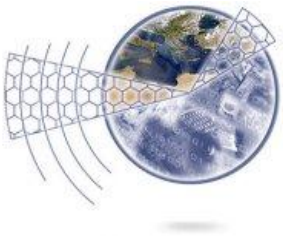
Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων



Πτυχιακή εργασία

Τίτλος:

Ανάπτυξη Παιχνιδιού σε Πλατφόρμα Unity3d με Θέμα την Κνωσό

Καρέντζος Κωνσταντίνος (ΑΜ:1622)
Χατζηδήμος Αλέξανδρος (ΑΜ: 1804)

Επιβλέπων καθηγητής : Μαλάμος Αθανάσιος

Ευχαριστίες

Θέλουμε να ευχαριστήσουμε όσους βοήθησαν να δημιουργηθεί αυτή η πτυχιακή**.** Η στήριξη τους, με τον, οποιοδήποτε τρόπο, ήταν απαραίτητη μιας και ήταν η πρώτη φορά που ασχοληθήκαμε με ένα project μεγαλύτερου μεγέθους και γενικότερα με τον τομέα του game development. Ελπίζουμε το αποτέλεσμα να δικαιώσει όχι μόνο εμάς αλλά και εκείνους, που υπομείνανε τα περίεργα ωράρια μας και ήταν εκεί όταν χρειαστήκαμε το οτιδήποτε.

Επιπλέον θέλουμε να τους αφιερώσουμε την πτυχιακή αυτή ελπίζοντας πως θα είναι το ξεκίνημα για μια επιτυχημένη καριέρα στο game development και πως θα μας στηρίζουν σε οτιδήποτε αποφασίσουμε να δημιουργήσουμε στο μέλλον.

Τέλος θέλουμε να ευχαριστήσουμε θερμά τον Ανδρέα Σταμούλια ο οποίος μας παρείχε την βοήθειά του όποτε την χρειαζόμασταν.

Abstract

Our Thesis is a three dimensional adventure game created for educational purposes so the player can gain some knowledge about the great city of Minoan Crete, Knossos. By interacting with the game, the player will learn about antiquities found in ancient Crete. Through various restoration mechanisms he will interact with those antiquities to restore them so he can see how they looked back then.

The player, who is an archaeologist, begins his adventure inside his lab. The purpose of the lab is twofold. First he uses one of four mechanisms for restoring the antiquities. Once restored, they are displayed in the exhibition room where the player can interact and see some information about them. The mechanisms are:

- Cleaning an object from dust.
- Rotating and cleaning an object from mud.
- A scanner that x-rays objects that have accumulated rocks over time.
- A fun mini game where the player has to connect fragments that are broken.

Each object that is restored is put on display in the display room atop some pedestals or inside the main marble display with a glass casing.

We chose to create our game using the popular game engine Unity 3d. It offers a very clean and understandable graphical user interface that has a small learning curve to master, easy importing system for assets and packages, easy animation system, intuitive particle system and many more features that a first time user can use quite easily of the box. It supports the following scripting languages. Javascript, boo and C#. We chose to develop our game using Javascript because it is more versatile in its use and implementation.

For our assets creation we used a plethora of software such as: Autodesk's 3dStudio Max, Google Sketchup, Autodesk's motion builder, Pixologic's Zbrush, N-Sided's Quidam, Photoshop and Audacity. Our export format, for the 3d assets, was the .fbx format which is supported by all these programs and gives many options upon exporting.

Σύνοψη

Η πτυχιακή μας εργασία είναι ένα τρισδιάστατο παιχνίδι περιπέτειας που δημιουργήθηκε για εκπαιδευτικούς σκοπούς, ώστε ο παίκτης μπορεί να αποκτήσει κάποιες γνώσεις σχετικά με τη μεγάλη πόλη της Μινωικής Κρήτης, την Κνωσό. Αλληλεπιδρώντας με το παιχνίδι, ο παίκτης θα μάθει για διάφορες αρχαιότητες που βρέθηκαν στην αρχαία Κρήτη. Μέσω διαφόρων μηχανισμών αποκατάστασης αυτών των αρχαιοτήτων θα μπορέσει να πάρει μια ιδέα για το πώς μπορεί να φαινόταν τότε.

Ο παίκτης, ο οποίος είναι ένας αρχαιολόγος, ξεκινά την περιπέτειά του μέσα στο εργαστήριό του. Ο σκοπός του εργαστηρίου είναι διττός. Πρώτα χρησιμοποιεί έναν από τους τέσσερις μηχανισμούς για την αποκατάσταση των αρχαιοτήτων. Μόλις ένα αντικείμενο αποκατασταθεί, εμφανίζεται στον εκθεσιακό χώρο, όπου ο παίκτης μπορεί να αλληλεπιδράσει μαζί του και να δει κάποιες πληροφορίες γι 'αυτό. Οι μηχανισμοί είναι:

- Καθαρισμός ενός αντικείμενου από τη σκόνη.
- Περιστροφή καθαρισμός ενός αντικειμένου από λάσπη.
- Ένας σαρωτής αντικειμένων για αντικείμενα που έχουν καλυφθεί με πέτρες με την πάροδο του χρόνου.
- Ένα mini game, όπου ο παίκτης πρέπει να συνδέσει μεταξύ τους κομμάτια από ένα αντικείμενο που έχει καταστραφεί.

Κάθε αντικείμενο που έχει αποκατασταθεί εμφανίζεται σαν έκθεμα στην μουσειακή αίθουσα πάνω σε κάποιο βάθρο ή στο κέντρο της αίθουσας στο κεντρικό μαρμάρινο βάθρο.

Επιλέξαμε να δημιουργήσουμε το παιχνίδι μας με την δημοφιλή game engine Unity 3d. Προσφέρει μια πολύ καθαρή και κατανοητή γραφική διεπαφή χρήστη που έχει μια μικρή καμπύλη εκμάθησης, εύκολο σύστημα εισαγωγής asset και packages, εύκολο σύστημα animation, έξυπνο particle system και πολλά άλλα χαρακτηριστικά που ο χρήστης μπορεί να χρησιμοποιήσει άμεσα με λίγη εκμάθηση . Υποστηρίζει τις ακόλουθες γλώσσες προγραμματισμού. Javascript, Boo και C #. Εμείς επιλέξαμε να αναπτύξουμε το παιχνίδι σε Javascript, επειδή είναι πιο ευέλικτη στη χρήση της και την εφαρμογή της.

Για την δημιουργία των assets μας χρησιμοποιήσαμε μια πληθώρα λογισμικού, όπως: 3dStudio Max της Autodesk, το Google SketchUp, το Motion Builder της Autodesk, το ZBrush της Pixologic του, το Photoshop και το Audacity. Για την εξαγωγή των 3d assets που χρησιμοποιήσαμε, χρησιμοποιήσαμε την .FBX μορφή που υποστηρίζεται από όλα αυτά τα προγράμματα και δίνει πολλές επιλογές κατά την εξαγωγή.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Λίστα Πινάκων

1. Εισαγωγή

Το video game development είναι η διαδικασία δημιουργίας ενός ηλεκτρονικού παιχνιδιού είτε για ηλεκτρονικό υπολογιστή, κονσόλα ή κινητή συσκευή. Η ανάπτυξη του παιχνιδιού γίνεται από έναν game developer, από μια μικρή ομάδα ή από μια μεγάλη επιχείρηση. Η συνήθης πρακτική είναι τα παιχνίδια αυτά να χρηματοδοτούνται από έναν publisher και συνήθως θέλουν κάποια χρόνια για να ολοκληρωθούν. Υπάρχει βέβαια και η κατηγορία των Indie Games(ομάδες πολλών ή ενός ατόμου που δημιουργούν video games χωρίς κάποιον publisher), όπου τα παιχνίδια μπορεί να χρειάζονται λιγότερο καιρό για να δημιουργηθούν αλλά και μικρότερο budget.

Τα πρώτα ηλεκτρονικά παιχνίδια αναπτύχθηκαν την δεκαετία του εξήντα, αλλά δεν έγιναν εμπορικά διαθέσιμα λόγω της έλλειψης προσωπικών ηλεκτρονικών υπολογιστών στα νοικοκυριά. Την δεκαετία του εβδομήντα βλέπουμε τα πρώτα ηλεκτρονικά παιχνίδια να γίνονται εμπορικά διαθέσιμα. Λόγω του μικρού κόστους για ανάπτυξη, εκείνη την εποχή, ένας προγραμματιστής μόνος του μπορούσε να αναπτύξει ένα ολοκληρωμένο παιχνίδι. Όμως από τις απόμενες δεκαετίες μέχρι σήμερα και καθώς οι απαιτήσεις συστημάτων αυξάνονταν έτσι αυξανόταν και ο προϋπολογισμός για την δημιουργία ενός παιχνιδιού. Πλέον για τα mainstream παιχνίδια, το budget κυμαίνεται από 5 εκ$ μέχρι 20εκ$. Τα Indie games βλέπουν μια άνθιση τον τελευταίο καιρό λόγω εύκολα προσβάσιμων game engine, όπως το Unity 3d, και πλατφορμών χρηματοδότησης όπως το Kickstarter.

Ο πραγματικός εμπορικός σχεδιασμός και ανάπτυξη ηλεκτρονικών παιχνιδιών ξεκίνησε τη δεκαετία του 1970, όταν τα arcade ηλεκτρονικά παιχνίδια και οι κονσόλες πρώτης γενιάς κυκλοφόρησαν στο εμπόριο . Το 1971, το Computer Space ήταν το πρώτο ηλεκτρονικό παιχνίδι, που λειτουργούσε με κέρματα, το οποίο εμφανίστηκε στην αγορά. Χρησιμοποιούσε μια μαυρόασπρη τηλεόραση σαν συσκευή εξόδου και το υπολογιστικό του σύστημα αποτελούνταν από μια σειρά εβδομήντα τεσσάρων TTL chips. . Το 1972, κυκλοφόρησε η πρώτη κονσόλα οικιακής χρήσης , που αναπτύχθηκε από τον Ralph H. Baer . Το ίδιο έτος, η Atari κυκλοφόρησε Pong , ένα arcade παιχνίδι που αύξησε την δημοτικότητα των video games. Η εμπορική επιτυχία του Pong οδήγησε άλλες εταιρείες να αναπτύξουν κλώνους του Pong, πράγμα που ξεκίνησε, ουσιαστικά, την βιομηχανία βιντεοπαιχνιδιών .

Ωστόσο, η πλημμύρα του κλώνων του Pong, εκείνη την περίοδο, οδήγησε στη πτώση των video games του 1977 , η οποίο τελικά ήρθε στο τέλος της με την mainstream επιτυχία της Taito , το παιχνίδι Space Invaders (1978) , που σηματοδοτεί την έναρξη της χρυσής εποχής των arcade παιχνίδιων και εμπνέει δεκάδες κατασκευαστές να εισέλθουν στην αγορά. Σύντομα μεταφέρθηκε στο Atari 2600 και έγινε ο λόγος για τον τετραπλασιασμό των πωλήσεων της κονσόλας. Την ίδια στιγμή, οι προσωπικοί υπολογιστές εμφανίστηκαν στην αγορά, επιτρέποντας σε προγραμματιστές και χομπίστες να αναπτύξουν παιχνίδια. Αυτό επέτρεψε τον κατασκευαστή του υλικού και κατασκευαστές λογισμικού να ενεργήσει χωριστά.

Με την ολοένα αυξανόμενη επεξεργαστική και γραφική δυνατότητα του arcade, των κονσολών και προϊόντων πληροφορικής, σε συνδυασμό με την αύξηση των προσδοκιών των χρηστών, ο σχεδιασμός των παιχνιδιών κινήθηκε πέρα από την ιδέα ενός μόνο προγραμματιστή να πρέπει να παράγει ένα εμπορεύσιμο προϊόν σε συγκεκριμένο χρονικό διάστημα. Αυτό πυροδότησε την έναρξη της ανάπτυξης με ομάδες.

Από την τρίτη γενιά κονσολών , η βιομηχανία ηλεκτρονικών παιχνιδιών αυξάνεται και επεκτείνεται. Τα έσοδα της βιομηχανίας έχουν αυξηθεί τουλάχιστον πέντε φορές από το 1990. Το 2007, τα έσοδα των ηλεκτρονικών παιχνιδιών ήταν 9,5 δισεκατομμύρια δολάρια, ξεπερνώντας τα έσοδα της βιομηχανίας του κινηματογράφου.

1.1 Περίληψη

Σκοπός της πτυχιακής αυτής ήταν η δημιουργία ενός τρισδιάστατου παιχνιδιού τρίτου προσώπου με εκπαιδευτικό προσανατολισμό. Κεντρικό της θέμα και μοτίβο ήταν η Κνωσός της Μινωικής Κρήτης, από την οποία εμπνευστήκαμε για να μπορέσει ο παίκτης να ανακαλύψει, και να γνωρίσει, αρχαιότητες από διάφορες εποχές και περιοχές .

Πέρα από τον εκπαιδευτικό χαρακτήρα που θελήσαμε, στο καλύτερο των δυνατοτήτων μας, να δώσουμε στο παιχνίδι έπρεπε και ο τρόπος που θα βρίσκει ο παίκτης τις αρχαιότητες και θα τις επεξεργάζεται να είναι διασκεδαστικός αρκετά για να του τραβήξει το ενδιαφέρον.

Έτσι αποφασίσαμε να κρατήσουμε την βασική ουσία από την διαδικασία ανάκτησης αντικειμένων από αρχαιολογικούς χώρους και την επεξεργασία τους για να αναστηλωθούν όσο πιο πιστά στην αρχική τους κατάσταση. Όμως το να προσομοιώσουμε κατά γράμμα τις διαδικασίες αυτές, εκτός από αρκετά περίπλοκο, έπρεπε οι μηχανισμοί του παιχνιδιού να είναι πιο προσιτοί και απλοί για να παραμείνει η ισορροπία ανάμεσα στο παιχνίδι και την εκμάθηση. Έτσι καταλήξαμε σε πέντε απλούς μηχανισμούς.

Ο Βασικός μηχανισμός είναι το σκάψιμο που γίνεται στο terrain για να ανακτήσει τις αρχαιότητες. Στην συνέχεια οι υπόλοιποι τέσσερις μηχανισμοί εκτελούνται στο εργαστήριο του αρχαιολόγου. Εκεί, πρέπει να καθαρίσει κάποια αντικείμενα από τη σκόνη, κάποια άλλα από λάσπη, κάποια τα οποία μπορεί να βρίσκονται σε κάποιο πέτρωμα περνάνε μέσα από ένα scanner για να αποκαλυφθεί η αρχαιότητα και τέλος ένα mini game που πρέπει να κολλήσει μεταξύ τους κομμάτια από ένα σπασμένο αντικείμενο.

Μετά από αυτές τις απλοποιημένες διαδικασίες ο παίκτης περνάει και στο εκπαιδευτικό κομμάτι του παιχνιδιού όταν παρατηρεί από κοντά τα αντικείμενα που συνέλεξε και διαβάζει μια μικρή περιγραφή για το τι είναι το καθένα και από ποια εποχή της Μινωικής Κρήτης.

1.2 Κίνητρο για την Διεξαγωγή της Εργασίας

Βασικό κίνητρο που είχαμε για την δημιουργία της πτυχιακής, ήταν η αγάπη μας για τα Video games για τα οποία έχουμε αφιερώσει πολύ χρόνο από παιδιά μέχρι σήμερα. Έτσι ήταν απόλυτα λογικό κάποια στιγμή να θελήσουμε να δημιουργήσουμε ένα δικό μας παιχνίδι.

Το να προτείνουμε και να διαλέξουμε μια πτυχιακή η οποία σχετίζεται και βρίσκεται μέσα στον κλάδο του game development ήταν φυσικό επόμενο. Ήταν μια πρώτης τάξεως ευκαιρία να δούμε από πρώτο χέρι ποιες είναι οι διαδικασίες ανάπτυξης ενός ηλεκτρονικού παιχνιδιού και πως θα μπορούσαμε να ανταπεξέλθουμε σε αυτές.

1.3 Σκοπός και Στόχοι Εργασίας

Σκοπός της πτυχιακής, πέρα από το τελικό αποτέλεσμα που θέλαμε να καταφέρουμε να φέρουμε εις πέρας, ήταν να αναπτύξουμε τις ικανότητες μας σε όλα τα στάδια δημιουργίας ενός ηλεκτρονικού παιχνιδιού. Τα στάδια αυτά και οι διαδικασίες είναι πολλά και περίπλοκα. Έπρεπε να αναπτύξουμε τις ικανότητες μας στο 3d modeling και όλα τα παράγωγα αυτού, το animation και το scripting.

Οι στόχοι αυτοί που θέσαμε ήταν βασικοί και έπρεπε, μέχρι ένα σημείο, να καταφέρουμε να ανταπεξέλθουμε στις απαιτήσεις που θα παρουσιαζόντουσαν. Εξάλλου όλοι αυτοί οι τομείς που αναφέραμε, σε μια μεγάλη παραγωγή είναι ξεκάθαρα διακριτοί και ξεχωριστοί και τους αναλαμβάνουν συγκεκριμένες ομάδες με πείρα και πολύ εμπειρία.

1.4 Δομή Εργασίας

Η Δομή της εργασίας είναι η εξής:

- Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή για να καταλάβει ο αναγνώστης τον σκοπό της εργασίας και αναλύονται συνοπτικά τα κίνητρα και ο σκοπός της.
- Στο κεφάλαιο δύο θα δούμε και θα αναλύσουμε τα στάδια δημιουργίας ενός ηλεκτρονικού παιχνιδιού.
- Στο κεφάλαιο τρία θα ασχοληθούμε με τα προγράμματα λογισμικού που χρησιμοποιήσαμε για να δημιουργήσουμε τα assets της πτυχιακής και κυρίως να αναλύσουμε το βασικό πρόγραμμα που είναι ο συνδετικός κρίκος όλης της εργασίας, το Unity 3d. Για το κάθε λογισμικό θα υπάρχει μια μικρή εισαγωγή για να μπορέσει ο αναγνώστης να καταλάβει ποιά είναι η λειτουργία του και πως το χρησιμοποιήσαμε εμείς για να πετύχουμε το τελικό αποτέλεσμα.
- Στο κεφάλαιο τέσσερα θα κάνουμε ένα overview του παιχνιδιού για να εξηγήσουμε στον αναγνώστη πως παίζεται το παιχνίδι και ποιοι είναι οι μηχανισμοί.

- Στο κεφάλαιο πέντε θα αναλύσουμε αναλυτικά τους μηχανισμούς του παιχνιδιού και εξηγήσουμε τον κώδικα που κρύβεται από πίσω τους.
- Στο κεφάλαιο έξι θα δούμε πως δημιουργούμε γραφικά περιβάλλοντα στο Unity.
- Στο κεφάλαιο επτά θα δούμε τι είναι, πως λειτουργούν οι shaders και ποια είναι τα είδη τους.
- Στο κεφάλαιο οκτώ θα ασχοληθούμε με μελλοντικά implementations του παιχνιδιού.
- Στο κεφάλαιο εννέα θα συνοψίσουμε και θα κάνουμε τις τελικές παρατηρήσεις της εργασίας.

2.Ανάπτυξη ηλεκτρονικών παιχνιδιών

Όπως είδαμε και στην εισαγωγή της εργασίας, το video game development, είναι διαδικασία δημιουργίας και παραγωγής ενός ηλεκτρονικού παιχνιδιού. Πλέον, με την διάδοση και πιο εύκολη πρόσβαση σε επαγγελματικά εργαλεία, εκτός από τις μεγάλες εταιρίες παραγωγής έχουν εμφανιστεί πολλές ανεξάρτητες εταιρίες οι οποίες μέσω του crowd funding μπορούν να φέρουν εις πέρας τα project τους.

Τα ηλεκτρονικά παιχνίδια δημιουργούνται ως δημιουργική διέξοδος, από ερασιτέχνες, και για την παραγωγή κέρδους από mainstream αλλά και από ανεξάρτητες εταιρείες. Η ανάπτυξη του παιχνιδιού λαμβάνει χρηματοδότηση, συνήθως, από έναν εκδότη (publisher). Ηλεκτρονικά παιχνίδια υψηλής ποιότητας θα αποφέρουν στην εταιρεία κέρδη άμεσα. Παρ' όλα αυτά, είναι σημαντικό να ληφθούν υπόψη οι οικονομικές απαιτήσεις που θα χρειαστεί το παιχνίδι, όπως τα κόστη ανάπτυξης των επιμέρους χαρακτηριστικών του.

Η βιομηχανία παιχνιδιών απαιτεί καινοτομίες καθώς οι εκδότες δεν μπορούν να επωφεληθούν από την ανάπτυξη επαναλαμβανόμενων παιχνιδιών . Κάθε χρόνο εμφανίζονται νέες ανεξάρτητες εταιρείες ανάπτυξης ηλεκτρονικών παιχνιδιών, οι οποίες καταφέρνουν να παράγουν έναν τουλάχιστον επιτυχημένο τίτλο. Ομοίως, πολλοί developers κλείνουν επειδή δεν μπορούν να βρουν μια εκδοτική σύμβαση ή γιατί οι παραγωγές τους δεν είναι επικερδής. Είναι δύσκολο να ξεκινήσει μια νέα εταιρεία λόγω της υψηλής αρχικής επένδυσης που απαιτείται. Παρ 'όλα αυτά, η ανάπτυξη casual παιχνιδιών (ιδιαίτερα για κινητά τηλέφωνα), επέτρεψε σε μικρότερες ομάδες να εισέλθουν στην αγορά. Μόλις οι επιχειρήσεις σταθεροποιηθούν οικονομικά, μπορεί να επεκταθούν και να πάρουν το ρίσκο να αναπτύξουν μεγαλύτερης κλίμακας παιχνίδια.

Η ανάπτυξη ηλεκτρονικών παιχνιδιών (Game Development) είναι μια διαδικασία που ξεκινά από μια ιδέα ή έννοια. Συχνά, η ιδέα βασίζεται σε μια τροποποίηση κάποιου ήδη υπάρχοντος concept παιχνιδιού. Η ιδέα του παιχνιδιού μπορεί να εμπίπτει ανάμεσα σε ένα ή περισσότερα είδη. Οι σχεδιαστές συχνά πειραματίζονται με διαφορετικούς συνδυασμούς παιχνιδιών. Στην συνέχεια ο game designer παράγει ένα πρώτο αρχείο προτάσεων, το οποίο περιέχει το concept, το gameplay, την λίστα των χαρακτηριστικών του παιχνιδιού, την τοποθεσία και την ιστορία, το κοινό που στοχεύει, τις απαιτήσεις και το budget και πολλά άλλα. Η κάθε εταιρεία έχει την δική της φιλοσοφία πάνω στο στάδιο αυτό αλλά υπάρχουν πολλές κοινές πρακτικές.

Η ανάπτυξη ξεκινάει από τον Game developer, που μπορεί να είναι από ένα άτομο μέχρι μια μεγάλη εταιρεία. Οι ανεξάρτητοι developers χρειάζονται την οικονομική στήριξη κάποιου μεγάλου εκδότη . Συνήθως, πρέπει να αναπτύξουν κάποιου είδους πρωτότυπο χωρίς καμία χρηματοδότηση μέχρι να το παρουσιάσουν στον publisher.

Κατασκευαστές κονσολών, όπως η Microsoft , Nintendo , ή η Sony , έχουν ένα τυποποιημένο σύνολο τεχνικών προδιαγραφών, με το οποίο το παιχνίδι προς παραγωγή πρέπει να συμφωνεί για να πάρει έγκριση. Επιπλέον, το concept του παιχνιδιού πρέπει να εγκριθεί και από τον κατασκευαστή ο οποίος μπορεί και να αρνηθεί να εγκρίνει κάποιον τίτλο.

Τα έσοδα από τις πωλήσεις του ηλεκτρονικού παιχνιδιού μοιράζονται κατά μήκος της αλυσίδας παραγωγής του παιχνιδιού αυτού, όπως ο developer, ο εκδότης, τα καταστήματα που θα πουλήσουν το παιχνίδι κτλ. Δυστυχώς, πολλοί developers δεν καταφέρνουν να επωφεληθούν από αυτή τη διαδικασία και πτωχεύουν.

2.1 Διαδικασία ανάπτυξης ηλεκτρονικού παιχνιδιού

Η Ανάπτυξη εποπτεύεται από εσωτερικούς και εξωτερικούς παραγωγούς (Producers). Ο παραγωγός που εργάζεται για τον developer είναι γνωστός ως ο εσωτερικός παραγωγός και διαχειρίζεται την ομάδα ανάπτυξης, ωράρια, τις εκθέσεις προόδου, προσλαμβάνει και κατανέμει το προσωπικό, και ούτω καθεξής. Ο παραγωγός που εργάζεται για τον εκδότη είναι γνωστός ως εξωτερικός παραγωγός και επιβλέπει την πρόοδο του έργου και τον προϋπολογισμό. Οι ευθύνες του παραγωγού περιλαμβάνουν δημόσιες σχέσεις , τη διαπραγμάτευση των συμβάσεων, επαφές μεταξύ του προσωπικού και των ενδιαφερομένων μερών, το χρονοδιάγραμμα και τη συντήρηση του προϋπολογισμού, τη διασφάλιση της ποιότητας , διαχείριση του beta testing, καθώς και το localization. Αυτός ο ρόλος μπορεί επίσης να αναφέρεται ως project manager, project lead, ή διευθυντής (director).

Οι προγραμματιστές μπορούν να κυμαίνονται σε μέγεθος από μικρές ομάδες που αναπτύσουν casual games μέχρι εκατοντάδες επαγγελματίες που αναπτύσσουν πολύπλοκα παιχνίδια εκατομμυρίων. Οι εταιρείες χωρίζουν τις δευτερεύουσες εργασίες κατά την ανάπτυξη του παιχνιδιού. Η ομάδα ανάπτυξης αποτελείται από πολλά μέλη. Ορισμένα μέλη της ομάδας μπορεί να έχουν περισσότερους από έναν ρόλο, όπως και περισσότερες από μια εργασίες μπορούν να ανατεθούν σε ένα άτομο. Το μέγεθος της ομάδας μπορεί να ποικίλλει, 20 με 100 ή και περισσότερα μέλη, ανάλογα με το μέγεθος του παιχνιδιού. Οι πιο αντιπροσωπευτικές ομάδες είναι οι καλλιτέχνες, , προγραμματιστές, σχεδιαστές, στη συνέχεια, και, τέλος, εδικούς για τον ήχο, με δύο έως τρεις παραγωγούς στον τομέα της διαχείρισης. Αυτές οι θέσεις απασχολούν άτομα με πλήρες ωράριο. Άλλες θέσεις, όπως δοκιμαστές, παρέχουν μερική απασχόληση.

Ένας σχεδιαστής (game designer) είναι ένα πρόσωπο που σχεδιάζει το gameplay , την σύλληψη και τον σχεδιασμό των κανόνων και τη δομή του παιχνιδιού. Οι ομάδες ανάπτυξης έχουν συνήθως έναν επικεφαλής σχεδιαστή ο οποίος συντονίζει το έργο των άλλων σχεδιαστών. Αυτός είναι ο κύριος οραματιστής του παιχνιδιού. Ένας από τους ρόλους του σχεδιαστή είναι να είναι ο συγγραφέας, που πρέπει να συλλάβει την αφήγηση του παιχνιδιού, τον διάλογο, τα σχόλια, τα cutscene , κλπ. Σε μεγάλα έργα, υπάρχουν συχνά ξεχωριστοί σχεδιαστές για διάφορα μέρη του παιχνιδιού, όπως, οι μηχανισμοί , το user interface , χαρακτήρες , διάλογος , κλπ.

Ένας καλλιτέχνης παιχνιδιών (game artist) είναι ο εικαστικός καλλιτέχνης που δημιουργεί την τέχνη και το concept art του παιχνιδιού. Εποπτεύεται από έναν καλλιτεχνικό διευθυντή ή art lead, που φροντίζει να ακολουθείται η καλλιτεχνική κατεύθυνση του παιχνιδιού . Ο καλλιτεχνικός διευθυντής διαχειρίζεται την ομάδα τέχνης, τον προγραμματισμό και τον συντονισμό εντός της ομάδας ανάπτυξης.

Η δουλειά του καλλιτέχνη μπορεί να έχει δισδιάστατο προσανατολισμό ή τρισδιάστατο προσανατολισμό. Καλλιτέχνες που εργάζονται σε δισδιάστατα παιχνίδια μπορούν να παράγουν concept art,sprites, υφές , περιβαλλοντικά σκηνικά και user interfaces. 3D καλλιτέχνες μπορούν να παράγουν μοντέλα, animation, τρισδιάστατα περιβάλλοντα και κινηματογραφικά cut scenes.

Ένας προγραμματιστής παιχνιδιού είναι ένας μηχανικός λογισμικού που αναπτύσσει κυρίως ηλεκτρονικά παιχνίδια ή σχετικό λογισμικό (όπως εργαλεία ανάπτυξης παιχνιδιών). Υπάρχουν συνήθως ένας έως μερικοί lead προγραμματιστές , που υλοποιούν το αρχικό codebase του παιχνιδιού και αναλαμβάνουν την επισκόπηση για την μελλοντική ανάπτυξη και την κατανομή προγραμματιστών σε επιμέρους ενότητες.

Οι μηχανικοί ήχου είναι επαγγελματίες τεχνικοί υπεύθυνοι για ηχητικά εφέ και την τοποθέτηση του ήχου. Πολλές φορές επιβλέπουν το voice acting και την δημιουργία των sound assets. Οι συνθέτες που δημιουργούν την μουσική επένδυση του παιχνιδιού συνοδεύονται και αυτοί από μια ομάδα μηχανικών ήχου, αν και συχνά το έργο αυτό ανατίθεται σε εξωτερικούς συνεργάτες.

Η διασφάλιση της ποιότητας του ηλεκτρονικού παιχνιδιού προς παραγωγή πραγματοποιείται από δοκιμαστές παιχνιδιών(game testers). Ένας tester αναλύει το παιχνίδι για να ανακαλύψει και να τεκμηριώσει τα ελαττώματα του λογισμικού. Το game testing είναι ένα σημαντικό κομμάτι του τεχνικού τομέα και απαιτεί υπολογιστική εμπειρία και αναλυτική ικανότητα.

Οι δοκιμαστές διασφαλίζουν ότι το παιχνίδι εμπίπτει στο προτεινόμενο σχέδιο. Η δουλειά τους περιλαμβάνει τον έλεγχο όλων των δυνατοτήτων και μηχανισμών του παιχνιδιού, τη συμβατότητα, εντοπισμός bug , κλπ..

3. Προγράμματα Λογισμικού

Για αυτή την εργασία χρησιμοποιήθηκαν τα εξής προγράμματα:

- Google Sketchup
- Autodesk 3d studio max
- Blender
- Autodesk Motionbuilder
- Pixologic ZBrush

3.1 Google Sketchup

Το Sketchup της Google είναι ένα πρόγραμμα για 3d modeling που χρησιμοποιείται για αρχιτεκτονικό, μηχανικό, πολιτικό μηχανικό σχέδιο όπως και για ταινίες και video game design. Το βασικό χαρακτηριστικό του είναι η ευκολία χρήσης του, καθώς παρέχει ένα καθαρό Interface και απλά εργαλεία κατασκευής τρισδιάστατων αντικειμένων.

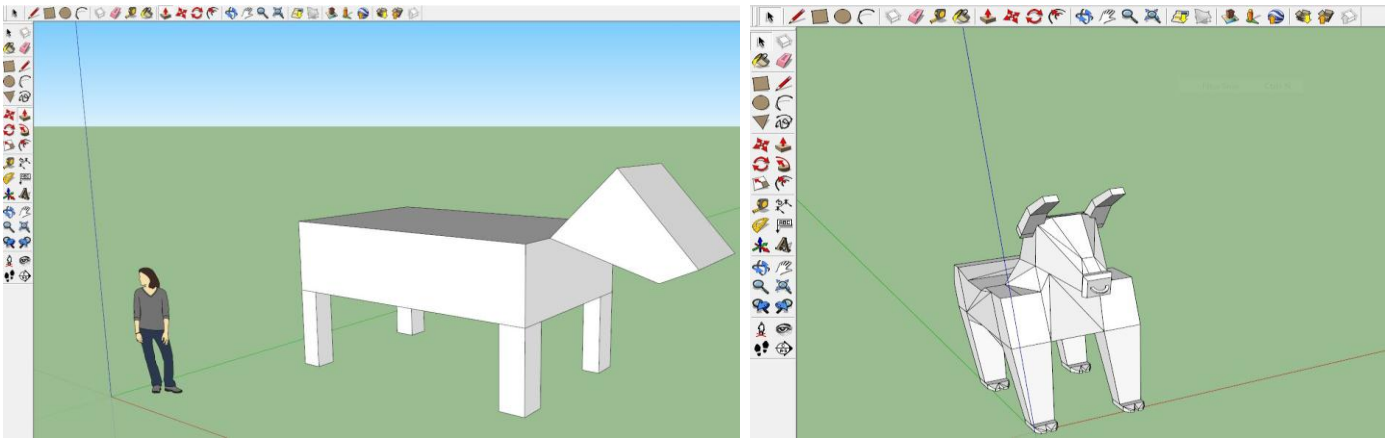
Χρησιμοποιώντας τρία βασικά primitive shapes, το παραλληλόγραμμο, τον κύκλο και το τρίγωνο δημιουργούμε την πρώτη βάση για το αντικείμενο που θέλουμε να φτιάξουμε. Επιλέγουμε το εργαλείο για να σχεδιάσουμε το παραλληλόγραμμο και το σχεδιάζουμε με βάση τους άξονες.

Το επόμενο βήμα είναι να δημιουργήσουμε ένα τρισδιάστατο κύβο από το παραλληλόγραμμο που μόλις φτιάξαμε. Εδώ φαίνεται η ευκολία χρήσης του Sketchup σε σχέση με προγράμματα για 3d modeling όπως το 3ds max. Επειδή ο σχεδιαστής δεν ασχολείται με τα πολύγωνα του μοντέλου του και την τοπολογία του χρησιμοποιεί το απλό εργαλείο push/pull για φέρει το απλό primitive στην τρίτη διάσταση.

Επιλέγοντας το εργαλείο push/pull και τραβώντας την πλευρά που επιθυμούμε, δημιουργούμε το τρισδιάστατο σχήμα που θέλουμε. Στο νέο μοντέλο που δημιουργήθηκε μπορούμε, αν το επιθυμούμε, να τραβήξουμε μια από τις υπάρχουσες πλευρές για να το επεκτείνουμε ή να το μικρύνουμε. Συνεχίζοντας με αυτήν την λογική, φτιάχνουμε τις τρισδιάστατες μορφές που θα αποτελούν το αντικείμενο μας και τις ενώνουμε για να πάρει την πρωταρχική του μορφή.

Άλλα εργαλεία που χρησιμοποιούμε για να δώσουμε λεπτομέρεια, είναι το rotate για να περιστρέφουμε τα αντικείμενα στους άξονες που θέλουμε, το make group για να επεξεργαζόμαστε το αντικείμενο σαν σύνολο, το scale για να αυξομειώνουμε το μέγεθος των αντικειμένων και το move tool για να τραβάμε πλευρές και να μετακινούμε αντικείμενα.

Επαναλαμβάνοντας την διαδικασία αυτή για κάθε ξεχωριστό μέλος του μοντέλου μας θα πάρουμε το τελικό αποτέλεσμα. Εδώ απαιτείται προσοχή όλα τα εξωτερικά faces να είναι front faces και όχι back faces για να μην υπάρξει μετά πρόβλημα με τα materials. Επίσης κατά την εξαγωγή του αντικειμένου, επιλέγουμε το fbx format και προσέχουμε, στις επιλογές εξαγωγής, να οι πλευρές να είναι διπλής όψης, και τα τελικά πολύγωνα να είναι τριγωνικά έτσι ώστε το μοντέλο μας να είναι όσο πιο συμβατό γίνεται με τις απαιτήσεις του Unity 3d.



Εικόνα1- Δημιουργία βασικού κορμού τρισδιάστατου αντικειμένου και τελικό αποτέλεσμα μοντελοποίησης στο sketchup

3.2 Pixologic ZBrush

Το ZBrush της Pixologic είναι ένα εργαλείο ψηφιακής γλυπτικής για modeling αντικειμένων. Χρησιμοποιείται για να δημιουργήσει υψηλής ανάλυσης μοντέλα (μέχρι 10εκ πολύγωνα), για περαιτέρω χρήση τους σε ταινίες, παιχνίδια και animations. Πρέπει να αναφερθεί ότι το χρησιμοποιούν μεγάλες εταιρίες όπως η Electronic Arts, ένας από τους μεγαλύτερους developers και publisher στον χώρο των video games και η Industrial Light&Magic, η εταιρεία ειδικών εφέ του George Lucas.

Στο Zbrush εισάγουμε τα μοντέλα που φτιάχνουμε στο Sketchup για να τα εξομαλύνουμε και να τους δώσουμε μια πιο smoothed επιφάνεια. Το πρόγραμμα παρέχει μια μεγάλη γκάμα τρισδιάστατων πινέλων γλυπτικής που έχουν διάφορες ιδιότητες, όπως σχήμα κατά την γλυπτική, δύναμη και διαφάνεια. Παρακάτω ακολουθεί μια εικόνα του interface του ZBrush, το οποίο δεν είναι το πιο εύχρηστο γραφικό περιβάλλον, ειδικά για έναν αρχάριο χρήστη και θέλει λίγη εκμάθηση για να χρησιμοποιηθεί.

Το πρώτο βήμα είναι να εισάγουμε το αντικείμενο από το Sketchup, το οποίο πρέπει να είναι σε μορφή object. Αυτό το επιτυγχάνουμε πατώντας import από το μενού στα δεξιά. Μετά διαλέγουμε το αρχείο με το οποίο θα δουλέψουμε. Μια προεπισκόπηση του τρισδιάστατου αντικειμένου μας εμφανίζεται στο δεξί μέρος. Πατώντας τη, κάνουμε κλικ στην κεντρική οθόνη, όπου θα εργαστούμε, και σέρνουμε τον κέρσορα για να αρχικοποιήσουμε το αντικείμενο. Εδώ είναι πολύ σημαντικό να προσέξουμε μια λαπτομέρεια. Επειδή το ZBrush δουλεύει όχι μόνο με τρισδιάστατα μοντέλα, αλλά χρησιμοποιείται και για 2.5d, πρέπει να μπούμε σε edit mode πατώντας το ‘T’ στο πληκτρολόγιο.

Έχοντας το μοντέλο μας πλέον έτοιμο για επεξεργασία, επιλέγουμε ένα από τα πολλά πινέλα που παρέχει το ZBrush, πειράζουμε τις ρυθμίσεις για το intensity και ξεκινάμε να ομαλοποιούμε το μοντέλο προσθέτοντας και αφαιρώντας λεπτομέρεια όπου χρειάζεται. Σε κάποια μοντέλα βολεύει να χρησιμοποιούμε την επιλογή της συμμετρίας για να γίνονται ομοιόμορφα οι αλλαγές. Επίσης μια ακόμα σημαντική λειτουργία είναι το smoothing που ενεργοποιείται κρατώντας πατημένο το ‘Shift’ κάνοντας κλικ με τον κέρσορα στα σημεία που θέλουμε.

Όταν είμαστε ευχαριστημένοι με το αποτέλεσμα διαλέγουμε την ανάλυση που θέλουμε να έχει το unified skin που θα κάνουμε export καθώς και την τιμή του smooth που θα του δώσουμε. Αυτό το μενού επιλογών βρίσκεται στο κάτω μέρος του δεξιού menu bar. Όταν κάνουμε τις ρυθμίσεις πατάμε make unified skin και το αντικείμενο είναι έτοιμο για εξαγωγή.

Το τελικό αποτέλεσμα βλέπουμε πως είναι πολύ πιο λεπτομερές άρα και πιο κοντά στο αντικείμενο που θέλουμε να αποδώσουμε στον τρισδιάστατο κόσμο. Οι δυνατότητες του ZBrush είναι πολλές και μπορεί να προσφέρει μεγάλη ευελιξία στον χρήστη για να δημιουργήσει πολύ εντυπωσιακά τρισδιάστατα μοντέλα, με άφθονη λεπτομέρεια. Όπως επισημάναμε και πιο πάνω, η μόνη αρνητική πτυχή του προγράμματος είναι το interface του το οποίο είναι αρκετά δύστροπο.

Εικόνα2-Τελικό αποτέλεσμα του sculpting στο ZBrush

3.3 Autodesk 3d Studio Max

Το 3ds Max είναι ίσως ένα από τα πιο γνωστά και αναγνωρισμένα επαγγελματικά προγράμματα στον κλάδο του 3d modeling. Χρησιμοποιείται ευρύτατα από developers ηλεκτρονικών παιχνιδιών και μεγάλα studio του Hollywood για να καλύψουν τις ανάγκες τους όσον αφορά την δημιουργία τρισδιάστατων μοντέλων, animation και γενικά ότι μπορεί να χρειαστούν στην ενδιάμεση παραγωγή αυτών.

Η λογική πίσω από τον σχεδιασμό τρισδιάστατων μοντέλων στο 3ds Max είναι αυτή του box modeling. Αυτός είναι ο de facto τρόπος σχεδιασμού και ο πιο αποδοτικός, εν αντιθέσει με το sketchup, γιατί κάθε πολύγωνο του μοντέλου αντιμετωπίζεται ξεχωριστά. Αυτό επιτρέπει μεγάλη ευλυγισία στον modeler, ο οποίος μπορεί να βελτιστοποιήσει στο έπακρο το μοντέλο του καταφέρνοντας έτσι να έχει μια τέλεια τοπολογία. Αυτό είναι ιδιαίτερα σημαντικό για τα υπόλοιπα στάδια δημιουργίας ενός χαρακτήρα, όπως το skinning.

Η δημιουργία ενός τρισδιάστατου, ανθρώπινου, μοντέλου στο 3d studio max είναι μια διαδικασία η οποία ορίζεται κυρίως από το πόσο έμπειρος είναι κάποιος στην εκτέλεση αυτής. Δηλαδή δεν υπάρχει κάποιος de facto τρόπος δουλειάς αλλά περισσότερο κάποιες συνήθεις πρακτικές οι οποίες διευκολύνουν τον designer. Εδώ θα αναλύσουμε κάποια βασικά βήματα για την δημιουργία ενός τρισδιάστατου μοντέλου ανθρώπου.

Το πρώτο βήμα είναι να βρούμε μια εικόνα ενδεικτική ενός ανθρώπινου σώματος σε t pose. Αυτή η εικόνα ονομάζεται reference image. Θα χρειαστούμε μια που να απεικονίζει τον χαρακτήρα en face, και μια προφίλ. Αυτές οι εικόνες θα μας βοηθήσουν να μοντελοποιήσουμε βήμα βήμα το μοντέλο μας. Μόλις βρούμε τις απαραίτητες εικόνες πρέπει να δημιουργήσουμε δύο planes για να τις ορίσουμε σαν textures και να μπορούμε να δουλέψουμε.

Έχοντας έτοιμα τα planes το μόνο που μένει είναι να ορίσουμε στο καθένα την ανάλογη εικόνα. Αυτό γίνεται ανοίγοντας τον material editor του 3d studio max. Επιλέγοντας κάποιο material μπορούμε να αναθέσουμε ένα texture είτε ψάχνοντας το μέσα από το εργαλείο είτε κάνοντας drag and drop την εικόνα απευθείας.

Σε αυτό το σημείο περνάμε στον βασικό κορμό του character modeling και στην κύρια ιδέα που κρύβεται από πίσω του. Χρησιμοποιώντας βασικά γεωμετρικά σχήματα τα οποία θα τροποποιήσουμε στην πορεία κατάλληλα ώστε να ταυτιστούν με τα reference images, θα χτίσουμε σιγά σιγά τον χαρακτήρα μας. Ξεκινώντας από τον κορμό θα δημιουργήσουμε έναν κύβο τον οποίο θα τοποθετήσουμε κατάλληλα εκμεταλλευόμενοι τα πολλαπλά viewports.

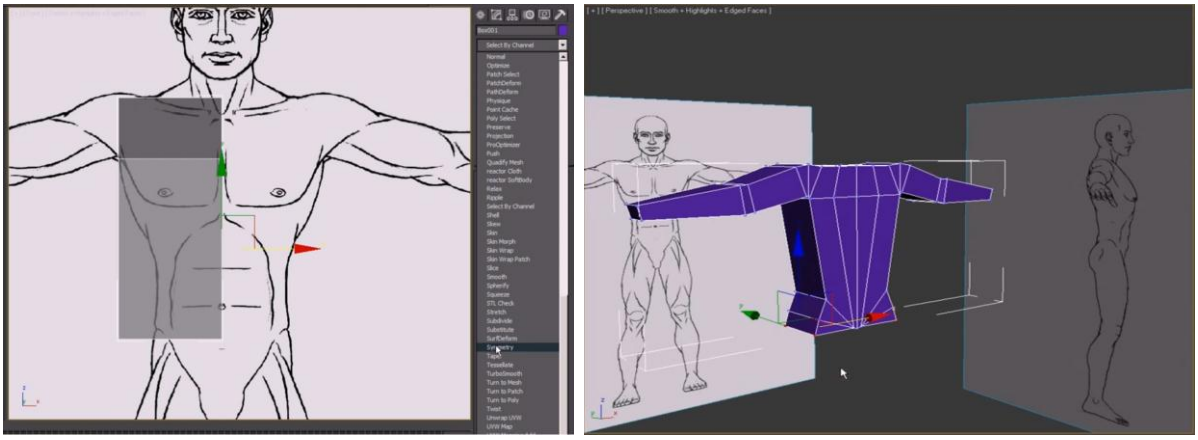
Τώρα θα πρέπει να αρχίσουμε να μεταβάλουμε τον κύβο ώστε να αρχίσει να ταυτίζεται με τα reference images. Πριν ξεκινήσουμε αυτήν τη διαδικασία είναι σημαντικό να τον προετοιμάσουμε πρώτα κάνοντας κάποιες απαραίτητες ενέργειες που θα μας διευκολύνουν στην σχεδίαση του μοντέλου.

- Επειδή δεν θέλουμε ο κύβος να κρύβει το reference image πατώντας τον συνδυασμό alt+x θα κάνει το αντικείμενο διάφανο.
- Η βασικότερη κίνηση είναι να μετατρέψουμε τον κύβο σε convertible polygon. Αυτό σημαίνει ότι θα μπορούμε να διαχειριστούμε της ακμές, τις πλευρές και τα πολύγωνα του όπως εμείς θέλουμε. Αυτό γίνεται κάνοντας δεξί κλικ στο αντικείμενο και επιλέγοντας convert to editable poly.
- Ο κύβος που δημιουργήσαμε είναι ένα καλό αρχικό σχήμα αλλά κατά την διάρκεια της μοντελοποίησης θα χρειαστούμε αρκετές φορές να προσθέσουμε νέες πλευρές για να ακολουθήσουμε το reference image. Από το νέο toolbar που έχει εμφανιστεί διαλέγουμε την επιλογή για τον χειρισμό πολυγώνων και πατάμε extrude. Πλέον έχουμε ένα νέο πολύφωνο το οποίο θα φέρουμε στο σημείο που επιθυμούμε.

Εδώ πρέπει να προσέξουμε μια πολύ βασική λεπτομέρεια. Καθώς δουλεύουμε το μοντέλο δεν θέλουμε να προσαρμόζουμε την μια μεριά και μετά την άλλη. Η καλύτερη λύση (μιας και το σχήμα μας είναι συμμετρικό) να κρατήσουμε την μια μεριά του πολύγωνου και ότι αλλαγές γίνονται σε αυτήν να εφαρμόζονται και από την άλλη. Για να το καταφέρουμε αυτό θα πρέπει να κόψουμε το αντικείμενο στην μέση. Αρχικά θα κάνουμε μια τομή στην μέση του πολυγώνου επιλέγοντας μια ακμή του πολυγώνου και πατώντας ring. Στην συνέχεια επιλέγοντας connect βλέπουμε ότι εμφανίζεται μια

κάθετη γραμμή που χωρίζει το αντικείμενο στην μέση. Τώρα μπορούμε να διαγράψουμε την δεξιά μερία.

Εδώ φτάνουμε στους modifiers του 3d studio max. Συγκεκριμένα θα ασχοληθούμε με τον symmetry modifier. Αυτός βρίσκεται στην λίστα με τους modifiers όπως απεικονίζεται στην εικόνα 16. Αμέσως μετά επιλέγουμε flip. Έτσι αποκτάμε πλέον ένα συμμετρικό σχήμα το οποίο θα μεταβάλλεται και από τις δύο μεριές με τον ίδιο τρόπο.



Εικόνα3-Symmetry modifier και βασική δημιουργία κορμού

Σε αυτό το σημείο μπορούμε να ξεκινήσουμε να ταιριάζουμε τον κύβο με το reference image χρησιμοποιώντας τις ακμές του. Στο selection menu επιλέγουμε το submenu των ακμών (vertices). Χρησιμοποιώντας τον κέρσορα και επιλέγοντας τις ακμές που θέλουμε μπορούμε να μεταβάλλουμε τον κύβο όπως επιθυμούμε. Χρειάζεται προσοχή όμως, να μην επιλέξουμε μια μεμονωμένη ακμή γιατί αυτές που βρίσκονται από πίσω της δεν θα αλλάξουν θέση.

Από εδώ και πέρα το πώς θα επιλέξουμε να κινηθούμε είναι καθαρά δική μας επιλογή. Το σίγουρο είναι ότι επιλέγοντας ακμές μορφοποιούμε το αντικείμενο, κάνοντας extrude τα πολύγωνα μπορούμε να επεκταθούμε στο reference image, με το εργαλείο ring και connect δημιουργούμε καινούργιες ακμές όπου στην συνέχεια θα μπορέσουμε να μεταχειριστούμε τις κορυφές τους. Επίσης έχουμε και στην διάθεση μας το scale tool το οποίο αυξομειώνει τα πολύγωνα σε μέγεθος έτσι ώστε να ταιριάζουμε το μοντέλο μας στην εικόνα.

3.4 Blender

Το Blender είναι ένα open source πρόγραμμα για την δημιουργία τρισδιάστατων μοντέλων. Χρησιμοποιείται ευρέως για την δημιουργία ταινιών animation, ειδικών εφέ για ταινίες, ηλεκτρονικών παιχνιδιών και τρισδιάστατων εφαρμογών. Οι δυνατότητες του καλύπτουν ένα ευρύ φάσμα λειτουργιών, όπως 3d modeling, UV unwrapping, texturing, rigging, skinning και πολλά άλλα. Να σημειωθεί επίσης πως παρέχει την δικιά του game engine. Το Blender θα το χρησιμοποιήσουμε για να δούμε πως γίνεται rigged ένας χαρακτήρας και τι ακριβώς είναι το rigging.

Όταν ένα τρισδιάστατο μοντέλο δημιουργηθεί στο 3d studio max, για να μπορέσει να δεχθεί κάποιο animation, θα πρέπει να περάσει από μια διαδικασία η οποία ονομάζεται rigging. Το rigging είναι πολύ βασικό και αναγκαίο γιατί στην ουσία προσθέτει στο μοντέλο ένα είδος σκελετού (Armature) ο οποίος αποτελείται από έναν αριθμό κόκκαλων (bones). Αυτός ο ενσωματωμένος σκελετός λειτουργεί όπως ο ανθρώπινος έτσι ώστε το τρισδιάστατο μοντέλο να μπορεί να κινείται φυσιολογικά.

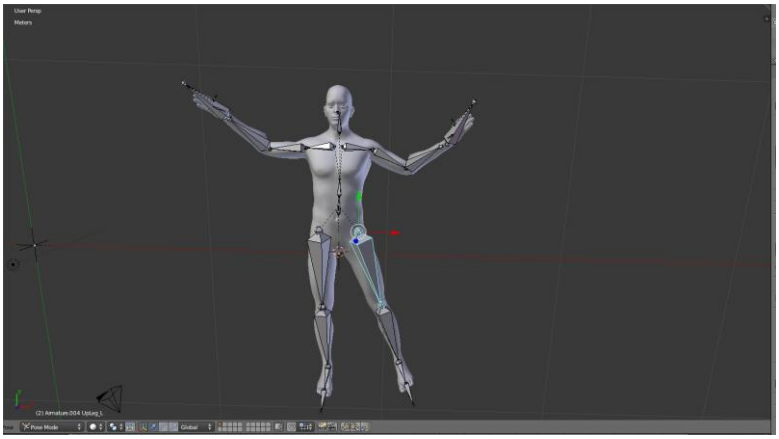
Αφού το επιθυμητό μοντέλο φορτωθεί στο Blender, η διαδικασία του rigging θα ξεκινήσει τοποθετώντας το πρώτο bone (το οποίο ονομάζεται root bone) στο μοντέλο πατώντας τον συνδυασμό πλήκτρων Shift+A. Από αυτό το πρώτο Bone θα δημιουργήσουμε και όλα τα υπόλοιπα τα οποία θα τοποθετηθούν στο υπόλοιπο σώμα.

Στην συνέχεια επιλέγοντας edit mode από το drop down menu κάτω αριστερά, θα μετακινήσουμε και θα αυξομειώσουμε το bone ώστε να τοποθετηθεί και να καλύψει την περιοχή της

σπονδυλικής στήλης. Αυτό γίνεται χρησιμοποιώντας τον άνω κύκλο του bone ο οποίος αλλάζει το μέγεθος του και τον κάτω κύκλο ο οποίος το μετακινεί στον χώρο. Επειδή δεν θέλουμε μόνο ένα Bone θα το διαιρέσουμε σε έναν αριθμό μικρότερων, κάτι το οποίο είναι καθαρά εμπειρική επιλογή. Έχοντας το επιλεγμένο πατώντας το πλήκτρο W επιλέγουμε subdivide όσες φορές θέλουμε να το διαιρέσουμε. Άλλη μια σημαντική εντολή είναι το extrude bone, το οποίο σημαίνει ότι από ένα ήδη υπάρχον θα δημιουργηθεί ένα καινούργιο. Πατώντας τον συνδυασμό E+Z, θα φτιάξουμε το bone του κεφαλιού.

Συνεχίζοντας με αυτόν τον τρόπο θα καλύψουμε όλο το σώμα του μοντέλου έχοντας σαν παράδειγμα τον ανθρώπινο σκελετό. Αυτό είναι λογικό εξάλλου αφού αυτόν προσπαθούμε να εξομοιώσουμε. Πέρα από τις τεχνικές που είδαμε μέχρι τώρα υπάρχει άλλη μια τελευταία πριν καταλήξουμε να δούμε το τελικό αποτέλεσμα. Όταν θέλουμε να βάλουμε συμμετρικά bones, αντί να κάνουμε extrude με το, πλήκτρο E, ένα τη φορά, αν το κρατήσουμε πατημένο και ταυτόχρονα πατήσουμε τον άξονα που θέλουμε να τοποθετηθεί και το μέγεθός του θα μπορούμε εύκολα να δημιουργήσουμε με τον ίδιο τρόπο το συμμετρικό σε αυτό Bone.

Το μόνο που μένει πλέον είναι το μοντέλο να γίνει παιδί του σκελετού που μόλις φτιάξαμε έτσι ώστε κάθε φορά που κινείται ένα bone, να κινείται και το αντίστοιχο γκρουπ πολυγώνων που του αντιστοιχεί. Γυρνώντας σε object mode και επιλέγοντας το μοντέλο και τον σκελετό, πατάμε ctrl-p και επιλέγουμε set automatic weights. Επιλέγοντας Pose mode μπορούμε να δούμε ακριβώς πως μεταβάλλεται το μοντέλο όταν κινούνται τα bones.



Εικόνα4-Τελικό αποτέλεσμα rigging

3.5 Motion Builder

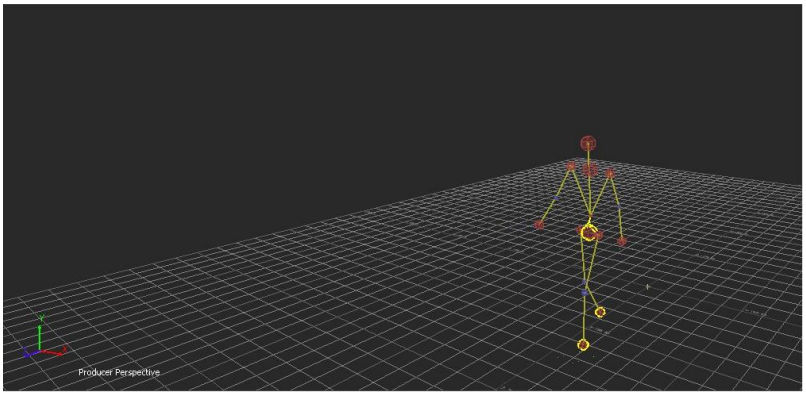
Το MotionBuilder της Autodesk είναι ένα επαγγελματικό πρόγραμμα για 3d animation. Χρησιμοποιείται ευρέως για παραγωγή ψηφιακού περιεχομένου, επαγγελματικό motion capture αλλά και για παραδοσιακό key frame animation. Χρησιμοποιείται στην παραγωγή παιχνιδιών, κινηματογραφικών ταινιών, τηλεοπτικών εκπομπών. Κάποιοι mainstream τίτλοι που έχουν επωφεληθεί από αυτό είναι τα ηλεκτρονικά παιχνίδια Assassins creed, Killzone 2 και στην ταινία Avatar. Επίσης αξίζει να αναφέρουμε ότι το πολυχρησιμοποιημένο format fbx για ανταλλαγή δεδομένων τρισδιάστατου περιεχομένου ξεκίνησε και πρωτοχρησιμοποιήθηκε από το MotionBuilder. Σε αυτό το υποκεφάλαιο θα δούμε πως γίνεται σε ένα rigged τρισδιάστατο μοντέλο να εφαρμόσουμε ένα αρχείο κίνησης τύπου bvh.

Για να εφαρμόσουμε κάποιο animation στο μοντέλο που θέλουμε πρέπει να ακολουθήσουμε την προβλεπόμενη διαδικασία που ορίζει το MotionBuilder. Σε αντίθεση με ότι έχουμε δει μέχρι τώρα το MotionBuilder έχει μια πολύ συγκεκριμένη διαδικασία που πρέπει να ακολουθηθεί. Το πρώτο πράμα που πρέπει να κάνουμε είναι να φορτώσουμε το bvh αρχείο που θέλουμε να εφαρμόσουμε στον χαρακτήρα μας στο πρόγραμμα. Με το που το κάνουμε αυτό θα δούμε τον σκελετό του animation να εμφανίζεται στο interface. Εάν πατήσουμε το κουμπί play στην μπάρα ελέγχου θα δούμε το animation εν κινήσει.

Το αρχείο που μόλις εισαγάγαμε, δεν μπορούμε να το εφαρμόσουμε απευθείας πάνω στον σκελετό του rigged μοντέλου μας. Πρέπει πρώτα να δημιουργήσουμε ένα characterized σκελετό, πάνω στον οποίο θα περάσουμε όλα τα δεδομένα του animation και στην συνέχεια θα συνδέσουμε αυτόν τον σκελετό με το μοντέλο μας ώστε να δεχτεί το animation. Άρα το πρώτο πράγμα που θα κάνουμε είναι να πούμε στο MotionBuilder πως θέλουμε να προβούμε στην δημιουργία του ειδικού σκελετού. Επιλέγοντας από το μενού κάτω δεξιά την επιλογή characters, θα εμφανιστεί ένα νέο μενού.

Στο δεξί μέλος έχουμε όλα τα μέλη του σκελετού που δεν έχει χαρακτηριστεί και επομένως, τα μέλη αυτά, δεν έχουν όνομα. Στο αριστερό μέρος κάνοντας collapse το bvh αρχείο εμφανίζονται όλα τα μέλη (χαρακτηρισμένα) του animation. Αυτό που θα πρέπει να κάνουμε για να χαρακτηριστεί ο σκελετός (control rig), είναι με drag and drop να κάνουμε τις σωστές αντιστοιχίες όπως φαίνεται στην εικόνα 21. Εδώ το βασικό που πρέπει να έχουμε στο νου μας είναι ότι πρέπει να αντιστοιχιστεί οπωσδήποτε το base. Με το που τελειώσει η αντιστοίχιση τσεκάρουμε το κουτάκι που γράφει characterize και στο νέο pop up window ενημερώνουμε το MotionBuilder ότι το control rig θα εφαρμοστεί σε biped. Το τελικό βήμα είναι να πατήσουμε create και η διαδικασία αυτή θα λάβει τέλος.

Στην οθόνη τώρα βλέπουμε το characterized control rig. Αν πατήσουμε play βέβαια θα παρατηρήσουμε πως δεν κινείται. Αυτό συμβαίνει διότι το animation πρέπει να γίνει bake στον νέο μας σκελετό. Από το menu character controls πάνω δεξιά θα πρέπει να επιλέξουμε none στο δεύτερο drop down menu. Στην συνέχεια κάνοντας κλικ στο μπλε κουμπί επιλέγουμε Bake to control rig. Με το που γίνει bake αν πατήσουμε play θα δούμε πωw το control rig έλαβε επιτυχώς τα δεδομένα του bvh αρχείου.



Εικόνα5-Το control rig

Τώρα το μόνο που μένει είναι να γίνει import το μοντέλο που θέλουμε να αποκτήσει το animation. Αυτό θα γίνει μετατρέποντας τον σκελετό του σε control rig και λέγοντας του να ακολουθεί το προηγούμενο control rig που φτιάξαμε. Όταν είμαστε ευχαριστημένοι με το αποτέλεσμα μπορούμε να το κάνουμε bake όπως και πριν και πλέον να διαγράψουμε το παλιό control rig.

3.6 Unity 3d

Το Unity 3d είναι μια από τις δημοφιλέστερες Game engines που κυκλοφορούν. Η επιτυχία της βασίζεται κυρίως στο γεγονός ότι οι δημιουργοί της είχαν σαν σκοπό να φτιάξουν μια μηχανή που θα ανταποκρινόταν στις ανάγκες των ανεξάρτητων δημιουργών ηλεκτρονικών παιχνιδιών (Independent Game Developers ή Indie developers), οι οποίο δεν μπορούσαν να φτιάξουν την δική τους game engine ή να αγοράσουν ακριβά licenses γνωστών μηχανών.

3.6.1 Game engines

Μια μηχανή παιχνιδιού είναι ένα σύστημα λογισμικού σχεδιασμένο για τη δημιουργία και την ανάπτυξη βιντεοπαιχνιδιών. Υπάρχουν πολλές μηχανές παιχνιδιών οι οποίες είναι σχεδιασμένες να δουλεύουν σε κονσόλες βιντεοπαιχνιδιών και λειτουργικά συστήματα επιτραπέζιων υπολογιστών

όπως τα Microsoft Windows, το Linux, και το Mac OS X. Η κεντρική λειτουργικότητα που παρέχεται τυπικά από μια μηχανή παιχνιδιού περιλαμβάνει μια μηχανή φωτοαπόδοσης ("renderer") για 2D ή 3D γραφικά, μια μηχανή φυσικής ή εντοπισμού συγκρούσεων (collision detection, καθώς και collision response), ήχο, scripting, animation. τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης, νήματα (threading), υποστήριξη τοπικοποίησης, και ένα γράφο σκηνής (scene graph). Η διαδικασία της ανάπτυξης παιχνιδιού συχνά οικονομικοποιείται με το ότι σε μεγάλο μέρος η ίδια μηχανή παιχνιδιού επαναχρησιμοποιείται για να δημιουργηθούν διαφορετικά παιχνίδια.

Οι μηχανές παιχνιδιών παρέχουν μια σουίτα οπτικών εργαλείων ανάπτυξης εκτός από επαναχρησιμοποιήσιμα στοιχεία λογισμικού. Αυτά τα εργαλεία γενικά παρέχονται σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης ώστε να καθιστούν ικανή την απλή, γρήγορη ανάπτυξη παιχνιδιών με ένα οδηγούμενο από δεδομένα τρόπο. Αυτές οι μηχανές παιχνιδιών συχνά καλούνται "game middleware" επειδή, όπως με την εμπορική έννοια του όρου, παρέχουν μια ευέλικτη και επαναχρησιμοποιήσιμη πλατφόρμα λογισμικού η οποία παρέχει όλη την κεντρική λειτουργικότητα που απαιτείται, αμέσως έξω από το κουτί, για την ανάπτυξη μια εφαρμογής παιχνιδιού ενώ ταυτόχρονα μειώνει τα κόστη, τις πολυπλοκότητες, και το χρόνο μέχρι την αγορά (time-to-market) — όλοι κρίσιμοι παράγοντες στην υψηλά ανταγωνιστική βιομηχανία βιντεοπαιχνιδιών.[2]

Όπως άλλες λύσεις middleware, οι μηχανές παιχνιδιών συνήθως παρέχουν μια αφαίρεση της πλατφόρμας, επιτρέποντας στο ίδιο παιχνίδι να τρέχει σε διάφορες πλατφόρμες συμπεριλαμβανομένων των κονσολών παιχνιδιών και των προσωπικών υπολογιστών με λίγες, αν όχι καθόλου, αλλαγές στον πηγαίο κώδικα του παιχνιδιού. Συχνά, το middleware παιχνιδιού σχεδιάζεται με μια βασισμένη σε στοιχεία αρχιτεκτονική η οποία επιτρέπει σε συγκεκριμένα συστήματα στη μηχανή να αντικατασταθούν ή να επεκταθούν με πιο εξειδικευμένα (και συχνά πιο ακριβά) στοιχεία middleware όπως το Havok για φυσική, το FMOD για ήχο, ή το Scaleform για UI και Βίντεο. Μερικές μηχανές παιχνιδιών όπως η RenderWare είναι και ακόμη σχεδιασμένες ως μια σειρά χαλαρά συνδεδεμένων στοιχείων middleware τα οποία μπορούν να συνδυαστούν κατά βούληση για τη δημιουργία μια προσαρμοσμένης μηχανής, αντί για την πιο κοινή προσέγγιση της επέκτασης ή της προσαρμογής μια ευέλικτης ενσωματωμένης λύσης. Με οποιοδήποτε τρόπο και αν η επεκτασιμότητα επιτυγχάνεται, παραμένει μια υψηλή προτεραιότητα στις μηχανές παιχνιδιών λόγω της ευρείας ποικιλίας χρήσεων για την οποίες αυτές εφαρμόζονται. Παρά την συγκεκριμένη έννοια του ονόματος, οι μηχανές παιχνιδιών συχνά χρησιμοποιούνται για άλλα είδη διαδραστικών εφαρμογών με πραγματικού χρόνου γραφικές απαιτήσεις όπως επιδείξεις μάρκετινγκ, αρχιτεκτονικές οπτικοποιήσεις, εκπαιδευτικές εξομοιώσεις, και περιβάλλοντα μοντελοποίησης.[3]

Μερικές μηχανές παιχνιδιών παρέχουν μόνο ικανότητες φωτοαπόδοσης 3D πραγματικού χρόνου (real time 3D rendering) αντί της ευρείας γκάμας λειτουργικότητας που απαιτείται από τα παιχνίδια. Αυτές οι μηχανές βασίζονται στον δημιουργό του παιχνιδιού να εφαρμόσει το υπόλοιπο αυτής της λειτουργικότητας ή να το συνθέσει από άλλα στοιχεία middleware παιχνιδιού. Αυτού του τύπου οι μηχανές γενικά αναφέρονται ως «μηχανή γραφικών», «μηχανή rendering», ή «μηχανή 3D» αντί για τον πιο περιληπτικό όρο «μηχανή παιχνιδιού». Ωστόσο, αυτή η ορολογία χρησιμοποιείται ανακόλουθα καθώς πολλές πλήρεις σε χαρακτηριστικά μηχανές 3D παιχνιδιών αναφέρονται απλά ως «μηχανές 3D». Μερικά παραδείγματα μηχανών γραφικών είναι οι εξής: RealmForge, Truevision3D, OGRE, Crystal Space, Genesis3D, Irrlicht και JMonkey Engine. Οι μοντέρνες μηχανές παιχνιδιών ή γραφικών γενικά παρέχουν ένα scene graph, ο οποίος είναι μια αντικειμενοστρεφής αναπαράσταση του 3D κόσμου του παιχνιδιού η οποία συχνά απλοποιεί τον σχεδιασμό του παιχνιδιού και μπορεί να χρησιμοποιηθεί για πιο αποδοτικό rendering τεράστιων εικονικών κόσμων.

Πιο συχνά, οι 3D μηχανές ή τα συστήματα rendering στις μηχανές παιχνιδιών κατασκευάζονται πάνω σε ένα API γραφικών, όπως τα Direct3D ή OpenGL, το οποίο παρέχει μια αφαιρετικότητα λογισμικού της GPU ή της κάρτας βίντεο. Βιβλιοθήκες χαμηλού επιπέδου όπως τα DirectX, SDL και OpenAL επίσης χρησιμοποιούνται συχνά σε παιχνίδια καθώς παρέχουν πρόσβαση ανεξάρτητη από το υλικό σε άλλο υλικό του υπολογιστή όπως συσκευές εισόδου (ποντίκι (υπολογιστές), πληκτρολόγιο και joystick), κάρτες δικτύου και κάρτες ήχου. Πριν από τα επιταχυνόμενα από το υλικό 3D γραφικά, είχαν χρησιμοποιηθεί renderers λογισμικού. Το rendering λογισμικού ακόμα χρησιμοποιείται σε κάποια εργαλεία μοντελοποίησης ή για εικόνες ακίνητα rendered όταν η οπτική ακρίβεια αποτιμάται πάνω από την επίδοση πραγματικού χρόνου (καρέ ανά

δευτερόλεπτο) ή όταν το υλικό του υπολογιστή δεν συναντά απαιτήσεις όπως υποστήριξη shader ή, στην περίπτωση των Windows Vista, υποστήριξη για το Direct3D 10.

Με την ανατολή της επιταχυνόμενης από το υλικό επεξεργασίας φυσικής, διάφορα API φυσικής όπως το PAL και οι επεκτάσεις φυσικής του COLLADA (μιας ανταλλακτικής μορφής για 3D στοιχεία) έγιναν διαθέσιμα για να παρέχουν μια αφαιρετικότητα λογισμικού της μονάδας επεξεργασίας φυσικής διαφορετικών παρόχων middleware και πλατφορμών κονσολών.

Η πρώτη γενιά μηχανών γραφικών τρίτων μερών ή renderers (και προγόνων αυτών που γνωρίζουμε ως μηχανές) κυριαρχούταν από τρεις παίκτες: την BRender από την Argonaut Software, την Renderware από την Criterion Software Limited και την Reality Lab της RenderMorphics. Η Reality Lab ήταν η ταχύτερη από τις τρεις και ήταν η πρώτη που αναλήφθηκε σε μια επιθετική κίνηση της Microsoft. Η ομάδα της RenderMorphics, οι Servan Keondjian, Kate Seekings και Doug Rabson, ακολούθως προσχώρησαν στο εγχείρημα της Microsoft το οποίο μετέτρεψε την Reality Lab στο Direct3D, πριν οι Keondjian και Rabson φύγουν για να ξεκινήσουν μια άλλη εταιρεία middleware Qube Software. Η Renderware τελικά αγοράστηκε από την EA αλλά τέθηκε στην άκρη από τον γίγαντα των παιχνιδιών.

Ο όρος «μηχανή παιχνιδιού» ήρθε στην επιφάνεια στα μέσα της δεκαετίας του 1990, ειδικά σε σχέση με 3D παιχνίδια όπως τα παιχνίδια βολών πρώτου προσώπου (FPS). Τόση ήταν η δημοτικότητα των παιχνιδιών Doom και Quake που, αντί να δουλέψουν από μηδενική βάση, άλλοι δημιουργοί πήραν άδεια για τα κεντρικά κομμάτια του λογισμικού και σχεδίασαν τα δικά τους γραφικά, χαρακτήρες, όπλα και επίπεδα – το «περιεχόμενο του παιχνιδιού» ή τα «στοιχεία του παιχνιδιού». Ο διαχωρισμός των ειδικών για το παιχνίδι κανόνων και δεδομένων από βασικές έννοιες όπως ο έλεγχος σύγκρουσης και οι οντότητες του παιχνιδιού σήμαινε ότι οι ομάδες μπορούσαν να μεγαλώσουν και να ειδικευτούν.

3.6.2 Overview του Unity 3d

Το Unity 3d είναι μια από τις δημοφιλέστερες Game engines που κυκλοφορούν. Η επιτυχία της βασίζεται κυρίως στο γεγονός ότι οι δημιουργοί της είχαν σαν σκοπό να φτιάξουν μια μηχανή που θα ανταποκρινόταν στις ανάγκες των ανεξάρτητων δημιουργών ηλεκτρονικών παιχνιδιών (Independent Game Developers ή Indie developers), οι οποίο δεν μπορούσαν να φτιάξουν την δική τους game engine ή να αγοράσουν ακριβά licenses γνωστών μηχανών.

Το Unity 3d παρέχει ένα καθαρό και πολύ εύχρηστο γραφικό περιβάλλον το οποίο ο χρήστης μπορεί να αναπροσαρμόσει όπως επιθυμεί. Αποτελείται από πέντε παράθυρα. Στο πρώτο, scene/game, στήνουμε τις σκηνές του παιχνιδιού μας και όταν πατάμε Play εκεί εμφανίζεται η προεπισκόπηση. Το δεύτερο παράθυρο, console, είναι αυτό στο οποίο εμφανίζονται όλα τα μηνύματα συστήματος, τα οποία μπορούμε εύκολα να διαχειριστούμε. Στο παράθυρο Hierarchy μπορούμε να δούμε όλα τα αντικείμενα που είναι τοποθετημένα στην σκηνή. Όλοι οι φάκελοι και τα αρχεία του παιχνιδιού απεικονίζονται στο παράθυρο Project. Τέλος στο παράθυρο inspector βλέπουμε λεπτομέρειες για κάθε επιλεγμένο στοιχείο.

Το Unity 3d παρέχει όλα τα εργαλεία που θα περίμενε κανείς από μια game engine. Παρέχει την δυνατότητα δημιουργίας primitives (όπως είδαμε και στο Sketchup), κατασκευή terrain για τη δημιουργία πιστών, δυνατότητες φυσικής, εισαγωγή δυναμικού φωτισμού στη σκηνή και πολλά άλλα χρήσιμα εργαλεία. Επίσης παρέχεται ο editor mono develop για συγγραφή και διαχείριση των script.

Ξεκινώντας ένα νέο project στο Unity πρέπει να έχουμε στο μυαλό μας είναι η οργάνωση των φακέλων και των αρχείων. Είναι καλή πρακτική να δημιουργούνται φάκελοι με ένα περιγραφικό όνομα για να τοποθετούνται όλα τα αρχεία. Για παράδειγμα, οι βασικοί φάκελοι πρέπει να είναι οι scenes, scripts, materials και animations. Γενικά οτιδήποτε μπορεί να ταξινομηθεί σε μια ομάδα είναι καλό να τοποθετείται σε ένα ξεχωριστό φάκελο. Οι φάκελοι που δημιουργήσαμε εμείς φαίνονται στην παρακάτω εικόνα:

Ξεκινώντας το project έπρεπε να δημιουργήσουμε το terrain στο οποίο ο παίκτης θα ψάχνει τα αρχαιολογικά αντικείμενα. Ο terrain editor του Unity 3d είναι ένα πολύ χρήσιμο και εύχρηστο εργαλείο το οποίο παρέχει πολλές δυνατότητες. Ο χρήστης μπορεί να δημιουργήσει το terrain δημιουργώντας υψώματα και λακούβες, να του αναθέσει textures και να προσθέσει πολλές λεπτομέρειες όπως δέντρα, θάμνους και γρασίδι.

Κάνοντας κλικ στο terrain στο άνω toolbar και επιλέγοντας new terrain δημιουργούμε ένα καινούργιο terrain. Πριν ξεκινήσουμε όμως πρέπει να ορίσουμε κάποιες σημαντικές λεπτομέρειες όπως, ανάλυση του terrain, μέγεθος κτλ. Πηγαίνοντας ξανά στο toolbar βλέπουμε ότι εμφανίζονται όλες οι επιλογές που πριν ήταν γκριζαρισμένες.

- Set Resolution: Στο νέο παράθυρο που ανοίγει ρυθμίζουμε την ανάλυση του terrain και των λεπτομερειών του.
- Flatten Heightmap: Εδώ μπορούμε να δώσουμε μια τιμή έτσι ώστε να μπορούμε να σκάψουμε το terrain.

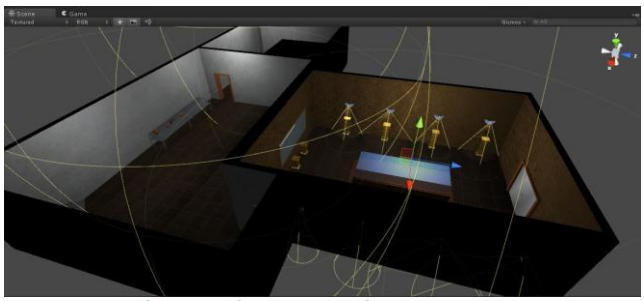
Πλέον μπορούμε να ξεκινήσουμε να δίνουμε μορφή στο terrain. Έχοντας το επιλεγμένο στο hierarchy μπορούμε να δούμε στον inspector όλα τα εργαλεία που έχουμε στην διάθεσή μας. Μπορούμε να υψώσουμε το terrain (κρατώντας πατημένο το shift μπορούμε να το σκάψουμε), να το εξομαλύνουμε κτλ. Επίσης μπορούμε να επιλέξουμε τα textures που θα χρησιμοποιήσουμε και με τα οποία θα βάψουμε το terrain.

Για το εισαγωγικό scene φτιάξαμε τα τρισδιάστατα γράμματα στο sketchup, τοποθετήσαμε και διακοσμήσαμε την περιοχή την οποία θα κοιτάει η κάμερα και εφαρμόσαμε τα image effects, στην κάμερα, που προσφέρει το Unity. Είναι μια απλή διαδικασία η οποία μπορεί να κάνει την διαφορά στην αισθητική του παιχνιδιού. Τα image effects βρίσκονται στο menu components. Αυτά που χρησιμοποιήσαμε εμείς είναι τα εξής:

- Screen Overlay
- Vignetting
- Contrast Enhance
- Bloom and Lens Flares

Έχοντας επιλέξει κάποιο εφέ, στον Inspector εμφανίζονται όλες οι ρυθμίσεις που μπορούμε να κάνουμε. Τα φίλτρα που παρέχονται είναι πολλά και με πολλές ρυθμίσεις το καθένα. Οι όποιες γνώσεις ορολογίας του Photoshop θα φανούν σίγουρα χρήσιμες.

Το επόμενο βασικό scene που έπρεπε να στήσουμε ήταν αυτό του εργαστηρίου του αρχαιολόγου. Έχοντας φτιάξει τα δωμάτια και τα διακοσμητικά αντικείμενα στο sketchup η διαδικασία του στησίματος, αφού έχουμε κάνει import τα assets, είναι πολύ απλή. Για τον φωτισμό τοποθετήσαμε spotlights στο δωμάτιο με τα εκθέματα, ένα point light σε κάθε δωμάτιο και ένα directional light για να δώσει μια πρώτη στρώση φωτισμού. Οι πηγές φωτός βρίσκονται στο toolbar στην επιλογή gameobjects. Όλες οι ρυθμίσεις για μια επιλεγμένη πηγή φωτός βρίσκονται στον Inspector.



Εικόνα6-Στήσιμο σκηνής στο Unity 3d

4.Overview του παιχνιδιού

Όπως αναφέραμε και στην εισαγωγή, το παιχνίδι είναι μια περιπέτεια τρίτου προσώπου με εκπαιδευτικά στοιχεία για να μάθει ο παίκτης μερικά πράγματα για την Μινωική Κρήτη. Παιχνίδι τρίτου προσώπου σημαίνει πως ο χρήστης χειρίζεται τον χαρακτήρα βλέποντας τον από πίσω. Κατά την διάρκεια του παιχνιδιού θα πρέπει να ψάξει τον αρχαιολογικό χώρο για να ξεθάψει μινωικά αντικείμενα και στην συνέχεια, να μπει στο εργαστήριο του για να τα καθαρίσει και να τα προβάλει στην μικρή αίθουσα εκθεμάτων.

4.1.Αρχική οθόνη και εργαστήριο

Το πρώτο πράγμα που αντικρίζει ο παίκτης ξεκινώντας το παιχνίδι είναι η αρχική οθόνη. Σε αυτήν μπορεί να επιλέξει να ξεκινήσει ένα νέο παιχνίδι ή να κλείσει την εφαρμογή. Επιλέγοντας ένα νέο παιχνίδι ο χρήστης θα αντικρίσει το εργαστήριο του αρχαιολόγου στο οποίο μπορεί να πλοηγηθεί. Ο παίκτης κινείται μπροστά με το πλήκτρο ‘W’, πίσω με το πλήκτρο ‘S’ και στρίβει δεξιά ή αριστερά κινώντας το ποντίκι στην επιθυμητή κατεύθυνση. Το εργαστήριο του παίκτη αποτελείται από δύο μέρη. Το πρώτο μέρος είναι ο εκθεσιακός χώρος ο οποίος προς το παρόν είναι κενός.

Το δεύτερο μέρος είναι το υπόλοιπο εργαστήριο στο οποίο υπάρχουν οι πάγκοι στους οποίους ο παίκτης θα αλληλεπιδράσει με τα αρχαιολογικά αντικείμενα. Προχωρώντας προς την πόρτα και πατώντας το πλήκτρο ‘Ε’ ο παίκτης μεταφέρεται στον αρχαιολογικό χώρο.

4.2.Terrain

Ο Αρχαιολογικός χώρος είναι ένα μεγάλο terrain το οποίο έχει αρκετή λεπτομέρεια. Διάσπαρτα στον χώρο ο παίκτης πρέπει να ψάξει να βρει που μπορεί να είναι θαμμένα αρχαιολογικά ευρήματα. Συνήθως τα σημεία στα οποία βρίσκεται κάτι είναι πιο εμφανή γιατί το texture και το terrain είναι λίγο διαφορετικά. Επίσης υπάρχουν και κάποιες λεπτομέρειες αρχαιολογικού χώρου. Τα πιο δύσκολα βρίσκονται σε μέρη που είναι πιο δασώδη.

Μόλις βρει κάποιο σημείο το οποίο μπορεί να έχει κάποιο αντικείμενο πατώντας το πλήκτρο ‘Ε’ ο παίκτης σκάβει το έδαφος. Πατώντας ξανά το ‘Ε’, καθώς είναι πάνω από το αντικείμενο, το συλλέγει.

4.3.Εργαστήριο

Έχοντας μαζέψει τα αντικείμενα που θέλει, ο παίκτης μπορεί να επιστρέψει στο εργαστήριο για να τα επεξεργαστεί και να περάσει στο δεύτερο βασικό μέρος του παιχνιδιού που είναι η ανάκτηση των αρχαιολογικών αντικειμένων. Αυτό το καταφέρνει πατώντας το πλήκτρο ‘Ε’. Μόλις μπει στο εργαστήριο πρέπει να κατευθυνθεί προς τους πάγκους και το scanner για να δει τα αντικείμενα που μάζεψε πως μπορεί να τα ανακτήσει και να τα αναγνωρίσει.

Ο πρώτος τρόπος ανάκτησης είναι ο καθαρισμός των αρχαιολογικών αντικειμένων από την σκόνη. Πλησιάζοντας τον πάγκο και πατώντας το πλήκτρο ‘Ε’ ο παίκτης μεταφέρεται στην σκηνή του καθαρισμού. Στην πάνω αριστερή γωνία βρίσκεται ένας αριθμός κουμπιών. Ο αριθμός αυτός καθορίζεται ανάλογα με το πόσα αντικείμενα που καθαρίζονται από την σκόνη έχει μαζέψει ο παίκτης. Πατώντας τα κουμπιά αυτά, το ανάλογο αντικείμενο εμφανίζεται στην οθόνη. Πατώντας το πλήκτρο ‘Space’ κάθε φορά το αντικείμενο καθαρίζεται αποκαλύπτοντας την καθαρή επιφάνεια. Πατώντας το πλήκτρο ‘Ε’ ο παίκτης επιστρέφει στο εργαστήριο.

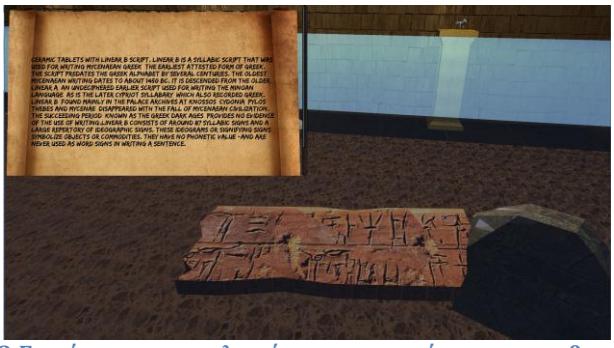
Ο δεύτερος τρόπος ανάκτησης είναι ο καθαρισμός αντικειμένων από την λάσπη. Ενεργοποιώντας τον πάγκο ο παίκτης μεταφέρεται στην ανάλογη σκηνή. Εδώ εμφανίζονται πάλι τα κουμπιά που αναλογούν στα αντικείμενα που έχουν λάσπη και μπορούν να καθαριστούν. Πατώντας όποιο κουμπί θέλει ο παίκτης, το αντικείμενο εμφανίζεται στην οθόνη. Με το δεξί κλικ του ποντικιού και κινώντας στην επιθυμητή κατεύθυνση το αντικείμενο κάνει rotate για να μπορεί να φανεί ποια σημεία του ,εσωτερικά ή εξωτερικά, μπορούν να καθαριστούν. Κάνοντας αριστερό κλικ με το ποντίκι στα σημεία που υπάρχει λάσπη αυτή καθαρίζεται κάθε φορά.

Ο τρίτος τρόπος ανάκτησης είναι χρησιμοποιώντας το scanner του εργαστηρίου. Στην σκηνή αυτή και πάλι ο παίκτης θα επιλέξει τα αντικείμενα που συνέλεξε μέσω των κουμπιών στην πάνω αριστερή γωνία της οθόνης. Επιλέγοντας τα, αυτά εμφανίζονται στην αρχή του scanner και κυλάνε προς την οθόνη όπου θα εμφανιστεί το περιεχόμενο του πετρώματος.

Ο τελευταίος τρόπος ανάκτησης αρχαιολογικών αντικειμένων είναι ένα μικρό mini game που πρέπει να παίξει ο παίκτης. Επιλέγοντας το αντικείμενο που θέλει μέσω των κουμπιών στην άνω αριστερή γωνία της οθόνης, αυτό εμφανίζεται σπασμένο σε κομμάτια. Σκοπός του είναι να ενώσει αυτά τα κομμάτια για να ξαναγίνει το αντικείμενο ενιαίο. Ένα από τα κομμάτια παραμένει σταθερό στη θέση του και πρέπει να σύρει τα υπόλοιπα στις σωστές θέσεις.

Εικόνα7-Μηχανισμοί καθαρισμού αντικειμένων

Κάθε φορά που ένα αντικείμενο καθαρίζεται ή ανακτάται, αυτό τοποθετείται στην μικρή εκθεσιακή αίθουσα. Εδώ ο παίκτης περνάει στο τρίτο μέρος του παιχνιδιού, που είναι το εκπαιδευτικό κομμάτι. Τα ανακτημένα αντικείμενα τοποθετούνται στα βάθρα που βρίσκονται περιμετρικά της αίθουσας ή στο κεντρικό μαρμάρινο βάθρο. Αν ο παίκτης επιθυμεί να μάθει κάποια πράγματα για τα αρχαιολογικά αντικείμενα που ξέθαψε και καθάρισε, αρκεί να πλησιάσει τα βάθρα. Κάνοντας το αυτό η κάμερα τοποθετείται σε σημείο τέτοιο ώστε να προβάλει μόνο τα αντικείμενα. Επίσης στην πάνω αριστερή γωνία εμφανίζεται ένα μικρό κείμενο με σχετικές πληροφορίες.



Εικόνα8-Εμφάνιση αναστυλωμένου αντικειμένου στον εκθεσιακό χώρο

5.Ανάλυση μηχανισμών και κώδικας

Η γλώσσα προγραμματισμού JavaScript δημιουργήθηκε αρχικά από τον Brendan Eich της εταιρείας Netscape με την επωνυμία Mocha. Αργότερα, Mocha μετονομάστηκε σε LiveScript, και τελικά σε JavaScript, κυρίως επειδή η ανάπτυξή της επηρεάστηκε περισσότερο από τη γλώσσα προγραμματισμού Java. LiveScript ήταν το επίσημο όνομα της γλώσσας όταν για πρώτη φορά κυκλοφόρησε στην αγορά σε βήτα (beta) εκδόσεις με το πρόγραμμα περιήγησης στο Web, Netscape Navigator εκδοχή 2.0 τον Σεπτέμβριο του 1995. LiveScript μετονομάστηκε σε JavaScript σε μια κοινή ανακοίνωση με την εταιρεία Sun Microsystems στις 4 Δεκεμβρίου, 1995, όταν επεκτάθηκε στην έκδοση του προγράμματος περιήγησης στο Web, Netscape εκδοχή 2.0B3.

Η JavaScript απέκτησε μεγάλη επιτυχία ως γλώσσα στην πλευρά του πελάτη (client-side) για εκτέλεση κώδικα σε ιστοσελίδες, και περιλήφθηκε σε διάφορα πρόγραμματα περιήγησης στο Web. Κατά συνέπεια, η εταιρεία Microsoft ονόμασε την εφάρμογή της σε JScript για να αποφύγει δύσκολα θέματα εμπορικών σημάτων. JScript πρόσθεσε νέους μεθόδους για να διορθώσει τα Y2K-προβλήματα στην JavaScript, οι οποίοι βασίστηκαν στην Java.util.Date τάξη της Java. JScript περιλήφθηκε στο πρόγραμμα Internet Explorer εκδοχή 3.0, το οποίο κυκλοφόρησε τον Αύγουστο του 1996.

Τον Νοέμβριο του 1996, η Netscape ανακοίνωσε ότι είχε υποβάλει τη γλώσσα JavaScript στο Ecma International (μια οργάνωση της τυποποίησης των γλωσσών προγραμματισμού) για εξέταση ως βιομηχανικό πρότυπο, και στη συνέχεια το έργο είχε ως αποτέλεσμα την τυποποιημένη μορφή που ονομάζεται ECMAScript.

Η JavaScript έχει γίνει μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού ηλεκτρονικών υπολογιστών στον Παγκόσμιο Ιστό (Web). Αρχικά, όμως, πολλοί επαγγελματίες προγραμματιστές υποτίμησαν τη γλώσσα διότι το κοινό της ήταν ερασιτέχνες συγγραφείς ιστοσελίδων και όχι επαγγελματίες προγραμματιστές (και μεταξύ άλλων λόγων). Με με τη χρήση της τεχνολογίας Ajax, η JavaScript γλώσσα επέστρεψε στο προσκήνιο και έφερε πιο επαγγελματική προσοχή προγραμματισμού. Το αποτέλεσμα ήταν ένα καινοτόμο αντίκτυπο στην εξάπλωση των πλαισίων και των βιβλιοθηκών, τη βελτίωση προγραμματισμού με JavaScript, καθώς και αυξημένη χρήση της JavaScript έξω από τα προγράμματα περιήγησης στο Web.

Τον Ιανουάριο του 2009, το έργο CommonJS ιδρύθηκε με στόχο τον καθορισμό ενός κοινού προτύπου βιβλιοθήκης κυρίως για την ανάπτυξη της JavaScript έξω από το πρόγραμμα περιήγησης και μέσα σε άλλες τεχνολογίες (π.χ. server-side).

5.1 Μηχανισμός Κίνησης Χαρακτήρα

Για να δημιουργήσουμε τον δικό μας custom character controller έπρεπε να ορίσουμε κάποιες μεταβλητές οι οποίες θα είχαν νόημα σε περίπτωση που τις μεταβάλαμε θα μεταβάλλονταν και οι ιδιότητες της κίνησης του χαρακτήρα μέσα στο παιχνίδι. Στον παρακάτω πίνακα βλέπουμε ποιές είναι οι μεταβλητές αυτές.

```
private var charController:CharacterController;
var walkSpeed:float = 6;
var runSpeed:float = 40;
var walkAnimSpeed:float =1;
var rotationSpeed:float = 80;
var gravityFall:float = 1;
var gravity:float = 2;
var walking: AudioClip;
```

Πίνακας 1 - Δηλώσεις μεταβλητών

- walkSpeed: Υποδεικνύει την ταχύτητα με την οποία θα περπατάει ο χαρακτήρας
- runSpeed: Υποδεικνύει την ταχύτητα με την οποία θα τρέχει ο χαεακτήρας.
- walkAnimSpeed: Η ταχύτητα με την οποία θα παίζουν τα animations.
- rotationSpeed: Η ταχύτητα με την οποία θα περιστρέφεται ο χαρακτήρας.

- gravityFall:
- gravity: Πόσο δυνατή θα είναι η δύναμη της βαρύτητας που ασκείται στον χαρακτήρα.
- charController: Σε αυτή τη μεταβλητή θα αναθέσουμε τον character controller.

Στον πίνακα 2 βλέπουμε πως γίνεται η ανάθεση του character controller στην μεταβλητή που ορίσαμε. Στον κώδικά μας πλέον θα αναφερόμαστε στον χαρακτήρα μέσω αυτής της μεταβλητής. Όπως θα δούμε και παρακάτω μέσω της μεταβλητής αυτής, που περιέχει τον controller καλούμε όλες τις μεθόδους που χρειάζονται για να τον κάνουμε να κινείται. Η μέθοδος Start(), όπως υποδικνύει και το όνομά της καλείται στην φόρτωση της σκηνής και είναι κατάλληλη για αρχικοποίηση μεταβλητών και αντικειμένων.

```
function Start()
{
    charController= GetComponent(CharacterController);
}
```

Πίνακας 2 – Ανάθεση character controller

Έχοντας αρχικοποιήσει στην μέθοδο Start() τον character controller, περνάμε στην επόμενη πολύ σημαντική μέθοδο, την Update(). Το κάθε script που γράφεται και γίνεται compiled μπορεί να περιέχει μια μέθοδο Update() και Start(). Κάνοντας το τελικό Build η κάθε Update() και Start() ενώνονται. Η Update() εκτελείται κάθε frame και εδώ είναι που μπαίνει ο κώδικας ο οποίος θα κάνει τον χαρακτήρα μας να κινείται. Δηλαδή σε κάθε frame υπολογίζεται η θέση του χαρακτήρα, το animation που θα πρέπει να παίξει καθώς και ο ανάλογος ήχος περπατήματος αν τρέχει, αν περπατάει ή αν βρίσκεται στο νερό.

Στον πίνακα 3 θα δούμε αρχικά τον κώδικα που χρειάζεται για να ακούγονται οι κατάλληλοι ήχοι ανάλογα με την ταχύτητα που περπατάει ο παίκτης και πάνω σε τι επιφάνεια περπατάει ο παίκτης.

```
function Update () {

if(excavate.waterWalking==false)
{

if (Input.GetKeyDown(KeyCode.W))
{
    audio.clip = walking;
    audio.Play();
    audio.loop = true;
}

if (Input.GetKeyUp(KeyCode.W))
{
    audio.clip = walking;
    audio.Stop();
}

if (Input.GetKeyDown(KeyCode.S))
{
    audio.clip = walking;
    audio.Play();
    audio.loop = true;
}

if (Input.GetKeyUp(KeyCode.S))
{
    audio.clip = walking;
```

```
    audio.Stop();

}

if (Input.GetKeyDown(KeyCode.W && KeyCode.LeftShift))
{
    audio.clip = walking;
    audio.Play();
    audio.loop = true;
    audio.pitch=2;
}



if (Input.GetKeyUp( KeyCode.LeftShift))
{
   audio.pitch=1.25;
}
}
```

Πίνακας 3 – Κώδικας για αναπαραγωγή ήχου κατά το περπάτημα σε στέρεο έδαφος

Ο κώδικας είναι χωρισμένος σε κάποιες *if* statements, οι οποίες περιγράφουν όλες τις πιθανές περιπτώσεις που μπορεί να συμβούν. Για παράδειγμα κανονικό περπάτημα σε στέρεο έδαφος, τρέξιμο σε στέρεο έδαφος, περπάτημα σε νερό κτλ. Πρέπει να αναφέρουμε ότι ο κώδικας αυτός είναι τελείως ξεχωριστός από τον κώδικα που χρειάζεται για να κινηθεί ο χαρακτήρας με τον προβλεπόμενο τρόπο.

Αρχικά επειδή στο παιχνίδι υπάρχουν δύο επιφάνειες (νερό και στέρεο έδαφος) ο κώδικας θα πρέπει να αναγνωρίζει ανά πάσα στιγμή τι βρίσκεται κάτω από τα πόδια του παίκτη. Έτσι ο κώδικας χωρίζεται σε δυο κομμάτια.

Στο πρώτο κομμάτι ελέγχεται αν ο παίκτης δεν βρίσκεται πάνω στο νερό. Αυτό γίνεται με την πρώτη γραμμή.

```
if(excavate.waterWalking==false)
```

Πίνακας 4 – Έλεγχος αν βρίσκεται νερό κάτω από τον παίκτη

Εδώ μέσω του script excavate ελέγχεται αν η Boolean μεταβλητή waterWalking είναι true ή false. Ο τρόπος που γίνεται ο έλεγχος αυτός είναι ο εξής. Ένας κύβος τροποποιημένου μεγέθους είναι τοποθετημένος μπροστά στον χαρακτήρα, σε κατάλληλο ύψος ώστε να έχει επαφή με το έδαφος. Αυτός ο κύβος γίνεται παιδί του χαρακτήρα έτσι ώστε να κινείται μαζί του. Το script excavate είναι τοποθετημένο σε αυτόν τον κύβο. Ο παρακάτω κώδικας αναγνωρίζει αν κάτω από τον παίκτη υπάρχει νερό.

```
function OnTriggerStay(other : Collider)
{
    if (other.tag == "water" )
    {
        waterWalking=true;
    }
    if (other.tag != "water" )
    {
        waterWalking=false;
    }
```

Πίνακας 5 – Λειτουργία ελέγχου επιφάνειας νερού

Η μέθοδος `OnTriggerStay` ελέγχει με τι αντικείμενα κάνει collide το αντικείμενο στο οποίο είναι προσαρτημένο το script. Η μεταβλητή other χρησιμοποιείται κάθε φορά για να μπορεί να υπάρξει αναφορά στο αντικείμενο που θέλουμε να ελεγχθεί. Έτσι μέσω της other ελέγχεται πλέον το tag του αντικειμένου με το οποίο γίνεται collision. Χρησιμοποιώντας τον έλεγχο `if (other.tag == "water" )` και το αντίθετο `if (other.tag != "water" )` μπορούμε εύκολα να αλλάξουμε την μεταβλητή waterWalking από true σε false και το αντίθετο.

Στην περίπτωση που ελέγχουμε τώρα ο παίκτης δεν βρίσκεται πάνω σε νερό οπότε μπορούμε να προχωρήσουμε στις συνθήκες που θα επιτρέψουν την αναπαραγωγή των σωστών ήχων. Στην πρώτη `if` () βλέπουμε ότι μπαίνει σαν συνθήκη το Input. `GetKeyDown` (). Η κλάση Input χρησιμοποιείται για να μπορεί να καταλάβει η Update() δύο πράγματα. Αρχικά με ποιόν τρόπο θα πατηθεί το πλήκτρο που θα μπεί σαν όρισμα και δεύτερον ποιο πλήκτρο θα είναι αυτό. Για την αναγνώριση του πλήκτρου καλείται η μέθοδος, `GetKeyDown` (),`GetKeyUp` (). Η μία ορίζει αν το πλήκτρο πατιέται και η δεύτερη αν το πλήκτρο αφήνεται. Στην συνέχεια σαν ορίσματα μπαίνουν ποια πλήκτρα θέλουμε να ενεργοποιούν τις επιθυμητές ενέργειες.

Μπαίνοντας στον κώδικα για το τι θα γίνεται αν έχουν πατηθεί τα κατάλληλα πλήκτρα έχουμε τα εξής:

- `audio.clip = walking;` : Το ηχητικό κλίπ που θα παιχτεί κατά την διάρκεια θα είναι αυτό που έχουμε αναθέσει στον editor με drag and drop, δηλαδή την μεταβλητή με το όνομα walking.
- `audio.Play();`: Δίνουμε την εντολή να παίξει ο ήχος που ορίσαμε.
- `audio.loop = true;` : Δίνουμε την εντολή ο ήχος να παίζει σε επανάληψη.
- `audio.pitch=2;` : Δίνουμε την εντολή ο ήχος να παίζει σε επανάληψη.

Στην συνέχεια θα αναλύσουμε τον κώδικα για την κίνηση του χαρακτήρα καθώς και την σωστή μίξη των αντίστοιχων στις κινήσεις animations. Εδώ θα χρησιμοποιήσουμε και πάλι κάποια if statements τα οποία όταν ικανοποιούνται θα καλούν τις κατάλληλες μεθόδους οι οποίες με την σειρά τους θα κινούν τον χαρακτήρα στην κατεύθυνση που πρέπει με την ταχύτητα που απαιτείται.
Για την περίπτωση που ο παίκτης βρίσκεται σε επιφάνεια νερού η διαδικασία είναι σχεδόν η ίδια. Σε αυτήν την περίπτωση ο αρχικός έλεγχος είναι `if(excavate.waterWalking==true)` . Με τον ίδιο τρόπο ελέγχεται η Boolean μεταβλητή waterWalking του Script excavate. Οι περιπτώσεις που πρέπει να ελεγχθούν είναι όταν πατιέται το κουμπί w για να προχωρήσει ο παίκτης μπροστά και να παίξει ο ήχος (μέσω της `Input.GetKeyDown(KeyCode.W)`), όταν αφήσει το κουμπί w για να σταματήσει να παίζει ο ήχος (μέσω της `Input.GetKeyUp(KeyCode.W)`) και όταν τρέχει ο πάικτης (μέσω της `Input.GetKeyDown(KeyCode.W && KeyCode.LeftShift)`) κρατώντας πατημένο το αριστερό Shift.

```
if(excavate.waterWalking==true)
{
    if(audio.clip==walking)
    {
        audio.clip = waterWalking;
        audio.Play();
        audio.pitch=1.5;
    }
    if (Input.GetKeyDown(KeyCode.W))
    {
        audio.clip = waterWalking;
        audio.Play();
        audio.loop = true;
        audio.pitch=1.5;
    }
    if (Input.GetKeyUp(KeyCode.W))
    {
        audio.clip = waterWalking;
        audio.Stop();
```

```
    }
   if (Input.GetKeyDown(KeyCode.S))
    {
        audio.clip = waterWalking;
        audio.Play();
        audio.loop = true;
        audio.pitch=1.5;
    }
    if (Input.GetKeyUp(KeyCode.S))
    {
        audio.clip = waterWalking;
        audio.Stop();
    }
    if (Input.GetKeyDown(KeyCode.W && KeyCode.LeftShift))
    {
        audio.clip = waterWalking;
        audio.Play();
        audio.loop = true;
        audio.pitch=2.5;
    }
   if (Input.GetKeyUp( KeyCode.LeftShift))
    {
        audio.pitch=1.5;
   }
}
```

Πίνακας 6 – Λειτουργία ελέγχου επιφάνειας νερού

Η συνέχεια του κώδικα καθορίζει πως και προς τα πού θα κινείται ο χαρακτήρας, καθώς και τα αντίστοιχα animations που θα πρέπει να παίζουν και πως θα εναλλάσσονται μεταξύ τους με φυσικό τρόπο. Η κίνηση του χαρακτήρα καθορίζεται αρχικά από την εντολή `Input.GetAxis("Vertical")` . Ο αρχικός έλεγχος που γίνεται είναι αν ο χαρακτήρας έχει κατεύθυνση προς τα μπροστά ή προς τα πίσω. Αυτό γίνεται χωρίζοντας τον κώδικα σε δύο μέρη. Όταν η παραπάνω εντολή είναι θετική ο χαρακτήρας θα προχωράει μπροστά. Όταν είναι αρνητική ο χαρακτήρας θα προχωράει προς τα πίσω. Στο παρακάτω μπλοκ κώδικα ο χαρακτήρας τρέχει μπροστά.

```
animation.CrossFade("indIdle");
if(Input.GetAxis("Vertical")>0)
{
      if(Input.GetButton("Shift"))
      {
        animation['indRun'].speed= walkAnimSpeed;
        animation.CrossFade("indRun");
        charController.Move(transform.forward*runSpeed*Time.deltaTime);
      }
```

Πίνακας 7 – Τρέξιμο προς τα εμπρός

Αρχικά παίζει το πρώτο animation, που είναι το idle μέσω της εντολής `animation.CrossFade("indIdle");`. Δηλαδή όταν ο χαρακτήρας είναι ακίνητος και αδρανής. Στη συνέχεια γίνεται ο βασικός έλεγχος αν ο χαρακτήρας κινείται μπροστά ή προς τα πίσω. Αυτό γίνεται ελέγχοντας αν το Vertical Axis είναι θετικό ή αρνητικό (`if(Input.GetAxis("Vertical")>0)`). Στην συνέχεια γίνεται έλεγχος αν είναι πατημένο το κουμπί τρεξίματος, δηλαδή το αριστερό Shift μέσω της εντολής `if(Input.GetButton("Shift"))` . Αν είναι πατημένο τότε εκτελούνται οι επόμενες εντολές:

- `animation['indRun'].speed= walkAnimSpeed;` Αρχικά δίνεται μια ταχύτητα στο animation που θα παιχτεί .
- `animation.CrossFade("indRun");` Μέσω της εντολής CrossFade θα παιχτεί το animation του τρεξίματος, το οποίο θα εναλλαχτεί ομαλά με το idle animation.
- `charController.Move(transform.forward*runSpeed*Time.deltaTime);` Μέσω της μεθόδου Move() ο Character Controller κινείται στον χώρο. Στην παρένθεση βρίσκονται τα ορίσματα τα οποία θα καθορίσουν την ταχύτητα και την κατεύθυνση. Η κατεύθυνση ορίζεται από το `transform.forward` ,η ταχύτητα από την μεταβλητή `runSpeed` και τέλος πολλαπλασιάζουμε με τον χρόνο `Time.deltaTime` .

Το επόμενο μπλοκ κώδικα κινεί τον χαρακτήρα μπροστά αλλά με ταχύτητα περπατήματος. Ουσιαστικά είναι η συνέχεια του προηγούμενου μπλοκ το οποίο ξεκίνησε με την if να ελέγχει αν το κουμπί Shift του τρεξίματος είναι πατημένο. Η else λοιπόν ελέγχει την περίπτωση που δεν είναι πατημένο το Shift, άρα ο χαρακτήρας περπατάει.

```
Else
 {
        animation['indWalk'].speed= walkAnimSpeed;
        animation.CrossFade("indWalk");
        charController.Move(transform.forward* walkSpeed*Time.deltaTime);
 }
}
```

Πίνακας 8 – Περπάτημα προς τα εμπρός

Εδώ βλέπουμε πως ο κώδικας επαναλαμβάνεται αλλά πλέον το animation που θα παίξει, θα είναι του περπατήματος.

Το επόμενο μπλοκ κώδικα αναλαμβάνει να κινήσει τον χαρακτήρα προς τα πίσω παίζοντας τα κατάλληλα animation αν τρέχει ή περπατάει με την κατάλληλη ταχύτητα.

```
else if(Input.GetAxis("Vertical")<0){
       if(Input.GetButton("Shift")){
        animation['indRun'].speed= -walkAnimSpeed;
        animation.CrossFade("indRun");
        charController.Move(transform.forward* -runSpeed*Time.deltaTime);
       }
```

Πίνακας 9 – Τρέξιμο προς τα πίσω

Θέτοντας το walkAnimSpeed με αρνητική τιμή διασφαλίζουμε ότι το animation αυτή τη φορά θα παίξει ανάποδα. Έτσι ο χαρακτήρας θα φαίνεται ότι τρέχει προς τα πίσω. Επίσης στην μέθοδο Move η μεταβλητή runSpeed θα πρέπει και αυτή να πάρει αρνητικό πρόσημο. Έτσι έχοντας – transform.forward ο χαρακτήρας θα κινηθεί προς τα πίσω.

```
Else
{
        if (!animation.IsPlaying("indWalk"))
        animation['indWalk'].time = animation['indWalk'].length;
        animation['indWalk'].speed= -walkAnimSpeed;
        animation.CrossFade("indWalk");
        charController.Move(transform.forward* -walkSpeed*Time.deltaTime);
       }
   }
```

Πίνακας 10 – Περπάτημα προς τα πίσω

Το παρακάτω μπλοκ κώδικα είναι αυτό το οποίο δίνει βαρύτητα στον χαρακτήρα όταν πηδάει, ή γενικά όταν δεν βρίσκεται στον αέρα. Είναι πολύ σημαντικό σε ένα παιχνίδι να ελέγχεται αυτή η περίπτωση και να γράφεται ο ανάλογος κώδικας που να εξομοιώνει την βαρύτητα. Όταν ένας character controller κινείται στον χώρο και φτάσει σε κάποιο σημείο το οποίο, όπως ένας γκρεμός, για να λειτουργήσει φυσιολογικά και να δοθεί στον παίκτη η εντύπωση ότι ο χαρακτήρας έχει βάρος και υπόσταση πρέπει να εφαρμοστεί η δύναμη της βαρύτητας.

```
if (!charController.isGrounded)
    {
        charController.Move(transform.up * gravityFall * Time.deltaTime);
        gravityFall -= gravity;
    }
```

Πίνακας 11 – Εφαρμογή βαρύτητας

Η πρώτη if ελέγχει αν ο χαρακτήρας είναι στον αέρα. `if (!charController.isGrounded)` . Σε αυτήν την περίπτωση σημαίνει πως ο παίκτης είτε έχει πηδήξει, είτε μόλις προχώρησε από την άκρη ενός υψηλότερου επίπεδου. Αν δεν γραφτεί κώδικας για εξομοίωση βαρύτητας αυτό που θα γίνει είναι ότι ο χαρακτήρας θα συνεχίσει να περπατάει στον αέρα. Για να αποφευχθεί αυτό, μέσω της μεθόδου Move, κινούμε τον χαρακτήρα προς τα πάνω (`transform.up`) και πολλαπλασιάζουμε με την μεταβλητή `gravityFall`. Έτσι εξομοιώνεται η δύναμη της βαρύτητας και ο χαρακτήρας θα πέσει προς τα κάτω. Αφαιρώντας κάθε φορά από την προηγούμενη μεταβλητή μία άλλη τιμή(`gravityFall -= gravity;`), θα παρατηρηθεί και μια επιτάχυνση στην πτώση του παίκτη στο κενό.

Τέλος ο κώδικας που κάνει rotate τον χαρακτήρα είναι ο παρακάτω. Όπως χρησιμοποιούμε την μέθοδο Move για να κινήσουμε τον χαρακτήρα, είναι λογικό να υπάρχει και μια αντίστοιχη μέθοδος για το rotation του χαρακτήρα. Αυτή η μέθοδος είναι η Rotate και συντάσσεται έτσι:

```
transform.Rotate(Vector3(0,Input.GetAxis("MouseX")*rotationSpeed*Time.delta
Time,0));
```

Πίνακας 12 – Rotation του χαρακτήρα

Η μέθοδος Rotate παίρνει ως ορίσματα ένα Vector3 δηλαδή ένα πίνακα τριών τιμών. Μέσω της Input δέχεται σε ποιόν άξονα θέλουμε να στρίβει ο χαρακτήρας `Input.GetAxis("MouseX")`, με τι ταχύτητα `rotationSpeed` , και τον χρόνο.

Καταλήξαμε να έχουμε έναν χαρακτήρα με custom σύστημα κίνησης, παίζοντας τους απαραίτητους ήχους ανάλογα σε τι επιφάνεια περπατάει, με τα κατάλληλα animations για την κάθε περίπτωση περπατήματος ή τρεξίματος προς όλες τις κατευθύνσεις. Επίσης του δόθηκε βαρύτητα ώστε να πέφτει όταν υπάρχει κενό ή όταν από ένα υψηλότερο κατεβαίνει σε ένα χαμηλότερο.

5.2 Manager Script

Το επόμενο Script προς παρουσίαση είναι και από τα πιο σημαντικά αφού βρίσκεται σε λειτουργία καθ' όλη την διάρκεια του παιχνιδιού και παρασκηνιακά ελέγχει κάποιες πτυχές του. Σε κάθε παιχνίδι πρέπει να υπάρχει τουλάχιστον ένα τέτοιο script το οποίο κρατάει αντικείμενα ανέπαφα από την μεταφορά ανάμεσα σε σκηνές, ή κρατάει ανέπαφους πίνακες οι οποίοι έχουν αποθηκευμένα πολύ σημαντικά στοιχεία για το παιχνίδι. Το δικό μας Manager Script εξυπηρετεί του εξής σκοπούς:

- Την δημιουργία πινάκων που θα αποθηκεύονται τα ονόματα των αντικειμένων που θα μαζεύει ο παίκτης.
- Την διατήρηση του Manager Script κατά την διάρκεια αλλαγής σκηνών.

- Τον έλεγχο για την σωστή εμφάνιση των tutorials.
- Την σωστή τοποθέτηση του χαρακτήρα στον χώρο κάθε φορά που μπαίνει και βγαίνει από ένα mode καθαρισμού αντικειμένων.

5.2.1 Δημιουργία πινάκων αποθήκευσης String αντικειμένων

Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο υπάρχουν τέσσερα modes καθαρισμού των αντικειμένων. Κάθε φορά που ο χαρακτήρας μαζεύει αντικείμενα αυτά πρέπει κάπως να αναγνωρίζονται και με κάποιο τρόπο να αποθηκεύεται ποια έχει συλλέξει. Μια πρώτη σκέψη ήταν να αποθηκεύονται απευθείας σαν game objects αλλά το Unity 3d έχει θέμα με την απευθείας αποθήκευση αντικειμένων.

Έτσι τα αντικείμενα δεν αποθηκεύονται τα ίδια σε κάποιον πίνακα, αλλά η ονομασία τους σε έναν πίνακα τύπου String. Για να αποθηκευτούν σε κάποιον πίνακα το κάθε αντικείμενο πρέπει να έχει ξεχωριστό όνομα και να έχει ένα tag το οποίο περιγράφει την ομάδα αντικειμένων στην οποία ανήκει. Τα tags είναι τα ακόλουθα:

- Mud
- Dirty
- Scanner
- Broken

Έτσι, μέσω του Excavate Script, που αναλύσαμε και παραπάνω, όταν αναγνωριστεί κάποιο αντικείμενο μέσω του tag γίνεται push στον κατάλληλο πίνακα. Δηλαδή διαβάζεται το όνομα του αντικειμένου και το string αυτό αποθηκεύεται πλέον στον πίνακα που έχει δημιουργηθεί στο manager script. Ο κώδικας του excavate script που το καθιστά δυνατό είναι ο εξής:

```
else if (other.tag == "mud" && Input.GetKeyDown(KeyCode.E))
    {

Instantiate(pickup_particles,other.transform.position,Quaternion.identity);
        manager.mud.Push(other.gameObject.name);
        Destroy(other.gameObject);
        audio.PlayOneShot(sound_pickup);
        display_message_pickUp=false;
    }
```

Πίνακας 12 – Αναγνώριση αντικειμένων στο excavate script

Έτσι, μέσω του Excavate Script, που αναλύσαμε και παραπάνω, όταν αναγνωριστεί κάποιο αντικείμενο μέσω της μεταβλητής other και του tag που του έχουμε δώσει (`other.tag == "mud"`) και έχοντας πατήσει το E (`&& Input.GetKeyDown(KeyCode.E)` θα γίνουν τα παρακάτω. Αρχικά, θα εμφανιστούν τα particle σκόνης τα οποία δημιουργήσαμε και τα οποία θα αναλύσουμε σε άλλο κεφάλαιο(`Instantiate(pickup_particles,other.transform.position,Quaternion.identity);`). Στην συνέχεια το όνομα του αντικειμένου που αναγνωρίστηκε θα προωθηθεί στον κατάλληλο πίνακα του manager script(`manager.mud.Push(other.gameObject.name);`). Η εντολή αυτή ουσιαστικά λέει, στο script manager, στον πίνακα mud, κάνε push(δηλαδή βάλε) το string, δηλαδή το όνομα του game object που αναγνωρίστηκε μόλις τώρα. Στην συνέχεια καταστρέφεται το αντικείμενο γιατί δεν χρειάζεται να υπάρχει στην σκηνή μέσω της destroy (`Destroy(other.gameObject);`), παίζει ο ήχος που αναθέσαμε στο μάζεμα των αντικειμένων(`audio.PlayOneShot(sound_pickup);`). Η χρήση της επόμενης εντολής θα αναλυθεί σε επόμενο κεφάλαιο.

Όμως πως δηλώνονται και ποια είναι η σύνταξη για την δημιουργία των πινάκων αυτών; Οι πίνακες που δημιουργήθηκαν είναι χωρισμένοι σε δύο ομάδες. Η μία ομάδα αποτελεί τους πίνακες όπου αποθηκεύονται τα ονόματα των αντικειμένων πριν καθαριστούν και η δεύτερη ομάδα πινάκων

υπάρχει για να αποθηκεύονται τα ονόματα των αντικειμένων που έχουν καθαριστεί και είναι έτοιμα να μπουν στην μουσειακή αίθουσα. Οι πίνακες της πρώτης ομάδας είναι οι:

- Mud
- Dirty
- Scanner
- Broken

Οι πίνακες της δεύτερης ομάδας είναι οι:

- Display_mud
- Display_dirty
- Display_scanner
- Display_broken

Ο παρακάτω πίνακας παρουσιάζει τον κώδικα για την δημιουργία αυτών των πινάκων:

```
static var mud = new Array ();
static var display_mud = new Array();

static var dirty = new Array ();
static var display_dirty = new Array();

static var scanner = new Array();
static var display_scanner = new Array();

static var broken = new Array ();
static var display_broken = new Array ();
```

Πίνακας 13 – Δημιουργία πινάκων

Όλοι οι πίνακες θα δηλωθούν ως static για να μπορούν να χρησιμοποιηθούν από άλλα scripts. Οποιαδήποτε μεταβλητή που πρέπει να χρησιμοποιηθεί από εξωτερικά scripts πρέπει να δηλώνεται ως static. Για να κληθεί από άλλο script πρώτα γράφεται το όνομα του script από το οποίο προέρχεται η μεταβλητή που θέλουμε να χρησιμοποιήσουμε, μετά τελεία και μετά την μεταβλητή.

5.2.2 Αλλαγή σκηνών και διατήρηση script

Το manager script το οποίο αναλύουμε, όπως και όλα τα scripts σε ένα project του Unity 3d, πρέπει να προσαρτάται σε κάποιο Game Object. Είτε αυτό το Game Object είναι κάποιο αντικείμενο της σκηνής το οποίο φαίνεται και υφίσταται (όπως πχ ο character controller), είτε είναι ένα άδειο Game Object είναι καθαρά λειτουργική επιλογή ώστε να εξυπηρετεί τις ανάγκες μας. Εμείς δημιουργήσαμε ένα empty Game Object, το οποίο ονομάσαμε και αυτό manager, και του προσαρτήσαμε το script manager. Για την δημιουργία empty Game Object επιλέγουμε από το πάνω toolbar το game object και μετά empty. Αυτό δημιουργεί το άδειο αντικείμενο και το τοποθετεί στην σκηνή. Από το hierarchy μπορεί να μετονομαστεί σε ότι βολεύει τον χρήστη. Το empty game object δεν γίνεται rendered και δεν θα φανεί ποτέ στον παίκτη είτε στον χρήστη, για αυτό το λόγο είναι κατάλληλο για τοποθέτηση script. Κάνοντας drag and drop το script στο empty game object το συσχετίζουμε.

Τώρα που το script έχει συσχετιστεί με κάποιο αντικείμενο στην σκηνή μπορεί με το που γίνει αλλαγή σκηνής να μην καταστραφεί και να συνεχίσει να υπάρχει όπως ήταν και στην προηγούμενη κατάσταση του και στην επόμενη και στην όποια σκηνή. Αυτό γίνεται γιατί ο κώδικας που εφαρμόζεται για την διατήρηση του manager αντικειμένου ισχύει και για το script το οποίο είναι προσαρτημένο στο αντικείμενο αυτό. Η εντολή για την διατήρηση ενός αντικειμένου κατά την διάρκεια αλλαγής σκηνής είναι η DontDestroyOnLoad() την οποία θα δούμε παρακάτω. Επί τη ευκαιρία θα δούμε και την εντολή με την οποία με το πάτημα ενός πλήκτρου αλλάζει η σκηνή. Στον παρακάτω πίνακα φαίνεται ο απαραίτητος κώδικας που υλοποιεί την αλλαγή σκηνής και την διατήρηση του αντικειμένου manager μαζί με το script:

```
function Update()
{
     if (Input.GetKeyDown(KeyCode.L))
          Application.LoadLevel ("Lab");


 DontDestroyOnLoad (this.transform.gameObject);


}
```

Πίνακας 14 – Αλλαγή σκηνής διατήρηση αντικειμένου

Στον πίνακα 14 βλέπουμε τον κώδικα για αυτά την αλλαγή σκηνής και την διατήρηση ενός αντικειμένου ανέπαφου. Όποιο αντικείμενο δεν διατηρηθεί από την `DontDestroyOnLoad()`, θα καταστραφεί μαζί με όλα τα υπόλοιπα αντικείμενα της σκηνής και δεν θα υπάρχει στην νέα σκηνή, άρα δεν θα μπορεί να χρησιμοποιηθεί.

Αρχικά για να αλλάξει μια σκηνή χρησιμοποιείται η μέθοδος LoadLevel() η οποία παίρνει σαν όρισμα το String, το όνομα, της σκηνής που θέλουμε να φορτώσουμε. Στον παραπάνω κώδικα φαίνεται πως φορτώνεται η σκηνή του εργαστηριού του χαρακτήρα την οποία έχουμε ονομάσει 'Lab'. Όπως έχουμε δει και σε προηγούμενα με την χρήση της GetKeyDown() πατώντας, στην συγκεκριμένη περίπτωση, το 'L' θα μεταφερθούμε στον χώρο του εργαστηρίου.

Στην συνέχεια η `DontDestroyOnLoad()`, κρατλαε το αντικείμενο ανέπαφο κατά την μετάβαση. Το όρισμα της είναι απλός το αντικείμενο που πρέπει να διατηρηθεί. Στην συγκεκριμένη περίπτωση επειδή το script βρίσκεται προσαρτημένο στο αντικείμενο που πρέπει να παραμείνει ανέπαφο χρησιμοποιείται το `this.` . Το 'this' δηλαδή αναφέρεται πάντα στο αντικείμενο το οποίο φιλοξενεί το script που τρέχει εκείνη την στιγμή. Το 'this' εδώ δηλαδή αναφέρεται στο αντικείμενο manager. Για να ολοκληρωθεί το όρισμα αναφερόμαστε στο αντικείμενο μέσω του transform. Πάντα όταν γίνεται αναφορά σε ένα αντικείμενο χρησιμοποιείται το `transform.gameObject`.

5.2.3 Έλεγχος για την σωστή εμφάνιση των Tutorials

Τα tutorials στο παιχνίδι εμφανίζονται σε συγκεκριμένες στιγμές και είναι φτιαγμένα έτσι ώστε να καθοδηγήσουν τον παίκτη στα πρώτα του βήματα. Για τον λόγο αυτό δεν εμφανίζεται ένα κείμενο το οποίο επεξηγεί όλες τις δυνατότητες του παιχνιδιού και τους στόχους του, αλλά κάθε μήνυμα που εμφανίζεται οδηγεί τον παίκτη στο τι πρέπει να κάνει μετά. Επίσης όταν ο παίκτης εμφανίζεται πρώτη φορά στον χώρο του εργαστηρίου με το που ξεκινήσει το παιχνίδι επειδή δεν έχει συλλέξει ακόμα κανένα αντικείμενο οι colliders των πάγκων που αλλάζουν την σκηνή και οδηγούν στα τέσσερα modes είναι απενεργοποιημένοι.

Ο παίκτης πρέπει πρώτα να διαβάσει το tutorial το οποίο θα τον ενημερώσει πως πρέπει να βγει στον αρχαιολογικό χώρο να ψάξει και να βρει αντικείμενα και μόλις επιστρέψει να πλησιάσει τους πάγκους για να δει μπει στα διάφορα modes. Σε εκείνο το σημείο οι colliders ενεργοποιούνται και ο παίκτης μόλις πλησιάσει τους πάγκους θα δει να εμφανίζεται ειδοποιητικό μήνυμα ότι μπορεί να κάνει χρήση των modes.

Ο τρόπος με τον οποίο μπορεί να υλοποιηθεί το παραπάνω σενάριο είναι αν ανά πάσα στιγμή μπορεί να είναι γνωστό ποιά σκηνή είναι φορτωμένη. Επίσης πρέπει να είναι γνωστό πόσες φορές έχει φορτωθεί μια σκηνή. Γενικά αυτές οι λεπτομέρειες μπορούν να χειριστούν με κάποιους integers οι οποίοι καθώς αυξομειώνονται, σε συνδυασμό με την γνώση του ποια σκηνή είναι φορτωμένη, εμφανίζει τα tutorials σε λογικά και κομβικά σημεία που καθοδηγούν τον παίκτη.

Άρα λοιπόν, με βάση τα παραπάνω, άλλη μια πολύ σημαντική συνάρτηση είναι η (`OnLevelWasLoaded (level : int)`). Αυτή είναι η εντολή με την οποία μπορούμε ανά πάσα στιγμή να γνωρίζουμε ποια σκηνή έχει φορτωθεί και να κάνουμε τους υπολογισμούς που θέλουμε. Το όρισμα που δέχεται είναι απλά ένα ακέραιος ο οποίος αντιπροσωπεύει σαν index number την κάθε σκηνή. Ο αριθμός για κάθε σκηνή μπορεί να βρεθεί πηγαίνοντας στο file και μετά επιλέγοντας την επιλογή build settings. Στα build settings θα αναφερθούμε σε επόμενο κεφάλαιο. Άρα ξέροντας τα

index των σκηνών μπορούμε μόλις φορτωθεί μια σκηνή που επιθυμούμε να εισάγουμε κώδικα που θα χειρίζεται τα tutorials που θέλουμε. Στον παρακάτω πίνακα φαίνεται ο κώδικας:

```
function OnLevelWasLoaded (level : int)
{
        if (level == 3)
         {
            in_main=false;
            t_cnt=t_cnt+1;

       if(level==1 && t_cnt!=0)
         {
            in_main=true;
         }

      if(level==1)
         {

           t2_cnt=t2_cnt+1;
         }

      if(level==3 && t2_cnt!=0)
        {
          t2_cnt=t2_cnt+1;


         }



}
```

Πίνακας 15 – Εμφάνιση των tutorials

Για να εξηγηθεί ο κώδικας εμφάνισης των tutorials καλύτερα έχοντας τα index numbers της κάθε σκηνής θα βοηθήσει στο να κατανοήσουμε καλύτερα τον κώδικα. Αρχικά βλέπουμε πως η σκηνή του εργαστηρίου είναι αυτή με τον index αριθμό τρία. Είναι λογικό να ξέρουμε και να κρατάμε κάθε φορά την πληροφορία του ότι όταν είμαστε στην σκηνή του εργαστηρίου δεν γίνεται να είμαστε ταυτόχρονα και στην main σκηνή. Άρα στην πρώτη if το πρώτο βήμα είναι να έχουμε μια Boolean μεταβλητή, την in_main, η οποία θα γίνεται true ή false ανάλογα με την σκηνή που βρισκόμαστε. Άρα αρχικά είναι false γιατί βρισκόμαστε στο εργαστήριο. Στο εργαστήριο θα εμφανιστούν Tutorials δύο φορές. Μόλις μπει ο παίκτης για πρώτη φορά και μια δεύτερη μόλις γυρίσει από την σκηνή main. Άρα θέλουμε να ξέρουμε και να μετρήσουμε πόσες φορές φορτώθηκε η σκηνή του εργαστηρίου. Για αυτό το λόγο χρησιμοποιούμε έναν counter, τον t_cnt, ο οποίος θα αυξάνεται κατά ένα. Άρα όταν φτάσει στον αριθμό δύο θα ξέρουμε ότι ο παίκτης μόλις γύρισε από την main scene και μπορεί να εμφανιστεί το δεύτερο tutorial.

Στην επόμενη if συνθήκη αλλάζουμε την μεταβλητή in_main και την κάνουμε true γιατί πλέον ελέγχουμε αν έχει φορτωθεί η σκηνή ένα, δηλαδή η σκηνή του αρχαιολογικού χώρου. Εδώ αντίστοιχα χρησιμοποιούμε έναν counter για να ελέγχουμε πόσες φορές έχει μπει ο παίκτης στην σκηνή και τον ονομάζουμε t2_cnt ο οποίος και αυτός αυξάνει κάθε φορά.

Πλέον χρησιμοποιώντας άλλο ένα Script το οποίο ονομάζουμε Lab_tutorial μπορούμε να εμφανίσουμε και να εξαφανίσουμε τα tutorials όποτε θέλουμε. Αναλύοντας παρακάτω το script θα δούμε και την συνάρτηση onGui(). Η συνάρτηση αυτή, είναι που επιτρέπει την δημιουργία οποιουδήποτε γραφικού περιβάλλοντος στο παιχνίδι.

Στον πρώτο πίνακα θα δούμε πως εμφανίζεται το αρχικό εισαγωγικό tutorial με το που εμφανίζεται ο παίκτης στο παιχνίδι, δηλαδή στο εργαστήριο.

```
function OnGUI()
{

    if (manager.t_cnt==1 && manager.in_main==false)
    {

GUI.Box(Rect(Screen.width/2,Screen.height/2,Screen.width/2,Screen.height/2)
,"Welcome to the game! This is your lab. If you explore around you will
find some benches that will be used later for restoring archaeological
items. But first thing first. Head for the door to enter the dig site!
Click here to continue.",boxStyle20);

    if(Input.GetMouseButton(0))
    {
        Destroy(GameObject.Find("Lab_tutorial"));
    }

}
```

Πίνακας 16 – Εμφάνιση 1ου tutorial

Αρχικά χρησιμοποιούμε όπως είπαμε και πιο πάνω την συνάρτηση η οποία μας επιτρέπει να δημιουργούμε γραφικά περιβάλλοντα, την onGUI(). Μέσα σε αυτήν μπορούμε να κάνουμε όλους κατάλληλους ελέγχους για το ποιο σημείο βρίσκεται ο παίκτης και να εμφανίσουμε με γραφικό τρόπο το κείμενο που θέλουμε. Ο πρώτος έλεγχος που γίνεται εδώ βλέπουμε πως είναι ο έλεγχος της static μεταβλητής t_cnt που μιλήσαμε πιο πάνω. Αν αυτή ισούται αποκλειστικά με ένα και η static μεταβλητή in_main είναι false τότε αυτό σημαίνει ότι ο παίκτης μόλις μπήκε στο παιχνίδι. Οπότε εμφανίζεται το μήνυμα που θέλουμε. Για τις εντολές GUI θα αναφερθούμε σε επόμενο κεφάλαιο.

Η επόμενη if δίνει την εντολή αν πατηθεί το αριστερό κλικ του ποντικιού να καταστραφεί το αντικείμενο το οποίο φιλοξενεί το GUI με αποτέλεσμα να εξαφανιστεί από την οθόνη. Εδώ γίνεται η πρώτη γνωριμία με την συνάρτηση Destroy() και την μέθοδο Find(). Η συνάρτηση Destroy όπως λέει και το όνομα της καταστρέφει από την σκηνή ένα αντικείμενο. Ο καλύτερος τρόπος για να αναφερθούμε σε ένα αντικείμενο το οποίο βρίσκεται στην σκηνή είναι να χρησιμοποιήσουμε την μέθοδο Find. Στην συγκεκριμένη περίπτωση ψάχνουμε το αντικείμενο με όνομα 'Lab_tutorial'. Άρα βάζοντας αυτό το String σαν όρισμα η Find θα βρει αυτό το Game Object. Βάζοντας αυτό πλέον σαν όρισμα στην Destroy θα καταστραφεί το αντικείμενο αυτό με αποτέλεσμα να εξαφανιστεί το Tutorial.

Ο επόμενος πίνακας δείχνει πως καλείται το επόμενο Tutorial. Αυτό πρέπει να εμφανίζεται όταν ο παίκτης έχει επισκεφτεί το Main Scene την πρώτη φορά και έχει επιστρέψει έχοντας μαζέψει κάποια αρχαιολογικά αντικείμενα.

```
if (manager.t_cnt==2 && manager.in_main==false)
{

GUI.Box(Rect(Screen.width/2,Screen.height/2,Screen.width/2,Screen.height/2)
,"Now that you have gathered some items approach the benches and the
scanner, follow the instructions and start restoring! Each item you found
corresponds to only one restoring mode. Click here to
continue.",boxStyle20);

    if(Input.GetMouseButton(0))
    {
        Destroy(GameObject.Find("Lab_tutorial"));

    }

}
```

Πίνακας 17 – Εμφάνιση 2ου tutorial

Εδώ βλέπουμε ότι για να εμφανιστεί το δεύτερο tutorial πρέπει η static μεταβλητή t_cnt να ισούται αποκλειστικά με δύο και η static μεταβλητή in_main να είναι false δηλαδή ο παίκτης να είναι στο αρχαιολογικό εργαστήριο. Αν ισχύουν τα παραπάνω εμφανίζεται το επόμενο tutorial και κάνοντας κλικ επάνω του θα εξαφανιστεί εάν το επιθυμεί ο παίκτης.

Πιο πάνω αναφέραμε πως είναι λογικό ο παίκτης να μπορεί να έχει πρόσβαση στα modes μόνο από τη στιγμή που θα επιστρέψει την πρώτη φορά από το main scene. Αυτό σημαίνει ότι μόνο τότε θα μπορούν να εμφανιστούν τα μηνύματα τα οποία ενημερώνου τον παίκτη ότι μπορεί να εισέλθει στα modes πλησιάζοντας τους πάγκους.

Ο κάθε πάγκος έχει τοποθετημένο γύρο του έναν box collider ο οποίος είναι υπεύθυνος για τον έλεγχο της σύγκρουσης με τον παίκτη. Τοποθετώντας ένα script το οποίο ονομάζουμε κάθε φορά ανάλογα με το mode που ενεργοποιεί ο κάθε πάγκος, το οποίο κάνει τους απαραίτητους ελέγχους μπορεί να εμφανίζεται και να εξαφανίζεται το μήνυμα ειδοποίησης εισόδου σε mode. Στον επόμενο πίνακα είναι ο κώδικας.

```
var display_tutorial: boolean;
var mySkin : GUISkin;
static var PlayerPos:Vector3;
static var dirty_entered:boolean;

function Start()
{
   display_tutorial=false;
   dirty_entered=false;
}


function OnTriggerStay(other : Collider)
{
    if (other.tag == "Player" && manager.t_cnt>=2)
    {
        display_tutorial=true;

        if (Input.GetKeyDown(KeyCode.E))
        {
            dirty_entered=true;
            PlayerPos=GameObject.FindWithTag("Player").transform.position;

            Application.LoadLevel ("Dirty");

    }
    }
}
function OnTriggerExit(other:Collider)
{
     display_tutorial=false;
}

function OnGUI()
{
GUI.skin = mySkin;
    if (display_tutorial)
     GUI.Box(Rect(0,0,Screen.width/3,Screen.height/9),"Press E  to enter
Dirt Removal mode!");
}
```

Πίνακας 18 – Εμφάνιση μηνυμάτων έναρξης modes

Και σε αυτό το κομμάτι κώδικα θα δουλέψουμε με κάποια Boolean μεταβλητή, το tag του παίκτη και την μεταβλητή t_cnt που χρησιμοποιήσαμε ποιο πάνω. Αυτό που μας ενδιαφέρει εδώ είναι το μήνυμα να εμφανίζεται όταν ο παίκτης μπαίνει για δεύτερη φορά στο εργαστήριο και να παραμένει μόνο όσο βρίσκεται σε σύγκρουση με το box collider του πάγκου.

Αρχικά ξεκινάμε δηλώνοντας τις μεταβλητές και αρχικοποιώντας τες με την συνάρτηση Start(). Αυτό σημαίνει ότι με το που φορτωθεί η σκηνή η συνάρτηση display_tutorial θα είναι false. Στην συνέχεια, επειδή θέλουμε όσο βρίσκεται σε επαφή με τον box collider ο παίκτης να ελέγχουμε, θα χρησιμοποιήσουμε την συνάρτηση OntriggerStay(). Άρα αν το tag που κάνει collide με τον box collider είναι 'Player' και η μεταβλητή t_cnt έχει πάρει την τιμή δύο (`if (other.tag == "Player" && manager.t_cnt>=2)`), τότε και μόνο τότε η μεταβλητή display_tutorial θα γίνει true. Άρα για νε εμφανιστεί το GUI καλούμε την OnGUI() και χρησιμοποιώντας την συνθήκη (`if (display_tutorial)`), δηλαδή αν η μεταβλητή display tutorial είναι true.

Όμως, επειδή αν ο παίκτης χάσει την επαφή με τον Box collider του πάγκου το GUI μήνυμα θέλουμε να αξαφανιστεί πρέπει να χρησιμοποιήσουμε άλλη μια συνάρτηση. Αυτή η συνάρτηση είναι η OnTriggerExit(). Άρα κάθε φορά που απομακρύνεται ο παίκτης από τον πάγκο η μεταβλητή display_tutorial γίνεται false και το μήνυμα θα εξαφανίζεται σε κάθε update.

Πλέον το μόνο που απομένει είναι το Tutorial που εξηγεί στον παίκτη ποιοι είναι οι στόχοι του μόλις φορτωθεί η main σκηνή και εμφανιστεί στον αρχαιολογικό χώρο. Εδώ όπως αναφέραμε πιο πάνω θα χρησιμοποιηθεί η μεταβλητή ελέγχου t2_cnt η οποία ελέγχει πόσες φορές έχει μπεί ο χαρακτήρας στον αρχαιολογικό χώρο. Εμάς μας ενδιαφέρει να εμφανιστεί μόνο μια φορά το tutorial άρα αρκεί η μεταβλητή να είναι ένα.

```
if (manager.t2_cnt==1 )
{
GUI.Box(Rect(Screen.width/2,Screen.height/2,Screen.width/2,Screen.height/2)
,"This is the dig dite. Here you will unearth archaeologicl items. Just
follow the pillars of light to find them. Click here to
continue.",boxStyle20);


     if(Input.GetMouseButton(0))
        {
         Destroy(GameObject.Find("Main_tutorial"));

        }
}
```

Πίνακας 19 – Εμφάνιση 3ου tutorial

5.2.4 Σωστή τοποθέτηση χαρακτήρα κατά την έξοδο από Mode

Κάνοντας μια ανακεφαλαίωση, στο σημείο που βρισκόμαστε τώρα ο παίκτης έχει εμφανιστεί στο εργαστήριο και έχει διαβάσει το πρώτο Tutorial. Για το συγκεκριμένο υποκεφάλαιο θα προσπεράσουμε όλες τις διαδικασίες και θα εξετάσουμε τον κώδικα που επαναφέρει τον παίκτη στην θέση που ήταν πριν εισέλθει σε κάποιο mode καθαρισμού.

Στο κάθε script αρχείο που υπάρχει τοποθετημένο σε κάθε collider πάγκου παρατηρούμε δύο μεταβλητές ακόμα που δεν σχολιάσαμε πριν. Η μια είναι η dirty_entered και η άλλη είναι η Plyer_Pos. Να αναφέρουμε ότι για κάθε script πάγκου η μεταβλητή _entered είναι διαφορετική (scanner_entered, mud_entered, κτλ). Εδώ για να εξηγήσουμε θα συνεχίσουμε με την dirty_entered.

Η μεταβλητή dirty_entered είναι και αυτή Boolean και αρχικοποιείται σαν false με το που φορτώνει η σκηνή στην συνάρτηση start(). Η μεταβλητή Player_Pos είναι τύπου Vector3. Δηλαδή ένα πίνακας που δέχεται τρείς τιμές. Αυτές οι τιμές καθορίζουν τη θέση του παίκτη στον χώρο (x,y,z). Η μεταβλητή dirty_entered γίνεται true μόλις ο παίκτης πατήσει το κουμπί 'E' για να μεταφερθεί στην σκηνή που αντιστοιχεί στον πάγκο που διάλεξε. Όσο για την μεταβλητή Player_Pos βλέπουμε ότι για να της αποδώσουμε τιμή χρησιμοποιούμε μια γνώριμη μέθοδο

(`PlayerPos=GameObject.FindWithTag("Player").transform.position;`). Εδώ έχουμε την μέθοδο FindWithTag η οποία παίρνει σαν όρισμα κάποιο String το οποίο είναι tag κάποιου αντικειμένου. Έτσι βρίσκεται ο χαρακτήρας και χρησιμοποιώντας το transform.position παίρνουμε την θέση του στον χώρο(x,y,z) και την αποθηκεύουμε στην μεταβλητή PlayerPos.

Άρα πλέον έχουμε κρατήσει την θέση από την οποία έχει μπει ο παίκτης στην νέα σκηνή. Όντας πλέον στη νέα σκηνή αν θέλουμε να επιστρέψουμε στο εργαστήριο πατώντας το 'L' ο παίκτης θα εμφανιστεί στην ίδια θέση και αυτό χάρη στον παρακάτω κώδικα του manager script.

```
if(Dirt_Removal.dirty_entered==true)
            {

GameObject.FindWithTag("Player").transform.position=Dirt_Removal.PlayerPos;

            }

        if(lab_door.main_entered==true)
            {

GameObject.FindWithTag("Player").transform.position=lab_door.PlayerPos;

            }
        if(Mud_Removal.mud_entered==true)
            {

GameObject.FindWithTag("Player").transform.position=Mud_Removal.PlayerPos;

            }
        if(Broken_Removal.broken_entered==true)
            {

GameObject.FindWithTag("Player").transform.position=Broken_Removal.PlayerPo
s;
            }
        if(lab_scanner.scanner_entered==true)
            {

GameObject.FindWithTag("Player").transform.position=lab_scanner.PlayerPos;
            }
```

Πίνακας 20 – Ορισμός σωστής θέσης επαναφοράς του παίκτη

Βλέπουμε ότι η κάθε if σε αυτό το κομμάτι κώδικα ελέγχει το ανάλογο script του κάθε collider των πάγκων για να δει αν η μεταβλητή _entered είναι True. Εάν είναι αυτό που γίνεται είναι η νέα θέση που θα έχει ο παίκτης όταν ξαναμπεί στην σκηνή του εργαστηρίου θα εξισωθεί και θα πάρει την τιμή της μεταβλητής PlayerPos που είδαμε πιο πάνω, δηλαδή την τιμή της θέσης από όπου μπήκε ο παίκτης στο mode.

5.3 Scripts αρχαιολογικού χώρου

Μόλις ο παίκτης διαβάσει το πρώτο tutorial θα κινηθεί προς την πόρτα του εργαστηρίου και θα βρεθεί στον αρχαιολογικό χώρο. Όπως αναλύσαμε πριν το tutorial που θα εμφανιστεί εκεί θα εξηγήσει στον παίκτη ποιος είναι ο σκοπός του σε αυτήν τη σκηνή. Στον αρχαιολογικό χώρο ο παίκτης πρέπει να αναγνωρίσει τα σημεία όπου μπορεί να σκάψει και μετά να συλλέξει το αρχαιολογικό αντικείμενο.

Σε αυτό το σημείο του παιχνιδιού τα scripts που τρέχουν ενεργά είναι το manager(το οποίο τρέχει πάντα όπως έχουμε πει), το player script του χαρακτήρα βεβαίως και το excavate script το

οποίο είπαμε πως βρίσκεται προσαρτημένο στον collider που βρίσκεται μπροστά από τον παίκτη και που ελέγχει όλα τα collisions.

```
private var scrollPosition : Vector2;
static var waterWalking: boolean;
var someRect : Rect;
var hit : RaycastHit;
var excavate_particles : GameObject;
var pickup_particles : GameObject;
var display_message_excavate: boolean;
var display_message_pickUp: boolean;
var sound_excavate: AudioClip;
var sound_pickup: AudioClip;
var mySkin : GUISkin;

function Start ()
{

    display_message_excavate=false;
    display_message_pickUp=false;
    someRect=Rect(0,0,Screen.width/6,Screen.width/6);
    waterWalking=false;

}

function OnTriggerStay(other : Collider)
{

    if (other.tag == "water" )
    {
        waterWalking=true;
    }

     if (other.tag != "water" )
    {
        waterWalking=false;
    }



    if (other.tag == "cap" )
    {
        display_message_excavate=true;
    }

    else if (other.tag == "mud" || other.tag =="scanner" || other.tag
=="dirty" || other.tag =="broken"  )
    {
        display_message_pickUp=true;

    }



    if (other.tag == "cap" && Input.GetKeyDown(KeyCode.E))
    {
        Instantiate(excavate_particles, other.transform.position,
Quaternion.identity);
        Destroy(other.gameObject);
        audio.PlayOneShot(sound_excavate);
```

```
        display_message_excavate=false;




    }

    else if (other.tag == "mud" && Input.GetKeyDown(KeyCode.E))
    {

        Instantiate(pickup_particles, other.transform.position,
Quaternion.identity);
        manager.mud.Push(other.gameObject.name);
        Destroy(other.gameObject);
        audio.PlayOneShot(sound_pickup);
        display_message_pickUp=false;

    }

    else if (other.tag == "scanner" && Input.GetKeyUp(KeyCode.E))
    {
        Instantiate(pickup_particles, other.transform.position,
Quaternion.identity);
        manager.scanner.Push(other.gameObject.name);
        Destroy(other.gameObject);
        audio.PlayOneShot(sound_pickup);
        display_message_pickUp=false;
    }

    else if (other.tag == "dirty" && Input.GetKeyDown(KeyCode.E))
    {
        Instantiate(pickup_particles, other.transform.position,
Quaternion.identity);
        manager.dirty.Push(other.gameObject.name);
        Destroy(other.gameObject);
        audio.PlayOneShot(sound_pickup);
        display_message_pickUp=false;

    }

    else if (other.tag == "broken" && Input.GetKeyDown(KeyCode.E))
    {
        Instantiate(pickup_particles, other.transform.position,
Quaternion.identity);
        manager.broken.Push(other.gameObject.name);
        Destroy(other.gameObject);
        audio.PlayOneShot(sound_pickup);
        display_message_pickUp=false;

    }

}


function OnTriggerExit(other : Collider)
{

  display_message_pickUp=false;
  display_message_excavate=false;
}
```

```
function OnGUI()
{
   GUI.Box(Rect(0,0,Screen.width/3,Screen.height/9),"Press L if you want to
return to the lab");

   if(display_message_excavate)
   {
      GUI.Box(Rect(0,0,Screen.width/3,Screen.height/9),"Press E to excavate
here.");
   }

   if(display_message_pickUp)
   {
      GUI.Box(Rect(0,0,Screen.width/3,Screen.height/9),"Press E to pick up
this object");

   }
}
```

Πίνακας 21 – Excavate script

5.4 Μηχανισμοί καθαρισμού αρχαιολογικών αντικειμένων

Σε αυτό το σημείο ο παίκτης έχει μαζέψει κάποια, ή όλα τα αρχαιολογικά αντικείμενα και έχει επιστρέψει στον εργαστηριακό χώρο. Όπως τον έχει ενημερώσει και το tutorial, όπως είδαμε αναλυτικά παραπάνω, πλέον μπορεί να πλησιάσει τους πάγκους και να ενεργοποιήσει τα διάφορα modes καθαρισμού των αντικειμένων που έχει συλλέξει. Στα επόμενα υποκεφάλαια θα αναλύσουμε τους μηχανισμούς για κάθε ένα mode ξεχωριστά.

Όλοι οι μηχανισμοί έχουν κάποια κοινά στον τρόπο που έχουν δομηθεί με βάσει τα Scripts τους. Όλοι οι μηχανισμοί έχουν ένα empty game object το οποίο περιέχει ένα script το οποίο δημιουργεί το GUI με τα κουμπιά τα οποία ενεργοποιούν τα συλλεγμένα αντικείμενα. Επίσης αυτό το script ενεργοποιεί κάποιο άλλο script το οποίο είναι ενσωματωμένο στα αντικείμενα που έχει συλλέξει ο παίκτης.

Άλλο ένα κοινό χαρακτηριστικό δόμησης που μοιράζονται όλα τα modes καθαρισμού των αρχαιολογικών αντικειμένων είναι ο τρόπος με τον οποίο αλληλεπιδρά ο χρήστης με τα αντικείμενα. Ότι αντικείμενο έχει συλλέξει ο παίκτη, όπως είδαμε στην αρχή του κεφαλαίου τέσσερα, δίνει το όνομα του σε μια θέση ενός πίνακα από strings. Αυτέ οι θέσεις διαβάζονται και ανάλογα με το πλήθος τους δημιουργούν και τα ανάλογα κουμπιά σε αριθμό, και τίτλο.

Πριν ξεκινήσουμε την ανάλυση του κώδικα είναι καλό να έχουμε μια μικρή ιδέα για το τι γίνεται πίσω από την κάμερα σε κάθε scene. Η κάμερα της σκηνής είναι τοποθετημένη έτσι ώστε να στοχεύει και να δείχνει προς ένα στημένο χώρο όπου ο παίκτης θα μπορεί να αλληλεπιδράσει. Σε αυτόν το χώρο βρίσκονται επίσης τοποθετημένα τα audio sources(τα οποία βεβαίως δεν φαίνονται.). Το ενδιαφέρον είναι ότι πίσω από την κάμερα (ή τουλάχιστον εκτός της περιοχής που βλέπει) βρίσκονται τοποθετημένα όλα τα αντικείμενα τα οποία θα μπορέσει να αλληλεπιδράσει ο παίκτης. Με κατάλληλο κώδικα μόλις πατήσει κάποιο κουμπί επιλογής αντικειμένου αυτό εμφανίζεται στο σωστό σημείο και αρχίζει η αλληλεπίδραση.

Τέλος για όλα τα modes καθαρισμού αρχαιολογικών αντικειμένων, όταν ένα αντικείμενο θεωρείται πως έχει καθαριστεί και είναι έτοιμο να μπει στην μικρή μουσειακή αίθουσα, τότε το όνομά του διαγράφεται από τον πρώτο πίνακα που ήταν αποθηκευμένο και περνάει στον δεύτερο πίνακα που είδαμε που έχει template display_.

5.4.1 Mud Removal

Το πρώτο από τα τέσσερα modes καθαρισμού αρχαιολογικών αντικειμένων που θα ασχοληθούμε είναι αυτό του καθαρισμού λάσπης. Η ιδέα είναι ο παίκτης να αντιλαμβάνεται ότι το αντικείμενο που έχει μπροστά του είναι όντως λερωμένο και χρειάζεται καθαρισμό. Για αυτό το λόγο δημιουργήσαμε χοντροκομμένα κομμάτια λάσπης τα οποία είναι κολλημένα πάνω στα αρχαιολογικά αντικείμενα.

Επιπρόσθετα πρέπει ο παίκτης να μπορεί να χειρίζεται το αντικείμενο πραγματικά μιας και λειτουργεί σε τρισδιάστατο χώρο. Άρα η καλύτερη λύση ήταν να μπορεί να κάνει rotate το αντικείμενο προς όλες τις κατευθύνσεις και ταυτόχρονα να καταστρέφει με ικανοποιητικό τρόπο την συσσωρευμένη λάσπη.

Άρα τα scripts που θα αναλύσουμε σε αυτό το υποκεφάλαιο θα είναι τρία. Το πρώτο είναι το 'Destroy Dirt' το οποίο είναι αυτό το οποίο δημιουργεί τα κουμπιά του GUI, το δεύτερο είναι το 'Rotate' το οποίο, όπως υποδεικνύει και το όνομα του, είναι υπεύθυνο για το rotation του αντικειμένου και τέλος το script 'Destroy Mud' το οποίο είναι υπεύθυνο για την καταστροφή της λάσπης στα αντικείμενα.

Το πρώτο script που θα αναλύσουμε είναι το 'Destroy Dirt'. Όπως είπαμε κα στην αρχή του κεφαλαίου ένα από τα κοινά scripts, όσον αφορά σε λειτουργία, είναι αυτό το οποίο δημιουργεί τα κουμπιά του GUI με την λειτουργικότητα τους. Το 'Destroy Dirt' είναι αυτό το script. Τα κομμάτια που αφορούν την δημιουργία του γραφικού περιβάλλοντος θα τα αναλύσουμε σε επόμενο κεφάλαιο. Εδώ θα επικεντρωθούμε μόνο στην λειτουργία των κουμπιών του GUI. Θα ξεκινήσουμε από τις δηλώσεις μεταβλητών.

```
var boxStyle3 : GUIStyle;
var boxStyle10 : GUIStyle;
var cnt: int;
private var scrollPosition : Vector2;
private var scrollPosition2 : Vector2;
static var selected_object_mud: int;
var back: Texture2D;


var object_1: Transform;
var object_2: Transform;
var object_3: Transform;
var object_4: Transform;
var object_5: Transform;
var object_6: Transform;
var object_7: Transform;
```

Πίνακας 22 – Δηλώσεις μεταβλητών Destroy Dirt script

Η πρώτη μεταβλητή που μας ενδιαφέρει για την ανάλυση αυτού του script είναι η selected_object_mud. Είναι ένας ακέραιος ο οποίος θα μας βοηθήσει σε επόμενο script να διαπιστώσουμε ποιο αντικείμενο επεξεργάζεται ο παίκτης και να διαγραφεί από τον πρώτο πίνακα που βρίσκεται και να περάσει στον display πίνακα.

Στην συνέχεια έχουμε επτά δηλώσεις μεταβλητών τύπου transform (object_1, object_2, object_3 κτλ). Ουσιαστικά σε αυτές τις μεταβλητές, στον editor, θα αποθηκευτούν τα αντικείμενα τα οποία θα διαχειριστεί ο παίκτης και είναι κριμένα στην σκηνή.

Σε αυτό το σημείο είναι που παίζει βασικό και μεγάλο ρόλο ο πίνακας στον οποίο αποθηκεύονται τα ονόματα (strings) των αντικειμένων που έχει μαζέψει ο παίκτης στον αρχαιολογικό χώρο. Για να φτιαχτούν τα κουμπιά, που θα φέρνουν το αντικείμενο μπροστά στην κάμερα, πρέπει ο κώδικας να είναι ενήμερος σχετικά με το πόσα αντικείμενα έχουν συλλεχθεί.

```
if(manager.mud.length >=1)
```

```
    {
        if(GUILayout.Button (manager.mud[0],boxStyle10))
        {
GameObject.Find(manager.mud[0]).GetComponent(Rotate).enabled = true;

GameObject.Find(manager.mud[0]).transform.position=
instantiation_node.position;

selected_object_mud = 0;


        }
    }
```

Πίνακας 23– Δημιουργία ενός κουμπιού μέσω της συνάρτησης OnGUI()

Έχοντας τον πίνακα mud στο script manager με αποθηκευμένα τα ονόματα των αντικειμένων που έχει μαζέψει ο παίκτης μέσω του script είναι εύκολο να γνωρίζουμε το μέγεθός του μέσω της length (`manager.mud.length`). Ελέγχοντας το μήκος του πίνακα αν είναι μεγαλύτερος ή ίσος με ένα είμαστε σίγουροι ότι θα δημιουργηθεί ένα κουμπί. Το κουμπί αυτό δημιουργείται στην αμέσως επόμενη if και ταυτόχρονα ελέγχεται αν έχει πατηθεί έτσι ώστε να ξεκινήσει η λειτουργικότητα για την μετακίνηση των strings στους πίνακες που συζητήσαμε πιο πάνω (`if(GUILayout.Button (manager.mud[0],boxStyle10))`).

Άρα εάν πατηθεί το κουμπί (που σημαίνει ότι ο παίκτης επιθυμεί να φέρει στο προσκήνιο το συγκεκριμένο αντικείμενο για να το επεξεργαστεί) περνάμε στις επόμενες εντολές. Αρχικά θα πρέπει να ενεργοποιηθεί το Rotate script για να μπορεί ο παίκτης να γυρίζει το αντικείμενο στον χώρο (`GameObject.Find(manager.mud[0]).GetComponent(Rotate).enabled = true;`). Την μέθοδο Find() την είδαμε και σε προηγούμενο script και ξέρουμε ότι ψάχνει να βρει το αντικείμενο που έχει το όνομα που έχουμε βάλει σαν όρισμα. Στην συγκεκριμένη περίπτωση το string που μας ενδιαφέρει είναι αυτό που έχει αποθηκευτεί στην θέση 0 του πίνακα mud (`manager.mud[0]`). Άρα το script θα ψάξει στον πίνακα mud, του script manager τι string υπάρχει στη θέση 0. Πλέον μέσα στην σκηνή θα βρει το αντικείμενο με αυτό το όνομα.

Για να ενεργοποιηθεί το script Rotate στο αντικείμενο αυτό πρέπει να μπορούμε να έχουμε πρόσβαση στο script αυτό. Εδώ εμφανίζεται η μέθοδος GetComponent η οποία μπορεί να προσπελάσει όλα τα components (scripts, colliders, children κτλ) και να τα διαχειριστεί όπως επιθυμούμε. Εμείς θέλουμε να ενεργοποιήσουμε το script άρα αρκεί να του πούμε να το κάνει enable (`GetComponent(Rotate).enabled = true;`).

Έχοντας ενεργοποιήσει το script, το μόνο που μένει είναι να μεταφερθεί το αντικείμενο μπροστά στην κάμερα για να μπορεί να το χειριστεί ο παίκτης. Ξανά μέσω της μεθόδου Find() βρίσκουμε το αντικείμενο και βρίσκοντας την θέση του (`transform.position`) μπορούμε να του πούμε ποια θέλουμε να είναι η νέα θέση του. Αυτό το καταφέρνουμε έχοντας ένα αντικείμενο αναφοράς (reference object), χωρίς renderer για να μην εμφανίζεται, και εξισώνοντας το position του(`GameObject.Find(manager.mud[0]).transform.position=instantiation_node.position;`).

Τέλος για το πρώτο αντικείμενο η μεταβλητή selected_object θα πάρει την τιμή 0 έτσι ώστε όταν θα χρειαστεί να αναφερθούμε σε αυτό το αντικείμενο για να το διαγράψουμε από την σκηνή να μπορούμε εύκολα να το καταφέρουμε. Θα δούμε πως λειτουργεί σε επόμενο script αυτό.

Για να μπορέσει να δημιουργήσει τόσα κουμπιά όσα αντικείμενα έχει μαζέψει ο παίκτης ο κώδικας αυτός θα επαναληφθεί με τις ανάλογες if για να κάνει τους απαραίτητους ελέγχους. Κάθε φορά που μια if ισχύει τότε δημιουργείται και ένα κουμπί στην οθόνη.

```
function OnGUI()
{
  boxStyle3.wordWrap = true;
  boxStyle3.fontSize=18;
```

```
  boxStyle3.normal.background=back;
 GUI.Box(Rect(Screen.width-
Screen.width/3,Screen.heightScreen.height,Screen.width/3,Screen.height/5),"
To rotate the object, use the right mouse button. Using the left mouse
button will remove the mud. Each object cleaned will be displayed in the
displays room. Return to the lab by pressing L.",boxStyle3);


scrollPosition = GUILayout.BeginScrollView (scrollPosition, GUILayout.Width
(Screen.width/6), GUILayout.Height (Screen.height/3));


if(manager.mud.length >=1)
 {
  if(GUILayout.Button (manager.mud[0],boxStyle10))
   {
 GameObject.Find(manager.mud[0]).GetComponent(Rotate).enabled = true;

GameObject.Find(manager.mud[0]).transform.position =
instantiation_node.position;

selected_object_mud = 0;
     }
  }

if(manager.mud.length >=2)
{
  if(GUILayout.Button (manager.mud[1],boxStyle10))
     {
GameObject.Find(manager.mud[1]).GetComponent(Rotate).enabled = true;
 GameObject.Find(manager.mud[1]).transform.position =
instantiation_node.position;

selected_object_mud = 1;
      }
}

if(manager.mud.length >=3)
 {
  if(GUILayout.Button (manager.mud[2],boxStyle10))
{
 GameObject.Find(manager.mud[2]).GetComponent(Rotate).enabled = true;
GameObject.Find(manager.mud[2]).transform.position =
instantiation_node.position;

selected_object_mud = 2;
        }
    }

if(manager.mud.length >=4)
 {
 if(GUILayout.Button (manager.mud[3],boxStyle10))
    {
    GameObject.Find(manager.mud[3]).GetComponent(Rotate).enabled = true;
 GameObject.Find(manager.mud[3]).transform.position =
instantiation_node.position;

selected_object_mud = 3;
        }
}
```

```
if(manager.mud.length >=5)
 {
  if(GUILayout.Button (manager.mud[4],boxStyle10))
   {
GameObject.Find(manager.mud[4]).GetComponent(Rotate).enabled = true;
 GameObject.Find(manager.mud[4]).transform.position =
instantiation_node.position;
selected_object_mud = 4;
         }
    }

if(manager.mud.length >=6)
  {
  if(GUILayout.Button (manager.mud[5],boxStyle10))
   {
  GameObject.Find(manager.mud[5]).GetComponent(Rotate).enabled = true;
 GameObject.Find(manager.mud[5]).transform.position =
instantiation_node.position;
selected_object_mud = 5;
    }
 }
if(manager.mud.length >=7)
 {
 if(GUILayout.Button (manager.mud[6],boxStyle10))
   {
 GameObject.Find(manager.mud[6]).GetComponent(Rotate).enabled = true;
GameObject.Find(manager.mud[6]).transform.position =
instantiation_node.position;
selected_object_mud = 6;
    }
}
if(manager.mud.length >=8)
   {
   if(GUILayout.Button (manager.mud[7],boxStyle10))
   {
   GameObject.Find(manager.mud[7]).GetComponent(Rotate).enabled = true;
  GameObject.Find(manager.mud[7]).transform.position =
instantiation_node.position;
 selected_object_mud = 7;
         }
    }
    GUILayout.EndScrollView ();
}
```

Πίνακας 24– Δημιουργία κουμπιών μέσω της συνάρτησης OnGUI()

Το επόμενο script προς εξέταση θα είναι αυτό το οποίο ενεργοποίησε το Destroy_Dirt. Δηλαδή αυτό το οποίο δίνει την δυνατότητα στον παίκτη να περιστρέφει το αντικείμενο στον χώρο γύρω από τον εαυτό του. Για να δημιουργηθεί αυτός ο μηχανισμός πρέπει να σκεφτούμε τι έχουμε στην διάθεση μας και το χρειαζόμαστε. Αρχικά ο παίκτης θα χειρίζεται ένα τρσιδιάστατο αντικείμενο σε τρισδιάστατο χώρο. Αυτό θα πρέπει να γυρίζει ελεύθερα προς όλες τις κατευθύνσεις για να μπορεί ο παίκτης να εντοπίσει όλα τα σημεία όπου υπάρχει λάσπη. Η κίνηση θα πρέπει να είναι λοιπόν όσο πιο ομαλή γίνεται, άρα ο έλεγχος με το ποντίκι φαίνεται λογική επιλογή. Ας δούμε αρχικά τις μεταβλητές που θα χρειαστούμε.

```
var speed : int;
var friction : float;
var lerpSpeed : float;
```

```
private var xDeg : float;
private var yDeg : float;
private var fromRotation : Quaternion;
private var toRotation : Quaternion;
```

Πίνακας 25– Μεταβλητές για το Rotation Script

Αρχικά έχουμε την μεταβλητή (`var speed : int;`) η οποία είναι ένας ακέραιος που αντιπροσωπεύει την γενική ταχύτητα με την οποία θα γυρίζει το αντικείμενο. Στην συνέχεια είναι λογικό να δώσουμε στην όλη εξομοίωση μια δύναμη τριβής η οποία είναι η μεταβλητή (`var friction : float;`) και είναι δεκαδικός αριθμός. Για να είναι πιο ρεαλιστική η κίνηση καλό είναι να υπάρχει και μια μεταβλητή η οποία θα αντιπροσωπεύει την ταχύτητα ανάμεσα σε δύο σημεία. Αυτή είναι η (`var lerpSpeed : float;`). Επίσης χρειαζόμαστε δύο μεταβλητές για να μπορούμε να αναφερθούμε στις γωνίες περιστροφής. Αυτές θα είναι η (`private var xDeg : float;`) και η (`private var yDeg : float;`). Τέλος θέλουμε να ξέρουμε από ποιο σημείο μέχρι ποιο θα περιστραφεί το αντικείμενο. Το σημείο εκκίνησης θα αντιπροσωπεύεται από την μεταβλητή (`private var fromRotation : Quaternion;`) και το τελικό σημείο από την (`private var toRotation : Quaternion;`). Τα quaternions στο Unity 3d χρησιμοποιούνται για να αντιπροσωπεύουν περιστροφές.

Αφού έχουμε στήσει τις μεταβλητές τις οποίες θα χρειαστούμε το αμέσως επόμενο βήμα είναι να περάσουμε στον κώδικα τιμές για την ταχύτητα, την τριβή και την ταχύτητα ανάμεσα σε δύο σημεία.

```
speed=10;
friction=2;
lerpSpeed=3;
```

Πίνακας 26– Hardcoding τις μεταβλητές

Σε αυτό το σημείο είμαστε έτοιμοι να ασχοληθούμε με την συνάρτηση Update(), η οποία όπως θυμόμαστε, εκτελείται σε κάθε frame του παιχνιδιού και κάνει όλους τους υπολογισμούς που της έχουμε αναθέσει.

```
function Update (){
   if(Input.GetMouseButton(1))
     {
       xDeg -= Input.GetAxis("Mouse X") * speed * friction;
       yDeg += Input.GetAxis("Mouse Y") * speed * friction;
       fromRotation = transform.rotation;
       toRotation = Quaternion.Euler(yDeg,xDeg,0);
       transform.rotation =
Quaternion.Lerp(fromRotation,toRotation,Time.deltaTime  * lerpSpeed);
      }}
```

Πίνακας 27– Update του rotation script

Στην αρχή της Update() συναντάμε την γνώριμη πλέον εντολή (`if(Input.GetMouseButton(1))`) η οποία υποδεικνύει τι θα γίνει, από εκεί και πέρα, σε περίπτωση που ο χρήστης πατήσει το δεξί πλήκτρο του ποντικιού. Αρχικά οι μεταβλητές xDeg και yDeg θα πάρουν τιμές. Η xDeg θα προσαρμοστεί με βάσει τον X άξονα που βρίσκεται ο κέρσορας εκείνη τη στιγμή λαμβάνοντας υπόψη την ταχύτητα και την τριβή που έχουμε ορίσει εμείς πιο πάνω. Στην συνέχεια η μεταβλητή fromRotation η οποία αντιπροσωπεύει το σημείο εκκίνησης περιστροφής θα πάρει την τιμή της γωνίας που έχει εκείνη στιγμή το αντικείμενο μέσω της εντολής (`transform.rotation;`). Έχοντας την γωνία που βρίσκεται το αντικείμενο πρέπει να ορίσουμε την γωνία στην οποία θέλουμε να περιστραφεί τελικά δίνοντας μια τιμή στην μεταβλητή toRotation. Χρησιμοποιώντας τις X και Y μοίρες από πριν και μέσω του Euler θα υπολογιστεί η τελική γωνία (`Quaternion.Euler(yDeg,xDeg,0);`). Το μόνο που απομένει πλέον είναι να πάμε ομαλά από το

fromRotation στο toRotation με την μέθοδο Lerp(). Η μέθοδος αυτή δέχεται ένα σημείο εκκίνησης και ένα σημείο τερματισμού. Έτσι το αντικείμενο θα περιστραφεί ομαλά ανάμεσα σε αυτά τα δύο σημεία(`Quaternion.Lerp(fromRotation,toRotation,Time.deltaTime*lerpSpeed);`).

Όλα καλά λοιπόν με την περιστροφή του αντικειμένου αλλά μέσα σε αυτόν τον κώδικα που βρίσκονται οι εντολές για τον εντοπισμό και την καταστροφή των στοιχείων της λάσπης πάνω στο αντικείμενο; Αυτοί οι υπολογισμοί, προς χάριν ευκολίας, έχουν τοποθετηθεί στο τρίτο και τελικό script για αυτόν το μηχανισμό, το Destroy mud script. Σε αυτό θα δούμε μια καινούργια τεχνική η οποία είναι πάρα πολύ χρήσιμη και ονομάζεται Ray Casting. Ας δούμε και εδώ τι έχουμε στην διάθεση μας και τι θα χρειαστούμε. Το αντικείμενο με τα τρισδιάστατα μοντέλα της λάσπης (Τα οποία είναι ξεχωριστά αντικείμενα με την δική τους υπόσταση αλλά έχουν τεθεί ως παιδιά του αντικειμένου για να περιστρέφονται μαζί του.) βρίσκεται στην σκηνή και ο παίκτης χρησιμοποιώντας τον κέρσορα μπορεί και περιστρέφει το αντικείμενο.

Θα είναι πολύ βολικό αν καθώς ο παίκτης περνάει τον κέρσορα πάνω από το αντικείμενο, αν γίνεται να αναγνωρίζει το script τι είναι λάσπη και ποιο είναι το αρχαιολογικό αντικείμενο έτσι ώστε με το πάτημα του αριστερού κουμπιού του ποντικιού η λάσπη να διαλύεται σταδιακά. Αυτός ο τρόπος υπάρχει και μας δίνει την δυνατότητα αυτή η μέθοδος του Ray Casting, η οποία πετάει μια ακτίνα από το σημείο εκκίνησης που θέλουμε (στην συγκεκριμένη περίπτωση, την κάμερα) και εκεί όπου θα καταλήξει (στην συγκεκριμένη περίπτωση το σημείο που βρίσκεται ο κέρσορας), να μας δώσει όποια πληροφορία θέλουμε για το αντικείμενο αυτό. Επειδή εμείς θέλουμε να μπορούμε να αναγνωρίζουμε την λάσπη μέσω του Ray Casting θα ονομάσουμε το tag των game objects της λάσπης ως mud. Ας ξεκινήσουμε με τις μεταβλητές που θα χρησιμοποιήσουμε.

```
var ray : Ray;
var hit : RaycastHit;
var cnt: int;
var Sound_mud: AudioClip;
```

Πίνακας 28– Μεταβλητές του Destroy mud script

Στην προηγούμενη παράγραφο αναλύσαμε πως λειτουργεί το Ray Casting, άρα η πρώτη μας μεταβλητή θα είναι η (`var ray : Ray;`), η οποία αντιπροσωπεύει την ακτίνα που θα ξεκινάει από την κάμερα και θα καταλήγει στο σημείο που δείχνει ο κέρσορας. Άρα πλέον χρειαζόμαστε και μια μεταβλητή μέσω της οποίας θα μπορούμε να αναγνωρίσουμε το αντικείμενο που χτύπησε η ακτίνα. Αυτή θα είναι η (`var hit : RaycastHit;`). Χρειαζόμαστε και έναν ακέραιο ο οποίος θα αυξάνεται κάθε φορά που ο παίκτης θα πατάει το αριστερό πλήκτρο του ποντικιού για να καταστρέψει, σταδιακά, ένα κομμάτι λάσπης. Αυτή είναι η (`var cnt: int;`) και είναι ένας ακέραιος. Τέλος επειδή θέλουμε να ακούγεται κάποιος ήχος χρειαζόμαστε την μεταβλητή (`var Sound_mud: AudioClip;`) στην οποία θα αποθηκεύσουμε από τον editor τον ήχο που επιθυμούμε.
Στην συνέχεια καλούμε την συνάρτηση onMouseOver() η οποία είναι υπεύθυνη για οτιδήποτε συμβαίνει όταν ο κέρσορας περνάει πάνω από κάποιο αντικείμενο.

```
function OnMouseOver()
{
    ray= Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, hit, Mathf.Infinity))
    {
        if(hit.transform.tag=="mud")
        {
            mud=hit.collider.gameObject;
            if(Input.GetMouseButtonDown(0) && cnt<=3)
            {
                mud.transform.localScale/=1.5;
                audio.PlayOneShot(Sound_mud);
```

```
                cnt=cnt+1;
            }
            if(cnt==3)
            {
                Destroy(mud.gameObject);
                audio.PlayOneShot(Sound_mud);
            }
        }
                if(cnt == 3)
                {
                    manager.mud.RemoveAt(destroy_Dirt.selected_object_mud);
      manager.display_mud.Push(this.transform.root.gameObject.name);
                    Destroy(this.transform.root.gameObject, 3);
                    cnt=0;
                }
    }
}
```

Πίνακας 29– Εντοπισμός και καταστροφή λάσπης

Το πρώτο πράγμα που πρέπει να γίνει είναι να οριστεί η ακτίνα από πού θα ξεκινάει και που θα τελειώνει. Χρησιμοποιούμε την μέθοδο ScreenPointToRay() με όρισμα το σημείο που βρίσκεται ο κέρσορας εκείνη τη στιγμή (`Camera.main.ScreenPointToRay(Input.mousePosition);`). Αυτή η γραμμή κώδικα μας λέει ότι η ακτίνα ξεκινάει από σημείο της οθόνης και θα καταλήγει στον κέρσορα. Η αμέσως επόμενη συνθήκη if υπάρχει για να διαπιστώσει αν η ακτίνα (ray) έχει χτυπήσει κάτι (hit) έχοντας δώσει περιθώριο άπειρης απόστασης(Mathf.Infinity). Ο κώδικας πλέον θα μπει μέσα στην if αν και μόνο αν έχει χτυπήσει κάτι. Για να διαπιστώσουμε ποιο είναι το tag του αντικειμένου που χτύπησε προσθέτουμε την επόμενη if. Εάν η μεταβλητή hit, μέσω της οποίας μπορούμε να προσπελάσουμε τις πληροφορίες του αντικειμένου που χτυπήθηκε, αντιπροσωπεύει ένα game object με tag που ονομάζεται 'mud' (`if(hit.transform.tag=="mud")`), τότε θα αποθηκευτεί σε μια μεταβλητή τύπου Game object που ονομάζουμε mud το αντικείμενο αυτό (`mud=hit.collider.gameObject;`).

Πλέον έχουμε όλες τις πληροφορίες που χρειαζόμαστε για να μπορέσουμε να χειριστούμε και να μικρύνουμε τα αντικείμενα λάσπης μέχρι να εξαφανιστούν. Εδώ θα χρειαστούμε την μεταβλητή cnt, που μετράει πόσες φορές έκανε κλικ ο παίκτης για να μικρύνει την λάσπη, και κάποιες if συνθήκες για να χειριστούμε όλες τις περιπτώσεις. Η λάσπη θέλουμε να μειώνεται κατά μιάμιση φορά κάθε φορά που επιλέγει ο χρήστης να κάνει κλικ πάνω της. Επίσης θέλουμε στην τρίτη σμίκρυνση η λάσπη να εξαφανίζεται. Άρα η πρώτη if συνθήκη πρέπει να ελέγχει αν ο παίκτης έχει κάνει κλικ και ο μετρητής μας είναι μικρότερος από τον αριθμό τρία (`if(Input.GetMouseButtonDown(0) && cnt<=3)`). Σε αυτήν την περίπτωση το game object που αντιπροσωπεύει την λάσπη πρέπει να μικρύνει (`mud.transform.localScale/=1.5;`), να παίξει ο αντίστοιχος ήχος αφαίρεσης της λάσπης (`audio.PlayOneShot(Sound_mud);`), και ο μετρητής να αυξηθεί κατά ένα (`cnt=cnt+1;`).

Αν ο μετρητής φτάσει τον αριθμό τρία, δηλαδή ο παίκτης έχει κάνει κλικ τρείς φορές πάνω στην λάσπη, τότε πρέπει το game object της λάσπης να διαγραφεί από το scene μέσω της συνάρτησης Destroy() που είδαμε και σε προηγούμενο κεφάλαιο (`Destroy(mud.gameObject);`). Για το επόμενο βήμα θα παίξει ρόλο η μεταβλητή με template selected_object που έχουμε βάλει στο κάθε script δημιουργίας των κουμπιών για την προσπέλαση αντικειμένων. Για το συγκεκριμένο mode το script αυτό είναι το destroy_dirt. Ο ακέραιος αυτός χρησιμοποιείται για να ξέρουμε ανά πάσα στιγμή την θέση του ονόματος, του αρχαιολογικού αντικειμένου που επεξεργάζεται ο παίκτης, στον πίνακα που έχει αποθηκευτεί μόλις ο παίκτης το μάζεψε. Έτσι μπορούμε τώρα να καθαρίσουμε την θέση του πίνακα αυτού χρησιμοποιώντας την μέθοδο RemoveAt() έχοντας ως αναφορά τον πίνακα mud του script manager όπου βρίσκεται ο πίνακας (`RemoveAt(destroy_Dirt.selected_object_mud);`).

Πλέον το αντικείμενο είναι καθαρό και είναι έτοιμο να μπει στην αίθουσα των εκθεμάτων. Άρα όπως αναφέραμε σε προηγούμενο κεφάλαιο, το όνομα του αντικειμένου αυτού θα περάσει στο

display πίνακα της κατηγορίας του μέσω της μεθόδου Push έχοντας σαν αναφορά τον πίνακα display_mud (`manager.display_mud.Push(this.transform.root.gameObject.name);`). Το μόνο που μένει είναι να καταστραφεί το αντικείμενο από την σκηνή μέσω της Destroy() (βάζοντας στα ορίσματα έναν αριθμό θα περιμένει τόσα δευτερόλεπτα πριν το καταστρέψει) και ο μετρητής να ξαναμηδενιστεί για το επόμενο αντικείμενο.

5.4.2 Scanner Mode

Σε αυτό το υποκεφάλαιο θα ασχοληθούμε με τον μηχανισμό του scanner. Σε αυτόν ο παίκτης διαλέγει τα αντικείμενα, τα οποία βρίσκονται μέσα σε πετρώματα, από την λίστα και αυτά καθώς φτάνουν στην οθόνη του scanner αποκαλύπτουν το καθαρό αντικείμενο. Πριν ξεκινήσουμε την ανάλυση του κώδικα θα δούμε τον τρόπο με τον οποίο καθίσταται δυνατό αυτό.

Όλα τα αρχαιολογικά αντικείμενα στην σκηνή αυτή αποτελούνται από το αρχαιολογικό αντικείμενο προς εξέταση και άλλα δυο game objects (τα πετρώματα). Αυτά, στην ουσία, είναι το ίδιο game object το οποίο έχει προστεθεί σαν παιδί δυο φορές. Η μια έκδοση του πετρώματος είναι κανονική ώστε να είναι ορατή στον παίκτη με τον απλό diffuse shader. Η δεύτερη έκδοση έχει έναν διάφανο shader ώστε το πέτρωμα να μην είναι ορατό στο μάτι.

Μέσα στο scanner και στο σημείο στο οποίο θα σταματήσει το αντικείμενο έχει προστεθεί μια δεύτερη κάμερα η οποία βρίσκεται εκεί για να εμφανίζει το αντικείμενο μέσω ενός render texture το οποίο είναι τοποθετημένο στην οθόνη του scanner. Η λειτουργία ενός render texture είναι να προβάλει οτιδήποτε δείχνει μια κάμερα που έχουμε επιλέξει εμείς. Στην συγκεκριμένη περίπτωση, η κάμερα που βρίσκεται εντός του scanner.

Όμως αυτό δεν είναι αρκετό για να μπορέσουμε στην οθόνη του scanner να δούμε το αντικείμενο γυμνό. Εδώ φάνηκε χρήσιμη η χρήση των layers. Όπως αναθέτουμε στα game objects tags για να μπορούμε να τα ξεχωρίζουμε και να τα αναγνωρίζουμε βάση αυτών, μπορούμε να τους αναθέσουμε και σε ποιο layer θα βρίσκονται. Όπως και στο photoshop μπορούμε να επιλέξουμε ποια layers θα είναι ορατά, έτσι και στο Unity 3d μπορούμε να ορίσουμε, σε μια κάμερα, ποιο layer θα κάνει Render, δηλαδή ποιο θα εμφανίσει. Έτσι λοιπόν ορίζοντας στην εσωτερική κάμερα να κάνει render μονάχα τα αντικείμενα που βρίσκονται στο layer default (δηλαδή τους βράχους που είναι διάφανοι και τα αρχαιολογικά αντικείμενα που εσωκλείουν) καταφέρνουμε να εμφανίσουμε στην οθόνη του scanner (που της έχει ανατεθεί το render texture), το αντικείμενο γυμνό.

Επειδή σε αυτό το mode θέλουμε τα αντικείμενα να μεταφέρονται ομαλά από την τοποθεσία που είναι κρυμμένα, προς το τέλος του Scanner που βρίσκεται η οθόνη που θα προβληθούν, η λειτουργικότητα του είναι σχετικά απλή και χωρίζεται σε δύο scripts. Αυτά είναι το scanner script και το scanning script.

Το scanner script είναι αυτό το οποίο θα αναλάβει να δημιουργήσει όλα τα απαραίτητα κουμπιά ανάλογα με το πόσα συμβατά αντικείμενα μάζεψε ο παίκτης και να ενεργοποιήσει το scanning script με το που πατηθεί ένα από αυτά τα κουμπιά. Αυτό με την σειρά του θα αναλάβει να μεταφέρει το εν λόγω αντικείμενο, ομαλά με την χρήση της συνάρτησης Lerp(), στο τέλος της διαδρομής του, δηλαδή στην οθόνη του Scanner. Όπως θα παρατηρήσουμε το Scanner Script λειτουργικά είναι παρόμοιο με το Destroy Dirt script.

```
private var scrollPosition : Vector2;
static var selected_object: int;
var boxStyle4 : GUIStyle;
var boxStyle10 : GUIStyle;
var back:Texture2D;

var object_1: Transform;
var object_2: Transform;
var object_3: Transform;
var object_4: Transform;
var object_5: Transform;
var object_6: Transform;
var object_7: Transform;
```

```
function OnGUI()
{


//boxStyle="box";
boxStyle4.wordWrap = true;
boxStyle4.fontSize=18;

boxStyle4.normal.background=back;

GUI.Box(Rect(Screen.width-Screen.width/3,Screen.height-
Screen.height,Screen.width/3,Screen.height/5),"To scan an object simply
select it. Each object scanned will be displayed in the displays
room.Return to the lab by pressing L.",boxStyle4);


    scrollPosition = GUILayout.BeginScrollView (scrollPosition,
GUILayout.Width (Screen.width/6), GUILayout.Height (Screen.height/3));

if(manager.scanner.length >=1)
    {
        if(GUILayout.Button (manager.scanner[0],boxStyle10))
        {


GameObject.Find(manager.scanner[0]).GetComponent(scanning).enabled = true;
            selected_object = 0;
        }
    }

    if(manager.scanner.length >=2)
    {
        if(GUILayout.Button (manager.scanner[1],boxStyle10))
        {

GameObject.Find(manager.scanner[1]).GetComponent(scanning).enabled = true;
            selected_object = 1;
        }
    }

    if(manager.scanner.length >=3)
    {
        if(GUILayout.Button (manager.scanner[2],boxStyle10))
        {

GameObject.Find(manager.scanner[2]).GetComponent(scanning).enabled = true;
            selected_object = 2;
        }
    }

    if(manager.scanner.length >=4)
    {
        if(GUILayout.Button (manager.scanner[3],boxStyle10))
        {

GameObject.Find(manager.scanner[3]).GetComponent(scanning).enabled = true;
            selected_object = 3;
        }
    }
```

```
    if(manager.scanner.length >=5)
    {
        if(GUILayout.Button (manager.scanner[4],boxStyle10))
        {

GameObject.Find(manager.scanner[4]).GetComponent(scanning).enabled = true;
            selected_object = 4;
        }
    }

    if(manager.scanner.length >=6)
    {
        if(GUILayout.Button (manager.scanner[5],boxStyle10))
        {

GameObject.Find(manager.scanner[5]).GetComponent(scanning).enabled = true;
            selected_object = 5;
        }
    }

    if(manager.scanner.length >=7)
    {
        if(GUILayout.Button (manager.scanner[6],boxStyle10))
        {

GameObject.Find(manager.scanner[6]).GetComponent(scanning).enabled = true;
            selected_object = 6;
        }
    }

    if(manager.scanner.length >=8)
    {
        if(GUILayout.Button (manager.scanner[7],boxStyle10))
        {

GameObject.Find(manager.scanner[7]).GetComponent(scanning).enabled = true;
            selected_object = 7;
        }
    }
    GUILayout.EndScrollView ();
}
```

Πίνακας 30- Scanner script

Στη συνέχεια θα αναλύσουμε την λειτουργικότητα του Scanning script. Εδώ τα πράγματα είναι σαφώς πιο απλά γιατί ξέρουμε ήδη την αρχική θέση του κάθε αντικειμένου και το μόνο που πρέπει να προσθέσουμε στην σκηνή είναι ένα empty game object μέσα στο scanner, στην θέση που θέλουμε να σταματάει το αντικείμενο (ending). Επίσης, θέλουμε μια μεταβλητή τύπου Audio Clip ώστε να της αναθέσουμε στον editor το ηχητικό κλιπάκι που θα παίζει μόλις φτάσει το αντικείμενο στον τελικό προορισμό του (Sound_scanner). Άρα οι μεταβλητές που θα ορίσουμε είναι οι παρακάτω.

```
var ending : Transform;
var Sound_scanner: AudioClip;
```

Πίνακας 31- Μεταβλητές Scanning script

Αφού έχουμε ορίσει τις μεταβλητές που θα χρειαστούμε θα δούμε ποιες συναρτήσεις θα χρησιμοποιήσουμε. Αρχικά κάθε φορά που ένα αντικείμενο εμφανίζεται στο προσκήνιο πρέπει

απευθείας να ξεκινάει την πορεία του προς το ending game object όπου και θα σταματήσει. Άρα ο υπολογισμός της πορείας θα πρέπει να μπει στην Update() του script.

```
function Update ()
{
    transform.position = Vector3.Lerp(transform.position, ending.position,
Time.deltaTime * 0.8);
}
```

Πίνακας 32- Χρήση Lerp() για μεταφορά αντικειμένου ανάμεσα σε δυο Nodes

Έτσι λοιπόν βλέπουμε πως η θέση που θα έχει το αντικείμενο (`transform.position`) σε κάθε frame αλλάζει ανάλογα με αυτό που ορίζει η συνάρτηση Lerp(). Το πρώτο όρισμα που θα πάρει είναι η τρέχουσα θέση του αντικειμένου σε κάθε frame (`transform.position`), το δεύτερο όρισμα είναι η θέση που βρίσκεται το Game object ending (`ending.position`) και τέλος ορίζουμε τον χρόνο με τον οποίο θέλουμε να μετακινείται το αντικείμενο(`Time.deltaTime * 0.8`).

Πλέον έχουμε καλύψει τον τρόπο με τον οποίο γίνεται η μετακίνηση του αντικειμένου. Μόλις όμως φτάσει το αντικείμενο στον τελικό προορισμό του πρέπει, το script, να έχει έναν τρόπο να καταλάβει πότε έφτασε για να πάρει τις απαραίτητες ενέργειες. Σε αυτό το σημείο θα χρησιμοποιήσουμε την παραλλαγή μιας συνάρτησης που έχουμε ξανασυναντήσει. Σε προηγούμενο κεφάλαιο είδαμε τις OnTriggerStay() και OnTriggerExit(). Η πρώτη έλεγχε το collision όσο κάποιο αντικείμενο ήταν μέσα στην περιοχή του collider και η δεύτερη έλεγχε αν έχει βγει από την περιοχή αυτή. Στην συγκεκριμένη περίπτωση θα χρησιμοποιήσουμε άλλη μια παραλλαγή αυτών.

```
function OnTriggerEnter(other : Collider)
{
    if (other.gameObject.tag=="scanner_endpoint_detector")
    {
        audio.PlayOneShot(Sound_scanner);
        manager.scanner.RemoveAt(scanner.selected_object);
        Wait_please();
    }
}
```

Πίνακας 33- Χρήση της OnTriggerEnter

Η OnTriggerEnter() κάνει έλεγχο μόνο την στιγμή που γίνεται το collision των δυο αντικειμένων. Άρα αυτή η συνάρτηση θα δουλέψει με το που το αρχαιολογικό αντικείμενο χτυπήσει τον collider του ending game object. Με το που γίνει αυτό στη μεταβλητή other, που δέχεται σαν όρισμα(`OnTriggerEnter(other : Collider)`), θα ανατεθεί ο collider του ending αντικειμένου.

Έτσι ερχόμαστε στην πρώτη if συνθήκη η οποία ελέγχει αν το tag του αντικειμένου που χτυπήθηκε έχει το string 'scanner_endpoint_detector'. Αν η συνθήκη ισχύει, κάτι το οποίο σημαίνει ότι το αντικείμενο έφτασε στο τέλος της διαδρομής, τότε θα συμβούν τρία πράγματα. Θα παίξει ο κατάλληλος ήχος (`audio.PlayOneShot(Sound_scanner);`), θα αφαιρεθεί το string από τον πίνακα scanner(`manager.scanner.RemoveAt(scanner.selected_object);`) και θα κληθεί η συνάρτηση Wait_please().

```
function Wait_please()
{
    yield WaitForSeconds (5);
    manager.display_scanner.Push(this.gameObject.name);
    Destroy(this.gameObject);
}
```

Πίνακας 34- Custom συνάρτηση Wait_please()

Η συνάρτηση Wait_Please() εξυπηρετεί με την σειρά της τρεις σκοπούς. Αρχικά αναγκάζει το script να κάνει μια παύση πέντε δευτερολέπτων (`yield WaitForSeconds (5);`), πριν περάσει στην παρακάτω γραμμή κώδικα, προωθεί το όνομα του αντικειμένου στον πίνακα display() μέσω της Push(), μιας και είναι έτοιμο για να μπει στην αίθουσα εκθεμάτων και καταστρέφει το αντικείμενο μέσω της Destroy().

5.4.3 Dust removal mode

Στον μηχανισμό καθαρισμού σκόνης, ο παίκτης επιλέγει τα αντικείμενα που έχει συλλέξει και αφού αυτά τοποθετούνται κατάλληλα στην σκηνή, χρησιμοποιώντας το space bar τα καθαρίζει σταδιακά από την σκόνη που έχει συσσωρευτεί. Για να δημιουργηθεί η αίσθηση στον παίκτη ότι ξεσκονίζει τα αντικείμενα θα χρειαστούμε δυο πράγματα. Έναν δικό μας shader ο οποίος δέχεται σαν input δυο materials. Το ένα είναι το σκονισμένο και το δεύτερο είναι το καθαρό τελικό. Μια μεταβλητή float η οποία, όταν χρησιμοποιηθεί μέσω script, καθώς μειώνεται, μέσω μιας διαδικασίας που θα αναλύσουμε στο τέλος του υποκεφαλαίου, εμφανίζοντας σταδιακά το καθαρό. Άλλο ένα στοιχείο που θα πρέπει να προσέξουμε είναι να μπορέσουμε να παραστήσουμε γραφικά την σκόνη χρησιμοποιώντας particles. Τέλος η προσθήκη ήχου την στιγμή που ο παίκτης πατάει το space bar θα προσδώσει στην αίσθηση του καθαρισμού.

Έχοντας ορίσει ότι ο dirty shader θα χρησιμοποιηθεί στα αντικείμενα, θα πρέπει να γράψουμε δύο script. Το πρώτο θα είναι το dirty script. Αυτό είναι το γνώριμο πλέον script για την δημιουργία των κουπιών, ην ανάθεση των ονομάτων σε αυτά και τη χρησιμοποίηση της μεταβλητής selectedobject. Το δεύτερο είναι το dirtying, το οποίο θα αναλάβει για κάθε αντικείμενο που έχει έρθει στο προσκήνιο, να ελέγχει το ποσό της σκόνης και μόλις αυτό καθαρίσει τελείως να το διαγράψει από την σκηνή και να περάσει το όνομα του στον πίνακα display_dirty.

Όσον αφορά το dirty script δεν θα το εισάγουμε όλο αφού το έχουμε ξανασυναντήσει σε παρόμοιες μορφές. Στον παρακάτω πίνακα θα δείξουμε σαν παράδειγμα μόνο μια εμφωλευμένη if συνθήκη για να θυμηθούμε πως συντάσσεται και ποιες μεταβλητές χρησιμοποιούνται.

```
if(manager.dirty.length >=1)
    {
        if(GUILayout.Button (manager.dirty[0],boxStyle10))
        {

GameObject.Find(manager.dirty[0]).GetComponent(dirtying).enabled = true;
            GameObject.Find(manager.dirty[0]).transform.position =
instantiation_node.position;
            selected_object_dirty = 0;
        }
    }
```

Πίνακας 35- Στιγμιότυπο του Dirt Script

Για το dirtying script θα αναλύσουμε αρχικά τι μεταβλητές θα χρειαστούμε. Εδώ ο κώδικας δεν κάνει σύνθετα πράγματα οπότε θα χρειαστούμε μια float μεταβλητή για να μπορεί το script να επικοινωνεί με τη μεταβλητή του shader που ρυθμίζει το alpha του βρόμικου texture, μια μεταβλητή για να μπορούμε να ορίσουμε που θα ποια θα είναι τα particles που θα εμφανιστούν και μια μεταβλητή για να αποθηκεύσουμε τον ήχο που θέλουμε να αναπαράγεται.

```
var dirty_amount: float = 1.0;
var particle_instantiation : GameObject;
var sound_clean: AudioClip;
```

Πίνακας 36- Στιγμιότυπο του Dirt Script

Τώρα μπορούμε να προχωρήσουμε στην ανάλυση του υπόλοιπου κώδικα, ξεκινώντας από την Update() η οποία και φαίνεται στον πίνακα παρακάτω.

```
function Update ()
{

      if(this.dirty_amount>=0.1 && Input.GetKeyUp(KeyCode.Space))
            {
                Instantiate(particle_instantiation, transform.position,
Quaternion.identity);
                audio.PlayOneShot(sound_clean);
                dirty_amount = dirty_amount - 0.1;
                renderer.material.SetFloat("_dirtiness",dirty_amount);

            }

      else if(this.dirty_amount<=0.1 && Input.GetKeyUp(KeyCode.Space))
            {

                Wait_please();
            }


}
```

Πίνακας 37- Συνάρτηση Update() του dirtying script

Η πρώτη συνάρτηση που θα χρησιμοποιήσουμε σίγουρα είναι η Update() γιατί θέλουμε σε κάθε frame του παιχνιδιού να γίνεται έλεγχος αν πατήθηκε το space bar και αν καλύπτονται οι συνθήκες για να ξεκινήσει καθαρισμός. Αρχικά πρέπει να θέσουμε ποιο θα είναι το όριο στο οποίο θα θεωρείται το αντικείμενο καθαρό. Με άλλα λόγια, μέχρι ποια τιμή θα φτάσει η μεταβλητή dirty_amount.

Αυτό σημαίνει ότι θέλουμε μια if συνθήκη η οποία θα πραγματοποιεί αυτόν τον έλεγχο αν και μόνο αν έχει πατηθεί το space bar (`if(this.dirty_amount>=0.1&& Input.GetKeyUp(KeyCode.Space))`). Τα particles θα εμφανιστούν μέσω της συνάρτησης `Instantiate(particle_instantiation,transform.position,Quaternion.identity);` Το πρώτο όρισμα είναι η μεταβλητή την οποία συσχετίσαμε με τα particles που θέλουμε να εμφανιστούν. Το δεύτερο όρισμα θα τα εμφανίσει στην θέση που βρίσκεται το αντικείμενο που καθαρίζει ο παίκτης και το τρίτο ρυθμίζει το rotation. Στην συνέχεια θα παίξει ο ήχος που ορίσαμε στον editor (`audio.PlayOneShot(sound_clean);`). Πολύ σημαντικό είναι να μειώσουμε την μεταβλητή dirty_amount κατά 0.1 ώστε να αρχίσει να εμφανίζεται το καθαρό texture (`dirty_amount = dirty_amount - 0.1;`). Επειδή όμως αυτή η μεταβλητή ορίζει την αντίστοιχη τιμή στον shader, πρέπει να ανανεωθεί και εκεί για να δούμε το αποτέλεσμα. Θα πρέπει να επικοινωνήσουμε με το material του shader μέσω του renderer και καλώντας την μέθοδο setFloat() θα πάρει την νέα τιμή (`renderer.material.SetFloat("_dirtiness",dirty_amount);`).

Όταν η τιμή του dirty_amount γίνει μικρότερη του 0.1, δηλαδή το αντικείμενο έχει καθαριστεί θα πρέπει να ορίσουμε άλλη μια συνθήκη η οποία θα ολοκληρώσει την διαδικασία διαγράφοντας το αντικείμενο και ενημερώνοντας τον πίνακα display_dirty (). Αυτό θα γίνει καλώντας την custom συνάρτηση Wait_please().

```
function Wait_please()
{
    yield WaitForSeconds (2);
    manager.dirty.RemoveAt(dirty.selected_object_dirty);
    manager.display_dirty.Push(this.gameObject.name);
    Destroy(this.gameObject);

}
```

Πίνακας 38– Συνάρτηση Wait_please() του dirtying script

Με το που κληθεί η Wait_please() το script θα περιμένει δυο λεπτά μέσω της (`yield WaitForSeconds (2);`), θα αφαιρέσει το όνομα του αντικειμένου που καθαρίστηκε από τον αρχικό πίνακα (`manager.dirty.RemoveAt(dirty.selected_object_dirty);`), θα προωθήσει το όνομα στον πίνακα display_dirty (`manager.display_dirty.Push(this.gameObject.name);`) και τέλος θα καταστρέψει το αντικείμενο.

Πριν κλείσουμε αυτό το υποκεφάλαιο θα εξηγήσουμε πως δημιουργήθηκε ο shader που χρησιμοποιήσαμε. Με τον Strumpy Shader Editor μπορούμε να δημιουργήσουμε custom shaders χωρίς την χρήση της γλώσσας CG που έχει περίπλοκη δομή και σύνταξη. Ο Strumpy Shader Editor είναι δωρεάν εργαλείο που μπορεί κανείς να το βρει στο Unity Asset Store. Με αυτό το εργαλείο δημιουργούμε shaders με έναν απλό Graph Editor ο οποίος μας επιτρέπει να συνδέουμε σε γραφικό περιβάλλον κόμβους (nodes), ιδιότητες (properties), σταθερές (constants) και συναρτήσεις (functions) οι οποίες είναι διαθέσιμες από το API του Unity. Με αυτόν τον τρόπο μπορούμε να κατασκευάσουμε ακόμη και τους πιο περίπλοκους shaders με κόμβους αποφεύγοντας τον κώδικα σε CG. Το πρόβλημα είναι ο χρήστης θα πρέπει να έχει κάποιες βασικές γνώσεις πάνω στην γλώσσα CG, διαφορετικά δεν θα μπορεί να κάνει χρήση των κατάλληλων κόμβων και συναρτήσεων του SSE. Ο SSE κάνει χρήση των αυθεντικών ιδιοτήτων (τόσο στο όνομα, όσο και στην λειτουργία) της γλώσσας CG.

Για τον dirty_shader θα χρειαστούμε 3 ιδιότητες τις οποίες προσθέτουμε από το μενού Inputs:

- Το βασικό main_texture
- Το dirty_texture
- Το ποσοστό που θα ορίζει δυναμικά πόσο «dirty» είναι το main_texture (dirtiness).

Το dirtiness είναι ιδιότητα τύπου "Range" και μπορεί να πάρει τιμές από Α έως Β. Εδώ έχουμε βάλει τις αρχικές τιμές από 0 έως 1.

Σε αυτό το σημείο ορίσαμε τις 3 ιδιότητες οι οποίες θα είναι εμφανείς ως επιλογές πλέον στο material που θα κάνει χρήση τον shader αυτόν. Στο Graph Editor του SSE πατώντας δεξί κλικ σε κενό χώρο εμφανίζονται οι επιλογές για το τι θέλουμε να προσθέσουμε στο Graph. Από αυτό το μενού μπορούμε να προσθέσουμε όλες τις διαθέσιμες ιδιότητες, σταθερές, συναρτήσεις κλπ.

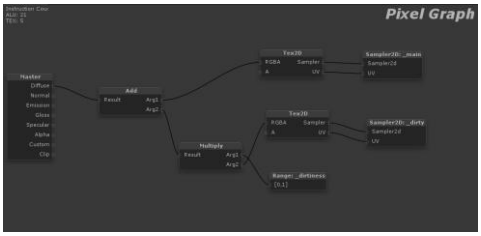
Από το Properties προσθέτουμε 2 Sampler2D και από το Functions προσθέτουμε 2 Tex2D. Για να μπορεί να εμφανίζει ένα υποτυπώδες texture ο shader χρειάζεται έναν Sampler2D και ένα Tex2D σαν συνάρτηση.

Από το μενού Nodes και έχοντας επιλεγμένο έναν Sampler2D ορίζουμε πιο Input θα κάνει χρήση. Επιλέγουμε τόσο για το main Sampler2D όσο και για το dirty Sampler2D την αντίστοιχη ιδιότητα.

Στον Graph Editor κάνουμε ένωση τον κάθε Sampler2D με το Tex2D τόσο για το main όσο και για το dirty. Πλέον στον Graph Editor θα έχουμε το παρακάτω:

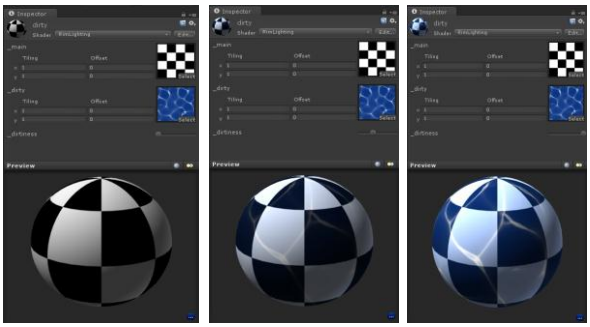
Ο shader στην παρούσα φάση μπορεί να φορτώσει 2 textures. Για την κατασκευή του dirty shader θα πρέπει να μπορούμε να ορίσουμε το ποσοστό του δεύτερου texture (dirty) έναντι του πρώτου texture (main). Για να επιτευχθεί αυτό θα πρέπει να πολλαπλασιάζουμε την ιδιότητα Range με το dirty texture.

Τέλος προσθέτουμε το αρχικό texture με το πολλαπλασιασμένο με την συνάρτηση Add. Ακόμη θα ενώσουμε το τελικό αποτέλεσμα με κανάλι Diffuse του shader. Ο Graph Editor θα έχει πλέον τα εξής:



Εικόνα 9 – Ολοκληρωμένο γράφημα

Αφού κάνουμε export τον shader θα είναι πλέον διαθέσιμος να τον προσθέσουμε σε όποιο material του Unity θέλουμε. Παρακάτω βλέπουμε παράδειγμα χρήσης του shader σε ένα material του Unity από τον Inspector:



Εικόνα 10 – Τιμές Dirtiness, 0, 0.35 και 1 αντίστοιχα

Με την χρήση του SSE αποφύγαμε τον κώδικα σε CG ο οποίος για τον dirty shader είχε μεταξύ άλλων τα εξής:

```
c.a = s.Alpha;
return c;

        }

        inline half4 LightingBlinnPhongEditor (EditorSurfaceOutput s, half3 lightDir, half3 viewDir, half atten)
        {
            half3 h = normalize (lightDir + viewDir);

            half diff = max (0, dot ( lightDir, s.Normal ));

            float nh = max (0, dot (s.Normal, h));
            float spec = pow (nh, s.Specular*128.0);

            half4 res;
            res.rgb = _LightColor0.rgb * diff;
            res.w = spec * Luminance (_LightColor0.rgb);
            res *= atten * 2.0;

            return LightingBlinnPhongEditor_PrePass( s, res );
        }

        struct Input {
            float2 uv_main;
float2 uv_dirty;

        };

        void vert (inout appdata_full v, out Input o) {
float4 VertexOutputMaster0_0_NoInput = float4(0,0,0,0);
float4 VertexOutputMaster0_1_NoInput = float4(0,0,0,0);
float4 VertexOutputMaster0_2_NoInput = float4(0,0,0,0);
float4 VertexOutputMaster0_3_NoInput = float4(0,0,0,0);
```

Εικόνα 11 – Κώδικας για Shader CG

5.4.4 Fragments recomposing mode

Το τέταρτο και τελευταίο mode προς ανάλυση είναι αυτό στο οποίο ο παίκτης επιλέγει αντικείμενα (όπως ταμπλέτες), τα οποία έχουν σπάσει σε κάποια κομμάτια και χρησιμοποιώντας τον κέρσορα πρέπει να συναρμολογήσει το τελικό αντικείμενο βάζοντας τα στην σωστή θέση. Εδώ θα χρειαστούμε τρία scripts. Το Breaking script είναι αυτό που ρυθμίζει τα αντικείμενα που βρίσκονται στην σκηνή, προσθέτοντας κουμπιά κτλ. Το Drag script εκτελεί όλες τις απαραίτητες εντολές για να μπορεί ο παίκτης να σέρνει τα διάφορα κομμάτια στον χώρο και να τα ενώνει μόλις αυτά φτάσουν σε ένα κοντινό σημείο που έχουμε ορίσει εμείς. Τέλος, το broken_main script αναλαμβάνει την διαχείριση των ξεχωριστών κομματιών του αντικειμένου και ενεργοποιεί το drag script στα αντικείμενα αυτά.

Όσον αφορά το Breaking script δεν θα αναφερθούμε εκτενώς αφού είναι και αυτό μια παραλλαγή των script που είδαμε πιο πάνω, για την δημιουργία κουμπιών και την διαχείριση των αντικειμένων που βρίσκονται στην σκηνή. Θα παραθέσουμε απλά ένα παράδειγμα του κώδικα, για να υπάρχει ως αναφορά.

```
if(manager.broken.length >=1)
    {
    if(GUILayout.Button (manager.broken[0],boxStyle10))

{GameObject.Find(manager.broken[0]).GetComponent(broken_main).enabled =
true;
            breaking_selected_object = 0;
        }
    }
```

Πίνακας 39– Κομμάτι του Breaking script

Ξεκινώντας την ανάλυση του Drag script θα πρέπει να σκεφτούμε και να αναλύσουμε τι μεταβλητές θα πρέπει να χρησιμοποιήσουμε. Όπως και στο Rotate script όλη η λειτουργικότητα της κίνησης των αντικειμένων έγκειται στον κέρσορα του ποντικιού. Άρα σίγουρα θα χρειαστεί να χρησιμοποιήσουμε τουλάχιστον έναν πίνακα τριών θέσεων για να αποθηκεύσουμε θέσεις αντικειμένων.

```
private var offset: Vector3;
private var curScreenPoint : Vector3;
static var curPosition : Vector3;
private var screenPoint : Vector3;
var myPos : Vector3;
var Sound_destroy: AudioClip;
```

Πίνακας 40– Μεταβλητές του Drag script

Η μεταβλητή (`private var offset: Vector3;`) θα χρησιμοποιηθεί αργότερα για να βρίσκουμε το offset ανάμεσα στην θέση του αντικειμένου που θέλουμε να σύρουμε και το world point που βρίσκεται ο κέρσορας. Η επόμενη μεταβλητή (`private var curScreenPoint : Vector3;`) θα χρησιμοποιηθεί για να βρίσκουμε την θέση που δείχνει ο κέρσορας ανά πάσα στιγμή. Η μεταβλητή (`private var screenPoint : Vector3;`) θα χρησιμοποιηθεί για την αποθήκευση της θέσης σε screen space.

Οι συναρτήσεις που θα χρειαστούμε για αυτό το script είναι τρείς. Μας ενδιαφέρει να κάνουμε υπολογισμούς όταν ο χρήστης κάνει κλικ με τον κέρσορα πάνω στο σπασμένο κομμάτι, όταν το σέρνει και όταν το αφήνει. Άρα θα χρειαστούμε την OnMouseDown(), την OnMouseDrag() και την OnMouseUp().

```
function OnMouseDown () {
screenPoint =Camera.main.WorldToScreenPoint(gameObject.transform.position);
offset = gameObject.transform.position - Camera.main.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, screenPoint.z));

Screen.showCursor = false;
    }
```

Πίνακας 41– Χρήση της OnMouseDown

Μόλις πατήσει ο παίκτης το αντικείμενο, στην μεταβλητή screenPoint θα αποθηκευτεί η θέση του αντικειμένου, όχι στο World space αλλά στο Screen space. Στην συνέχεια στην μεταβλητή offset θα αποθηκευτεί το offset των σημείων ανάμεσα στην θέση του αντικειμένου και στο που βρίσκεται ακριβώς ο κέρσορας στην οθόνη. Στην συνέχεια θέλουμε να εξετάσουμε τι θα κάνουμε κατά το drag του αντικειμένου.

```
function OnMouseDrag() {
curScreenPoint = new Vector3(Input.mousePosition.x, Input.mousePosition.y,
screenPoint.z);

curPosition = Camera.main.ScreenToWorldPoint(curScreenPoint) + offset;
transform.position = curPosition;
    }
```

Πίνακας 42– Χρήση της OnMouseDrag

Κατά την διάρκεια που θα σέρνει ο παίκτης το αντικείμενο θέλουμε να ανανεώνεται κάθε φορά η νέα του θέση. Άρα στην μεταβλητή curScreenPoint θα αποθηκεύεται, σε κάθε frame, η θέση του κέρσορα στους άξονες x και y. Για να μπορέσουμε όμως να εντοπίσουμε την θέση που θα έχει το

αντικείμενο κάθε φορά πρέπει να κάνουμε την μετατροπή στο World Space και να την αποθηκεύσουμε στην curPosition και εν τέλει να περάσουμε την τιμή αυτή στο position του αντικειμένου. Στην συνέχεια, στο τελικό κομμάτι θα δούμε τι γίνεται μόλις ο παίκτης αφήσει τον κέρσορα.

```
function OnMouseUp(){
    Screen.showCursor = true;
if (Vector3.Distance(transform.parent.position,transform.position)<1.5)
        {
          transform.position = myPos;
          broken_main.puzzle_cnt += 1;
          audio.PlayOneShot(Sound_destroy);
          Destroy(this);
        }
```

Πίνακας 43– Χρήση της OnMouseUp

Μόλις ο παίκτης αφήσει τον κέρσορα θα γίνει ένας έλεγχος μέσω της πρώτης συνθήκης if αν η απόσταση που έχει το κομμάτι που ελέγχει με το ακίνητο κομμάτι όπου πρέπει να προσαρτηθεί είναι μικρότερη του 1.5. Αυτό γίνεται μέσω της μεθόδου Distance(), η οποία δέχεται ως ορίσματα την θέση των δύο αντικειμένων και την απόσταση που θέλουμε να ελέγξουμε. Αν η απόσταση αυτή είναι όντως μικρότερη, τότε το κομμάτι θα πρέπει να προσαρτηθεί στο σωστό σημείο και ο counter του script broken_main θα αυξηθεί κατά ένα.

Προχωρώντας στο script broken_main θα δούμε πως γίνεται η μεταχείριση των κομματιών των αντικειμένων. Αρχικά θα χρειαστούμε έναν πίνακα που θα αποθηκεύει τα σπασμένα αντικείμενα, έναν πίνακα που θα αποθηκεύει την θέση τους στο χώρο, δύο float μεταβλητές οι οποίες θα χρησιμεύσουν στην τοποθέτηση των κομματιών στον χώρο και τον counter που αναφέραμε πριν.

```
var fragments : Transform[];
var fragmentsPos : Vector2[];
var minRAN : float;
var maxRAN : float;
static var puzzle_cnt: int = 0;
var instantiation_node: Transform;
```

Πίνακας 44– Μεταβλητές για το script broken_main

Στην συνέχεια του script στην συνάρτηση Start() θα πρέπει να γεμίσουμε τους πίνακες που δηλώσαμε στην αρχή με τα κομμάτια που θα υπάρχουν κάθε φορά στην σκηνή και με την τοποθεσία τους έτσι ώστε μετά, χρησιμοποιώντας τους δύο float, minRAN και maxRAN, να τοποθετούνται στον χώρο σε ελεγχόμενα τυχαίες θέσεις.

```
transform.position = instantiation_node.position;

    for(var i = 0; i < fragments.length; i++)
    {
        fragments[i].gameObject.AddComponent(drag);
        fragmentsPos[i] =
Vector2(fragments[i].transform.position.x,fragments[i].transform.position.y
);
    }
```

Πίνακας 45– Ορισμός θέσης parent κομματιού και γέμισμα πινάκων

Αρχικά θα πρέπει το parent κομμάτι, στο οποίο είναι προσαρτημένο αυτό το script, να μεταφερθεί στην θέση του instantiation_node. Το κομμάτι αυτό δεν θα κινηθεί ποτέ και όλα τα

υπόλοιπα κομμάτια του μοντέλου θα τοποθετηθούν γύρω του όπως θα δούμε παρακάτω. Έχοντας συσχετίσει τα κομμάτια στον editor με τον πίνακα fragments, μέσω μιας for loop θα ενεργοποιήσουμε το drag script. Στην συνέχεια στον πίνακα fragmentsPos θα αποθηκεύσουμε τις θέσεις των κομματιών έτσι ώστε στην επόμενη for να τα τοποθετήσουμε στον χώρο.

```
for(i = 0; i < fragments.length; i++)
    {
fragments[i].position =
Vector3(Random.Range((fragmentsPos[i].x+minRAN+2),(fragmentsPos[i].x+maxRAN
+2)),Random.Range((fragmentsPos[i].y+minRAN),(fragmentsPos[i].y+maxRAN)),tr
ansform.position.z);
    }
```

Πίνακας 46– Ελεγχόμενα τυχαία τοποθέτηση των κομματιών

Στην δεύτερη for loop θα διαβάσουμε τον πίνακα όπου αποθηκεύσαμε τις θέσεις των κομματιών και μέσω της Random.Range() τα κομμάτια θα τοποθετηθούν σε μια, ελεγχόμενα, τυχαία κλίμακα στον άξονα x και y. Το μόνο που μένει πλέον είναι η Update() να ελέγχει σε κάθε frame αν ο counter (ο οποίος όπως θυμόμαστε, μεταβάλλεται μέσω του drag script όταν τα κομμάτια κολλήσουν εκεί που πρέπει) έχει φτάσει τον αριθμό των κομματιών που έχουν αποθηκευτεί στον πίνακα fragments έτσι ώστε να τελειώσει την διαδικασία προωθώντας το όνομα του ολοκληρωμένου αντικειμένου στον πίνακα display_broken, να καταστρέψει το αντικείμενο και τέλος να ξανά μηδενίσει τον counter για το επόμενο αντικείμενο.

```
function Update ()
{
    if(puzzle_cnt == (fragments.length))
    {
        manager.display_broken.Push(this.gameObject.name);
        manager.broken.RemoveAt(Breaking.breaking_selected_object);
        Destroy(this.gameObject, 2);
        puzzle_cnt = 0;
    }
}
```

Πίνακας 47– Ολοκλήρωση της διαδικασίας για το συναρμολογημένο αντικείμενο

5.5 Scripts για την αίθουσα εκθεμάτων

Σε αυτό το υποκεφάλαιο θα ασχοληθούμε με το τι γίνεται αφού τα αντικείμενα καθαριστούν και είναι έτοιμα να παρουσιαστούν στην αίθουσα εκθεμάτων του αρχαιολογικού εργαστηρίου. Ο τρόπος που λειτουργεί η παρουσίαση είναι ο εξής. Όταν ο παίκτης πλησιάσει κάποιο έκθεμα, η κάμερα του απενεργοποιείται και ενεργοποιείται μια άλλη, τοποθετημένη με τέτοιο τρόπο ώστε να βλέπει το αντικείμενο ο παίκτης σε πρώτο πρόσωπο.

Πριν όμως δούμε τον κώδικα για την παραπάνω λειτουργία, πρέπει να εξετάσουμε πως θα εμφανίζονται τα αντικείμενα που έχουν καθαριστεί στην αίθουσα εκθεμάτων. Όπως είδαμε και στις σκηνές καθαρισμού των αρχαιολογικών αντικειμένων, όποιο game object θέλουμε να χειριστούμε βρίσκεται ήδη μέσα στην σκηνή απλά δεν είναι ορατό στον παίκτη. Έτσι και εδώ θα ακολουθήσουμε παρόμοια τακτική. Τα αντικείμενα θα είναι τοποθετημένα όπου πρέπει για να προβληθούν, αλλά θα ενεργοποιούνται μόνο όταν το αντικείμενο έχει καθαριστεί. Εδώ έρχονται να μας βοηθήσουν οι πίνακες με template display_* όπου είναι αποθηκευμένα τα ονόματα των έτοιμων αντικειμένων. Το script ψάχνοντας μέσα σε αυτούς τους πίνακες ενεργοποιεί όσα αντικείμενα στην σκηνή έχουν ίδιο όνομα.

Αρχικά για να μπορέσει το script να ψάξει στους display πίνακες, θα πρέπει να αναθέσουμε όλα τα αντικείμενα σε μεταβλητές τύπου game object. Μόλις φτιάξουμε τις μεταβλητές, στον editor κάνουμε την αντιστοίχιση μεταφέροντας τα αντικείμενα, με drag and drop, στις μεταβλητές του script.

```
var scanner_1 : GameObject;
var scanner_2 : GameObject;
var scanner_3 : GameObject;
var scanner_4 : GameObject;
var scanner_5 : GameObject;
var scanner_6 : GameObject;
var scanner_7 : GameObject;

var mud_1 : GameObject;
var mud_2 : GameObject;
var mud_3 : GameObject;
var mud_4 : GameObject;
var mud_5 : GameObject;
var mud_6 : GameObject;
var mud_7 : GameObject;

var dirt_1 : GameObject;
var dirt_2 : GameObject;
var dirt_3 : GameObject;
var dirt_4 : GameObject;
var dirt_5 : GameObject;
var dirt_6 : GameObject;
var dirt_7 : GameObject;

var broken_1 : GameObject;
var broken_2 : GameObject;
var broken_3 : GameObject;
var broken_4 : GameObject;
var broken_5 : GameObject;
```

Πίνακας 48– Μεταβλητές για ανάθεση game objects

Από εδώ και πέρα θα είναι πολύ εύκολο να δούμε αν το όνομα του κάθε game object που έχει αντιστοιχηθεί με τις παραπάνω μεταβλητές, υπάρχει μέσα σε κάποιον από τους display πίνακες.

Το μόνο που μένει τώρα είναι χρησιμοποιώντας κάποιες if συνθήκες, να ελέγξουμε ποια από τα ονόματα των παραπάνω αντικειμένων βρίσκονται στους πίνακες display που τους αντιστοιχούν. Ο έλεγχος αν βρίσκεται κάποιο στοιχείο σε κάποιον πίνακα γίνεται με την δεσμευμένη λέξη in. Για να ενεργοποιηθεί κάποιο αντικείμενο του οποίου το όνομα βρέθηκε, θα χρησιμοποιήσουμε την μέθοδο setActiveRecursively(). Ο λόγος είναι ότι κάποια αντικείμενα είναι πιθανόν να έχουν κάποια παιδιά, οπότε έτσι θα ενεργοποιηθούν και αυτά μαζί.

```
function Start ()
{
    if(scanner_1.name in manager.display_scanner)
    scanner_1.gameObject.SetActiveRecursively(true);

    if(scanner_2.name in manager.display_scanner)
    scanner_2.gameObject.SetActiveRecursively(true);

    if(scanner_3.name in manager.display_scanner)
    scanner_3.gameObject.SetActiveRecursively(true);

    if(scanner_4.name in manager.display_scanner)
    scanner_4.gameObject.SetActiveRecursively(true);

    if(scanner_5.name in manager.display_scanner)
    scanner_5.gameObject.SetActiveRecursively(true);

    if(scanner_6.name in manager.display_scanner)
```

```
    scanner_6.gameObject.SetActiveRecursively(true);

    if(scanner_7.name in manager.display_scanner)
    scanner_7.gameObject.SetActiveRecursively(true);


    if(mud_1.name in manager.display_mud)
    mud_1.gameObject.SetActiveRecursively(true);

    if(mud_2.name in manager.display_mud)
    mud_2.gameObject.SetActiveRecursively(true);

    if(mud_3.name in manager.display_mud)
    mud_3.gameObject.SetActiveRecursively(true);

    if(mud_4.name in manager.display_mud)
    mud_4.gameObject.SetActiveRecursively(true);

    if(mud_5.name in manager.display_mud)
    mud_5.gameObject.SetActiveRecursively(true);

    if(mud_6.name in manager.display_mud)
    mud_6.gameObject.SetActiveRecursively(true);

    if(mud_7.name in manager.display_mud)
    mud_7.gameObject.SetActiveRecursively(true);
 }
```

Πίνακας 49– Μεταβλητές για ανάθεση game objects

Τώρα που τα αντικείμενα που έχουν καθαριστεί έχουν εμφανιστεί στην αίθουσα εκθεμάτων πρέπει να ενεργοποιηθεί και η δυνατότητα προβολής τους σε πρώτο πρόσωπο, όπως είπαμε και πριν. Το πρώτο πράγμα που θα κάνουμε είναι να τοποθετήσουμε box colliders γύρω από τα βάθρα που είναι τοποθετημένα αντικείμενα. Αυτοί οι colliders θα είναι απενεργοποιημένοι μέχρι να διαπιστωθεί ότι το αντικείμενο που επιθυμεί να δει ο παίκτης έχει καθαριστεί. Επίσης κατά την ενεργοποίηση του εκάστοτε collider, θα ενεργοποιείται και η κάμερα για την προοπτική πρώτου προσώπου των αντικειμένων. Αρχικά θα δούμε τις μεταβλητές που θα χρησιμοποιήσουμε.

```
private var scrollPosition : Vector2;
var LinearB_camera: Camera;
var Cam_to_disable:Camera;
var display_message: boolean;
var someRect : Rect;
var someMessage : String;
var mySkin : GUISkin;
```

Πίνακας 50– Μεταβλητές για ανάθεση game objects

Οι μεταβλητές που θέλουμε να εξετάσουμε για το συγκεκριμένο κομμάτι που αναλύουμε είναι οι εξής:

- `var LinearB_camera: Camera;` Στον editor θα συσχετίσουμε την κάμερα πρώτου προσώπου με αυτήν την μεταβλητή για να μπορέσουμε να την ενεργοποιήσουμε.
- `var Cam_to_disable:Camera;` Στον editor θα συσχετίσουμε την κάμερα του χαρακτήρα που θέλουμε να απενεργοποιήσουμε με αυτήν τη μεταβλητή.

Η πρώτη συνθήκη if θα ελέγχει ότι το αντικείμενο είναι όντως στον πίνακα display. Εάν είναι ο collider θα ενεργοποιείται.

```
unction Start ()
{
    display_message=false;

       if("LinearB_1" in manager.display_broken || "LinearB_2" in
manager.display_broken )
            {

                this.gameObject.collider.enabled=true;
        }

}
```

Πίνακας 51– Μεταβλητές για ανάθεση game objects

Το επόμενο βήμα είναι να ενεργοποιείται η κάμερα πρώτου προσώπου (LinearB_camera) και να απενεργοποιείται η κάμερα του παίκτη (Cam_to_disable), όσο ο παίκτης βρίσκεται σε επαφή με τον collider. Άρα θα χρησιμοποιήσουμε την συνάρτηση OnCollisionStay().

```
function OnTriggerStay(other : Collider)
{
    if (other.tag == "Player")
    {
    LinearB_camera.camera.enabled=true;
    Cam_to_disable.camera.enabled=false;
    display_message=true;
    }
```

Πίνακας 52– Μεταβλητές για ανάθεση game objects

Όμως όταν χαθεί το collision ανάμεσα στον παίκτη και το βάθρο, πρέπει να ξανά ενεργοποιηθεί η κάμερά του. Οπότε αυτό μας λέει ότι χρειαζόμαστε την συνάρτηση OnCollisionExit() στην οποία θα απενεργοποιήσουμε την κάμερα πρώτου προσώπου και θα ενεργοποιήσουμε αυτή του παίκτη.

```
function OnTriggerExit(other : Collider)
{
   if (other.tag == "Player")
    {
    LinearB_camera.camera.enabled=false;
    Cam_to_disable.camera.enabled=true;
    display_message=false;

    }
```

Πίνακας 53– Μεταβλητές για ανάθεση game objects

6. Δημιουργία γραφικού περιβάλλοντος (GUI) στο Unity3d

Η δημιουργία ενός γραφικού περιβάλλοντος (GUI) είναι μια απαραίτητη αλλά και σχετικά χρονοβόρα και περίπλοκη διαδικασία στο Unity 3d. Για να δημιουργήσουμε ένα GUI αποτελεσματικά πρέπει αρχικά να σκεφτούμε τι θέλουμε να καταφέρουμε σαν τελικό αποτέλεσμα. Πρέπει να λάβουμε υπόψη ποια θα είναι η καλύτερη πρακτική ώστε ο χρήστης να μπορεί να κατατοπιστεί άνετα χωρίς να τον μπερδεύει το γραφικό περιβάλλον.

Αυτές είναι οι γενικές πρώτες σκέψεις που πρέπει να κάνουμε. Όμως υπάρχουν και πιο τεχνικά θέματα τα οποία θα μας απασχολήσουν. Το γραφικό περιβάλλον θα λειτουργεί για όλες τις αναλύσεις οθονών; Θα παρέχει λειτουργία κουμπιών; Θα έχει textures τα οποία θα το διακοσμούν; Ή μήπως θα είναι τρισδιάστατο; Στα επόμενα υποκεφάλαια θα δούμε πως λειτουργεί ο κώδικας για τα κουμπιά στις σκηνές καθαρισμού των αρχαιολογικών αντικειμένων. Επίσης θα δούμε πως λειτουργούν τα GUI styles.

6.1 Δημιουργία κουμπιών στο Unity3d

Ένα από τα βασικά στοιχεία γραφικού περιβάλλοντος που θα κληθούμε να φτιάξουμε σε ένα οποιοδήποτε ηλεκτρονικό παιχνίδι, είναι τα κουμπιά. Αρχικά για να δημιουργήσουμε οποιοδήποτε στοιχείο GUI στο Unity 3d θα πρέπει μέσω script να καλέσουμε την OnGUI(). Αυτή είναι η συνάρτηση η οποία αναλαμβάνει να ζωγραφίσει τα γραφικά στοιχεία που θα της πούμε εμείς. Ας δούμε πως φτιάχνεται μια κάθετη σειρά κουμπιών η οποία έχει την δυνατότητα scrolling.

Αρχικά στις δηλώσεις πρέπει να έχουμε μια μεταβλητή η οποία να είναι πίνακας δυο θέσεων ώστε να μπορούμε στην συνάρτηση OnGUI() να ορίσουμε από πού μέχρι που θα είναι η περιοχή scrolling.

```
private var scrollPosition : Vector2;
```

Πίνακας 54– Δήλωση της scrollPosition

Έχοντας δηλώσει την scrollPosition μεταβλητή, θα καλέσουμε την συνάρτηση onGUI() για να ορίσουμε ποια θα είναι τα περιθώρια που θα τοποθετηθούν τα κουμπιά που θα δημιουργήσουμε.

```
scrollPosition = GUILayout.BeginScrollView (scrollPosition, GUILayout.Width
(Screen.width/6), GUILayout.Height (Screen.height/2));
```

Πίνακας 55– Ορισμός περιθωρίων της σειράς κουμπιών

Για να οριστεί το περιθώριο που θέλουμε, βλέπουμε ότι καλούμε την μέθοδο BeginScrollView() η οποία θα αναλάβει να φέρει εις πέρας αυτό που θέλουμε ανάλογα με τα ορίσματα που θα της δώσουμε. Σε αυτό το σημείο πρέπει να αναφερθούμε σε κάτι σημαντικό όσον αφορά τα γραφικά περιβάλλοντα. Όταν δηλώνουμε που θέλουμε να τοποθετηθεί κάποιο γραφικό στοιχείο και τι μέγεθος να έχει πρέπει να σκεφτούμε αν η εφαρμογή μας θα τρέχει μόνο σε μια ανάλυση οθόνης. Αν δεν είναι συγκεκριμένη πρέπει πάντα να ορίζουμε τα μεγέθη με βάση το ύψος και το πλάτος της κάθε ανάλυσης. Για αυτόν το λόγο χρησιμοποιούμε την Screen.width και Screen.height. Επιστρέφοντας στα ορίσματα της BeginScrollView(), βλέπουμε πως αυτά είναι τρία. Το πρώτο είναι ο δυσδιάστατος πίνακας που ορίσαμε και τα άλλα δύο είναι το πόσο ύψος και πόσο πλάτος θα καταλαμβάνει η περιοχή. Δίνοντας σαν όρισμα Screen.width/6, θα καταλάβει το ένα έκτο της οθόνης σε πλάτος. Πλέον οτιδήποτε δημιουργήσουμε θα βρίσκεται μέσα σε αυτήν την περιοχή.

Στην συνέχεια θα δημιουργήσουμε τα κουμπιά που θα τοποθετηθούν στην περιοχή αυτή. Για να δούμε πως λειτουργεί αυτό, θα χρησιμοποιήσουμε ένα παράδειγμα script από την σκηνή με το scanner το οποίο, όπως έχουμε πει δημιουργεί τα κουμπιά ανάλογα με το πόσα αντικείμενα μάζεψε ο παίκτης.

```
private var scrollPosition : Vector2;

function OnGUI()
{

GUI.Box(Rect(Screen.width-Screen.width/3,Screen.height-
Screen.height,Screen.width/3,Screen.height/5),"Use the left mouse button to
piece together the artifact. Each object restored will be displayed in the
displays room. Return to the lab by pressing L.",boxStyle1);

scrollPosition = GUILayout.BeginScrollView (scrollPosition, GUILayout.Width
(Screen.width/6), GUILayout.Height (Screen.height/2));

 if(manager.broken.length >=1)
    {
        if(GUILayout.Button (manager.broken[0],boxStyle10))
        {
GameObject.Find(manager.broken[0]).GetComponent(broken_main).enabled =
true;
            breaking_selected_object = 0;
        }
    }

    if(manager.broken.length >=2)
    {
        if(GUILayout.Button (manager.broken[1],boxStyle10))
        {

GameObject.Find(manager.broken[1]).GetComponent(broken_main).enabled =
true;
            breaking_selected_object = 1;
        }
    }

    if(manager.broken.length >=3)
    {
        if(GUILayout.Button (manager.broken[2],boxStyle10))
        {

GameObject.Find(manager.broken[2]).GetComponent(broken_main).enabled =
true;
            breaking_selected_object = 2;
        }
    }
GUILayout.EndScrollView ();
```

Πίνακας 56- Δημιουργία κουμπιών

Η ανάλυση αυτού του κώδικα είναι ένα πολύ καλό παράδειγμα για να δούμε ποια είναι η διαφορά στην δημιουργία γραφικών αντικειμένων μέσα σε περιοχές και έξω από περιοχές, αν μπορούν να υπάρχουν άλλες εντολές άσχετα με τον σχεδιασμό αντικειμένων GUI και τι ακριβώς είναι τα boxStyles.

Ξεκινώντας, πριν ανοίξει η συνάρτηση onGUI() βλέπουμε τον ορισμό της scrollPosition όπως την αναλύσαμε πριν. Ξεκινώντας στην onGUI(), πρώτες γραμμές κώδικα, βλέπουμε το BeginScrollWindow όπως το περιγράψαμε πριν. Αμέσως μετά όμως βλέπουμε if και συνθήκες. Άρα

εδώ να επισημάνουμε πως άσχετα αν η onGUI() αναλαμβάνει να ζωγραφίσει τα γραφικά αντικείμενα που θα της ζητήσουμε εννοείται πως δέχεται οποιοδήποτε είδους κώδικα. Εξάλλου όπως θα δούμε αμέσως, έτσι μόνο γίνεται να δώσουμε δυναμικό χαρακτήρα στα κουμπιά που θα δημιουργήσουμε. Προσπερνώντας τις συνθήκες δημιουργίας των κουμπιών, τις οποίες αναλύσαμε σε προηγούμενο κεφάλαιο, βλέπουμε την δημιουργία ενός κουμπιού (`GUILayout.Button(manager.broken[0],boxStyle10)`). Το πρώτο όρισμα που παίρνει είναι το όνομα που θα δείχνει το κουμπί. Εδώ παίρνει το String που βρίσκεται στην θέση μηδέν του πίνακα broken. Το επόμενο όρισμα, κανονικά, είναι το μέγεθος και η τοποθεσία του κουμπιού. Αντί για αυτό βλέπουμε να γράφει boxStyle10. Εδώ θα αναφερθούμε στα GUIstyles.

Τα GUI styles είναι ένας τρόπος να επηρεάσουμε την οπτική αισθητική των γραφικών στοιχείων που θα χρησιμοποιήσουμε στο παιχνίδι. Μπορούμε να αναθέσουμε textures τα οποία μπορεί και να αλλάζουν ανάλογα με το αν, για παράδειγμα, ένα κουμπί πατιέται, μπορούμε να αλλάξουμε γραμματοσειρές κτλ. Οι τροποποιήσεις αυτές μπορούν να γίνουν μέσω editor ή και μέσω script. Η επόμενη εικόνα απεικονίζει το GUI style που διαλέξαμε εμείς για τα κουμπιά αυτής της σκηνής.

Εικόνα 12– Παράδειγμα GUI style μέσω editor

Στο συγκεκριμένο παράδειγμα διαλέξαμε το texture papyrus4 να εμφανίζεται όταν το κουμπί είναι αδρανές και όταν ο κέρσορας περάσει από πάνω του να εμφανίζεται ένα διάφανο texture. Επίσης έχουμε επιλέξει δική μας γραμματοσειρά, όπως και την θέση του text που θα εμφανίζεται.

Αν τώρα θέλουμε να προσθέσουμε κάποιο άλλο στοιχείο GUI έξω από την περιοχή σχεδίασης κουμπιών, σε κάποιο άλλο σημείο της οθόνης, μπορούμε να το κάνουμε πολύ εύκολα όπως φαίνεται και στον κώδικα που μελετάμε.

```
GUI.Box(Rect(Screen.width-Screen.width/3,Screen.height-
Screen.height,Screen.width/3,Screen.height/5),"Use the left mouse button to
piece together the artifact. Each object restored will be displayed in the
displays room. Return to the lab by pressing L.",boxStyle1);
```

Πίνακας 57– Δημιουργία GUI Box

Εδώ βλέπουμε την δημιουργία ενός GUI box το οποίο θα δημιουργηθεί σε ένα τελείως ξεχωριστό μέρος της οθόνης. Αν παρατηρήσουμε τα ορίσματα θα δούμε ότι όλα είναι αναλογικά με την ανάλυση της εκάστοτε οθόνης που θα τρέξει το project και αντιπροσωπεύουν το μέγεθος του κουτιού και την τοποθεσία του. Τα άλλα ορίσματα είναι το κείμενο που θα προβάλλεται και το GUI style που επιλέξαμε.

7.Τι είναι οι Shaders

Στον τομέα των γραφικών ηλεκτρονικών υπολογιστών , ένας shader είναι ένα πρόγραμμα υπολογιστή που τρέχει στην μονάδα επεξεργασίας γραφικών και χρησιμοποιείται για να παράγει σκίαση - την παραγωγή των κατάλληλων επιπέδων του φωτός και σκίασης μέσα σε μια εικόνα - ή, στη σύγχρονη εποχή, για την παραγωγή ειδικών εφέ ή postprocessing .

Οι shaders υπολογίζουν την απόδοση του rendering για το hardware γραφικών με υψηλό βαθμό ευελιξίας. Οι γλώσσες που χρησιμοποιούνται για το shading προγραμματίζουν τον προγραμματιζόμενο αγωγό απόδοσης της μονάδα επεξεργασίας γραφικών (GPU), ο οποίος έχει αντικαταστήσει, ως επί το πλείστον, τη σταθερή λειτουργία του αγωγού που επιτρέπει μόνο κοινή μεταμόρφωση γεωμετρίας και λειτουργίες pixel-shading. Χρησιμοποιώντας shaders, μπορούν να χρησιμοποιηθούν προσαρμοσμένα εφέ. Η θέση, η απόχρωση, ο κορεσμός, η φωτεινότητα και η αντίθεση όλων των εικονοστοιχείων (pixels) , κορυφές , ή υφές (textures) που χρησιμοποιούνται για την κατασκευή μιας τελικής εικόνας μπορούν να μεταβληθούν στην εύκολα, χρησιμοποιώντας αλγόριθμους που ορίζονται μέσα στον shader, και μπορεί να τροποποιηθεί μέσω εξωτερικών μεταβλητών ή υφών που εισήχθησαν απο το πρόγραμμα που κάλεσε τον shader.

Οι Shaders χρησιμοποιούνται ευρέως στον τομέα του postprocessing ταινιών κινηματογράφου, στην δημιουργία ψηφιακών εικόνων και ηλεκτρονικών παιχνιδιών για να παραχθεί ένα άπειρο εύρος εφέ. Πέρα από το φωτισμό μοντέλων πιο πολύπλοκες χρήσεις μπορεί να περιλαμβάνουν την τροποποίηση της απόχρωσης , του κορεσμού , της φωτεινότητα και της αντίθεσης της εικόνας, την παραγωγή blur , bokeh , σκίασης cel , ποστεροποίησης (posterization) , chroma keying , παραμόρφωση , chroma keying (η λεγόμενη " μπλε οθόνης / greenscreen "), ανίχνευση ακμών και κίνησης , ψυχεδελικά εφέ, καθώς και ένα ευρύ φάσμα άλλων.

Η σύγχρονη χρήση του όρου "shader" εισήχθη στο κοινό από την Pixar με το "RenderMan Interface Specification, έκδοση 3.0» που δημοσιεύθηκε τον Μάιο του 1988.

Καθώς οι μονάδες επεξεργασίας γραφικών εξελιχτήκαν, μεγάλες βιβλιοθήκες λογισμικού γραφικών όπως οι OpenGL και Direct3D άρχισαν να υποστηρίζουν τους shaders. Η πρώτη κάρτα γραφικών με δυνατότητα shading υποστήριζε μόνο Pixel shading, αλλά οι vertex shaders κυκλοφόρησαν γρήγορα όταν οι προγραμματιστές συνειδητοποίησε την δύναμη τους. Shaders γεωμετρίας εισήχθησαν πρόσφατα με Direct3D 10 και OpenGL 3.2, αλλά αυτή τη στιγμή υποστηρίζονται μόνο από high-end κάρτες γραφικών.

7.1 Τεχνολογική επισκόπηση Shader

Οι Shaders είναι προγράμματα που περιγράφουν τα χαρακτηριστικά είτε μιας κορυφής ή ενός εικονοστοιχείου (pixel) . Οι Vertex shaders περιγράφουν τα χαρακτηριστικά (θέση, συντεταγμένες υφή , τα χρώματα, κλπ.) μιας κορυφής, ενώ οι pixel shaders περιγράφουν τα χαρακτηριστικά (χρώμα, z βάθος και αξία άλφα) ενός pixel. Ένας vertex shader καλείται για κάθε κορυφή σε ένα primitive (βασικό γεωμετρικό σχήμα), ενδεχομένως μετά από tessellation, η μια ενημερωμένη κορυφή μετά την άλλη . Κάθε κορυφή στην συνέχεια γίνεται rendered ως μια σειρά από pixel σε μία επιφάνεια (μπλοκ της μνήμης) που τελικά θα σταλεί στην οθόνη.

Οι Shaders αντικαθιστούν ένα τμήμα της κάρτας γραφικών, που συνήθως ονομάζεται Σταθερή Λειτουργία Αγωγού (Fixed Function Pipeline) . Του έχει αποδοθεί αυτή η ονομασία επειδή εκτελεί τον φωτισμό και χαρτογράφηση των textures (texture mapping) με hard coded τρόπο. Οι Shaders παρέχουν μια προγραμματιζόμενη εναλλακτική λύση αυτής της προσέγγισης.

- Η κεντρική μονάδα επεξεργασίας (CPU) στέλνει οδηγίες και δεδομένα γεωμετρίας στη μονάδα επεξεργασίας γραφικών, που βρίσκεται στην κάρτα γραφικών.
- Μέσω του shader κορυφών (vertex shader) η γεωμετρία μετασχηματίζεται.
- Εάν ένας shader γεωμετρίας βρίσκεται στην μονάδα επεξεργασίας γραφικών και είναι ενεργός, η γεωμετρία στην σκηνή μπορεί να υποδιαιρεθεί.
- Η υπολογισμένη γεωμετρία διασταυρώνεται (υποδιαιρείται σε τρίγωνα).
- Τα τρίγωνα αναλύονται σε κομμάτια quads (ένα quad κομμάτι είναι ένα 2 × 2 fragment primitive).
- Τα fragment quad τροποποιούνται σύμφωνα με τον fragment shader.

- πραγματοποιείται ένα τεστ βάθους (depth test), όπου τμήματα που περνούν θα εμφανιστούν στην οθόνη και μπορούν αναμιχθούν στον frame buffer .

Ο γραφικός αγωγός χρησιμοποιεί αυτά τα βήματα για να μετατρέψει τα τρισδιάστατη (ή δύο δισδιάστατα) δεδομένα σε χρήσιμα δύο διαστάσεων δεδομένα για την εμφάνιση τους. Σε γενικές γραμμές, αυτό είναι μια μεγάλο μήτρα πίξελ (pixel matrix) ή frame buffer.

7.2 Τύποι Shader

Υπάρχουν τρεις τύποι shaders που χρησιμοποιούνται ευρέως. Ενώ παλαιότερες κάρτες γραφικών χρησιμοποιούν ξεχωριστές μονάδες επεξεργασίας για κάθε τύπο shader, νεότερες κάρτες διαθέτουν ένα ενιαίους shaders που είναι ικανοί να εκτελέσουν οποιοδήποτε τύπο του shader. Αυτό επιτρέπει στις κάρτες γραφικών να κάνουν πιο αποτελεσματική χρήση της επεξεργαστικής ισχύος.

Οι Vertex shaders εκτελούνται μία φορά για κάθε κορυφή που μεταφέρεται στον επεξεργαστή γραφικών. Ο σκοπός είναι να μετασχηματίσει η 3D θέση κάθε κορυφής, σε εικονικό χώρο, σε δισδιάστατες συντεταγμένες όπου και θα εμφανιστεί στην οθόνη (καθώς και μια τιμή βάθους για το Z-buffer). Οι Vertex shaders μπορούν να χειριστούν ιδιότητες, όπως η θέση, το χρώμα, τις συντεταγμένες των texture, αλλά δεν μπορούν να δημιουργήσουν νέες κορυφές. Η έξοδος του Vertex shader μεταφέρεται στο επόμενο στάδιο του αγωγού, το οποίο είναι είτε ένας shader γεωμετρίας εάν υπάρχει, ή διαφορετικά ο shader pixel και ο rasterizer. Προσφέρουν ισχυρό έλεγχο των στοιχείων της θέσης, της κίνησης, του φωτισμού και του χρώματος σε κάθε σκηνή που περιέχει 3D μοντέλα .

Οι Shaders Γεωμετρίας είναι ένας σχετικά νέος τύπος του shader, που εισήχθη στο Direct3D 10 και OpenGL 3.2. Πρώην διατίθεται σε OpenGL 2,0 με τη χρήση επεκτάσεων. Αυτό το είδος shader μπορεί να δημιουργήσει νέα γραφικά primitives, όπως σημεία, γραμμές, και τρίγωνα, από τα εν λόγω primitives που εστάλησαν στην αρχή του αγωγού γραφικών.

Τα προγράμματα shader γεωμετρίας εκτελούνται μετά τους vertex shaders. Παίρνουν ως είσοδο ένα σύνολο primitives, ενδεχομένως γειτνιάζουσες πληροφορίες. Για παράδειγμα, όταν λειτουργούν σε τρίγωνα, οι τρεις κορυφές είναι η είσοδος του shader γεωμετρίας. Ο shader μπορεί να εκπέμψει στη συνέχεια μηδέν ή περισσότερα primitives, τα οποία ραστεροποιούνται και θραύσματα τους περνάνε τελικά στον pixel shader .

Τυπικές χρήσεις ενός shader γεωμετρίας περιλαμβάνουν point sprite generation, γεωμετρία tessellation, shadow volume extrusion και single pass rendering σε ένα cube map. Ένα τυπικό παράδειγμα για τα οφέλη αυτού είναι η αυτόματη τροποποίηση πολυπλοκότητας των mesh. Μια σειρά λωρίδων γραμμών που αντιπροσωπεύει τα σημεία ελέγχου για μια καμπύλη έχουν περάσει στο shader γεωμετρίας και ανάλογα με την πολυπλοκότητα που απαιτείται ο shader μπορεί αυτόματα να δημιουργήσει επιπλέον γραμμές, καθεμία από τις οποίες παρέχει μια καλύτερη προσέγγιση της καμπύλης.

Ο Pixel shader, επίσης γνωστός ως fragment shader, υπολογίζει το χρώμα και άλλα χαρακτηριστικά του κάθε θραύσματος. Μπορούν να εμφανίζουν πάντα το ίδιο χρώμα, να αλλάζουν την τιμή του φωτισμού, να εφαρμόζουν bump mapping , εφαρμόζουν σκιές , specular highlights , translucency και άλλα φαινόμενα. Μπορούν να μεταβάλλουν το βάθος του θραύσματος (για Z-buffering), ή να εμφανίσουν περισσότερα από ένα χρώματα αν τα στοιχεία που θα γίνουν rendered είναι πολλαπλά. Σε 3D γραφικά, ένας pixel shader μόνος του δεν μπορεί να παράγει πολύπλοκα αποτελέσματα, επειδή λειτουργεί μόνο σε ένα μόνο κομμάτι, χωρίς γνώση της γεωμετρίας της σκηνής. Ωστόσο, οι pixel shaders έχουν γνώση των συντεταγμένων οθόνης στο στάδιο της επεξεργασίας, και μπορούν να κάνουν δειγματοληψία της οθόνη και των κοντινών pixels εάν το περιεχόμενο ολόκληρης της οθόνης περάσει ως texture στον shader. Η τεχνική αυτή μπορεί να επιτρέψει μια ευρεία ποικιλία δισδιάστατων postprocessing εφέ, όπως θάμπωμα, ή ανίχνευση άκρων, ενίσχυση για κινούμενα σχέδια και cel shadering. Μπορούν επίσης να εφαρμοστούν σε ενδιάμεσα στάδια σε κάθε δισδιάστατη εικόνα στο pipeline, ενώ σε αντίθεση οι vertex shaders απαιτούν πάντα ένα 3D μοντέλο. Για παράδειγμα, ένας pixel shader είναι το μόνο είδος shader που μπορεί να δράσει ως postprocessor ή φίλτρο για μια ροή βίντεο αφού έχει ραστεροποιούνται .

Οι Shaders γράφονται για να εφαρμόζουν μετασχηματισμούς σε ένα μεγάλο σύνολο στοιχείων την φορά, για παράδειγμα, σε κάθε εικονοστοιχείο σε μια περιοχή της οθόνης, ή για κάθε κορυφή ενός μοντέλου. Αυτή η πρακτική είναι κατάλληλη για παράλληλη επεξεργασία , και οι πιο

σύγχρονες κάρτες γραφικών έχουν πολλαπλούς shader pipelines για τη διευκόλυνση αυτή, βελτιώνοντας έτσι την αποτελεσματικότητα των υπολογισμών.

Η γλώσσα στην οποία προγραμματίζονται οι shaders εξαρτάται από το περιβάλλον που μας ενδιαφέρει. Η επίσημη OpenGL και OpenGL ES γλώσσα shading είναι η OpenGL Γλώσσα σκίασης , επίσης γνωστή ως GLSL, καθώς και η επίσημη γλώσσα Direct3D shading είναι υψηλού επιπέδου γλώσσα Shader , επίσης γνωστή ως HLSL. Ωστόσο, η Cg είναι μια γλώσσα σκίασης που αναπτύχθηκε από την Nvidia που εξάγει τους OpenGL και Direct3D shaders.

8.Μελλοντικά implementations του παιχνιδιού

Κατά την διάρκεια δημιουργίας ενός ηλεκτρονικού παιχνιδιού, οι ιδέες για τους μηχανισμούς που θα εισαχθούν σε συνάρτηση με το deadline που πρέπει να διατηρηθεί μπορεί να μην καταφέρουν να υλοποιηθούν πλήρως ή μπορεί και να μην εισαχθούν και καθόλου. Άλλοι βασικοί παράγοντες που μπορεί να περιορίσουν την λειτουργικότητα ενός παιχνιδιού είναι η εμπειρία των developers. Τα περισσότερα sequel παιχνιδιών παρουσιάζουν μεγάλη πρόοδο και είναι πολύ περισσότερο user friendly. Πολύ σημαντικό στοιχείο είναι το feedback των παικτών το οποίο συνήθως καθορίζει ποια στοιχεία θα βελτιωθούν και ποια στοιχεία θα εισαχθούν ώστε να βελτιώσουν την εμπειρία. Κάποιες εταιρείες αναλαμβάνουν μια ακόμη μεγαλύτερη πρόκληση. Αφού το παιχνίδι έχει ολοκληρωθεί και βγει στην αγορά, ακούγοντας τις συμβουλές των παικτών βελτιώνουν όλα τα προβληματικά κομμάτια και βελτιώνουν το υπάρχων παιχνίδι σημαντικά. Μια άλλη συνηθισμένη τακτική είναι οι εταιρείες να βγάζουν στην αγορά πακέτα τα οποία προσθέτουν νέες αποστολές και καμία φορά μηχανισμούς στο παιχνίδι, προς πληρωμή.

Για να μελετήσουμε τι θα μπορούσαμε να προσθέσουμε ή να βελτιώσουμε στην εφαρμογή που αναπτύξαμε, θα πρέπει να χωρίσουμε κάποιες κατηγορίες. Θα πρέπει να δούμε πως μπορούν να βελτιωθούν οι υπάρχοντες μηχανισμοί και αν μπορούν να προστεθούν παραπάνω, τα γραφικά του παιχνιδιού πως μπορούν να βελτιωθούν, που υστερεί ο τομέας του sound design και αν μπορεί να προστεθεί κάποιο σενάριο.

Οι μηχανισμοί που εισάγαμε στο παιχνίδι θέλαμε να είναι αρκετά απλοί για να μπορούν να είναι προσβάσιμοι από παίκτες οποιαδήποτε ηλικίας. Ένα βασικό χαρακτηριστικό που μπορεί να προστεθεί είναι ένας καθολικός μηχανισμός που θα προσθέτει πόντους καθώς ο παίκτης θα αλληλεπιδρά με τις διάφορες πτυχές του παιχνιδιού. Μαζεύοντας τα αντικείμενα ο παίκτης θα συλλέγει ταυτόχρονα πόντους τους οποίους θα μπορεί να χρησιμοποιεί για να ενεργοποιεί έναν μηχανισμό ο οποίος για ένα χρονικό διάστημα, ανάλογο με τους πόντους που έχει μαζέψει, θα τον οδηγεί προς τα κρυμμένα αρχαιολογικά αντικείμενα. Στην συνέχεια, για να παρουσιάζονται πιο ενδιαφέροντες και για να προσφέρουν μια μεγαλύτερη πρόκληση, οι μηχανισμοί καθαρισμού θα πρέπει να προστεθεί ένας timer όπου όσο πιο γρήγορα καθαρίσει ο παίκτης τα αντικείμενα που σύλλεξε θα παίρνει και περισσότερους πόντους. Επίσης ανάλογα με το πόσο καλά τα πήγε θα ξεκλειδώνουν περισσότερες πληροφορίες στην αίθουσα εκθεμάτων. Η δυνατότητα του timer μπορεί να επεκταθεί, έτσι ώστε να μπορεί ο παίκτης να προσπαθεί να ολοκληρώσει το παιχνίδι όσο το δυνατόν πιο γρήγορα, προσπαθώντας έτσι να κάνει Hi-score.

Το πώς παρουσιάζεται ένα ηλεκτρονικό παιχνίδι όσο αφορά τον τομέα των γραφικών είναι ένα πολύ σημαντικό κεφάλαιο γιατί επηρεάζει άμεσα το πώς αντιλαμβάνεται ο παίκτης το παιχνίδι. Βέβαια δεν πρέπει να συγχέονται τα γραφικά του παιχνιδιού με την αισθητική. Τα γραφικά αφορούν στο πόσο καλά αναπαριστάται ο κόσμος όσο αφορά κυρίως την γεωμετρία του κόσμου και των αντικειμένων. Η αισθητική είναι επίσης, ίσως και ακόμα πιο σημαντική, από την σωστή γραφική αναπαράσταση. Όσον αφορά τα γραφικά και τα μοντέλα, αποκτώντας παραπάνω εμπειρία στον τομέα του 3d modeling τα αρχαιολογικά αντικείμενα θα μπορούν να αναπαρασταθούν με πολύ μεγαλύτερη λεπτομέρεια, κάτι το οποίο είναι απαραίτητο για ένα παιχνίδι με εκπαιδευτικό προσανατολισμό. Όσον αφορά την αισθητική, τα φίλτρα που εφαρμόστηκαν στην αρχική σκηνή του παιχνιδιού δεν μπορούν να εφαρμοστούν και στο υπόλοιπο παιχνίδι λόγο ελλιπούς υπολογιστικής ισχύος.

Ο τομέας του sound design σε ένα παιχνίδι είναι σίγουρα από τους πιο εξεζητημένους και απαιτητικούς. Για να αποκτήσει ένα παιχνίδι τον δικό του χαρακτήρα χρειάζεται σίγουρα κάποιον συνθέτη που θα ασχοληθεί εξολοκλήρου με το soundtrack και με τα ηχητικά εφέ που θα ζωντανέψουν τον κόσμο του παιχνιδιού. Ο επαγγελματισμός στον ήχο κάνει τις περισσότερες φορές την διαφορά σε κάποιο ηλεκτρονικό παιχνίδι. Ένα καλοστημένο soundtrack, καθαρά ηχογραφημένα ηχητικά εφέ και δυναμικός ήχος ο οποίος μεταβάλλεται ανάλογα με το παιχνίδι θα δώσει στο παιχνίδι τον επαγγελματικό αέρα που χρειάζεται για να ξεχωρίσει. Όμως και εδώ η έλλειψη κατάλληλου hardware μπορεί πολύ εύκολα να σταθεί εμπόδιο.

Τέλος φτάνουμε στο σημείο όπου πρέπει να δούμε αν κάποιο είδος σεναρίου θα ωφελούσε το παιχνίδι ή αν δεν ταιριάζει με την κεντρική ιδέα των mini games και του εκπαιδευτικού χαρακτήρα. Ένα ολοκληρωμένο σενάριο που θα έβαζε τον παίκτη στον ρόλο ενός κεντρικού ήρωα θα αλλάξει σίγουρα τον αέρα του παιχνιδιού και θα μετέβαλε τον απλό χαρακτήρα του που το κάνει προσβάσιμο

σε όλες τις ηλικίες και θα απαιτούσε από τους παίκτες να επικεντρωθούν περισσότερο στην ιστορία παρά στην συλλογή και προβολή των αρχαιολογικών αντικειμένων. Βέβαια αυτό που θα μπορούσε να προστεθεί μελλοντικά είναι κάποιοι δευτερεύοντες χαρακτήρες οι οποίοι μπορούν να αλληλεπιδρούν με τον χαρακτήρα, πχ αντικαθιστώντας το υπάρχον tutorial, γενικά κάνοντας το παιχνίδι ποιο ζωντανό.

Συνοψίζοντας βλέπουμε πως όλοι οι τομείς του παιχνιδιού μπορούν να επωφεληθούν από την προσθήκη παραπάνω λεπτομερειών, όπως μηχανισμοί χρόνο και πόντων που θα προσθέσουν replayability και θα δώσουν στον παίκτη την επιθυμία να ξεκλειδώσει όλες τις λεπτομέρειες των αρχαιολογικών αντικειμένων. Όσον αφορά τα γραφικά την αισθητική και το sound design, η έλλειψη κατάλληλου hardware εμφανίζει σημαντικά εμπόδια. Τέλος καταλήξαμε ότι θα επωφελήσει το παιχνίδι, όχι ένα ολοκληρωμένο σενάριο, αλλά κάποιοι βοηθητικοί χαρακτήρες οι οποίοι θα ζωντανέψουν το παιχνίδι.

9.Συμπεράσματα από την πτυχιακή εργασία

Όπως αναφέραμε και στην εισαγωγή, αυτή η πτυχιακή εργασία ήταν η πρώτη μας επαφή με τον κόσμο του game development και όλες τις διαδικασίες που χρειάζεται να ακολουθηθούν για να δημιουργηθεί ένα ηλεκτρονικό παιχνίδι. Παρόλο που είχαμε κάποια εισαγωγική εμπειρία με κάποιους από τους τομείς που συμβάλλουν στην δημιουργία της εργασίας, το επίπεδο που απαιτείται πρέπει να είναι πολύ πιο υψηλό. Για να μπορέσουμε να βγάλουμε κάποια χρήσιμα συμπεράσματα καλό είναι να χωρίσουμε αυτό το κεφάλαιο στο τι μας δυσκόλεψε και στο τι μάθαμε από την όλη διαδικασία ανάπτυξης του παιχνιδιού.

Τα βασικά προβλήματα που αντιμετωπίσαμε ήταν τρία. Αρχικά η έλλειψη εμπειρίας και γνώσεων, τα προβλήματα συμβατότητας αρχείων ανάμεσα σε εφαρμογές και υπολογιστικά συστήματα τα οποία δεν μπορούσαν να ανταποκριθούν στις απαιτήσεις του software που χρησιμοποιήσαμε.

Έχοντας κάποια βασική εμπειρία στο modeling και scripting μέσω των μαθημάτων της σχολής ήταν σίγουρα μια καλή αρχή αλλά οι απαραίτητες γνώσεις που χρειαζόμασταν ήταν σίγουρα πολύ περισσότερες. Όπως είδαμε και σε προηγούμενα κεφάλαια εκτός από το modeling, το οποίο είναι από μόνο του μια απαιτητική διαδικασία, έπρεπε να μάθουμε την διαδικασία του rigging, του skinning και μην ξεχνάμε το animation. Πολλές δυσκολίες εμφανίστηκαν σε αυτόν τομέα καθώς αρχικά δοκιμάσαμε να δημιουργήσουμε τα animations του χαρακτήρα χρησιμοποιώντας το Kinect, open drivers και το motion builder. Έχοντας παρακολουθήσει αρκετά tutorials, αυτό το workflow φαινόταν πολύ ενδιαφέρον γεμάτο πολλές δυνατότητες. Δυστυχώς, προβλήματα compatibility με το hardware των υπολογιστών που δουλεύαμε δεν μας επέτρεψε να χρησιμοποιήσουμε το συγκεκριμένο workflow.

Έχοντας αποκτήσει λίγη εμπειρία παραπάνω στα προγράμματα που χρειαζόμασταν, εμφανίστηκε το δεύτερο σοβαρό πρόβλημα που έπρεπε να αντιμετωπίσουμε, το πρόβλημα της συμβατότητας ανάμεσα στα προγράμματα αυτά. Πολλές φορές η μεταφορά αρχείων .fbx από προγράμματα δημιουργίας μοντέλων στο Unity 3d ή τα animation μεταξύ blender και motion builder παρουσίαζε αρκετά προβλήματα τα οποία χρειαζόντουσαν αρκετό trial and error για να βρεθούν οι σωστές ρυθμίσεις. Αυτό με την σειρά του απαιτούσε καλή οργάνωση όλων των assets μας, κάτι το οποίο δεν ήταν πάντα εύκολο.

Αμέσως μετά, και καθώς το project προχωρούσε, εμφανίστηκε το επόμενο πρόβλημα το οποίο αν και δεν ήταν απαγορευτικό για την συνέχεια της πτυχιακής, ήταν σίγουρα ένας παράγοντας ο οποίος συνέβαλε στο να προχωράει αρκετά ποιο αργά καθώς κάθε νέο compile,για debugging, έπαιρνε πολύ περισσότερο χρόνο απ' ότι θα έπαιρνε σε πιο σύγχρονους υπολογιστές. Επίσης τα προβλήματα ασυμβατότητας που αναφέραμε στην προηγούμενη παράγραφο μας ανάγκαζαν να κάνουμε πολλαπλά exports από απαιτητικά, σε υπολογιστική ισχύ, κάτι το οποίο αποδείχθηκε ιδιαίτερα χρονοβόρο.

Βέβαια πρέπει να αναφέρουμε πως τα προβλήματα που εμφανιζόντουσαν σε πολλά βήματα της εκπόνησης της εργασίας σε τελική ανάλυση μας βοήθησαν να καταλάβουμε ποιες είναι οι απαιτήσεις για την δημιουργία ενός ηλεκτρονικού παιχνιδιού και πως μπορούμε να ανταπεξέλθουμε στα μελλοντικά projects που θα αναλάβουμε, καθώς θέλουμε να ασχοληθούμε επαγγελματικά με τον τομέα του game development.

Μέσω αυτών των δυσκολιών καταφέραμε να αποκτήσουμε παραπάνω γνώσεις σε τομείς που κατείχαμε τα βασικά, αλλά και σε άλλους που δεν είχαμε ασχοληθεί ποτέ μέχρι τώρα. Καταλάβαμε ότι το modeling τρισδιάστατων αντικειμένων ενώ είναι μια διαδικασία πιο πολύπλοκη από όσο ξέραμε, παρουσίασε μεγάλο ενδιαφέρον και πολύ μεγάλη ικανοποίηση. Το βασικό πρόγραμμα που μας ενδιέφερε να μάθουμε να χρησιμοποιούμε, το Unity 3d, μας έδωσε καλές βάσεις για το επόμενο παιχνίδι που θα δημιουργήσουμε.

Άλλο ένα σημαντικό σημείο το οποίο χρειάστηκε να δουλέψουμε, ήταν η συνεργασία μεταξύ μας για την εκπόνηση των διάφορων εργασιών. Η συνεργασία μεταξύ ατόμων σε μια ομάδα είναι ένα περίπλοκο θέμα το οποίο αν δεν οργανωθεί σωστά μπορεί να δημιουργήσει κολλήματα στο workflow της ομάδας. Το πρόβλημα αυτό σίγουρα θα πολλαπλασιάζεται όταν τα μέλη της ομάδας είναι πάνω από δύο. Αν όμως υπάρχει σωστή οργάνωση, το project που πρέπει να υλοποιηθεί θα προχωρήσει χωρίς προβλήματα.

Κλείνοντας πρέπει να αναφέρουμε πως η όλη πτυχιακή εργασία ήταν μια πολύ καλή εισαγωγή σχετικά με το τι γνώσεις πρέπει να αποκτήσουμε, σε ποιους τομείς, πώς να ξεπερνάμε τα προβλήματα που θα εμφανίζονται, την σωστή προετοιμασία και την οργανωμένη ομαδική προσπάθεια και δουλειά. Το Unity 3d αποδείχθηκε μια πάρα πολύ καλή game engine, φιλική προς τον αρχάριο χρήστη, με πολύ καλό documentation, πολύ καλή κοινότητα, πληθώρα λειτουργιών που μπορούν να δώσουν σε δημιουργικά μυαλά τα εργαλεία για να δημιουργήσουν μεγάλο αριθμό ηλεκτρονικών παιχνιδιών. Η όλη εμπειρία βέβαια μας έδειξε πως έχουμε να μάθουμε ακόμα πολλά πράγματα όσον αφορά τον τομέα του modeling του rigging και του animation. Το πιο ευχάριστο και δημιουργικό κομμάτι βέβαια ήταν το scripting των assets στο Unity 3d. Ο συνδυασμός της δημιουργίας των δικών μας assets με την δυνατότητα να τα προγραμματίσουμε για να αλληλεπιδρά ο παίκτης όπως το φανταζόμαστε, ήταν μια επιβράβευση από μόνο του. Πιστεύουμε πως η πτυχιακή αυτή μας έδωσε τις βάσεις και την άνεση για να ξεκινήσουμε τα επόμενα projects που θέλουμε να υλοποιήσουμε σε επαγγελματικό επίπεδο.