ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΝΑΠΤΥΞΗ ΚΑΙ ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΧΡΗΣΗ ΤΗΣ ΜΗΧΑΝΗΣ UNITY

ΚΑΡΑΒΑΣΙΛΗΣ ΛΟΥΚΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
Διδάσκων

Λαμία 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΝΑΠΤΥΞΗ ΚΑΙ ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΧΡΗΣΗ ΤΗΣ ΜΗΧΑΝΗΣ UNITY

ΚΑΡΑΒΑΣΙΛΗΣ ΛΟΥΚΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
Διδάσκων

Λαμία 2025

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

GAME DEVELOPMENT AND DESIGN USING UNITY ENGINE

KARAVASILIS LOUKAS

FINAL THESIS

ADVISOR

KOLOMVATSOS KONSTANTINOS
Professor

Lamia 2025

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί π****αράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 19/9/2025

Ο Δηλών

ΚΑΡΑΒΑΣΙΛΗΣ ΛΟΥΚΑΣ

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Το παιχνίδι υπήρξε διαχρονικά αναπόσπαστο στοιχείο της ανθρώπινης φύσης. Με την αλματώδη πρόοδο της τεχνολογίας, η ψυχαγωγία εξελίχθηκε σε ψηφιακή μορφή, οδηγώντας στην δημιουργία των βιντεοπαιχνιδιών, τα οποία σήμερα αποτελούν την πλέον διαδεδομένη μορφή παιχνιδιού. Η μεγάλη ποικιλία τους έχει οδηγήσει στην κατηγοριοποίηση τους, ενώ η ανάπτυξη και υλοποίηση τους καθίσταται εφικτή μέσω εξειδικευμένων λογισμικών, γνωστών ως μηχανές ανάπτυξης βιντεοπαιχνιδιών.

Στο πλαίσιο της παρούσας πτυχιακής εργασίας πραγματοποιείται αρχικά μια σύντομη ιστορική αναδρομή της μηχανής Unity.Παρουσιάζονται τα στάδια σχεδιασμού και ανάπτυξης ενός βιντεοπαιχνιδιού και εν συνεχεία αναλύεται η διαδικασία δημιουργίας του βιντεοπαιχνιδιού Ghost Seekers το οποίο αναπτύχθηκε στο πλαίσιο της εργασίας.

ABSTRACT

Playing Games has always been an integral part of human nature. With the rapid advancement of technology, entertainment has evolved into a digital form, leading to the creation of video games, which today constitute the most widespread form of playing. Their great variety has resulted in their categorization, while their development and implementation are made possible through specialized software, known as Game Engines.

Withing the framework of this thesis, a brief historical overview of the Unity Engine is initially presented. The stages of designing and developing a video game are then outlined, followed by an analysis of the creation process of the video game ‘Ghost Seekers’, which was developed as part of this thesis.

Table of Contents

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Ιστορία των Βιντεοπαιχνιδιών

Η Ιστορία των βιντεοπαιχνιδιών ξεκίνησε την δεκαετία του 1950 και του 1960, όταν επιστήμονες άρχισαν να σχεδιάζουν απλά παιχνίδια σε μίνι-υπολογιστές και κεντρικούς υπολογιστές. Το πρώτο παιχνίδι που δημιουργήθηκε ήταν το Spacewar! ,για τον υπολογιστή PDP-1, το οποίο αναπτύχθηκε από φοιτητές του Ινστιτούτου Τεχνολογίας της Μασαχουσέτης (MIT), και αποτέλεσε ένα από τα πρώτα παιχνίδια που χρησιμοποιούσαν οθόνη.
Στις αρχές του 1970 κυκλοφόρησε το πρώτο υλικό βιντεοπαιχνιδιών καθώς και η πρώτη οικιακή κονσόλα βιντεοπαιχνιδιών Magnavox Odyssey, ενώ τα πρώτα arcade παιχνίδια ήταν το Computer Space και Pong.

Στα μέσα της δεκαετίας του 1970, οι χαμηλού κόστους προγραμματιζόμενοι επεξεργαστές αντικατέστησαν τα πρώιμα κυκλώματα λογικής από τρανζίστορ, ενώ κυκλοφόρησαν οι πρώτες κασέτες ROM για οικιακές κονσόλες, συμπεριλαμβάνοντας το Atari Video Computer (VCS).

Το 1983 λόγω της κυκλοφορίας μεγάλου αριθμού παιχνιδιών, συχνά χαμηλής ποιότητας ή αντιγραφών, υπήρξε κρίση στις Ηνωμένες Πολιτείες με αποτέλεσμα να πάρει ηγετικό ρόλο στην βιομηχανία βιντεοπαιχνιδιών η Ιαπωνία, όπου το 1985 η Nintendo κυκλοφόρησε στις Ηνωμένες Πολιτείες της Αμερικής το Nintendo Entertainment System, συμβάλλοντας στην ανάκαμψη της αγοράς βιντεοπαιχνιδιών.

Την δεκαετία του 1990 εμφανίστηκαν οι πρώτες φορητές συσκευές με πρωτοπόρο το Gameboy.

Στις αρχές της δεκαετίας 1990 η εξέλιξη της τεχνολογίας των μικροεπεξεργαστών επέτρεψε την ανάπτυξη γραφικών τρισδιάστατης απόδοσης σε πραγματικό χρόνο τόσο στις κονσόλες όσο και στους υπολογιστές με την βοήθεια από κάρτες γραφικών.
Άρχισε να εφαρμόζεται η χρήση οπτικών μέσων όπως τα CD-ROM σε υπολογιστές και κονσόλες, μεταξύ αυτών και στην νέα και διάσημη κονσόλα PlayStation της Sony που είχε τεράστιο αντίκτυπο σε άλλες βιομηχανίες όπως η Sega και η Nintendo.

Στις αρχές του 2000 η Microsoft εισήλθε στην αγορά με την κυκλοφορία του Xbox. Καθώς η Sony και η Microsoft συνέχιζαν να επικεντρώνονται στην ανάπτυξη νέων κονσόλων υψηλών προδιαγραφών, η Nintendo επέλεξε να επικεντρωθεί στην καινοτομία του τρόπου παιχνιδιού, παρουσιάζοντας του Wii με χειριστήρια ανίχνευσης κίνησης, το οποίο αποτέλεσε στρατηγική για την κυκλοφορία του Nintendo Switch.

Από την δεκαετία του 2000 έως και του 2010, ή αγορά γνώρισε σημαντικές αλλαγές στο κοινό καθώς τα παιχνίδια για κινητές συσκευές σε smartphones και tablets εκτόπισαν τις κονσόλες, ενώ τα παιχνίδια για περιστασιακούς παίκτες κατέκτησαν μεγαλύτερο μερίδιο αγοράς. Παράλληλα αυξήθηκε σημαντικά ο αριθμός παικτών στην Κίνα και σε άλλες περιοχές που δεν είχαν στενή σχέση με την βιομηχανία. Για να αξιοποιηθούν αυτές οι αλλαγές, τα παραδοσιακά μοντέλα εσόδων αντικαταστάθηκαν από νέα μοντέλα όπως τα free-to-play, το freemium και τις συνδρομητικές υπηρεσίες. Η πρόοδος στο υλικό και το λογισμικό συνεχίζεται μέχρι και σήμερα και βελτιώνει την εμπειρία των βιντεοπαιχνιδιών, υποστηρίζοντας υψηλή ανάλυση, υψηλούς ρυθμούς καρέ και τεχνολογίες εικονικής και επαυξημένης πραγματικότητας.

1.2 Είδη Ψηφιακών Παιχνιδιών

Ένα είδος ψηφιακού παιχνιδιού αποτελεί μια συγκεκριμένη κατηγορία παιχνιδιών που συνδέονται μέσω παρομοίων χαρακτηριστικών εμπειρίας. Τα είδη παιχνιδιών συνήθως δεν ορίζονται από το περιβάλλον ή την πλοκή του παιχνιδιού, καθώς και ούτε από το μέσο το οποίο παίζονται, αλλά από τον τρόπο με τον οποίο ο παίκτης αλληλοεπιδρά με το παιχνίδι.

Για παράδειγμα ένα παιχνίδι τύπου First Person Shooter παραμένει στην ίδια κατηγορία ανεξάρτητα από το αν εκτυλίσσεται σε περιβάλλον επιστημονικής φαντασίας, γουέστερν αρκεί να περιλαμβάνει κάμερα που προσομοιώνει την οπτική γωνία του πρωταγωνιστή και την εμπειρία που επικεντρώνεται στην χρήση όπλων.

Μερικά από τα είδη αυτά είναι:

Παιχνίδια Δράσης (Action Games) :

Τα παιχνίδια δράσης δίνουν έμφαση σε φυσικές προκλήσεις που απαιτούν συντονισμό χεριού-ματιού και κινητικές δεξιότητες για την υπέρβαση τους. Επικεντρώνονται στον παίκτη, ο οποίος ελέγχει το μεγαλύτερος μέρος της δράσης

Παιχνίδια Μάχης (Fighting Games):

Τα παιχνίδια μάχης επικεντρώνονται σε μάχες μικρής εμβέλειας, οι οποίες συνήθως πραγματοποιούνται ένας-εναντίον-ενός ή εναντίον μικρού αριθμού αντιπάλων με ισοδύναμη ισχύ, και συχνά περιλαμβάνουν βίαιες και υπερβολικές άοπλες επιθέσεις. Τα περισσότερα παιχνίδια μάχης διαθέτουν μεγάλο αριθμό παικτών-χαρακτήρων προς επιλογή καθώς και ανταγωνιστική λειτουργία πολλών παικτών. Πολλά παιχνίδια μάχης ενσωματώνουν επιθέσεις εμπνευσμένες από διάφορα συστήματα πολεμικών τεχνών.

Παιχνίδια Πλατφόρμας (Platform Games):

Τα παιχνίδια πλατφόρμας χαρακτηρίζονται από περιβάλλοντα με κατακόρυφη διάταξη, όπου ο παίκτης πρέπει να πραγματοποιεί άλματα και αναρριχήσεις μεταξύ πλατφόρμων, εξ ου και η ονομασία τους. Η εμπειρία στα παιχνίδια πλατφόρμας επικεντρώνεται κυρίως στο άλμα και στην αναρρίχηση, προκειμένου ο παίκτης να εξερευνήσει ή να διασχίσει το περιβάλλον του παιχνιδιού. Μπορεί να περιλαμβάνονται εχθροί ή εμπόδια που πρέπει να αποφευχθούν ή να αντιμετωπιστούν, ενώ σε ορισμένες περιπτώσεις το παιχνίδι μπορεί να συνίσταται αποκλειστικά σε γρίφους άλματος.

ΚΕΦΑΛΑΙΟ 2 Μηχανές Δημιουργίας Παιχνιδιών

2.1 Εισαγωγή

Ένα Game Engine είναι ένα πλαίσιο λογισμικού που έχει σχεδιαστεί κυρίως για την ανάπτυξη βιντεοπαιχνιδιών, το οποίο περιλαμβάνει σχετικές βιβλιοθήκες και προγράμματα υποστήριξης.

Χρησιμοποιούνται από προγραμματιστές για την δημιουργία παιχνιδιών για διάφορες πλατφόρμες (PlayStation, Xbox, Android/IOS, PC).

Υπάρχουν πολλές μηχανές παραγωγής παιχνιδιών, παρόλα αυτά έχουν διακριθεί μέσα σε όλες αυτές τρείς μηχανές, οι οποίες μέχρι και σήμερα υποστηρίζονται πλήρως και ενημερώνονται κατάλληλα για να ανταπεξέλθουν στις απαιτήσεις του κοινού. Αυτές οι μηχανές είναι:

1) Unreal Engine
2) Unity Engine
3) Godot

2.2 UNREAL ENGINE

Η Unreal Engine είναι μια από τις πιο διάσημες μηχανές παραγωγής παιχνιδιών. Δημιουργήθηκε από την Epic Games και συγκεκριμένα από τον ευρετή της Tim Sweeney το 1995 και κυκλοφόρησε το 1998. Η μηχανή Unreal Engine είναι γραμμένη σε C++. Διαθέτει πληθώρα εργαλείων για την απλούστευση της δημιουργίας κώδικα καθώς και επεξεργασίας των υλικών και είναι η πιο διαδεδομένη μηχανή που χρησιμοποιείται από τις AAA (Triple-A) εταιρείες.

Είναι μια από τις πιο επιτυχημένες μηχανές παραγωγής παιχνιδιών καθώς έχει βραβευτεί με το World Guinness Record (2015) στον χώρο αυτό .

2.3 GODOT

Η Godot είναι μια ανοιχτού κώδικα μηχανή αναζήτησης για ανάπτυξη παιχνιδιών. Αναπτύχθηκε στο Μπουένος Άιρες από τους Αργεντινούς προγραμματιστές Juan Linietsky και Ariel Manzur για διάφορες εταιρείες στην Λατινική Αμερική. Κυκλοφόρησε δημόσια το 2014. Το περιβάλλον ανάπτυξης λειτουργεί σε πολλές πλατφόρμες και μπορεί να εξάγει έργα σε ακόμα περισσότερες. Χρησιμοποιείται για παιχνίδια 2D και 3D για υπολογιστές, κινητές συσκευές, διαδικτυακές εφαρμογές , καθώς και για VR, AR και MR πραγματικότητας. Είναι γραμμένη σε C++, C# και GDScript.

Το GDScript είναι η ενσωματωμένη προγραμματιστική γλώσσα υψηλού επιπέδου με σταδιακή τυποποίηση , η οποία παρουσιάζει συντακτικές ομοιότητες με την γλώσσα Python.

Αν και χρησιμοποιείται λιγότερο από τις άλλες δύο μηχανές, η εξέλιξη της με τις νέες εκδόσεις της την καθιστούν ένα πολύ εύχρηστο περιβάλλον για έναν 'καινούργιο' προγραμματιστή παιχνιδιών.

ΚΕΦΑΛΑΙΟ 3 UNITY GAME ENGINE

3.1 Εισαγωγή

Στο πλαίσιο της παρούσας πτυχιακής εργασίας, επιλέχθηκε η **Unity** ως η κύρια μηχανή ανάπτυξης βιντεοπαιχνιδιών, λόγω της ευρείας διάδοσής της, της ευελιξίας της και της καταλληλόλητάς της για τη δημιουργία τόσο δισδιάστατων(2D) όσο και τρισδιάστατων(3D) εφαρμογών. Η ιστορική πορεία του Unity αντικατοπτρίζει την εξέλιξη της τεχνολογίας ανάπτυξης παιχνιδιών και την αυξανόμενη ζήτηση για εργαλεία που συνδυάζουν τη λειτουργικότητα με τη φιλικότητα προς τον χρήστη.

Η **Unity** παρουσιάστηκε για πρώτη φορά το 2005 από την εταιρεία **Unity Technologies** ως ένα εργαλείο ανάπτυξης για το λειτουργικό σύστημα **Mac OS X**, με κύριο στόχο να καταστήσει τη διαδικασία δημιουργίας βιντεοπαιχνιδιών πιο προσιτή σε ανεξάρτητους προγραμματιστές, σε μικρές ομάδες ή ακόμα και σε νέους προγραμματιστές που επιθυμούν να ενταχθούν στον τομέα του Game Development. Η αρχική της επιτυχία οφείλεται σε ένα μεγάλο βαθμό στην υποστήριξη της για πολλαπλές πλατφόρμες, επιτρέποντας στους δημιουργούς να αναπτύσσουν παιχνίδια για υπολογιστές, διαδικτυακές εφαρμογές, κονσόλες και κινητές συσκευές, χωρίς την ανάγκη εκτεταμένων αλλαγών στον κώδικα.

Κατά τη διάρκεια της δεκαετίας του 2010, η Unity εξελίχθηκε σε μία από τις πιο διαδεδομένες μηχανές ανάπτυξης παιχνιδιών παγκοσμίως. Η εισαγωγή του **Unity Asset Store** το 2010 διευκόλυνε ακόμη περισσότερο την ανάπτυξη, παρέχοντας στους δημιουργούς πρόσβαση σε έτοιμα μοντέλα, σενάρια και εργαλεία, ενισχύοντας την παραγωγικότητα και μειώνοντας τον χρόνο ανάπτυξης. Παράλληλα, η συνεχής υποστήριξη για νέες τεχνολογίες, όπως η εικονική(VR) και η επαυξημένη πραγματικότητα (AR), εδραίωσε τη Unity ως βασικό εργαλείο όχι μόνο για βιντεοπαιχνίδια αλλά και για εφαρμογές προσομοίωσης, εκπαίδευσης και διαδραστικής απεικόνισης.

Σήμερα, η Unity χρησιμοποιείται από ανεξάρτητους δημιουργούς αλλά και από μεγάλες εταιρείες του χώρου, προσφέροντας μια ολοκληρωμένη πλατφόρμα ανάπτυξης που συνδυάζει ισχυρή τεχνική υποδομή με ευκολία χρήσης. Η ιστορική της εξέλιξη αντανακλά τη συνεχή πρόοδο στη βιομηχανία του ψηφιακού παιχνιδιού και τον καθοριστικό ρόλο των μηχανών ανάπτυξης στη διαμόρφωση της σύγχρονης διαδραστικής εμπειρίας. (Haas, 2014)

3.2 Unity 1.0

Η μηχανή ανάπτυξης παιχνιδιών κυκλοφόρησε το 2005 με σκοπό να μπορέσουν περισσότεροι προγραμματιστές να έχουν πρόσβαση στον τομέα του Game Development. Παρουσιάστηκε στο κοινό από τον Scott Forstall για το Mac OS X. Βραβεύτηκε με την δεύτερη θέση το 2006 στα Apple Inc. 's Apple Design Awards, και αργότερα άρχισε να υποστηρίζεται για συστήματα της Microsoft και για προγράμματα περιήγησης Ιστού. (Smykil, 2006)

3.3 Unity 2.0

Κυκλοφόρησε το 2007 με περίπου 50 καινούργιες λειτουργίες καθώς και υποστήριξη του DirectX. Στην έκδοση αυτή μερικές από τις λειτουργίες που προστέθηκαν ήταν ένα βελτιστοποιημένο σύστημα διαχείρισης εδάφους (terrain engine)που έδινε την δυνατότητα δημιουργίας τρισδιάστατων περιβαλλόντων με λεπτομέρεια. Ακόμα ενσωματώθηκαν στην

έκδοση αυτή δυναμικές σκιές σε πραγματικό χώρο, καθώς και κατευθυντικός φωτισμός και προβολείς.
Τέλος αποτέλεσε σημαντική καινοτομία η Στρώση Δικτύωσης (Networking Layer), η οποία επέτρεπε στους προγραμματιστές την δημιουργία Multiplayer παιχνιδιών χρησιμοποιώντας το πρωτόκολλο UDP(User Datagram Protocol) , με υποστήριξη για NAT(Network Address Translation), State Synchronization και RPCs (Remote Procedure Calls).

Τέλος με την έκδοση της Unity 2.5 ήταν πλέον δυνατή και η υποστήριξη των Windows δίνοντας έτσι την δυνατότητα και σε άλλους χρήστες να έχουν πρόσβαση σε αυτήν την μηχανή ανάπτυξης παιχνιδιών.

3.4 Unity 3.0

Η έκδοση του Unity 3.0 κυκλοφόρησε το Σεπτέμβριο του 2010, προσθέτοντας νέες σημαντικές βελτιώσεις στα γραφικά για υπολογιστές και κονσόλες παιχνιδιών. Πλέον υπήρχε υποστήριξη για το λογισμικό Android και επιπλέον ενσωμάτωσε ένα νέο εργαλείο με όνομα Beast Lightmap της εταιρείας Illuminate Labs για προηγμένο φωτισμό, προστέθηκε η τεχνική Deferred Rendering για βελτιωμένη απόδοση γραφικών, έναν ενσωματωμένο επεξεργαστή δένδρων (Tree Editor),Native Font Rendering, Αυτόματη Χαρτογράφηση UV και τέλος εισήγαγε ηχητικά φίλτρα.

Το 2012 σύμφωνα με μια έρευνα από το VentureBeat αναφέρθηκε ότι είναι από τις μεγαλύτερες εταιρείες που έχουν συμβάλει τόσο σημαντικά στην ανάπτυξη των βιντεοπαιχνιδιών, καθώς επισημάνθηκε ότι πάνω από 1.3 εκατομμύρια χρήστες χρησιμοποιούσαν τα εργαλεία της Unity Technologies για την δημιουργία παιχνιδιών.

Επιπλέον μέσω έρευνα του Game Developer Magazine τον Μάιο του 2012 η Unity αναδείχθηκε ως η κορυφαία μηχανή παραγωγής παιχνιδιών για τις πλατφόρμες Mobile. (GameDeveloper, 2012)

3.5 Unity 4.0

Η Unity Technologies κυκλοφόρησε τον Νοέμβριο του 2012 την έκδοση Unity 4.0 που παρείχε σημαντικές βελτιώσεις. Συγκεκριμένα παρείχε υποστήριξη για το DirectX 11, το Adobe Flash, νέα εργαλεία κινούμενης εικόνας με όνομα Mecanim και τέλος υποστήριζε την προεπισκόπηση σε συστήματα Linux.

Το 2013 η Facebook ενσωμάτωσε ένα SDK (Software Development Kit) για παιχνίδια που αναπτύσσονταν με την μηχανή Unity, παρέχοντας εργαλεία για άμεση σύνδεση των χρηστών από αναρτήσεις στα κοινωνικά δίκτυα σε συγκεκριμένα σημεία των παιχνιδιών, καθώς και δυνατότητα εύκολης κοινοποίησης εικόνων εντός παιχνιδιού.

Το 2016 το Facebook ανέπτυξε μια νέα πλατφόρμα παιχνιδιών για υπολογιστές σε συνεργασία με την Unity, η οποία προσέφερε υποστήριξη για τις πλατφόρμες αυτές. Έτσι παρείχε την υποστήριξη στους προγραμματιστές να εξάγουν και να δημοσιεύουν παιχνίδια στο Facebook γρηγορότερα, ευκολότερα και πιο αποτελεσματικά.

3.6 Unity 5.0

Η κυκλοφορία της έκδοσης Unity 5 το 2015, αποτέλεσε σημαντικό σταθμό για τη δημοφιλή μηχανή ανάπτυξης παιχνιδιών. Σύμφωνα με το The Verge , η Unity είχε ως βασικό στόχο την ευρύτερη και καθολική πρόσβαση στην δημιουργία ψηφιακών παιχνιδιών, και η πέμπτη γενιά του

λογισμικού θεωρήθηκε ένα καθοριστικό βήμα προς αυτήν την κατεύθυνση. Η συγκεκριμένη έκδοση παρουσίασε σημαντικές τεχνικές αναβαθμίσεις όπως βελτιωμένο σύστημα φωτισμού και ήχου, καθώς και την υποστήριξη WebGL ,που επέτρεψε στους δημιουργούς να διαθέτουν τα έργα σε διαδικτυακούς φυλλομετρητές, χωρίς την ανάγκη πρόσθετων πακέτων . (Kumparak, 2014)

Επιπλέον η Unity 5.0 Εισήγαγε νέες τεχνολογίες όπως ο πραγματικός φωτισμός σε πραγματικό χρόνο, οι δυναμικές προεπισκόπησης φωτισμού, την πλατφόρμα Unity Cloud , Ένα ανανεωμένο σύστημα ήχου καθώς και ενσωμάτωση της μεγάλης φυσικής Nvidia PhysX 3.3. Σημαντική καινοτομία αποτέλεσε επίσης η εισαγωγή των Cinematic Image Effects, οι οποίες συνέβαλαν πάνω στην αισθητική βελτίωση των παιχνιδιών που δημιουργούνταν με τη μηχανή, μειώνοντας την ομοιομορφία στην οπτική τους ταυτότητα.

Τον Αύγουστο του 2015 η Unity παρουσίασε μια πειραματική, αν και μη υποστηριζόμενη, έκδοση του Editor της για το λειτουργικό σύστημα Linux , ενώ η έκδοση 5.6 εμπλούτισε την μηχανή με νέες δυνατότητες, όπως προηγμένα εφέ φωτισμού και particles, καλύτερη συνολική απόδοση με το νέο γραφικό περιβάλλον με ονομασία Vulkan, καθώς και υποστήριξη για πλατφόρμες όπως το Nintendo Switch και το Facebook Gameroom και το Google Daydream. Τέλος προστέθηκε ένα σύστημα για αναπαραγωγή βίντεο ικανό να μπορεί να διαχειριστεί αναλύσεις μέχρι και 4K και υποστήριξη βίντεο 360° για εφαρμογές εικονικής πραγματικότητας. (Grubb, 2017)

Ωστόσο , η ευρεία διάδοση της Unity οδήγησε και σε ορισμένες αρνητικές κριτικές , καθώς η εύκολη πρόσβαση στην μηχανή αυτή συνδέθηκε με την παραγωγή παιχνιδιών χαμηλής ποιότητας από λιγότερο έμπειρους δημιουργούς. Ο διευθύνων σύμβουλος της εταιρείας John Riccitiello τόνισε ότι αυτή η κατάσταση αποτελεί φυσικό αποτέλεσμα της προσπάθειας εκδημοκρατισμού της ανάπτυξης παιχνιδιών, θεωρώντας ωστόσο θετικό το γεγονός ότι η πλατφόρμα ενθαρρύνει όλο και περισσότερους ανθρώπους να μην είναι μόνο καταναλωτές της τεχνολογίας αλλά και δημιουργοί της.

3.7 Unity 6.0

Η Unity Technologies ανακοίνωσε στις 16 Νοεμβρίου του 2023 την επόμενη έκδοση της μηχανής με το όνομα Unity 6. Η έκδοση αυτή κυκλοφόρησε δημόσια στις 17 Οκτωβρίου του 2024, εισάγοντας νέες δυνατότητες ,όπως τα AI εργαλεία ***Unity Muse*** και ***Unity Sentis***.

Αυτές οι εξελίξεις καθιστούν το Unity όχι απλώς μια μηχανή ανάπτυξης παιχνιδιών, αλλά και ένα ισχυρό περιβάλλον για το AI.

3.7.1 Unity Muse

Η ***Unity Muse*** αποτελεί ένα σύνολο εργαλείων δημιουργίας περιεχομένου μέσω AI. Περιλαμβάνει ενσωματωμένα εργαλεία όπως τα:

- ***Muse Chat***: παρέχει την δυνατότητα για διαρθρωμένες απαντήσεις και την δημιουργία λειτουργικού κώδικα από φυσικές γλώσσες.
- ***Muse Sprite*** και ***Muse Texture***: δίνει την δυνατότητα για γρήγορη δημιουργία Sprites και Texture με χρήση εξατομικευμένων μοντέλων Deep Learning που βασίζονται σε datasets αποκλειστικά υπό την ιδιοκτησία της Unity

3.7.2 Unity Sentis

Η ***Unity Sentis*** (Μετονομάστηκε σε Inference Engine) αποτελεί runtime λύση για ενσωμάτωση μοντέλων τεχνητής νοημοσύνης μέσα στο παιχνίδι.

Επιτρέπεται στους προγραμματιστές να χρησιμοποιήσουν AI μοντέλα τοπικά χωρίς να απαιτείται σύνδεση δικτύου ή Cloud με αποτέλεσμα της εξάλειψη καθυστέρησης και κόστους.

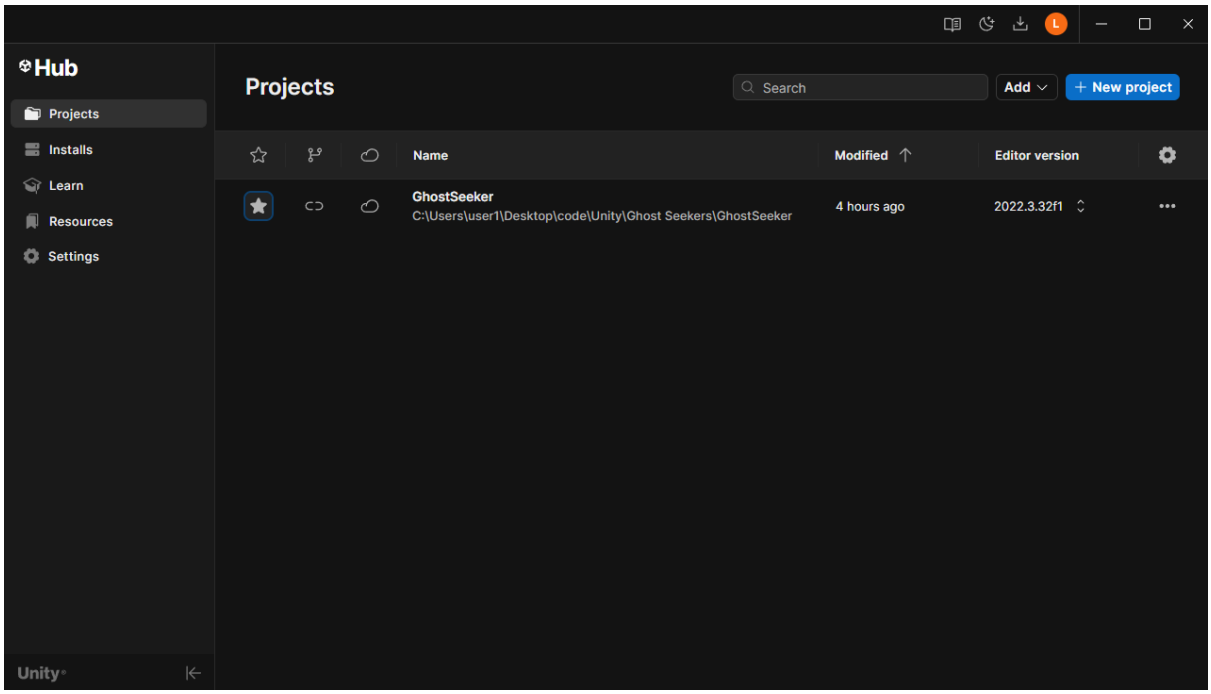
Τέλος, παρέχει υποστήριξη για λειτουργίες όπως:

- Αναγνώριση Αντικειμένων
- Έλεγχο Οπτικού Βάθους σε παιχνίδια AR
- Αναγνώριση Ομιλίας
- Έξυπνη συμπεριφορά για τα NPC

3.8 Πως λειτουργεί το Unity Engine

3.8.1 Έναρξη Unity Hub

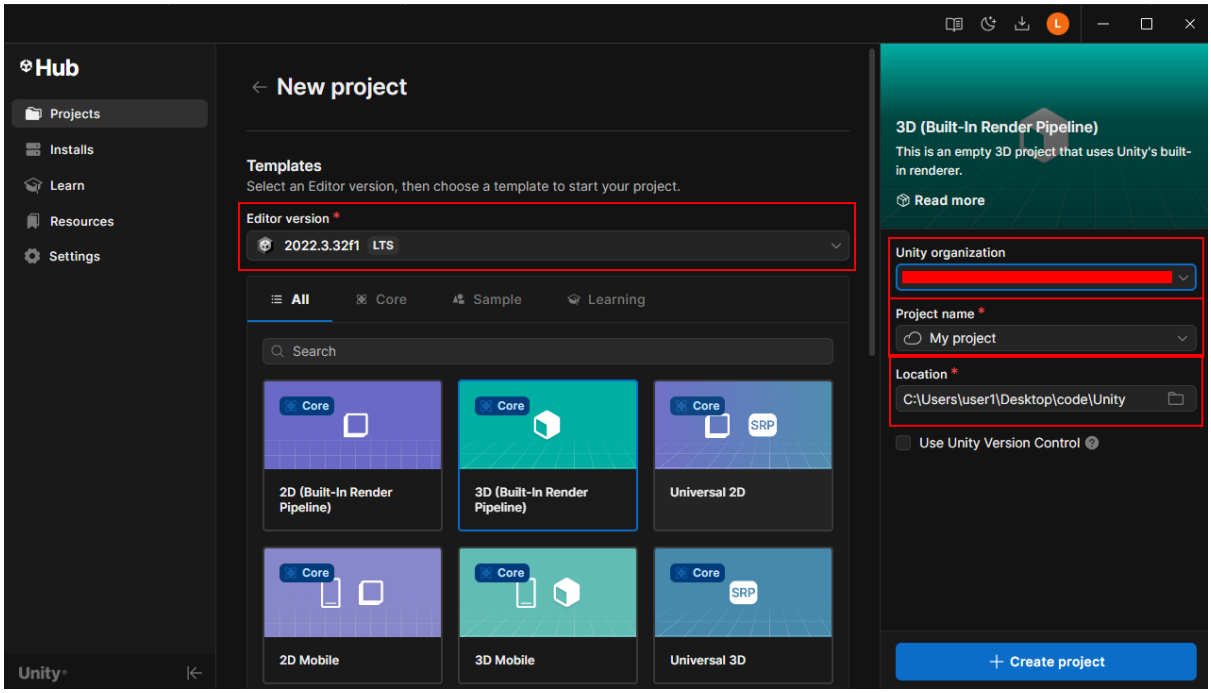
Κατά την έναρξη του Unity Hub, αφού δημιουργήσουμε λογαριασμό και συνδεθούμε μετά στον λογαριασμό μας, θα παρουσιαστεί το παρακάτω παράθυρο:



Unity Hub Εικόνα

Στο συγκεκριμένο παράθυρο φαίνονται τα Project τα οποία έχουμε δημιουργήσει και όσα είναι αποθηκευμένα στην συσκευή μας.

Πατώντας το κουμπί "+ New Project " θα μας οδηγήσει σε ένα νέο παράθυρο με αυτήν την μορφή:



Unity's Project Creation Menu

Στο παράθυρο αυτό μπορούμε να επιλέξουμε και να αλλάξουμε τις ρυθμίσεις που επιθυμούμε για το Project, όπως για παράδειγμα την έκδοση στην οποία θα αναπτύξουμε το παιχνίδι μας, το όνομα της εταιρείας μας, το όνομα του Project μας και τέλος αν θέλουμε να ενεργοποιήσουμε την λειτουργία "Unity Version Control" της οποίας η λειτουργία είναι να ενημερώσει και να επιτρέπει σε μια ομάδα προγραμματιστών να έχουν όλοι την ίδια έκδοση για την ανάπτυξη του Project.

Επίσης ο χρήστης έχει την δυνατότητα να επιλέξει ένα είδος Template για να ξεκινήσει το Project του. Μερικές δυνατές επιλογές είναι:

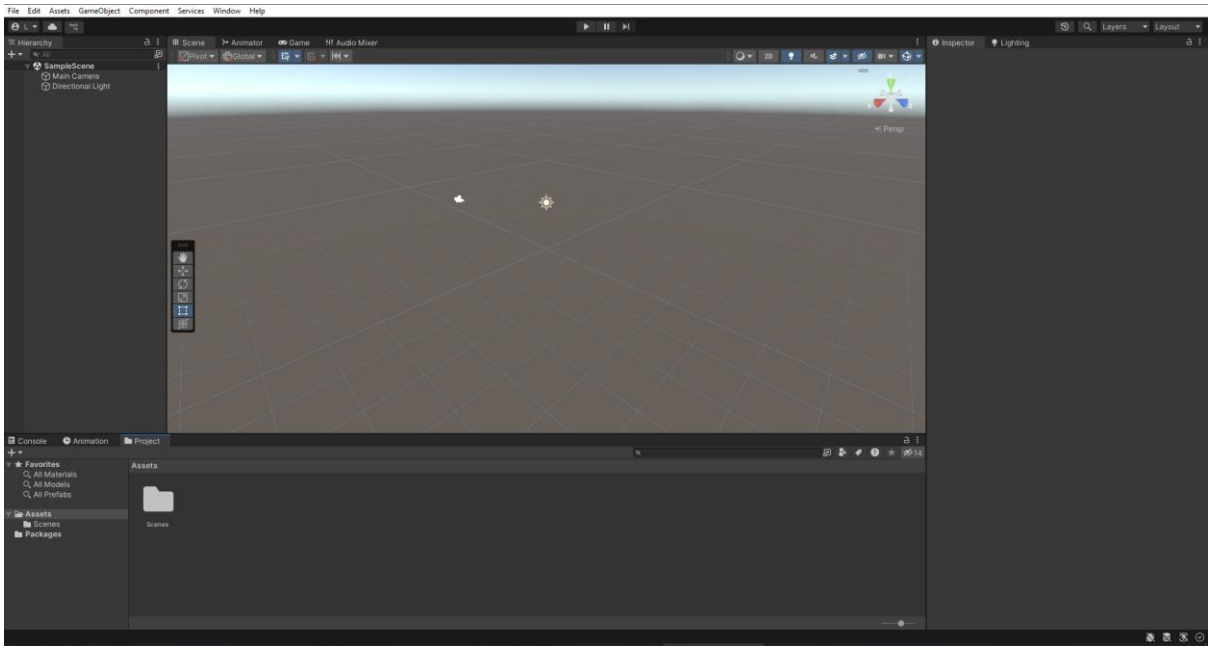
- Δημιουργία 2D ή 3D περιβάλλοντος για υπολογιστή
- Δημιουργία 2D ή 3D περιβάλλοντος για κινητές συσκευές
- Δημιουργία περιβάλλοντος για AR, VR ή MR

Το Ghost Seekers είναι χτισμένο σε Universal 3d περιβάλλον ή αλλιώς URP, και η διαφορά του με ένα απλό 3D περιβάλλον είναι η ποιότητα των γραφικών καθώς και επιπρόσθετες βιβλιοθήκες για την καλύτερη απόδοση σκιών, φωτισμών, εφέ κάμερας και ήχου αλλά και άλλων λεπτομερειών πάνω στα αντικείμενα μας.

Στην συνέχεια πατάμε το κουμπί "+ Create Project" για την δημιουργία του περιβάλλοντος πάνω στο οποίο θα αναπτυχθεί το παιχνίδι μας.

3.8.2 Το περιβάλλον του Unity και Βασικές Λειτουργείες

Αφού το Project μας δημιουργηθεί θα βρεθούμε σε ένα νέο παράθυρο που θα ανταποκρίνεται στο Template που επιλέξαμε. Αυτό είναι η λεγόμενη Σκηνή μας ή 'Scene', η οποία θα είναι πάντα κενή στην αρχή.

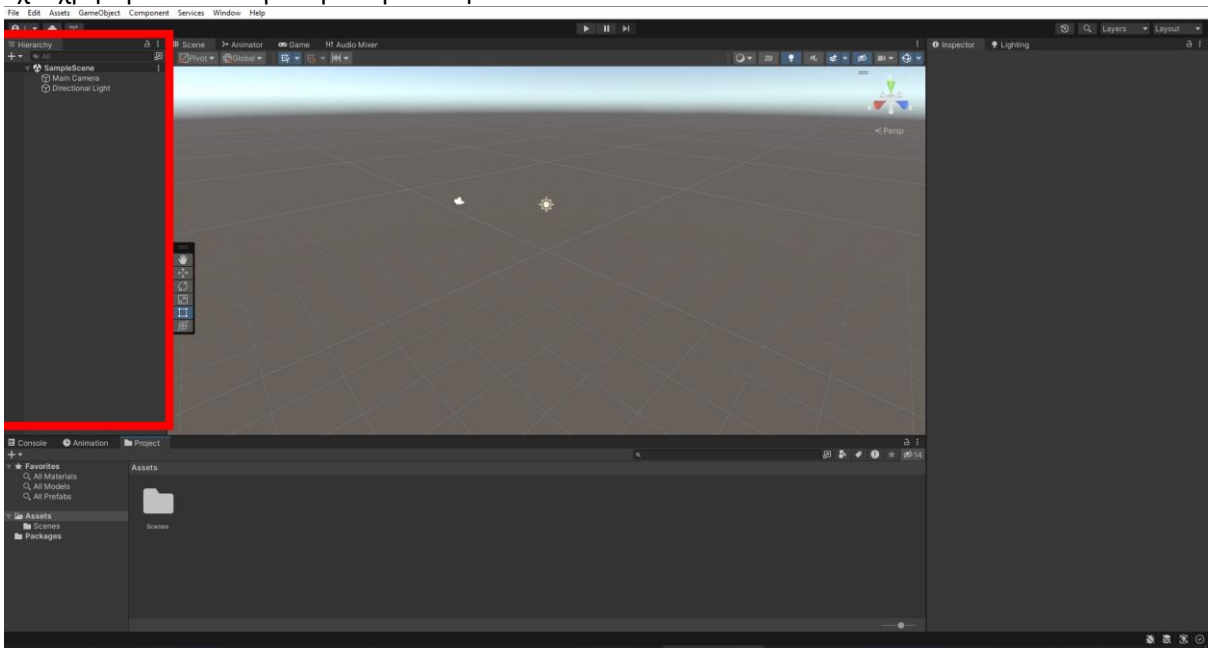


Unity Scene

Στα αριστερά του παραθύρου , μπορούμε να παρατηρήσουμε μια στήλη ή οποία περιέχει για αρχή 2 αντικείμενα. Αυτά τα αντικείμενα ή αλλιώς GameObjects θα δημιουργούνται πάντα μαζί με την έναρξη της σκηνής.

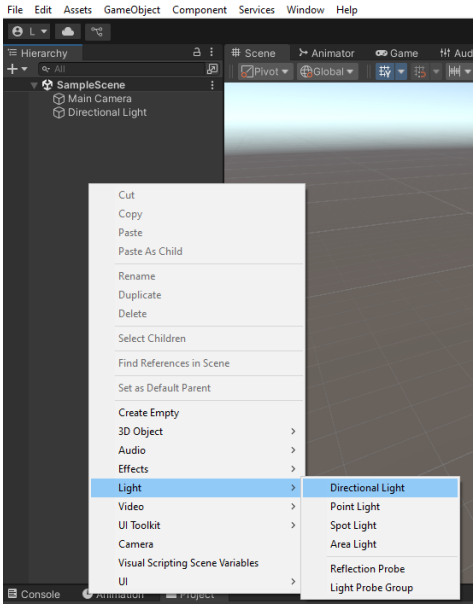
Το πρώτο αντικείμενο με όνομα Main Camera είναι η κάμερα την οποία ο παίκτης μπορεί να χρησιμοποιήσει για να έχει όραση στο περιβάλλον.

Το δεύτερο αντικείμενο με όνομα Directional Light είναι ο φωτισμός του περιβάλλοντος και έχει χρησιμοποιείται για προσομοίωση των του Ήλιου.

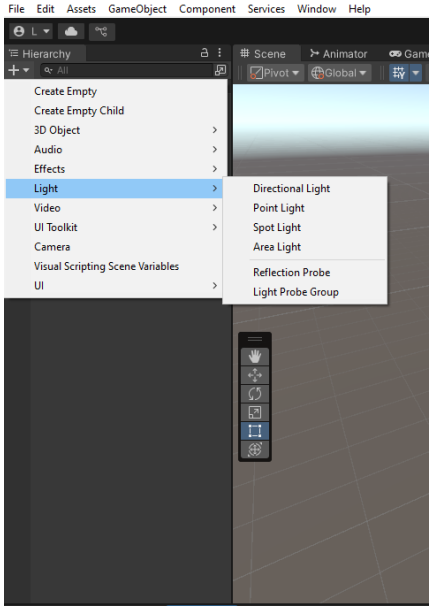


Unity Hierarchy

Στην περίπτωση που διαγράψουμε κάποιο από αυτά τα αντικείμενα μπορούμε εύκολα να δημιουργήσουμε άλλο: 1) Πατώντας το '+' πάνω-αριστερά του Hierarchy, 2) Κάνοντας Δεξί Κλικ και επιλέγοντας το αντίστοιχο αντικείμενο που θέλουμε να δημιουργήσουμε μέσα από την λίστα αυτή.

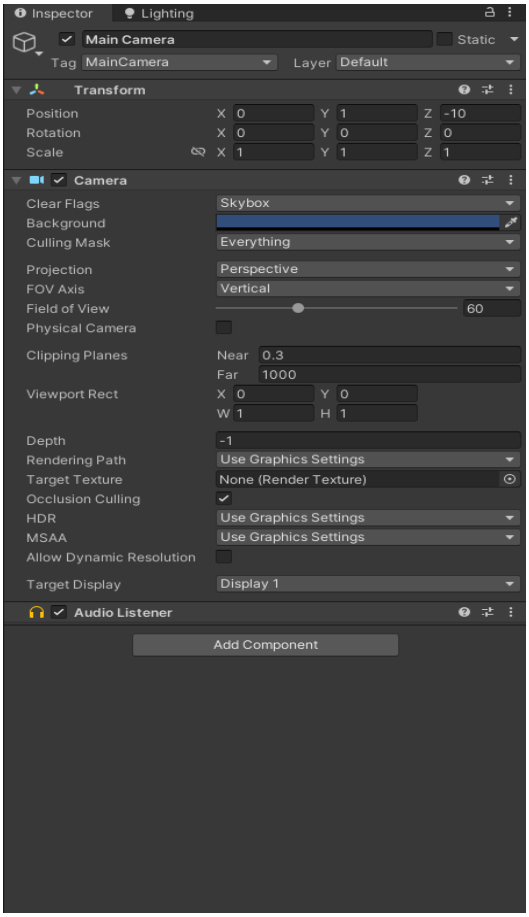


Creating Object
Using Right Click



Creating Object
Using '+' Button

Επιλέγοντας ένα στοιχείο στο Hierarchy θα παρατηρήσουμε πως στην δεξιά μεριά του παραθύρου και συγκεκριμένα στην στήλη με όνομα Inspector, θα εμφανιστούν ορισμένες ρυθμίσεις.



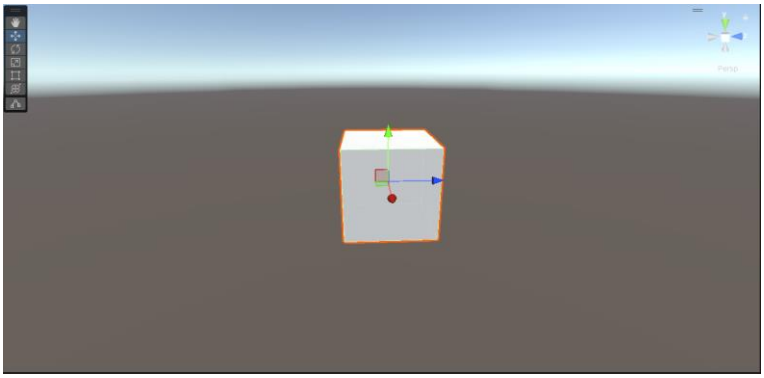
Inspector Tab

Στην στήλη αυτή πλέον μας δίνεται η δυνατότητα να ρυθμίσουμε παραμέτρους και άλλες ιδιότητες του αντικειμένου του οποίου επιλέξαμε από το Hierarchy.
Πιο συγκεκριμένα η πρώτη ιδιότητα που μπορούμε να ρυθμίσουμε ονομάζεται Transform.

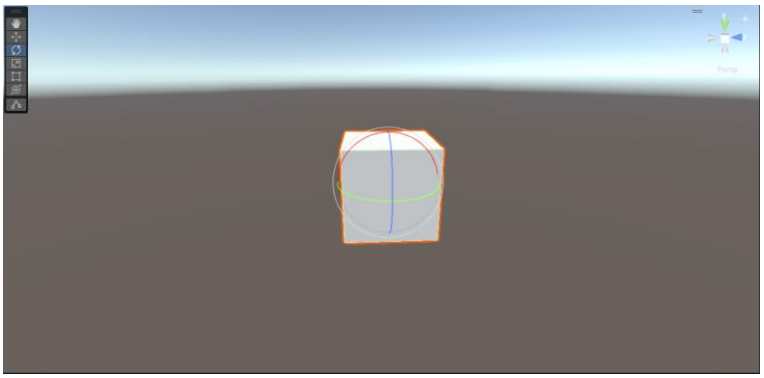
Αυτή η ιδιότητα βρίσκεται σε οποιοδήποτε αντικείμενο καθώς επηρεάζει :

1. Την θέση του στον χώρο
2. Την περιστροφή του σε μοίρες
3. Το μέγεθος του στο χώρο ανά μονάδες Unit (1 Unit ≈ 1m)

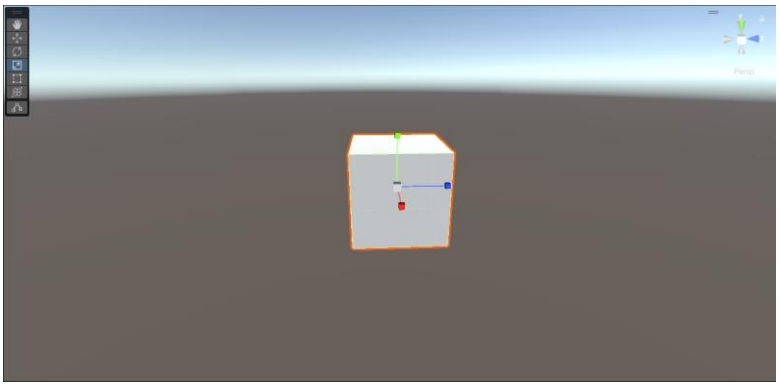
Αυτά τα πεδία μπορούμε να τα επεξεργαστούμε χειροκίνητα, θέτοντας του τιμές που επιθυμούμε ή μπορούμε μέσα από το ίδιο το περιβάλλον χρησιμοποιώντας μετακινώντας το GameObject μας με την χρήση συντομεύσεων και με τα βέλη ανάλογα την κατεύθυνση που επιθυμούμε αν μετακινήσουμε το αντικείμενο μας ή ακόμα και να το περιστρέψουμε με τον ίδιο τρόπο ή και να το μεγαλώσουμε και να το μικρύνουμε , όπως φαίνεται και στις παρακάτω εικόνες για την κάθε περίπτωση.



Position Changer for Object



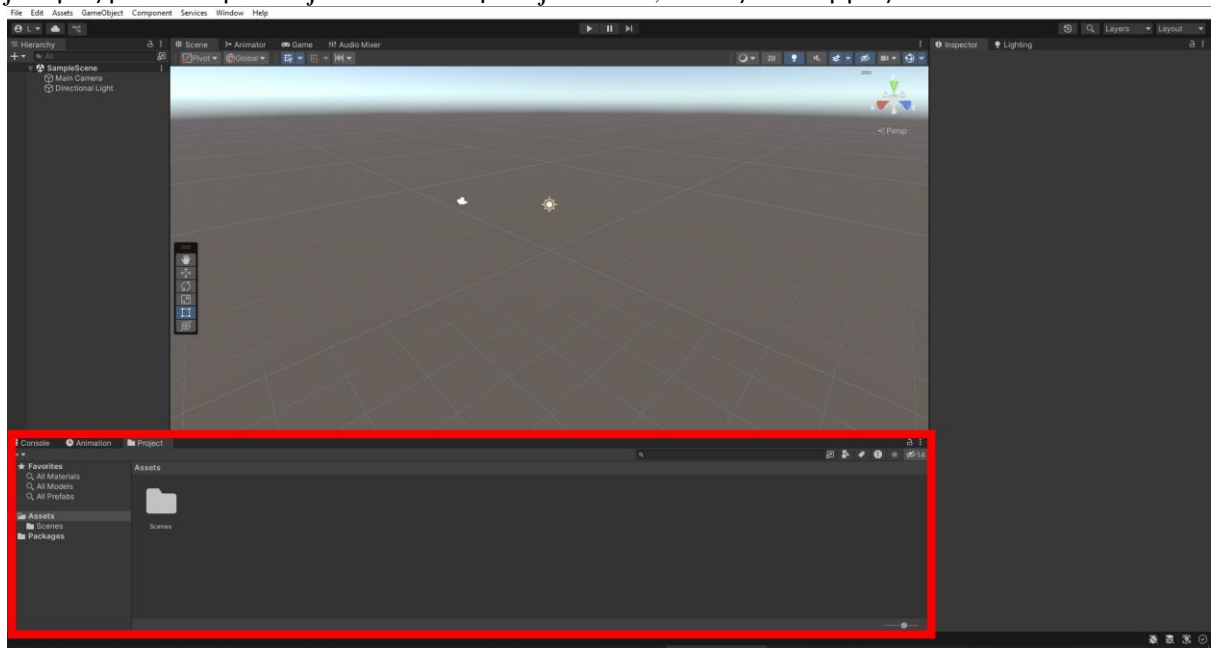
Rotation Changer for Object



Scale Changer for Object

Στην συνέχεια οι επόμενες ιδιότητες που θα εμφανιστούν κάτω από το Transform θα είναι διαφορετικές με βάση το τύπο του αντικειμένου που έχουμε επιλέξει.

Τέλος, έχει μεγάλη σημασία να μιλήσουμε για το πλαίσιο που περιλαμβάνει τα Assets του Project μας με το όνομα Project Window ή Project View, και τις λειτουργίες του.



Assets Display Panel

Στο συγκεκριμένο πλαίσιο, ο χρήστης μπορεί να διαχειριστεί:

- Τους φακέλους του και την δημιουργία νέων φακέλων.
- Να αποθηκεύσει νέα Assets για Textures, Μοντέλα κ.λπ.
- Να δημιουργήσει και να αποθηκεύσει Scripts.
- Να δημιουργήσει και να αποθηκεύσει Materials για τα αντικείμενα που χρησιμοποιεί.
- Να δημιουργήσει, να επεξεργάζεται Animations.
- Να αποθηκεύσει και να ρυθμίζει Prefabs.

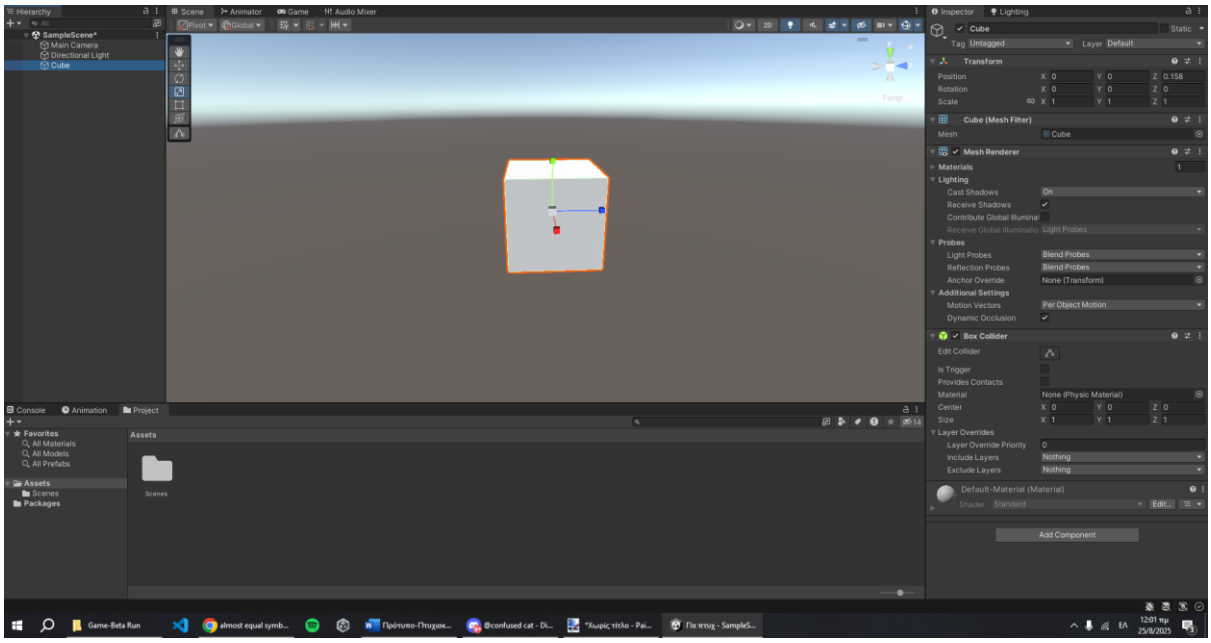
3.8.3 Η Λειτουργία των Prefabs και η Χρησιμότητα τους

Τα prefabs στον περιβάλλον του Unity αποτελούν αναπόσπαστο κομμάτι για τους προγραμματιστές καθώς προσφέρει πολλαπλά πλεονεκτήματα για την ανάπτυξη του παιχνιδιού τόσο στην ταχύτητα όσο και στην απόδοση του παιχνιδιού.

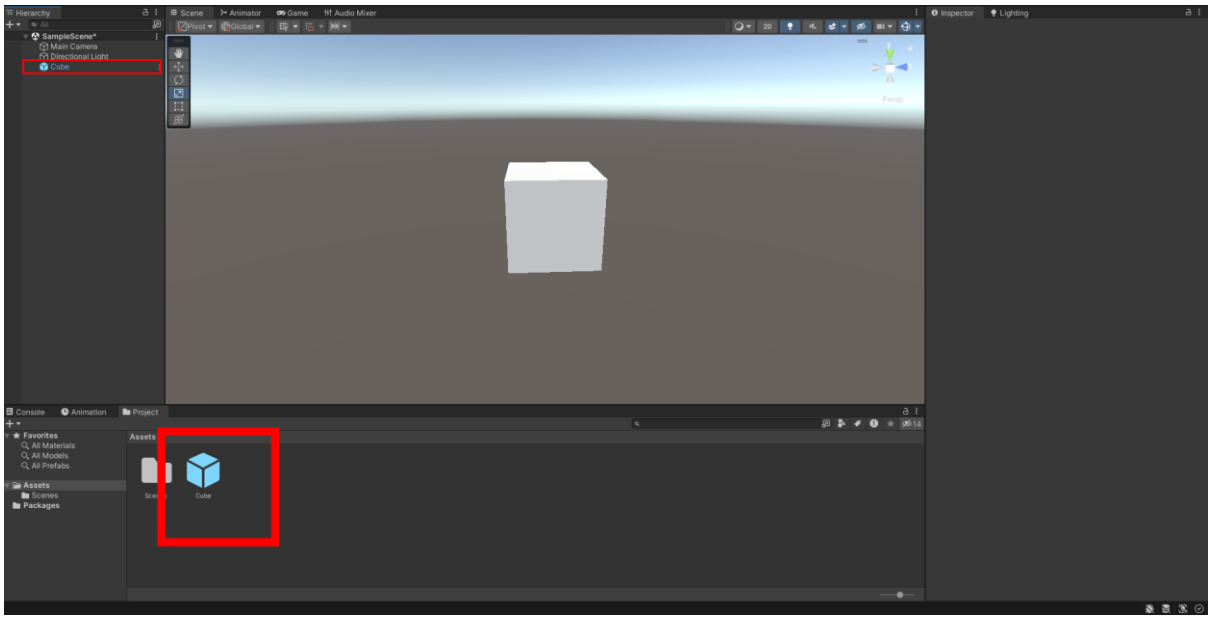
Στην ουσία ένα Prefab είναι ένα έτοιμο, προκαθορισμένο αντικείμενο ή ακόμα και μια συλλογή από αντικείμενα, το οποίο διατηρεί αποθηκευμένα σε μια ενιαία μορφή όλες τους τις ιδιότητες (π.χ. μέγεθος, περιστροφή), τα Components του (π.χ. Colliders, Φωτισμούς, Scripts), ώστε να μπορούν να χρησιμοποιηθούν εύκολα και γρήγορα σε διάφορα σημεία ενός παιχνιδιού με μία απλή τοποθέτηση τους στο χώρο, με την λειτουργία Drag and Drop ή ακόμα και μέσω ενός Script.

3.8.3.1 Δημιουργία Prefab

Για να δημιουργήσουμε ένα Prefab θα χρειαστεί να έχουμε ήδη δημιουργήσει ένα αντικείμενο στο Hierarchy. Αφού το δημιουργήσουμε, το μόνο που χρειάζεται να κάνουμε είναι να ‘σύρουμε’ το αντικείμενο από το Hierarchy στο Asset Panel ή σε οποιοδήποτε φάκελο μέσα σε αυτό. Με την ολοκλήρωση αυτού του βήματος θα παρατηρήσουμε ότι θα αλλάξει και το εικονίδιο που είχε προηγουμένως.



Object Before Becoming Prefab

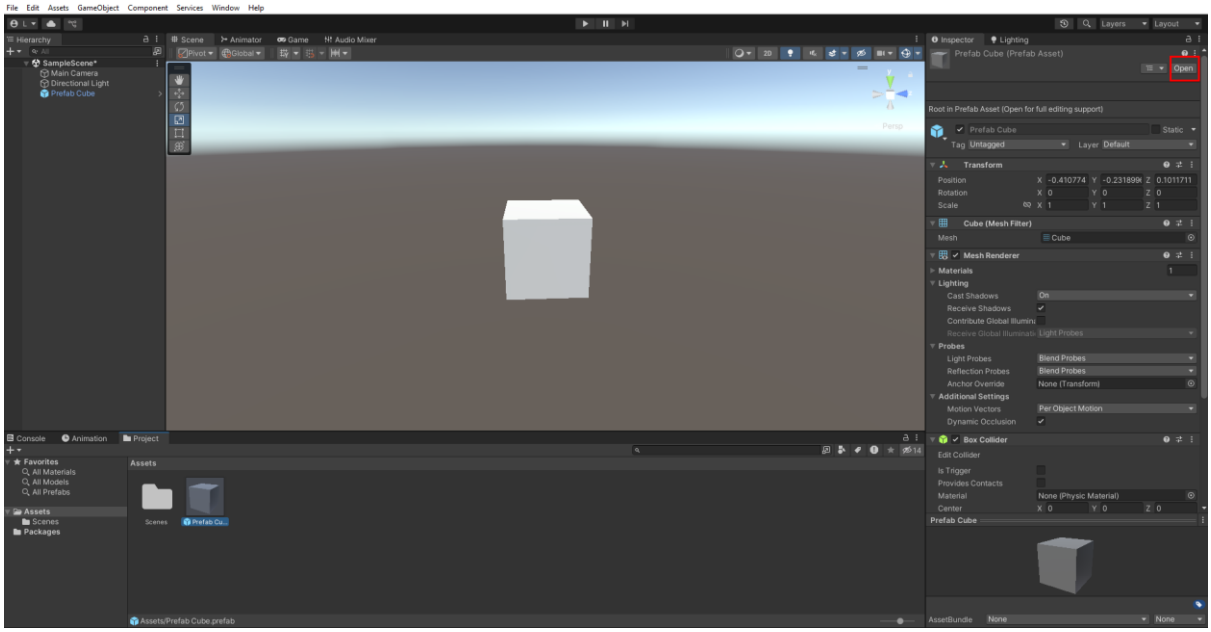


Object after Becoming a Prefab

3.8.3.2 Ιδιότητες ενός Prefab

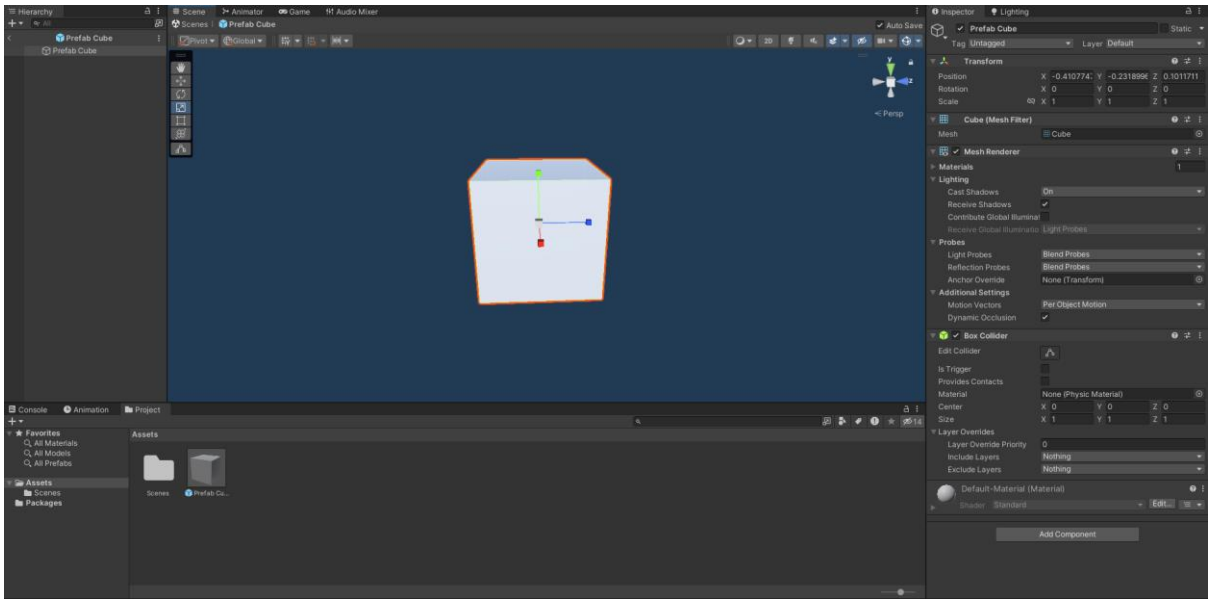
Αφού δημιουργήσουμε το Prefab μας μπορούμε να προσθέσουμε διάφορους κώδικες πάνω σε αυτό ή διάφορα άλλα Components από το Inspector.

Για να το κάνουμε αυτό και να επεξεργαστούμε απευθείας το Prefab αντικείμενο μας πρέπει να επιλέξουμε πρώτα το Prefab από το Project Window και να πατήσουμε το κουμπί 'Open' στον Inspector, όπως φαίνεται στην εικόνα από κάτω:



Prefab Inspector

Πατώντας το παραπάνω Κουμπί θα ανοίξει ένα νέο παράθυρο με αυτήν την μορφή:

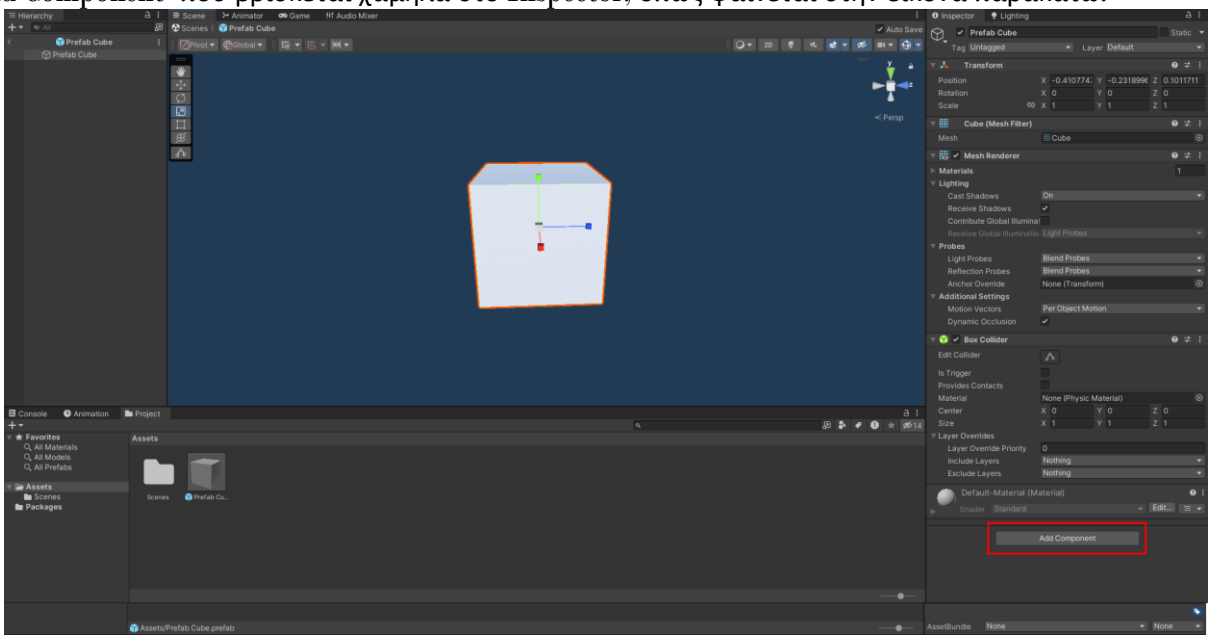


Onbject Prefab

Σε αυτό το παράθυρο ο χρήστης έχει την δυνατότητα να αλλάξει το Prefab εξ ολοκλήρου, με κάθε αλλαγή να εφαρμόζεται επίσης σε κάθε κλώνο του Prefab μέσα στις Σκηνές μας. Δηλαδή αν αλλάξουμε το χρώμα του κύβου από λευκό σε κόκκινο, όλοι οι κύβοι, σε κάθε σκηνή θα αποκτήσουν αυτό το χρώμα. Αυτό συμβαίνει για κάθε ιδιότητα που επηρεάσουμε στον κύβο.

Επιπλέον στο παράθυρο αυτό εκτός από το να επεξεργαστούμε τις ήδη υπάρχουσες ιδιότητες του αντικειμένου μας, μπορούμε να προσθέσουμε και άλλα Components (π.χ. Scripts), τα οποία θα εφαρμοστούν σε όλα τα prefab του αντικειμένου που αλλάξαμε.

Για να προσθέσουμε ένα νέο Component μπορούμε είτε να το 'σύρουμε' από το Project Window μέσα στο Inspector του επιλεγμένου αντικειμένου ή μπορούμε να πατήσουμε το κουμπί 'Add Component' που βρίσκεται χαμηλά στο Inspector, όπως φαίνεται στην εικόνα παρακάτω:



Add Component to Prefab

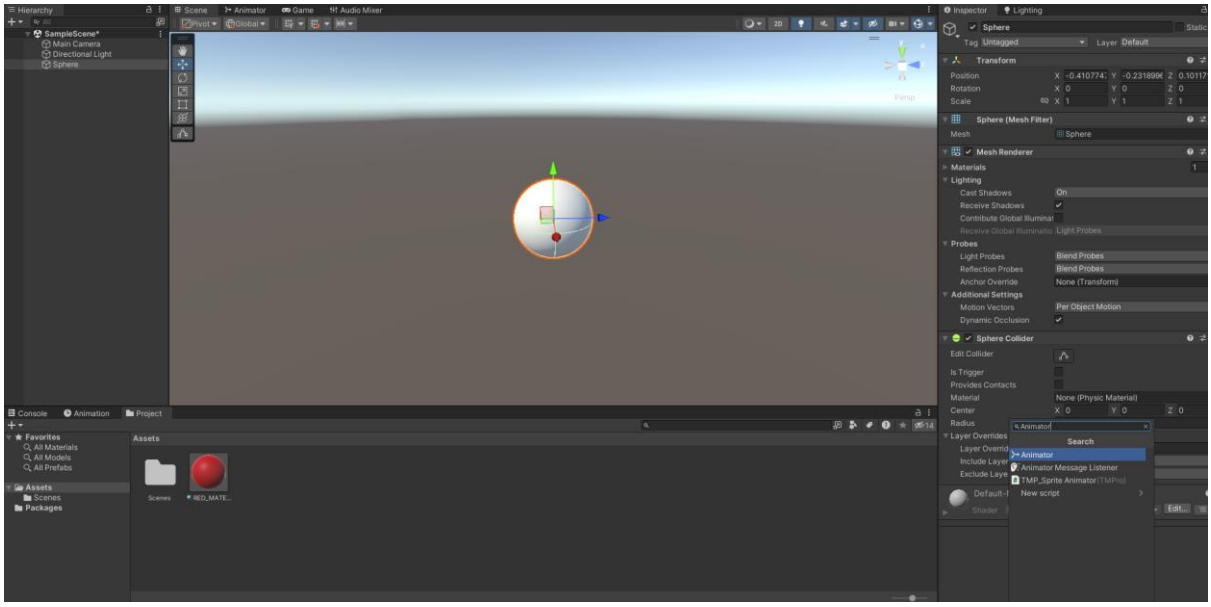
Πατώντας το κουμπί αυτό, θα εμφανιστεί μια λίστα με μια μπάρα αναζήτησης όπου μπορούμε να αναζητήσουμε το Component που επιθυμούμε να εισάγουμε στο αντικείμενο μας, ανεξάρτητα αν αυτό είναι Prefab ή όχι.

3.8.4 Animations

Ένα πολύ σημαντικό κομμάτι στα παιχνίδια για να προσφέρουν ζωντάνια και παραστατικότητα στους χαρακτήρες αλλά και στα αντικείμενα είναι τα Animations. Τα Animations μπορεί ο χρήστης να τα δημιουργήσει μέσα στο περιβάλλον Unity αλλάζοντας μεταβλητές από ιδιότητες πάνω σε αντικείμενα ή ακόμα και να τα προσθέσει από εξωτερικές πηγές.

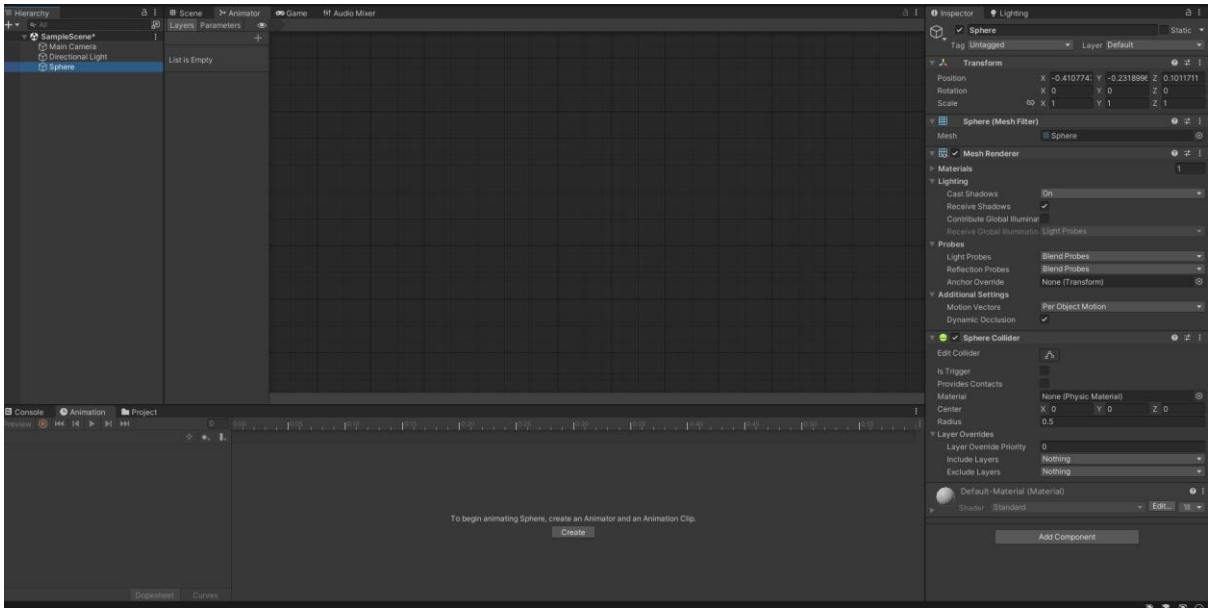
3.8.4.1 Δημιουργία Animation

Για την δημιουργία Animation το πρώτο πράγμα που χρειαζόμαστε είναι ένα οποιοδήποτε αντικείμενο στο Hierarchy. Αφού το επιλέξουμε, πρέπει να το εισάγουμε μέσω του Inspector το κατάλληλο Component με το όνομα 'Animator'.



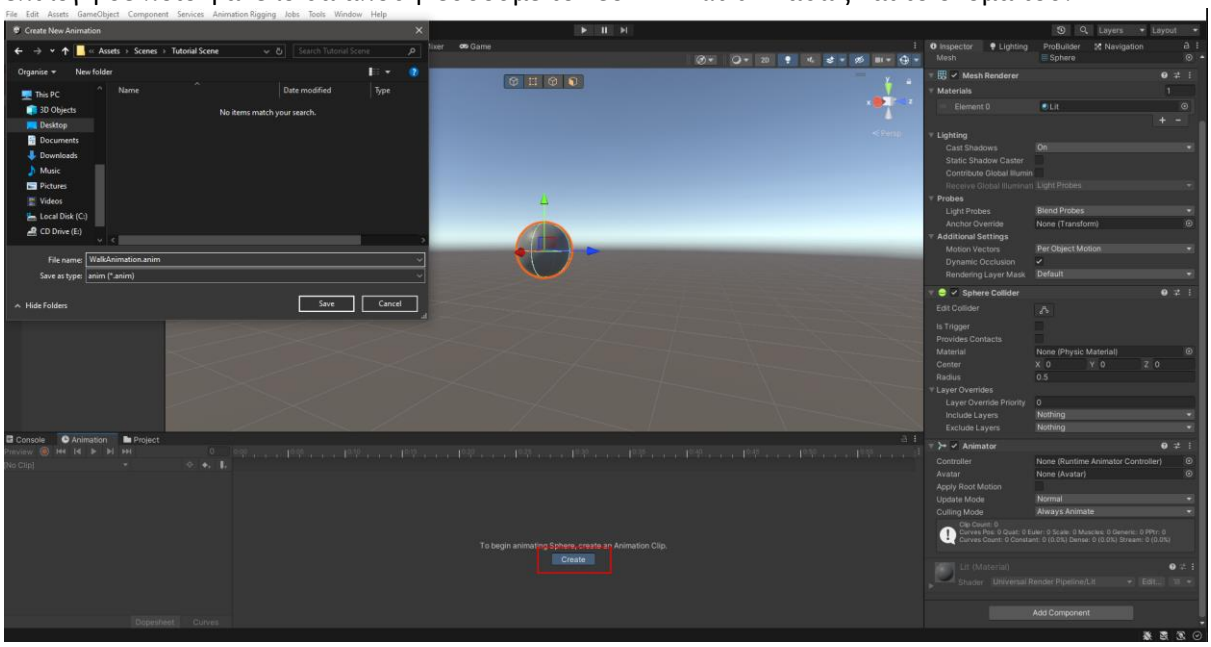
Animator Component

Αφού το προσθέσουμε στο αντικείμενο μας, ανοίγουμε το παράθυρο με όνομα Animator. Σε περίπτωση που δεν είναι ορατό στο περιβάλλον μας, μπορούμε να το εμφανίσουμε πατώντας στην κορυφή του παραθύρου του Unity : Window → Animation → Animator. Αντίστοιχα και για το παράθυρο που θα χρειαστούμε αργότερα για την δημιουργία των Animation θα πατήσουμε: Window → Animation → Animation. Αφού εμφανιστούν και ανοίξουμε και τα δύο παράθυρα θα πρέπει να επιλέξουμε το αντικείμενο το οποίο προσθέσαμε το Component: Animator. Κάνοντας τα παραπάνω βήματα θα έχουμε καταλήξει την μορφή που δείχνει η παρακάτω εικόνα:



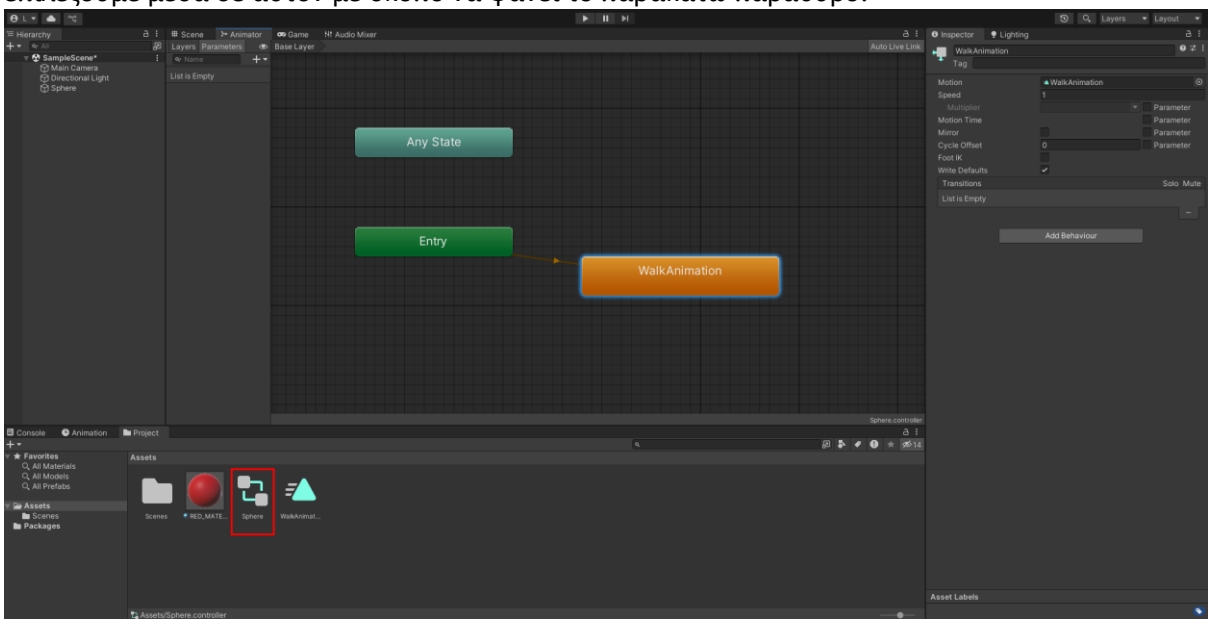
Animator & Animation Windows

Στην συνέχεια ο χρήστης πρέπει να πατήσει το κουμπί Create στο παράθυρο του Animation και η επιλογή σε ποιο φάκελο θα αποθηκεύσουμε το νέο Animation καθώς και το όνομα του.



Create Animation

Με την δημιουργία του Animation στον φάκελο που επιθυμούμε μπορούμε πλέον να το επιλέξουμε μέσα σε αυτόν με σκοπό να φανεί το παρακάτω παράθυρο:



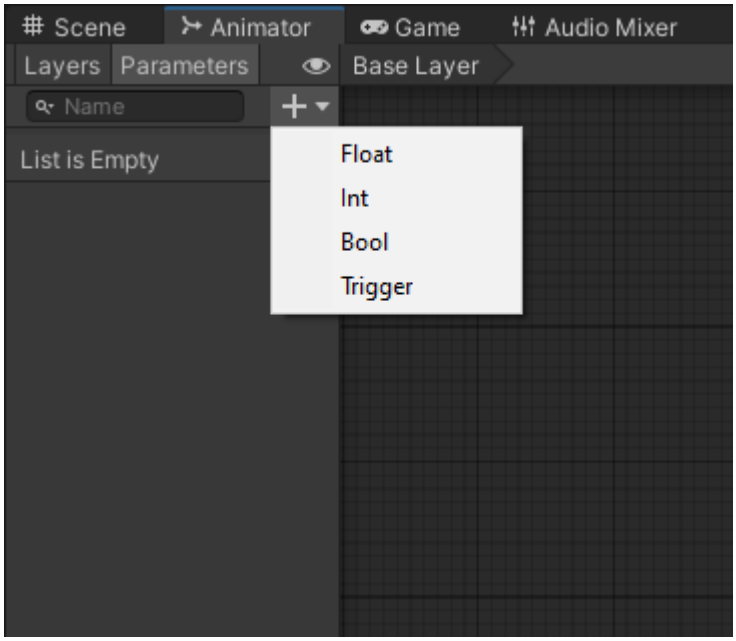
Animator Display

Στο παράθυρο του Animator μπορούμε να ρυθμίσουμε πλέον τα Animation που δημιουργούμε για το αντικείμενο μας. Όπως φαίνεται όταν το παιχνίδι ξεκινήσει θα λειτουργήσει απευθείας το Animation καθώς από το Entry → WalkAnimation (ή όπως αλλιώς το ονομάσει ο παίκτης). Αυτή η συνθήκη είναι πάντα αληθής καθώς ο χρήστης δεν έχει θέσει παραμέτρους για να ελέγχει πότε θα λειτουργεί το Animation μας.

3.8.4.2 Παράμετροι για Έλεγχο Animation

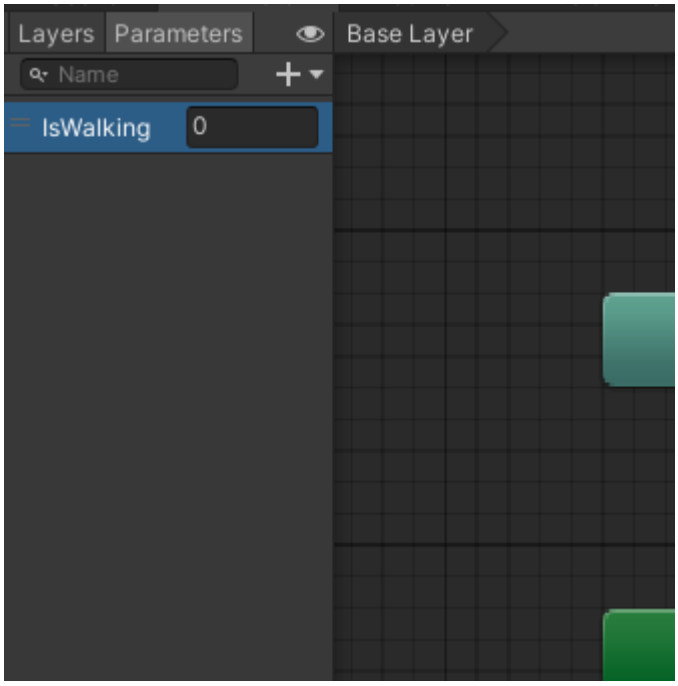
Πολλές φορές ο χρήστης θα χρειαστεί να χρησιμοποιήσει ένα Animation σε συγκεκριμένα σημεία του παιχνιδιού (π.χ. Για εναλλαγή της κίνησης του χαρακτήρα από περπάτημα σε τρέξιμο).

Για να συμβεί αυτό, μπορούμε να θέσουμε διαφόρων τύπων παραμέτρους (Integers, Float , Booleans, Toggle), με σκοπό να ελέγχουμε τα στάδια του αντικειμένου μας. Για να έχουμε πρόσβαση στις παραμέτρους πρέπει να πατήσουμε στο κουμπί Parameters που βρίσκεται πάνω αριστερά του παραθύρου Animator και στην συνέχεια το κουμπί '+' που θα βρίσκεται ακριβώς από κάτω.



Parameter Types

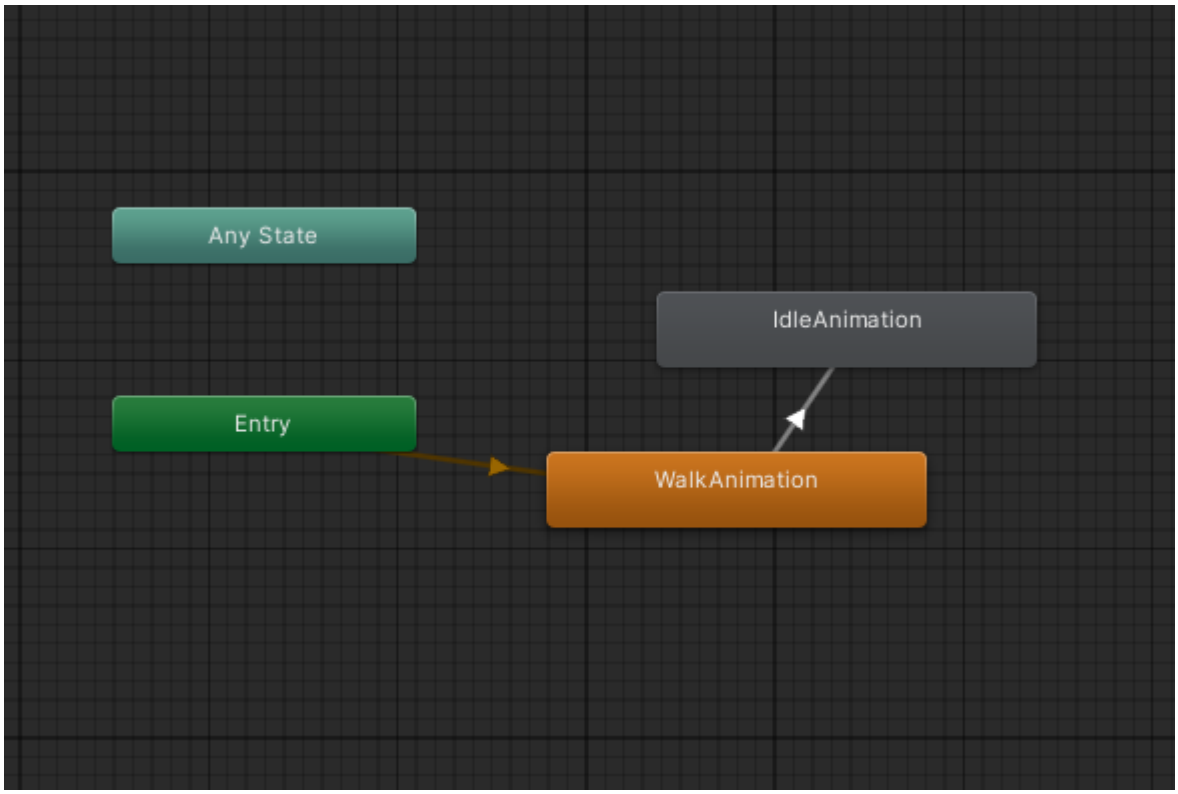
Αφού ο χρήστης επιλέξει μια κατηγορία (π.χ. Float) η παράμετρος αυτή θα εμφανιστεί στην λίστα από κάτω με αυτή την μορφή όπου και μπορεί να γίνει μετονομασία της:



Float Parameter isWalking

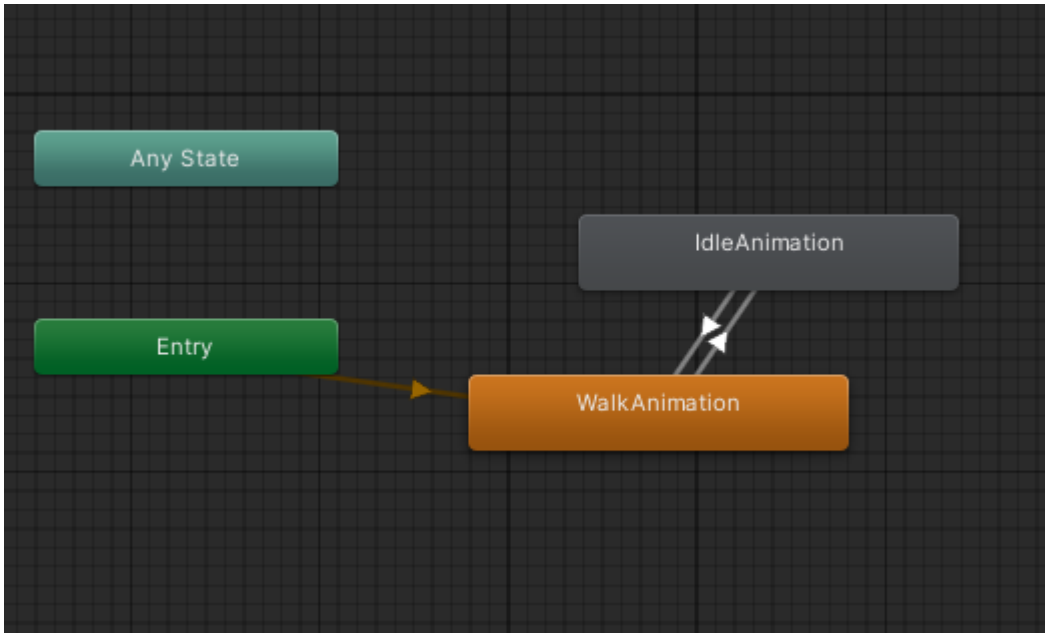
Για να μπορέσουμε πλέον να χρησιμοποιήσουμε την μεταβλητή αυτή θα πρέπει να δημιουργήσουμε συνδέσεις μεταξύ καταστάσεων του αντικειμένου ή όπως αλλιώς ονομάζονται **Transitions**. Οπότε έστω για παράδειγμα ότι θέλουμε να ελέγξουμε αν ο παίκτης μας περπατάει ή όχι. Θα δημιουργήσουμε ένα νέο Animation στο Project View με την χρήση του δεξιού κλικ → Create → Animation. Στην συνέχεια το νέο Animation το κάνουμε Drag and Drop στον Animator και θα έχουμε το αποτέλεσμα αυτής της μορφής:

Από το σημείο αυτό για να προσθέσουμε Transitions κάνουμε δεξί κλικ ένα Animation State, δηλαδή ένα από τα κουτάκια με το όνομα των Animation που φτιάξαμε, και στην συνέχεια **Make Transition** όπου και θα επιλέξουμε ως προς ποιο state θέλουμε να μεταφερθούμε.



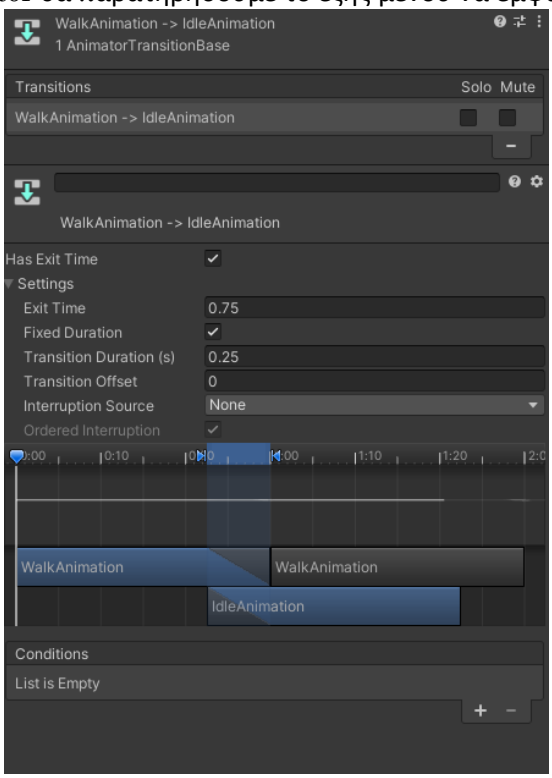
Animator

Για να ολοκληρωθεί σαν κύκλωμα η λογική των Animation πρέπει να φέρνουμε και ένα Transition πίσω στο αρχικό Animation (Στην δικιά μας περίπτωση WalkAnimation), με τον ίδιο ακριβώς τρόπο που δημιουργήσαμε πριν. Με αποτέλεσμα να προκύψει η παρακάτω εικόνα:



Animator Transition

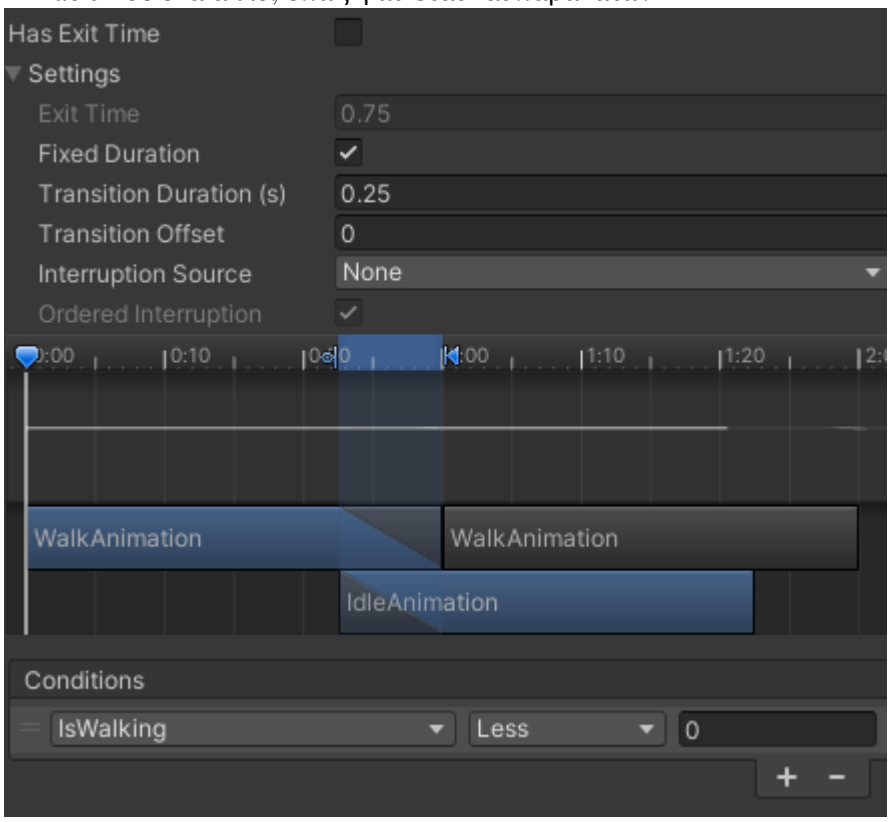
Για την ολοκλήρωση των παραμέτρων ώστε να έχουν όντως ρόλο στην εναλλαγή των Animation πρέπει να κάνουμε κλικ στα βελάκια (π.χ. στο → από WalkAnimation σε Idle Animation) και στον Inspector θα παρατηρήσουμε το εξής μενού να εμφανίζεται:



Animator Transition Settings

Στο συγκεκριμένο μενού θα μπορέσουμε να ρυθμίσουμε τις παραμέτρους μας ανάλογα με το κριτήριο που θέλουμε. Συνήθως είναι καλό να κάνουμε Uncheck το κουμπί **Has Exit Time** καθώς

προκαλεί μια καθυστέρηση στο να «παίξει» το Animation μας, καθώς και να μηδενίσουμε το πεδίο **Transition Duration (s).**
Στην συνέχεια στο πλαίσιο με την λέξη Conditions θα πατήσουμε το κουμπί με το σύμβολο '+' , όπου θα εμφανιστεί η παράμετρος μας ή αν έχουμε περισσότερες να επιλέξουμε πια θα χρησιμοποιήσουμε, και να θέσουμε την συνθήκη που θέλουμε για να είναι επιτυχής η μεταφορά από το ένα Animation σε ένα άλλο, όπως φαίνεται και παρακάτω:



Animator Conditions

Το ίδιο θα κάνουμε και για το αντίστοιχο βελάκι μεταξύ των δύο states αλλά με την αντίθετη συνθήκη για να μπορεί να γίνει σωστά η εναλλαγή.
Τέλος για δοθούν οι τιμές στις παραμέτρους αυτές ο χρήστης πρέπει μέσω Script και ειδικών συναρτήσεων να θέτει την τιμή των παραμέτρων.

π.χ. Για την αλλαγή της τιμής **IsWalking** η εντολή που θα χρησιμοποιηθεί όταν ο παίκτης πατάει το πλήκτρο για να περπατήσει θα είναι της μορφής:

animator.SetFloat("IsWalking", speed);

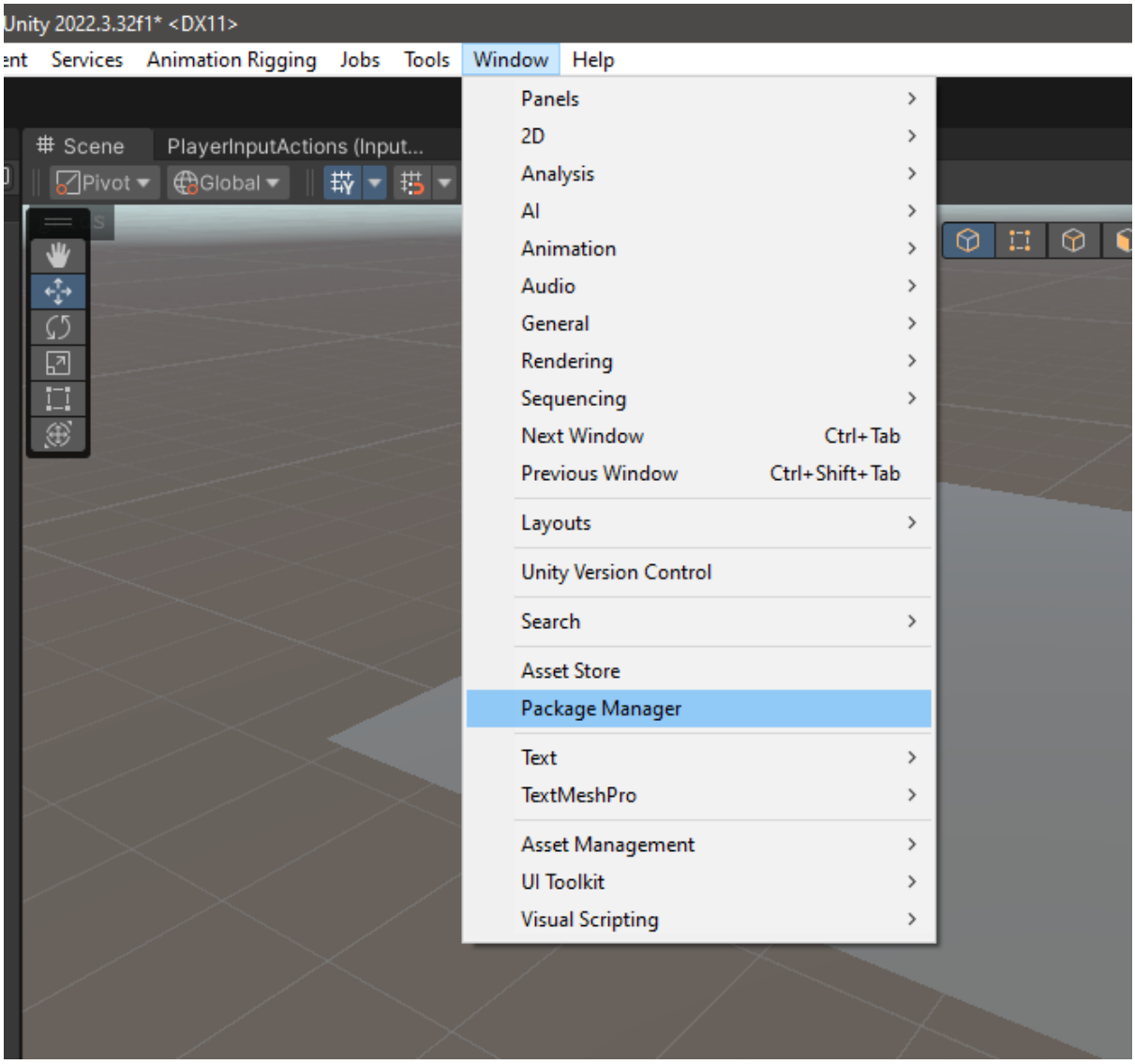
Όπου **IsWalking** πρέπει να είναι το όνομα της παραμέτρου μας , με μεγάλη προσοχή να γραφθεί σωστά αλλιώς δεν θα λειτουργήσει , και **speed** η τιμή float που θα ελέγχεται από τον Animator.

3.8.5 AI Navigation Package

Το AI Navigation Package του Unity Αποτελεί ένα ολοκληρωμένο σύνολο εργαλείων και βιβλιοθηκών που έχουν ως στόχο την υποστήριξη της ανάπτυξης αυτόνομων χαρακτήρων και πρακτόρων μέσα σε τρισδιάστατα ή δισδιάστατα περιβάλλοντα. Βασίζεται κυρίως στην χρήση της πλοήγησης μέσω του NavMesh ή αλλιώς Navigation Mesh, που είναι μια αναπαράσταση του χώρου σε μορφή πολυγωνικού πλέγματος το οποίο καθορίζει τις περιοχές που είναι βατές για κίνηση του μοντέλου που θα χρησιμοποιήσουμε. Ακόμα το πακέτο αυτό αξιοποιείται εκτενώς σε εφαρμογές εικονικής πραγματικότητας και βιντεοπαιχνίδια όπου η ομαλή, ρεαλιστική και αποδοτική πλοήγηση πρακτόρων ή αλλιώς agents, είναι κρίσιμη.

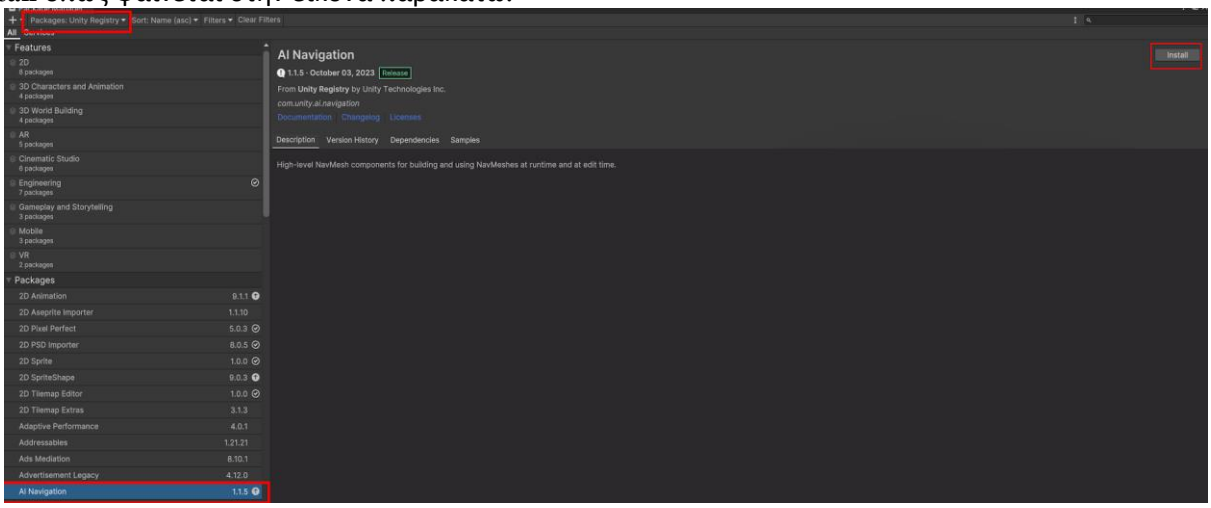
Το NavMesh Surface είναι ένα Component που επιτρέπει τη δημιουργία και διαχείριση του πλέγματος πλοήγησης με μεγαλύτερη ευελιξία.

Στο Unity για να έχουμε πρόσβαση στο πακέτο αυτό πρέπει να το εγκαταστήσουμε μέσω του PackManager , το οποίο μπορεί να βρεθεί πατώντας Window → Package Manager



Opening Package Manager

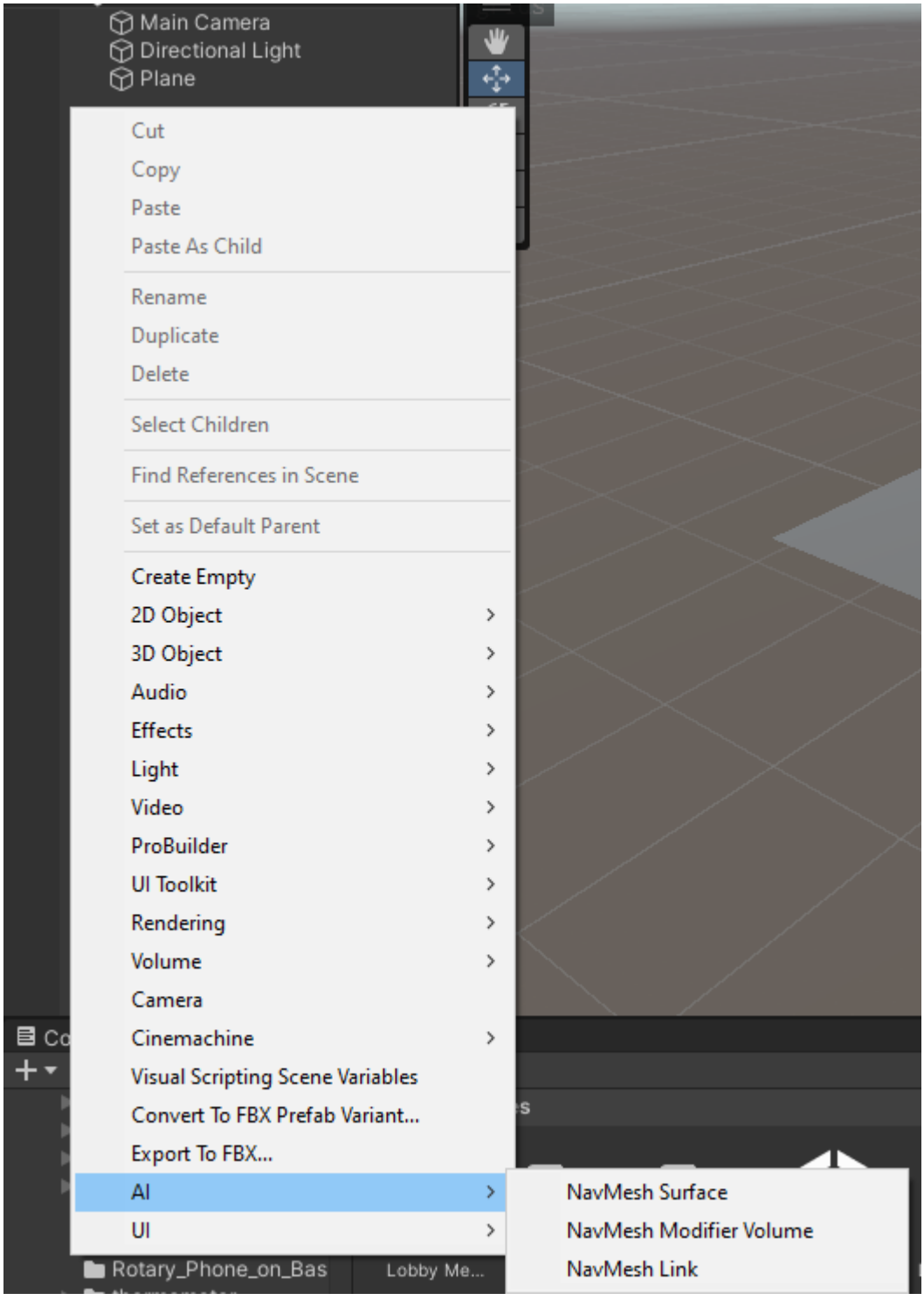
Με την επιλογή του Package Manager ο χρήστης θα πρέπει να στην τοποθεσία Unity Registry ώστε να μπορέσει να επιλέξει το πακέτο AI Navigation και να το εγκαταστήσει με το κουμπι Install όπως φαίνεται στην εικόνα παρακάτω:



Package Manager – AI Navaigation Install

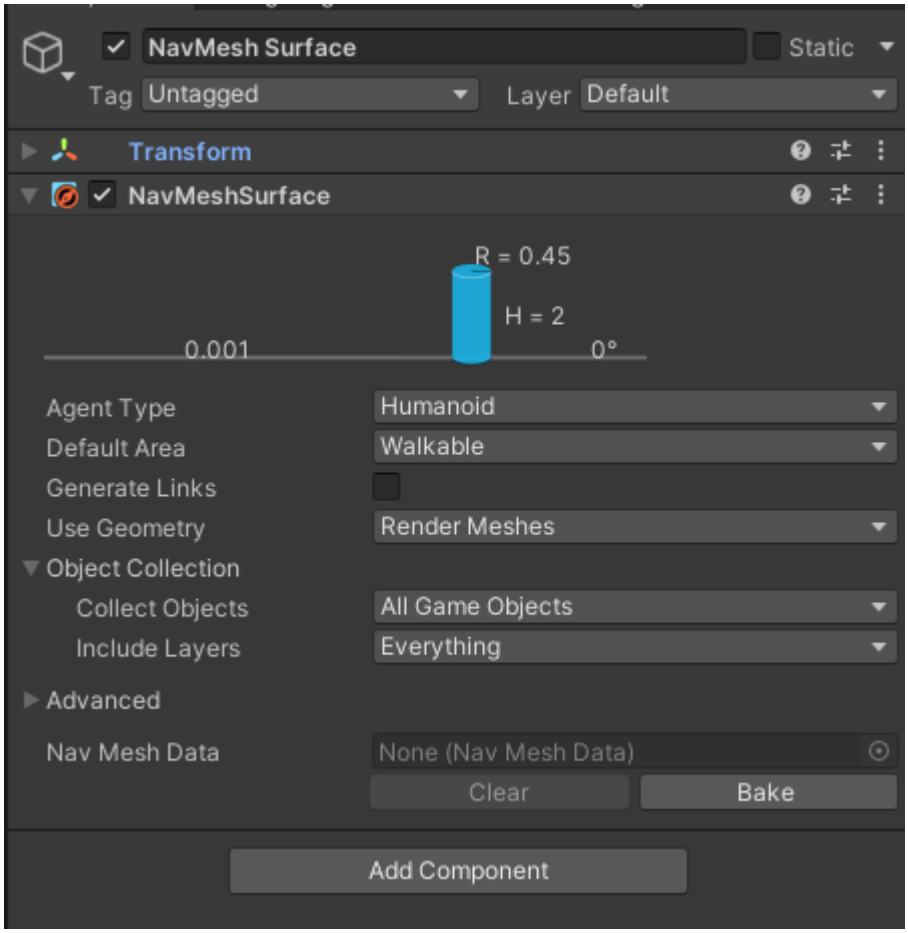
Εφόσον ολοκληρωθεί η λήψη του πακέτου αυτού, μπορούμε να κλείσουμε το παράθυρο και να μεταβούμε ξανά στην σκηνή μας.

Θα δημιουργήσουμε ένα 3D Object → Plane , ώστε να έχουμε ένα πάτωμα. Στην συνέχεια για να χρησιμοποιήσουμε το πακέτο που μόλις κατεβάσαμε , θα πατήσουμε δεξί κλικ στο Hierarchy → AI → NavMesh Surface όπως στην παρακάτω εικόνα:



NavMesh Surface Creation

Αυτή η ενέργεια θα δημιουργήσει ένα αντικείμενο στο Hierarchy με όνομα NavMesh Surface , και θα έχει ενσωματωμένο στο Inspector ένα Component με όνομα NavMeshSurface.



NavMesh Surface Component

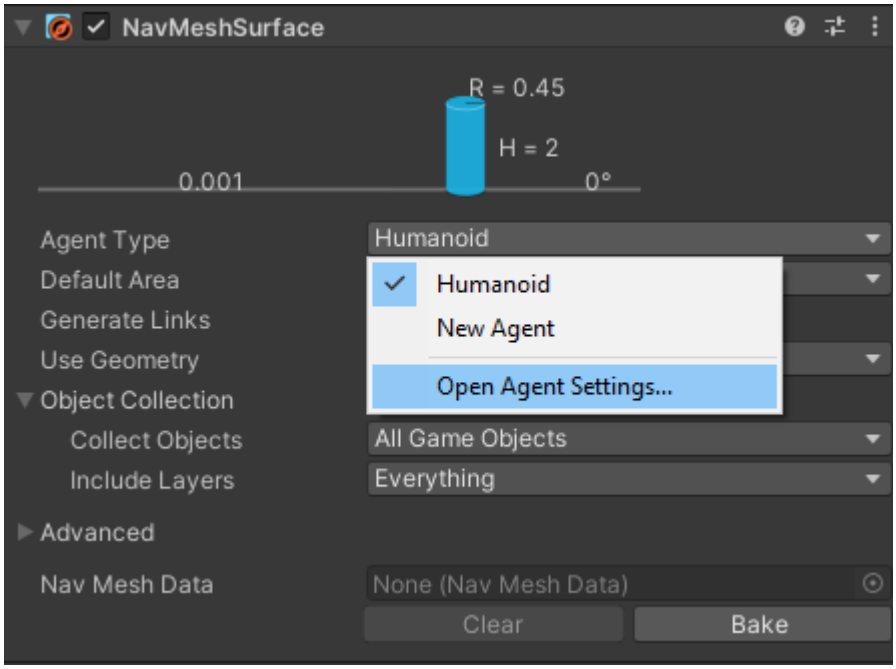
Στο συγκεκριμένο παράθυρο ο χρήστης μπορεί να επηρεάσει διάφορες μεταβλητές καθώς τον τύπο του Agent, την περιοχή που θα λαμβάνει υπόψη του για να μπορεί να βαδίσει, καθώς και ποια Layers να περιλαμβάνει μέσα στα μονοπάτια του.

Όπως φαίνεται το μοντέλο του Agent : Humanoid αποτελείται από κάποιες συγκεκριμένες τιμές στην αναπαράσταση οι οποίες αφορούν:

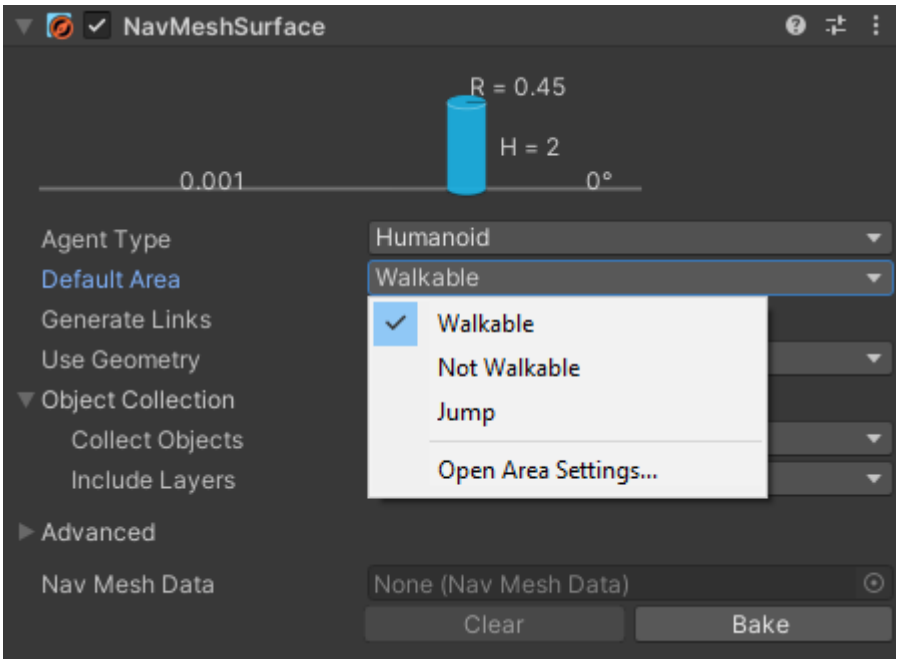
1) R = το μέγεθος του πράκτορα
2) H = το ύψος του πράκτορα
3) Step Height = 0.001, ελέγχει το ύψος του βήματος για τον πράκτορα (π.χ. για να ανέβει σκαλιά),
4) Mas Slope = 0° , ελέγχει την μέγιστη κλίση που μπορεί να κάνει βήμα

Αυτές οι τιμές δεν είναι μόνιμε καθώς ο παίκτης έχει την δυνατότητα να τις αλλάξει πατώντας πάνω στο κάθετο βελάκι στο Agent Type → Open Agent Settings.

Επιπλέον ο παίκτης έχει την δυνατότητα να διαχειριστεί τι μπορεί να θεωρηθεί περιοχή βατή για το AI πατώντας το κάθετο βελάκι στο πεδίο Default Area → Open Area Settings



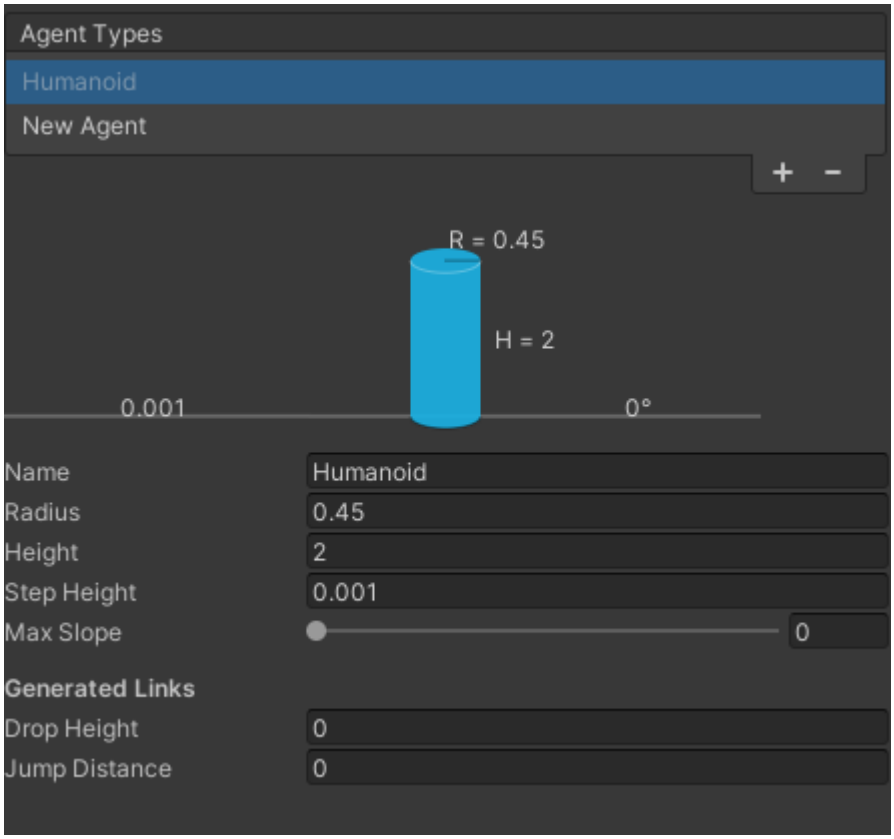
Open Agent Settings



Open Area Settings

3.8.5.1 Agent Settings

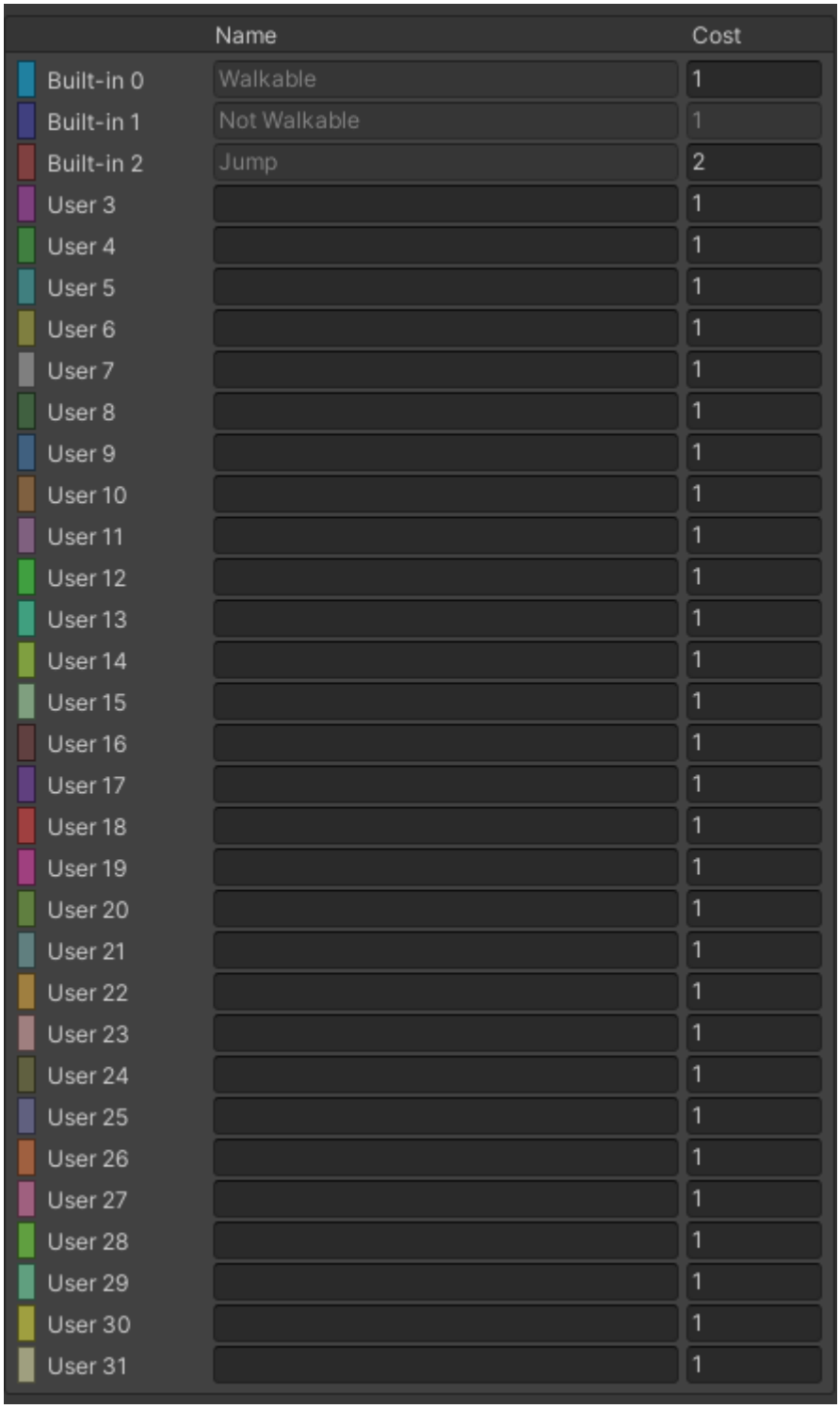
Με το πάτημα του κουμπιού ο χρήστης θα μπορεί να αλλάξει όλες τις διαθέσιμες τιμές του Agent ή και ακόμα να δημιουργήσει τον δικό του Agent με διαφορετικές τιμές πατώντας το '+' κουμπί , όπως παρουσιάζεται στην παρακάτω εικόνα.



Agent Settings

Η δημιουργία πολλαπλών Agent είναι πολύ σημαντική καθώς σε ένα παιχνίδι δεν θέλουμε όλοι οι χαρακτήρες να ακολουθούν τις ίδιες κινήσεις και μονοπάτια. Για παράδειγμα, θέλουμε διαφορετικό ύψος και step height για έναν άνθρωπο και διαφορετικό για ένα σκύλο ή μια γάτα.

3.8.5.2 Area Settings



Area Settings

Στο παράθυρο αυτό μπορούμε να παρατηρήσουμε τις επιφάνειες που είναι built-in από το πακέτο και έτσι να ορίσουμε που μπορεί να έχει πρόσβαση το AI μοντέλο μας, ή όπως φαίνεται ακόμα και να κάνει άλμα αλλά με μεγαλύτερο κόστος. Το κόστος που αναφέρεται είναι στην ουσία αφορά την δυσκολία που απαιτείται για την κίνηση στην συγκεκριμένη κατηγορία. Στην περίπτωση των built – in περιοχών, το Walkable έχει κόστος 1, ενώ το Jump έχει κόστος 2. Αυτό σημαίνει ότι στην περίπτωση που μπορούν να λειτουργήσουν και οι δύο επιλογές στο περιβάλλον, το AI θα επιλέξει

την περιοχή που είναι Walkable καθώς έχει λιγότερο βάρος. Εκτός από τα built – in μπορεί ακόμα και ο χρήστης να ορίσει δικές του κατηγορίες και το επιθυμητό βάρος (π.χ. Swim με κόστος 2.5).

ΚΕΦΑΛΑΙΟ 4 Ghost Seekers

4.1 Σενάριο του Ghost Seekers

Το παιχνίδι Ghost Seekers διαδραματίζεται σε ένα σύμπαν μυστηρίου και υπερφυσικών φαινομένων, όπου οι παίκτες καλούνται να αναλάβουν τον ρόλο επαγγελματιών ερευνητών παραφυσικών φαινομένων. Η κεντρική αφήγηση περιστρέφεται γύρω από την εξερεύνηση εγκαταλελειμμένων τοποθεσιών, στις οποίες έχουν καταγραφθεί ανεξήγητα φαινόμενα, με στόχο τον εντοπισμό, την ταυτοποίηση και εν τέλει την κατανόηση των φαινομένων αυτών.

Ο παίκτης καλείται να λύσει το μυστήριο κάθε τοποθεσίας προσπαθώντας να αποφύγει τους κινδύνους που θα δημιουργεί η οντότητα με σκοπό να εμποδίσει τον παίκτη να ολοκληρώσει την έρευνα του. Η τελική πρόκληση κάθε αποστολής απαιτεί σχεδιασμό καθώς η οντότητα μπορεί να γίνει ιδιαίτερα επιθετική όσο πλησιάζει η στιγμή της αποκάλυψης.

Το παιχνίδι αποτελείται από 4 σκηνές στο σύνολο. Οι σκηνές αυτές είναι το 1) Entry Scene, 2) Lobby Scene, 3) Map, 4) Tutorial Scene.

4.1.1. Ghost Seekers Logo

Εικόνα 4.1.1

4.2 Entry Menu

Κατά την έναρξη του παιχνιδιού, ο χρήστης θα βρίσκεται στην πρώτη Σκηνή του Project, με το όνομα Entry Scene.

Entry Menu

Στην σκηνή αυτή ο παίκτης θα έχει την δυνατότητα:

1. Να Παίξει Singleplayer
2. Να Παίξει Multiplayer (Μη Διαθέσιμο)
3. Να Μάθει το παιχνίδι παίζοντας ένα Tutorial
4. Να αλλάξει τις ρυθμίσεις του παιχνιδιού και να μάθει τα πλήκτρα του παιχνιδιού
5. Να κλείσει το Παιχνίδι

Με το πάτημα του κουμπιού 'Singleplayer', θα μεταφερθούμε στην επόμενη σκηνή με το όνομα Lobby Scene, όπου από εκεί ο παίκτης μπορεί να επιλέξει την τοποθεσία που θέλει να ερευνήσει για παραφυσικά φαινόμενα.

Ο παίκτης με την επιλογή του κουμπιού 'Settings' θα βρεθεί σε ένα νέο παράθυρο, στο οποίο θα έχει την δυνατότητα να αλλάξει τις ρυθμίσεις του παιχνιδιού του.

Οι ρυθμίσεις που προς το παρών έχει ο παίκτης στην διάθεση του:

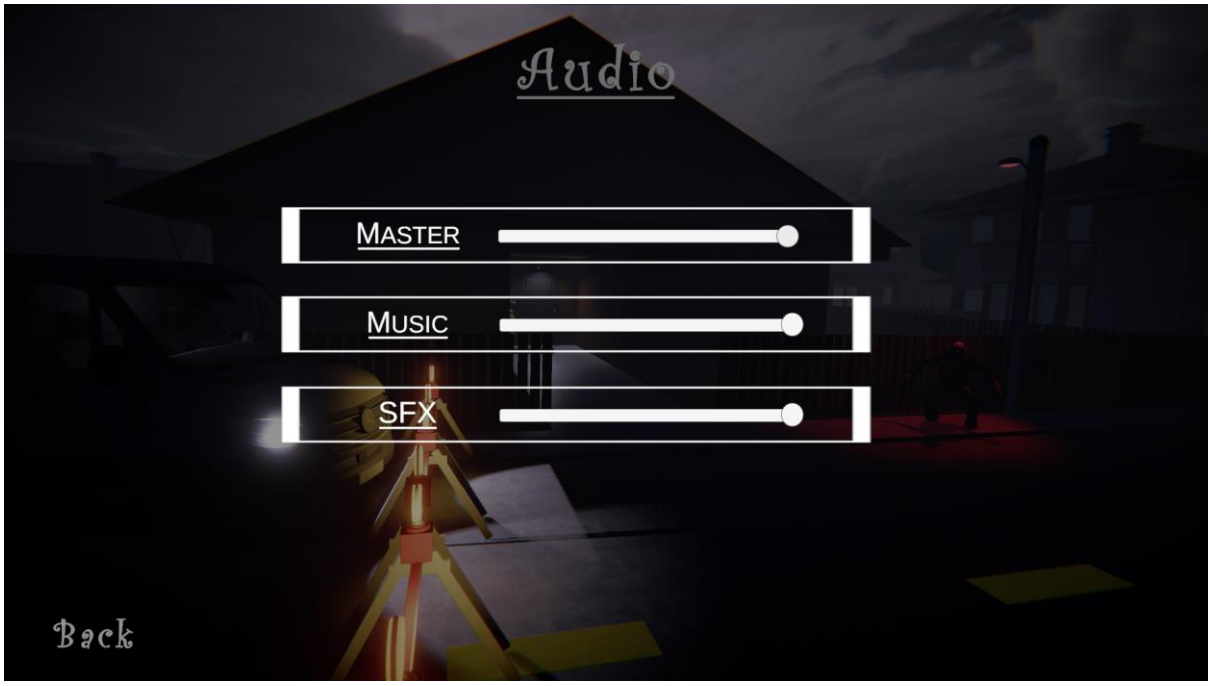
1) Graphics Settings
   - Αλλαγή Resolution
   - Αλλαγή Ποιότητας Γραφικών

- Εναλλαγή Fullscreen/Windowed

2) Audio Settings
   - Master Volume (Ελέγχει όλους τους ήχους του παιχνιδιού)
   - Music Volume (Ελέγχει την μουσική του παιχνιδιού)
   - SFX Volume (Ελέγχει τους ήχους από πιο ειδικά ηχητικά εφέ , π.χ. περπάτημα, κλείσιμο πόρτας)
3) Control Settings
   - Ο χρήστης έχει την δυνατότητα να δει τα κουμπιά και την λειτουργία τους μέσα από αυτό το παράθυρο.

Settings Menu Εικόνα

4.2.1 Audio Panel

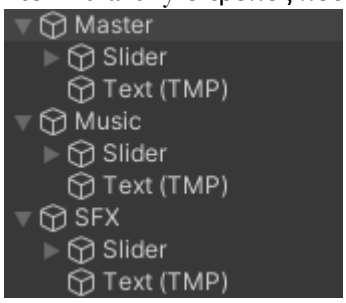


Audio Settings Εικόνα

Στο Audio Panel, μπορούμε να ρυθμίσουμε τον ήχο ανάλογα με την κατηγορία που επιθυμούμε.

Ο τρόπος με τον οποίο λειτουργεί είναι με την βοήθεια των Slider και φυσικά με την χρήση Script για την ομαλή αλλαγή της έντασης του ήχου.

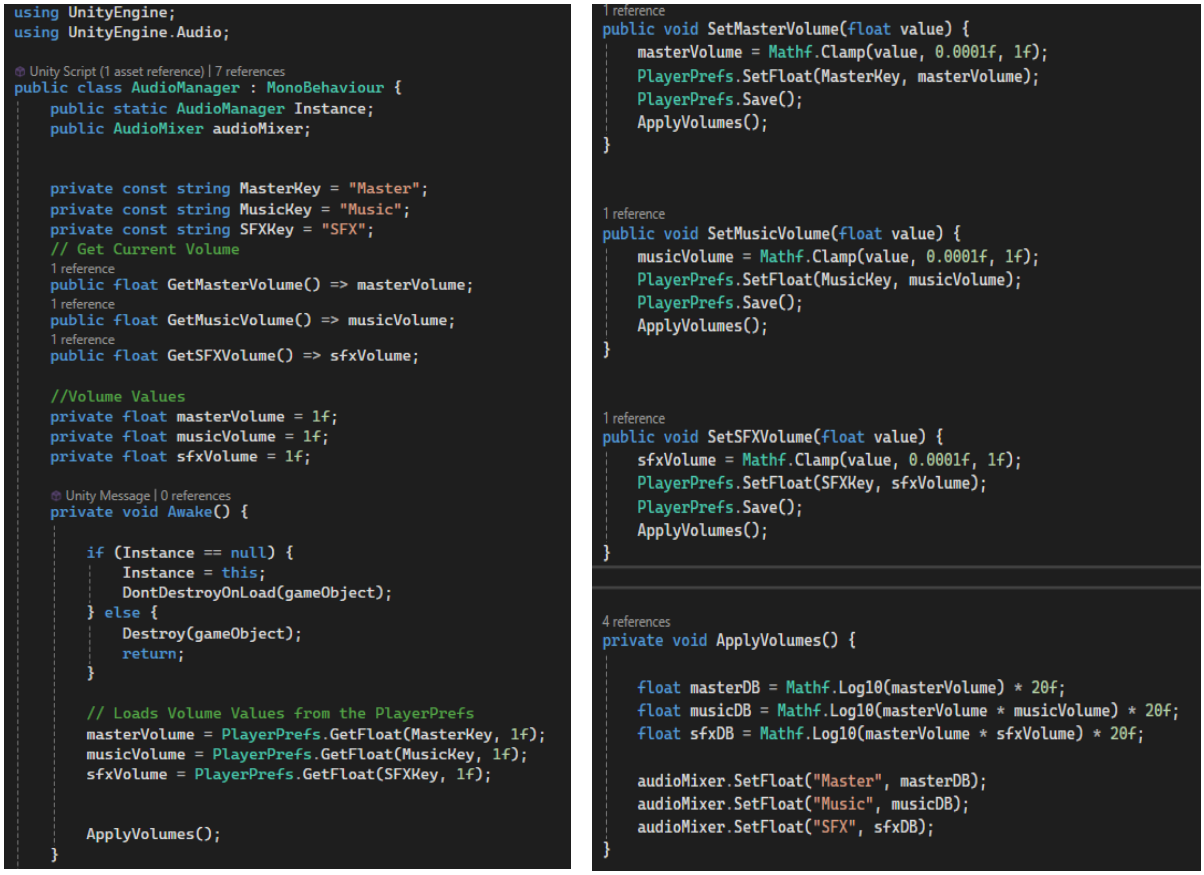
Στο Hierarchy ο τρόπος που είναι δομημένα τα Sliders είναι της παρακάτω μορφής:



Για κάθε κατηγορία έχουμε το Parent GameObject (Master, Music, SFX), και σαν Children έχουμε τα Slider, τα οποία είναι αντικείμενα UI που δημιουργούνται με την χρήση του:
Δεξί Κλικ → UI → Slider.

Το Text (TMP) χρησιμοποιείται για να εμφανιστεί στο UI ο τίτλος που ανταποκρίνεται στην κάθε κατηγορία ήχου.

Το Script που δίνει την δυνατότητα να ρυθμίσουμε και να αλλάξουμε τους ήχους είναι το AudioManager.cs



```
using UnityEngine;
using UnityEngine.Audio;

Unity Script (1 asset reference) | 7 references
public class AudioManager : MonoBehaviour {
    public static AudioManager Instance;
    public AudioMixer audioMixer;


    private const string MasterKey = "Master";
    private const string MusicKey = "Music";
    private const string SFXKey = "SFX";
    // Get Current Volume
    1 reference
    public float GetMasterVolume() => masterVolume;
    1 reference
    public float GetMusicVolume() => musicVolume;
    1 reference
    public float GetSFXVolume() => sfxVolume;

    //Volume Values
    private float masterVolume = 1f;
    private float musicVolume = 1f;
    private float sfxVolume = 1f;

    Unity Message | 0 references
    private void Awake() {

        if (Instance == null) {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        } else {
            Destroy(gameObject);
            return;
        }

        // Loads Volume Values from the PlayerPrefs
        masterVolume = PlayerPrefs.GetFloat(MasterKey, 1f);
        musicVolume = PlayerPrefs.GetFloat(MusicKey, 1f);
        sfxVolume = PlayerPrefs.GetFloat(SFXKey, 1f);


        ApplyVolumes();
    }
```

```
1 reference
public void SetMasterVolume(float value) {
    masterVolume = Mathf.Clamp(value, 0.0001f, 1f);
    PlayerPrefs.SetFloat(MasterKey, masterVolume);
    PlayerPrefs.Save();
    ApplyVolumes();
}


1 reference
public void SetMusicVolume(float value) {
    musicVolume = Mathf.Clamp(value, 0.0001f, 1f);
    PlayerPrefs.SetFloat(MusicKey, musicVolume);
    PlayerPrefs.Save();
    ApplyVolumes();
}


1 reference
public void SetSFXVolume(float value) {
    sfxVolume = Mathf.Clamp(value, 0.0001f, 1f);
    PlayerPrefs.SetFloat(SFXKey, sfxVolume);
    PlayerPrefs.Save();
    ApplyVolumes();
}


4 references
private void ApplyVolumes() {

    float masterDB = Mathf.Log10(masterVolume) * 20f;
    float musicDB = Mathf.Log10(masterVolume * musicVolume) * 20f;
    float sfxDB = Mathf.Log10(masterVolume * sfxVolume) * 20f;

    audioMixer.SetFloat("Master", masterDB);
    audioMixer.SetFloat("Music", musicDB);
    audioMixer.SetFloat("SFX", sfxDB);
}
```

AudioManager.cs

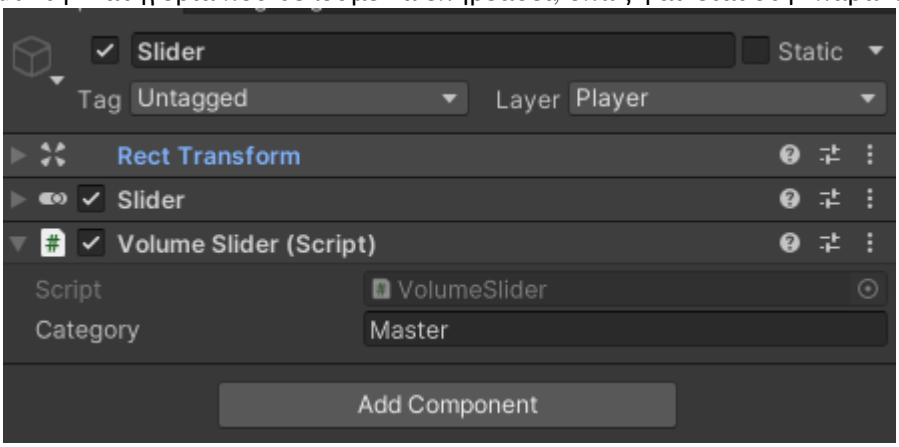
Στο συγκεκριμένο Script αρχικοποιούνται οι τιμές που είχε θέσει ο χρήστης κατά την ρύθμιση των Slider με την χρήση getter/setter , και το πρόγραμμα τις θέτει στην αρχή του παιχνιδιού με την συνάρτηση ApplyVolumes(). Η χρήση του Mathf.Log10() μέσα στην συνάρτηση ApplyVolumes χρησιμοποιείται για να είναι ομαλή η αλλαγή του ήχου χωρίς να χρειαστεί να επηρεάσουμε το AudioMixer (AudioMixer είναι το κεντρικό μενού ελέγχου των ήχων, όπου μπορούμε να διαχωρίσουμε τους ήχους σε κατηγορίες, και να επιλέξουμε κάθε αντικείμενο σε ποια κατηγορία θα 'στέλνει' τον ήχο) που έχουμε δημιουργήσει.

Για να έχουν όμως επίδραση τα Slider πάνω στον ήχο χρησιμοποιούμε το VolumeSlider.cs το οποίο επιτρέπει στην κάθε αλλαγή τιμής του Slider να δίνει την νέα τιμή στο AudioManager.cs όπου και ολοκληρώνεται η ρύθμιση της.

```
public class VolumeSlider : MonoBehaviour {
    public string category;
    private Slider slider;

    Unity Message | 0 references
    private void Start() {
        slider = GetComponent<Slider>();


        float currentValue = 1f;

        switch (category) {
            case "Master": currentValue = AudioManager.Instance.GetMasterVolume(); break;
            case "Music": currentValue = AudioManager.Instance.GetMusicVolume(); break;
            case "SFX": currentValue = AudioManager.Instance.GetSFXVolume(); break;
        }

        slider.value = currentValue;


        slider.onValueChanged.AddListener(OnSliderChanged);
    }

    1 reference
    private void OnSliderChanged(float value) {
        switch (category) {
            case "Master": AudioManager.Instance.SetMasterVolume(value); break;
            case "Music": AudioManager.Instance.SetMusicVolume(value); break;
            case "SFX": AudioManager.Instance.SetSFXVolume(value); break;
        }
    }
}
```

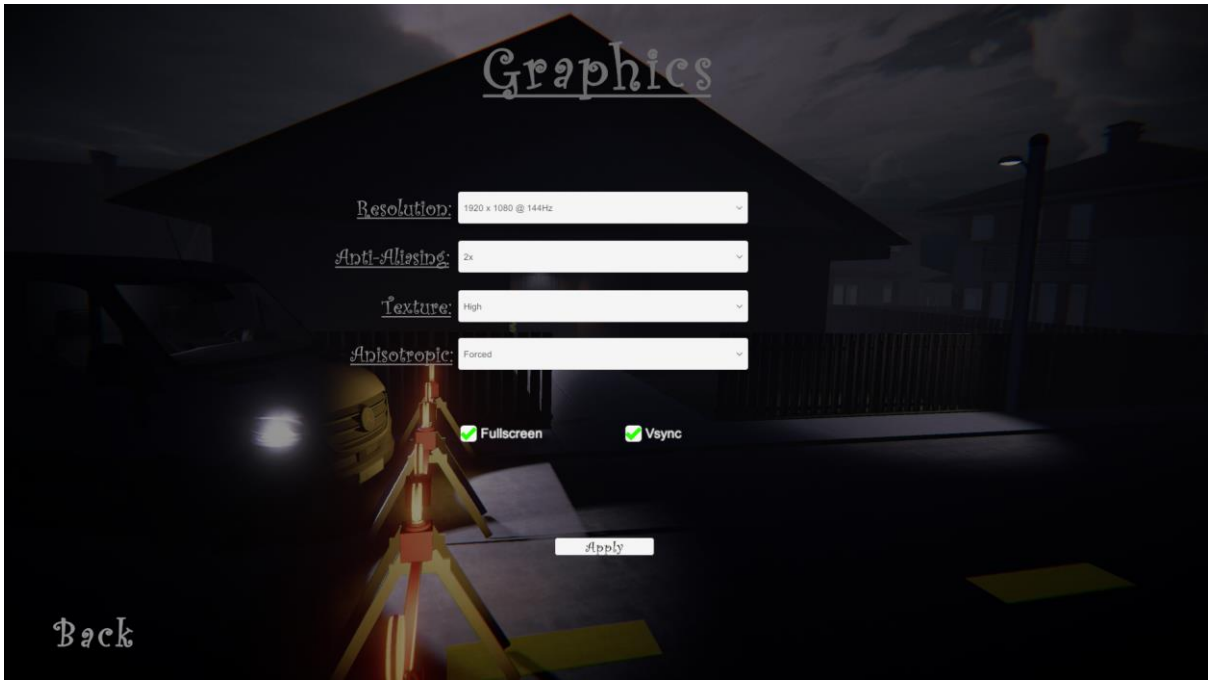
Volume Slider.cs

Το συγκεκριμένο script βρίσκεται σε κάθε Slider που αναφέραμε όπου και θέτουμε στο Inspector του κάθε Slider την κατηγορία που θέλουμε να επηρεάσει, όπως φαίνεται στην παρακάτω εικόνα:



Αντίστοιχα, για τις υπόλοιπες κατηγορίες στο πλαίσιο αυτό θα γράψουμε την κατηγορία που θα επηρεάσουν (Music, SFX).

4.2.2 Graphics Panel



Graphics Panel Εικόνα

Στο παραπάνω παράθυρο μπορούμε να αλλάξουμε τις ρυθμίσεις που αφορούν την εμφάνιση του παιχνιδιού. Οι ρυθμίσεις έχουμε την δυνατότητα να αλλάξουμε είναι:

1. Resolution
2. Anti-Aliasing
3. Texture
4. Anisotropic
5. Fullscreen Toggle
6. Vsync Toggle

Όλες οι ρυθμίσεις διαχειρίζονται από ένα script με το όνομα SettingsMenu.cs.

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

Unity Script (2 asset references) | 0 references
public class SettingsMenu : MonoBehaviour {
    public Dropdown resolutionDropdown;
    public Toggle fullscreenToggle;

    public Toggle vsyncToggle;
    public Dropdown aaDropdown;
    public Dropdown shadowDropdown;
    public Dropdown textureDropdown;
    public Dropdown anisotropicDropdown;

    private List<Resolution> resolutionsList = new List<Resolution>();

    Unity Message | 0 references
    void Start() {
        LoadResolutions();

        LoadAADropdown();
        LoadShadowDropdown();
        LoadTextureDropdown();
        LoadAnisotropicDropdown();
        LoadOtherSettings();

        LoadPlayerPrefs();
    }
```

SettingsMenu.cs

4.2.2.1 Resolution

```
1 reference
void LoadResolutions() {
    resolutionDropdown.ClearOptions();
    resolutionsList.Clear();

    Resolution[] allRes = Screen.resolutions;
    HashSet<string> addedOptions = new HashSet<string>();
    List<string> options = new List<string>();

    foreach (Resolution r in allRes) {

        string option = $"{r.width} x {r.height} @ {Mathf.RoundToInt((float)r.refreshRateRatio.value)}Hz";

        if (!addedOptions.Contains(option)) {
            addedOptions.Add(option);
            options.Add(option);
            resolutionsList.Add(r);
        }
    }

    resolutionDropdown.AddOptions(options);

    // Find current resolution index
    int currentResIndex = 0;
    for (int i = 0; i < resolutionsList.Count; i++) {

        if (Screen.currentResolution.width == resolutionsList[i].width &&
            Screen.currentResolution.height == resolutionsList[i].height &&
            Mathf.RoundToInt((float)Screen.currentResolution.refreshRateRatio.value) == Mathf.RoundToInt((float)resolutionsList[i].refreshRateRatio.value)) {
            currentResIndex = i;
            break;
        }

    }

    resolutionDropdown.value = currentResIndex;
    resolutionDropdown.RefreshShownValue();
```

LoadResolutions Function

Η συνάρτηση LoadResolutions() έχει ως σκοπό να εντοπίσει όλες τις διαθέσιμες αναλύσεις της οθόνης του συστήματος και να εμφανίσει στο Dropdown αφαιρώντας τις διπλότυπες αναλύσεις.

Πιο συγκεκριμένα, από την αρχή του κώδικα χρησιμοποιούμε την εντολή resolution.Dropdown.ClearOptions() που αφαιρεί όλες τι υπάρχουσες επιλογές από το Dropdown και στην συνέχεια το resolutionsList.Clear() που είναι η λίστα που αποθηκεύει όλα τα αντικείμενα τύπου Resolution.

Στην συνέχεια αρχικοποιούμε έναν πίνακα allRes με τις αναλύσεις που είναι δυνατή να έχει η οθόνη μας, ένα HashSet που χρησιμοποιείται για να διασφαλίσουμε ότι καμία ανάλυση δεν θα εμφανιστεί δύο φορές και τέλος μία λίστα που αποθηκεύει τις επιλογές που θα προστεθούν στο Dropdown.

Με τον βρόγχο φροντίζουμε για κάθε ανάλυση να την τυπώσει σε συγκεκριμένο format και φροντίζουμε μην είναι διπλότυπη. Και αφού γίνει ο έλεγχος τις προσθέτουμε στο Dropdown.

Τέλος , ελέγχουμε ποια από τις τρέχουσες αναλύσεις του Dropdown ανταποκρίνεται στην τρέχουσα ανάλυση της οθόνης μας καθώς και τον ρυθμό ανανέωσης και την εμφανίζουμε ως την τρέχουσα ανάλυση στο Dropdown.

4.2.2.2 Anti-Aliasing

```
1 reference
void LoadAADropdown() {
    aaDropdown.ClearOptions();
    List<string> aaOptions = new List<string>() { "Disabled", "2x", "4x", "8x" };
    aaDropdown.AddOptions(aaOptions);

    int currentAA = 0;
    switch (QualitySettings.antiAliasing) {
        case 2: currentAA = 1; break;
        case 4: currentAA = 2; break;
        case 8: currentAA = 3; break;
    }
    aaDropdown.value = currentAA;
    aaDropdown.RefreshShownValue();
}
```

LoadAADropdown Function

Ο παραπάνω κώδικας αφορά την ρύθμιση του Anti-Aliasing στο παιχνίδι μας. Για αρχή το πρόγραμμα αφαιρεί όλες τις επιλογές από το Dropdown με την χρήση του aaDropdown.ClearOptions(). Στην συνέχεια δημιουργούμε μια λίστα με προκαθορισμένες τιμές και την προσθέτουμε στο Dropdown. Τέλος βρίσκουμε την τρέχουσα ρύθμιση για το Anit-Aliasing και εμφανίζει την τιμή αυτή στο Dropdown.

4.2.2.5 Textures

```
// Texture quality options
1 reference
void LoadTextureDropdown() {
    textureDropdown.ClearOptions();
    List<string> textureOptions = new List<string>() { "Low", "Medium", "High" };
    textureDropdown.AddOptions(textureOptions);

    int currentTexture = 2 - QualitySettings.globalTextureMipmapLimit;
    textureDropdown.value = currentTexture;
    textureDropdown.RefreshShownValue();
}
```

LoadTextureDropdown Function

Όμοια με τις προηγούμενες συναρτήσεις η LoadTextureDropdown έχει ως σκοπό να φορτώσει στο Dropdown της τις τιμές που δημιουργούμε στο textureOptions ως επιλογές. Τέλος αναγνωρίζει την τρέχουσα ρύθμιση μέσω του QualitySettings.globalTextureMipmapLimit. Ο λόγος για τον οποίο χρησιμοποιούμαι την εντολή "int currentTexture = 2-QualitySettings.globalTextureMipmapLimit;"
Είναι διότι μέσω του συστήματος οι τιμές που μας δίνονται είναι της μορφής:

- 0 → High
- 1 → Medium
- 2 → Low

Οπότε ο σκοπός της εντολής που αναφέρθηκε παραπάνω είναι για να αντιστρέψει την λογική αυτή. Τέλος ενημερώνουμε το Dropdown με την τρέχουσα τιμή που αντιστοιχεί στην ρύθμιση του παιχνιδιού

4.2.2.6 Anisotropic

```
1 reference
void LoadAnisotropicDropdown() {
    anisotropicDropdown.ClearOptions();
    List<string> anisoOptions = new List<string>() { "Disabled", "Enabled", "Forced" };
    anisotropicDropdown.AddOptions(anisoOptions);

    anisotropicDropdown.value = (int)QualitySettings.anisotropicFiltering;
    anisotropicDropdown.RefreshShownValue();
}
```

LoadAnisotropicDropdown Function

Η συνάρτηση LoadAnisotropicDropdown() χειρίζεται την φόρτωση των επιλογών που δημιουργούνται στο anisoOptions και την εμφάνιση τους στο Dropdown, αφού έχουμε φροντίσει να αφαιρέσουμε τις προηγούμενες τιμές από το Dropdown.

4.2.2.7 Fullscreen & Vsync

```
1 reference
void LoadOtherSettings() {
    fullscreenToggle.isOn = Screen.fullScreen;
    vsyncToggle.isOn = QualitySettings.vSyncCount > 0;
}
```

LoadOtherSettings Function

Η συνάρτηση LoadOtherSettings() είναι υπεύθυνη για την φόρτωση και λειτουργία του Fullscreen και του Vsync.

Για την ρύθμιση fullscreen χρησιμοποιούμε μια τιμή Boolean η οποία ελέγχει αν η κατάσταση του παιχνιδιού είναι στην ρύθμιση fullscreen. Αν το παιχνίδι είναι στην κατάσταση Fullscreen τότε η τιμή που θα επιστραφεί στο fullscreenToggle.isOn είναι True διαφορετικά θα είναι False.

Για την ρύθμιση του Vsync χρησιμοποιούμε το QualitySettings.vSyncCount > 0; που μετατρέπει το αποτέλεσμα σε μορφή Boolean. Με βάση την λειτουργία του Unity το Vsync ορίζεται ως εξής:

- 0 → VSync Disabled
- 1 → Vsync Enabled

Οπότε η τιμή Boolean είναι True όταν το Vsync είναι ενεργοποιημένο και False όταν είναι απενεργοποιημένο.

4.2.2.8 Apply Settings

```
public void ApplySettings() {
    Resolution resolution = resolutionsList[resolutionDropdown.value];
    FullScreenMode screenMode = fullscreenToggle.isOn ? FullScreenMode.ExclusiveFullScreen : FullScreenMode.Windowed;

    Screen.SetResolution(resolution.width, resolution.height, screenMode, resolution.refreshRateRatio);



    QualitySettings.vSyncCount = vsyncToggle.isOn ? 1 : 0;

    int aaLevel = 0;
    switch (aaDropdown.value) {
        case 1: aaLevel = 2; break;
        case 2: aaLevel = 4; break;
        case 3: aaLevel = 8; break;
    }
    QualitySettings.antiAliasing = aaLevel;



    QualitySettings.globalTextureMipmapLimit = 2 - textureDropdown.value;

    switch (anisotropicDropdown.value) {
        case 0: QualitySettings.anisotropicFiltering = AnisotropicFiltering.Disable; break;
        case 1: QualitySettings.anisotropicFiltering = AnisotropicFiltering.Enable; break;
        case 2: QualitySettings.anisotropicFiltering = AnisotropicFiltering.ForceEnable; break;
    }
    SavePlayerPrefs();
}
```

ApplySettings Function

Η συνάρτηση ApplySettings() έχει ως σκοπό, να εφαρμόσει τις επιθυμητές αλλαγές που επέλεξε ο χρήστης από τα Dropdown που αναφέρθηκαν.

Για αρχή ορίζουμε σε μια μεταβλητή τύπου Resolution την τρέχουσα τιμή που βρίσκεται στο πεδίο του Dropdown. Στην συνέχεια ελέγχουμε την μεταβλητή fullscreenToggle.isOn και ανάλογα με το αποτέλεσμα της (True / False) θέτουμε στην μεταβλητή τύπου FullSreenMode το αποτέλεσμα μεταξύ ExculsiveFullScreen και Windowed.

Με τις νέες αυτές μεταβλητές μπορούμε πλέον να αλλάξουμε την ανάλυση της οθόνης μας με βάση το μήκος, ύψος, τον τύπο του παραθύρου και τον ρυθμό ανάλυσης.

Στην συνέχεια με βάση την μεταβλητή vSyncToggle.isOn μπορούμε να αλλάξουμε την ακέραια τιμή το vSyncCount σε 1 ή 0.

Τέλος για κάθε μια από τις κατηγορίες των Dropdown Γραφικών που αναφέραμε ελέγχουμε την κατάσταση που βρίσκεται το Dropdown της κάθε κατηγορίας γραφικών, και την θέτουμε ως την τρέχουσα ρύθμιση για το σύστημα μας.

Πριν ολοκληρωθεί η συνάρτηση ,φροντίζουμε οι αλλαγές που έγιναν να αποθηκευτούν στον PlayerPrefs για την χρήση τους σε περίπτωση επιστροφής στο παιχνίδι.

4.2.3 Controls Panel

Το παραπάνω Panel αφορά τον χειρισμό του παιχνιδιού και δίνει την δυνατότητα στον χρήστη να μάθει τα κουμπιά για να παίξει το παιχνίδι.

4.3 Tutorial Scene

Εφόσον ο παίκτης επιλέξει την επιλογή 'Tutorial' θα οδηγηθεί σε μια νέα σκηνή η οποία αφορά την εκμάθηση του παιχνιδιού. Στο Tutorial αυτό ο παίκτης θα μάθει:

- Τα διαθέσιμα εργαλεία που μπορεί να χρησιμοποιήσει στην έρευνα του για να ταυτοποιήσει το φάντασμα.
- Να μάθει πως φανερώνονται τα στοιχεία που θα αποκαλύψουν το φάντασμα.

- Τον χειρισμό του παιχνιδιού
- Τον τρόπο που αλληλοεπιδράει το φάντασμα με το περιβάλλον

Ο παίκτης θα εμφανιστεί στο Tutorial , και θα πρέπει να ακολουθήσει τους οδηγίες που του δίνονται για να μάθουν όλα όσα έχει να προσφέρει αυτή η εκμάθηση του παιχνιδιού.

Tutorial Entry Room

4.3.1 Tutorial First Room

Tutorial First Room

Στο πρώτο δωμάτιο του Tutorial ο παίκτης θα μπορέσει να μάθει για την χρήση του Φακό (Flashlight). Πιο συγκεκριμένα, θα μάθει πως:

1. Να μπορεί να σηκώσει αντικείμενα στο χέρι του (Μέχρι 3 θέσεις για αποθήκευση αντικειμένων)
2. Πώς να χρησιμοποιεί τον φακό
3. Και πώς να χρησιμοποιεί τους διακόπτες φωτός για να σβήσει τα φώτα

4.3.1.1 PickUpController Script

Ο παίκτης μπορεί να σηκώσει αντικείμενα χάρις το script PickUpController(). Το συγκεκριμένο script είναι ενσωματωμένο πάνω σε κάθε αντικείμενο το οποίο θέλουμε να επιτρέψουμε να μπορεί να σηκωθεί από τον παίκτη.

```
public class PickUpController : MonoBehaviour {
    [Tooltip("Scripts to enable when picked up, and disable when dropped.")]
    public List<MonoBehaviour> enableOnPickup;
    GameObject[] slotGameObjects;
    public Rigidbody rb;
    public BoxCollider coll;
    public Transform player, slotContainer, fpsCam;
    private Transform[] slots;
    public static GameObject currentItem = null;
    public static int currentSlotIndex = -1;




    public float pickUpRange = 3f;
    public float dropForwardForce = 2f, dropUpwardForce = 1f;

    public bool equipped;
    public static bool slotFull;

    Unity Message | 0 references
    private void Start() {
        rb = GetComponent<Rigidbody>();
        coll = GetComponent<BoxCollider>();


        if (coll == null) {
            coll = GetComponentInChildren<BoxCollider>();
        }
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
        slotContainer = GameObject.Find("Slots").GetComponent<Transform>();
        slots = new Transform[slotContainer.childCount];

        slotGameObjects = new GameObject[slotContainer.childCount];
        for (int i = 0; i < slotContainer.childCount; i++) {
            Transform child = slotContainer.GetChild(i);
            slots[i] = child;
            slotGameObjects[i] = child.gameObject;
        }
        fpsCam = GameObject.Find("Camera").GetComponent<Transform>();


        SetItemState(equipped);
    }
```

PickUpController.cs

Η λειτουργία του script είναι να δώσει την δυνατότητα στον χρήστη να μπορεί να κρατήσει στο χέρι του ορισμένα αντικείμενα, και να ενεργοποιεί ορισμένα script μόνο όταν κρατάει το αντικείμενο αυτό.

Στην συνάρτηση Start ανακτούμε τα στοιχεία από διάφορα Components του ίδιου του αντικειμένου (π.χ. Rigidbody, Collider) και ορισμένα άλλα αντικείμενα τα οποία βρίσκονται ως Components πάνω στον παίκτη μας. Πιο συγκεκριμένα:

- Η μεταβλητή player ανακτά την τοποθεσία του παίκτη με την βοήθεια της εύρεσης του Tag: Player.
- Η μεταβλητή slotContainer ανακτά την τοποθεσία του Inventory που έχει ο παίκτης έτσι ώστε όταν ο παίκτης σηκώσει ένα αντικείμενο να μπορεί να προσαρμοστεί σωστά η θέση του αντικειμένου και να φαίνεται ομοιόμορφα στην κάμερα.
- Το slots με την βοήθεια του slotsContainer που λάβαμε προηγουμένως δέχεται τον αριθμό των διαθέσιμων θέσεων στο Inventory. Στην συνέχεια θέτουμε ότι κάθε μια από αυτές τις θέσεις θα μπορεί να υποστηρίξει ένα GameObject θέτοντας τα κατάλληλα indexes με βάση τις θέσεις.
- Τέλος ανακτούμε και την θέση της κάμερας του παίκτη διότι αργότερα με την χρήση το πλήκτρου 'Q' για ρίψη του αντικειμένου, θέλουμε να κάνουμε πιο ομοιόμορφη την ρίψη αυτή.

Έτσι μετά από όλες τις αρχικοποιήσεις και τις ανακτήσεις τιμών μπορούμε να θέσουμε το state του αντικειμένου με την χρήση της SetItemState().

4.3.1.2 SetItemState Function

```
7 references
public void SetItemState(bool state) {
    foreach (var script in enableOnPickup) {
        if (script != null) {
            var remote = script as TVRemoteScript.TV_Remote_Script;
            if (remote != null) {
                remote.playerControlEnabled = state;
                continue;
            }
            script.enabled = state;
        }
    }
    rb.isKinematic = state;
    coll.isTrigger = state;
```

SetItemState Function

Στην συγκεκριμένη συνάρτηση λειτουργούμε έναν βρόγχο με σκοπό να μπορέσουμε να ενεργοποιήσουμε ή απενεργοποιήσουμε τα κατάλληλα script που είναι πάνω στο αντικείμενο μας.
Κατά την αρχή του προγράμματος επειδή ο παίκτης δεν έχει προλάβει να σηκώσει κάποιο αντικείμενο, πρώτα θέτουμε όλα τα script ανενεργά (εκτός από το PickUpController.cs) για να μην μπορεί ο παίκτης να αλληλεπιδράσει με το αντικείμενο πατώντας κάποιο κουμπί ενώ δεν κρατάει το αντικείμενο. Τέλος θέτουμε στο Component Rigidbody την μεταβλητή isKinematic σύμφωνα με την κατάσταση του αντικειμένου (Αν κρατιέται τότε είναι true, διαφορετικά False). Η ιδιότητα IsKinematic σημαίνει πως δεν επιτρέπει στο αντικείμενο να έχει βαρύτητα και έτσι είναι 'παγωμένο'. Αντίστοιχα το "coll.isTrigger = state" λειτουργεί με τον ίδιο ακριβώς τρόπο, και όταν είναι το state = True, τότε το αντικείμενο θα θεωρείται ότι δεν έχει σημεία επαφής για να κάνει σύγκρουση με το περιβάλλον.

Εφόσον έχουμε αρχικοποιήσει πλήρως πλέον τα αντικείμενα μας με τις συναρτήσεις, για να μπορέσει ο παίκτης να κρατήσει ένα αντικείμενο πρέπει να καλέσει την συνάρτηση PickUp() που

βρίσκεται μέσα στο script. Για να το κάνει αυτό χρησιμοποιείται ένα script που βρίσκεται πάνω στον παίκτη.

4.3.1.3 Camera Script

```
void Update() {
    RaycastHit hit;
    bool hitSomething = Physics.Raycast(transform.position, transform.forward, out hit, DistanceOpen, Physics.DefaultRaycastLayers, QueryTriggerInteraction.Ignore);


    if (hitSomething) {
        DoorScript.Door door = hit.transform.GetComponent<DoorScript.Door>();
        ExitDoorScript.ExitDoor exitDoor = hit.transform.GetComponent<ExitDoorScript.ExitDoor>();
        LightScript.LightSwitch lightSwitch = hit.transform.GetComponent<LightScript.LightSwitch>();
        currentPickup = hit.transform.GetComponent<PickUpScript.PickUpController>();
        ScriptVinyl.VinylScript vinyl = hit.transform.GetComponent<ScriptVinyl.VinylScript>();

        if (door != null) {
            cursor_img.sprite = cursor_focused;
            if (Input.GetKeyDown(KeyCode.E))
                door.OpenDoor();

        } else if (exitDoor != null) {
            cursor_img.sprite = cursor_focused;
            if (Input.GetKeyDown(KeyCode.E))
                exitDoor.ExitLobby(scene_index);
                exitDoor.fadeoutCanvas = fadeCanvas;
        } else if (lightSwitch != null) {
            cursor_img.sprite = cursor_focused;
            if (Input.GetKeyDown(KeyCode.E))
                lightSwitch.Switch();

        } else if (currentPickup != null) {
            cursor_img.sprite = cursor_focused;
            if (Input.GetKeyDown(KeyCode.E) && !currentPickup.equipped && !PickUpController.slotFull) {
                currentPickup.PickUp();
                equippedPickup = currentPickup; // Set the equipped item
            }

        } else if (vinyl !=null) {
            cursor_img.sprite = cursor_focused;
            if (Input.GetKeyDown(KeyCode.E))
                vinyl.CycleAudioState();

        } else {
            cursor_img.sprite = cursor_unfocused; // Hit something, but it's not interactable
            currentPickup = null;
        }
    } else {
        cursor_img.sprite = cursor_unfocused; // Didn't hit anything
        currentPickup = null;
    }

    //allow dropping independently
    if (equippedPickup != null && Input.GetKeyDown(KeyCode.Q)) {
        equippedPickup.Drop();

    }
```

CameraScript.cs

Το συγκεκριμένο script εκτός από την λειτουργία του παίκτη να μπορεί να σηκώνει αντικείμενα εμπεριέχει κα άλλες λειτουργίες που βοηθούν τον παίκτη να αλληλεπιδράσει με αρκετά ακόμα αντικείμενα (π.χ. Πόρτες, τηλεκοντρόλ, Διακόπτες), καθώς και αλλάζει το εικονίδιο στο κέντρο της οθόνης για να δείξει ότι μπορεί να γίνει αλληλεπίδραση με το αντικείμενο που σημαδεύει ο παίκτης.

Πιο συγκεκριμένα, μέσα στην Update(), κάθε frame εκτελούμε ένα Raycast με την χρήση ακτίνας στο οποίο δηλώνουμε:

1. Την έναρξη της ακτίνας
2. Την κατεύθυνση της ακτίνας που θα εξάγουμε από την κάμερα
3. Δηλώνουμε το σημείο με το "out hit" που θα συγκρουστεί η ακτίνα
4. Την απόσταση που έχουμε θέσει εμείς όπου μπορεί να ταξιδέψει η ακτίνα μέχρι να συγκρουστεί.
5. Τα layers που μπορεί να συγκρουστεί
6. Σε περίπτωση που συναντήσει κάποιο αντικείμενο με Collider που έχει θέσει την μεταβλητή IsTrigger = true, να το αγνοήσει

Στην συνέχεια ελέγχουμε με πολλαπλές "if" αν η ακτίνα μας έχει χτυπήσει κάποιο αντικείμενο που να εμπεριέχει κάποιο συγκεκριμένο script. Για παράδειγμα οι πόρτες στο παιχνίδι , έχουν όλες τους ενσωματωμένο το script Door.cs. Αν η ακτίνα συγκρουστεί με ένα τέτοιο αντικείμενο, τότε θα αλλάξει και το εικονίδιο του κέρσορα του παίκτη διότι είναι δυνατή η αλληλεπίδραση με το συγκεκριμένο αντικείμενο, αλλά και θα ενεργοποιήσει κάποιο script δεδομένου ότι ο παίκτης πατήσει το απαιτούμενο κουμπί .

Αντίστοιχα για την λειτουργία του PickUpController ο παίκτης πρέπει να βρει ένα αντικείμενο που να εμπεριέχει αυτό το script ως Component και να πατήσει το 'E' για να καλέσει την συνάρτηση PickUp(). Εφόσον σηκωθεί το αντικείμενο φροντίζουμε να αλλάξουμε την μεταβλητή equippedPickup να είναι το αντικείμενο που ακριβώς κοιτούσε ο παίκτης.

Τέλος όταν δεν υπάρχει κάποιο αντικείμενο μπροστά στον παίκτη που να μπορεί να αλληλεπιδράσει τότε και το εικονίδιο του κέρσορα επανέρχεται μέχρι να βρεθεί ξανά.

4.3.1.4 PickUp Function

```
public void PickUp() {
    bool foundSlot = false;
    int assignedSlotIndex = -1;

    // Find first empty slot
    for (int i = 0; i < slots.Length; i++) {
        if (slots[i].childCount == 0) {
            transform.SetParent(slots[i]);
            transform.localPosition = Vector3.zero;
            transform.localRotation = Quaternion.identity;
            transform.localScale = Vector3.one;
            foundSlot = true;
            assignedSlotIndex = i;
            break;
        }
    }

    if (!foundSlot) {
        Debug.LogWarning("No available slot to pick up the item.");
        slotFull = true;
        return;
    } else {
        slotFull = false;
    }

    // If there is no currently equipped item, this becomes the active one
    if (currentItem == null) {
        currentItem = gameObject;
        currentSlotIndex = assignedSlotIndex;
        gameObject.SetActive(true);
        SetItemState(true);
        equipped = true;
    } else {
        // Already have an item equipped, keep this one inactive until switched
        gameObject.SetActive(false);
        SetItemState(false);
        equipped = false;
    }
}
```

PickUp Function

Η συγκεκριμένη συνάρτηση αρχικοποιεί μια Boolean μεταβλητή foundSlot Και έναν index για να διαχειριστούμε το inventory του παίκτη καθώς και την αλλαγή αντικειμένων που κρατάμε στο χέρι.

Ο βρόγχος μας επιτρέπει να βρούμε την πρώτη θέση διαθέσιμη για να αποθηκεύσει ο παίκτης το αντικείμενο που σήκωσε ρυθμίζοντας κατάλληλα ορισμένες μεταβλητές όπως την κατεύθυνση και την περιστροφή του αντικειμένου.

Εκτός του βρόγχου ελέγχουμε αν υπάρχει διαθέσιμη θέση για να σηκωθεί το αντικείμενο, διαφορετικά δεν μπορούμε να το σηκώσουμε.

Τέλος, ελέγχουμε αν δεν υπάρχει κάποιο αντικείμενο ήδη στα χέρια του παίκτη. Σε περίπτωση που δεν υπάρχει τότε θέτουμε αυτό το αντικείμενο ως αυτό που θα φαίνεται ορατό και θα μπορεί να χρησιμοποιεί ο παίκτης.
Σε περίπτωση που ο παίκτης κρατάει ένα αντικείμενο τότε το αντικείμενο που προσπάθησε να σηκώσει ο παίκτης αποθηκεύεται στον Inventory.

4.3.1.5 Drop Function

Ο παίκτης εφόσον έχει κάποιο αντικείμενο στο χέρι του, έχει την δυνατότητα να το πετάξει με την χρήση της συνάρτησης Drop(). Η συνάρτηση καλείται από το Camera Script εφόσον ο παίκτης πατήσει το 'Q'.

```
public void Drop() {
    if (currentItem == null) {
        Debug.LogWarning("No item currently equipped to drop.");
        return;
    }

    // Drop the currently held item
    GameObject itemToDrop = currentItem;
    int slotToClear = currentSlotIndex;

    // Detach the dropped item from slot
    itemToDrop.transform.SetParent(null);

    // Enable physics and visuals on dropped item
    itemToDrop.SetActive(true);
    var pickUpCtrl = itemToDrop.GetComponent<PickUpController>();
    if (pickUpCtrl != null) {
        pickUpCtrl.SetItemState(false);
        pickUpCtrl.equipped = false;
        pickUpCtrl.rb.isKinematic = false;

        // Apply throw force
        if(itemToDrop.name != "Writing Book") {
            Vector3 throwDirection = fpsCam.forward + fpsCam.up * 0.5f;
            pickUpCtrl.rb.AddForce(throwDirection * dropForwardForce, ForceMode.Impulse);
            pickUpCtrl.rb.AddTorque(Random.insideUnitSphere * 10f, ForceMode.Impulse);
        }

    }

    // Clear the slot and current item references
    currentItem = null;
    currentSlotIndex = -1;


    CompactSlots(slotToClear);

    // Find the next available item in slots to equip
    for (int i = 0; i < slots.Length; i++) {
        if (slots[i].childCount > 0) {
            GameObject nextItem = slots[i].GetChild(0).gameObject;
            var nextPickUpCtrl = nextItem.GetComponent<PickUpController>();
            if (nextPickUpCtrl != null) {
                currentItem = nextItem;
                currentSlotIndex = i;

                nextItem.SetActive(true);
                nextPickUpCtrl.SetItemState(true);
                nextPickUpCtrl.equipped = true;

                break;
            }
        }
    }
}
```

Drop Function

Κατά την έναρξη της συνάρτησης, ελέγχουμε πρώτα αν ο παίκτης διαθέτει κάποιο αντικείμενο στα χέρια του, διαφορετικά διακόπτεται η ροή της συνάρτησης.

Εφόσον έχουμε διαβεβαιώσει ότι υπάρχει αντικείμενο για ρίψη, δεχόμαστε σε μια μεταβλητή itemToDrop το αντικείμενο που κρατάμε και σε έναν index την θέση στην οποία βρίσκεται μεταξύ των τριών που ο παίκτης έχει διαθέσιμες.

Στην συνέχεια θέτουμε ότι το αντικείμενο δεν ανήκει πλέον πάνω στον παίκτη με την αλλαγή Parent, και καλούμε ξανά την συνάρτηση SetItemState() με την διαφορά ότι σαν παράμετρο θέτουμε το state = false. Ταυτόχρονα ενημερώνουμε μεταβλητές για να δηλώσουμε την έλλειψη του αντικειμένου από το Inventory του παίκτη καθώς και την βαρύτητα του σε ενεργή.

Στην συνάρτηση αυτή κάνουμε συγκεκριμένα αναφορά στο αντικείμενο και ελέγχουμε αν είναι το "Writing Book" διότι το αντικείμενο αυτό συνδέεται με περισσότερους τρόπους στην συνάρτηση αυτή και δεν θέλουμε να του εφαρμόσουμε κάποια δύναμη που να οδηγεί σε ρίψη.

Ολοκληρώνουμε την ρίψη του αντικειμένου καλώντας την συνάρτηση CompactSlots() και ψάχνοντας το επόμενο δυνατό αντικείμενο που θέλουμε να εμφανιστεί στα χέρια το παίκτη εφόσον γίνει η ρίψη του προηγούμενου αντικειμένου

4.3.1.6 CompactSlots Function

```
void CompactSlots(int emptySlotIndex) {
    for (int i = emptySlotIndex + 1; i < slots.Length; i++) {
        if (slots[i].childCount > 0) {
            Transform item = slots[i].GetChild(0);

            // Move item to previous slot
            item.SetParent(slots[i - 1]);
            item.localPosition = Vector3.zero;
            item.localRotation = Quaternion.identity;
            item.localScale = Vector3.one;

            // If this is the currentItem, update its slot index
            if (item.gameObject == currentItem) {
                currentSlotIndex = i - 1;
            }
        } else {

            break;A
        }
    }
}
```

CompactSlots Function

Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για να αναπροσαρμόσει τα indexes των αντικειμένων καθώς ο παίκτης προσθέτει η αφαιρεί αντικείμενα στο Inventory του.

4.3.1.7 SwitchToSlot Function

```
public void SwitchToSlot(int index) {
    if (index < 0 || index >= slots.Length) {
        Debug.LogWarning("Slot index out of range");
        return;
    }

    Transform slot = slots[index];

    // Check if the slot has an item still parented there
    if (slot.childCount == 0) {
        Debug.Log("No item in this slot");
        return;
    }

    GameObject itemInSlot = slot.GetChild(0).gameObject;

    // Don't switch to the same item again
    if (itemInSlot == currentItem)
        return;

    // Deactivate previous item
    if (currentItem != null) {
        currentItem.SetActive(false);
        currentItem.GetComponent<PickUpController>()?.SetItemState(false);
    }

    // Switch to new item
    currentItem = itemInSlot;
    currentItem.SetActive(true);
    currentItem.GetComponent<PickUpController>()?.SetItemState(true);
    currentSlotIndex = index;

    Debug.Log("Switched to item in slot " + index);
}
```

```
Unity Message | 0 references
private void Update() {
    if (Input.GetKeyDown(KeyCode.Alpha1)) {
        SwitchToSlot(0);
    }
    if (Input.GetKeyDown(KeyCode.Alpha2)) {
        SwitchToSlot(1);
    }
    if (Input.GetKeyDown(KeyCode.Alpha3)) {
        SwitchToSlot(2);
    }
}
```

SwitchToSlot Function

Στην συνάρτηση Update ελέγχουμε αν ο χρήστης πατήσει μεταξύ των κουμπιών 1-2-3.

Σε οποιαδήποτε από τις παραπάνω περιπτώσεις το πρόγραμμα καλεί την συνάρτηση SwitchSlot() με παράμετρο έναν ακέραιο αριθμό από το 0 → 2.

Στην συνάρτηση ελέγχουμε αν υπάρχουν άλλα αντικείμενα στο Inventory και εφόσον υπάρχουν απενεργοποιούμε το τρέχων αντικείμενο και καλούμε την συνάρτηση για το **συγκεκριμένο** αυτό αντικείμενο και ενημερώνουμε στην συνέχεια το index των αντικειμένων. Τέλος θέτουμε για το αντικείμενο που επιλέξαμε να ενεργοποιηθεί , να ενημερώσουμε το index και να καλέσουμε ξανά την συνάρτηση SetItemState() με την παράμετρο να είναι true.

4.3.2. Tutorial Second Room

Tutorial Second Room

Στο δεύτερο δωμάτιο του Tutorial, ο παίκτης θα έρθει σε επαφή με το πρώτο του εργαλείο που θα μπορεί να αποκαλύψει ένα από τα στοιχεία.
Το στοιχείο αυτό ονομάζεται Ultaviolet, και μπορεί να φανερωθεί μόνο όταν πληρούνται τα παρακάτω κριτήρια:

1. Το φάντασμα πρέπει να διαθέτει σαν Evidences το Ultraviolet. Αν δεν το εμπεριέχει δεν είναι δυνατόν να φανερωθεί αυτό το στοιχείο
2. Το φάντασμα πρέπει να αλληλεπιδράσει με κάποιο αντικείμενο από τα παρακάτω:
   - Άνοιγμα/Κλείσιμο Πόρτας
   - Άνοιγμα/Κλείσιμο Διακόπτη Φωτός
3. Ο παίκτης έχει στην διάθεση του 20 δευτερόλεπτα για να χρησιμοποιήσει το UV Flashlight ώστε να ελέγξει την περιοχή που αλληλεπίδρασε το φάντασμα, διαφορετικά το αποτύπωμα του φαντάσματος θα εξαφανιστεί.

Στο συγκεκριμένο δωμάτιο ο παίκτης θα πρέπει να χρησιμοποιήσει τον UV Flashlight με σκοπό να ερευνήσει το δωμάτιο για τυχόν αποτυπώματα από το φάντασμα. Τα αποτυπώματα αυτά δεν θα εξαφανιστούν καθώς ο παίκτης βρίσκεται στο Tutorial.

Μορφή UV Handprint

Τα script που χρησιμοποιούνται στον UV Flashlight είναι ακριβώς τα ίδια με τα Script του Flashlight , με την διαφορά ότι στο Flashlight.cs όλοι οι UV Flashlights έχουν την Boolean μεταβλητή : IsUVFlashlight = True;

Ο τρόπος με τον οποίο μπορούμε και διαχειριζόμαστε τα αποτυπώματα ώστε να λειτουργούν μόνο με τους συγκεκριμένους φακούς είναι με το UVHandprint.cs που φαίνεται παρακάτω:

```
public class UVHandprint : MonoBehaviour {
    private Renderer rend;
    Flashlight[] flashlights;
    Unity Message | 0 references
    void Start() {
        rend = GetComponent<Renderer>();
        rend.enabled = false;
        flashlights = FindObjectsOfType<Flashlight>();
    }

    Unity Message | 0 references
    void LateUpdate() {


        bool isLitByUV = false;

        foreach (Flashlight f in flashlights) {
            if (f.isUVFlashlight && f.on && f.spotlight != null) {
                Vector3 toHandprint = transform.position - f.spotlight.transform.position;
                float angle = Vector3.Angle(f.spotlight.transform.forward, toHandprint);


                if (angle < (f.spotlight.spotAngle / 2f) && toHandprint.magnitude <= f.spotlight.range) {

                    Ray ray = new Ray(f.spotlight.transform.position, toHandprint.normalized);
                    RaycastHit[] hits = Physics.RaycastAll(ray, f.spotlight.range);

                    foreach (RaycastHit hit in hits) {
                        if (hit.collider.gameObject == gameObject) {
                            isLitByUV = true;
                            break;
                        }
                    }

                    if (isLitByUV) break;
                }
            }
        }

        rend.enabled = isLitByUV;
    }

    0 references
    public void SetHandprintVisible(bool isVisible) {
        rend.enabled = isVisible;
    }
}
```

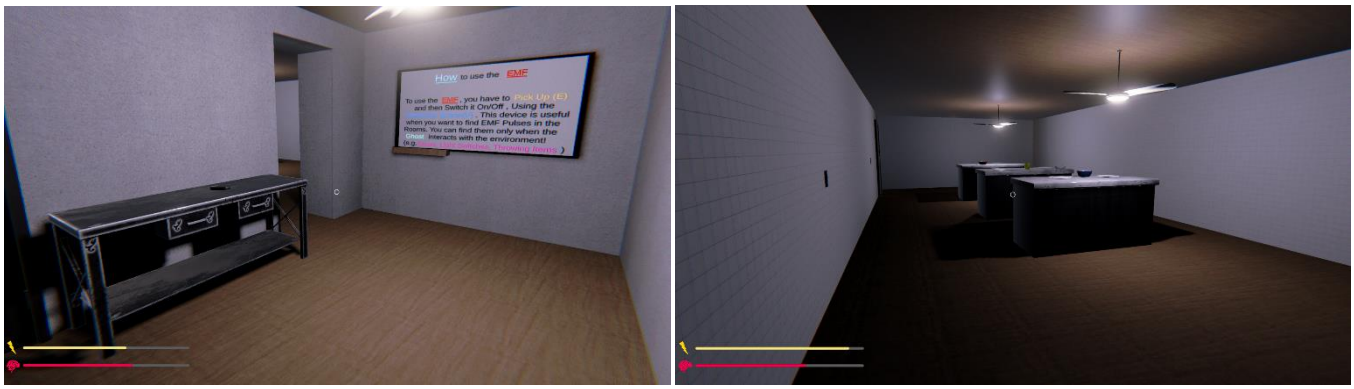
UVHanprint.cs

Το συγκεκριμένο script είναι ενσωματωμένο ως Component πάνω σε όλα τα GameObject που θέλουμε να είναι αποτυπώματα.
Στην συνάρτηση Start() , ανακτούμε την μεταβλητή τύπου Render και τις τιμές τις με την εντολή "rend = GetComponent<Renderer>();". Στην συνέχεια απενεργοποιούμε το Component Renderer από το αντικείμενο μας ώστε να είναι αρχικά αόρατο στον παίκτη. Τέλος βρίσκουμε όλα τα αντικείμενα που περιέχουν αυτό το script για να ανακτήσουμε πρόσβαση στις μεταβλητές τους.

Στην συνέχεια στην LateUpdate() θέτουμε τον renderer του αντικειμένου μας σε απενεργοποιημένο. Για κάθε φακό ελέγχουμε αν είναι UV και στην περίπτωση που ο φακός έχει την τιμή "IsUVFlashlight = True;" , Τότε υπολογίζεται η απόσταση από τον φακό προς το αντικείμενο και η γωνία μεταξύ του φακού προς το αντικείμενο. Αν το αντικείμενο βρίσκεται εντός αυτής της ακτίνας και της γωνίας, εξάγεται μια Ray από τον φακό προς το αντικείμενο. Στην περίπτωση που κάποια από τι ακτίνες συγκρουστούν με κάποιο από αυτά τα αντικείμενα που

εμπεριέχουν το script αυτό θεωρούμε ότι πλέον φωτίζετε από το UV Flashlight. Στο τέλος ο renderer του αντικειμένου ενεργοποιείται μόνο αν η συνθήκη που αφορά τον φωτισμό του είναι αληθής.

4.3.3 Tutorial Third Room



Tutorial Third Room

Στο τρίτο δωμάτιο ο παίκτης καλείται να μάθει την λειτουργία του επόμενου αντικειμένου με την ονομασία EMF Reader.
Το αντικείμενο αυτό δίνει την δυνατότητα στον παίκτη να αναγνωρίσει ένα ακόμα στοιχείο του φαντάσματος το οποίο ονομάζεται "EMF 5".

Ο παίκτης θα πρέπει να εισέλθει στο δωμάτιο της δεξιάς εικόνας, και με την χρήση του EMF Reader να αναμένει κάποια αλληλεπίδραση του φαντάσματος με το περιβάλλον.

Για αρχή θα εμβαθύνουμε στην ίδια την λειτουργία του EMF Reader καθώς και υπό ποια κριτήρια λειτουργεί και στην συνέχεια θα περιγράψουμε την περίπτωση για την εύρεση του στοιχείου.

4.3.3.1 EMF Reader

Σκοπός του EMF Reader είναι η εύρεση παραφυσικής δραστηριότητας στον περιβάλλον, εφόσον ο παίκτης έχει ενεργοποιήσει την συσκευή , και το φάντασμα έχει αλληλεπιδράσει με το περιβάλλον εκτελώντας διάφορες ενέργειες:

- Άνοιγμα/Κλείσιμο Πόρτας
- Άνοιγμα/Κλείσιμο Διακόπτη Φωτός
- Αλληλεπίδραση με Βιβλίο
- Εκτέλεση μικρού event (π.χ. κουδούνισμα τηλεφώνου)
- Ρίψη Αντικειμένου

Στην περίπτωση που κάνει οποιοδήποτε από τις παραπάνω ενέργειες, τότε θα εκτελεστούν ορισμένα script, ορισμένα εκ των οποίων θα εξηγηθούν αργότερα στην κατηγορία για το Φάντασμα.

4.3.3.2 EMF Reader Script

```
Unity Message | 0 references
private void OnTriggerEnter(Collider other) {
    if (other.CompareTag("EMF_Pulse")) {
        EMF_Pulse pulse = other.GetComponent<EMF_Pulse>();
        if (pulse != null && !activePulses.Contains(pulse) && sphere_collider.enabled == true) {
            activePulses.Add(pulse);
            UpdateEmission();
        }
    }
}

Unity Message | 0 references
private void OnTriggerExit(Collider other) {
    if (other.CompareTag("EMF_Pulse")) {
        EMF_Pulse pulse = other.GetComponent<EMF_Pulse>();
        if (pulse != null && activePulses.Contains(pulse) && sphere_collider.enabled == true) {
            activePulses.Remove(pulse);
            UpdateEmission();
        }
    }
}
```

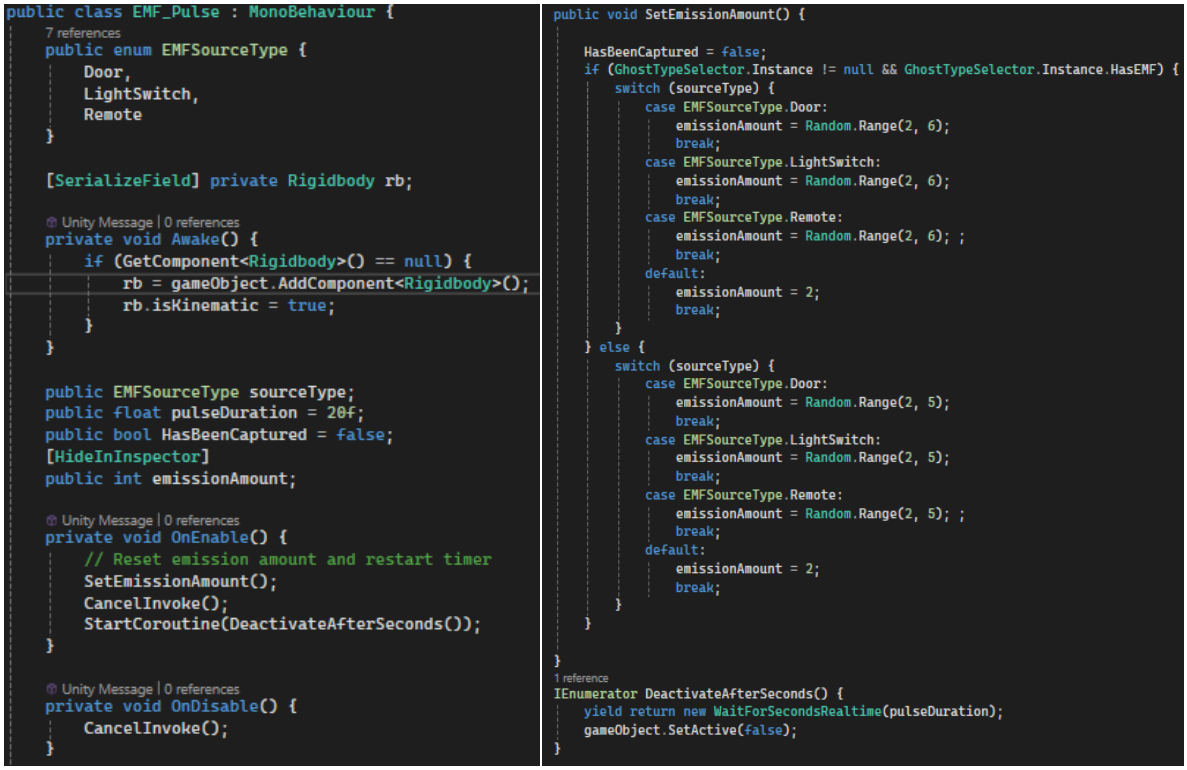
EMF TriggerEntry

Το αντικείμενο EMF Reader έχει ως Component ένα SphereCollider με την ρύθμιση να είναι Trigger, δηλαδή να μην έχει σημεία επαφής στερεά αλλά αόρατα με σκοπό να ελέγχουν αν κάποιο αντικείμενο εισέλθει ή εξέρθει μέσα από αυτό το Collider.

Έτσι στην πρώτη συνάρτηση ελέγχουμε την περίπτωση που κάποιο αντικείμενο με μια συγκεκριμένη ιδιότητα (Να έχει tag: EMF_Pulse) βρίσκεται μέσα στον Collider του EMF Reader. Στην περίπτωση που είναι τότε λαμβάνουμε υπόψη το Script με όνομα EMF_Pulse και λαμβάνουμε τις τιμές και μεταβλητές του από το κάθε αντικείμενο.
Στην συνέχεια εφόσον βρέθηκε ένα τέτοιο αντικείμενο προσθέτουμε σε μια λίστα τον αριθμό των Pulse που είναι ακέραιος αριθμός και προκύπτει από το EMF_Pulse Script.

Αντιθέτως κατά την έξοδο του αντικειμένου από το SphereCollider που ελέγχει τον τύπο των αντικειμένων, εκτελούμε την ακριβώς αντίθετη διαδικασία αφαιρώντας από την λίστα activePulses το συγκεκριμένο επίπεδο pulse του αντικειμένου.

4.3.3.3 EMF_Pulse Script



```
public class EMF_Pulse : MonoBehaviour {
    7 references
    public enum EMFSourceType {
        Door,
        LightSwitch,
        Remote
    }

    [SerializeField] private Rigidbody rb;

    Unity Message | 0 references
    private void Awake() {
        if (GetComponent<Rigidbody>() == null) {
            rb = gameObject.AddComponent<Rigidbody>();
            rb.isKinematic = true;
        }
    }

    public EMFSourceType sourceType;
    public float pulseDuration = 20f;
    public bool HasBeenCaptured = false;
    [HideInInspector]
    public int emissionAmount;

    Unity Message | 0 references
    private void OnEnable() {
        // Reset emission amount and restart timer
        SetEmissionAmount();
        CancelInvoke();
        StartCoroutine(DeactivateAfterSeconds());
    }

    Unity Message | 0 references
    private void OnDisable() {
        CancelInvoke();
    }
```

```
public void SetEmissionAmount() {

    HasBeenCaptured = false;
    if (GhostTypeSelector.Instance != null && GhostTypeSelector.Instance.HasEMF) {
        switch (sourceType) {
            case EMFSourceType.Door:
                emissionAmount = Random.Range(2, 6);
                break;
            case EMFSourceType.LightSwitch:
                emissionAmount = Random.Range(2, 6);
                break;
            case EMFSourceType.Remote:
                emissionAmount = Random.Range(2, 6); ;
                break;
            default:
                emissionAmount = 2;
                break;
        }
    } else {
        switch (sourceType) {
            case EMFSourceType.Door:
                emissionAmount = Random.Range(2, 5);
                break;
            case EMFSourceType.LightSwitch:
                emissionAmount = Random.Range(2, 5);
                break;
            case EMFSourceType.Remote:
                emissionAmount = Random.Range(2, 5); ;
                break;
            default:
                emissionAmount = 2;
                break;
        }
    }

}
1 reference
IEnumerator DeactivateAfterSeconds() {
    yield return new WaitForSecondsRealtime(pulseDuration);
    gameObject.SetActive(false);
}
```

EMF_Pulse.cs

Το συγκεκριμένο Script είναι υπεύθυνο για την σωστή λειτουργία τον EMF_Pulses που δρουν πάνω σε όλα τα αντικείμενα που μπορεί το φάντασμα να αλληλεπιδράσει.
Ενσωματώνεται πάνω σε οποιοδήποτε αντικείμενο και απλώς θέτουμε με βάση το προκαθορισμένο enum EMFSourceType τι τύπο αντικειμένου θα αφορά στο Inspector.

Στην αρχή του παιχνιδιού, ελέγχουμε εάν το αντικείμενο που έχουμε εμπεριέχει Component Rigidbody και στην περίπτωση που δεν υπάρχει το εισάγουμε μέσω του script.

Το script από την αρχή βρίσκεται ενσωματωμένο πάνω σε ένα αντικείμενο που είναι Child στο βασικό αντικείμενο με όνομα EMF_Pulse και tag : EMF_Pulse.

Όσα αντικείμενα θέλουμε να μπορούν να εξάγουν κάποια ένδειξη από το EMF Reader πρέπει να βρίσκονται σε αυτή την μορφή στο Hierarchy :

(Parent) Αντικείμενο/Μοντέλο Αντικειμένου
|
|
(Child) EMF_Pulse(With Tag: EMF_Pulse)

Η εντολή CancelInvoke Χρησιμοποιείται για να σταματήσουμε κάθε είδος λειτουργία να εξάγεται από αυτό το script.
Τέλος ξεκινάμε ένα Coroutine για να δηλώσουμε ότι θα διατηρήσουμε το συγκεκριμένο επίπεδο EMF σταθερό για 20 δευτερόλεπτα. Από εκεί και πέρα το φάντασμα πρέπει να αλληλεπιδράσει ξανά με το ίδιο αντικείμενο για να ενεργοποιηθεί νέο Pulse.

4.3.3.4 UpdateEmission – EMF Reader Script

```
private void Update() {
    // Remove deactivated pulses
    bool changed = activePulses.RemoveAll(p => p == null || !p.gameObject.activeInHierarchy) > 0;
    if (changed) {
        UpdateEmission();
    }
}

3 references
void UpdateEmission() {
    int totalEmission = 0;

    foreach (var pulse in activePulses) {
        totalEmission += pulse.emissionAmount;
    }

    if (totalEmission > 0 && (!emissionActive || currentEmissionLevel != totalEmission)) {
        ApplyEmission(totalEmission);
        emissionActive = true;
        currentEmissionLevel = totalEmission;
    } else if (totalEmission == 0 && emissionActive) {
        DisableAllEmission();
        emissionActive = false;
        currentEmissionLevel = 0;
    }
}
```

UpdateEmission Function

Η συνάρτηση αυτή εκτελείται στην Update() και είναι υπεύθυνη για να ενημερώνει κατάλληλα για το κάθε πιθανό αντικείμενο τον αριθμό τον Pulse που δέχτηκε από το EMF_Pulse.cs. Εφόσον δεχτεί την τιμή αυτή, ενημερώνει την συσκευή EMF για να εμφανίζει κατάλληλα το επίπεδο παραφυσικής δραστηριότητας με τις διαθέσιμες ενδείξεις από φωτάκια του EMF. Για παράδειγμα:

Αριστερή Εικόνα: EMF Κλειστό
Μεσαία Εικόνα: EMF Ανοιχτό – Μόνιμα στο Level 1
Δεξιά Εικόνα: EMF- Εντοπισμός Αντικειμένου με Pulse Level 4

Το EMF Reader έχει την δυνατότητα να φτάσει μέχρι το Level 5. Για να συμβεί αυτό όμως προϋποτίθεται το φάντασμα να έχει την δυνατότητα να το παράγει.

Έτσι οδηγούμαστε στο δεύτερο Evidence που μπορεί να εμφανίσει το φάντασμα, και το πώς μπορούμε να ερευνήσουμε την ύπαρξη του με την συσκευή EMF Reader.

4.3.3.5 ApplyEmission Function

```
void ApplyEmission(int amount) {

    foreach (var audio in audioSources) {
        audio.Stop();
    }


    int relativeIndex = amount - 2;


    if (relativeIndex >= 0 && relativeIndex < audioSources.Count) {
        audioSources[relativeIndex].Play();
    }


    for (int i = 0; i < materialsToToggle.Count; i++) {
        if (i <= relativeIndex) {
            materialsToToggle[i].EnableKeyword("_EMISSION");
        } else {
            materialsToToggle[i].DisableKeyword("_EMISSION");
        }
    }
}
```

ApplyEmission Function

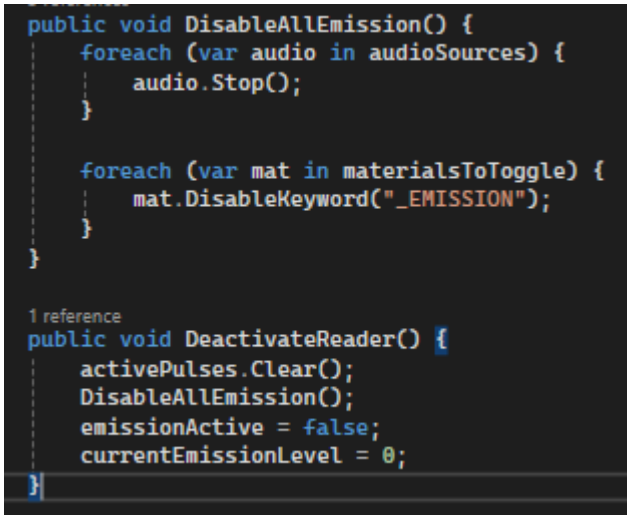
Η παραπάνω συνάρτηση είναι υπεύθυνη για την εμφάνιση του κατάλληλου επιπέδου πάνω στην συσκευή EMF Reader. Για αρχή σταματάει όλους τους ήχους που προκαλούνται από το EMF Reader με την χρήση του βρόγχου.

Κατά την ανάκτηση τιμών από το script EMF_Pulse οι τιμές του emissionpulse ξεκινάνε από το 2 →5 ή 2 → 6(Στην περίπτωση που το φάντασμα έχει σαν στοιχείο το EMF Level 5), επειδή όμως λειτουργούμε μία λίστα οι τιμές που θέλουμε να αλληλεπιδράμε είναι από 0 → 3 ή 0 → 4. Για αυτό τον λόγο αφαιρούμε από το amount τον αριθμό 2 για να ανταποκρίνεται στις τιμές αυτές.

Επομένως, αφού προσαρμόσουμε σωστά την τιμή του amount, ενεργοποιούμε και τον κατάλληλο ήχο που ανταποκρίνεται στον επίπεδο EMF που ανακτήθηκε.

Οπότε με τον τελευταίο βρόγχο, για κάθε ένδειξη που δόθηκε από το EMF_Pulse.cs αλλάζουμε την ιδιότητα Emission σε ενεργή για να παρομοιάσουμε την ένδειξη «φωτισμού».

4.3.3.6 Disable Emission & DeactivateReader Functions



```
public void DisableAllEmission() {
    foreach (var audio in audioSources) {
        audio.Stop();
    }

    foreach (var mat in materialsToToggle) {
        mat.DisableKeyword("_EMISSION");
    }
}

1 reference
public void DeactivateReader() {
    activePulses.Clear();
    DisableAllEmission();
    emissionActive = false;
    currentEmissionLevel = 0;
}
```

Disable Emission & DeactivateReader Functions

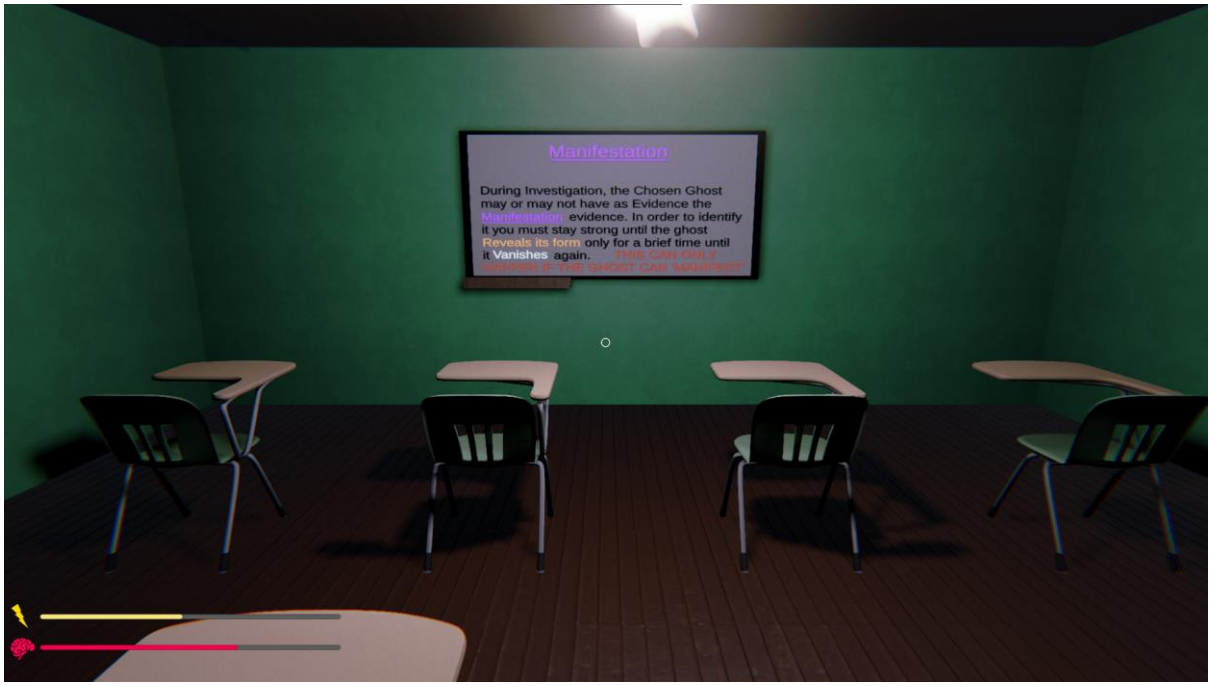
Τέλος, χρησιμοποιούμε δύο συναρτήσεις οι οποίες λειτουργούν μόνο όταν ο παίκτης απενεργοποιεί το EMF Reader. Ο τρόπος με τον οποίο ο παίκτης μπορεί να λειτουργήσει την συσκευή αυτή είναι με το παρακάτω script.

```
public class EMF_Reader_Toggle : MonoBehaviour {
    public GameObject targetObject;
    public GameObject emissionObject;
    public Material material1;
    public Material material2;
    public SphereCollider sphere_collider;
    private bool isFirstMaterial = true;
    public EMF_Reader_Script emfReader;

    Unity Message | 0 references
    private void Start() {
        sphere_collider.enabled = false;
    }
    Unity Message | 0 references
    void Update() {
        if (Input.GetKeyDown(KeyCode.F)) {
            ToggleMaterialAndEmission();
        }
    }

    1 reference
    void ToggleMaterialAndEmission() {
        if (targetObject == null || material1 == null || material2 == null || emissionObject == null) {
            Debug.LogWarning("Assign all objects and materials.");
            return;
        }

        //---MATERIAL TOGGLE---
        MeshRenderer targetRenderer = targetObject.GetComponent<MeshRenderer>();
        if (targetRenderer == null) return;

        Material newMaterial = isFirstMaterial ? material2 : material1;
        targetRenderer.material = newMaterial;

        //---EMISSION TOGGLE---
        MeshRenderer emissionRenderer = emissionObject.GetComponent<MeshRenderer>();
        if (emissionRenderer == null) return;

        Material emissionMat = emissionRenderer.material;

        if (emissionMat.IsKeywordEnabled("_EMISSION"))

            emissionMat.DisableKeyword("_EMISSION");
        else
            emissionMat.EnableKeyword("_EMISSION");

        bool isNowEnabled = !sphere_collider.enabled;
        sphere_collider.enabled = isNowEnabled;

        if (sphere_collider.enabled == false ) {
            emfReader.DisableAllEmission();
            emfReader.DeactivateReader();
        }


        isFirstMaterial = !isFirstMaterial;
    }
}
```

Ο παίκτης με την χρήση του πλήκτρου ‘F’ μπορεί να ενεργοποιήσει και να απενεργοποιήσει το EMF Reade. Κατά την ενεργοποίηση του, αλλάζουμε την ιδιότητα του EMISSION σε true μόνο στην πρώτη ένδειξη του EMF (Level 1) και ενεργοποιούμε ένα SphereCollider σε μορφή IsTrigger το οποίο χρησιμοποιείται για να μπορέσουμε να σκανάρουμε τα αντικείμενα στο περιβάλλον.

Αντίθετα κατά την απενεργοποίηση του EMF, το SphereCollider απενεργοποιείται και καλούμε τις δύο συναρτήσεις που αφορούν όλα τα επίπεδα ένδειξης του EMF όσο η συσκευή μας παραμένει κλειστή.

4.3.4 Tutorial Fourth Room



Tutorial Fourth Room Classroom

Tutorial Fourth Room Test Area

Στην επόμενη δοκιμασία ο παίκτης θα πρέπει να έρθει σε επαφή με το επόμενο στοιχείο που μπορεί να αναδείξει το φάντασμα κατά την διάρκεια του παιχνιδιού. Το στοιχείο αυτό έχει την ονομασία 'Manifestation'. Στην ουσία το φάντασμα έχει την δυνατότητα να εμφανίσει την μορφή του στον παίκτης για ένα πολύ μικρό χρονικό διάστημα(Συγκεκριμένα 1.5 δευτερόλεπτα). Αυτό γίνεται τυχαία χρησιμοποιώντας ένα script που είναι ενσωματωμένο πάνω στο φάντασμα με

όνομα GhostAppearEvidence.cs . Θα γίνει πιο λεπτομερής αναφορά σε όλα τα script του φαντάσματος σε παρακάτω κεφάλαιο.

4.3.5 Tutorial Fifth Room

Tutorial Fifth Room

Στο επόμενο στάδιο του Tutorial ο παίκτης καλείται να μάθει για ένα ακόμα εργαλείο που έχει στην διάθεση του και αυτό είναι το θερμόμετρο.

Το θερμόμετρο αποτελεί ένα από τα πιο χρήσιμα εργαλεία του παιχνιδιού καθώς όχι μόνο βοηθάει στην εύρεση Evidence του φαντάσματος, αλλά βοηθάει στην εύρεση του δωματίου που το φάντασμα έχει στοιχειώσει και χρησιμοποιεί σαν 'φωλιά' του καθ' όλη την διάρκειας της έρευνας.

Με λίγα λόγια το θερμόμετρο , μπορεί να χρησιμοποιηθεί είτε ο παίκτης το κρατάει στα χέρια του, είτε το έχει πετάξει στο πάτωμα. Όσο το θερμόμετρο αλλάζει δωμάτιο θα αναγράφει και την θερμοκρασία του δωματίου, στην περίπτωση εύρεσης του δωματίου του φαντάσματος, η ένδειξη του θερμομέτρου θα αρχίσει να μειώνεται πιο απότομα, και ανάλογα αν το φάντασμα διαθέτει το Evidence: Freezing Temperatures, η θερμοκρασία θα φτάσει κάτω από τους 0 βαθμούς Κελσίου ή να παραμείνει ελάχιστα πάνω από αυτούς.

Οπότε ο παίκτης σε αυτήν την δοκιμασία καλείται να ερευνήσει το κάθε δωμάτιο , και να αναγνωρίσει ποιο από αυτά αποτελεί την 'φωλιά' του φαντάσματος

Thermometer Device

Το θερμόμετρο λειτουργεί με την χρήση δύο script. Το πρώτο script αφορά την λειτουργία του θερμόμετρου για την ενημέρωση της ένδειξης του με όνομα ThermometerDisplay.cs και το δεύτερο script ελέγχει αν το θερμόμετρο είναι μέσα στο δωμάτιο του φαντάσματος.

```
private void Start() {
    // Initialize with a slightly random temperature for realism
    currentTemperature = Random.Range(15.5f, 25.5f);
}

Unity Message | 0 references
void Update() {
    timer += Time.deltaTime;

    if (timer >= updateInterval) {
        if (isInRoom) {
            if (!IsTutorial) {
                if (GhostTypeSelector.Instance != null && GhostTypeSelector.Instance.HasFreezingTemps) {
                    float targetTemperature = -6.5f;
                    currentTemperature = Mathf.MoveTowards(currentTemperature, targetTemperature, temperatureChangeRate);
                } else {
                    float targetTemperature = .2f;
                    currentTemperature = Mathf.MoveTowards(currentTemperature, targetTemperature, temperatureChangeRate);
                }
            } else {
                float targetTemperature = -6.5f;
                currentTemperature = Mathf.MoveTowards(currentTemperature, targetTemperature, temperatureChangeRate+1f);
            }


                // Move toward cold temperature gradually

        } else {
            // Float around 20°C naturally
            float randomOffset = Random.Range(-0.3f, 0.3f);
            currentTemperature = Mathf.Clamp(currentTemperature + randomOffset, 19.5f, 20.5f);
        }

        // Update temperature text
        if (temperatureText != null)
            temperatureText.text = $"{currentTemperature:F1}°C";

        // Update icon sprite
        if (statusIcon != null && coldSprite != null && normalSprite != null) {
            if (currentTemperature < 7.5f)
                statusIcon.sprite = coldSprite;
            else
                statusIcon.sprite = normalSprite;
        }

        timer = 0f;
    }
}
```

ThermometerDisplay.cs

Το παραπάνω script ρυθμίζει την ένδειξη του θερμομέτρου ελέγχοντας την Boolean τιμή isInRoom. Σε περίπτωση που είναι αληθής η συνθήκη τότε ελέγχουμε αν ο παίκτης παίζει το Tutorial, με σκοπό να αναδειχθεί η θερμοκρασία ταχύτερα από ότι θα συμβαίνει στο Singleplayer.
Στην συνέχεια πριν θέτουμε το κατώτατο όριο θερμοκρασίας , το οποίο εξαρτάται από το αν το φάντασμα διαθέτει σαν Evidence το Freezing Temperatures.

Τέλος μέχρι να βρεθεί το δωμάτιο του φαντάσματος οι θερμοκρασίες θα κυμαίνονται κοντά στους 20 βαθμούς Κελσίου.
Αν η θερμοκρασία είναι μικρότερη από το 7.5 τότε αλλάζουμε την ένδειξη του θερμομέτρου σε ένα διαφορετικό εικονίδιο.

Ο τρόπος με τον οποίο λαμβάνουμε την τιμή isInRoom είναι μέσω του δεύτερου script που αναφέρθηκε με το όνομα ThermometerRoomTrigger.cs

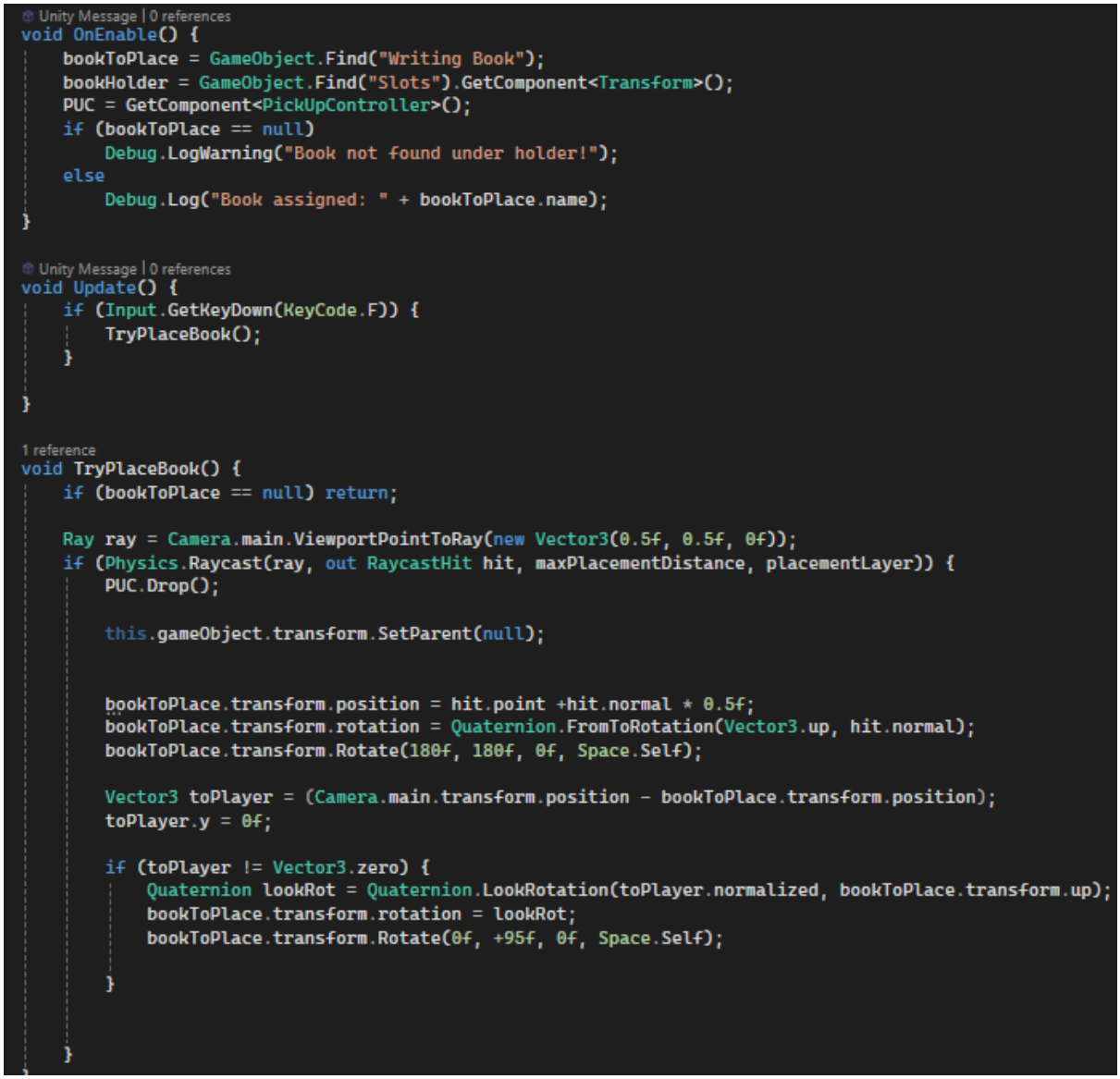
```
public class ThermometerRoomTrigger : MonoBehaviour {
    public ThermometerDisplay thermometerDisplay;

    Unity Message | 0 references
    void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Ghost Room") ) {
            thermometerDisplay.SetInRoom(true);
        }
    }

    Unity Message | 0 references
    void OnTriggerExit(Collider other) {
        if (other.CompareTag("Ghost Room") ) {
            thermometerDisplay.SetInRoom(false);
        }
    }
}
```

Το συγκεκριμένο script το μόνο που είναι υπεύθυνο να πραγματοποιήσει είναι ο έλεγχος για το αν το θερμόμετρο βρίσκεται στον χώρο του φαντάσματος τον οποίο σηματοδοτούμε με την χρήση του Tag = Ghost Room. Πιο συγκεκριμένα χρησιμοποιούμε ένα SphereCollider το οποίο θέτουμε στο Component Rigidbody το isTrigger = True. Αυτό μας δίνει την δυνατότητα να ελέγχουμε αν κάνει Collide με ένα άλλο Collider που έχει το επιθυμητό tag. Στην περίπτωση που κάνει τότε επιστρέφουμε στο προηγούμενο script την τιμή true , διαφορετικά επιστρέφουμε false.

4.3.6 Tutorial Sixth Room

Στο επόμενο δωμάτιο, ο παίκτης θα έχει την δυνατότητα να μάθει για την χρήση του βιβλίου (Writing Book), το οποίο αποτελεί το επόμενο διαθέσιμο Evidence που μπορεί το φάντασμα να αποκαλύψει.

4.3.6.1 BookPlacer.cs



```
Unity Message | 0 references
void OnEnable() {
    bookToPlace = GameObject.Find("Writing Book");
    bookHolder = GameObject.Find("Slots").GetComponent<Transform>();
    PUC = GetComponent<PickUpController>();
    if (bookToPlace == null)
        Debug.LogWarning("Book not found under holder!");
    else
        Debug.Log("Book assigned: " + bookToPlace.name);
}

Unity Message | 0 references
void Update() {
    if (Input.GetKeyDown(KeyCode.F)) {
        TryPlaceBook();
    }

}

1 reference
void TryPlaceBook() {
    if (bookToPlace == null) return;

    Ray ray = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0f));
    if (Physics.Raycast(ray, out RaycastHit hit, maxPlacementDistance, placementLayer)) {
        PUC.Drop();

        this.gameObject.transform.SetParent(null);


        bookToPlace.transform.position = hit.point +hit.normal * 0.5f;
        bookToPlace.transform.rotation = Quaternion.FromToRotation(Vector3.up, hit.normal);
        bookToPlace.transform.Rotate(180f, 180f, 0f, Space.Self);

        Vector3 toPlayer = (Camera.main.transform.position - bookToPlace.transform.position);
        toPlayer.y = 0f;

        if (toPlayer != Vector3.zero) {
            Quaternion lookRot = Quaternion.LookRotation(toPlayer.normalized, bookToPlace.transform.up);
            bookToPlace.transform.rotation = lookRot;
            bookToPlace.transform.Rotate(0f, +95f, 0f, Space.Self);

        }


    }
```

BookPlacer.cs

Το παραπάνω script αφορά τον τρόπο με τον οποίο ο παίκτης μπορεί να ‘αφήσει’ το βιβλίο στο πάτωμα με σκοπό το φάντασμα να έχει αργότερα την δυνατότητα να γράψει πάνω σε αυτό

Κατά την αρχή του script γίνονται κάποιοι έλεγχοι για ορισμένες μεταβλητές με σκοπό να προσομοιωθεί η ενέργεια του παίκτη να αφήσει το βιβλίο. Εφόσον ο παίκτης προσπαθήσει να αφήσει το βιβλίο με το πάτημα του πλήκτρου ‘F’, χρησιμοποιούμε ένα Raycast από την κάμερα του παίκτη ,προσπαθώντας να συγκρουστεί με ένα αντικείμενο που έχει Layer : Floor το οποίο το θέτουμε στο Inspector. Εφόσον συγκρουστεί με ένα τέτοιο αντικείμενο ορίζουμε το position και Rotation το βιβλίου με σκοπό να προσαρμοστεί σωστά στο έδαφος, και καλούμε την συνάρτηση Drop από το PickUpController που θα ενημερώσει το Inventory του παίκτη κατάλληλα για την αφαίρεση του Writing Book από το Inventory.

4.3.7 Tutorial Seventh Room

Tutorial Seventh Room

Στο τελευταίο δωμάτιο ο παίκτης μαθαίνει για την χρήση του Sanity Medication.Η χρήση του Sanity Medication είναι η αναπλήρωση του χαμένου Sanity που έχει υποστεί ο παίκτης από την συνεχόμενη αλληλεπίδραση του παίκτη με το φάντασμα ή και από την παραμονή του παίκτη στο περιβάλλον.

```
public class SanityMed : MonoBehaviour
{
    public SanityManager sanity;
    public bool used;
    public AudioSource asource;
    public AudioClip med_taken;
    Unity Message | 0 references
    private void Start() {
        used = false;

    }

    Unity Message | 0 references
    private void Update() {
        if (!used && Input.GetKeyDown(KeyCode.F) && (sanity.sanity + 30f) <= 100f ) {

            sanity.RestoreSanity(30f);
            asource.PlayOneShot(med_taken);
            used = true;

        } else {
            Debug.Log("Already used Sanity Med!");
        }

    }


    1 reference
    public void SetPlayer(SanityManager playerSanity) {
        sanity = playerSanity;

    }

}
```

SanityMed.cs

Το παραπάνω script αφορά το ίδιο το Sanity Medication και όχι την μπάρα Sanity του παίκτη. Για αρχή χρησιμοποιούμε σαν αναφορά μια μεταβλητή με όνομα sanity που αφορά το script SanityManager.cs που είναι ενσωματωμένο πάνω στον παίκτη, για να ρυθμίσουμε την τιμή του Sanity του παίκτη και να ενημερώσουμε την μπάρα.

Στην συνέχεια ελέγχουμε αν το Sanity Medication που διαθέτει ο παίκτης έχει χρησιμοποιηθεί , αν ο παίκτης έχει αρκετή έλλειψη sanity , και αν πάτησε το σωστό πλήκτρο. Σε περίπτωση που όλα τα παραπάνω κριτήρια τότε αναπληρώνεται 30.0 βαθμοί Sanity στον παίκτη και στην συνέχεια θέτουμε το Medication ως used = true , για να μην μπορεί να γίνει ξανά χρήση.

4.4 Lobby Menu

Με την ολοκλήρωση του Tutorial ο παίκτης θα μεταφερθεί απευθείας στην επόμενη σκηνή που είναι το Lobby. Στην σκηνή αυτή ο παίκτης θα έχει την δυνατότητα επιλογής της περιοχής που θα προσπαθήσει να ερευνήσει, καθώς και την δυνατότητα να μαζέψει πόντους και λεφτά μέσα από αποστολές που πρέπει να καταφέρει να βγάλει εις πέρας σε κάθε έρευνα.

Lobby Menu

Από το Entry Scene , έχουν περάσει και σε αυτή την σκηνή ορισμένα GameObjects τα οποία έχουν τεθεί σε DontDestroyOnLoad καθώς και είναι Static. Τα αντικείμενα με αυτές τις ιδιότητες έχουν την ευκολία να μπορούν να ελεγχθούν από οποιαδήποτε script σε οποιαδήποτε σκηνή και έτσι να μπορούμε να διαβιβάζουμε τιμές από μια σκήνη σε μία άλλη (π.χ. οι ρυθμίσεις για τον ήχο).

Πιο συγκεκριμένα θα μιλήσουμε για δύο GameObjects:

1) AudioManager
2) LevelManager

4.4.1 AudioManager

```
public class AudioManager : MonoBehaviour {
    public static AudioManager Instance;
    public AudioMixer audioMixer;


    private const string MasterKey = "Master";
    private const string MusicKey = "Music";
    private const string SFXKey = "SFX";

    1 reference
    public float GetMasterVolume() => masterVolume;
    1 reference
    public float GetMusicVolume() => musicVolume;
    1 reference
    public float GetSFXVolume() => sfxVolume;


    private float masterVolume = 1f;
    private float musicVolume = 1f;
    private float sfxVolume = 1f;

    Unity Message | 0 references
    private void Awake() {

        if (Instance == null) {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        } else {
            Destroy(gameObject);
            return;
        }

        // Loads values from PlayerPrefs
        masterVolume = PlayerPrefs.GetFloat(MasterKey, 1f);
        musicVolume = PlayerPrefs.GetFloat(MusicKey, 1f);
        sfxVolume = PlayerPrefs.GetFloat(SFXKey, 1f);


        ApplyVolumes();
    }
```

```
1 reference
public void SetMasterVolume(float value) {
    masterVolume = Mathf.Clamp(value, 0.0001f, 1f);
    PlayerPrefs.SetFloat(MasterKey, masterVolume);
    PlayerPrefs.Save();
    ApplyVolumes();
}


1 reference
public void SetMusicVolume(float value) {
    musicVolume = Mathf.Clamp(value, 0.0001f, 1f);
    PlayerPrefs.SetFloat(MusicKey, musicVolume);
    PlayerPrefs.Save();
    ApplyVolumes();
}


1 reference
public void SetSFXVolume(float value) {
    sfxVolume = Mathf.Clamp(value, 0.0001f, 1f);
    PlayerPrefs.SetFloat(SFXKey, sfxVolume);
    PlayerPrefs.Save();
    ApplyVolumes();
}


4 references
private void ApplyVolumes() {

    float masterDB = Mathf.Log10(masterVolume) * 20f;
    float musicDB = Mathf.Log10(masterVolume * musicVolume) * 20f;
    float sfxDB = Mathf.Log10(masterVolume * sfxVolume) * 20f;

    audioMixer.SetFloat("Master", masterDB);
    audioMixer.SetFloat("Music", musicDB);
    audioMixer.SetFloat("SFX", sfxDB);
}
```

Ο παραπάνω κώδικας αφορά τις ρυθμίσεις του ήχου που ελέγχονται μέσα από το Pause Menu → Settings → Audio Settings.

Για αρχή αρχικοποιούμε ορισμένες μεταβλητές με την τρέχουσα τιμή που θα έχουν στην αρχή του παιχνιδιού. Στην συνέχεια μέσα στην Awake (η οποία τρέχει πριν από την Start, και είναι χρήσιμη για πιο σημαντικές αρχικοποιήσεις), δημιουργούμε και θέτουμε ένα Instance.

Το Instance λειτουργεί για να μπορούμε να καλέσουμε με ευκολία το script και τις διαθέσιμες μεταβλητές του χωρίς να υπάρχει θέμα για αδυναμία πρόσβασης. Επίσης επειδή θέτουμε και το αντικείμενο που βρίσκεται το script απάνω ως DontDestroyOnLoad(), το αντικείμενο αυτό θα λειτουργεί σε όλες τις σκηνές. Τέτοια αντικείμενα μπορούμε να τα ονομάσουμε και ως Singleton. Τέλος θέτουμε ότι αν σε μια σκηνή προϋπάρχει ήδη το αντικείμενο αυτό, πρώτα θα διαγράψουμε το παλιό.

Στην συνέχεια λαμβάνουμε από το PlayerPrefs τις τιμές της έντασης του κάθε ήχου για την κάθε κατηγορία που διαθέτουμε, και τις αποθηκεύουμε. Τέλος με την χρήση της AplyVolumes() οδηγούμαστε σε μια νέα συνάρτηση όπου με βάση τις τιμές που λήφθηκαν από το PlayerPrefs αλλάζουμε και την ένταση των ήχων.

Στην περίπτωση που ο παίκτης αποφασίσει να αλλάξει τις ρυθμίσεις ενεργοποιούνται ανάλογα με την κατηγορία οι συναρτήσεις SetMasterVolume, SetMusicVolume, SetSFXVolume.
Μέσα σε αυτές εκτός από την αλλαγή της έντασης, αποθηκεύουμε στο PlayerPrefs την νέα τιμή που επιλέχθηκε.

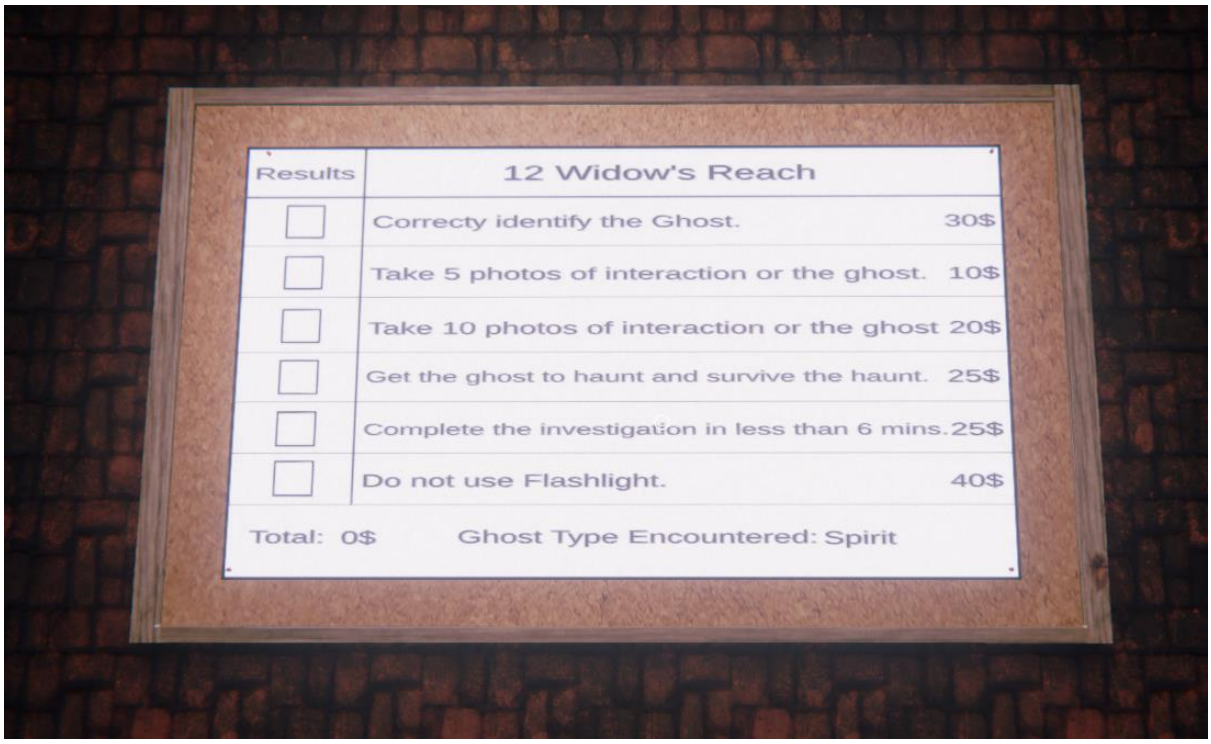
Ο τρόπος με τον οποίο καλούνται οι συναρτήσεις αυτές είναι με το script VolumeSlider το οποίο έχει αναλυθεί προηγουμένως.

4.4.2 LevelManager

Ο LevelManager είναι υπεύθυνος για την λειτουργία της ανταμοιβής μετά από κάθε έρευνα που κάνει ο παίκτης. Πιο συγκεκριμένα ο παίκτης ολοκληρώνοντας έρευνες στις διάφορες στοιχειωμένες περιοχές έχει την δυνατότητα να ολοκληρώσει επιπλέον αποστολές οι οποίες του παρέχουν χρήματα καθώς και πόντους εμπειρίας που βοηθάνε στην ανάπτυξη των στατιστικών του προφίλ του.

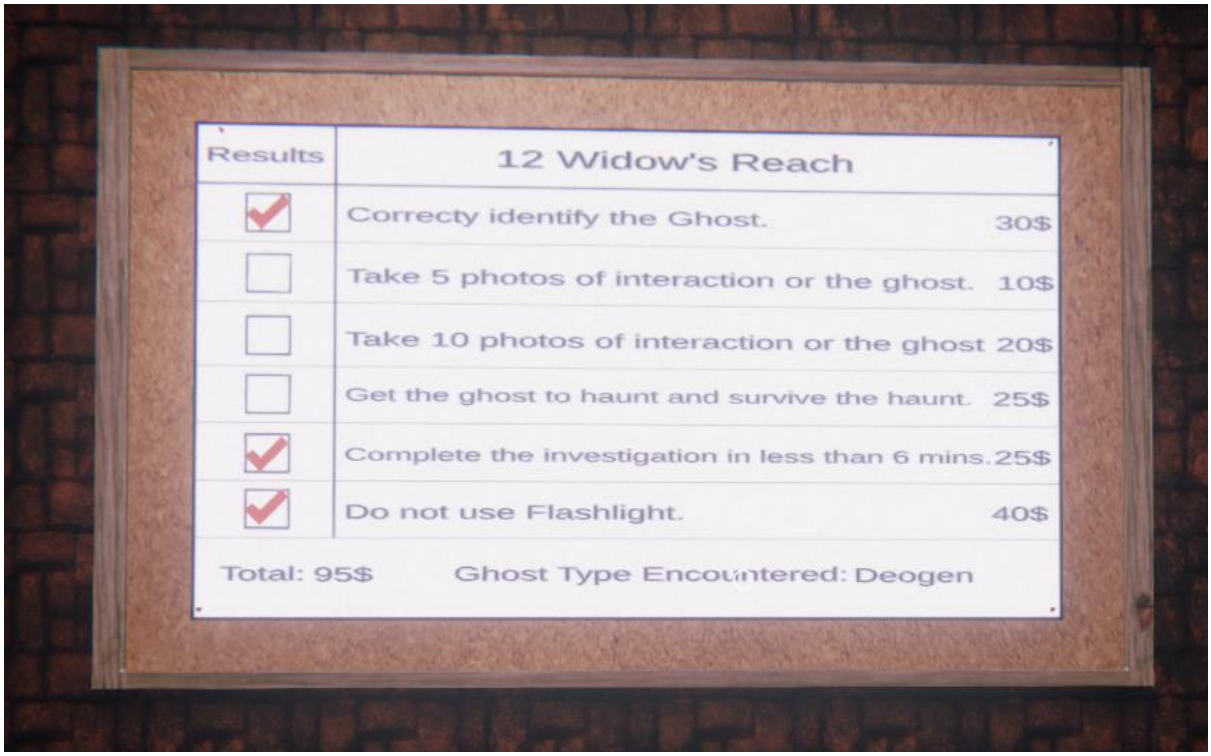
Οι αποστολές που μπορεί ο παίκτης να ολοκληρώσει είναι:

1) Να ταυτοποιήσει σωστά το φάντασμα – Ανταμοιβή: 30$
2) Να έχει 5 έγκυρες φωτογραφίες είτε από το φάντασμα, είτε από διάφορες αλληλεπιδράσεις του φαντάσματος με το περιβάλλον – Ανταμοιβή: 10$
3) Να έχει 10 έγκυρες φωτογραφίες είτε από το φάντασμα, είτε από διάφορες αλληλεπιδράσεις του φαντάσματος με το περιβάλλον – Ανταμοιβή: 20$
4) Να επιβιώσει από το φάντασμα εφόσον κυνηγήσει τον παίκτη – Ανταμοιβή: 25$
5) Να ολοκληρώσει την έρευνα σε λιγότερο από 6 λεπτά – Ανταμοιβή: 25$
6) Να μην χρησιμοποιήσει Φακό κατά την διάρκεια της έρευνας – Ανταμοιβή: 40$



Side Missions List

Εφόσον ο παίκτης επιτύχει στην εκπλήρωση οποιουδήποτε από τους παραπάνω στόχους θα σημειωθεί η εκπλήρωση της συγκεκριμένης αποστολής καθώς και τα ανάλογα χρήματα που αντιστοιχούν από την εκπλήρωση τους, όπως φαίνεται παρακάτω :



Side Missions Completed

Ο τρόπος με τον οποίο λειτουργεί ο παραπάνω πίνακας είναι με την βοήθεια του script με όνομα ResultsUI.cs

4.4.2.1 ResultsIU & ResultsManager

```
public class ResultsUI : MonoBehaviour {
    [Header("UI Image to show when correct")]
    public GameObject stat1Image;
    public GameObject stat2Image;
    public GameObject stat3Image;
    public GameObject stat4Image;
    public GameObject stat5Image;
    public GameObject stat6Image;
    public TMP_Text Total_Text;
    public TMP_Text Ghost_Name;
    private int InvestigationsDone;
    private int DeathsCtr;
    private int CorrectGuess;
    private int money;

    Unity Message | 0 references
    void Start() {
        if (ResultsManager.Instance == null) {
            Debug.LogError("No ResultsManager instance found!");
            return;
        }

        var r = ResultsManager.Instance;
        InvestigationsDone = 1;
        if (r.isCorrect) { money += 30; stat1Image.SetActive(true); CorrectGuess = 1; }
        if (r.PhotosTaken >= 5) { money += 10; stat2Image.SetActive(true); }
        if (r.PhotosTaken == 10) { money += 20; stat3Image.SetActive(true); }
        if (!r.DeadHaunt) { money += 25; stat4Image.SetActive(true); } else { DeathsCtr = 1; }
        if (r.TimeSpentInGame < 6f) { money += 25; stat5Image.SetActive(true); }
        if (r.FlashlightOFF) { money += 40; stat6Image.SetActive(true); }

        Total_Text.text = money + "$";
        Ghost_Name.text = r.actualGhostType.ToString();

        // Add money and update panel
        if (PlayerLevelnMoney.Instance != null) {
            PlayerLevelnMoney.Instance.AddMoney(money);
            PlayerLevelnMoney.Instance.SetOtherStatValues(DeathsCtr, CorrectGuess, InvestigationsDone);
            // Update stats panel if it exists
            FindObjectOfType<PlayerStatsUI>()?.UpdateStatsUI();
        } else {
            Debug.LogError("PlayerLevelnMoney instance not found!");
        }
    }
```

ResultsUI.cs

Για αρχή θέτουμε ορισμένα GameObjects ώστε να μπορέσουμε να θέσουμε στο Inspector τα αντικείμενα για τα οποία μιλάμε, καθώς και τα κείμενα και μεταβλητές που αφορούν στατιστικά για το προφίλ του παίκτη και των χρημάτων.

Πιο συγκεκριμένα κάνουμε μια αναφορά στο ResultsManager το οποίο είναι αντικείμενο Singleton και ο σκοπός του είναι να λάβει τις τιμές που θα προέρχονται από την έρευνα του παίκτη και θα τις στέλνουμε στο ResultsUI μέσω της αναφοράς του με την εντολή var r = ResultsManager.Instance . Με αυτό τον τρόπο μπορούμε να έχουμε πρόσβαση πάνω σε κάθε τιμή που βρίσκεται στο ResultsManager.cs και έτσι κάνουμε τους αντίστοιχους ελέγχους με την χρήση των if.

Στην συνέχεια γίνονται οι παρακάτω έλεγχοι:

1) Ελέγχουμε αν η επιλογή του φαντάσματος αντιστοιχεί στον τύπο του φαντάσματος που υπήρχε στην έρευνα. Αν ναι τότε ανταμείβουμε τον παίκτη με χρήματα, και ενημερώνουμε

την εκπλήρωση της αποστολής καθώς και την μεταβλητή CorrectGuess = 1, ώστε να προστεθεί αργότερα στα στατιστικά του παίκτη.

2) Ελέγχουμε αν ο παίκτης έβγαλε 5 έγκυρες φωτογραφίες, αν ναι τον ανταμείβουμε κατάλληλα και ενημερώνουμε και την εκπλήρωση της αποστολής.
3) Ομοίως κάνουμε έλεγχο για 10 φωτογραφίες και εκπληρώνουμε τις ίδιες ακριβώς δράσεις για την ενημέρωση του πίνακα αποστολών
4) Ελέγχουμε αν ο παίκτης επέζησε από το φάντασμα που τον κυνήγησε. Αν ναι τότε επιβραβεύουμε τον παίκτη με τα χρήματα που του αναλογούν, ενημερώνουμε κατάλληλα τον πίνακα. Αν όχι τότε σημειώνουμε ότι ο παίκτης πέθανε μέσω της μεταβλητής DeathCtr.
5) Ελέγχουμε αν ο χρόνος που χρειάστηκε ο παίκτης για την έρευνα του κράτησε λιγότερο από 6 λεπτά. Στην περίπτωση που ισχύει η συνθήκη , τον ανταμείβουμε με τα ανάλογα λεφτά και ενημερώνουμε τον πίνακα κατάλληλα.
6) Ελέγχουμε αν ο φακός έχει χρησιμοποιηθεί μέσω της Boolean τιμής FlashlightOFF. Αν ισχύει, τότε ανταμείβουμε με λεφτά τον παίκτη και ενημερώνουμε τον πίνακα.

Τέλος , εμφανίζουμε στον πίνακα τα συνολικά λεφτά που λήφθηκαν από την πρόσφατη έρευνα, καθώς και τον τύπο του φαντάσματος που καλούμασταν να αντιμετωπίσουμε. Μετά από την διαδικασία αυτή ενημερώνουμε το Singleton GameObject : LevelManager, που περιέχει το script PlayerLevelnMoney και είναι υπεύθυνο για την αποθήκευση των τιμών και στατιστικών στο προφίλ του παίκτη και τέλος καλούμε την συνάρτηση UpdateStatsUI() που θα ενημερώσει κατάλληλα τα στατιστικά του παίκτη σε ένα ειδικό UI που θα επεκταθούμε αργότερα.

```
public class ResultsManager : MonoBehaviour {
    public static ResultsManager Instance;

    public GhostTypeSelector.GhostType actualGhostType;
    public GhostTypeSelector.GhostType playerGuess;
    public bool isCorrect;
    public float TimeSpentInGame=100;
    public int PhotosTaken;
    public bool DeadHaunt = true;
    public bool FlashlightOFF;

    Unity Message | 0 references
    void Awake() {
        if (Instance == null) {
            Instance = this;
            DontDestroyOnLoad(gameObject); // Keep across scenes
        } else {
            Destroy(gameObject);
        }
    }


    1 reference
    public void SetResults(GhostTypeSelector.GhostType guess, GhostTypeSelector.GhostType actual) {
        playerGuess = guess;
        actualGhostType = actual;
        isCorrect = (playerGuess == actual);
    }

    1 reference
    public void SetAdditionalStats(float time, int photos,bool died, bool NoFlashlight) {
        TimeSpentInGame = time;
        PhotosTaken = photos;
        DeadHaunt = died;
        FlashlightOFF = NoFlashlight;
    }


}
```

ResultsManager.cs

Όλες οι τιμές του ResultsManager θέτονται κατά την αποχώρηση από την κάθε έρευνα με βάση ένα script που βρίσκεται στην σκηνή όπου διεξάγεται η έρευνα.

Στην Awake θέτουμε το αντικείμενο που βρίσκεται το script πάνω ως Singleton. Στην συνέχεια ακολουθούν συναρτήσεις όπου:

Η SetResults() είναι υπεύθυνη για την συμφωνία της επιλογής του παίκτη για τον τύπο φαντάσματος και τον πραγματικό τύπο φαντάσματος. Σε περίπτωση συμφωνίας των τιμών αυτών ενημερώνουμε την τιμή isCorrect νε σκοπό να περαστεί στο ResultsUI

Η SetAdditionalStats() δέχεται όλες τις υπόλοιπες μεταβλητές που αφορούν τα SideMissions που αναφέρθηκαν παραπάνω με σκοπό να περαστούν στο ResultsUI.

4.4.2.2 PlayerLevelnMoney

```
public class PlayerLevelnMoney : MonoBehaviour {
    public static PlayerLevelnMoney Instance;

    [Header("Player Stats")]
    public int totalMoney = 0;
    public int playerLevel = 1;
    public int currentXP = 0;
    public int xpToNextLevel = 40;
    public int TotalDeaths = 0;
    public int TotalInvestigations = 0;
    public int CorrectInvestigations = 0;




    [Header("Level Settings")]
    public float xpIncreaseRate = 1.2f;
    public int maxLevel = 99;

    Unity Message | 0 references
    private void Awake() {
        if (Instance == null) {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            LoadPlayerData(); // load saved data
        } else {
            Destroy(gameObject);
        }
    }

    Unity Message | 0 references
    private void OnApplicationQuit() {
        SavePlayerData();
    }
```

LevelnMoney.cs Part1

Για αρχή θέτουμε μεταβλητές που θα χρησιμοποιήσουμε στο script μας αλλά και σε άλλα script, όπως το συνολικό αριθμό χρημάτων, συνολικοί πόντοι εμπειρίας κ.λπ.

Στην συνέχεια ορίζουμε το αντικείμενο ως Singleton και φορτώνουμε τις τιμές μέσω της συνάρτησης LoadPlayerData() που φαίνεται παρακάτω:

```
public void LoadPlayerData() {
    totalMoney = PlayerPrefs.GetInt("TotalMoney", 0);
    playerLevel = PlayerPrefs.GetInt("PlayerLevel", 1);
    currentXP = PlayerPrefs.GetInt("CurrentXP", 0);
    xpToNextLevel = PlayerPrefs.GetInt("XPToNextLevel", 40);
    TotalDeaths = PlayerPrefs.GetInt("TotalDeaths", 0);
    TotalInvestigations = PlayerPrefs.GetInt("TotalInvestigations", 0);
    CorrectInvestigations = PlayerPrefs.GetInt("CorrectInvestigations", 0);
}
```

LoadPlayerData Function

Ο τρόπος με τον οποίο λειτουργεί η συνάρτηση είναι ότι λαμβάνει τις τιμές από το πεδίο με το αντίστοιχο όνομα που καλούμε, εφόσον βρεθεί το πεδίο αυτό φορτώνουμε την αποθηκευμένη τιμή **διαφορετικά** φορτώνουμε την τιμή δίπλα από το πεδίο.

Δηλαδή για την πρώτη εντολή:

totalMoney = PlayerPrefs.GetInt(“TotalMoney”,0);

Ζητάμε να πάρουμε την τιμή που είναι αποθηκευμένη στο πεδίο “TotalMoney” , αν το πεδίο αυτό δεν υπάρχει τότε θέτουμε απευθείας την τιμή 0. Αντίστοιχα και για τις υπόλοιπες μεταβλητές.

Όμοια λειτουργεί και SavePlayerData που φαίνεται παρακάτω:

```
public void SavePlayerData() {
    PlayerPrefs.SetInt("TotalMoney", totalMoney);
    PlayerPrefs.SetInt("PlayerLevel", playerLevel);
    PlayerPrefs.SetInt("CurrentXP", currentXP);
    PlayerPrefs.SetInt("XPToNextLevel", xpToNextLevel);
    PlayerPrefs.SetInt("TotalDeaths", TotalDeaths);
    PlayerPrefs.SetInt("TotalInvestigations", TotalInvestigations);
    PlayerPrefs.SetInt("CorrectInvestigations", CorrectInvestigations);
    PlayerPrefs.Save();
}
```

SavePlayerData Function

Η οποία χρησιμοποιεί διαφορετικό τρόπο για να αποκτήσει πρόσβαση στα πεδία αυτά.

Πιο συγκεκριμένα με το SetInt(“TotalMoney”, totalMoney), δεν αποκτάμε πρόσβαση στο πεδίο αυτό την πρώτη φορά, αλλά αντιθέτως το δημιουργούμε. Εφόσον αυτό το πεδίο έχει δημιουργηθεί αποθηκεύουμε σε αυτό το πεδίο την μεταβλητή που επιθυμούμε.

```
public void AddMoney(int amount) {
    totalMoney += amount;
    AddXP(amount); // gain XP equal to money
    SavePlayerData();
}

1 reference
public void AddXP(int amount) {
    if (playerLevel >= maxLevel) return;

    currentXP += amount;

    while (currentXP >= xpToNextLevel && playerLevel < maxLevel) {
        currentXP -= xpToNextLevel; // overflow XP
        LevelUp();
    }

    SavePlayerData();
}

1 reference
private void LevelUp() {
    playerLevel++;
    xpToNextLevel = Mathf.RoundToInt(xpToNextLevel * xpIncreaseRate);


    if (TryGetComponent<AudioSource>(out AudioSource audio)) {
        audio.Play();
    }

    if (playerLevel >= maxLevel) {
        currentXP = 0;
        xpToNextLevel = 0;
    }
}

1 reference
public void SetOtherStatValues(int Deaths, int ValidGuesses, int Investigations) {
    TotalDeaths += Deaths;
    TotalInvestigations += Investigations;
    CorrectInvestigations += ValidGuesses;
```

LevelnMoney Functions

Στην συνέχεια ακολουθούν συναρτήσεις που εκτελούν την αύξηση χρημάτων, την απόκτηση πόντων εμπειρίας και την Αύξηση Επιπέδου του παίκτη.
Η χρήση των χρημάτων στο παιχνίδι συνδέεται άμεσα με τους πόντους εμπειρίας , οπότε όσα χρήματα λάβει ο παίκτης, τόσους πόντους εμπειρίας θα λάβει με σκοπό να ανέβει κατάταξη.

Συγκεκριμένα:

Η AddMoney() δέχεται σαν παράμετρο το ποσό των χρημάτων που πάρθηκαν. Ελέγχουμε αν το παίκτης έχει φτάσει το ανώτατο όριο της κατάταξης. Εφόσον δεν έχει φτάσει αυξάνουμε τους πόντους εμπειρίας του και ελέγχουμε την περίπτωση να ανέβει κατάταξη. Στην περίπτωση που ισχύει, προσέχουμε να καλύψουμε την διαφορά που απέμεινε , και να την εντάξουμε την διαφορά στην πρόοδο για το επόμενο επίπεδο.
Εφόσον ο παίκτης ανέβηκε επίπεδο θα πρέπει να ενημερώσουμε και τις τιμές που αφορούν την πρόοδο του παίκτη, καλώντας το LevelUp().

Η LevelUp() για αρχή αυξάνει το επίπεδο του παίκτη κατά 1 και στην συνέχεια αυξάνουμε τους απαιτούμενους πόντους εμπειρίας για την προσπέραση του επόμενου επιπέδου κατά 20% του

προηγούμενου. Επειδή μια τέτοια πράξη θα έχει αποτέλεσμα δεκαδικό αριθμό την μετατρέπουμε σε Ακέραιο.

Τέλος, ακούγεται ένας ήχος που αναδεικνύει την απόκτηση νέου επιπέδου και ελέγχουμε ότι αν πρόκειται για το μέγιστο επίπεδο, να μην θέσουμε άλλο ανώτατο όριο για αύξηση επιπέδου.

Η συνάρτηση SetOtherValues() αφορά την προσμέτρηση των στατιστικών που προκύπτουν από τα SideMissions και αναφερθήκαμε προηγουμένως στο ResultsUI.

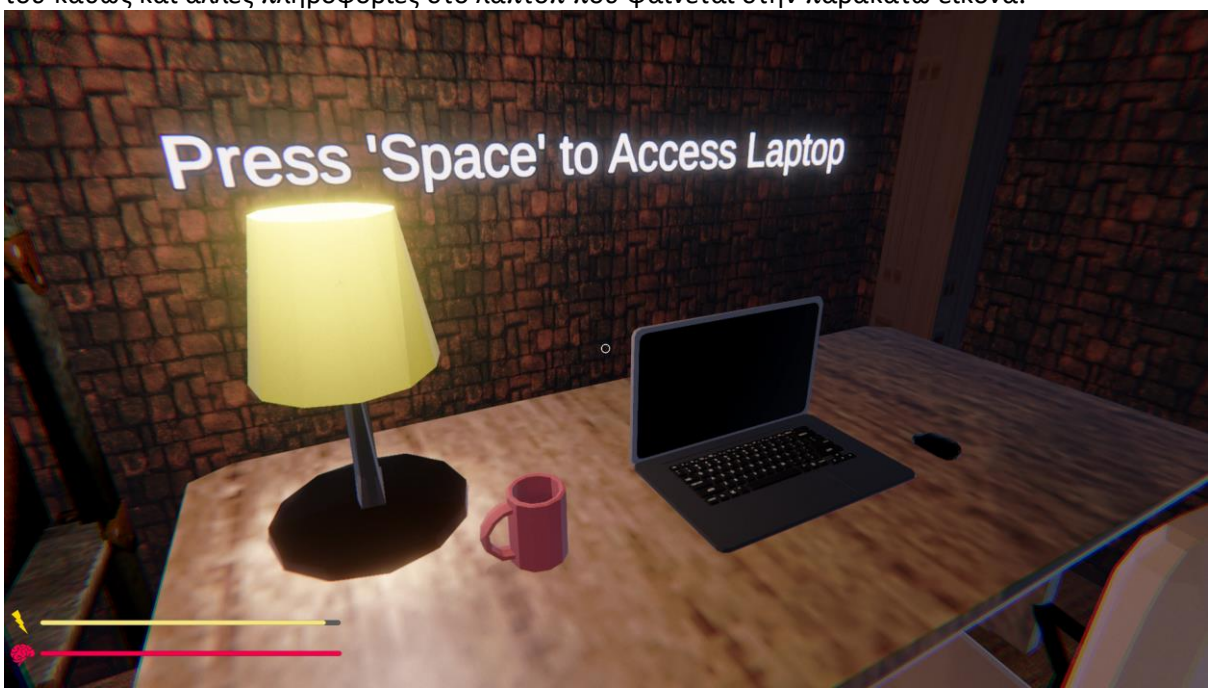
Με την σκέψη ότι μπορεί κάποια στιγμή ο παίκτης να θέλει να επαναφέρει για κάποιο λόγο τα στατιστικά του, υπάρχει και η παρακάτω συνάρτηση ResetPlayerData(), η οποία κάνει ακριβώς αυτήν την ενέργεια μηδενίζοντας όλες τις τιμές σαν να ξεκινάει πρώτη φορά ο παίκτης.

```
1 reference
public void ResetPlayerData() {
    PlayerPrefs.DeleteAll();
    totalMoney = 0;
    playerLevel = 1;
    currentXP = 0;
    xpToNextLevel = 40;
    TotalDeaths = 0;
    TotalInvestigations = 0;
    CorrectInvestigations = 0;
    SavePlayerData();
}
```

ResetPlayerData Function

4.4.3 LaptopUI & Player Stats

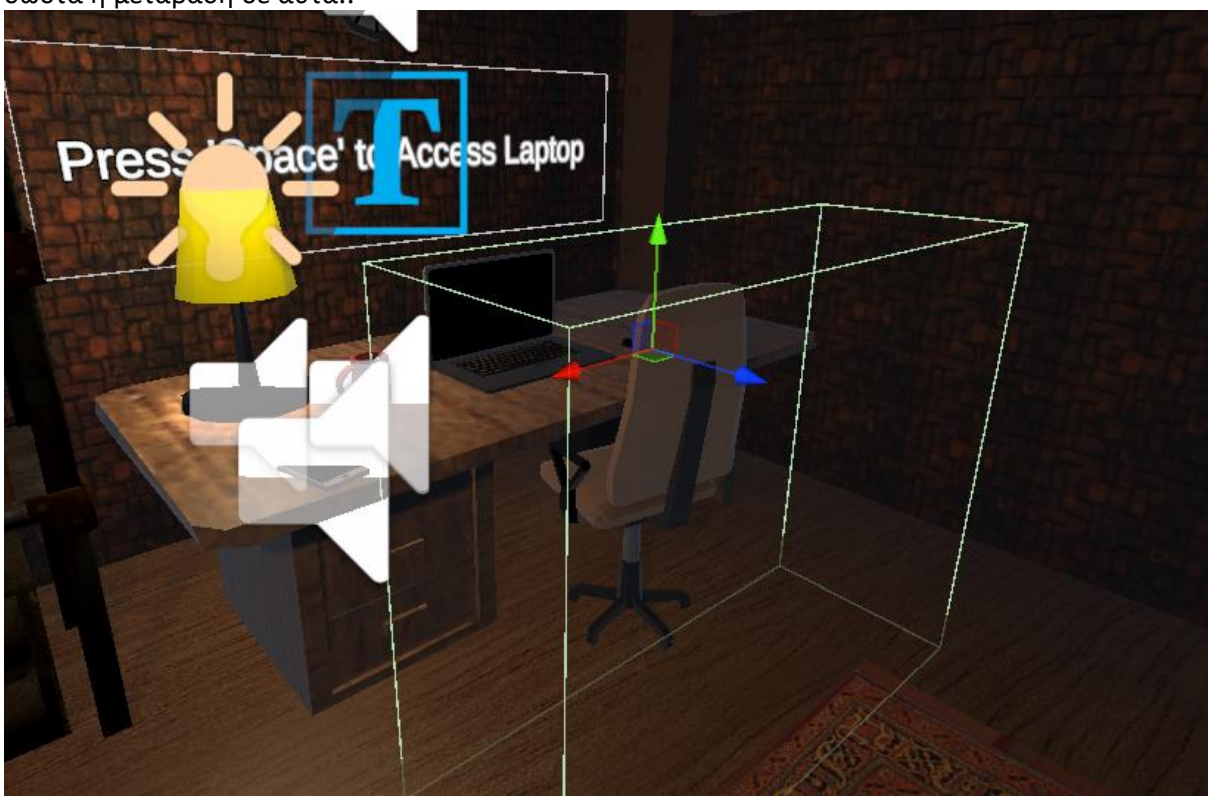
Εφόσον ο παίκτης έχει επιστρέψει από την έρευνα του, μπορεί να παρακολουθήσει τα στατιστικά του καθώς και άλλες πληροφορίες στο λάπτοπ που φαίνεται στην παρακάτω εικόνα:



Laptop UI

Εφόσον ο παίκτης βρεθεί κοντά στο λάπτοπ θα εμφανιστεί η ένδειξη να πατήσει το πλήκτρο ‘Space’ στην περίπτωση που θέλει να μάθει πληροφορίες ή να ελέγξει τα στατιστικά του.

Ο τρόπος με τον οποίο λειτουργεί αυτή η μετάβαση είναι μέσω αντικειμένων με tag CameraSpot που βρίσκονται στο χώρο μπροστά από το λάπτοπ και ελέγχονται με μερικά script για να γίνει σωστά η μετάβαση σε αυτά..



CameraSpot GameObject

Στην πράσινη περιοχή αναδεικνύεται σε ποιο σημείο πρέπει να βρίσκεται ο παίκτης για να εμφανιστεί το κείμενο από πάνω, καθώς και για να μπορέσει ο παίκτης να εισέλθει στο λάπτοπ.

Σε όλα τα σημεία με tag CameraSpot χρησιμοποιούμε δύο script.

4.4.3.1 IsNear Script & CameraSpotInfo Script

```
public class IsNearScript : MonoBehaviour
{
    public bool IsNear = false;
    private GameObject player;
    public GameObject CanvasInstruction;

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        if (other.gameObject.CompareTag("Player")) {
            IsNear = true;
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other) {
        if (other.gameObject.CompareTag("Player")) {
            IsNear = false;
        }
    }

    Unity Message | 0 references
    private void Update() {
        if (IsNear) {

            CanvasInstruction.SetActive(true);
        } else {

            CanvasInstruction.SetActive(false);

        }
    }

}
```

Το συγκεκριμένο script είναι υπέυθυνο να αναγνωρίσει μέσω αν το αντικείμενο που εισήλθε στην πράσινη περιοχή είναι ο παίκτης, ελέγχοντας το tag του αν είναι "Player". Αν είναι ο παίκτης τότε εμφανίζει την πληροφορία για το πώς θα εισέλθει.

```
using UnityEngine;


public class CameraSpotInfo : MonoBehaviour {
    6 references
    public enum UIType { None, MapSelector, Laptop, MapLeaver }
    public UIType uiType = UIType.None;
}
```

Το δεύτερο script βοηθάει στο να θέσουμε ποιος τύπος από όλους τους CameraSpot είναι ο συγκεκριμένους που προσπαθούμε να αλληλεπιδράσουμε. Έτσι μας δίνεται η επιλογή να θέσουμε στο Inspector για το συγκεκριμένο CameraSpot ποιο τύπο UI θέλουμε να αναφέρεται ώστε να το ενεργοποιήσουμε.

4.4.3.2 MovePlayerCamera Script

Το παρακάτω script είναι αυτό που ελέγχει την τιμή του CameraSpotInfo.cs καθώς και πολλαπλές άλλες μεταβλητές για να επιτύχει σωστές και ομαλές μεταβάσεις της κάμερας του παίκτη στο επιθυμητό CameraSpot.

```
public class MovePlayerCamera : MonoBehaviour {
    private Transform targetTransform;
    private Transform camTransform;
    public FPSController fpsController;
    public IsNearScript CheckNear;

    private bool isMoving = false;
    public float moveSpeed = 4.0f;
    private float lerpProgress = 0f;

    private Vector3 startPosition;
    private Quaternion startRotation;
    private Vector3 endPosition;
    private Quaternion endRotation;

    private bool isAtTarget = false;

    public GameObject MapSelectorUI;
    public GameObject MapLeaverUI;
    public GameObject LaptopUI;

    private GameObject[] cameraSpots;
```

MovePlayerCamera.cs Part1

Για αρχή, αρχικοποιούμε ορισμένες μεταβλητές με σκοπό να έχουμε αναφορές σε:

1) Υπάρχων Script όπως IsNearScript (Το FPSController θα εξηγηθεί στην κατηγορία του Παίκτη)
2) Στο Transform Component αντικειμένων (τα οποία θέτονται παρακάτω στον κώδικα)
3) Αντικειμένων – GameObejct που αφορούν τα UI Panels των σημείων που έχουν GameObject με tag CameraSpot
4) Ένα πίνακα από GameObjects στον οποίο θα βάλουμε όλα τα δυνατά GameObjects με tag CameraSpot .

```
void Start() {
    // Find all CameraSpots in the scene
    cameraSpots = GameObject.FindGameObjectsWithTag("CameraSpot");
    if (cameraSpots.Length == 0) {
        Debug.LogWarning("No CameraSpots found in the scene!");
    }

    fpsController = GetComponent<FPSController>();
    if (fpsController == null) {
        Debug.LogWarning("FPSController not found on this GameObject.");
    }

    if (Camera.main != null) {
        camTransform = Camera.main.transform;
    } else {
        Debug.LogError("Main camera not found! Disabling script.");
        enabled = false;
        return;
    }
}
```

MovePlayerCamera.cs Part2

Στην συνέχεια ακολουθεί η μέθοδος Start() η οποία στην αρχή λειτουργίας του script, προσπαθεί να βρει όλα τα διαθέσιμα αντικείμενα με tag CameraSpot. Επιπλέον ανακτά τις τιμές και μεταβλητές του FPSController (αφορά κυρίως την κίνηση του παίκτη), και ανακτούμε το στοιχείο Transform της κάμερα του παίκτη.

```
void Update() {
    if (Input.GetKeyDown(KeyCode.Space)) {
        GameObject nearestSpot = GetNearestCameraSpot();
        if (nearestSpot != null) {
            CheckNear = nearestSpot.GetComponent<IsNearScript>();
            targetTransform = nearestSpot.transform;

            if (CheckNear != null && CheckNear.IsNear) {
                if (MainMenuFucntions.CurrentUI == MainMenuFucntions.UIState.None ||
                    MainMenuFucntions.CurrentUI == MainMenuFucntions.UIState.MapSelector ||
                    MainMenuFucntions.CurrentUI == MainMenuFucntions.UIState.MapLeaver ||
                    MainMenuFucntions.CurrentUI == MainMenuFucntions.UIState.Laptop) {
                    ToggleCameraPosition();
                }
            }
        }
    }

    if (isMoving) {
        lerpProgress += Time.deltaTime * moveSpeed;
        camTransform.position = Vector3.Lerp(startPosition, endPosition, lerpProgress);
        camTransform.rotation = Quaternion.Slerp(startRotation, endRotation, lerpProgress);

        if (lerpProgress >= 1f)
            isMoving = false;
    }
}
```

MovePlayerCamera.cs Part3

Με την χρήση της μεθόδου Update() ελέγχουμε αν πατηθεί το πλήκτρο Space. Εφόσον πατηθεί, με την χρήση της συνάρτησης GetNearestCameraSpot() λαμβάνουμε το πιο κοντινό αντικείμενο τύπου CameraSpot και το ορίζουμε στην μεταβλητή τύπου GameObject ως το τρέχων CameraSpot που βρέθηκε. Ελέγχουμε αν το script IsNear στο τρέχων CameraSpot είναι true και πρώτού ξεκινήσει η μετάβαση της κάμερα στο επιθυμητό σημείο ελέγχουμε μην υπάρχει άλλο UI Panel ανοιχτό μέσω του MainmenuFunctions.cs (θα αναλυθεί στην κατηγορία του Παίκτη).

Εφόσον πραγματοποιηθούν όλοι οι έλεγχοι καλείται η συνάρτηση ToggleCameraPosition() . Ταυτόχρονα ελέγχεται όπως φαίνεται στην εικόνα η μεταβλητή isMoving. Η συγκεκριμένη μεταβλητή αποτρέπει την κίνηση της κάμερας μέχρι να έχει ολοκληρωθεί επιτυχώς η μετάβαση της.

```
private GameObject GetNearestCameraSpot() {
    if (cameraSpots.Length == 0) return null;

    GameObject nearest = null;
    float minDistance = float.MaxValue;

    foreach (GameObject spot in cameraSpots) {
        float distance = Vector3.Distance(transform.position, spot.transform.position);
        if (distance < minDistance) {
            minDistance = distance;
            nearest = spot;
        }
    }

    return nearest;
}
```

MovePlayerCamera.cs Part4

```
public void ToggleCameraPosition() {
    lerpProgress = 0f;
    isMoving = true;

    if (!isAtTarget) {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;

        startPosition = camTransform.position;
        startRotation = camTransform.rotation;

        endPosition = targetTransform.position;
        endRotation = targetTransform.rotation;

        // Get CameraSpotInfo and start coroutine to open UI after delay
        CameraSpotInfo spotInfo = targetTransform.GetComponent<CameraSpotInfo>();
        if (spotInfo != null) {
            StartCoroutine(OpenUIWithDelay(spotInfo, 0.5f)); // 0.5 second delay
        }

        if (fpsController != null)
            fpsController.enabled = false;
    } else {
        // Hide all UIs when returning to player
        MapSelectorUI?.SetActive(false);
        LaptopUI?.SetActive(false);
        MapLeaverUI?.SetActive(false);

        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;

        startPosition = camTransform.position;
        startRotation = camTransform.rotation;

        endPosition = transform.position + new Vector3(0, 1.6f, 0);
        endRotation = transform.rotation;

        if (fpsController != null)
            fpsController.enabled = true;

        MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.None;
    }

    isAtTarget = !isAtTarget;
}
```

MovePlayerCamera.cs Part5

Παραπάνω φαίνεται η συνάρτηση ToggleCameraPosition() η οποία θέτει σαν τιμές στην αρχή της το lerpProgress = 0 και isMoving = true

Το lerpProgress έχει ως σκοπό να θέσουμε μια αρχή για να ρυθμίσουμε την ομαλότητα με την οποία η κάμερα θα κουνηθεί στο σημείο που θέλουμε.

Στην συνέχεια ελέγχουμε αν η κάμερα έχει φτάσει στην επιθυμητή τοποθεσία για την ανάδειξη των UI Panel μέσω του isAtTarget. Θέτουμε ότι πλέον ο κέρσορας θα είναι φανερός και θα μπορούμε να τον ελέγξουμε, θέτουμε το αρχικό σημείο και περιστροφή που ξεκινάει η κάμερα και το σημείο που θέλουμε να καταλήξει. Τέλος ανακτούμε τις τιμές από το CameraSpotInfo και ξεκινάμε ένα Coroutine και μέσω του FPSController κλειδώνουμε την κίνηση του παίκτη.

Το Coroutine είναι ένας μηχανισμός που επιτρέπει την εκτέλεση κώδικα σε πολλαπλά frame χωρίς όμως να παγώνει το παιχνίδι. Χρησιμοποιείται κυρίως για να προκαλούμε παύσεις στο χρόνο του παιχνιδιού , χωρίς όμως να επηρεάζεται ολόκληρη η ροή του.

Σε περίπτωση που η κάμερα βρίσκεται σε κάποιο ανοιχτό Panel (δηλαδή isAtTarget == true), απενεργοποιούμε όλα τα Panel, κλειδώνουμε ξανά την χρήση του κέρσορα, και πραγματοποιούμε με μετάβαση της κάμερας πίσω στον παίκτη αλλάζοντας τις μεταβλητές startPosition, startRotation και endPosition, endRotation, με σκοπό να είναι τελικός προορισμός της κάμερα ο παίκτης ξανά. Τέλος επαναφέρουμε την κίνηση του παίκτη ώστε να μπορέσει να κινηθεί ξανά στο χώρο.

```
private IEnumerator OpenUIWithDelay(CameraSpotInfo spotInfo, float delay) {
    yield return new WaitForSeconds(delay);

    // Disable all first
    MapSelectorUI?.SetActive(false);
    LaptopUI?.SetActive(false);
    MapLeaverUI?.SetActive(false);

    // Activate the correct UI
    switch (spotInfo.uiType) {
        case CameraSpotInfo.UIType.MapSelector:
            MapSelectorUI?.SetActive(true);
            MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.MapSelector;
            break;
        case CameraSpotInfo.UIType.Laptop:
            LaptopUI?.SetActive(true);
            MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.Laptop;
            break;
        case CameraSpotInfo.UIType.MapLeaver:
            MapLeaverUI?.SetActive(true);
            MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.MapLeaver;
            break;
        case CameraSpotInfo.UIType.None:
            MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.None;
            break;
    }
}
```

MoevPlayerCamera.cs Part6

Στην παραπάνω εικόνα παρουσιάζεται πως δηλώνουμε μια συνάρτηση που είναι Coroutine. Σε σχέση με τις συνηθισμένες συναρτήσεις που δεν χρειάζονται κάποια εντολή return , η συναρτήσεις αυτές απαιτούν εντολή return που είναι της μορφής " **yield return . . .**"

Υπάρχουν πολλαπλοί τρόποι συμπλήρωσης της εντολής. Στην συγκεκριμένη περίπτωση χρησιμοποιούμε **"yield return new WaitForSeconds()"**.

Αυτό σημαίνει οποιαδήποτε εντολή βρίσκεται κάτω από αυτήν που αναφέρθηκε, δεν εκτελείται μέχρις ότου να ολοκληρωθεί ο χρόνος που θέσαμε σαν παράμετρο σε αυτήν. Εφόσον ολοκληρωθεί, Επιλέγουμε αν δεν βρισκόμαστε σε κάποιο Panel, την εμφάνιση του επιθυμητού Panel, και την ενημέρωση του στο MainMenuFucnctions.

4.4.3.3 Laptop Apps and UIs

Εφόσον ολοκληρώθηκε η εμφάνιση στο UI του λάπτοπ θα έρθουμε σε επαφή με αυτό το Panel παρακάτω:

Laptop Panel

Στο συγκεκριμένο Panel , ο παίκτης μπορεί να λειτουργήσει σαν μια συνηθισμένη Επιφάνεια Εργασίας , πατώντας κάποια από τις διαθέσιμες εφαρμογές της.

4.4.3.3.1 Ghost Wiki

Η εφαρμογή αυτή είναι αρκετά χρήσιμή αν κάποιος παίκτης θελήσει να ενημερωθεί για τους τύπους των φαντασμάτων, τα Evidences που μπορούν να εμφανίσουν, καθώς και τον τρόπο συμπεριφοράς τους (ο οποίος δεν αποτελεί παρά μόνο κομμάτι περιγραφής και όχι gameplay).

Χρησιμοποιώντας το Scrollbar της αριστερής στήλης ο παίκτης μπορεί να επιλέξει μεταξύ των 10 τύπων φαντασμάτων και να μάθει πληροφορίες για αυτά.

Ο τρόπο που λειτουργεί το UI αυτό είναι με την χρήση του script NameListUI.cs.

```
public class NameListUI : MonoBehaviour {
    [Header("UI References")]
    public GameObject buttonPrefab;
    public Transform contentParent;
    public TMP_Text infoPanelText;

    [Header("Names and Texts")]
    public List<string> names;
    [TextArea]
    public List<string> infoTexts;

    Unity Message | 0 references
    private void Start() {
        GenerateButtons();
    }

    1 reference
    void GenerateButtons() {
        if (names.Count != infoTexts.Count) {
            Debug.LogError("Names and InfoTexts lists must be the same length!");
            return;
        }


        for (int i = 0; i < names.Count; i++) {
            int index = i;
            GameObject newButton = Instantiate(buttonPrefab, contentParent);
            newButton.GetComponentInChildren<TMP_Text>().text = names[i];


            newButton.GetComponent<Button>().onClick.AddListener(() => OnNameClicked(index));
        }
    }

    1 reference
    void OnNameClicked(int index) {
        infoPanelText.text = infoTexts[index];
    }
}
```

NameListUI.cs

Στο παραπάνω script αρχικοποιούμε τρείς μεταβλητές οι οποίες αφορούν:

1) Ένα Prefab κουμπιού το οποίο θα δημιουργηθεί όσες φορές επιλέξουμε στον scrollbar
2) Την τοποθεσία του παραθύρου που είναι το Scrollbar
3) Το κείμενο που εμφανίζεται στο πλαίσιο δίπλα από το Scrollbar

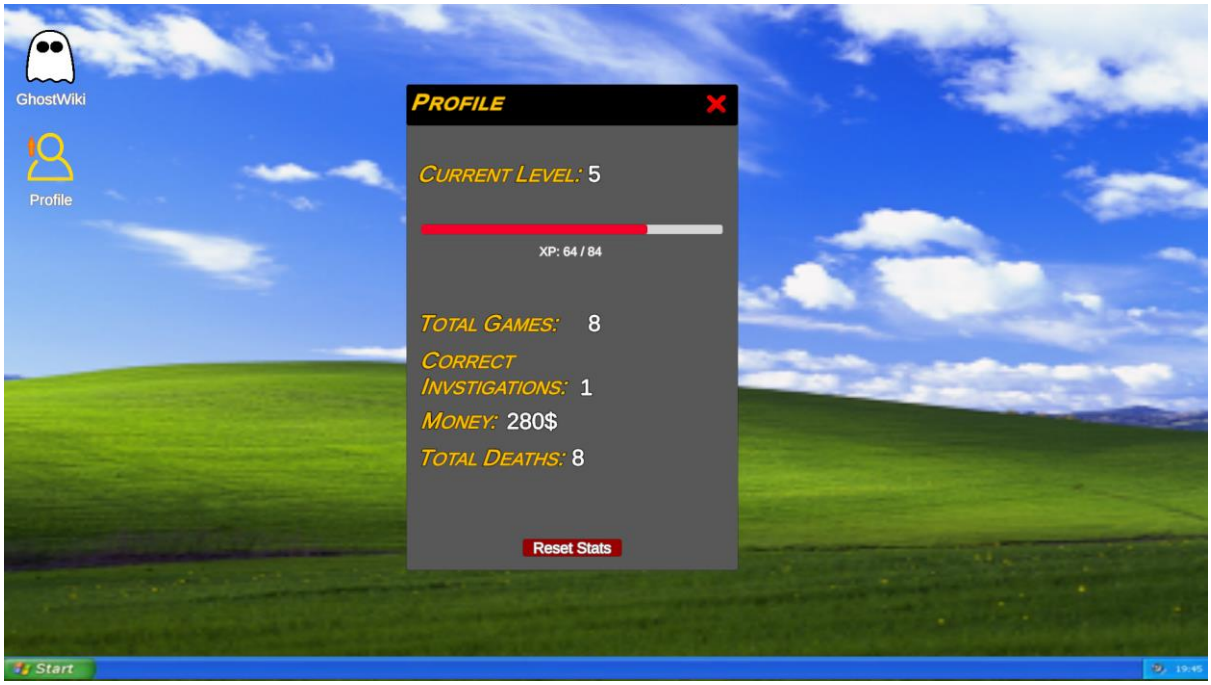
Εκτός από αυτά αρχικοποιούμε δύο λίστες οι οποίες αφορούν το όνομα των φαντασμάτων, και την περιγραφή – κείμενο που θα εμφανίζεται. Αυτές τις τιμές και συμβολοσειρές τι θέτουμε στο Inspector.

Κατά την αρχή του προγράμματος στην μέθοδο Start() , δημιουργούμε τα κουμπιά με την χρήση της GenerateButtons().

Μέσα στην συνάρτηση αυτή, με βάση των αριθμό των ονομάτων που θέσαμε από το Inspector δημιουργούμε τον αντίστοιχο αριθμό Prefab Κουμπιών , το καθένα με δικό του όνομα τύπου φαντάσματος και περιγραφής του. Τέλος [προσθέτουμε ένα onClick Listener, που ελέγχει τι θα συμβαίνει κάθε φορά που πατιέται ένα κουμπί.

Εφόσον κάποιο από αυτά πατηθεί καλείται η συνάρτηση OnNameClicked, η οποία με βάση το index με το οποίο δημιουργήθηκε το κουμπί εμφανίζει το αντίστοιχο κείμενο του ίδιου index.

4.4.3.3.2 Profile



Profile App

Με το πάτημα της εφαρμογής Profile θα ανοίξει το παραπάνω παράθυρο. Στο συγκεκριμένο παράθυρο ο παίκτης έχει την δυνατότητα να δει τα στατιστικά του.

Η μόνη λειτουργία που μπορεί να πραγματοποιήσει στο παράθυρο αυτό είναι να επαναφέρει τα στατιστικά του από την αρχή.

Το παράθυρο αυτό λειτουργεί με την βοήθεια του παρακάτω script:

```
public class PlayerStatsUI : MonoBehaviour {
    [Header("UI References")]
    public TMP_Text moneyText;
    public TMP_Text levelText;
    public TMP_Text xpText;
    public Slider xpSlider;
    public TMP_Text gamesText;
    public TMP_Text correctInvsText;
    public TMP_Text deathsText;
    Unity Message | 0 references
    private void OnEnable() {
        UpdateStatsUI();
    }

    3 references
    public void UpdateStatsUI() {
        if (PlayerLevelnMoney.Instance == null) {
            Debug.LogWarning("PlayerLevelnMoney instance not found!");
            return;
        }

        var player = PlayerLevelnMoney.Instance;

        moneyText.text = player.totalMoney + "$";
        levelText.text = player.playerLevel.ToString();
        xpText.text = "XP: " + player.currentXP + " / " + player.xpToNextLevel;
        gamesText.text = player.TotalInvestigations.ToString();
        correctInvsText.text = player.CorrectInvestigations.ToString();
        deathsText.text = player.TotalDeaths.ToString();


        if (xpSlider != null) {
            xpSlider.minValue = 0;
            xpSlider.maxValue = player.xpToNextLevel;
            xpSlider.value = player.currentXP;
        }
    }
}
```

PlayerStatsUI.cs

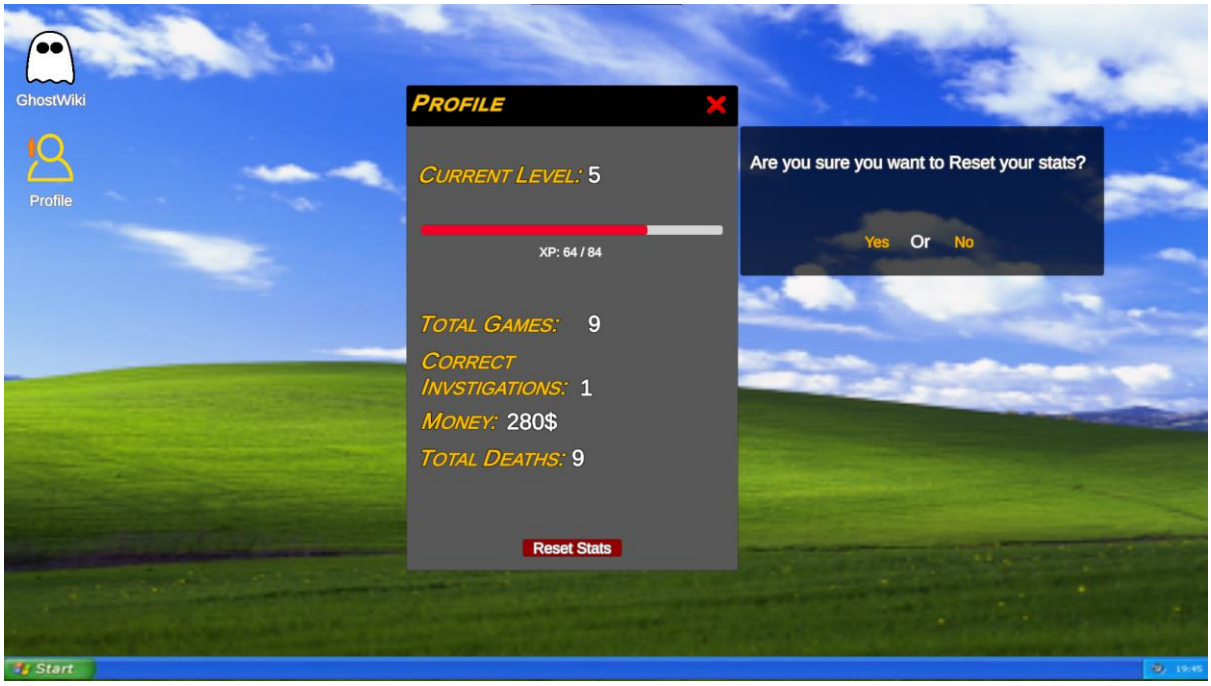
Κατά την αρχή του παιχνιδιού και του ίδιου του script στην μέθοδο OnEnable() καλούμε την συνάρτηση UpdateStatsUI().

Η συγκεκριμένη συνάρτηση επίσης καλούταν από το ResultsUI.cs κάθε φόρα που επέστρεφε ο παίκτης και ενημερώνονταν οι τιμές στον πίνακα.

Η UpdateStatsUI ανακτά τις τιμές από το Instance του PlayerLevelnMoney και για κάθε μια κατηγορία ενημερώνει κάθε κατηγορία, ορισμένες με πιο συγκεκριμένο format.

Τέλος φροντίζει να αλλάζει η τιμή του Slider με βάση τους πόντους εμπειρίας που έχουν υπολογιστεί και εξηγήσαμε νωρίτερα.

Στην περίπτωση που ο παίκτης αποφασίσει να επαναφέρει τα στατιστικά του, θα εμφανιστεί το παρακάτω παράθυρο:



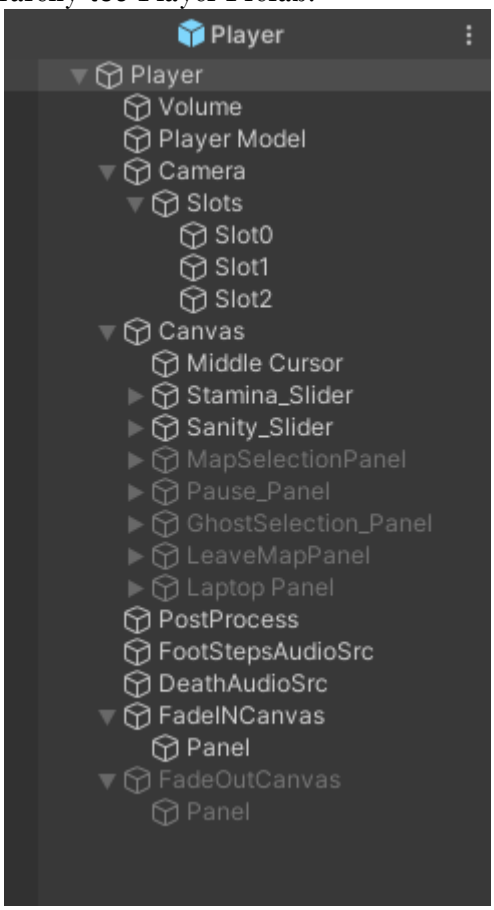
Reset Stats Panel

Αυτό γίνεται αφού θέσουμε σαν λειτουργία στο κουμπί να ανοίγει το Reset Stats Panel μέσω του Inspector. Αν ο παίκτης αποφασίσει ότι θέλει να μηδενίσει τα στατιστικά του, τότε με το πάτημα του κουμπιού **"Yes"** θα καλέσουμε την συνάρτηση ResetPlayerData() που αναλύσαμε στο script PlayerLevelnMoney.cs

4.4.4 Player Prefab

Στην συνέχεια θα μιλήσουμε για τον παίκτη τον ίδιο ο οποίος αποτελεί ένα αντικείμενο Prefab, καθώς και για όλα τα script και Components που είναι ενσωματωμένα πάνω του.

Παρακάτω φαίνεται το Hierarchy του Player Prefab:



Player Prefab Hierarchy

Για αρχή έχουμε ένα GameObject που λειτουργεί ως Parent για όλα τα υπόλοιπα GameObjects που είναι ενσωματωμένα στον παίκτη. Στο GameObject Player έχουμε θέσει το Tag του ως **"Player"**.

Πιο συγκεκριμένα στο Player έχουμε ενσωματωμένα πολλαπλά script που βοηθούν στην κίνηση του παίκτη, στην λειτουργία των Panel αλλά και άλλων ρυθμίσεων.

4.4.4.1 FPSController Script

```
Unity Script (2 asset references) | 7 references
public class FPSController : MonoBehaviour {
    [Header("Movement Settings")]
    public float walkingSpeed = 4.5f;
    public float runningSpeed = 6f;
    public float crouchSpeed = 2.5f;
    public float gravity = 20.0f;

    [Header("Look Settings")]
    public float lookSpeed = 100f;
    public float lookXLimit = 90.0f;

    [Header("References")]
    public Camera playerCamera;
    public Slider staminaSlider;

    [Header("Stamina Settings")]
    public float maxStamina = 5f;
    public float staminaDrainRate = 1f;
    public float crouchSprintDrainMultiplier = 1.25f;
    public float staminaRecoveryRate = 0.5f;
    public float staminaRecoveryDelay = 4f;

    [Header("Crouch Settings")]
    public float standingHeight = 3.0f;
    public float crouchHeight = 1.8f;
    public float cameraCrouchOffset = -0.1f;
    public float crouchTransitionSpeed = 8f;

    private CharacterController characterController;
    private Vector3 moveDirection = Vector3.zero;
    private float rotationX = 0f;
    private Vector3 cameraInitialLocalPos;

    [HideInInspector]
    public bool canMove = true;

    private bool isCrouching = false;
    private float currentStamina;
    private float staminaRecoveryTimer = 0f;
```

FPSController.cs

Στην αρχή του κώδικα αρχικοποιούμε μεταβλητές οι οποίες αφορούν την κίνηση του παίκτη.

Οι περισσότερες μεταβλητές είναι κατανοητές για την ενέργεια που θα επιτελούν, παρόλα αυτά υπάρχουν και κάποιες οι οποίες πρέπει να εξηγηθούν.

Πιο συγκεκριμένα:

- staminaSlider = αναφερόμαστε στο Slider που είναι εμφανές στην εικόνα του παίκτη και είναι η ένδειξη για την διάρκεια που μπορεί να τρέξει ο παίκτης μέχρι να επιστρέψει ξανά στο περπάτημα.
- maxStamina = η μέγιστη τιμή του Stamina του παίκτη για να μπορεί να τρέξει
- standHeight = το ύψος του παίκτη, το οποίο θα συμβαδίζει και με το Component Character Controller
- crouchHeight = το ύψος του παίκτη όταν βρίσκεται στην λειτουργία Crouch
- cameraCrouchOffset = Είναι το ύψος που θα μεταφερθεί η κάμερα όταν ο παίκτης είναι Crouch
- crouchTransitionSpeed = αφορά την ταχύτητα που θα γίνει η μετάβαση από Standing → Crouch και το αντίθετο.
- cameraInitialLocalPos = χρησιμοποιείται για αρχικοποίηση της θέσης της κάμερας του παίκτη

```
void Awake() {
    characterController = GetComponent<CharacterController>();
    cameraInitialLocalPos = playerCamera.transform.localPosition;

    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;

    currentStamina = maxStamina;
    if (staminaSlider != null) {
        staminaSlider.maxValue = maxStamina;
        staminaSlider.value = maxStamina;
    }
}

Unity Message | 0 references
void Update() {
    HandleCrouchInput();
    HandleMovement();
    UpdateStaminaUI();
}

Unity Message | 0 references
private void LateUpdate() {
    HandleLook();
}
```

FPSController.cs Part2

Στην Awake ανακτούμε τιμές και μεταβλητές από το Component CharacterController, και το position της κάμερας του παίκτη. Χρησιμοποιούμε το localPosition για να ανακτήσουμε την τοποθεσία της κάμερας σε σχέση με το Parent GameObject.
Στην συνέχεια κλειδώνουμε τον κέρσορα μας και τον θέτουμε ως αόρατο, και ορίζουμε τις μεταβλητές που αφορούν την μέγιστη τιμή του Stamina Slider καθώς και την παρούσα τιμή που θα ξεκινήσει.

Στην Update() χρησιμοποιούνται τρείς συναρτήσεις για τον έλεγχο της κίνησης του παίκτη και του Stamina Slider, ενώ την LateUpdate() την χρησιμοποιούμε μόνο για την κίνηση της κάμερας του παίκτη.

Ας αναλύσουμε μία προς μία τις συναρτήσεις:

HandleMovement Function

```
1 reference
void HandleMovement() {
    if (!canMove) return;

    Vector3 forward = transform.TransformDirection(Vector3.forward);
    Vector3 right = transform.TransformDirection(Vector3.right);

    bool isRunningKey = Input.GetKey(KeyCode.LeftShift);
    bool canSprint = currentStamina > 0;


    bool isRunning = isRunningKey && canSprint;

    float speed;
    if (isCrouching) {
        if (isRunning) {
            speed = 5f;
        } else {
            speed = crouchSpeed;
        }
    } else {
        speed = isRunning ? runningSpeed : walkingSpeed;
    }


    float inputX = Input.GetAxis("Horizontal");
    float inputY = Input.GetAxis("Vertical");

    Vector3 horizontalMove = (forward * inputY + right * inputX) * speed;
    moveDirection.x = horizontalMove.x;
    moveDirection.z = horizontalMove.z;

    if (characterController.isGrounded) {
        moveDirection.y = -1f;
    } else {
        moveDirection.y -= gravity * Time.deltaTime;
    }

    characterController.Move(moveDirection * Time.deltaTime);
```

```
    if (isRunning && (inputX != 0 || inputY != 0)) {
        float drain = staminaDrainRate;


        if (isCrouching) {
            drain *= crouchSprintDrainMultiplier;
        }

        currentStamina -= drain * Time.deltaTime;
        staminaRecoveryTimer = 0f;
    } else {

        if (currentStamina < maxStamina) {
            staminaRecoveryTimer += Time.deltaTime;
            if (staminaRecoveryTimer >= staminaRecoveryDelay) {
                currentStamina += staminaRecoveryRate * Time.deltaTime;
            }
        }
    }


    currentStamina = Mathf.Clamp(currentStamina, 0f, maxStamina);
}
```

HandleMovement Function

Η παραπάνω συνάρτηση είναι υπεύθυνη για την κίνηση του παίκτη.

Για αρχή ελέγχουμε αν ο παίκτης μπορεί να κινηθεί ελέγχοντας την Boolean μεταβλητή canMove, εφόσον μπορεί να κινηθεί , ορίζουμε δύο νέες μεταβλητές τύπου Vector3. Τέτοιου είδους μεταβλητές χρησιμοποιούνται για να εκφράσουν συντεταγμένες στους άξονες X,Y,Z.

Επιπλέον αρχικοποιούμε δύο νέες μεταβλητές Boolean που δέχονται απευθείας τιμή. Η isRunningKey είναι true όταν είναι πατημένο το πλήκτρο LeftShift, και η canSprint είναι true όταν ο παίκτης διαθέτει Stamina για να μπορεί να τρέξει. Η σύζευξη αυτών των δύων συνθηκών μας δίνει στην συνέχεια την Boolean μεταβλητή isRunning.

Στην συνέχεια ελέγχουμε αν ο παίκτης είναι σε Crouch λειτουργία. Αν ο παίκτης είναι Crouch τότε ελέγχουμε αν τρέχει ή όχι για να θέσουμε κατάλληλα την ταχύτητα του. Αν δεν είναι σε λειτουργία Crouch τότε με μια συνθήκη if θέτουμε την ταχύτητα του με κεντρική συνθήκη το isRunning, η οποία αν είναι true τότε ορίζουμε σαν ταχύτητα του παίκτη το runningspeed αλλιώς ορίζουμε ως ταχύτητα το walkingspeed.

Οι μεταβλητές inputX και inputY χρησιμοποιούνται για να ανακτήσουμε την κατεύθυνση του παίκτη. Αυτό συμβαίνει με την εντολή GetAxis() και σαν παράμετρο το όνομα του πεδίου Horizontal (A ή D) ή Vertical (W ή S). Οι παράμετροι αυτοί είναι built-in σε κάθε project του unity και ο χρήστης μπορεί να τις επεξεργαστεί ή και να προσθέσει δικές του παραμέτρους μέσω των ρυθμίσεων. Η τιμή που επιστρέφουν για τον κάθε άξονα είναι -1 ή 1 εφόσον πατηθεί το αντίστοιχο πλήκτρο για κίνηση.

Εφόσον ληφθούν και αυτές οι τιμές, υπολογίζεται ένα Vector3 με όνομα horizontalmove όπου προσθέτουμε τις κατευθύνσεις και δίνουμε στον παίκτη την επιθυμητή ταχύτητα με τον πολλαπλασιασμό με το speed.
Έτσι το αποτέλεσμα μας αποτελεί το τελικό διάνυσμα της κατεύθυνσης του παίκτη.

Τέλος υπολογίζουμε το moveDirection του παίκτη ανάλογα με τον άξονα που θα κινηθεί ο παίκτης. Ο άξονας y δεν χρησιμοποιείται καθώς αφορά άλλες κινήσεις όπως άλματα και βαρύτητα, το οποίο και χρησιμοποιούμε παρακάτω για να ορίσουμε στον παίκτη δύναμη τεχνητής βαρύτητας.

Κλείνοντας με την απλή κίνηση του παίκτη ενημερώνουμε το charactercontroller ότι πρόκειται ο παίκτης να κινηθεί με την συνάρτηση Move() και σαν παράμετρο το moveDirection * Time.deltaTime.

Το Time.deltaTime είναι **πολύ** χρήσιμο ειδικά στην κίνηση, αλλά και σε άλλες ενέργειες, του παίκτη καθώς χωρίς αυτό, ολόκληρη η κίνηση του παίκτη θα βασιζόταν μόνο στην δύναμη του συστήματος του.
Με άλλα λόγια ένας ισχυρός υπολογιστής θα μετακινούταν μεγαλύτερη απόσταση από ότι ένας αδύναμος υπολογιστής. Έτσι το Time.deltaTime εξισορροπεί την διαφορά των συστημάτων αυτών και θέτει την κίνηση του παίκτη με βάση τον αριθμό καρέ του (FPS).

Τέλος για να ολοκληρωθεί με επιτυχία η κίνηση του παίκτη, πρέπει να ελέγξουμε να μικραίνει το διαθέσιμο Stamina του παίκτη όσο ο παίκτης τρέχει και ταυτόχρονα κινείται. Εάν ο παίκτης βρίσκεται σε λειτουργία Crouch τότε ο ρυθμός που αφαιρείται το stamina αυξάνεται και έτσι αφαιρούμε από το Slider το διαθέσιμο stamina μέχρι να σταματήσει ο παίκτης να τρέχει . Εφόσον το stamina του παίκτη δεν έχει την μέγιστη τιμή, θα γίνεται προσπάθεια αναπλήρωσης του.
Να σημειωθεί ότι σε τέτοιου είδους ενέργειες χρησιμοποιούμε ξανά το Time.deltaTime καθώς θέλουμε να ισοσταθμίσουμε την λειτουργία του τρεξίματος με αυτήν της αφαίρεσης stamina χωρίς να εξαρτάται από τα καρέ του παίκτη.

HandleLook Function

```
void HandleLook() {
    if (!canMove) return;

    float mouseY = Input.GetAxis("Mouse Y") * lookSpeed * Time.deltaTime;
    float mouseX = Input.GetAxis("Mouse X") * lookSpeed * Time.deltaTime;

    rotationX += -mouseY;
    rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);

    Quaternion xRotation = Quaternion.Euler(rotationX, 0, 0);
    playerCamera.transform.localRotation = xRotation;

    transform.rotation *= Quaternion.Euler(0, mouseX, 0);
}
```

HandleLook Function

Η συνάρτηση αυτή χρησιμοποιείται για να μπορεί ο παίκτης να χειρίζεται την κάμερα με την βοήθεια του ποντικιού.

Για αρχή γίνεται έλεγχος αν ο παίκτης μπορεί να κουνηθεί. Αυτός ο έλεγχος πραγματοποιείται με σκοπό να μην έχει την δυνατότητα ο παίκτης να χειρίζεται την κάμερα ή το CharacterController σε περιπτώσεις όπως :

1) Ανοιχτό Pause Menu και Οποιοδήποτε Panel ή Menu
2) Μετακίνηση κάμερας σε οποιοδήποτε Camera Spot

Εφόσον επαληθεύσουμε πως ο παίκτης έχει την δυνατότητα να κουνηθεί, διαβάζουμε σε δύο μεταβλητές mouseY, mouseX τύπου float την κατεύθυνση τους στον κάθετο και οριζόντιο άξονα.

Στην συνέχεια με τους κατάλληλους υπολογισμούς μετατρέπουμε τις τιμές αυτές σε συντεταγμένες για την περιστροφή της κάμερα σε μοίρες, οι οποίες προκύπτουν από την χρήση των Quaternion.Euler(), όπου κάθε παράμετρος της αφορά τους άξονες x,y,z.

HandleCrouchInput Function

```
void HandleCrouchInput() {
    if (Input.GetKeyDown(KeyCode.C)) {
        isCrouching = !isCrouching;
    }

    float targetHeight = isCrouching ? crouchHeight : standingHeight;
    characterController.height = Mathf.Lerp(characterController.height, targetHeight, Time.deltaTime * crouchTransitionSpeed);

    Vector3 targetCameraPos = cameraInitialLocalPos + (isCrouching ? new Vector3(0, cameraCrouchOffset, 0) : Vector3.zero);
    playerCamera.transform.localPosition = Vector3.Lerp(playerCamera.transform.localPosition, targetCameraPos, Time.deltaTime * crouchTransitionSpeed);
}
```

HandleCrouchInput Function

Η συγκεκριμένη συνάρτηση χειρίζεται την λειτουργία του Crouch με τον έλεγχο για το πάτημα του πλήκτρου C. Εφόσον το πλήκτρο αυτό πατηθεί, υπολογίζουμε το νέο ύψος του παίκτη που θα έχει ανάλογα με το αν είναι Crouched ή όχι, καθώς και η ομαλότητα να προσαρμοστεί η κάμερα και ο ίδιος ο παίκτης , με την βοήθεια της συνάρτησης Mathf.Lerp().

4.4.4.2 MainMenuFunctions.cs

Το script MainMenuFunctions είναι υπεύθυνο για την απεικόνιση των στοιχείων UI στην εικόνα του παίκτη.

```
public class MainMenuFucntions : MonoBehaviour {

    public GameObject CursorImg;
    public GameObject StaminaBar;
    public GameObject SanityBar;
    public GameObject PauseMenu;
    public GameObject LaptopPanel;
    private GameObject player;
    private FPSController fpsController;
    public GameObject Tablet;

    private AudioSource[] audioSources;



    public GameObject AudioPanel;
    public GameObject ControlsPanel;
    public GameObject GraphicsPanel;
    public GameObject SETTINGSPANEL;
    public GameObject MainPanel;



    23 references
    public enum UIState { None, PauseMenu, Tablet, MapSelector, MapLeaver, Laptop }
    21 references
    public static UIState CurrentUI { get;  set; } = UIState.None;
```

MainMenuFunctions.cs Part1

Το πρόγραμμα ξεκινάει με το να αρχικοποιούμε όλα τα διαθέσιμα Panel που έχει στην διάθεση του ο παίκτης. Κάθε ένα από αυτά τα αντικείμενα που είναι public , το θέτουμε από το Inspector του παίκτη. Όσα GameObjects είναι private τα λαμβάνουμε μέσω αναφορών σε αυτά μέσα από το ίδιο το script.

Τα UIState τα χρησιμοποιούμε για να μπορούμε να δηλώσουμε σε οποιοδήποτε script την κατάσταση που βρίσκεται το UI του παίκτη. π.χ. όταν ο παίκτης ανοίξει το PauseMenu , αυτό θα θέσουμε ως τρέχων UIState σε όσα script επηρεάσουν το UI. Επιπλέον θέτουμε μια μεταβλητή τύπου UIState με όνομα CurrentUI για να μπορούμε να κάνουμε αναφορά στην διαχείριση των Panel.

```
private void Awake() {

    Cursor.lockState = CursorLockMode.Confined;
    player = GameObject.FindGameObjectWithTag("Player");

    if (player != null) {
        fpsController = player.GetComponent<FPSController>();
    }

}
Unity Message | 0 references
private void Start() {
    audioSources = FindObjectsOfType<AudioSource>(true);
}

Unity Message | 0 references
private void Update() {

    if (Input.GetKeyDown(KeyCode.Escape)) {
        if (CurrentUI == UIState.None && SceneManager.GetActiveScene().name != "Entry Scene") PauseGame();
        else if (CurrentUI == UIState.PauseMenu && SceneManager.GetActiveScene().name != "Entry Scene") ContinueGame();
    }

    if (Input.GetKeyDown(KeyCode.B)) {
        if (CurrentUI == UIState.None && SceneManager.GetActiveScene().name != "Entry Scene") OpenTablet();
        else if (CurrentUI == UIState.Tablet && SceneManager.GetActiveScene().name != "Entry Scene") CloseTablet();
    }

}
```

MainMenuFunctions.cs Part2

Στην μέθοδο Awake(), δηλώνουμε τον κέρσορα να ξεκινάει σε μορφή Confined που σημαίνει ότι ο κέρσορας δεν θα μπορεί να μετακινηθεί έξω από το παράθυρο του παιχνιδιού.

Στην μέθοδο Start(), βρίσκουμε και προσαρτούμε στον πίνακα audioSources όλα τα αντικείμενα με Component : AudioSource. Η παράμετρος true χρησιμοποιείται για τον έλεγχο αν συμπεριλάβουμε και τα ανενεργά GameObjects που εμπεριέχουν αυτό το Component.

Στην μέθοδο Update() ελέγχουμε με βάση το πλήκτρο Escape ή B ποιο Panel θα ανοίξουμε. Και στις δύο περιπτώσεις , ελέγχουμε αν δεν υπάρχει άλλο Panel ενεργό και αν η σκηνή στην οποία βρισκόμαστε δεν είναι η Entry Scene, τότε καλούμε την κατάλληλη συνάρτηση.

PauseGame Function

```
public void PauseGame() {
    CurrentUI = UIState.PauseMenu;
    StaminaBar.SetActive(false);
    SanityBar.SetActive(false);
    CursorImg.SetActive(false);
    Time.timeScale = 0f;
    PauseMenu.SetActive(true);


    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;

    if (fpsController != null) {
        fpsController.canMove = false;
    }

    foreach (var audioSource in audioSources) {
        if (audioSource != null) {
            audioSource.Pause();
        }
    }
}
```

PauseGame Function

Με την κλήση της PauseGame, ενημερώνουμε το UIState σε PauseMenu και απενεργοποιούμε όλα τα UI Elements με σκοπό να είναι πιο καθαρό το παράθυρο . Ακόμα με την εντολή Time.timeScale = 0f; Ελέγχουμε την ροή του χρόνου του παιχνιδιού (Αν θέσουμε ως τιμή 0f τότε σημαίνει παγώνουμε την ροή του χρόνου στο παιχνίδι μας, διαφορετικά αν η τιμή είναι 1f τότε η ροή του χρόνο είναι κανονική). Εμφανίζουμε το PausePanel, και θέτουμε τον κέρσορα σε ορατή μορφή. Στην συνέχεια αλλάζουμε την Boolean τιμή canMove σε false . Με την εντολή αυτή θέτουμε στο script FPSController ότι ο παίκτης δεν μπορεί κουνηθεί ούτε ο ίδιος ούτε η κάμερα, όσο είναι ανοιχτό το Panel. Τέλος κάνουμε παύση όλους τους ήχου κατά την διάρκεια του που ο παίκτης είναι στο Pause Menu.

ContinueGame Function

```
public void ContinueGame() {
    CurrentUI = UIState.None;
    StaminaBar.SetActive(true);
    SanityBar.SetActive(true);
    CursorImg.SetActive(true);
    Time.timeScale = 1f;
    PauseMenu.SetActive(false);


    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;

    if (fpsController != null) {
        fpsController.canMove = true;
    }


    foreach (var audioSource in audioSources) {
        if (audioSource != null) {
            audioSource.UnPause();
        }
    }
    if (AudioPanel != null) AudioPanel.SetActive(false);
    if (ControlsPanel != null) ControlsPanel.SetActive(false);
    if (GraphicsPanel != null) GraphicsPanel.SetActive(false);
    if (SETTINGSPANEL != null) SETTINGSPANEL.SetActive(false);
    MainPanel.SetActive(true);
}
```

ContinueGame Function

Η παραπάνω συνάρτηση πραγματοποιεί την ακριβώς αντίθετη διαδικασία από την PauseGame() που μελετήσαμε παραπάνω . Με λίγα λόγια στην συνάρτηση αυτή ενημερώνουμε το UIState και εμφανίζουμε τα UIElements, ξεπαγώνουμε την ροή του χρόνου και θέτουμε τον κέρσορα σε αόρατη λειτουργία.

Στην συνέχεια επιτρέπουμε ξανά την κίνηση του παίκτη, και για κάθε Audiosource που υπάρχει στην σκηνή. Τέλος επειδή ο παίκτης είναι πιθανό να κλείσει το PauseMenu ενώ βρίσκεται σε διαφορετικό Panel από το αρχικό (π.χ. ο παίκτης έχει ανοίξει το μενού γραφικών και πατάει το κουμπί Escape για έξοδο από όλα τα menus) φροντίζουμε να απενεργοποιήσουμε κατά την έξοδο από το PauseMenu όλα τα υπόλοιπα Panels και να ενεργοποιήσουμε μόνο το αρχικό MAIN PANEL.

PauseMenu Button Functions

```
0 references
public void SinglePlayer() {
    SceneManager.LoadScene("Lobby Menu", LoadSceneMode.Single);
}

0 references
public void StartTutorial() {
    SceneManager.LoadScene("Tutorial Scene", LoadSceneMode.Single);
}

0 references
public void QuitGame() {
    Application.Quit();
}

0 references
public void BackToMainMenu() {
    Time.timeScale = 1f;

    SceneManager.LoadScene("Entry Scene", LoadSceneMode.Single);

}
```

Pause Menu Functions

1) Η συνάρτηση SinglePlayer φορτώνει την σκηνή Lobby Menu . Το LoadSceneMode.Single χρησιμοποιείται για να διακόψουμε την λειτουργία των υπόλοιπων σκηνών και την φόρτωση επόμενης σκηνής.
2) Η συνάρτηση StartTutorial φορτώνει την σκηνή Tutorial Scene
3) Η συνάρτηση QuitGame() κλείνει την εφαρμογή του παιχνιδιού
4) Η συνάρτηση BackToMainMenu() επιτρέπει την επιστροφή της αρχικής σκηνής και θέτει την ροή του χρόνου σε κανονική μορφή για λόγους ασφαλείας.

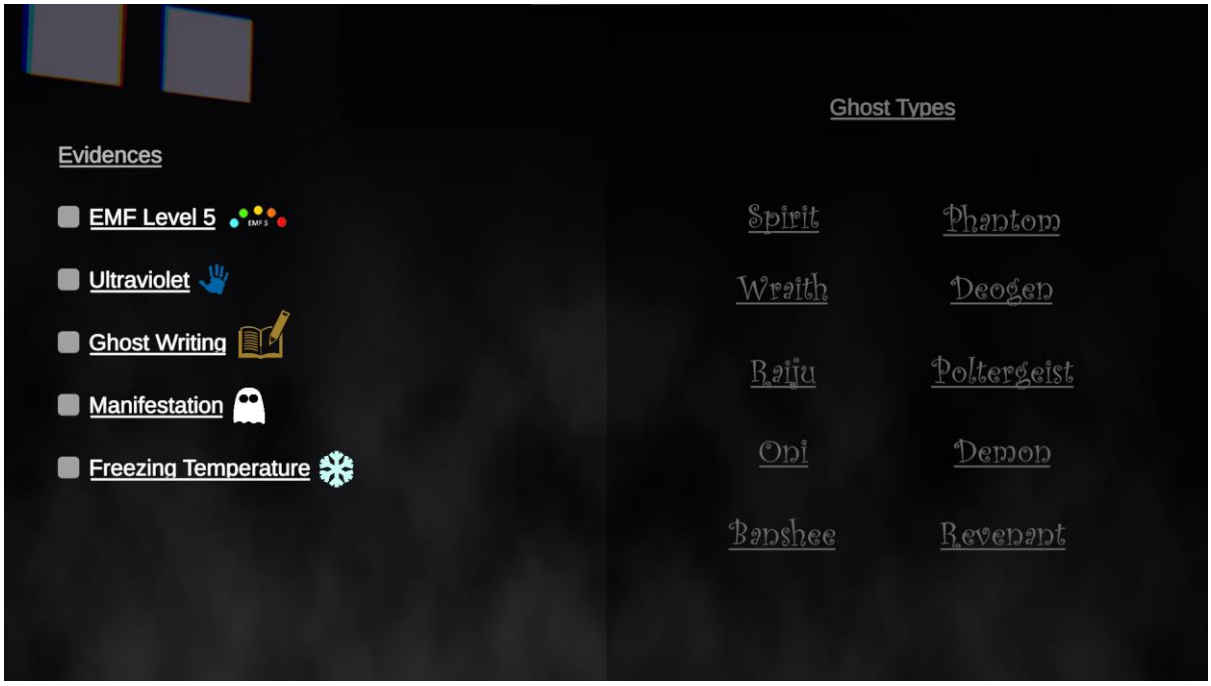
Open – Close Tablet Functions

```
public void OpenTablet() {
    CurrentUI = UIState.Tablet;
    StaminaBar.SetActive(false);
    SanityBar.SetActive(false);
    CursorImg.SetActive(false);
    Time.timeScale = 0f;

    Tablet.SetActive(true);
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;

    if (fpsController != null) {
        fpsController.canMove = false;
    }
    foreach (var audioSource in audioSources) {
        if (audioSource != null) {
            audioSource.Pause();
        }
    }
}

1 reference
public void CloseTablet() {
    CurrentUI = UIState.None;
    StaminaBar.SetActive(true);
    SanityBar.SetActive(true);
    CursorImg.SetActive(true);
    Time.timeScale = 1f;

    Tablet.SetActive(false);
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;

    if (fpsController != null) {
        fpsController.canMove = true;
    }

    foreach (var audioSource in audioSources) {
        if (audioSource != null) {
            audioSource.UnPause();
        }
    }
}
```

Open - Close Tablet Functions

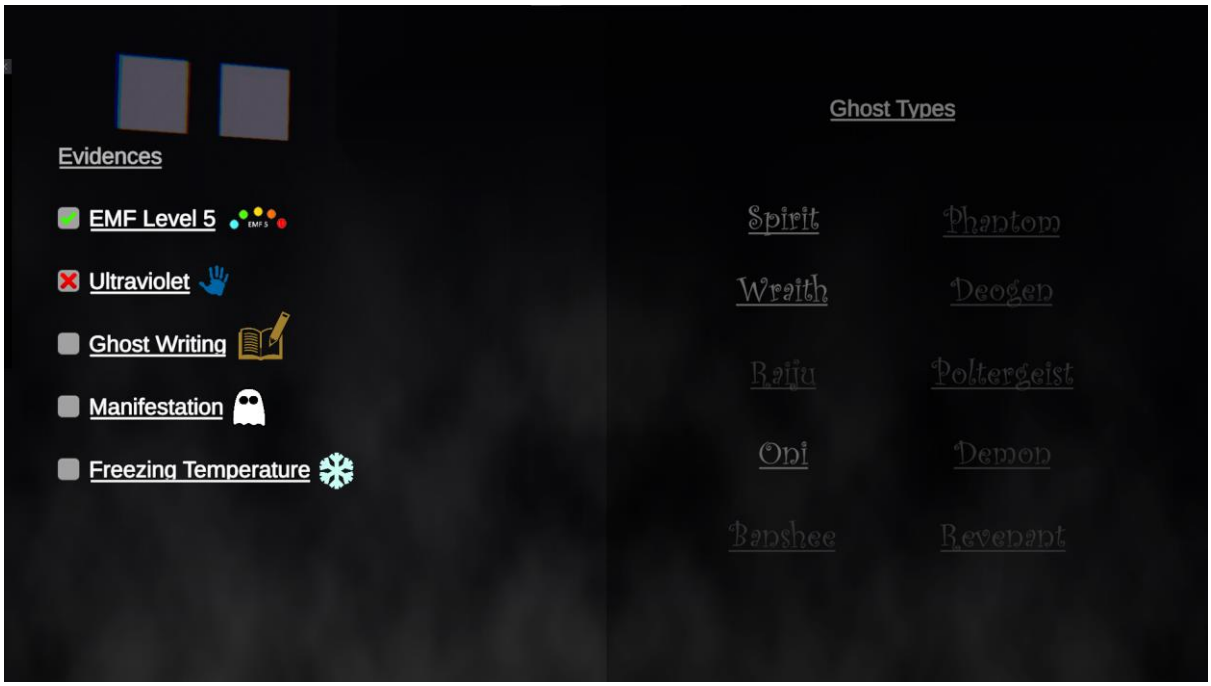
Οι παραπάνω συναρτήσεις λειτουργούν όμοια με τις συναρτήσεις PauseGame και ContinueGame. Αναλόγως την ενέργεια του παίκτη ενεργοποιούν ή απενεργοποιούν ορισμένα UI Elements, κάνουν ορατό τον κέρσορα, επιτρέπουν την δυνατότητα κίνησης μέσω της μεταβλητής canMove, και ανάλογα με την ενέργεια του παίκτη θέτουν όλους τους ήχους Pause ή Unpause λειτουργία.

Το Open – Close Tablet χρησιμεύει όταν ο παίκτης θέλει να σημειώσει και να επιλέξει τα στοιχεία που εμφάνισε το φάντασμα. Η μορφή του παραθύρου είναι η εξής:



GhostSelectionPanel

Στο αριστερό μέρος του Ghost Selection Panel ο παίκτης μπορεί να επιλέξει πιο από τα στοιχεία ανακάλυψε να εμφανίζει το φάντασμα. Εκτός από την σημείωση ότι βρέθηκε κάποιο στοιχείο ο παίκτης μπορεί επίσης να σημειώσει ότι κάποιο στοιχείο δεν χρησιμοποιείται από το φάντασμα, όπως φαίνεται παρακάτω και για τις δύο περιπτώσεις:



Valid and Invalid Evidence Selection

Καθώς ο παίκτης πατήσει κάποιο από τα κουμπιά , αυτομάτως τα ονόματα στην δεξιά πλευρά αλλάζουν το χρώμα τους σαν να σβήνονται από την λίστα για να δηλωθεί ότι το φάντασμα δεν είναι κάποιο από αυτές τις κατηγορίες.

Η λειτουργία όλου του παραπάνω Panel είναι συνδυασμός αρκετών script.
Τα script που χρησιμοποιούνται και θα αναλύσουμε παρακάτω είναι τα:

- GhostFilterManager.cs
- EvidenceCheckBox.cs
- GhostSelectedEffect.cs
- GhostTypeSelector.cs
- GhostEvidenceDatabase.cs

GhostFilterManager.cs

```
public static GhostFilterManager Instance;

private List<EvidenceCheckBox> evidenceBoxes = new List<EvidenceCheckBox>();

[Header("UI Ghost Labels")]
public List<GameObject> ghostLabelObjects;

private Dictionary<GhostTypeSelector.GhostType, GameObject> ghostLabelMap;

Unity Message | 0 references
private void Awake() {
    if (Instance == null) Instance = this;
    BuildGhostLabelMap();
}


1 reference
private void BuildGhostLabelMap() {
    ghostLabelMap = new Dictionary<GhostTypeSelector.GhostType, GameObject>();

    foreach (var obj in ghostLabelObjects) {
        string objName = obj.name.Trim();

        if (System.Enum.TryParse(objName, out GhostTypeSelector.GhostType ghostType)) {
            ghostLabelMap[ghostType] = obj;
        }
    }
}
```

GhostFilterManager.cs Part 1

Για αρχή θέτουμε το script αυτό ως static καθώς θα χρειαστεί σε πολλαπλά script να έχουμε πρόσβαση στις τιμές του και το ορίζουμε ως Singleton μέσα στην Awake(). Δημιουργούμε μια λίστα που συγκεντρώνει στοιχεία του τύπου EvidenceCheckBox, μία λίστα που συγκεντρώνει αντικείμενα τύπου GameObject που αφορά τους τύπους φαντασμάτων. Στην συνέχεια με την κλήση της BuildGhostLabelMap() κατασκευάζουμε ένα Dictionary, με σκοπό να συσχετίσουμε κάθε τιμή από το GhostTypeSelector.GhostType (πρόκειται για enum που περιέχει τους τύπους των φαντασμάτων) με το αντίστοιχο GameObject. Κάνουμε τις κατάλληλες αλλαγές πάνω στα ονόματα για να μην έχουν κενά με το Trim() και τέλος γίνεται προσπάθεια μετατροπής του ονόματος σε έγκυρη τιμή του enum . Εφόσον επιτύχει η μετατροπή τότε ο συνδυασμός αυτών των δύο αποθηκεύεται στο Dictionary.

```
public void RegisterCheckBox(EvidenceCheckBox box) {
    if (!evidenceBoxes.Contains(box)) {
        evidenceBoxes.Add(box);
    }
}

1 reference
public void UpdateGhostVisibility() {
    List<GhostTypeSelector.GhostType> validGhosts = new List<GhostTypeSelector.GhostType>();

    foreach (var entry in GhostEvidenceDatabase.GhostEvidenceMap) {
        bool isValid = true;

        foreach (var box in evidenceBoxes) {
            var type = box.evidenceType;


            if (box.isChecked && !entry.Value.Contains(type)) {
                isValid = false;
                break;
            }

            if (box.isX && entry.Value.Contains(type)) {
                isValid = false;
                break;
            }
        }

        if (isValid) {
            validGhosts.Add(entry.Key);
        }
    }

    foreach (var pair in ghostLabelMap) {
        bool isValid = validGhosts.Contains(pair.Key);

        CanvasGroup canvasGroup = pair.Value.GetComponent<CanvasGroup>();
        if (canvasGroup == null) {
            canvasGroup = pair.Value.AddComponent<CanvasGroup>();
        }

        canvasGroup.alpha = isValid ? 1f : 0.2f;
    }
}
```

GhostFilterManager.cs Part2

Η συνάρτηση RegisterCheckBox ελέγχει αν κάποιο από τα κουτιά δεν έχει καταχωρηθεί μέσα στην λίστα evidenceBoxes και σε περίπτωση που δεν έχει το καταχωρεί σε αυτήν.

Στην συνέχεια στην μέθοδο UpdateGhostVisibility , δημιουργούμε μια λίστα όπου αποθηκεύουμε τα "έγκυρα" φαντάσματα με βάση τα στοιχεία τους που επιλέγει ο χρήστης.

Για αρχή θέτουμε για κάθε τύπο φαντάσματος ότι είναι έγκυρο. Στην συνέχεια ακολουθούν έλεγχοι για τα evidenceboxes ανάλογα με το αν ο παίκτης έχει θέσει ως έγκυρα ή άκυρα και αν ανταποκρίνεται το κουτί που επέλεξε ο παίκτης στα διαθέσιμα στοιχεία του φαντάσματος.
Μετά από τους ελέγχους αυτούς προσθέτουμε στην λίστα validGhosts όσα φαντάσματα ήταν έγκυρα.

Τέλος ενημερώνουμε το UI με βάση την λίστα αυτή και προσαρμόζουμε το transparency των μη έγκυρων φαντασμάτων για να φαίνονται τα ονόματα τους ως σβησμένα.

EvidenceCheckBox

```
public class EvidenceCheckBox : MonoBehaviour {
    public EvidenceType evidenceType;
    public GameObject Check_Sprite;
    public GameObject X_Sprite;
    public Button mybutton;
    [HideInInspector] public bool isChecked = false;
    [HideInInspector] public bool isX = false;
    [HideInInspector] public bool isEmpty = true;

    Unity Message | 0 references
    private void Awake() {
        GhostFilterManager.Instance?.RegisterCheckBox(this);
        mybutton.onClick.AddListener(ManageBox);
    }

    1 reference
    public void ManageBox() {
        if (isEmpty) {
            CheckBox();
        } else if (isChecked) {
            X_Box();
        } else if (isX) {
            EmptyBox();
        }
        GhostFilterManager.Instance?.UpdateGhostVisibility();
    }

    1 reference
    public void CheckBox() {
        isChecked = true;
        isEmpty = false;
        isX = false;

        Check_Sprite.SetActive(true);
        X_Sprite.SetActive(false);
    }

    1 reference
    public void X_Box() {
        isChecked = false;
        isX = true;

        Check_Sprite.SetActive(false);
        X_Sprite.SetActive(true);
    }

    1 reference
    public void EmptyBox() {
        isX = false;
        isEmpty = true;

        X_Sprite.SetActive(false);
        Check_Sprite.SetActive(false);
    }
}
```

EvidenceCheckBox.cs

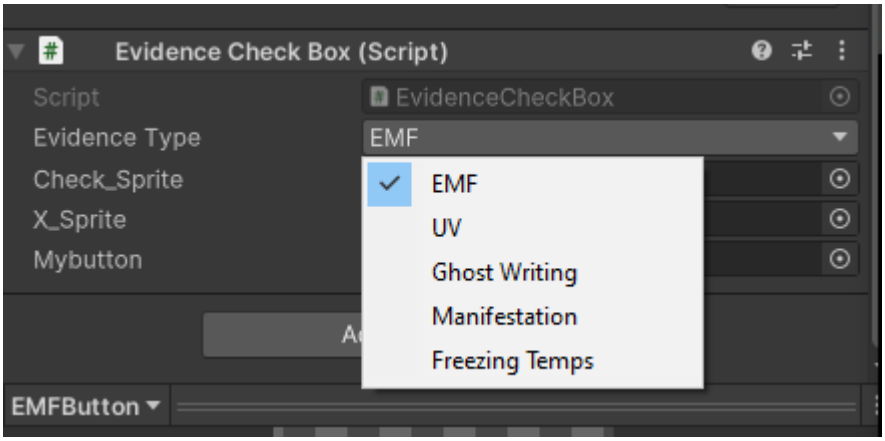
Στην αρχή του κώδικα μέσα στην μέθοδο Awake, κάνουμε αναφορά στο script GhostFilterManager και καλούμε την συνάρτηση RegisterCheckBox() και σαν παράμετρο βάζουμε το κουμπί που είναι ήδη πάνω το script ως Component.
Στην συνέχεια προσθέτουμε έναν Listener για το κουμπί αυτό , και κάθε φορά που πατιέται θα καλείται η συνάρτηση ManageBox().

Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για την σωστή εναλλαγή μεταξύ των καταστάσεων που θέτει ο παίκτης το κουμπί. Οι τρείς αυτές καταστάσεις είναι

1) isEmpty δηλαδή ο παίκτης δεν έχει αποφασίσει αν το στοιχείο είναι ένα από αυτά που διαχειρίζεται το φάντασμα
2) isChecked δηλαδή ο παίκτης έχει ανακαλύψει ότι το συγκεκριμένο στοιχείο ανήκει στο φάντασμα
3) isX δηλαδή η συγκεκριμένη κατηγορία δεν αποτελεί στοιχείο που μπορεί να εμφανίσει το φάντασμα.

Ανάλογα το ποια κατάσταση από τις παραπάνω έχουμε ενημερώνουμε και κατάλληλα το UI με την εμφάνιση συγκεκριμένων ενδείξεων της επιλογής του παίκτη.

Το συγκεκριμένο script το εισάγουμε σε κάθε κουμπί που αφορά τα evidences και από το inspector θέτουμε μέσα από το enum EvidenceType σε ποιο στοιχείο αναφέρεται το κουμπί μας.



Inspector Button Evidence Type Initialize

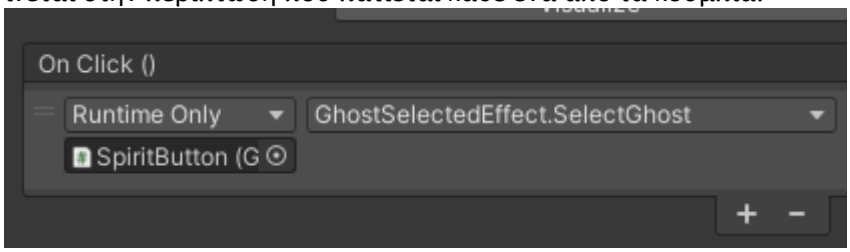
GhostSelectedEffect.cs

```
public class GhostSelectedEffect : MonoBehaviour {
    public GameObject img;
    public string ghostName;
    public TMP_Text ghostText;
    private bool isSelected = false;

    private static List<GhostSelectedEffect> allGhosts = new List<GhostSelectedEffect>();
    5 references
    public static GhostSelectedEffect CurrentlySelected { get; private set; }

    Unity Message | 0 references
    private void Awake() {
        allGhosts.Add(this);
    }

    Unity Message | 0 references
    private void OnDestroy() {
        allGhosts.Remove(this);
    }

    0 references
    public void SelectGhost() {
        foreach (GhostSelectedEffect ghost in allGhosts) {
            if (ghost != this) {
                ghost.Deselect();
            }
        }

        isSelected = !isSelected;
        img.SetActive(isSelected);

        CurrentlySelected = isSelected ? this : null;
        ghostText.text = ghostName;
    }

    1 reference
    public void Deselect() {
        isSelected = false;
        img.SetActive(false);

        if (CurrentlySelected == this) {
            CurrentlySelected = null;
        }
    }
```

GhostSelectedEffect.cs

Το συγκεκριμένο script είναι υπεύθυνο για να προσαρμόσει την κατάσταση των φαντασμάτων στο UI αλλά και να ενημερώσει την τιμή του τρέχοντος φαντάσματος που έχει επιλέξει . Πιο

συγκεκριμένα στην αρχή δημιουργούμε μια λίστα που θα αφορά όλα τα φαντάσματα που έχουν αυτό το συγκεκριμένο script. Προσθέτουμε μέσα σε αυτήν την λίστα το αντικείμενο που βρίσκεται το script αυτό ενσωματωμένο και επειδή το αντικείμενο μας είναι Button, θέτουμε από τον Inspector τι θα γίνεται στην περίπτωση που πατιέται κάθε ένα από τα κουμπιά.



OnClick() Running SelectGhost Function

Στην συνάρτηση SelectGhost προσπαθούμε να διασφαλίσουμε ότι μόνο ένα φάντασμα είναι ενεργό σαν επιλογή του παίκτη, οπότε στην περίπτωση που ο παίκτης έχει ήδη επιλέξει έναν τύπο φαντάσματος και αποφασίσει να αλλάξει την επιλογή του, τότε θα καλέσουμε την Deselect() η οποία θα κάνει σβήσει την επιλογή του παίκτη και θα θέση την επιλογή του σε null.

GhostTypeSelector.cs

```
15 references
public static GhostTypeSelector Instance { get; private set; }

28 references
public enum GhostType {
    Spirit,
    Wraith,
    Raiju,
    Oni,
    Banshee,
    Phantom,
    Deogen,
    Poltergeist,
    Demon,
    Revenant
}

40 references
public enum EvidenceType {
    EMF,
    UV,
    GhostWriting,
    Manifestation,
    FreezingTemps
}

public GhostType selectedGhostType;
public List<EvidenceType> selectedEvidences;

Unity Message | 0 references
private void Awake() {

    if (Instance != null && Instance != this) {
        Destroy(gameObject);
    } else {
        Instance = this;
    }
}

Unity Message | 0 references
private void Start() {
    SelectRandomGhost();
}
```

GhostTypeSelector.cs Part 1

Το script αυτό είναι υπεύθυνο να θέσει στο φάντασμα μια κατηγορία οι οποίες θέτουμε μέσα στα enums GhostType σε συνδυασμό με τα EvidenceType. Πιο συγκεκριμένα αυτό το script χρησιμοποιείται επίσης ως Singleton και εφόσον δημιουργήσουμε τα enums που αφορούν τον τύπο του φαντάσματος και τα διαθέσιμα στοιχεία, καλούμε στην Start() την συνάρτηση SelectRandomGhost().

```
private void Start() {
    SelectRandomGhost();
}

1 reference
void SelectRandomGhost() {
    var ghostTypes = System.Enum.GetValues(typeof(GhostType));
    selectedGhostType = (GhostType)ghostTypes.GetValue(Random.Range(0, ghostTypes.Length));
    selectedEvidences = GhostEvidenceDatabase.GhostEvidenceMap[selectedGhostType];
    Debug.Log("Selected Ghost: " + selectedGhostType);
    Debug.Log("Evidences :" + selectedEvidences[0] + " and "+ selectedEvidences[1]);
}


5 references
public bool HasEvidence(EvidenceType evidence) {
    return selectedEvidences != null && selectedEvidences.Contains(evidence);
}

1 reference
public bool HasGhostWriting => HasEvidence(EvidenceType.GhostWriting);
1 reference
public bool HasEMF => HasEvidence(EvidenceType.EMF);
1 reference
public bool HasFreezingTemps => HasEvidence(EvidenceType.FreezingTemps);
1 reference
public bool HasUV => HasEvidence(EvidenceType.UV);
1 reference
public bool HasManifestation => HasEvidence(EvidenceType.Manifestation);
```

GhostTypeSelector.cs Part 2

Η συνάρτηση αυτή έχει ως σκοπό μέσα από το GhostEvidenceDatabase να λάβει ένα τυχαίο τύπο φαντάσματος, καθώς και το ζευγάρι των Evidence που έχουμε θέσει σε αυτό με την βοήθεια των mapping, και αρχικοποιούμε τις μεταβλητές selectedGhostType και selectedEvidences με τις τιμές αυτές.

Τέλος ανακτούμε σε Boolean μεταβλητές αν το φάντασμα περιέχει ορισμένα στοιχεία με σκοπό να χρησιμοποιηθούν ως συνθήκες σε παρακάτω scripts. Παρακάτω φαίνεται το Database από όπου έγινε η ανάκτηση στοιχείων για το GhostTypeSelector.cs

```
using System.Collections.Generic;
using static GhostTypeSelector;

2 references
public static class GhostEvidenceDatabase {
    public static Dictionary<GhostType, List<EvidenceType>> GhostEvidenceMap = new Dictionary<GhostType, List<EvidenceType>>()
    {
        { GhostType.Spirit, new List<EvidenceType> { EvidenceType.EMF, EvidenceType.GhostWriting } },
        { GhostType.Wraith, new List<EvidenceType> { EvidenceType.EMF, EvidenceType.FreezingTemps } },
        { GhostType.Raiju, new List<EvidenceType> { EvidenceType.EMF, EvidenceType.UV } },
        { GhostType.Oni, new List<EvidenceType> { EvidenceType.EMF, EvidenceType.Manifestation } },
        { GhostType.Banshee, new List<EvidenceType> { EvidenceType.UV, EvidenceType.Manifestation } },
        { GhostType.Phantom, new List<EvidenceType> { EvidenceType.UV, EvidenceType.FreezingTemps } },
        { GhostType.Deogen, new List<EvidenceType> { EvidenceType.UV, EvidenceType.GhostWriting } },
        { GhostType.Poltergeist, new List<EvidenceType> { EvidenceType.GhostWriting, EvidenceType.FreezingTemps } },
        { GhostType.Demon, new List<EvidenceType> { EvidenceType.GhostWriting, EvidenceType.Manifestation } },
        { GhostType.Revenant, new List<EvidenceType> { EvidenceType.Manifestation, EvidenceType.FreezingTemps } },
    };
```

GhostEvidenceDatabase.cs

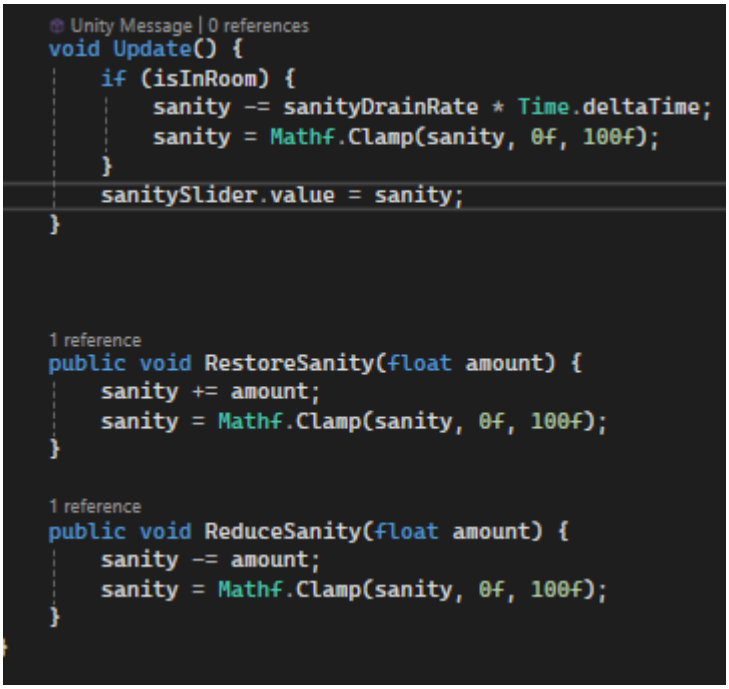
Το συγκεκριμένο script είναι κατά βάση ένα Dictionary το οποίο εμπεριέχει τον τύπο του φαντάσματος με τα στοιχεία που θα τον αφορούν και θα χρησιμοποιεί το φάντασμα στο παιχνίδι.

4.4.4.3 SanityManager.cs

```
Unity Script (1 asset reference) | 7 references
public class SanityManager : MonoBehaviour {
    public static SanityManager Instance;
    public float sanity = 100f;
    public float sanityDrainRate = .1f;
    public Slider sanitySlider;

    private bool isInRoom = false;



    Unity Message | 0 references
    private void Awake() {
        if (Instance == null) Instance = this;
    }
    Unity Message | 0 references
    void Start() {
        if (sanitySlider != null) {
            sanitySlider.maxValue = 100f;
            sanitySlider.value = sanity;
        }



        if (SceneManager.GetActiveScene().name == "Tutorial Scene") {
            ReduceSanity(35f);
        }
    }

    Unity Message | 0 references
    void OnTriggerStay(Collider other) {
        if (other.CompareTag("Room") || other.CompareTag("Ghost Room")) {
            isInRoom = true;
        }
    }

    Unity Message | 0 references
    void OnTriggerExit(Collider other) {
        if (other.CompareTag("Room") || other.CompareTag("Ghost Room")) {
            isInRoom = false;
        }
    }
```

SanityManager.cs Part 1

Το SanityManager.cs έχει ως λειτουργία να απεικονίζει σε μορφή UI τις μεταβολές του Sanity Slider του παίκτη. Για αρχή θέτουμε το αρχικό sanity του παίκτη να έχει την τιμή 100 και τον ρυθμό που θα αδειάζει το Sanity **μόνο** όταν ο παίκτης βρίσκεται μέσα στους χώρους που μπορεί να στοιχειώνει το φάντασμα. Σε περίπτωση που βρίσκεται μέσα σε έναν από αυτούς του χώρους ορίζουμε την τιμή isInRoom = true. Επιπλέον στην μέθοδο Start δημιουργούμε έναν πίνακα που θα περιλαμβάνει τα SanityMeds που θα έχει στην διάθεση του ο παίκτης, με σκοπό να κάνουμε αλλαγές στις τιμές του Sanity Slider.



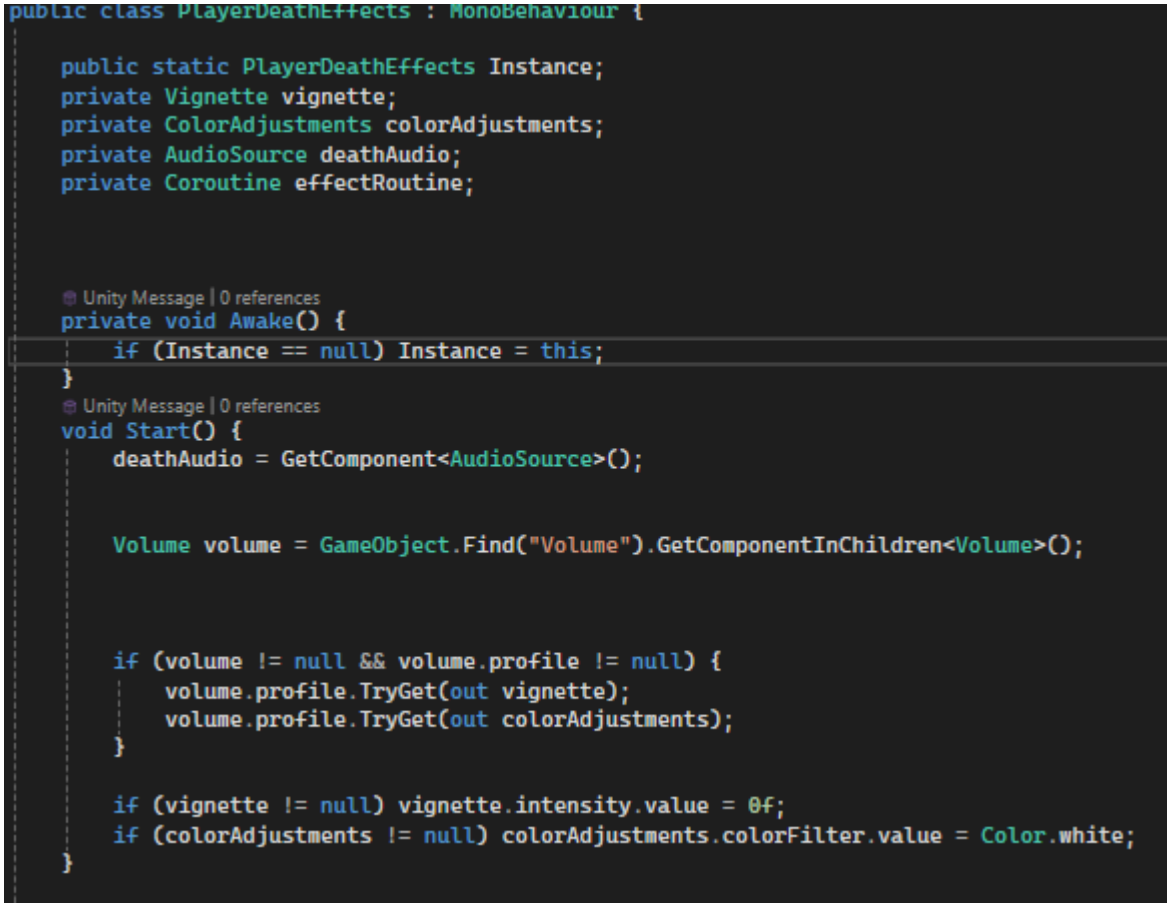
```
Unity Message | 0 references
void Update() {
    if (isInRoom) {
        sanity -= sanityDrainRate * Time.deltaTime;
        sanity = Mathf.Clamp(sanity, 0f, 100f);
    }
    sanitySlider.value = sanity;
}


1 reference
public void RestoreSanity(float amount) {
    sanity += amount;
    sanity = Mathf.Clamp(sanity, 0f, 100f);
}

1 reference
public void ReduceSanity(float amount) {
    sanity -= amount;
    sanity = Mathf.Clamp(sanity, 0f, 100f);
}
```

SanityManager.cs Part 2

Στην μέθοδο Update ενημερώνουμε την μεταβολή της τιμής του sanity, καθώς και την γραφική απεικόνιση πάνω στο Sanity Slider.

Τέλος χρησιμοποιούμε δύο συναρτήσεις με σκοπό να κάνουμε πιο άμεσες μεταβολές στο sanity του παίκτη, όπως αναπλήρωση και αφαίρεση από αυτό.

4.4.4.5 PlayerDeathEffects.cs



```
public class PlayerDeathEffects : MonoBehaviour {

    public static PlayerDeathEffects Instance;
    private Vignette vignette;
    private ColorAdjustments colorAdjustments;
    private AudioSource deathAudio;
    private Coroutine effectRoutine;



    Unity Message | 0 references
    private void Awake() {
        if (Instance == null) Instance = this;
    }
    Unity Message | 0 references
    void Start() {
        deathAudio = GetComponent<AudioSource>();


        Volume volume = GameObject.Find("Volume").GetComponentInChildren<Volume>();



        if (volume != null && volume.profile != null) {
            volume.profile.TryGet(out vignette);
            volume.profile.TryGet(out colorAdjustments);
        }

        if (vignette != null) vignette.intensity.value = 0f;
        if (colorAdjustments != null) colorAdjustments.colorFilter.value = Color.white;
    }
```

PlayerDeathEffects.cs Part 1

```
1 reference
public void PlayDeathEffects(float duration) {
    if (vignette == null || colorAdjustments == null) {
        Debug.LogWarning("Volume overrides not found! Check Volume setup.");
        return;
    }

    if (deathAudio != null) deathAudio.Play();

    if (effectRoutine != null) StopCoroutine(effectRoutine);
    effectRoutine = StartCoroutine(SmoothDeath(1f, Color.black, duration));
}

1 reference
private IEnumerator SmoothDeath(float targetVignette, Color targetColor, float duration) {
    float startVignette = vignette.intensity.value;
    Color startColor = colorAdjustments.colorFilter.value;
    float time = 0f;

    while (time < duration) {
        time += Time.deltaTime;
        float t = time / duration;

        vignette.intensity.value = Mathf.Lerp(startVignette, targetVignette, t);
        colorAdjustments.colorFilter.value = Color.Lerp(startColor, targetColor, t);

        yield return null;
    }

    vignette.intensity.value = targetVignette;
    colorAdjustments.colorFilter.value = targetColor;
}
```

PlayerDeathEffects.cs Part 2

Το συγκεκριμένο script ενεργοποιείται από την πλευρά τους φαντάσματος, όταν καταφέρει και σκοτώσει τον παίκτη από ένα κυνήγι που θα εκτελέσει. Για αρχή το ορίζουμε ως Singleton, και στην Start() ανακτούμε τις πληροφορίες από το AudioSource. Στην συνέχεια ψάχνουμε την σκηνή για το αντικείμενο με όνομα Volume για να ανακτήσουμε τις τιμές και μεταβλητές του.
Στην ουσία το Volume αφορά εφέ και φίλτρα τα οποία προστίθενται στην κάμερα του παίκτη για να το κάνουν πιο σκοτεινό, φωτεινό και να προσθέσουν περισσότερα εφέ φωτισμού και σκιών. Στην δικιά μας περίπτωση θέλουμε να διαχειριστούμε την ρύθμιση που αφορά τα εφέ των χρωμάτων που αποδίδονται ως εξτρά στην κάμερα του παίκτη, καθώς και μια λειτουργία εφέ που ονομάζεται Vignette.

Εφόσον ληφθούν αυτές οι τιμές και το φάντασμα καλέσει την συνάρτηση PlayDeathEffects() θα προσομοιάσουμε το κλείσιμο των ματιών καθώς και μια μουσική που θα υποδηλώνει την λήξη του παιχνιδιού .

4.4.4.6 Map Selection

Εφόσον ο παίκτης είναι έτοιμος να παίξει θα χρειαστεί με την χρήση του πλήκτρου 'Space' να μεταφερθεί η κάμερα στο παρακάτω πίνακα, ο παίκτης θα έχει την δυνατότητα να επιλέξει την τοποθεσία που θα δοκιμάσει να εξιχνιάσει την ταυτότητα του φαντάσματος.

Map Selection

Με το που μεταφερθεί η κάμερα, ο παίκτης θα έχει στην διάθεση του την επιλογή οποιασδήποτε από της διαθέσιμες πίστες. Εφόσον πατήσει με το ποντίκι την επιθυμητή πίστα, μπορεί να ανοίξει την πόρτα από το Lobby για να μεταφερθεί στην επόμενη σκηνή με όνομα 'Map'.

4.4.4.7 Panels

Παρακάτω θα γίνει μία αναφορά σε όλα τα διαθέσιμα Panel του παίκτη.

4.5 Map Scene

Εφόσον γίνει η μεταφορά του παίκτη, θα βρεθούμε στο περιβάλλον που παρουσιάστηκε και στο Entry Menu, με την διαφορά ότι πλέον πρέπει να ερευνήσουμε τα παραφυσικά φαινόμενα του σπιτιού με τον εξοπλισμό που μας δίνεται και βρίσκεται πάνω στο τραπέζι.

Map Scene

Ο παίκτης πλέον καλείται να αντιμετωπίσει το φάντασμα και να το ταυτοποιήσει με την είσοδο του στο σπίτι. Εφόσον ο παίκτης μπει στο σπίτι θα περάσει από ένα αντικείμενο Trigger που θα δημιουργήσει το φάντασμα στην σκηνή , και θα του παρέχει όλα τα scripts για την λειτουργία του.

4.5.1 House

Για αρχή ας κάνουμε μια μικρή εισαγωγή στους χώρους του σπιτιού. Κάθε δωμάτιο του σπιτιού έχει ως Tag το "Room", και περιέχει ως Component ένα Box Collider σε λειτουργία Trigger η οποία έχει ως σκοπό να δηλώσει την περιοχή που είναι δυνατή το φάντασμα να δημιουργηθεί. Ο τρόπος με τον οποίο το φάντασμα δημιουργείτε και επιλέγει δωμάτιο είναι μέσω του script με όνομα GhostManager.cs

House – Hall

House – Living Room

House – Kitchen

House – Blue Bedroom

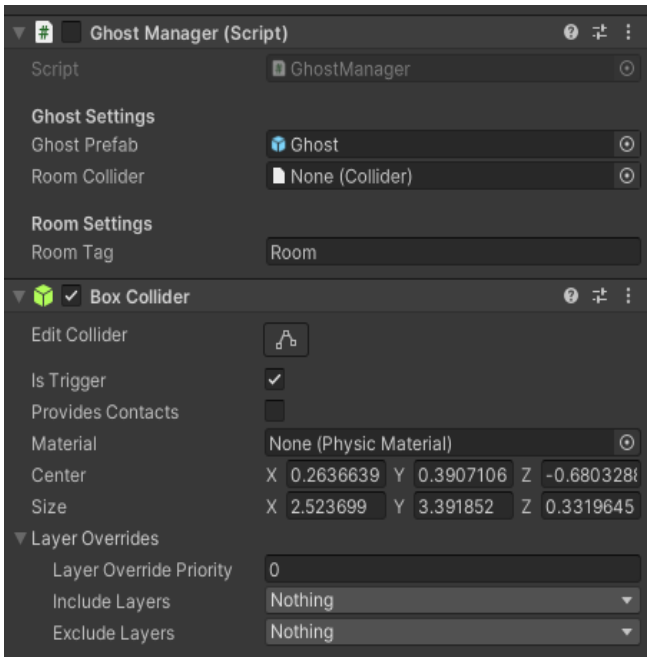
House – Bathroom

House – Orange Bedroom

4.5.1 Ghost Manager.cs & Ghost Enabler.cs

```
public class GhostManager : MonoBehaviour {
    [Header("Ghost Settings")]
    public GameObject ghostPrefab;
    public Collider roomCollider;

    [Header("Room Settings")]
    public string roomTag = "Room";

    Unity Message | 0 references
    void Start() {
        SpawnGhostInRandomRoom();
    }
```

```
public class GhostEnabler : MonoBehaviour
{
    private bool IsSpawned = false;
    [SerializeField]private MonoBehaviour GhostSpawner_Script;
    Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        if (!IsSpawned) {
            GhostSpawner_Script.enabled = true;
            IsSpawned = true;
        }
    }
}
```



Το GhostEnabler χρησιμοποιείται μόνο για τον έλεγχο του BoxCollider για να ελέγξει αν ο παίκτης πέρασε μέσα από το αντικείμενο, στην περίπτωση που περάσει τότε ενεργοποιούμε το GhostManager με σκοπό να δημιουργήσουμε το φάντασμα.

Στο script GhostManager δημιουργούμε στην αρχή ένα GameObject που θα αφορά το φάντασμα που θα δημιουργήσουμε, και δεύτερον ένα string RoomTag με σκοπό να δηλώσουμε από το Inspector και για τις δύο τιμές:

1) Ποιο Prefab θέλουμε να δημιουργήσουμε στην σκηνή μας
2) Με ποιο tag θέλουμε να ψάξουμε τα δωμάτια που μπορεί το φάντασμα να δημιουργηθεί

Στην συνέχεια καλούμε την συνάρτηση SpawnGhostInRandomRoom() που μας οδηγεί στην τυχαία επιλογή δωματίου για το φάντασμα αυτό.

```
public void SpawnGhostInRandomRoom() {

    GameObject[] rooms = GameObject.FindGameObjectsWithTag(roomTag);
    if (rooms.Length == 0) {
        Debug.LogWarning("No rooms found with tag: " + roomTag);
        return;
    }


    GameObject selectedRoom = rooms[Random.Range(0, rooms.Length)];
    roomCollider = selectedRoom.GetComponent<Collider>();

    if (roomCollider == null) {
        Debug.LogWarning("Selected room has no collider.");
        return;
    }


    selectedRoom.tag = "Ghost Room";

    Debug.Log("Ghost will be spawned in room: " + selectedRoom.name + " (tag changed to Ghost Room)");


    Vector3 spawnPos = GetRandomPointInRoom(roomCollider);
    if (spawnPos == Vector3.zero) {
        Debug.LogWarning("Failed to find a valid spawn point in room.");
        return;
    }


    GameObject ghost = Instantiate(ghostPrefab, spawnPos, Quaternion.identity);
    GhostAI ghostAI = ghost.GetComponent<GhostAI>();

    if (ghostAI != null) {
        ghostAI.SetCurrentRoom(roomCollider);
    } else {
        Debug.LogWarning("Spawned ghost does not have GhostAI script.");
    }
}
```

SpawnGhostInRandomRoom Function

Ξεκινώντας λαμβάνουμε σε έναν πίνακα από GameObjects όλα τα δωμάτιο με το ζητούμενο tag = Room. Μέσω της συνάρτησης Random.Range() επιλέγουμε μία τυχαία τιμή μέσα στον παραπάνω πίνακα και με την επιλογή αυτή ανακτούμε και τις τιμές του Collider του δωματίου αυτού.

Για να μπορούμε να διαχωρίσουμε το δωμάτιο του φαντάσματος με τα υπόλοιπα , αλλάζουμε το tag του σε Ghost Room, και στην συνέχεια οδηγούμαστε στην επιλογή μια τυχαίας θέσης μέσα στον χώρο που είναι δηλωμένος ως GhostRoom και περιορίζεται από τα όρια του Collider του δωματίου , επιστρέφοντας την θέση αυτή σε μια μεταβλητή τύπου Vector3. Αμέσως μετά δημιουργούμε το φάντασμα με την εντολή Instantiate() , που δέχεται ως παραμέτρους:

1) Το Ghost Prefab
2) Την θέση που βρέθηκε μέσω της συνάρτησης GetRandomPointInRoom()
3) Την φορά που θα κοιτάει το φάντασμα

Τέλος ανακτούμε τις τιμές από το script GhostAI για να μπορέσουμε να δηλώσουμε στο φάντασμα τον χώρο που θα έχει την δυνατότητα να κινείται.

```
1 reference
private Vector3 GetRandomPointInRoom(Collider room) {
    Vector3 center = room.bounds.center;
    Vector3 extents = room.bounds.extents;

    for (int i = 0; i < 10; i++) {
        Vector3 randomPos = center + new Vector3(
            Random.Range(-extents.x, extents.x),
            0,
            Random.Range(-extents.z, extents.z)
        );

        if (NavMesh.SamplePosition(randomPos, out NavMeshHit hit, 1f, NavMesh.AllAreas)) {
            return hit.position;
        }
    }

    return Vector3.zero;
}
```

GetRandomPointInRoom Function

Στην συνάρτηση αυτή εξετάζουμε και ανακτούμε τις τιμές του Collider που έχουμε στο επιλεγμένο δωμάτιο. Ο βρόγχος εκτελείται 10 φορές με σκοπό να βρεθεί ένα έγκυρο σημείο πάνω στο δάπεδο που να μπορεί το φάντασμα να θέσει ως περιοχή για να δημιουργηθεί , και ταυτόχρονα να είναι περιοχή του NavMesh του φαντάσματος. Εφόσον βρεθεί αυτή η περιοχή επιστρέφουμε την τιμή τύπου Vector3 που επιλέχθηκε .

4.5.2 GhostAI.cs

```
public class GhostAI : MonoBehaviour {
    [Header("Roaming Settings")]
    public float roamRadius = 3f;
    public float idleTimeAfterArrival = 2f;
    public float returnDelay = 5f;



    private NavMeshAgent agent;
    private Collider currentRoom;
    private bool isReturning = false;
    private bool isRoaming = false;

    Unity Message | 0 references
    void Start() {
        agent = GetComponent<NavMeshAgent>();
        agent.updateRotation = true;

        if (currentRoom != null) {
            StartRoaming();
        } else {
            Debug.LogWarning("No room assigned. Use SetCurrentRoom().");
        }
    }


    1 reference
    public void SetCurrentRoom(Collider room) {
        currentRoom = room;

        if (agent != null && !isRoaming) {
            StartRoaming();
        }
    }

    2 references
    void StartRoaming() {
        isRoaming = true;
        StartCoroutine(RoamLoop());
    }
```

GhostAI.cs

Το GhostAI είναι υπεύθυνο για τον τρόπο με τον οποίο το φάντασμα θα μετακινείται στους χώρους. Για αρχή του θέτουμε μερικές μεταβλητές που θα αφορούν την επιλογή της επόμενης θέσης που θα κινηθεί.
Χρησιμοποιούμε την roamRadius για να δηλώσουμε το μέχρι πόσο μακριά μπορεί να επιλέξει μια νέα θέση να κινηθεί, το idleTimeAfterArrival για να δηλώσουμε την καθυστέρηση μέχρι να επιλέξουμε την επόμενη του κίνηση, και το returnDelay για να δηλώσουμε ότι στην περίπτωση που το φάντασμα αποφασίσει να επιλέξει κινηθεί εκτός των ορίων του δωματίου, να επιστρέψει μετά από τον συγκεκριμένο χρόνο.

Στην συνέχεια στην Start() ανακτούμε τις μεταβλητές και τιμές από το Component NavMesh που έχει το φάντασμα και επιτρέπουμε τον αυτόματο χειρισμό της περιστροφής του φαντάσματος από το NavMesh, διαφορετικά θα έπρεπε να γίνει χειροκίνητα μέσω κώδικα. Εφόσον υλοποιηθούν όλα τα παραπάνω, καλούμε την συνάρτηση StartRoaming() που αποσκοπεί στην δήλωση της έναρξης της κίνησης του φαντάσματος με την Boolean τιμή isRoaming = true, και το κάλεσμα της συνάρτησης RoamLoop() που οδηγεί σε έναν ατέρμων βρόγχο.

```
1 reference
IEnumerator RoamLoop() {
    while (true) {

        // Skip if returning to room
        if (isReturning || currentRoom == null) {
            yield return null;
            continue;
        }

        Vector3 roamPoint = GetRandomPointInRoom(currentRoom);
        agent.SetDestination(roamPoint);

        // Wait until ghost reaches destination
        while (agent.pathPending || agent.remainingDistance > agent.stoppingDistance) {
            yield return null;
        }

        // Short pause before next move

        yield return new WaitForSeconds(idleTimeAfterArrival);

    }
}
```

RoamLoop Function

Όπως αναφέρθηκε η συνάρτηση αυτή εκτελεί έναν ατέρμων βρόγχο κατά τον οποίο, επιλέγουμε για το φάντασμα ένα νέο σημείο μέσα στο δωμάτιο για να μετακινηθεί και με την εντολή agent.SetDestination() και ως παράμετρο το σημείο , το φάντασμα εκτελεί την κίνηση.

Πριν γίνουν όμως αυτές οι κινήσεις γίνονται έλεγχοι όπως:

1) Να μην γίνει επόμενη κίνηση όσο το φάντασμα επιστρέφει από εξωτερικό δωμάτιο
2) Να μην γίνει επόμενη κίνηση όταν το φάντασμα δεν έχει ολοκληρώσει την κίνηση που του δόθηκε προηγουμένως

Για να ελέγξουμε όμως αν το φάντασμα βρέθηκε εκτός του δωματίου του, το πραγματοποιούμε με τον παρακάτω κώδικα:

```
Unity Message | 0 references
private void OnTriggerExit(Collider other) {
    if (other == currentRoom && !isReturning) {
        StartCoroutine(ReturnToRoom());
    }
}

1 reference
private IEnumerator ReturnToRoom() {
    isReturning = true;
    agent.ResetPath();

    yield return new WaitForSeconds(returnDelay);

    if (currentRoom != null) {
        Vector3 returnPoint = GetRandomPointInRoom(currentRoom);
        agent.SetDestination(returnPoint);

        // Wait until the agent reaches the return point
        while (agent.pathPending || agent.remainingDistance > agent.stoppingDistance) {
            yield return null;
        }
    }

    isReturning = false;
}
```

ReturnToRoom Function

Ελέγχουμε κάθε φορά που το φάντασμα εγκαταλείπει το δωμάτιο με την συνάρτηση OnTriggerExit() , δηλαδή κάθε φορά που το φάντασμα βγαίνει εκτός του Box Collider του δωματίου του, και στην περίπτωση που αληθεύει ξεκινάμε ένα Coroutine που πραγματοποιεί την επιστροφή του στο δωμάτιο.

Πιο συγκεκριμένα αφού διαπιστωθεί πως το φάντασμα βρίσκεται έξω από το δωμάτιο, θέτουμε την Boolean τιμή isReturning = true , σταματάμε την διαδρομή που είχε σκοπό να κάνει και του θέτουμε μια νέα αναγκαστική διαδρομή στο Ghost Room μέσα από την συνάρτηση GetRandomPointInRoom(). Η διαδικασία αυτή θα πραγματοποιείται μέχρι να επιστρέψει πίσω στο δωμάτιο όπου και θα αλλάξουμε την μεταβλητή isReturning σε false.

4.5.3 GhostInteractionManager.cs

Εκτός από την κίνηση του φαντάσματος θα χρειαστεί να διαχειριστούμε και τις ενέργειες που θα εκτελεί καθ' όλη την διάρκεια του παιχνιδιού. Για αυτό το λόγο θα χρησιμοποιήσουμε το script GhostInteractionManager.cs

```
public class GhostInteractionManager : MonoBehaviour {
    public float interactionInterval = 10f;
    public float interactionChance = 0.5f;

    private GhostLightInteraction lightInteraction;
    private GhostDoorInteraction doorInteraction;
    private BookInteraction bookInteraction;
    private GhostAppearEvidence ManifestInteraction;
    private SpecialObjectsEvents SpecialInteraction;
    private GhostEvent EventInteraction;

    [Header("Set to TRUE to only control light switches")]public bool MustSwitchLight = false;
    [Header("Set to TRUE to only Ghost Write")] public bool mustWrite = false;
    [Header("Set to TRUE to only Manifest")] public bool mustManifest = false;
    Unity Message | 0 references
    void Start() {

        lightInteraction = GetComponent<GhostLightInteraction>();
        doorInteraction = GetComponent<GhostDoorInteraction>();
        bookInteraction = GetComponent<BookInteraction>();
        ManifestInteraction = GetComponent<GhostAppearEvidence>();
        SpecialInteraction = GetComponent<SpecialObjectsEvents>();
        EventInteraction = GetComponent<GhostEvent>();
        Debug.Log("[GhostInteractionManager] Start called.");
        StartCoroutine(InteractionLoop());
    }
```

GhostInteractionManager.cs Part 1

Στο συγκεκριμένο script διαχειριζόμαστε σχεδόν όλες τις ενέργειες που μπορεί να εκτελέσει το φάντασμα μέσα στο παιχνίδι. Για αρχή ορίζουμε κάποιες μεταβλητές που αφορούν την καθυστέρηση από το να κάνει ένα νέο Interaction, και την πιθανότητα να κάνει ένα Interaction.

Στην συνέχεια κάνουμε αναφορά σε ορισμένα script που θα μπορεί το φάντασμα να καλέσει, και ανακτούμε τις τιμές από αυτά τα script μέσα στην μέθοδο Start().

Επιπλέον χρησιμοποιούμε στην αρχικοποίηση των μεταβλητών ορισμένες Boolean μεταβλητές, με σκοπό να αναγκάσουμε το φάντασμα να πραγματοποιήσει ένα συγκεκριμένο ή και παραπάνω interaction. Αυτή η λειτουργία χρησιμοποιήθηκε και στο Tutorial Room για την καλύτερη ανάδειξη των στοιχείων χωρίς το φάντασμα να πραγματοποιεί και άλλα Interactions.

Εφόσον πραγματοποιηθούν όλα τα παραπάνω καλούμε το Coroutine με όνομα InteractionLoop που αποτελεί επίσης έναν ατέρμων βρόγχο.

```
1 reference
IEnumerator InteractionLoop() {
    while (true) {

        yield return new WaitForSeconds(interactionInterval);

        if (Random.value <= interactionChance) {

            TryRandomInteraction();
        }
    }
}
```

InteractionLoop Function

Στο συγκεκριμένο βρόγχο ελέγχουμε με τυχαίες τιμές αν το φάντασμα θα καλέσει την TryRandomInteraction(). Διαφορετικά θα συνεχιστεί ο ίδιος βρόγχος μετά από την καθυστέρηση

που θέσαμε στην αρχή. Το Random.value μας δίνει μία τιμή float από το 0.0 - 1.0. Οπότε με την πιθανότητα που έχουμε θέσει ως 0.5, έχει πιθανότητα 50% να κάνει κάποιο Interaction.

```
void TryRandomInteraction() {

    int randomChoice;
    if (MustSwitchLight) {
        randomChoice = 0;
    }else if (mustWrite) {
        randomChoice = 2;

    }else if (mustManifest) {
        randomChoice = 3;
    } else {
        randomChoice = Random.Range(0, 5);
    }
    Debug.Log("Random is "+ randomChoice);
    switch (randomChoice) {
        case 0:
            lightInteraction?.TryInteract();
            break;
        case 1:
            doorInteraction?.TryInteract();
            break;
        case 2:
            if (!mustWrite) {

                if (GhostTypeSelector.Instance != null && GhostTypeSelector.Instance.HasGhostWriting) {
                    bookInteraction?.TryInteract();
                } else {

                }
            } else {

                bookInteraction?.TryInteract();
            }
            break;
        case 3:
            if (!mustManifest) {
                if (GhostTypeSelector.Instance != null && GhostTypeSelector.Instance.HasManifestation) {
                    ManifestInteraction?.TryManifest();
                } else {
                    Debug.Log("Ghost doesn't Manifest");
                }
            } else {
                ManifestInteraction?.TryManifest();
            }

            break;
        case 4:
            int select_event = Random.Range(0, 5);
            if (select_event <= 2) {
                SpecialInteraction?.TryInteract();
            } else {
                EventInteraction?.TryScare();
            }
            break;
    }
```

TryRandomInteraction Function

Η TryRandomInteraction αποτελείται από 5 διαφορετικά Interactions που έχει την δυνατότητα το φάντασμα να εκτελέσει. Ξεκινώντας προσπαθούμε να πάρουμε μία τιμή από 0 ως 5 στην τύχη με την χρήση της Random, εκτός αν έχουμε θέσει από το Inspector ότι θέλουμε κάποιο συγκεκριμένο Interaction. Στην περίπτωση που θέλουμε , αναγκάζουμε την επιλογή του randomChoice στο ζητούμενο αριθμό και προχωράμε στην switch – case, διαφορετικά επιλέγουμε έναν τυχαίο αριθμό στην κλίμακα που αναφέραμε.

Εφόσον προκύψει αυτός ο αριθμός, με την χρήση της switch – case ελέγχουμε για την κάθε πιθανή περίπτωση του randomChoice την ενέργεια που θα πραγματοποιηθεί.

Οι περιπτώσεις του randomChoice είναι οι εξής:

1) Αν το randomChoice = 0, τότε το φάντασμα θα δοκιμάσει να πειράξει κάποιον από τους κοντινούς διακόπτες φωτός
2) Αν το randomChoice = 1, τοτε το φάντασμα θα προσπαθήσει να επηρεάσει κάποια από τις κοντινές πόρτες.
3) Αν το randomChoice = 2, τότε το φάντασμα θα δοκιμάσει να γράψει στο Writing Book **εφόσον** διαθέτει αυτό το στοιχείο.
4) Αν το randomChoice = 3,τότε το φάντασμα θα δοκιμάσει να κάνει Manisfest , **εφόσον** διαθέτει το στοιχείο αυτό.
5) Αν το randomChoice = 4, τότε το φάντασμα θα επιχειρήσει να εκτελέσει ένα πιο ειδικό event ως προς τον παίκτη το οποίο χωρίζεται σε δύο περιπτώσεις. Η πρώτη περίπτωση αφορά την δημιουργία κάποιου ιδιαίτερου ήχου μέσα στο σπίτι , και η δεύτερη περίπτωση θα αφορά ένα event με σκοπό να εμφανιστεί στον παίκτη, να τον τρομάξει με διάφορες ενέργειες και να του μειώσει το διαθέσιμο sanity.

Ξεκινώντας από την πρώτη κατηγορία θα αναφερθούμε σε καθένα από τα script.

4.5.3.1 BookInteraction.TryInteract

Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για να μπορέσει το φάντασμα να γράψει στο βιβλίο, δεδομένου ότι διαθέτει το στοιχείο αυτό και είναι εντός της περιοχής που θέτουμε.

```
public class BookInteraction : MonoBehaviour {
    [Header("Text Fade Settings")]
    public TMP_Text[] textMeshes;
    public float fadeDuration = 4f;

    [Header("Writing Interaction Radius")]
    public float interactionRadius = 6f;

    public bool includeTriggers = true;

    [Header("Names & Tags")]
    public string bookTag = "Book";
    public string emfPulseName = "EMF_Pulse";

    private bool Used = false;
    private bool isFading = false;
    private float fadeTimer = 0f;
    public GameObject emf_Pulse;
```

BookInteraction.cs

Για αρχή δηλώνουμε ορισμένες μεταβλητές οι οποίες αποτελούνται από έναν πίνακα τύπου TMP_Text που αφορά το κείμενο του βιβλίου , το fadeDuration που αποσκοπεί στην διάρκεια που θα χρειαστεί να εμφανιστεί το κείμενο από όταν το φάντασμα κάνει Interact με αυτό (χρησιμοποιείται για λόγους αισθητικής, αντί να εμφανιζόταν το κείμενο απευθείας), και ορισμένες bool μεταβλητές που μας βοηθάνε σε διάφορους ελέγχους για τις συνθήκες.

```
public void TryInteract() {

    var colliders = includeTriggers
        ? Physics.OverlapSphere(transform.position, interactionRadius, ~0, QueryTriggerInteraction.Collide)
        : Physics.OverlapSphere(transform.position, interactionRadius);

    if (colliders == null || colliders.Length == 0) {
        Debug.Log("[BookInteraction] No colliders found in range.");
        return;
    }

    foreach (var col in colliders) {

        Transform t = col.transform;
        Transform bookRoot = null;
        while (t != null) {
            if (t.CompareTag(bookTag)) { bookRoot = t; break; }
            t = t.parent;
        }

        if (bookRoot == null) continue;


        if (isFading || Used) return;
        if (Random.value > 0.3f) return;


        var allTransforms = bookRoot.GetComponentsInChildren<Transform>(true);
        var emfTr = allTransforms.FirstOrDefault(x => x.name == emfPulseName);
        if (emfTr == null) {
            Debug.LogError($"[BookInteraction] '{emfPulseName}' not found under '{bookRoot.name}'.");
            return;
        }

        emf_Pulse = emfTr.gameObject;


        textMeshes = bookRoot.GetComponentsInChildren<TMP_Text>(includeInactive: true);
        if (textMeshes == null || textMeshes.Length == 0) {
            Debug.LogError($"[BookInteraction] No TMP_Text found under '{bookRoot.name}'.");
            return;
        }


        foreach (var tm in textMeshes) {
            if (tm == null) continue;
            tm.enabled = true;
            var c = tm.color; c.a = 0f; tm.color = c;
            tm.alpha = 0f;
            tm.ForceMeshUpdate();
        }

        isFading = true;
        fadeTimer = 0f;

        if (emf_Pulse != null) emf_Pulse.SetActive(true);

        Debug.Log($"[BookInteraction] Triggered on '{bookRoot.name}'. Texts: {textMeshes.Length}");
        return;
    }
}
```

TryInteract Function

Η μεταβλητή colliders περιέχει έναν πίνακα που αποθηκεύει όλα τα Collider που βρίσκονται μέσα σε μια σφαίρα που δημιουργείτε με γύρω από την θέση του φαντάσματος (transform.position), με συγκεκριμένη εμβέλεια (interactionRadius) , θα δέχεται όλα τα Layers το οποίο δηλώνεται με το ~0, και τέλος επιτρέπει τα collider με την λειτουργία Is Trigger να συμπεριληφθούν στον πίνακα. Διαφορετικά αν δεν επιστρέψει τέτοιου είδους αντικείμενα θα δημιουργήσει μία όμοια σφαίρα η οποία δεν θα εμπεριέχει τα αντικείμενα με Collider στην λειτουργία isTrigger.

Εφόσον ολοκληρωθεί αυτή η διαδικασία, για κάθε ένα από αυτά παίρνουμε το Component Transform και ελέγχουμε αν ανήκει σε αντικείμενο με τα tag Book, το οποίο το θέσαμε στην αρχή. Κάνουμε αυτόν τον έλεγχο πηγαίνοντας στο Hierarchy μέχρι το Parent Object, με σκοπό να βρούμε αντικείμενο με αυτό το tag.

Στην συνέχεια ελέγχουμε με το Random.value την πιθανότητα να μπορέσει το φάντασμα να γράψει στο βιβλίο με την διαφορά ότι με την συνθήκη **if(Random.value > 0,3f) return;** , ελέγχει για πιθανότητα 70%, αφού αν είναι αληθής η συνθήκη αυτή θα κάνει **return** . Όμοια κάνουμε και έλεγχο για το βιβλίο έχει χρησιμοποιηθεί ή χρησιμοποιείται εκείνη την στιγμή.

Στην συνέχεια για όλα τα Transform που υπάρχουν στο βιβλίο αναζητούμε το GameObject emfPulse
Με το επιθυμητό όνομα, αναζητούμε επίσης στο βιβλίο τα κείμενα για να ανακτήσουμε τις τιμές τους, και για κάθε ένα από τα κείμενα θέτω την τιμή που αφορά το transparency = 0.

Με την χρήση του ForceMeshUpdate() στην ουσία βοηθάμε τα κείμενα να εμφανιστούν σωστά και ομοιόμορφα.

Τέλος ορίζουμε ότι το βιβλίο θα αρχίσει να εμφανίζει τα γράμματα, θέτοντας και το fadeTimer σαν χρονόμετρο από το 0, και ενεργοποιούμε το emf_Pulse.
Μετά από ολόκληρη την παραπάνω διαδικασία, η Update() φροντίζει να γίνει η εναλλαγή του transparency από 0 σε 255.

```
Unity Message | 0 references
private void Update() {
    if (!isFading) return;

    fadeTimer += Time.deltaTime;
    float a = Mathf.Clamp01(fadeTimer / fadeDuration);

    // Apply alpha for TMP color and vertex alpha
    foreach (var tm in textMeshes) {
        if (tm == null) continue;
        var c = tm.color; c.a = a; tm.color = c;
        tm.alpha = a;
    }

    if (a >= 1f) {
        Used = true;
        isFading = false;
        Debug.Log("[BookInteraction] Text fully revealed.");
    }
}
```

BookInteraction – Update()

Σε κάθε στάδιο του παιχνιδιού η Update φροντίζει να ελέγχει για την μεταβλητή isFading. Όταν η συνθήκη γίνει αληθής θα εμφανίζει τα κείμενα με ομαλό τρόπο μέσα στην διάρκεια που έχουμε θέσει. Αφού το transparency γίνει μέγιστο ορίζουμε πως το συγκεκριμένο βιβλίο έχει χρησιμοποιηθεί.

4.5.3.2 ManifestInteraction.TryInteract

```
Unity Script (1 asset reference) | 2 references
public class GhostAppearEvidence : MonoBehaviour
{
    public GameObject GhostBody;
    public float flashDuration = 0.3f;



    2 references
    public void TryManifest() {
        // 10% chance
        if (Random.value <= 0.1f) {
            StartCoroutine(Flash());
        }
    }

    1 reference
    private IEnumerator Flash() {
        if (GhostBody != null) {
            GhostBody.SetActive(true);
            yield return new WaitForSeconds(flashDuration);
            GhostBody.SetActive(false);
        }
    }
}
```

GhostAppearEvidence.cs

Στο συγκεκριμένο script, το οποίο καλείται από το GhostInteractionManager.cs αρχικοποιούμε ένα GameObject που θα αναφέρεται στο σώμα του φαντάσματος, το οποίο θα το αρχικοποιήσουμε από το Inspector.

Με την κλήση της TryManifest() το φάντασμα έχει 10% πιθανότητα να ξεκινήσει ένα Coroutine στο οποίο θα εμφανίζει το μοντέλο του για ένα μικρό χρονικό διάστημα και στην συνέχεια θα εξαφανίζεται πάλι.

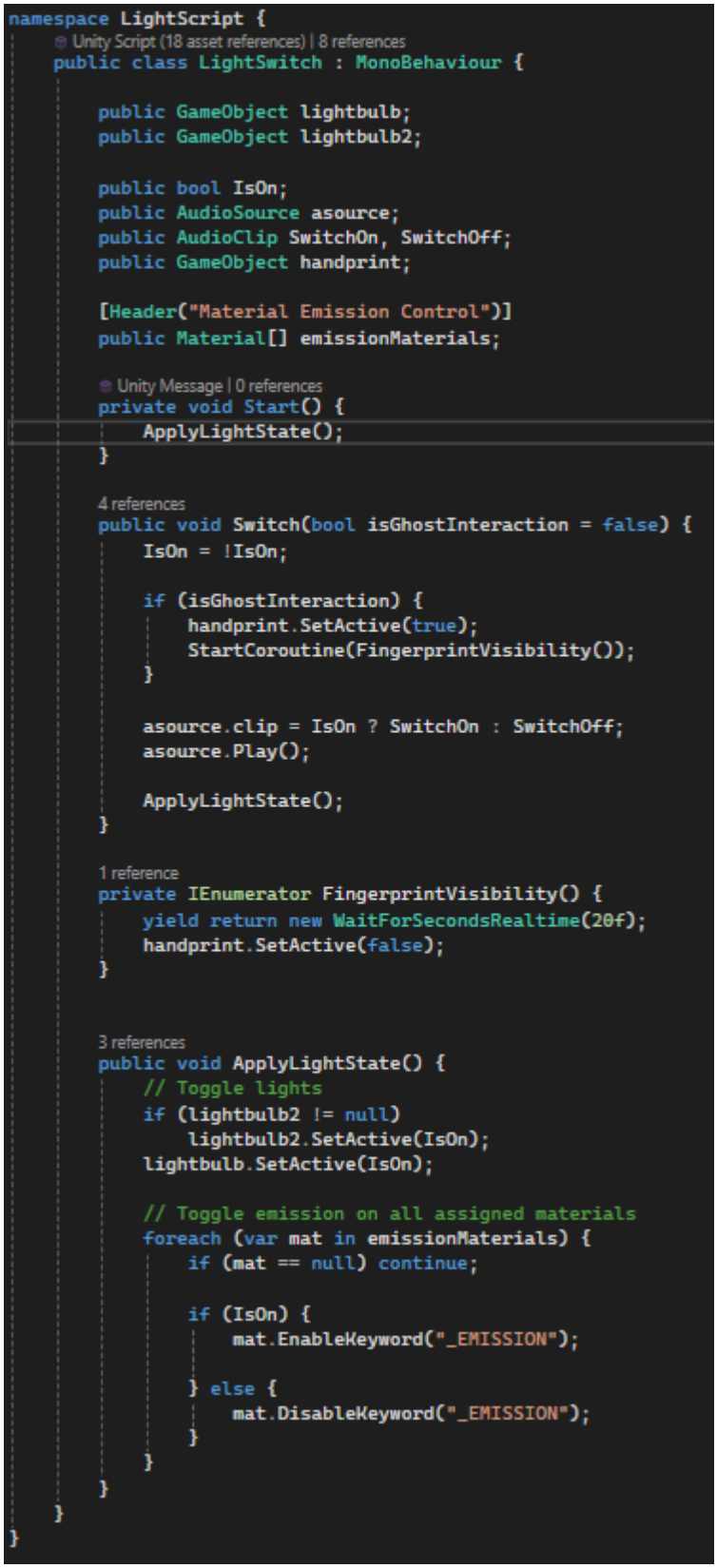
4.5.3.3 LightInteraction.TryInteract

```
Unity Script (1 asset reference) | 2 references
public class GhostLightInteraction : MonoBehaviour {
    [Header("Light Switch (ON) Interaction Chance")] public float intchanceOn = 0.45f;
    [Header("Light Switch (OFF) Interaction Chance")] public float intchanceOff = 0.02f;
    1 reference
    public void TryInteract() {
        Collider[] lights = Physics.OverlapSphere(transform.position, 5f);

        foreach (Collider col in lights) {
            var light = col.GetComponent<LightSwitch>();
            if (light != null) {
                if (Random.value <= intchanceOn && light.IsOn) {
                    light.Switch(isGhostInteraction: true);
                    Debug.Log("Ghost turned OFF the light.");
                } else if (Random.value <= intchanceOff && !light.IsOn) {
                    light.Switch(isGhostInteraction: true);
                    Debug.Log("Ghost turned ON the light.");
                }
                var Emf = light.transform.Find("EMF_Pulse");
                Emf.gameObject.SetActive(true);
            }
        }
    }
}
```

GhostLightInteraction.cs

Το συγκεκριμένο script δημιουργεί ένα πίνακα που αποθηκεύει όλα τα Colliders εντός εμβέλειας 5unit, και για κάθε ένα από αυτά ελέγχουμε αν περιέχει το script LightSwitch. Σε περίπτωση που το έχει ως Component, για το αντικείμενο αυτό ελέγχουμε αν το φως είναι αναμμένο , και εφόσον επιτύχει και η πιθανότητα με την βοήθεια του Random.value , το φάντασμα θα σβήσει το φως, διαφορετικά στην αντίθετη συνθήκη θα ανάψει το, πράγμα λιγότερο πιθανό λόγω της δοσμένης πιθανότητας.

Το script για το LightSwitch που αναφέραμε είναι το ακόλουθο:



```
namespace LightScript {
    Unity Script (18 asset references) | 8 references
    public class LightSwitch : MonoBehaviour {

        public GameObject lightbulb;
        public GameObject lightbulb2;

        public bool IsOn;
        public AudioSource asource;
        public AudioClip SwitchOn, SwitchOff;
        public GameObject handprint;

        [Header("Material Emission Control")]
        public Material[] emissionMaterials;

        Unity Message | 0 references
        private void Start() {
            ApplyLightState();
        }

        4 references
        public void Switch(bool isGhostInteraction = false) {
            IsOn = !IsOn;

            if (isGhostInteraction) {
                handprint.SetActive(true);
                StartCoroutine(FingerprintVisibility());
            }

            asource.clip = IsOn ? SwitchOn : SwitchOff;
            asource.Play();

            ApplyLightState();
        }

        1 reference
        private IEnumerator FingerprintVisibility() {
            yield return new WaitForSecondsRealtime(20f);
            handprint.SetActive(false);
        }


        3 references
        public void ApplyLightState() {
            // Toggle lights
            if (lightbulb2 != null)
                lightbulb2.SetActive(IsOn);
            lightbulb.SetActive(IsOn);

            // Toggle emission on all assigned materials
            foreach (var mat in emissionMaterials) {
                if (mat == null) continue;

                if (IsOn) {
                    mat.EnableKeyword("_EMISSION");

                } else {
                    mat.DisableKeyword("_EMISSION");
                }
            }
        }
    }
}
```

LightSwitch.cs

Για αρχή , θέτουμε ορισμένα GameObjects τα οποία θα τα αρχικοποιήσουμε από το Inspector.
Ο κάθε διακόπτης μπορεί να επηρεάζει μέχρι δύο λάμπες με βάση αυτό το script.
Στην Start() καλούμε την συνάρτηση ApplyState() με σκοπό να θέσουμε τα φώτα ως κλειστά.

Στην συνάρτηση Switch() δεχόμαστε ως παράμετρο μία Boolean τιμή η οποία αν είναι true θα υποδηλώνει ότι το φάντασμα αλληλεπίδρασε με τον διακόπτη και όχι ο παίκτης. Έτσι αν τα φάντασμα διαθέτει ως στοιχείο το Ultraviolet , θα εμφανίσει και αποτυπώματα πάνω στον διακόπτη. Ανάλογα την κατάσταση της μεταβλητής IsOn μέσω του script ακούγεται και ο κατάλληλος ήχος. Και τέλος στο ApplyLightState() ελέγχουμε την κατάσταση της μεταβλητής IsOn και αναλόγως σβήνουμε ή ανάβουμε τα φώτα καθώς και απενεργοποιούμε ή ενεργοποιούμε στο Material Component την λειτουργία Emission για να φαίνεται πως η λάμπα έχει έντονο φωτισμό.

4.5.3.4 DoorInteraction.TryInteract

```
Unity Script (1 asset reference) | 2 references
public class GhostDoorInteraction : MonoBehaviour {
    1 reference
    public void TryInteract() {
        Collider[] doors = Physics.OverlapSphere(transform.position, 5f);

        foreach (Collider col in doors) {
            var door = col.GetComponent<DoorScript.Door>();
            if (door != null && Random.value <= 0.5f) {
                door.OpenDoor(isGhostInteraction: true);
                Debug.Log("Ghost opened a door.");

                var Emf = door.transform.Find("EMF_Pulse");
                Emf.gameObject.SetActive(true);
            }
        }
    }
}
```

GhostDoorInteraction.cs

Όμοια και με το GhostLightInteraction.cs, στην αρχή μέσα σε μία εμβέλεια 5 unit , αποθηκεύουμε σε έναν πίνακα όλα τα Colliders, και εφόσον κάποιο από αυτά τα Colliders διαθέτει πάνω του το Door.cs , με βάση την πιθανότητα (η οποία είναι 50%) το φάντασμα θα επιχειρήσει να κάνει interact με την πόρτα.

Το Door.cs ακολουθεί παρακάτω:

```
namespace DoorScript {
    [RequireComponent(typeof(AudioSource))]
    Unity Script (108 asset references) | 7 references
    public class Door : MonoBehaviour {
        public bool open;
        public bool interactedByGhost = false;
        public float smooth = 1.0f;
        float DoorOpenAngle = -90.0f;
        float DoorCloseAngle = 0.0f;
        public AudioSource asource;
        public AudioClip openDoor, closeDoor;
        public GameObject handprint1, handprint2;

        Unity Message | 0 references
        void Start() {
            asource = GetComponent<AudioSource>();
        }

        Unity Message | 0 references
        void Update() {
            if (open) {
                var target = Quaternion.Euler(0, DoorOpenAngle, 0);
                transform.localRotation = Quaternion.Slerp(transform.localRotation, target, Time.deltaTime * 5 * smooth);
            } else {
                var target1 = Quaternion.Euler(0, DoorCloseAngle, 0);
                transform.localRotation = Quaternion.Slerp(transform.localRotation, target1, Time.deltaTime * 5 * smooth);
            }
        }

        4 references
        public void OpenDoor(bool isGhostInteraction = false) {
            open = !open;


            if (isGhostInteraction) {
                if (GhostTypeSelector.Instance != null && GhostTypeSelector.Instance.HasUV) {
                    handprint1.SetActive(true);
                    handprint2.SetActive(true);
                    StartCoroutine(HandprintVisibility());
                } else {
                    Debug.Log("There is no fingees");
                }

            }

            asource.clip = open ? openDoor : closeDoor;
            asource.Play();
        }

        1 reference
        private IEnumerator HandprintVisibility() {

            yield return new WaitForSecondsRealtime(20f);
            handprint1.SetActive(false);
            handprint2.SetActive(false);
        }

    }
}
```

Door.cs

Το παραπάνω script έχει αρκετές ομοιότητες με το LightSwitch.cs . Για αρχή θέτουμε μερικές Boolean μεταβλητές για να ελέγχουμε την κατάσταση της πόρτας. Ακόμα χρησιμοποιούμε και μία μεταβλητή η οποία ελέγχει αν το φάντασμα έκανε Interact με την πόρτα ή όχι. Τέλος ορίζουμε τα όρια για το άνοιγμα της πόρτας καθώς και την προσθήκη των ήχων για το άνοιγμα- κλείσιμο της πόρτας και τα Handprints.

Στην Update() ελέγχουμε με βάση την μεταβλητή open την γωνία που θα θέσουμε την πόρτα και να χρησιμοποιήσουμε το DoorOpenAngle ή DoorCloseAngle.

Η συνάρτηση OpenDoor , επίσης παίρνει σαν παράμετρο το isGhostInteraction με σκοπό να ελέγξει αν εκτελέστηκε η συνάρτηση από το GhostInteractionManager.cs. Σε περίπτωση που έκανε interact το φάντασμα με την πόρτα εμφανίζουμε τα Handprints τα οποία διαρκούν για 20 δευτερόλεπτα όπου μετά απενεργοποιούνται.

4.5.3.5 SpecialInteraction.TryInteract

```
public class SpecialObjectsEvents : MonoBehaviour {
    private AudioSource phoneSrc;
    private AudioSource clockSrc;
    private TV_Remote_Script mytv;

    private GameObject phoneObj;
    private GameObject clockObj;
    private GameObject remoteObj;

    Unity Message | 0 references
    void Start() {

        phoneObj = GameObject.FindGameObjectWithTag("Device");
        clockObj = GameObject.FindGameObjectWithTag("Clock");
        remoteObj = GameObject.FindGameObjectWithTag("Remote");

        if (phoneObj != null) phoneSrc = phoneObj.GetComponent<AudioSource>();
        if (clockObj != null) clockSrc = clockObj.GetComponent<AudioSource>();
        if (remoteObj != null) mytv = remoteObj.GetComponent<TV_Remote_Script>();


    }

    1 reference
    public void TryInteract() {
        float r = Random.value;

        if (r < 0.1f && phoneSrc != null) {
            phoneSrc.Play();
            var emf = phoneObj.transform.Find("EMF_Pulse");
            if (emf != null) emf.gameObject.SetActive(true);
        } else if (r < 0.2f && clockSrc != null) {
            clockSrc.Play();
            var emf = clockObj.transform.Find("EMF_Pulse");
            if (emf != null) emf.gameObject.SetActive(true);
        } else if (r < 0.3f && mytv != null) {

            mytv.ToggleTV();

            var emf = remoteObj.transform.Find("EMF_Pulse");
            if (emf != null) emf.gameObject.SetActive(true);
        }
    }
}
```

SpecialObjectsEvents.cs

Το παραπάνω script αφορά events τα οποία είναι μεν για να φοβίσουν τον παίκτη, αλλά δεν επηρεάζουν καθόλου την τιμή του Sanity του παίκτη σε σχέση με το Event που θα παρουσιάσουμε στην συνέχεια.

Στο συγκεκριμένο script ανακτούμε σε μεταβλητές με βάση το tag τα αντικείμενα που θέλουμε το φάντασμα να επηρεάσει, και ανακτούμε τις τιμές από τα AudioSources και το TV_Remote_Script

Στην συνέχεια με την χρήση πιθανοτήτων εξετάζουμε και επιλέγουμε ποιο από τα τρία αυτά αντικείμενα θα χρησιμοποιήσει το φάντασμα ή αν δεν επιτύχει να επηρεάσει κάποιο. Σε κάθε ένα από αυτά τα αντικείμενα ενεργοποιούμε και το emf που μας δίνει την δυνατότητα να το ελέγξουμε για το στοιχείο EMF Level 5.

Το script ToggleTV() φαίνεται παρακάτω:

```
public class TV_Remote_Script : MonoBehaviour
{
    public GameObject tvpanel;
    public AudioSource turnOn;
    public AudioSource turnOff;
    public AudioSource TVSound;
    public bool playerControlEnabled = true;
    public bool on;
    public bool off;

    Unity Message | 0 references
    void Start() {
        off = true;
        tvpanel.SetActive(false);
    }

    Unity Message | 0 references
    void Update() {
        if (!playerControlEnabled) return;

        if (off && Input.GetButtonDown("F")) {
            ToggleTV();
        } else if (on && Input.GetButtonDown("F")) {
            ToggleTV();
        }
    }

    3 references
    public void ToggleTV() {
        if (off) {
            tvpanel.SetActive(true);
            turnOn.Play();
            off = false;
            on = true;
            TVSound.Play();
        } else {
            turnOff.Play();
            tvpanel.SetActive(false);
            off = true;
            on = false;
            TVSound.Stop();
        }
    }
```

TV_Remote_Script.cs

Για αρχή το συγκεκριμένο script χρησιμοποιεί μεταβλητές bool για τον έλεγχο της κατάστασης της τηλεόρασης, καθώς και AudioSources για τον ήχο που θα κάνει το χειριστήριο της τηλεόρασης αλλά και η ίδια η τηλεόραση όταν ανάβει. Επιπλέον χρησιμοποιούμε και ένα GameObject tvpanel που μέχρι ο παίκτης ή το φάντασμα να ανάψουν την τηλεόραση είναι απενεργοποιημένο και με την χρήση της συνάρτησης το ενεργοποιούμε και το απενεργοποιούμε.

TV Turned On

4.5.3.6 GhostEvent.cs

```
public class GhostEvent : MonoBehaviour {
    public Transform player;
    public float scareChance = 0.1f;

    public AudioSource asource;
    public AudioClip event_clip;
    public GameObject ghostmodel;
    public AnimationClip anim;
    private Volume globalVolume;
    public bool canScare = true;
    private NavMeshAgent agent;
    private Animator animator;
    private bool isScaring = false;

    private ColorAdjustments colorAdjustments;
    private Color originalColor;

    Unity Message | 0 references
    void Awake() {
        agent = GetComponent<NavMeshAgent>();
        animator = GetComponentInChildren<Animator>();

        GameObject playerObject = GameObject.FindGameObjectWithTag("Player");

        player = playerObject.transform;


        globalVolume = GameObject.Find("Volume").GetComponent<Volume>();

        if (globalVolume != null) {

            globalVolume.profile = Instantiate(globalVolume.profile);


            if (!globalVolume.profile.TryGet(out colorAdjustments)) {

                colorAdjustments = globalVolume.profile.Add<ColorAdjustments>(true);
                colorAdjustments.active = true;
            }


            colorAdjustments.colorFilter.overrideState = true;


            originalColor = colorAdjustments.colorFilter.value;
        } else {
            Debug.LogError("No Volume found on Player object or its children!");
        }

    }
```

GhostEvent.cs Part 1

Στο συγκεκριμένο script σε σχέση με το SpecialObjectsEvents.cs , το φάντασμα το ίδιο εμφανίζεται στον παίκτη επηρεάζοντας την εικόνα του και μειώνοντας το Sanity του παίκτη.

Για αρχή αρχικοποιούμε μεταβλητές όπως η τοποθεσία του παίκτη, η πιθανότητα το φάντασμα να τρομάξει τον παίκτη, καθώς και ήχους και το GameObject που αφορά τα εφέ της κάμερας για να πραγματοποιηθούν οι κατάλληλες ενέργειες ως προς τον παίκτη. Ακόμα λαμβάνουμε το NavMesh του φαντάσματος και χρησιμοποιούμε Boolean μεταβλητές για να ελέγχουμε τα γεγονότα.

Στην Awake() κάνουμε ανάκτηση των τιμών από το NavMesh, το Animator του φαντάσματος, τον ίδιο τον παίκτη και την τοποθεσία του, και τέλος κάνουμε αναφορά στο αντικείμενο globalVolume

το οποίο βρίσκεται ενεργό στην σκήνη και το αναζητούμε με την εύρεση του ονόματος του για να λάβουμε τιμές του καθώς και στοιχεία του που χρειαζόμαστε όπως το Color Adjustments.

```
public void TryScare() {
    if (!canScare || isScaring) return;


    Debug.Log("AN FANEI PAME KALA");
    if (Random.value < scareChance ) {

        StartCoroutine(DoScareEvent());
    }
}

1 reference
IEnumerator DoScareEvent() {
    isScaring = true;
    Debug.Log("SCARE");
    agent.isStopped = true;
    animator.SetBool("isWalking", false);
    agent.updateRotation = false;
    SanityManager.Instance.ReduceSanity(Random.Range(5, 12));

    Vector3 direction = (player.position - transform.position).normalized;
    direction.y = 0f;
    transform.rotation = Quaternion.LookRotation(direction);


    animator.SetTrigger("Event");
    ghostmodel.SetActive(true);
    SetColorFilter(Color.red);
    asource.PlayOneShot(event_clip);

    yield return new WaitForSecondsRealtime(anim.length);


    TurnOffAllNormalLights();
    ghostmodel.SetActive(false);
    SetColorFilter(originalColor);
    asource.Stop();

    agent.updateRotation = true;
    agent.isStopped = false;
    animator.SetBool("isWalking", true);
    isScaring = false;
}

2 references
void SetColorFilter(Color color) {
    if (colorAdjustments != null) {
        colorAdjustments.colorFilter.value = color;
    }
}

1 reference
void TurnOffAllNormalLights() {
    LightSwitch[] myLights = GameObject.FindObjectsOfType<LightSwitch>();

    foreach (LightSwitch light in myLights) {
        light.IsOn = false;
        light.ApplyLightState();
    }


}
```

GhostEvent.cs Part 2

Μέσω το GhostInteractionManager.cs το φάντασμα θα καλέσει την συνάρτηση TryScare(). Η συγκεκριμένη συνάρτηση στην αρχή ελέγχει αν το φάντασμα εκτελεί ήδη την ενέργεια αυτή ή αν

το επιτρέπεται να την εκτελέσει. Εφόσον έχει την δυνατότητα αυτή, ξεκινάει η διαδικασία για να τρομάξει τον παίκτη , κατά την οποία θέτουμε την isScaring = true για να αποτρέψουμε το φάντασμα από το να εκτελέσει ξανά event και θέτουμε στο Animator ότι θα σταματήσει το Animation που αφορά το περπάτημα με την εντολή : **animator.SetBool("isWalking", false);**
Αποτρέπουμε στο NavMesh να ελέγχει το rotation του φαντάσματος καθώς σκοπεύουμε να το στρέψουμε να κοιτάει τον παίκτη κατά την διάρκεια του event χειροκίνητα.
Στην συνέχεια μειώνουμε το sanity του παίκτη με μία τυχαία τιμή μεταξύ 5- 12 πόντους sanity, Και αλλάζουμε την τιμή του πεδίου Event στον Animator , για να μπορέσει να λειτουργήσει ένα συγκεκριμένο Animation για την διάρκεια του Event. Τέλος εμφανίζουμε το μοντέλο του φαντάσματος, αλλάζουμε μέσω της συνάρτησης SetColorFilter() το χρώμα που αποδίδει το Global Volume/Color Adjustments στην κάμερα του παίκτη, και παίζουμε έναν ήχο για να υποδηλώσουμε ότι έγινε event.

Εφόσον πραγματοποιηθεί το event αυτό αντιστρέφουμε όλες τις παραπάνω διαδικασίες για να επιστρέψει το παιχνίδι στην κανονική του μορφή χωρίς τα εφέ και χωρίς το φάντασμα να είναι φανερό, επιτρέποντας του ξανά να μετακινείτε στον χώρο και να εκτελεί διάφορα events.

Η συνάρτηση SetColorFilter() δέχεται σαν παράμετρο το χρώμα που επιθυμούμε να αλλάξουμε το χρώμα του Global Volume/Color Adjustments .

Η συνάρτηση TurnOffAllNormalLights() λαμβάνει όλα τα αντικείμενα που διαθέτουν το script LightSwitch (στην ουσία όλους τους διακόπτες) και θέτει τις τιμές IsOn = false και καλεί την συνάρτηση ApplyLightState() για ενημέρωση της κατάστασης για κάθε διακόπτη και φως.

4.5.3 GhostThrowItems.cs

```
public class GhostThrowItems : MonoBehaviour {
    [Header("Throw Settings")]
    [Range(0f, 1f)]
    public float throwChance = 0.25f;
    public float throwForce = 15f;
    public float minThrowInterval = 2f;
    public float maxThrowInterval = 5f;

    private List<Rigidbody> nearbyItems = new List<Rigidbody>();

    Unity Message | 0 references
    private void Awake() {

        SphereCollider collider = GetComponent<SphereCollider>();
        if (collider == null || !collider.isTrigger) {
            Debug.LogError("This script requires a SphereCollider set to 'Is Trigger'.");
        }
    }

    Unity Message | 0 references
    private void Start() {

        StartCoroutine(ThrowItemRoutine());
    }

    1 reference
    private IEnumerator ThrowItemRoutine() {
        while (true) {

            float randomInterval = Random.Range(minThrowInterval, maxThrowInterval);
            yield return new WaitForSeconds(randomInterval);


            AttemptThrowItem();
        }
    }

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {

        Rigidbody itemRigidbody = other.GetComponent<Rigidbody>();
        if (itemRigidbody != null) {

            nearbyItems.Add(itemRigidbody);
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other) {

        Rigidbody itemRigidbody = other.GetComponent<Rigidbody>();
        if (itemRigidbody != null) {

            nearbyItems.Remove(itemRigidbody);
        }
    }
```

GhostThrowItems.cs

Το παραπάνω script είναι υπεύθυνο για την λειτουργία "Ρίψης Αντικειμένων" από το φάντασμα. Στην αρχή δηλώνουμε μεταβλητές που ορίζουν την πιθανότητα να πετάξει κάτι το φάντασμα, την δύναμη που θα χρησιμοποιήσει για την ρίψη, καθώς και την μικρότερη και μεγαλύτερη καθυστέρηση μεταξύ ρίψεων. Τέλος δηλώνουμε μία λίστα αντικειμένων η οποία θα αποτελείται από Rigidbody και θα ενημερώνεται τακτικά μέσω των OnTriggerEnter και OnTriggerExit.

Στην Start() καλούμε την συνάρτηση ThrowItemRoutine που αποτελεί Coroutine και συγκεκριμένα έναν ατέρμων βρόγχο με σκοπό να γίνονται οι προσπάθειες ρίψης καθ' όλη την διάρκεια του παιχνιδιού.

Στον OnTriggerEnter ελέγχουμε για κάθε αντικείμενο που εισέρχεται μέσα στο SphereCollider του φαντάσματος με την λειτουργία Is Trigger, αν περιέχει Rigidbody και το προσθέτουμε στην λίστα, αντιθέτως στην OnTriggerExit αν ένα αντικείμενο που βρισκόταν προηγουμένως μέσα στον εμβέλεια του SphereCollider πλέον βρεθεί εκτός αυτού, θα ενημερώσουμε την λίστα να το αφαιρέσει από αυτήν.

```
private void AttemptThrowItem() {

    if (nearbyItems.Count == 0) {

        return;
    }


    if (Random.value <= throwChance) {

        Rigidbody itemToThrow = nearbyItems[Random.Range(0, nearbyItems.Count)];


        Vector3 throwDirection = (itemToThrow.position - transform.position).normalized;


        itemToThrow.AddForce(throwDirection * throwForce, ForceMode.Impulse);

        Transform emfPl = itemToThrow.transform.Find("EMF_Pulse");
        if (emfPl != null) {
            emfPl.gameObject.SetActive(true);
        }

        nearbyItems.Remove(itemToThrow);


    }
```

AttemptThrowItem Function

Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για τον έλεγχο και την διαχείριση της ρίψης του αντικείμενου. Πιο συγκεκριμένα αν η συνθήκη που ελέγχει την πιθανότητα ρίψης είναι αληθής, τότε μέσα από την λίστα των αντικειμένων που συλλέχθηκαν από την OnTriggerEnter() , επιλέγουμε ένα τυχαίο και του ασκούμε μία δύναμη για να προσομοιάσει την ρίψη. Τέλος ενεργοποιούμε το GameObject emfPL που μας δίνει την δυνατότητα να το ελέγξουμε για πιθανό στοιχείο EMF Level 5.

4.5.4 Haunt Manager & GhostHaunt Scripts

Από τα τελευταία script του φαντάσματος είναι τα HauntManager.cs και GhostHaunt.cs

Συγκεκριμένα ο ρόλος του Haunt Manager είναι να ελέγχει ανά διαστήματα των 10 δευτερολέπτων αν το φάντασμα έχει την δυνατότητα να κυνηγήσει, αν μπορεί τότε καλεί την συνάρτηση TryStartHauntBasedOnSanity()

```
Unity Script (1 asset reference) | 0 references
public class HauntManager : MonoBehaviour {
    public GhostHaunt ghostHaunt;
    public SanityManager sanity;
    public float checkInterval = 10f;
    private float timer = 0f;



    Unity Message | 0 references
    private void Start() {
        GameObject playerRef = GameObject.FindGameObjectWithTag("Player");
        sanity = playerRef.GetComponent<SanityManager>();
    }
    Unity Message | 0 references
    void Update() {
        timer += Time.deltaTime;

        if (timer >= checkInterval && ghostHaunt.isHaunting == false) {

            ghostHaunt.TryStartHauntingBasedOnSanity();
            timer = 0f;

        }
    }
}
```

HauntManager.cs

Εφόσον καλέσουμε την παραπάνω συνάρτηση η ροή του κώδικα θα μεταφερθεί στον παρακάτω script με όνομα GhostHaunt.cs

```
public class GhostHaunt : MonoBehaviour {
    public float updateRate = 0.2f;
    public float lostSightDelay = 3f;
    public LayerMask sightLayers;
    public GameObject GhostModel;
    public float hauntRetryCooldown = 10f;
    public MonoBehaviour[] scriptsToDisableDuringHaunt;
    public AudioSource HauntScream;
    private Vector3 lastKnownPlayerPosition;
    public bool PlayerDied;

    private Transform player;
    private NavMeshAgent agent;
    private GhostAI ghostAI;

    public EndHandler endgame;
    public bool isHaunting = false;
    private bool hasLineOfSight = false;
    private float lostSightTimer = 0f;
    private bool hasTriedHaunt = false;

    Unity Message | 0 references
    void Start() {
        PlayerDied = true;
        agent = GetComponent<NavMeshAgent>();
        ghostAI = GetComponent<GhostAI>();
        GameObject playertr = GameObject.FindGameObjectWithTag("Player");
        endgame = playertr.GetComponentsInChildren<EndHandler>(true)
                          .FirstOrDefault(e => e.gameObject.name == "LeaveMapPanel");


    }

    1 reference
    public void TryStartHauntingBasedOnSanity() {
        if (isHaunting || hasTriedHaunt) return;



        if (SanityManager.Instance.sanity < 60f) {
            Debug.Log("Try Haunt");
            hasTriedHaunt = true;
            float hauntChance = Random.Range(0f, 1f);
            if (hauntChance < 0.3f) {
                Debug.Log("Success");
                StartHaunting();
            } else {
                Debug.Log("Failed");
                StartCoroutine(ResetHauntTryAfterDelay());
            }
        }
    }
```

GhostHaunt.cs Part 1

Για αρχή ορίζουμε αρκετές μεταβλητές στο ξεκίνημα του κώδικα. Οι πιο σημαντικές για αναφορά είναι:

1) updateRate: Αφορά τον ρυθμό με τον οποίο μετράμε την διάρκεια που κυνηγάει το φάντασμα μέχρι και να διακοπεί το κυνήγι
2) lostSightDelay: Αφορά τον χρόνο που χρειάζεται για να θεωρηθεί από το φάντασμα ότι ο παίκτης είναι εκτός οπτικού πεδίου
3) sightLayers : Τα Layers που θα ελέγχει το φάντασμα κατά την διάρκεια του κυνηγιού (το συγκεκριμένο το ορίζουμε από το Inspector = Player)
4) hauntRetryCooldown: Το χρονικό διάστημα μέχρι το φάντασμα να προσπαθήσει να κυνηγήσει ξανά τον παίκτη μετά από ένα αποτυχημένο κυνήγι.

5) scriptstoDisableDuringhaunt: Αφορά τα script τα οποία θα πρέπει να απενεργοποιηθούν όταν ξεκινήσει το κυνήγι.
6) lastKnownPlayerPosition: Είναι μια μεταβλητή τύπου Vector 3 (συντεταγμένες x, y, z) που διατηρούν την τοποθεσία του παίκτη για ένα μικρό χρονικό διάστημα αφού το φάντασμα χάσει την οπτική επαφή με τον παίκτη.

Εφόσον δηλώσαμε τις τιμές μας, στην Start() ανακτούμε τιμές από τα Navmesh και τον παίκτη.

Στην συνέχεια εφόσον από το HauntManager.cs καλέσαμε την TryStartHauntingBasedOnSanity()
, θα ελέγξουμε μέσα σε αυτήν αν το φάντασμα κυνηγάει ή αν βρίσκεται σε παύση μέχρι να μπορεί να κυνηγήσει ξανά. Σε περίπτωση που δεν είναι, θα προσπαθήσουμε να εκτελέσουμε κυνήγι **μόνο** αν ο παίκτης διαθέτει λιγότερο από 60% sanity και μόνο αν επιτύχει η πιθανότητα της συνθήκης (η οποία είναι 30%).

Σε περίπτωση επιτυχίας θα καλέσουμε την συνάρτηση StartHaunting() αλλιώς θα καλέσουμε την συνάρτηση ResethauntTryAfterDelay() η οποία αναμένει όσο έχουμε θέσει την τιμή hauntRetryCooldown, και στην συνέχεια προσπαθεί ξανά για κυνήγι.

```
1 reference
IEnumerator ResetHauntTryAfterDelay() {
    yield return new WaitForSeconds(hauntRetryCooldown);
    hasTriedHaunt = false;
}

1 reference
public void StartHaunting() {
    foreach (var script in scriptsToDisableDuringHaunt) {
        if (script != null) {
            script.enabled = false;
            // If it has canScare, disable it
            var ghostEvent = script as GhostEvent;
            if (ghostEvent != null) ghostEvent.canScare = false;
        }
    }
    GhostModel.SetActive(true);
    HauntScream.Play();

    GameObject myplayer_ref = GameObject.FindGameObjectWithTag("Player");
    player = myplayer_ref.transform;

    isHaunting = true;
    ghostAI.enabled = false;
    agent.isStopped = false;
    agent.updatePosition = true;
    agent.updateRotation = true;

    // Immediately move to player's position
    if (player != null)
        agent.SetDestination(player.position);

    StartCoroutine(HauntLoop());
}
```

GhostHaunt.cs Part 2

Στην συνάρτηση StartHaunting, εκτελούμε τις κατάλληλες ενέργειες πριν ξεκινήσουμε το κυνήγι, Πιο συγκεκριμένα, απενεργοποιούμε τα script που θέσαμε στο Inspector,εμφανίζουμε την μορφή του φαντάσματος καθώς και την παραγωγή ενός ήχου που αναδεικνύει το ξεκίνημα ενός Haunt, και ενημερώνουμε κατάλληλα το φάντασμα για την θέση του παίκτη και οδηγώντας το προς αυτόν.

Παράλληλα απενεργοποιούμε το GhostAI.cs με σκοπό να μην ξεφύγει το φάντασμα από την λογική του Haunting. Μετά από όλες τις παραπάνω ενέργειες , καλούμε το HauntLoop().

```
1 reference
IEnumerator HauntLoop() {
    float hauntDuration = 30f;
    float hauntTimer = 0f;


    float searchTimer = 0f;
    bool isSearching = false;

    while (isHaunting && hauntTimer < hauntDuration) {
        if (player == null) yield break;

        hauntTimer += updateRate;

        hasLineOfSight = false;
        int rayCount = 36;
        float angleStep = 360f / rayCount;
        Vector3 origin = transform.position + Vector3.up * 1.5f;

        for (int i = 0; i < rayCount; i++) {
            float angle = i * angleStep;
            Vector3 direction = Quaternion.Euler(0, angle, 0) * transform.forward;

            if (Physics.Raycast(origin, direction, out RaycastHit hit, 10f, sightLayers)) {
                if (hit.transform.CompareTag("Player")) {
                    hasLineOfSight = true;
                    break;
                }
            }
        }

        if (hasLineOfSight) {
            lostSightTimer = 0f;
            searchTimer = 0f;
            isSearching = false;

            lastKnownPlayerPosition = player.position;
            agent.SetDestination(player.position);
        } else {
            lostSightTimer += updateRate;

            if (!isSearching && lostSightTimer >= lostSightDelay) {
                isSearching = true;
                searchTimer = 0f;
                agent.SetDestination(lastKnownPlayerPosition);
                StartCoroutine(LookAround(lastKnownPlayerPosition, 2f));
            }

            if (isSearching) {
                searchTimer += updateRate;

                if (!agent.hasPath || agent.remainingDistance < 0.5f) {
                    agent.SetDestination(GetRandomNearbyPoint());
                }
            }
        }

        yield return new WaitForSeconds(updateRate);
    }
    StopHaunting(false);
}
```

GhostHaunt.cs Part 3

Η συνάρτηση HauntLoop ξεκινάει με την δήλωση της διάρκειας του Haunting καθώς και του χρονόμετρού hauntTimer.
Επειδή στην αρχή το φάντασμα θα γνωρίζει την θέση του παίκτη θεωρούμε την Boolean isSearching = false.

Στην συνέχεια μέσω του βρόγχου while ελέγχουμε αν το φάντασμα κυνηγάει ακόμα ή αν τελείωσε η χρονική διάρκεια που μπορούσε να κυνηγάει. Εφόσον δεν ισχύει , θα περάσουμε στις εντολές του βρόγχου οι οποίες ενημερώνουν το hauntTimer και θέτουμε ότι το φάντασμα δεν έχει οπτική επαφή. Στην συνέχεια από το φάντασμα δηλώνουμε ότι θα φύγουν 36 ακτίνες με σκοπό να ελέγχει το φάντασμα σε 360 μοίρες το περιβάλλον για την παρουσία του παίκτη. Στην συνέχεια για κάθε μία από αυτές τις ακτίνες ελέγχουμε αν συγκρουστούν με κάποιο αντικείμενο που έχει ως tag το Player. Εφόσον υπάρξει αυτό το αντικείμενο τότε ενημερώνουμε το hasLineOfSight = true.

Τέλος γίνεται έλεγχος για το αν το hasLineOfSight είναι αληθές. Στην περίπτωση που είναι ενημερώνουμε το φάντασμα με την πιο πρόσφατη θέση του παίκτη και την θέτουμε ως προορισμό για το φάντασμα. Διαφορετικά αυξάνεται το χρονόμετρο που αφορά την έλλειψη της οπτικής επαφής του παίκτη με το φάντασμα. Έτσι το φάντασμα θα κάνει μία τελευταία προσπάθεια να επισκεφθεί τη τελευταία τοποθεσία που είδε τον παίκτη και στην συνέχεια θα επιλέγει διάφορα σημεία στην περιοχή μέχρι να λήξει ο χρόνος του κυνηγιού με την GetRandomNearbyPoint().
Εφόσον συμπληρωθούν τα 30 δευτερόλεπτα το κυνήγι θα σταματήσει με το κάλεσμα της συνάρτησης StopHaunting() και μία παράμετρο Boolean που είναι false .

```
public void StopHaunting(bool isDead) {

    foreach (var script in scriptsToDisableDuringHaunt) {
        if (script != null) {
            script.enabled = true;
            var ghostEvent = script as GhostEvent;
            if (ghostEvent != null) ghostEvent.canScare = true;
        }
    }


    HauntScream.Stop();
    GhostModel.SetActive(false);

    if (isDead) {
        isHaunting = true;
        PlayerDied = true;
        StartCoroutine(HandleDeathSequence());


    } else {
        isHaunting = false;
        PlayerDied = false;
    }
    hasTriedHaunt = false;
    player = null;

}

1 reference
IEnumerator LookAround(Vector3 position, float duration) {
    float timer = 0f;
    while (timer < duration && isHaunting) {
        Vector3 direction = (position - transform.position).normalized;
        direction.y = 0f;
        transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(direction), 0.05f);
        timer += Time.deltaTime;
        yield return null;
    }
}
1 reference
private IEnumerator HandleDeathSequence() {
    PlayerDeathEffects.Instance.PlayDeathEffects(4f);      // Play effects first
    yield return new WaitForSeconds(4f); // Wait for them to finish
    endgame.OnYesButtonPressed();       // Then end the game
}
```

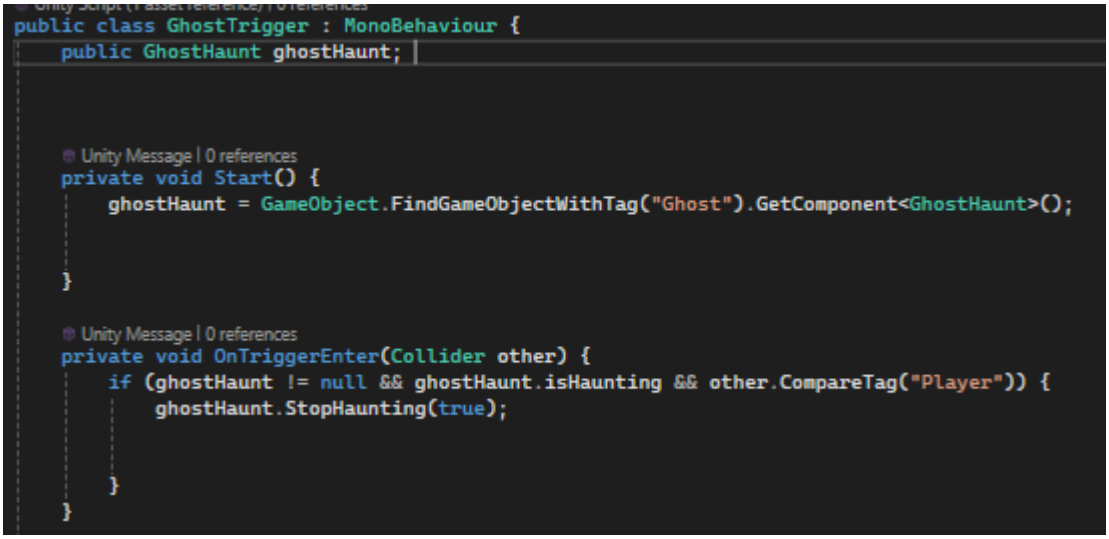
GhostHaunt.cs Part 3

Η συνάρτηση StopHaunting καλείται με παράμετρο μία Boolean μεταβλητή. Ξεκινώντας, αυτή η συνάρτηση επαναφέρει όλα τα script που είχαν απενεργοποιηθεί, καθώς και εξαφανίζει το μοντέλο του φαντάσματος ξανά. Στην συνέχεια ελέγχουμε αν ο παίκτης έχει πεθάνει από το φάντασμα. Στην περίπτωση που αληθεύει , ορίζουμε το isHaunting = true για να μην προκαλέσει άλλο Haunting το φάντασμα μέχρι να αλλάξει σκηνή το παιχνίδι, και την κατάσταση του παίκτη με το PlayerDied.

Η συνάρτηση LookAround χρησιμοποιείται για την γρήγορη εναλλαγή τοποθεσίας του φαντάσματος με σκοπό να βρει ξανά τον παίκτη εφόσον έχασε οπτική επαφή για ένα μικρό χρονικό διάστημα.

Η συνάρτηση HandleDeathSequence() ενεργοποιεί την συνάρτηση από το PlayerDeathEffects που είναι ενσωματωμένο στον παίκτη, για να προσομοιάσει το θάνατο του παίκτη, και καλεί την συνάρτηση OnYesButtonPressed() από το script EndGame.cs .

Κλείνοντας από το φάντασμα και τα script του για το κυνήγι, για να ενεργοποιήσουμε και να ενημερώσουμε το StopHaunting() ότι ο παίκτης πέθανε, χρησιμοποιούμε το GhostTrigger.cs



```
public class GhostTrigger : MonoBehaviour {
    public GhostHaunt ghostHaunt;

    Unity Message | 0 references
    private void Start() {
        ghostHaunt = GameObject.FindGameObjectWithTag("Ghost").GetComponent<GhostHaunt>();

    }

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other) {
        if (ghostHaunt != null && ghostHaunt.isHaunting && other.CompareTag("Player")) {
            ghostHaunt.StopHaunting(true);

        }
    }
```

GhostTrigger.cs

Το οποίο ελέγχει αν το Trigger Collider πάνω στο φάντασμα συγκρουστεί με τον παίκτη με την βοήθεια της OnTriggerEnter(). Στην περίπτωση που συμβεί, καλούμε την συνάρτηση StopHaunting() με παράμετρο true που θα δοθεί στην μεταβλητή isDead του Ghosthaunt.cs .

4.5.5 EndHandler.cs

```
public class EndHandler : MonoBehaviour {



    private Flashlight flashlight_script;
    private CameraPhotoCapture camera_script;
    private GhostHaunt ghost_haunt;
    private float endtimer = 0;
    private bool Deathresult;



    Unity Message | 0 references
    private void Update() {
        endtimer += Time.deltaTime;
    }

    1 reference
    public void OnYesButtonPressed() {
        flashlight_script = GameObject.FindGameObjectWithTag("Flashlight").GetComponent<Flashlight>();
        camera_script = GameObject.FindGameObjectWithTag("PHOTOCAMERA").GetComponent<CameraPhotoCapture>();
        ghost_haunt = GameObject.FindGameObjectWithTag("Ghost")?.GetComponent<GhostHaunt>();
        if (ghost_haunt == null) {

            Deathresult = true;
        } else {

            Deathresult = ghost_haunt.PlayerDied;
            Debug.Log(Deathresult);
        }

        var guessedGhost = GhostSelectedEffect.CurrentlySelected != null
            ? (GhostTypeSelector.GhostType)System.Enum.Parse(
                typeof(GhostTypeSelector.GhostType),
                GhostSelectedEffect.CurrentlySelected.ghostName
            )
            : GhostTypeSelector.GhostType.Spirit;

        var actualGhost = GhostTypeSelector.Instance != null
            ? GhostTypeSelector.Instance.selectedGhostType
            : GetRandomGhostExcludingSpirit();

        ResultsManager.Instance.SetResults(
            guessedGhost,
            actualGhost
        );


        ResultsManager.Instance.SetAdditionalStats(
            time: endtimer/60,
            photos: camera_script.validphotos,
            died: Deathresult,
            NoFlashlight:!flashlight_script.FlashlightUsed
        );
        MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.None;
        SceneManager.LoadScene("Lobby Menu");
    }
```

EndHandler.cs

Το παραπάνω script είναι το τελευταίο script που καλείται από την σκηνή με τίτλο Map πριν επιστρέψουμε στον Lobby Menu. Μπορεί να ενεργοποιηθεί με δύο τρόπους ο πρώτος είναι με τον θάνατο του παίκτη από το φάντασμα που αναφέρθηκε παραπάνω και ο δεύτερος τρόπος είναι μέσα από το φορτηγό που βρίσκεται μπροστά από στον παίκτη μόλις δημιουργηθεί, πηγαίνοντας δίπλα από την πόρτα του οδηγού.

Van EndGame

Με το πάτημα του πλήκτρου "Space" ο παίκτης θα ενεργοποιήσει το script CameraToggle.cs που αναφέραμε , και θα μετακινηθεί μέσα στο φορτηγό του, όπως φαίνεται στην παρακάτω εικόνα:

LeaveMapPanel

Με το πάτημα του κουμπιού "Yes" ο παίκτης δέχεται να φύγει από την πίστα με τα τρέχοντα αποτελέσματα από την έρευνα του. Διαφορετικά αν πατήσει "No" ο παίκτης θα βγει αυτομάτως από το φορτηγό για να συνεχίσει την έρευνα.

Οι μεταβλητές που δηλώνουμε στην αρχή αφορούν τα side missions που αναλύσαμε προηγουμένως με την βοήθεια του ResultsManager.cs

Η Update μετράει από την αρχή του script τον χρόνο που ο παίκτης βρίσκεται στην πίστα μέχρι και να φύγει.

Η συνάρτηση OnYesButtonPressed ανακτάει τις τιμές από τα διάφορα script που χρειάζονται για τα side missions, για να ελέγξουμε και να στείλουμε τα αποτελέσματα από τις μεταβλητές αυτές στο ResultsManager.cs
Στην συνέχεια ελέγχουμε αν το φάντασμα είχε δημιουργηθεί στην πίστα, αν ναι τότε το guessedGhost λαμβάνει την τιμή που δόθηκε στο φάντασμα κατά την δημιουργία του, αλλιώς του παραθέτει ένα τυχαίο τύπο φαντάσματος **εκτός** του Spirit. Αντίστοιχα για την επιλογή του παίκτη, αν ο παίκτης δεν επιλέξει κάποιο τότε θα θεωρηθεί αυτόματα σαν επιλογή το Spirit, αλλιώς θα οριστεί η επιλογή του από το GhostSelection Panel. Στην συνέχεια περνάμε μέσω του ResultsManager.Instance τα αποτελέσματα από την πίστα και πραγματοποιείται έξοδος στο Lobby Menu για την ανακοίνωση των αποτελεσμάτων.

```
1 reference
private GhostTypeSelector.GhostType GetRandomGhostExcludingSpirit() {
    var ghostValues = System.Enum.GetValues(typeof(GhostTypeSelector.GhostType));
    GhostTypeSelector.GhostType randomGhost;
    do {
        int randomIndex = Random.Range(0, ghostValues.Length);
        randomGhost = (GhostTypeSelector.GhostType)ghostValues.GetValue(randomIndex);
    } while (randomGhost == GhostTypeSelector.GhostType.Spirit);
    return randomGhost;
}

0 references
public void OnNoButtonPressed() {
    MainMenuFucntions.CurrentUI = MainMenuFucntions.UIState.None;
}
```

GetRandomGhostExcludingSpirit Function

Στην συγκεκριμένη συνάρτηση λαμβάνουμε στο ghostValues όλες τις διαθέσιμες τιμές από το enum GhostType, και λειτουργούμε έναν βρόγχο μέχρις ότου να μην επιλεχθεί ως φάντασμα το Spirit.

ΚΕΦΑΛΑΙΟ 5 Συμπεράσματα και Μελλοντικές Προσεγγίσεις

5.1. Συμπεράσματα

Η παρούσα πτυχιακή εργασία ανέδειξε τη διαχρονική εξέλιξη της βιομηχανίας των βιντεοπαιχνιδιών, από τις πρώτες μορφές διαδραστικής ψυχαγωγίας έως τις σύγχρονες προσεγγίσεις που αξιοποιούν εξελιγμένες μηχανές γραφικών και τεχνητής νοημοσύνης όπως η μηχανή παραγωγής παιχνιδιών Unity.

Με την ανάπτυξη του παιχνιδιού Ghost Seekers, Έγινε εφαρμογή των θεωρητικών γνώσεων στην πράξη. Το έργο αυτό απέδειξε πως συνδυάζονται μηχανισμοί πλοήγησης, τεχνητής νοημοσύνης και διαδραστικού gameplay, Δημιουργώντας μια εμπειρία εξερεύνησης και αλληλεπίδρασης σε ατμοσφαιρικό περιβάλλον.

5.2 Μελλοντικές Προσεγγίσεις

Παρά την ολοκληρωμένη παρουσίαση και ανάπτυξη, η έρευνα και η υλοποίηση μπορούν να επεκταθούν περαιτέρω με διάφορους τρόπους :

1) **Επέκταση του gameplay**
   - Δυνατότητα εμπλουτισμού του σεναρίου με περισσότερες αποστολές και περιοχές για έρευνα φαντασμάτων.
   - Δημιουργία περισσότερων Scare Event καθώς και Animation
   - Δημιουργία περισσότερων στοιχείων καθώς και τύπους φαντασμάτων
2) **Βελτίωση της τεχνητής νοημοσύνης με την Unity 6**
   - Χρήση της έκδοσης Unity 6 με τα νέα μοντέλα Machine Learning για πιο δυναμική και προσαρμοστική συμπεριφορά των εχθρών.
   - Δημιουργία οντοτήτων που έχουν την δυνατότητα να "μαθαίνουν" από τις στρατηγικές του παίκτη και προσαρμόζουν την δράση τους.
3) **Δημιουργία Multiplayer Mode**
   - Υλοποίηση Multiplayer Mode με έμφαση στους τρόπους συνεργασίας μεταξύ των παικτών
4) **Επέκταση Γραφικών**
   - Με την χρήση της Unity 6 , δίνεται η δυνατότητα για ενσωμάτωση πιο ρεαλιστικών φωτισμών και σκιών για ενίσχυση ατμόσφαιρας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Contributors, W. (2025, March 14). *Unity (game engine)*. Ανάκτηση από https://en.wikipedia.org/wiki/Unity_(game_engine)

GameDeveloper. (2012, May 24). *Mobile game developer survey leans heavily toward iOS, Unity*. Ανάκτηση από https://www.gamedeveloper.com/audio/mobile-game-developer-survey-leans-heavily-toward-ios-unity

Grubb, J. (2017, March 31). *Unity 5.6 launches with support for Vulkan graphics, Nintedo Switch, and more*. Ανάκτηση από https://web.archive.org/web/20190320210243/https://venturebeat.com/2017/03/31/unity-5-6-launches-with-support-for-vulkan-graphics-nintendo-switch-and-more/

Haas, J. K. (2014). *A history of the unity game engine*. Ανάκτηση από http://www.daelab.cn/wp-content/uploads/2023/09/A_History_of_the_Unity_Game_Engine.pdf

Kumparak, G. (2014, March 18). *Unity 5 Announced With Better Lighting, Better Audio, And "Early" Support For Plugin-Free Browser Games*. Ανάκτηση από https://techcrunch.com/2014/03/18/unity-5-announced-with-early-support-for-plugin-free-browser-games/

Smykil, J. (2006, August 10). *Apple Design Award winners announced*. Ανάκτηση από https://arstechnica.com/gadgets/2006/08/4937/