρειαζόμαστε ένα 3D Text και τρεις κύβους για στόχους. Επίσης πρέπει να προσθέσουμε τρεις νέες ετικέτες, μια για κάθε στοιχείο μενού. Η πρώτη θα ονομάζεται Reload, η δεύτερη Menu και η τρίτη Exit. Με αυτή την απλή λύση θα ξέρουμε τι έχει επιλέξει ο παίχτης κάθε φορά χωρίς κόπο. Επιπλέον, εφόσον θέλουμε να φτιάξουμε μενού με στόχους, μια ενδιαφέρουσα ιδέα είναι να βάλουμε δικό μας Material για κάθε στόχο με το εικονίδιο που αντιπροσωπεύει κάθε επιλογή. Τα εικονίδια που χρησιμοποιήθηκαν είναι τα παρακάτω:



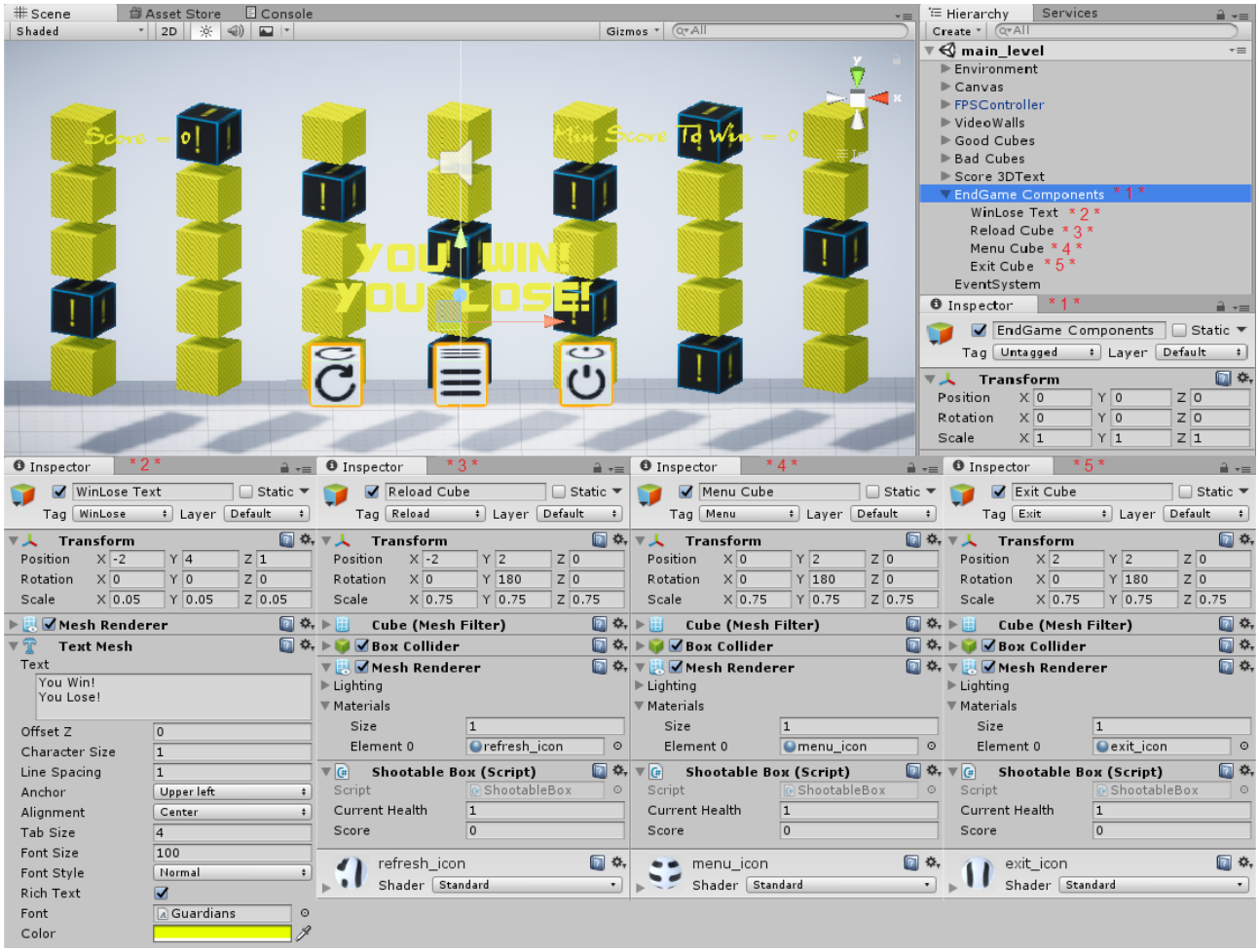
Reload Material

Menu Material



Exit Material

Τα ονόματα, οι θέσεις, οι διαστάσεις και η τελική εικόνα των αντικειμένων μενού στον χώρο πρέπει να είναι η εξής:



Εικόνα 2.9.vi: Κατασκευή του utility menu για χρήση ως Pause και EndGame Menu

Αφού έχουμε φτιάξει όλα τα αντικείμενα που χρειαζόμαστε για το παιχνίδι, ας ολοκληρώσουμε τα script. Ας ξεκινήσουμε με τον κώδικα που θα παράγει ψευδοτυχαία τα spawns των στόχων που θα ενεργοποιούνται και απενεργοποιούνται σε κάθε χτύπημα του ρυθμού. Χρειαζόμαστε να καλούμε ένα αντικείμενο τύπου AudioProcessor (που υλοποιήσαμε στο προηγούμενο κεφάλαιο) και να ελέγχουμε εάν

χτύπησε ρυθμός για να εμφανίζουμε στόχους. Επίσης, πρέπει να βάλουμε την δυνατότητα ο παίχτης να κάνει παύση στο παιχνίδι του πατώντας το πλήκτρο Enter. Για να πετύχουμε αυτό το αποτέλεσμα πρέπει να αναπτύξουμε τον ακόλουθο κώδικα με ονομασία random_spawn και να εντάξουμε το script του στον φάκελο Scripts_Beat των Assets.

random_spawn.cs

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

public class random_spawn : MonoBehaviour
{
      //whichLine: which line of the cube grid is selected
      public int whichLine;
      //cubeNumber: which column of the cube grid is selected
      public int cubeNumber;
      //spawns: number of active cubes on each song beat
      public int spawns;

      //textObject: 3D Text GameObject of Scene
      TextMesh textObject;

      AudioSource audioSource;

      //minPointsToWin: the minimum amount of points to win the stage/based
on the live beat count
      public int minPointsToWin;
      //score: player's in game score
      public int score;

      //difficulty: keeps the level difficulty the player selected/saved in
PlayerPrefs
      public int difficulty;

      //isinpause: checks if the player has pressed the Pause button
      public int isinpause;
      //songtime: total length of the song in seconds
      public float songtime;
      //timeinpause: sum of seconds between hitting the pause and unpause
button
      public float timeinpause;
      //timepause: timemark that the player paused the game
      public float timepause;
      //timeunpause: timemark that the player unpaused the game
      public float timeunpause;
      //overaltime: the sum of seconds since the load of the scene in seconds
(song time + time in pause)
      public float overaltime;

      //reloadbox: the shootable box that reloads the scene
      public GameObject reloadbox;
      //menubox: the shootable box that returns the player to the main menu
      public GameObject menubox;
      //exitbox: the shootable box that quits the game
```

```
public GameObject exitbox;

//boxij: the grid of shootable boxes we have on the scene
public GameObject box11;
public GameObject box12;
public GameObject box13;
public GameObject box14;
public GameObject box15;
public GameObject box16;
public GameObject box17;

public GameObject box21;
public GameObject box22;
public GameObject box23;
public GameObject box24;
public GameObject box25;
public GameObject box26;
public GameObject box27;

public GameObject box31;
public GameObject box32;
public GameObject box33;
public GameObject box34;
public GameObject box35;
public GameObject box36;
public GameObject box37;

public GameObject box41;
public GameObject box42;
public GameObject box43;
public GameObject box44;
public GameObject box45;
public GameObject box46;
public GameObject box47;

public GameObject box51;
public GameObject box52;
public GameObject box53;
public GameObject box54;
public GameObject box55;
public GameObject box56;
public GameObject box57;

//public AudioClip clip;


void BoxDeactivator ()
{
      //deactivate all the boxes in the grid
      box11.SetActive(false);
      box12.SetActive(false);
      box13.SetActive(false);
      box14.SetActive(false);
      box15.SetActive(false);
      box16.SetActive(false);
      box17.SetActive(false);

      box21.SetActive(false);
```

```
            box22.SetActive(false);
            box23.SetActive(false);
            box24.SetActive(false);
            box25.SetActive(false);
            box26.SetActive(false);
            box27.SetActive(false);

            box31.SetActive(false);
            box32.SetActive(false);
            box33.SetActive(false);
            box34.SetActive(false);
            box35.SetActive(false);
            box36.SetActive(false);
            box37.SetActive(false);

            box41.SetActive(false);
            box42.SetActive(false);
            box43.SetActive(false);
            box44.SetActive(false);
            box45.SetActive(false);
            box46.SetActive(false);
            box47.SetActive(false);

            box51.SetActive(false);
            box52.SetActive(false);
            box53.SetActive(false);
            box54.SetActive(false);
            box55.SetActive(false);
            box56.SetActive(false);
            box57.SetActive(false);
      }

      void RandomBoxGenerator()
      {
            //how many spawns we will have active, min 3 - max 6
            spawns = UnityEngine.Random.Range(3, 7);

            int s;

            //for each one of the spawns to be activated
            for (s=1; s<=spawns; s++)
                  {
                        //we select randomly a cube, line and column
                        whichLine = UnityEngine.Random.Range(1, 6); //line
number, is exclusive on the 2nd part
                        cubeNumber = UnityEngine.Random.Range(1, 8); //for
integers between 1-7
                        if (whichLine == 1)
                        {
                              if (cubeNumber == 1) {
                                    box11.SetActive(true);
                              }
                              else if (cubeNumber == 2) {
                                    box12.SetActive(true);
                              }
                              else if (cubeNumber == 3) {
                                    box13.SetActive(true);
                              }
```

```
            else if (cubeNumber == 4) {
                  box14.SetActive(true);
            }
            else if (cubeNumber == 5) {
                  box15.SetActive(true);
            }
            else if (cubeNumber == 6) {
                  box16.SetActive(true);
            }
            else {
                  box17.SetActive(true);
            }
      }
      if (whichLine == 2)
      {
            if (cubeNumber == 1) {
                  box21.SetActive(true);
            }
            else if (cubeNumber == 2) {
                  box22.SetActive(true);
            }
            else if (cubeNumber == 3) {
                  box23.SetActive(true);
            }
            else if (cubeNumber == 4) {
                  box24.SetActive(true);
            }
            else if (cubeNumber == 5) {
                  box25.SetActive(true);
            }
            else if (cubeNumber == 6) {
                  box26.SetActive(true);
            }
            else {
                  box27.SetActive(true);
            }
      }
      if (whichLine == 3)
      {
            if (cubeNumber == 1) {
                  box31.SetActive(true);
            }
            else if (cubeNumber == 2) {
                  box32.SetActive(true);
            }
            else if (cubeNumber == 3) {
                  box33.SetActive(true);
            }
            else if (cubeNumber == 4) {
                  box34.SetActive(true);
            }
            else if (cubeNumber == 5) {
                  box35.SetActive(true);
            }
            else if (cubeNumber == 6) {
                  box36.SetActive(true);
            }
            else {
```

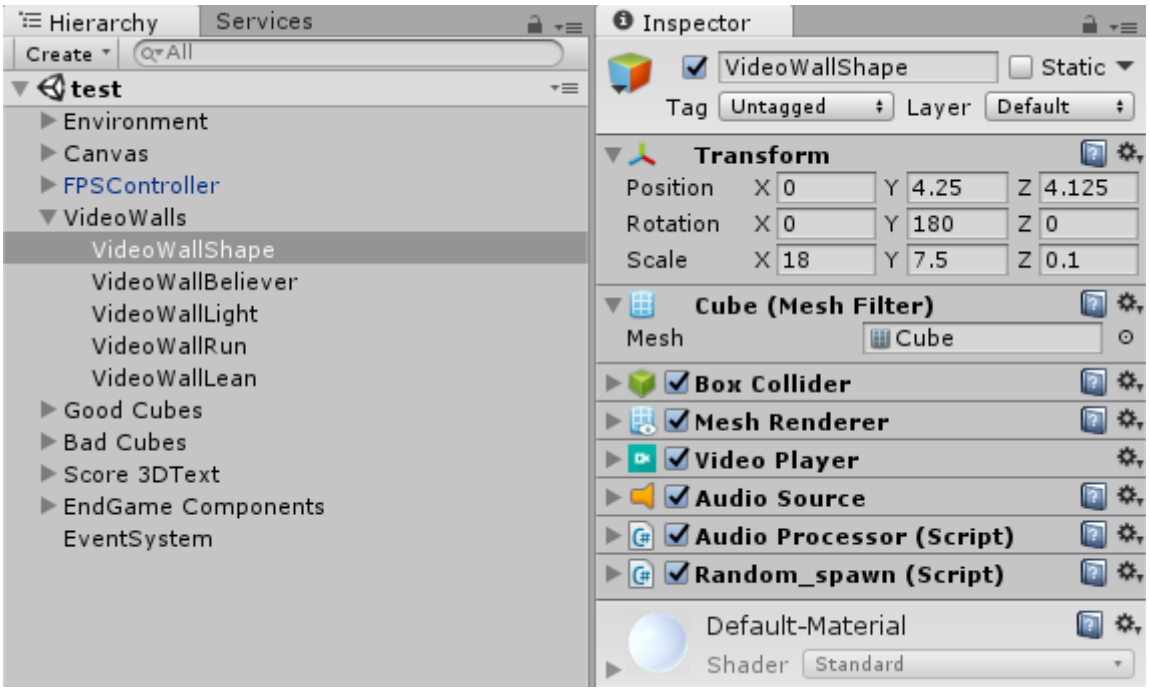
```
                        box37.SetActive(true);
                    }
                }
                if (whichLine == 4)
                {
                    if (cubeNumber == 1) {
                        box41.SetActive(true);
                    }
                    else if (cubeNumber == 2) {
                        box42.SetActive(true);
                    }
                    else if (cubeNumber == 3) {
                        box43.SetActive(true);
                    }
                    else if (cubeNumber == 4) {
                        box44.SetActive(true);
                    }
                    else if (cubeNumber == 5) {
                        box45.SetActive(true);
                    }
                    else if (cubeNumber == 6) {
                        box46.SetActive(true);
                    }
                    else {
                        box47.SetActive(true);
                    }
                }
                if (whichLine == 5)
                {
                    if (cubeNumber == 1) {
                        box51.SetActive(true);
                    }
                    else if (cubeNumber == 2) {
                        box52.SetActive(true);
                    }
                    else if (cubeNumber == 3) {
                        box53.SetActive(true);
                    }
                    else if (cubeNumber == 4) {
                        box54.SetActive(true);
                    }
                    else if (cubeNumber == 5) {
                        box55.SetActive(true);
                    }
                    else if (cubeNumber == 6) {
                        box56.SetActive(true);
                    }
                    else {
                        box57.SetActive(true);
                    }
                }
            } // end of for
    }

    void Start ()
    {
        //we will select the instance of AudioProcessor and pass a
reference to this object
```

```
            AudioProcessor processor = FindObjectOfType<AudioProcessor> ();
            processor.onBeat.AddListener (onOnbeatDetected);
            processor.onSpectrum.AddListener (onSpectrum);

            //we need to connect the audioSource of the object this script is
attached to
            audioSource = GetComponent<AudioSource>();

            //variable initialization
            minPointsToWin = 0;

            timeinpause = 0;
            overaltime = 0;
            songtime = audioSource.clip.length;
            timepause = 0;
            timeunpause = 0;

            isinpause = 0; //the game when the stage loads is not in pause

            BoxDeactivator(); //box grid not active when the stage begins

            //the utility boxes are not active either
            reloadbox.SetActive(false);
            menubox.SetActive(false);
            exitbox.SetActive(false);

            //there is no visible 3D Text in the screen with the game results
            textObject =
GameObject.FindWithTag("WinLose").GetComponent<TextMesh>();
            textObject.text = " ";
      }

      // fixed update time can be changed from the platform settings
      void FixedUpdate()
      {
            //if the player presses the Enter button on their keyboard
            if (Input.GetKeyDown (KeyCode.Return))
            {
                  AudioSource audio = GetComponent<AudioSource>();
                  VideoPlayer videoPlayer = GetComponent<VideoPlayer> ();
                  //if the video is not already paused
                  if (isinpause == 0){
                        //pause the video and the music
                        videoPlayer.Pause ();
                        audio.Pause();
                        //deactivate all the active spawns so the player can
not shoot them
                        BoxDeactivator();
                        //use the WinLose 3D Text to show the message Paused
                        //there is no need to have a different 3D Text object
                        textObject.text = " Paused!";
                        Debug.Log ("Pause");
                        //activate the utility boxes
                        reloadbox.SetActive(true);
                        menubox.SetActive(true);
                        exitbox.SetActive(true);
                        //declare that we are in pause condition
                        isinpause = 1;
```

```
                        //keep the timestamp that the player performed this
action
                        timepause = Time.time;
                  }
                  else {
                        //if we are in pause condition and Enter is pressed
again
                        //we set that we are no longer paused
                        isinpause = 0;//unpause
                        //keep the timestamp that the player performed this
action
                        timeunpause = Time.time;
                        //calculate the seconds that passed while in pause
                        timeinpause += (timeunpause - timepause);
                        //start video and music in the VideoPlayer
                        videoPlayer.Play ();
                        audio.Play();
                        //generate new spawns in case the player memorised
their position
                        RandomBoxGenerator();
                        //hide the 3D Text
                        textObject.text = " ";
                        Debug.Log ("Unpause");
                        //deactivate the utility boxes
                        reloadbox.SetActive(false);
                        menubox.SetActive(false);
                        exitbox.SetActive(false);
                  }
            }
      }

      //this event will be called every time a beat is detected
      //you can change the threshold parameter in the inspector to adjust the
sensitivity
      void onOnbeatDetected ()
      {
            Debug.Log ("Beat!!!");
            //get the difficulty the player has selected from the menu
            difficulty = PlayerPrefs.GetInt("SelectedDifficulty");
            if (difficulty == 0){
                  //if it is on easy add 2 points on the win score minimum
every time we find a beat
                  minPointsToWin += 2;
            }
            else if (difficulty == 1){
                  //if it is on hard, then we add 4 points
                  minPointsToWin += 4;
            }

            //Update the 3D Text with the tag MinScore
 and keep the current score in PlayerPrefs
            TextMesh textObject =
GameObject.FindWithTag("MinScore").GetComponent<TextMesh>();
            textObject.text = "Min Score To Win = "+minPointsToWin;
            PlayerPrefs.SetInt("MinScoreToWin", minPointsToWin);

            Debug.Log ("Min Points To Win = "+minPointsToWin);
```

```
            //deactivate the old boxes
            BoxDeactivator();
            //and generate new spawns
            RandomBoxGenerator();
      }

      //runs once per frame
      void Update()
     {
            //we calculate sum of the duration of each song and the time in
pause
            overaltime = timeinpause + songtime;
            //we check if the song is not playing and if the current timestamp
matches the stage-end timestamp
            if ((Time.timeSinceLevelLoad >= overaltime) && (!
audioSource.isPlaying))
            {
                  //if we use Time.Time it does not work properly because it
calculates the time
                  //since the beginning of the game and continues to count
even when we reload or
                  //change scenes
                  //must not use Time.Time because it will always activate on
reload and show Win/Lose on Pause
                  BoxDeactivator();
             Debug.Log ("End of Song!!!");
                  //check if the player has reached the required score to win
and show the appropriate message
                  score = PlayerPrefs.GetInt("Score");
                  if (score >= minPointsToWin)
                  {
                        textObject.text = "You Win!";
                  }
                  else
                  {
                        textObject.text = "You Lose!";
                  }
                  //also activate the utility boxes to allow the player to
play again, return to the main menu or exit
                  reloadbox.SetActive(true);
                  menubox.SetActive(true);
                  exitbox.SetActive(true);
            }
     }

      //This event will be called every frame while music is playing
      void onSpectrum (float[] spectrum)
      {
            //The spectrum is logarithmically averaged to 12 bands
            for (int i = 0; i < spectrum.Length; ++i) {
                  Vector3 start = new Vector3 (i, 0, 0);
                  Vector3 end = new Vector3 (i, spectrum [i], 0);
                  Debug.DrawLine (start, end);
            }
      }
}
```

Με την ολοκλήρωση του script, πρέπει να το εντάξουμε πλέον και στα Videowall. Στο κεφάλαιο που δουλέψαμε πάνω στα Videowall είπαμε πως δεν ήταν αναγκαίο να τα φτιάξουμε όλα πλήρως γιατί θα έπρεπε να κάνουμε κάποιες σημαντικές αλλαγές. Εντάσσοντας το script random_spawn στο VideoWallShape μαζί με το AudioProcessor και κάνοντας την ορθή αντιστοίχηση των στόχων του grid με τα αντικείμενα του random_spawn, μπορούμε πλέον να κάνουμε σωστούς κλώνους και για τα λοιπά υποαντικείμενα βάση του VideoWallShape. Απλά πρέπει να ορίσουμε σωστά τα μουσικά κομμάτια και βίντεο για κάθε οθόνη προβολής στα εκάστοτε Video Player και Audio Source.



Εικόνα 2.9.vii: Τελική μορφή των στοιχείων των Videowall

Σε αυτό το στάδιο, μένει μόνο να ανανεώσουμε τον κώδικα του script ShootableBox για να ελέγχουμε την ετικέτα του κάθε αντικειμένου που πυροβολούμε έτσι ώστε να γνωρίζουμε εάν είναι καλός στόχος, κακός στόχος ή απλά ένα αντικείμενο του βοηθητικού μενού. Η τελική μορφή του script πρέπει να είναι η ακόλουθη:

ShootableBox.cs

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

public class ShootableBox : MonoBehaviour {

	//The box's current health point total - 1 hit kill
	public int currentHealth = 1;
	public int score;
	//The idea of adding an Audioclip to match the box that was hit was an
	//initial idea but was rejected because it distracted the player from
the beat.
	//public AudioClip goodclip;
	//public AudioClip badclip;
```

```
	public void Damage(int damageAmount)
	{
		//subtract damage amount when Damage function is called
		currentHealth -= damageAmount;

		//check if health has fallen below zero
		if (currentHealth <= 0)
		{
			score = PlayerPrefs.GetInt("Score");
			//we need to see if the player hit a good or a bad box
			if (gameObject.CompareTag ("Good"))
			{
				//if the player shoots a good box, points are added
				//AudioSource.PlayClipAtPoint(goodclip,  new  Vector3(0,
0, 0));
				score += 10;
			}
			if (gameObject.CompareTag ("Bad"))
			{
				//if the player shoots a bad box, points are subtracted
				//AudioSource.PlayClipAtPoint(badclip,  new  Vector3(0,
0, 0));
				score -= 10;
			}
			//the next cases are for the utility boxes only
			if (gameObject.CompareTag ("Reload"))
			{
				//if  the  player  hits  reload  then  the  same  level  is
loaded, on the same difficulty
				//the  selected  song  is  the  same  and  is  stored  on
PlayerPrefs
				int                 selectedDifficulty                =
PlayerPrefs.GetInt("SelectedDifficulty");
				if (selectedDifficulty == 0){
				SceneManager.LoadScene("main_level");
				//Application.LoadLevel("main_level");        //obsolete
expression
				}
				else if (selectedDifficulty == 1){
				SceneManager.LoadScene("main_level_hard");
				}
				//SceneManager.GetActiveScene();   //   does   not   work
properly with the pause addition
				//SceneManager.LoadScene(SceneManager.GetActiveScene().
name);
			}
			if (gameObject.CompareTag ("Menu"))
			{
				SceneManager.LoadScene("menu"); //load menu scene
			}
			if (gameObject.CompareTag ("Exit"))
			{
				Debug.Log ("Game is exiting...");
				Application.Quit();
```

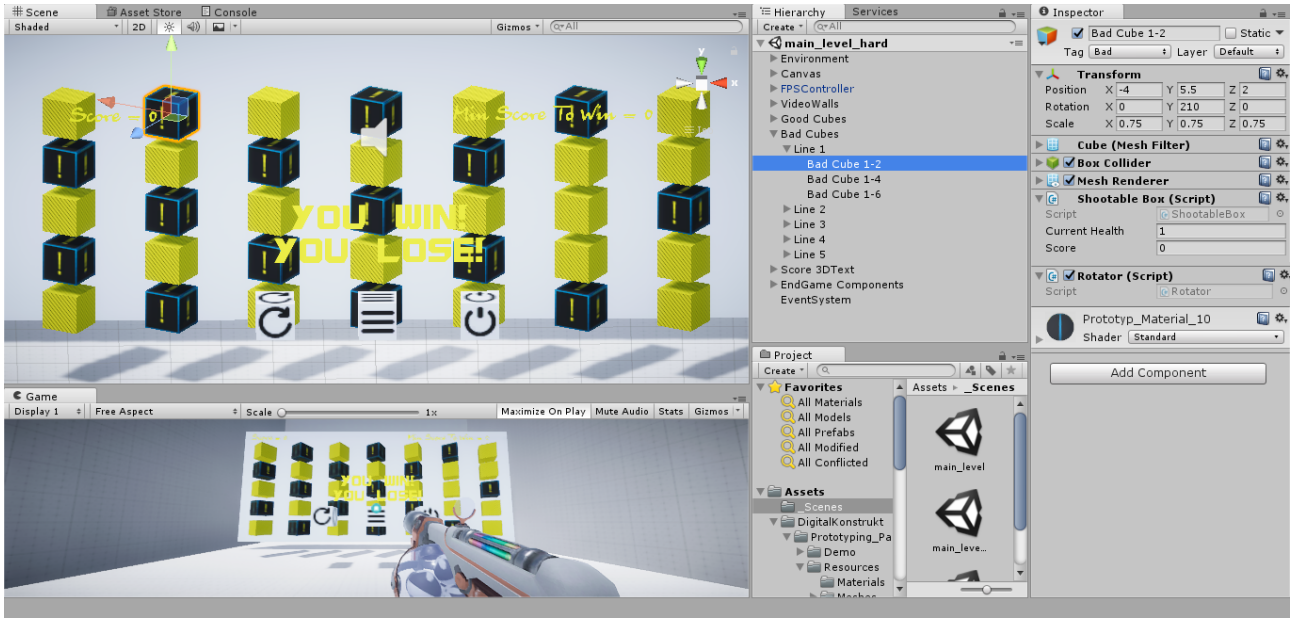
```
                }
                //deactivate the box with 0 health
                gameObject.SetActive (false);
                //update the score in PlayerPrefs and the scene 3D Text as
well
                PlayerPrefs.SetInt("Score", score);
                TextMesh                    textObject                    =
GameObject.FindWithTag("Score").GetComponent<TextMesh>();
                textObject.text = "Score = "+score;
            }
      }
}
```

Τον συγκεκριμένο κώδικα πρέπει να τον βάλουμε σε όποιο αντικείμενο μπορεί να πυροβοληθεί, δηλαδή και σε όλους τους στόχους του πλέγματος αλλά και σε όλα τα κουμπιά/στόχους του βοηθητικού μενού. Επιπλέον, εάν θέλουμε να κάνουμε ένα τελευταίο βήμα προς την ωραιοποίηση του πλέγματος με τους στόχους, μπορούμε να βάλουμε σε αυτούς ένα script που θα τους περιστρέφει κάνοντας τους να "χορεύουν" στο ρυθμό της μουσικής. Το script αυτό θα το ονομάσουμε Rotator και θα το βάλουμε στον φάκελο Scripts_Beat των Assets. Αυτό το βήμα είναι καθαρά προαιρετικό αλλά προσδίδει ένα πολύ ευχάριστο αισθητικό αποτέλεσμα. Το Rotator είναι βέλτιστο να χρησιμοποιηθεί μόνο στο πλέγμα και όχι στους στόχους του βοηθητικού μενού.

Rotator.cs

```
using UnityEngine;
using System.Collections;

public class Rotator : MonoBehaviour {

      //public float speed = 10f;
      void Update ()
      {
            transform.Rotate (new Vector3 (15, 30, 45) * Time.deltaTime);
            //transform.Rotate(Vector3.up, speed * Time.deltaTime);
            //transform.Rotate(0,20*Time.deltaTime,0);
            //transform.Rotate(0, Time.deltaTime * speed , 0, Space.Self);
            //transform.RotateAround(transform.position, transform.up,
Time.deltaTime * 90f);
      }
}
```

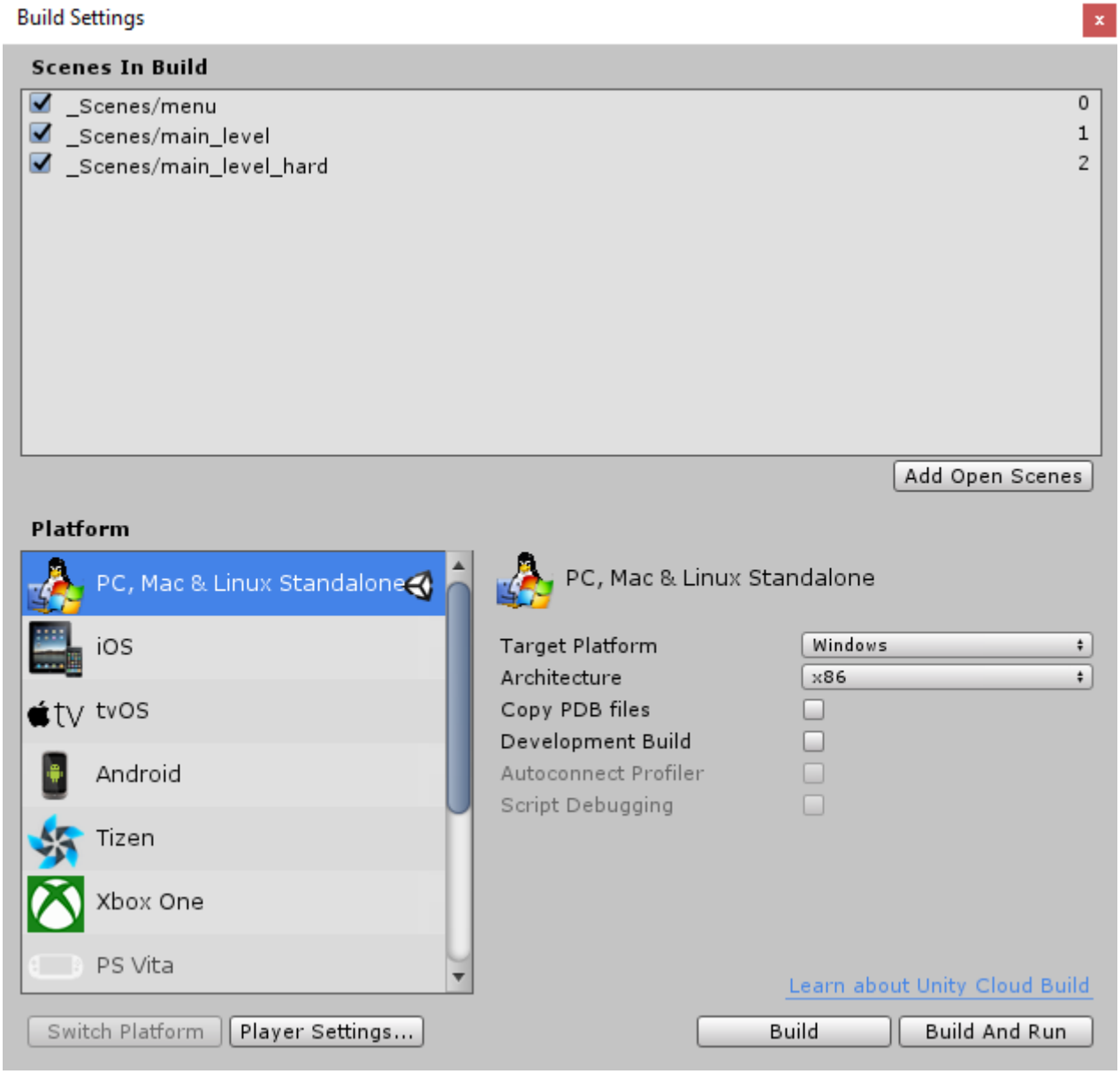
Σε αυτό το στάδιο μπορούμε με χαρά να δηλώσουμε ότι βασικό σκέλος του παιχνιδιού μας έχει ολοκληρωθεί. Μένει μόνο να τροποποιήσουμε αυτή τη σκηνή για να είναι κατάλληλη για το δύσκολο επίπεδο. Αρχικά θα κάνουμε αποθήκευση ενός αντίγραφου της σκηνής με το όνομα main_level_hard. Επειδή τα script μας αναλαμβάνουν όλες τις ενέργειες που πρέπει να γίνουν στο δύσκολο επίπεδο, το μόνο που πρέπει να αλλάξουμε είναι να προσθέσουμε περισσότερους κακούς στόχους στο πλέγμα, να απενεργοποιήσουμε τους αντίστοιχους καλούς στόχους και να εντάξουμε τους νέους κακούς στόχους στα Inspector των VideoWall για το script random_spawn. Η τελική εικόνα του πλέγματος πρέπει να είναι η ακόλουθη:



Εικόνα 2.9.viii: Δομή πλέγματος για τα spawn του Hard επιπέδου

Σημείωση: πρέπει να αναφερθεί ότι αυτή η σχεδίαση με τις δύο διαφορετικές πίστες δεν είναι ούτε δεσμευτική ούτε η απόλυτα βέλτιστη εκδοχή της εφαρμογής. Φυσικά και θα μπορούσε να χρησιμοποιηθεί μόνο ένα main_level αντί για δύο και απλά να γινόταν διαφορετική αντιστοίχιση των καλών/κακών στόχων στο script των spawn. Ήταν καθαρά μια σχεδιαστική επιλογή διότι είχε πιο καθαρό και οργανωμένο οπτικό αποτέλεσμα μεταξύ των σκηνών δημιουργώντας μια φυσική διαφορά μεταξύ ενός εύκολου και δύσκολου επιπέδου. Επίσης, οι δύο διαφορετικές σκηνές μπορούν να εξελιχθούν τελείως διαφορετικά δίνοντας το ενδεχόμενο για μεγαλύτερη δημιουργική παρέμβαση σε μελλοντικές εκδοχές της εφαρμογής. Για παράδειγμα, ίσως έναν AI (Artificial Intelligence) αντίπαλο για ένα Versus Game Stage.

Για να μπορεί να λειτουργεί και μέσω του κεντρικού μενού του παιχνιδιού και αυτή η σκηνή, πρέπει για ακόμα μια φορά να την εντάξουμε στο Build του παιχνιδιού κάνοντας File → Build Settings… και να κάνουμε Add Open Scene επιλέγοντας το main_level_hard να ακολουθεί τις δύο προ-υπάρχουσες πίστες. Ακολουθεί εικόνα που δείχνει πως θα πρέπει να οριστεί το τελικό Build για να μπορούμε να εξάγουμε το παιχνίδι μας ως αυτόνομη εφαρμογή.



Εικόνα 2.9.ix: Τελικό Build για να μπορούμε να τρέχουμε πλήρως το παιχνίδι μας

Ακολουθεί η πρώιμη εκδοχή του random_spawn script που απορρίφθηκε για λόγους Performance:

random_spawn.cs

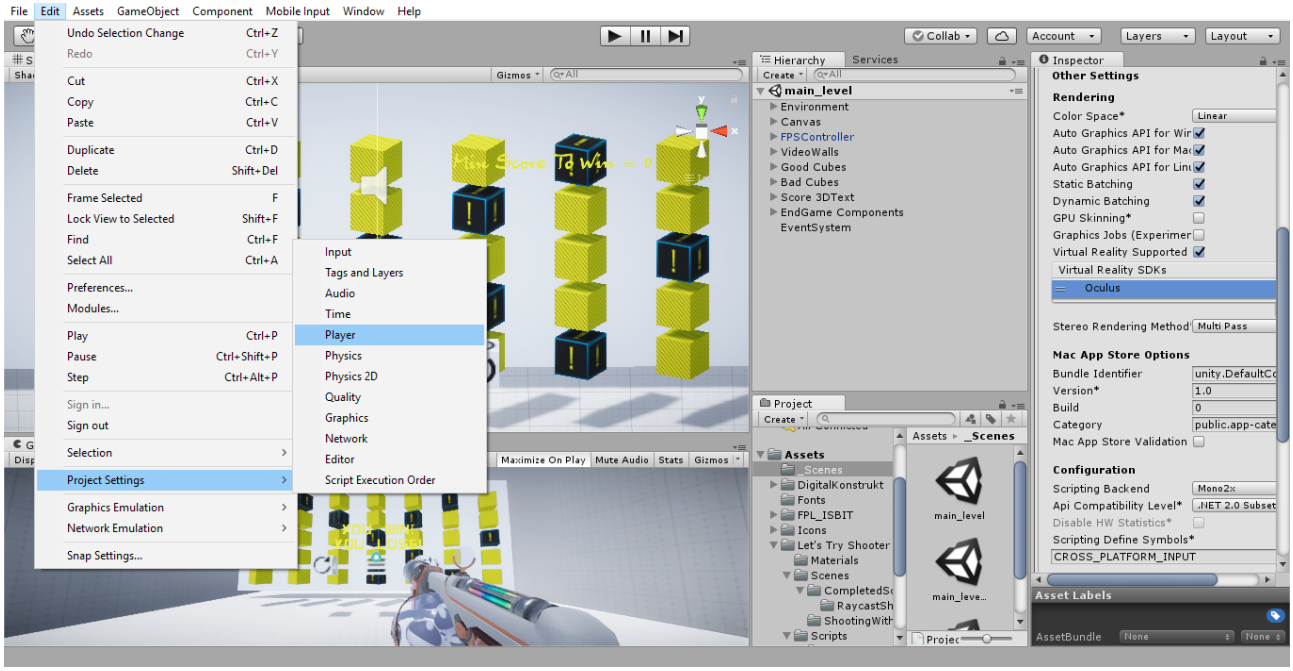
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;
using UnityEngine.UI;

public class random_spawn : MonoBehaviour {

      public int whichCube;
      public int timeAlive;
      public int killTime;
      public GameObject good;
      public GameObject bad;
```

```
	public GameObject goodCube;
	public GameObject badCube;

	AudioSource audioSource;

	void Start ()
	{
		//we will select the instance of AudioProcessor and pass a
reference to this object
		AudioProcessor processor = FindObjectOfType<AudioProcessor> ();
		processor.onBeat.AddListener (onOnbeatDetected);
		processor.onSpectrum.AddListener (onSpectrum);

		//we need to connect the audioSource of the object this script is
attached to
		audioSource = GetComponent<AudioSource>();

		//we deactive the objects to make them invisible to the player
		good.SetActive(false);
		bad.SetActive(false);
		//goodCube.SetActive(false);
		//badCube.SetActive(false);
	}

	//this event will be called every time a beat is detected
	//you can change the threshold parameter in the inspector to adjust the
sensitivity
	void onOnbeatDetected ()
	{
		//in this version of the code we make instances of the parent
cubes and
		//keep them alive for a set period of time adding more instances
on each beat
		whichCube = Random.Range(0, 9); //0 to 9 -> 0-6 for good/7-9 for
bad

		timeAlive = Random.Range(3, 6); //how much time the object will be
active in seconds
		killTime = (int) Time.time + timeAlive;

		//find a random position in front of the VideoWall
		int x = Random.Range (-6, 6);
		int y = Random.Range (1, 5);
		float z = 4.125;

		//yield return new WaitForSeconds(time);
		foreach (Transform child in transform) {
			if (whichCube >= 0 && whichCube <=6)
			{
				//good
				//transform
				//GameObject good = Instantiate (goodCube,
child.transform.position, child.transform.rotation) as GameObject;
				//loacation is a Vector3(x, y, z)
				GameObject good = Instantiate (goodCube, Vector3(x, y,
z), child.transform.rotation) as GameObject;
				good.transform.parent = child;
```

```
                        yield return new WaitForSeconds(killTime);
                        Destroy (GameObject);
                  }
                  else
                  {
                        //bad
                        //transform
                        GameObject bad = Instantiate (badCube, Vector3(x, y,
z), child.transform.rotation) as GameObject;
                        bad.transform.parent = child;
                        yield return new WaitForSeconds(killTime);
                        Destroy (GameObject);
                  }
                  //AudioSource.PlayClipAtPoint(clip, transform.position);

            }

      }

      //This event will be called every frame while music is playing
      void onSpectrum (float[] spectrum)
      {
            //The spectrum is logarithmically averaged to 12 bands
            for (int i = 0; i < spectrum.Length; ++i) {
                  Vector3 start = new Vector3 (i, 0, 0);
                  Vector3 end = new Vector3 (i, spectrum [i], 0);
                  Debug.DrawLine (start, end);
            }
      }
}
```

2.10 Αλλάζοντας το Rendering για VR Compatibility

Σε όλη τη διάρκεια του project αναφερόμασταν σε VR Compatibility αλλά έχουμε φτάσει στο τέλος της εφαρμογής και δεν έχουμε προσθέσει ακόμα κάποιο επιπλέον στοιχείο για την διευθέτηση του ζητήματος. Στην πραγματικότητα όμως, σε μια desktop εφαρμογή που απλά θέλουμε να αλλάζουμε το Viewpoint της κάμερας (αντί με το ποντίκι) με το Gyroscope της μάσκας VR, χρειάζεται απλά η αλλαγή του τελικού Rendering σε στερεοσκοπικό.
Στο VR Compatible rendering, ελέγχεται εάν υπάρχει συνδεμένο Oculus στον υπολογιστή και αλλάζει την εξαγόμενη εικόνα της εκάστοτε σκηνής σε δύο διαφορετικές, μια για κάθε μάτι, με την αντίστοιχη γωνία όρασης που προσφέρει την πλήρη οπτική του χώρου από το σημείο εκκίνησης του χαρακτήρα. Με κάθε κίνηση του κεφαλιού κοιτάζουμε και σε ένα διαφορετικό σημείο της σκηνής, ακριβώς σαν να δουλεύαμε την κάμερα με ένα ποντίκι.
Η Unity υποστηρίζει πλήρως εξαγωγή project σε Oculus Rift VR Headset που ήταν και ο στόχος μας. Απλά απαιτείται να προστεθεί στις σκηνές του παιχνιδιού η επιλογή Virtual Reality Supported και το SDK για το Oculus. Η ρύθμιση προσθέτεται εύκολα επιλέγοντας Edit → Project Settings → Player → Other Settings → Virtual Reality Supported και Virtual Reality SDKs = Oculus.



Εικόνα 2.10.i: Προβολή του σημείου ρύθμισης της κάθε σκηνής για VR Compatibility

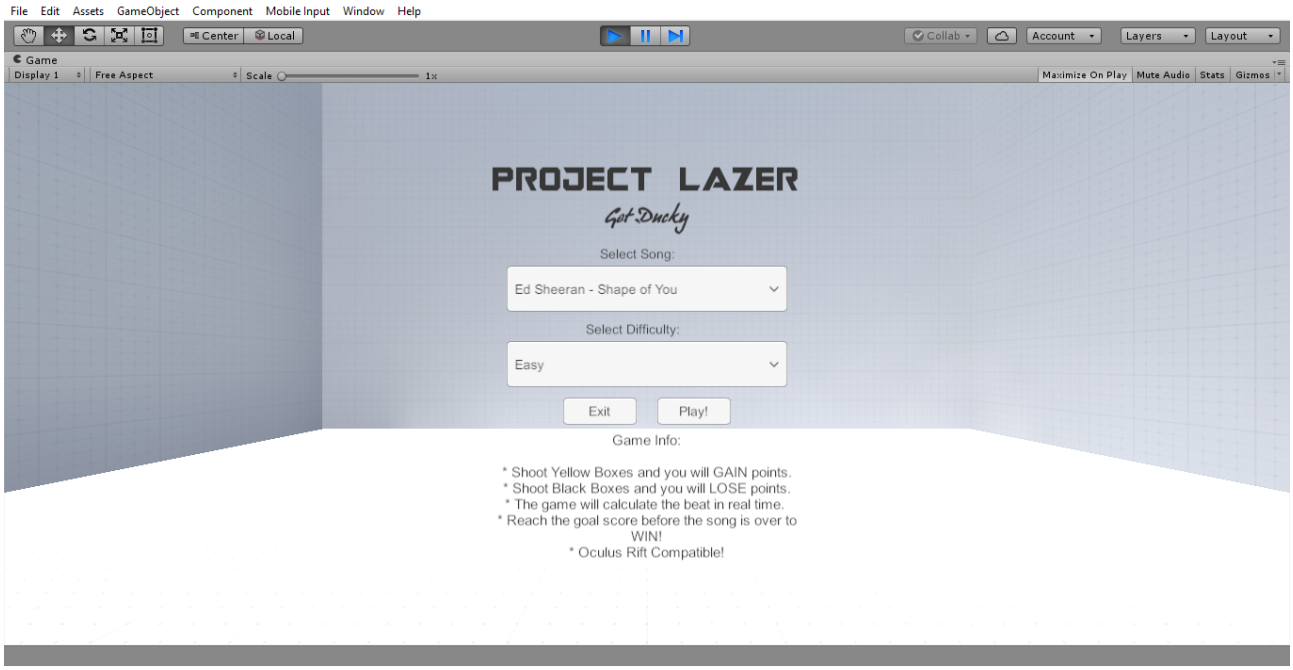
Σε περίπτωση που θέλουμε να κάνουμε το παιχνίδι μας VR Compatible με το Headset HTC Vive, τότε η διαδικασία που πρέπει να ακολουθήσουμε είναι διαφορετική. Η συγκεκριμένη μάσκα εικονικής πραγματικότητας λειτουργεί σε συνεργασία με Steam Powered Software οπότε πρέπει να προσαρμοστούμε κατάλληλα.
Αρχικά θα πρέπει να δημιουργηθεί Steam Account για να έχουμε την δυνατότητα να κατεβάσουμε το Steam VR Runtime Software στον υπολογιστή που θα δοκιμάσουμε το παιχνίδι. Στην συνέχεια, πρέπει να πάμε στο ίδιο μας το project και να βάλουμε ένα συμβατό Asset με το Runtime Software έτσι ώστε να επικοινωνούν μεταξύ τους. Το πακέτο που χρειαζόμαστε είναι το SteamVR Plugin της Valve Corporation που διατίθεται δωρεάν στο Asset Store. Εφόσον το κατεβάσουμε και το εισάγουμε στο παιχνίδι μας, πρέπει να χρησιμοποιήσουμε την κάμερα του πακέτου στην θέση της υπάρχουσας κάμερας του παιχνιδιού. Προσοχή να γίνει απενεργοποίηση της παλιάς κάμερας για να μην υπάρχει πρόβλημα στο εξαγόμενο οπτικό feed.
Σε κάθε περίπτωση, είναι σημαντικό να αναφέρουμε ότι πολλές φορές το οπτικό αποτέλεσμα σε ένα VR Headset και στην οθόνη ενός υπολογιστή υπάρχει πιθανότητα να μην είναι απόλυτα ίδιο οπότε

συνίσταται να γίνουν δοκιμές για τις ρυθμίσεις φωτισμού και θέσεων των αντικειμένων της σκηνής πριν την τελική έκδοση του παιχνιδιού.

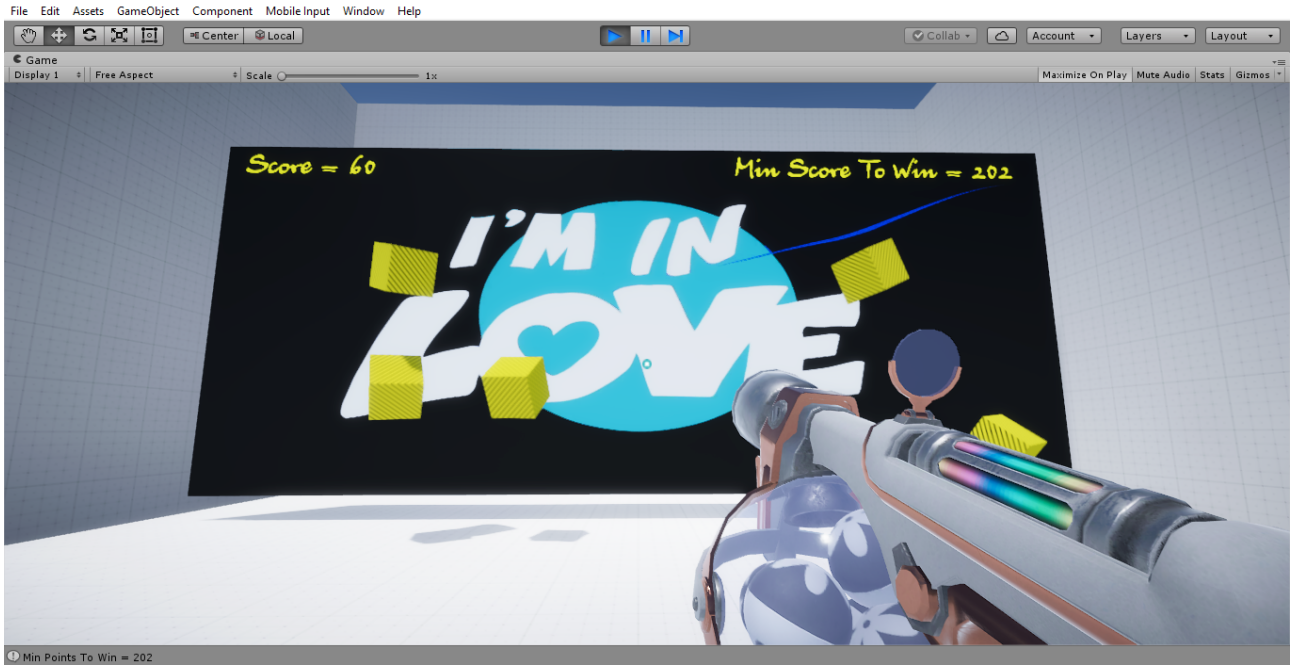
Εικόνα 2.10.ii: SteamVR Plugin για συμβατότητα με το Headset HTC Vive

2.11 Τελική Εικόνα του Παιχνιδιού

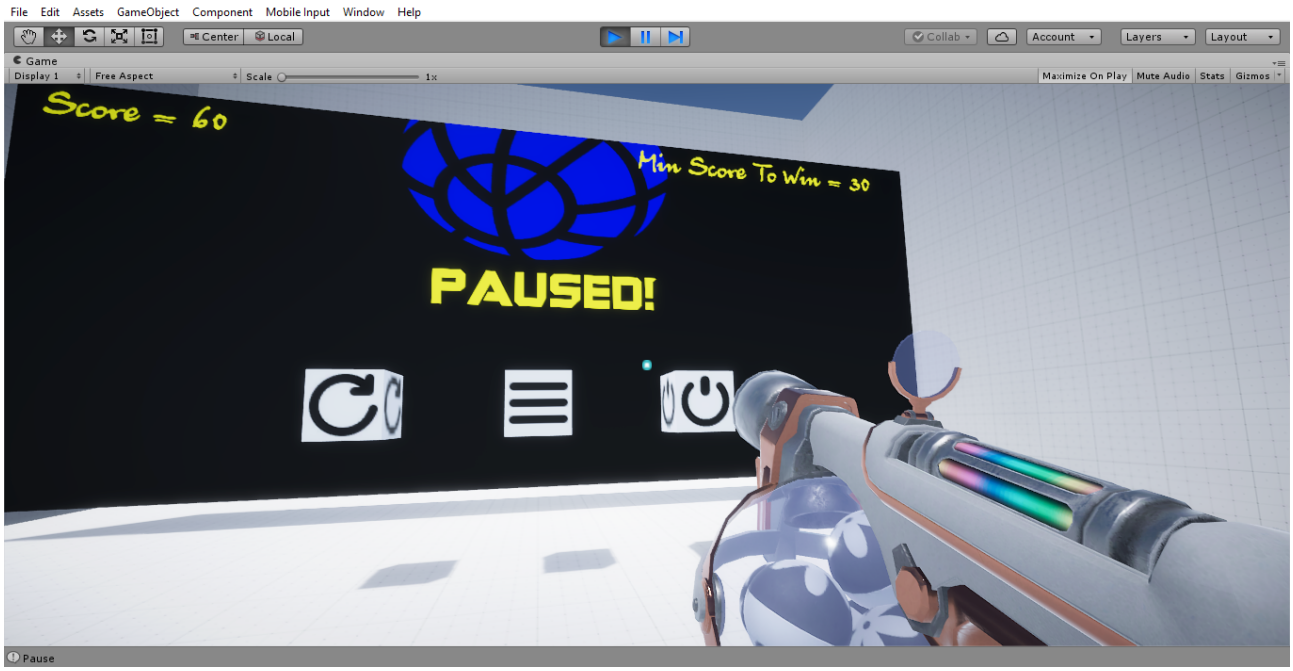
Τώρα που έχουμε φτάσει στο τέλος της υλοποίησης του παιχνιδιού μας, ας παραθέσουμε μερικά στιγμιότυπα από το λογισμικό εν ώρα λειτουργίας σε διάφορα στάδια χρήσης του. Ενδεικτικά θα δείξουμε το κεντρικό μενού που υποδέχεται τον παίχτη, πώς φαίνεται το παιχνίδι μας εν ώρα λειτουργίας με την δυναμική αλλαγή στόχων στον ρυθμό της μουσικής, πώς είναι το βοηθητικό μενού παύσης του παιχνιδιού και το αποτέλεσμα που βλέπει ο παίχτης εάν πετύχει ή όχι το ελάχιστο score για να κερδίσει το παιχνίδι.



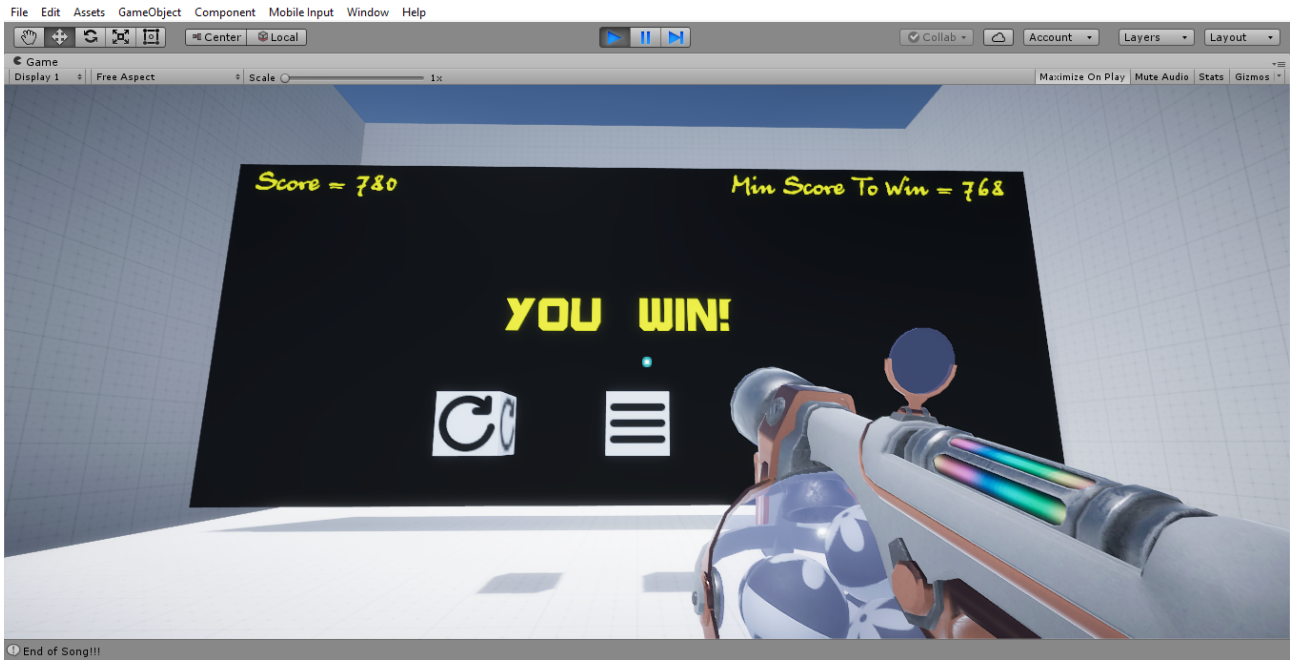
1) Main Menu για επιλογή Τραγουδιού και Επιπέδου Δυσκολίας του παιχνιδιού



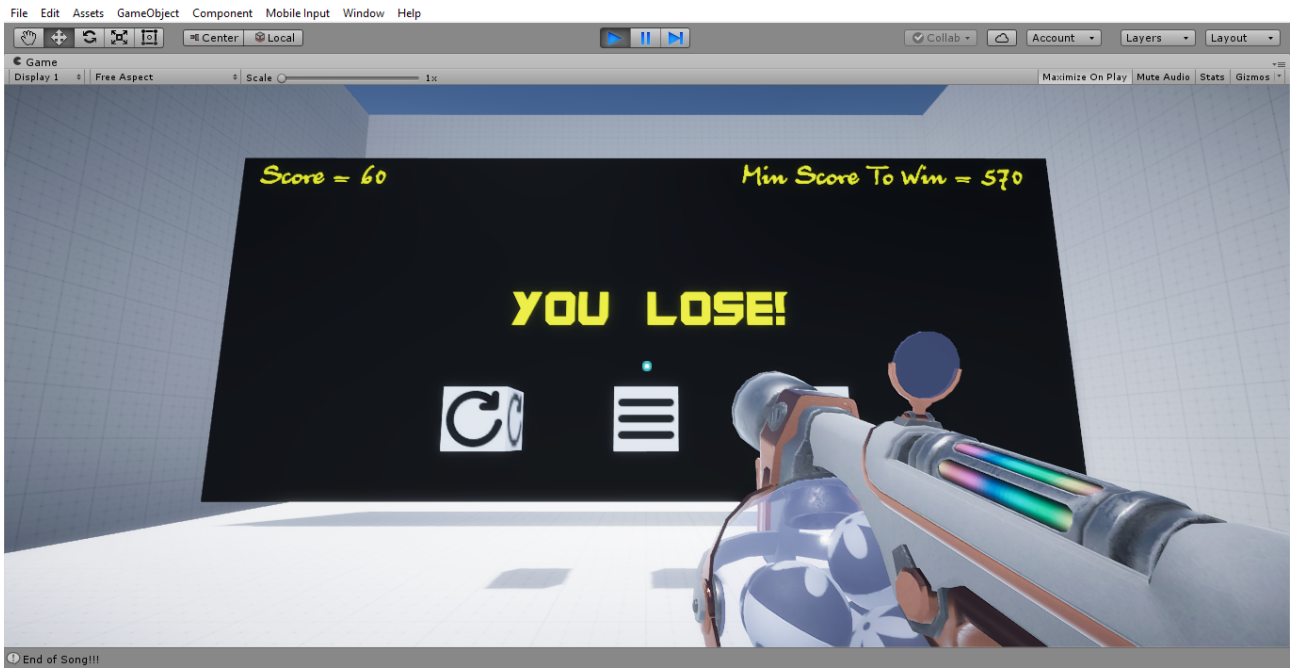
2) Στιγμιότυπο από το Easy Επίπεδο εν ώρα παιχνιδιού



3) Το παιχνίδι σε παύση



4) Το παιχνίδι όταν έχουμε κερδίσει, δηλαδή το Score μας είναι πάνω από το min όριο

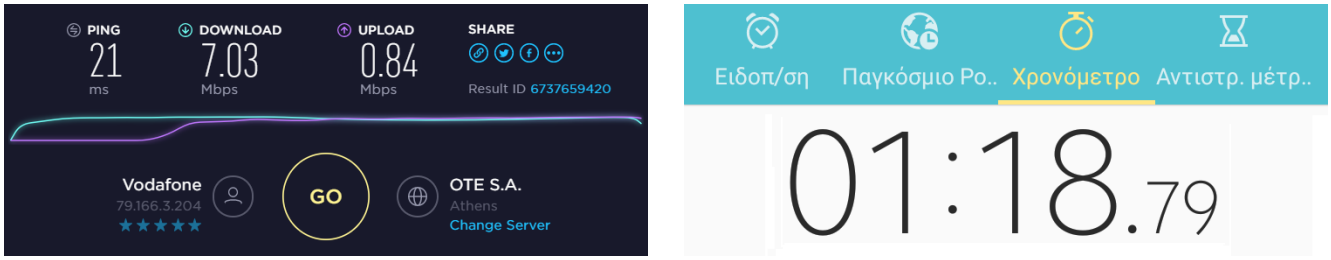


5) Το παιχνίδι όταν δεν έχουμε καταφέρει να φτάσουμε το επιθυμητό Score

Μελλοντικές Εξελίξεις της Εφαρμογής

Σε αυτό το σημείο, έχουμε φτάσει επίσημα στο τέλος της υλοποίησης μας. Όμως, θα ήθελα πριν κλείσουμε την ανάλυση αυτή, να μιλήσουμε για μερικές ιδέες που υπήρχαν κατά την ανάπτυξη του παιχνιδιού αλλά δυστυχώς δεν βρήκαν πρακτική εφαρμογή λόγω τεχνικών δυσκολιών και προβλημάτων της πλατφόρμας. Οι ιδέες αυτές με συντρόφεψαν κατά τη διάρκεια της ανάπτυξης του παιχνιδιού και παρέμειναν στο μυαλό μου ως βελτιώσεις της δομής, του gameplay και του περιβάλλοντος σε μελλοντικές εκδόσεις αυτού. Οι τεχνικές που επιθυμώ να μελετήσω εκτενέστερα και να υλοποιήσω εφόσον υπάρχει δυνατότητα είναι οι εξής:

- Να βρεθεί τρόπος να ενταχθεί online streaming (on player demand) για όποιο μουσικό video θέλει να δοκιμάσει ο παίχτης με seamless αποτέλεσμα και χωρίς καθυστερήσεις λόγω buffering, είτε από δικό μας Server είτε μέσω Youtube. Οι τεχνικές για επίτευξη αυτών των δύο ιδεών είναι εκ διαμέτρου διαφορετικές καθώς σε ένα Server έχουμε απευθείας access σε ένα αρχείο video ενώ μέσω Youtube πρέπει να κάνουμε fetch τα Multimedia στοιχεία της ιστοσελίδας καθώς τα αρχεία είναι προστατευμένα. Χαρακτηριστικά στα πλαίσια δοκιμών, δοκιμάστηκε να υλοποιηθεί το Online Streaming με πρόσβαση σε Server με δυστυχώς πολύ άσχημα αποτελέσματα στον τομέα του performance. Πέρα από την τεχνική δυσλειτουργία της πλατφόρμας να προβάλει διαφορετικά video σε ένα videowall, δοκιμάσαμε την υλοποίηση εξαιρώντας απλά το VR Compatibility. Το αποτέλεσμα ήταν αποθαρρυντικό καθώς κάθε video απαιτούσε σημαντικό χρόνο loading για να προβληθεί τοπικά. Ενδεικτικά, χρονομετρήθηκε το ακριβές χρονικό διάστημα που χρειάστηκε για να κατέβει το δοκιμαστικό, συμβατό με την πλατφόρμα video “Cosmos Reel” (τύπου .ogv) (http://unityinstitute.esy.es/TutorialAssets/CosmosReel.ogv), διάρκειας 3:29 λεπτών (όσο ένα μέσο μουσικό κομμάτι), και η καθυστέρηση για το buffering ήταν 1:18.79. Ο συγκεκριμένος χρόνος είναι αδικαιολόγητος ακόμα και για loading περίτεχνων σκηνών σε Role Playing Games όπως το Witcher III. Η ταχύτητα πρόσβασης στο διαδίκτυο του τερματικού σε συνθήκες δοκιμών ήταν η εξής:



Εικόνα 3.0: Ταχύτητα πρόσβασης στο διαδίκτυο και χρόνος buffering για Online Streaming των Video

Ενδεικτικά παραθέτω και τον κώδικα που δοκιμάστηκε για να είναι πλήρως κατανοητή η τεχνική που εφαρμόστηκε για την πρόσβαση των αρχείων στον Server.

PlayVideoFromOnlineSource.cs

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

public class PlayVideoFromOnlineSource : MonoBehaviour {

	#if UNITY_STANDALONE || UNITY_EDITOR || UNITY_WEBPLAYER
	public string urlVideo =
"http://unityinstitute.esy.es/TutorialAssets/CosmosReel.ogv";
```

```
	public MovieTexture	movieTexture;
	public Text progresion;
	protected bool streamReady = false;
	AudioSource		sound;

	void Start()
	{
		//progression text indicating when the video will be stream ready
		//initially blank but when space is pressed it begins to increase
		//until it reaches the maximum of 100% (video will begin to play)
		progresion.text = "";
	}

	void Update()
	{
		//if the player pressed the space button and the video is ready to
stream
		if (Input.GetKeyDown ("space") && !streamReady) {
			//start streaming the video
			StartCoroutine (StartStream (@urlVideo));
		}
	}

	protected IEnumerator StartStream (string url)
	{
		WWW videoStreamer = new WWW (url);

		//while we buffer the video, the progress indicator will gradually
increase
		//when the indicator has reached 100% it is ready to play and we
will be able
		//to use the streamed video as a texture to the object the script
is on
		while (videoStreamer.isDone == false) {
			if(progresion	!=	null)	progresion.text	=	(int)(100.0f	*
videoStreamer.progress) + "%";
			yield return 0;
		}
		movieTexture = videoStreamer.GetMovieTexture();
		while (!movieTexture.isReadyToPlay) {
			yield return 0;
		}

		//we also need to seperate the audioclip from the video in order
to play it
		sound = GetComponent<AudioSource> ();
		sound.clip = movieTexture.audioClip;
		//we set the streamed video as the texture of the object
		GetComponent<MeshRenderer> ().material.mainTexture = movieTexture as
MovieTexture;

		//when everything is ready to play, we start playing the sound and
the video at the same time
```

```
            //we also delete the progresion indicator since it is no longer
required
            if (videoStreamer.isDone) {
                  progresion.text = "";
                  sound.Play ();
                  movieTexture.Play ();
                  streamReady = true;
            }
      }
      #endif
      }
```

- Σε περίπτωση που επιτυγχάναμε ένα ικανοποιητικό αποτέλεσμα με τους χρόνους buffering των video και φυσικά διορθώνονταν το πρόβλημα της πλατφόρμας με την εναλλαγή video σε ένα αντικείμενο, μια πολύ ενδιαφέρουσα προσθήκη θα ήταν να έχει το παιχνίδι Online Multiplayer Feature. Η υλοποίηση θα επιτρέπει σε παραπάνω από έναν παίχτες να παίζουν ταυτόχρονα το ίδιο τραγούδι και στο τέλος θα συγκρίνει το score τους για να κρίνει τον νικητή.

- Επίσης, θα ήταν πολύ ενδιαφέρον να δοκιμάσουμε να προσαρμόσουμε τα controls του παιχνιδιού από ποντίκι και πληκτρολόγιο (όπως ένα τυπικό FPS) σε VR Gun Controller για να έχει τη συνολική αίσθηση του Duck Hunt αλλά μέσα από το VR.

- Θα ήταν επίσης μια ενδιαφέρουσα εκδοχή να προσαρμόσουμε αυτό το παιχνίδι για mobile devices αλλάζοντας εξ’ ολοκλήρου τα controls και τον τρόπο σχεδίασης της εφαρμογής και του κώδικα για ακόμα πιο lightweight αποτέλεσμα. Η συγκεκριμένη ιδέα σίγουρα απαιτεί την λειτουργικότητα του Online Streaming καθώς δεν θα μπορούμε να κρατάμε native στην συσκευή τα μουσικά video.

- Να δοκιμαστεί να υλοποιηθεί η εφαρμογή σε διαφορετική πλατφόρμα που δεν θα αντιμετωπίζει τις τεχνικές δυσλειτουργίες που αντιμετωπίστηκαν σε αυτή την εργασία. Επιπλέον, είναι σημαντικό να κάνουμε αυτές τις δοκιμές και σε υπολογιστή με μεγαλύτερες δυνατότητες. Θα μας επέτρεπε την χρήση λεπτομερέστερων μοντέλων και σκηνών κατάλληλων ακόμα και για animated sequences.

Όπως καταλαβαίνουμε, πρόκειται για ένα μεγάλο εύρος αλλαγών αλλά θεωρώ πώς αν επιτύχουμε στην εκτέλεση αυτών, το τελικό αποτέλεσμα δεν θα προσεγγίσει απλά μια επαγγελματική εφαρμογή αλλά μια πραγματικά καινοτόμα εκδοχή των Rhythm Games δίνοντας την δυνατότητα στους παίχτες όχι μόνο να επιλέγουν οι ίδιοι με ποιο τραγούδι θα δοκιμάσουν τις ικανότητές τους αλλά να προκαλέσουν και φίλους στους, ιδέες που μέχρι σήμερα δεν έχει αποπειραθεί κανένα Development Studio.

Βιβλιογραφία

Γενική Βιβλιογραφία

Unity 3D (2017). Unity User Manual (2017.2). Ανακτήθηκε στις 14 Ιανουαρίου 2017 από: https://docs.unity3d.com/Manual/index.html

Unity 3D (2017). AR/VR (XR) Discussion. Ανακτήθηκε στις 10 Απριλίου 2017 από: https://forum.unity.com/forums/ar-vr-xr-discussion.80/

MathWorks (2017). Fourier Transforms. Ανακτήθηκε στις 26 Ιανουαρίου 2017 από: https://www.mathworks.com/help/matlab/math/fourier-transforms.html

MathWorks (2017). FFT for Spectral Analysis. Ανακτήθηκε στις 26 Ιανουαρίου 2017 από: https://www.mathworks.com/help/matlab/examples/fft-for-spectral-analysis.html

Ashcraft, Brian (2008). Arcade Mania! The Turbo-Charged World of Japan's Games Centers. Kodansha International Publishing House.

Rollings, Andrew & Adams, Ernest (2006). Fundamentals of Game Design. Prentice Hall Publishing House.

Steinberg, Scott (2011). Music Games Rock. Power Play Publishing House.

Ravyse, Werner Siegfried (2016). Success factors for serious games to enhance learning: a systematic review. Ανακτήθηκε στις 8 Ιανουαρίου 2017 από: https://dspace.nwu.ac.za/handle/10394/17123

Knoeferle, Klemens (2016). When brands come to life: Experimental research on the vividness effect of Virtual Reality in transformational marketing communications. Ανακτήθηκε στις 3 Ιουνίου 2017 από: https://cris.vub.be/files/34677982/Manuscript_VR_authors_copy.pdf

Isaac, Joseph (2016). Step into a new world - Virtual Reality (VR) (Basic Concepts of Virtual Reality along with Research Challenges explained in simple words). Ανακτήθηκε στις 12 Ιουλίου 2017 από: https://www.completegate.com/2016070154/blog/virtual-reality-explained

Mailman Joshua (2013). Improvising Synesthesia: Comprovisation of Generative Graphics and Music. Ανακτήθηκε στις 7 Μαρτίου 2017 από: http://www.leoalmanac.org/wp-content/uploads/2013/07/LEAVol19No3-Mailman.pdf

Huilberts, Sander (2010). Captivating Sound, The Role of Audio for Immersion in Computer Games. Ανακτήθηκε στις 10 Φεβρουαρίου 2017 από: http://download.captivatingsound.com/Sander_Huiberts_CaptivatingSound.pdf

Chen, Chun Han & Lo, Chien Shun (2016). The Development of a Music Rhythm Game with a Higher Level of Playability. Ανακτήθηκε στις 23 Ιουλίου 2017 από: http://ieeexplore.ieee.org/abstract/document/7840256/?reload=true

Yi, Joshua & Lai, Jordan (2017). Virtual Reality Rhythm Game. Ανακτήθηκε στις 26 Αυγούστου 2017 από: http://scholarcommons.scu.edu/cseng_senior/97/

Hobb, Toby & Fisher, Jolene. Examination of the Relationship Between Gender, Performance, and Enjoyment of a First-Person Shooter Game. Ανακτήθηκε στις 27 Απριλίου 2017 από: http://journals.sagepub.com/doi/abs/10.1177/1046878117693397

Πηγές Εικόνων Κειμένου

PlayStation Store (2017): Project Diva: Future Tone Screenshots. Ανακτήθηκε στις 14 Μαΐου 2017 από: https://store.playstation.com/#!/en-us/games/hatsune-miku-project-diva-future-tone/cid=UP0177-CUSA06093_00-PROJECTDIVA00000

PlayStation (2017): Parappa The Rapper. Ανακτήθηκε στις 15 Μαΐου 2017 από: https://www.playstation.com/en-us/games/parappa-the-rapper-remastered-ps4/

Unilad (2017): Guitar Hero. Ανακτήθηκε στις 15 Μαΐου 2017 από: https://www.unilad.co.uk/video/footage-of-a-cancelled-guitar-hero-mmo-surfaces-online/

Nintendo (2014): Duck Hunt. Ανακτήθηκε στις 16 Μαΐου 2017 από: http://www.nintendolife.com/news/2014/11/the_original_nes_duck_hunt_is_coming_soon_to_the_virtual_console_on_wii_u

Unity 3D (2017). Raycast Examples. Ανακτήθηκε στις 11 Ιανουαρίου 2017 από: https://unity3d.com/learn/tutorials/projects/lets-try-assignments/lets-try-shooting-raycasts-article

Πηγές Πληροφοριών για τον Κώδικα

Unity Docs (2017). Movie Texture. Ανακτήθηκε στις 28 Ιανουαρίου 2017 από: https://docs.unity3d.com/Manual/class-MovieTexture.html

Unity Answers (2017). Texture Mapping. Ανακτήθηκε στις 4 Μαρτίου 2017 από: http://answers.unity3d.com/questions/157391/why-do-textures-map-upside-down.html

Unity Answers (2017). Random Range. Ανακτήθηκε στις 7 Ιουνίου 2017 από: http://answers.unity3d.com/questions/233436/random-issue.html

Unity Answers (2017). New Dropdown Menu. Ανακτήθηκε στις 11 Ιουνίου 2017 από: http://answers.unity3d.com/questions/1066135/new-dropdown-menu-sample.html

Unity Answers (2017). Referencing Non-Static Variables. Ανακτήθηκε στις 27 Απριλίου 2017 από: http://answers.unity3d.com/questions/42843/referencing-non-static-variables-from-another-scri.html

Unity Answers (2017). Reach Variables in Other Scenes. Ανακτήθηκε στις 27 Απριλίου 2017 από: http://answers.unity3d.com/questions/969403/how-to-reach-a-variable-in-another-scene.html

Unity Answers (2017). Set Destruction Time in Object Instantiate. Ανακτήθηκε στις 18 Ιουνίου 2017 από: http://answers.unity3d.com/questions/1034351/instantiate-an-object-and-destroy-it-after-given-t.html

Unity Forum (2008). Blurry 3D Text. Ανακτήθηκε στις 27 Ιουλίου 2017 από: https://forum.unity.com/threads/3d-text-always-blurry.18712/

Unity Tutorials (2017). The Video Player Component. Ανακτήθηκε στις 10 Αυγούστου 2017 από: https://unity3d.com/learn/tutorials/topics/graphics/videoplayer-component

Unity Answers (2017). Reset Time. Ανακτήθηκε στις 18 Αυγούστου 2017 από: http://answers.unity3d.com/questions/728588/reset-the-timetime-back-to-00.html