ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
«Εφαρμογή
για τη δημιουργία 3D χαρακτήρων »

Του φοιτητή
Ρωσσίδη Ευάγγλου
Αρ. Μητρώου: 113789

Επιβλέπων
Χριστίνα Βολιώτη
Βαθμίδα Μετα-Διδάκτορας

Ημερομηνία 04/09/2022

Εφαρμογή για τη δημιουργία 3D χαρακτήρων
Κωδικός Π.Ε. 22158
Ρωσσίδης Ευάγγελος
Χριστίνα Βολιώτη
30-03-2022
04-09-2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Αφιέρωση»

Πρόλογος

Η πτυχιακή εργασία που ανέλαβα έχει ως θέμα την δημιουργία 3D χαρακτήρων μέσω ενός customization μενού, με το οποίο μπορεί ο παίκτης να δημιουργήσει δικό του χαρακτήρα και να τον περάσει μέσα στο παιχνίδι για να παίξει.

Ο λόγος που ανέλαβα αυτή την πτυχιακή, είναι επειδή μου αρέσει πολύ η δημιουργία παιχνιδιών και ήθελα να εξασκηθώ, ώστε να ακολουθήσω στο μέλλον καριέρα πάνω στη δημιουργία παιχνιδιών. Έχω ξανασχοληθεί στο παρελθόν και μόνος μου με το game developing, κυρίως με τη χρήση του Unreal Engine, το οποίο θεωρώ ότι είναι ένα πολύ καλό πρόγραμμα δημιουργίας παιχνιδιών, αλλά πιστεύω πως είναι καλό να μάθω να χρησιμοποιώ και το Unity, ώστε να μπορώ να ασχοληθώ και με τα 2 engines αν χρειαστεί.

Περίληψη

Το θέμα της πτυχιακές εργασίες που ανέλαβα, είναι η δημιουργία και το customization 3D χαρακτήρων, μέσω ενός παιχνιδιού στο οποίο μπορεί ο παίκτης να δημιουργήσει τον δικό του χαρακτήρα και να παίξει.

Η υλοποίηση της πτυχιακής έγινε μέσω των προγραμμάτων Blender (για την δημιουργία των 3D μοντέλων) και Unity (για το στήσιμο του παιχνιδιού) και η συγγραφή του κώδικα, έγινε με τη γλώσσα προγραμματισμού C#. Δημιουργώντας αυτό το παιχνίδι μπόρεσα να βελτιώσω και να εξασκήσω τις γνώσεις μου πάνω σε αυτόν τον τομέα, καθώς και να μάθω διάφορες καινούργιες τεχνικές για την δημιουργία παιχνιδιών, οι οποίες θα μου χρησιμεύσουν στο μέλλον.

Το παιχνίδι που έφτιαξα έχει ένα μενού στο οποίο επιλέγεις την φυλή του χαρακτήρα σου και στη συνέχεια αλλάζεις τα διάφορα στοιχεία του χαρακτήρα, με βάση τη φυλή που έφτιαξες. Κάθε φυλή έχει κάποια διαφορετικά στοιχεία, όπως για παράδειγμα ο δαίμονας έχει κέρατα τα οποία δεν υπάρχουν σε άλλη φίλη. Επίσης ο παίκτης μπορεί να αλλάξει τα χρώματα του χαρακτήρα και μετά στο επόμενο μενού μπορεί να διαμορφώσει το πρόσωπο του χαρακτήρα όπως το θέλει. Στην συνέχεια ο παίκτης επιλέγει διάφορα ρούχα για τον χαρακτήρα του και αφού ολοκληρώσει το customization του χαρακτήρα του, μπαίνει στο παιχνίδι στο οποίο έχει ένα σπαθί και πολεμάει κάποια τέρατα, τα οποία τριγυρνάνε μέσα στην πίστα.

«Application for 3D character customization»

«Rossidis Evangelos»

Abstract

The topic of my thesis, is the creation and customization of 3D characters, through a game in which the player can create his own character and play.

My thesis was implemented through the programs Blender (to create the 3D models and the animations) and Unity (to set up the game) and the code was written using the C# programming language. By creating this game I was able to improve and practice my existing knowledge in this field, as well as learn several new techniques for game creation, which will serve me well in the future.

The game I made has a menu where you select your character's race and then change the various elements of the character based on the race you chose. Each race has some different elements, like for example the demon has horns which no other race has. Also the player can change the colors of the character and then in the next menu he can shape the face of the character as he wants. Then the player chooses various clothes for his character and after completing the customization of his character, he enters the game in which he has a sword and fights some monsters, which roam inside the map.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
A.I.	Artificial Intelligence
GUI	Graphical User Interface
UI	User Interface

Κεφάλαιο 1ο: Εισαγωγή

Η πτυχιακή εργασία που ανέλαβα έχει ως θέμα την δημιουργία 3D χαρακτήρων μέσω ενός customization μενού, με το οποίο μπορεί ο παίκτης να δημιουργήσει δικό του χαρακτήρα και να τον περάσει μέσα στο παιχνίδι για να παίξει. Η δημιουργία των μοντέλων υλοποιήθηκε μέσω του blender και το παιχνίδι φτιάχτηκε στο Unity engine. Επίσης η γλώσσα προγραμματισμού που χρησιμοποιήθηκε ήταν η C#.

Ο λόγος που ανέλαβα αυτή την πτυχιακή, είναι επειδή μου αρέσει πολύ η δημιουργία παιχνιδιών και ήθελα να εξασκηθώ, ώστε να ακολουθήσω στο μέλλον καριέρα πάνω στη δημιουργία παιχνιδιών. Έχω ξανασχοληθεί στο παρελθόν και μόνος μου με το game developing, κυρίως με τη χρήση του Unreal Engine, το οποίο θεωρώ ότι είναι ένα πολύ καλό πρόγραμμα δημιουργίας παιχνιδιών, αλλά πιστεύω πως είναι καλό να μάθω να χρησιμοποιώ και το Unity.

Το κείμενο της πτυχιακής είναι χωρισμένο σε 7 κεφαλαία, τα οποία με τη σειρά τους είναι χωρισμένα σε αριθμημένες ενότητες και υποενότητες.
Το πρώτο κεφάλαιο είναι η εισαγωγή στο οποίο γράφω λίγα λόγια για την πτυχιακή μου και αναφέρω τι θα ακολουθήσει στα επόμενα κεφάλαια.

Το δεύτερο κεφάλαιο έχει θέμα τη δημιουργία 3D μοντέλων στο Blender, στο οποίο αναλύω τη διαδικασία που ακολούθησα για να δημιουργήσω και να επεξεργαστώ τα μοντέλα που χρησιμοποίησα στο παιχνίδι. Αναλύω δηλαδή σε στάδια πως δημιούργησα το μοντέλο του χαρακτήρα όπως επίσης και ένα παράδειγμα για το πως δημιούργησα ένα από τα μοντέλα των στοιχείων του χαρακτήρα, δηλαδή μοντέλα όπως αυτιά, κέρατα και τα λοιπά. Τέλος αναφέρω ποια asset είναι αυτά που δημιουργήθηκαν στο blender. Το μόνο πράγμα που δεν αναλύω σε αυτό το κεφάλαιο είναι ότι έχει να κάνει με το animation του χαρακτήρα, διότι αυτό το αναλύω ξεχωριστά στο επόμενο κεφάλαιο.

Το 3ο κεφάλαιο έχει θέμα την δημιουργία animation στο blender. Εδώ αναλύω σε στάδια την διαδικασία που ακολούθησα για να δημιουργήσω τον σκελετό του μοντέλου και στην συνέχεια να

δημιουργήσω το animation χρησιμοποίησα στις σκηνές του μενού στο παιχνίδι. Επίσης αναλύω την διαδικασία που ακολούθησα ώστε να επεξεργαστώ και να προσαρμόσω τα έτοιμα μοντέλα των ρούχων, ώστε να μπορούν να χρησιμοποιηθούν με τα animation του παιχνιδιού και τέλος αναφέρω τις ρυθμίσεις που έβαλα ώστε να κάνω εξαγωγή αυτά τα μοντέλα και το animation για να χρησιμοποιηθούν αργότερα στο Unity.

Το τέταρτο κεφάλαιο έχει ως θέμα το στήσιμο του παιχνιδιού στο Unity. Εδώ αναλύω ολόκληρο το στήσιμο το παιχνιδιού εκτός από τον κώδικα, τον οποίο εξηγώ στο επόμενο κεφάλαιο. Στις πρώτες ενότητες αυτού του κεφαλαίου αναφέρω την προέλευση των asset που χρησιμοποίησα καθώς και πώς τα έβαλα μέσα στο project μου και μετά πώς τα ρύθμισα μέσα από τους φακέλους. Στην συνέχεια αναλύω πως δημιούργησα τις σκηνές και τι gameobject έβαλα μέσα σε αυτές τις σκηνές. Αυτή η ενότητα είναι χωρισμένη σε 5 υποενότητες για καθεμία από τις ομάδες σκηνών που δημιούργησα. Έπειτα σε άλλη ενότητα αναλύω όλες τις ρυθμίσεις τον component της προηγούμενης ενότητας. Αυτή η ενότητα, όπως και η προηγούμενη, είναι χωρισμένη σε πέντε υποενότητες, μία για κάθε ομάδα σκηνών. Τέλος αναλύω το στήσιμο των State machine του παιχνιδιού, καθώς και τα build settings του παιχνιδιού.

Το πέμπτο κεφάλαιο έχει ως θέμα τον κώδικα του παιχνιδιού στο Unity. Εδώ αναλύω όλον τον κώδικα που έγραψα για το παιχνίδι τον οποίο έχω χωρισμένο σε ενότητες με βάση την λειτουργία των διαφόρων script που έφτιαξα. στην συνέχεια αυτές οι ενότητες είναι χωρισμένες σε επιπλέον υποενότητες, μία για κάθε ξεχωριστό script. Και τέλος είναι χωρισμένο και σε άλλες υποενότητες, ώστε να εξηγώ ξεχωριστά, τον κώδικα της κάθε μεθόδου για κάθε script, αν βέβαια το script ήταν αρκετά μεγάλο ώστε να δικαιολογείται ο χωρισμός.

Το έκτο κεφάλαιο έχει ως θέμα ένα παράδειγμα χρήσης του παιχνιδιού από τον παίκτη. Σε αυτό το κεφάλαιο εξηγώ το παιχνίδι και τους διάφορους χειρισμούς του από την πλευρά ενός απλού χρήστη ο οποίος παίζει το παιχνίδι, για να ξέρει τι κάνει το κάθε κουμπί και πώς να δημιουργήσει τον χαρακτήρα του και να παίξει. Αυτό το κεφάλαιο είναι επίσης χωρισμένο σε ενότητες, για κάθε ομάδα σκηνών του παιχνιδιού.

Το 7ο κεφάλαιο έχει θέμα κάποιες προτάσεις βελτιώσεις του παιχνιδιού. Εδώ δηλαδή αναφέρω μερικά παραδείγματα σχετικά με τι θα μπορούσα προσθέσω, ώστε να βελτιωθεί το παιχνίδι.
Και τέλος έχω προσθέσει τη βιβλιογραφία της πτυχιακής μου στην οποία έχω βάλει links σε όσες σελίδες χρησιμοποίησα, για να πάρω τα πράγματα που χρησιμοποίησα για την υλοποίηση της πτυχιακής μου.

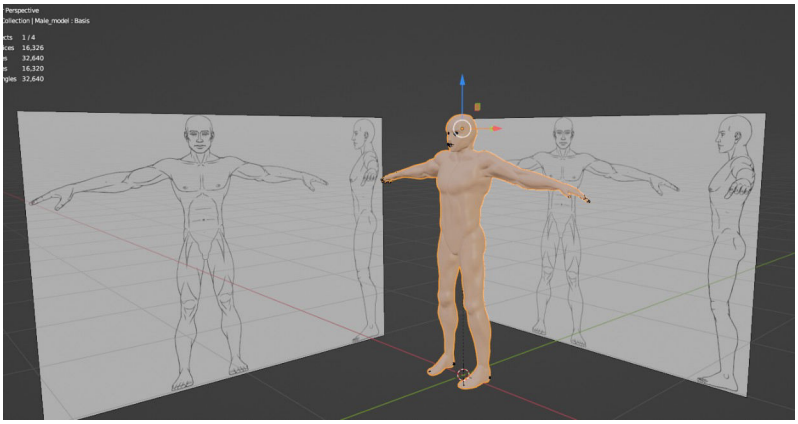
Κεφάλαιο 2ο: Δημιουργία 3D Μοντέλων στο Blender

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο εξηγώ πρώτα πως δημιούργησα το αρχικό low-resolution μοντέλο του χαρακτήρα και μετά πως έκανα το sculpting για να προσθέσω παραπάνω λεπτομέρια στο μοντέλο (high-resolution). Εδώ αναλύω μόνο το μοντέλο του χαρακτήρα, διότι τα υπόλοιπα μοντέλα, όπως το μοντέλο των αυτιών, έγινε με την ίδια διαδικασία. Το μόνο επιπλέον μοντέλο που εξηγώ είναι τα κέρατα γιατί χρησιμοποίησα κάποια διαφορετικά εργαλεία. Μετά εξηγώ πως έκανα το UV Unwrapping του μοντέλου, δηλαδή πως το ξεδίπλωσα ώστε να δημιουργήσω μία 2D αναπαράσταση του 3D μοντέλου του χαρακτήρα και στην συνέχεια την δημιουργία του Normal Map του μοντέλου, ώστε να προσθέσω την λεπτομέρια του high-resolution μοντέλου, πάνω στο low-resolution μοντέλο, χωρίς να επιβαρύνει τους πόρους του συστήματος. Μετά εξηγώ πως χρωμάτισα τον χαρακτήρα και έπειτα πως δημιούργησα τα shape keys (τα οποία λέγονται blendshapes στο Unity), για να διαμορφώνει ο παίχτης το πρόσωπο του χαρακτήρα. Στην συνέχεια εξηγώ πως έφτιαξα τα κέρατα και τέλος αναφέρω ποιά μοντέλα κατασκεύασα ή επεξεργάστηκα μέσα στο blender.

2.2 Δημιουργία Γεωμετρίας Μοντέλου

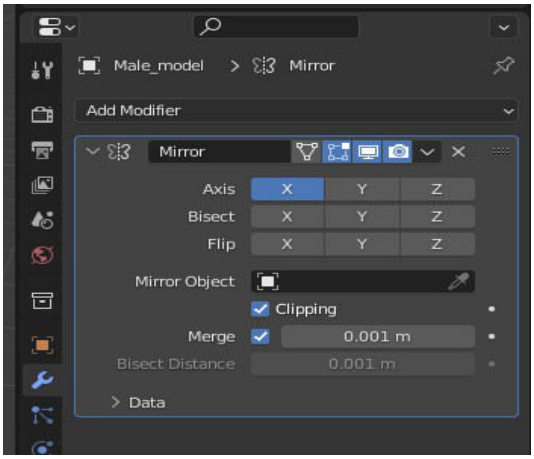
Εδώ περιγράφω το αρχικό στάδιο της δημιουργίας του βασικού μοντέλου. Ξεκινώντας διέγραψα όλα τα αντικείμενα που υπήρχαν μέσα στην σκηνή του blender και πατώντας “Shift + A”, άνοιξα το μενού προσθήκης αντικειμένων και επέλεξα “Image → Reference”, ώστε να προσθέσω μία εικόνα ως blueprint [1] για να χτίσω το μοντέλο μου πάνω σε αυτήν (αντίστοιχη διαδικασία έγινε και για τα αυτιά [2]). Μετά έστησα την εικόνα να στέκεται όρθια και την έβαλα λίγο πίσω ώστε να μην είναι ακριβώς στο κέντρο, επειδή εκεί θα μπεί ο χαρακτήρας. Έπειτα δημιούργησα ένα duplicate αυτής της εικόνας το έκανα rotate 90 μοίρες πάνω στον άξωνα “Z” και το μετακίνησα, ώστε να σχηματίζει μία γωνία με την πρώτη εικόνα και ανάμεσα τους θα φτιάξω τον χαρακτήρα. Επίσης αφού τοποθέτησα σωστά αυτές τις εικόνες, τις έβαλα σε έναν φάκελο και απενεργοποίησα το την δυντότητα επιλογής των εικόνων με το ποντίκι (αυτό γίνεται πατώντας το εικονίδιο του ποντικιού δίπλα από τα αντικείμενα στο Outliner παράθυρο πάνω δεξιά), ώστε να μην επιλέγω και μετακινώ κατά λάθος τις εικόνες, όσο δουλεύω πάνω στο μοντέλο μου.



Εικόνα 2.1 Στήσιμο Reference Image

Στην συνέχεια δημιούργησα ένα UV Shpere, πατώντας “Shift + A” και μετά “Mesh → UV Shpere” και το οποίο εμφανίζεται στο σημείο που βρίσκεται ο Cursor (ένας κύκλος που φαίνεται στην σκηνή). Αν ο cursor για κάποιο λόγο δεν είναι στο world origin, μπορείς να τον επαναφέρεις πατώντας “Shift + S” και μετά “Cursor to World Origin”.

Μετά μετακίνησα το UV Sphere προς τα πάνω εκεί που οι εικόνες δίχνουν το κεφάλι του χαρακτήρα. Για βλέπω καλύτερα που μετακινώ το mesh, πάτησα τα πλήκτρα από το Numpad τα οποία ελέγχουν τις κάμερες θέασης της σκηνής, από τις οποίες κυρίως χρησιμοποίησα την front view (πλήκτρο “1” στο Numpad) και το side view (πλήκτρο “3”). Έπειτα μπήκα στο Edit Mode και διέγραψα το μισό mesh πάνω στο άξωνα “x” και μετά μπήκα στο Object Mode (πατώντας “Ctrl + Tab”) και μετά από τα modifiers στο παράθυρο Properties στο δεξί μενού, πρόσθεσα έναν Mirror modifier και επέλεξα τον X axis και ενεργοποίησα το clipping.

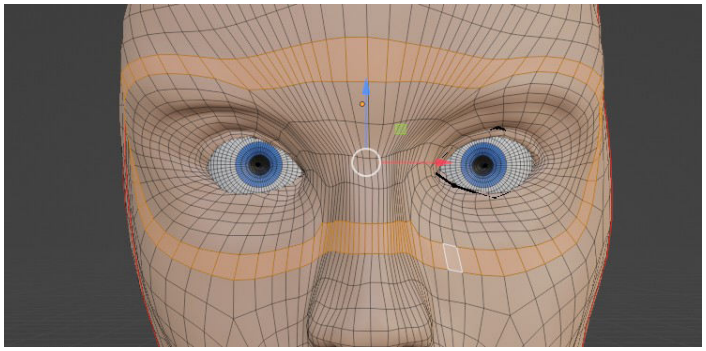


Εικόνα 2.2: Properties Window (Mirror Modifier)

Αυτό έγινε ώστε να διαμορφώνω μόνο την μία μεριά του μοντέλου και να σχηματίζεται συμμετρικα και η άλλη μεριά αυτόματα και με το clipping ενώνονται τα vertices στην μέση. Στην συνέχεια μπήκα πάλι στο Edit Mode και άνοιξα το Proportional Editing (πατώντας το πλήκτρο “O”), το οποίο

μετακινεί ομαλά όλα τα vertices που βρίσκω κοντά στο vertice που μετακινείς, ανάλογα με το μέγεθος του Proportional Editing, το οποίο αλλάζεις με το mouse wheel. Έτσι άρχισα διαμορφώνω το μοντέλο ξεκινώντας από το κεφάλι και σιγά σιγά πήγα προς τα κάτω.

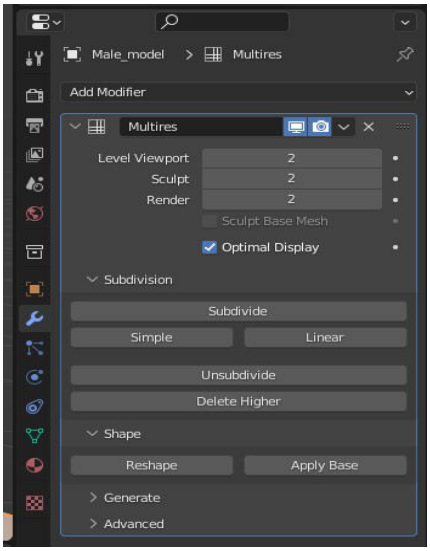
Αφού ολοκλήρωσα το σχήμα του κεφαλιού, επέλεξα κάποια faces (εκεί που θα είναι ο λαιμός) και έκανα extrude (πλήκτρο "E"), για να δημιουργήσω καινούργιο geometry προς τα κάτω. Επίσης έκανα μερικά loop cuts (πατώντας "Ctrl + R" και μετά επιλέγεις πόσα θέλεις με το scroll wheel), το οποίο χωρίζει το mesh σε περισσότερα τμήματα για έχεις περισσότερο geometry για να δουλέψεις. Επιπλέον όλο το topology του μοντέλου διαμορφώθηκε ώστε όλα τα faces να έχουν 4 γωνίες, και τα faces να σχηματίζουν loops, το οποίο βοηθάει πολύ, ειδικά όταν πρόκειται για animated χαρακτήρες ώστε να μην παραμορφώνονται περίεργα με τα animations. Αναδιαμόρφωσα επίσης και το πρόσωπο ώστε να σχηματίζει τέτοια loops πχ γύρω από τα μάτια, το στόμα κτλ, χρησιμοποιώντας και το Knife tool (πλήκτρο "K").



Εικόνα 2.3: Παράδειγμα Ενός Loop Στο Πρόσωπο

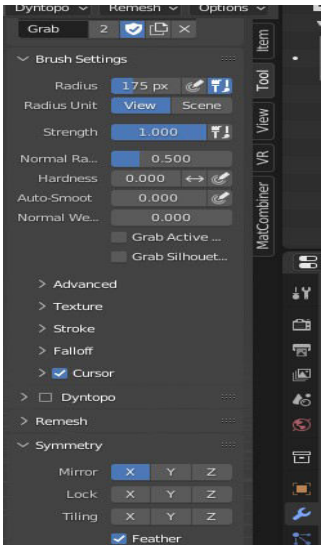
2.3 Sculpting

Εδώ περιγράφω το sculpting του μοντέλου ώστε να του προσθέσω περισσότερη λεπτομέρια. Ξεκινώντας αφού ολοκλήρωσα το βασικό geometry του μοντέλου στο Edit Mode, μπήκα στο Object Mode (πατώντας "Ctrl + Tab") και μετά από τα modifiers στο παράθυρο Properties στο δεξί μενού, έκανα apply τον mirror modifier και μετά πρόσθεσα έναν MultiResolution modifier, με τον οποίο αύξησα το geometry του μοντέλου. Σε αυτόν τον modifier πάτησα το κουμπί Subdivide μέχρι να φτάσει στην τιμή 2 και μετά μπήκα στο Sculpt Mode και πατώντας το πλήκτρο "N", άνοιξα το options panel στο παράθυρο της σκηνής και επέλεξα την καρτέλα Tool.



Εικόνα 2.4: Multi-Resolution Modifier

Έπειτα πήγα στο Symmetry μενού και επέλεξα τον άξωνα "x", ώστε ότι φτιάχνω στο sculpting στην μία μεριά να αλλάζει και στην άλλη. Αυτό έγινε επειδή η συμμετρία στο sculpting, δεν μπορεί να λειτουργήσει με το Mirror modifier , όταν υπαρχει και το MultiResolution modifier (παράγει περίεργα αποτελέσματα όταν κάνεις apply το mirror).



Εικόνα 2.5: Symmetry Menu

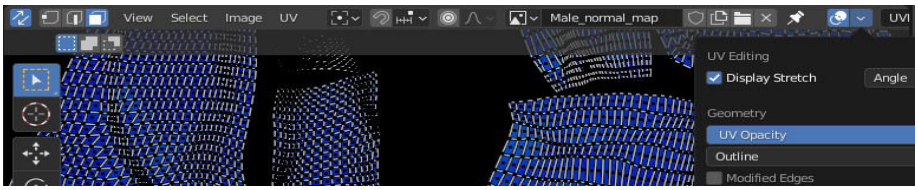
Στην συνέχεια έπλασα τον χαρακτήρα με την χρήση των διαφόρων brushes του sculpt mode, τα οποία βρίσκονται στο μενού αριστερά που εμφανίζεται πατώντας το πλήκτρο "T". Από αυτά τα brushes, αυτά που χρησιμοποίησα περισσότερο ήταν τα crease, smooth, scrape και grap.

Μόλις ολοκλήρωσα το sculpting, πήγα στο shape και πάτησα το κουμπί "Apply Base" (βλ. Εικόνα 2.4), ώστε να φέρω το αρχικό μοντέλο (αυτό που φαίνεται στο Edit Mode), όσο πιο κοντά γίνεται με το sculpted μοντέλο.

2.4 UV Unwrapping και Δημιουργία Normal Map Μοντέλου

Εδώ περιγράφω την διαδικασία του UV Unwrapping και της δημιουργίας του Normal map του μοντέλου. Το UV Unwrapping είναι η διαδικασία με την οποία ξεδιπλώνεις το 3D μοντέλο, ώστε να απλώσεις τις επιφάνειες του μέσα σε μία 2D image (UV map). Το Normal map είναι ένα texture, στο οποίο κάνεις bake τις λεπτομέριες του high-resolution μοντέλου (του sculpted μοντέλου), μέσα στο UV map και μετά το προσθέτεις πάνω στην low-res εκδοχή του μοντέλου, ώστε να δώσει την ψευδαίσθηση οτι το μοντέλο έχει περισσότερη λεπτομέρια από ότι έχει στην πραγματικότητα. Έτσι το low res μοντέλο δίχνει σχεδόν ίδιο με το high-res μοντέλο χωρίς να επιβαρύνει τους πόρους του συστήματος.

Για να κάνω UV Unwrap το μοντέλο μου, αρχικά άνοιξα ένα δεύτερο παράθυρο στο οποίο έβαλα τον UV Editor (ή απλα πατώντας την καρτέλα UV Editing, η οποία ανοίγει αυτόματα το UV Editor δίπλα στο 3D Viewport). Μετά επέλεξα το μοντέλο και μπήκα στο edit mode ώστε να εμφανιστούν κάποιες επιπλέον επιλογές στον UV editor οι οποίες εμφανίζονται μόνο στο edit mode. Μέσα στον UV editor άνοιξα την ρύθμιση UV sync selection (το εικονίδιο με τα δύο βελάκια που κοιτάνε σε αντίθετες κατευθύνσεις), ώστε να εμφανίζονται όλα τα vertices μέσα στον UV editor ακόμα και όταν δεν είναι επιλεγμένα από το μοντέλο στο viewport. Επίσης ενεργοποίησα την επιλογή Show Overlays και από το μενού του overlays επέλεξα το display stretch ώστε όταν κάνω UV unwrap, να εμφανίζει με χρώματα αν υπάρχει παραμόρφωση στο UV texture.

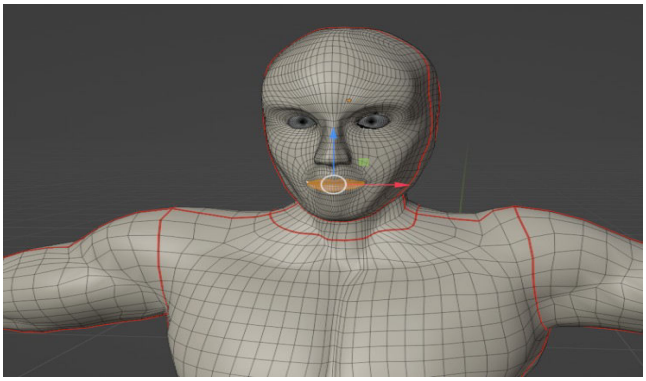


Εικόνα 2.6: UV Sync Selection & Show Overlays Settings

Στην συνέχεια πάνω στο μοντέλο μέσα από το edit mode, επέλεξα που θα μπουν οι εγκοπές/ραφές οι οποίες δίνουν τα σημεία στα οποία θα κοπεί και θα ξεδιπλωθεί το 3D μοντέλο κατά το UV unwrapping. Αυτό έγινε επιλέγοντας τα edges που θέλω και στη συνέχεια πάτησα το πλήκτρο "U" και μετά την επιλογή mark seam. Αυτά τα seam τα έβαλα σε σημεία μου φαινόντουσαν λογικό να μπούνε.

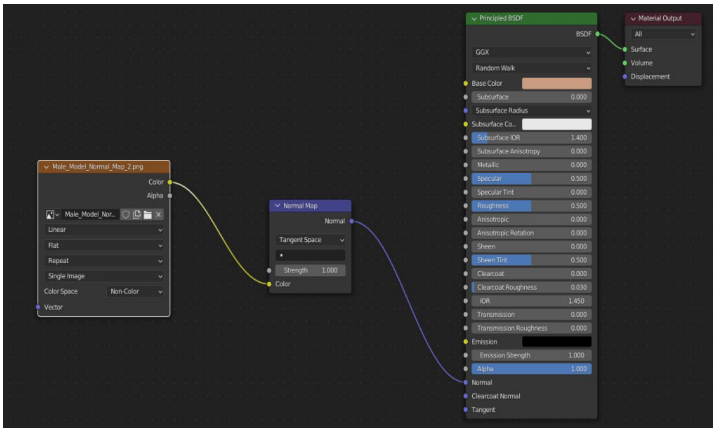
Έπειτα αφού έβαλα όλα τα seam, επέλεξα όλα τα vertices του μοντέλου και πάτησα το πλήκτρο "U" και μετά την επιλογή Unwrap. Αν υπάρχουν περιοχές οι οποίες δεν έχουνε μπλε χρώμα σημαίνει ότι παραμορφώνονται, οπότε πρόσθεσα κάποια επιπλέον seams για να το διορθώσω.

Στην συνέχεια ακολούθησε το baking του normal map. Αρχικά με το μοντέλο επιλεγμένο πήγα στο παράθυρο properties και μετά στην καρτέλα materials και δημιούργησα ένα καινούργιο Material και το έκανα assign στο μοντέλο. Μετά μέσα από τον UV editor δημιούργησα ένα καινούργιο image στο οποίο έβαλα διαστάσεις 4096 x 4096 και το ονόμασα Normal_Map. Έπειτα άνοιξα και ένα τρίτο παράθυρο στο οποίο έβαλα τον Shader Editor, μέσα στον οποίο εμφανίζει κάποια nodes από το επιλεγμένο Material. Τα nodes που δημιουργούνται μόνα τους είναι το principled BSDF και το Material Output. Μετά πρόσθεσα το Normal Map node πατώντας "Shift + A" και επέλεξα



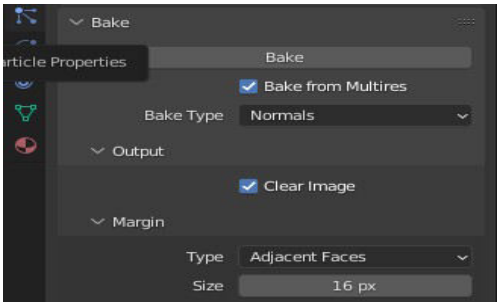
Εικόνα 2.7: Mark Seam Example (Red Lines)

Vector → Normal map και ένωσα το “Normal” output του Normal Map node, με το “Normal” input του Principled BSDF node. Επίσης πρόσθεσα το image texture node πατώντας "Shift + A" και μετά Texture → Image Texture και μέσα σε αυτό το node έβαλα το Normal_Map image που έφτιαξα και μετά επέλεξα από την επιλογή “Color Space”, το Non-Color και τέλος ένωσα το Image Texture node με το Normal Map node.



Εικόνα 2.8: Nodes Inside Shader Editor

Αφού έγιναν όλες αυτές οι ρυθμίσεις, για να κάνω bake το normal map, πήγα στο παράθυρο properties και μετά στην καρτέλα render και από εκεί άνοιξα το μενού του Bake ενεργοποίησα την επιλογή “Bake From Multires”, επέλεξα το “Normals” στο “Bake Type” και πάτησα Bake.



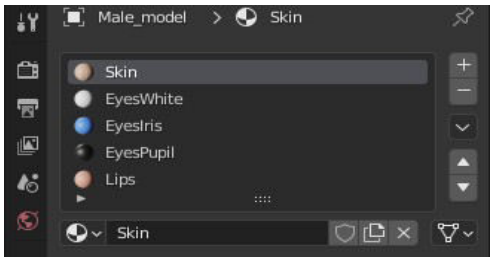
Εικόνα 2.9: Normal Map Baking Settings

Τέλος αφού με την επεξεργασία του μοντέλου έκανα Apply το Multires modifier.

2.5 Texture Painting

Για να προσθέσεις χρώματα ή textures στο χαρακτήρα, μπορείς να το κάνεις από το texture painting για περισσότερη λεπτομέρεια, αλλά εγώ απλά έβαλα κάποια χρώματα πάνω σε συγκεκριμένα faces το μοντέλου.

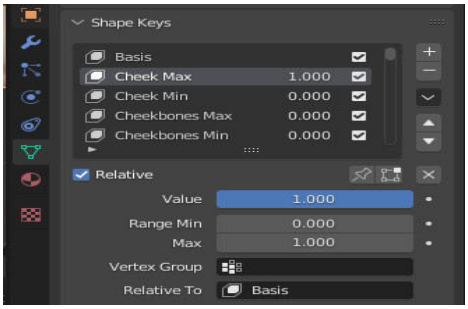
Για αρχή μπήκα μέσα στο Material στο οποίο πρόσθεσα το normal map και μέσα στον Shader Editor άλλαξα το base color του Principled BSDF, σε ένα χρώμα του δέρματος και ονόμασα αυτό το material “Skin”. Έπειτα δημιούργησα μερικά ακόμα materials τα οποία έκανα assign στα κατάλληλα faces του μοντέλου και τους έβαλα το κατάλληλο χρώμα. Αυτά τα materials τα ονόμασα EyesWhite, EyesIris, EyesPupil και Lips.



Εικόνα 2.10: Materials Of Character Model

2.6 Προσθήκη Shape Keys Για Το Πρόσωπο

Τα Shape Keys χρησιμοποιούνται για να παραμορφώσεις/τροποποιήσεις ένα αντικείμενο και να αποθηκεύσεις αυτές τις τροποποιήσεις, χωρίς να πειράξεις το αρχικό μοντέλο. Μπορείς δηλαδή να αλλάξεις το σχήμα ενός μοντέλου και να αποθηκεύσεις αυτή την τροποποίηση μέσα σε ένα Shape Key, το οποίο μπορείς να ενεργοποιήσεις και να απενεργοποιήσεις όποτε θέλεις, ελέγχοντας ακόμα και τον βαθμό που επηρεάζει αυτό το shape key το μοντέλο, με την χρήση ενός value που παίρνει τιμές από 0 έως 1.



Εικόνα 2.11: Shape Keys Window

Αυτά τα Shape Keys έχουν διάφορες χρήσεις, κάποιες από τις οποίες είναι για να δημιουργήσεις τα animation του προσώπου (πχ για να ανοιγοκλείνουν τα μάτια, να κουνάει τα φρύδια του, το στόμα του κτλ) ή για να διορθώσεις κάποια σημεία πάνω στα animation του μοντέλου (corrective shape keys) τα οποία δεν γίνονται σωστά μόνο με τη χρήση του σκελετού (rig), ώστε τα animation σου να παραμορφώνουν το μοντέλο με μεγαλύτερη ακρίβεια. Τα shape keys μπορούν επίσης να ενεργοποιηθούν μέσω κάποιων drivers.

Στην δική μου περίπτωση όμως, τα χρησιμοποίησα ώστε να αποθηκεύσω κάποιες τροποποιήσεις στο πρόσωπο του μοντέλου μου, ώστε μετά σκαλίζοντας αυτά τα shape keys, να μπορεί ο παίκτης να διαμορφώσει το πρόσωπο το χαρακτήρα του (στο Unity αυτά είναι τα blendshapes από τις σκηνές διαμόρφωσης προσώπου βλ. Ενότητες 4.4.3 & 5.3).

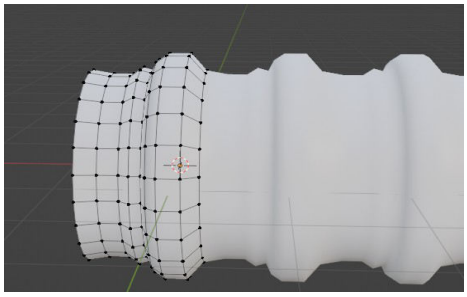
Για να δημιουργήσω αυτά τα shape keys, πήγα στο παράθυρο properties και πάτησα στην καρτέλα “object data properties” (βλ. Εικόνα 2.11). Από εκεί πήγα στα shape keys πάτησα να προσθέσω ένα shape key. Το πρώτο που προστίθεται ονομάζεται basis και είναι το αρχικό μοντέλο (βάση) σε σχέση με το οποίο δημιουργούνται όλα τα υπόλοιπα shape keys. Το basis shape key δεν μπορείς να το πειράξεις. Στην συνέχεια δημιούργησα διάφορα shape keys, τα οποία αυτόματα έχουν στο πεδίο “relative to”, το πρώτο βασικό shape key (basis), ως σημείο αναφοράς. Έπειτα για να επεξεργαστώ τα shape keys που έφτιαξα, επέλεξα το shape key που ήθελα και μπήκα στο edit mode ώστε να επεξεργαστώ το μοντέλο και να αποθηκευτεί αυτή η τροποποίηση στο συγκεκριμένο shape key. Έπειτα μπαίνοντας στο object mode, για να δεις την αλλαγή πρέπει να επιλέξεις το τροποποιημένο shape key και να αυξήσεις την τιμή Value, η οποία όσο πιο πολύ την αυξάνεις, παραμορφώνεται το μοντέλο περισσότερο. Τα shape keys που έφτιαξα ήταν 36 συνολικά και τα οποία ήταν σε ζευγάρια των 2 (Min και Max για κάθε σημείο του προσώπου πχ Max για να φουσκώσεις τα μάγουλα και Min για να τα ρουφήξεις προς τα μέσα).

2.7 Μοντέλα Στοιχείων Χαρακτήρα

Τα μοντέλα των στοιχείων του χαρακτήρα έγιναν με παρόμοιες διαδικασίες με την διαφορά κάποιων μοντέλων όπως τα κέρατα. Σε αυτά χρησιμοποίησα τα NurbsPath με τα οποία όρισα μία διαδρομή που ακολουθεί το mesh για να σχηματίσω την καμπύλη των κεράτων.

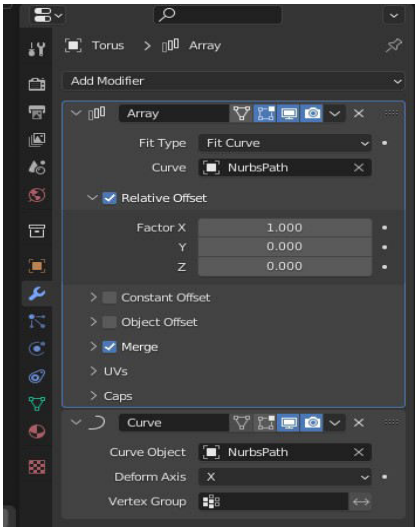
Στη ουσία δημιούργησα ένα NurbsPath και του έδωσα την καμπύλη που ήθελα και έβαλα κάποιες ρυθμίσεις μέσα από το Object Data Properties, στην καρτέλα του παραθύρου Properties.

Έπειτα δημιούργησα ένα mesh στο οποίο σχεδίασα μόνο ένα μικρό κομμάτι από τα κέρατα και μετά αυτό το έβαλα πάνω στο path, ώστε να πολλαπλασιάσω το ίδιο mesh πολλές φορές, ακολουθώντας το path και έτσι να σχηματιστεί το κέρατο.



Εικόνα 2.12: Δημιουργία Ενός Κομματιού Και Το Υπόλοιπο Πολλαπλασιάζεται

Δηλαδή αφού έφτιαξα το κομμάτι του μοντέλου του κεράτου, πρόσθεσα έναν Array modifier, με το οποίο φτιάχνω πολλά αντίγραφα του μοντέλου που έφτιαξα και τα οποία ενώνονται σε ένα ενιαίο mesh και μετά πρόσθεσα και ένα Curve modifier, στο οποίο έβαλα το NurbsPath που έφτιαξα, ώστε το mesh να ακολουθήσει την πορεία του path.



Εικόνα 2.13: Array & Curve Modifiers

2.8 Assets Που Κατασκευάστηκαν Με Το Blender

Τα assets που έφτιαξα στο blender, ήταν τα εξής:

- Το μοντέλο του χαρακτήρα του παιχνιδιού.
- Το Idle animation του χαρακτήρα που παίζει στις σκηνές του μενού στο Unity.

- Τα διάφορα στοιχεία του χαρακτήρα, όπως τα αυτιά, τα κέρατα, οι ουρές κτλ.
- Οι πανοπλίες [3-4], οι οποίες ήταν έτοιμες αλλά παρ' όλα αυτά ήθελαν πολύ επεξεργασία, η οποία έγινε μέσα από το blender (βλ. Ενότητα 3.4).

Κεφάλαιο 3ο: Δημιουργία Animation στο Blender

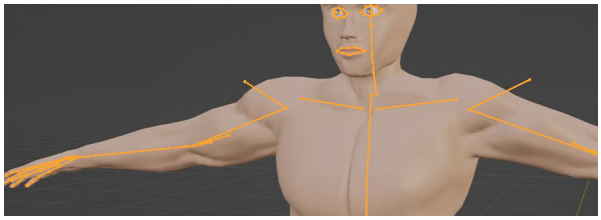
3.1 Εισαγωγή

Σε αυτό το κεφάλαιο εξηγώ πως έκανα το rigging του χαρακτήρα, δηλαδή την δημιουργία και την ενσωμάτωση του σκελετού στο μοντέλο, ώστε να μπορώ να δημιουργήσω τα animation καθώς και το weight painting του μοντέλου, με το οποίο όρισα πόσο θα παραμορφώνει το κάθε bone του σκελετού, τα διάφορα vertices του μοντέλου. Στην συνέχεια εξηγώ πως δημιούργησα το Idle animation που παίζει στις σκηνές του μενού στο Unity. Έπειτα στην επόμενη ενότητα εξηγώ πως επεξεργάστηκα τα έτοιμα μοντέλα των πανοπλιών, ώστε να προσαρμωστούν πάνω στο μοντέλο του χαρακτήρα και να μπορούν να χρησιμοποιηθούν με τα animation. Και τέλος εξηγώ τις ρυθμίσεις που έκανα για να κάνω export τα μοντέλα και τον σκελετό ώστε να τα περάσω μετά στο Unity.

3.2 Rigging Model & Weight Painting

Εδώ περιγράφω την διαδικασία του rigging του μοντέλου, δηλαδή την δημιουργία του σκελετού του μοντέλου με τον οποίο μπορείς να φτιάξεις τα animation, καθώς και δημιουργία των weights του μοντέλου, τα οποία ορίζουν τον βαθμό στον οποίο θα παραμορφώνεται το μοντέλο όταν κουνάς τον σκελετό.

Για το rigging, δημιούργησα έναν σκελετό (armature) πατώντας “Shift + A” και επέλεξα “Armature → Human (Meta-Rig)”. Από αυτόν το σκελετό διέγραψα τα bones που δεν χρειάζομαι και μετά προσάρμωσα τον σκελετό, για να μπούν τα bones στις σωστές θέσεις πάνω στο μοντέλο του χαρακτήρα.



Εικόνα 3.1: Προσαρμωγή Rig Στον Χαρακτήρα

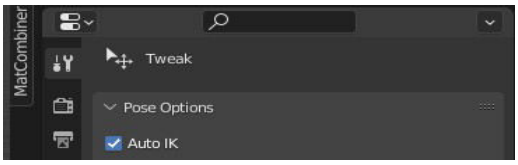
Στην συνέχεια, επέλεξα το μοντέλο και μετά τον σκελετό και πάτησα “Ctrl + P” για να ενώσω το μοντέλο με τον σκελετό με automatic weights, το οποίο προσθέτει μόνο του τα weights στον χαρακτήρα. Έπειτα μπήκα στο Weight Painting Mode, ενεργοποίησα την επιλογή “Auto-Normalize” από την καρτέλα “Active Tools and Workspace Setting” στο παράθυρο Properties και

διόρθωσα τα weights όπου χρειαζόταν, κυρίως με την χρήση των Draw, Blur και Smear tools του weight painting.

3.3 Δημιουργία Του Animation

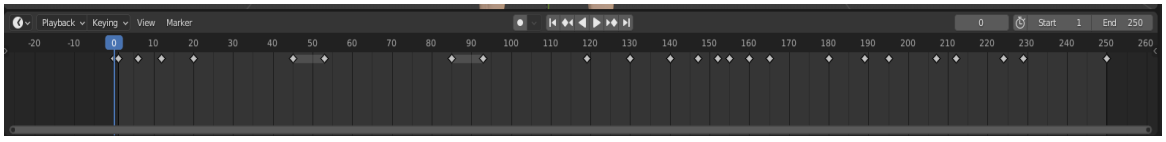
Εδώ περιγράφω την δημιουργία του Idle animation που έφτιαξα, το οποίο παίζει στις σκηνές του μενού στο Unity.

Αρχικά για να φτίαξω το animation, επέλεξα τον σκελετό και μπήκα στο Pose Mode και ενεργοποίησα την επιλογή "Auto-IK" από την πρώτη καρτέλα στο παράθυρο των Properties, με το οποίο επιλέγωντας ένα bone, κάνει και τα σχετικά bones με αυτό να κουνιούντε μαζί του, δηλαδη πχ αν επιλέξω το bone του καρπού και το κουνίσω, να κουνηθούν μαζί του και τα υπόλοιπα bones του χεριού.



Εικόνα 3.2: Pose Mode Auto IK

Έπειτα μέσα στο παράθυρο "Timeline", που βρίσκεται κάτω στο blender, έθεσα το start frame και το end frame για το animation. Στην συνέχεια, επέλεξα όλα τα bones του σκελετού (στο Pose Mode) και στο frame 0, αποθήκευσα το pose του σκελετού, πατώντας το πλήκτρο "I", για να ανοίξει το "Insert Keyframe Menu" και μετά επέλεξα "Location, Rotation & Scale", για να αποθηκεύσω όλο το transform των bones. Τέλος μετακινούσα σιγά σιγά τον σκελετό σε διάφορα στάδια του animation και αποθήκευα στα ενδιάμεσα frames, διάφορα keyframes με τον ίδιο τρόπο, μέχρι να ολοκληρωθεί το animation. Αν κάποιες κινήσεις δεν φαινόντουσαν σωστές, τότε πρόσθετα κάποια επιπλέον keyframe στο ενδιάμεσο, μέχρι οι κινήσεις να γίνουν όπως τις θέλω.



Εικόνα 3.3: Timeline & Keyframes

3.4 Επεξεργασία Έτοιμων Μοντέλων Για Χρήση Στα Animations

Εδώ εξηγώ πως επεξεργάστικα τα έτοιμα μοντέλα των πανοπλιών [3-4], ώστε να μπορούν να χρησιμοποιηθούν στο παιχνίδι με τα animation. Αυτά τα μοντέλα μπορεί να ήταν έτοιμα, αλλά ήθελαν πολύ επεξεργασία για να μπορέσουν να εφαρμοστούν επάνω στο project μου επειδή η γεωμετρία τους ήταν πολύ χαοτική και έπρεπε να την διορθώσω ώστε να γίνουν τα μοντέλα των πανοπλιών και πιο ελαφριά για το παιχνίδι, αλλά και να μπορεί να χρησιμοποιηθεί μαζί με τα animation του παιχνιδιού. Δηλαδή εκτός από το να διορθώσω απλά την γεωμετρία των μοντέλων έπρεπε και να τους βάλω και

ένα σκελετό για να τα φέρω στην ίδια πόζα με το μοντέλο του χαρακτήρα για να μπορώ να χρησιμοποιήσω τα animation πάνω σε αυτά. Στην συνέχεια χώρισα το μοντέλο του χαρακτήρα σε modular κομμάτια και μετά ένωσα τα κομμάτια της πανοπλίας, με τα modular κομμάτια και τον σκελετό του χαρακτήρα και έσβησα την γεωμετρία, η οποία δεν φαίνεται από κάτω, για εξοικονόμηση πόρων. Στη συνέχεια πρόσθεσα κάποια έξτρα κόκαλα στο σκελετό, για να μπορούν τα κομμάτια της πανοπλίας να κινούνται πιο ομαλά πάνω στον χαρακτήρα. Πιο αναλυτικά η διαδικασία ήταν η παρακάτω.

Αρχικά, χώρισα το μοντέλο του χαρακτήρα που έφτιαξα, σε κομμάτια (modular parts), ώστε να ενσωματώσω τα κομμάτια των πανοπλιών πάνω στον χαρακτήρα και μετά στο Unity να αλλάζω απλά τα modular κομμάτια πχ από το απλό γυμνό chest, στο chest που φοράει την πανοπλία. Αυτό έγινε επιλέγοντας τα edge loops γύρω από τα σημεία του χαρακτήρα, στα οποία θέλω να γίνει ο χωρισμός πχ γύρω από τους καρπούς ώστε να χωρίσω το modular κομμάτι που ονόμασα “hands” κτλ. Δημιούργησα σύνολο 5 κομμάτια (head, chest, hands, legs και feet).

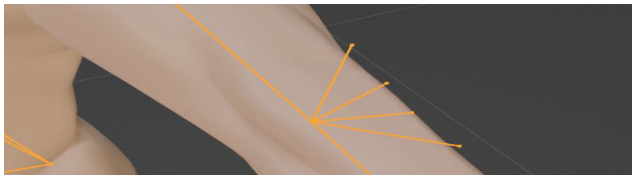
Στην συνέχεια μετέτρεψα τα faces στα μοντέλα των πανοπλιών από τρίγωνα σε τετράγωνα για να μειώσω τον αριθμό των faces και επίσης μπορώ έτσι να δημιουργήσω face loops. Μετά για να δημιουργήσω μερικά τέτοια face loops, προσπάθησα να διορθώσω την τοπολογία των μοντέλων, διαγράφοντας edges και προσθέτοντας καινούργια edges με την χρήση του knife tool.

Έπειτα επειδή οι πανοπλίες δεν ήταν φτιαγμένες με ένα ενιαίο mesh, αλλά ήταν είτε χωρισμένο σε πάρα πολλά μικρά κομμάτια (διαφορετικά mesh το κάθε ένα), είτε ήταν ένα mesh άλλα μέσα στο mesh, τα vertices δεν ήταν όλα ενωμένα μεταξύ τους, οπότε έπρεπε να τα ενώσω για να μπορούν να χρησιμοποιηθούν, διότι όταν το μοντέλο έχει πολλά κομμάτια, δεν λειτουργούν σωστά τα animation (τα weights πάνω στο rig ήταν τελείως λάθος και τα κομμάτια μετακινόντουσαν ξεχωριστά και έτσι έδειχναν πολύ άσχημα). Οπότε ομαδοποίησα τα διάφορα κομμάτια σε ομάδες, ανάλογα με τα modular κομμάτια του χαρακτήρα (πχ όλα τα μικρά κομμάτια από τα γάντια) και τα ένωσα για να δημιουργήσω ένα ενιαίο mesh. Για να τα ενώσω έπρεπε πρώτα να επιλέξω τα meshes στο Object Mode και να πατήσω “Ctrl + J” για να γίνουν ένα mesh και μετά επέλεξα τα κομμάτια και πατώντας το πλήκτρο “M” (στο Edit Mode) επέλεξα να τα ενώσω με βάση την απόσταση, ώστε να ενωθούν τα vertices που βρίσκονταν κοντά. Επίσης πολλά vertices, επειδή ήταν σε περίεργες θέσεις, έπρεπε να τα ενώσω χειροκίνητα με την χρήση των άλλων επιλογών του merge. Επίσης σε κάποιες περιπτώσεις έπρεπε να κάνω το αντίθετο, δηλαδή να χωρίσω ένα mesh σε διαφορετικά κομμάτια, αντί να τα ενώσω, το οποίο έγινε με την χρήση του rip region tool, όπως επίσης και πατώντας το πλήκτρο “P” στο Edit Mode και επιλέγοντας “Separate by loose parts”.

Στην συνέχεια επειδή οι πανοπλίες ήταν σε διαφορετική πόζα από το μοντέλο του χαρακτήρα, για να μπορέσουν να χρησιμοποιηθούν με τα animation που χρησιμοποίησα, έπρεπε να φέρω την πανοπλία στην ίδια πόζα με το μοντέλο του χαρακτήρα. Για να το κάνω αυτό, δημιούργησα ένα αντίγραφο του rig του χαρακτήρα και το προσάρμωσα πάνω στην πανοπλία, ώστε να αλλάξω έτσι την πόζα της πανοπλίας. Έπειτα έκανα parent το μοντέλο στο rig με automatic weights, για να δωθούν τα κύρια weights διόρθωσα τα vertex group που ανατέθηκαν στα bones, επειδή πολλά από αυτά ήταν λάθος (πχ. στο vertex group του αριστερού χεριού, υπήρχαν vertices από τα πόδια τις μπότες και έτσι

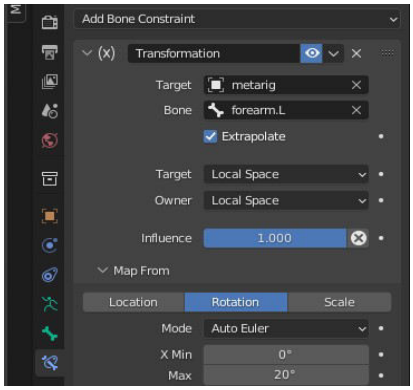
κουνώντας το χέρι παραμορφώνονταν και οι μπότες). Επίσης αυτά τα κομμάτια από την πανοπλία έπρεπε να τα κάνω parent στα bones ξεχωριστά γιατί όταν πας να κάνεις πολλά meshes μαζί, τα weights δεν γίνονται σωστα και επιπλέον πολλά weights τα έφτιαξα χειροκίνητα.

Έπειτα αφού έφερα τις πανοπλίες στην ίδια πόζα με τον χαρακτήρα, έκανα Apply το armature και το διέγραψα. Επίσης πρόσθεσα κάποια επιπλέον bones στον σκελετό του χαρακτήρα, για να κινούνται τα διάφορα κομμάτια της πανοπλίας πιο σωστά, χωρίς να λυγίζουν (πχ έβαλα bones στον ώμο και στους αγκώνες, ώστε τα μεταλικά κομμάτια της πανοπλίας που βρίσκονται σε αυτά τα σημεία να μην λυγίζουν πολύ, αλλά να μετακινούνται πιο ομαλά).



Εικόνα 3.4: Extra Bones Στον Αγκώνα

Για να φτίαξω αυτά τα επιπλέον bones, πρόσθεσα και κάποια Transformation bone constraints από την καρτέλα “Bone Constraints” στο παράθυρο “Properties”. Αυτά τα constraints περιορίζουν το πόσο θα μετακινούνται τα συγκεκριμένα bones (πόσες μοίρες), σε σχέση με την κίνηση ενός κεντρικού bone που όρισα (πχ τα επιπλέον bones του αγκώνα, κινούνται σε σχέση με το “forearm” bone του χαρακτήρα).



Εικόνα 3.5: Transformation Bone Constraint

Τέλος το ξαναέκανα parent πάνω στο armature του χαρακτήρα με automatic weights, διόρθωσα χειροκίνητα όλα τα weights που ήταν λάθος και ένωσα την πανοπλία με τα διάφορα κομμάτια του χαρακτήρα, διαγράφωντας ταυτόχρονα όσα faces του χαρακτήρα κρύβονταν κάτω από την πανοπλία.

3.5 Εξαγωγή Μοντέλων Σε Μορφή FBX Για Χρήση Στο Unity

Εδώ περιγράφω πως έγινε η εξαγωγή των μοντέλων από το blender, ώστε να τα εισάγω στο Unity.

Αρχικά έπρεπε να κάνω Apply τα modifiers για να μπορέσω να εξάγω τα shape keys (blendshapes στο Unity), αλλιώς τα blendshapes δεν εμφανίζονταν στο Unity. Έπειτα επέλεξα το μοντέλο του χαρακτήρα και τον σκελετό και πάτησα στο File → Export → FBX. Στην συνέχεια πάτησα στην επιλογή "Selected Objects", ώστε να κάνω export μόνο τα επιλεγμένα αντικείμενα. Μετά ενεργοποίησα το "Bake Animations", για να περαστούν τα animation και επίσης επέλεξα, μέσα στο μενού του Transform, τις επιλογές "Apply Unit" και "Use Space Transform" και τέλος έβαλα στον άξωνα Forward το "-Z forward" και στον άξωνα Up το "Y up" και έκανα export.

Η ίδια διαδικασία ακολουθήθηκε και για τα υπόλοιπα μοντέλα (αυτιά, πανοπλίες κτλ).

Εικόνα 3.6: Export Settings

Κεφάλαιο 4ο: Στήσιμο Παιχνιδιού στο Unity

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο αναλύω τα πάντα για το στήσιμο του παιχνιδιού, εκτός του κώδικα, ο οποίος θα συζητηθεί στο επόμενο κεφάλαιο. Θα εξηγήσω από πού προέρχονται τα assets του project μου, τα οποία περιλαμβάνουν και έτοιμα assets αλλά και asset τα οποία έφτιαξα στο Blender και στο Photoshop και πώς τα πέρασα στο Unity. Έπειτα αναφέρω όλα τα αντικείμενα που πρόσθεσα μέσα στις σκηνές του παιχνιδιού, όπως και τα διάφορα components που πρόσθεσα σε αυτά τα αντικείμενα. Επίσης εξηγώ το πώς έστησα το graphical user interface σε κάθε μία από τις σκηνές δηλαδή τα διάφορα κουμπιά, sliders, εικόνες και όλα τα άλλα γραφικά που μπορεί να βλέπει και να αλληλεπιδρά ο χρήστης. Τέλος εξηγώ το στήσιμο του State machine του παιχνιδιού το οποίο χρησιμοποιείται για την εναλλαγή των διαφόρων καταστάσεων, τόσο του παίκτη όσο και των εχθρών στο παιχνίδι.

4.2 Εισαγωγή Asset Στο Unity

4.2.1 Προέλευση Των Asset Του Project Μου

Κάποια asset που χρησιμοποίησα στο project μου τα έφτιαξα εγώ στο blender, κάποια τα πήρα από το Unity Store και κάποιες άλλες ιστοσελίδες και ένα animation το πήρα από το Mixamo.

Από το mixamo πήρα το animation του παίκτη για την κίνηση του σπαθιού όταν ο παίχτης επιτίθεται [5].

Από το Unity Store πήρα α) το Third Person Controller του Unity [6], β) το μοντέλο του τέρατος που πολεμάς μέσα στο παιχνίδι [7], γ) το σπαθί που κρατάει ο παίκτης [8], δ) κάποια γενικά αντικείμενα για το background στις σκηνές [9] και ε) κάποια εικονίδια για τα κουμπιά [10-11].

Επίσης τα μοντέλα για τις πανοπλίες τα πήρα έτοιμα από άλλες ιστοσελίδες [3-4], αλλά ήθελαν πολύ επεξεργασία για να μπορέσουν να εφαρμοστούν επάνω στο project μου επειδή η γεωμετρία τους ήταν πολύ χαοτική και έπρεπε να την διορθώσω ώστε να γίνουν τα μοντέλα των πανοπλιών και πιο ελαφριά για το παιχνίδι, αλλά και να μπορεί να χρησιμοποιηθεί μαζί με τα animation του παιχνιδιού. Δηλαδή εκτός από το να διορθώσω απλά την γεωμετρία των μοντέλων έπρεπε και να τους βάλω και ένα σκελετό για να τα φέρω στην ίδια πόζα με το μοντέλο του χαρακτήρα για να μπορώ να χρησιμοποιήσω τα animation πάνω σε αυτά. Μετά τα ένωσα με το μοντέλο και το σκελετό του χαρακτήρα και έσβησα την γεωμετρία, η οποία δεν φαίνεται από κάτω, για εξοικονόμηση πόρων. Στη συνέχεια πρόσθεσα κάποια έξτρα κόκαλα στο σκελετό, για να μπορούν τα κομμάτια της πανοπλίας να κινούνται πιο ομαλά πάνω στον χαρακτήρα.

Στην συνέχεια πήρα τα μοντέλα των μαλλιών από 3rd party ιστοσελίδες [12-13], όπως και το background image του μενού [14]

Και τέλος τα asset που δημιούργησα ο ίδιος από την αρχή ήταν α) το μοντέλο του χαρακτήρα, β) το animation του χαρακτήρα στην οθόνη του μενού και γ) όλα τα κομμάτια του σώματος που επιλέγει ο παίκτης στις οθόνες του μενού όπως είναι τα αυτιά, κέρατα, ουρές και τα λοιπά, εκτός των μαλλιών.

4.2.2 Εισαγωγή Δικών Μου Μοντέλων Από Το Blender Στο Unity

Για να εισάγω τα μοντέλα από το Blender στο Unity, απλά πήρα τα FBX αρχεία που δημιούργησα στο Blender (βλ. Ενότητα 3.5) και με drag & drop τα έβαλα μέσα στο Project Window του Unity. Η μόνη διαφορά με τα μοντέλα των πανοπλιών ήταν οτι έπρεπε πρώτα να περάσω τον φάκελο με τα Textures των μοντέλων και μετά να περάσω τα μοντέλα.

4.2.3 Εισαγωγή Έτοιμων Μοντέλων Από Το Unity Store

Για να εισάγω έτοιμα μοντέλα από το Unity Store, μπήκα στην ιστοσελίδα του Unity Store (https://assetstore.unity.com/) και επέλεξα το Asset που ήθελα και πάτησα Add to my assets. Μετά πάτησα “Open in Unity” το οποίο ανοίγει το “Package Manager” του Unity. Έπειτα από εκεί πάτησα

download και μόλις ολοκληρώθηκε το κατέβασμα, πάτησα Import, για να προστεθεί το package με τα Assets, μέσα στους φακέλους του Project window.

4.3 Ρυθμίσεις Των Asset Από Το Project Layout Window

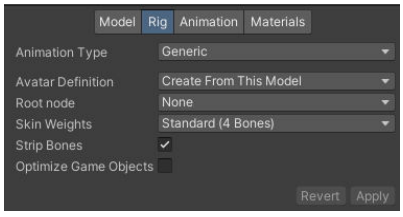
Εδώ εξηγώ πως έχω ρυθμίσει τα assets μόνο μέσα στο project layout window. Δεν αναλύω τις ρυθμίσεις των asset μέσα στις σκηνές. Αυτό το εξηγώ στην παρακάτω ενότητα της δημιουργίας των σκηνών του παιχνιδιού.

4.3.1 human_male_model_modular & human_male_model_modular_humanoid Assets

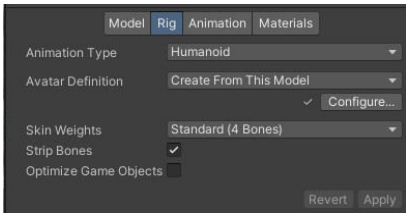
Μέσα στα assets στον φάκελο Custom Assets, υπάρχουν δύο εκδοχές του μοντέλου του χαρακτήρα που έφτιαξα, που ονομάζονται "human_male_model_modular" και "human_male_model_modular_humanoid".

Εικόνα 4.1: Generic & Humanoid Rig Models

Αυτά τα δύο είναι ακριβώς το ίδιο μοντέλο απλώς είναι ρυθμισμένα διαφορετικά μέσα στο Unity ώστε το πρώτο να μπορεί να χρησιμοποιηθεί στις σκηνές του μενού με το animation που έφτιαξα στο blender και το δεύτερο για χρήση στη σκηνή του παιχνιδιού ώστε να μπορεί να λειτουργήσει με τα animation του third person controller. Οι διαφορές τους είναι ότι το πρώτο για να χρησιμοποιηθεί στις σκηνές του μενού με το animation που έφτιαξα στο blender, η ρύθμιση του animation type στην καρτέλα rig του asset, πρέπει να είναι ρυθμισμένη ως generic, ενώ το άλλο για να χρησιμοποιηθεί στη σκηνή του παιχνιδιού με τα animation του third person controller, πρέπει αυτή η ρύθμιση να είναι στο humanoid.



Εικόνα 4.3: Generic Rig Type



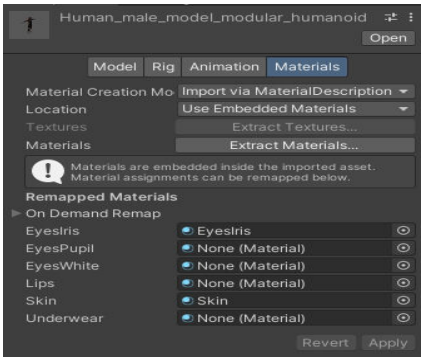
Εικόνα 4.2: Humanoid Rig Type

Έπειτα και οι δύο εκδοχές έχουν ρυθμισμένο το avatar definition στην καρτέλα rig του asset ως create from this model. Αυτό δημιουργεί το avatar μέσα στο asset (όταν το κάνεις expand), που έχει το εικονίδιο μιας φιγούρας ανθρώπου (βλ. Εικόνα 4.4 κάτω δεξιά εικονίδιο).

Εικόνα 4.4: Expanded Model & Avatar

Στην humanoid εκδοχή σε περίπτωση που τα mappings, πατώντας το κουμπί configure (βλ. Εικόνα 4.2), δεν είναι σωστά ρυθμισμένα αυτόματα, για να ταιριάζουν με το σκελετό του third person controller, έπρεπε να επιλεχθούν χειροκίνητα τα σωστά κόκαλα στις σωστές θέσεις.

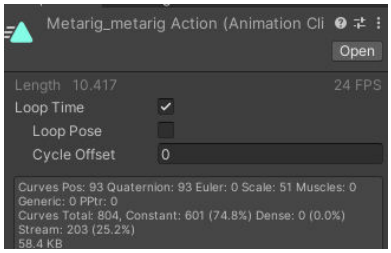
Έπειτα δημιούργησα duplicates των material του μοντέλου για όσα materials θέλω να μπορώ να επεξεργάζομαι και τα ανέθεσα μέσα στο remap materials της καρτέλας materials και στις δύο εκδοχές των asset.



Εικόνα 4.5: Remap Materials With Duplicates

Δημιούργησα duplicates των material, μόνο από το μοντέλο το οποίο ρύθμισα ως generic στην καρτέλα rig και όχι από το 2ο το οποίο ρύθμισα ως humanoid, εφόσον τα materials είναι ίδια και στα δύο και αυτό έγινε για δύο λόγους. Πρώτον και οι δύο εκδοχές παίρνουν αυτά τα material από το ίδιο ακριβώς σημείο ώστε να μην πρέπει ξεχωριστά να αλλάξω το χρώμα και στα δύο. Έτσι όταν αλλάζω, για παράδειγμα το χρώμα του skin material ενημερώνει αυτόματα και τα δύο μοντέλα. Δεύτερον επειδή τα αντικείμενα παιδιά του asset (δηλαδή ότι βρίσκεται μες στο asset όταν το κάνεις expand) είναι κλειδωμένα και δεν μπορώ να τα επεξεργαστώ απευθείας.

Τέλος δημιούργησα επίσης και ένα duplicate του animation που έφτιαξα στο blender από την generic εκδοχή ώστε να μπορεί και αυτό να χρησιμοποιηθεί έξω από την ιεραρχία του asset. Έπειτα ενεργοποίησα την επιλογή loop time του animation ώστε να παίζει το animation μες στις σκηνές του μενού επανειλημμένα. Αυτό το animation θα χρησιμοποιηθεί μέσα στο human male animator controller που δημιούργησα.



Εικόνα 4.6: Loop Time Inside Duplicate Animation

4.3.2 Human_Male_Animator_Controller Asset

Μέσα στα assets, στον φάκελο custom assets, πατώντας δεξί κλικ->Create->animator Controller, δημιούργησα ένα animator controller το οποίο ονόμασα Human_Male_Animator_Controller και το οποίο θα χρησιμοποιηθεί για να περάσω και να τρέξω το animation που έφτιαξα στο blender (συγκεκριμένα το duplicate animation που ανέφερα στην ενότητα 4.3.1), για χρήση στις σκηνές του μενού.

Εικόνα 4.7: Animator Controller

4.3.3 Δημιουργία Prefabs Assets Από Τα Μοντέλα Των Στοιχείων Του Χαρακτήρα

Τα μοντέλα των στοιχείων του χαρακτήρα είναι χωρισμένα σε φακέλους ανά κατηγορία (ears, horns, tails, hair) και μέσα σε κάθε έναν από αυτούς τους φακέλους υπάρχει ένας φάκελος που λέγεται prefabs ο οποίος περιέχει τα prefabs αυτών των μοντέλων.

Τα prefabs είναι προδιαμορφωμένα, επαναχρησιμοποιούμενα game objects αυτών των μοντέλων τα οποία διαμορφώθηκαν μέσα στη σκηνή ακόμα και σε συνδυασμό με άλλα στοιχεία της σκηνής και μετά τραβήχτηκαν και περάστηκαν στο project window ως prefab assets για μελλοντική χρήση, με έτοιμες ρυθμίσεις.

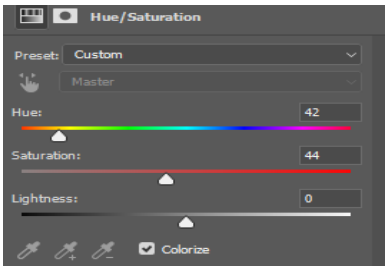
Η δημιουργία αυτών των prefabs έγινε ως εξής. Πρόσθεσα τα μοντέλα των στοιχείων του χαρακτήρα μέσα στην σκηνή και ρύθμισα το μέγεθος και την τοποθεσία τους πάνω στο μοντέλο του χαρακτήρα. Μετά δημιούργησα ένα empty game object ως παιδί του κόκαλου "face" του χαρακτήρα και το οποίο βρίσκεται στο κεφάλι του (για τα μοντέλα που θα βρίσκονται στο κεφάλι πχ αυτιά, κέρατα κτλ) και για τις ουρές έκανα το ίδιο αλλά στο κόκαλο “Spine.001”. Ονόμασα αυτό το empty game object "pivot", μηδένισα τη θέση του ως προς την θέση του κόκαλου και το χρησιμοποίησα ως parent καθενός στοιχείου ξεχωριστά για να δημιουργήσω τα prefabs τραβώντας αυτό το pivot μαζί με τα στοιχεία για να έχουν όλα τα στοιχεία το pivot ως κοινό point of origin.

Εικόνα 4.8: Pivot As Child Of Face To Create Body Parts Prefabs

Δεν πρόσθεσα ως παιδί όλα τα στοιχεία μαζί. Απλά πρόσθετα το ένα στοιχείο, έφτιαχνα το prefab και μετά το αφαιρούσα και πρόσθετα το επόμενο κ.ο.κ.. Αυτό βοήθησε στο script της αλλαγής των στοιχείων του χαρακτήρα, ώστε να προσθέτω δυναμικά τα στοιχεία στο runtime και να έρχονται όλα στη σωστή τους θέση, μηδενίζοντας απλά τη θέση του pivot αντί να πρέπει να θέσω διαφορετικές συντεταγμένες για κάθε στοιχείο.

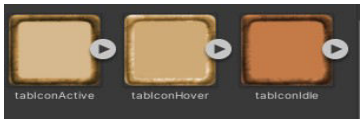
4.3.4 Δημιουργία Sprite Images Assets Στο Photoshop

Τα sprites που χρησιμοποίησα, τα επεξεργάστηκα μέσω του photoshop για να τους αλλάξω χρώμα. Αρχικά πήρα την εικόνα του μενού [14] και άνοιξα το παράθυρο των properties του hue/saturation και πάτησα το colorize για να χρωματίσω την εικόνα και μετά απλά άλλαξα τις ρυθμίσεις για να το κάνω στο χρώμα που ήθελα.



Εικόνα 4.9: Hue/Saturation Στο Photoshop

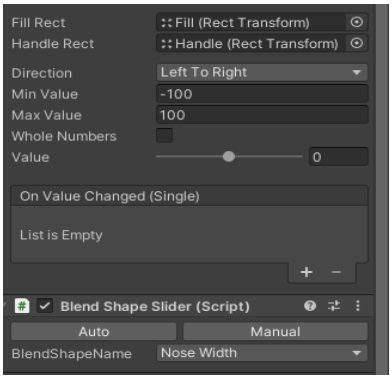
Στην συνέχεια έφτιαξα τα background των κουμπιών με την χρήση 2 images από το UI pack [10] (ένα image για το frame γύρω από το κουμπί και ένα για το εσωτερικό του) και τα έβαλα στο photoshop για να τα ενώσω και ακολούθησα την ίδια διαδικασία για να χρωματίσω το frame και το εσωτερικό του κουμπιού και δημιούργησα 3 παραλλαγές. Μία για idle button, μία για hover button και μία για active button. Και τέλος έκανα την ίδια διαδικασία και για το background του tab menu.



Εικόνα 4.10: 3 Παραλλαγές Για
Το State Των Κουμπιών

4.3.5 Blendshape Slider Prefab Asset

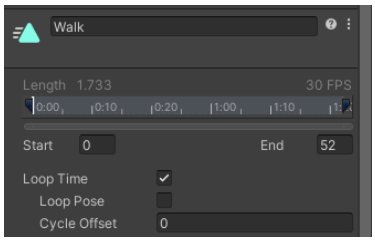
Δημιούργησα ένα prefab το οποίο ονόμασα Blendshape Slider και το αποθήκευσα μέσα στον φάκελο resources ο οποίος βρίσκεται μέσα στον φάκελο custom assets. Μετά ρύθμισα το prefab μέσα στη σκηνή να έχει ως minimum value το -100 και ως maximum value το 100 και του πρόσθεσα ως component το BlendShapeSlider script.



Εικόνα 4.11: BlendshapeSlider
Prefab Settings

4.3.6 Enemy Skeleton Animations Assets

Τα animation του enemy skeleton βρίσκονται διαδρομή Assets->FantasyMonster->Skeleton->Animations. Ενεργοποίησα την επιλογή loop time στην καρτέλα animation του κάθε asset εκτός από τα animation damage, die και knockback.



Εικόνα 4.12: Loop Time
Animation Setting For Skeleton

4.3.7 Skeleton Animator Controller Asset

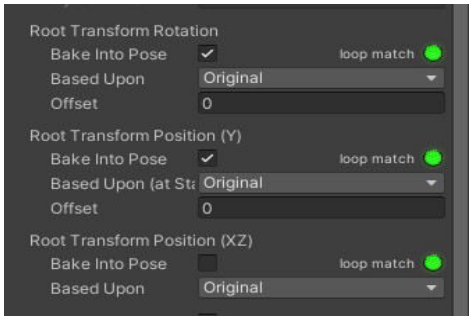
Μέσα στην διαδρομή Assets->FantasyMonster->Skeleton, πατώντας δεξί κλικ Create->animator controller, δημιούργησα ένα animator controller το οποίο ονόμασα SkeletonAnimatorController και το οποίο θα χρησιμοποιηθεί για την εναλλαγή των animation και των διάφορων σταδίων του enemy AI.

4.3.8 Player Animations Assets

Μέσα στο φάκελο των animation του third person controller πρόσθεσα το animation sword slash και έθεσα τις εξής ρυθμίσεις πάνω σε αυτό.

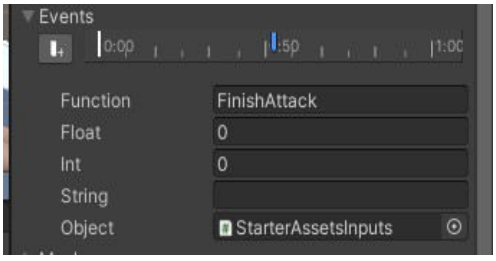
Στην καρτέλα rig επέλεξα humanoid animation type και στο avatar definition επέλεξα create from this model (βλ. Εικόνα 4.2).

Στην καρτέλα animation άλλαξα όλες τις ρυθμίσεις με τίτλο "based upon" σε "original" και επέλεξα τις δύο πρώτες επιλογές "Bake into Pose" για να παίζει πιο σωστά το animation με τον σκελετό του χαρακτήρα.



Εικόνα 4.13: Animation Tab Settings For SwordSlash Animation

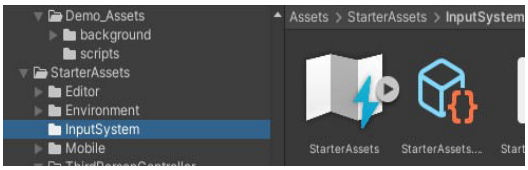
Επίσης στην ίδια καρτέλα, πήγα στα events και πρόσθεσα ένα event στην μέση του animation, στο frame που τελειώνει η κίνηση της επίθεσης του σπαθιού και έβαλα στο πεδίο function το όνομα της μεθόδου (FinishAttack) που θέλω να καλεί αυτόματα κάθε φορά που το animation φτάνει σε αυτό το frame και στο πεδίο object έβαλα την κλάση (StarterAssetsInputs) από την οποία θα καλεί αυτό το function.



Εικόνα 4.14: SwordSlash Event On Frame

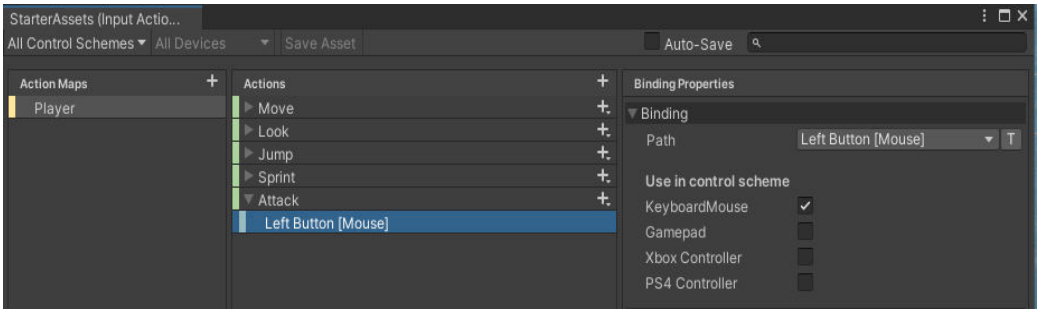
4.3.9 StarterAssets Input Action Asset

Άνοιξα το starter assets input action asset στο οποίο προσθέτεις και ρυθμίζεις τα διάφορα Action Maps του παιχνιδιού δηλαδή τον τρόπο που χειρίζεται ο παίκτης τον χαρακτήρα του χρησιμοποιώντας συγκεκριμένα keybindings.



Εικόνα 4.15: StarterAssets Input Action

Εκεί πρόσθεσα στα actions (πατώντας τον σταυρό) μία καινούργια επιλογή που την ονόμασα Attack. Μετά επέκτεινα το Attack και επέλεξα στο binding path το αριστερό κουμπί του ποντικιού και επέλεξα την επιλογή KeyboardMouse. Επίσης πατώντας πάνω στο Attack, επέλεξα στην επιλογή Action Type το Button.



Εικόνα 4.16: Create Attack Action With Left Mouse

4.4 Δημιουργία Σκηνών Και GUI

Για το στήσιμο των σκηνών το παιχνιδιού, χώρισα τις σκηνές σε 5 ομάδες σκηνών. Οι πρώτες τέσσερις ομάδες σκηνών έχουν η καθεμιά από 5 σκηνές όσες είναι δηλαδή και οι φυλές που μπορεί να επιλέξει ο παίκτης για το χαρακτήρα του. Ο παίκτης περνάει μόνο από μία σκηνή ανά ομάδα ανάλογα με ποια φυλή διάλεξε. Αυτό έγινε ώστε κάθε διαφορετική σκηνή ανά ομάδα να έχει είτε διαφορετικές επιλογές είτε διαφορετικό background ανάλογα με τη φυλή που επέλεξε ο παίκτης.

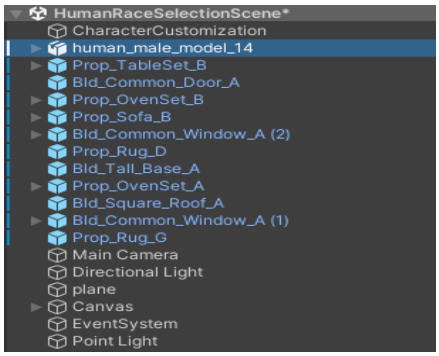
Οι ομάδες των σκηνών είναι οι σκηνές επιλογής φυλής, οι σκηνές επιλογής στοιχείων χαρακτήρα, οι σκηνές διαμόρφωσης προσώπου χαρακτήρα, οι σκηνές επιλογής ρούχων χαρακτήρα και τέλος η σκηνή παιχνιδιού, η οποία είναι ακριβώς ίδια ανεξαρτήτως ποια φυλή επέλεξε ο παίκτης. Με την έναρξη του παιχνιδιού ο παίκτης επιλέγει ποια φυλή θέλει και φορτώνεται η αντίστοιχη σκηνή με κάποια τυχαία χαρακτηριστικά της φυλής, μετά προχωράει στην επόμενη σκηνή στην οποία επιλέγει τα στοιχεία του χαρακτήρα ανάλογα με τη φυλή που διάλεξε και οι επιλογές του μενού μπορεί να είναι διαφορετικές με βάση τη φυλή και μετά περνάει στις επόμενες δύο ομάδες σκηνών

του μενού οι οποίες έχουν ίδιο menu αλλά είναι χωρισμένες σε διαφορετικές σκηνές μέσα στις ομάδες, σε περίπτωση που θέλω η σκηνή κάθε φυλής να έχει διαφορετικό background στο μενού.

Παρακάτω εξηγώ πως έχω στήσει κάθε σκηνή, σε ενότητες ανά ομάδα σκηνών. Αναλύω δηλαδή ποια gameobjects έχω προσθέσει στις σκηνές και πως έχει στηθεί το graphical user interface, δηλαδή τα διάφορα κουμπιά, εικόνες, sliders και τα λοιπά, καθώς και τα components που πρόσθεσα στα gameobjects. Στην ενότητα Ρύθμιση GameObject's Component UI Παιχνιδιού, αναλύω τις ρυθμίσεις που μπορεί να βάλει ο χρήστης μέσα σε αυτά τα components.

4.4.1 Σκηνή Επιλογής Φυλής

Οι σκηνές επιλογής φυλής χαρακτήρα έχουν τα εξής gameobject μέσα στη σκηνή:



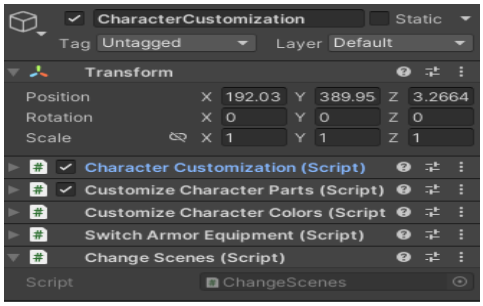
Εικόνα 4.17: GameObjects Της Σκηνής Επιλογής Φυλής

4.4.1.1 CharacterCustomization GameObject

Ένα empty gameobject το οποίο ονόμασα "CharacterCustomization" στο οποίο έχω περάσει σαν components τα διάφορα scripts που έφτιαξα και με το οποίο διαχειρίζομαι όλο το customization του χαρακτήρα σε όλες τις σκηνές του μενού. Αυτό το αντικείμενο περιέχει και τα scripts από τις επόμενες σκηνές και είναι αντικείμενο το οποίο περνιέται και στις επόμενες σκηνές χωρίς να καταστραφεί. Τα components που πέρασα σε αυτό το αντικείμενο είναι τα εξής:

- Το character customization script το οποίο διαχειρίζεται τα blendshapes του χαρακτήρα. Βρίσκεται στον φάκελο BlendShapes στα scripts.
- Το customize character parts script το οποίο περιέχει και διαχειρίζεται τα διάφορα μοντέλα των χαρακτηριστικών του χαρακτήρα, όπως αυτιά, κέρατα και τα λοιπά. Βρίσκεται στο φάκελο customize character parts μέσα στα scripts.
- Το customize character colors script το οποίο περιέχει και διαχειρίζεται τα χρώματα του χαρακτήρα, όπως το χρώμα του δέρματος, των ματιών και τα λοιπά. Το script βρίσκεται μέσα στο φάκελο customize character parts μέσα στα scripts.

- Το switch armor equipment script το οποίο περιέχει και διαχειρίζεται όλα τα ρούχα του χαρακτήρα. Το script βρίσκεται στον φάκελο switch armor μέσα στα scripts.
- Το change scences script το οποίο διαχειρίζεται την αλλαγή των σκηνών. Το script βρίσκεται μέσα στο φάκελο script.



Εικόνα 4.18: CharacterCustomization GameObject With Minimized Components

4.4.1.2 human_male_model_14 GameObject

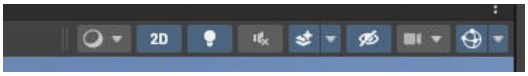
Το modular μοντέλο του χαρακτήρα το οποίο ονόμασα "human_male_model_14", μες την σκηνή και του πέρασα ως component, το RotateCharacter script, το οποίο χρησιμοποιείται για να περιστρέφεις τον χαρακτήρα στο μενού του customization, χρησιμοποιώντας το αριστερό κλικ του ποντικιού . Το asset πρέπει να έχει και το animator component, αλλά αυτό προστέθηκε από μόνο του, όταν επέλεξα avatar definition-> create from this model. (βλ. Ενότητα 4.3.1). Τέλος προσάρμοσα το position, το rotation και το scale του χαρακτήρα ώστε να έρθει μπροστά στην main camera της σκηνής, για να μπορείς να τον δεις στο runtime.

4.4.1.3 Γενικά Gameobjects

Οι σκηνές περιέχουν επίσης και μερικά γενικά gameobjects για να λειτουργήσουν. Αυτά είναι η main camera, η οποία είναι η κάμερα μέσα από την οποία θα βλέπει ο παίκτης το GUI της σκηνής, το directional light το οποίο είναι ο γενικός φωτισμός της σκηνής, το plane το οποίο είναι το πάτωμα πάνω στο οποίο στέκονται τα υπόλοιπα game objects και διάφορα άλλα background αντικείμενα για τη διακόσμηση της σκηνής.

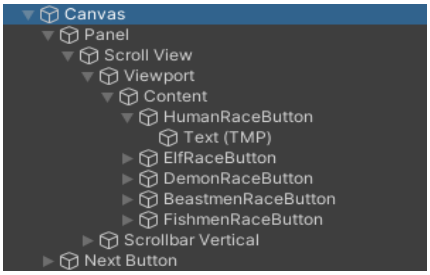
4.4.1.4 GUI Σκηνής Επιλογής Φυλής

Στις σκηνές επιλογής φυλής για να δημιουργήσω το graphical user interface του παιχνιδιού, έκανα δεξί κλικ μέσα στην ιεραρχία της σκηνής και επέλεξα UI -> canvas το οποίο εμπεριέχει όλα τα GUI του παιχνιδιού. Για να δω σωστά το canvas, πάτησα την επιλογή 2D, η οποία βρίσκεται πάνω δεξιά στο scene window του Unity και μετά επέλεξα το canvas και πάτησα το κουμπί “F”.



Εικόνα 4.19: 2D View

Μετά δημιούργησα μέσα στο canvas, δηλαδή ως παιδιά του, τα εξής UI αντικείμενα:



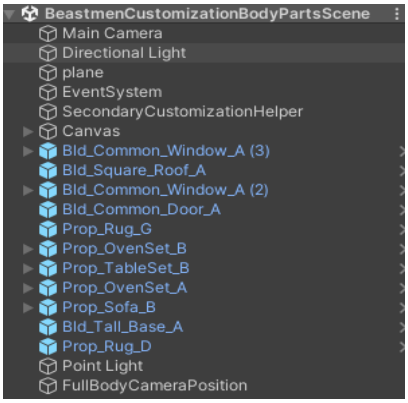
Εικόνα 4.20: Ιεραρχία Των UI του Canvas Σκηνή Επιλογής Φυλής

- Ένα panel, το οποίο θα περιέχει απλώς μία background εικόνα, για την διακόσμηση του μενού και την τοποθέτησα στην αριστερή μεριά του canvas και προσάρμοσα το μέγεθός της να εφαρμόζει στα πάνω και στα κάτω περιθώρια του canvas.
- Ένα scrollview το οποίο χρησιμοποιείται για να προσθέσω μέσα του όσα UI θέλω να μπορώ να πλοηγούμαι μεταξύ τους (καθέτως), μέσω ενός scrollbar. Έπειτα μέσα στο scrollview στην ιεραρχία της σκηνής, διέγραψα το object scrollbar horizontal επειδή χρειάζομαι μόνο το vertical scrollbar για πλοήγηση. Όταν επεκτείνεις το scrollview, μέσα στην ιεραρχία της σκηνής, υπάρχει ένα αντικείμενο που λέγεται viewport και μέσα σε αυτό υπάρχει ένα άλλο αντικείμενο που λέγεται content. Ότι προσθέτω μέσα στο content μπορεί να μετακινηθεί με το scrollbar. Τοποθέτησα το scrollview πάνω στο panel και το προσάρμοσα ώστε το πλάτος του να είναι ίδιο με του panel, αλλά το ύψος του να είναι μικρότερο ώστε ότι UI αντικείμενα βρίσκονται μέσα στο scrollview, όταν χειρίζεσαι το scrollbar, να μην ξεφεύγουν από τα όρια που θέλω μέσα στο background image του panel και να μην πατάνε πάνω στο frame της εικόνας. Επίσης το content μπορεί να περιέχει πάρα πολλά UI αντικείμενα μέσα του, αλλά αυτά που φαίνονται είναι μόνο όσα βρίσκονται μέσα στο περιθώριο του viewport, οπότε όταν θέλω να προσθέσω πολλά άλλα αντικείμενα στο content, τα οποία να φαίνονται μόνο πηγαίνοντας την scrollbar προς τα κάτω, πρέπει να μεγαλώσω τις διαστάσεις του content και ανάλογα με το μέγεθος του content (σε σύγκριση με το μέγεθος του viewport) προσαρμόζεται και η scrollbar. Αν οι διαστάσεις του content είναι μικρότερες ή ίσες με το viewport, τότε η scrollbar είναι σαν να είναι απενεργοποιημένη. Στην περίπτωση αυτών των σκηνών, έκανα το content ίδιο μέγεθος με τον viewport, επειδή είχα μόνο 5 κουμπιά και δεν χρειαζόμουν την scrollbar. Απλώς την άφησα σε περίπτωση που στο μέλλον θελήσω να προσθέσω παραπάνω επιλογές.

- Ένα button-TextMeshPro, το οποίο ονόμασα next button, και με το οποίο προχωράς στην επόμενη σκηνή, η οποία είναι η σκηνή επιλογής στοιχείων χαρακτήρα. Το τοποθέτησα κάτω δεξιά στο canvas.
- Μέσα στο content πρόσθεσα από ένα κουμπί (button-TextMeshPro) για κάθε φυλή του παιχνιδιού και τους έδωσα τις κατάλληλες ονομασίες.

Πέρα από τα ήδη υπάρχοντα components που δημιουργήθηκαν αυτόματα με αυτά τα αντικείμενα, το μόνο επιπρόσθετο component που πρόσθεσα, είναι το component vertical layout group που πρόσθεσα στο content ώστε να ρυθμίσει ομοιόμορφα το layout όλων των κουμπιών που βρίσκονται μέσα στο content.

4.4.2 Σκηνή Επιλογής Στοιχείων Χαρακτήρα



Εικόνα 4.21: GameObjects Της Σκηνής Επιλογής Στοιχείων Χαρακτήρα

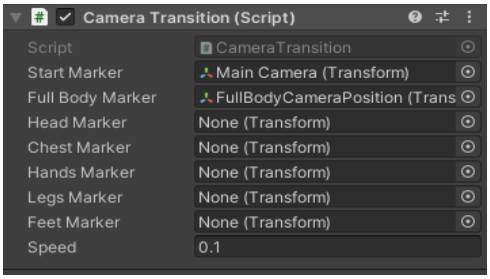
4.4.2.1 CharacterCustomization GameObject & human_male_model_14 GameObject

Αυτά τα δύο game objects είναι απαραίτητα για τη λειτουργία αυτών των σκηνών και χρησιμοποιούνται στις σκηνές αλλά δεν έχουν στηθεί απευθείας μέσα σε αυτές τις σκηνές χειροκίνητα, επειδή μεταφέρονται από την προηγούμενη σκηνή της επιλογής φυλής, με χρήση κώδικα. Αυτό σημαίνει ότι όλες οι σκηνές έχουν από κοινού αυτά τα δύο αντικείμενα και οποιεσδήποτε αλλαγές γίνονται στις προηγούμενες σκηνές, μεταφέρονται και στις επόμενες.

4.4.2.2 Γενικά Gameobjects

Οι σκηνές περιέχουν επίσης και μερικά γενικά gameobjects για να λειτουργήσουν, τα οποία είναι ίδια με αυτά της ενότητας 4.4.1.3. Τα αντικείμενα αυτά τα αντέγραψα από την πρώτη σκηνή της επιλογής φυλής και τα έκανα επικόλληση εδώ, ώστε να έρθουν ακριβώς στην ίδια θέση, με τη

διαφορά ότι σε αυτή τη σκηνή πρόσθεσα στη main camera, ως component, το CameraTransition script που έφτιαξα για να αλλάζει θέση η κάμερα σε Full Body View και ένα Empty GameObject που ονόμασα "FullBodyCameraPosition". Η διαδικασία είναι παρόμοια με την σκηνή της αλλαγής ρούχων, αλλά εδώ γίνεται μόνο για full body view (βλ. Ενότητα 4.4.4.1).



Εικόνα 4.22: Camera Transition Component
Σκηνής Επιλογής Στοιχείων Χαρακτήρα

4.4.2.3 SecondaryCustomizationHelper Gameobject

Το Secondary Customization Helper Gameobject είναι ένα empty gameobject που πρόσθεσα στην σκηνή και είναι βοηθητικό customization manager για την διαμόρφωση του μοντέλου του χαρακτήρα στις σκηνές του μενού. Έχει παρόμοια λειτουργία με το CharacterCustomization Gameobject που δημιούργησα στην σκηνή της επιλογής φυλής. Του έχω βάλει δύο scripts, ως components, το ChangeScenes script και το SecondaryCustomizationHelper script. Η δουλειά του είναι να διαχειρίζεται την αλλαγή των σκηνών στο μενού μέσω του ChangeScenes script και μέσω των μεθόδων του SecondaryCustomizationHelper script, να καλεί διάφορες άλλες μεθόδους σχετικά με την διαμόρφωση του χαρακτήρα, οι οποίες βρίσκονται μέσα στα scripts CustomizeCharacterParts και SwitchArmorEquipment και τα οποία βρίσκονται με την σειρά τους, μέσα στο Character Customization Gameobject της σκηνής για την επιλογή φυλής. Οι λόγοι που δημιούργησα αυτό το δευτερεύων βοηθητικό empty gameobject, το οποίο καλεί απλώς μεθόδους από τα scripts του Character Customization Gameobject, αντί να αντιγράψω απλά το κύριο CharacterCustomization Gameobject στις άλλες σκηνές, ήταν οι εξής:

- Πρώτον, όπως και για το Character Customization Gameobject, έγινε ώστε να είναι όλα τα scripts σχετικά με την διαμόρφωση του χαρακτήρα και την εναλλαγή των σκηνών, οργανωμένα στο ίδιο σημείο. Θα μπορούσα για παράδειγμα, να βάλω το ChangeScenes script, απευθείας μέσα στο next button, ως component του και μετά να περάσω στο πεδίο object του OnClick Unity Event, τον εαυτό του και να καλέσω έτσι την μέθοδο για αλλαγή σκηνής, αλλά θα έπρεπε για όλα τα κουμπιά σε κάθε σκηνή, να περνάω τα ίδια script ξανά και ξανά και μετά να καλώ τον εαυτό τους, ενώ έτσι απλώς πέρασα όλα τα scripts μια φορά, τα οργάνωσα σε

ένα σημείο και μετά απλώς περνάω παντού το ίδιο αντικείμενο σαν object του OnClick Unity Event και καλώ την κατάλληλη μέθοδο.

- Δεύτερον, επειδή ήθελα οι ρυθμίσεις όλων αυτών των scripts που βρίσκονται μέσα στο Character Customization Gameobject, να μπορούν να επεξεργαστούν και να ενημερωθούν, για παράδειγμα με καινούργια μοντέλα αυτιών μέσα στα πεδία τους, από μία κεντρική σκηνή, η οποία μετά θα περνάει στις επόμενες σκηνές, χωρίς να πρέπει να ενημερώνω τα ίδια πράγματα ξανά και ξανά για κάθε σκηνή ξεχωριστά.

- Τρίτον, επειδή ορισμένα script, κατά το στήσιμο τους, χρειάζονται να είναι παρών στην σκηνή, αντικείμενα του μοντέλου του χαρακτήρα, τα οποία όμως θα περαστούν αργότερα στην σκηνή κατά το runtime. Κάποια από αυτά τα αντικείμενα είναι για παράδειγμα το κόκαλο του προσώπου του χαρακτήρα που πρέπει να περαστεί ως πεδίο ώστε να μπορεί το script να το χρησιμοποιήσει ως parent εισαγωγή μοντέλων αυτιών, κεράτων κτλ, πάνω στον χαρακτήρα ή το material του δέρματος ώστε να μπορεί το script να αλλάξει χρώμα το δέρμα του χαρακτήρα. Έτσι χωρίς να είναι παρών το μοντέλο του χαρακτήρα μέσα στην σκηνή, δεν μπορώ να προσθέσω αυτά τα αντικείμενα στα πεδία των script κατά το στήσιμο της σκηνής.

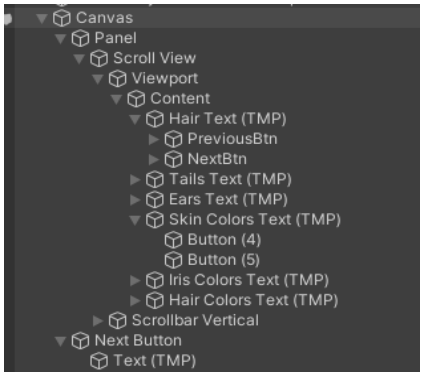
- Χωρίς να είναι το Character Customization Gameobject στις σκηνές (εφόσον όπως είπα περνιέται μετά κατά το runtime στις επόμενες σκηνές), δεν μπορώ να χρησιμοποιήσω ούτε τις διάφορες μεθόδους των script που βρίσκονται μέσα στο Character Customization Gameobject, επειδή τα κουμπιά θέλουν να τους προσθέσεις το object που περιέχει το script και μετά να καλέσεις τις μεθόδους αυτού του script. Οπότε με το secondary customization object, μπορώ να προσθέσω αυτό σαν object στο OnClick Event των κουμπιών και με το script του, να αναζητήσω το Character Customization Gameobject κατά το runtime, όταν αυτό θα έχει μεταφερθεί στην τρέχουσα σκηνή και να το χρησιμοποιήσω σαν μεσάζοντα για να καλέσω τις μεθόδους του.

- Και τέλος επειδή θέλω να κρατάω τις αλλαγές των τιμών των μεταβλητών στις επόμενες σκηνές, ώστε να διατηρούνται ότι αλλαγές έγιναν πάνω στον χαρακτήρα. Αν αντέγραφα το CharacterCustomization στις επόμενες σκηνές, όλες οι αλλαγές στα scripts θα μηδενίζονταν, ενώ περνώντας το ίδιο gameobject (με την εντολή DontDestroyOnLoad), το οποίο περιέχει τα scripts, στις επόμενες σκηνές, διατηρούνται όλες οι αλλαγές στις μεταβλητές.

Έτσι χρησιμοποιώντας το secondary customization helper, σε συνδυασμό με το CharacterCustomization, μπορώ να καλέσω όλες τις μεθόδους του CharacterCustomization στις επόμενες σκηνές, διατηρώντας όλα τα scripts συγκεντρωμένα σε ένα κεντρικό σημείο, για ευκολία τόσο χρήσης, όσο και ενημέρωσης τους, και διατηρώντας ταυτόχρονα τις μεταβλητές στις επόμενες σκηνές.

4.4.2.4 GUI Σκηνής Επιλογής Στοιχείων Χαρακτήρα

Στις σκηνές Επιλογής Στοιχείων Χαρακτήρα για να δημιουργήσω το graphical user interface του παιχνιδιού, δημιούργησα μέσα στο canvas τα εξής UI αντικείμενα:



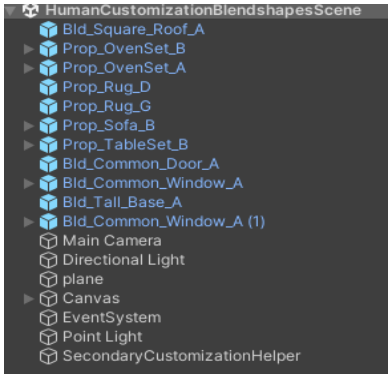
Εικόνα 4.23: Ιεραρχία Των UI του Canvas Σκηνή Επιλογής Στοιχείων Χαρακτήρα

- Ένα **panel**, το οποίο θα περιέχει απλώς μία background εικόνα. Η διαδικασία είναι ίδια όπως περιγράφεται στο GUI σκηνής επιλογής φυλής (βλ. Ενότητα 4.4.1.4).
- Ένα **scrollview** το οποίο χρησιμοποιείται για να προσθέσεις μέσα του όσα UI θέλεις να μπορείς να πλοηγείσαι μεταξύ τους (καθέτως), μέσω ενός scrollbar (βλ. Ενότητα 4.4.1.4 για περισσότερες λεπτομέριες σχετικά με το scrollview).
- Ένα **button-TextMeshPro**, το οποία ονόμασα **next button**, και με το οποίο προχωράς στην επόμενη σκηνή (σκηνή διαμόρφωσης προσώπου). Το next button το τοποθέτησα κάτω δεξιά στο canvas.
- Μέσα στο content πρόσθεσα κάποια **Text-TextMeshPro**, τα οποία είναι labels για διάφορες ομάδες κουμπιών και επίσης τα έθεσα ως parent object αυτών των ομάδων κουμπιών για να τα οργανώσω. Αυτές οι ομάδες κουμπιών χωρίζονται σε 2 κατηγορίες, την κατηγορία χρωμάτων και την κατηγορία στοιχείων χαρακτήρα.
- Μέσα σε αυτά τα labels που ανέφερα παραπάνω έβαλα ως child objects κάποια **button-TextMeshPro** για την αλλαγή χρωμάτων του χαρακτήρα και για αλλαγή των διάφορων στοιχείων πάνω στον χαρακτήρα (αυτιά, κέρατα κτλ), ανάλογα με το ποιά φυλή αφορά η κάθε σκηνή. Για όσα κουμπιά ανήκουν στις ομάδες κουμπιών της κατηγορίας χρωμάτων διέγραψα το text object τους και πρόσθεσα τόσα κουμπιά όσα και τα χρώματα που θέλω να υπάρχουν. Στις ομάδες κουμπιών της κατηγορίας στοιχείων χαρακτήρα, πρόσθεσα 2 κουμπιά ανά ομάδα, ένα previous και ένα next για σειριακή αλλαγή των στοιχείων.

Πέρα από τα ήδη υπάρχοντα components που δημιουργήθηκαν αυτόματα με αυτά τα αντικείμενα, πρόσθεσα τα εξής **components**:

- Στα **Text-TextMeshPro** της κατηγορίας χρωμάτων πρόσθεσα το component **Grid Layout Group**, ενώ σε αυτά της κατηγορίας στοιχείων χαρακτήρα πρόσθεσα το **Horizontal Layout Group**.

4.4.3 Σκηνή Διαμόρφωσης Προσώπου Χαρακτήρα



Εικόνα 4.24: GameObjects Της Σκηνής Διαμόρφωσης Προσώπου Χαρακτήρα

4.4.3.1 CharacterCustomization GameObject & human_male_model_14 GameObject

Εδώ ισχύει το ίδιο που ανέφερα στην ενότητα 4.4.2.1.

4.4.3.2 SecondaryCustomizationHelper Gameobject

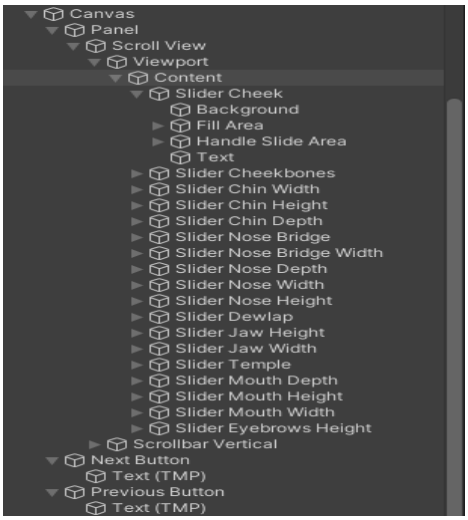
To Secondary Customization Helper Gameobject είναι ένα empty gameobject που πρόσθεσα στην σκηνή και είναι βοηθητικό customization manager. Εδώ ισχύει ότι αναφέρω στην ενότητα 4.4.2.3.

4.4.3.3 Γενικά Gameobjects

Οι σκηνές περιέχουν επίσης και μερικά γενικά gameobjects για να λειτουργήσουν, τα οποία είναι ίδια με αυτά της ενότητας 4.4.1.3. Τα αντικείμενα αυτά τα αντέγραψα από την πρώτη σκηνή της επιλογής φυλής και τα έκανα επικόλληση εδώ, ώστε να έρθουν ακριβώς στην ίδια θέση.

4.4.3.4 GUI Σκηνής Διαμόρφωσης Προσώπου Χαρακτήρα

Στις σκηνές Διαμόρφωσης Προσώπου Χαρακτήρα για να δημιουργήσω το graphical user interface του παιχνιδιού, δημιούργησα μέσα στο canvas τα εξής UI αντικείμενα:



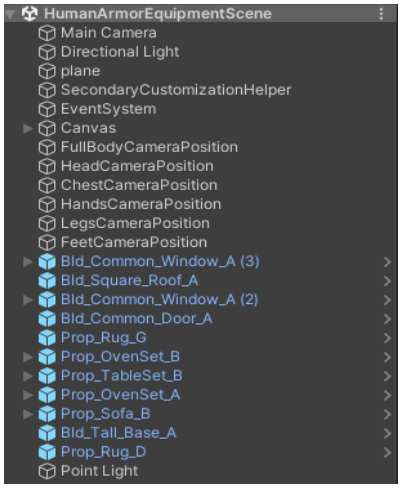
Εικόνα 4.25: Ιεραρχία Των UI του Canvas Σκηνή Διαμόρφωσης Προσώπου Χαρακτήρα

- Ένα **panel**, το οποίο θα περιέχει απλώς μία background εικόνα,. Η διαδικασία είναι ίδια όπως περιγράφεται στο GUI σκηνής επιλογής φυλής (βλ. Ενότητα 4.4.1.4).
- Ένα **scrollview** το οποίο χρησιμοποιείται για να προσθέσεις μέσα του όσα UI θέλεις να μπορείς να πλοηγείσαι μεταξύ τους (καθέτως), μέσω ενός scrollbar (βλ. Ενότητα 4.4.1.4 για περισσότερες λεπτομέρειες σχετικά με το scrollview).
- Δύο **button-TextMeshPro**, τα οποία ονόμασα **next button** και **previous button**, και με τα οποία είτε προχωράς στην επόμενη σκηνή (σκηνή επιλογής ρούχων χαρακτήρα) ή γυρνάς στην προηγούμενη (σκηνή επιλογής στοιχείων χαρακτήρα). Το next button το τοποθέτησα κάτω δεξιά στο canvas, και το άλλο κάτω αριστερά.
- Μέσα στο **content** πρόσθεσα **18 blendshape sliders**, όσα είναι δηλαδή και τα blendshapes από το μοντέλο του χαρακτήρα που έφτιαξα στο blender. Τα blendshapes με τα επιθέματα Min και Max, θεωρούνται ένα. Οπότε το slider έχει σαν αφετηρία τη μέση η οποία είναι το μηδέν και όταν το πηγαίνεις προς τα αριστερά ελέγχει τα blendshape με το επίθεμα Min, ενώ όταν το πηγαίνεις δεξιά ελέγχει τα blendshape με το επίθεμα Max. Η προσθήκη αυτών των sliders έγινε μέσω των scripts που έφτιαξα για τα blendshapes. Με αυτά τα scripts έφτιαξα επίσης και ένα custom editor το οποίο εμφανίζεται στο inspector, για την πιο εύκολη δημιουργία των sliders, με έτοιμες όλες τις απαραίτητες ρυθμίσεις. Σε αυτό το editor επιλέγεις το μοντέλο το οποίο περιέχει τα blendshapes, βάζεις τις ονομασίες των επιθεμάτων και σου βγάζει από μόνο του σε ένα drop-down μενού, με όλα τα blendshape του χαρακτήρα. Από εκεί επιλέγεις ένα-ένα τα blendshapes και πατάς Create Slider. Όλα τα

scripts που αφορούν την διαμόρφωση του χαρακτήρα στο μενού, ήθελα να βρίσκονται, ως components, μέσα στο ίδιο empty gameobject (δηλαδή το "CharacterCustomization" gameobject που πρόσθεσα στην αρχική σκηνή της επιλογής φυλής) το οποίο θα λειτουργεί ως customization manager και θα περνιέται από σκηνή σε σκηνή. Έτσι επειδή αυτά τα scripts για την διαχείριση και δημιουργία των blendshape sliders, βρίσκονται μέσα στο character customization game object της σκηνής επιλογής φυλής (το οποίο μεταφέρεται στην τρέχουσα σκηνή κατά το runtime), για να χρησιμοποιήσω το editor της δημιουργίας των sliders, έπρεπε να πάω στην σκηνή επιλογής φυλής, να τα δημιουργήσω εκεί και μετά τα αντιγράψω από τη σκηνή επιλογής φυλής στη σκηνή διαμόρφωσης προσώπου.

Πέρα από τα ήδη υπάρχοντα components που δημιουργήθηκαν αυτόματα με αυτά τα αντικείμενα, το μόνο επιπρόσθετο component που πρόσθεσα, είναι το component **vertical layout group** που πρόσθεσα στο content ώστε να ρυθμίσει ομοιόμορφα το layout όλων των sliders που βρίσκονται μέσα στο content.

4.4.4 Σκηνή Επιλογής Ρούχων Χαρακτήρα



Εικόνα 4.26: GameObjects Της Σκηνής Επιλογής Ρούχων Χαρακτήρα

Το "CharacterCustomization GameObject & human_male_model_14 GameObject" (βλ. Ενότητα 4.4.2.1), "Γενικά Gameobjects" (βλ. Ενότητα 4.4.1.3) και "SecondaryCustomizationHelper Empty Gameobject" (βλ. Ενότητα 4.4.2.3) είναι ακριβώς τα ίδια, με τη διαφορά ότι σε αυτή τη σκηνή πρόσθεσα στη main camera, ως component, το CameraTransition script που έφτιαξα για να αλλάζει θέση η κάμερα.

4.4.4.1 Camera Position Empty GameObjects

Τα Camera Position gameobjects, είναι empty gameobjects τα οποία δημιούργησα και τοποθέτησα μέσα στη σκηνή σε συγκεκριμένες τοποθεσίες. Μέσα σε αυτή τη σκηνή θέλω η κάμερα του παιχνιδιού να μπορεί να αλλάξει θέσεις κατά τη διάρκεια του runtime και να μετακινηθεί σε συγκεκριμένες τοποθεσίες για να μπορεί ο χρήστης να δει το μοντέλο του χαρακτήρα από διαφορετικές οπτικές γωνίες. Αυτά τα gameobjects δίνουν αυτές τις συγκεκριμένες και προκαθορισμένες τοποθεσίες, τις οποίες χρησιμοποιεί η κάμερα για να βρει πού θα μετακινηθεί.

Εικόνα 4.27: Empty Gameobjects For Camera Position

Μέσα στη σκηνή δημιούργησα 6 τέτοια αντικείμενα, τα οποία είναι τα εξής:

- Το **FullBodyCameraPosition**, το οποίο είναι για να δείχνει η κάμερα ολόκληρο το μοντέλο το χαρακτήρα από μακριά και αυτή είναι η default θέση όταν φορτώνεται αυτή η σκηνή.

- Το **HeadCameraPosition**, το οποίο είναι για να έρχεται η κάμερα πιο κοντά και να δείχνει το κεφάλι του χαρακτήρα.

- Το **ChestCameraPosition**, το οποίο είναι για να έρχεται η κάμερα κοντά και να δείχνει το πάνω μέρος του σώματος από τη μέση έως και το λαιμό περίπου.

- Το **HandsCameraPosition**, το οποίο είναι για να μετακινείται η κάμερα στα πλάγια και να δείχνει από κοντά τα χέρια.

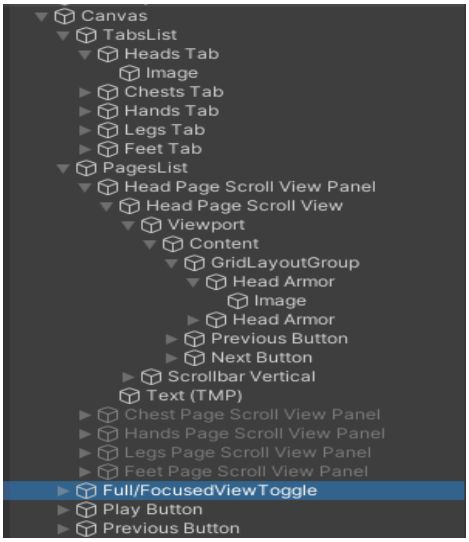
- Το **LegsCameraPosition**, το οποίο είναι για να μετακινείται η κάμερα και να δείχνει τα πόδια από κοντά.

- Το **FeetCameraPosition**, το οποίο είναι για να μετακινείται η κάμερα λίγο στα πλάγια και να δείχνει τα πόδια από τον αστράγαλο και κάτω.

Για να θέσω σωστά την τοποθεσία αυτών των αντικειμένων, μετακινούσα την main Camera στις θέσεις που την ήθελα και μετά αντέγραψα το position και το rotation της και τα περνούσα σε αυτά τα αντικείμενα. Οπότε όταν ήθελα η κάμερα να μετακινηθεί, έπαιρνα το transform αυτών των αντικειμένων για να βρει η κάμερα που θα πάει.

4.4.4.2 GUI Σκηνής Επιλογής Ρούχων Χαρακτήρα

Στις σκηνές Επιλογής Ρούχων Χαρακτήρα για να δημιουργήσω το graphical user interface του παιχνιδιού δημιούργησα μέσα στο canvas τα εξής UI αντικείμενα:



Εικόνα 4.28: Ιεραρχία Των UI του Canvas Σκηνή Επιλογής Ρούχων Χαρακτήρα

- Δύο **button-TextMeshPro**, τα οποία ονόμασα **play button** και **previous button**, και με τα οποία είτε αποθηκεύεις όλες τις αλλαγές και προχωράς στην επόμενη σκηνή (σκηνή παιχνιδιού) ή γυρνάς στην προηγούμενη (σκηνή διαμόρφωσης προσώπου). Το play button το τοποθέτησα κάτω δεξιά στο canvas, και το άλλο κάτω αριστερά.
- Ένα **empty gameobject**, το οποίο ονόμασα **TabsList**. Αυτό το αντικείμενο, με την χρήση ενός script που έφτιαξα (TabGroup script), χρησιμοποιείται για να ομαδοποιήσει 5 panels, τα οποία είναι απλά UI Panels αλλά δρουν ως Tab Buttons με τη χρήση ενός άλλου script που έφτιαξα (TabButton script), ώστε πατώντας σε αυτά τα Tab Buttons, να εναλλάσσονται οι καρτέλες του μενού των ρούχων ανάλογα με τα μέρη του σώματος του χαρακτήρα που ντύνεις.
- 5 **Tab Buttons**, ένα για κάθε μέρος του σώματος του χαρακτήρα και τα πέρασα ως παιδιά του TabsList. Αυτά τα Tab Buttons τα ονόμασα **α) Heads Tab, β) Chest Tab, γ) Hands Tab, δ) Legs tab και ε) Feet Tab**. Αυτά τα Tab Buttons, στην πραγματικότητα είναι panels αλλά λειτουργούν ως κουμπιά λόγω του Tab Button script που τους πρόσθεσα. Κάθε ένα από αυτά περιέχει ως παιδί του ένα αντικείμενο image το οποίο θα περιέχει ένα γενικό εικονίδιο για κάθε μέρος του σώματος του χαρακτήρα πχ εικονίδιο για γάντια, παπούτσια, παντελόνια και τα λοιπά.

- Ένα **empty gameobject**, το οποίο ονόμασα **PagesList** και το τοποθέτησα στην αριστερή μεριά του canvas και προσάρμοσα το μέγεθός του να εφαρμόζει στα πάνω και στα κάτω περιθώρια του canvas. Αυτό το αντικείμενο χρησιμοποιείται για να ομαδοποιήσει 5 panels, ως παιδιά του, τα οποία είναι απλά UI Panels, που θα περιέχουν απλώς μία background εικόνα, για την διακόσμηση του μενού και μέσα τους έχουν τα scrollviews που θα περιγράψω παρακάτω.
- 5 **panels**, ένα για κάθε μέρος του σώματος του χαρακτήρα και τα πέρασα ως παιδιά του PagesList. Αυτά τα panels τα ονόμασα **α) Head Page Scroll View Panel, β) Chest Page Scroll View Panel, γ) Hands Page Scroll View Panel, δ) Legs Page Scroll View Panel και ε) Feet Page Scroll View Panel**. Κάθε ένα από αυτά περιέχει ως παιδί του ένα UI scrollview το οποίο θα περιέχει όλο το περιεχόμενο των tabs για κάθε μέρος του σώματος του χαρακτήρα πχ το Grid Menu με τα εικονίδια για όλα τα μοντέλα από τα γάντια, παπούτσια, παντελόνια και τα λοιπά, τα οποία θα πατάει ο παίκτης για να αλλάζει ρούχα. Και τέλος κάθε panel έχει ως παιδιά του και από ένα UI text-textmeshpro, τα οποία περιέχουν ένα label του περιεχομένου (head, chest, hands, legs, feet).
- 5 **scrollviews**, ένα για κάθε μέρος του σώματος του χαρακτήρα, το οποίο χρησιμοποιείται για να προσθέσεις μέσα του όσα UI θέλεις να μπορείς να πλοηγείσαι μεταξύ τους (καθέτως), μέσω ενός scrollbar και τα τοποθέτησα πάνω στα αντίστοιχα panel. Η διαδικασία είναι ίδια με το scrollview της ενότητας 4.4.1.4. Αυτά τα scrollviews τα ονόμασα **Head Page Scroll View, Chest Page Scroll View, Hands Page Scroll View, Legs Page Scroll View και Feet Page Scroll View.**
- 5 **Text-TextMeshPro** τα οποία πρόσθεσα ως παιδιά σε κάθε ένα από τα πέντε panels που προανέφερα (Head Page Scroll View Panel κτλ) και περιέχουν απλώς ως κείμενο το όνομα του κάθε περιεχομένου (head, chest, hands, legs, feet).
- Από δύο ακόμα **button-TextMeshPro** ως παιδία του αντικειμένου content των scrollviews για κάθε ένα από τα scrollviews, τα οποία ονόμασα next button και previous button και τα οποία χρησιμοποιούνται για την σειριακή εναλλαγή των ρούχων, από τις λίστες όλων των ρούχων που υπάρχουν.
- Από ένα **empty gameobject** για κάθε ένα από τα scrollviews, και ορίστηκαν ως παιδιά του content των scrollviews. Αυτά τα αντικείμενα ονομάστηκαν **GridLayoutGroup**. Τα GridLayoutGroup χρησιμοποιήθηκαν ως TabGroup, με παρόμοια χρήση με το αντικείμενο TabsList που ανέφερα παραπάνω, με τη διαφορά ότι το TabsList περιείχε TabButtons για εναλλαγή UI Panels, ενώ αυτό περιέχει TabButtons για εναλλαγή ρούχων πάνω στο χαρακτήρα. Το ένα δηλαδή αλλάζει το μενού της οθόνης ενώ το άλλο αλλάζει ρούχα. Τα scripts TabGroup και TabButton αρχικά δημιουργήθηκαν για την εναλλαγή των tabs το μενού για αυτό και ονομάστηκαν έτσι αλλά επειδή το Grid Menu των ρούχων έχει παρόμοια λειτουργία με τα Tabs του μενού, με κάποιες μικρές διαφορές για αυτό χρησιμοποίησα τα ίδια script και απλώς πρόσθεσα κάποια έξτρα λειτουργία μες στο κώδικα αυτών των script.

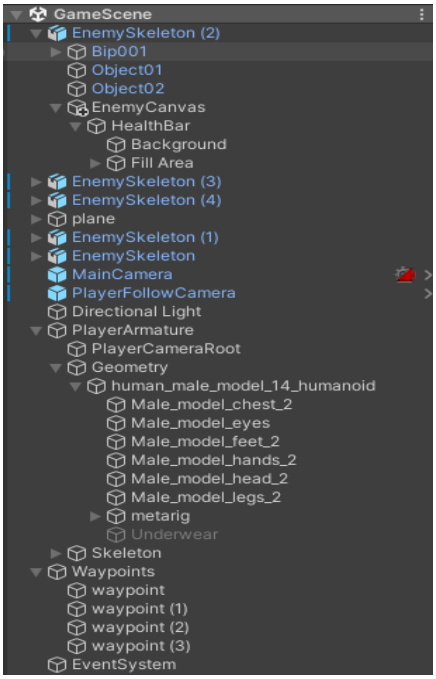
- Διάφορα **TabButtons** (τα οποία εδώ λειτουργούν ως Grid Buttons), ως παιδιά των GridLayoutGroup που βρίσκονται μέσα σε κάθε ένα από τα scrollviews. Ο αριθμός αυτών των Grid Buttons, μέσα στη σκηνή, εξαρτάται από το πόσα ρούχα υπάρχουν στο παιχνίδι. Κάθε Grid Button αντιστοιχεί σε ένα ρούχο και πατώντας το Grid Button περνιέται αυτό το ρούχο πάνω στο χαρακτήρα. Η ονομασία αυτών των Grid Button πρέπει να είναι συγκεκριμένη μέσα στη σκηνή και ανάλογη με τον τύπο του ρούχου, επειδή έτσι το έχω ορίσει μέσα στο script για να μπορεί να λειτουργεί ο κώδικας. Αν το ρούχο είναι για παράδειγμα παντελόνι, τότε πρέπει το Grid Button να ονομαστεί "Legs Armor". Όσα Grid Button και να έχεις για παντελόνια πρέπει όλα να ονομαστούν με ακριβώς τον ίδιο τρόπο. Αν το ρούχο είναι παπούτσι, τότε πρέπει να ονομαστεί "Feer Armor" και ούτω καθεξής. Επίσης κάθε ένα από αυτά περιέχει ως παιδί του ένα αντικείμενο image το οποίο θα περιέχει το εικονίδιο του αντίστοιχου ρούχου.
- Ένα **UI toggle**, το οποίο ονόμασα **Full/FocusedViewToggle** και βρίσκεται ως παιδί του canvas και χρησιμοποιείται ώστε όταν είναι επιλεγμένο να λειτουργούν οι focused κάμερες, αυτές δηλαδή που δείχνουν το μοντέλο από κοντά σε κάθε κομμάτι του σώματός του (head, chest, hands, legs και feet camera) και όταν το βγάζεις να λειτουργεί απλά zoomed out camera που δείχνει ολόκληρο το σώμα. (βλ. Ενότητα 4.4.4.1)

Πέρα από τα ήδη υπάρχοντα components που δημιουργήθηκαν αυτόματα με αυτά τα αντικείμενα, πρόσθεσα τα εξής components:

- Στο αντικείμενο TabsList πρόσθεσα ως components το vertical layout group για την ομοιόμορφη οργάνωση των tabs και το script που έφτιαξα το οποίο λέγεται TabGroup.
- Στα Heads Tab, Chest Tab, Hands Tab, Legs tab και Feet Tab πρόσθεσα ως components το vertical layout group για να κεντράρει το εικονίδιο που υπάρχει μέσα και το script που έφτιαξα το οποίο λέγεται TabButton.
- Στα GridLayoutGroup μέσα στο content κάθε scrollview, πρόσθεσα ως components το grid layout group για να οργανώσει τα grid buttons και το script που έφτιαξα το οποίο λέγεται TabGroup.
- Σε όλα τα Head Armor, Chest Armor, Hands Armor, Legs Armor και Feet Armor, πρόσθεσα ως components το script που έφτιαξα το οποίο λέγεται TabButton.

4.4.5 Σκηνή Παιχνιδιού

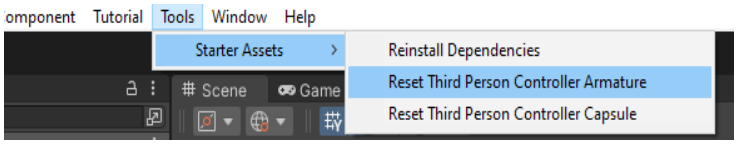
Η σκηνή παιχνιδιού είναι η σκηνή που ξεκινάει το παιχνίδι αφού φτιάξεις τον χαρακτήρα σου από τις σκηνές του μενού και έχει τα εξής Gameobjects:



Εικόνα 4.29: Gameobjects Της Σκηνής Παιχνιδιού

4.4.5.1 PlayerArmature, Main Camera & PlayerFollowCamera GameObjects

Αυτά τα 3 gameobjects πάνε πακέτο και τα πρόσθεσα στην σκηνή επιλέγοντας από το μενού του Unity την επιλογή Tools → Starter Assets → Third Person Controller Armature.

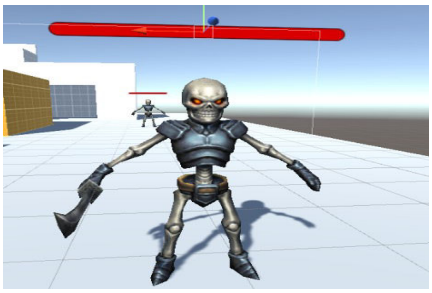


Εικόνα 4.30: Create ThirdPersonContoller Armature

Μετά έκανα Unpack το prefab “PlayerArmature” για να μπορώ να το επεξεργαστώ, πατώντας δεξί κλικ πάνω του και μετά επέλεξα “prefab → Unpack Completely” και στην συνέχεια, μέσα στο PlayerArmature υπάρχει ένα child gameobject, το οποίο λέγεται Geometry (βλ. Εικόνα 4.29) και μέσα του έχει το default μοντέλο του third person controller, το οποίο διέγραψα και το αντικατέστησα με το δικό μου μοντέλο, που το ονόμασα human_male_model_14_humanoid (το οποίο είναι το humanoid rig μοντέλο βλ. Ενότητα 4.3.1). Επίσης πρόσθεσα ένα tag στο “face” bone (“GameSceneFaceBone” tag) και ένα tag στο “Spine.001” bone (“GameSceneTailBone” tag) του humanoid μοντέλου, ώστε να ξεχωρίζουν από τα αντίστοιχα bone του generic rig μοντέλου μέσα στο script (βλ. Ενότητα 5.2.2.1 μεταβλητή newFace και newTailBone) και άλλαξα τα ονόματα των child object του human_male_model_14_humanoid, για να έχουν την κατάληξη “_2”, ώστε και αυτά να ξεχωρίζουν

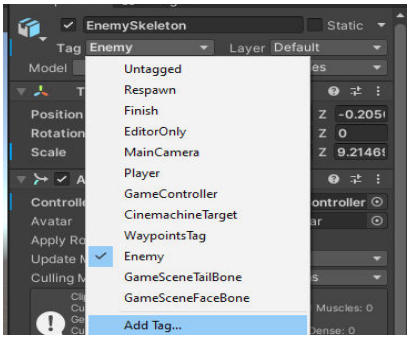
μέσα στο script (βλ. Εικόνα 4.29). Και τέλος έβαλα ως component του human_male_model_14_humanoid, το SetPlayableCharacterAppearance script.

4.4.5.2 EnemySkeleton GameObject



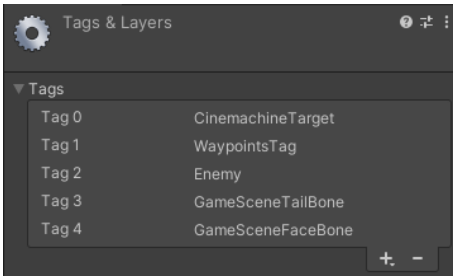
Εικόνα 4.31: Enemy Skeleton Model & HealthBar

Αυτό το gameobject, είναι το μοντέλο του εχθρού που πολεμάς μέσα στην σκηνή και είναι ένα έτοιμο μοντέλο που πήρα από το Unity Store. Μέσα στο EnemySkeleton πρόσθεσα ως child, ένα UI canvas και το ονόμασα EnemyCanvas (βλ. Εικόνα 4.29). Έπειτα μέσα στο EnemyCanvas, έβαλα ένα slider το οποίο ονόμασα HealthBar και λειτουργεί ως ένδειξη για το HP του εχθρού. Επίσης πρόσθεσα στο EnemySkeleton, ένα tag που έφτιαξα (μέσα στον inspector πατώντας το dropdown Tag που βρίσκεται πάνω πάνω και μετά την επιλογή Add Tag)



Εικόνα 4.32: How To Add Tag

και το ονόμασα Enemy για να πάρω αυτό το gameobject μέσα στον κώδικα με την εντολή FindGameObjectWithTag.



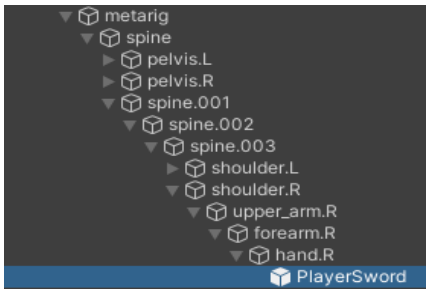
Εικόνα 4.33: GameScene Tags

Στην συνέχεια τα επιπλέον components που έβαλα σε αυτό το gameobject και τα παιδιά του ήταν τα εξής:

- Ένα Box Collider, το οποίο πρόσθεσα μέσα στο EnemySkeleton, το οποίο δημιουργεί το collision του μοντέλου και το χρησιμοποίησα κυρίως για να το αναγνωρίζω ως collision όταν επιτίθεται ο παίχτης με το σπαθί του.
- Ένα Nav Mesh Agent, το οποίο πρόσθεσα στο EnemySkeleton και χρησιμοποιείται για να πλοηγείται το AI του εχθρού μέσα στην σκηνή.
- Το Enemy script που έφτιαξα, το οποίο πρόσθεσα στο EnemySkeleton και χρησιμοποίησα για να μπορεί ο παίχτης να κάνει damage στον εχθρό και να τον σκοτώνει.
- Το FaceHPSliderAtPlayer script που έφτιαξα, το οποίο πρόσθεσα στο EnemyCanvas και κάνει το canvas να κοιτάει πάντα προς τον παίχτη για να μπορείς να δεις καλά το HP του εχθρού.

4.4.5.3 PlayerSword GameObject

Αυτό το gameobject, είναι το σπαθί με το οποίο πολεμάει ο παίχτης. Το μοντέλο το πήρα έτοιμο από το Unity Store και το πρόσθεσα ως child στο “hand.R” bone του σκελετού που βρίσκεται εσωτερικά του human_male_model_humanoid gameobject, ώστε να κινείται μαζί με τον χαρακτήρα και το τοποθέτησα στην παλάμη του χαρακτήρα.



Εικόνα 4.34: Player Sword Inside hand.R Bone

Όταν ο παίχτης επιτίθεται και ταυτόχρονα ακουμπίσει το collider του όπλου με το collider του εχθρού, τότε τρώει damage ο εχθρός.

Αυτό το gameobject, έχει τα εξής επιπρόσθετα components:

- Το WeaponCollider script που έφτιαξα, το οποίο χρησιμοποίησα για να κάνει damage ο παίχτης στον εχθρό, όταν συγκρούονται τα colliders του όπλου με του εχθρού.
- Ένα RigidBody component, για να μπορέσει να λειτουργήσει το collider.
- Ένα Box Collider, για να χρησιμοποιηθεί με το WeaponCollider script.

4.4.5.4 Waypoints GameObject

Αυτό το gameobject, είναι ένα Empty Gameobject, το οποίο ομαδοποιεί ως παιδιά του, μερικά άλλα Empty Gameobjects (βλ. Εικόνα 4.29), τα οποία δίνουν απλά τα σημεία που θα τριγυρνάει το Enemy AI, όταν βρίσκεται στο Walk State, δηλαδή όταν κάνει patrol. Μέσα στο κεντρικό waypoints

gameobject, μπορώ να προσθέσω όσα waypoints θέλω και να τα βάλω στις τοποθεσίες που θέλω μέσα στην σκηνή, ώστε ο εχθρός να παίρνει αυτές τις τοποθεσίες για να κάνει patrol. Επίσης πρόσθεσα στο waypoints, ένα tag που έφτιαξα (μέσα στον inspector πατώντας το dropdown Tag που βρίσκεται πάνω πάνω και μετά την επιλογή Add Tag) και το ονόμασα WaypointsTag για να πάρω αυτό το gameobject μέσα στον κώδικα με την εντολή FindGameObjectWithTag.

4.4.5.5 GameObjects Για Το Περιβάλλον Της Σκηνής

Μέσα στην σκηνή πρόσθεσα και μερικά gameobjects για να δημιουργήσω το περιβάλλον της σκηνής του παιχνιδιού και έβαλα όλα τα αντικείμενα του περιβάλλοντος, ως παιδιά μέσα σε ένα κεντρικό gameobject, το οποίο ήταν το “plane” (το έδαφος).

4.5 Ρύθμιση GameObject's Component UI Παιχνιδιού

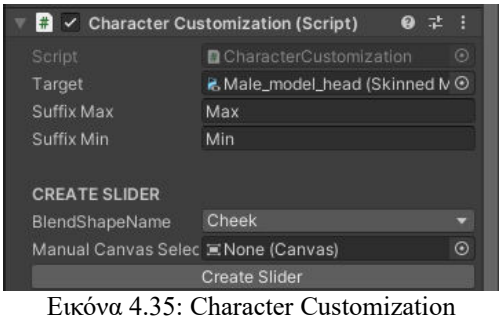
Εδώ αναλύω τις διάφορες ρυθμίσεις που μπορεί να βάλει ένας developer χρήστης (όχι ένας απλός παίκτης κατά τη διάρκεια του runtime) στα διάφορα component που πρόσθεσα (βλ. Ενότητα 4.4 Δημιουργία Σκηνής Παιχνιδιού Και GUI) για να προσαρμόσει το παιχνίδι. Αναλύω μόνο τις ρυθμίσεις που πείραξα, όχι όλες που υπάρχουν.

4.5.1 Component UI Σκηνής Επιλογής Φυλής

4.5.1.1 CharacterCustomization

Στα components του CharacterCustomization Gameobject έχω βάλει τις εξής ρυθμίσεις:

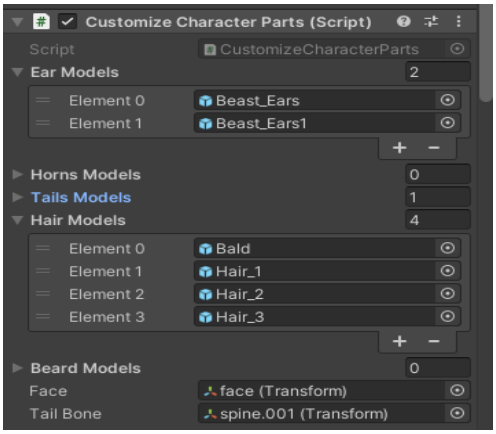
- Character Customization (Script): Στο πεδίο Target έβαλα το “Male_model_head (Skinned Mesh Renderer)” από την σκηνή. Στο πεδίο Suffix Max έβαλα την λέξη Max και στο Suffix Min έβαλα την λέξη Min.



Εικόνα 4.35: Character Customization (Script) Component

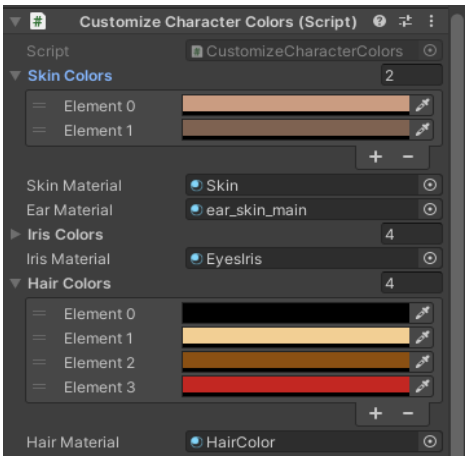
- Customize Character Parts (Script): Στις λίστες των μοντέλων πρόσθεσα όσα μοντέλα έχω φτιάξει, ανάλογα όμως με την φυλή του χαρακτήρα. Στην σκηνή με την φυλή ανθρώπου για παράδειγμα, έβαλα μόνο ανθρώπινα αυτιά και όσα πράγματα δεν έχει ένας άνθρωπος, όπως κέρατα και ουρές, τα άφησα κενά. Στην σκηνή των δαιμόνων έβαλα όλων των ειδών αυτιά,

επειδή ήθελα οι δαίμονες να έχουν αυτιά που μοιάζουν και με human και με elf και με beastmen κτλ. Επίσης τους πρόσθεσα και κέρατα. Οι beastmen έχουν μόνο αυτιά ζώου και ουρές κ.ο.κ. Τέλος έβαλα στο πεδίο Face το transform του "face" bone από τον σκελετό του χαρακτήρα στην σκηνή και στο πεδίο Tail Bone το transform του "Spine.001" bone .



Εικόνα 4.36: Customization Character Parts (Script) Component

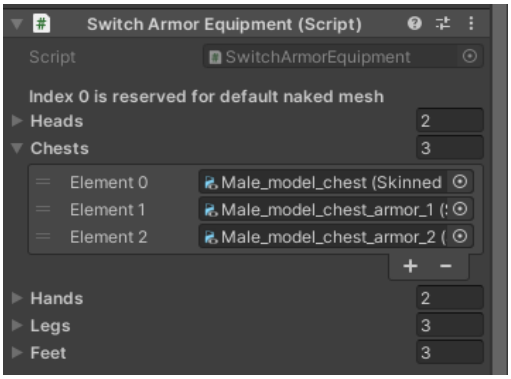
- Customize Character Colors (Script): Στις λίστες skin και iris colors πρόσθεσα κάποια χρώματα και έβαλα στα πεδία skin material και iris material, τα material με ονόματα skin και EyesIris που βρίσκονται στον φάκελο Assets → Custom Assets (αυτά που είχα δημιουργήσει σαν duplicates, όχι αυτά που βρίσκονται μέσα στην ιεραρχία του human_male_model). Και τέλος έβαλα στο πεδίο Ear Material, το material που βρίσκεται στον φάκελο Assets → Custom Assets → Ears (αυτό που είχα δημιουργήσει σαν duplicate, όχι αυτό που βρίσκεται μέσα στην ιεραρχία των μοντέλων των αυτιών).



Εικόνα 4.37: Customize Character Colors (Script) Component

- Switch Armor Equipment (Script): Στις λίστες των armor, έβαλα στην πρώτη θέση κάθε λίστας τα skinned mesh renderers του μοντέλου human_male_model_modular που βρίσκεται

στον φάκελο Assets → Custom Assets για να μπορεί να αλλάξει ο χαρακτήρας στο αρχικό μοντέλο χωρίς ρούχα και στα υπόλοιπα πεδία των λιστών έβαλα όσα armor έφτιαξα, τα οποία βρίσκονται μέσα στους υποφακέλους του φάκελου Assets → Custom Assets → Armors.

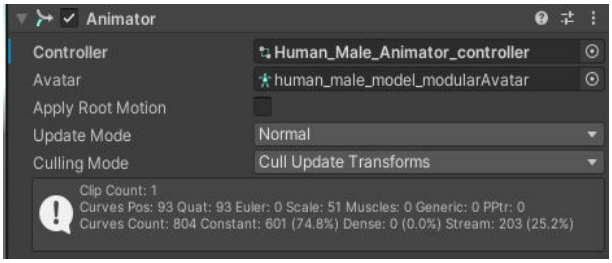


Εικόνα 4.38: Switch Armor Equipment (Script) Component

4.5.1.2 human_male_model_14

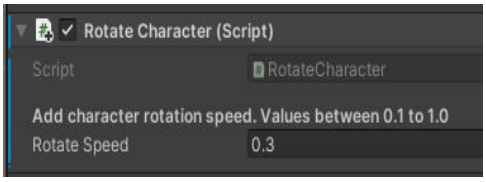
Στα components του human_male_model_14 Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Animator: Στο πεδίο Controller έβαλα το Human_Male_Animator_Controller που βρίσκεται στον φάκελο Assets → Custom Assets.



Εικόνα 4.39: human_male_model_14 Animator Component

- Rotate Character (Script): Στο πεδίο Rotate Speed έβαλα την τιμή 0,3.

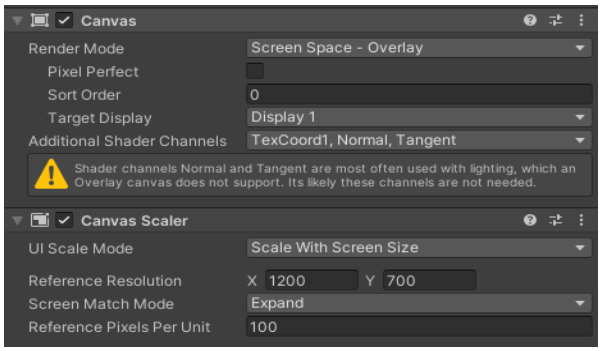


Εικόνα 4.40: Rotate Character (Script) Component

4.5.1.3 GUI Canvas

Στα components του Canvas Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Canvas: Στο dropdown Render Mode επέλεξα “Screen Space – Overlay”.
- Canvas Scaler: Στο dropdown UI Scale Mode επέλεξα Scale With Screen Size, στο Reference Resolution έβαλα 1200x700 και στο dropdown Screen Match Mode επέλεξα Expand.

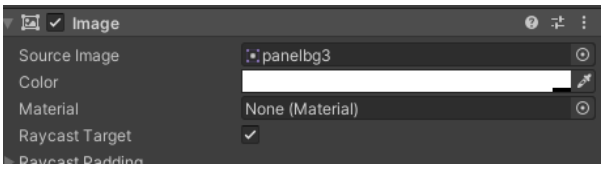


Εικόνα 4.41: Canvas & Canvas Scaler Components

4.5.1.4 GUI Panel

Στα components του Panel Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Image: Στο πεδίο Source Image έβαλα το sprite “panelbg3” που βρίσκεται στον φάκελο Assets → Custom Assets → Sprites.

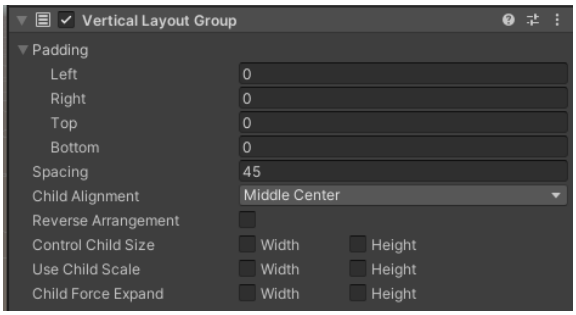


Εικόνα 4.42: Panel Image Component

4.5.1.5 GUI Content του Scroll View

Στα components του Content Gameobject που είναι child object του Scroll View έχω βάλει τις εξής ρυθμίσεις:

- Vertical Layout Group: Έβγαλα όλα τα checkboxes, έβαλα στο πεδίο spacing την τιμή 45 και επέλεξα στο Child Alignment το Middle Center.

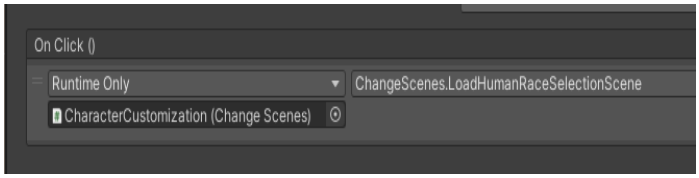


Εικόνα 4.43: Vertical Layout Group Component

4.5.1.6 GUI RaceButtons Του Content

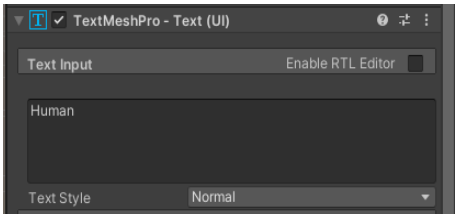
Στα components των HumanRaceButton, ElfRaceButton, DemonRaceButton, BeastmenRaceButton και FishmenRaceButton Gameobjects που είναι childen objects του Content Gameobject μέσα στο Scroll View, έχω βάλει τις εξής ρυθμίσεις:

- Button: Μέσα στα Onclick() Events του κάθε κουμπιού πέρασα ως αντικείμενο το CharacterCustomization gameobject που βρίσκεται στην σκηνή και επέλεξα μια από τις function LoadRaceSelectionScene() της αντίστοιχης φυλής από το script ChangeScenes (πχ LoadHumanRaceSelectionScene()).



Εικόνα 4.44: Onclick Event

- TextMeshPro – Text (UI): Στο πεδίο του κειμένου κάθε κουμπιού, έβαλα το όνομα της αντίστοιχης φυλής.



Εικόνα 4.45: Button's Text Area

4.5.1.7 GUI Next Button & Text (TMP) Του Next Button

Στα components των Next Button Gameobject & Text (TMP) Gameobject που είναι child object του Next Button Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Button: Μέσα στο Onclick() Event πέρασα ως αντικείμενο το CharacterCustomization gameobject που βρίσκεται στην σκηνή και επέλεξα μια από τις function LoadCustomizationBodyPartsScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes (πχ LoadHumanCustomizationBodyPartsScene()).
- TextMeshPro – Text (UI): Στο πεδίο του κειμένου έβαλα την λέξη Next.

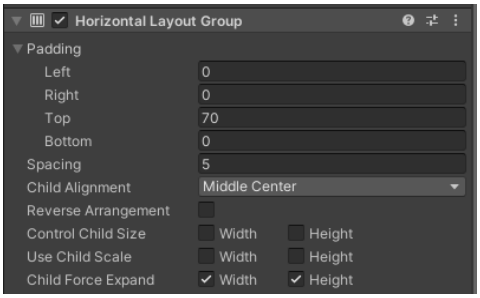
4.5.2 Component UI Σκηνής Επιλογής Στοιχείων Χαρακτήρα

Οι ρυθμίσεις των components των GUI Canvas και GUI Panel, είναι ακριβώς οι ίδιες με τις ενότητες 4.5.1.3 και 4.5.1.4. Επίσης οι ρυθμίσεις της main camera, είναι ίδιες με την ενότητα 4.5.4.1 με την διαφορά ότι εδώ πρόσθεσα μόνο το Start και FullBody Marker.

4.5.2.1 GUI Text-TextMeshPro του Content

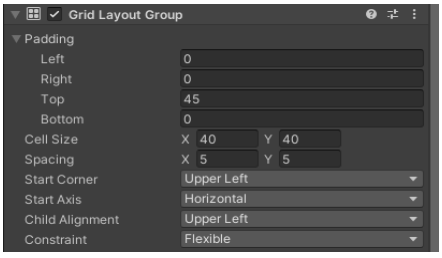
Στα components των Text-TextMeshPro Gameobject που είναι child object του Content και τα οποία χωρίζονται σε 2 κατηγορίες (κατηγορία χρωμάτων και κατηγορία στοιχείων χαρακτήρα), έχω βάλει τις εξής ρυθμίσεις:

- Horizontal Layout Group: Στο Top Padding έβαλα την τιμή 70, στο πεδίο spacing την τιμή 5 και επέλεξα στο Child Alignment το Middle Center.



Εικόνα 4.46: Horizontal Layout Group Component

- Grid Layout Group: Στο Top Padding έβαλα την τιμή 45, στο πεδίο spacing τις τιμές x=5, y=5 και στο Cell Size 40x40. Στα Child Alignment και Start Corner έβαλα το Upper Left, στο Start Axis έβαλα Horizontal και στο Constraint έβαλα Flexible.



Εικόνα 4.47: Grid Layout Group Component

Επίσης στα κουμπιά της κατηγορίας χρωμάτων διέγραψα το text object τους και τους άλλαξα το χρώμα.

4.5.2.2 GUI Button- TextMeshPro Των Text-TextMeshPro

Στα components των Next Button Gameobject, Previous Button Gameobject & τα Text (TMP) Gameobjects των δύο αυτών κουμπιών που είναι child object τους, έχω βάλει τις εξής ρυθμίσεις:

- Button (κατηγορία στοιχείων χαρακτήρα): Μέσα στο Onclick() Event και των δύο κουμπιών πέρασα ως αντικείμενο το SecondaryCustomizationHelper gameobject που βρίσκεται στην σκηνή και επέλεξα στο Next Button μια από τις function για αλλαγή στο επόμενο μοντέλο στοιχείων χαρακτήρα ανάλογα με την ομάδα που ήθελα πχ NextEarsModel() και μια από τις function για αλλαγή στο προηγούμενο μοντέλο στοιχείων χαρακτήρα ανάλογα με την ομάδα που ήθελα πχ PreviousHornsModel().
- Button (κατηγορία χρωμάτων): Μέσα στο Onclick() Event και των δύο κουμπιών πέρασα ως αντικείμενο το SecondaryCustomizationHelper gameobject που βρίσκεται στην σκηνή και επέλεξα στο Next Button μια από τις function LoadCustomizationBlendshapesScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes (πχ LoadHumanCustomizationBlendshapesScene()), ενώ στο Previous Button μια από τις function LoadRaceSelectionScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes.
- TextMeshPro – Text (UI): Στο πεδίο του κειμένου έβαλα την λέξη Next στο ένα και την λέξη Previous στο άλλο.

4.5.2.3 GUI Next Button, Previous Button & Text (TMP) Των Button

Στα components των Next Button Gameobject, Previous Button Gameobject & τα Text (TMP) Gameobjects των δύο αυτών κουμπιών που είναι child object τους, έχω βάλει τις εξής ρυθμίσεις:

- Button: Μέσα στο Onclick() Event και των δύο κουμπιών πέρασα ως αντικείμενο το SecondaryCustomizationHelper gameobject που βρίσκεται στην σκηνή και επέλεξα στο Next Button μια από τις function LoadCustomizationBlendshapesScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes (πχ LoadHumanCustomizationBlendshapesScene()), ενώ στο Previous Button μια από τις function LoadRaceSelectionScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes .
- TextMeshPro – Text (UI): Στο πεδίο του κειμένου έβαλα την λέξη Next στο ένα και την λέξη Previous στο άλλο.

4.5.3 Component UI Σκηνής Διαμόρφωσης Προσώπου Χαρακτήρα

Οι ρυθμίσεις των components των GUI Canvas και GUI Panel, είναι ακριβώς οι ίδιες με τις ενότητες 4.5.1.3 και 4.5.1.4.

4.5.3.1 GUI Content του Scroll View

Στα components του Content Gameobject που είναι child object του Scroll View έχω βάλει τις εξής ρυθμίσεις:

- Vertical Layout Group: Έβγαλα όλα τα checkboxes, έβαλα στο πεδίο spacing την τιμή 40 και επέλεξα στο Child Alignment το Middle Center.

4.5.3.2 GUI Next Button, Previous Button & Text (TMP) Των Button

Στα components των Next Button Gameobject, Previous Button Gameobject & τα Text (TMP) Gameobjects των δύο αυτών κουμπιών που είναι child object τους, έχω βάλει τις εξής ρυθμίσεις:

- Button: Μέσα στο Onclick() Event και των δύο κουμπιών πέρασα ως αντικείμενο το SecondaryCustomizationHelper gameobject που βρίσκεται στην σκηνή και επέλεξα στο Next Button μια από τις function LoadArmorEquipmentScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes (πχ LoadHumanArmorEquipmentScene()), ενώ στο Previous Button μια από τις function LoadCustomizationBodyPartsScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes .
- TextMeshPro – Text (UI): Στο πεδίο του κειμένου έβαλα την λέξη Next στο ένα και την λέξη Previous στο άλλο.

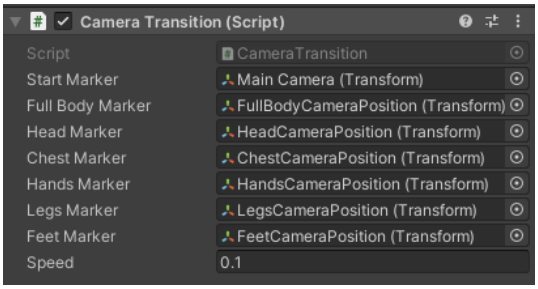
4.5.4 Component UI Σκηνής Επιλογής Ρούχων Χαρακτήρα

Οι ρυθμίσεις των components του GUI Canvas, είναι ακριβώς οι ίδιες με την ενότητα 4.5.1.3.

4.5.4.1 Main Camera

Στα components του Main Camera Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Camera Transition (Script): Στο πεδίο Start Marker έβαλα την main camera από την σκηνή και στα υπόλοιπα Markers (Full Body Marker, Head Marker κτλ) έβαλα τα αντίστοιχα CameraPosition objects από την σκηνή, με τις αντίστοιχες ονομασίες (FullBodyCameraPosition, HeadCameraPosition κτλ). Και τέλος στο πεδίο του Speed έβαλα την τιμή 0,01.

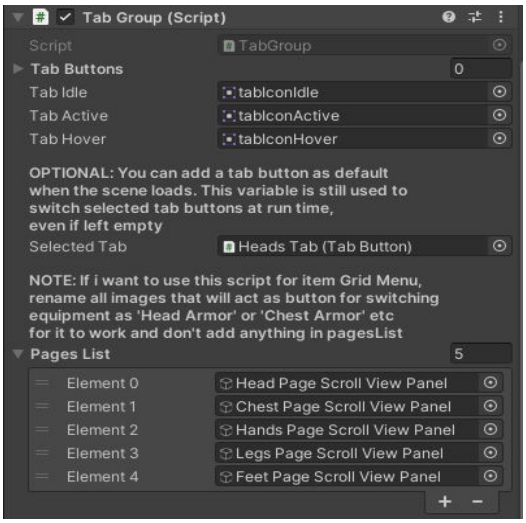


Εικόνα 4.48: Camera Transition (Script)
Component For Armor View

4.5.4.2 GUI TabsList

Στα components του TabsList Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Image: Στο Source Image έβαλα το sprite tabPanelIcon που βρίσκεται στον φάκελο Assets → Custom Assets → Sprites.
- Vertical Layout Group: Επέλεξα στο Child Alignment το Middle Center.
- Tab Group (Script): Στα πεδία Tab Idle, Tab Active και Tab Hover, έβαλα τα sprites tabIconIdle, tabIconActive και tabIconHover αντίστοιχα, τα οποία βρίσκονται στον φάκελο Assets → Custom Assets → Sprites. Στο πεδίο Selected Tab έβαλα το Heads Tab (Tab Button) από την σκηνή. Και τέλος στην λίστα των Pages List, έβαλα τα 5 Page Scroll View Panel για τα head, chest, hands, legs και feet αντίστοιχα.

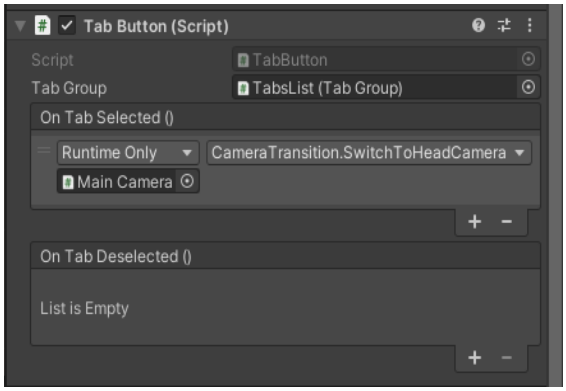


Εικόνα 4.49: Tab Group (Script) Component
For Tab Menu Use

4.5.4.3 GUI Heads Tab, Chests Tab, Hands Tab, Legs Tab, Feet Tab

Στα components των Heads Tab, Chests Tab, Hands Tab, Legs Tab και Feet Tab Gameobject, που είναι child object του TabsList, έχω βάλει τις εξής ρυθμίσεις:

- Image: Στο πρώτο Tab (Heads Tab) έβαλα στο πεδίο Source Image το sprite tabIconActive, ως προεπιλογή, ενώ στα υπόλοιπα tabs έβαλα το sprite tabIconIdle.
- Vertical Layout Group: Επέλεξα στο Child Alignment το Middle Center.
- Tab Button (Script): Στο πεδίο Tab Group έβαλα το TabsList (Tab Group) gameobject από την σκηνή και στο OnTabSelected() Event όλων των tabs, έβαλα το main camera gameobject από την σκηνή και επέλεξα την αντίστοιχη SwitchToCamera() Function της κλάσης CameraTransition ανάλογα με το tab, δηλαδή SwitchToHeadCamera για το Heads Tab, SwitchToChestCamera για το Chests Tab κ.ο.κ.

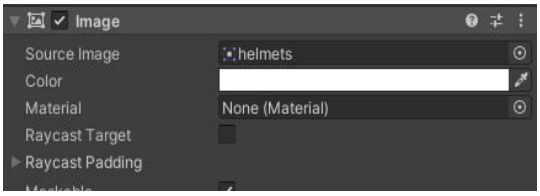


Εικόνα 4.50: Tab Button (Script) Component

4.5.4.4 GUI Image Των Tabs

Στα components των Image Gameobject που είναι child object των Heads Tab, Chests Tab, Hands Tab, Legs Tab και Feet Tab Gameobject, έχω βάλει τις εξής ρυθμίσεις:

- Image: Στο πεδίο Source Image κάθε Tab έβαλα το αντίστοιχο generic sprite που βρίσκεται στον φάκελο των sprites (helmets, chest, gloves, pants και boots) και απενεργοποίησα την επιλογή Raycast Target.



Εικόνα 4.51: Image Component Of Tab Button

4.5.4.5 GUI Page Scroll View Panels

Στα components των Head Page Scroll View Panel, Chest Page Scroll View Panel, Hands Page Scroll View Panel, Legs Page Scroll View Panel και Feet Page Scroll View Panel Gameobjects, έχω βάλει τις εξής ρυθμίσεις:

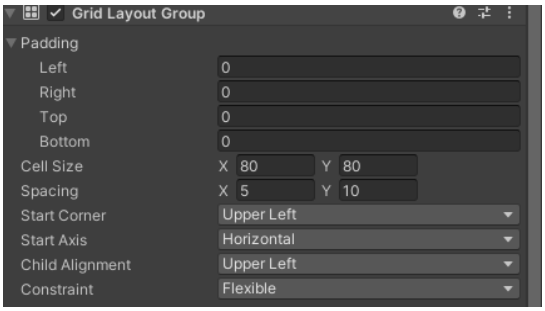
- Image: Στο πεδίο Source Image έβαλα το sprite “panelbg3” που βρίσκεται στον φάκελο Assets → Custom Assets → Sprites.

Επίσης απενεργοποίησα όλα τα Panels εκτός του πρώτου, μέσα από το παράθυρο του inspector, βγάζοντας την επιλογή του checkbox πάνω αριστερά στον inspector, επειδή θέλω να εμφανίζεται μονο ένα όταν φορτώσει η σκηνή και μετά θα εναλλάσσονται μέσω του script.

4.5.4.6 GUI GridLayoutGroup

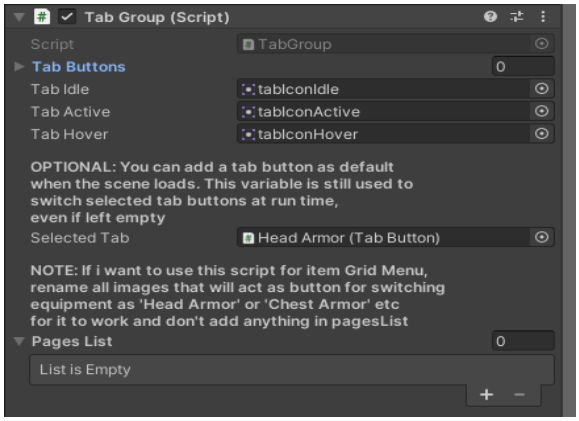
Στα components των 5 GridLayoutGroup Gameobject, που είναι child object του Content σε κάθε ένα από τα Scrollview, έχω βάλει τις εξής ρυθμίσεις:

- Grid Layout Group: Στο πεδίο Cell Size έβαλα x=80 και y=80. Στο πεδίο Spacing έβαλα x=5 και y=10. Στο Start Corner έβαλα Upper Left. Στο Start Axis έβαλα Horizontal. Στο Child Alignment επέλεξα το Upper Left. Και τέλος στο Constraint επέλεξα Flexible.



Εικόνα 4.52: Grid Layout Group Component

- Tab Group (Script): Στα πεδία Tab Idle, Tab Active και Tab Hover, έβαλα τα sprites tabIconIdle, tabIconActive και tabIconHover αντίστοιχα, τα οποία βρίσκονται στον φάκελο Assets → Custom Assets → Sprites. Στο πεδίο Selected Tab έβαλα το Head Armor (Tab Button) από την σκηνή. Και τέλος στην λίστα των Pages List, δεν έβαλα τίποτα επειδή εδώ το script έχει λειτουργία grid menu για αλλαγή armor και όχι tab menu για αλλαγή panel.

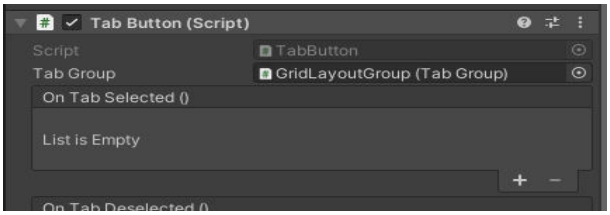


Εικόνα 4.53: Tab Group (Script) For Grid Menu Use

4.5.4.7 GUI Head Armor, Chests Armor, Hands Armor, Legs Armor, Feet Armor

Στα components των Heads Armor, Chests Armor, Hands Armor, Legs Armor και Feet Armor Gameobject, που είναι child object των 5 GridLayoutGroup, έχω βάλει τις εξής ρυθμίσεις:

- Image: Στο πρώτο Armor gameobject κάθε μίας κατηγορίας που αντιπροσωπεύει το κενό armor έβαλα στο πεδίο Source Image το sprite tabIconActive, ως προεπιλογή, ενώ στα υπόλοιπα gameobjects έβαλα το sprite tabIconIdle.
- Tab Button (Script): Στο πεδίο Tab Group έβαλα το GridLayoutGroup (Tab Group) gameobject από την σκηνή.



Εικόνα 4.54: Tab Button (Script) Component For Grid

4.5.4.8 GUI Image Των Tabs

Στα components των Image Gameobject που είναι child object όλων των Heads Armor, Chests Armor, Hands Armor, Legs Armor και Feet Armor Gameobjects, έχω βάλει τις εξής ρυθμίσεις:

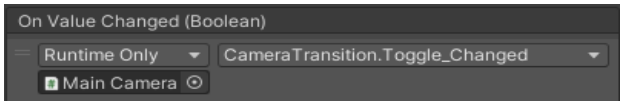
- Image: Στο πρώτο Armor gameobject κάθε μίας κατηγορίας που αντιπροσωπεύει το κενό armor έβαλα στο πεδίο Source Image το sprite EmptyIcon, ως προεπιλογή επειδή το πρώτο θα είναι το κενό που δεν έχει επιλεγεί καμία πανοπλία, ενώ στα υπόλοιπα gameobjects έβαλα τα sprites της κατάλληλης πανοπλίας από τους υποφακέλους που αρχίζουν με την λέξη Armor, και βρίσκονται μέσα στον φάκελο Assets → Custom Assets → Sprites. Και τέλος απενεργοποίησα την επιλογή Raycast Target.

Εικόνα 4.55: Grid Menu With Icons For Armor

4.5.4.9 GUI Full/FocusedViewToggle

Στα components του Full/FocusedViewToggle Gameobject έχω βάλει τις εξής ρυθμίσεις:

- Toggle: Στο On Value Changed (Boolean) Event έβαλα το main camera gameobject από την σκηνή και επέλεξα την Toggle_Changed() Function της κλάσης CameraTransition.



Εικόνα 4.56: Toggle's OnValueChange Event

4.5.4.10 GUI Play Button, Previous Button & Text (TMP) Των Button

Στα components των Play Button Gameobject, Previous Button Gameobject & τα Text (TMP) Gameobjects των δύο αυτών κουμπιών που είναι child object τους, έχω βάλει τις εξής ρυθμίσεις:

- Button: Μέσα στο Onclick() Event και των δύο κουμπιών πέρασα ως αντικείμενο το SecondaryCustomizationHelper gameobject που βρίσκεται στην σκηνή και επέλεξα στο Play Button την function LoadGameScene() από το script SecondaryCustomizationHelper, ενώ στο

Previous Button μια από τις function LoadCustomizationBlendshapesScene() της φυλής της τρέχουσας σκηνής από το script ChangeScenes (πχ LoadHumanCustomizationBlendshapesScene().

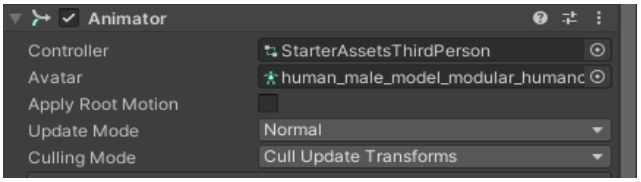
- TextMeshPro – Text (UI): Στο πεδίο του κειμένου έβαλα την λέξη Play στο ένα και την λέξη Previous στο άλλο.

4.5.5 Component UI Σκηνής Παιχνιδιού

4.5.5.1 PlayerArmature

Μέσα στα component του PlayerArmature έβαλα τις εξής ρυθμίσεις:

- Animator: Στο animator component του PlayerArmature, επέλεξα ως Controller το StarterAssetsThirdPerson animator controller, το οποίο βρίσκεται στον φάκελο Assets → StarterAssets → ThirdPersonController → Character → Animations και στο πεδίο Avatar, έβαλα το "human_male_model_humanoid Avatar", το οποίο βρίσκεται μέσα στο human_male_model_modular_humanoid assets στον φάκελο Assets → Custom Assets.

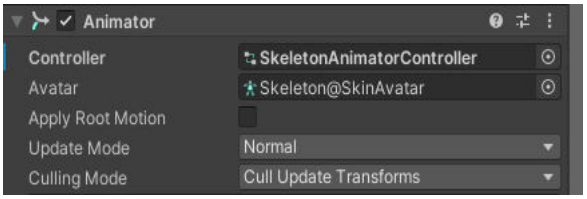


Εικόνα 4.57: PlayerArmature Animator Component

4.5.5.2 EnemySkeleton & EnemyCanvas

Μέσα στα component του EnemySkeleton έβαλα τις εξής ρυθμίσεις:

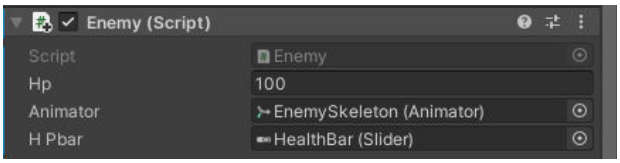
- Animator: Στο animator component του EnemySkeleton, επέλεξα ως Controller το SkeletonAnimatorController που δημιούργησα (βλ. Ενότητα 4.3.7), το οποίο βρίσκεται στον φάκελο Assets → FantasyMonster → Skeleton.



Εικόνα 4.58: EnemySkeleton Animator Component

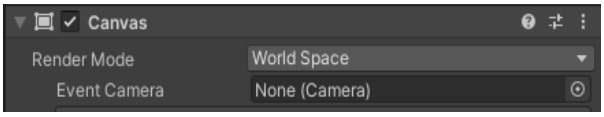
- NavMeshAgent: Σε αυτό το component στο Steering έβαλα Speed = 1.5, Angular Speed = 90, Acceleration = 8 και στο Obstacle Avoidance έβαλα Radius = 0.05 και Height = 0.15.

- Enemy (Script): Σε αυτό το component, έβαλα στο πεδίο HP την τιμή 100, στο πεδίο Animator έβαλα το animator component του EnemySkeleton και στο πεδίο HP bar, έβαλα το HealthBar slider του EnemyCanvas.



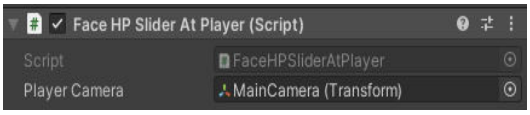
Εικόνα 4.59: Enemy (Script) Component

- Canvas: Στο canvas component του EnemyCanvas, επέλεξα στο Render Mode, το World Space.



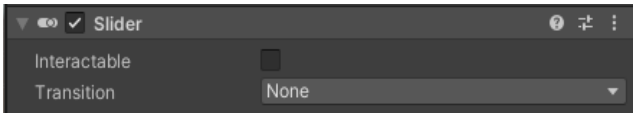
Εικόνα 4.60: EnemyCanvas Canvas Component

- FaceHPSliderAtPlayer (Script): Σε αυτό το component που βρίσκεται στο EnemyCanvas, έβαλα στο πεδίο Player Camera, την Main Camera από την σκηνή.



Εικόνα 4.61: Face HP Slider At Player (Script) Component

- Slider: Στο component Slider του HealthBar, έβγαλα την επιλογή Interactable.



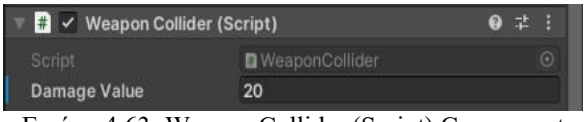
Εικόνα 4.62: HealthBar Slider Component

- Image: Στο "Background" child object του HealthBar έβαλα το κόκκινο χρώμα και στο "Fill" child object του HealthBar, έβαλα το πράσινο χρώμα.

4.5.5.3 PlayerSword

Μέσα στα component του PlayerSword έβαλα τις εξής ρυθμίσεις:

- WeaponCollider (Script): Σε αυτό το component, έβαλα στο πεδίο "Damage Value" την τιμή 20.

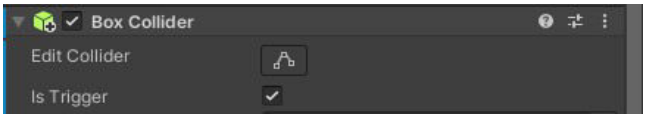


Εικόνα 4.63: Weapon Collider (Script) Component

- RigidBody: Σε αυτό το component, έβγαλα την επιλογή "Use Gravity".

Εικόνα 4.64: RigidBody Component

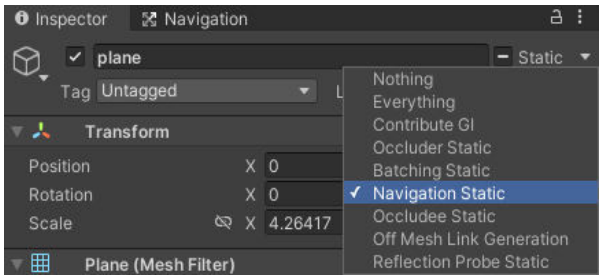
- Box Collider: Σε αυτό το component, έβαλα τις κατάλληλες διαστάσεις και ενεργοποίησα την επιλογή “Is Trigger”.



Εικόνα 4.65: Weapon's Box Collider Component

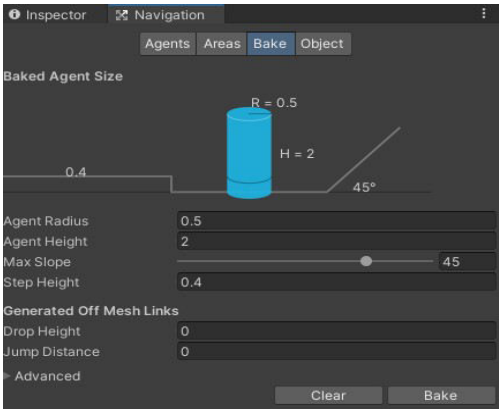
4.5.5.4 Gameobjects Περιβάλλοντος

Αρχικά επέλεξα το plane και μέσα από τον inspector (τέρμα πάνω δεξιά, δίπλα από εκεί που γράφει “static”), πάτησα το βελάκι και επέλεξα “Navigation Static” και μετά “Yes, change children”, για να αλλάξει και τα παιδιά του, ώστε να μπορεί να χρησιμοποιηθεί το περιβάλλον για να φτιαχτεί το navigation για τα EnemyAI.



Εικόνα 4.66: Set Plane & Children To Navigation Static

Στην συνέχεια πήγα πάνω στο μενού του Unity και επέλεξα Window → AI → Navigation, το οποίο ανοίγει το Navigation window. Από εκεί πήγα στην καρτέλα Bake και πάτησα το κουμπί Bake για να χαρτογραφηθεί το περιβάλλον, μαζί με όλα τα αντικείμενα που έχει, ώστε να μπορεί το EnemyAI να ξέρει πως να μετακινηθεί μέσα στην πίστα με την χρήση της εντολής SetDestination του NavMeshAgent (βλ. Ενότητα 5.5.2).



Εικόνα 4.67: Navigation Bake

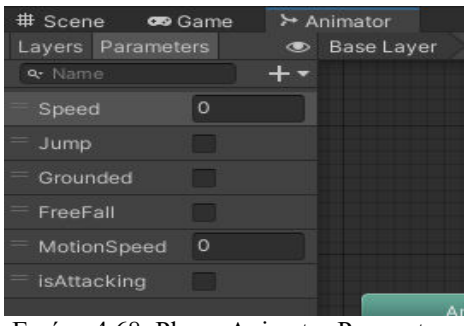
4.6 Animator State Machine

Μέσα στο animator window των διαφόρων χαρακτήρων, υπάρχει αυτό που αποκαλείται State Machine. Το State Machine, είναι το σύνολο των states (καταστάσεων του χαρακτήρα όπως πχ Idle, Walk, Run κτλ), των transition με τα οποία μπορείς να ορίσεις τον τρόπο πού αλλάζει ο χαρακτήρας από το ένα state στο άλλο, και τις παραμέτρους που χρησιμοποιείς ως συνθήκες, για να γίνουν αυτές οι εναλλαγές. Αυτό γίνεται, και για να ορίσεις το πότε θα παίξει κάποιο animation, αλλά και για την λογική εναλλαγή των states, δηλαδή για παράδειγμα να μην πηγαίνεις κατευθείαν από το Jump State στο Run State, να πηγαίνεις από το Jump στο Land State και μετά στο Run.

4.6.1 Player Animator State Machine

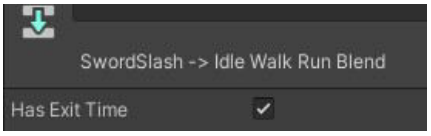
Εδώ περιγράφω τις ρυθμίσεις του StarterAssetsThirdPerson (μόνο για την επίθεση που πρόσθεσα εγώ), που ελέγχει τον παίχτη, μέσα από το Animator Window.

Σε αυτό το Animator Controller, μέσα στην καρτέλα “Parameters” (πάνω αριστερά), δημιούργησα μία boolean παράμετρο την οποία ονόμασα “isAttacking”, η οποία χρησιμοποιείται για την εναλλαγή μεταξύ του “Idle Walk Run Blend” state και του “SwordSlash” state (που περιέχει το animation που ο παίχτης επιτίθεται με το σπαθί).



Εικόνα 4.68: Player Animator Parameters

Στο transition από το “Idle Walk Run Blend” προς το “SwordSlash”, έβαλα ως condition την παράμετρο isAttacking, η οποία όταν είναι true ξεκινάει το transition. Επίσης έβγαλα την επιλογή “Has Exit Time”. Για το transition προς την άλλη κατεύθυνση, δεν έβαλα κανένα condition και απλά ενεργοποίησα την επιλογή “Has Exit Time”, ώστε με το που τελειώσει το animation, να επιστρέφει στο προηγούμενο state.



Εικόνα 4.69: Has Exit Time

4.6.2 Enemy AI Animator State Machine

Εδώ περιγράφω τις ρυθμίσεις του SkeletonAnimatorController, που ελέγχει τον εχθρό, μέσα από το Animator Window.

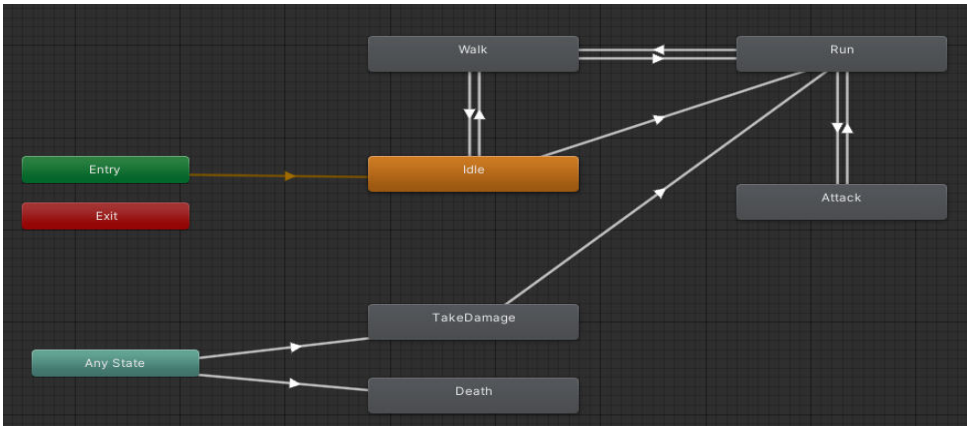
4.6.2.1 Animator Parameters

Εικόνα 4.70: Enemy Animator Parameters

Σε αυτό το Animator Controller, μέσα στην καρτέλα “Parameters” (πάνω αριστερά), δημιούργησα τις εξής παραμέτρους:

- isPatrolling: Αυτή η boolean παράμετρος, χρησιμοποιείται για την εναλλαγή μεταξύ των Idle και Walk State.
- isChasing: Αυτή η boolean παράμετρος, χρησιμοποιείται για την εναλλαγή μεταξύ των Walk και Run State, όπως επίσης και για να πάει προς μία κατεύθυνση από το Idle State στο Run State.
- isAttacking: Αυτή η boolean παράμετρος, χρησιμοποιείται για την εναλλαγή μεταξύ των Run και Attack State.
- damage: Αυτή η trigger παράμετρος, χρησιμοποιείται για να πάει από οποιοδήποτε State, στο TakeDamage State. Οι παράμετροι trigger, λειτουργούν σαν boolean, αλλά γίνονται αυτόματα reset από τον controller, μόλις χρησιμοποιηθούν σε κάποιο State Transition.
- die: Αυτή η trigger παράμετρος, χρησιμοποιείται για να πάει από οποιοδήποτε State, στο Death State.

4.6.2.2 Εισαγωγή Animations



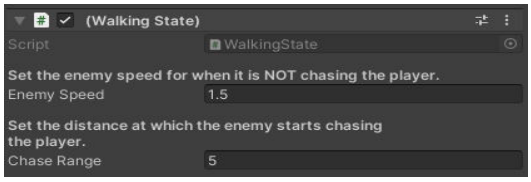
Εικόνα 4.71: Enemy State Machine

Σε αυτό το Animator Contoller, έβαλα κάποια animation (drag & drop από τον φάκελο) του EnemySkeleton (βλ. Ενότητα 4.3.6) και τους πέρασα κάποια scripts. Αυτά τα animation, είναι τα εξής:

- Idle Animation (State): Σε αυτό το state έβαλα ως component, το IdleState script, και πέρασα στο πεδίο Chase Range, την τιμή 5, η οποία δίνει την απόσταση που μπορεί ο εχθρός να ανιχνεύσει τον παίχτη.

Εικόνα 4.72: Idle State (Script) Component

- Walk Animation (State): Σε αυτό το state έβαλα ως component, το WalkingState script, και πέρασα στο πεδίο Chase Range, την τιμή 5, η οποία δίνει την απόσταση που μπορεί ο εχθρός να ανιχνεύσει τον παίχτη. Επίσης στο πεδίο Enemy Speed, πέρασα την τιμή 1.5, η οποία δίνει την ταχύτητα του εχθρού όταν δεν έχει ανιχνεύσει τον παίχτη και απλός patrol.



Εικόνα 4.73: Walking State (Script) Component

- Run Animation (State): Σε αυτό το state έβαλα ως component, το ChasingState script, και πέρασα στο πεδίο Chase Range, την τιμή 5, η οποία δίνει την απόσταση που ο εχθρός σταματάει να κυνηγάει τον παίχτη. Επίσης στο πεδίο Enemy Speed, πέρασα την τιμή 3.5, η οποία δίνει την ταχύτητα του εχθρού όταν έχει πλέον ανιχνεύσει τον παίχτη και τον κυνηγάει

και στο πεδίο Attack Range, την τιμή 2, η οποία δίνει τη απόσταση που ο εχθρός ξεκινήσει να επιτίθεται τον παίχτη.

Εικόνα 4.74: Chasing State (Script) Component

- Attack Animation (State): Σε αυτό το state έβαλα ως component, το AttackState script, και πέρασα στο πεδίο Attack Range, την τιμή 2, η οποία δίνει τη απόσταση που ο εχθρός σταματάει να επιτίθεται τον παίχτη.

Εικόνα 4.75: Attack State (Script) Component

- TakeDamage & Death Animations (States): Σε αυτά τα states, δεν πέρασα κανένα script.

4.6.2.3 Animations Transition

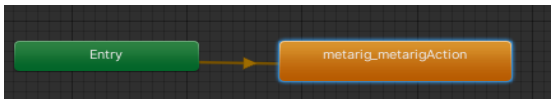
Στο Animator Controller, έβαλα και κάποια transitions μέσω των οποίων γίνεται η εναλλαγή μεταξύ των διαφόρων States. Επίσης όλα τα states, έχουν μία επιλογή που λέγεται “Has Exit Time”, η οποία όταν είναι επιλεγμένη, σημαίνει οτι με το που τελειώσει το τρέχον animation, μεταφέρεται αυτόματα στο επόμενο state. Αυτή η επιλογή είναι απενεργοποιημένη σε όλα τα states, εκτός από το TakeDamage state, επειδή θέλω την αλλαγή να την ελέγχουν τα scripts και οι παράμετροι που έφτιαξα. Το TakeDamage από την άλλη, έχει αυτή την επιλογή ενεργοποιημένη, επειδή θέλω να παίζει μία φορά το animation όταν τρώει damage ο εχθρός και μετά να επιστρέφει στο Run state. Επίσης αυτό έγινε ώστε ακόμα και αν δεν σε είχε δει ο εχθρός, όταν τον χτυπάς να αρχίσει να σε κυνηγάει. Αυτά τα transitions λοιπόν, είναι τα εξής:

- Idle/Walk Transition: Αυτά τα states, έχουν αμφίδρομο transition. Και στα 2 πέρασα, ως condition, την παράμετρο isPatrolling, η οποία όταν είναι true, αλλάζει από Idle σε Walk και όταν είναι false, αλλάζει από Walk σε Idle.
- Idle/Run Transition: Αυτά τα states, έχουν μονής κατεύθυνσης transition. Πέρασα, ως condition, την παράμετρο isChasing, η οποία όταν είναι true, αλλάζει από Idle σε Run.
- Walk/Run Transition: Αυτά τα states, έχουν αμφίδρομο transition. Και στα 2 πέρασα, ως condition, την παράμετρο isChasing, η οποία όταν είναι true, αλλάζει από Walk σε Run και όταν είναι false, αλλάζει από Run σε Walk.

- Run/Attack Transition: Αυτά τα states, έχουν αμφίδρομο transition. Και στα 2 πέρασα, ως condition, την παράμετρο isAttacking, η οποία όταν είναι true, αλλάζει από Run σε Attack και όταν είναι false, αλλάζει από Attack σε Run.
- Any State/TakeDamage Transition: Σε αυτό το transition, πέρασα ως condition, το damage trigger, το οποίο όταν ενεργοποιηθεί, μεταφέρει τον εχθρό από οποιοδήποτε state, στο TakeDamage state.
- Any State/Death Transition: Σε αυτό το transition, πέρασα ως condition, το die trigger, το οποίο όταν ενεργοποιηθεί, μεταφέρει τον εχθρό από οποιοδήποτε state, στο Death state.

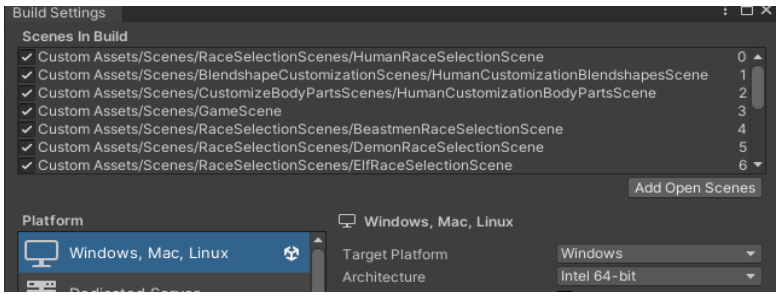
4.6.3 Human Male Model Animator State Machine

Το παράθυρο του animator ανοίγει επιλέγοντας animation και μετά animator. Αφού ανοίξει το παράθυρο animator επέλεξα το Human Male Model Animator Controller και μετά τράβηξα και πέταξα το duplicate animation (βλ. Ενότητα 4.3.1) από τον φάκελο Custom Assets μέσα στο animator controller. Εφόσον το animator controller ήταν κενό το transition από το entry προς το animation δημιουργείται αυτόματα.



Εικόνα 4.76: Idle Animation For Menu Scenes

4.7 Build Settings



Εικόνα 4.77: Build Settings

Τα Build Settings ανοίγουν πηγαίνοντας στο File → Build Settings, του Unity. Ανοίγωντας το, επέλεξα ως Platform το Windows, και έκανα drag & drop, όλες τις σκηνές που έφτιαξα μέσα στο “Scenes In Build”, για να μπορεί να τις φορτώσει το παιχνίδι. Πρώτη έβαλα την σκηνή HumanRaceSelectionScene, για να είναι αυτή που εμφανίζεται στην αρχή όταν τρέξεις το παιχνίδι.

Κεφάλαιο 5ο: Κώδικας Παιχνιδιού Στο Unity

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο, αναλύω όλο τον κώδικα του παιχνιδιού, τον οποίο έχω χωρίσει σε ενότητες οι οποίες ομαδοποιούν τα scripts με βάση την λειτουργία τους. Μετά χώρισα αυτές τις ενότητες σε άλλες υποενότητες για κάθε script ξεχωριστά και τέλος το κάθε script το χώρισα σε υποενότητες για κάθε μέθοδο, αν το script έχει αρκετές μεθόδους ώστε να δικαιολογεί τον χωρισμό.

5.2 Εναλλαγή Στοιχείων Του Χαρακτήρα

5.2.1 CustomizeCharacterColors Script

Το CustomizeCharacterColors Script είναι το script το οποίο χειρίζεται την αλλαγή χρωμάτων του χαρακτήρα, όπως είναι το χρώμα του δέρματος και το χρώμα των ματιών. Το Script αυτό βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → CustomizeCharacterParts.

5.2.1.1 Μεταβλητές Κλάσης

```
public Color[] skinColors; //array that holds all the skin colors to use
public Material skinMaterial; //skin material to use the colors on
public Material earMaterial; //ear material to use the colors on

public Color[] irisColors; //array that holds all the iris colors to use
public Material irisMaterial; //iris material to use the colors on
```

Εικόνα 5.1: Μεταβλητές Colors & Materials

Μέσα στην κλάση έχω δηλώσει τις **public μεταβλητές**:

- Έναν πίνακα τύπου **Color** και τον ονόμασα **skinColors**. Αυτός ο πίνακας δέχεται μέσα από τον inspector όλα τα χρώματα που θα χρησιμοποιηθούν για το δέρμα.
- Τα **skinMaterial** και **earMaterial** που είναι τύπου **Material** και τα οποία δέχονται μέσα από τον inspector το material του δέρματος του μοντέλου του χαρακτήρα και το material των αυτιών, πάνω στα οποία θα χρησιμοποιηθούν τα χρώματα του δέρματος.
- Έναν πίνακα τύπου **Color** και τον ονόμασα **irisColors**. Αυτός ο πίνακας δέχεται μέσα από τον inspector όλα τα χρώματα που θα χρησιμοποιηθούν για το χρώμα των ματιών.
- Το **irisMaterial** που είναι τύπου **Material** και το οποίο δέχεται μέσα από τον inspector το material της ίριδας των ματιών, πάνω στο οποίο θα χρησιμοποιηθούν τα χρώματα του πίνακα **irisColors**.
- Έναν πίνακα τύπου **Color** και τον ονόμασα **hairColors**. Αυτός ο πίνακας δέχεται μέσα από τον inspector όλα τα χρώματα που θα χρησιμοποιηθούν για το χρώμα των μαλλιών.

- Το **hairMaterial** που είναι τύπου **Material** και το οποίο δέχεται μέσα από τον inspector το material των μαλλιών, πάνω στο οποίο θα χρησιμοποιηθούν τα χρώματα του πίνακα **hairColors**.

5.2.1.2 Μέθοδοι changeSkinColor(), changeSkinColor() & changeHairColor()

Αυτή η κλάση έχει μόνο 3 functions. Την changeSkinColor(int), την changeIrisColor(int) και την hangeHairColor(int). Και οι 3 δέχονται ως παράμετρο ένα int το οποίο είναι το index του χρώματος μέσα από τους πίνακες των χρωμάτων. Η changeSkinColor(int) αλλάζει το χρώμα του skin αλλά και του ear material χρησιμοποιώντας τον πίνακα των χρωμάτων του δέρματος. Οι άλλες 2 μέθοδοι, κάνουν ακριβώς το ίδιο αλλά για το χρώμα των ματιών και των μαλλιών αντίστοιχα, χρησιμοποιώντας τους κατάλληλους πίνακες. Η κλήση αυτών των μεθόδων γίνεται μέσα από την σκηνή με την χρήση των Onclick Event στα κουμπιά και βάζοντας από εκεί το index του χρώματος που θέλω.

```
1 reference
public void changeSkinColor(int colorIndex)
{
    skinMaterial.color = skinColors[colorIndex];
    earMaterial.color = skinColors[colorIndex];
}

1 reference
public void changeIrisColor(int colorIndex)
{
    irisMaterial.color = irisColors[colorIndex];
}
```

Εικόνα 5.2: Μέθοδοι Αλλαγής Χρωμάτων

5.2.2 CustomizeCharacterParts Script

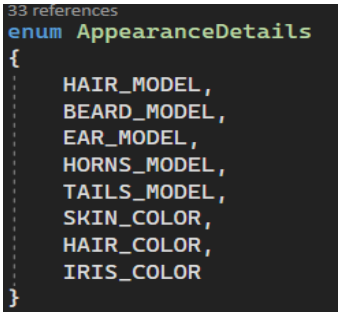
Το CustomizeCharacterParts Script είναι το script το οποία χειρίζεται την αλλαγή των στοιχείων του χαρακτήρα, όπως είναι τα αυτιά, τα κέρατα κτλ. Το Script αυτό βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → CustomizeCharacterParts.

5.2.2.1 Μεταβλητές Κλάσης

Μέσα στην κλάση έχω δηλώσει τις **private μεταβλητές**:

- Μία **Enum** μεταβλητή που την ονόμασα **AppearanceDetails** και χρησιμοποιείται για την switch της μεθόδου που αλλάζει τα μοντέλα αλλά και ως dictionary key για αποθήκευση των αλλαγών ώστε τα φορτώσει μετά στην σκηνή του παιχνιδιού.

- 5 πίνακες τύπου **GameObject** τους οποίους ονόμασα **earModels, hornsModel, tailsModels, hairModels και beardModels** και οι οποίοι είναι private αλλά παρόλα αυτά εμφανίζονται στον inspector λόγω της εντολής [SerializeField]. Κάθε public μεταβλητή γίνεται αυτόματα



Εικόνα 5.3: Enum

serialize ενώ οι private μεταβλητές δεν γίνονται. Χρησιμοποιώντας την [SerializeField], αναγκάζεις την μεταβλητή να γίνει serialize παραμένοντας ταυτόχρονα private. Όταν μια μεταβλητή γίνεται serialize σημαίνει ότι αποθηκεύεται από την ram στον σκληρό δίσκο για μελλοντική χρήση και ταυτόχρονα εμφανίζεται στον inspector.

```
[SerializeField] private GameObject[] earModels;
[SerializeField] private GameObject[] hornsModels;
[SerializeField] private GameObject[] tailsModels;
[SerializeField] private GameObject[] hairModels;
[SerializeField] private GameObject[] beardModels;
```
Εικόνα 5.4: GameObject Arays

- Μία **serialized** μεταβλητή τύπου **Transform** που την ονόμασα **face** και η οποία θα πάρει το transform του κόκαλου (bone) του προσώπου του χαρακτήρα που βρίσκεται στις σκηνές του μενού και το οποίο θα χρησιμοποιηθεί ως parent object των μοντέλων των αυτιών, κεράτων κτλ μέσα στην σκηνή και μία ίδια μεταβλητή για το **tailBone** για τις ουρές.

```
[SerializeField] private Transform face; //The
[SerializeField] private Transform tailBone;
```
Εικόνα 5.5: Μεταβλητές face & tailBone

- 5 μεταβλητές τύπου **GameObject** που τις ονόμασα **activeEars, activeHorns, activeTails, activeHair και activeBeard** και οι οποίες αποθηκεύουν το τρέχον επιλεγμένο GameObject από τους πίνακες των μοντέλων (βλ. Εικόνα 5.4).

```
GameObject activeEars;
GameObject activeHorns;
GameObject activeTails;
GameObject activeHair;
GameObject activeBeard;
```
Εικόνα 5.6: Μεταβλητές GameObject

- 5 **int** μεταβλητές που τις ονόμασα **earsIndex, hornsIndex, tailsIndex, hairIndex και beardIndex** και οι οποίες δίνουν την επιλεγμένη θέση μέσα στους πίνακες (βλ. Εικόνα 5.4).

```
int earsIndex = 0;
int hornsIndex = 0;
int tailsIndex = 0;
int hairIndex = 0;
int beardIndex = 0;
```
Εικόνα 5.7: Index Μεταβλητές

- Ένα **Dictionary** που δέχεται ως key το AppearanceDetails Enum και ως value ένα int (το οποίο είναι το index των πινάκων με τα μοντέλα) και το ονόμασα **savedAppearance**. Αυτό το dictionary χρησιμοποιείται για να αποθηκεύω τις επιλογές των στοιχείων του χαρακτήρα και να τα φορτώνω μετά μέσα στην σκηνή του παιχνιδιού.

```
//Dictionary that saves the model selection for use in gameScene. Saves enum for model type and int for selected index
Dictionary<AppearanceDetails, int> savedAppearance;
```
Εικόνα 5.8: Dictionary

- Μία **GameObject** μεταβλητή την οποία ονόμασα **newFace** και χρησιμοποιείται όπως η μεταβλητή face, με την διαφορά οτι εδώ θα μπει το face bone του χαρακτήρα από την σκηνή του παιχνιδιού (δηλαδή από το humanoid rig μοντέλο), για να φορτώσω τις αποθηκευμένες επιλογές των μοντέλων από το dictionary και το ίδιο για την μεταβλητή **newTailBone**.

5.2.2.2 Μέθοδοι NextEarsModel() & PreviousEarsModel() κτλ

Ακριβώς τα ίδια που θα περιγράψω εδώ, ισχύουν και για τις άλλες παρόμοιες μεθόδους απλά αφορούν διαφορετικά στοιχεία του χαρακτήρα, όπως κέρατα NextHornsModel() & PreviousHornsModel() κτλ.

Η μέθοδος NextEarsModel() είναι void μέθοδος άρα δεν επιστρέφει τίποτα. Μέσα έχει μία if statement η οποία ελέγχει αν η μεταβλητή earsIndex βγαίνει εκτός των ορίων του πίνακα earModels. Αν δεν ξεπερνάει τα όρια του πίνακα τότε αυξάνει την μεταβλητή earsIndex κατά ένα για να πάρει το επόμενο μοντέλο από τον πίνακα και αν ξεπερνάει τα όρια τότε πηγαίνει στην αρχή του πίνακα. Αφού τελειώσει ο έλεγχος καλεί την μέθοδο ChangeBodyParts (βλ. Ενότητα 5.2.2.3) στην οποία βάζει στην πρώτη παράμετρο την ανάλογη τιμή της AppearanceDetails Enum (για τα αυτιά είναι η AppearanceDetails.EAR_MODEL) και στην δεύτερη παράμετρο το αλλαγμένο earsIndex.

```
1 reference
public void NextEarsModel()
{
    if (earsIndex < earModels.Length - 1)
        earsIndex++;
    else
        earsIndex = 0;

    ChangeBodyPartsModel(AppearanceDetails.EAR_MODEL, earsIndex);
}
```
Εικόνα 5.9: Μέθοδος Αλλαγής Επόμενου Μοντέλου

Στη μέθοδο PreviousEarsModel ισχύει το ίδια απλά αντίστροφα.

```
1 reference
public void PreviousEarsModel()
{
    if (earsIndex > 0)
        earsIndex--;
    else
        earsIndex = earModels.Length - 1;

    ChangeBodyPartsModel(AppearanceDetails.EAR_MODEL, earsIndex);
}
```

Εικόνα 5.10: Μέθοδος Αλλαγής Προηγούμενου Μοντέλου

5.2.2.3 Μέθοδος ChangeBodyParts()

Αυτή η μέθοδος χρησιμοποιείται για την αλλαγή των μοντέλων των στοιχείων του χαρακτήρα. Παίρνει ως παραμέτρους την τιμή του AppearanceDetails Enum, η οποία ονομάζεται detail και ανάλογα σε ποια κατηγορία μοντέλων θέλω να γίνει η αλλαγή (πχ αυτιά), καλεί την κατάλληλη μέθοδο (βλ. Ενότητα 5.2.2.2). Επίσης παίρνει μια int παράμετρο που ονομάζεται id και είναι η θέση του πίνακα (αντιστοιχεί σε μια από τις μεταβλητές Index).

```
15 references
void ChangeBodyPartsModel(AppearanceDetails detail, int id)
{
```

Εικόνα 5.11: Δήλωση Μεθόδου ChangeBodyParts

Αρχικά ελέγχει τι τιμή έχει η μεταβλητή detail (με switch case) και αναλόγως χρησιμοποιεί την κατάλληλη GameObject μεταβλητή (πχ activeEars). Μέσα στην case ελέγχει αν υπάρχει κάποιο τρέχον αποθηκευμένο μοντέλο στην activeEars και αν υπάρχει το καταστρέφει, ώστε να διαγραφεί από την σκηνή το τρέχον επιλεγμένο μοντέλο αυτιών.

```
switch (detail) //switch with enum variable for model types
{
    case AppearanceDetails.EAR_MODEL:
        if (activeEars != null) //checks if there is an active object on the character and destroys it before setting a new model
            GameObject.Destroy(activeEars);
```

Εικόνα 5.12: Switch

Μετά αποθηκεύει ένα αντίγραφο του καινούργιου μοντέλου απο των πίνακα earModels χρησιμοποιώντας το id και θέτει ως parent του μοντέλου το face bone του generic rig χαρακτήρα από την σκηνή του μενού (το ίδιο γίνεται και για τις ουρές αλλά με την χρήση του tailBone). Τέλος θέτει

την θέση του μοντέλου πάνω στον χαρακτήρα (σε σχέση με την θέση του face bone ή του tailBone αντίστοιχα) μέσω της μεθόδου ResetBodyPartsTransform που υλοποίησα στην κλάση UtilSetPartsTransform, η οποία είναι μια extension μέθοδος που επεκτείνει την built-in λειτουργικότητα της κλάσης transform του Unity (βλ. Ενότητα 5.2.3).

```
activeEars = GameObject.Instantiate(earModels[id]); //Creates a clone of the object inside the array and sets it as active
activeEars.transform.SetParent(face); //set parent bone
activeEars.transform.ResetBodyPartsTransform(); //resets body parts model transform relative to the bone
break;
```

Εικόνα 5.13: Εσωτερικά του 1ου Case της Switch

5.2.2.4 Μέθοδος SaveAppearance()

Αυτή η μέθοδος αποθηκεύει τις επιλογές των στοιχείων του χαρακτήρα, που διάλεξε ο παίχτης στην σκηνή επιλογής στοιχείων χαρακτήρα. Πρώτα αρχικοποιώ το savedAppearance dictionary το οποίο αποθηκεύει αυτές τις επιλογές και μετά αποθηκεύω τις επιλογές με την χρήση της μεθόδου Add του dictionary, βάζοντας ως key μία μία όλες τις τιμές της AppearanceDetails Enum και ως value την αντίστοιχη μεταβλητή (earsIndex, hornsIndex κτλ). Αυτή η μέθοδος καλείται στην τελευταία σκηνή του μενού ακριβώς πριν περάσεις στην σκηνή του παιχνιδιού.

```
//Function to save customized body parts appearance in dictionary for use in game scene
1 reference
public void SaveAppearance()
{
    savedAppearance = new Dictionary<AppearanceDetails, int>();

    savedAppearance.Add(AppearanceDetails.EAR_MODEL, earsIndex);
    savedAppearance.Add(AppearanceDetails.HORNS_MODEL, hornsIndex);
    savedAppearance.Add(AppearanceDetails.TAILS_MODEL, tailsIndex);
    savedAppearance.Add(AppearanceDetails.HAIR_MODEL, hairIndex);
    savedAppearance.Add(AppearanceDetails.BEARD_MODEL, beardIndex);
}
```

Εικόνα 5.14: Αποθήκευση Επιλογών

5.2.2.5 Μέθοδος LoadAppearance()

Αυτή η μέθοδος φορτώνει στον humanoid rig χαρακτήρα, που βρίσκεται στην σκηνή παιχνιδιού, τις αποθηκευμένες επιλογές του savedAppearance dictionary.

Αρχικά βρίσκω τα gameobject με τα “GameSceneFaceBone” και “GameSceneTailBone” tags από την σκηνή του παιχνιδιού, τα οποία είναι τα “face” και “Spine.001” bones του humanoid rig χαρακτήρα, με την χρήση της μεθόδου FindGameObjectWithTag της GameObject κλάσης του Unity και μετά τα περνάω στις μεταβλητές newFace και newTailBone αντίστοιχα. Ο υπόλοιπος κώδικας είναι ίδιος με την μέθοδο ChangeBodyParts (βλ. Ενότητα 5.2.2.3), με τις εξής διαφορές:

- Εδώ δεν υπάρχει switch case για επιλογή της κατηγορίας χαρακτήρα. Απλά περνάει από όλες τις κατηγορίες (αυτιά, κέρατα κτλ) και βάζει όλες τις επιλογές στον χαρακτήρα.
- Αντί για την παράμετρο id που δεχόταν η μέθοδος ChangeBodyParts κατά την κλήση της για την επιλογή της θέσης από τον πίνακα των μοντέλων, εδώ χρησιμοποιώ το savedAppearance dictionary, για να βάλω τις αποθηκευμένες επιλογές των μοντέλων του παίχτη. Στο dictionary έβαλα τα keys από την AppearanceDetails Enum για να πάρω τις αποθηκευμένες int τιμές ως index για τον πίνακα πχ savedAppearance[AppearanceDetail.EAR_MODEL].

```
//Instantiate model using dictionary with saved details from the menu scenes
activeEars = GameObject.Instantiate(earModels[savedAppearance[AppearanceDetails.EAR_MODEL]]);
```

Εικόνα 5.15: Instantiate Model Using Dictionary

- Έθεσα ως parent των μοντέλων το transform του newFace GameObject (newFace.transform), αντί για την μεταβλητή face ή το transform του newTailBone αν πρόκειται για τις ουρές.

```
activeHorns.transform.SetParent(newFace.transform);
```

Εικόνα 5.16: newFace Transform

- Σε όλες τις κατηγορίες μοντέλων έβαλα έναν έλεγχο για να μην εκτελέσει το κομμάτι του κώδικα της αντίστοιχης κατηγορίας μοντέλων, αν ο πίνακας μοντέλων της κατηγορίας στοιχείων είναι κενός (αν δηλαδή δεν υπάρχουν κέρατα στην σκηνή αυτής της φυλής). Η μόνη κατηγορία που δεν έχει αυτόν τον έλεγχο είναι η κατηγορία των αυτιών επειδή όλες οι φυλές έχουν αυτιά.

```
if (hornsModels.Length != 0)
{
```

Εικόνα 5.17: Don't Execute Code Inside If Array Of Models Is Empty

5.2.2.6 Μέθοδος Randomize()

Αυτή η μέθοδος βάζει κάποια τυχαία μοντέλα στοιχείων του χαρακτήρα (ανάλογα με την φυλή της τρέχουσας σκηνής) και καλείται κατά το άνοιγμα της σκηνής στο runtime μέσω της μεθόδου Start() που έχουν οι κλάσεις του Unity.

Αρχικά δίνω μια τυχαία int τιμή (από 0 μέχρι και όσο είναι το μέγεθος του αντίστοιχου πίνακα πχ του πίνακα earModels) σε κάθε μια από τις 5 Index μεταβλητές ξεχωριστά (earsIndex, hornsIndex κτλ), με την χρήση της μεθόδου Random.Range().

```
earsIndex = Random.Range(0, earModels.Length);
hornsIndex = Random.Range(0, hornsModels.Length);
tailsIndex = Random.Range(0, tailsModels.Length);
hairIndex = Random.Range(0, hairModels.Length);
beardIndex = Random.Range(0, beardModels.Length);
```

Εικόνα 5.18: Random Number Within Range

Έπειτα ελέγχω αν ο πίνακας της κάθε κατηγορίας στοιχείων χαρακτήρα δεν είναι κενός (για όλες τις κατηγορίες εκτός της κατηγορίας των αυτιών αφού όλες οι φυλές έχουν αυτιά) και καλώ την μέθοδο ChangeBodyParts (βλ. Ενότητα 5.2.2.3), βάζοντας ως 1η παράμετρο τις διάφορες τιμές της AppearanceDetails Enum και ως 2η παράμετρο την αντίστοιχη μεταβλητή Index στις οποίες έδωσα τυχαίες τιμές.

```
ChangeBodyPartsModel(AppearanceDetails.EAR_MODEL, earsIndex);

if (hornsModels.Length != 0)
    ChangeBodyPartsModel(AppearanceDetails.HORNS_MODEL, hornsIndex);

if (tailsModels.Length != 0)
    ChangeBodyPartsModel(AppearanceDetails.TAILS_MODEL, tailsIndex);

if (tailsModels.Length != 0)
    ChangeBodyPartsModel(AppearanceDetails.HAIR_MODEL, hairIndex);

if (tailsModels.Length != 0)
    ChangeBodyPartsModel(AppearanceDetails.BEARD_MODEL, beardIndex);
```

Εικόνα 5.19: Κλήση της Μεθόδου ChangeBodyPartsModel

5.2.2.7 Μέθοδος HideHairWhenWearingHelmet()

Αυτή η μέθοδος χρησιμοποιείται για να κρυφτούν τα μαλλιά του χαρακτήρα, αν φοράει κράνος και καλείται μέσα από τις μεθόδους ChangeArmorPart και LoadArmor του SwitchArmorEquipment script (βλ. Ενότητα 5.4.4.2 και 5.4.4.3).

```
public void HideHairWhenWearingHelmet(int helmetID)
{
    //if helmet ID is not 0, meaning that character w
    if (helmetID != 0)
        activeHair.SetActive(false);
    else
        activeHair.SetActive(true);
}
```

Εικόνα 5.20: HideHairWhenWearingHelmet Method

Η μέθοδος δέχεται ως παράμετρο το ID (index) του πίνακα που περιέχει τα headModels και αν το ID έχει την τιμή 0 σημαίνει οτι δεν φοράει κράνος (στο 0 αποθηκεύεται το μοντέλο χωρίς κράνος) και άρα ενεργοποιεί τα μαλλιά, αλλιώς τα απενεργοποιεί.

5.2.3 UtilSetPartsTransform Script

Αυτή η κλάση (class) είναι μια static class με την οποία υλοποίησα μια extension μέθοδο για την Transform class του Unity. Το Script αυτό βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → CustomizeCharacterParts. Οι extension μέθοδοι, είναι μέθοδοι οι οποίες προσδίδουν επιπλέον λειτουργία σε ήδη υπάρχων τύπους μεταβλητών όπως είναι για παράδειγμα η tranform,

GameObject, int κτλ, χωρίς να πειράξεις τον κώδικα της κλάσης του τύπου. Επίσης δεν χρειάζεται να προσθέσω το script σε κάποιο αντικείμενο στην σκηνή, μπορώ απλώς να καλέσω την μέθοδο από οπουδήποτε, ακριβώς όπως θα καλούσα κάθε άλλη μέθοδο της transform. Χρησιμοποίησα αυτή την extension μέθοδο για να μηδενίσω την θέση του ενεργού μοντέλου των στοιχείων του χαρακτήρα, πάνω στο face bone (ή στο tail bone για τις ουρές) του χαρακτήρα (βλ. Ενότητες 5.2.2.3 & 5.2.2.5). Πιο συγκεκριμένα μηδένισα το "pivot" parent object των prefab μοντέλων (βλ. Ενότητα 4.3.3), κατά το runtime, ώστε να έρθουν τα μοντέλα στην θέση τους πάνω στον χαρακτήρα.

5.2.3.1 Μέθοδος ResetBodyPartsTransform()

```
public static class UtilSetPartsTransform
{

    10 references
    public static void ResetBodyPartsTransform(this Transform _transform)
    {
        _transform.localPosition = Vector3.zero; //Sets position to zero
        _transform.localEulerAngles = Vector3.zero; //Sets rotation to zero
        _transform.localScale = Vector3.one; //Sets scale to one
    }

}
```

Εικόνα 5.21: Extension Method

Η υλοποίηση της extension μεθόδου έγινε φτιάχνοντας μια static class την οποία ονόμασα UtilSetPartsTransform και μετά μια static μέθοδο που την ονόμασα ResetBodyPartsTransform. Η μέθοδος για να λειτουργήσει ως extension, πέρασα ως παράμετρο το "this" ακολουθούμενο από τον τύπο του οποίου θέλω να επεκτείνω την λειτουργία του, δηλαδή "this Transform" (και μετά ένα όνομα). Αν ήθελα να περάσω και άλλη παράμετρο, θα έπρεπε και πάλι να περάσω ως πρώτη παράμετρο το ίδιο πράγμα και μετά να βάλω πχ την παράμετρο int και αν ήθελα να καλέσω αυτή την μέθοδο, θα έπρεπε κατά την κλήση της μεθόδου να περάσω τιμή μόνο στις υπόλοιπες παραμέτρους εκτός της πρώτης (πχ. ResetBodyPartsTransform(int)).

Έπειτα μηδένισα όλες τις τιμές (x,y,z) του position και του rotation του αντικειμένου πάνω στο οποίο θα χρησιμοποιηθεί η μέθοδος, με το parent object ως σημείο αναφοράς και έκανα το ίδιο και για το scale, με την διαφορά οτι σε αυτό έβαλα την τιμή 1.

5.3 Ρύθμιση Blendshapes Του Χαρακτήρα

5.3.1 Blendshape Script

Αυτή η κλάση (class) είναι για να δημιουργήσω έναν τύπου "Blendshape", ο οποίος περιέχει το index του blendershape με το επίθεμα "Max" και το index του blendershape με το επίθεμα "Min". Το Script αυτό βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → Blendshapes. Σε

αυτή την κλάση δημιούργησα δύο auto-properties, τα οποία είναι κάτι σαν συντομογραφία για να ορίσεις properties και μετά ο compiler το δημιουργεί σε κανονικό property. Είναι απλά ένας τρόπος για να ορίσεις property με get και set πιο σύντομα. Αυτά τα properties τα ονόμασα positiveIndex και negativeIndex και μετά δημιούργησα ένα constructor για αυτά τα δύο properties. Επίσης αυτή η κλάση δεν κάνει inherit την MonoBehaviour του Unity.

```
//The index of the max suffixes of the blendshapes. Their position inside the blendshape list in the inspector
10 references
public int positiveIndex { get; set; }
//The index of the min suffixes of the blendshapes. Their position inside the blendshape list in the inspector
10 references
public int negativeIndex { get; set; }

1 reference
public Blendshape(int positiveIndex, int negativeIndex)
{
        this.positiveIndex = positiveIndex;
        this.negativeIndex = negativeIndex;
}
```

Εικόνα 5.22: Auto-Properties & Constructor

5.3.2 Singleton Script

Εδώ δημιούργησα μία generic singleton class, η οποία βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → Blendshapes. Η singleton είναι μία κλάση η οποία είναι προσβάσιμη από παντού, αλλά υπάρχει μόνο μία φορά μέσα στη σκηνή, δηλαδή δεν μπορείς να δημιουργήσεις πάνω από ένα instance αυτής της κλάσης. Στο Unity οι κλάσεις που κάνουν inherit την κλάση MonoBehaviour βασίζονται πάνω στην ιδέα των components. Η MonoBehaviour class κάνει inherit την Behaviour class και αυτή με την σειρά της κάνει inherit την Component class. Για κάθε script που εισάγω ως component μέσα σε ένα GameObject, δημιουργείται ένα instance αυτού του script. Στο Unity δεν υπάρχει ο συντελεστής "new", οπότε αν θέλω να δημιουργήσω ένα καινούργιο instance μέσω του κώδικα, χωρίς δηλαδή να προσθέσω το script ως component από τον Editor, τότε πρέπει να χρησιμοποιήσω την εντολή AddComponent (στην ουσία είναι ο αντίστοιχος συντελεστής new του Unity). Η singleton δέχεται ως όρισμα μία άλλη κλάση. Η κλάση που έχει μπει ως όρισμα της singleton, απαγορεύει την πρόσβαση στον constructor της και μπορεί να δημιουργήσει ένα και μοναδικό instance μόνο μέσω αυτής της singleton class. Επίσης για να καλέσεις αυτό το instance της κλάσης που έχει μπει ως όρισμα, και να αποκτήσεις πρόσβαση σε αυτή την κλάση, μπορείς να το κάνεις απευθείας μέσω του Instance property της singleton class. Έτσι αντί να δημιουργήσεις ένα αντικείμενο της κλάσης σε κάθε script που θέλεις να έχεις πρόσβαση σε αυτήν την κλάση και μετά να πάρεις το instance με την εντολή GetComponent, μπορείς να έχεις πρόσβαση από οπουδήποτε, απευθείας μέσω του instance της singleton γράφοντας πχ CharacterCustomization.Instance.MyMethod.

Σε αυτή τη singleton υλοποίησα ως όρισμα τον generic type (T) το οποίο σημαίνει ότι αυτή singleton μπορεί να χρησιμοποιηθεί για οποιαδήποτε κλάση. Όρισα δηλαδή την singleton ως

"Singleton<T>" και μετά στην κλάση για την οποία θέλω να χρησιμοποιηθεί αυτή η singleton, την όρισα ως "CharacterCustomization : Singleton <CharacterCustomization>". Αυτό σημαίνει ότι η κλάση character customization ορίζεται ως singleton και περνάει σαν όρισμα τον εαυτό της και μετά μέσα στη singleton αντικαθιστάται το generic type "T", με το όνομα της κλάσης που κάνει inherit την singleton. Επίσης η singleton κάνει inherit την MonoBehaviour class πού είναι η κλάση από την οποία προέρχονται όλα τα script του Unity.

```
public class Singleton<T> : MonoBehaviour
```

Εικόνα 5.23: Δήλωση Singleton Με Generic Type T

Για να υλοποιήσω την singleton, αρχικά απαγόρευσα πρόσβαση στον constructor της κλάσης που θέλω να κάνει inherit την singleton (δηλαδή την CharacterCustomization class), κάνοντας private τον constructor της. Έπειτα μέσα στην singleton όρισα μία private static μεταβλητή “instance” του generic type "T", ο οποίος είναι ο τύπος της κλάσης η οποία κάνει inherit την singleton.

```
private static T instance;
```

Εικόνα 5.24: Static Variable instance

Οι static μεταβλητές είναι μεταβλητές από τις οποίες δεν μπορείς να δημιουργήσεις πολλαπλά instance, οπότε εφόσον θέλω η singleton να δημιουργεί μόνο ένα instance της κλάσης, έθεσα αυτή τη μεταβλητή ως static. Μετά μέσα στην Awake function του Unity ελέγχω αν υπάρχει ήδη κάποιο instance μέσω της μεταβλητής που έφτιαξα. Αν υπάρχει ήδη, τότε καταστρέφω το καινούργιο instance component και κρατάω μόνο το πρώτο instance του script, επειδή θέλω να υπάρχει μόνο ένα component του CharacterCustomization script μέσα στην σκηνή. Αν δεν υπάρχει κάποιο instance, τότε αποθηκεύω το instance με την χρήση του GetComponent<T> (το οποίο παίρνει από το GameObject της σκηνής, πάνω στο οποίο έχει περαστεί το CharacterCustomization script και κατ' επέκταση και την Singleton class αφού την κάνει inherit), το component που που αντιστοιχεί στον generic type “T”, δηλαδή σε αυτή την περίπτωση το CharacterCustomization Script component. Πιο αναλυτικά, το script CharacterCustomization των blendshapes, είναι περασμένο ως component μέσα στο CharacterCustomization GameObject της σκηνής. Όταν φορτώνεται η σκηνή, πρώτα καλούνται οι μέθοδοι Awake της Unity. Εφόσον η CharacterCustomization class είναι περασμένη στο gameobject της σκηνής, με το φόρτωμα της σκηνής καλείται αυτή η κλάση και επειδή αυτή η κλάση κάνει inherit την singleton, με την σειρά της καλείται και η Awake function της Singleton. Έτσι περνιέται στην singleton η κλάση CharacterCustomization μέσω του generic type T και μετά το GetComponent<T> παίρνει το CharacterCustomization component του gameobject της σκηνής, με το οποίο αποθηκεύει αυτό το instance της κλάσης. Αν υπάρχει ήδη ένα component instance αυτής της κλάσης μέσα στην σκηνή, τότε καταστρέφει όλο το αντικείμενο που περιέχει το καινούργιο component instance.

Επίσης για να λειτουργήσει η εντολή GetComponent<T>, έπρεπε να προσθέσω ένα generic constraint, το οποίο σιγουρεύει οτι το T θα πρέπει αναγκαστικά να προέρχονται από κλάση που κάνει inherit την MonoBehaviour. Αυτό έγινε επειδή χώρις αυτό το Unity δεν ήξερε τι είναι αυτό το T και δεν το αναγνώσιζε ως Unity component. Έτσι με αυτό το constraint δήλωσα οτι το T θα είναι σίγουρα

```
Unity Message | 0 references
public void Awake()
{
    //If instance doesn't exist set the instance and if it does exist destroy the game object and only keep the first instance
    if (instance == null)
    {
        instance = GetComponent<T>();
    }
    else
        Destroy(gameObject);
}
```

Εικόνα 5.25: Get instance & Destroy Any Other instance

ένα Unity component. Η δήλωση ήταν η εξής "public class Singleton<T> : MonoBehaviour where T : MonoBehaviour".

```
public class Singleton<T> : MonoBehaviour where T : MonoBehaviour
```

Εικόνα 5.26: Generic Constraint

Και τέλος πρόσθεσα ένα public static property του generic type "T" και το ονόμασα Instance. Τα properties, είναι ένας ευέλικτος τρόπος για να ορίσω τιμές σε private μεταβλητές, καθώς και να τις τραβήξω. Δηλαδή μπορώ να ορίσω τον τρόπο που θέλω να χειρίζεται τις μεταβλητές κάνοντας επιπλέον πράγματα όπως για παράδειγμα να τραβάει μία μεταβλητή και να δείχνει ένα μήνυμα. Η υλοποίηση της γίνεται βάζοντας απλά την εντολή "get" και μέσα στις αγκύλες βάζω τον τρόπο που θέλω να τραβήξει την τιμή και προαιρετικά μπορώ να βάλεις και ένα "set" και μες στις αγκύλες να ορίσω το τρόπο που θέλω να αλλάξει την τιμή της μεταβλητής. Αυτό το property επιστρέφει απλός την private μεταβλητή instance, επειδή δεν μπορείς να την καλέσεις απευθείας λόγω του οτι είναι private. Του δήλωσα μόνο το "get", γιατί δεν χρειάζομαι να κάνει "set", καμία τιμή (αυτό γίνεται αυτόματα).

```
//Creates Instance property with only "get" because we don't want to "set" the instance, it is done automatically
2 references
public static T Instance
{
    get
    {
        if (instance == null)
            print("Instance of GameObject does not exist!");

        return instance;
    }
}
```

Εικόνα 5.27: Static Property Instance

5.3.3 BlendShapeSlider Script

Το Script αυτό βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → Blendshapes. Για αρχή πρόσθεσα το UI library με την "using UnityEngine.UI" ώστε να μπορώ να δημιουργήσω μεταβλητή για το slider και δήλωσα ως dependency το slider component με την εντολή [RequireComponent(typeof(Slider))]. Αυτό σημαίνει ότι στο αντικείμενο που προστίθεται αυτό το script πρέπει οπωσδήποτε να υπάρχει ένα slider και αν δεν υπάρχει τότε δημιουργεί αυτόματα ένα

slider component. Η εντολή αυτή μπαίνει πάνω από τη δήλωση της κλάσης. Έπειτα όρισα ένα Header το οποίο είναι απλώς ένα μήνυμα πού εμφανίζεται πάνω από τη μεταβλητή στον inspector, για να δώσω κάποιες οδηγίες για την εισαγωγή των τιμών στη μεταβλητή και μετά δήλωσα δύο μεταβλητές:

- Μία public string μεταβλητή που ονόμασα blendShapeName και η οποία θα αποθηκεύει το όνομά του blendshape
- Μία private slider μεταβλητή που την ονόμασα slider και η οποία αποθηκεύει το slider component από τα gameobject.

Στην συνέχεια μέσα στην start μέθοδο του Unity χρησιμοποίησα τη μέθοδο trim πάνω στην μεταβλητή blendShapeName, η οποία αφαιρεί τυχόν κενούς χαρακτήρες που μπορεί να υπάρχουν στην αρχή ή στο τέλος του ονόματος, γιατί αυτό μπορεί να δημιουργήσει προβλήματα μετά και πέρασα το καινούργιο string μέσα στην ίδια μεταβλητή. Μετά τράβηξα το slider component από το αντικείμενο μέσω της εντολής GetComponent και το πέρασα στη μεταβλητή slider.

```
blendShapeName = blendShapeName.Trim(); //Trims spaces at start or end of name that might exist
slider = GetComponent<Slider>();
```

Εικόνα 5.28: Trim & Component Slider

Τέλος χρησιμοποίησα τη μεταβλητή slider για να καλέσω το event OnValueChanged του slider, το οποίο ενεργοποιείται όταν αλλάζει τιμή το slider και πέρασα την αλλαγμένη τιμή στη μέθοδο ChangeBlendshapeValue της CharacterCustomization class, την οποία κάλεσα με τη χρήση του Instance property που δημιούργησα με τη singleton. Σαν ορίσματα της μεθόδου πέρασα την μεταβλητή blendShapeName, που έχει το όνομα του blendshape, καθώς και την καινούργια τιμή από το slider.

```
slider.onValueChanged.AddListener(value => CharacterCustomization.Instance.ChangeBlendshapeValue(blendShapeName, value));
```

Εικόνα 5.29: Create Listener For OnValueChanged Of Slider

5.3.4 CharacterCustomization Script

Το Script αυτό βρίσκεται μέσα στον φάκελο Assets → Custom Assets → Scripts → Blendshapes και είναι το κύριο script, που μπαίνει στο CharacterCustomization GameObject της σκηνής, και μέσα από το οποίο γίνονται οι ρυθμίσεις των blendshapes. Επίσης μπορεί αν υπάρχει μόνο ένα instance αυτού του script στην σκηνή, το οποίο εξασφαλίζεται μέσω της singleton που δημιούργησα (βλ. Ενότητα 5.3.2). Για να χρησιμοποιηθεί η singleton, από την μεριά αυτού του script, πρέπει να κάνει inherit την singleton δηλώνοντας “public class CharacterCustomization : Singleton<CharacterCustomization>” και μετά να απαγορεύσω την πρόσβαση στον constructor, κάνοντας τον private.

```
public class CharacterCustomization : Singleton<CharacterCustomization>
```

Εικόνα 5.30: Δήλωση Κλάσης Ως Singleton

5.3.4.1 Μεταβλητές Κλάσης

```
public SkinnedMeshRenderer target;
public string suffixMax = "Max", suffixMin = "Min";

//private constructor in order to prevent access to the constructor from any class.
//It can only be done through the singleton
0 references
private CharacterCustomization() { }

private SkinnedMeshRenderer skmr;
private Mesh mesh;

private Dictionary<string, Blendshape> blendShapeDatabase = new Dictionary<string, Blendshape>();
private Dictionary<string, float> savedBlendshapeValuesDatabase = new Dictionary<string, float>();
```

Εικόνα 5.31: Μεταβλητές της CharacterCustomization class

Σε αυτή την κλάση δημιούργησα τις εξής μεταβλητές:

- Μία public μεταβλητή τύπου SkinnedMeshRenderer που την ονόμασα “target”. Αυτή η μεταβλητή αποθηκεύει το skinnedMeshRenderer του μοντέλου που περιέχει τα blendshapes, δηλαδή το κεφάλι του modular χαρακτήρα.
- Δύο public string μεταβλητές που ονόμασα suffixMax και suffixMin. Αυτές αποθηκεύουν τα επιθέματα των blendshape. Έχουν μια αρχική τιμή αλλά μπορεί αν αλλαχτεί και από τον inspector.
- Μία private μεταβλητή τύπου SkinnedMeshRenderer που την ονόμασα “skmr”. Αυτή η μεταβλητή αποθηκεύει το skinnedMeshRenderer του μοντέλου από την target μεταβλητή.
- Μία private Mesh μεταβλητή που αποθηκεύει το mesh του μοντέλου.
- Ένα private dictionary που ονόμασα blendShapeDatabase και αποθηκεύει όλα τα blendshapes (χωρίς τα επιθέματα) και τα index τους (και αυτών με το max και αυτών με το min). Το dictionary δέχεται το όνομα του blendshape (χωρίς τα επιθέματα) και μια μεταβλητή τύπου Blendshape (βλ. Ενότητα 5.3.1).
- Ένα private dictionary που ονόμασα savedBlendshapeValuesDatabase και αποθηκεύει τις επιλογές των blendshapes που έγιναν από τον παίχτη. Το dictionary δέχεται το όνομα του blendshape (χωρίς τα επιθέματα) και μια μεταβλητή τύπου float που είναι η τιμή του slider (από -100 μέχρι 100), που επέλεξε ο παίχτης.

5.3.4.2 Μέθοδος Initialize()

Αυτή η μέθοδος καλείται από την Start μέθοδο του Unity και αποθηκεύει στην μεταβλητή “skmr”, το skinnedmeshrenderer του κεφαλιού από το “target”, καθώς και το Mesh του “target” το

οποίο αποθηκεύεται στην μεταβλητή “mesh” και μετά καλεί την private μέθοδο “ParseBlendShapesToDictionary” (βλ. Ενότητα 5.3.4.3).

```
3 references
public void Initialize()
{
    skmr = target;
    mesh = skmr.sharedMesh;

    ParseBlendShapesToDictionary();
}
```

Εικόνα 5.32: Μέθοδος Initialize

5.3.4.3 Μέθοδος ParseBlendShapesToDictionary()

Αυτή η μέθοδος περνάει όλα τα blendshapes μέσα στο blendShapeDatabase dictionary που δημιούργησα, δηλαδή περνάει το όνομα των blendshapes χωρίς τα επιθέματα και την μεταβλητή τύπου Blendshape, που περιέχει το positiveIndex και το negativeIndex.

Αρχικά δημιουργώ μία λίστα που ονόμασα "blendshapeNames" και με τη χρήση της εντολής "Enumerable.Range", περνάω όλα τα ονόματα των blendshapes από το Mesh του μοντέλου σε ένα IEnumerable<string>, το οποίο στη συνέχεια μετατρέπω σε λίστα και αποθηκεύεται μέσα στη λίστα που ονόμασα blendshapeNames. Η εντολή "Enumerable.Range", στην ουσία δημιουργεί μία σειρά από integer μεταξύ του διαστήματος που ορίζω (0 έως τον αριθμό των blendshapes) και μετά μέσα στην "Select" χρησιμοποιώ αυτά τα integer για να επιστρέψω τα ονόματα των blendshapes τα οποία βρίσκονται στις αντίστοιχες θέσεις μέσα στα blendshapes του μοντέλου.

```
//Get all blendshape names.
//Loop through all blendshapes and passes the names to a IEnumerable<String>, which then converts to a List<string>
List<string> blendshapeNames = Enumerable.Range(0, mesh.blendShapeCount).Select(x => mesh.GetBlendShapeName(x)).ToList();
```

Εικόνα 5.33: Enumerable.Range

Στην συνέχεια δημιουργώ ένα "for" loop το οποίο περνάει όλα τα ονόματα τον blendshape της λίστας που δημιούργησα (blendshapeNames). Σε αυτό το loop δεν αυξάνω την μεταβλητή "i". Αντί για αυτό το "i" παραμένει μηδέν, παίρνω πάντα το πρώτο αντικείμενο της λίστας, μετά βρίσκω το αντίθετο του μέσα στην λίστα (δηλαδή πχ αν στην πρώτη θέση της λίστας υπάρχει το cheekMax, βρίσκω και το cheekMin)

```
if (blendshapeNames[i].EndsWith(suffixMax))
{
    altSuffix = noSuffix + " " + suffixMin;
```

Εικόνα 5.34: Αποθήκευση Αντίθετου Blendshape Name Σε Μεταβλητή

και στο τέλος του loop αφαιρώ τα blendshapes (βλ. Εικόνα 5.41) με τα δύο διαφορετικά επιθέματα "max" και "min" (αν υπάρχουν δύο γιατί υπάρχει και περίπτωση να υπάρχει μόνο ένα, δηλαδή να έχει

μόνο θετικές τιμές), τα οποία επεξεργάστηκα και έτσι μειώνονται τα αντικείμενα που υπάρχουν μέσα στη λίστα. Το loop τελειώνει όταν η λίστα αδειάσει. Επίσης το "i" είναι πάντα 0, επειδή μετά τη διαγραφή των blendshapes που επεξεργάστηκα, το επόμενο αντικείμενο της λίστας στη θέση 1 πηγαίνει στη θέση 0 και έτσι στο επόμενο loop επεξεργάζεται το επόμενο blendshape.

```
for (int i = 0; blendshapeNames.Count > 0;)
```

Εικόνα 5.35: For Loop Χωρίς Αύξηση του i

Διαμόρφωσα το loop με αυτό τον τρόπο για τους εξής λόγους:

- Το "foreach" loop δεν θα λειτουργούσε τόσο καλά επειδή θέλω τα blendshape που έχουν τα δύο διαφορετικά επιθέματα "max" και "min", να αναγνωρίζονται ως ένα blendshape (δηλαδή με κάθε πέρασμα να επεξεργάζεται και το Max και το Min), ενώ το "foreach" loop θα περνούσε ένα-ένα τα αντικείμενα ξεχωριστά μέσα από τη λίστα και επίσης όπως είπα και πριν, κάποια blendshapes μπορεί να έχουνε μόνο θετικούς αριθμούς, δηλαδή να πηγαίνουν από 0 έως 100 αντί για -100 έως 100, οπότε θα μπερδευόταν πολύ ο κώδικας.
- Το κανονικό “for” loop που αυξάνει το “i”, επίσης δεν θα λειτουργούσε καλά για τους ίδιους λόγους και επίσης τα δύο blendshapes με τα διαφορετικά επιθέματα θα μπορούσαν να βρίσκονται σε τυχαίες θέσεις μέσα στη λίστα των blendshapes και όχι στη σειρά το ένα μετά το άλλο. Οπότε ακόμα και αν έβαζα έναν έλεγχο για να ελέγξει αν το συγκεκριμένο blendshape έχει και θετικές και αρνητικές τιμές, ώστε να επεξεργαστεί και τα δύο, θα έπρεπε μετά, είτε να αυξήσω το “i” +2 ώστε να προσπεράσει και τα 2 blendshapes (αν αυτά βρίσκονταν στη σειρά) ή να αποθηκεύσω κάπου την αντίστοιχη θέση του αντίθετου blendshape, ώστε να την προσπεράσει μετά. Για παράδειγμα αν το cheekMax είναι στη θέση 3 και το cheekMin είναι σε θέση 6, θα έπρεπε να αυξήσω το “i” +1 για να πάει από το 3 στο 4 και μετά να προσπεράσω τη θέση 6 ή αν ήταν στη σειρά, για παράδειγμα στις θέσεις 3 και 4, θα έπρεπε να αυξήσω το “i” +2 και αν το cheek είχε μόνο ένα blendshape (χωρίς Max και Min), θα έπρεπε να το αυξήσω +1.

Μετά εσωτερικά του loop δημιουργώ δύο string μεταβλητές altSuffix και noSuffix (οι οποίες θα αποθηκεύουν αντίστοιχα, το όνομα του αντιθέτου blendshape και το όνομα του blendshape χωρίς επιθέματα), δύο string που τα ονόμασα positiveName και negativeName (τα οποία θα κρατάνε το όνομα του Max και το όνομά του Min blendshape), μία μεταβλητή boolean την οποία ονόμασα exists (η οποία χρησιμοποιείται για να ελέγξω αν το συγκεκριμένο blendshape έχει και αντίθετο και έχει την αρχική τιμή false) και τέλος δύο int μεταβλητές που τις ονόμασα positiveIndex και negativeIndex (οι οποίες αποθηκεύουν την θέση του θετικού και του αρνητικού blendshape, μέσα από τη λίστα των blendshape του μοντέλου). Στη συνέχεια παίρνω το όνομα του τωρινού blendshape και με την εντολή trimEnd κόβω από το τέλος το επίθεμα Max (αν δεν υπάρχει, δεν κόβει τίποτα), μετά το ξαναπερνάω από την εντολή trimEnd, αλλά αυτή τη φορά κόβοντας το επίθεμα Min (αν δεν υπάρχει, δεν κόβει τίποτα) και τέλος το ξανά περνάω από ένα trim, για να κόψει και τυχόν κενούς χαρακτήρες.

```
noSuffix = blendshapeNames[i].TrimEnd(suffixMax.ToCharArray()).TrimEnd(suffixMin.ToCharArray()).Trim();
```
Εικόνα 5.36: Trim Suffixes and Spaces

Έπειτα κάνω τρεις ξεχωριστούς ελέγχους για να δω αν το όνομα του τωρινού blendshape, τελειώνει με το επίθεμα Max ή αν τελειώνει με το επίθεμα Min ή αν δεν έχει κανένα από τα δύο επιθέματα.

- Αν τελειώνει με το επίθεμα Max, τότε αποθηκεύω στη μεταβλητή altSuffix το όνομα του αντίθετου blendshape, βάζοντας το όνομα του blendshape που έχω αποθηκεύσει χωρίς τα επιθέματα και κολλάω στο τέλος το επίθεμα Min. Μετά αποθηκεύω στη μεταβλητή positiveName το τωρινό blendshape και στην μεταβλητή negativeName το αντίθετο blendshape, χρησιμοποιώντας τη μεταβλητή altSuffix που έφτιαξα πριν λίγο. Έπειτα ελέγχω αν η λίστα των ονομάτων περιέχει αυτό το αντίθετο όνομα που έφτιαξα με τη μεταβλητή altSuffix, δηλαδή ελέγχει αν το συγκεκριμένο blendshape έχει και θετικές και αρνητικές τιμές και αν έχει τότε θέτω τη μεταβλητή exists σε true. Στην συνέχεια χρησιμοποιώντας το όνομα που έβαλα στο positiveName, παίρνω το Index του blendshape από τη λίστα των blendshapes του μοντέλου, και το βάζω στη μεταβλητή positiveIndex. Το ίδιο κάνω και για το αντίθετο blendshape, με τη χρήση των μεταβλητών negativeIndex και altSuffix, αφού όμως πρώτα ελέγξω αν υπάρχει το αντίθετο blendshape, με τη χρήση της μεταβλητής exists.

```
//If Suffix is Positive (Max)
if (blendshapeNames[i].EndsWith(suffixMax))
{
    altSuffix = noSuffix + " " + suffixMin;

    positiveName = blendshapeNames[i];
    negativeName = altSuffix;

    if (blendshapeNames.Contains(altSuffix))
        exists = true;

    positiveIndex = mesh.GetBlendShapeIndex(positiveName);

    if (exists)
        negativeIndex = mesh.GetBlendShapeIndex(altSuffix);
}
```
Εικόνα 5.37: Ends With Suffix Max

- Αν τελειώνει με το επίθεμα Min, τότε κάνω ακριβώς τα ίδια απλώς με ανεστραμμένα τα θετικά με τα αρνητικά.

- Αν δεν έχει κανένα από τα δύο επιθέματα, τότε αποθηκεύει απλά το Index του blendshape μέσα στη μεταβλητή positive Index και μετά αποθηκεύει και το όνομα του blendshape χωρίς τα επιθέματα, με τη χρήση της μεταβλητής noSuffix. Ο λόγος που αντιγράφω το όνομα από την μεταβλητή noSuffix στην μεταβλητή positiveName, είναι γιατί οι μεταβλητές positiveName και negativeName είναι αυτές που χρησιμοποιούνται στο τέλος του loop ώστε να αφαιρέσουν τα αντικείμενα που επεξεργάστηκαν από τη λίστα και έτσι όταν αδειάσει η λίστα να τελειώσει το loop. Αν κάποιο blendshape δεν αποθηκευτεί, σε καμία από αυτές τις δύο μεταβλητές, τότε δεν θα αφαιρεθεί ποτέ από τη λίστα και έτσι το loop θα γίνει ατέρμονο.

```
else
{
    positiveIndex = mesh.GetBlendShapeIndex(blendshapeNames[i]);

    positiveName = noSuffix; //This is here so it will remove it (for loop condition) so it's not infinite loop.
}
```

Εικόνα 5.39: No Suffix

```
    if (blendshapeNames.Contains(altSuffix))
        exists = true;

    negativeIndex = mesh.GetBlendShapeIndex(negativeName);

    if (exists)
        positiveIndex = mesh.GetBlendShapeIndex(altSuffix);
}
```

Εικόνα 5.38: Ends With Suffix Min

Μετά από αυτούς τους ελέγχους κάνω έναν έλεγχο για να δω αν υπάρχει μέσα στο blendShapeDatabase dictionary (βλ. Ενότητα 5.3.4.1), κάποια εγγραφή με κλειδί το όνομα του blendshape χωρίς τα επιθέματα. Αν υπάρχει, σημαίνει ότι αυτό το blendshape έχει ήδη περαστεί μέσα στο dictionary που περιέχει όλα τα blendshapes και βγάζει ένα error που λέει όταν το συγκεκριμένο blendshape υπάρχει στη βάση και έτσι σταματάει η εκτέλεση του προγράμματος. Αν δεν υπάρχει, τότε προσθέτει μία καινούργια εγγραφή μέσα στη βάση για αυτό το blendshape.

```
if (blendShapeDatabase.ContainsKey(noSuffix))
    Debug.LogError(noSuffix + " already exists within the Database!");

blendShapeDatabase.Add(noSuffix, new Blendshape(positiveIndex, negativeIndex));
```

Εικόνα 5.40: Add New Blendshape in Dictionary

Τέλος κάνω έναν έλεγχο για να δω αν η μεταβλητή positiveName έχει κάποια τιμή και αν έχει τότε αφαιρώ αυτό το όνομα από τη λίστα με τα ονόματα των blendshape που ονόμασα blendshapeNames. Εδώ το positiveName μπορεί να περιέχει είτε το όνομα ενός blendshape με το επίθεμα Max ή το όνομα ενός blendshape χωρίς κανένα από τα δύο επιθέματα. Έπειτα ο ίδιος έλεγχος γίνεται και για το negativeName, ώστε να ελέγξει αν υπάρχει αυτό το blendshape και με το επίθεμα Min, για να αφαιρέσει και αυτό από τη λίστα και έτσι μόλις αδειάσει η λίστα, να τελειώσει το loop (βλ. Εικόνα 5.35).

```
//Remove selected indexes from the list, in order for the loop to end
if (positiveName != string.Empty)
    blendshapeNames.Remove(positiveName);
if (negativeName != string.Empty)
    blendshapeNames.Remove(negativeName);
```

Εικόνα 5.41: Remove Items of List to End Loop

5.3.4.4 Μέθοδος ChangeBlendshapeValue()

Αυτή είναι η μέθοδος η οποία αλλάζει τις τιμές των slider (αυτά που είναι μέσα στον inspector) από τα blendshapes του μοντέλου και παίρνει ως ορίσματα μία string μεταβλητή που ονομάζεται blendshapeName και περιέχει το όνομα του blendshape και μία float που ονομάζεται Value και περιέχει την τιμή των slider (αυτά που βρίσκονται μέσα στο GUI του μενού του παιχνιδιού),

η οποία κυμαίνεται από - 100 έως 100. Έτσι αλλάζοντας τα slider που βρίσκονται μέσα στο παιχνίδι αλλάζουν και αναλόγως τα slider του μοντέλου, που βρίσκονται μέσα στον editor. Η μέθοδος αυτή καλείται μέσω του OnValueChanged event στην BlenderShapeSlider class (βλ. Ενότητα 5.3.3).

Αρχικά ελέγχω αν στο blendShapeDatabase dictionary (βλ. Ενότητα 5.3.4.1), υπάρχει κλειδί με αυτό το blendshapeName και αν δεν υπάρχει βγάζει ένα error και τερματίζει το πρόγραμμα. Μετά περνάω το Value μέσω της Mathf.Clamp, ώστε να περιορίσει την τιμή του Value μεταξύ των επιτρεπτών ορίων που έχω ορίσει (από -100 έως 100).

```
value = Mathf.Clamp(value, -100, 100);
```

Εικόνα 5.42: Clamp

Αν η τιμή είναι μεταξύ των ορίων τότε επιστρέφει απλά την τιμή και αν είναι εκτός ορίων πότε επιστρέφει τη μέγιστη ή την ελάχιστη τιμή, που έχω ορίσει, αντίστοιχα. Οι αρνητικές τιμές αλλάζουν το slider (του μοντέλου μέσα στον editor, όχι αυτό στο GUI) με το επίθεμα Min και θετικές τιμές αλλάζουν το slider με το επίθεμα Max (ή το slider που δεν έχει κανένα από τα δύο επιθέματα σε περίπτωση που υπάρχει μόνο ένα).

Μετά δημιουργώ μία μεταβλητή τύπου Blendshape (βλ. Ενότητα 5.3.1) και της περνάω την αντίστοιχη μεταβλητή τύπου Blendshape, μέσα από το blendShapeDatabase dictionary. Έπειτα ελέγχω αν μέσα στο savedBlendshapeValuesDatabase dictionary (μέσα στο οποίο αποθηκεύονται οι επιλογές του παίκτη για να περαστούν αργότερα μέσα στη σκηνή του παιχνιδιού βλ. Ενότητα 5.3.4.1) υπάρχει κλειδί με όνομα που αντιστοιχεί σε αυτό το blendshape και αν δεν υπάρχει, τότε προσθέτω μία καινούργια εγγραφή με το όνομα του blendshape ως κλειδί και ως τιμή το Value του slider, το οποίο κυμαίνεται από - 100 έως 100. Αν όμως υπάρχει αυτό το blendshape μέσα στο dictionary, τότε απλά αλλάζει την τιμή της υπάρχων εγγραφής. Επιπλέον μέσα σε αυτή τη μέθοδο περνιούνται μόνο τα blendshape των οποίων το slider έχεις αλλάξει μέσα από το GUI του παιχνιδιού. Αυτό σημαίνει ότι και σε αυτό το dictionary αποθηκεύονται μόνο τα blendshape τα οποία έχεις πειράξει, καθώς τα υπόλοιπα blendshape δεν χρειάζεται να αποθηκευτούν, γιατί φορτώνουν απλά οι προεπιλεγμένες τιμές, οι οποίες είναι το μηδέν.

```
//If statement to save in dictionary only the values of the blendshapes that have changed. The rest are 0 by default
//in the inspector.
if (!savedBlendshapeValuesDatabase.ContainsKey(blendshapeName))
    savedBlendshapeValuesDatabase.Add(blendshapeName, value);
else
    savedBlendshapeValuesDatabase[blendshapeName] = value;
```

Εικόνα 5.43: Save Blendshape Value To Load On Character In Game Scene

Στην συνέχεια ελέγχω αν το value του slider είναι θετικός ή αρνητικός αριθμός. Αν είναι θετικός τότε σημαίνει οτι αναφέρεται στο blendshape με επίθεμα "Max". Σε αυτή την περίπτωση ελέγχω πρώτα αν αυτό το blendshape έχει positiveIndex (δηλαδή αν το positiveIndex είναι διάφορο του -1, που σημαίνει, είτε ότι υπάρχει blendshape με το επίθεμα Max ή χωρίς κανένα από τα δύο επιθέματα). Αν δεν υπάρχει τότε τελειώνει η κλήση της μεθόδου. Αν όμως υπάρχει τότε

χρησιμοποιώντας την μεταβλητή skmr (βλ. Ενότητα 5.3.4.1) αλλάζω την τιμή του blendshape, πού βρίσκεται σε αυτό το index, για να είναι ίδια με την τιμή του Value. Έπειτα ελέγχω αν το συγκεκριμένο blendshape υπάρχει και με το επίθεμα Min και αν υπάρχει, τότε αλλάζω την τιμή του blendshape πού βρίσκεται σε αυτό το index, σε μηδέν. Αυτό γίνεται επειδή τα blendshapes του μοντέλου μέσα στον editor έχουν δύο διαφορετικά blendshape sliders για κάθε ένα από τα blendshape με διαφορετικό επίθεμα (δηλαδή ξεχωριστό slider για το blendshape με επίθεμα Max και διαφορετικό για το επίθεμα Min). Όλα αυτά τα slider μέσα στον editor παίρνουν τιμές από 0 έως 100, όποτε θέλω όταν το ένα slider αυξάνεται το άλλο να μηδενίζεται.

```
if (blendshape.positiveIndex == -1)
    return;
//Set blendshape value for blendshape in this index. Could be blendshape with Max or no suffix.
skmr.SetBlendShapeWeight(blendshape.positiveIndex, value);

//If negativeIndex is -1 then the blendshape has no suffix, but if it is not -1,
//then min suffix exists and so it sets it to 0
if (blendshape.negativeIndex == -1)
    return;
skmr.SetBlendShapeWeight(blendshape.negativeIndex, 0);
```

Εικόνα 5.44: Set Blendshape If Value is Positive

Έπειτα κάνω το ίδιο και για τους αρνητικούς αριθμούς του Value με τη διαφορά ότι το κάνω ανάποδα, δηλαδή πρώτα ελέγχω για τον negativeIndex και βάζω το value εκεί και μετά μηδενίζω το blendshape slider στο positiveIndex. Επιπλέον επειδή όλα τα slider μέσα στον editor παίρνουν τιμές από 0 έως 100, αλλά το slider μέσα στη σκηνή του παιχνιδιού όταν θέλω να αναφερθεί στα blendshape με το επίθεμα Min (δηλαδή όταν το slider στη σκηνή του παιχνιδιού πηγαίνει προς τα αριστερά και έτσι παίρνει τιμές από -1 έως -100), παίρνει αρνητικές τιμές, πρέπει η τιμή του Value να γίνει από αρνητική σε θετική. Αυτό έγινε απλώς βάζοντας το μείον(“-”) μπροστά από μεταβλητή Value.

```
//If slider move to the left between -100 and -1 which is the min suffix of the blendshape
else
{
    if (blendshape.negativeIndex == -1)
        return;
    skmr.SetBlendShapeWeight(blendshape.negativeIndex, -value);
    if (blendshape.positiveIndex == -1)
        return;
    skmr.SetBlendShapeWeight(blendshape.positiveIndex, 0);
}
```

Εικόνα 5.45: Set Blendshape If Value is Negative

5.3.4.5 Μέθοδος LoadBlendshapesToCharacter()

Αυτή η μέθοδος είναι παρόμοια με τη μέθοδο ChangeBlendshapeValue της ενότητας 5.4.4.4, με τη διαφορά ότι αυτή παίρνει τις τιμές για τα blendshape, από το dictionary αντί για τα slider μέσα στο μενού και χρησιμοποιείται για να φορτώσει τις αλλαγές που έκανε ο παίχτης πάνω στα blendshape και οι οποίες αποθηκεύτηκαν μέσα στο savedBlendshapeValuesDatabase dictionary (βλ.

Ενότητα 5.3.4.1) με τη χρήση της μεθόδου ChangeBlendshapeValue. Η μέθοδος αυτή καλείται μέσα από SetPlayableCharacterAppearance Script (βλ.Ενότητα 5.10.1) το οποίο εισάγεται, ως component, μέσα στο humanoid rig μοντέλο του χαρακτήρα στη σκηνή του παιχνιδιού, για να εισάγει στον χαρακτήρα τις διάφορες επιλογές πού έγιναν στις σκηνές του μενού.

Σε αυτή τη μέθοδο περνάω ως παράμετρο το SkinnedMeshRenderer του μοντέλου που περιέχει τα blendshapes, πάνω στο οποίο θέλω να περαστούν οι αποθηκευμένες ρυθμίσεις των blendshape (δηλαδή το humanoid rig μοντέλο της σκηνής). Μετά δημιουργώ μία μεταβλητή τύπου Blendshape που την ονόμασα "blendshape" και μία μεταβλητή τύπου float που την ονόμασα "value". Στην συνέχεια χρησιμοποιώντας την foreach, περνάω από όλες τις εγγραφές του dictionary "savedBlendshapeValuesDatabase" και βάζω την τιμή της κάθε εγγραφής, μέσα στη μεταβλητή value που έφτιαξα και χρησιμοποιώντας το κλειδί της κάθε εγγραφής, παίρνω από το blendShapeDatabase dictionary, την τιμή τύπου Blendshape και την περνάω μέσα στη μεταβλητή blendshape που έφτιαξα μέσα σε αυτή τη μέθοδο.

```
foreach (KeyValuePair<string, float> item in savedBlendshapeValuesDatabase)
{
    value = item.Value; //Get saved blendshape float value from current item
    blendshape = blendShapeDatabase[item.Key]; //Get blendshape from blendShapeDatabase dictionary
```

Εικόνα 5.46: Εισαγωγή Τιμών Από SavedBlendshapeValuesDatabase Dictionary

Ο υπόλοιπος κώδικας ρυθμίζει απλώς τα blendshape sliders στον editor, με τον ίδιο τρόπο που έγινε και στη μέθοδο ChangeBlendshapeValue (βλ. Ενότητα 5.3.4.4).

5.3.4.6 Μικρές Μέθοδοι

Τέλος υπάρχουν και μερικές ακόμα μέθοδοι οι οποίες είναι πολύ μικρές για να κάνω ξεχωριστές ενότητες για την κάθε μία. Αυτές οι μέθοδοι χρησιμοποιούνται για τα custom editor scripts που έφτιαξα (βλ. Ενότητες 5.3.5 & 5.3.6) και είναι οι εξής:

- GetBlendShapeNames(): Αυτή η μέθοδος επιστρέφει ένα πίνακα string που περιέχει όλα τα κλειδιά του blendShapeDatabase dictionary σε μορφή array, δηλαδή επιστρέφει ένα πίνακα με όλα τα ονόματα των blendshape, χωρίς τα επιθέματα.

```
public string[] GetBlendShapeNames()
{
    return blendShapeDatabase.Keys.ToArray(); //Create array from the dictionary keys
}
```

Εικόνα 5.47: Create Array Of Dictionary Keys

- GetNumberOfEntries(): Αυτή η μέθοδος επιστρέφει τον αριθμό των εγγράφων του blendShapeDatabase dictionary.
- GetBlendshape(): Αυτή η μέθοδος δέχεται ως όρισμα το όνομα ενός blendshape και επιστρέφει την μεταβλητή τύπου Blendshape από το blendShapeDatabase dictionary, που αντιστοιχεί σε αυτό το όνομα.

- DoesTargetMatchSkmr(): Αυτή η μέθοδος ελέγχει αν η μεταβλητή "target" τύπου SkinnedMeshRenderer, είναι ίδια με τη μεταβλητή "skmr" και αναλόγως επιστρέφει true ή false. Αυτό χρησιμοποιείται ώστε το custom editor που έφτιαξα, να ενημερώνετε αν έχει αλλαχθεί το πεδίο που περιέχει το SkinnedMeshRenderer του μοντέλου ώστε να ενημερώσει τα παιδιά του editor αναλόγως.
- ClearDatabase(): Αυτή η μέθοδος αδειάζει όλες τις εγγραφές το blendShapeDatabase dictionary.

5.3.5 BlendShapeSliderEditor Script

Με αυτό το script διαμορφώνω έναν custom editor για την BlendShapeSlider class. Με αυτό το editor μπορώ να επιλέξω το blendshape που θέλω να επηρεάζει το τρέχον slider, είτε μέσω ενός dropdown menu ώστε να αποφευχθούν τυχόν τυπογραφικά λάθη, είτε γράφοντας το χειροκίνητα, το οποίο μπορεί να χρησιμοποιηθεί και σαν αναζήτηση ώστε να μην ψάχνεις αυτό που θέλεις από τη λίστα σε περίπτωση που έχεις πάρα πολλές επιλογές. Αυτό το script βρίσκεται στον φάκελο Assets → Editor (όλα τα editor scripts πρέπει να βρίσκονται σε αυτό το φάκελο για να λειτουργήσουν όταν κάνεις build το project).

Αρχικά δηλώνω ότι αυτό το editor script είναι ένα custom editor για την BlendShapeSlider class με τη χρήση της εντολής “[CustomEditor(typeof(BlendShapeSlider))]”. Αυτό μπαίνει ώστε όταν προσθέτεις ως component το BlendShapeSlider script, να ξέρει το Unity να φορτώσει αυτό το custom editor, αντί για το default editor της Unity. Επίσης δήλωσα ότι αυτή η κλάση κάνει inherit την Editor class της Unity.

5.3.5.1 Μεταβλητές Κλάσης

Δημιούργησα τις εξής μεταβλητές:

- Μία public μεταβλητή τύπου Enum που την ονόμασα "State" και περιέχει δύο τιμές, την “auto” και την “manual”, οι οποίες χρησιμοποιούνται για να αλλάξουν την εμφάνιση του custom editor.
- Μία public μεταβλητή τύπου State (δηλαδή της Enum που έφτιαξα), και την ονόμασα "state" στην οποία αποθηκεύεται η τωρινή τιμή της Enum.
- Μία private μεταβλητή τύπου BlendShapeSlider (βλ. Ενότητα 5.3.5) που την ονόμασα "blendShapeSlider".

```
5 references
public enum State
{
    auto,
    manual
}

public State state;
private BlendShapeSlider blendShapeSlider;
```

Εικόνα 5.48: Μεταβλητές Slider Editor

5.3.5.2 Μέθοδος Override OnInspectorGUI

Αυτή η μέθοδος κάνει override τη μέθοδο OnInspectorGUI της base class "Editor", δηλαδή εκτελείτε αυτή αντί της κανονικής. Μετά ορίζω το layout του custom editor με τις εντολές "EditorGUILayout.BeginHorizontal();" και "EditorGUILayout.EndHorizontal();". Ότι μπαίνει ανάμεσα σε αυτές τις δύο εντολές διαμορφώνεται με οριζόντια στοίχιση μέσα στον editor. Μέσα σε αυτό το layout δημιούργησα δύο κουμπιά (το "Auto" και το "Manual"), τα οποία τα έβαλα απευθείας μέσα σε if condition. Όταν πατιέται το Auto κουμπί τότε βάζει μέσα στην state μεταβλητή την τιμή " auto" της Enum και όταν πατιέται το manual κουμπί βάζει την τιμή "manual".

```
EditorGUILayout.BeginHorizontal();
Debug.Log("test");
//Creates 2 buttons and when they are pressed they return true and change the enum state to auto or manual
//and then run the GUI_Auto() or GUI_Manual() respectively
if (GUILayout.Button("Auto"))
    state = State.auto;
if (GUILayout.Button("Manual"))
    state = State.manual;

EditorGUILayout.EndHorizontal(); //End of Editor GUI Layout group.
```

Εικόνα 5.49: Buttons Inside Horizontal Layout

Έπειτα παίρνω τη μεταβλητή target (η οποία προέρχεται από την Editor class την οποία κάνω inherit και η οποία επιστρέφει το αντικείμενο το οποίο επεξεργάζομαι, δηλαδή το επιλεγμένο slider που έχει ως component το script για το οποίο έχει φτιαχτεί αυτός ο editor π.χ. το slider που ονομάζεται "Slider Cheek") και το μετατρέπω (casting) σε τύπο BlendShapeSlider και το περνάω στην μεταβλητή blendShapeSlider που έφτιαξα.

```
blendShapeSlider = (BlendShapeSlider)target;
```

Εικόνα 5.50: Cast target (of Editor class) Into BlendShapeSlider Variable

Στην συνέχεια χρησιμοποιώ ως condition στο switch την μεταβλητή "state" και αν η τιμή της είναι manual, τότε καλώ τη μέθοδο GUI_Manual αλλιώς καλώ την μέθοδο GUI_Auto (βλ. Ενότητα 5.3.5.3).

5.3.5.3 Μέθοδοι GUI_Auto & GUI_Manual

Αυτές οι δύο μέθοδοι χρησιμοποιούνται για να διαλέξω το editor που θα εμφανιστεί (default ή custom editor) στον inspector. Η manual μέθοδος εμφανίζει κάτω από τα δύο κουμπιά που έφτιαξα, το μενού του default editor που περιέχει όλα τα serialized στοιχεία που δήλωσα στην κλάση BlendShapeSlider, ενώ η auto μέθοδος, εμφανίζει κάτω από τα δύο κουμπιά που έφτιαξα, ένα άλλο custom μενού για το editor, που υλοποιώ μέσα σε αυτή την κλάση.

Δημιουργώντας αυτόν τον custom editor με αυτή την κλάση και κάνοντας override την μέθοδο OnInspectorGUI, έχει ως αποτέλεσμα να μην εμφανίζεται το editor με τον κλασικό τρόπο. Οπότε ότι ρυθμίσεις έχω κάνει μέσα στην κλάση BlendShapeSlider (όπως διάφορες serialized μεταβλητές, μηνύματα και τα λοιπά), δεν εμφανίζονται μέσα στον editor. Παρόλα αυτά μέσα στη μέθοδο GUI_Manual έχω βάλει την εντολή "base.OnInspectorGUI();", η οποία όταν εκτελεστεί, καλεί παράλληλα και την αρχική μέθοδο OnInspectorGUI της base (Editor) class. Έτσι μαζί με τα κουμπιά που έφτιαξα μέσα στο δικό μου OnInspectorGUI, εμφανίζεται και το αρχικό editor με όλα τα serialized στοιχεία που έχω δηλώσει στην κλάση BlendShapeSlider.

Πιο συγκεκριμένα, η μέθοδος OnInspectorGUI εκτελείται κάθε φορά που μετακινώ το ποντίκι μέσα στον inspector ή αλληλεπιδρώ με τον inspector με οποιοδήποτε τρόπο. Έτσι όταν πατάω ένα από τα δύο κουμπιά εκτελείται από την αρχή η μέθοδος OnInspectorGUI και εμφανίζει το layout που έχω δηλώσει μέσα σε αυτή τη μέθοδο δηλαδή τα κουμπιά. Στην συνέχεια, όταν πατάς σε ένα από τα δύο κουμπιά ενημερώνεται η μεταβλητή state και εκτελείται η αντίστοιχη μέθοδος. Αν καλέσει την GUI_Manual, τότε κάτω από τα κουμπιά εμφανίζει και τον default editor, αλλιώς εμφανίζει τις υπόλοιπες ρυθμίσεις του custom editor, που έχω βάλει μέσα στην μέθοδο GUI_Auto.

Η μέθοδος GUI_Auto, βρίσκει και αποθηκεύει το CharacterCustomization gameobject από την σκηνή, παίρνει το dictionary με τα blendshapes και εμφανίζει ένα drop down menu με το οποίο επιλέγεις το blendshape που θέλεις να επηρεάζει αυτό το slider. Η υλοποίηση του γίνεται ως εξής.

Δημιουργώ μία μεταβλητή τύπου CharacterCustomization που ονόμασα “characterCustomization” και περνάω μέσα της το gameobject που βρίσκω στην σκηνή, το οποίο έχει μέσα του την CharacterCustomization class.

```
//Get characterCustomization object from the scene
CharacterCustomization characterCustomization = GameObject.FindObjectOfType<CharacterCustomization>();
```

Εικόνα 5.51: Get GameObject That Contains CharacterCustomization Component

Μετά παίρνω τον αριθμό των εγγράφων μέσα στο dictionary που περιέχει τα blendshapes (βλ. Ενότητα 5.3.4.6) και ελέγχω αν το dictionary είναι άδειο, ώστε αν είναι άδειο να το γεμίσω με τα blendshapes (βλ. Ενότητα 5.3.4.2).

Στην συνέχεια αποθηκεύω μέσα σε ένα πίνακα τα ονόματα όλων των blendshapes, χωρίς τα επιθέματα (βλ. Ενότητα 5.3.4.6). Μετά με ένα loop περνάω από όλες τις εγγραφές του πίνακα με τα ονόματα των

```
//Gets the number of blendshapes without the suffixes (Meaning max and min are treated as one)
//from the dictionary and checks if it is filled
//If the dictionary is empty, it parses the blendshapes into the dictionary
if (characterCustomization.GetNumberOfEntries() <= 0)
    characterCustomization.Initialize();
```

Εικόνα 5.52: If Dictionary Is Empty, Parse Blendshape Into It

blendshapes και ελέγχω αν το όνομα του επιλεγμένου slider υπάρχει μέσα στη βάση των blendshape sliders, και αν υπάρχει τότε αποθηκεύω το index του.

```
//The current selectedIndex from the current blendshape UI slider
int blendShapeID = 0;

for (int i = 0; i < blendShapeNames.Length; i++)//Loops through all blendshapes
    //Checks if the blendshape name that it gets from the current blendShapeSlider, inside the inspector,
    //exists in the Blendshape Database and if it does it assigns its index to the blendShapeID variable
    if (blendShapeSlider.blendShapeName == blendShapeNames[i])
        blendShapeID = i;
```

Εικόνα 5.53: If Current Blendshape Exists In Dictionary, Save Its Index

Τέλος δημιουργώ ένα dropdown menu (EditorGUILayout.Popup) στο οποίο βάζω ως ορίσματα, ένα όνομα (για το label δίπλα από το dropdown menu) , το επιλεγμένο Index και τον πίνακα με τα ονόματα και μετά περνάω την int τιμή που επιστρέφει αυτή η εντολή σε μία μεταβλητή ώστε την χρησιμοποιήσω για να ενημερώσω και το πεδίο του default editor που περιέχει το όνομα του blendshape, ώστε όταν το αλλάξεις να εμφανίζεται και εκεί το επιλεγμένο όνομα.

```
//Creates a dropdown menu in the inspector, with BlendShapeName as a label, blendShapeID the default selected index
//and an array of strings that have the blendShapeNames and it then returns the selected index
blendShapeID = EditorGUILayout.Popup("BlendShapeName", blendShapeID, blendShapeNames);

//Sets the blendShapeName field from the blendShapeSlider class(that you can only see when you switch to manual),
//to match the new selected blendshape name from the dropdown menu in the GUI auto layout
blendShapeSlider.blendShapeName = blendShapeNames[blendShapeID];
```

Εικόνα 5.54: Create DropDown Menu & Copy Selected Blendshape Name In Blendshape Name Field Of Default Editor

5.3.6 CharacterCustomizationEditor Script

Με αυτό το script δημιούργησα ένα custom editor, το οποίο εμφανίζει μία λίστα με όλα τα blendshape του χαρακτήρα και μπορώ να επιλέξω όποιο blendshape θέλω και να δημιουργήσω ένα slider μέσα στην σκηνή, με έτοιμες όλες τις απαραίτητες ρυθμίσεις, πατώντας το κουμπί "Create slider". Όρισα αυτή την κλάση ως custom editor του CharacterCustomization script με τον ίδιο τρόπο όρισα και το άλλο editor (βλ. Εισαγωγή ενότητας 5.4.5). Μετά δήλωσα μία private μεταβλητή τύπου int που την ονόμασα "blendshapeSelectedIndex", μία private μεταβλητή "characterCustomization" και μία public μεταβλητή τύπου Canvas. Στην συνέχεια έκανα override τη μέθοδο OnInspectorGUI και μέσα της αρχικά κάλεσα την OnInspectorGUI μέθοδο της base (Editor) class για να εμφανιστεί και το default editor. Έπειτα άφησα 3 κενές σειρές μέσα στον editor και πέρασα το αντικείμενο που περιέχει την κλάση CharacterCustomization, μέσα στη μεταβλητή που έφτιαξα.

```
base.OnInspectorGUI();

//Three spaces between the base.OnInspectorGUI() fields and the overriden OnInspectorGUI() fields
EditorGUILayout.Space(); EditorGUILayout.Space(); EditorGUILayout.Space();

//target derives from Editor class and is the current object that is being inspected
characterCustomization = (CharacterCustomization)target;
```

Εικόνα 5.55: Call Base OnInspectorGUI, Create Spaces & Cast target Variable Into characterCustomization

Έπειτα ελέγχω αν το SkinnedMeshRenderer που περιέχει τα blendshapes, έχει αλλάξει με τη μέθοδο DoesTargetMatchSkmr (βλ. Ενότητα 5.3.4.6), ώστε αν έχει αλλάξει να αδειάσω το dictionary που περιέχει όλα τα blendshapes και να του περάσω τα καινούργια, από το καινούργιο μοντέλο και μετά αποθηκεύω σε ένα πίνακα τα ονόματα των blendshapes.

```
//If target has been changed then clear blendshapes database in order to update to new target with the next if statement below
if (characterCustomization.DoesTargetMatchSkmr())
    characterCustomization.ClearDatabase();

//Initialize Blendshapes and get from database
if (characterCustomization.GetNumberOfEntries() <= 0)
    characterCustomization.Initialize();

string[] blendShapeNames = characterCustomization.GetBlendShapeNames();
```

Εικόνα 5.56: If Model Changed Clear Dictionary, Parse New Values & Save Names In Array

Στην συνέχεια δημιουργώ ένα drop down menu στο οποίο περνάω ένα label, το επιλεγμένο Index και τα ονόματα των blendshapes και αποθηκεύω το Index το οποίο επιστρέφει το drop down όταν επιλέγεις κάποια από τις επιλογές του, μέσα σε μία μεταβλητή.

```
blendshapeSelectedIndex = EditorGUILayout.Popup("BlendShapeName", blendshapeSelectedIndex, blendShapeNames);
```

Εικόνα 5.57: DropDown Menu To Select Blendshape

Επίσης δημιουργώ ένα πεδίο το οποίο δέχεται ένα αντικείμενο τύπου Canvas από τη σκηνή, σε περίπτωση που θέλω να προσθέσω ένα canvas χειροκίνητα, αν έχει πολλά μέσα στη σκηνή.

```
canvas = EditorGUILayout.ObjectField("Manual Canvas Selection:", canvas, typeof(Canvas), true) as Canvas;
```

Εικόνα 5.58: Object Field For Manual Canvas Selection From Scene

Μετά δημιούργησα ένα κουμπί που το ονόμασα “Create slider”. Όταν πατήσω αυτό το κουμπί εκτελούνται οι παρακάτω εντολές:

Αν δεν επέλεξα κανένα canvas επιλέγει μόνο του ένα μέσα από την σκηνή.
Μετά δημιουργεί ένα αντίγραφο του prefab "Blendshape Slider" που είχα δημιουργήσει (βλ. Ενότητα 3.3.5) και το περνάει στην GameObject μεταβλητή που ονόμασα "sliderPrefab".

```
if (GUILayout.Button("Create Slider"))
//Instantiate Slider from root Resource folder within path Resources/"Blendshape Slider"
GameObject sliderPrefab = Instantiate(Resources.Load("Blendshape Slider", typeof(GameObject))) as GameObject;
```

Εικόνα 5.60: Load And Instantiate Prefab

```
    canvas = GameObject.FindObjectOfType<Canvas>();

    //If canvas doesn't exist, throw exception
    if (canvas == null)
    {
        throw new System.Exception("Please add a canvas into your scene!");
    }

}
```

Εικόνα 5.59: Automatically Find Canvas From Scene

Στην συνέχεια παίρνω το BlendShapeSlider component που έχει αυτό το prefab και το αποθηκεύω σε μία μεταβλητή που ονόμασα "BShapeSlider" και έπειτα αποθηκεύω στο πεδίο blendShapeName του BShapeSlider (δηλαδή το πεδίο του επιλεγμένου blendshape, το οποίο δήλωσα μέσα στην BlendShapeSlider class και εμφανίζεται στον default editor), το όνομα του blendshape πού βρίσκεται στο επιλεγμένο Index, από το drop down menu.

```
//Get preset component from prefab slider
BlendShapeSlider BShapeSlider = sliderPrefab.GetComponent<BlendShapeSlider>();
//Fill in the name of the selected Blendshape Name. Returns array with the names and you get the name from the  selected index
BShapeSlider.blendShapeName = characterCustomization.GetBlendShapeNames()[blendshapeSelectedIndex];
```

Εικόνα 5.61: GetComponent Slider & Save Dropdown Menu's Selected Blendshape In Slider's Blendshape Name Field

Μετά θέτω ως parent του BShapeSlider το canvas της σκηνής και αλλάζω την ονομασία αυτού του slider, μέσα στη σκηνή, σε "Slider " + την ονομασία του επιλεγμένου blendshape. Επίσης βάζω το όνομα του επιλεγμένου blendshape, ως κείμενο στο text component, το οποίο είναι παιδί αυτού του slider και τέλος θέτω κάποιες προεπιλεγμένες διαστάσεις για αυτό το slider.

```
//Parent slider to canvas
BShapeSlider.transform.parent = canvas.transform;
//Set slider game object name
BShapeSlider.name = "Slider " + BShapeSlider.blendShapeName;
//Change the Label text for the blendshape from the component in the children object of the slider
BShapeSlider.transform.GetComponentInChildren<Text>().text = BShapeSlider.blendShapeName;
//Set default dimensions for the slider
BShapeSlider.GetComponent<RectTransform>().sizeDelta = new Vector2(240f, 30f);
```

Εικόνα 5.62: Set Parent, Name Slider, Set Text In Child Object & Set Default Slider Dimensions

Στην συνέχεια ελέγχω αν το blendshape αυτού του slider μπορεί να πάρει αρνητικές τιμές (δηλαδή αν αυτό το blendshape υπάρχει και με το επίθεμα Min) και αν όχι, θέτω την ελάχιστη τιμή αυτού του slider σε μηδέν και το ίδιο κάνω για τις θετικές τιμές και τέλος θέτω την προεπιλεγμένη τιμή του slider σε μηδέν.

```
//Set slider's min and max value based on the blendshape
if (blendShape.negativeIndex == -1)
    slider.minValue = 0f;
else if (blendShape.positiveIndex == -1)
    slider.maxValue = 0f;
else
    throw new System.Exception("Blendshape doesn't exist!");

//Set default slider value to 0
slider.value = 0f;
```

Εικόνα 5.63: Set Slider's Min Value, Max Value & Default Value

5.4 Επιλογή Ρούχων Χαρακτήρα

Όλα τα scripts της αλλαγής ρούχων βρίσκονται στον φάκελο Assets → Custom Assets → SwitchArmor.

5.4.1 TabGroup Script

Ξεκινώντας η κλάση αποθηκεύει το component SwitchArmorEquipment που βρίσκεται στο CharacterCustomization gameobject, για να χρησιμοποιηθεί μετά για την αλλαγή ρούχων πατώντας κάποιο από τα κουμπιά που περιέχουν ένα ρούχο.

```
Unity Message | 0 references
void Start()
{
    //SwitchArmorEquipment Class Object in order to call function from there for the added functionality of
    //switching armor from the armor grid menu
    switchArmor = GameObject.Find("CharacterCustomization").GetComponent<SwitchArmorEquipment>();
}
```

Εικόνα 5.64: Μέθοδος Start

Μετά υπάρχει μία μέθοδος που ονομάζεται Register και η οποία κάνει εγγραφή όλα τα κουμπιά τύπου TabButton σε μία λίστα.

```
1 reference
public void Register(TabButton button)
{
    if(tabButtons == null) //if tabButtons list is null, create an empty one
    {
        tabButtons = new List<TabButton>();
    }

    tabButtons.Add(button); //add button to list
}
```

Εικόνα 5.65: Μέθοδος Register

Έπειτα υπάρχει μία μέθοδος OnTabEnter η οποία καλείται από την TabButton class, όταν βάζεις το ποντίκι πάνω στο κουμπί και η οποία καλεί με την σειρά της, την μέθοδο ResetTabs, η οποία βάζει σε όλα τα κουμπιά το idle sprite εκτός από το επιλεγμένο κουμπί και μετά αν το κουμπί στο οποίο έχεις βάλει πάνω το ποντίκι δεν είναι το επιλεγμένο κουμπί (δηλαδή δεν έχεις κάνει κλικ πάνω του), τότε βάζει το hover sprite.

```
public void OnTabEnter(TabButton button)
{
    ResetTabs(); //Reset all tab buttons except the selected one to the idle sprite

    //check if the tab button you hovered over is the selected one and if not it sets it to the hover sprite
    if (selectedTab == null || selectedTab != button)
    {
        button.background.sprite = tabHover;
    }
}
```

Εικόνα 5.66: Μέθοδος OnTabEnter

Έπειτα υπάρχει μία μέθοδος OnTabExit η οποία καλείται από την κλάση TabButton όταν κουνάς το ποντίκι έξω από το κουμπί και καλεί την μέθοδο ResetTabs. Στη συνέχεια υπάρχει η μέθοδος OnTabSelected, η οποία επίσης καλείται από την κλάση TabButton, όταν κάνεις όμως κλικ πάνω σε ένα κουμπί. Ξεκινώντας αυτή η μέθοδος, ελέγχει αν υπήρχε πριν κάποιο επιλεγμένο κουμπί. Αν υπήρχε τότε κάνοντας κλικ στο τωρινό κουμπί, σημαίνει ότι το προηγούμενο κουμπί δεν είναι πλέον επιλεγμένο, άρα εκτελεί το Unity event που έφτιαξα που ονομάζεται Deselect (μέσα στην κλάση TabButton) και μετά θέτει ως επιλεγμένο κουμπί το τωρινό και εκτελεί το Unity event Select (που επίσης έφτιαξα μέσα στην κλάση TabButton) και εκτελείται μόλις επιλέξεις ένα κουμπί.

```
//Function to set the selected tab button sprite to the active sprite and enable the tab panel corresponding to that tab and
This method has 1 reference(s). (Alt+3) the tab panels. Also added armor grid functionality to switch armor pieces
1 reference
public void OnTabSelected(TabButton button)
{
    //When a button is clicked, if there was already a selected tab button, it runs the Deselect() function which invokes
    //the unity event that is set in the TabButton class. This unity event shows up in the inspector just like in
    //a normal button and you can add any script to it.
    if (selectedTab != null)
    {
        selectedTab.Deselect();
    }

    selectedTab = button; //set clicked button as selected

    selectedTab.Select(); //New selected button is set above and it invokes the selected unity event
```

Εικόνα 5.67: Μέθοδος OnTabSelected

Στη συνέχεια εκτελείται η μέθοδος ResetTabs, η οποία επαναφέρει τα sprite όλων των κουμπιών (εκτός από το επιλεγμένο) στην idle κατάσταση (sprite) και μετά βάζει το Active Sprite στο επιλεγμένο κουμπί. Μετά παίρνω το Index αυτού του κουμπιού, με την εντολή GetSiblingIndex, η οποία επιστρέφει την θέση αυτού του κουμπιού μέσα στην ιεραρχία του parent object.

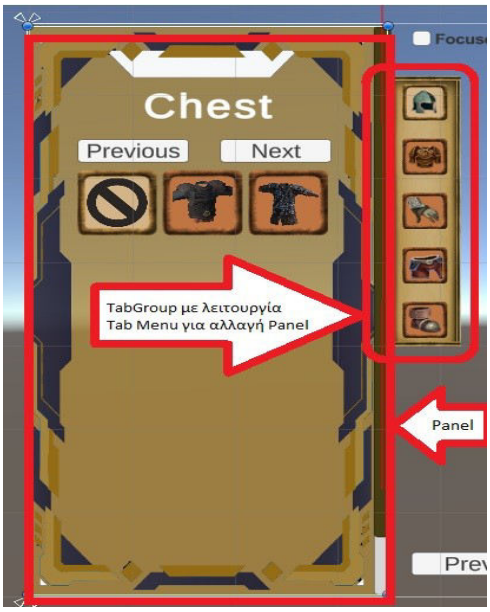
```
ResetTabs(); //Reset all tab buttons except the selected one to the idle sprite
button.background.sprite = tabActive; //Set sprite for the selected tab button to the active sprite

//Get the index of the tab button among all the objects with the same parent. Every child of the same parent object (siblings)
//in the scene, have an index based on their position in the hierarchy. This gets that index.
int index = button.transform.GetSiblingIndex();
```

Εικόνα 5.68: Εσωτερικά της Μεθόδου OnTabSelected

Στη συνέχεια υπάρχει μία if condition, η οποία ελέγχει την χρήση αυτού του TabGroup και αναλόγως εκτελεί διαφορετικές λειτουργίες. Πιο συγκεκριμένα, αυτή η κλάση χρησιμοποιείται, είτε για να δημιουργήσω ένα Grid μενού που θα έχει κουμπιά, με όλα τα ρούχα του χαρακτήρα και πατώντας κάποιο από τα κουμπιά, εμφανίζεται το ρούχο (που αντιστοιχεί σε αυτό το κουμπί), πάνω στον χαρακτήρα,

Εικόνα 5.69: Grid Menu Functionality (TabGroup)

είτε χρησιμοποιείται για να δημιουργήσω ένα Tab μενού, με το οποίο πατώντας κάποιο από τα Tabs, αλλάζει το Panel στο canvas.



Εικόνα 5.70: Tab Menu Functionality (TabGroup)

Οπότε αυτό το if condition, ελέγχει αν το όνομα του κουμπιού έχει μία συγκεκριμένη μορφή ονομασίας ώστε να ξεχωρίσει, για ποια λειτουργία θέλω το TabGroup. Στην ουσία αν το όνομα του κουμπιού έχει σαν ονομασία, κάποιο μέρος του σώματος και στο τέλος τη λέξη Armor, σημαίνει ότι προορίζεται για Grid μενού και άρα μπαίνει μέσα στο πρώτο σκέλος του if, και αναλόγως αν το όνομα του κουμπιού είναι πχ “Head Amor” ή “Chest Armor” εκτελεί την αντίστοιχη μέθοδο για αλλαγή του ρούχου πάνω στο χαρακτήρα, παίρνοντας ως παράμετρο το Index του κουμπιού.

```
//Added functionality for use with armor grid menu instead of tab menu!
#region Armor Grid Functionality
//checks if the clicked tab button's name is any of the below names and if so, it knows to treat it as an armor grid menu button.
//Tab buttons objects in the scene need to follow this name convention for armor grid functionality to work.
if (button.name == "Head Armor" || button.name == "Chest Armor" || button.name == "Hands Armor"
    || button.name == "Legs Armor" || button.name == "Feet Armor")
{
    //Based on the button type (head, chest etc), calls the corresponding function to switch armor and passes the armor grid
    //tab button's sibling index
    switch (button.name)
    {
        case "Head Armor":
            switchArmor.SwitchHeadArmor(index);
            break;
        case "Chest Armor":
            switchArmor.SwitchChestArmor(index);
            break;
        case "Hands Armor":
            switchArmor.SwitchHandsArmor(index);
            break;
        case "Legs Armor":
            switchArmor.SwitchLegsArmor(index);
            break;
        case "Feet Armor":
            switchArmor.SwitchFeetArmor(index);
            break;
    }
}
#endregion
```

Εικόνα 5.71: Μέθοδος OnTabSelected (Armor Gird Functionality)

Αν από την άλλη, τα κουμπιά δεν έχουν κάποια από τα ονόματα της μορφής “Head Amor”, “Chest Armor” κτλ, τότε σημαίνει ότι αυτό το TabGroup έχει λειτουργία Tab μενού και άρα εκτελεί τις εντολές που υπάρχουν στο else. Σε αυτή την περίπτωση με τη χρήση ενός loop περνάει από τη λίστα όλων των Panel/σελίδων και θέτει ως active μόνο το panel που έχει ίδιο index με το τωρινό κουμπί (βλ. Εικόνα 5.68).

```
#region Tab Menu Functionality
else // tab menu functionality
{
    //loops through all tab panels and only enables the tab panel corresponding to the clicked tab button sibling index
    for (int i = 0; i < pagesList.Count; i++)
    {
        if (i == index)
        {
            pagesList[i].SetActive(true);
        }
        else
        {
            pagesList[i].SetActive(false);
        }
    }
}
#endregion
```

Εικόνα 5.72: Μέθοδος OnTabSelected (Tab Menu Functionality)

Τέλος υπάρχει η μέθοδος ResetTabs που ανέφερα και παραπάνω, η οποία περνάει από όλα τα TabButtons τα οποία έκανα εγγραφή στη λίστα με τη μέθοδο Register και ελέγχει αν το κουμπί που κοιτάει, είναι το επιλεγμένο. Αν είναι το επιλεγμένο, τότε απλά το προσπερνάει και αν δεν είναι το επιλεγμένο, τότε του βάζει το idle sprite. Έτσι θέτει όλα τα κουμπιά, εκτός του επιλεγμένου, στην idle κατάσταση.

```
3 references
public void ResetTabs()
{
    //Loops through all tab buttons from the list of a specific tab group instance
    foreach(TabButton button in tabButtons)
    {
        //if selected is not empty and selected tab button equals the current tab button item from the loop
        //then continue to the next loop item without executing the rest of the code inside the loop
        if (selectedTab != null && selectedTab == button)
        {
            continue;
        }

        button.background.sprite = tabIdle; //sets every tab button except the selected one to the idle sprite
    }
}
```

Εικόνα 5.73: Μέθοδος ResetTabs

5.4.2 TabButton Script

Αυτή η κλάση χρησιμοποιείται για να δημιουργήσω κάποια δικά μου custom κουμπιά, τα οποία ονόμασα TabButtons. Στην πραγματικότητα αυτά τα κουμπιά είναι Panel GameObjects, δηλαδή gameobjects τα οποία περιέχουν απλώς το image component και τους πρόσθεσα εγώ αυτό το script, το οποίο δημιουργεί custom button λειτουργία. Στην ουσία αυτά τα gameobject, δεν έχουν το button component του Unity, αλλά έχουν ως component αυτό το script, το οποίο τα κάνει να λειτουργούν σαν

κουμπιά, ώστε να έχω περισσότερη ευελιξία για τη χρήση αυτών των κουμπιών. Επίσης όλα αυτά τα κουμπιά, περνιούνται μέσα σε ένα TabGroup (βλ. Ενότητα 5.4.1), το οποίο τα ομαδοποιεί και χειρίζεται τις λειτουργίες τους. Η υλοποίηση τους έγινε ως εξής:

Αρχικά όρισα το image component ως dependency αυτής της κλάσης, δηλαδή το αντικείμενο που περιέχει αυτό το script, πρέπει οπωσδήποτε να έχει το image component αλλιώς δημιουργεί μόνο του ένα, και μετά εκτός από την MonoBehaviour class την οποία κάνω inherit, κάνω επίσης και implement τα interface (IPointerClickHandler, IPointerEnterHandler, IPointerExitHandler), τα οποία χρησιμοποιούνται για να υλοποιήσω τις μεθόδους, οι οποίες θα επιτρέπουν την εκτέλεση κάποιων event, όταν κάνω κλικ πάνω στα κουμπιά ή όταν βάζω το ποντίκι πάνω στο κουμπί ή όταν το ποντίκι είναι ήδη πάνω σε ένα κουμπί και το βγάζω εκτός του κουμπιού.

```
[RequireComponent(typeof(Image))] //requires image component and if it doesn't exist it creates one

//Implements IPointerClickHandler, IPointerEnterHandler, IPointerExitHandler interfaces in order to use
Unity Script (90 asset references) | 8 references
public class TabButton : MonoBehaviour, IPointerClickHandler, IPointerEnterHandler, IPointerExitHandler
```

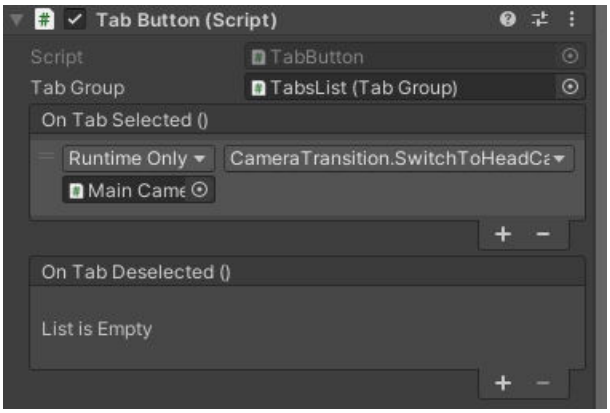
Εικόνα 5.74: Dependency & Interfaces

Στην συνέχεια δημιουργώ μία μεταβλητή TabGroup, στην οποία θα περαστούν όλα τα TabButton και μία public μεταβλητή image την οποία ονόμασα background και η οποία αποθηκεύει το image component του εκάστοτε κουμπιού. Επίσης έκρυψα αυτή τη μεταβλητή από τον Inspector, επειδή δεν θέλω να εμφανίζεται, αλλά παρόλα αυτά θέλω να είναι public για να μπορεί να χρησιμοποιηθεί στην κλάση TabGroup.

```
[HideInInspector] public Image background;
```

Εικόνα 5.75: HideInIspector Public Image

Έπειτα δημιούργησα δύο μεταβλητές τύπου UnityEvent, οι οποίες θα χρησιμοποιηθούν για να εκτελούν κάποια events, όταν επιλέγεις κάποιο κουμπί ή όταν το αποεπιλέγεις. Αυτά τα events δηλαδή, εμφανίζονται στον inspector ακριβώς όπως εμφανίζεται το OnClick event του button component και μπορείς μέσα τους να προσθέσεις ένα αντικείμενο που έχει κάποιο script component και να εκτελέσεις όποια μέθοδο θέλεις. Έτσι αυτά τα κουμπιά πέρα από τη στάνταρ λειτουργία που τους έχω δώσει, έχουν τη δυνατότητα να τους προσθέσω ξεχωριστά (χωρίς να πειράξω τον κώδικα του TabGroup ή του TabButton class), επιπλέον λειτουργία, προσθέτωντας άλλα scripts με το OnClick event όπως έχει και το κανονικό button component του Unity και επίσης υπάρχει και το deselect event, στο οποίο μπορώ επίσης να περάσω κάποιο άλλο script όταν το κουμπί αποεπιλέγεται.



Εικόνα 5.76: OnTabSelected (OnClick) &
OnTabDeselected UnityEvents On TabButtons

Στην συνέχεια μέσα στην start μέθοδο του Unity, περνάω το image component του κουμπιού και κάνω εγγραφή αυτό το κουμπιού μέσα στο tabGroup (βλ. Ενότητα 5.4.1, Εικόνα 5.65).

```
void Start()
{
    background = GetComponent<Image>();
    tabGroup.Register(this); //Register t
}
```

Εικόνα 5.77: Get Image Component & Register
TabButton

Έπειτα ορίζω τις μεθόδους για τα Mouse Events, καλώντας τις αντίστοιχες μεθόδους (OnTabSelected, OnTabEnter, OnTabExit), που υλοποιώ στην TabGroup class (βλ. Ενότητα 5.4.1).

```
0 references
public void OnPointerClick(PointerEventData eventData)
{
    tabGroup.OnTabSelected(this); //execute function from TabGroup class using this tab button
}
```

Εικόνα 5.78: OnPointerClick Event

Τέλος έφτιαξα δύο μεθόδους την Select και την Deselect, οι οποίες εκτελούν τα script που έχω περάσει μέσα στα Callback Unity Events που δημιούργησα (βλ. Εικόνα 5.76). Αυτές οι μέθοδοι καλούνται μέσα από την TabGroup class.

```
public void Select()
{
    if (onTabSelected != null)
    {
        onTabSelected.Invoke(); //Runs all registered scripts inside this unity event
    }
}

1 reference
public void Deselect()
{
    if (onTabDeselected != null)
    {
        onTabDeselected.Invoke(); //Runs all registered scripts inside this unity event
    }
}
```

Εικόνα 5.79: Execute Registered Callback Events

5.4.3 CameraTransition Script

Αυτή η κλάση χρησιμοποιείται για να αλλάξει θέση η κάμερα της σκηνής όταν χρησιμοποιείς τις καρτέλες του μενού αλλαγής ρούχων. Δηλαδή για παράδειγμα, όταν πατάς στην καρτέλα που περιέχει τα παντελόνια, η κάμερα μετακινείται για να βλέπεις το παντελόνι πάνω στο χαρακτήρα από κοντά κ.ο.κ. Επίσης χρησιμοποιείται για στις σκηνές επιλογής στοιχείων χαρακτήρα, αλλά μόνο για να αλλάξει η κάμερα σε full body view.

5.4.3.1 Μεταβλητές Κλάσης

```
public Transform startMarker; //S

//Markers to set the different ca
public Transform fullBodyMarker;
public Transform headMarker;
public Transform chestMarker;
public Transform handsMarker;
public Transform legsMarker;
public Transform feetMarker;

private Transform endMarker;// En
```

Εικόνα 5.80: Transform Μεταβλητές Για Τις Θέσεις Της Κάμερας

Αρχικά δηλώνω κάποιες μεταβλητές τύπου Transform, στις οποίες περνάω κάποια Empty GameObjects από την σκηνή και τα οποία χρησιμοποιούνται για να πάρω τη θέση τους μέσα στη σκηνή και να μετακινήσω εκεί την κάμερα του παιχνιδιού. Αυτές οι μεταβλητές είναι η startMarker στην οποία περνάω την ίδια την κάμερα του παιχνιδιού και είναι το σημείο αφετηρίας της κάμερας, και στις υπόλοιπες 6 Transformers μεταβλητές περνάω αυτά τα Empty GameObjects τα οποία έχω βάλει στις θέσεις που θέλω (5 από αυτά δείχνουν το χαρακτήρα από κοντά σε διαφορετικά σημεία του σώματος και η μία δείχνει ολόκληρο τον χαρακτήρα). Επίσης υπάρχει και μία ακόμα private Transform μεταβλητή στην οποία μεταφέρω ένα από τα προαναφερθέντα Empty GameObject (τα

οποία παίρνω από τις προηγούμενες μεταβλητές), για να θέσω την θέση του ως τον τωρινό προορισμό. Τέλος υπάρχει μία Enum μεταβλητή, την οποία ονόμασα CameraViews και έχει 5 τιμές για κάθε ένα από τα μέρη του σώματος που χώρισα στον χαρακτήρα και χρησιμοποιείται για να αποθηκεύσω ποια ήταν η τελευταία κάμερα που χρησιμοποιήθηκε.

```
enum CameraViews
{
    HeadView,
    ChestView,
    HandsView,
    LegsView,
    FeetView
}

CameraViews camera;
```

Εικόνα 5.81: CameraViews Enum

5.4.3.2 Μέθοδοι Start & Update

Ξεκινώντας, μέσα στη μέθοδο Start του Unity, αποθηκεύω σε μία μεταβλητή τον τωρινό χρόνο ως αφετηρία (τον χρόνο που πέρασε από το ξεκίνημα του παιχνιδιού) και αποθηκεύω στη μεταβλητή endMarker, το default σημείο προορισμού για όταν φορτώνει η σκηνή. Επίσης αποθηκεύω την απόσταση μεταξύ του σημείου αφετηρίας και του σημείου προορισμού.

```
void Start() //Switch to fullbody cam view by default when loading scene
{
    // Keep a note of the time the movement started.
    startTime = Time.time;

    endMarker = fullBodyMarker; //Set endmarker to the fullbodymarker position fo

    // Calculate the journey length.
    journeyLength = Vector3.Distance(startMarker.position, endMarker.position);
}
```

Εικόνα 5.82: Get Current Time, Get Destination & Calculate Distance To Travel

Στην συνέχεια μέσα στη μέθοδο update του Unity, η οποία καλείται σε κάθε frame, υπολογίζω αρχικά την απόσταση που έχει καλυφθεί από την κάμερα, πολλαπλασιάζοντας την ταχύτητα κίνησης της κάμερας με τον χρόνο που έχει περάσει (αφαιρώ τον αρχικό χρόνο που είχα αποθηκεύσει από τον τωρινό για να βρω πόσος χρόνος πέρασε), για να βρω πόση απόσταση έχει διανύσει.

```
float distanceCovered = (Time.time - startTime) * speed;
```

Εικόνα 5.83: Calculate Distance Covered

Έπειτα υπολογίζω το ποσοστό του ταξιδιού που έχει καλυφθεί, σε σχέση με την συνολική απόσταση που πρέπει να καλυφθεί, δηλαδή διαιρώ την απόσταση που έχει καλυφθεί, την οποία υπολόγισα παραπάνω (βλ. Εικόνα 5.83), με τη συνολική απόσταση του ταξιδιού (βλ. Εικόνα 5.82).

```
float percentageOfJourneyCovered = distanceCovered / journeyLength;
```
Εικόνα 5.84: Percentage Of Total Distance Covered

Τέλος αλλάζω το position και το rotation της κάμερας, με ομαλό τρόπο ώστε να έρθει στο ίδιο position και rotation με το Empty Gameobject που έχει τεθεί ως προορισμός. Αυτό γίνεται με τη χρήση της εντολής Lerp, στην οποία θέτεις ένα σημείο αφετηρίας, ένα σημείο προορισμού και ως 3η παράμετρο, παίρνει έναν δεκαδικό αριθμό μεταξύ του 0 και του 1 και μετακινεί το αντικείμενο μεταξύ των δύο σημείων, σύμφωνα με αυτή την τιμή. Για παράδειγμα, αν αυτή η τιμή είναι 0.3 τότε μετακινεί το αντικείμενο (Main Camera) στο σημείο που βρίσκεται στο 1/3 της διαδρομής. Έτσι βάζοντας τη μεταβλητή που αποθηκεύει το ποσοστό του ταξιδιού που καλύφθηκε (ή καλύτερα, το ποσοστό του ταξιδιού που θα έπρεπε να καλυφθεί με βάση τον χρόνο που πέρασε), μετακινεί και το αντικείμενο αναλόγως.

```
transform.position = Vector3.Lerp(startMarker.position, endMarker.position, percentageOfJourneyCovered);
transform.rotation = Quaternion.Lerp(startMarker.rotation, endMarker.rotation, percentageOfJourneyCovered);
```
Εικόνα 5.85: Linearly Interpolate Between 2 Point For Position & Rotation

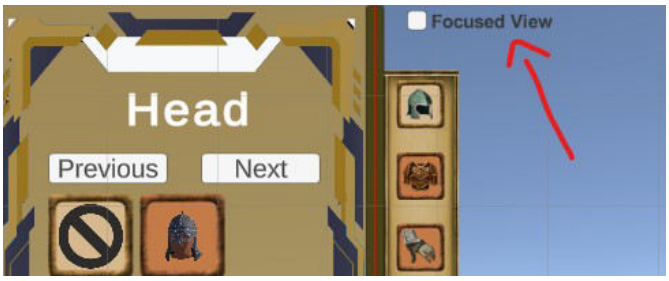
Στην πραγματικότητα δηλαδή, το Lerp από μόνο του δεν μετακινεί την κάμερα με βάση τον χρόνο που πέρασε, αλλά την μετακινεί με βάση ένα ποσοστό το οποίο υπολογίζω εγώ ξεχωριστά, αφαιρώντας τον χρόνο που είχε περάσει τη στιγμή που ξεκίνησε να μετακινείται η κάμερα (πχ στο 10ο δευτερόλεπτο), με τον τωρινό χρόνο του κάθε frame (πχ 15ο δευτερόλεπτο), έτσι ώστε να υπολογίσω τον χρόνο που πέρασε (με βάση το παράδειγμα πέρασαν 5 δευτερόλεπτα) από την εκκίνηση και μετά το πολλαπλασιάζω αυτό με την ταχύτητα (πχ 1 μέτρο το δευτερόλεπτο), ώστε να βγάλω την απόσταση που θα πρέπει να διανυθεί (5 μέτρα) , και μετα διαιρώ όλο αυτό με την συνολική απόσταση του ταξιδιού (πχ 10 μέτρα), ώστε να πάρω αυτό το ποσοστό (0.5) που θα χρησιμοποιήσω μέσα στο Lerp, για να μετακινήσω την κάμερα στην κατάλληλη θέση (άρα σε αυτό το frame η κάμερα θα μετακινηθεί στην μέση της διαδρομής, δηλαδή στο 5ο μέτρο).

5.4.3.3 Μέθοδοι SwitchTo{Name}Camera

Στην συνέχεια, δημιούργησα 6 μεθόδους (μία για κάθε θέση της κάμερας), οι οποίες είναι ίδιες μεταξύ τους εκτός από την SwitchToFullBodyCamera, η οποία έχει κάποιες μικρές διαφορές. Η μέθοδος ξεκινάει αποθηκεύοντας, με τη χρήση της Enum CameraViews, ποια είναι η κάμερα στην οποία μετακινείται, το οποίο θα χρησιμοποιηθεί μετά στη μέθοδο Toggle_Changed (βλ. Ενότητα 5.4.3.4).

```
camera = CameraViews.HeadView;
```
Εικόνα 5.86: Save Camera Choice

Μετα ελέγχω αν το σημείο αφετηρίας είναι το ίδιο με του προορισμού, άρα να μην κάνω τίποτα και επίσης ελέγχω αν η μεταβλητή focusedViewToggle είναι true, το οποίο σημαίνει ότι το Toggle (βλ. Ενότητα 4.4.4.2) στη σκηνή, είναι ενεργοποιημένο.



Εικόνα 5.87: Focused Camera View Toggle

```
if (endMarker != headMarker && focusedViewToggle)
```
Εικόνα 5.88: Check if Starting Position And Destination Are the Same & Check For Toggle

Αυτό το Toggle όταν είναι true, τότε η κάμερα δείχνει τον χαρακτήρα από κοντά σε συγκεκριμένα σημεία του σώματος, αλλιώς όποιο TabButton (του Tab Menu βλ. Ενότητα 5.4.1 Εικόνα 5.70) και να έχεις πλεγμένο, δείχνει απλά ολόκληρο τον χαρακτήρα από μακριά. Ο υπόλοιπος κώδικας είναι ο ίδιος με αυτόν στη μέθοδο start, απλά με διαφορετικό destination (βλ. Ενότητα 5.4.3.2 Εικόνα 5.82). Όσο για την μέθοδο SwitchToFullBodyCamera, η μόνη διαφορά είναι ότι δεν αποθηκεύει κάτι στη μεταβλητή camera και δεν ελέγχει για το Toggle μέσα στην if.

5.4.3.4 Μέθοδος Toggle_Changed

Αυτή η μέθοδος χρησιμοποιείται ώστε να αλλάζει από full body view στις υπόλοιπες κάμερες που δείχνουν το χαρακτήρα από κοντά και επίσης ελέγχει ποια είναι η κάμερα του TabButton που βρίσκεσε από το Tab Menu (με τη χρήση μεταβλητής camera), ώστε όταν αλλάξεις το Toggle σε full body camera και μετά το ξαναβάλεις στο Focused View να επιστρέψει στην κάμερα του συγκεκριμένου tab. Αυτή η μέθοδος εκτελείται όταν αλλάζεις τιμή στο Toggle (βλ. Ενότητα 5.4.3.3 Εικόνα 5.87) και περνάει ως παράμετρο την τιμή του, η οποία αποθηκεύεται στη μεταβλητή focusedViewToggle. Έπειτα ελέγχει αν το focusedViewToggle είναι false, τότε σε πηγαίνει στην full body camera, αλλιώς ανάλογα με το τι τιμή έχει η μεταβλητή camera, σε πηγαίνει στην αντίστοιχη κάμερα.

```
if (!focusedViewToggle) //if focused
{
    SwitchToFullBodyCamera();
}
else
{
    switch (camera)
    {
        case CameraViews.HeadView:
            SwitchToHeadCamera();
            break;
        case CameraViews.ChestView:
            SwitchToChestCamera();
            break;
        case CameraViews.HandsView:
            SwitchToHandsCamera();
            break;
        case CameraViews.LegsView:
            SwitchToLegsCamera();
            break;
        case CameraViews.FeetView:
            SwitchToFeetCamera();
            break;
```

Εικόνα 5.89: Switch Between FullBody Camera & Focused Cameras

5.4.4 SwitchArmorEquipment Script

Αυτή η κλάση χρησιμοποιείται για να αλλάξεις τα ρούχα του modular χαρακτήρα. Ξεκινώντας μέσα στην μέθοδο Awake, βρίσκει και αποθηκεύει τα SkinnedMeshRenderers, για το κάθε κομμάτι του modular χαρακτήρα που βρίσκεται στο μενού (δηλαδή του generic rig μοντέλου), ξεχωριστά.

5.4.4.1 Μέθοδοι NextHeadArmor & PreviousHeadArmor κτλ

Αυτές οι μέθοδοι χρησιμοποιούνται για τη σειριακή αλλαγή ρούχων σε κάθε κομμάτι του σώματος. Αυτές οι μέθοδοι χρησιμοποιούνται για τη σειριακή αλλαγή ρούχων σε κάθε κομμάτι του σώματος και υπάρχουν ίδιες μέθοδοι και για τα υπόλοιπα 4 κομμάτια του modular χαρακτήρα, δηλαδή για chest, hands, legs και feet. Η υλοποίηση αυτών των μεθόδων είναι ακριβώς ίδια με αυτή της σειριακής αλλαγής των αυτιών κεράτων κτλ, απλά αυτή τη φορά, για ρούχα (βλ. Ενότητα 5.2.2.2 Εικόνες 5.9 & 5.10).

```
public void NextHeadArmor()
{
    if (headsIndex < heads.Length - 1)
        headsIndex++;
    else
        headsIndex = 0;

    ChangeArmorPart(ArmorParts.HEAD_ARMOR, headsIndex);
}
```

Εικόνα 5.90: Switch To Next Armor Model

Επίσης έφτιαξα παρόμοιες μέθοδοι οι οποίες αλλάζουν τα ρούχα απευθείας σε συγκεκριμένο Index.

```
public void SwitchHeadArmor(int i)
{
    headsIndex = i; //save value in this variable for use in SaveArmor()
    ChangeArmorPart(ArmorParts.HEAD_ARMOR, i);
}
```

Εικόνα 5.91: Switch To Amor In Specified Index

5.4.4.2 Μέθοδος ChangeArmorPart

Αυτή η μέθοδος καλείται μέσα από τις μεθόδους της ενότητας 5.5.4.1 και δέχεται ως παραμέτρους μία μεταβλητή τύπου ArmorParts και ένα int που περιέχει το index για τον πίνακα. Στην συνέχεια χρησιμοποιεί την μεταβλητή armor για να καταλάβει σε ποιο κομμάτι του σώματος αναφερόμαστε και μετά αλλάζει τα Mesh και τα Materials του χαρακτήρα, με αυτά που βρίσκονται σε αυτό το Index του αντίστοιχου πίνακα. Επίσης καλεί την μέθοδο HideHairWhenWearingHelmet (βλ. Ενότητα 5.2.2.7) περνώντας το id, ώστε να κρύψει τα μαλλιά αν φοράει ο χαρακτήρας κράνος.

```
void ChangeArmorPart(ArmorParts armor, int id)
{
    switch (armor)
    {
        case ArmorParts.HEAD_ARMOR:
            headSkmr.sharedMesh = heads[id].sharedMesh; //change the mesh with the
            headSkmr.sharedMaterials = heads[id].sharedMaterials; //change the mat
            GetComponent<CustomizeCharacterParts>().HideHairWhenWearingHelmet(id);
            break;
        case ArmorParts.CHEST_ARMOR:
            chestSkmr.sharedMesh = chests[id].sharedMesh;
            chestSkmr.sharedMaterials = chests[id].sharedMaterials;
            break;
        case ArmorParts.HANDS_ARMOR:
```

Εικόνα 5.92: Load Armor Mesh & Materials To Character Model & Hide Hair

5.4.4.3 Μέθοδοι SaveArmor & LoadArmor

Η μέθοδος SaveArmor αποθηκεύει τις επιλογές ρούχων του παίκτη για να τις φορτώσει στον χαρακτήρα της σκηνής του παιχνιδιού (humanoid rig μοντέλο) και εκτελείται μέσα από το SecondaryCustomizationHelper script, ακριβώς πριν φορτωθεί η σκηνή του παιχνιδιού**.** Αυτό γίνεται αποθηκεύοντας τα Index από τους πίνακες μέσα σε ένα dictionary.

Στην συνέχεια καλείται η μέθοδος LoadArmor μέσα από το SetPlayableCharacterAppearance script που είναι περασμένο ως component στο humanoid rig μοντέλο του χαρακτήρα, μέσα στη σκηνή του παιχνιδιού και φορτώνει τις αποθηκευμένες επιλογές πάνω στο χαρακτήρα. Ο κώδικας αυτής της μεθόδου είναι παρόμοιος με αυτόν της μεθόδου ChangeArmorPart (βλ. Ενότητα 5.4.4.2 Εικόνα 5.92), με τη διαφορά ότι εδώ δεν υπάρχει το switch και αντί για τη μεταβλητή id, εδώ δίνω τον Index των πινάκων, με τη χρήση του dictionary που έφτιαξα στην μέθοδο SaveArmor.

```
public void SaveArmor()
{
    savedArmor = new Dictionary<ArmorParts, int>();

    savedArmor.Add(ArmorParts.HEAD_ARMOR, headsIndex);
    savedArmor.Add(ArmorParts.CHEST_ARMOR, chestsIndex);
    savedArmor.Add(ArmorParts.HANDS_ARMOR, handsIndex);
    savedArmor.Add(ArmorParts.LEGS_ARMOR, legsIndex);
    savedArmor.Add(ArmorParts.FEET_ARMOR, feetIndex);
}
```

Εικόνα 5.93: Save Player Selections to Dictionary

Επίσης στην αρχή της μεθόδου περνάω σε κάποιες καινούργιες μεταβλητές τα κομμάτια του

```
gameSceneHeadSkmr.sharedMesh = heads[savedArmor[ArmorParts.HEAD_ARMOR]].sharedMesh;
gameSceneHeadSkmr.sharedMaterials = heads[savedArmor[ArmorParts.HEAD_ARMOR]].sharedMaterials;
GetComponent<CustomizeCharacterParts>().HideHairWhenWearingHelmet(savedArmor[ArmorParts.HEAD_ARMOR]);

gameSceneChestSkmr.sharedMesh = chests[savedArmor[ArmorParts.CHEST_ARMOR]].sharedMesh;
gameSceneChestSkmr.sharedMaterials = chests[savedArmor[ArmorParts.CHEST_ARMOR]].sharedMaterials;
```

Εικόνα 5.94: Load Player Selections to Character Model in Game Scene

humanoid rig χαρακτήρα και χρησιμοποίησα αυτά τα κομμάτια, για να φορτώσω τις επιλογές του παίχτη, πάνω στο χαρακτήρα της σκηνής παιχνιδιού.

```
GameObject gameSceneHeadObj = GameObject.Find("Male_model_head_2");
GameObject gameSceneChestObj = GameObject.Find("Male_model_chest_2");
GameObject gameSceneHandsObj = GameObject.Find("Male_model_hands_2");
GameObject gameSceneLegsObj = GameObject.Find("Male_model_legs_2");
GameObject gameSceneFeetObj = GameObject.Find("Male_model_feet_2");

SkinnedMeshRenderer gameSceneHeadSkmr = gameSceneHeadObj.GetComponent<SkinnedMeshRenderer>();
SkinnedMeshRenderer gameSceneChestSkmr = gameSceneChestObj.GetComponent<SkinnedMeshRenderer>();
SkinnedMeshRenderer gameSceneHandsSkmr = gameSceneHandsObj.GetComponent<SkinnedMeshRenderer>();
SkinnedMeshRenderer gameSceneLegsSkmr = gameSceneLegsObj.GetComponent<SkinnedMeshRenderer>();
SkinnedMeshRenderer gameSceneFeetSkmr = gameSceneFeetObj.GetComponent<SkinnedMeshRenderer>();
```

Εικόνα 5.95: Get SkinnedMeshRenderers From Humanoid Rig Model In Game Scene

5.5 Κώδικας AI Του Εχθρού

Αυτά τα scripts ελέγχουν το AI του εχθρού και βρίσκονται όλα μέσα στον φάκελο Assets → Custom Assets → EnemyAI. Οι κλάσεις που κάνουν inherit την Κλαση StateMachineBehaviour, έχουν κάποιες override μεθόδους από τις οποίες εγώ χρησιμοποίησα τις 3. Αυτές είναι η OnstateEnter, η OnStateUpdate και η OnstateExit. Η OnstateEnter εκτελείται μόλις ξεκινήσει το συγκεκριμένο State (μέσα στο οποίο έβαλα το ανάλογο script ως component) από το State Machine, η OnStateExit εκτελείται όταν βγεί από το συγκεκριμένο State και η OnStateUpdate εκτελείται σε κάθε frame μεταξύ των OnStateEnter και OnStateExit.

5.5.1 IdleState Script (StateMachineBehaviour)

Αυτή η κλάση διαχειρίζεται την λειτουργία του Enemy AI, όταν ο εχθρός βρίσκεται στο Idle State, δηλαδή όταν είναι ακίνητος και περιμένει. Η κλάση έχει μία Transform μεταβλητή, η οποία αποθηκεύει την θέση του παίχτη μέσα στην σκηνή, ένα timer που χρησιμοποιείται για την εναλλαγή μεταξύ των states και serialized float μεταβλητή που ονόμασα "chaseRange", με την οποία ορίζω την απόσταση στην οποία ο εχθρός ανιχνεύει τον παίχτη.

Στην συνέχεια, μέσα στην OnStateEnter μέθοδο, μηδενίζω το timer και παίρνω την θέση του παίχτη με την χρήση ενός tag που έφτιαξα και πρόσθεσα στο μοντέλο του χαρακτήρα του παίχτη.

```
timer = 0;
playerPosition = GameObject.FindGameObjectWithTag("Player").transform;
```

Εικόνα 5.96: Reset Timer & Get Player Position

Έπειτα μέσα στην μέθοδο OnStateUpdate προσθέτω σε κάθε frame το DeltaTime, το οποίο επιστρέφει χρόνο που πέρασε μεταξύ κάθε frame και έτσι παίρνω τον πραγματικό χρόνο που πέρασε, ανεξάρτητα απο το frame rate του παιχνιδιού. Οπότε αν πέρασαν 5 δευτερόλεπτα από την στιγμή που ξεκίνησε αυτό το state, τότε θέτω την παράμετρο "isPatrolling" (την οποία πρόσθεσα μέσα στην καρτέλα parameters στο animator window του εχθρού), ώστε ο εχθρός να βγεί από το Idle State και να μπεί στο Walk State, το οποίο είναι το state που τριγυρνάει ο εχθρός σε διάφορα σημεία στην σκηνή.

```
timer += Time.deltaTime;
if (timer > 5) // if 5 seconds pass
    //Set bool parameter i created inside the ani
    //to transition from idle state to walking/pa
    animator.SetBool("isPatrolling", true);
```

Εικόνα 5.97: Start Patrolling After 5 Sec

Τέλος υπολογίζω την απόσταση μεταξύ του παίχτη και του εχθρού και αν η απόσταση είναι μικρότερη της απόστασης, που ο εχθρός ανιχνεύει τον παίχτη, τότε θέτω την παράμετρο "isChasing" του animator, το οποίο σημαίνει οτι ο εχθρός ανίχνευσε τον παίκτη και αρχίζει να τον κυνηγάει, αλλάζοντας από το Walk State στο Run State.

```
//Get distance between player and enemy
float distance = Vector3.Distance(playerPosition.position, animator.transform.position);
if (distance < chaseRange) //if enemy within range from the player
    //Set bool parameter i created inside the animator to true for use in controller's sta
    //to transition from idle or walking state to run state and from run state to walking
    animator.SetBool("isChasing", true);
```

Εικόνα 5.98: Calculate Distance & Start Chasing

5.5.2 WalkingState Script (StateMachineBehaviour)

Αυτή η κλάση διαχειρίζεται την λειτουργία του Enemy AI, όταν ο εχθρός βρίσκεται στο Walk State, δηλαδή όταν τριγυρνάει στα διάφορα waypoints που έβαλα μέσα στην σκηνή (patrolling).

Αρχικά στην μέθοδο OnStateEnter, κάνω reset το timer, δίνω μια τυχαία τιμή από 5 έως 10 δευτερόλεπτα για να σταματήσει να κινείται ο εχθρός και παίρνω το NavMesh του εχθρού.

```
timer = 0;
navAgent = animator.GetComponent<NavMeshAgent>();
```

Εικόνα 5.99: Reset Timer & Get NavMesh

Μετά βρίσκω και αποθηκεύω το αντικείμενο που περιέχει όλα τα waypoints που μπορεί να μετακινηθεί ο εχθρός, με τη χρήση του WaypointsTag που δημιούργησα. Μετά βρίσκω τον παίκτη με τη χρήση Player tag και παίρνω τη θέση του και έπειτα ορίζω την ταχύτητα του εχθρού όταν κάνει patrolling.

```
GameObject waypointsObject = GameObject.FindGameObjectWithTag("WaypointsTag");

playerPosition = GameObject.FindGameObjectWithTag("Player").transform; //Get pl
navAgent.speed = enemySpeed; //Set the enemy's speed from inside the navmeshage
```

Εικόνα 5.100: Get Waypoints With Tag, Get Player Position & Set Enemy Speed On Patrol

Τέλος με ένα foreach, αποθηκεύω τις θέσεις των waypoints μέσα σε μία λίστα και τέλος θέτω τον προορισμό σε ένα τυχαίο waypoint από την λίστα.

```
foreach (Transform t in waypointsObject.transform)
    waypoints.Add(t);

//Set a random destination among all the waypoints, for the enemy to move to
navAgent.SetDestination(waypoints[Random.Range(0, waypoints.Count)].position);
```

Εικόνα 5.101: Add Waypoints To List & Set Random Waypoint As Destination

Στην συνέχεια στη μέθοδο OnStateUpdate, ξεκινάω ελέγχοντας αν το remainingDistance του NavAgent του εχθρού, είναι μικρότερο ή ίσο με το stoppingDistance. Η εντολή remainingDistance δίνει την εναπομείναντα απόσταση μεταξύ της τωρινής τοποθεσίας του εχθρού και του προορισμού του. Το stoppingDistance δίνει την απόσταση από τον προορισμό στην οποία πρέπει να σταματήσει να κινείται ο εχθρός (στην περίπτωση μου, αυτό σημαίνει απλά οτι ο εχθρός έχει φτάσει στον προορισμό του και άρα διαλέγει έναν καινούργιο προορισμό, επειδή το έφτιαξα έτσι ώστε να σταματάει αφού περάσουν κάποια δευτερόλεπτα, και όχι όταν φτάσει στον προορισμό του. Επίσης όρισα το stoppingDistance ως 0 μέσα από το NavMeshAgent component στον inspector). Οπότε αν το remainingDistance είναι μικρότερο από το stoppingDistance σημαίνει ότι εχθρός βρίσκεται ήδη στον προορισμό, και έτσι δίνω έναν καινούργιο τυχαίο προορισμό για να πάει. Για παράδειγμα, αν η απόσταση από τον προορισμό, στην οποία πρέπει να σταματήσει να κινείται (stoppingDistance), είναι 5 μέτρα και η εναπομείναντα απόσταση που έχει μείνει να διανύσει ο εχθρός για να φτάσει στον προορισμό του (remaining distance) είναι μικρότερη από 5, σημαίνει ότι ο εχθρός βρίσκεται ήδη σε απόσταση 5 μέτρων από τον προορισμό του, οπότε θεωτείται ότι έφτασε στον τωρινό προορισμό και συνεχίζει προς τον επόμενο.

```
if (navAgent.remainingDistance <= navAgent.stoppingDistance)
    navAgent.SetDestination(waypoints[Random.Range(0, waypoints.Count)].position);
```
Εικόνα 5.102: RemainingDistance & StoppingDistance

Αυτό μπορεί να σημαίνει είτε ότι διάλεξε σαν προορισμό του το ίδιο σημείο δύο φορές στη σειρά και άρα βρίσκεται ήδη εκεί, είτε ότι ο επόμενος προορισμός είναι πολύ κοντά στον προηγούμενο (δηλαδή εντός του stopping distance), είτε ότι ο εχθρός έφτασε στον προορισμό του σε λιγότερο από 5-10 δευτερόλεπτα (το οποίο είναι ο χρόνος για τον οποίο κινείται ο εχθρός και δίνεται τυχαία μεταξύ των 5 με 10 δευτερολέπτων) και άρα διάλεξε απευθείας επόμενο προορισμό χωρίς να αλλάξει πρώτα στο Idle State. Δηλαδή, στην ουσία ο εχθρός δεν μπαίνει στο Idle state όταν φτάνει στον προορισμό του, αλλά μπαίνει στο Idle state, αφού περάσει το χρονικό διάστημα που έχω ορίσει (5-10 sec) και άρα θα μπορούσε να ταξιδέψει σε πάνω από έναν προορισμό (waypoints) μέσα στο χρονικό περιθώριο. Επίσης αυτό σημαίνει ότι εχθρός μπορεί να σταματήσει σε οποιαδήποτε τοποθεσία μεταξύ των waypoints, αν έχει περάσει ο χρόνος, και όχι μόνο στις ακριβείς τοποθεσίες στις οποίες βρίσκονται τα waypoints.

Στην συνέχεια, μετράω τον χρόνο που πέρασε με το DeltaTime και αν πέρασε το χρονικό περιθώριο που έχω ορίσει, θέτω την παράμετρο “isPatrolling” σε false, ώστε να σταματήσει να τριγυρνάει και να μπει πάλι στο Idle State.

```
timer += Time.deltaTime;
if (timer > timeToStopMoving) //Stop patrolling after 5-10 secs
    animator.SetBool("isPatrolling", false);
```
Εικόνα 5.103: Stop Patrolling After 5-10 Seconds

Έπειτα υπολογίζω την απόσταση μεταξύ του παίχτη και του εχθρού και αν η απόσταση είναι μικρότερη της απόστασης που μπορεί ο εχθρός να ανιχνεύσει τον παίχτη, θέτω την παράμετρο “isChasing” σε true και έτσι μπαίνει στο Run State και αρχίζει να κυνηγάει τον παίχτη.

```
float distance = Vector3.Distance(playerPosition.position, animator.transform.position);
if (distance < chaseRange)
    animator.SetBool("isChasing", true);
```
Εικόνα 5.104: Calculate Distance & Start Chasing

Τέλος μέσα στην OnStateExit μέθοδο, θέτω ως προορισμό την τωρινή θέση στην οποία βρίσκεται ήδη ο εχθρός, ώστε να σταματήσει να κινείται.

```
navAgent.SetDestination(navAgent.transform.position);
```
Εικόνα 5.105: Stop Moving

5.5.3 ChasingState Script (StateMachineBehaviour)

Αυτή η κλάση διαχειρίζεται την λειτουργία του Enemy AI, όταν ο εχθρός βρίσκεται στο Run State, δηλαδή όταν κυνηγάει τον παίχτη.

Αρχικά μέσα στην μέθοδο OnStateEnter, παίρνω το NavMeshAgent του εχθρού και την θέση του παίχτη και ορίζω την ταχύτητα του εχθρού όταν κυνηγάει τον παίχτη.

```
navAgent = animator.GetComponent<NavMeshAgent>();
playerPosition = GameObject.FindGameObjectWithTag("Player").transform;
navAgent.speed = enemySpeed;
```

Εικόνα 5.106: Get NavMesh, Get Player Position & Set Speed

Στην συνέχεια δίνω ως προορισμό του εχθρού, την τοποθεσία του παίχτη, ώστε να κυνηγάει τον παίχτη και μετά υπολογίζω την απόσταση μεταξύ του εχθρού και του παίχτη ώστε αν ο παίχτης απομακρυνθεί πολύ από τον εχθρό, να σταματήσει ο εχθρός να τον κυνηγάσει και να γυρίσει στο Walk State.

```
float distance = Vector3.Distance(playerPosition.position, animator.transform.position);
if (distance > chaseRange)
    animator.SetBool("isChasing", false);
```

Εικόνα 5.107: If Player Too Far Away Stop Chasing

Αν από την άλλη ο εχθρός πλησιάσει πολύ τον παίχτη, τότε αλλάζει από το Run State στο Attack State, ώστε να αρχίσει να επιτίθεται τον παίχτη.

```
if (distance < attackRange)
    animator.SetBool("isAttacking", true);
```

Εικόνα 5.108: If Close Enough Start Attacking

Τέλος μέσα στην OnStateExit μέθοδο, θέτω ως προορισμό την τωρινή θέση στην οποία βρίσκεται ήδη ο εχθρός, ώστε να σταματήσει να κινείται.

5.5.4 AttackState Script (StateMachineBehaviour)

Αυτή η κλάση διαχειρίζεται την λειτουργία του Enemy AI, όταν ο εχθρός βρίσκεται στο Attack State, δηλαδή όταν έχει πλησιάσει τον παίχτη και αρχίζει να επιτίθεται.

Αρχικά μέσα στην μέθοδο OnStateEnter, παίρνω απλά την θέση του παίχτη. Στην συνέχεια μέσα στην μέθοδο OnStateUpdate, λέω στον εχθρό να κοιτάει τον παίχτη όταν επιτίθεται και υπολογίζω την απόσταση μεταξύ του εχθρού και του παίχτη, ώστε αν ο παίχτης βγεί εκτός της εμβέλειας, στην οποία φτάνει ο εχθρός για να επιτεθεί, τότε σταματάει να επιτίθεται και μπαίνει πάλι στο Run State, ώστε να αρχίσει πάλι να κυνηγάει τον παίχτη.

```
animator.transform.LookAt(playerPosition);
float distance = Vector3.Distance(playerPosition.position, animator.transform.position);

//If player moves far enough from enemy, it stops attacking
if (distance > attackRange)
    animator.SetBool("isAttacking", false);
```

Εικόνα 5.109: Look At Player When Attacking & If Too Far Away Stop Attacking

5.5.5 Enemy Script

Αυτό το script χειρίζεται το Health του εχθρού και τις λειτουργίες που γίνονται, όταν ο εχθρός τρώει damage από τον παίχτη.

Αρχικά παίρνω το HealthBar slider του εχθρού από το EnemyCanvas και θέτω την μέγιστη τιμή του slider μέσα στην Start μέθοδο, ώστε να είναι ίδια με το συνολικό health του εχθρού που όρισα μέσα στην serialized hp μεταβλητή.

Στην συνέχεια, μέσα στην Update μέθοδο, περνάω την τιμή του health του εχθρού ώστε να εμφανίζεται στο HealthBar slider. Μετά ελέγχω αν το health του εχθρού είναι μικρότερο ή ίσο του 0 (που σημαίνει ότι πέθανε) και αν η boolean που ελέγχει αν μπορεί να εκτελεστεί το slow motion effect (δηλαδή αν έχει εκτελεστεί ήδη μία φορά ώστε να μην εκτελείται επ' άπειρον) είναι true, τότε εκτελώ τις ακόλουθες λειτουργίες για να γίνει ένα slow motion effect όταν πεθάνει ο εχθρός. Μέσα σε αυτή την if ελέγχω αν ο χρόνος που πέρασε από το ξεκίνημα του slow motion είναι μικρότερος των 2 δευτερολέπτων (τον οποίο μετράω με το DeltaTime), τότε θέτω το time scale του παιχνιδιού (δηλαδη την ταχύτητα με την οποία κυλάει ο χρόνος), να κυλάει 2 φορές πιο αργά. Αν από την άλλη έχουν περάσει πάνω από 2 δευτερόλεπτα, τότε τελειώνει το slow motion και θέτω την τιμή του time scale σε 1 (δηλαδή να κυλάει κανονικά ο χρόνος), μηδενίζω το timer και θέτω την μεταβλητή canExecuteSlowMo σε false, ώστε να μην ξαναμπεί σε slowmotion. Επίσης επειδή το DeltaTime επηρεάζεται από το TimeScale, μέσα στην πρώτη if, πολλαπλασιάζω τον χρόνο που θέλω να διαρκεί το slowmotion (2 δευτερόλεπτα) με το τωρινό TimeScale, ώστε να μετράει κανονικά τα δευτερόλεπτα σε πραγματικό χρόνο. Δηλαδή όταν το TimeScale γίνεται 0.5 σημαίνει οτι και το DeltaTime κυλάει 2 φορές πιο αργά και άρα τα 2 δευτερόλεπτα του παιχνιδιού περνάνε σε 4 δευτερόλεπτα πραγματικού χρόνου. Έτσι πολλαπλασιάζοντας τα 2 δευτερόλεπτα με το timescale, τα 2 δευτερόλεπτα του DeltaTime περνάνε σε πραγματικό χρόνο.

```
if (hp <= 0 && canExecuteSlowMo)
{
    slowMoTimer += Time.deltaTime;

    if (slowMoTimer < 2 * Time.timeScale)
        Time.timeScale = 0.5f;
    else
    {
        Time.timeScale = 1f;
        slowMoTimer = 0;
        canExecuteSlowMo = false;
    }
```

Εικόνα 5.110: Slow Motion Effect

Τέλος μέσα στην μέθοδο TakeDamage, αφαιρώ από το health του εχθρού, το DamageValue το οποίο δίνεται από το όπλο του παίχτη και αν το health του εχθρού φτάσει το 0 τότε ενεργοποιώ το die trigger

που δημιούργησα στο state machine του εχθρού, ώστε να εκτελεστεί το Death State και μετά απενεργοποιώ το collider του εχθρού για να μην μπορεί ο παίχτης να συνεχίσει να τον επιτίθεται αφού πεθάνει. Αν από την άλλη το health του εχθρού είναι μεγαλύτερο του 0, τότε ενεργοποιώ το damage trigger, ώστε να εκτελεστεί το TakeDamage State του εχθρού κάθε φορά που του επιτίθεται ο παίχτης.

```
hp -= damageValue;//Decrease enemy health based
if (hp <= 0)
{
    //set the trigger parameter in state machine
    animator.SetTrigger("die");
    //Disable enemy's collider so that take dama
    GetComponent<Collider>().enabled = false;
}
else
{
    //set the trigger parameter in state machine
    //Also no need to add transition condition f
    //with just an empty transition, so that the
    animator.SetTrigger("damage");
}
```

Εικόνα 5.111: TakeDamage Method

5.5.6 FaceHPSliderAtPlayer Script

Αυτό το script περιστρέφει απλά το EnemyCanvas, ώστε να κοιτάει πάντα στην κάμερα του παίχτη, για να μπορεί ο παίχτης να βλέπει σωστά το HealthBar του εχθρού από όποια μεριά και αν τον κοιτάει.

Αυτό γίνεται με την εντολή LookAt, την οποία έβαλα μέσα στην LateUpdate μέθοδο του Unity, η οποία σε κάθε frame μετά την Update μέθοδο, ώστε να περιστρέφει το EnemyCanvas, αφού πρώτα γυρίσεις την κάμερα του παίχτη με την χρήση του ποντικού.

5.6 Collider Όπλου

Η λειτουργία με την οποία το όπλο μπορεί να κάνει damage τον εχθρό βρίσκεται μέσα σε αυτά τα 3 scripts. Το Collider Script είναι ένα script που έφτιαξα εγώ και περιέχει το κύριο κομμάτι αυτού τοθ κώδικα. Τα άλλα 2 scripts υπήρχαν ήδη από το ThirdPersonController και απλά τους πρόσθεσα κάποια επιπλέον πράγματα για να μπορεί να λειτουργήσει το όπλο.

5.6.1 StarterAssetsInputs Script (Μόνο Ένα Μέρος Του Script)

Σε αυτό το script εκτελούνται κάποιες μέθοδοι, οι οποίες εκτελούνται όταν ο παίχτης πατήσει κάποιο από τα κουμπία για τον χειρισμό του χαρακτήρα, τα οποία κουμπιά ορίστηκαν μέσα στο “Starter Assets (Input Action Asset)” (βλ. Ενότητα 4.3.9) και μετά αυτές οι μέθοδοι αποθηκεύουν τιμές σε μεταβλητές, ώστε να ξέρει το Unity ποίο action εκτέλεσαι ο παίχτης.

Μέσα σε αυτό το script, πρόσθεσα μία bool μεταβλητή που την ονόμασα “attack”, η οποία γίνεται true όταν ο παίχτης επιτεθεί. Μετά έφτιαξα την μέθοδο OnAttack, η οποία εκτελείται όταν

πατήσει ο παίχτης το αριστερό κλικ στο ποντίκι (βλ. Ενότητα 4.3.9). Αυτή η μέθοδος καλεί την μέθοδο AttackInput και της περνάει ως παράμετρο το value του κουμπιού, δηλαδή το true και μετά η μέθοδος AttackInput αποθηκεύει αυτή την τιμή μέσα στην μεταβλητή attack.

Επίσης έφτιαξα μία μέθοδο την οποία ονόμασα FinishAttack, η οποία καλείται μέσω ενός Event που ενσωμάτωσα μέσα σε συγκεκριμένο frame του SwordSlash animation (βλ. Ενότητα 4.3.8) και όταν παίξει το SwordSlash animation και φτάσει στο συγκεκριμένο frame εκτελείται αυτή η μέθοδος. Αυτή η μέθοδος θέτει την τιμή της "attack" bool σε false όταν τελειώσει η κίνηση του σπαθιού. Αυτό έγινε ώστε όταν ο παίχτης πατήσει το αριστερό κλικ να γίνει η "attack" true και μετά μέσα στην μέθοδο Attack του ThirdPersonController script (βλ. Ενότητα 5.6.2) να γίνει true η παράμετρος "isAttacking" του Player State Machine και να μπει ο παίχτης στο SwordSlash State και στην συνέχεια όταν ολοκληρωθεί η κίνηση του σπαθιού, να γίνει η "attack" false και με την σειρά της να γίνει reset σε false η παράμετρος "isAttacking", ώστε να μην ξαναμπει στο SwordSlash State αυτόματα χωρίς να ξαναπατήσεις το αριστερό κλικ.

5.6.2 ThirdPersonController Script (Μόνο Ένα Μέρος Του Script)

Αυτό το script περιέχει διάφορες λειτουργίες για το ThirdPersonController. Εδώ έφτιαξα μία μέθοδο που την ονόμασα Attack και η οποία καλείται μέσα από την Update μέθοδο. Η μέθοδος Attack, ελέγχει αν το input attack είναι true (δηλαδή αν ο παίχτης πάτησε το αριστερό κλικ για να επιτεθεί) και αν ο παίχτης δεν τρέχει (επειδή δεν θέλω να επιτίθεται όταν τρέχει). Αν αυτή η συνθήκη είναι true τότε κάνω true την "isAttacking" παράμετρο που έφτιαξα μέσα στο Player State Machine, ώστε να μπει ο παίχτης στο SwordSlash State και να αρχίσει να επιτίθεται. Αν η συνθήκη είναι false, τότε κάνω την παράμετρο false για να μην εκτελείται το SwordSlash πολλές φορές με ένα κλικ που κάνει ο παίχτης.

5.6.3 WeaponCollider Script

Αρχικά παίρνω το μοντέλο του χαρακτήρα από την σκηνή του παιχνιδιού με την χρήση του Player tag και μετά παίρνω το StarterAssetsInputs του player, το οποίο περιέχει τις λειτουργίες που γίνονται με τα κουμπιά με τα οποία χειρίζεται ο παίχτης τον χαρακτήρα, όπως πχ το jump, move, sprint κτλ.

Στην συνέχεια έφτιαξα την OnTriggerEnter μέθοδο, η οποία ενεργοποιείται όταν το collider του όπλου του παίχτη, ακουμπίσει κάποιο άλλο collider. Αυτό για να λειτουργήσει, ενεργοποίησα την επιλογή "IsTrigger" στο collider του όπλου του παίχτη (βλ. Ενότητα 4.5.5.3), το οποίο του δίνει λειτουργία trigger και επίσης το όπλο δεν έχει φυσική υπόσταση, δηλαδή περνάει από μέσα από όλα τα άλλα colliders, χωρίς να τρακέρνει.

Μετά η μέθοδος OnTriggerEnter περνάει ως παράμετρο το collider του αντικειμένου, το οποίο ακούμπισε το collider του όπλου. Σε όλα gameobject στην σκηνή, τα οποία θεωρούνται εχθροί και θέλω δηλαδή να μπορώ να τους κάνω damage, τους έχω προσθέσει το "Enemy" tag που

δημιούργησα (βλ. Ενότητα 4.4.5.2), ώστε να αναγνωρίζονται ως εχθροί. Οπότε μέσα στην μέθοδο ελέγχω αν το collider με το οποίο ακούμπισε το όπλο, έχει αυτό το tag, ώστε να μπορέσω να του κάνω damage και επίσης ελέγχω αν το input attack (δηλαδή αν πάτησε ο παίχτης το αριστερό κλικ για να επιτεθεί (βλ. Ενότητα 5.6.1), ώστε να μην κάνει damage στον εχθρό κατά λάθος, αν ακουμπίσουν τα colliders όταν ο παίχτης κρατάει απλά το σπαθί, χωρίς να επιτεθεί). Οπότε αν ο παίχτης επιτίθεται και το collider του όπλου ακούμπισε ένα collider, το οποίο έχει το "Enemy" tag, τότε καλώ την μέθοδο TakeDamage από το Enemy script περνώντας το damageValue του όπλου, ώστε να κάνω damage στον εχθρό. Τέλος κάνω και από εδώ το input attack false, ώστε να μην φάει πάνω από μία φορά damage ο εχθρός με μία κίνηση του όπλου, αν τα colliders ακουμπήσουν κατά λάθος πάνω από μία φορά με μία κίνηση.

```
private void OnTriggerEnter(Collider other)
{
    //checks if the other collider's tag is "enemy" (which i
    //enemies and be able to take damage by the player) and
    //player won't damage the enemy if the weapon touches th
    if (other.tag == "Enemy" && _input.attack)
    {
        other.GetComponent<Enemy>().TakeDamage(damageValue);

        //make it false so it doesn't take damage more than
        //the enemy multiple times with one swing
        //NOTE: This is also set to false with an event i se
        //at a specific frame (when player finishes his swin
        //StarterAssetsInputs class
        _input.attack = false;
```

Εικόνα 5.112: OnTriggerEnter Event

5.7 Εναλλαγή Σκηνών Παιχνιδιού (ChangeScenes Script)

Από αυτό το script φορτώνω όλες τις σκηνές του παιχνιδιού. Μέσα στο script, έχω φτιάξει μία μέθοδο για την κάθε σκηνή και απλά φορτώνω τις σκηνές με την εντολή LoadScene. Επίσης μόνο στις μεθόδους για το άνοιγμα των σκηνών επιλογής στοιχείων χαρακτήρα (οι οποίες είναι δεύτερες σε σειρά εμφάνισης στο μενού), έχω ορίσει ότι με την αλλαγή των σκηνών θέλω να μην καταστρέφονται το μοντέλο του generic rig χαρακτήρα και το CharacterCustomization gameobject, επειδή θα τα χρειαστώ και στις επόμενες σκηνές (με την χρήση του DontDestroyOnLoad). Στις επόμενες σκηνές μετά τις σκηνές επιλογής στοιχείων χαρακτήρα, δεν ξανά πρόσθεσα το DontDestroyOnLoad, επειδή από την στιγμή που το έβαλα στις προηγούμενες σκηνές, παραμένει σε όλες τις επόμενες σκηνές από εκεί και πέρα. Επίσης αυτός είναι ένας τρόπος για να μεταφέρεις μεταβλητές από την μία σκηνή στην άλλη, δηλαδή ότι μεταβλητές έχουν αλλάξει τιμές σε scripts τα οποία ήταν component του gameobject το οποίο έστειλα στην επόμενη σκηνή, κρατάνε όλες τις τιμές τους.

Στην συνέχεια βρίσκω και αποθηκεύω το CharacterCustomization gameobject από την σκηνή, το οποίο χρησιμοποιώ μέσα στην μέθοδο LoadGameScene, ώστε να αποθηκεύσω όλες τις επιλογές

```
public void LoadHumanCustomizationBodyPartsScene()
{
    DontDestroyOnLoad(GameObject.Find("CharacterCustomization"));
    DontDestroyOnLoad(GameObject.Find("human_male_model_14"));
    SceneManager.LoadScene("HumanCustomizationBodyPartsScene");
}
```

Εικόνα 5.113: Dont Destroy On Load & Load Scene

του χαρακτήρα, που έγιναν από τον παίχτη, για να τις φορτώσω μετά στην σκηνή του παιχνιδιού. Έτσι καλώ την μέθοδο SaveAppearance από το CustomizeCharacterParts script για να αποθηκεύσω τα στοιχεία του χαρακτήρα, μετά καλώ την μέθοδο SaveArmor από το SwitchArmorEquipment script, για να αποθηκεύσω τις επιλογές της πανοπλίας και τέλος φορτώνω την σκηνή του παιχνιδιού. Η αποθήκευση των blendershape γίνεται απευθείας μέσα στην μέθοδο ChangeBlendshapeValue της CharacterCustomization class (βλ. Ενότητα 5.3.4.4).

```
characterCustomization.GetComponent<CustomizeCharacterParts>().SaveAppearance();
characterCustomization.GetComponent<SwitchArmorEquipment>().SaveArmor();
SceneManager.LoadScene("GameScene");
```

Εικόνα 5.114: Save Selections & Load Game Scene

5.8 Περιστροφή Του Χαρακτήρα Στο Menu Επιλογής

5.8.1 RotateCharacter Script

Αυτό το script χρησιμοποιείται για να μπορείς να περιστρέψεις τον χαρακτήρα μέσα στις σκηνές του μενού, ώστε να τον δείς από πολλές οπτικές γωνίες.

Αρχικά μέσα στην Update μέθοδο, πήρα το Canvas component, το οποίο το έκανα μέσα από την Update αντί για την Start μέθοδο, επειδή όταν το script μεταφέρεται από την μία σκηνή στην άλλη μέσω του DontDestroyOnLoad, η μεταβλητή canvas γίνεται null και μετά δεν μπορεί να ξαναεκτελεστεί η Start για να πάρει το καινούργιο canvas από την τρέχουσα σκηνή. Μετά ελέγχω αν την ώρα που πατήθηκε το αριστερό κλικ, δηλαδή στο πρώτο frame που έγινε το κλικ, το ποντίκι βρισκόταν έξω από το panel του μενού και τότε θέτω την BtnPressedOutsideMenu bool σε true. Αυτή η bool μεταβλητή, παραμένει true για όσο κρατάω το αριστερό κλικ πατημένο και όταν το αφήσω, γίνεται false. Οπότε για να μπορέσω να περιστρέψω τον χαρακτήρα πρέπει να κάνω κλικ έξω από το panel του μενού και να κρατήσω το κλικ πατημένο καθώς το κουνάω και όταν το αφήσω σταματάει να περιστρέφεται. Αν κάνω κλικ μέσα στο μενού και κρατώντας το κλικ πατημένο μετακινήσω το ποντίκι οπουδήποτε, ακόμα και έξω από το μενού, τότε ο χαρακτήρας δεν περιστρέφεται.

Επίσης τα όρια του μενού panel πάνω στον οριζόντιο άξωνα “x”, τα πήρα με την εντολή “canvas.renderingDisplaySize.x * 0.3206”. Το μενού panel πιάνει σχεδόν το 32% της οθόνης, αλλά επειδή τα pixel του άξωνα “x” της οθόνης είναι αναλογικά με την ανάλυση της οθόνης, δεν μπορούσα να πάρω μία fixed τιμή για να ορίσω ότι μέχρι αυτό το pixel θεωρείται οτι το ποντίκι είναι πάνω στο

μενού (επειδή το όριο θα μετακινούταν ανάλογα με την οθόνη σου), άρα πήρα τα συνολικά pixel από την ανάλυση της οθόνης (η εντολή renderingDisplaySize, δίνει τις διαστάσεις του canvas ανάλογα με την ανάλυση της οθόνης σου) και το πολλαπλασίασα με το 0.3206, ώστε να πω οτι μέχρι και το pixel, το οποίο βρίσκεται περίπου στο 32% της οθόνης (ως προς τον άξωνα "x"), είναι τα όρια του μενού panel, οπότε μόνο αν κάνεις κλικ από εκεί και μετά μπορείς να περιστρέψεις τον χαρακτήρα.

```
if (Input.GetMouseButtonDown(0) && Input.mousePosition.x > canvas.renderingDisplaySize.x * 0.3206)
    BtnPressedOutsideMenu = true;
else if(!Input.GetMouseButton(0)) //set bool to false only when you release the pressed button
    BtnPressedOutsideMenu = false;
```

Εικόνα 5.115: Check If Mouse Was Initially Clicked Inside Menu Panel Boundaries

Στην συνέχεια ελέγχω αν η μεταβλητή είναι true και μετά επεξεργάζομαι την μεταβλητή rotateSpeed (στην οποία δίνω μια τιμή από τον inspector), ώστε η τιμή της να είναι μέσα στα όρια που έχω ορίσει (0.1 με 1) με την χρήση της εντολής Clamp. Έπειτα υπολογίζω και αποθηκεύω στην μεταβλητή "mousePositionDelta", την διαφορά μεταξύ της τωρινής θέσης του ποντικιού και της θέσης του ποντικιού στο προηγούμενο frame (η τελευταία θέση του ποντικιού αποθηκεύεται σε κάθε frame στο τέλος της Update μεθόδου).

```
rotateSpeed = Mathf.Clamp(rotateSpeed, 0.1f, 1);

//Difference between the current mouse position and the last frame's mouse position
mousePositionDelta = Input.mousePosition - mouseLastPosition;
```

Εικόνα 5.116: Clamp Rotate Speed & Get Mouse Position Delta

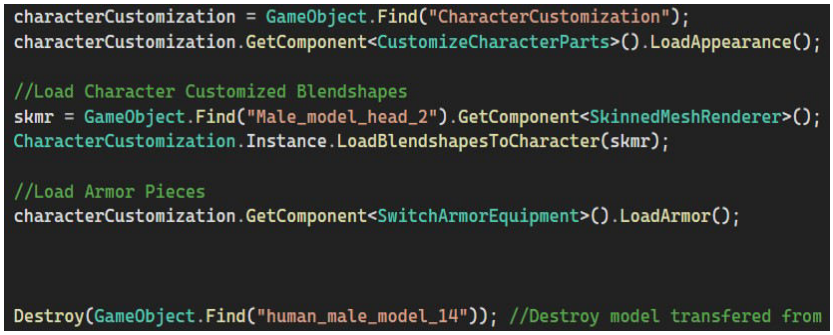
Τέλος περιστρέφω τον χαρακτήρα με την εντολή Rotate, δίνοντας ως παραμέτρους τον άξωνα γύρω από τον οποίο θα περιστρέφεται (ο οποίος είναι ο άξωνας που κοιτάει προς τα πάνω, δηλαδή ο πράσινος "y" άξωνας), τις μοίρες που θα περισταφεί και τον χώρο σε σχέση με τον οποίο θα περιστρέφεται (world space). Επίσης ως μοίρες περιστροφής, δίνω το dot product μεταξύ του mousePositionDelta και του άξωνα "x" της Main Camera, το οποίο πολλαπλασιάζω με την μεταβλητή rotateSpeed (την οποία ανέφερα παραπάνω), ώστε να ρυθμίζω την ταχύτητα περιστροφής.

```
transform.Rotate(transform.up, Vector3.Dot(mousePositionDelta, Camera.main.transform.right) * rotateSpeed, Space.World);
```

Εικόνα 5.117: Rotate Character

5.9 Φόρτωση Όλων Των Επιλογών Από Το Menu Στο Παιχνίδι

Το SetPlayableCharacterAppearance Script είναι το script το οποίο είναι περασμένο ως component πάνω στο humanoid rig μοντέλο του χαρακτήρα στην σκηνή του παιχνιδιού και μέσω αυτού, φορτώνω τις επιλογές που έκανε ο παίχτης από τις σκηνές του μενού, στην σκηνή του παιχνιδιού.



Εικόνα 5.118: Load Player Selections

Αρχικά μέσα στην μέθοδο Start, παίρνω το CharacterCustomization gameobject, ώστε να πάρω το CustomizeCharacterParts script και να καλέσω την μέθοδο LoadAppearance με την οποία φορτώνω τα στοιχεία του χαρακτήρα που επέλεξε ο παίχτης (αυτιά, κέρατα κτλ). Στην συνέχεια παίρνω το SkinnedMeshRenderer component από το Male_model_head_2 gameobject (το οποίο είναι το κεφάλι από το modular humanoid rig μοντέλο το χαρακτήρα), για να περάσω τις επιλογές των blendshapes. Η μέθοδος που περνάει τα blendshapes είναι η LoadBlendshapesToCharacter του CharacterCustomization script, την οποία καλώ απευθείας μέσω του Instance της singleton (βλ. Ενότητα 5.3.2). Έπειτα καλώ και την μέθοδο LoadArmor του SwitchArmorEquipment script, ώστε να περάσω και τις επιλεγμένες πανοπλίες και τέλος καταστρέφω το generic rig μοντέλο του χαρακτήρα (human_male_model_14) από την σκηνή του παιχνιδιού, το οποίο μεταφέρθηκε από τις σκηνές του μενού με την DontDestroyOnLoad εντολή (βλ. Ενότητα 5.7), ώστε να υπάρχει στην σκηνή του παιχνιδιού μόνο το humanoid rig μοντέλο το οποίο θα χειρίζεται ο παίχτης μέσω του ThirdPersonController.

5.10 SecondaryCustomizationHelper Script

Την χρήση αυτού του script την εξήγησα μέσα στην ενότητα 4.4.2.3, οπότε εδώ θα εξηγήσω απευθείας τον κώδικα του.

Αρχικά μέσα στην μέθοδο Awake ελέγχω αν η μεταβλητή characterCustomization είναι κενή και της περνάω το CharacterCustomization gameobject. Στην συνέχεια με την χρήση του CharacterCustomization gameobject, παίρνω τα script που είναι περασμένα μέσα σε αυτό το gameobject, όταν αυτό το gameobject μεταφερθεί στην τρέχουσα σκηνή μέσω της εντολής DontDestroyOnLoad (βλ. Ενότητα 5.7), και καλώ τις διάφορες μεθόδους για την αλλαγή των στοιχείων του χαρακτήρα, των χρωμάτων και των πανοπλιών. Αυτές οι μέθοδοι μετά θα κληθούν μέσα από τα κουμπιά στο GUI των σκηνών του μενού. Στην ουσία δημιουργώ μεθόδους οι οποίες δρουν ως μεσάζοντες για την κλήση των κανονικών μεθόδων από τα αντίστοιχα scripts, για τους λόγους που ανέφερα στην ενότητα 4.4.2.3.

Κεφάλαιο 6ο: Παράδειγμα Χρήσης Παιχνιδιού από Παίχτη

6.1 Εισαγωγή

Σε αυτό το κεφάλαιο δίνω ένα παράδειγμα χρήσης του παιχνιδιού από την μεριά του παίχτη, δηλαδή όταν ανοίγει το παιχνίδι για να παίξει και εξηγώ τους χειρισμούς του παιχνιδιού. Το κεφάλαιο είναι χωρισμένο σε ενότητες για κάθε σκηνή του παιχνιδιού που περνάει ο παίχτης.

6.2 Σκηνή Επιλογής Φυλής Χαρακτήρα

Εικόνα 6.1: Σκηνή Επιλογής Φυλής

Ανοίγωντας το παιχνίδι, ο παίχτης μπαίνει στην σκηνή της επιλογής χαρακτήρα. Εκεί υπάρχει αριστερά το μενού και δεξιά η οθόνη με τον χαρακτήρα στον οποίο παίζει ένα Idle animation. Μέσα στο μενού υπάρχουν 5 κουμπιά (Human, Elf, Demon, Beastmen, Fishmen), με τα οποία επιλέγεις την φυλή του χαρακτήρα σου. Μόλις επιλέγεις μία φυλή φορτώνονται αυτόματα κάποια τυχαία στοιχεία με βάση την φυλή που επέλεξες (αυτιά, κέρατα κτλ). Αν ξαναπατήσεις το ίδιο κουμπί μίας φυλής απλά αλλάζουν τα τυχαία στοιχεία του χαρακτήρα. Επίσης αν πατήσεις το αριστερό κλικ του ποντικιού, μέσα στην οθόνη με τον χαρακτήρα (αν πατήσεις μέσα στο panel του μενού δεν λειτουργεί) και το κρατήσεις πατημένο, μπορείς να κουνήσεις το ποντίκι για να περιστρέψεις τον χαρακτήρα ώστε να τον δεις καλύτερα. Αυτό γίνεται σε όλες τις σκηνές του μενού. Τέλος αφού επιλέξεις την φυλή που θέλεις πατάς στο κουμπί Next για να προχωρίσεις στο επόμενο μενού.

6.3 Σκηνή Επιλογής Στοιχείων Χαρακτήρα

Σε αυτήν τη σκηνή επιλέγεις τα στοιχεία του χαρακτήρα ανάλογα με την φυλή που διάλεξες. Αν διάλεξες πχ demon μπορείς να αλλάξεις διάφορα αυτιά και κέρατα (κέρατα έχει μόνο ο demon).

Μέσα στο μενού υπάρχουν στην αρχή κάποια κουμπιά που γράφουν Next και Previous με τα οποία αλλάζεις σειριακά τα μοντέλα των στοιχείων του χαρακτήρα (πχ στον demon υπάρχουν πρώτα τα αυτιά και μετά τα κέρατα) και πατώντας τα κουμπιά πας μπρος και πίσω στα μοντέλα. Μετά από κάτω υπάρχουν τα color palettes, με μερικά χρώματα για την αλλαγή του χρώματος του δέρματος ή των ματιών και πατώντας κάποιο χρώμα, εμφανίζεται το χρώμα που πάτησες πάνω στον χαρακτήρα. Όταν επιλέξεις τα στοιχεία και τα χρώματα που θέλεις πατάς στο κουμπί Next κάτω δεξιά για να περάσεις στην επόμενη σκηνή.

Εικόνα 6.2: Σκηνή Επιλογής Στοιχείων Χαρακτήρα

6.4 Σκηνή Διαμόρφωσης Προσώπου Χαρακτήρα

Εικόνα 6.3: Σκηνή Διαμόρφωσης Προσώπου Χαρακτήρα

Σε αυτή την σκηνή είναι το μενού για την διαμόρφωση του προσώπου του χαρακτήρα. Εδώ υπάρχουν σύνολο 18 sliders για διάφορα σημεία του προσώπου. Κάθε slider έχει το value του στην μέση που είναι το default μοντέλο του χαρακτήρα. Για να δεις όλες τις επιλογές πρέπει να μετακινήσεις το scrollbar του μενού είτε επιλέγοντας την scrollbar με το ποντίκι και μετακινώντας την πάνω κάτω ή με την χρήση του mouse wheel. Κατα κύριο λόγο, μετακινώντας τα slider προς τα αριστερά αλλάζουν τα αντίστοιχα σημεία του προσώπου προς τα μέσα (ρουφιούνται) και όταν τα μετακινείς προς τα δεξιά αλλάζουν τα αντίστοιχα σημεία του προσώπου προς τα έξω (φουσκώνουν). Αν πχ επιλέξεις το slider για τα μάγουλα και το μετακινήσεις προς τα αριστερά τα μάγουλα ρουφιούνται προς τα μέσα ενώ αν το μετακινήσεις προς τα δεξιά, τα μάγουλα φουσκώνουν. Αφού διαμορφώσεις το πρόσωπο όπως το θέλεις, μπορείς να πατήσεις το κουμπί Next κάτω δεξιά στην οθόνη του χαρακτήρα για να πας στο επόμενο μενού ή αν θες μπορείς να γυρίσεις στο προηγούμενο μενού, με το κουμπί Previous που βρίσκεται κάτω αριστερά στην οθόνη με τον χαρακτήρα.

6.5 Σκηνή Επιλογής Ρούχων Χαρακτήρα

Σε αυτό το μενού γίνεται η επιλογή της πανοπλία του χαρακτήρα. Αρχικά μπαίνοντας σε αυτό το μενού η κάμερα μετακινείται ομαλά προς τα έξω για να δεις ολόκληρο τον χαρακτήρα. Δεξιά από το μενού υπάρχει ένα tab μενού με 5 κουμπιά. Κάθε ένα από αυτά τα κουμπιά αλλάζει το μενού στο αντίστοιχο σημείο του σώματος. Τα κουμπιά αυτά (τα οποία έχουν και το ανάλογο εικονίδιο) είναι το κουμπί που σε πάει στο μενού με τα helmets, το κουμπί με τα Chests, το κουμπί με τα Gloves, το κουμπί με τα Pants και το κουμπί με τα Boots. Από αυτά τα μενού το προεπιλεγμένο μενού είναι το Helmets.

Στην συνέχεια κάθε μενού έχει μέσα του 2 κουμπιά Next και Previous, με τα οποία μπορείς να αλλάξεις σειριακά τα διάφορα μοντέλα των πανοπλιών, και ένα Grid μενού που έχει όλες τις πανοπλίες μαζί με το εικονίδιο της αντίστοιχης πανοπλίας, ώστε να μπορείς να τα δεις καλύτερα και να επιλέξεις. Μέσα στο Grid μενού, ως πρώτη default επιλογή είναι το κομμάτι του σώματος χωρίς κάποια πανοπλία και μπορείς να την αλλάξεις πατώντας τα κουμπιά του Grid μενού.

Εικόνα 6.4: Σκηνή Επιλογής Ρούχων (Full Body View)

Επιπλέον, υπάρχει ένα checkbox πάνω στην οθόνη (δεξιά από το μενού), το οποίο γράφει Focused View και είναι απενεργοποιημένο. Όταν αυτό το checkbox είναι απενεργοποιημένο και αλλάζεις τα μενού με τα tabs, η κάμερα παραμένει να δείχνει ολόκληρο τον χαρακτήρα. Αν όμως το επιλεξεις, τότε με κάθε tab που αλλάζεις, μετακινείται και η κάμερα στο αντίστοιχο σημείο από κοντά ώστε να δεις το σημείο που ντύνεις καλύτερα. Όταν ολοκληρώσεις τις επιλογές σου πατάς το κουμπί Play για να μπείς μέσα στο παιχνίδι ή αν θέλεις μπορείς να πατήσεις το κουμπί Previous για να επιστρέψεις στην προηγούμενη σκηνή.

6.6 Σκηνή Παιχνιδιού

Εικόνα 6.6: Game Scene With Created Character

Μπαίνοντας στην σκηνή του παιχνιδιού φορτώνεται ο χαρακτήρας με τις επιλογές που επέλεξε ο παίκτης και με ένα σπαθί στο χέρι. Για να κουνήσεις τον χαρακτήρα χρησιμοποιούνται τα πλήκτρα "WASD" και για να πηδήξεις το πλήκτρο “space”. Επίσης μπορείς να τρέξεις πιο γρήγορα κρατώντας πατημένο το “shift” και για να επιτεθείς με το σπαθί πατάς το αριστερό κλικ στο ποντίκι. Μέσα στην σκηνή υπάρχουνε μερικοί εχθροί οι οποίοι τριγυρνάνε σε διάφορα σημεία του χάρτη. Όταν πλησιάσεις κοντά σε έναν εχθρό ο εχθρός σε ανιχνεύει και αρχίζει να τρέχει και να σε κυνηγάει. Μόλις φτάσει κοντά σου αρχίζει να σου επιτίθεται και άμα απομακρυνθείς αρκετά από τον εχθρό σε χάνει και αρχίζει πάλι απλά να τριγυρνάει μες στη σκηνή. Αν πατήσεις το αριστερό κλικ για να επιτεθείς και το σπαθί σου ακουμπήσεις τον εχθρό, ο εχθρός τρώει damage και μειώνεται το Health του μέχρι να φτάσει στο μηδέν όπου πεθαίνει. Επίσης όταν πεθαίνει ο εχθρός δημιουργείται για δύο δευτερόλεπτα ένα slow motion effect.

Κεφάλαιο 7ο: Προτάσεις βελτίωσης

Σε αυτό το κεφάλαιο αναφέρω μερικές προτάσεις βελτίωσης του παιχνιδιού, τις οποίες δεν πρόλαβα να προσθέσω και οι οποίες είναι οι εξής:

- Η δημιουργία ενός αρχικού μενού το οποίο θα έχει την επιλογή Create character ή την επιλογή που λέει play στην οποία θα μπαίνεις στο παιχνίδι, επιλέγοντας έναν χαρακτήρα που έχεις ήδη φτιάξει. Η οθόνη επιλογής υπάρχων χαρακτήρα, θα μπορούσε να έχει όλους τους χαρακτήρες που έχεις δημιουργήσει στην σειρά για να μπορείς να τους βλέπεις και να επιλέγεις.
- Η δημιουργία μηχανισμού για να μπορείς να βάζεις το όπλο στην θήκη του και να το βγάζεις, καθώς και να αλλάζεις σε διαφορετικά όπλα όπως π.χ. Δόρυ, τόξο και τα λοιπά.
- Η προσθήκη ηχητικών εφέ στον εχθρό π.χ. όταν τρώει damage ή όταν πεθαίνει.
- Η προσθήκη οπτικών εφέ όπως π.χ. αίμα.

- Ο εμπλουτισμός του περιβάλλοντος των σκηνών και η χρήση διαφορετικού περιβάλλοντος για κάθε διαφορετική φυλή π.χ. τα Ξωτικά να έχουν background ένα μέρος που να θυμίζει πόλη ξωτικών, οι beastmen να έχουν ως background ένα δάσος, οι δαίμονες να έρθουν ως background ένα σκοτεινό μέρος και τα λοιπά.
- Η βελτίωση του Enemy AI, ώστε να ανιχνεύει όταν μπεις στο οπτικό πεδίο, καθώς και με τον ήχο και όταν σε χάσει από το οπτικό του πεδίο να έρχεται στην τελευταία τοποθεσία που σε είδε και να σε ψάχνει εκεί.
- Η προσθήκη face animations στον παίχτη με την χρήση των blendshapes.
- Η προσθήκη physics στα μαλλιά και τις ουρές, ώστε να κουνιούνται.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Athletic male Blueprint
[1] https://drawingdatabase.com/athletic-male/ Το reference image για την δημιουργία του μοντέλου του χαρακτήρα.

Human Ear Topology
[2] https://gr.pinterest.com/pin/397864948330111902/ Το reference image για την δημιουργία του μοντέλου του αυτιού.

Jorah Mormont inspired armor
[3]https://sketchfab.com/3d-models/jorah-mormont-inspired-armor-442b204b4b3141ba9e6d034e80cf6ae1 Το μοντέλο της πανοπλίας.

Viking lamellar armor

[4]https://sketchfab.com/3d-models/viking-lamellar-armor-297c2536534c4045aa0be028d10d39d4 Το μοντέλο της πανοπλίας.

Sword Slash Animation
[5] https://www.mixamo.com/#/?page=1&query=sword+slash&type=Motion%2CMotionPack Το Sword Slash animation του παίχτη.

Starter Assets - Third Person Character Controller
[6] https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-196526 Τα starter assets του Unity με το third person controller.

Fantasy Monster – Skeleton
[7] https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy-monster-skeleton-35635 Το asset του enemy skeleton.

Melee Weapons Pack [Swords – Axes]
[8] https://assetstore.unity.com/packages/3d/props/weapons/melee-weapons-pack-swords-axes-121237 Το asset του σπαθιού του παίχτη.

Sun Temple
[9] https://assetstore.unity.com/packages/3d/environments/sun-temple-115417#description Τα asset για την διακόσμηση των σκηνών του μενού.

Free UI Pack
[10] https://assetstore.unity.com/packages/2d/gui/icons/free-ui-pack-170878 Το asset για τα frames των GUI buttons.

RPG Inventory Icons
[11] https://assetstore.unity.com/packages/2d/gui/icons/rpg-inventory-icons-56687 Το asset για τα εικονίδια των GUI buttons.

3D Toon Dreadlock Ponytail Hairstyle model
[12] https://www.turbosquid.com/3d-models/3d-toon-dreadlock-ponytail-hairstyle-model-1828694 Το μοντέλο για τα μαλλιά.

FREE Sample - Low Poly Hairstyles 3D model
[13] https://www.turbosquid.com/3d-models/hair-low-poly-hairstyles-free-sample-3d-model-1836795 Το μοντέλο για τα μαλλιά του παίχτη.

Gradient twitch panels pack

[14]https://www.freepik.com/free-vector/gradient-twitch-panels-pack_21075385.htm#query=game%20panel&position=22&from_view=keyword To asset για background image των μενού.