Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Πτυχιακή Εργασία

"Ανάπτυξη 3D παιχνιδιού με JavaScript και κύρια εργαλεία το Unity και το Blender"

Σταμούλιας Ανδρέας (ΑΜ: 1934)

Επιβλέπων Καθηγητής: κ. Μαλάμος Αθανάσιος

Επιτροπή αξιολόγησης:

Ηράκλειο, Νοέμβριος 2011

Abstract

The aim of this thesis is to set out the procedures to create a modern game, from the earliest stages of model and environment design patterns to programming and creating the executable file. During this thesis, we will discuss the techniques and methods for an optimal effect between visual presentations and performance. Finally we will refer to the tools used to design this game and the techniques followed in them.

The first chapter is referred to Autodesk 3ds Max, the technologies offered and the manner used by us. Also presented through examples the box modeling technique and the process of uvw unwrap.

The second chapter presents the program Pixologic ZBrush, mentioned technologies but also new technologies that we meet as the pixol, polypaint and sculpt modeling. Finally, continuing the example of the previous chapter, we show the procedure followed for the creation of hi-poly mesh, and how to use the program to extract the textures.

In the next section we refer to another 3D content design program, Blender game engine. Presents techniques for creating skeletal system (Armature), the use of the Inverse Kinematics modifier, the manner and process of linking the mesh with armature and the weight painting process of bones. Finally we talk about animation creation techniques, Action, NLA and Ipo Curve editors, as well as ways to export animations.

The fourth chapter is an extensive reference to Unity game engine outlining all the features and technologies offered. Then we present the editor, through a description of the basic menus and windows, what they represent, and how to use them. Still referring to the components supplied, illustrating WrapMode and Layer functions of animations and analyze basic function, such as Awake, Start, Update. Finally presented and illustrated through examples, several points of the code developed

The final chapter will present the final result of this thesis, through the analysis of game-play, the description of the graphic display and the various functions that we developed.

Σύνοψη

Στόχος της πτυχιακής εργασίας είναι να καταγραφούν οι διαδικασίες δημιουργίας ενός σύγχρονου παιχνιδιού, από τα πρώτα στάδια της σχεδίασης των μοντέλων και του περιβάλλοντος, μέχρι τον προγραμματισμό και την δημιουργία του εκτελέσιμου αρχείου. Κατά την διάρκεια αυτής, θα ασχοληθούμε με τις τεχνικές και τις μεθόδους για το βέλτιστο αποτέλεσμα μεταξύ εμφάνισης και λειτουργικότητας. Τέλος θα αναφερθούμε στα εργαλεία που χρησιμοποιήθηκαν για την σχεδίαση αυτού του παιχνιδιού και στις τεχνικές που ακολουθήθηκαν μέσα σε αυτά.

Στο πρώτο κεφάλαιο αναφερόμαστε στο πρόγραμμα 3ds Max της Autodek, στις τεχνολογίες που προσφέρει αλλά και στον τρόπο που χρησιμοποιήθηκε από εμάς. Ακόμα παρουσιάζεται μέσα από παραδείγματα η τεχνική box modeling και η διαδικασία του uvw unwrap.

Στο δεύτερο κεφάλαιο παρουσιάζεται το πρόγραμμα ZBrush της Pixologic, αναφέρονται οι τεχνολογίες που προσφέρει αλλά και για τις νέες τεχνολογίες που συναντάμε, όπως αυτή του pixol, polypaint και του sculpt modeling. Τέλος συνεχίζοντας το παράδειγμα του προηγούμενου κεφαλαίου, δείχνουμε την διαδικασία που ακολουθήσαμε για την δημιουργία του hi-poly mesh, αλλά και τους τρόπους χρήσης του προγράμματος για την εξαγωγή των textures.

Στο επόμενο κεφάλαιο αναφερόμαστε σε ένα ακόμα πρόγραμμα σχεδίασης 3D περιεχομένου, το Blender game engine. Παρουσιάζονται τεχνικές για την δημιουργία σκελετικού συστήματος (Armature) και χρήση του modifier Inverse Kinematics, τον τρόπο και την διαδικασία σύνδεσης του mesh με το armature και την διαδικασία weight painting των bones. Τέλος αναφερόμαστε στις τεχνικές για την δημιουργία animation, στους Action, NLA και Ipo Curve editors, αλλά και στους τρόπους export των animations.

Στο τέταρτο κεφάλαιο γίνεται μια εκτενή αναφορά στο Unity game engine περιγράφονται όλες της δυνατότητες και τεχνολογίες που προσφέρει. Στην συνέχεια παρουσιάζουμε τον editor, μέσα από περιγραφή του βασικού μενού και των παραθύρων του, τι αντιπροσωπεύουν, καθώς και τον τρόπο χρήσης τους. Ακόμα αναφερόμαστε στα components που παρέχει, επεξηγούμε τις WrapMode και Layer λειτουργίες των animations και αναλύουμε την λειτουργία βασικών function, όπως τις Awake, Start, Update. Τέλος παρουσιάζονται και επεξηγούνται μέσα από παραδείγματα, αρκετά σημεία του κώδικα που αναπτύξαμε

Στο τελευταίο κεφάλαιο θα παρουσιάσουμε το τελικό αποτέλεσμα της πτυχιακής μας, μέσα από την ανάλυση του game-play, την περιγραφή των γραφικών στοιχείων της οθόνης και τις διάφορες λειτουργίες που έχουμε αναπτύξει.

Περιεχόμενα

Πίνακας Εικόνων

Κεφάλαιο 1

Κεφάλαιο 2

Κεφάλαιο 3

Κεφάλαιο 1

Autodesk 3ds Max

1.1 Ιστορική αναδρομή

Αρχικά το εργαλείο αυτό είχε δημιουργηθεί από την Yost Group και εκδοθεί από την Autodesk για DOS λειτουργικό σύστημα. Μετά την τέταρτη έκδοσή του σχεδιάστηκε και ξαναγράφτηκε από την ίδια εταιρία για λειτουργικό σύστημα Windows NT και μετονομάστηκε σε 3D Studio MAX. Εκδόθηκε αυτήν την φορά από την Kinetix, μια εταιρία που υπαγόταν εκείνη την περίοδο στο τμήμα πολυμέσων και ψυχαγωγίας της Autodesk. Αργότερα η Autodesk αγόρασε το εργαλείο αυτό και ξανά άλλαξε το όνομα του σε 3d max που αυτήν την φορά ήταν υπό την εποπτεία της Καναδικής εταιρίας Discreet. Από την όγδοη έκδοση και μετά το λογότυπο του εργαλείου είναι πλέον αυτό της Autodesk και από την ένατη έκδοση υπάρχει και στον τίτλο του.

1.2 Το εργαλείο σήμερα

Σήμερα το εργαλείο 3d studio Max της Autodesk παρέχει ένα πλήρες και ολοκληρωμένο σύστημα 3d μοντελοποίησης, animation αλλά και ένα rendering σύστημα που αποτελεί λύση για σχεδιαστές παιχνιδιών, γραφίστες και άλλου είδους καλλιτέχνες..

Αποτελεί ένα ισχυρό εργαλείο 3d μοντελοποίησης με ποικίλες τεχνικές όπως το box-modeling και το nurbs modeling. Η μετατροπή του κάθε αντικειμένου σε polygon, mesh, nurbs, η χρήση των build-in και plugged-in modifier αλλά και μια ακόμα πληθώρα εργαλείων, βοηθάνε τον κάθε χρήστη να προσαρμόσει το συγκεκριμένο εργαλείο στις ανάγκες του. Η εργαλειοθήκη του συστήματος texture mapping προσφέρει ένα πλήρες σύστημα UVW Map και Unwrap, ευελιξία και ταχύτητα στον χρήστη.

Επίσης προσφέρει ένα ισχυρό και προχωρημένο σύστημα character rigging και animation (Character Animation Toolkit). Υποστηρίζει λύσεις renderer όπως τον ενσωματωμένο 3ds max scan line και τον mental ray, ο οποίος υποστηρίζει και δικτυακό rendering.

Τέλος προγραμματιστές μπορούν μέσω του MAXscript και της C++ και της .NET να επεκτείνουν το εργαλείο στις ανάγκες τους, να δημιουργήσουν plug-in εργαλεία και compilers. Σε όλα αυτά έρχεται να προστεθεί μια τεράστια λίστα import/export καταλήξεων, όπως το .fbx, .3ds, .dxf, .ls, .obj, .xml .wrl και το .x3d, που υποστηρίζονται από πληθώρα σύγχρονων εργαλείων.

1.3 Τα σημεία χρήσης του στην ανάπτυξη της πτυχιακής

Ένα σύγχρονο 3d παιχνίδι αποτελείτε από πολλά στατικά και μη αντικείμενα με τουλάχιστον δυο ειδών maps (texture map και normal map), τα οποία δέχονται αλλά και δημιουργούν σκιές στο περιβάλλον, εκπέμπουν φώς, λειτουργούν σαν colliders, αλληλεπιδρούν μέσω physics με τον περιβάλλον και τον χρήστη, ενεργοποιούνται ή απενεργοποιούνται. Όλα αυτά παίζουν σημαντικό ρόλο στον ρυθμό των καρέ ανά δευτερόλεπτο ή fps. Ακόμα για την δημιουργία ενός ρεαλιστικού περιβάλλοντος είναι αναγκαίο να γεμίσει η σκηνή όσο το δυνατόν περισσότερο με αντικείμενα που θα φαίνονται αληθινά. Τα αντικείμενα αυτά χρειάζονται χρόνο για να γίνουν render και draw calls από τον επεξεργαστή, για αυτό σημαντικό είναι να τηρούνται κάποιοι κανόνες.

Έχοντας υπόψη τα παραπάνω και ξεκινώντας την δημιουργία ενός αντικειμένου, πρέπει να έχουμε υπόψη μας ότι το αντικείμενο πρέπει να στηρίζεται σε κάποιους κανόνες. Πρέπει να ορίσουμε την λειτουργικότητα του, δηλαδή να μην είναι ένα περιττό αντικείμενο στην σκηνή μας. Επίσης πρέπει να κάνουμε εξοικονόμηση στα πολύγωνα, δηλαδή να σχεδιάσουμε έξυπνα και να είμαστε σε θέση να αφαιρέσουμε περιττές λεπτομέρειες που ίσως και να μην φανούν στο παιχνίδι μας. Τέλος είναι καλό να μπορούμε να επαναχρησιμοποιούμε τα μοντέλα μας, όπως επίσης και τα texture μας καθώς αυτό μειώνει δραματικά την μνήμη που χρησιμοποιεί και τον όγκο των δεδομένων που πρέπει να επεξεργαστεί.

Στην συγκεκριμένη πτυχιακή εργασία το εργαλείο 3ds max χρησιμοποιήθηκε για την δημιουργία μέσω της τεχνικής box modeling των base mesh μοντέλων δηλαδή χρησιμοποιήθηκε για την δημιουργία του αρχικού σταδίου των 3D μοντέλων. Το κάθε μοντέλο σε αυτήν την φάση, αποτελεί την βάση πάνω στην οποία θα στηριχτούμε για την δημιουργία του τελικού αποτελέσματος. Το μοντέλο γίνεται export σε άλλα προγράμματα για την δημιουργία του hi-poly mesh, των texture maps και των animations. Όλη αυτή την διαδικασία θα την περιγράψουμε παρακάτω αφού όμως πρώτα γνωρίσουμε το 3ds Max Ο καλύτερος τρόπος να καλύψουμε όλα τα βασικά σημεία και να δείξουμε τις τεχνικές που χρησιμοποιήθηκαν είναι μέσα από ένα παράδειγμα.

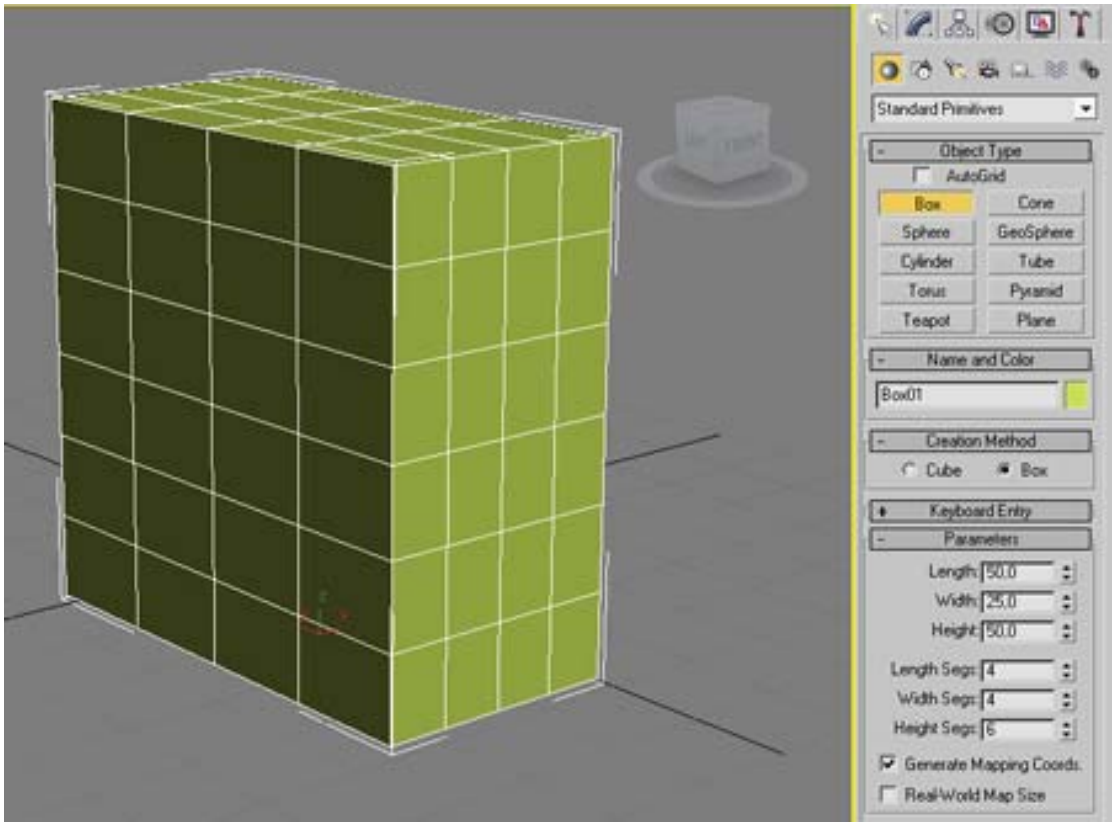
1.4 Παράδειγμα χρήσης της τεχνικής box modeling

Στο 3ds Max μέσω της τεχνικής box modeling δημιουργήσαμε το μεγαλύτερο μέρος των μοντέλων μας. Η τεχνική box modeling ονομάζεται η διαδικασία μοντελοποίησης με την χρήση primitive σχημάτων σε μια προσπάθεια δημιουργίας μιας πρώιμης φιγούρας του τελικού αποτελέσματος. Κατά την διάρκεια αυτής της διαδικασίας χρησιμοποιούμε τεχνικές extruding και scaling στα faces καθώς και μετακινήσεις των vertices. Στο παράδειγμα που ακολουθεί θα δείξουμε τα βασικά βήματα της δημιουργίας του κύριου χαρακτήρα, που αργότερα θα προσθέσουμε στο παιχνίδι μας.

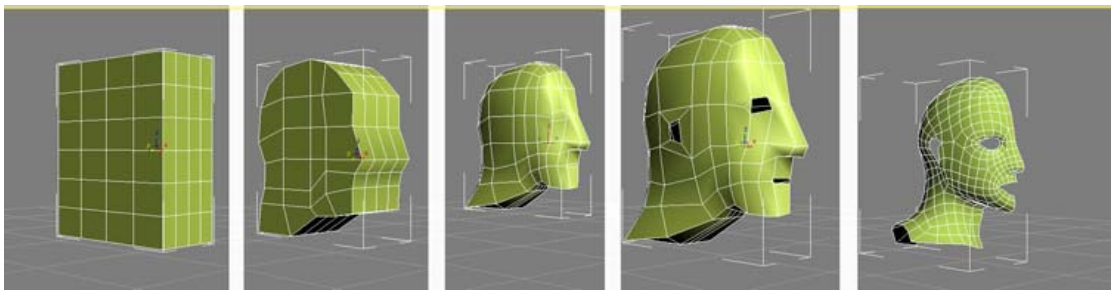
Αρχικά ξεκινάμε δημιουργώντας ένα primitive Box πηγαίνοντας στο μενού Create→Standard Primitives→Box ή από το Create panel, και αφού είμαστε ευχαριστημένοι με τις παραμέτρους Length, Width και Height, όπως επίσης και με τον αριθμό των Segments των πλευρών του, κάνουμε δεξί κλικ πάνω στο αντικείμενο μας και στο μενού που εμφανίζεται, επιλέγουμε Convert To→Convert to Editable Poly. Μετατρέποντας το Box σε

Editable Poly μας δίνεται ή δυνατότητα να το επεξεργαστούμε επιλέγοντας κορυφές, ακμές, πολύγωνα ή ολόκληρα αντικείμενα σε περίπτωση που έχουμε προσκολλημένα άλλα meshes σε αυτό.

Στην εικόνα 1 εμφανίζεται το Create panel του 3ds Max, από όπου δημιουργήσαμε το Box καθώς και τις παραμέτρους που επιλέξαμε, ενώ στην επόμενη εμφανίζεται μια αλληλουχία από εικόνες εμφανίζοντας την διαδικασία δημιουργίας του προσώπου του χαρακτήρα μας.



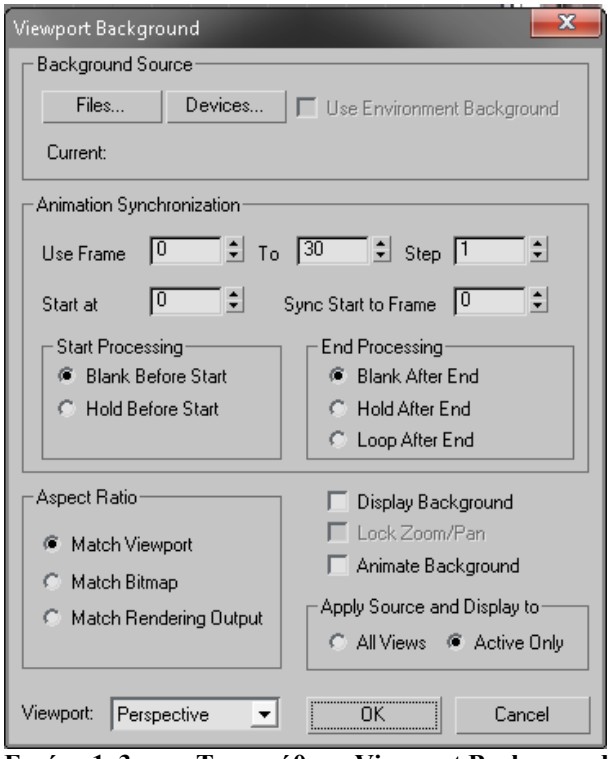
Εικόνα 1. 1 Δημιουργία primitive Box μέσω του Creation panel.



Εικόνα 1. 2 Δημιουργία του κεφαλιού του χαρακτήρα μας χρησιμοποιώντας την τεχνική box modeling

Η διαδικασία αυτή ακολουθήθηκε για την δημιουργία όλων των μοντέλων μας, ενώ σε πολλές περιπτώσεις χρησιμοποιήθηκαν φωτογραφίες ως σημεία αναφοράς στο background των viewports, ώστε να μπορούμε να δημιουργήσουμε το μοντέλο μας με μεγαλύτερη ακρίβεια. Κάτι τέτοιο μπορεί να γίνει επιλέγοντας το viewport στο οποίο

θέλουμε να βάλουμε μια εικόνα ως φόντο και στην συνέχεια πηγαίνουμε στο μενού View→Viewport Background→Viewport Background ή πατώντας Alt + B και στο παράθυρο που εμφανίζεται δηλώνουμε την εικόνα ή το βίντεο καθώς επίσης και το Aspect Ratio, animation synchronization και διάφορες άλλες παραμέτρους όπως βλέπουμε στην εικόνα 3.

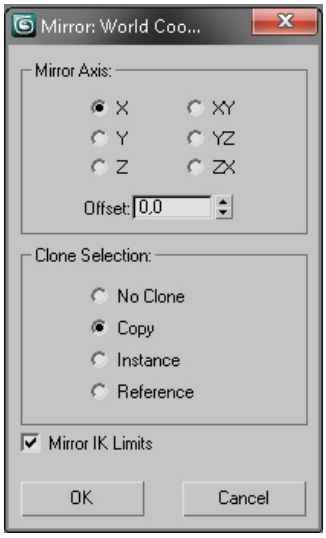


Εικόνα 1. 3 Το παράθυρο Viewport Background

Έχοντας πλέον δηλώσει τις κατάλληλες φωτογραφίες ως φόντο στα viewports μπορούμε μέσω τεχνικών box modeling να δημιουργήσουμε σιγά-σιγά τα διάφορα κομμάτια του χαρακτήρα μας. Στην αρχή μας ενδιαφέρει μόνο να δώσουμε το σωστό σχήμα στα primitive meshes μας έτσι ώστε να είναι όσο πιο κοντά στην reference εικόνα του φόντου μας. Μέσω διαδικασιών extrude, bevel, outline, insert, split, cut, slice, tessellate, relax, weld, bridge και break, όπως επίσης και με της μετακινήσεις ή διαγραφές vertices, edges και polygons μπορούμε να δημιουργήσουμε μια πρώτη φιγούρα του αντικειμένου που στοχεύουμε.

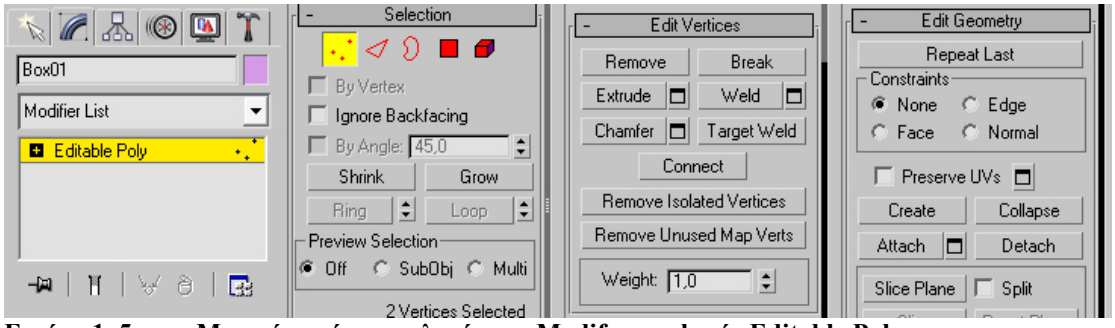
Αφού ολοκληρώσουμε την σχεδίαση της μιας πλευράς του μοντέλου μας, μέσω του Mirror μπορούμε να δημιουργήσουμε το άλλο μισό του. Έχοντας επιλεγμένο το μοντέλο μας, πηγαίνουμε στο Tools→Mirror, όπου επιλέγουμε να δημιουργήσουμε ένα αντίγραφου του αντικειμένου μας ανεστραμμένο ως προς τον άξονα x. Σε περίπτωση που το μοντέλο μας δεν είναι ευθυγραμμισμένο με την οθόνη μας, μπορούμε να το κάνουμε rotate ή να επιλέξουμε να δημιουργήσουμε ένα mirrored αντίγραφο σε άλλον άξονα. Επίσης μπορούμε να δώσουμε μια τιμή στο Offset ώστε να μετακινηθεί δεξιά ή αριστερά στον επιλεγμένο άξονα το αντίγραφο μας, ενώ σε περίπτωση που θέλουμε να δουλέψουμε ακόμα στο αντικείμενο μας, αλλά να βλέπουμε και τις δύο πλευρές, μπορούμε να επιλέξουμε να δημιουργήσουμε ένα Reference, έτσι ώστε ότι αλλαγές γίνονται στο πρωτότυπο να μεταφέρονται και στο mirrored.

Στην εικόνα 4 μπορούμε να δούμε το παράθυρο του εργαλείου Mirror με τις επιλογές σε άξονες που μπορούμε να επιλέξουμε για την δημιουργία του ανεστραμμένου αντικειμένου μας, όπως επίσης και τις cloning επιλογές του.

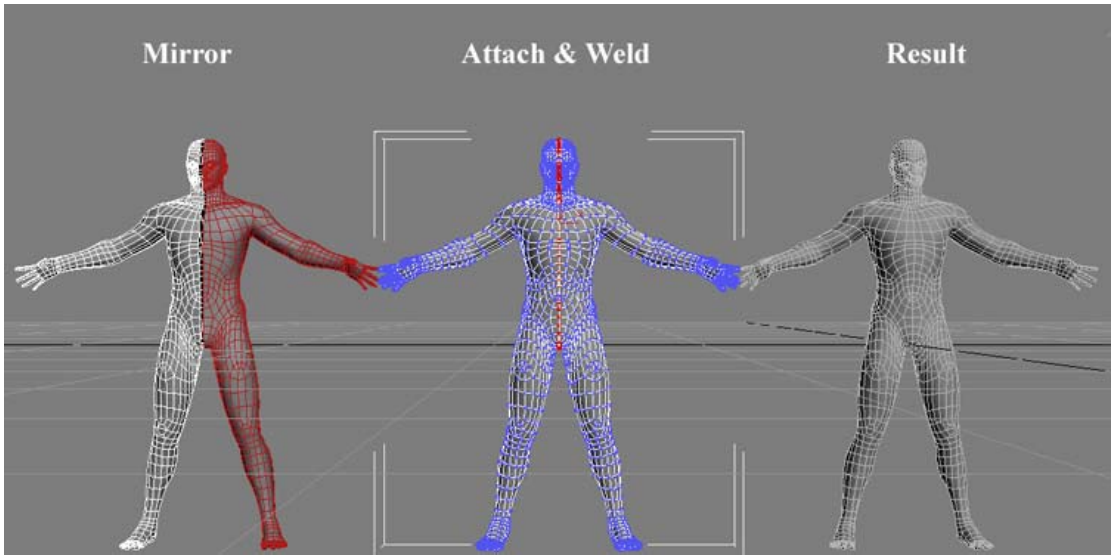


Εικόνα 1. 4 Το παράθυρο του εργαλείου Mirror

Αφού πλέον έχουμε δημιουργήσει το mirrored αντίγραφο, επιλέγουμε ένα από τα δυο κομμάτια και πηγαίνουμε στον Modify panel και στην κατηγορία Edit Geometry, πατάμε το κουμπί Attach και στην συνέχεια επιλέγουμε το άλλο κομμάτι του χαρακτήρα μας. Με αυτήν την διαδικασία έχουμε ενώσει τις δυο πλευρές σε ένα αντικείμενο και τώρα μένει να κλείσουμε το κενό μεταξύ του αριστερού και δεξιού κομματιού. Πλέον αφού έχουμε μόνο ένα αντικείμενο και μπορούμε να επεξεργαστούμε και τις δυο πλευρές ταυτόχρονα, πηγαίνουμε στο Modify panel και επιλέγουμε Vertex από την κατηγορία Selection. Ενεργοποιώντας την επιλογή Vertex βλέπουμε ότι όλα τα vertices του μοντέλου μας γίνονται ενεργά και έτσι μπορούμε να επεξεργαστούμε όλα τα σημεία του. Κρατώντας πατημένο το πλήκτρο Ctrl, έτσι ώστε να μπορούμε να επιλέξουμε πολλαπλά σημεία, επιλέγουμε όλα εκείνα τα vertices που εφάπτονται μεταξύ τους ή που πιστεύουμε ότι θα έπρεπε να ενωθούν, και στην συνέχεια πατάμε το κουμπί Weld της κατηγορία Edit Vertices του Modify panel. Σε περίπτωση που υπάρχουν ανεπιθύμητες ενώσεις μεταξύ των σημείων ή και καθόλου ενώσεις, μπορούμε να κάνουμε Weld ένα-ένα τα σημεία που επιθυμούμε ή αντίστοιχα να αυξήσουμε στις ρυθμίσεις του κουμπιού Weld την τιμή Weld Threshold.



Εικόνα 1. 5 Μερικές από τις επιλογές του Modify panel ενός Editable Poly με ενεργοποιημένα τα vertices



Εικόνα 1. 6 μοντέλο μας κατά την διαδικασία Mirror και συνένωσής του

1.5 Παράδειγμα χρήσης της τεχνικής uvw unwrap

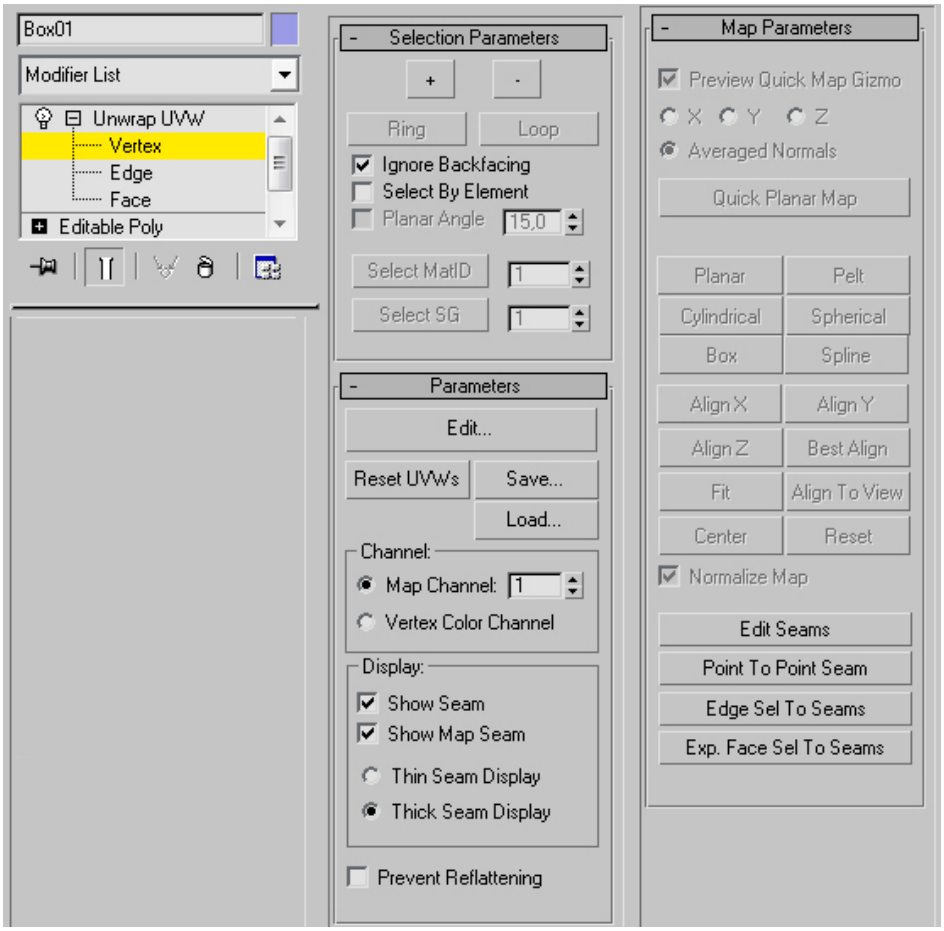
Τώρα αφού πλέον έχουμε τελειώσει με την σχεδίαση του χαρακτήρα μας, μένει να δείξουμε τα βήματα που ακολουθήσαμε στην διαδικασία του uvw mapping και uvw unwrapping. Με το uvw mapping στα γραφικά υπολογιστών εννοούμε μια μαθηματική τεχνική με την οποία μπορούμε να μετατρέψουμε μια 2D εικόνα σε ένα 3D αντικείμενο συγκεκριμένης τοπολογίας UVW, δηλαδή να αποδώσουμε στα σημεία της εικόνας τιμές σε τρεις διαστάσεις όπως σε ένα καρτεσιανό σύστημα συντεταγμένων. Η τρίτη διάσταση επιτρέπει στους χάρτες να τυλίγονται με πολύπλοκους τρόπους σε ανώμαλες επιφάνειες, ενώ το κάθε σημείο του UVW χάρτη αντιστοιχεί σε ένα σημείο του 3D αντικειμένου. Με την διαδικασία uvw unwrapping στην ουσία προσπαθούμε να σχηματίσουμε αυτόν τον 2D χάρτη που αναφέραμε παραπάνω επιλέγοντας τον τρόπο και την θέση στην οποία θα εμφανίζονται τα σημεία του μοντέλου μας στον χάρτη αυτόν.

Το 3ds Max, όπως και το κάθε πρόγραμμα δημιουργίας γραφικών που υποστηρίζει αυτήν την λειτουργία, έχει ενσωματωμένη και υποστηρίζει αυτήν την μαθηματική συνάρτηση με την οποία μπορεί να απεικονιστούν όλα τα σημεία ενός μοντέλου σε δυο διαστάσεις και το ανάποδο. Το 3ds Max προσφέρει διάφορους τρόπους απεικόνισης του 3D αντικειμένου στον χάρτη, δημιουργώντας gizmos σε primitive σχήματα όπως Box, Cylindrical, Spherical, Planar και με διαφορετικές ευθυγραμμίσεις στους άξονες x,y,z αλλά και σε σχέση με την οθόνη, μέσω των οποίων ξεδιπλώνεται το μοντέλο μας. Η χρησιμότητα των gizmos συνίσταται μόνο σε απλά γεωμετρικά σχήματα, καθώς σε πολύπλοκα αντικείμενα δεν υπάρχει σωστός τρόπος υπολογισμού του σημείου από το οποίο θα ξεκινήσει το "ξεδίπλωμα".

Στην τεχνική uvw unwrapping πολύπλοκων αντικειμένων, όπως του χαρακτήρα μας, δημιουργούμε χειροκίνητα τους χάρτες μας, αφού σημειώνουμε πάνω στο μοντέλο μας τον τρόπο με τον οποίο επιθυμούμε να "ξεδιπλωθεί", χωρίζοντας το σε κομμάτια όπου κρίνουμε ότι είναι αναγκαίο. Στην διαδικασία αυτήν δεν υπάρχει σωστός ή λάθος τρόπος, ο καθένας μπορεί να εφαρμόσει τις δικές του τεχνικές, όμως ο χάρτης που θα παραχθεί θα πρέπει να

είναι ευδιάκριτος, όλα τα σημεία θα πρέπει να είναι τακτοποιημένα μέσα στα όρια του uvw template δίνοντας μεγαλύτερο χώρο στα σημεία που μας ενδιαφέρουν περισσότερο και τέλος δεν θα πρέπει να υπάρχουν faces και vertices πάνω από άλλα.

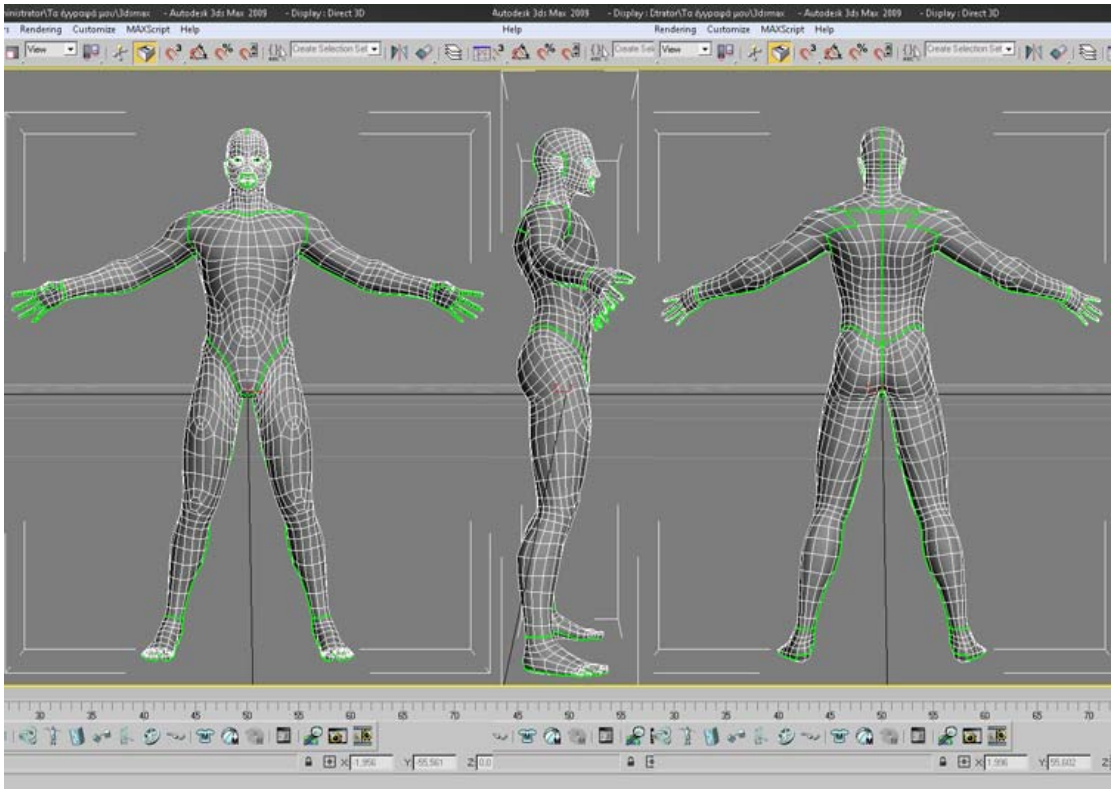
Συνεχίζοντας το παράδειγμα θα δείξουμε την διαδικασία δημιουργία αυτού του χάρτη και τον τρόπο με τον οποίο δηλώνουμε τα σημεία πάνω στο αντικείμενο μας και τον τρόπο που θέλουμε να "ξεδιπλωθεί". Αρχικά εφαρμόζουμε στο αντικείμενο μας τον modifier Unwrap UVW, ο οποίος βρίσκεται στο Modify panel→Modifier List→Unwrap UVW. Εφαρμόζοντας τον Unwrap UVW modifier βλέπουμε, ότι πλέον δεν μπορούμε να κάνουμε αλλαγές στο μοντέλο μας, καθώς αν γίνει κάτι τέτοιο θα χαθούν όλες οι απαραίτητες πληροφορίες συντεταγμένων που έχουμε ορίσει. Επίσης στον Modify panel εμφανίζονται τρεις κατηγορίες με διάφορες παραμέτρους, Selection Parameters, Parameters και Map Parameters, όπως φαίνονται και στην εικόνα 7 παρακάτω.



Εικόνα 1. 7 Οι παράμετροι του Unwrap UVW Modifier

Για τον ορισμό των σημείων πάνω στο μοντέλο, όπου θέλουμε να δημιουργήσουμε τις "ραφές" και οι οποίες θα αποτελούν τις άκρες του χάρτη συντεταγμένων μας, πρέπει να ενεργοποιήσουμε τον Unwrap UVW modifier επιλέγοντας μια από τις επιλογές Vertex, Edge, Face, έτσι ώστε να μπορούμε να επιλέξουμε τις παραμέτρους του. Στην κατηγορία Map Parameters ενεργοποιούμε το κουμπί Point To Point Seam, το οποίο μας επιτρέπει να

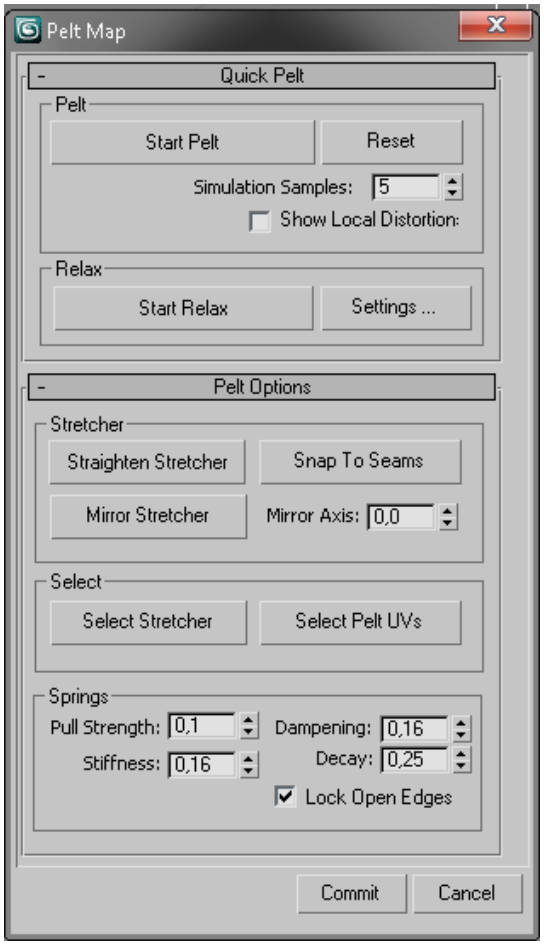
διαμορφώσουμε πατώντας πάνω στα vertices του αντικειμένου μια διαδρομή. Χρησιμοποιώντας την τεχνική Point To Point Seam μπορούμε να χωρίσουμε το μοντέλο μας σε κομμάτια ή να διαμορφώσουμε τον τρόπο που θα ξεδιπλωθεί κάποιο από αυτά, επιλέγοντας τα σημεία τομής του. Στην εικόνα 8 παρακάτω απεικονίζεται το μοντέλο μας μετά το πέρας της διαδικασίας αυτής, ενώ με πράσινο χρώμα εμφανίζονται οι "ραφές" που έχουμε δημιουργήσει.



Εικόνα 1. 8 Το μοντέλο με ενεργοποιημένα τα seams, σε front, left και back view, όπως φαίνεται μέσα από το 3ds Max

Τελειώνοντας αυτήν την διαδικασία, έχουμε ορίσει τα σημεία τομής του μοντέλου μας και τώρα μας μένει να απεικονίσουμε αυτά τα 3D κομμάτια σε μια 2D εικόνα, και τέλος να δημιουργήσουμε τον χάρτη μας. Για να το κάνουμε αυτό, αλλάζουμε από vertex σε faces, έτσι ώστε να μπορούμε να επιλέγουμε πολύγωνα και στην συνέχεια πατάμε το κουμπί Edit της κατηγορίας Parameters. Αμέσως εμφανίζεται στην οθόνη μας το παράθυρο Edit UVWs, μέσα στο οποίο θα δημιουργήσουμε τον χάρτη μας. Επιλέγουμε τα faces κάθε κομματιού που δημιουργήσαμε, είτε με το mouse είτε με το Expand Selection της κατηγορίας Selection Mode του Edit UVWs παραθύρου και στην συνέχεια πατάμε το κουμπί Pelt της κατηγορίας Map Parameters του Modify panel.

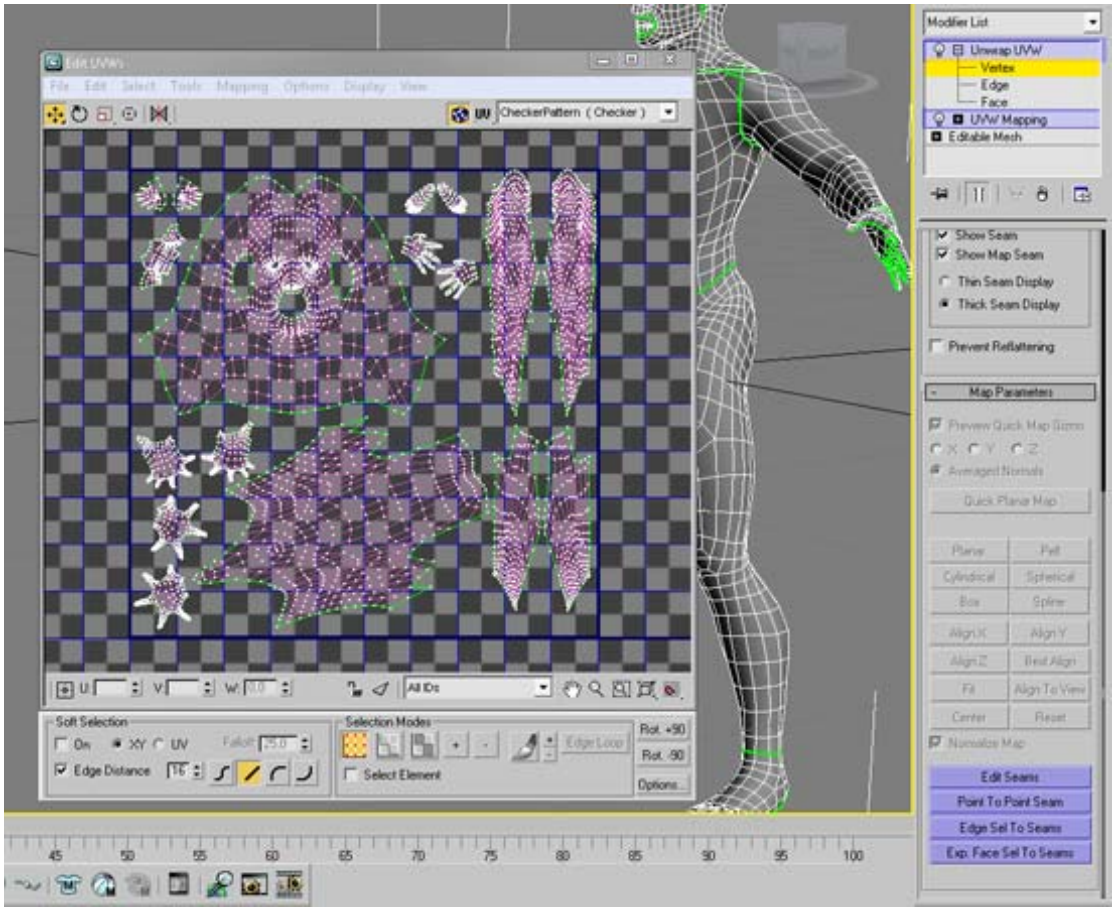
Με την λειτουργία Pelt μπορούμε να ξεδιπλώσουμε τα σημεία του μοντέλου μας, σαν να τραβάγαμε της άκρες του, δηλαδή στην πραγματικότητα τεντώνει το μοντέλο μας τραβώντας το από τα seams που δημιουργήσαμε. Στο παράθυρο Pelt Map πατάμε Start Pelt και όταν είμαστε ευχαριστημένοι από το αποτέλεσμα κάνουμε Stop Pelt και στην συνέχεια Commit. Εάν έχει γίνει σωστή δημιουργία των seams, η διαδικασία Pelt δεν εμφανίζει ποτέ προβλήματα και η εξομοίωση είναι στιγμιαία, παρόλα αυτά σε περίπτωση που έχουμε ανεπιθύμητα αποτελέσματα μπορούμε να αλλάξουμε τις παραμέτρους του από το Pelt Options και να ξανακάνουμε Pelt ή να αλλάξουμε τα seams μας.



Εικόνα 1. 9 Το εργαλείο Pelt και οι ρυθμίσεις του

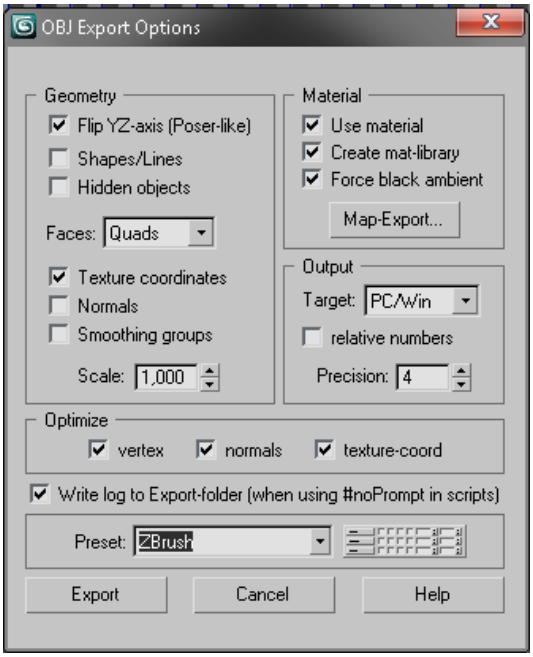
Όταν έχουμε τελειώσει πλέον με την διαδικασία του Pelt σε ολόκληρο το μοντέλο μας, με την χρήση των Move, Rotate, Scale, Freeform Mode και Mirror εργαλείων του Edit UVWs παραθύρου, δημιουργούμε μέσα στο προκαθορισμένο τετράγωνο τον χάρτη μας, όπως φαίνεται στην εικόνα 10 παρακάτω. Σε αυτήν την διαδικασία δίνουμε μεγαλύτερο βάρος, δηλαδή αυξάνουμε το μέγεθος στα κομμάτια του μοντέλου που είναι άμεσα ορατά ή που θέλουμε να δημιουργήσουμε περισσότερες λεπτομέρειες, όπως κάναμε στην περίπτωσή μας στο πρόσωπο.

Μέσα από το Edit UVWs παράθυρο μπορούμε να δημιουργήσουμε τον χάρτη UVW του μοντέλου μας πηγαίνοντας στο Tools→Render UVW Template και αφού ρυθμίσουμε την ανάλυση που θέλουμε πατάμε Render UV Template, σώνουμε την εικόνα και στην συνέχεια μπορούμε να την επεξεργαστούμε και να την χρωματίσουμε στο Photoshop ή σε άλλο παρόμοιο πρόγραμμα. Όμως στην συγκεκριμένη πτυχιακή εργασία η δημιουργία των maps θα γίνει από άλλο πρόγραμμα 3D και συγκεκριμένα από το ZBrush στο οποίο θα αναφερθούμε παρακάτω.



Εικόνα 1. 10 Απεικόνιση του χάρτη UVW του μοντέλου, όπως δημιουργήθηκε στο 3ds Max

Με την διαδικασία του UVW Unwrapping τελειώνει και η χρησιμότητα του συγκεκριμένου προγράμματος και το μόνο που μένει είναι να κάνουμε export σε κατάλληλη μορφή το μοντέλο μας για την επόμενη φάση της επεξεργασίας του. Στην συνέχεια το μοντέλο θα εισαχθεί στο ZBrush για την δημιουργία της hi-poly έκδοσης του και των κατάλληλων maps. Για να γίνει αυτό, επιλέγουμε File→Export στην κύρια γραμμή μενού του 3ds Max και αφού δηλώσουμε την διεύθυνση που θέλουμε να δημιουργήσουμε το αρχείο, επιλέγουμε την μορφή .obj, αφού είναι και η μόνη που υποστηρίζεται από το ZBrush, και πατάμε αποθήκευση. Στο παράθυρο που εμφανίζεται επιλέγουμε από το Preset το ZBrush, έτσι ώστε να μετατραπούν κατάλληλα οι άξονες xyz, το scale size και άλλες ρυθμίσεις των παραμέτρων του και στην συνέχεια πατάμε Export.



Εικόνα 1. 11 Οι ρυθμίσεις του export option της μορφής .obj για χρήση στο πρόγραμμα ZBrush

Κεφάλαιο 2

Pixologic ZBrush

2.1 Πρόλογος

Το ZBrush είναι ένα εργαλείο 3D γλυπτικής που συνδυάζει 3D/2.5D μοντελοποίηση, τεχνικές texturing και ζωγραφικής. Το πρόγραμμα χρησιμοποιεί την τεχνολογία "pixol" ή οποία αποθηκεύει φωτισμό, χρώμα και το υλικό του αντικειμένου, καθώς και πληροφορίες για το βάθος. Η κύρια διαφορά ανάμεσα στο ZBrush και σε άλλα παρόμοια πακέτα είναι ότι εδώ οι διαδικασίες που ακολουθούνται μοιάζουν ποιο πολύ με γλυπτική. Το ZBrush χρησιμοποιείται ευρέως από τις βιομηχανίες ταινιών, animation και παιχνιδιών, αφού είναι ικανό να παράγει υψηλής ανάλυσης 3D μοντέλα με πάνω από 10 εκατομμύρια πολύγωνα, ενώ παράλληλα δίνει την δυνατότητα στο χρήστη να κινηθεί ανάμεσα στα επίπεδα των αναλύσεων και κάνει αλλαγές.

2.2 Ιστορική αναδρομή

Το ZBrush δημιουργήθηκε από την εταιρία Pixologic Inc. πού ιδρύθηκε από τον Ofer Alon και τον Jack Rimokh και παρουσιάστηκε για πρώτη φορά το 1999 στο ετήσιο συνέδριο των computer graphics (CG), SIGGRAPH. Η πρώτη demo έκδοση του προγράμματος, το ZBrush 1.55, παρουσιάστηκε το 2002 και ακολούθησαν οι εκδόσεις 3.1 το 2007, 3.5 το 2009 και ZBrush 4 το 2010.

2.3 Το εργαλείο σήμερα

Το ZBrush στην τωρινή του έκδοση προσφέρει μοναδικά χαρακτηριστικά για την δημιουργία και επεξεργασία των meshes. Έχει ένα μεγάλο αριθμό από 3D brushes για την επεξεργασία των μοντέλων, με διαφορετικά χαρακτηριστικά όπως την σκληρότητα, την τιμή alpha, το σχήμα, αλλά και τον τρόπο που εφαρμόζουν πάνω στο μοντέλο. Επίσης μέσα από το σύστημα Polypaint προσφέρει την δυνατότητα να χρωματίσουμε την επιφάνεια ενός αντικειμένου σε 3D περιβάλλον χωρίς να έχουμε ορίσει ένα texture map. Με το σύστημα Illustration το ZBrush μας δίνει την δυνατότητα να μεταφερθούμε σε ένα 2.5D περιβάλλον και να χρωματίσουμε, να αλλάξουμε την επιφάνεια, το υλικό, την θέση και τον φωτισμό του αντικειμένου και εν τέλει να μεταφέρουμε όλες αυτές τις αλλαγές στο μοντέλο μας μέσω του

συστήματος pixol. Ακόμα το πρόγραμμα μέσω του Transpose παρέχει παρόμοιες τεχνικές σκελετικού συστήματος, όπως άλλα 3D πακέτα, με το οποίο μπορούμε να αλλάξουμε την πόζα του μοντέλου μας χωρίς όμως την ανάγκη να εφαρμοστούν διαδικασίες skeletal rigging. Ένα μοναδικό χαρακτηριστικό του ZBrush είναι το ZSpheres, το οποίο επιτρέπει την δημιουργία ενός base mesh, από ένα σύνολο σφαιρών το οποίο αφού του σώσουμε το βασικό σχήμα του αντικειμένου που θέλουμε να δημιουργήσουμε, μετατρέπεται σε ένα επεξεργάσιμο αντικείμενο. Τέλος το ZBrush μέσω του συστήματος GoZ επιτρέπει την σύνδεση του με άλλα 3D προγράμματα μέσω μιας αυτόματης διαδικασίας όπου μεταφέρονται τα αντικείμενα καθώς και πληροφορίες των normal, displacement και texture maps. Το σύστημα αυτό επιτρέπει την σύνδεση του προγράμματος με το 3DS MAX, Maya, Modo, Cinema 4D αλλά και με το Photoshop και είναι υπεύθυνο για την διαχείριση όλης της διαδικασίας καθώς και για την διόρθωση σημείων, την τακτοποίηση των meshes, και την αλλαγή των αξόνων.

2.4 Τα σημεία χρήσης του στην ανάπτυξη της πτυχιακής

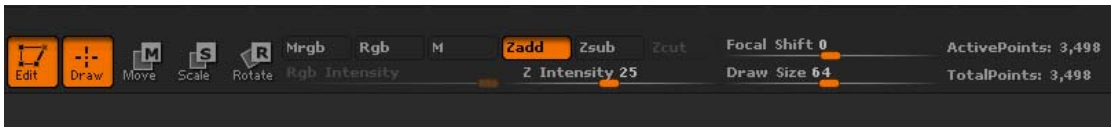
Το πρόγραμμα αυτό χρησιμοποιήθηκε στην πτυχιακή για την δημιουργία των hi-poly meshes και εν συνεχεία για την δημιουργία όλων των απαραίτητων texture maps. Σε πολλές περιπτώσεις χρησιμοποιήθηκε ακόμα και στις διαδικασίες του uvw mapping και uv unwrapping. Το πρόγραμμα ZBrush επιλέχθηκε για αυτήν την διαδικασία καθώς προσφέρει μια ιδιαίτερη τεχνική μοντελοποίησης που μοιάζει αρκετά με αυτή της γλυπτικής. Παρόμοιες διαδικασίες συναντάμε και στο Blender, που όμως δεν χρησιμοποιήθηκε σε αυτήν την φάση καθώς το ZBrush προσφέρει αρκετά πιο πολλές, βελτιωμένες και εξειδικευμένες λειτουργίες και τεχνικές, όπως επίσης και την δυνατότητα δημιουργίας εκατομμύριων πολυγώνων χωρίς προβλήματα σε ότι αφορά την μνήμη.

2.5 Γνωριμία με τον editor και παράδειγμα δημιουργίας ενός hi-poly mesh

Στο παράδειγμα μας, θα δείξουμε όλες τις διαδικασίες που ακολουθήθηκαν για την επεξεργασία του μοντέλου μας κατά την χρήση του προγράμματος ZBrush. Αρχικά εισάγουμε το μοντέλο που κάναμε export στην μορφή Wavefront .obj από το 3ds Max πηγαίνοντας στο Tools→Import.

Εικόνα 2. 1 Το Tool μενού του ZBrush

Αφού φορτωθεί το μοντέλο μας στο πρόγραμμα, για να το εμφανίσουμε στην οθόνη πρέπει να σύρουμε κρατώντας πατημένο το αριστερό κλικ του ποντικιού πάνω στην άδεια σκηνή του ZBrush και να το αφήσουμε όταν δούμε ότι εμφανίζεται στην οθόνη. Το μοντέλο μας, ή SubTool όπως ονομάζεται από το ZBrush βρίσκεται πλέον στην οθόνη αλλά δεν μπορούμε να επεξεργαστούμε αν πρώτα δεν πατήσουμε το πλήκτρο "T" ή Edit από τα εργαλεία της σκηνής όπως φαίνονται και στην εικόνα 2.



Εικόνα 2. 2 Τα εργαλεία της σκηνής του ZBrush για την διαχείριση των SubTools

Έχοντας ενεργοποιήσει το Edit Mode του αντικειμένου μπορούμε να ξεκινήσουμε την επεξεργασία του, να το χρωματίσουμε, να αλλάξουμε θέση, γωνία ή μέγεθος στο αντικείμενο μας ή σε μέρος αυτού, να διαλέξουμε material, να κάνουμε extract άλλων subtools μέσω masking διαδικασιών αλλά και να δημιουργήσουμε μέσω του subdivide mesh με διάφορες ρυθμίσεις ένα hi-poly mesh του μοντέλου μας, ώστε να είμαστε σε θέση να δημιουργήσουμε περισσότερες λεπτομέρειες και να δώσουμε καλύτερη οπτική απόδοση στο αντικείμενο μας.

Για να αυξήσουμε τον αριθμό των πολυγώνων του μοντέλου ή όπως αναφέρονται από το ZBrush τον αριθμό των Points πηγαίνουμε στο Tools→Geometry και πατάμε το κουμπί Divide έχοντας ενεργοποιημένο το Subdivide Smooth Modifier, έτσι ώστε να κάνουμε την επιφάνεια πιο λεία. Κάνοντας subdivide στο subtool μας βλέπουμε ότι δημιουργούνται επίπεδα ανάλυσης (subdivision levels) στα οποία μπορούμε να μετακινηθούμε χωρίς περιορισμούς, μπορούμε δηλαδή να μετακινηθούμε στο subdivision level 1, να κάνουμε μια αλλαγή στην επιφάνεια του μοντέλου και επιστρέφοντας στην τελευταίο επίπεδο ανάλυσης να έχουμε την ίδια αλλαγή, η οποία θα έχει διαμορφωθεί κατάλληλα από τους ίδιους modifiers που έχουμε επιλέξει σε κάθε επίπεδο ανάλυσης.

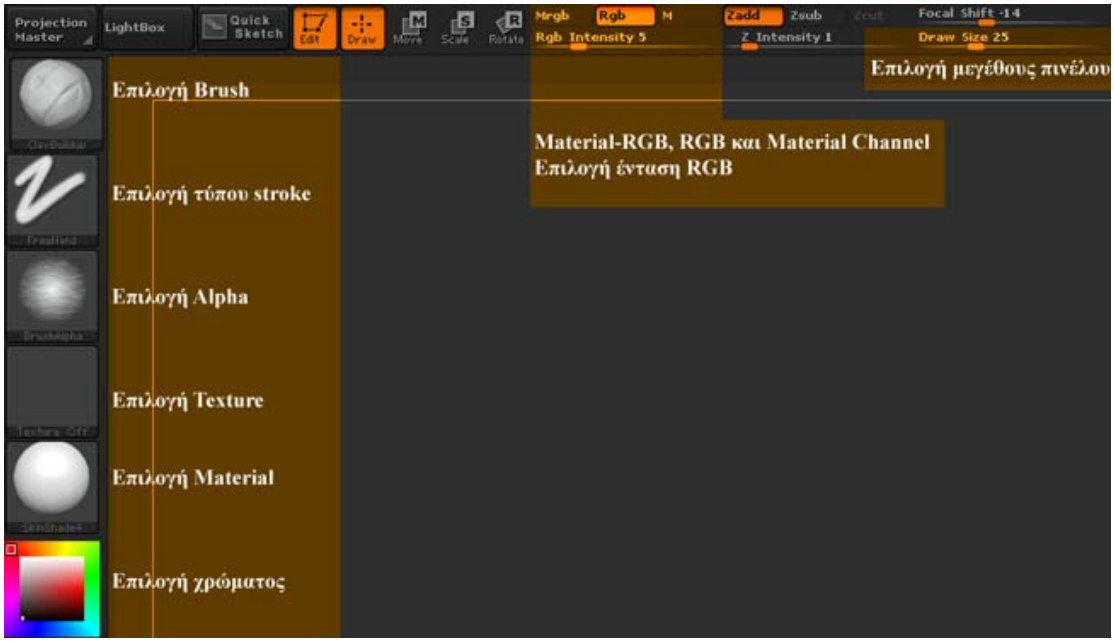
Εικόνα 2. 3 Το Geometry Tool για την δημιουργία των subdivision levels

Έχοντας δημιουργήσει μερικά επίπεδα ανάλυσης ξεκινάμε να δημιουργούμε λεπτομέρειες χρησιμοποιώντας τα κατάλληλα brushes με χρήση διάφορων alpha και strokes. Το ZBrush προσφέρει από μόνο του μια αρκετά μεγάλη λίστα από brushes, alphas και materials για να διαλέξουμε, ενώ υπάρχουν πολλά ακόμα που μπορούμε να προσθέσουμε κατεβάζοντας τα από την διαδικτυακή σελίδα του.

Για να δημιουργήσουμε ομοιόμορφα και συμμετρικά τις λεπτομέρειες και τα χαρακτηριστικά του μοντέλου μας, ενεργοποιούμε στο μενού Transform το κουμπί Activate Symmetry και επιλέγουμε τον άξονα x, έτσι ώστε να δημιουργούμε και στις δύο πλευρές τα ίδια χαρακτηριστικά.

Εικόνα 2. 4 Η κατηγορία Activate Symmetry του Transform μενού

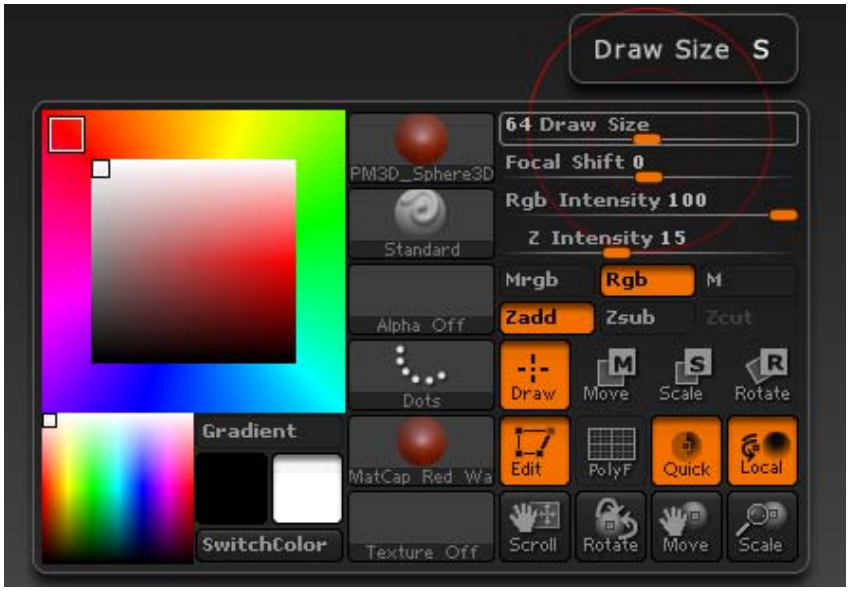
Στην συνέχεια επιλέγουμε από τα διαθέσιμα brushes, ένα που θα μας βοηθήσει να χτίσουμε το αντικείμενο μας, σε μια διαδικασία που μοιάζει σαν να βάζουμε ή βγάζουμε πηλό, όπως είναι το ClayBuildUp και ένα κατάλληλο Alpha για την κεφαλή του συγκεκριμένου πινέλου. Ακόμα ρυθμίζουμε την ένταση, το μέγεθος του πινέλου, ενεργοποιούμε το κουμπί Zadd ή Zsub αναλόγως τι θέλουμε να σχηματίσουμε προσθέτοντας ή αφαιρώντας πηλό και ξεκινάμε να δημιουργούμε το hi-poly μοντέλο μας. Εάν θέλουμε να χρωματίζουμε παράλληλα το μοντέλο μας καθώς δημιουργούμε τις λεπτομέρειες, ενεργοποιούμε το κουμπί RGB Channel των εργαλείων της σκηνής και επιλέγουμε την ένταση καθώς και το χρώμα.



Εικόνα 2. 5 Παρουσίαση των εργαλείων επιλογής brush, stroke alpha, texture, material και color που αναφέραμε παραπάνω

Αφού πλέον έχουμε κάνει όλα τα παραπάνω, μπορούμε με κινήσεις σαν να ζωγραφίζουμε να δημιουργήσουμε λεπτομέρειες, ενώ παράλληλα μπορούμε να κινούμαστε σε μικρότερα επίπεδα subdivision και να κάνουμε αλλαγές. Το ZBrush προφέρει Clay, Layer, Move, Polish, Magnify, Smooth και πολλά ακόμα είδη πινέλων τα οποία μπορούμε να χρησιμοποιήσουμε για να επιτύχουμε το οποιοδήποτε αποτέλεσμα.

Το πινέλο smooth από default μπορούμε να το χρησιμοποιήσουμε γρήγορα κρατώντας πατημένο το πλήκτρο Shift, ή μπορούμε να δηλώσουμε κάποιο άλλο στην θέση του, ενώ κρατώντας πατημένο το Ctrl ενεργοποιούμε το πινέλο με το οποίο μπορούμε να κάνουμε masking στο μοντέλο μας. Ακόμα κρατώντας πατημένο το συνδυασμό Ctrl + Shift, μπορούμε να επιλέξουμε ένα μόνο μέρος του μοντέλου μας να εμφανίζεται στην οθόνη, μια τεχνική που βοηθάει πολύ αφού συγκεντρωνόμαστε στο συγκεκριμένο σημείο αλλά μειώνει και την χρησιμοποιούμενη μνήμη αφού το πρόγραμμα διαχειρίζεται λιγότερα Active Points. Τέλος πατώντας δεξί κλικ σε οποιοδήποτε μέρος της σκηνής εμφανίζεται ένα παράθυρο με όλα τα απαραίτητα εργαλεία συγκεντρωμένα, έτσι ώστε να γλυτώνουμε χρόνο αποφεύγοντας μετακινήσεις του κέρσορα σε ολόκληρη την οθόνη, αλλά και να παραμένουμε συγκεντρωμένοι στην δουλειά μας.



Εικόνα 2. 6 Το παράθυρο με τα απαραίτητα εργαλεία που εμφανίζεται πατώντας δεξί κλικ στην σκηνή μας

Όλες οι παραπάνω τεχνικές που αναφέραμε θα μας βοηθήσουν να διαμορφώσουμε αποτελεσματικά το τελική έκδοση του μοντέλου μας και αφού αφιερώσουμε τον απαραίτητο χρόνο και έχουμε καταλήξει σε ένα αποτέλεσμα όπως αυτό της εικόνας 7, είμαστε σε θέση να προχωρήσουμε στο επόμενο βήμα.

Εικόνα 2. 7 Τα subdivision levels του χαρακτήρα που δημιουργήσαμε

Αφού πλέον έχουμε τελειώσει με το μοντέλο μας και είμαστε ευχαριστημένοι από αυτό που βλέπουμε, μένει να δημιουργήσουμε από το hi-poly τα texture, normal και όποια ακόμα maps νομίζουμε ότι χρειαζόμαστε και τα οποία θα εφαρμόσουμε αργότερα στον low poly χαρακτήρα που θα χρησιμοποιήσουμε στο παιχνίδι μας. Το ZBrush δίνει την δυνατότητα δημιουργίας texture, normal, displacement, ambient occlusion και cavity maps, ενώ παράλληλα μπορούμε να επιλέξουμε το subdivision level από το οποίο θα κάνουμε export.

2.5 Παράδειγμα της διαδικασίας δημιουργίας των maps.

Έχοντας κάνει UVW unwrap μέσα στο 3ds Max, έχουμε ετοιμάσει τον χάρτη συντεταγμένων του 3d μοντέλου μας και μπορούμε να δημιουργήσουμε το texture map πηγαίνοντας στο Tools→Texture Map→New From PolyPaint, το normal map πηγαίνοντας Tools→Normal Map→Create NormalMap και το displacement map στο Tools→Displacement Map→Create DispMap.

Εικόνα 2. 8 Texture Map, Displacement Map και Normal Map panel.

Στην συγκεκριμένη πτυχιακή παρ' όλα αυτά χρησιμοποιήθηκε ένα plug in που είναι διαθέσιμο για download από την official σελίδα του ZBrush, που μας επιτρέπει να δημιουργήσουμε ακόμα Cavity και Ambient Occlusion Maps. Μέσω του συγκεκριμένου plug-in που βρίσκεται στο Zplugin→Multi Map Exporter, μπορούμε να επιλέξουμε ένα ή και όλα τα είδη maps για κάθε ορατό subtool και μέσω μιας αυτόματης διαδικασίας να μας κάνει export στην διαδρομή που θα ορίσουμε όλους τους χάρτες γλιτώνοντας αρκετό χρόνο.

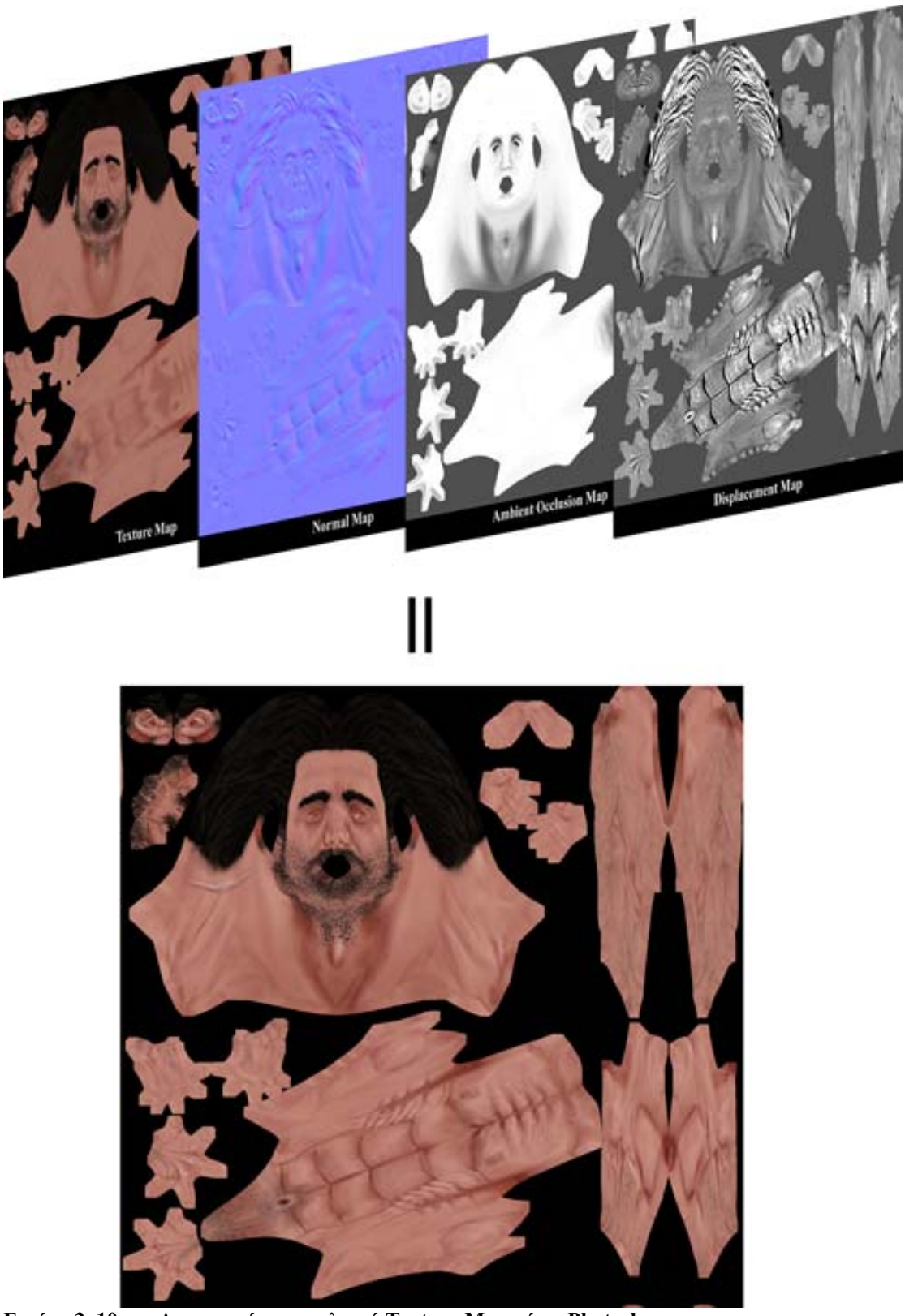
Για το συγκεκριμένο μοντέλο δημιουργήσαμε όλα τα διαθέσιμα είδη Map, με της ρυθμίσεις που φαίνονται στην εικόνα 9 παρακάτω, και στην συνέχεια τα επεξεργαστήκαμε κατάλληλα στο Photoshop, για την δημιουργία του τελικού Texture Map που θα χρησιμοποιήσουμε.

Εικόνα 2. 9 Οι ρυθμίσεις των παραμέτρων όλων των Map που δημιουργήθηκαν μέσω του Multi Map Exporter plug-in

Τους χάρτες που δημιουργήσαμε στην συνέχεια τους εισάγουμε στο Photoshop και τους τοποθετούμε σε ένα νέο document, τον καθένα σε διαφορετικό layer. Η διαδικασία που ακολουθούμε είναι απλή. Ο στόχος είναι να δημιουργήσουμε, επεξεργαζόμενοι το κάθε χάρτη και αλλάζοντας layer blending modes, ένα Texture Map με περισσότερες λεπτομέρειες, δηλαδή "ψήνουμε" επιπρόσθετες πληροφορίες στον χάρτη χρώματος (Texture Map). Αρχικά τοποθετείται σαν βάση το Texture Map και στην συνέχεια ο Normal Map, ο Ambient Occlusion και ο Displacement, αν και ο τελευταίο συνιστάται για μοντέλα μεγαλύτερης ανάλυσης απ' ότι το δικό μας, που θα χρησιμοποιηθεί στο subdivision level 1.

Τον Normal Map που τοποθετούμε ακριβώς ένα layer πιο πάνω και λόγω της ιδιομορφίας του τον μετατρέπουμε σε ασπρόμαυρο, πηγαίνοντας στο Image→Adjustments→Black & White και αφού επιλέξουμε από το Preset το Maximum Black, πατάμε Ok. Αυτή η διαδικασία μπορεί να διαφέρει από μοντέλο σε μοντέλο, αναλόγως με το οπτικό αποτέλεσμα που θέλουμε να δημιουργήσουμε. Η διαδικασία αυτή μας βοηθάει να κρατήσουμε αρκετές πληροφορίες, όπως αυτή του βάθους, στον ασπρόμαυρο χάρτη, και επιλέγοντας διαφορετικό Blending Mode στο layer, όπως είναι το Overlay, Soft Light, Vivid Light, Hard Light και Linear Light, να αποτυπώσουμε αυτές τις πληροφορίες στο προηγούμενο layer, που δεν είναι άλλο από το Texture Map.

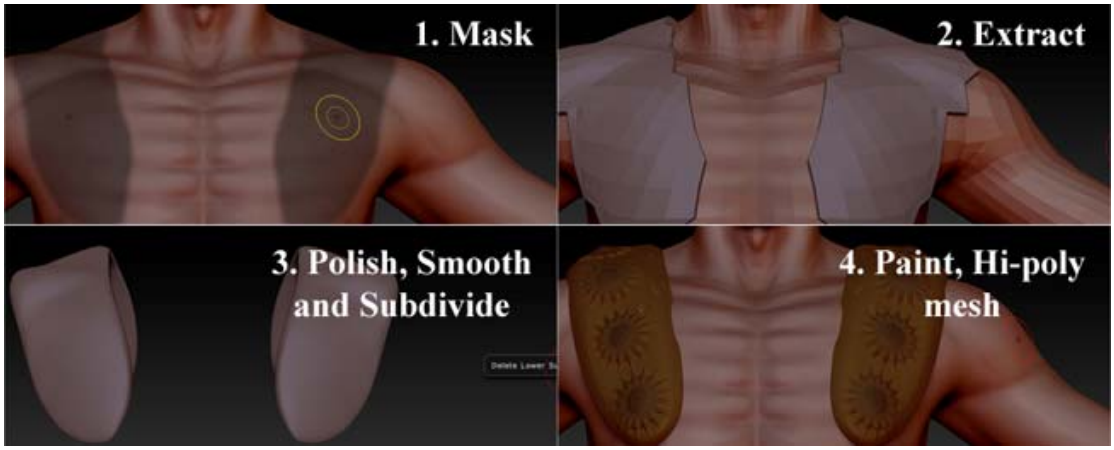
Η ίδια διαδικασία ακολουθείτε και για τα επόμενα layer, βρίσκοντας το κατάλληλο Blending Mode, πειράζοντας το Opacity και Fill του layer, την φωτεινότητα, την χροιά και την έκθεση τους στο φως. Εκτός από τα συγκεκριμένα layers μπορούμε να δημιουργήσουμε επιπλέον λεπτομέρειες και να εφαρμόσουμε διάφορα effects, ή ζωγραφίζοντας να δημιουργήσουμε παραπάνω λεπτομέρειες, αφού τώρα πλέον ξέρουμε το κάθε σημείο του χάρτη μας σε ποιο σημείο αντιστοιχεί στον 3D μοντέλο μας. Παρακάτω, στην εικόνα 10 δείχνουμε τα layers, αλλά και το τελικό αποτέλεσμα του Texture Map που δημιουργήσαμε.



Εικόνα 2. 10 Δημιουργία του τελικού Texture Map μέσω Photoshop

Πριν προχωρήσουμε στο επόμενο κεφάλαιο που περιλαμβάνει την δημιουργία animation με την χρήση του Blender, θα αφιερώσουμε λίγο χρόνο για να δείξουμε έναν γρήγορο και αποτελεσματικό τρόπο δημιουργίας subtools, χρησιμοποιώντας ως βάση το υπάρχον αντικείμενο μας. Τέτοια subtools είναι τα ρούχα και η πανοπλία, αφού θέλουμε να έχουν ένα παρόμοιο σχήμα με αυτό του χαρακτήρα μας. Μέσω της τεχνικής masking και extract που θα εφαρμόσουμε πάνω στο μοντέλο μας, είμαστε σε θέση να δημιουργήσουμε νέα meshes, στηριζόμενοι στο σχήμα της περιοχής που έχουμε κάνει mask.

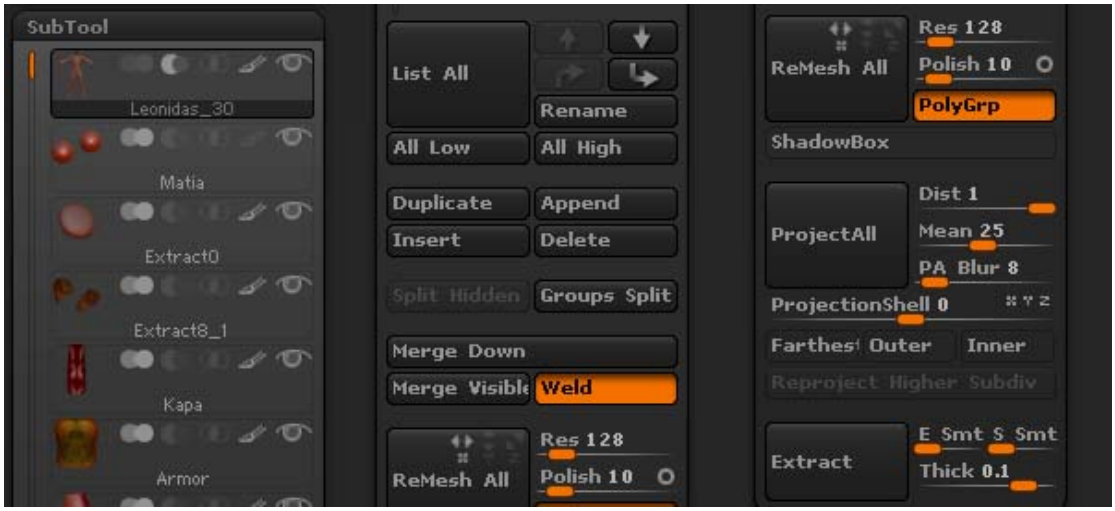
Αρχικά όπως φαίνεται και στην εικόνα 11, επιλέγουμε με το Masking Tool την επιθυμητή περιοχή κρατώντας πατημένο το Ctrl ή διορθώνοντας με τον συνδυασμό Ctrl + Alt, απλά σαν να ζωγραφίσουμε στην επιφάνεια του χαρακτήρα μας, επιλέγοντας το μέγεθος του πινέλου, τον τρόπο αλλά και το alpha της μύτης του. Πριν την διαδικασία του extract, μπορούμε να επιλέξουμε το smoothness των άκρων και της επιφάνειας, όπως επίσης και το πάχος του νέου subtool.



Εικόνα 2. 11 Δημιουργία νέου subtool, μέσω masking και extract

Όταν τελειώσουμε με όλα τα παραπάνω και είμαστε έτοιμοι πατάμε το κουμπί Extract που βρίσκεται στο Tool→SubTool, και αυτόματα δημιουργεί το νέο subtool, το οποίο μπορούμε να επεξεργαστούμε με παρόμοιο τρόπο, όπως αυτόν που δείξαμε παραπάνω. Σημαντικό σε αυτήν την φάση είναι να είμαστε σε ένα χαμηλό subdivision level όταν θα το δημιουργήσουμε, αφού το νέο subtool, έχει τον ίδιο αριθμό points με την επιλεγμένη περιοχή.

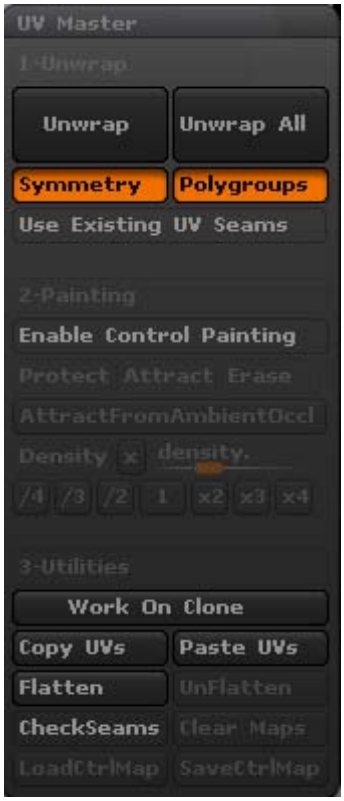
Δημιουργώντας subtools, ή προσθέτοντας μέσω των επιλογών insert και append, δημιουργούμε μια λίστα, την οποία μπορούμε να διαχειριστούμε πηγαίνοντας στο Tool→SubTool. Από εκεί μπορούμε να διαγράψουμε ένα subtool, να δημιουργήσουμε ένα αντίγραφο, να ενώσουμε ή να χωρίσουμε μερικά, να κάνουμε μετονομασία ή να τα μετακινήσουμε πάνω και κάτω στην λίστα. Ακόμα μπορούμε να μετακινούμαστε μεταξύ των subtools, επιτρέποντας μας να μπαίνουμε σε Edit Mode για το καθένα από αυτά, αλλά και να εμφανίζουμε ή να κρύβουμε από την οθόνη τα subtools της επιλογής μας.



Εικόνα 2. 12 Η λίστα των subtools που έχουμε δημιουργήσει, καθώς και όλα τα εργαλεία του SubTool panel του ZBrush.

Τα νέα subtools που δημιουργούμε, δεν περιέχουν uv map και έτσι πρέπει είτε να ορίσουμε εμείς κάνοντας export το μοντέλο στο 3ds Max, με την ίδια διαδικασία που ακολουθήσαμε στο παράδειγμα μας, είτε μέσω του ZBrush και του plug-in UV Master. Για να κάνουμε uvw unwrap μέσω του 3ds Max στο subtool που δημιουργήσαμε μπορούμε να ακολουθήσουμε δύο διαφορετικούς τρόπους. Μπορούμε να κάνουμε export στο πρώτο subdivision level το μοντέλο μας σε μορφή .obj και να το εισάγουμε στο 3ds Max, όπου θα δημιουργήσουμε το uvw χάρτη και αφού το κάνουμε export και πάλι import στο ZBrush, θα αντικαταστήσουμε το αρχικό subtool με αυτό που έχει uvw συντεταγμένες και θα συνεχίσουμε την επεξεργασία του. Ο δεύτερος τρόπος είναι παρόμοιος, όμως δεν μας ενδιαφέρει το subdivision level, αφού μπορούμε να κάνουμε uvw unwrap και να μέσω του plug-in UV Master να κάνουμε copy-paste τον χάρτη συντεταγμένων που στο αρχικό subtool. Ο μόνος περιορισμός είναι ότι πρέπει να γίνει paste στο ίδιο subdivision level με αυτό που δουλέψαμε στο 3ds Max.

Παρόλα αυτά το ZBrush μέσω του plug-in UV Master δίνει την δυνατότητα δημιουργίας uvw maps μέσω μιας αυτόματης διαδικασίας. Το UV Master δημιουργεί ένα αντίγραφο του πρώτου subdivision level του μοντέλου που δουλεύουμε, και πάνω σε αυτό επιλέγουμε σημεία που θέλουμε να προστατέψουμε και σημεία που θέλουμε να προσελκύσουν τα seams, και έτσι πατώντας Unwrap, δημιουργεί ένα χάρτη συντεταγμένων. Η διαδικασία αυτή δεν είναι κατάλληλη για μεγάλα και περίπλοκα αντικείμενα, με ανώμαλες επιφάνειες, αφού δίνει ένα αποτέλεσμα κατά προσέγγιση. Άλλοι τρόποι δημιουργίας seams στο ZBrush είναι μέσω των polygroups, ή την αυτόματη δημιουργία των seams βασισμένη στο ambient occlusion, δηλαδή στα σημεία που δεν είναι πολύ εμφανή.



Εικόνα 2. 13 Το plug-in εργαλείο UV Master και οι παράμετροι του

Έχοντας πλέον δημιουργήσει όλα τα απαραίτητα subtools, τα κάνουμε export στο μικρότερο επίπεδο ανάλυσης, δηλαδή στο subdivision level 1, δημιουργούμε όλα τα απαραίτητα maps και συνεχίζουμε στην επόμενη φάση και στην δημιουργία των animations.

Κεφάλαιο 3

Blender Game Engine

3.1 Πρόλογος

Το Blender είναι ένα open source λογισμικό που χρησιμοποιείται για την δημιουργία animated ταινιών, οπτικών effect, διαδραστικές 3D εφαρμογές και video παιχνιδιών. Μερικά από τα χαρακτηριστικά που προσφέρει είναι η δυνατότητα ανάπτυξης 3D μοντέλων, το UV Unwrapping και Texturing, διαδικασίες rigging και skinning, εξομοίωση particles, υγρών και καπνού και τέλος τις διαδικασίες animating, rendering, επεξεργασία βίντεο και compositing.

3.2 Ιστορική αναδρομή

Το Blender αρχικά δημιουργήθηκε σαν μια μικρή εφαρμογή γραφείου από την Ολλανδική εταιρία animation studio NeoGeo και τον συνιδρυτή της Ton Roosendaal. Γρήγορα η NeoGeo έγινε ένα από τα πιο διαδεδομένα animation studio στην Ολλανδία και το 1995 αποφασίστηκε να ξαναγραφτεί το εργαλείο από την αρχή. Αργότερα το 1998 ο Ton Roosendaal ίδρυσε μια νέα εταιρία με το όνομα Nan (Not a Number Technologies) για την περεταίρω ανάπτυξη και προώθηση του συγκεκριμένου εργαλείου, καθώς επίσης την παροχή υπηρεσιών και εμπορικών προϊόντων. Στόχος της εταιρίας ήταν η δημιουργία ενός ελεύθερου στην χρήση προγράμματος δημιουργίας 3D περιεχομένου, που όμως παρόλο την μεγάλη ανάπτυξη και τις εταιρίες που επένδυσαν σε αυτό μέχρι το 2000, δύο χρόνια αργότερα η εταιρία αναγκάστηκε να κλείσει λόγω οικονομικών προβλημάτων.

Το κλείσιμο της εταιρίας σταμάτησε την ανάπτυξη του Blender, όμως οι ενθουσιώδης κοινότητες χρηστών και πελατών δεν σταμάτησαν ποτέ να ασχολούνται με το συγκεκριμένο εργαλείο και έτσι ο Ton Roosendaal αποφάσισε να δημιουργήσει το μη κερδοσκοπικό ίδρυμα Blender, όπου έβαλε ως στόχο την ανάπτυξη και προώθηση του εργαλείου ως ένα ελεύθερο λογισμικό ανοικτού κώδικα βασισμένο στην κοινότητα χρηστών. Τον Οκτώβριο του 2002 το Blender παραδόθηκε στο κοινό κάτω από τους όρους του GNU General Public License και έκτοτε αναπτύσσεται με την εθελοντική εργασία προγραμματιστών ανά τον κόσμο.

3.3 Το εργαλείο σήμερα

Το Blender τρέχει στα λειτουργικά συστήματα Linux, Mac OS X, Microsoft Windows και FreeBSD και αν και έχει ένα σχετικά μικρό μέγεθος εγκατάστασης, έχει χαρακτηριστικά και παρέχει λύσεις ενός high-end 3D λογισμικού. Υποστηρίζει ένα μεγάλο αριθμό primitive meshes, polygon meshes, fast subdivision surface modeling, Bezier curves, NURBS surfaces, meatballs, digital sculpting, outline font, και το νέο n-gon σύστημα μοντελοποίησης που ονομάζεται B-mesh. Ακόμα υποστηρίζει το YafaRay , ελεύθερο πρόγραμμα ray tracer, εσωτερική render engine με ray tracing, indirect lighting και ambient occlusion, ένα πλήρες ολοκληρωμένο node based compositor που υποστηρίζεται στην διαδικασία rendering, modifiers που βοηθούν στην διαδικασία μοντελοποίησης και βασικές τεχνικές επεξεργασίας ήχου και βίντεο. Στον τομέα animation προσφέρει keyframed animation εργαλεία, inverse kinematics, ένα εργαλείο δημιουργίας σκελετικού συστήματος (armature), curve και lattice based deformations, shape keys, vertex weighting, soft bodies, fluid και bullet rigid body dynamics και mesh collision detection.

Το Blender λειτουργεί ακόμα και σαν game engine, αφού προσφέρει collision detection, dynamics engine, όπως particle συστημάτων και εξομοίωση καπνού και νερού και τέλος παρέχει την δυνατότητα προγραμματισμού σε Python. Μέσω Python scripts εκτός από τον προγραμματισμό του περιβάλλοντος του παιχνιδιού, μπορούμε να δημιουργήσουμε εξειδικευμένα εργαλεία, να αυτοματοποιήσουμε διαδικασίες και να δημιουργήσουμε import/export plug-ins για διάφορες μορφές αρχείων.

3.4 Τα σημεία χρήσης του στην ανάπτυξη της πτυχιακής

Στην συγκεκριμένη πτυχιακή χρησιμοποιήθηκε το Blender στην έκδοση 2.49b, κυρίως για την δημιουργία των σκελετικών συστημάτων (armature) των μοντέλων και για την δημιουργία των animations. Αν και αποτελεί μια ολοκληρωμένη λύση για την δημιουργία ενός παιχνιδιού, από την διαδικασία της μοντελοποίησης, έως και τον προγραμματισμό, εδώ χρησιμοποιήθηκε σε έναν συγκεκριμένο τομέα, αυτόν του animation, αφού δίνει μεγάλη ελευθερία και ευκολία στον χρήστη. Παρακάτω θα δείξουμε μέσω ενός παραδείγματος, πως χρησιμοποιήσαμε το Blender.

3.5 Παράδειγμα δημιουργίας του armature

Συνεχίζοντας το παράδειγμα που παρουσιάσαμε σε προηγούμενα κεφάλαια, πρώτα μέσω του 3ds Max και μετά μέσω του ZBrush, θα δείξουμε την διαδικασία δημιουργίας των animations, καθώς και τις τεχνικές που ακολουθήσαμε.

Πριν ξεκινήσουμε όμως να σχηματίζουμε το σκελετό για τον χαρακτήρα μας, ας μιλήσουμε λίγο για την διαδικασία rigging. Το rigging μπορούμε να το περιγράψουμε ως την

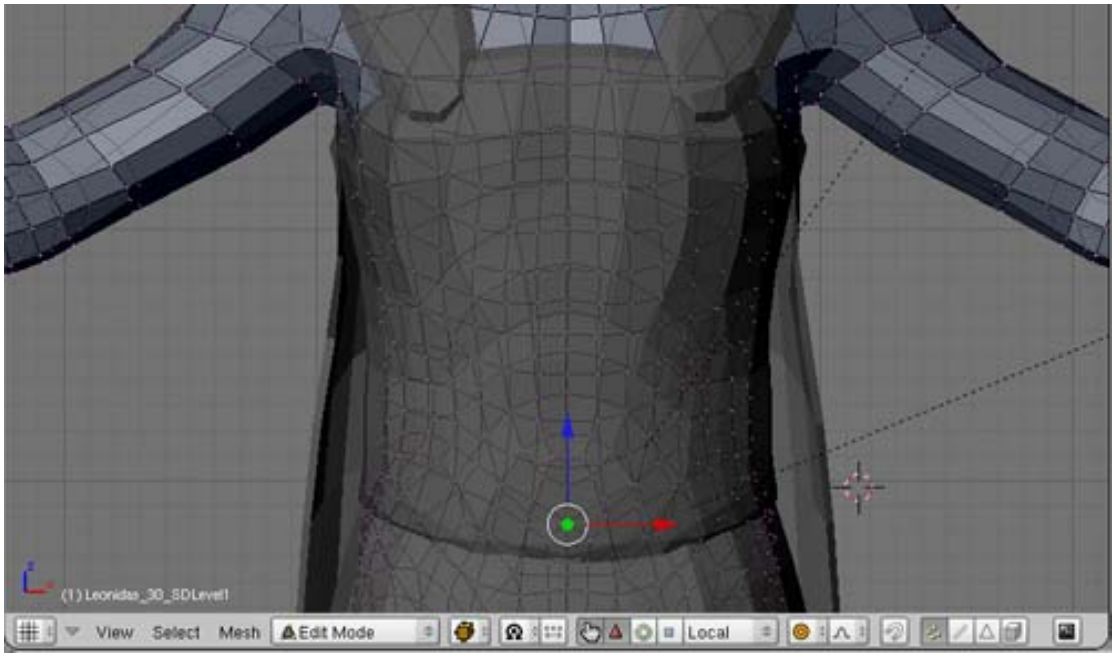
διαδικασία μέσω της οποίας δημιουργούμε απλά “controller” αντικείμενα, μέσω τον οποίων μπορούμε να επηρεάσουμε άλλα πιο περίπλοκα αντικείμενα. Στο Blender οι “controller” αυτοί καλούνται armatures, και μπορούν να ελέγχουν το σχήμα (geometry) των meshes και όλων των ειδών των επιφανειών, όπως επίσης και τις ιδιότητες των αντικειμένων, όπως την θέση, το μέγεθος και την περιστροφή. Τα armatures λοιπόν μιμούνται έναν σκελετό, τόσο στην δομή, όσο και στην συμπεριφορά, γι’ αυτό λέμε ότι αποτελούνται από bones.

Σε κάθε bone του armature μπορούμε να συνδέσουμε ένα ή περισσότερα αντικείμενα, ή μόνο ένα μέρος της επιφάνειας ενός αντικειμένου, μέσω μιας διαδικασίας ορισμού σχέσης γονέα-παιδιού μεταξύ armature και αντικειμένου. Μέσω αυτής της σχέσης μπορούμε μετακινώντας τα bones, να αλλάζουμε τις ιδιότητες των συνδεδεμένων αντικειμένων. Ακόμα το κάθε armature έχει επιπλέον εκτός από το Object Mode και Edit Mode, το Pose Mode, μέσω του οποίου γίνονται όλα τα παραπάνω δυνατά. Ας δείξουμε τώρα τα βασικά βήματα δημιουργίας ενός armature και τον τρόπο σύνδεσης του χαρακτήρα μας σε αυτό.

Αρχικά κάνουμε import στο Blender τον χαρακτήρα μας και όλα τα επιμέρους subtools που δημιουργήσαμε στο ZBrush, έτσι ώστε να έχουμε στην οθόνη μας όλα τα αντικείμενα στα οποία θα βασιστούμε για το σχήμα του armature που θα σχεδιάσουμε. Αφού τελειώσουμε την εισαγωγή όλων των αρχείων μας, είμαστε σε θέση να ξεκινήσουμε να δημιουργούμε το σκελετό και όλα τα επιμέρους bones ελέγχου.

Για την δημιουργίας armature πηγαίνουμε στο Add του βασικού μενού και επιλέγουμε Armature, ή μέσω των συντομεύσεων Shift+A και Space, όπου επιλέγουμε Add→Armature στο μενού που εμφανίζεται στο σημείο που έχουμε τον κέρσορα του ποντικιού. Το armature, όπως και το κάθε αντικείμενο που εισάγουμε μέσα στο σκηνή μας πατώντας Add, δημιουργείται στην θέση που είναι ο κέρσορας. Μπορούμε να μεταφέρουμε τον κέρσορα στην θέση 0,0,0 πατώντας Shift+C, ή να επιλέξουμε εμείς μια άλλη πατώντας αριστερό κλικ οπουδήποτε στην οθόνη. Όμως εμείς χρειαζόμαστε η βάση του armature να ταυτίζεται απόλυτα με ένα σημείο του κέντρου του χαρακτήρα μας, έτσι ώστε να έχουμε αριστερά και δεξιά ομοιόμορφα bones.

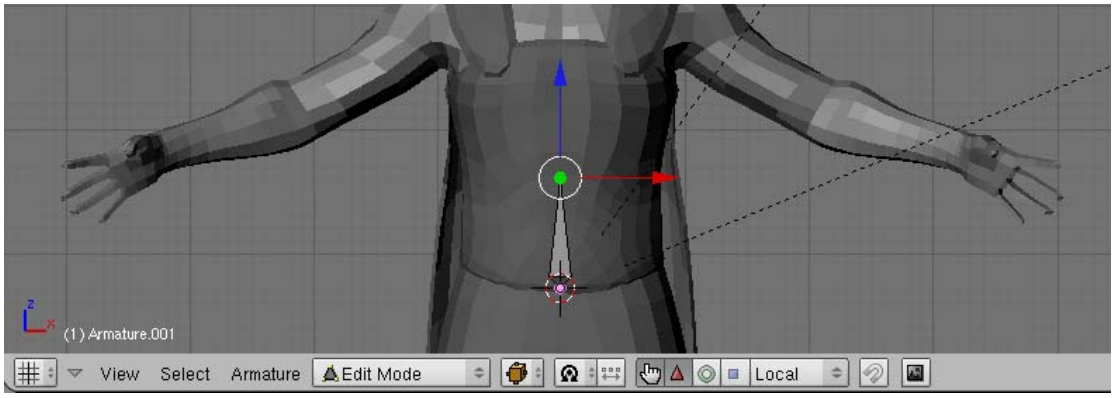
Για να μεταφέρουμε τον κέρσορα στο μέσο του χαρακτήρα μας και στην επιθυμητή θέση, ακολουθούμε τα παρακάτω βήματα. Επιλέγουμε το mesh του χαρακτήρα μας και μπαίνουμε σε Edit Mode πατώντας Tab ή αλλάζοντας το Object Mode από την μπάρα εργαλείων της σκηνής. Στην συνέχεια αλλάζουμε το view από perspective σε front και αφού επιλέξουμε Vertex Select Mode και εμφανιστούν όλα τα vertices του μοντέλου μας, επιλέγουμε το μεσαίο vertex που βρίσκεται στο ύψος της λεκάνης του, πατώντας αριστερό κλικ πάνω του, όπως φαίνεται στην εικόνα 1.



Εικόνα 3. 1 Το μοντέλο μας όπως φαίνεται στο Edit Mode, με επιλεγμένο το Vertex Selection Mode

Έχοντας επιλεγμένο το επιθυμητό vertex, πατάμε Shift+S και επιλέγουμε το Cursor -> Selection, το οποίο μεταφέρει τον κέρσορα στο επιλεγμένο σημείο. Πλέον έχουμε ορίσει το σημείο στο οποίο θα δημιουργήσουμε την βάση του armature και βγαίνουμε από το Edit Mode πατώντας Tab.

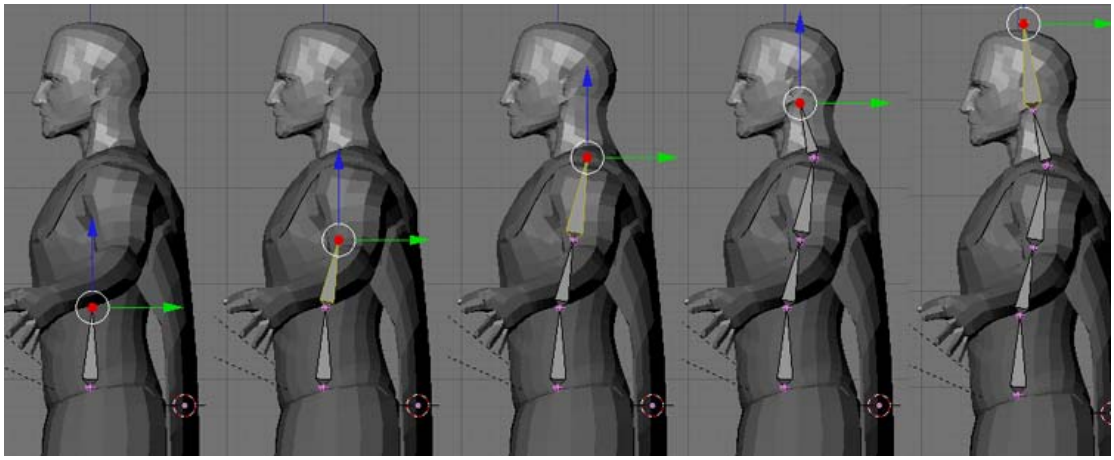
Εφόσον είναι όλα έτοιμα πατάμε Add→Armature και εμφανίζεται το πρώτο μας bone. Αμέσως αλλάζουμε σε Edit Mode, επιλέγουμε την κορυφή του bone και το κάνουμε scale down στον Z άξονα, όπως φαίνεται στην εικόνα 2. Επίσης για να είναι εμφανές τα bones μέσα από τα meshes, ενεργοποιούμε από τα Editing Options του Armature, το κουμπί X-Ray.



Εικόνα 3. 2 Το armature σε Edit Mode, με ενεργοποιημένο το κουμπί X-Ray

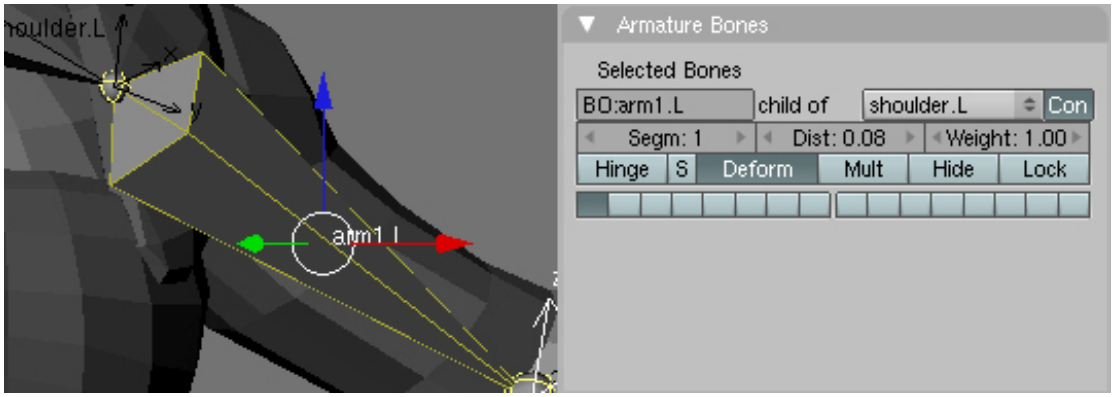
Στην συνέχεια πατάμε το κουμπί 3 για να αλλάξουμε σε left view και συνεχίζουμε να δημιουργούμε bones, πατώντας το πλήκτρο "E" και κάνοντας drag στην οθόνη, όπως φαίνεται στην εικόνα 3. Σε περίπτωση που θέλουμε το bone που θα δημιουργήσουμε να είναι

ευθυγραμμισμένο με κάποιον άξονα, πατάμε “E” για να το δημιουργήσουμε και ένα από τα x,y,z για να επιλέξουμε τον συγκεκριμένο άξονα.

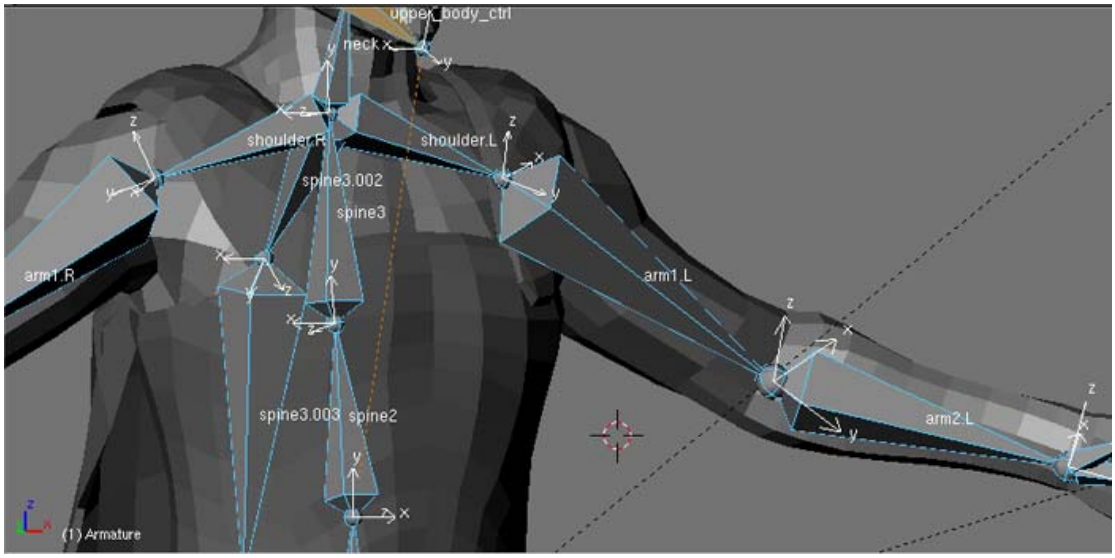


Εικόνα 3. 3 Στιγμιότυπα από την δημιουργία του σκελετικού συστήματος

Για να δημιουργήσουμε bones συμμετρικά ως προς τον άξονα x, δηλαδή ταυτόχρονα δεξιά και αριστερά, πατάμε “1” έτσι ώστε να γυρίσουμε στο front view και ενεργοποιούμε από τα Editing Options το X-Axis Mirror. Στην συνέχεια δημιουργούμε bones, όπως και πριν αλλά πατώντας Shift+E. Αφού προσθέσουμε όλα τα απαραίτητα bones, κάνουμε όλες τις απαραίτητες μικρές διορθώσεις και ελέγχουμε αν είναι όλα στην σωστή θέση τους γυρνώντας την κάμερα και αλλάζοντας view. Πριν βγούμε από το Edit Mode, κάνουμε μετονομασία τα bones με ονόματα σχετικά με τα σημεία που ελέγχουν στον χαρακτήρα μας, έτσι ώστε αργότερα να διευκολυνθούμε. Επίσης η μετονομασία των συμμετρικών bones, είναι σημαντικό να γίνει επιλέγοντας ονόματα όπως bone.R και bone.L, ή Right.bone και Left.bone, αφού έτσι το Blender αναγνωρίζει τι είναι δεξιά και τι αριστερά, δίνοντας μας την δυνατότητα να αντιγράφουμε πόζες συμμετρικά, μια τεχνική που θα μας βοηθήσει πολύ στην διαδικασία του animation. Στις δύο εικόνες παρακάτω φαίνονται το panel Selected Bones, όπου γίνεται η μετονομασία των bones και η αλλαγή του parent τους, όπως επίσης και ένα στιγμιότυπο με τα ονόματα και τους άξονες των bones που δημιουργήσαμε.



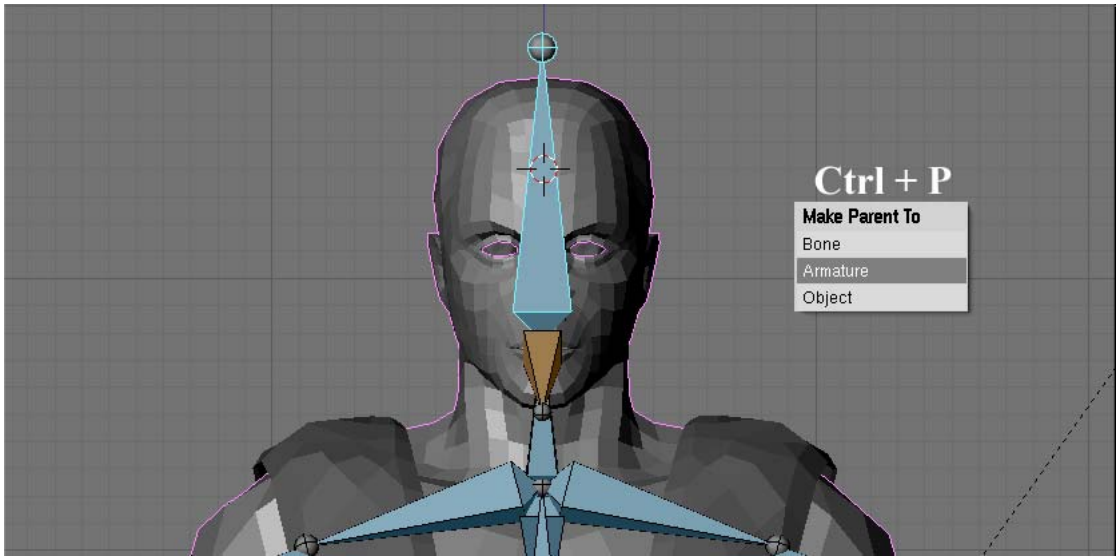
Εικόνα 3. 4 Το panel Selected Bones, από όπου έγινε η μετονομασία, όπως εμφανίζεται στο Edit Mode



Εικόνα 3. 5 Στιγμιότυπο από το τελικό αποτέλεσμα του armature, με τα ονόματα των bones, καθώς και οι άξονες τους

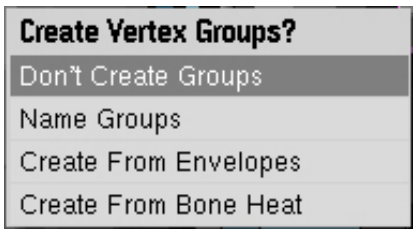
Έχοντας σχεδιάσει όλα τα bones του χαρακτήρα, προχωράμε στο επόμενο βήμα του Rigging, που είναι η διαδικασία Skinning. Με το Skinning περιγράφεται η διαδικασία σύνδεσης του armature, με άλλα αντικείμενα, μέσω της οποίας το armature επηρεάζει τις ιδιότητες ή το σχήμα αυτών. Στο Blender υπάρχουν δύο βασικοί τρόποι skinning που μπορούν να επιτευχθούν, πρώτον μέσω της σύνδεσης των αντικειμένων με parent και constraint στα bones και δεύτερον με την εφαρμογή ολόκληρου του armature modifier σε ένα mesh. Στην πρώτη περίπτωση όταν μετακινούμε ή περιστρέφουμε τα bones στο Pose Mode, το ίδιο συμβαίνει και στα παιδιά τους, δηλαδή στα αντικείμενα που τους έχουμε κάνει link. Επίσης σε αυτήν την περίπτωση τα αντικείμενα (children) δεν γίνονται deform, δηλαδή δεν δέχονται καμία αλλαγή στην επιφάνεια τους. Αντιθέτως στην δεύτερη περίπτωση όπου εφαρμόζουμε το armature modifier σε ένα αντικείμενο, μπορούμε να κάνουμε deform, αφού τα bones του armature ελέγχουν διαφορετικά σημεία του mesh. Αυτή η διαδικασία είναι μια περίπλοκη και ισχυρή μέθοδος, που στην ουσία αποτελεί τον μόνο πραγματικό τρόπο ελέγχου του σχήματος ενός mesh, δηλαδή τον έλεγχο της σχετικής θέσης των vertices του mesh μέσα στο 3D περιβάλλον.

Αφού περιγράψαμε το skinning, και πώς μπορεί να επιτευχθεί, θα δούμε τώρα μέσα από το Blender, πώς γίνεται στην πράξη, συνεχίζοντας το παράδειγμά μας. Αρχικά επιλέγουμε το armature και ενεργοποιούμε το Pose Mode ώστε να μπορούμε να επιλέξουμε τα bones του. Στην συνέχεια επιλέγουμε τον χαρακτήρα μας, βεβαιωνόμαστε ότι είναι σε Object Mode και έχοντας πατημένο το πλήκτρο Shift, επιλέγουμε ένα από τα bones του armature. Είναι σημαντικό να γίνει με αυτήν την σειρά, καθώς πρώτα επιλέγουμε το child και μετά το parent του. Αφού επιλέξουμε και τα δύο πατάμε Ctrl+P, το οποίο θα εμφανίσει στην οθόνη μας το μενού "Make Parent To" και τρεις επιλογές, τις Bone, Armature και Object. Εμείς θέλουμε να εφαρμόσουμε το armature που σχεδιάσαμε πάνω στο σώμα του χαρακτήρα μας και έτσι επιλέγουμε το Make Parent To→Armature.



Εικόνα 3. 6 Δημιουργία σχέσης γονέα-παιδιού μεταξύ armature και mesh, πατώντας Ctrl + P

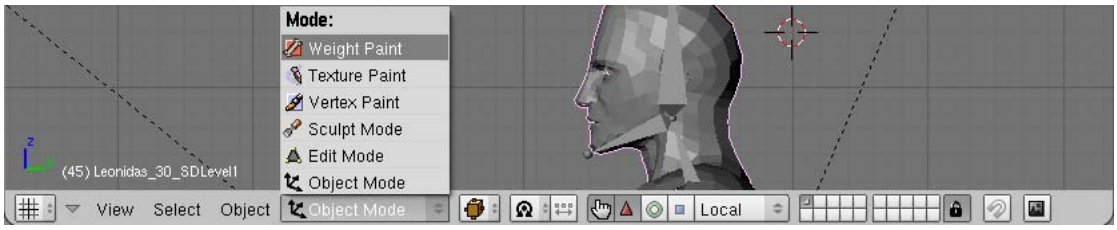
Επιλέγοντας το Armature για parent, εμφανίζεται ένα ακόμα μενού με τίτλο "Create Vertex Groups?" από το οποίο καλούμαστε να επιλέξουμε μια από τις τέσσερις τεχνικές δημιουργίας "vertex groups" της διαδικασίας skinning, όπως φαίνεται και στην εικόνα 7.



Εικόνα 3. 7 Το μενού της δημιουργίας Vertex Groups, της διαδικασίας skinning με parent το armature

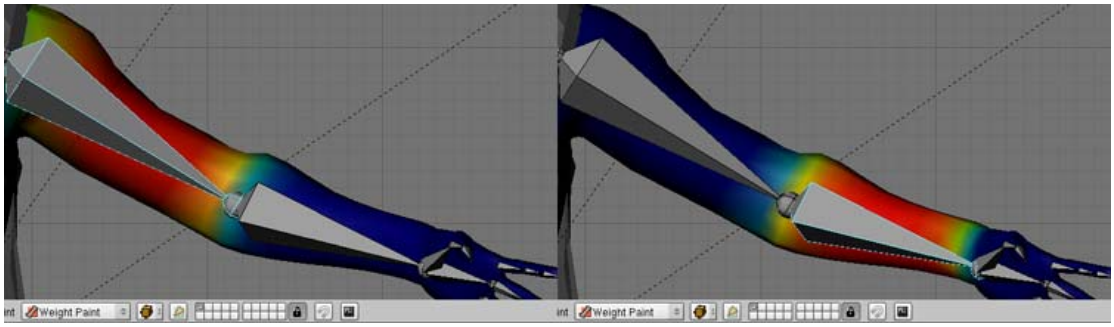
Με την επιλογή “Don’t Create Groups”, δεν δημιουργεί groups και κατά συνέπεια τα bones δεν θα επηρεάζουν καθόλου την επιφάνεια του mesh. Με το “Name Groups” δημιουργούνται, αν δεν υπάρχουν ήδη, άδεια vertex groups για το κάθε bone του skinned armature. Αυτή τον τρόπο τον επιλέγουμε αν έχουμε δημιουργήσει ήδη vertex groups και έχουμε ορίσει τα vertices που θα ελέγχουν μέσω του weight paint, για ολόκληρο το mesh μας. Ακόμα στο μενού βλέπουμε σαν τρίτη επιλογή το “Create From Envelopes”, που όπως και στο “Name Groups”, θα δημιουργήσει vertex groups για κάθε bone, όμως θα κάνει αυτόματα weight αναλόγως την περιοχή επιρροής του κάθε bone. Με αυτόν τον τρόπο ότι vertex groups έχουμε δημιουργήσει και κάνει weight αντικαθίστανται. Σαν τελευταία επιλογή βλέπουμε το “Create From Bone Heat”, που όπως και στην παραπάνω περίπτωση σχηματίζει αυτόματα vertex groups, όμως εδώ η διαδικασία επιλογής των vertices και του weight paint γίνονται μέσα από τον “bone heat” αλγόριθμο.

Στο παράδειγμα μας, επιλέγουμε να δημιουργήσουμε vertex groups μέσω του αλγορίθμου Bone Heat, καθώς δίνει αρκετά καλά αποτελέσματα, που στην συνέχεια μπορούμε να αλλάξουμε/διορθώσουμε χειροκίνητα. Έχοντας δηλώσει ως parent του χαρακτήρα μας το armature, βλέπουμε ότι μετακινώντας ή περιστρέφοντας τα bones, οι ίδιες αλλαγές γίνονται και στον χαρακτήρα μας. Επίσης μετακινώντας bones του armature μπορούμε να ελέγξουμε αν έχει γίνει σωστό weight paint σε αυτά, κάτι που στις περισσότερες περιπτώσεις θέλει μικρές διορθώσεις. Για να κάνουμε διορθώσεις ή για να επιλέξουμε εμείς τα vertices επιρροής ενός bone, επιλέγουμε το mesh και στην συνέχεια αλλάζουμε από Object Mode σε Weight Paint Mode όπως φαίνεται στην εικόνα παρακάτω.



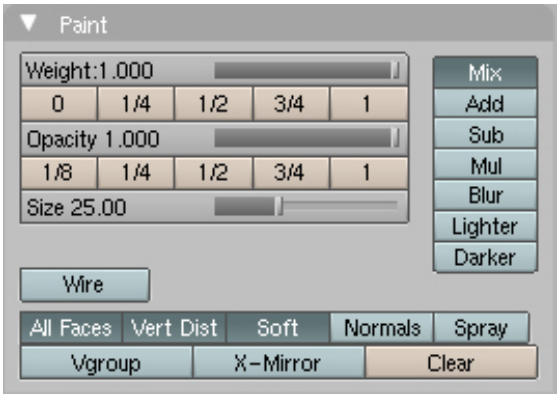
Εικόνα 3. 8 Επιλογή του Weight Paint Mode

Σε αυτό το Mode επιλέγοντας το κάθε bone μπορούμε να δούμε την περιοχή επιρροής του, αλλά και την δύναμη με την οποία ελέγχει το mesh. Το βάρος του bone στα διάφορα σημεία επιρροής του vertex group, φαίνεται μέσω του χρώματος. Το βάρος κυμαίνετε μεταξύ 0 και 1, με το 0 να μην επηρεάζει καθόλου την περιοχή και το 1 να ελέγχει 100% τα συγκεκριμένα vertices. Σε πολλά σημεία δύο ή και παραπάνω bones μπορεί να ελέγχουν τα ίδια vertices, με διαφορετικά βάρη. Τέτοια σημεία είναι αυτά όπου ενώνονται τα bones και στα οποία τα βάρη δημιουργούνται και μοιράζονται αυτόματα από τον αλγόριθμο “bone heat”.



Εικόνα 3. 9 Τα βάρη και η περιοχές επιρροής των bones, που δημιουργήθηκαν μέσω του "bone heat" αλγορίθμου στην διαδικασία skinning

Επίσης στο Weight Paint Mode βλέπουμε ότι στο Buttons Window, εμφανίζεται το Paint panel όπου βρίσκονται όλα τα απαραίτητα εργαλεία με τα οποία μπορούμε να χρωματίσουμε το μοντέλο μας και να δημιουργήσουμε ή αλλάξουμε τα βάρη των bones. Στο panel αυτό μπορούμε να επιλέξουμε την ένταση και το μέγεθος της επιφάνειας που θα βάψουμε, τον τρόπο, αλλά και διάφορες άλλες επιλογές, όπως φαίνονται στην εικόνα 10.



Εικόνα 3. 10 Το Paint panel του Buttons Window που εμφανίζεται στο Weight Paint Mode

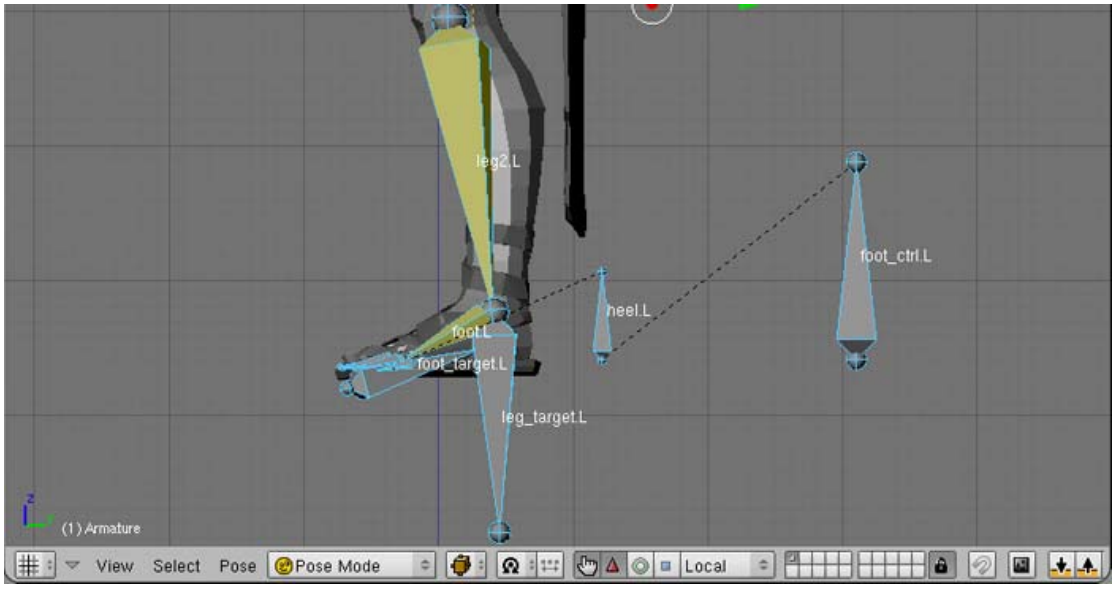
Τελειώνοντας με τον χαρακτήρα μας, επιλέγουμε και τα υπόλοιπα αντικείμενα μας και τα κάνουμε parent στο armature, επιλέγοντας Parent To→Armature, για αυτά που θέλουμε να γίνονται deform από τα bones και Parent To→Bone για αυτά που θέλουμε να μετακινούνται με τον ίδιο τρόπο όπως τα bones, αλλά να μην επιδέχεται καμία αλλαγή στο σχήμα τους.

Αφού ελέγξουμε, μετακινώντας όλα τα bones, ότι είναι σωστά δηλωμένα τα vertex groups όλων των αντικειμένων, και κάνουμε τις απαραίτητες αλλαγές μέσα από το Weight Paint Mode, είμαστε σε θέση να δημιουργήσουμε τα animations μας. Πρώτα όμως θα ασχοληθούμε λίγο με την δημιουργία κάποιων επιπλέων bones, μακριά από το mesh έτσι ώστε να είναι εύκολα προσβάσιμα και τα οποία δεν θα αποτελούν μέρος του σκελετού του χαρακτήρα μας, αλλά ένα είδος controller μιας ολόκληρης περιοχής του. Με την δημιουργία των control bones η διαδικασία του animation γίνεται ακόμα πιο εύκολη, αφού μέσω inverse kinematics, ελέγχουμε ολόκληρες ομάδες από bones, και όχι το καθένα ξεχωριστά. Η δημιουργία αυτών των bones γίνεται επίτηδες σε αυτό το σημείο, αφού έχουμε ήδη κάνει

parent το armature στο mesh και έτσι όλα τα νέα bones που σχεδιάζουμε δεν επηρεάζουν άμεσα το χαρακτήρα μας, κάτι που σε αντίθετη περίπτωση θα έπρεπε να διορθώσουμε εντοπίζοντας και σβήνοντας τα βάρη στα vertex groups που θα είχαν δημιουργηθεί.

Ένα από τα πιο κρίσιμα σημεία της σχεδίασης του armature είναι αυτό της δημιουργίας των control bones, στα πόδια. Αυτό συμβαίνει γιατί μετακινώντας την βάση του armature, δηλαδή του parent όλων των bones, θέλουμε τα πόδια του χαρακτήρα να παραμένουν σταθερά στο ίδιο σημείο έτσι ώστε να αποφύγουμε την δημιουργία ενός είδους κύλισης. Για τον λόγω αυτό δημιουργούμε τέσσερα control bones στο κάθε πόδι και τα συνδέουμε κατάλληλα μέσω της τεχνικής parent αλλά και μέσω constraint με άλλα.

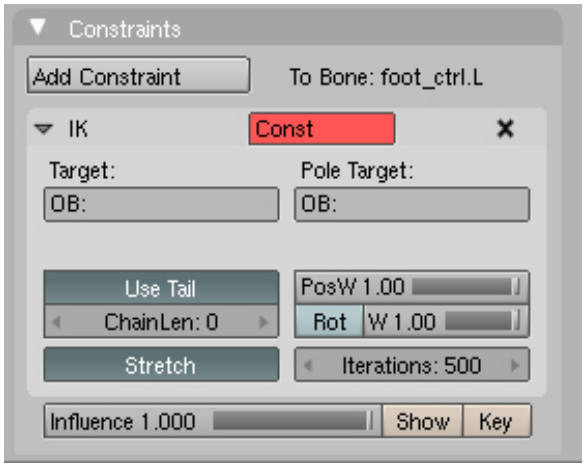
Αρχικά δημιουργούμε τα bones, όπως φαίνεται στην εικόνα 12, με τον ίδιο τρόπο που δείξαμε παραπάνω και στην συνέχεια τα μετακινούμε μακριά από τον σκελετό, απενεργοποιώντας το κουμπί Con του parent τους που βρίσκεται στο Selected Bones panel



Εικόνα 3. 11 Τα control bones του δεξιού ποδιού του armature

Αφού τα ονομάσαμε κατάλληλα ώστε να τα ξεχωρίζουμε, δηλώσαμε τους parent τους με τον εξής τρόπο. Στο foot_target δηλώσαμε ως parent το heel, στο Leg_target το foot_target, στο heel το foot_ctrl και τέλος στο foot_ctrl, που αποτελεί ουσιαστικά τον parent και των τριών, δεν δηλώσαμε κανένα γιατί δεν θέλουμε να εξαρτάται από κανένα άλλο. Επίσης δημιουργήσαμε μέσω inverse kinematics και του IK Solver constraint μια σύνδεση μεταξύ αυτών των bones και του leg2 και foot. Με το IK Solver δημιουργούμε ένα είδος σύνδεσης μεταξύ του επιλεγμένου bone που προσθέτουμε το συγκεκριμένο constraint και της βάσης του armature ή με ένα επιλεγμένο bone που το δηλώνουμε ως target.

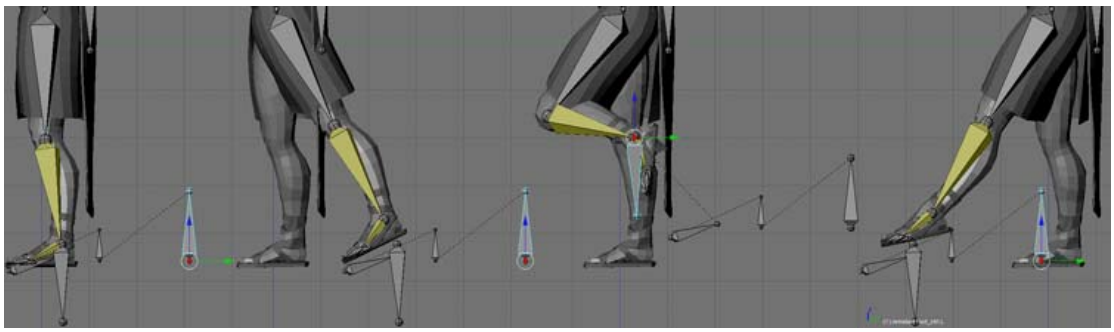
Στο παράδειγμά μας, εφόσον είμαστε σε Pose Mode, επιλέγουμε το bone με όνομα leg2, της δεξιάς και αριστερής πλευράς ξεχωριστά, πατάμε Add Constrain στο Constrain panel του και επιλέγουμε IK Solver. Αμέσως παρατηρούμε στην σκηνή μας, ότι το bone αλλάζει χρώμα, σημάδι ότι στο συγκεκριμένο bone έχουμε κάνει add ένα constraint, και επίσης εμφανίζεται στο Constrain panel ο IK Solver, όπως φαίνεται παρακάτω.



Εικόνα 3. 12 Το panel του IK Solver Constraint

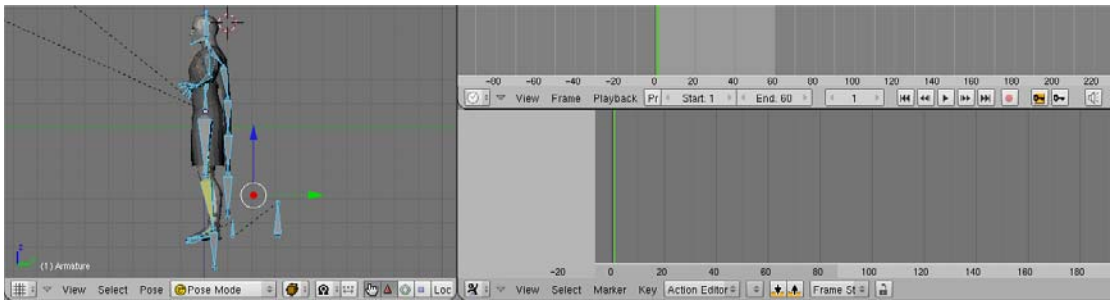
Στο IK Constraint panel δηλώνουμε στο Target OB το όνομα του armature, που στην περίπτωσή μας έχουμε κρατήσει το default όνομα Armature, στο Subtarget Bone BO το leg_target bone και στο ChainLen την τιμή 2. Το ίδιο κάνουμε και για το bone foot της δεξιάς και αριστερής πλευράς, με Target OB το armature, Subtarget Bone BO το foot_target και ChainLen την τιμή 1.

Έχοντας δημιουργήσει σχέσεις parent και constraint στα bones κάνοντας όλα τα παραπάνω, ελέγχοντας το foot_ctrl bone της κάθε πλευράς, είμαστε σε θέση να κινούμε το πόδι με φυσικό τρόπο, ενώ επιλέγοντας και κάνοντας περιστροφή ή μετακίνηση οποιουδήποτε άλλου control bone θα μπορούμε να ελέγχουμε μικρότερες ομάδες από τα bones του ποδιού. Στην παρακάτω εικόνα φαίνεται η κίνηση του ποδιού μέσα από την περιστροφή και μετακίνηση αυτών των bones.



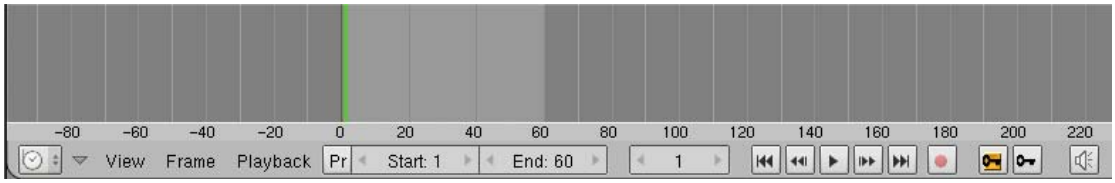
Εικόνα 3. 13 Κίνηση ολόκληρου του αριστερού ποδιού χρησιμοποιώντας τα control bones

Πλέον έχουμε τελειώσει με το armature, και είμαστε σε θέση να ξεκινήσουμε να δημιουργούμε τα animation μας. Για να το κάνουμε αυτό, χωρίζουμε το παράθυρο της σκηνής μας στα δύο, πατώντας στα όρια του view δεξί κλικ ή την ροδέλα και επιλέγουμε Split Area. Κάνουμε το ίδιο στην περιοχή που δημιουργήσαμε και το χωρίζουμε πάλι στα δύο, έτσι ώστε να έχουμε στην μισή οθόνη το 3D View και στην άλλη μισή στο πάνω μέρος το Timeline και στο κάτω τον Action Editor, όπως φαίνεται στην εικόνα 15. Για να αλλάξουμε τον τύπο του παραθύρου, πατάμε στο κουμπί Window type στο κάτω αριστερό μέρος της κάθε περιοχής.



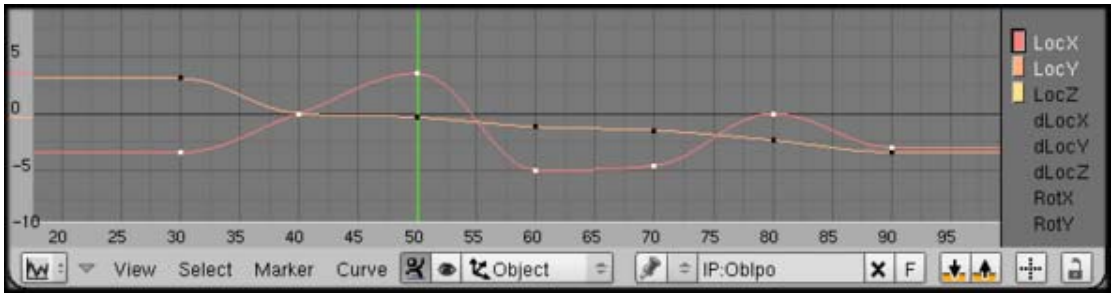
Εικόνα 3. 14 Τα παράθυρα 3D View, Timeline και Action Editor του Blender

Στο παράθυρο Timeline μπορούμε να δούμε σε ποιο frame ή δευτερόλεπτο βρισκόμαστε, ποιο είναι το αρχικό και τελικό frame του animation και να δούμε που είναι τα keyframes του επιλεγμένου αντικειμένου. Επίσης παρέχει ένα VTR-like control σύστημα, με το οποίο μπορούμε να το frame ου βρισκόμαστε αλλά και το frame range, να μετακινηθούμε στο επόμενο ή προηγούμενο frame, να "τρέξουμε" το animation, να δημιουργήσουμε ή να σβήσουμε keyframes και να συγχρονίσουμε έναν ήχο με το animation.

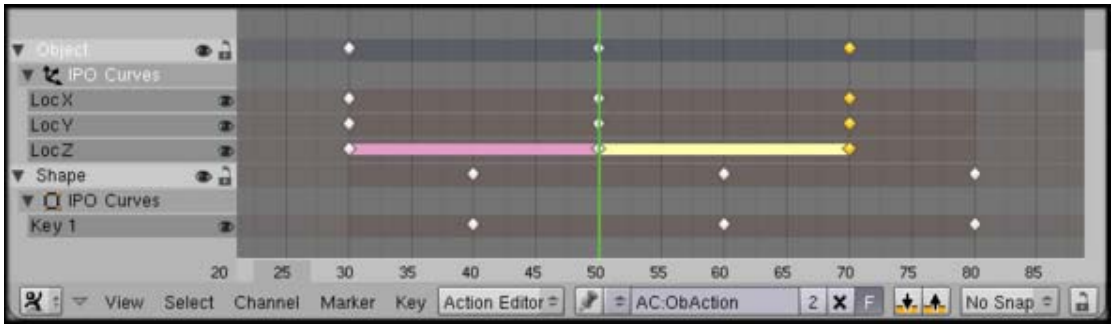


Εικόνα 3. 15 Το παράθυρο Timeline με την γραμμή εργαλείων του

Το animation είναι ένα αρκετά μεγάλο και σημαντικό κομμάτι της πτυχιακής, καθώς χρειάζεται αρκετό χρόνο και λεπτομερείς δουλειά. Στο Blender υπάρχουν τρεις διαφορετικοί Animation Editors, μέσα από τους οποίους μπορούμε να πετύχουμε τέλεια αποτελέσματα. Αυτοί οι Animation Editors είναι, ο Ipo Curve editor, ο Action editor και ο NLA editor, οι οποίοι έχουν δημιουργηθεί έτσι ώστε ο ένας να συνεχίζει την δουλειά του άλλου, αφού τα Ipo curves χρησιμοποιούνται από τα actions, τα οποία στην συνέχεια χρησιμοποιούνται από το NLA (non linear animation). Ο Ipo curve αποτελεί το χαμηλότερο επίπεδο editor, μέσα από το οποίο μπορούμε να ελέγξουμε μέσω καμπύλων της ρυθμίσεις και ιδιότητες του animation, ενώ ο Action editor, αν και είναι παρόμοιος με τον Ipo editor, δίνει μια πιο γενική εικόνα του animation και επιτρέπει την επεξεργασία πολλών bones και αντικειμένων ταυτοχρόνως. Τέλος ο NLA αποτελεί τον υψηλότερο επιπέδου animation editor του Blender, ο οποίος έχει βασιστεί στην ιδέα των non-linear video editors και επιτρέπει την επεξεργασία πολλών actions ταυτόχρονα, τα οποία παρουσιάζονται σαν στοιχεία και μπορούν να μετακινηθούν, να δημιουργηθούν αντίγραφα και να μεγαλώσουν ή μικρύνουν αυξάνοντας και μειώνοντας αντίστοιχα την ταχύτητα των animation clips (actions)



Εικόνα 3. 16 Ο Ipo Curve Editor. Χρησιμοποιείται για να ορίσουμε και επεξεργαστούμε interpolation curves



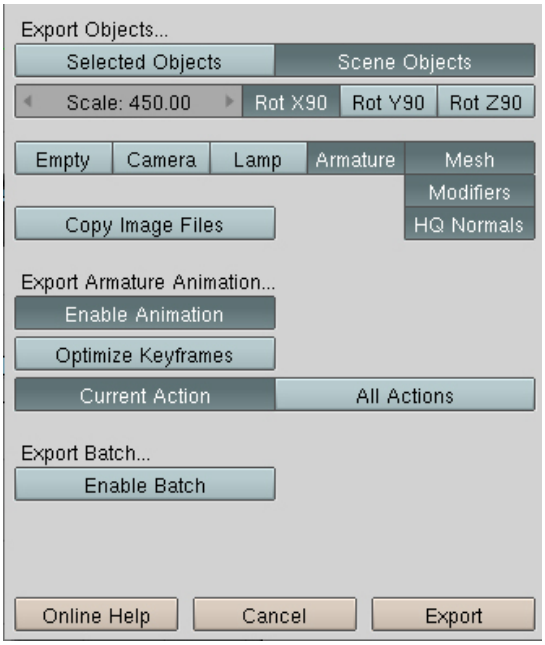
Εικόνα 3. 17 Ο Action Editor. Χρησιμοποιείται για την δημιουργία ενός action

Εικόνα 3. 18 Ο NLA Editor. Χρησιμοποιείται για να επεξεργαστούμε πολλά actions

Συνεχίζοντας το παράδειγμα μας, για να δημιουργήσουμε ένα animation, επιλέγουμε το armature του χαρακτήρα μας και μπαίνουμε σε Pose Mode, πηγαίνουμε στο παράθυρο Action Editor, επιλέγουμε Action Editor Mode εάν δεν είναι ήδη ενεργοποιημένο και κάνουμε ADD NEW action. Στην συνέχεια αλλάζουμε το όνομα του action στο επιθυμητό πατώντας πάνω του και αφού μεταφερθούμε στο πρώτο frame, επιλέγουμε όλα τα bones πατώντας "A" και από το Timeline Window πατάμε το κουμπί Insert Keyframe→VisualLocRot. Έχοντας δημιουργήσει keyframes και αφού ενεργοποιήσουμε το κουμπί "Automatic keyframe insertion for Objects and Bones" μπορούμε να δώσουμε την πόζα που θέλουμε στο μοντέλο μας, χωρίς να ανησυχούμε, αφού σε κάθε αλλαγή της θέσης ή περιστροφής των bones γίνεται αυτόματο keyframe. Με την ίδια λογική μετακινούμαστε στον χρόνο και δημιουργούμε νέες πόζες δημιουργώντας keyframes μέχρι να τελειώσουμε το animation μας, ενώ μπορούμε να το "τρέξουμε" ώστε να εντοπίσουμε τυχόν προβλήματα στην κίνηση και να μετακινηθούμε χειροκίνητη ή μέσω του VTR-like control bar του Timeline παραθύρου σε όποιο frame θέλουμε κάνοντας αλλαγές στο animation.

Αφού τελειώσουμε με το animation και το διορθώσουμε όπου χρειάζεται μέσα από το Ipo curve editor, συνεχίζουμε να δημιουργούμε άλλα κάνοντας ADD NEW στο Action παράθυρο. Τα animations τα κάνουμε στην συνέχεια export σε μορφή .FBX, που θα τα εισάγουμε αργότερα στο Unity game engine. Στην διαδικασία αυτήν μπορούμε να χρησιμοποιήσουμε δύο διαφορετικούς τρόπους. Μπορούμε να κάνουμε export το κάθε animation ξεχωριστά ή να δημιουργήσουμε ένα μεγάλο animation μέσα από το NLA editor και στην συνέχεια να το χωρίσουμε μέσα στο Unity.

Για να κάνουμε export το κάθε animation ξεχωριστά, επιλέγουμε το κάθε ένα από αυτά, πηγαίνουμε στην μπάρα του παραθύρου Timeline και αφού δηλώσουμε το "end frame of the animation" με τιμή ίδια με το τελευταίο frame του συγκεκριμένου action, πηγαίνουμε File→Export→Autodesk FBX. Στο παράθυρο που μας ανοίγει επιλέγουμε Scene Objects, το επιθυμητό Scale Size αν και μπορούμε να αλλάξουμε το scale size μέσα από το Unity αργότερα, απενεργοποιούμε τα κουμπιά Empty, Camera, Lamp και Optimize Keyframes και πατάμε export. Για να κάνουμε export όλα τα animations σε ένα .fbx αρχείο, πηγαίνουμε στο NLA editor και προσθέτουμε όλα τα actions που θέλουμε, πατώντας στο Strip→Add Action Strip. Αφού τα βάλουμε στην σωστή σειρά, βάζουμε στο "end frame of the animation" την τιμή ίση με το μήκος όλων των strips και κάνουμε export.



Εικόνα 3. 19 Το παράθυρο παραμέτρων export στην μορφή Autodesk FBX.

Κεφάλαιο 4

Unity Game Engine

4.1 Πρόλογος

Το Unity είναι ένα ολοκληρωμένο εργαλείο δημιουργίας 3D παιχνιδιών και άλλων διαδραστικών περιεχομένων, όπως η αρχιτεκτονική απεικόνιση ή το 3D animation πραγματικού χρόνου.

Το περιβάλλον ανάπτυξης του Unity τρέχει σε λειτουργικό σύστημα Microsoft Windows και Mac OS X, ενώ τα παιχνίδια και οι εφαρμογές που παράγει μπορούν να τρέξουν σε λειτουργικό Windows, Mac, Android και σε iOS, δηλαδή σε συσκευές iPad και iPhone, ενώ μπορούν να τρέξουν ακόμα και σε παιχνιδομηχανές όπως το Xbox 360, το PlayStation 3 και το Wii. Μπορεί να παράγει ακόμα και browser games χρησιμοποιώντας το Unity web player plug-in που υποστηρίζεται αποκλειστικά από τα λειτουργικά Windows και Mac, όπου στο τελευταίο χρησιμοποιείται επίσης για την ανάπτυξη των Mac widgets.

Το Unity αποτελείται από έναν editor για την ανάπτυξη και τον σχεδιασμό του περιεχομένου αλλά και από μια game engine για την εκτέλεση του τελικού προϊόντος. Είναι παρόμοιο με άλλες game engine που σαν κύρια μέθοδο ανάπτυξης έχουν τον editor και το γραφικό περιβάλλον, όπως το Director, το Blender game engine, το Virtools, το ShiVa3D, το Torque Game Builder και το Gamestudio.

Το Unity στην σύντομη πορεία του κέρδισε το βραβείο Wall Street Journal 2010 στην κατηγορία καινοτόμου τεχνολογίας. Το 2009 ονομάστηκε από την Gamasutra ως μια από τις 5 κορυφαίες εταιρίες παιχνιδιών παγκοσμίως, ενώ το 2006 διαγωνίστηκε στο Apple Design Awards και βγήκε η πρώτη επιλαχούσα για την καλύτερη χρήση γραφικών στο λειτουργικό σύστημα Mac OS X.

4.2 Κύρια χαρακτηριστικά

Το Unity game engine περιλαμβάνει ένα ολοκληρωμένο περιβάλλον ανάπτυξης με ιεραρχική και οπτική επεξεργασία, ένα λεπτομερές property inspector καθώς και ζωντανή προεπισκόπηση παιχνιδιού. Όλα τα στοιχεία φορτώνονται αυτόματα και ανανεώνονται κάθε φορά που ένα από αυτά έχει ενημερωθεί.

Υποστηρίζει μια πληθώρα τεχνικών χαρτογράφησης mapping, δυναμικό φωτισμό και effect πλήρους οθόνης. Υποστηρίζει γλώσσα ShaderLab για την χρήση και δημιουργία shader. Έχει ενσωματωμένη μηχανή φυσικής την Nvidia's PhysX με την υποστήριξη της Nvidia.

Τέλος ο κώδικας είναι βασισμένος στην open source εφαρμογή του .NET Framework, στην Mono, όπου ο προγραμματιστής μπορεί να επιλέξει ανάμεσα στην JavaScript, την C# και την Boo. Επίσης υποστηρίζει αναπαραγωγή βίντεο και ήχου, έχει

ενσωματωμένη μηχανή για την δημιουργία εδάφους και βλάστησης, υποστηρίζει Occlusion Culling, lightmapping, global illumination και Multiplayer networking.

4.2.1 Engine - Μηχανή γραφικών

Το Unity game engine στην τωρινή του έκδοση το Unity 3 προσφέρει εντυπωσιακή ποιότητα και ταχύτητα μέσω του rendering, του φωτισμού, των Nvidia PhysX, του ήχου, της δικτυακής δυνατότητας που προσφέρει αλλά και μέσω του ευέλικτου προγραμματισμού.

Όλα τα στοιχεία φορτώνονται αυτόματα στο Unity και ανανεώνονται κάθε φορά που ένα από αυτά έχει ενημερωθεί. Υποστηρίζεται ολοκληρωτικά από τα προγράμματα 3ds Max, Maya, Blender, Modo, Zbrush, Cinema 4D, Cheetah3D, Photoshop και το Allegorithmic Substance.

4.2.1.1 Rendering

Η μηχανή γραφικών χρησιμοποιεί Direct3D και OpenGL σε λειτουργικό Windows, OpenGL σε λειτουργικό Mac OSX, και OpenGL ES σε λειτουργικό iOS και Andoid.

Το Unity περιέχει ενσωματωμένους shaders από τους πιο απλούς όπως Diffuse, Glossy, εως τους πιο εξειδικευμένους όπως τον Self Illuminated Bumped Specular shader, ενώ προσφέρει την δυνατότητα δημιουργίας Surface Shaders, Vertex and Fragment Shaders και Fixed Function Shaders έτσι ώστε να μπορεί ο χρήστης να εφαρμόσει τον δικό του τρόπο αλληλεπίδρασης των επιφανειών με το φωτισμό, την σκίαση, την προβολή και την χρήση των post process effects.

Η τεχνολογία Deferred Lighting Rendering Path που υποστηρίζει την χρήση πολλαπλών σημείων φωτισμού ανά αντικείμενο με τις διάφορες τεχνικές τις, η τεχνολογία Batching που μειώνει τους κύκλους του επεξεργαστή και η τεχνολογία Occlusion Culling που παραβλέπει τα αντικείμενα που δεν φαίνονται από την κάμερα, απενεργοποιώντας τα, από την διαδικασία του rendering, προσφέρουν τεράστιες βελτιώσεις στον τομέα των επιδόσεων. Τέλος μέσω μίας εξομοίωσης ο χρήστης μπορεί να κάνει τις απαραίτητες αλλαγές και να σιγουρευτεί ότι η εφαρμογή του υποστηρίζεται τέλεια από το hardware στο οποίο στοχεύει να λειτουργεί.

4.2.1.2 Lighting

Το Unity υποστηρίζει την δυνατότητα σκίασης σε πραγματικό χρόνο, όπου απαλές ή έντονες σκιάσεις μπορούν να δημιουργηθούν και να εφαρμοστούν σε αντικείμενα από κάθε

πηγή φωτός στην σκηνή. Με την χρήση των post process effects που εφαρμόζουν στην κάμερα όπως του Screen Space Ambient Occlusion (SSAO) μπορούμε να βελτιώσουμε το οπτικό αποτέλεσμα αλλά και να αλληλεπιδράσουμε με το φως και την σκίαση της σκηνής μας.

Επίσης προσφέρει μια πλήρως ενσωματωμένη τεχνολογία ρεαλιστικής απεικόνισης της ατμόσφαιρας της σκηνής, που λέγεται lightmapping. Στο Unity 3 χρησιμοποιείται ένας από τους πλέον γνωστούς και διαδεδομένους lightmapper, ο Beast της Autodesk. Ο Beast αποτελείται από ένα σύνολο εργαλείων υπεύθυνα για την δημιουργία ρεαλιστικού φωτισμού, προσομοιώνοντας effect φυσικού φωτισμού, όπως color bounce, απαλές σκιές, δυναμικό φωτισμό και φωτισμό σε κινούμε αντικείμενα. Εκτός των άλλων ο Autodesk Beast προσφέρει την δυνατότητα να ενσωματωθεί εύκολα σε έναν level-editor κάτι που στην περίπτωση του Unity 3 έχει γίνει και πλέον λειτουργεί σαν αναπόσπαστο κομμάτι της διαδικασίας του rendering παίρνοντας υπόψη τα meshes, τα textures, τα materials και τον φωτισμό των αντικειμένων.

4.2.1.3 Terrain editor και Substances

Το Unity 3 έχει ενσωματωμένο έναν terrain editor, όπου ο χρήστης μπορεί να σχεδιάσει και να αλλάξει τη μορφολογία του εδάφους, να χρωματίσει και να γεμίσει με χλωρίδα το έδαφος απλά "ζωγραφίζοντας" πάνω σε ένα plane μέσα στο Scene View χρησιμοποιώντας ένα πλήθος εργαλείων.

Εικόνα 4. 1 Εργαλεία δημιουργίας του terrain editor

Επίσης προσφέρει έναν ενσωματωμένο tree creator, όπου ο χρήστης μπορεί να δημιουργήσει από την αρχή ένα δέντρο ή να το αλλάξει οποιαδήποτε στιγμή. Το προϊόν του tree creator μπορεί να χρησιμοποιηθεί σαν ένα GameObject ή να ενσωματωθεί στο terrain engine και να χρησιμοποιηθεί μέσω τον εργαλείων του.

Τέλος υποστηρίζει Allegorithmic Substances, δηλαδή assets που μπορούν να έχουν πολλών ειδών textures, χωρίς να γίνουν αλλαγές σε παραμέτρους. Μπορούμε να αλλάξουμε τις παραμέτρους του substance μέσα από το Unity, δημιουργώντας πολλές παραλλαγές των materials που διαθέτουμε, αφού υπάρχει ένα εύκολο στην χρήση API και είναι πλήρες scriptable. Τα substances συνήθως φτάνουν λίγα μόνο kb σε μέγεθος, διαδικασία που μπορεί να μειώσει το μέγεθος του παιχνιδιού σημαντικά χρησιμοποιώντας τα αντί των bitmaps.

4.2.1.4 Physics

Το Unity περιέχει την πανίσχυρη NVIDIA® PhysX® next-gen Physics Engine. Μέσω αυτής στο Unity 3 συναντάμε συστήματα και τεχνικές προσομοίωσης φυσικής όπως αυτές των Cloth, Soft Bodies, Rigidbodies, Ragdolls, Joints και Cars.

Το σύστημα Cloth διακρίνεται σε 2 κατηγορίες, σε Interactive Cloth και Skinned Cloth. Το πρώτο προσομοιώνει την συμπεριφορά ενός υφάσματος και αλληλεπιδρά πλήρως με την υπόλοιπη σκηνή, ενώ το δεύτερο προσομοιώνει την κίνηση ενός υφάσματος σε skinned animated characters. Με το Soft Bodies σου επιτρέπει να δημιουργείς ελαστικές επιφάνειες στα αντικείμενα σου, που αλληλοεπιδρούν ρεαλιστικά με το υπόλοιπο περιβάλλον.

Με την χρήση του Rigidbody στα αντικείμενα, ενεργοποιείται αυτόματα η φυσική αλληλεπίδρασή τους με το περιβάλλον, αφού επηρεάζονται αυτόματα από την βαρύτητα και από άλλα αντικείμενα αλληλεπιδρώντας μαζί τους μέσω των colliders και των rigidbodies, αλλά μπορούν να δεχτούν και άλλες εξωτερικές δυνάμεις μέσω προγραμματισμού, αυξάνοντας την ρεαλιστική απεικόνιση της σκηνής.

Το Ragdoll σύστημα συμπληρώνεται από ένα ragdoll wizard ενσωματωμένο στο Unity προσφέροντας έτσι στον χρήστη ευκολία και ταχύτητα κατά την δημιουργία ragdolls από τους animated χαρακτήρες του. Επίσης φροντίζει για την σωστή δημιουργία και τοποθέτηση όλων των αναγκαίων Colliders, Rigidbodies και Joints. Το σύστημα των Joints στο Unity περιέχει το Hinge Joint, το οποίο μπορεί να ενώσει 2 rigidbodies κάνοντάς τα να συμπεριφέροντε σαν να έχουν ενωθεί μέσω ενός μεντεσέ, το Spring Joint, που ενώνει 2 rigidbodies κάνοντάς τα να συμπεριφέροντε σαν να έχουν ενωθεί μέσω ενός ελατηρίου, και το Character Joint, που χρησιμοποιείται κυρίως για τα ragdolls και είναι κατάλληλο για την οριοθέτηση της κίνησης των κλειδώσεων σε όλους τους άξονες. Τέλος υπάρχει και αυτό του Configurable Joint που δίνει στον χρήστη απεριόριστη ελευθερία εμφανίζοντας του όλα τα properties σχετικά με τα joints της PhysX και μπορεί να παράγει δικούς του τύπους ενώσεων ή κάποιον από τους παραπάνω.

Στην κατηγορία των Colliders το Unity πέρα από τους κλασσικούς προσφέρει και έναν που προσομοιώνει την λειτουργία και την πολυπλοκότητα της κίνησης μίας ρόδας, τον Wheel Collider. Είναι ένας ειδικά σχεδιασμένος collider για οχήματα εδάφους και προσφέρει ενσωματωμένο collision detection, πιστή αναπαράσταση της φυσικής μίας ρόδας και ολίσθηση με βάση την τριβή των ελαστικών.

4.2.1.5 Audio - Ήχος

Το Unity 3 παρέχει ένα από τα δυνατότερα εργαλεία ήχου στην κατηγορία τον videogames , το FMOD. Δίνει την δυνατότητα αναπαραγωγής πολλών μορφών αρχεία ήχου όπως τα .AIF, .ASF, .ASX, .DLS, .FLAC, .FSB, .IT, .M3U, .MIDI, .MOD, .MP2, .MP3, Ogg Vorbis, .PLS, .S3M, .VAG, .WAX, .WAV, .WMA, .XM και .XMA σε πολλές διαφορετικές πλατφόρμες όπως τις Microsoft Windows (32-bit και 64-bit), Mac OS 8/9/X, Mac OS iPhone, Linux (32-bit και 64-bit), Solaris, Nintendo GameCube, Nintendo Wii, Nintendo 3DS, Xbox, Xbox 360, PlayStation 2/3 και PSP.

Όλες οι βασικές λειτουργίες των αρχείων ήχου μπορούν να ελεγχθούν από καμπύλες εξασθένισης της έντασης (attenuation curves) και φίλτρα όπως τα High/Low Pass, Distortion, Chorus, Echo και Reverb. Ακόμα προσφέρει ζωντανή αναπαραγωγή των αρχείων μέσα από

τον inspector αλλά και αναπαραγωγή μέσα από τον editor για την πραγματική αναπαραγωγή των ήχων της σκηνής και την σωστή αντίληψη του περιβάλλοντος από τον παίκτη.

4.2.1.6 Programming – Προγραμματισμός

Το Unity υποστηρίζει 3 γλώσσες προγραμματισμού, τις C#, JavaScript και μια παραλλαγή της Python την Boo. Και οι 3 είναι εξίσου γρήγορες και ευέλικτες, ενώ μπορούν να χρησιμοποιήσουν τις .NET βιβλιοθήκες και να υποστηρίξουν βάσεις δεδομένων, regular expressions, XML και Networking.

Προσφέρει γρήγορους ρυθμούς προσπέλασης και επανάληψης του κώδικα, είτε είναι JavaScript, C# ή Boo, αφού οι γλώσσες προγραμματισμού με τα native στοιχεία τους, τρέχουν σχεδόν τόσο γρήγορα όσο η C++. Επίσης το Unity είναι βασισμένο στην λογική της open source πλατφόρμα της .NET, την Mono. Αυτό δίνει την δύναμη, ευελιξία και ταχύτητα ενός κορυφαίου περιβάλλοντος προγραμματισμού στον κόσμο.

Επίσης προσφέρει μεγάλες ευκολίες στο event system αφού το Unity χειρίζεται όλες τις συνδέσεις αυτόματα, απλά γράφοντας μια function, ενώ μας δίνεται η δυνατότητα να χρησιμοποιήσουμε την SendMessage και να καλέσουμε οποιαδήποτε function σε οποιοδήποτε script.

Στην διαδικασία του debugging στην έκδοση 3 του Unity έχει ενσωματωθεί ένας debugger, όπου ο χρήστης απλά κάνοντας παύση στο παιχνίδι του, μπορεί να ρυθμίσει breakpoints, να παρακολουθήσει μεταβλητές, ή να προχωρήσει ένα βήμα γραμμή προς γραμμή.

Τέλος ή έκδοση Unity Pro προσφέρει ένα ακέραιο profiler, που βοηθά στην βελτιστοποίηση του παιχνιδιού, αφού ο προγραμματιστής έχει την δυνατότητα να δει και να αναλύσει σε κάθε frame στατιστικά, όπως το που αναλώνουμε τον χρόνο του επεξεργαστή και γιατί.

4.2.1.7 Networking – Δικτύωση

Το Unity στον τομέα της δικτύωσης προσφέρει Backend Connectivity, State Synchronization, Realtime Networking, Remote Procedure Calls, Web Browser Integration και Web Connectivity.

Οι βιβλιοθήκες της .NET μπορούν να χρησιμοποιηθούν για δικτύωση πραγματικού χρόνου, χρησιμοποιώντας μια TCP/IP υποδοχή ή στέλνοντας ένα UDP μήνυμα. Επίσης η σύνδεση σε μια βάση δεδομένων μέσω ODBC, αλλά και η χρήση XML γίνεται πραγματικά εύκολη.

Ο συγχρονισμός μεταξύ των χρηστών της θέσης των αντικειμένων, των διαφόρων δυνάμεων που ασκούνται, των animations και γενικά ολόκληρης της σκηνής μπορεί να επιτευχθεί μέσω του delta compression αλγορίθμου ή μέσω άλλων uncompressed unreliable στρατηγικών. Ενώ μπορούν πολύ εύκολα να κληθούν οποιεσδήποτε functions σε κάθε client

χωρίς την χρήση marshalling ή οποιασδήποτε άλλης τεχνικής δικτυακής διαπραγμάτευσης (network negotiation).

Όταν το παιχνίδι τρέχει σε έναν web browser, ο web player του Unity έχει την δυνατότητα να επικοινωνήσει αυτόματα με την σελίδα που τον φιλοξενεί μέσω JavaScript προσφέροντας μας τις πλήρες δυνατότητες του AJAX. Για την πρόσβαση σε ιστοσελίδες και web services το Unity προσφέρει ένα εύκολο στην χρήση interface. Το WWW interface μπορεί να υποστηρίξει και synchronous και asynchronous τύπους επικοινωνίας.

Τέλος το Unity στην τωρινή του έκδοση μπορεί να μην προσφέρει την δυνατότητα δημιουργίας Massive Multiplayer Online παιχνιδιών, όμως αρκετές εταιρίες σε συνεργασία και πλήρη υποστήριξη από την Unity Technologies, προσφέρουν λύσεις σε αυτόν τον τομέα. Οι Electrotank Universe Platform, Photon Socket Server και Smartfox Server είναι μερικές από αυτές.

4.2.2 Editor

Ο editor του Unity αποτελεί έναν πλήρως εξοπλισμένο world builder, προσφέροντας έναν πανίσχυρο ενσωματωμένο profiler, για την παρακολούθηση κάθε πτυχής του παιχνιδιού, φροντίζει για το αυτόματο unwrap των lightmapped μοντέλων της σκηνής και την δυνατότητα surface και vertex snapping στην διαδικασία της δημιουργίας μιας σκηνής.

Επίσης το Unity είναι σχεδιασμένο με ένα asset pipeline σύστημα, υποστηρίζοντας τα σημαντικότερα εργαλεία σχεδίασης που κυκλοφορούν. Έτσι αναγνωρίζει τα αρχεία του απλά με ένα drag and drop στον Asset φάκελο του project ή τα ενημερώνει απλά κάνοντας save στις αλλαγές. Όλα τα assets έχουν μια προεπισκόπηση στον editor για να εντοπίζονται πιο εύκολα, ενώ το σύστημα tag, layer και μια search διαδικασία το κάνουν ακόμα ευκολότερο στην χρήση. Τέλος όλα τα assets με ένα απλό drag and drop εισάγονται στην σκηνή, ενώ όλες οι public μεταβλητές τους εμφανίζονται στον Inspector του editor για ευκολότερη και γρηγορότερη αλλαγή.

4.2.2.1 Integrated Editor

Ο editor πέραν του διαδραστικού preview, προσφέρει την δυνατότητα δημιουργίας prefabs πολύπλοκων αντικειμένων της σκηνής. Τα prefabs μπορούν εύκολα να τοποθετηθούν ξανά στην σκηνή, είτε κατευθείαν με drag and drop είτε μέσω προγραμματισμού, δημιουργώντας εύκολα και γρήγορα το περιβάλλον μας. Τα αρχικά prefabs μεταδίδουν όλες τις αλλαγές που δέχονται, είτε είναι μεγάλες, είτε μικρές, σε όλα τα εξαρτώμενα αντικείμενα της σκηνής.

Η δυνατότητα live preview του editor βοηθάει στην διαδικασία της δημιουργίας του περιβάλλοντος ενός παιχνιδιού, αφού επιτρέπει στιγμιαία την προεπισκόπηση του παιχνιδιού σε οποιαδήποτε πλατφόρμα και ρυθμίσεις έχουμε επιλέξει. Επίσης παράλληλα με το live preview, μπορούν να γίνουν αλλαγές σε assets, ρυθμίσεις, materials και scripts χωρίς να επηρεαστεί η σκηνή, με άμεση προεπισκόπηση των αλλαγών, που βοηθά στην άμεση και γρήγορη εξερεύνηση όλων των περιπτώσεων που θέλουμε να εξετάσουμε.

Τέλος το Unity από πλευράς customization δίνει την δυνατότητα να δημιουργηθούν εξειδικευμένα εργαλεία επεξεργασίας μέσα στο παράθυρο του editor και να ενσωματωθούν σε αυτόν, αφού οι δυνάμεις του Unity GUI συστήματος, δημιουργούν τις κατάλληλες προϋποθέσεις για δημιουργία οποιουδήποτε εργαλείου, βρεθεί αναγκαίο για την διεκπεραίωση κάποιου project.

4.2.2.2 Scene Construction

Το Unity προσφέρει μεγάλη ευκολία στην διαδικασία του scene construction, αφού απλά με ένα drag and drop των meshes στον Assets folder και μετά στο Scene window, τα 3d μοντέλα είναι έτοιμα να τους εφαρμοστούν materials, scripts και colliders. Μπορούμε να αλλάξουμε την θέση, την γωνία και το μέγεθος των αντικειμένων της σκηνής, ενώ τα εργαλεία grid snapping και surface snapping μαζί με το vertex snapping tool τοποθετήσουμε τα αντικείμενα μας με μεγάλη ακρίβεια.

Μέσα στον editor αντικείμενα όπως κάμερες, φώτα, colliders, audio reverb zones εμφανίζουν χαρακτηριστικά gizmos με την θέση τους και το πεδίο δράσης τους. Ο editor είναι εξαιρετικά ευέλικτος, αφού με την χρήση των layouts μπορούμε να επικεντρωθούμε σε ένα σημείο της σκηνής μας απλά εμφανίζοντας και εξαφανίζοντας ολόκληρα κομμάτια της.

4.2.2.3 Asset Pipeline

Το Unity είναι σχεδιασμένο με ένα asset pipeline σύστημα που υποστηρίζει τα σημαντικότερα εργαλεία σχεδίασης όπως τα Maya, 3ds Max, Cinema 4D και Blender. Το Unity υποστηρίζει 3d models, bone system και textures σχεδόν από όλα τα 3D προγράμματα, ενώ ανανεώνει τα ήδη υπάρχοντα assets απλά με ένα save.

Επίσης υποστηρίζει TrueType fonts, δυνατότητα δημιουργίας normal map από οποιοδήποτε texture, δημιουργία υψηλής ποιότητας mipmaps, δυνατότητα αλλαγής του μεγέθους και της μεθόδου κωδικοποίησης των textures και τέλος παρέχει την δυνατότητα εισαγωγής οποιασδήποτε μορφής αρχείων ήχου υποστηρίζεται από την FMOD και την αλλαγή του σε Ogg Vorbis για την εξοικονόμηση μεγέθους του published παιχνιδιού.

Τέλος στην έκδοση 3.4 του Unity υποστηρίζεται η χρήση των substances, δηλαδή μπορεί να εισαχθεί ένα Allegorithmic Substance κατευθείαν μέσα στο Unity και μέσω του editor να γίνουν αλλαγές των παραμέτρων του ή μέσω κώδικα με την χρήση του substance importer class να έχουμε άμεση επαφή με procedural material.

4.2.2.4 Unity Asset Server

Το Unity Asset Server είναι ένα add-on προϊόν, που προσφέρει την δυνατότητα ολόκληρες ομάδες να δουλεύουν ταυτόχρονα με τα ίδια assets, είτε είναι στον ίδιο χώρο, είτε απομακρυσμένα.

Έχει σχεδιαστεί έτσι ώστε να υποστηρίζεται από τα λειτουργικά συστήματα Microsoft Windows, Mac OS X και Linux, προσφέροντας ευελιξία, ενώ η PostgreSQL βάση δεδομένων που υποστηρίζει, προσφέρει αξιοπιστία, ακεραιότητα των δεδομένων, ευκολία στην διαχείριση και την δημιουργία αντιγράφων ασφαλείας. Τα αρχεία που έχουν τροποποιηθεί, μετακινηθεί ή διαγραφεί, ενημερώνονται αυτόματα και στιγμιαία χωρίς σφάλματα.

4.2.3 Publishing

Το Unity προσφέρει την δυνατότητα να μετατραπεί το παιχνίδι που έχει αναπτυχθεί για οποιαδήποτε πλατφόρμα λειτουργικού συστήματος στοχεύει ο developer. Επίσης μπορεί να επιλεχτεί μέσω του project settings ρυθμίσεις για την ποιότητα των γραφικών και να ρυθμιστούν τι χαρακτηριστικά θα εμφανίζονται σε κάθε μια από της προεπιλεγμένες ρυθμίσεις γραφικών όπως : fast, simple, good, beautiful. Ακόμα με την χρήση .psd αρχείων textures, μπορούμε να ρυθμίσουμε τον τρόπο συμπίεσης και το μέγεθος ανάλυσης των textures που θα χρησιμοποιούνται σε κάθε πλατφόρμα.

Το Unity μπορεί να χρησιμοποιηθεί για την δημιουργία PC και Mac standalone αρχείων, έκδοση του standalone αλλά για χρήση στο διαδίκτυο μέσω του unity web player, για χρήση σε περιβάλλοντα iOs και Android, όπως επίσης και για την δημιουργία παιχνιδιών για της κονσόλες Wii, Playstation 3 και Xbox 360.

4.2.3.1 Standalone Microsoft Windows OS και Mac OS

Το Unity έχει βελτιστοποιηθεί να δουλεύει άψογα και στα δυο λειτουργικά συστήματα και εξακολουθεί να αναβαθμίζεται με κάθε αλλαγή που δέχονται. Η χρήση του Direct3D σε Windows standalone παιχνίδια δίνει εντυπωσιακά αποτελέσματα με την χρήση των τελευταίων τεχνικών και τεχνολογιών γραφικών, ενώ το standalone σε Mac OSX συστήματα χρησιμοποιεί OpenGL.

Ακομα το Unity 3 προσφέρει υψηλής απόδοσης χαρακτηριστικά rendering όπως το deferred lighting, τα πλήρως customizable effects, την δυνατότητα παραμετροποίησης και δημιουργίας shader, την τεχνολογία Dual-Lightmapping που δίνει ποιότητα και οπτική πιστότητα του περιβάλλοντος και τέλος μέσω του Occlusion Culling που κρατά υψηλά τα fps του παιχνιδιού αφαιρώντας από την διαδικασία του rendering αντικείμενα που δεν εμφανίζονται στην κάμερα.

Το Unity έχει περάσει από ένα τεστ συμβατότητας με μια μεγάλη λίστα hardware και έτσι το standalone δεν χρειάζονται κάποιον ειδικό driver και επιπλέον εγκατάσταση του για να τρέξει σε κανένα μηχάνημα, είτε είναι καινούργιο με τους τελευταίους drivers, είτε είναι παλιό.

4.2.3.2 Web publishing

Το Unity δημιουργεί υψηλής ποιότητας παιχνίδια ακόμα και για χρήση σε browser, ενώ δίνεται και η δυνατότητα ανάπτυξης και δημιουργίας 3D dashboard widgets για Mac OS X λειτουργικά συστήματα. Σήμερα με πάνω από 60 εκατομμύρια εγκατεστημένους web players, το Unity πρωτοπορεί στην χρήση high-end περιεχομένου. Η έκδοση 3 υποστηρίζει την Java WebStart και έτσι οι χρήστες μπορούν να κάνουν εγκατάσταση του web player με ένα απλό mouse click.

Επίσης προσφέρει πλήρες browser integration, που σημαίνει ότι μπορεί να ενσωματωθεί πλήρως σε μια διαδικτυακή σελίδα κοινωνικού δικτύου ή σε μια άλλη τεχνολογία, αφού μπορούμε να καλέσουμε functions μέσω του web player στην σελίδα ή και το ανάποδο.

Ακόμα είναι σχεδιασμένο έτσι ώστε να δουλεύει άψογα με οποιονδήποτε από τους κορυφαίους browser, Internet Explorer, Mozilla Firefox, Opera ή Google Chrome χρησιμοποιεί ο χρήστης, ενώ δίνει την δυνατότητα να επιλέξει ο χρήστης μεταξύ τριών διαφορετικών προ-εγκατεστημένων WebPlayer templates ή να δημιουργήσει έναν δικό του μέσω HTML και να τον προσθέσει στην λίστα.

Τέλος η τεχνολογία WWW streaming και auto-streaming συνεργάζονται άψογα μεταξύ τους μειώνοντας τον χρόνο φόρτωσης της σελίδας και του παιχνιδιού, ενώ υπάρχει η δυνατότητα να αλλαχτεί η progress bar, το χρώμα και τα γραφικά, έτσι ώστε να γίνει πιο ευχάριστη η αναμονή.

4.2.3.3 iOS και Android publishing

Οι iOS και Android συσκευές είναι πλέον από τις πιο διαδεδομένες στον κόσμο. Στις πρώτες έχουν αναπτυχθεί μέσω του Unity και είναι αυτήν την στιγμή live στο App Store πάνω από χίλια παιχνίδια για iPhone, iPod Touch και iPad. Ενώ Οι συσκευές android έχουν αποδειχτεί από τις πιο γρήγορα αναπτυσσόμενες πλατφόρμες για κινητά τηλέφωνα και έχουν κερδίσει μια μεγάλη μερίδα κόσμου. Για την ανάπτυξη παιχνιδιών σε πλατφόρμες iOS και Android υπάρχει ένα τεράστιο community με παραδείγματα και βοήθεια σε εξειδικευμένα χαρακτηριστικά της κάθε συσκευής, όπως και τα αντίστοιχα APIs.

Ο Unity Editor μπορεί να μιμηθεί τις γραφικές δυνατότητες Android συσκευών και των διαφορετικών iOS συσκευών για την αποφυγή ανομοιομορφιών στο οπτικό αποτέλεσμα μεταξύ της published έκδοσης και αυτής του σταδίου ανάπτυξης. Επίσης με το Unity Remote, μπορούμε να χρησιμοποιήσουμε μια πραγματική συσκευή Android, iPhone, iPod Touch ή iPad για να δούμε και να “τεστάρουμε” το παιχνίδι μας ζωντανά.

Οι νέες συσκευές iOS και Android υποστηρίζουν OpenGLES 2.0, το οποίο δίνει την δυνατότητα στο Unity, την δημιουργία παιχνιδιών με χρήση shaders για ρεαλιστικά αποτελέσματα, ενώ με την τεχνολογία Occlusion Culling της Umbra μπορούμε να κάνουμε τα περιβάλλοντα μας να τρέχουν γρηγορότερα κάνοντας bake δεδομένα στα ήδη υπάρχοντα textures.

Τέλος μέσα από το Player Settings και πριν κάνουμε built το παιχνίδι μας, μπορούμε να ορίσουμε το μέγεθος ανάλυσης και το orientation (κάθετα ή οριζόντια) του παιχνιδιού, αλλά και άλλες λεπτομέρειες όπως το εικονίδιο και το splash screen του. Όλα αυτά, όπως και όλη η διαδικασία μετατροπής του παιχνιδιού στην κατάλληλη μορφή γίνονται χωρίς να χρειάζεται να ασχοληθούμε καθόλου με κώδικα.

4.2.3.4 Wii, PlayStation και Xbox publishing

Το Unity δίνει την δυνατότητα να αναπτυχθούν παιχνίδια για τις τρεις αυτές κονσόλες παιχνιδιών, εύκολα, δίνοντας άμεσα και απλά ολοκληρωμένες λύσεις σε scripting classes για την χρήση και των προγραμματισμό των εξειδικευμένων controllers.

Για την δημιουργία ή/και προώθηση στην αγορά ενός παιχνιδιού για οποιαδήποτε από αυτές τις τρεις κονσόλες Xbox 360, Playstation 3, Wii, χρειάζεται η πιστοποίηση εξουσιοδοτημένου προγραμματιστή από τις Microsoft, Sony και Nintendo αντίστοιχα. Η ανάπτυξη του παιχνιδιού μπορεί να επιτευχθεί όπως ακριβώς θα γινόταν για πλατφόρμες Microsoft Windows ή Mac OS X μέχρι τέλος και για όσο αναμένεται η εξουσιοδότηση από την εταιρία.

4.2.4 Γνωρίζοντας και χρησιμοποιώντας το Unity Game Engine για την υλοποίηση της πτυχιακής.

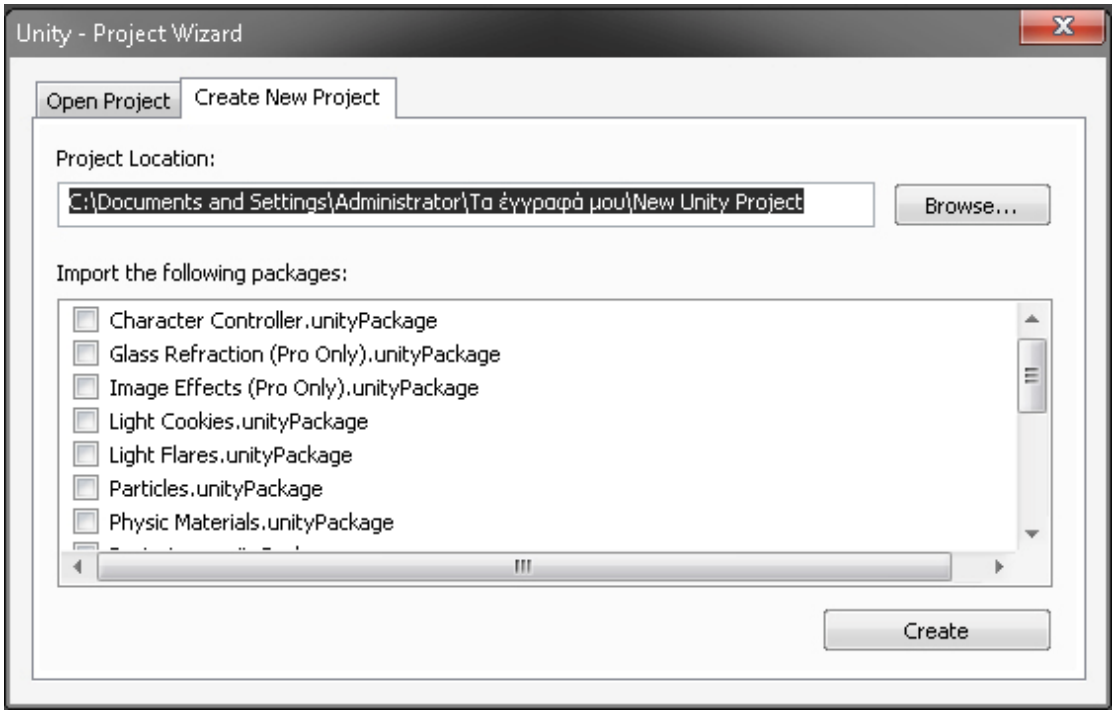
Αφού αναφερθήκαμε στα κύρια χαρακτηριστικά του Unity και στις δυνατότητες που προσφέρει, ήρθε η ώρα να δούμε το πρόγραμμα από μέσα, με την χρήση παραδειγμάτων. Θα αναφερθούμε στα βασικά εργαλεία του, στην διαδικασία εισαγωγής και διαχείρισης των assets, στην χρήση των physics, lights, audio, και τέλος θα δείξουμε την διαδικασία υλοποίησης και προγραμματισμού διάφορων βασικών στοιχείων του παιχνιδιού.

4.2.4.1 Δημιουργία νέου project

Ξεκινώντας το πρόγραμμα εμφανίζεται ένα παράθυρο και καλούμαστε να ανοίξουμε ένα υπάρχον project ή να δημιουργούμε ένα καινούργιο, ορίζοντας του ένα όνομα και το path που θέλουμε να αποθηκευτεί. Επίσης ο χρήστης καλείτε να επιλέξει αν θέλει να εισάγει packages, δηλαδή assets που προσφέρονται από default με το Unity ή που έχει κατεβάσει/αγοράσει ο ίδιος. Με την κάθε δημιουργία ενός νέου project το πρόγραμμα δημιουργεί ένα φάκελο, με το όνομα που δηλώσαμε, στο συγκεκριμένο path και μέσα του δημιουργεί τρείς ακόμα φακέλους, τον Assets, τον Library και τον Temp.

Ο φάκελος Assets είναι ο φάκελος στον οποίο αποθηκεύονται όλα τα αντικείμενα, δηλαδή όλα τα 3d modes, textures, αρχεία βίντεο και ήχου, fonts, και scripts που θέλουμε να χρησιμοποιήσουμε στο project μας ή που δημιουργούμε στην πορεία. Τα αντικείμενα που προστίθενται στον φάκελο αυτό εισάγονται αυτόματα μέσα στο Unity και μπορούμε να τα διαχειριστούμε μέσα από το Project panel.

Οι φάκελοι Library και Temp είναι εξίσου απαραίτητοι και μεγάλης σημασίας. Το Unity χρησιμοποιεί αυτούς τους φακέλους για να αποθηκεύει δεδομένα για την συσχέτιση μεταξύ των assets και πως χρησιμοποιούνται στο project. Ο φάκελος Temp διαγράφεται από το Unity κατευθείαν μετά τον τερματισμό του project, ενώ ο Library παραμένει ως έχει. Είναι ζωτικής σημασίας για το project τα αποθηκευμένα δεδομένα στους φακέλους αυτούς να μην τροποποιούνται ή διαγράφονται.



Εικόνα 4. 2 Unity Project Wizard. Παράθυρο δημιουργίας νέου project και επιλογή εισαγωγής packages

Ανοίγοντας το project βλέπουμε στο πάνω μέρος της οθόνης το βασικό μενού με τα κουμπιά File, Edit, Assets, GameObject, Component, Terrain, Window και Help, όπως επίσης

Play, Pause, Frame Step κουμπιά για την εξομοίωση της σκηνής και κάποια εργαλεία για την μετακίνηση των αντικειμένων και εμάς στην σκηνή. Τέλος βλέπουμε έξι panels, τα Scene, Game, Inspector, Hierarchy, Project και Console, σε διάφορα σημεία στην οθόνη και τα οποία μπορούμε επιλέγοντας τα, να τα μετακινήσουμε μέσα στο πρόγραμμα και έτσι να δημιουργήσουμε μια διάταξη που μας βολεύει.

4.2.4.2 Η κεντρική γραμμή μενού και τα βασικά εργαλεία του Unity

Παρακάτω θα αναφερθούμε αναλυτικά στα κουμπιά του menu και στο πως μπορούμε να τα χρησιμοποιήσουμε.

Αρχικά πατώντας στο κουμπί File, βλέπουμε ότι μπορούμε να δημιουργήσουμε, να ανοίξουμε, ή να σώσουμε μια σκηνή ή ένα project. Επίσης έχει το κουμπί build, με το οποίο μπορούμε να δημιουργήσουμε μια standalone έκδοση του παιχνιδιού μας, ενώ με το build settings μπορούμε να εισάγουμε της σκηνές που θέλουμε στην διαδικασία του build και να ρυθμίσουμε τα χαρακτηριστικά του, όπως μέγεθος default ανάλυσης, εικονίδιο του εκτελέσιμου αρχείου και διάφορες μορφές rendering path. Τέλος μέσω του File→Exit μπορούμε να κάνουμε έξοδο από το πρόγραμμα.

Στο κουμπί Edit θα βρούμε τα κλασσικά Undo/Redo, Cut/Copy/Paste, Duplicate και Delete κουμπιά, όπως επίσης πλήκτρα εύρεσης ενός αντικειμένου, focus και zoom, επιλογής όλων των αντικειμένων και τα πλήκτρα Play/Pause/Step. Ακόμα θα βρούμε ένα κουμπί Preferences για την εξατομίκευση του Unity, τα κουμπιά Project και Render Settings, στα οποία θα αναφερθούμε αργότερα, κουμπί Graphic και Network Emulation για την εξομοίωση του παιχνιδιού με διαφορετικές ρυθμίσεις σύνδεσης στο διαδίκτυο και με την χρήση διαφορετικών μοντέλων shader και τέλος ένα κουμπί για την ρύθμιση του Snapping Tool.

Στην κατηγορία Assets βρίσκουμε κουμπιά για την δημιουργία, εισαγωγή, διαγραφή και ενημέρωση των assets, όπως επίσης και κουμπιά για την εισαγωγή, εξαγωγή, ενημέρωση και ρύθμιση των Unity packages. Πατώντας Assets→Create μπορούμε να δημιουργήσουμε έναν φάκελο ή scripts σε Java, C Sharp και Boo, να δημιουργήσουμε Shader, Prefab, Animation, Material, GUI Skin και πολλά ακόμα. Όλα τα assets εμφανίζονται μέσα στο project panel, όπου μπορούμε να τα διαχειριστούμε και να τα τροποποιήσουμε ανάλογα με τις ανάγκες μας.

Επιλέγοντας το κουμπί GameObject μπορούμε να δημιουργήσουμε αντικείμενα όπως φώτα, κάμερες, ragdolls, ζώνες ήχου και ανέμου αλλά και primitive meshes όπως κύβους, σφαίρες, κυλίνδρους, κάψουλες και επίπεδα. Επίσης από εκεί μπορούμε να διαχειριστούμε τα αντικείμενα της σκηνής μας και να δημιουργήσουμε ή να καταργήσουμε σχέσεις γονέα-παιδιού, αλλά και να ευθυγραμμίσουμε τα αντικείμενα μεταξύ τους και με το view ή το view με αυτά. Τα GameObjects είναι αντικείμενα που δημιουργούνται αυτόματα στην σκηνή και δεν εμφανίζονται μέσα στο project panel ή στον φάκελο Assets, παρά μόνο αν δηλωθούν σαν prefabs.

Στην κατηγορία Component του μενού θα βρούμε όλα τα απαραίτητα συστατικά που χρειάζονται τα αντικείμενα μας για την χρήση τους στο παιχνίδι. Είναι ταξινομημένα για την διευκόλυνση μας στις κατηγορίες Mesh, Particles, Physics, Audio, Rendering, Miscellaneous και Scripts. Στην κατηγορία Component→Mesh θα βρούμε ένα Mesh Filter, Text Mesh και τον Mesh Renderer που είναι το αναγκαίο συστατικό για να γίνει αντιληπτό το αντικείμενο από την κάμερα.

Στο Particles βρίσκονται δύο ειδών particle emitter, ο ellipsoid και ο mesh, όπως και οι renderer τους. Με τα particles μπορούμε να δημιουργήσουμε στοιχεία όπως νερό, χιόνι, φωτιά και βροχή, ενώ με την χρήση του particle animator και world particle collider μπορούμε να δημιουργήσουμε αληθοφανές, κινούμενο και ζωντανό αποτέλεσμα. Τέλος με τον Trail renderer μπορούμε να δημιουργήσουμε μια ουρά σε κινούμενα αντικείμενα με διάφορες ρυθμίσεις.

Στην κατηγορία Component→Physics θα βρούμε διάφορα είδη Collider για τα αντικείμενα μας, το Rigidbody που είναι το απαραίτητο συστατικό για να αλληλεπιδρούν με την βαρύτητα και με άλλες εξωτερικές δυνάμεις και το περιβάλλον εφόσον έχουν έναν Collider εφαρμοσμένο, Joints για την σύνδεση αντικειμένων και το Character Controller που είναι ένα είδος Collider με διάφορα έτοιμα χαρακτηριστικά. Επίσης σε αυτήν την κατηγορία θα βρούμε το Cloth components για την δημιουργία και εξομοίωση αντικειμένων με την συμπεριφορά υφάσματος, όπως και διάφορες μορφές Joints για την ένωση αντικειμένων με την συμπεριφορά αρθρώσεων.

Στο Component→Audio βρίσκονται ένας Audio Listener το συστατικό που εφαρμόζεται συνήθως στην κάμερα και "διαβάζει" τους ήχους του περιβάλλοντος, ένα Audio Source, δηλαδή πηγές ήχου που μπορεί να εφαρμοστεί σε οποιοδήποτε αντικείμενο της σκηνής και ένα Audio Reverb Zone, όπου ένα αντικείμενο μπορεί να δημιουργεί μια ζώνη γύρω του στην οποία ο ήχος παίζει κάτω από διάφορες ρυθμίσεις δημιουργώντας διαφορετικά αποτελέσματα. Ακόμα υπάρχουν διάφορα φίλτρα, όπως Low/High Pass Filter, Echo, Distortion, Reverb και Chorus Filter που μπορούν να εφαρμοστούν σε πηγές ήχου.

Στην κατηγορία Rendering υπάρχουν διάφορα components όπως κάμερα, πηγή φωτός, halo, lens flare, projector, GUITextures και GUIText. Σε αντίθεση με την δημιουργία GameObject, αυτά τα components εφαρμόζονται κατευθείαν πάνω σε οποιοδήποτε αντικείμενο έχουμε επιλεγμένο χωρίς να δημιουργούν επιπλέον gameObjects στην σκηνή, αλλά απλά προσδίδουν επιπλέον ιδιότητες.

Στην κατηγορία Miscellaneous μπορούμε να επιλέξουμε να εφαρμόσουμε στα αντικείμενα μας διάφορα components όπως το Line Renderer, το Raycast Collider, το Network View, το Wind Zone και το Animation.

Τέλος στην κατηγορία Scripts εμφανίζονται όλα τα scripts που εισάγουμε μέσα στον φάκελο Assets ή που δημιουργούμε εμείς κατά την ανάπτυξη του παιχνιδιού μας. Τα scripts εμφανίζονται επίσης και στο project panel από όπου μπορούμε να τα εφαρμόσουμε με drag and drop στο αντικείμενο που θέλουμε στην σκηνή μας.

Το επόμενο κουμπί που συναντάμε είναι το κουμπί Terrain, με το οποίο μπορούμε να δημιουργήσουμε ένα Terrain και να ρυθμίσουμε την ανάλυσή του, να εισάγουμε και να εξάγουμε Heightmap – Raw data, να δημιουργήσουμε ένα terrain με την μορφολογία ενός mesh και τέλος να τροποποιήσουμε και να ελέγξουμε ένα πλήθος ρυθμίσεων. Την διαδικασία και τον τρόπο δημιουργίας ενός Terrain θα την αναλύσουμε παρακάτω, μαζί με την λεπτομερή αναφορά στα εργαλεία του Terrain Editor.

Στην κατηγορία Window της γραμμής μενού βρίσκονται όλα τα κουμπιά για την εμφάνιση των panel, την αλλαγή, επαναφορά, αποθήκευση ή διαγραφή των layout του Unity, όπως επίσης και κουμπιά για την εμφάνιση παραθύρων για την χρήση Lightmapping και Occlusion Culling.

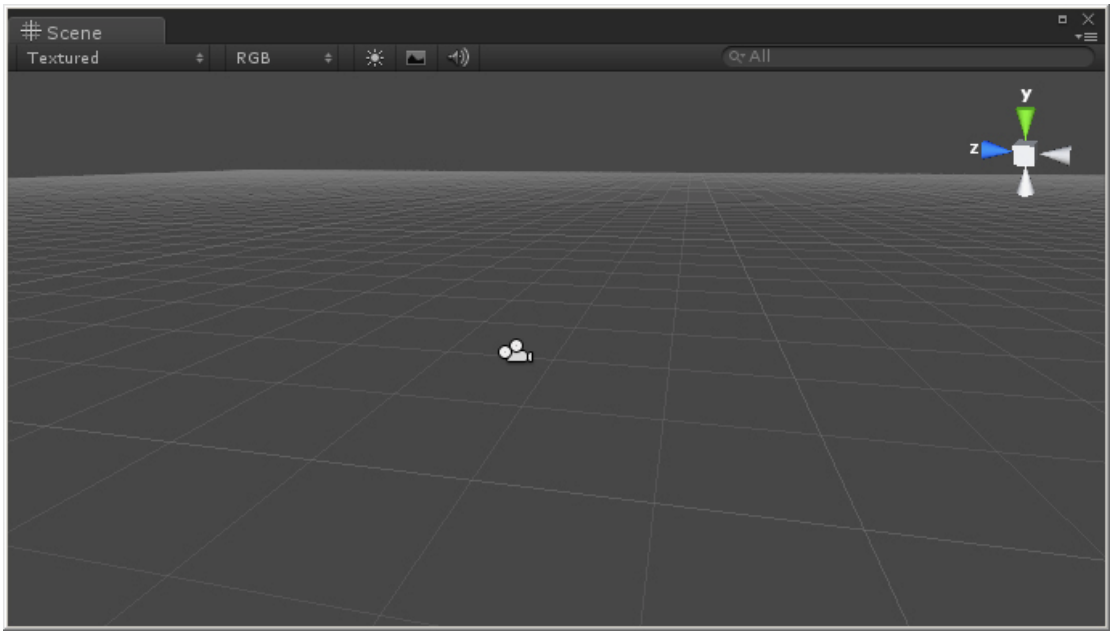
Τέλος βρίσκεται το κουμπί Help, όπου μπορούμε να βρούμε πληροφορίες για το Unity, να οδηγηθούμε στα Unity και Reference Manuals, να οδηγηθούμε στις ιστοσελίδες του Unity Forum, του Unity Answers και του Unity Feedback, να ελέγξουμε για ενημερώσεις και να στείλουμε Bug Report στην εταιρία.

4.2.4.3 Τα παράθυρα και το γραφικό περιβάλλον του Unity

Το γραφικό περιβάλλον του Unity αποτελείτε από το βασικό μενού, τα εργαλεία της σκηνής και τα panels. Τα panels είναι παράθυρα τα οποία έχουν σχεδιαστεί έτσι ώστε να μπορούν να εφαρμοστούν στο γραφικό περιβάλλον του προγράμματος σε διάφορες διατάξεις, όπως επιθυμεί ο χρήστης. Εξυπηρετούν στην ανάπτυξης ενός παιχνιδιού, σε διαφορετικά σημεία και με πολλούς τρόπους, αφού διευκολύνουν τον προγραμματιστή/σχεδιαστή στην δουλειά του. Όπως είπαμε και πιο πάνω είναι χωρισμένα στα Scene, Game, Inspector, Hierarchy, Project και Console, όμως υπάρχουν ακόμα και τα Animation, Profiler, Asset Store και Asset Server. Παρακάτω θα αναφερθούμε στην χρησιμότητα του κάθε ένα από αυτά.

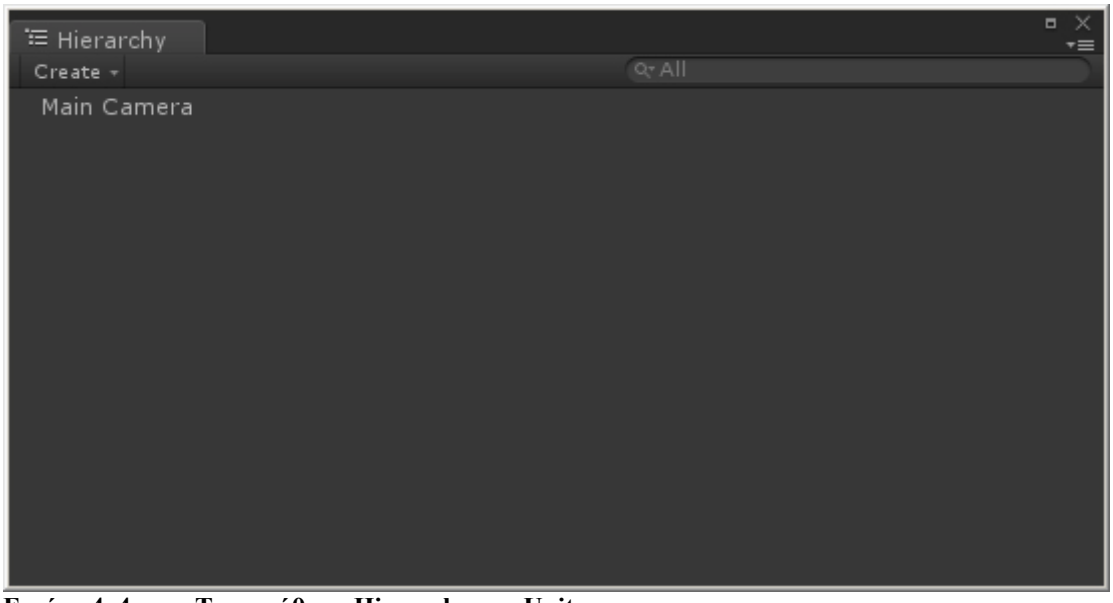
Το πρώτο παράθυρο που βλέπει κανείς ξεκινώντας το πρόγραμμα είναι το Scene. Αρχικά στο παράθυρο Scene βλέπουμε μια άδεια σκηνή με μόνο μια κάμερα εφαρμοσμένη σε αυτήν. Στο πάνω μέρος του παραθύρου, όπως φαίνεται και στην εικόνα 1.3.3.α, βρίσκεται μια γραμμή εργαλείων με διάφορα κουμπιά. Αρχικά υπάρχει ένα κουμπί με το οποίο μπορούμε να αλλάξουμε τον τρόπο προβολής της σκηνής σε Textured, σε Wireframe, σε Texture – Wireframe, σε προβολή των Render Paths και σε προβολή των Lightmaps Resolution. Δίπλα υπάρχει ένα πλήκτρο, όπου μπορούμε να επιλέξουμε ανάμεσα σε χρωματισμό RGB, σε χρήση του Alpha καναλιού των αντικειμένων μας, σε προβολή της σκηνής με Overdraw, αλλά και με την χρήση των Mipmaps. Ακόμα βλέπουμε τρία μικρά εικονίδια, με τα οποία μπορούμε ενεργοποιώντας τα να εξομοιώσουμε και να αναπαράγουμε μέσα στον Editor διάφορα στοιχεία της σκηνής. Πατώντας το κουμπί με την λάμπα, μπορούμε να εξομοιώσουμε τα effect φωτισμού και σκίασης από όλες της πηγές φωτός της σκηνής, ενώ με την χρήση του κουμπιού με το τοπίο, ατμοσφαιρικά effect, όπως ομίχλη, ambient φωτισμός αλλά και η χρήση του Skybox ενεργοποιούνται. Τέλος με το κουμπί του ήχου ενεργοποιούνται και αναπαράγονται όλες οι πηγές ήχου, ενώ σαν δέκτης χρησιμοποιείτε το σημείο στο οποίο βρισκόμαστε μέσα στην σκηνή. Στα δεξιά του παραθύρου διακρίνεται ένα εργαλείο αναζήτησης με επιλογή μεταξύ των τριών φίλτρων, All, Name, Type και με το οποίο μπορούμε να ψάξουμε για αντικείμενα που βρίσκεται στην σκηνή μας.

Ακόμα μέσα στο Scene panel παρατηρούμε στο πάνω-δεξί μέρος ένα είδος πυξίδας με το οποίο μπορούμε να προσανατολιστούμε, να μετακινήσουμε τα αντικείμενα μας στην κατάλληλη κατεύθυνση αλλά και να αλλάξουμε μεταξύ έξι διαφορετικών προεπιλεγμένων View της σκηνής. Πατώντας στον κύβο που βρίσκεται στο κέντρο αλλάζουμε από Isometric σε Perspective View, ενώ πατώντας πάνω σε κάθε ένα από τα βελάκια του x,y,z άξονα μπορούμε να εναλλαχθούμε μεταξύ των Top, Down, Front, Back, Right και Left Isometric Views.Το παράθυρο Scene αποτελεί το μέρος στο οποίο συνδέεται και συνδυάζεται το οπτικό κομμάτι του παιχνιδιού. Εδώ τοποθετούνται όλα τα αντικείμενα, δημιουργούνται τα game levels και το περιβάλλον.



Εικόνα 4. 3 Το παράθυρο Scene του Unity

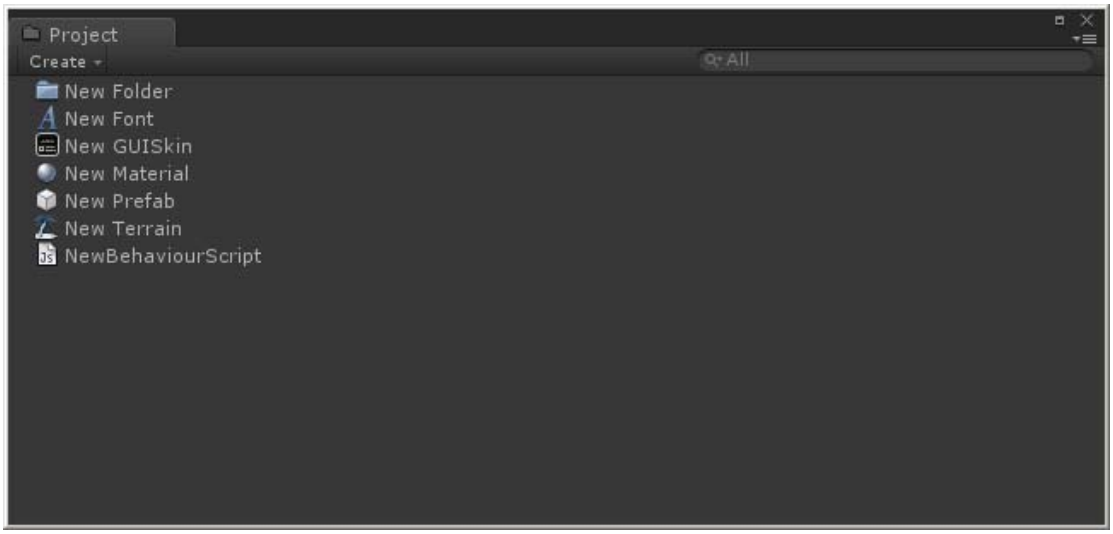
Στο παράθυρο Hierarchy εμφανίζονται όλα τα αντικείμενα της σκηνής σε μια λίστα από τα ονόματά τους. Οτιδήποτε υπάρχει στην σκηνή, υπάρχει και στο panel αυτό και οτιδήποτε διαγραφεί από αυτό, διαγράφεται και από την σκηνή. Όπως φαίνεται και στην εικόνα 1.3.3.β, στο panel αυτό υπάρχει μόνο ένα πεδίο αναζήτησης και το κουμπί Create, που ουσιαστικά αποτελεί ένα shortcut του GameObject→Create Other της γραμμής μενού του Unity



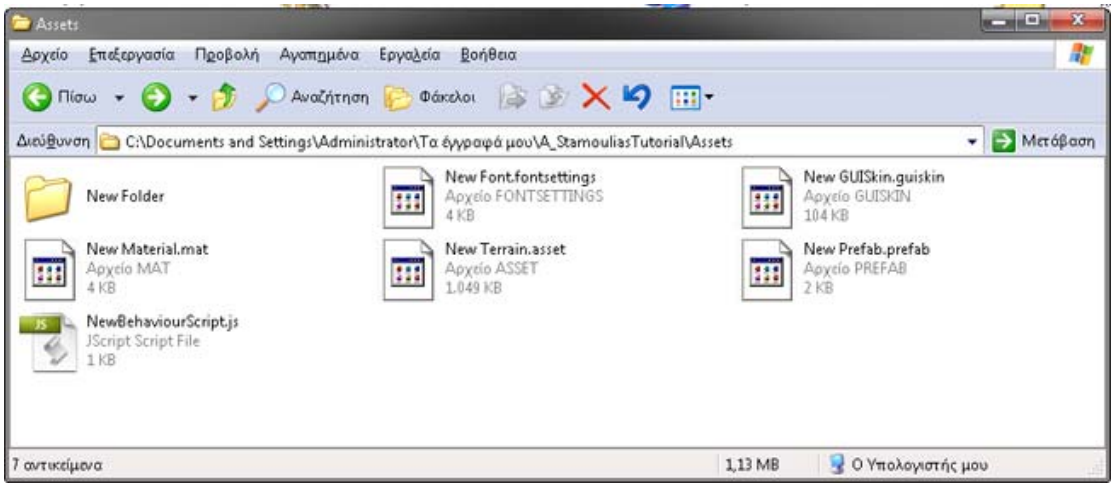
Εικόνα 4. 4 Το παράθυρο Hierarchy του Unity

Το Project panel είναι το σημείο στο οποίο εμφανίζονται όλα τα assets που έχουμε εισάγει ή δημιουργήσει στο project μας. Όπως φαίνεται και στην εικόνα 1.3.3.γ, έχουμε δημιουργήσει διάφορα Assets στο project μας, τα οποία και εμφανίζονται σε μια κατακόρυφη λίστα με τα ονόματα τους και το χαρακτηριστικό τους εικονίδιο. Βλέποντας τώρα την εικόνα 1.3.3.δ παρατηρούμε ότι τα ίδια αντικείμενα έχουν δημιουργηθεί αυτόματα και στον Assets φάκελο, χωρίς να κάνουμε αποθήκευση ή να τα έχουμε χρησιμοποιήσει. Όλα τα αντικείμενα ανανεώνονται αυτόματα από το Unity μετά την κάθε τροποποίηση τους και οι αλλαγές τους γίνονται ορατές και στο project panel. Κάθε αντικείμενο που βρίσκεται εκεί και κατά συνέπεια στον Asset φάκελο, μπορεί να εφαρμοστεί στην σκηνή μας είτε με drag and drop, είτε μέσω προγραμματισμού, ενώ δεν υπάρχει περιορισμός στον αριθμό των αντιγράφων που θα δημιουργήσουμε ή θα καλέσουμε σε αυτήν. Ακόμα η διαγραφή αντικειμένων από το Project panel ισοδυναμεί με την διαγραφή τους από τον Asset φάκελο και δεν υπάρχει η δυνατότητα αναίρεσης. Το παράθυρο Project λοιπόν, μπορούμε να πούμε οτι αποτελεί την βιβλιοθήκη μας, όπου αποθηκεύονται όλα εκείνα τα υλικά με τα οποία μπορούμε να σχεδιάσουμε, προγραμματίσουμε και υλοποιήσουμε το παιχνίδι μας.

Τέλος παρατηρώντας το παράθυρο Project του Unity, βλέπουμε ότι υπάρχει ένα κουμπί Create, όπου πατώντας το μπορούμε να δημιουργήσουμε όλα αυτά τα αντικείμενα που βρίσκονται στην κεντρική γραμμή μενού, στην κατηγορία Assets→Create. Επίσης στην καρτέλα αυτή εμφανίζεται και ένα πεδίο αναζήτησης για τα περιεχόμενα του φακέλου Assets, με φίλτρα All, Name, Type και Label.

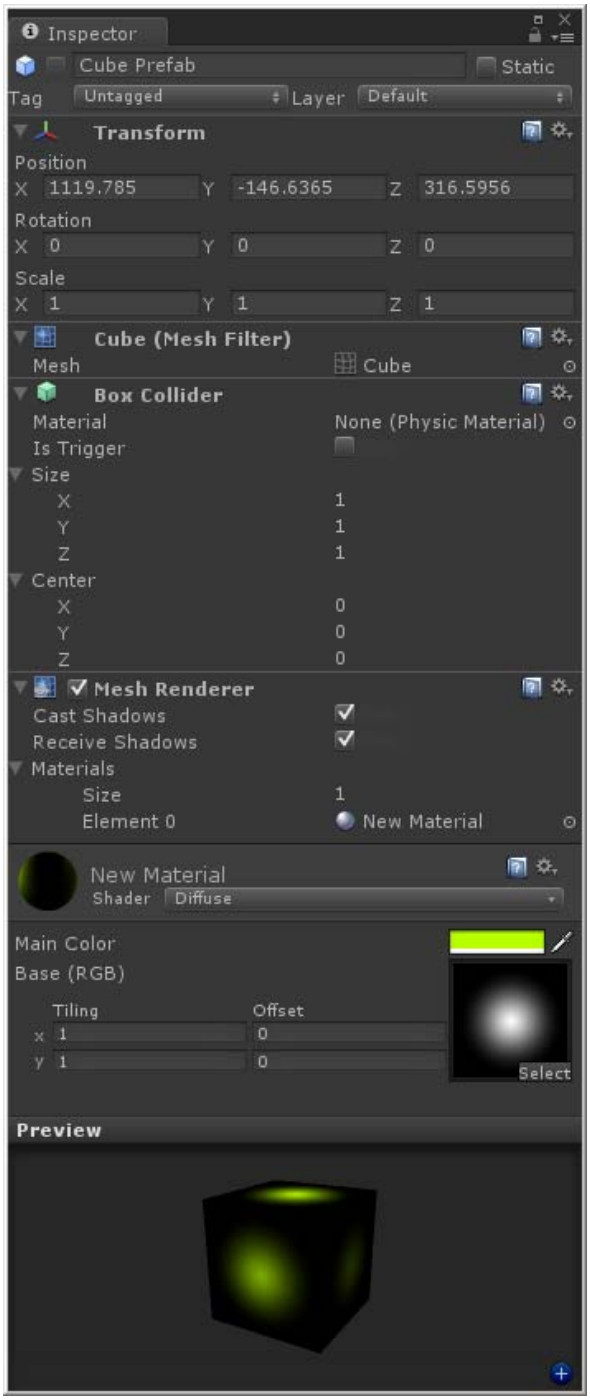


Εικόνα 4. 5 Το παράθυρο Project του Unity με μερικά Assets



Εικόνα 4. 6 Τα περιεχόμενα του φακέλου Assets του project μας

Επιλέγοντας ένα αντικείμενο από το παράθυρο Project ή Hierarchy, ή ακόμα και επιλέγοντας ένα αντικείμενο της σκηνής, αμέσως εμφανίζονται τα χαρακτηριστικά του στο παράθυρο Inspector. Το παράθυρο αυτό χρησιμοποιείται αποκλειστικά για την απεικόνιση και τροποποίηση των gameObjects, είτε αυτά είναι meshes, materials, prefabs ή scripts. Στον Inspector εμφανίζονται όλα τα χαρακτηριστικά των gameObjects από τα πιο απλά όπως ενός κύβου, έως και στα πιο περίπλοκα όπως ένας χαρακτήρα με materials, scripts, rigidbody, character controller και ένα σωρό ακόμα components εφαρμοσμένα πάνω του.



Εικόνα 4. 7 Απεικόνιση των components ενός prefab και των παραμέτρων τους στο παράθυρο Inspector.

Στην εικόνα 4.7 βλέπουμε τι εμφανίζεται στον Inspector επιλέγοντας ένα prefab ενός απλού primitive Cube που βρίσκεται στο Project panel του Unity. Στην κορυφή εμφανίζεται το όνομα του αντικειμένου, ενώ αριστερά του διακρίνετε ένα εικονίδιο ενός κύβου, που υποδεικνύει ότι το επιλεγμένο αντικείμενο είναι ένα prefab. Επίσης βλέπουμε δυο dropdown menu με τα ονόματα Tag και Layer. Τα Tags χρησιμοποιούνται για να ομαδοποιήσουμε αντικείμενα ως προς το είδος τους, ενώ είναι ένας πολύ χρήσιμος τρόπος για να βρίσκουμε αντικείμενα χρησιμοποιώντας τα tags τους και για να αναγνωρίζουμε αντικείμενα που περνάνε μέσα από triggered περιοχές, έρχονται σε επαφή με colliders, ή για να

αναγνωριστούν μέσω τεχνικών raycasting. Τα Layers χρησιμοποιούνται κυρίως από την κάμερα για την διαδικασία render ενός μόνο μέρους της σκηνής, από τα φώτα για τον φωτισμό μονό ενός μέρους της σκηνής, αλλά μπορούν επίσης να χρησιμοποιηθούν και από το raycasting για την δημιουργία collision ή την παράβλεψη αυτών. Μας δίνεται η δυνατότητα να δημιουργήσουμε νέα Tags και Layers πηγαίνοντας στο Edit→Project Settings→Tags και να τα προσθέσουμε στα GameObject μας.

Ακόμα βρίσκουμε ένα checkbox με όνομα Static, το οποίο χρησιμοποιείται στην διαδικασία του Occlusion Culling. Επιλέγοντας το αντικείμενο να είναι static, ουσιαστικά του δίνουμε την ικανότητα να κρύβει τα αντικείμενα που βρίσκονται πίσω του από την κάμερα, αφαιρώντας τα από την διαδικασία του Rendering και συνεπώς αυξάνοντας τις επιδόσεις του παιχνιδιού. Κάθε αντικείμενο της σκηνής μας, το οποίο δεν πρόκειται να μετακινηθεί κατά την διάρκεια του παιχνιδιού, είναι καλό να επιλέγεται ως static.

Κάθε αντικείμενο έχει επίσης ένα Transform Component, το οποίο καθορίζει την θέση, γωνία και κλίμακα στους άξονες x,y,z. Όλες οι ιδιότητες του Transform καθορίζονται από τον γονέα (parent) του αντικειμένου και σε περίπτωση που δεν έχει γονέα, οι ιδιότητες του Transform υπολογίζονται σε σχέση με το World Space. Τα αντικείμενα με τον Transform Component εμφανίζουν 3 άξονες, τους x,y,z στο 3D περιβάλλον τους και χρωματίζονται με διαφορετικά χρώματα τον κάθε ένα από αυτούς για ευκολία στον χειρισμό, ενώ ακολουθούν τον κανόνα XYZ = RGB, που σημαίνει ότι οι τρεις άξονες x,y,z έχουν κόκκινο, πράσινο και μπλε χρώμα αντίστοιχα.

Ο Transform Component μπορεί να ελεγχθεί μέσω του Inspector και αλλάζοντας την τιμή μίας μεταβλητής του Position, Rotation και Scale ή μέσω των εργαλείων Move, Rotate και Scale τα οποία είναι τοποθετημένα στην πάνω αριστερή γωνία του Unity. Χρησιμοποιώντας αυτά τα εργαλεία, μπορούμε να χειριστούμε τα αντικείμενα μας απευθείας μέσα από το Scene panel, ενώ το Unity θα αλλάζει ελαφρά το gizmo του Transform Component κάθε φορά που επιλέγουμε ένα διαφορετικό εργαλείο, όπως φαίνεται στην εικόνα 4.8 παρακάτω.

Εικόνα 4. 8 Απεικόνιση των gizmos των εργαλείων Move, Rotate και Scale

Στην εικόνα 4.7 βλέπουμε ότι το αντικείμενο μας έχει ένα Mesh Filter αλλά και έναν Mesh Renderer. Αυτά τα δυο components είναι υπεύθυνα για την απεικόνιση του αντικειμένου στην οθόνη. Κάθε φορά που εισάγουμε ή δημιουργούμε ένα mesh στο project

μας, αυτόματα δημιουργείται ένα Skinned Mesh Renderer αν το mesh είναι skinned, ή ένα Mesh Filter και ένας Mesh Renderer αν δεν είναι. Το Mesh Filter παίρνει το mesh από τον Assets φάκελο και τον στέλνει στον Mesh Renderer έτσι ώστε να απεικονιστεί στην οθόνη μας στην θέση που έχει καθοριστεί μέσα από τον Transform Component. Στον Mesh Renderer μπορούμε επίσης να επιλέξουμε αν θέλουμε το αντικείμενο μας να δημιουργεί και να δέχεται σκιές, όπως επίσης να δηλώσουμε ένα ή παραπάνω materials.

Το κάθε material που είναι εφαρμοσμένο σε ένα αντικείμενο, εμφανίζεται μέσα στον Inspector. Τα materials χρησιμοποιούνται από τα Mesh και Particle Renderers των GameObjects και παίζουν σημαντικό ρόλο στο πώς εμφανίζεται ένα αντικείμενο στην οθόνη, ενώ σε περίπτωση που δεν έχουμε δηλώσει materials στους Mesh και Particle Renderers, δεν εμφανίζονται σωστά ή και καθόλου στην οθόνη μας. Οι συνήθεις ιδιότητες των materials είναι ο Shader, το Main Color και το Base Texture. Ανάλογα τον τύπο του Shader που θα επιλέξουμε περισσότερες ιδιότητες μπορούν να εμφανιστούν στον Inspector. Με το Main Color ουσιαστικά μπορούμε να αλλάξουμε την απόχρωση, ενώ το Base αποτελείται από το Texture, που θέλουμε να εφαρμόσουμε στο αντικείμενο μας.

Τέλος στο αντικείμενο μας παρατηρούμε ένα Box Collider, που δεν είναι τίποτε άλλο από έναν Collider σε σχήμα κύβου. Ο κάθε Collider έχει σαν ιδιότητες το Material, που είναι μια αναφορά σε ένα Physic Material το οποίο καθορίζει τον τρόπο που αλληλεπιδρά ο Collider με τους άλλους, το Is Trigger το οποίο αν ενεργοποιηθεί ο Collider αγνοείται από την physics engine και χρησιμοποιείται μόνο για triggering events, το Size που καθορίζει το μέγεθος του στους x,y,z άξονες και το Center που καθορίζει την θέση του Collider σε σχέση με την θέση του αντικειμένου.

Σε όλα τα αντικείμενα τα Components μπορούν να διαγραφούν, να επαναφερθούν στην αρχική τους κατάσταση ή σε αυτή του prefab σε περίπτωση που έχουμε αλλάξει ένα αντίγραφο, όπως επίσης μπορούμε να εισάγουμε νέα Components ή να τα αντικαταστήσουμε.

Όπως φαίνεται και στην εικόνα 4.9 παρακάτω, το Unity προσφέρει ένα Play/Pause/Step σύστημα κουμπιών με το οποίο μπορούμε να τρέξουμε την σκηνή μας μέσα από τον Editor.

Εικόνα 4. 9 Απεικόνιση του συστήματος κουμπιών Play Mode του Editor

Πατώντας Play το Unity φέρνει στο προσκήνιο το παράθυρο Game. Στο παράθυρο αυτό, εξομοιώνεται η σκηνή μας και μπορούμε να παίξουμε έτσι ακριβώς όπως θα ήταν στην περίπτωση που είχαμε κάνει publish το παιχνίδι μας. Όμως κατά την διάρκεια του Play Mode το Unity μας επιτρέπει να κάνουμε προσωρινές αλλαγές στην σκηνή μας, ώστε να εξερευνήσουμε διαφορετικές περιπτώσεις και να αποφασίσουμε ευκολότερα σε πιθανές αλλαγές.

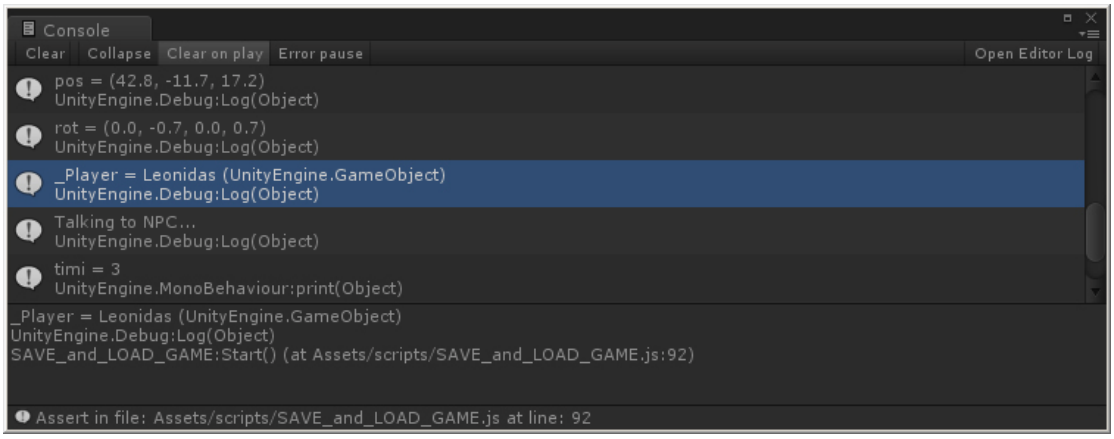
Στην μπάρα του Game View υπάρχει ένα Aspect drop-down μενού, με το οποίο μπορούμε να αλλάξουμε την ανάλυση του παιχνιδιού και να δοκιμάσουμε πώς θα φαίνεται το παιχνίδι μας σε διαφορετικού τύπου οθόνες. Στα δεξιά υπάρχει το κουμπί Maximize on Play, το οποίο μεγεθύνει το Game View στο 100% του Editor παραθύρου την επόμενη φορά που θα ξανατρέξουμε το παιχνίδι μας. Το κουμπί Gizmos, ενεργοποιεί στο Game View όλα τα gizmos των αντικειμένων, εμφανίζοντας τα όπως ακριβώς στο Scene View. Τέλος έχουμε το

κουμπί Stats, το οποίο ενεργοποιώντας το εμφανίζει, όπως φαίνεται και στην εικόνα 4.10, στατιστικά Rendering. Το παράθυρο αυτό είναι πολύ χρήσιμο, αφού μπορούμε να δούμε κατά την διάρκεια του παιχνιδιού διάφορα στατιστικά σχετικά με τον frame rate, τα draw calls και την RAM.

Εικόνα 4. 10 Στιγμιότυπο του παιχνιδιού στην φάση σχεδίασης της σκηνής, με τα στατιστικά του, όπως φαίνεται μέσα από το Game View του Editor

Καθώς τρέχουμε το παιχνίδι μας παρατηρούμε στο κάτω μέρος της οθόνης μας ένα Status Bar, όπου εμφανίζονται διάφορα μηνύματα. Πατώντας διπλό click πάνω τους, ή πηγαίνοντας στο Window→Console εμφανίζεται το παράθυρο Console. Σε αυτό το παράθυρο εμφανίζονται διάφορα μηνύματα, errors, warnings και debug μηνύματα του παιχνιδιού. Έχουμε την δυνατότητα να στέλνουμε τα δικά μας μηνύματα στην Console χρησιμοποιώντας την εντολή print(), την Debug.Log(), την Debug.LogWarning() και την Debug.LogError(). Με διπλό click σε όλα τα μηνύματα του Console, οδηγούμαστε στο script και στην γραμμή του κώδικά μας, από όπου δημιουργείται.

Όπως παρατηρούμε και στην εικόνα 1.3.3.ι, το παράθυρο Console έχει μια toolbar διαχείρισης που βοηθάει στο φιλτράρισμα των μηνυμάτων. Πατώντας το κουμπί **Clear** καθαρίζονται όλα τα μηνύματα της Console μέχρι εκείνη την χρονική στιγμή, ενεργοποιώντας το πλήκτρο **Collapse**, ίδια μηνύματα που γράφονται στην Console σε διάφορα χρονικά σημεία εμφανίζονται πλέον μόνο μια φορά, ενεργοποιώντας το **Clear on play**, η Console ρυθμίζεται να καθαρίζει τα μηνύματα κάθε φορά που μπαίνουμε σε Play Mode, ενώ ενεργοποιώντας το πλήκτρο **Error pause**, τα μηνύματα που στέλνονται στην Console με Debug.LogError() κάνουν παύση στο Play Mode του παιχνιδιού. Τέλος πατώντας το **Open Editor Log**, ανοίγει το Editor Log σε έναν Text Editor.

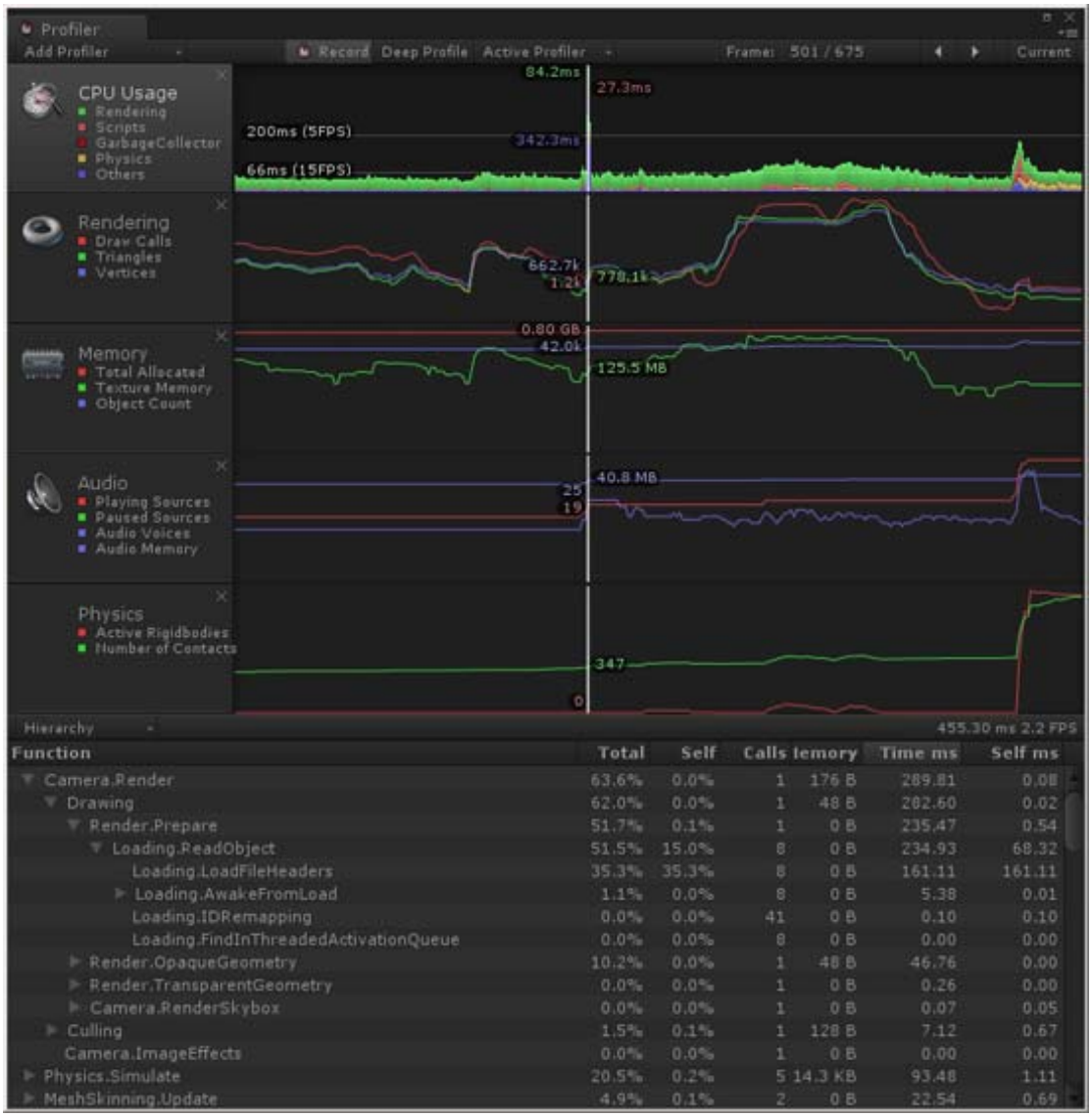


Εικόνα 4. 11 Στιγμιότυπο από το Console Window

Το Unity Pro προσφέρει ένα ακόμα εργαλείο, που βοηθάει τον χρήστη να βελτιώσει το παιχνίδι του. Το εργαλείο αυτό δεν είναι άλλο από τον Profiler, με το οποίο μπορούμε να δούμε για κάθε frame πόσος χρόνος και μνήμη χρειάζεται σε διάφορες πτυχές του παιχνιδιού μας, όπως στην διαδικασία του rendering και του animating. Έχοντας ενεργοποιήσει τον Profiler πηγαίνοντας στο Window→Profiler και πατώντας Play στον Editor καταγράφονται όλα τα δεδομένα σχετικά με τις επιδόσεις του παιχνιδιού. Όλα τα δεδομένα αυτά εμφανίζονται μέσα στον Profiler σε γραφικές παραστάσεις χαρισμένες στις πέντε κατηγορίες, CPU Usage, Rendering, Memory, Audio και Physics. Πατώντας οπουδήποτε στην χρονική γραμμή που έχει σχηματιστεί, δηλαδή επιλέγοντας ένα frame, εμφανίζονται στο κάτω μέρος του Profiler, αναλυτικές πληροφορίες για το συγκεκριμένο frame.

Όπως παρατηρούμε στην εικόνα 4.12 παρακάτω ο Profiler έχει μια control toolbar στο πάνω μέρος του παραθύρου, με την οποία μπορούμε να ενεργοποιήσουμε ή απενεργοποιήσουμε την διαδικασία του profiling πατώντας το **Record**, να περιηγηθούμε στα καταγεγραμμένα frames ή να προσθέσουμε έναν από τους πέντε Profiler με το πλήκτρο **Add Profiler**. Όταν ενεργοποιήσουμε το πλήκτρο **Deep Profile**, ο Profiler διεισδύει βαθειά μέσα στα scripts καταγράφοντας δεδομένα για όλες τις κλήσεις των συναρτήσεων, δηλαδή καταγράφει τον χρόνο που σπαταλά μέσα στον κώδικα του παιχνιδιού. Η διαδικασία του Deep Profile παρόλο που μπορεί να αποδειχτεί σημαντική κατά την διάρκεια της βελτίωσης και διόρθωσης του παιχνιδιού, χρησιμοποιεί ένα πολύ μεγάλο μέρος της μνήμης και σε έναν περίπλοκο κώδικα μπορεί να μην είναι δυνατή η χρήση του.

Στον Profiler μπορούμε να κλείσουμε όποια είδη Profiler δεν μας ενδιαφέρουν, ή ακόμα και να κρατήσουμε μόνο ένα έτσι ώστε να έχουμε κάθε φορά μια καλύτερη εικόνα του τι συμβαίνει στο τομέα που μας ενδιαφέρει. Πατώντας σε κάθε ένα από τα πέντε είδη Profiler, CPU Usage, Rendering, Memory, Audio και Physics, βλέπουμε διαφορετικά στατιστικά στοιχεία και σχετικά με της παραπάνω διαδικασίες. Στον CPU Usage Profiler έχουμε την δυνατότητα να αλλάξουμε θέση στα στοιχεία, Rendering, Scripts, GarbageCollector, Physics και Other που καταγράφονται, εμφανίζοντας την γραφική παράσταση με τρόπο που μας βολεύει.

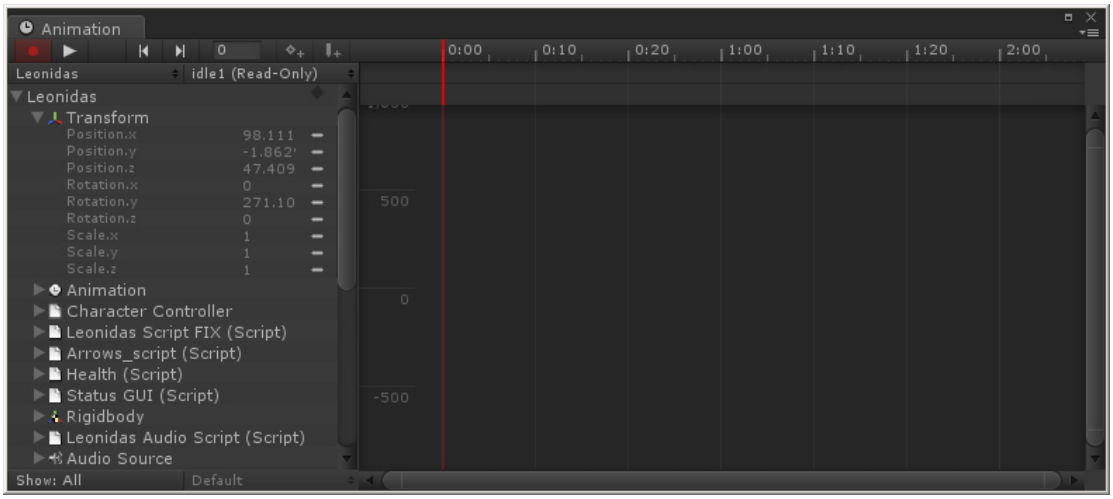


Εικόνα 4. 12 Στιγμιότυπο από τον Profiler με τα αναλυτικά καταγεγραμμένα στατιστικά στοιχεία της κατηγορίας CPU Usage του frame 501

Ακόμα ένα παράθυρο που είναι δυνατόν να ενσωματώσουμε στο Unity Editor πηγαίνοντας στο Window→Animation ή πατώντας Ctrl+6 είναι το Animation View με το οποίο μπορούμε να δούμε ή να επεξεργαστούμε ένα animation clip. Στο παράθυρο Animation εμφανίζεται πληροφορίες και τα διάφορα components του GameObject που έχουμε επιλέξει μέσω του Hierarchy View ή του Scene View, ενώ τα αντικείμενα που έχουν έναν Animation Component αλλά και animation clips είναι δυνατόν να τα επεξεργαστούμε.

Στο πάνω μέρος του παραθύρου βρίσκεται μια μπάρα διαχείρισης των animations με κουμπιά play, next/previous key frame, record, add key frame και add event, όπως επίσης και μια μπάρα timeline. Ακόμα υπάρχει η δυνατότητα να επιλέξουμε μέσα από μια λίστα τα animation clips που είναι δηλωμένα στο Animation Component του αντικειμένου.

Το Animation View είναι φτιαγμένο έτσι ώστε να λειτουργεί σαν ένα συμπληρωματικό εργαλείο διόρθωσης και επεξεργασίας ή σαν ένα εργαλείο δημιουργίας απλών animation clip. Ακόμα μας προσφέρει την δυνατότητα να δρούμε πάνω σε μεταβλητές των materials και άλλων Components, όπως επίσης και να καλούμε functions σε επιλεγμένο χρονικό σημείο με την χρήση του Animation Event.



Εικόνα 4. 13 Το παράθυρο Animation του Unity

Αφού περιγράψαμε όλα τα βασικά panels καθώς και τον τρόπο με τον οποίο τα χρησιμοποιούμε, τώρα θα δείξουμε την διαδικασία και τις τεχνικές που ακολουθήθηκαν για την δημιουργία του περιβάλλοντος και των χαρακτήρων μέσα στο Unity. Επίσης θα κάνουμε μια εκτενείς αναφορά στον προγραμματισμό των αντικειμένων μας, ώστε να δείξουμε, αρκετές από τις δυνατότητες που προσφέρει το Unity.

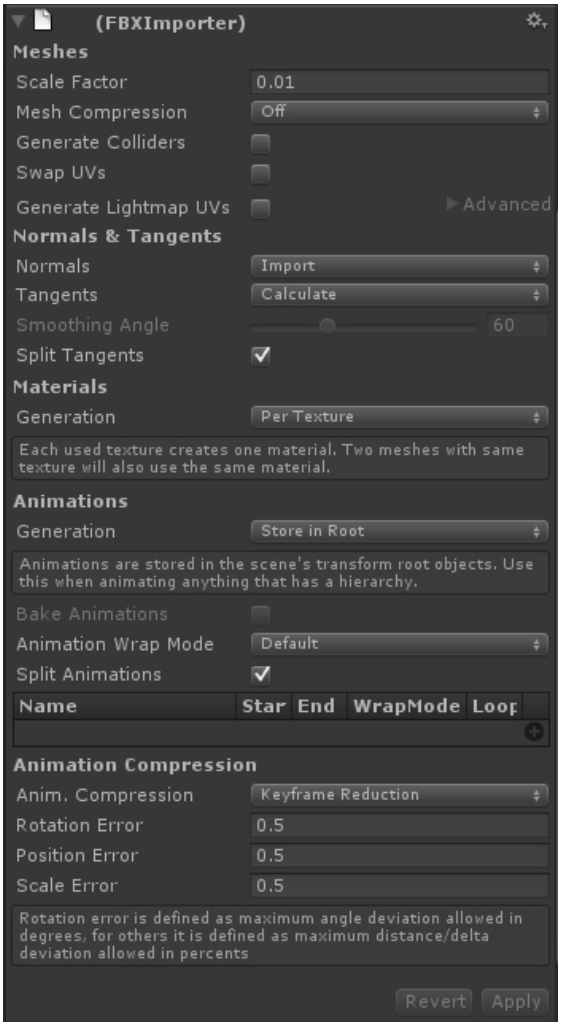
4.2.5 Παραδείγματα

4.2.5.1 Η διαδικασία import του FBX και επεξήγηση του Character Controller Component

Σε κάθε κεφάλαιο αυτής της πτυχιακής δείξαμε τις διαδικασίες που ακολουθήθηκαν, μέσα από διαφορετικά προγράμματα, για την δημιουργία ενός 3D χαρακτήρα. Τώρα στο παράδειγμα αυτό θα δείξουμε πώς θα χρησιμοποιήσουμε τον χαρακτήρα αυτό μέσα στο Unity αναλύοντας τα scripts που δημιουργήσαμε και όλα τα υπόλοιπα components που χρησιμοποιεί.

Αρχικά, αφού έχουμε κάνει import στον φάκελο Assets το .fbx που κάναμε export από το Blender, όπως δείξαμε παραπάνω, το βρίσκουμε στο project panel και το επιλέγουμε, εμφανίζοντας στον Inspector το FBXImporter. Στον FBXImporter μπορούμε να επεξεργαστούμε το μοντέλο κάνοντας scale, να το συμπιέσουμε, να δημιουργήσουμε colliders, να αλλάξουμε το UV και να πειράξουμε τα normal maps και τα materials. Επίσης

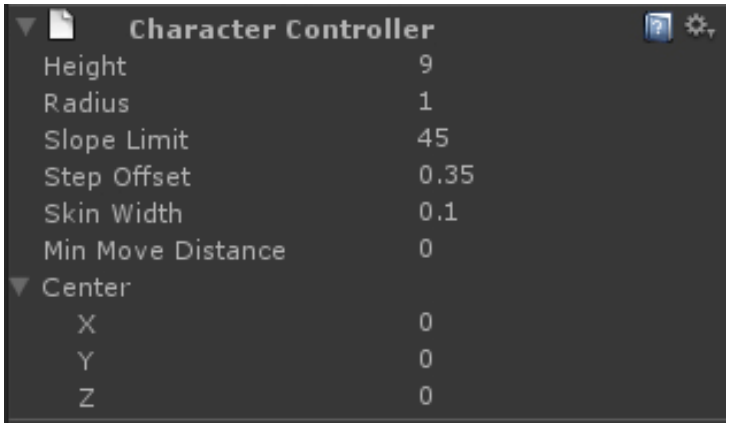
στον FBXImporter βλέπουμε και μια κατηγορία Animation, στην οποία μπορούμε να επιλέξουμε wrap mode στα animation clips, τον τρόπο που θα αποθηκεύονται και μια μέθοδο συμπίεσης. Εδώ σε περίπτωση που έχουμε κάνει export όλα τα animations σε ένα .fbx αρχείο, μέσω του NLA editor του Blender, μπορούμε να τα χωρίσουμε δημιουργώντας καινούργια animation clips και επιλέγοντας αρχικό και τελικό frame καθώς και wrap Mode. Τα animation clips που δημιουργούμε τα δηλώνουμε στο Animation Component του .fbx έτσι ώστε να μπορούμε να τα χρησιμοποιήσουμε και να τα καλέσουμε μέσω script.



Εικόνα 4. 14 Ο FBX Importer του Unity

Γενικά ακόμα και αν προσθέσει στην σκηνή ένα αντικείμενο, μπορούμε να συνεχίσουμε να κάνουμε αλλαγές στον FBXImporter, αφού οι αλλαγές αυτές γίνονται αυτόματα από το Unity σε όλα τα instances του, εφόσον πατήσουμε Apply. Αφού έχουμε κάνει όλες τις απαραίτητες αλλαγές, εισάγουμε το μοντέλο μας στην σκηνή και επιλέγοντας το κάθε mesh ξεχωριστά, δηλώνουμε στον Inspector panel τα texture και normal maps που έχουμε δημιουργήσει, όπως επίσης και τους κατάλληλους shaders. Στον χαρακτήρα μας έχουμε χρησιμοποιήσει Diffuse και Bumped Diffuse shaders για τα ρούχα και το σώμα, Bumped Specular και Reflective/Bumped Specular shaders για την πανοπλία και τα όπλα

Στην συνέχεια επιλέγουμε το μοντέλο μας (τον parent όλων των meshes) και του εφαρμόζουμε το Character Controller component πηγαίνοντας Component→Physics→Character Controller. Το συγκεκριμένο component είναι ένας Capsule collider ειδικά σχεδιασμένος από το Unity για χαρακτήρες, αφού μπορούμε να δηλώσουμε εκτός από το ύψος, την ακτίνα και το κέντρο της κάψουλας, το Slope Limit, το Step Offset, το Min Move Distance και το Skin Width. Στον Character Controller το Slope Limit περιορίζει τον collider του χαρακτήρα να ανεβαίνει σε άλλους colliders με γωνία κλίσης μικρότερη ή ίση της τιμής που δηλώνουμε. Με το Step Offset δηλώνουμε την μέγιστη απόσταση από το πάτωμα, στο οποίο μπορεί να περπατήσει ο χαρακτήρας και να ανέβει άλλους colliders. Στο Min Move Distance ορίζουμε την ελάχιστη τιμή για την οποία ο χαρακτήρας δεν θα μετακινηθεί σε περίπτωση που το επιχειρήσουμε για απόσταση μικρότερη ή ίση με αυτήν. Τέλος με το Skin Width ορίζουμε μια τιμή για την οποία ο collider μπορεί να εισχωρήσει σε έναν άλλον. Η βέλτιστη τιμή του Skin Width είναι ίση με το 10% της ακτίνας του collider, αφού σε περίπτωση που ορίσουμε μια χαμηλότερη ο χαρακτήρας μπορεί να κολλήσει.



Εικόνα 4. 15 Το Character Controller Component

Αφού κάνουμε τις κατάλληλες ρυθμίσεις στον Character Controller, είμαστε έτοιμοι να προγραμματίσουμε τον χαρακτήρα μας. Αρχικά δημιουργούμε ένα νέο JavaScript πηγαίνοντας **Assets→Create→JavaScript** ή μέσω του **Project panel στο Create→JavaScript**. Το νέο αυτό script δημιουργείται στον φάκελο Asset με όνομα **NewBehaviourScript** και εμφανίζεται στο Project panel, ενώ αν έχουμε επιλεγμένο ένα φάκελο στο Project panel, το νέο script θα δημιουργηθεί μέσα σε αυτόν. Στην συνέχεια αφού κάνουμε μετονομασία στο JavaScript, το ανοίγουμε κάνουμε double click πάνω του μέσα στο Project panel. Όλες οι scripting διαδικασίες γίνονται σε έναν default εξωτερικό text editor, ο οποίος μπορεί να αλλάξει από το μενού Edit→Preferences→General.

4.2.5.2 Η χρήση των βασικών function και η σημασία τους

Πριν ξεκινήσουμε να δείχνουμε κομμάτια κώδικα του χαρακτήρα αλλά και του περιβάλλοντος, θα αναφερθούμε στις βασικές functions και στην χρήση τους, ώστε να έχουμε

μια γενική εικόνα. Στο Unity, υπάρχουν αρκετές event functions που εκτελούνται με μια προκαθορισμένη σειρά, καθώς τρέχουμε ένα script. Ξεκινώντας το παιχνίδι και όταν φορτώσει την σκηνή, καλείται η συνάρτηση **Awake**, για κάθε αντικείμενο σε αυτήν. Αμέσως μετά και πριν ξεκινήσει το πρώτο frame καλείτε η συνάρτηση **Start**. Οι δύο αυτές συναρτήσεις καλούνται μόνο μια φορά για κάθε αντικείμενο είτε όταν φορτωθεί η σκηνή, είτε όταν κάνουμε instantiate ένα αντικείμενο στην σκηνή μας.

Η βασική function μέσω της οποίας γίνεται η μεγαλύτερη δουλειά είναι η **Update()**; Η συνάρτηση αυτή καλείται μια φορά για κάθε frame και έτσι χρησιμοποιείται για την δημιουργία του game logic, για της αλληλεπιδράσεις, για την χρήση και ρύθμιση των animations, για την παρακολούθηση και ρύθμιση της θέσης μια κάμερας, κτλ. Εκτός από την Update υπάρχουν δύο ακόμα συναρτήσεις, οι **LateUpdate** και **FixedUpdate** που μπορούν να χρησιμοποιηθούν για παρόμοια χρήση. Η LateUpdate καλείτε και αυτή μια φορά για κάθε frame, όμως αμέσως μετά την Update, που σημαίνει ότι οτιδήποτε είναι μέσα στην Update θα έχει ολοκληρωθεί όταν ξεκινήσει αυτή. Συνήθως η LateUpdate χρησιμοποιείται για την ρύθμιση της θέσης της κάμερας του χαρακτήρα, αφού καλείτε αμέσως μετά την Update, δηλαδή μετά το τέλος της κίνησης του χαρακτήρα και βεβαιώνει ότι θα τον ακολουθεί. Αντιθέτως η FixedUpdate καλείται πιο συχνά από την Update και παρέχει ένα πιο αξιόπιστο ρολόι, ανεξάρτητο από το frame rate του παιχνιδιού. Αυτό συμβαίνει αφού η συγκεκριμένη συνάρτηση, μπορεί να κληθεί και αρκετές φορές σε ένα frame αν το frame rate είναι χαμηλό ή να μην κληθεί καμία φορά ανάμεσα στα frames, αν είναι μεγάλο.

Στην διαδικασία του rendering η σειρά με την οποία καλούνται οι συναρτήσεις είναι η εξής. Πριν η κάμερα διαβάσει την σκηνή για να εμφανίσει τα ορατά αντικείμενα, καλείται η συνάρτηση **OnPreCull** και στην συνέχεια για κάθε αντικείμενο που είναι ορατό ή αόρατο καλείτε η **OnBecameVisible** και **OnBecameInvisible** αντίστοιχα. Αν το αντικείμενο είναι ορατό καλείτε η συνάρτηση **OnWillRenderObject**, μια φορά για κάθε κάμερα και ακολουθεί η **OnPreRender** πριν ξεκινήσει η κάμερα να κάνει render την σκηνή. Στην συνέχεια καλείται η **OnRenderObject**, όπου κάνει render, όλα τα ορατά αντικείμενα που βρήκε παραπάνω και αμέσως μόλις ολοκληρωθεί η διαδικασία αυτή του render για το συγκεκριμένο frame, καλείτε η **OnPostRender** και **OnRenderImage** μέσα στις οποίες μπορεί να γίνει χρήση των post effects. Τέλος στην διαδικασία του rendering, ανήκει και η συνάρτηση **OnGUI**, με την διαφορά ότι καλείτε πολλές φορές στην διάρκεια ενός frame. Αρχικά εκτελούνται οι διεργασίες Layout και Repaint και μετά όλες οι διεργασίες για της εντολές του keyboard και mouse.

Πολύ χρήσιμες συναρτήσεις είναι αυτές των Coroutines. Οι συναρτήσεις αυτές μπορούν να αναστείλουν την εκτέλεσή τους μέχρι να τελειώσει η συγκεκριμένη YieldInstruction που ορίζουμε. Η χρήση των Coroutines μπορεί να γίνει με την yield, όπου η συνάρτηση θα συνεχίσει στο επόμενο frame και αφού έχει τελειώσει η Update, με την **yield WaitForSeconds("*seconds*")**, όπου συνεχίζει μετά τον χρόνο που του ορίζουμε, **yield WaitForFixedUpdate**, όπου συνεχίζει όταν έχουν ολοκληρωθεί όλες οι FixedUpdate όλων των scripts, μέσω της **yield WWW**, όπου συνεχίζει αφού έχει ολοκληρωθεί η λήψη και τέλος μέσω της **yield StartCoroutine(function)**, όπου περιμένει ώσπου να ολοκληρωθεί πρώτα η function που του ορίζουμε.

Τέλος εξίσου σημαντικές είναι και οι συναρτήσεις OnCollisionEnter/OnCollisionExit και OnTriggerEnter/OnTriggerExit, αφού μπορούμε να ελέγξουμε αν ακουμπάνει δύο colliders, ή αν βρίσκεται ο ένας μέσα στον άλλον. Καθώς επίσης και η OnMouseEnter και OnMouseExit με τις οποίες μπορούμε να αλληλεπιδράσουμε μέσω του ποντικιού με κάποιον collider.

4.2.5.3 Το σύστημα animation του Unity και παραδείγματα χρήσης του

Αφού είδαμε τις βασικές function και την σειρά με την οποία εκτελούνται, θα συνεχίσουμε το παράδειγμά μας, δείχνοντας βασικά scripts που δημιουργήσαμε για την λειτουργία του χαρακτήρα αλλά και για το υπόλοιπο περιβάλλον. Στο script του χαρακτήρα μας αρχικά, δηλώνουμε μέσα στην Start() function το Wrap Mode των animations, το layer και το speed τους. Το κάνουμε αυτό για κάθε animation, έτσι ώστε να ρυθμίσουμε τον τρόπο που θα αναπαράγεται, το επίπεδο στο οποίο θα βρίσκεται, ώστε να μπορούμε να κάνουμε ενός είδους μίξη μεταξύ των animation clip, αλλά και να δηλώσουμε την σειρά προτεραιότητα τους, καθώς και την ταχύτητα με την οποία θα παίζει το κάθε ένα.

Ξεκινάμε λοιπόν δημιουργώντας την συνάρτηση Start γράφοντας, function Start(){} και στην συνέχεια μέσα σε αυτήν για κάθε ένα animation γράφουμε τον τρόπο αναπαραγωγής του. Τα είδη WrapModes που υποστηρίζει το Unity είναι: Default, Once, Loop, ClampForever και PingPong. Το default είναι το Once, ή αυτό που δηλώνουμε στον FBXImporter αν το έχουμε αλλάξει. Το WrapMode.Once σταματάει το animation όταν φτάσει στο τέλος του, με το Loop ξεκινάει το animation κάθε φορά που φτάνει στο τέλος του, με το ClampForever παιζει το animation μια μόνο φορά αλλά όταν φτάσει στο τέλος του παίζει το τελευταίο frame συνέχεια, ενώ τέλος με το PingPong πηγαίνει από την αρχή στο τέλος και πάλι πίσω. Μερικά παραδείγματα χρήσης του wrapMode φαίνονται στις παρακάτω γραμμές κώδικα.

```
animation["idle"].wrapMode = WrapMode.Loop;

animation["slash"].wrapMode = WrapMode.Once;

animation["aim"].wrapMode = WrapMode.ClampForever;
```

Στο animation με όνομα idle χρησιμοποιούμε WrapMode.Loop καθώς θέλουμε να παίζει συνέχεια το συγκεκριμένο clip, ενώ το animation του χτυπήματος θέλουμε να παίζει μια μόνο φορά πατώντας ένα πλήκτρο. Τέλος θέλουμε το animation[“aim”] να συνεχίσει να παίζει το τελευταίο frame για όσο εμείς θα έχουμε πατημένο ένα πλήκτρο για αυτό δηλώνουμε στο wrapMode του, το ClampForever.

Εφόσον έχουμε δηλώσει το wrapMode σε όλα τα animation clips, στην συνέχεια τα ταξινομούμε σε layers. Το animation σύστημα του Unity προσφέρει αυτήν την δυνατότητα, μέσω της οποίας μπορούμε να ομαδοποιήσουμε όσα animation clips θέλουμε μέσα σε layers και να κάνουμε μια μίξη μεταξύ τους. Στην ουσία αυτή η μίξη γίνεται προσθέτοντας βάρη στα animation clips που ανήκουν στα layers και τα οποία χρησιμοποιούνται για να καθοριστεί πιο clip έχει μεγαλύτερη προτεραιότητα. Τα animations μέσω της animation.CrossFade(); Μοιράζονται αυτόματα βάρη από το Unity που όταν αθροίζονται φτάνουν στο 100% για κάθε layer. Η ταξινόμηση των animations σε layers γίνεται με τον εξής τρόπο.

```
animation["idle"].layer = -1;

animation["run"].layer = 1;
```

```
animation["slash1"].layer = 2;

animation["slash2"].layer = 2;
```

Στις παραπάνω γραμμές κώδικα γίνεται ταξινόμηση αυτών των animations σε διαφορετικά layers. Στο χαμηλότερο layer, που στο παράδειγμα μας είναι το -1, τοποθετούμε το idle animation, καθώς θέλουμε να παίζει πάντα αλλά να φαίνεται μόνο όταν δεν παίζει κάποιο άλλον μεγαλύτερης προτεραιότητας. Στο layer 2 έχουμε τοποθετήσει δύο animation clips, καθώς θέλουμε να δημιουργήσουμε μια μίξη μεταξύ τους, δηλαδή όταν θα καλείται το slash1, να γίνεται fade out στο slash2 και το ανάποδο. Σε αυτήν την περίπτωση χρησιμοποιείται το animation.CrossFade(); για αυτόματη ή χειροκίνητη κατανομή βάρους.

Ακόμα στην Start function μπορούμε να δηλώσουμε την ταχύτητα με την οποία θα παίζει το κάθε animation, αφού μπορούμε να αυξήσουμε ή να μειώσουμε τον αριθμό των frames που χρειάζεται για να ολοκληρωθεί η αναπαραγωγή τους. Αυτή η τεχνική εκτός της βοήθειας που προσφέρει στην προσαρμογή της ταχύτητας του animation, μας δίνει επίσης την δυνατότητα να χρησιμοποιούμε μικρό αριθμό frames στα animations και συνεπώς να κρατήσουμε μικρό μέγεθος στα animation που κάνουμε export. Στις παρακάτω γραμμές βλέπουμε πώς μπορούμε να ρυθμίσουμε την ταχύτητα τους και συγκεκριμένα του animation[“idle”], που θα αναπαράγεται με το 80% της αρχικής του ταχύτητας, το “run” που θα κάνει να ολοκληρωθεί στο διπλάσιο του αρχικού χρόνου του και το slash, που θα χρειαστεί το μισό του χρόνο.

```
animation["idle"].speed = 0.8;

animation["run"].speed = 0.5;

animation["slash"].speed = 2;
```

Στην συνέχεια και ενώ είμαστε μέσα στην Start(), βάζουμε τον χαρακτήρα μας να παίζει το idle animation, έτσι ώστε να ξεκινήσει αρχίζοντας το παιχνίδι.

```
animation.Play("idle");
```

4.2.5.4 Επεξήγηση των βασικών σημείων του κώδικα της κίνησης του χαρακτήρα

Έχοντας τελειώσει με τα animations, μπορούμε να προχωρήσουμε στον προγραμματισμό της κίνησης του χαρακτήρα. Ο βασικός κώδικας της κίνησης του, γίνεται

μέσα στην Update και είναι σημαντικό να γίνει εκεί, καθώς αυτή η function καλείτε μια φορά για κάθε frame. Η κίνηση του θα γίνει μέσω του Character Controller component, που έχουμε προσθέσει στον χαρακτήρα μας, χρησιμοποιώντας την function Move(). Ο Character Controller εκτός από την Move(), προσφέρει την δυνατότητα να χρησιμοποιήσουμε μια αρκετά πιο απλή function, την SimpeMove(), η οποία λαμβάνει υπόψη του μόνο την ταχύτητα που εφαρμόζεται, ο y άξονας παραβλέπεται και η βαρύτητα εφαρμόζεται αυτόματα. Παρόλα αυτά, εμείς θα χρησιμοποιήσουμε την Move(), αφού η συγκεκριμένη συνάρτηση προσπαθεί να μετακινήσει τον controller μέσα από την οποιαδήποτε κίνηση και περιορίζεται μόνο από collisions.

Αρχικά δηλώνουμε στο script μας μεταβλητές, για την ταχύτητα με την οποία θα κινείτε ο controller, την δύναμη άλματος και την βαρύτητα που θα ασκείτε πάνω του, καθώς η Move() σε αντίθεση με την SimpeMove() δεν την εφαρμόζει αυτόματα.

```
var speed : float = 30.0;

var jumpSpeed : float = 50.0;

var gravity : float = 55.0;
```

Στην συνέχεια δημιουργούμε ένα τρισδιάστατο vector, ο οποίος θα χρησιμοποιηθεί για την μετακίνηση του controller και τον αρχικοποιούμε με τιμή (0,0,0) και επίσης δηλώνουμε τον CharacterController Component στην μεταβλητή heroController, έτσι ώστε να είναι πιο εύκολα προσβάσιμος.

```
private var moveDirection : Vector3 = Vector3.zero;

var heroController : CharacterController = GetComponent(CharacterController);
```

Μέσα στην Update τώρα κάνουμε έλεγχο αν ο heroController είναι στο έδαφος, δηλαδή αν επιστρέφει true στο CollisionFlags.Below. Στην περίπτωση που είναι true επιτρέπουμε την μετακίνηση του και συγκεκριμένα, στον κάθετο άξονα.

```
if(heroController.isGrounded){

    if(Input.GetAxis("Vertical") > .2 && !Input.GetButton("Action")){

        if(!Leonidas_SearchInteractPoint.InteractingWithObject)
        {
            moveDirection = Vector3(0,0, Input.GetAxis("Vertical"));
            moveDirection = transform.TransformDirection(moveDirection);
            moveDirection *= boostIncr * speed;
        }

        animation["run"].speed = 1;
        if(SelectWeaponLeftHand.bowActive && !ShiftRun && !Leonidas_SwimControll.CanSwim){
        if(Leonidas_SwimControll.CanSwim){
        if(ShiftRun && !Leonidas_SwimControll.CanSwim)
        {
        if(!ShiftRun && !SelectWeaponLeftHand.bowActive && !Leonidas_SwimControll.CanSwim)
        {
            boostIncr = 1;
            animation.CrossFade("run");
        }
    }


    else if(Input.GetAxis("Vertical") < -.2 && !Input.GetButton("Action"))
    {

        if(!Leonidas_SearchInteractPoint.InteractingWithObject)
        {
            moveDirection = Vector3(0,0, Input.GetAxis("Vertical"));
            moveDirection = transform.TransformDirection(moveDirection);
            moveDirection *= boostIncr * speed;
        }

        if(Leonidas_SwimControll.CanSwim){
        else{
            animation["run"].speed =-0.5;
            animation.CrossFade("walk_backwards");
            boostIncr = 0.5;
        }
    }

    else if(!Input.GetAxis("Vertical")){
    animation.Stop("run");
    animation.Stop("walk_backwards");
    animation.Stop("shift_run");
    animation.Stop("run_with_bow");
    animation.Stop("swim");
    //transform.Translate(Vector3.forward * 15 * Time.deltaTime, Space.Self);
    //animation.Play("stop_run");
    moveDirection = Vector3(0,0,0);
    moveDirection = transform.TransformDirection(moveDirection);
    moveDirection *= boostIncr * speed;
    }
```

Επίσης ελέγχει εάν έχει πατηθεί πλήκτρο για τον οριζόντιο άξονα και περιστρέφει τον controller κατάλληλα.

```
if(Input.GetAxis("Horizontal") && !Input.GetAxis("Vertical") && !Leonidas_SearchInteractPoint.InteractingWithObject){

    if(Input.GetAxis("Horizontal") < .1){
        //animation.Play("turn_left");
        //transform.Translate(Vector3.left * 15 * Time.deltaTime, Space.Self);
        //transform.Translate(Vector3.up * 5 * Time.deltaTime, Space.Self);
        transform.eulerAngles.y += Input.GetAxis("Horizontal") * 2;
    }

    if(Input.GetAxis("Horizontal") > 0){
        //animation.Play("turn_right");
        //transform.Translate(Vector3.right * 15 * Time.deltaTime, Space.Self);
        //transform.Translate(Vector3.up * 5 * Time.deltaTime, Space.Self);
        transform.eulerAngles.y += Input.GetAxis("Horizontal") * 2;
    }

}
```

Ακόμα ελέγχει αν έχει πατηθεί το πλήκτρο Jump και εάν επιστρέψει true, εφαρμόζει στον άξονα y του Vector3 που δηλώσαμε την jumpSpeed.

```
if (Input.GetButton ("Jump")) {
    if(ShiftRun){
        moveDirection.y = jumpSpeed;
        animation.Play("run_jumpHi");
    }
    else{
        moveDirection.y = jumpSpeed;
        animation.Play("jumpHi");
    }
}
```

Τέλος εφαρμόζει την βαρύτητα αλλά και την μετακίνηση του Controller με βάση την τιμή του Vector3 moveDirection.

```
if(!Leonidas_SearchInteractPoint.InteractingWithObject){
    transform.eulerAngles.y += Input.GetAxis("Horizontal") * 2;
}
moveDirection.y -= gravity * Time.deltaTime*2;
heroController.Move(moveDirection * Time.deltaTime);
}
```

4.2.5.5 Ο Input Manager και παράδειγμα προγραμματισμού των πλήκτρων

Τα πλήκτρα που είναι υπεύθυνα για τις κινήσεις του χαρακτήρα έχουν ρυθμιστεί μέσα στον Input Manager, με συγκεκριμένα ονόματα, τα οποία και καλούμε στο script μας. Ο προγραμματισμός των πλήκτρων μπορεί να γίνει είτε με την Input.GetKey(name : String) που παίρνει σαν είσοδο το όνομα του πλήκτρου, είτε με την Input.GetButton(buttonName :

String) που παίρνει τα ονόματα από τον Input Manager. Στο πτυχιακή αυτή επιλέχθηκε να γίνει χρήση της Input.GetButton για το πληκτρολόγιο και Input.GetMouseButton για το ποντίκι, αφού έχοντας δηλώσει τα επιθυμητά πλήκτρα στον Input Manager, οι συνάρτήσεις ψάχνουν για το όνομα σε αυτόν και όχι σε ποιο πλήκτρο αναφέρεται και έτσι μπορούμε να κάνουμε αλλαγές στα πλήκτρα χωρίς να χρειάζεται να επέμβουμε περαιτέρω στον κώδικα μας.

Παρακάτω δείχνουμε τον Input Manager με αρκετά από τα πλήκτρα που έχουμε δημιουργήσει εκεί.

Εικόνα 4. 16 Ο Input Manager

Μερικές γραμμές κώδικα που δείχνουν τον τρόπο που χρησιμοποιήθηκαν τα δηλωμένα πλήκτρα στον Input Manager για τις κινήσεις του χαρακτήρα μας.

```
if(Input.GetButtonDown("Fire1") && !InventoryMouseOver && SelectWeaponRightHand.swordActive){
    if(!animation["slash1"].enabled && !animation["slash2"].enabled && !animation["slash3"].enabled){
        attacking = true;
        ComboAttack();
    }
}


if(Input.GetButton("Fire2") /*&& heroController.isGrounded*/ && SelectWeaponLeftHand.bowActive)
{
    animation["attack_bow"].wrapMode = WrapMode.ClampForever;
    speed = 0;
    animation.CrossFade("attack_bow");
    if(Input.GetButtonDown("Fire1") && !Input.GetAxis("Vertical")){
        animation.Play("arrow_left");
        TakeArrowAndAim();
    }

}

if(Input.GetButtonUp("Fire2")){
    animation["attack_bow"].wrapMode = WrapMode.Once;
}
```

Η Input.GetButton επιστρέφει true για όσο έχουμε πατημένο το επιλεγμένο πλήκτρο που παίρνει σαν είσοδο, ενώ με την Input.GetButtonDown επιστρέφει true στο frame στο οποίο πατάμε το πλήκτρο. Τέλος για να ελέγξουμε αν αφήσαμε κάποιο πατημένο πλήκτρο χρησιμοποιούμε την Input.GetButtunUp, που επιστρέφει true στο πρώτο frame στο οποίο αφήνετε το πλήκτρο.

4.2.5.6 Χρήση της κλάσης GUI, παρουσίαση και επεξήγηση του Health script

Ένα αρκετά μεγάλο και σημαντικό μέρος της πτυχιακής ήταν η δημιουργία των διαλόγων, του συστήματος αποστολών, του Inventory, της γραφικής αναπαράστασης των στατιστικών του χαρακτήρα, όπως ζωή, πόντοι, κτλ. Όλα αυτά δημιουργήθηκαν μέσω της κλάσης GUI του Unity. Το GUI (Graphical User Interface) στο Unity αναφέρεται ως UnityGUI και μας επιτρέπει να δημιουργήσουμε, τοποθετήσουμε και να ορίσουμε τα δικά μας GUI controls άμεσα και γρήγορα μέσα από το ίδιο script. Ένα απλό παράδειγμα χρήσης της κλάσης GUI, φαίνεται παρακάτω μέσα από την δημιουργία του script ζωής του χαρακτήρα μας.

Αρχικά δημιουργούμε τις int μεταβλητές μας, MaxHealth και health και την boolean alive. Η MaxHealth ορίζει την μέγιστη τιμή της health, καθώς η health μεταβάλλεται κατά την διάρκεια του παιχνιδιού. Επίσης δηλώνουμε ένα δυσδιάστατο vector για την θέση του label που θα δημιουργήσουμε στην OnGUI() και θα τυπώνεται η μεταβλητή health, δηλαδή η ζωή μας. Ακόμα δημιουργούμε ένα GUISkin μέσα από το Assets→Create→GUI Skin και το δηλώνουμε στην HealthGUISkin μεταβλητή. Το GUI Skin χρησιμοποιείτε για την δημιουργία ενός skin με ρυθμίσεις για την γραμματοσειρά, το μέγεθος, το χρώμα, κτλ της health μεταβλητής που θα τυπώνεται στην οθόνη μας.

```
@script ExecuteInEditMode()
static var MaxHealth : int = 100;
static var health : int = MaxHealth;
static var alive : boolean = true;

var healthPos = Vector2(0,0);
var HealthGUISkin : GUISkin;
```

Μέσα στην OnGUI function δηλώνουμε το HealthGUISkin στο GUI.skin ώστε να χρησιμοποιήσουμε αυτό που θα δηλώσουμε μέσα από τον Inspector και επιλέγουμε χρώμα. Επίσης κάνουμε έλεγχο στην Boolean μεταβλητή alive, τυπώνοντας την μεταβλητή health μέσα στο string της GUI.Label, αν είναι true, ή την λέξη Dead αν δεν είναι.

```
function OnGUI(){

    GUI.skin = HealthGUISkin;

    GUI.DrawTexture(Rect(0,0,300,150), healthTexture);

    GUI.color=Color.white;

    if(alive)
    {
        GUI.Label(Rect(healthPos.x,healthPos.y,250,100), "" + health.ToString());
    }
    else if(!alive)
    {
        GUI.Label(Rect(healthPos.x,healthPos.y,250,100), "Dead");
    }

}
```

Με την GUI.Label μπορούμε να δημιουργήσουμε και να εμφανίσουμε στην οθόνη ένα κείμενο ή μια εικόνα, ενώ οι τρόποι σύνταξης της Label είναι οι εξής:

static function Label (position : Rect, text : String) : **void**
static function Label (position : Rect, image : Texture) : **void**
static function Label (position : Rect, content : GUIContent) : **void**
static function Label (position : Rect, text : String, style : GUIStyle) : **void**
static function Label (position : Rect, image : Texture, style : GUIStyle) : **void**
static function Label (position : Rect, content : GUIContent, style : GUIStyle) : **void**

Στην παρακάτω γραμμή κώδικα το healthPos.x και healthPos.y είναι οι x,y συντεταγμένες του Vector2 που δηλώσαμε στην αρχή και έχουμε ρυθμίσει στον Inspector.

Στην συνέχεια επιλέγουμε το μέγεθος του πλαισίου (250, 100), μέσα στο οποίο θα τυπώσει η συνάρτηση την τιμή της health μέσω της health.ToString().

GUI.Label(Rect(healthPos.x, healthPos.y,250,100), “” + health.ToString());

Όπως είπαμε και παραπάνω η GUI function τρέχει πιο συχνά και από την Update, που σημαίνει ότι μπορεί να γίνει έλεγχος για την τιμή της ζωής, όμως παρόλα αυτά για λόγους επιδόσεων θα χρησιμοποιήσουμε την Update, αφού μας αρκεί έλεγχος μια φορά ανά frame. Μέσα στην Update θα γίνεται έλεγχος της τιμής της health για την αλλαγή του Texture που θα χρησιμοποιήσουμε και κατά συνέπεια της alive, ώστε να αλλάξει το Sting του Label σε περίπτωση που η health γίνει μικρότερη ή ίση με το μηδέν. Μεσα στην OnGUI() είδαμε πως δηλώσαμε στην GUI.DrawTexture() να εμφανίζεται το healthTexture, το οποίο θα αλλάζει αναλόγως την τιμή της health, αφού μέσα στην Update για κάθε 10 μονάδες της health, θα παίρνει ένα διαφορετικό Texture μέσα σε ένα Texture2D πίνακα που έχουμε γεμίσει μέσω του Inspector.

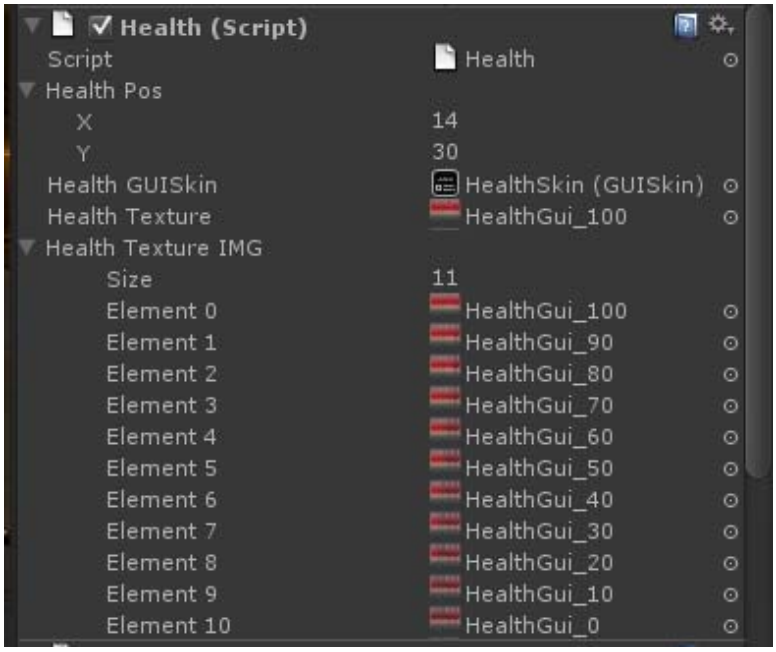
```
var healthTexture : Texture2D;
var healthTextureIMG : Texture2D[];

function Update(){

    if(health == MaxHealth){
        healthTexture = healthTextureIMG[0];
    }
    else if(health <100 && health > 90){
        healthTexture = healthTextureIMG[1];
    }
    else if(health <90 && health > 80){
        healthTexture = healthTextureIMG[2];
    }
    else if(health <80 && health > 70){
        healthTexture = healthTextureIMG[3];
    }
    else if(health <70 && health > 60){
        healthTexture = healthTextureIMG[4];
    }
    else if(health <60 && health > 50){
        healthTexture = healthTextureIMG[5];
    }
    else if(health <50 && health > 40){
        healthTexture = healthTextureIMG[6];
    }
    else if(health <40 && health > 30){
        healthTexture = healthTextureIMG[7];
    }
    else if(health <30 && health > 20){
        healthTexture = healthTextureIMG[8];
    }
    else if(health <20 && health > 0){
        healthTexture = healthTextureIMG[9];
    }
    else if (health <= 0  && alive){
        healthTexture = healthTextureIMG[10];
        PlayDead();
    }

}
```

Μέσα στον Inspector επιλέγουμε για το πίνακα healthTextureIMG τον αριθμό 11 στο size και δηλώνουμε τις .jpg εικόνες με τα στάδια της health bar που έχουμε δημιουργήσει στο Photoshop. Οι ρυθμίσεις των public μεταβλητών του συγκεκριμένου script φαίνονται στην παρακάτω εικόνα.



Εικόνα 4. 17 Το Health script του χαρακτήρα, όπως φαίνεται στον Inspector

Το health script έχει επίσης μια function για την αφαίρεση πόντων ζωής και μια για την αύξηση τους. Και οι δύο αυτές function καλούνται μέσα από scripts άλλων αντικειμένων.

```
function LeonidasIsBleeding (dmg : int){

    health -= dmg;
    BroadcastMessage("HitAnim");

  }


function AddHealth(addHealthPot : int){

    if (health < MaxHealth){
        health += addHealthPot;
    }

    if (health >= MaxHealth){
        health = MaxHealth;
    }

  }
```

Τέλος στο script αυτό υπάρχει μια ακόμα function, η οποία εκτελείτε όταν η τιμή της μεταβλητής health γίνει μηδέν ή αρνητική. Σε αυτήν γίνεται false η alive, ενημερώνοντας αμέσως την GUI.Label να αλλάξει το string, παίζει το κατάλληλο animation του χαρακτήρα και αφού περιμένει όσο χρόνο χρειάζεται για να ολοκληρωθεί το animation[“dead”], κάνει μέσω της Time.timeScale pause και στην συνέχεια φορτώνει μέσω της Application.LoadLevel μια άλλη σκηνή.

```
function PlayDead(){

    alive = false;
    animation.Play("dead");
    yield new WaitForSeconds(animation["dead"].clip.length);
    Time.timeScale = 0;
    Application.LoadLevel("DeadLevel");

}
```

Με παρόμοιο τρόπο δουλεύει και το σύστημα των πόντων και των άλλων στατιστικών που εμφανίζονται στην οθόνη. Συνολικά ο κύριος χαρακτήρας του παιχνιδιού έχει 19 scripts πάνω του και στα αντικείμενα που κουβαλά. Ένα εξίσου βασικό script είναι αυτό που είναι υπεύθυνο για την εφαρμογή ζημιάς στα αντικείμενα και τους εχθρούς.

Σε κάθε prefab εχθρού έχουμε ορίσει το tag ως Enemy, και έτσι κάνοντας Physics.Linecast μεταξύ δύο σημείων, αν βρει εχθρό στέλνουμε μήνυμα στην function μείωσης ζωής του στο κατάλληλο script που έχουμε δημιουργήσει, εφαρμόζοντας ζημιά. Ο εντοπισμός του εχθρού για την αποστολή του μηνύματος γίνεται μέσα από έλεγχο του Collider του component Character Controller που βρίσκουμε μέσω Linecast και του tag του αντικειμένου που βρίσκετε μεταξύ των σημείων. Η function αυτή συντάσσεται ως εξής.

static function Linecast (**start :** Vector3, **end :** Vector3, layerMask : **int** = kDefaultRaycastLayers) : **boolean**

Τα start και end είναι τα σημεία στον χώρου που υποδηλώνουν την αρχή και το τέλος της ευθείας στην οποία θα ψάχνει για colliders, ενώ το layerMask, χρησιμοποιείται για να αγνοήσουμε αντικείμενα που βρίσκονται σε κάποιο layer.

4.2.5.7 Παρουσίαση και επεξήγηση του AI script των εχθρών

Στον παρακάτω εικόνα φαίνεται ο τρόπος με τον οποίον χρησιμοποιήθηκε για τον συγκεκριμένο σκοπό που μιλήσαμε παραπάνω. Η διαδικασία γίνεται μέσα στην Update ώστε να κάνει Linecast για όσο χρειαζόμαστε σε κάθε frame. Αρχικά δηλώνουμε μια local μεταβλητή την hitObj, η οποία θα παρέχει πληροφορίες για τον collider, όταν η συνάρτηση επιστρέφει true, και αφού κάνουμε όσα Plysics.Linecast χρειαζόμαστε, ψάχνουμε για collider το οποίο είναι σε αντικείμενο με tag Enemy και αφού βρεθεί, τρέχουμε την function Bleed του script EnemyHealth.

```
function Update () {
    var hitObj : RaycastHit;
    if ((Physics.Linecast (transform.position, target1.position, hitObj)) ||
        (Physics.Linecast (transform.position, target2.position, hitObj)) ||
        (Physics.Linecast (transform.position, target3.position, hitObj)) ||
        (Physics.Linecast (transform.position, target4.position, hitObj)) ||
        (Physics.Linecast (transform.position, target5.position, hitObj)) ||
        (Physics.Linecast (transform.position, target6.position, hitObj))) {

        Debug.DrawLine(transform.position, target1.position, Color.red);
        Debug.DrawLine(transform.position, target2.position, Color.red);
        Debug.DrawLine(transform.position, target3.position, Color.red);
        Debug.DrawLine(transform.position, target4.position, Color.red);
        Debug.DrawLine(transform.position, target5.position, Color.red);
        Debug.DrawLine(transform.position, target6.position, Color.red);
        /*

        */
        if(hitObj.collider.gameObject.tag=="Enemy"){
            currentEnemy = hitObj.collider.gameObject.transform;

            Debug.Log("name of enemy is: "+currentEnemy);

            Bloodclone = Instantiate(BloodEffect,transform.position,transform.rotation);

            Destroy(Bloodclone, 0.5);
            currentEnemy.GetComponent("EnemyHealth").Bleed(15);

        }
```

Στους εχθρούς έχουμε δημιουργήσει ένα script για τον έλεγχο της κατάστασης τους, όπως για παράδειγμα, για την κίνηση τους στον χάρτη, για τον εντοπισμό και περιστροφή προς τον παίκτη, για την επίθεση κτλ. Στο script αυτό, αρχικά το κάθε instance, ψάχνει στην σκηνή να βρει τα waypoints που του έχουμε ορίσει, βρίσκει το κοντινότερο και μετακινείτε προς αυτό, ελέγχοντας παράλληλα αν βρίσκετε ο παίκτης μας κοντά. Η διαδικασία αυτή φαίνεται στις παρακάτω γραμμές κώδικα.

```
function Patrol () {
    var wPointWayPoint = AutoWayPoint_Zombie_Frouros.FindClosest(transform.position);
    while (true) {
        var waypointPosition = wPointWayPoint.transform.position;
        // An eimaiste konta se waypoint - pigene sto epomeno
        if (Vector3.Distance(waypointPosition, transform.position) < pickNextWaypointDistance){
            wPointWayPoint = PickNextWaypoint (wPointWayPoint);
        }
        if (CanSeeTarget()){
            yield StartCoroutine("AttackPlayer");
        }
        // Pigene pros to meros toy target
        MoveTowards(waypointPosition);
        yield;
    }
}
```

Η function CanSeeTarget κάνει Physics.Linecast και βρίσκοντας τον target, δηλαδή τον παίκτη μας, επιστρέφει true, κάνοντας yield στην Patrol() και ξεκινάει την function AttackPlayer.

```
function CanSeeTarget () : boolean {

    if(transform != null && target != null && !animation["stunned"].enabled){
        if (Vector3.Distance(transform.position, target.position) > attackRange){
            return false;
        }
        var hit : RaycastHit;
        if (Physics.Linecast (transform.position, target.position, hit)){
            return hit.transform == target;
        }
        return false;
    }

}
```

```
function AttackPlayer () {

    if(transform != null && target != null && !animation["stunned"].enabled){
        var lastVisiblePlayerPosition = target.position;
        while (true) {
            if (CanSeeTarget ()) {
                // an o target einai nekros - stamata
                if (target == null){
                    return;
                }
                // An o target einai makria-stamata
                var distance = Vector3.Distance(transform.position, target.position);
                if (distance > shootRange * 3){
                    return;
                }
                lastVisiblePlayerPosition = target.position;
                if (distance > dontComeCloserRange){
                    MoveTowards (lastVisiblePlayerPosition);
                }
                else{
                    RotateTowards(lastVisiblePlayerPosition);
                }
                var forward = transform.TransformDirection(Vector3.forward*2);
                var targetDirection = lastVisiblePlayerPosition - transform.position;
                targetDirection.y = 0;
                animation.CrossFade("walk");
                speed = RunSpeed;
                if (speed > 10){
                    speed = 10;
                }
                var angle = Vector3.Angle(targetDirection, forward);
                // an o target einai sto shootRange - xtypa
                if (distance < shootRange && angle < shootAngle){
                    yield StartCoroutine("Shoot");
                }
            }
            else {
                yield StartCoroutine("SearchPlayer", lastVisiblePlayerPosition);
                // an o target dn fainetai- einai makria- stamata
                if (!CanSeeTarget ()){
                    return;
                }
            }
            yield;
        }
    }

}
```

Στην AttackPlayer κοιτάει την απόσταση μεταξύ της θέσης του transform και του παίκτη, σε περίπτωση που είναι αρκετά μακριά σταματά και επιστρέφει στην Patrol(), αλλιώς πλησιάζει τον παίκτη και ξεκινάει την function Shoot() η οποία παίζει το κατάλληλο animation και στέλνει μέσω BroadcastMessage σε άλλο script μήνυμα ενεργοποίησης function για την εφαρμογή ζημιάς.

```
function Shoot () {
    if(!target.animation["defence"].enabled && !animation["stunned"].enabled){
        var AttackMethod = Random.Range(0,2);
        if(AttackMethod == 0 && !animation["idle1"].enabled){
            animation.CrossFade("attack1");
            yield WaitForSeconds(delayShootTime);
            BroadcastMessage("Hitting");
            yield WaitForSeconds(animation["attack1"].length - delayShootTime);
        }
        else if(AttackMethod == 1 && !animation["attack1"].enabled){
            animation.CrossFade("idle1");
            yield WaitForSeconds(animation["idle1"].length-1);
        }
    }
}
```

Επίσης η function σε περίπτωση που η CanSeeTarget() επιστρέψει false, μετακινεί το transform στην θέση που είχε εντοπιστεί τελευταία φορά ο target και ψάχνει ξανά. Επιστρέφοντας false και αυτή, ο εχθρός μετακινείται προς το επόμενο κοντινότερο waypoint στην σκηνή και συνεχίζει την Patrol(); Η μετακίνηση γίνεται μέσω της MoveTowards()

```
function MoveTowards (position : Vector3) {

    if(transform != null && target != null && !animation["stunned"].enabled){
        var direction = position - transform.position;
        direction.y = 0;
        if (direction.magnitude < 0.5) {
            return;
        }
        // Rotate pros ton target
        transform.rotation = Quaternion.Slerp (transform.rotation,
        Quaternion.LookRotation(direction), rotationSpeed * Time.deltaTime);
        transform.eulerAngles = Vector3(0, transform.eulerAngles.y, 0);
        // slow dpwn speed
        var forward = transform.TransformDirection(Vector3.forward);
        var speedModifier = Vector3.Dot(forward, direction.normalized);
        speedModifier = Mathf.Clamp01(speedModifier);
        // Move--
        speed = NormalSpeed;
        direction = forward * speed * speedModifier;
        GetComponent (CharacterController).SimpleMove(direction);
        animation.CrossFade("walk");
    }

}
```

Παρακάτω φαίνεται επίσης και η RotateTowards() που είναι υπεύθυνη για την στροφή του transform προς το την τελευταία γνωστή θέση του target.

```
function RotateTowards (position : Vector3) {

    if(transform != null && target != null && !animation["stunned"].enabled){

        var direction = position - transform.position;
        direction.y = 0;
        if (direction.magnitude < 0.1){
            return;
        }
        // Rotate pros ton target
        transform.rotation = Quaternion.Slerp (transform.rotation,
        Quaternion.LookRotation(direction), rotationSpeed * Time.deltaTime);
        transform.eulerAngles = Vector3(0, transform.eulerAngles.y, 0);
    }

 }
```

Η επιλογή των waypoints μεταξύ των οποίων μετακινούνται οι συγκεκριμένοι εχθροί, γίνεται με τον παρακάτω κώδικα.

```
function PickNextWaypoint (currentWaypoint : AutoWayPoint_Zombie_Frouros) {

    // H dieythinsi poy perpatame
    var forward = transform.TransformDirection(Vector3.forward);
    var best = currentWaypoint;
    var bestDot = -10.0;
    for (var wPoint : AutoWayPoint_Zombie_Frouros in currentWaypoint.connected) {
        var direction = Vector3.Normalize(wPoint.transform.position - transform.position);
        var dot = Vector3.Dot(direction, forward);
        if (dot > bestDot && wPoint != currentWaypoint) {
            bestDot = dot;
            best = wPoint;
        }
    }

    return best;
 }
```

Παρακάτω δείχνουμε τον τρόπο με τον οποίον γεμίζει το waypoint array με τα αντικείμενα που αποτελούν τα σημεία μεταξύ των οποίων μπορεί να μετακινηθεί το transform και στα οποία έχουμε εφαρμόσει τον κώδικα AutoWaypoint του συγκεκριμένου είδους prefab, καθώς και την δημιουργία των ενώσεων, κατά την οποίο γεμίζει το array connected του κάθε waypoint.

```
function RebuildWaypointList () {
    var objects : Object[] = FindObjectsOfType(AutoWayPoint_Zombie_Frouros);
    waypoints = Array(objects);

    for (var point : AutoWayPoint_Zombie_Frouros in waypoints) {
        point.RecalculateConnectedWaypoints();
    }
 }

function RecalculateConnectedWaypoints (){
    connected = Array();

    for (var other : AutoWayPoint_Zombie_Frouros in waypoints) {
        // min to enwseis me ton eayto mas
        if (other == this){
            continue;
        }
        // An den yparxei empodio metaksy twn waypoints - enwse ta
        if (!Physics.CheckCapsule(transform.position, other.transform.position, kLineOfSightCapsuleRadius)) {
            connected.Add(other);
        }
    }
 }
```

Τέλος παρακάτω δείχνουμε την function, που είναι υπεύθυνη για την επιλογή του επόμενου σημείου στο οποίο θα μετακινηθεί ο εχθρός, δηλαδή για την επιλογή της κοντινότερης διαδρομής.

```
static function FindClosest (pos : Vector3) : AutoWayPoint_Zombie_Frouros {
    // oso pio konta vriskontai oi 2 vectors, toso megalytero dot paragoun.
    var closest : AutoWayPoint_Zombie_Frouros;
    var closestDistance = 100000.0;
    for (var cur : AutoWayPoint_Zombie_Frouros in waypoints) {
        var distance = Vector3.Distance(cur.transform.position, pos);
        if (distance < closestDistance)
        {
            closestDistance = distance;
            closest = cur;
        }
    }

    return closest;
 }
```

4.2.5.8 Παρουσίαση και επεξήγηση των βασικών σημείων της διαδικασίας save και load του παιχνιδιού

Στο παιχνίδι που αναπτύξαμε σε αυτήν την πτυχιακή, αναπτύχθηκαν επίσης σύστημα Inventory, όπου μπορούμε να συλλέγουμε και να αποθηκεύουμε αντικείμενα, ένα σύστημα πώλησης αντικειμένων, ένα σύστημα αποστολών τα οποία ακολουθεί ο χαρακτήρας για την εξέλιξη του παιχνιδιού, και ένα σύστημα Save-Load, το οποίο αλληλεπιδρά λίγο πολύ με όλα τα παραπάνω αφού αποθηκεύονται, εκτός από τα στατιστικά του χαρακτήρα, το level στο οποίο βρίσκεται και η θέση του στον χάρτη, τα αντικείμενα του inventory, η πρόοδος των αποστολών και τέλος η κατάσταση αρκετών αντικειμένων, ώστε να φορτώνουμε το κάθε level όσο πιο κοντά γίνεται στην τελευταία του κατάσταση. Παρακάτω θα δείξουμε συνοπτικά μέσα από ένα παράδειγμα τον τρόπο που γίνεται η αποθήκευση και η μετατροπή των στοιχείων σε xml και το ανάποδο.

Αρχικά δημιουργούμε ένα νέο script και αφού εισάγουμε τις απαραίτητες βιβλιοθήκες, δημιουργούμε μια κλάση η οποία θα χρησιμοποιείτε για να δηλώσουμε όλες εκείνες τις μεταβλητές που θέλουμε να αποθηκεύουμε στο xml. Στις παρακάτω γραμμές κώδικα θα δείξουμε την διαδικασία που ακολουθήσαμε για την αποθήκευση της θέσης του χαρακτήρα, του level που βρίσκεται, αλλά και την τιμή της ζωή του.

```
import System.Collections;
import System.Xml;
import System.Xml.Serialization;
import System.IO;
import System.Text;

class GameData
{
    var x : float;
    var y : float;
    var z : float;
    var name : String;
    var hp : int;
    var gameLevelName : String;

}

class PlayerData{

  public var _iUser : GameData = new GameData();

}
```

Επίσης δηλώνουμε τις local μεταβλητές που θα χρησιμοποιηθούν για την προσωρινή αποθήκευση των στατιστικών μας.

```
// Dhlwnoyme tis local metavlites
private var _Save : Rect;
private var _Load : Rect;
private var _FileLocation : String;
private var _FileName : String = "SaveData.xml";
var _Player : GameObject;
var _PlayerName : String = "Andreas Stamoulias";
var _PlayerHealthPoints : int;
var _PlayerGameLevel : String;
private var myData : PlayerData;
private var _data : String;
private var PlayerPosition : Vector3;
```

Στην συνέχεια στην Start function δηλώνουμε το μέγεθος και την θέση των Save και Load button, την θέση του φακέλου Assets, που αργότερα θα αποτελεί την θέση του φακέλου του παιχνιδιού, που θα χρησιμοποιηθεί για την θέση αποθήκευσης του xml και τέλος δημιουργούμε ένα νέο instance του PlayerData με τις πληροφορίες του GameData.

```
function Start () {

    _Save=new Rect(10,250,100,20);
    _Load=new Rect(10,270,100,20);

    _FileLocation=Application.dataPath;
    myData=new PlayerData();

}
```

Το κουμπί Save που δημιουργήσαμε παραπάνω στην θέση (10, 250), το χρησιμοποιούμε μέσω της OnGUI function όπως φαίνεται παρακάτω.

```
function OnGUI(){
    if (GUI.Button(_Save,"Save")) {

        var _PlayerHealthPoints = Health.health;
        var _PlayerGameLevel = Application.loadedLevelName;

        myData._iUser.x = _Player.transform.position.x;
        myData._iUser.y = _Player.transform.position.y;
        myData._iUser.z = _Player.transform.position.z;
        myData._iUser.name = _PlayerName;
        myData._iUser.hp = _PlayerHealthPoints;
        myData._iUser.gameLevelName = _PlayerGameLevel;


        _data = SerializeObject(myData);

        CreateXML();
        Debug.Log(_data);
    }
}
```

Πατώντας το κουμπί Save δηλώνουμε στην _PlayerHealthPoints την ζωή του παίκτη, η οποία είναι δηλωμένη ως static και στην _PlayerGameData το όνομα του level στο οποίο βρισκόμαστε. Στην συνέχεια μεταφέρουμε την τιμή και την συμβολοσειρά στις μεταβλητές της κλάσης GameData, καθώς και την θέση του χαρακτήρα και κάνουμε serialize χρησιμοποιώντας τον παρακάτω κώδικα.

```
function SerializeObject(pObject : Object){

    var XmlizedString : String = null;
    var memoryStream : MemoryStream = new MemoryStream();
    var xs : XmlSerializer = new XmlSerializer(typeof(PlayerData));
    var xmlTextWriter : XmlTextWriter  = new XmlTextWriter(memoryStream, Encoding.UTF8);
    xs.Serialize(xmlTextWriter, pObject);
    memoryStream = xmlTextWriter.BaseStream; // (MemoryStream)
    XmlizedString = UTF8ByteArrayToString(memoryStream.ToArray());
    return XmlizedString;

}


function UTF8ByteArrayToString(characters : byte[] ){

    var encoding : UTF8Encoding = new UTF8Encoding();
    var constructedString : String = encoding.GetString(characters);
    return (constructedString);

}
```

Αφού κάνει serialize τα PlayerData objects του myData, δημιουργεί ένα έγγραφο xml στην θέση και με το όνομα που του δηλώσαμε ή αντικαθιστά διαγράφοντας το προηγούμενο με το ίδιο όνομα.

```
function CreateXML(){

    var writer : StreamWriter;
    var t : FileInfo = new FileInfo(_FileLocation+"/"+UserCollector.CurrentUserName+"_"+ _FileName);
    if(!t.Exists){
        writer = t.CreateText();
    }
    else{
        t.Delete();
        writer = t.CreateText();
    }
    writer.Write(_data);
    writer.Close();
    Debug.Log("File written.");

}
```

Όπως και στο Save, έτσι και με την Load στο παράδειγμα μας θα χρησιμοποιήσουμε την OnGUI function. Πατώντας Load βρίσκουμε τον παίκτη μας σε περίπτωση που δεν τον έχουμε δηλώσει στον Inspector, ψάχνοντας για gameObject με tag Player, και στην συνέχεια τρέχουμε την function LoadXML.

```
function LoadXML(){

    var r : StreamReader = File.OpenText(_FileLocation+"/"+UserCollector.CurrentUserName+"_"+ _FileName);
    var _info : String = r.ReadToEnd();
    r.Close();
    _data =_info;
    Debug.Log("File Read.");

}
```

Η LoadXML διαβάζει το έγγραφο _FileName.xml στην θέση _FileLocation, περνάει στο string _info ολόκληρο το κείμενο που έχει γίνει Serialize και το αποθηκεύει στο _data, το οποίο ελέγχεται και στην περίπτωση που δεν είναι κενό, γίνεται deserialize, όπως φαίνεται παρακάτω.

```
function DeserializeObject(pXmlizedString : String){

    var xs : XmlSerializer = new XmlSerializer(typeof(PlayerData));
    var memoryStream : MemoryStream = new MemoryStream(StringToUTF8ByteArray(pXmlizedString));
    var xmlTextWriter : XmlTextWriter = new XmlTextWriter(memoryStream, Encoding.UTF8);
    return xs.Deserialize(memoryStream);

}


function StringToUTF8ByteArray(pXmlString : String){

    var encoding : UTF8Encoding = new UTF8Encoding();
    var byteArray : byte[] = encoding.GetBytes(pXmlString);
    return byteArray;

}
```

Στην συνέχεια και αφού τελειώσει η παραπάνω διαδικασία, φορτώνουμε μέσω του Application.LoadLevel το string με το όνομα του level, που είχαμε αποθηκεύσει στο xml, τους πόντους ζωής και την θέση του χαρακτήρα, η οποία χρησιμοποιείτε για την μεταφορά του θέτοντας το transform.position στην θέση του Vector3 που δημιουργήσαμε με τις x,y,z float τιμές που περάσαμε στο xml.

```
if (GUI.Button(_Load,"Load")) {

    if (_Player == null && GameObject.FindWithTag("Player"))
        _Player = GameObject.FindWithTag("Player");

    LoadXML();

    if(_data.ToString() != ""){
        myData = DeserializeObject(_data);

        Application.LoadLevel(myData._iUser.gameLevelName);
        PlayerPosition= Vector3(myData._iUser.x,myData._iUser.y,myData._iUser.z);
        _Player.transform.position=VPosition;
        Health.health = myData._iUser.hp;

        Debug.Log(myData._iUser.name);
        Debug.Log("x="+myData._iUser.x + " y="+myData._iUser.y + " z="+myData._iUser.z);
        Debug.Log("hp="+myData._iUser.hp);
        Debug.Log("current Game Level="+myData._iUser.gameLevelName);
    }
}
```

4.2.5.9 Παράδειγμα χρήσης εξειδικευμένων function της κλάσης GUI και παρουσίαση μέρους του Merchant script

Τέλος θα παρουσιάσουμε τους τρόπους δημιουργίας παραθύρων μέσω της OnGUI function, τον τρόπο ορισμού textures σε κουμπιά, την αλλαγή χρώματος και διαφάνειας, την δημιουργία groups GUI στοιχείων και την χρήση της GUILayout για την ευθυγράμμιση αυτών. Για να γίνουν τα παραπάνω πιο κατανοητά θα χρησιμοποιήσουμε κομμάτι του κώδικα που γράφτηκε για τον χαρακτήρα NPC Merchant που βρίσκεται στο παιχνίδι μας, ενώ παράλληλα θα δείχνουμε και το οπτικό αποτέλεσμα.

Αρχικά ελέγχουμε την απόσταση του χαρακτήρα από τον Merchant χρησιμοποιώντας την Vector3.Distance και εφόσον είμαστε στην επιθυμητή απόσταση που έχουμε ορίσει, πατώνταντας το πλήκτρο F, κάνουμε την μεταβλητή TalkToMerchant true και ενεργοποιούμε στην OnGUI function την εμφάνιση του αρχικού μενού και τον κουμπιών.

```
    var distance = Vector3.Distance(InteractTransform.transform.position, player.position);
    if(Input.GetButtonDown("Interact") && distance < InteractRange && !TalkToMerchant){

        Debug.Log("Talking to merchant...");
        TalkToMerchant = true;
    }

    if(distance > InteractRange && TalkToMerchant){

        Debug.Log("Leaving from merchant...");
        TalkToMerchant=false;
        exitMerchant = true;

    }
```

Έχοντας κάνει true την TalkToMerchant, εμφανίζουμε στην οθόνη το Texture2D που έχουμε δηλώσει στον Inspector στην MerchantTexture σε ένα πλαίσιο 400 x 250 στο σημείο 10 pixel δεξιά από την αρχή της οθόνης κατά πλάτος και 200 pixel χαμηλότερα από την αρχή της οθόνης στο ύψος.

```
function OnGUI () {

    if(TalkToMerchant){
        showMainMenu = true;
        LeonidasScriptFIX.InventoryMouseOver = true;
        GUI.DrawTexture (Rect (Screen.width/2-(Screen.width/2 - 10),
        Screen.height/2-(Screen.height/2-200),400,250), MerchantTexture);
    }
    else {
        LeonidasScriptFIX.InventoryMouseOver = false;
        showMainMenu = false;
        showQuests = false;
        showBuySell = false;
    }
```

Στην συνέχεια δημιουργούμε τα τρία κουμπιά “Buy-Sell”, “Quests” και “Exit” με χρήση της GUI.Button, στις θέσεις και με το μέγεθος που βλέπουμε παρακάτω, ενώ πατώντας πάνω σε ένα από αυτά τα κουμπιά αλλάζουμε την κατάσταση στις μεταβλητές showBuySell, showQuests και TalkToMerchant αντίστοιχα, με το οποίο οδηγούμαστε σε άλλες περιπτώσεις.

```
    if(showMainMenu){
        if(GUI.Button (Rect (Screen.width/2-(Screen.width/2 - 160),
        Screen.height/2-(Screen.height/2-260),100,25), "Buy-Sell")){
            showBuySell = true;
        }
        if(GUI.Button (Rect (Screen.width/2-(Screen.width/2 - 160),
        Screen.height/2-(Screen.height/2-320),100,25), "Quests")){
            showQuests = true;
        }
        if(GUI.Button (Rect (Screen.width/2-(Screen.width/2 - 160),
        Screen.height/2-(Screen.height/2-380),100,25), "Exit")){
            TalkToMerchant=false;
        }
    }
```

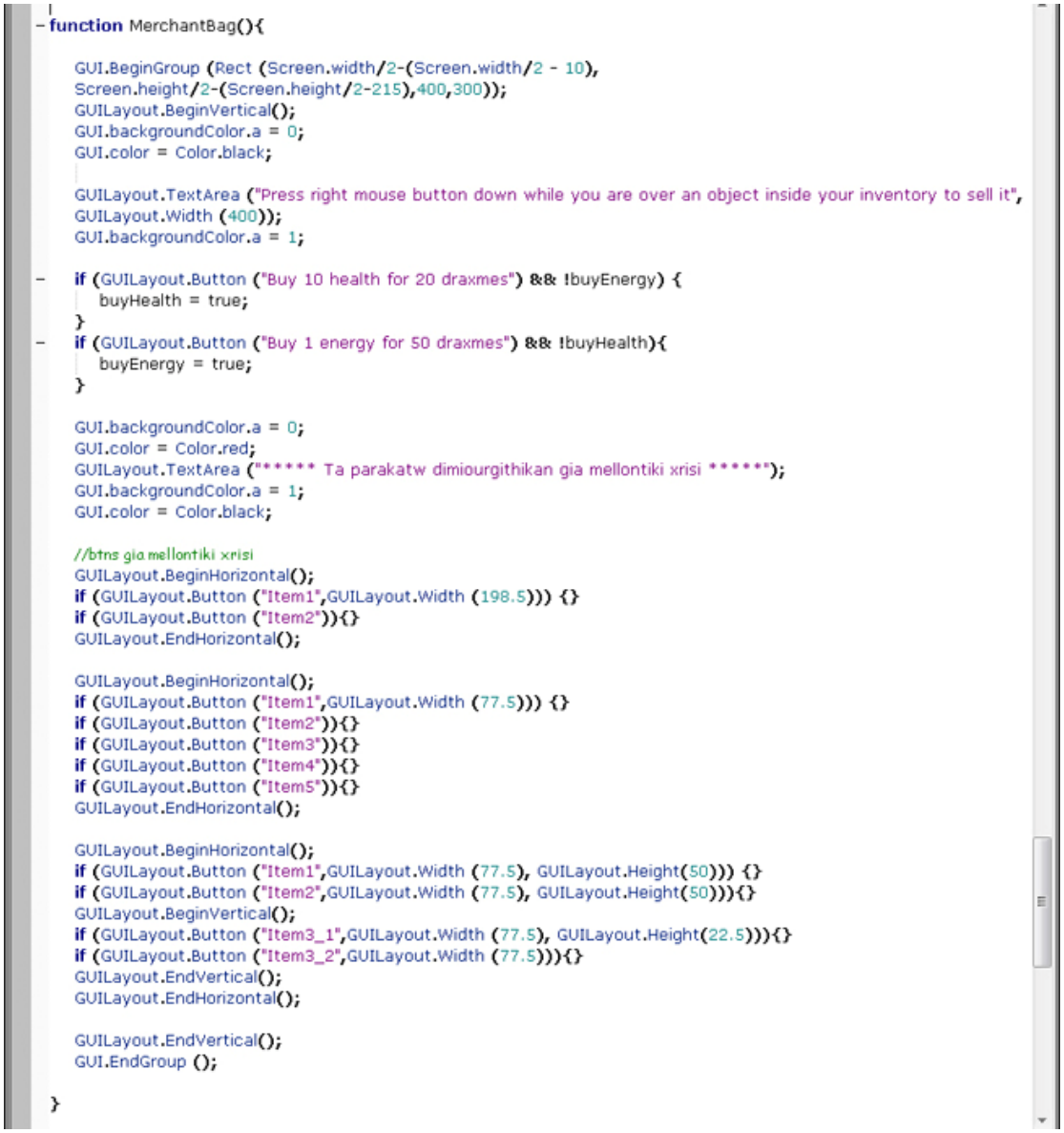
Εικόνα 4. 18 Το μενού που δημιουργήσαμε όπως εμφανίζεται μέσα στο Unity

Όπως παρατηρούμε και στην παραπάνω εικόνα, στην οθόνη μας εμφανίζεται το Texture2D που φτιάξαμε στο Photoshop, καθώς και τα τρία κουμπιά "Buy-Sell", "Quests", "Exit". Παρακάτω θα δείξουμε την λειτουργία του κουμπιού "Buy-Sell", αφού έχει γίνει χρήση της τεχνικής GUI.BeginGroup και GUILayout και παρουσιάζει ιδιαίτερο ενδιαφέρον.

```
if(showBuySell){
    showQuests = false;
    showMainMenu = false;
    GUI.DrawTexture (Rect (Screen.width/2-(Screen.width/2 - 10),
    Screen.height/2-(Screen.height/2-200),400,250), MerchantTexture);
    if (GUI.Button (Rect (Screen.width/2-(Screen.width/2 - 320),
    Screen.height/2-(Screen.height/2-400),50,25), "Back")) {
        showBuySell = false;
        showMainMenu = true;
    }
    GUI.backgroundColor.a = 0;
    CanBuySell = true;
    Inventory_PositionController.MakeInvObjectsSellable = true;
    MerchantBag();
}

if(!showBuySell){
    CanBuySell = false;
    showMainMenu = true;
    Inventory_PositionController.MakeInvObjectsSellable = false;
}
```

Έχοντας πατήσει το "Buy-Sell" κουμπί, κάναμε true την showBuySell, με την οποία δημιουργούμε το κουμπί "Back" για επιστροφή στο προηγούμενο μενού και καλούμε την MerchantBag function, στην οποία δημιουργούμε ένα νέο group από GUI elements και τα τακτοποιούμε κάθετα μέσω της GUILayout.BeginVertical.



```
function MerchantBag(){

    GUI.BeginGroup (Rect (Screen.width/2-(Screen.width/2 - 10),
    Screen.height/2-(Screen.height/2-215),400,300));
    GUILayout.BeginVertical();
    GUI.backgroundColor.a = 0;
    GUI.color = Color.black;

    GUILayout.TextArea ("Press right mouse button down while you are over an object inside your inventory to sell it",
    GUILayout.Width (400));
    GUI.backgroundColor.a = 1;

    if (GUILayout.Button ("Buy 10 health for 20 draxmes") && !buyEnergy) {
        buyHealth = true;
    }
    if (GUILayout.Button ("Buy 1 energy for 50 draxmes") && !buyHealth){
        buyEnergy = true;
    }

    GUI.backgroundColor.a = 0;
    GUI.color = Color.red;
    GUILayout.TextArea ("***** Ta parakatw dimiourgithikan gia mellontiki xrisi *****");
    GUI.backgroundColor.a = 1;
    GUI.color = Color.black;

    //btns gia mellontiki xrisi
    GUILayout.BeginHorizontal();
    if (GUILayout.Button ("Item1",GUILayout.Width (198.5))) {}
    if (GUILayout.Button ("Item2")){}
    GUILayout.EndHorizontal();

    GUILayout.BeginHorizontal();
    if (GUILayout.Button ("Item1",GUILayout.Width (77.5))) {}
    if (GUILayout.Button ("Item2")){}
    if (GUILayout.Button ("Item3")){}
    if (GUILayout.Button ("Item4")){}
    if (GUILayout.Button ("Item5")){}
    GUILayout.EndHorizontal();

    GUILayout.BeginHorizontal();
    if (GUILayout.Button ("Item1",GUILayout.Width (77.5), GUILayout.Height(50))) {}
    if (GUILayout.Button ("Item2",GUILayout.Width (77.5), GUILayout.Height(50))){}
    GUILayout.BeginVertical();
    if (GUILayout.Button ("Item3_1",GUILayout.Width (77.5), GUILayout.Height(22.5))){}
    if (GUILayout.Button ("Item3_2",GUILayout.Width (77.5))){}
    GUILayout.EndVertical();
    GUILayout.EndHorizontal();

    GUILayout.EndVertical();
    GUI.EndGroup ();

}
```

Στην function MerchantBag αρχικά κάνουμε GUI.BeginGroup και δημιουργούμε μια ομάδα από GUI elements. Σημαντικό είναι να αναφέρουμε ότι ξεκινώντας ένα group το σύστημα συντεταγμένων ορίζεται στο (0,0), δηλαδή στο πάνω αριστερό μέρος της οθόνης. Επίσης τα groups μπορούν να είναι nested μέσα σε άλλα group, έχοντας ως αρχή στο σύστημα συντεταγμένων το πάνω αριστερό μέρος του προηγούμενου group.

Στην συνέχεια καλούμε την GUILayout.BeginVertical, με την οποία στοιχίζουμε τα GUI Control κάθετα. Επίσης σημαντικό είναι να αναφέρουμε ότι πλέον για να δημιουργήσουμε ένα control, κάνουμε χρήση της GUILayout. Δηλώνοντας το πλάτος του πρώτου στοιχείου, αυτόματα το GUILayout σε κάθετο control group, ορίζει και τα υπόλοιπα στο ίδιο μέγεθος, ενώ σε οριζόντιο control group ορίζοντας το ύψος του πρώτο στοιχείο, θέτει αυτόματα και για τα υπόλοιπα το ύψος αυτό. Σε ένα group έχουμε την δυνατότητα

αρκετές μικρότερες ομάδες από controls δημιουργώντας νέα GUILayout, τα οποία πλέον θα συμπεριφέροντε σαν controls και θα έχουν πλάτος ίσο με το πρώτο στοιχείο του group.

Ακόμα στις παραπάνω γραμμές κώδικα φαίνεται η χρήση του GUI.color, που είναι υπεύθυνο για την αλλαγή του χρώματος των κειμένων των GUI controls και το GUI.backgroundColor.a με το οποίο έχουμε την δυνατότητα να αλλάζουμε την διαφάνεια αυτών.

Στην παρακάτω εικόνα φαίνεται το μενού του "Buy-Sell" κουμπιού, δηλαδή αυτά που δημιουργούμε μέσα από την MerchantBag function που δείξαμε.

Εικόνα 4. 19 Το μενού που δημιουργήσαμε με χρήση του GUI.BeginGroup και των GUILayout.BeginVertical και GUILayout.BeginHorizontal

Στο μενού που δημιουργήσαμε παραπάνω μπορούμε να επιλέξουμε τα δυο πρώτα κουμπιά "Buy 10 health for 20 draxmes" και "Buy 1 energy for 50 draxmes", τα οποία πατώντας τα, κάνουμε true τις αντίστοιχες μεταβλητές buyHealth και buyEnergy που ελέγχονται μέσα στην OnGUI function. Εκεί δημιουργούνται για κάθε περίπτωση ένα νέο παράθυρο, μέσω της GUI.Window. Στο παράθυρο αυτό καλούμαστε να δώσουμε έναν μοναδικό αριθμό ID, την θέση του στην οθόνη και το όνομα της function του.

```
if(buyHealth){
    GUI.color = Color.white;
    GUI.Window (1, Rect (Screen.width/2,Screen.height/2,250,100), BuyHP, "Buy Health Potion");
}

if(buyEnergy){
    GUI.color = Color.white;
    GUI.Window (2, Rect (Screen.width/2,Screen.height/2,250,100), BuyE, "Buy Energy");
}
```

Η function του κάθε παραθύρου παίρνει σαν είσοδο τον ID του GUI.Window που δηλώσαμε, ενώ μέσα σε αυτές δημιουργούμε δυο κουμπιά, τα "Buy" και "Cancel", και κάθε φορά που επιλέγουμε να κάνουμε αγορά ελέγχει τα διαθέσιμα χρήματα που έχουμε και κάνει BroadcastMessage στις κατάλληλες function των script.

```
function BuyHP (windowID : int) {

    if(TalkToMerchant && showBuySell){
        LeonidasScriptFIX.InventoryMouseOver = true;
        GUI.backgroundColor.a = 1;
        GUI.color = Color.white;
        if (GUI.Button (Rect (10,50,55,25), "Buy") && (StatusGUI.draxmes >= 20)) {
            player.BroadcastMessage("AddHealth", 10);
            player.BroadcastMessage("RemoveDraxmes", 20);
            showBuyWindow = false;
            buyHealth = false;
        }
        if (GUI.Button (Rect (185,50,55,25), "Cancel")) {
            showBuyWindow = false;
            buyHealth = false;
        }
    }
    else{
        LeonidasScriptFIX.InventoryMouseOver = false;
        showBuyWindow = false;
        buyHealth = false;
    }

 }
```

Εικόνα 4. 20 Το παράθυρο που δημιουργείται μέσω της GUI.Window και της BuyHP function.

Κεφάλαιο 5

Επίλογος

5.1 Παρουσίαση του τελικού αποτελέσματος

Σε αυτό το κεφάλαιο θα παρουσιαστεί το τελικό αποτέλεσμα της πτυχιακής, θα δείξουμε μέσα από εικόνες το παιχνίδι που αναπτύχθηκε και όλες τις λειτουργίες που συναντάμε μέσα σε αυτό και τέλος θα μιλήσουμε για επεκτάσεις και βελτιώσεις του.

Ξεκινώντας το παιχνίδι φορτώνεται μια σκηνή στην οποία καλούμαστε να εισάγουμε τα στοιχεία μας username και password. Για την δημιουργία ενός νέου λογαριασμού (account), πατάμε το κουμπί Create Account και τυπώνουμε το username και password της επιλογής μας, επιβεβαιώνοντας το password στο confirm password textbox. Στην συνέχεια πατάμε Create και εφόσον το username δεν είναι πιασμένο, δημιουργείτε ο λογαριασμός μας.

Εικόνα 5. 1 Δημιουργία λογαριασμού στο παιχνίδι

Στην συνέχεια εισάγουμε τα στοιχεία μας στα κενά και πατάμε το κουμπί Login. Εφόσον τα στοιχεία μας είναι σωστά το παιχνίδι φορτώνει την δεύτερη σκηνή όπου βρίσκεται το βασικό μενού του παιχνιδιού. Εδώ σε μια προσπάθεια να δημιουργήσω μια καλή πρώτη εντύπωση στον παίκτη, δημιούργησα ένα αρκετά δυναμικό μενού, αφού ο παίκτης έρχεται σε θέση να επιλέξει τι θέλει να κάνει, μετακινώντας τον χαρακτήρα στον χώρο, ενώ ταυτόχρονα εξοικειώνεται έστω και σε αυτόν τον μικρό βαθμό με τα πλήκτρα κίνησης. Στην σκηνή αυτήν βλέπουμε τρία σπίτι, το κάθε ένα αντιπροσωπεύει ένα κουμπί ενός κλασικού μενού και συγκεκριμένα το Start Game, Load Game και Exit Game. Μετακινώντας τον χαρακτήρα στον εσωτερικό χώρο του κάθε σπιτιού, εμφανίζεται στην οθόνη μας το ανάλογο παράθυρο για την δημιουργία νέου παιχνιδιού, φόρτωση του τελευταίων σωσμένων δεδομένων για παλαιά accounts και έξοδος από αυτό, που μας επιστρέφει στα Windows.

Εικόνα 5. 2 Το σημείο επιλογής έναρξης νέου παιχνιδιού

Ξεκινώντας ένα νέο παιχνίδι βλέπουμε το βίντεο εισαγωγής, και εφόσον τελειώσει ή παρακάμπτοντας το πατώντας το κουμπί Escape, φορτώνεται η πρώτη πίστα του παιχνιδιού μας. Η περιήγηση στο παιχνίδι γίνεται με τα κουμπιά W, A, S, D ή τα Arrow keys για την μετακίνηση και περιστροφή του χαρακτήρα, με το κουμπί "Space" για πήδημα ή διπλό πήδημα, και το αριστερό "Shift" για την εναλλαγή σε τρέξιμο, γρήγορο τρέξιμο ή περπάτημα. Στην πάνω αριστερό μέρος της οθόνης βρίσκεται μια μπάρα που απεικονίζει την ζωή μας, επίσης βλέπουμε τους πόντους, την ενεργεία και τα χρήματα που έχουμε. Μέσα στον χώρο που βρισκόμαστε παρατηρούμε ότι είμαστε εγκλωβισμένοι και ο μόνος τρόπος για να βγούμε έξω και να προχωρήσουμε το παιχνίδι είναι με την ανάληψη της πρώτης αποστολής, μιλώντας στον αρχηγό της μονάδας. Πλησιάζοντας τον συγκεκριμένο NPC(Non Player Character), μπορούμε να αλληλεπιδράσουμε μαζί του, πατώντας το κουμπί "F". Εφόσον πάρουμε την πρώτη αποστολή ή πόρτα ξεκλειδώνει και μπορούμε να μετακινηθούμε στον επόμενο χάρτη.

Εικόνα 5. 3 Στιγμιότυπο από την αρχή του παιχνιδιού και επεξήγηση διάφορων στοιχείων

Στην παραπάνω εικόνα έχουμε αριθμήσει από 1-3 διάφορα γραφικά στοιχεία που βρίσκονται στην οθόνη. Στο 1 απεικονίζεται η μπάρα ζωής του χαρακτήρα, στο 2 το επιλεγμένο όπλο και στο 3 βρίσκονται τα κουμπιά για την εμφάνιση του inventory και του μενού.

Η εξέλιξη του παιχνιδιού γίνεται μέσα από την διεκπεραίωση αποστολών (quests). Για κάθε αποστολή που αναλαμβάνουμε εμφανίζεται στο δεξί μέρος της οθόνης επιγραμματικά ο τίτλος και η πορεία της αποστολής. Ακόμα υπάρχουν και κάποιες επιπλέον αποστολές, όχι απαραίτητες για την εξέλιξη του παιχνιδιού. Με κάθε NPC που μπορούμε να αλληλεπιδράσουμε, εμφανίζεται στο πάνω μέρος του κεφαλιού του, ένα σύμβολο. Το σύμβολο αυτό μπορεί να είναι ένα θαυμαστικό που σημαίνει ότι υπάρχει διαθέσιμη αποστολή προς ανάληψη, ή ένα ερωτηματικό, που δηλώνει ότι έχουμε αναλάβει κάποια αποστολή από τον συγκεκριμένο NPC.

Εικόνα 5. 4 Στιγμιότυπο από το παιχνίδι. Στην οθόνη μας εμφανίζεται η ενεργή αποστολή και το χαρακτηριστικό σύμβολο πάνω από τον NPC

Κατά την διάρκεια του παιχνιδιού ερχόμαστε να αντιμετωπίσουμε διάφορους εχθρούς, όπως ζωντανούς νεκρούς και σάτυρους. Κάθε εχθρός εμφανίζει στο πάνω μέρος του κεφαλιού του, μια μπάρα ζωής, ενώ σε περίπτωση που έχουμε προκαλέσει ακινητοποίηση, εμφανίζεται ένα σύμβολο με το γράμμα "F", κατά την διάρκεια του οποίου μπορούμε να χρησιμοποιήσουμε ένα ιδιαίτερο τύπο χτυπήματος πατώντας το κουμπί "F" σκοτώνοντας το ακαριαία. Επίσης έχοντας ενεργοποιημένο το τόξο, σημαδεύοντας κάποιον εχθρό ή αντικείμενο που καταστρέφεται, κεντράρει αυτόματα και εμφανίζεται από πάνω του το σύμβολο ενός στόχου, το οποίο υποδηλώνει ότι ο στόχος έχει κεντραριστεί και γίνεται να χτυπηθεί. Το συγκεκριμένο σύστημα, αυτόματης σκόπευσης το ανέπτυξα για αυτόν ακριβώς τον λόγω, διότι παρατήρησα ότι είναι αρκετά δύσκολο να πετύχεις κάτι από σχετικά μακρινή απόσταση χωρίς να έχεις εξοικειωθεί αρκετά με τον χαρακτήρα και τα πλήκτρα κίνησης. Επίσης σε μια αρκετά μεγάλη απόσταση πατώντας το κουμπί "Tab" επιλέγεται και σημαδεύεται αυτόματα ο κοντινότερος εχθρός εφόσον έχουμε ενεργοποιημένο το τόξο.

Εικόνα 5. 5 Στιγμιότυπα από το παιχνίδι, όπου φαίνονται η μπάρα ζωής των εχθρών, το σύμβολο αυτόματου σημαδέματος σε αντικείμενο και εχθρό και τέλος το σύμβολο ακινητοποιημένου εχθρού

Ας αναφερθούμε λίγο στα διαθέσιμα όπλα και τον τρόπο χτυπήματος τους. Πρώτα από' όλα θα ήθελα να αναφέρω ότι η αλλαγή των όπλων γίνεται με την χρήση του κουμπιών 1,2,3 του πληκτρολογίου, τα οποία με την σειρά είναι 1 = τόξο, 2 = σπαθί-ασπίδα και 3 = γροθιές. Έχοντας επιλέξει το τόξο, σημαδεύουμε πατώντας δεξί κλικ στο ποντίκι και κρατώντας το πατημένο, πατάμε αριστερό κλικ ελευθερώνοντας το βέλος. Σε περίπτωση που έχουμε αρκετή ενέργεια, μπορούμε να χρησιμοποιήσουμε το κουμπί "G" για την χρήση ενός είδους skill το οποίο χτυπά με πολλά βέλη την περιοχή κοντά στον χαρακτήρα μας, προκαλώντας ζημιά στους γύρω εχθρούς αλλά και στα αντικείμενα που σπάνε. Με την χρήση του σπαθιού και της ασπίδας μπορούμε να προκαλέσουμε ζημιά κάνοντας αριστερό κλικ, ή να κάνουμε άμυνα κατά την διάρκεια της οποίας δεν δεχόμαστε ζημιά κρατώντας πατημένο την ροδέλα. Επίσης να αναφέρω κάνοντας άμυνα, μπορούμε να προκαλέσουμε μια μικρή ζημιά στους εχθρούς και να σπάσουμε τα γύρω αντικείμενα πατώντας αριστερό κλικ. Ακόμα όπως και στην περίπτωση του τόξου, έτσι και εδώ πατώντας "G" δημιουργούνται ασπίδες με δόρατα κυκλικά με κέντρο τον χαρακτήρα, τα οποία προκαλούν ζημιά στους γύρο εχθρούς και αντικείμενα. Στην τρίτη περίπτωση, μπορούμε να προκαλέσουμε μικρή ζημιά, αφού είμαστε άοπλοι, στους εχθρούς και αντικείμενα, πατώντας αριστερό κλικ, ή να αλληλεπιδράσουμε και να καταστρέψουμε αντικείμενα που είναι πολύ κοντά στο δάπεδο, με κλωτσιά πατώντας δεξί κλικ. Τέλος να πω ότι υπάρχει και ένα γραφικό στοιχείο στο κάτω αριστερό μέρος της οθόνη που δείχνει την επιλογή που έχουμε κάνει σε όπλο, ενώ ταυτόχρονα τα όπλα που δεν χρησιμοποιούνται εμφανίζονται στην πλάτη του χαρακτήρα.

Κατά την διάρκεια του παιχνιδιού, μπορούμε να συλλέξουμε αντικείμενα στο inventory μας, κάνοντας αριστερό κλικ πάνω τους, ή πατώντας το "C" όταν είμαστε σε αρκετά κοντινή απόσταση. Στο κάθε αντικείμενο είτε είναι στο inventory μας, είτε στο έδαφος περνώντας το ποντίκι από πάνω του, δημιουργείτε ένα παράθυρο, όπου εμφανίζονται διάφορες πληροφορίες για αυτό. Για να ανοίξουμε το inventory μας, μπορούμε να πατήσουμε όποιον σάκο θέλουμε από τους δυο που βρίσκονται στο κάτω δεξί μέρος της οθόνης ή να πατήσουμε το κουμπί "I", που ανοίγει (ή κλείνει) και τους δύο αυτόματα. Τα αντικείμενα μπορούν να πουληθούν στον Merchant που βρίσκεται στην αρχή του παιχνιδιού, έναντι χρημάτων. Στον Merchant ακόμα, μπορούμε να αναπληρώσουμε την ζωή μας, να αγοράσουμε ενέργεια έναντι χρηματικού ποσού, ή να αναλάβουμε αποστολές.

Εικόνα 5. 6 Στις δύο αυτές εικόνες βλέπουμε ένα παράδειγμα, ενός αντικειμένου με το παράθυρο πληροφοριών του όταν βρίσκεται στο πάτωμα και μέσα στην τσάντα μας

Ακόμα έχουμε την δυνατότητα να κάνουμε save, οποιαδήποτε στιγμή πηγαίνοντας στα προκαθορισμένα σημεία που βρίσκονται σε επιλεγμένα σημεία στον χάρτη όπως φαίνονται στην παρακάτω εικόνα. Πατώντας πάνω τους, ενεργοποιείται στο αριστερό μέρος της οθόνης κάτω από την μπάρα ζωής, ένα κουμπί με όνομα Save. Πατώντας το κουμπί δημιουργείτε ένα αρχείο που αποθηκεύει μέσα του, την θέση του παίκτη, την ζωή, τους πόντους, την ενέργεια, την σκηνή στην οποία βρισκόμαστε, τα αντικείμενα του inventory, τις αποστολές μας και τέλος την κατάσταση των αντικειμένων που μπορούμε να αλληλεπιδράσουμε μέσα στο παιχνίδι, όπως, τα αντικείμενα που σπάνε και τα μπαούλα που βρίσκονται σε διάφορα σημεία μέσα στην πίστα. Κατά την διάρκεια ανάπτυξης του Save και Load, προσπάθησα να το δημιουργήσω με τέτοιον τρόπο ώστε να αποθηκεύονται όσον το δυνατόν περισσότερες πληροφορίες, προσπαθώντας να δώσω την αίσθηση ότι συνεχίζουμε από εκεί που σταματήσαμε κάθε φορά που κάνουμε load, όπως συμβαίνει με όλα τα σύγχρονα παιχνίδια.

Εικόνα 5. 7 Τα προκαθορισμένα σημεία, όπου μπορεί να γίνει σώσιμο του παιχνιδιού

Όπως σε όλα τα παιχνίδια, έτσι και εδώ, πατώντας το κουμπί escape θα εμφανιστεί στην οθόνη μας ένα μενού, όπου μπορούμε να επιλέξουμε μεταξύ των τριών, επιστροφή στην αρχική οθόνη μενού, load ή έξοδος από το παιχνίδι. Επίσης καθ' όλη την διάρκεια που είναι

ενεργοποιημένο το παράθυρο αυτό, στο παιχνίδι μας γίνεται παύση. Για να κλείσουμε το συγκεκριμένο παράθυρο δεν έχουμε να κάνουμε τίποτα παραπάνω από το να πατήσουμε ξανά το κουμπί escape.

Εικόνα 5. 8 Το παράθυρο του μενού που εμφανίζεται πατώντας το Escape

5.2 Μελλοντική εργασία και επεκτάσεις

Ήδη έχω δημιουργήσει προϋποθέσεις για μελλοντική εξέλιξη στο συγκεκριμένο παιχνίδι, αφού κατά την διάρκεια αυτού του εξαμήνου που ασχολήθηκα με την πτυχιακή εργασία μου, δημιούργησα στην πορεία ένα μεγάλο όγκο 3d μοντέλων, αλλά και μια μεγάλη λίστα από scripts που άλλα βρίσκονται στο παιχνίδι και άλλα όχι. Επιπλέον ένα μεγάλο μέρος του προγραμματισμού, έγινε λόγω του τεράστιου ενδιαφέρον που μου δημιούργησε το συγκεκριμένο θέμα, όπως το σύστημα αποθήκευσης (inventory), το εξειδικευμένο save-load σύστημα που συνεργάζεται με ένα μεγάλο μέρος του παιχνιδιού, η δημιουργία λογαριασμού για την υποστήριξη πολλών παικτών, το σύστημα των αποστολών, πολλές λεπτομερείς κινήσεις και αλληλεπιδράσεις του χαρακτήρα αλλά και πολλά effect που βρίσκονται μέσα στο παιχνίδι. Πολλά σημεία της πτυχιακής απαίτησαν αρκετές ώρες προγραμματισμού και σχεδιασμού, με πολλά προβλήματα και πολλές διορθώσεις με τον καιρό, ενώ σε αρκετές περιπτώσεις ακολούθησα τελείως διαφορετικές τεχνικές από αυτές που είχα χρησιμοποιήσει αρχικά.

Επίσης στα πλαίσια της πτυχιακής ασχολήθηκα και με την σύνδεση του παιχνιδιού με το Kinect του Xbox, προσπαθώντας να χρησιμοποιήσω, αλλά και να γνωρίσω μια πιο σύγχρονη τεχνολογία του gaming χώρου. Η διαδικασία αν και παρουσίασε αρκετά προβλήματα, στο τέλος κατάφερα να συνδέσω τις κινήσεις του χαρακτήρα με τις δικές μου,

μπροστά στην κάμερα. Πιστεύω ότι είναι μια άκρως ενδιαφέρουσα τεχνολογία η οποία θα μπορούσε να χρησιμοποιηθεί για την ανάπτυξη ενός τελείως διαφορετικού game-play του παιχνιδιού, με το οποία μελλοντικά θα ήθελα να ασχοληθώ περισσότερα.

Μια ακόμα εξέλιξη του παιχνιδιού θα μπορούσε να είναι η δημιουργία multiplayer συστήματος για την υποστήριξη πολλών χρηστών ταυτόχρονα σε τοπικό δίκτυο ή μέσω Internet, μια λειτουργία που το Unity μπορεί να υποστηρίξει.

Τέλος μια ακόμα επέκταση του παιχνιδιού θα μπορούσε να είναι η μετατροπή του για χρήση σε συσκευές Smartphone κινητών. Η διαδικασία μετατροπής του κώδικα του παιχνιδιού για χρήση κινητών τηλεφώνων λειτουργικού συστήματος iOS και Android, γίνεται αρκετά εύκολα με το Unity και με ελάχιστη επιπλέον προγραμματιστική δουλειά.

Βιβλιογραφία

http://unity3d.com/

http://unity3d.com/support/documentation/ScriptReference/index.html

http://www.unifycommunity.com/wiki/index.php?title=Main_Page

http://usa.autodesk.com/3ds-max/

http://www.pixologic.com/home.php

http://www.blender.org/

Προγράμματα που χρησιμοποιήθηκαν

Autodesk 3ds Max 2009
Pixologic ZBrush 4.0
Blender game engine 2.6.2
Unity game engine 3.2
Adobe Photoshop CS3