Παράρτημα Σπάρτης

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

“Ανάπτυξη Ψυχαγωγικής Εφαρμογής σε περιβάλλον Android”

Πτυχιακή Εργασία

Χρήστος Νιάσκος

Επιβλέπων Καθηγητής - Αριστομένης Θανόπουλος

Σπάρτη 2017

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

"Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάση επιστημονικής παράφρασης.

Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων.

Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δε μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας."

Όνομα και Επώνυμο Συγγραφέα (Με Κεφαλαία): ΧΡΗΣΤΟΣ ΝΙΑΣΚΟΣ

Υπογραφή (Ολογράφως, χωρίς μονογραφή): 

Ημερομηνία (Ημέρα – Μήνας – Έτος): 29/5/2017

ΠΕΡΙΛΗΨΗ

Το θέμα που πραγματεύεται η παρούσα πτυχιακή εργασία είναι η κατασκευή ενός τρισδιάστατου παιχνιδιού με τη μηχανή παιχνιδιών Unreal Engine 4 για συσκευές Android. Η δημιουργία του παιχνιδιού πραγματοποιήθηκε κυρίως με την χρήση UML διαγραμμάτων γνωστά ως blueprints και τη συγγραφή κώδικα σε C++. Η επιλογή της συγκεκριμένης μηχανής παιχνιδιών έγινε διότι θεωρείται η καλύτερη στο είδος της και γιατί έπειτα από δοκιμή άλλων μηχανών παιχνιδιών, η Unreal Engine αποδείχτηκε η καλύτερη επιλογή. Το παιχνίδι έχει κλίμα μεσαιωνικής εποχής και ο παίχτης είναι ένας πολεμιστής που πρέπει να νικήσει όλους τους αντιπάλους του για ολοκληρώσει την αποστολή του. Σκοπός μου είναι το παιχνίδι να κατασκευαστεί με τέτοιο τρόπο ώστε να είναι εύκολα επεκτάσιμο για μελλοντικές αναβαθμίσεις. Για τη δημιουργία του παιχνιδιού αλλά και τα επιμέρους αντικείμενα θα χρησιμοποιηθούν τα προγράμματα Visual Studio 2017, Adobe photoshop CC 2015, blender και 3ds Max, που επιλέγουν οι μεγαλύτερες εταιρείες στον κόσμο.

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1
Μηχανές Παιχνιδιών

1.1 Εισαγωγή

Για τη δημιουργία του παιχνιδιού που πραγματεύεται η παρούσα πτυχιακή εργασία χρησιμοποιήθηκε η μηχανή παιχνιδιών Unreal Engine 4. Η Unreal Engine από τον Μάρτιο του 2015 παρέχει όλα τα εργαλεία, τη διεπαφή προγραμματισμού εφαρμογών (API) και το πηγαίο κώδικα της μηχανής εντελώς δωρεάν.

Στα πρώτα χρόνια της δημιουργίας παιχνιδιών, τα παιχνίδια δημιουργούνταν από το μηδέν, χωρίς τη χρήση κάποιας πλατφόρμας. Στη σύγχρονη εποχή είναι αναγκαίο, για να υπάρξει το επιθυμητό αποτέλεσμα αλλά και για εξοικονόμηση χρόνου, η χρήση μιας πλατφόρμας, όπως μια μηχανή παιχνιδιών.

Στο συγκεκριμένο κεφάλαιο θα γίνει μια περιγραφή της μηχανής παιχνιδιών, καθώς και μια σύντομη περιγραφή των πιο διάσημων μηχανών παιχνιδιών για android. Επίσης, θα εξετάσουμε τη δομή και τα βασικά χαρακτηριστικά της μηχανής παιχνιδιών που χρησιμοποιήθηκε.

1.2 Στόχος της Εργασίας

Στόχος της πτυχιακής εργασίας είναι η εκμάθηση μιας επαγγελματικής μηχανής παιχνιδιών και η ανάπτυξη-υλοποίηση σε περιβάλλον android ενός τρισδιάστατου παιχνιδιού δράσης που ανήκει στις κατηγορίες “Τρίτου προσώπου” (First-Person) και “δράσης περιπέτειας” (Action-Adventure).

Η κατηγορία “τρίτου προσώπου” πραγματεύεται μάχες με τη χρήση κάποιου όπλου. Στα παιχνίδια τρίτου προσώπου η κάμερα βρίσκεται πίσω από τους χαρακτήρες, δίνοντας την αίσθηση στο χρήστη ότι είναι ο χαρακτήρας του παιχνιδιού.

Η κατηγορία “δράσης περιπέτειας” πραγματεύεται παιχνίδια με προβλήματα κατάστασης του παίχτη προς λύση, με αρκετή δράση και περιπέτεια όπου απαιτούνται καλά αντανακλαστικά. Επίσης τα παιχνίδια αυτά έχουν γρήγορο ρυθμό και περιλαμβάνουν τόσο σωματικές όσο και εννοιολογικές προκλήσεις.

1.3 Μηχανές παιχνιδιών

Οι περισσότερες εταιρίες που κατασκευάζουν παιχνίδια δημιουργούν ένα λογισμικό, το οποίο τους επιτρέπει την ευκολότερη σχεδίαση και υλοποίηση παιχνιδιών. Το λογισμικό αυτό χρησιμοποιείται για την ανάπτυξη όλων των παιχνιδιών της εταιρίας, μέχρι να θεωρηθεί παλιό, εξαιτίας της εξέλιξης της τεχνολογίας και να δώσει τη θέση του σε μια νέα έκδοση.

Αυτό το λογισμικό που δημιουργούν οι μεγάλες εταιρίες είναι μια μηχανή παιχνιδιών. Μια μηχανή παιχνιδιών κάνει εφικτή τη διασύνδεση που χρειάζεται ο προγραμματιστής για να επικοινωνήσει με το υλικό (hardware). Πλέον, στις διασημότερες μηχανές παιχνιδιών μπορούν να κατασκευαστούν παιχνίδια για όλα τα λειτουργικά που χρησιμοποιούνται από κονσόλες, smartphones, υπολογιστές, κλπ.

Επίσης η μηχανή περιλαμβάνει συστήματα και τεχνικές για τη διαχείριση διαδικασιών που περιλαμβάνουν τη τεχνητή νοημοσύνη, τα γραφικά, τον ήχο, τη διαχείριση της εισόδου από το χρήστη, τη φυσική, τη συγγραφή κώδικα κ.α.

Το σύστημα τεχνητής νοημοσύνης(A.I) είναι υπεύθυνο για την συμπεριφορά που παρουσιάζουν οι χαρακτήρες-παίκτες που δεν ελέγχονται από τον παίκτη. Χωρίς το σύστημα τεχνητής νοημοσύνης, το αποτέλεσμα των κινήσεων των αντιπάλων θα ήταν ανύπαρκτο ή αφύσικο.

Το σύστημα των γραφικών είναι για την απεικόνιση των γραφικών του παιχνιδιού στην οθόνη και αναλαμβάνει τους υπολογισμούς, για τους οποίους ο παίκτης σπάνια αναρωτιέται. Περιέχει τις βιβλιοθήκες και τις συναρτήσεις που καλούνται από την μηχανή παιχνιδιών για την αναπαράσταση όλων των οντοτήτων.

Η μηχανή φυσικής, εισάγει τους νόμους της φυσικής, δημιουργώντας ρεαλισμό για τον χρήστη. Για την εξομοίωση χρησιμοποιείται η βαρύτητα, η ανίχνευση συγκρούσεων, αλλά και άλλα στοιχεία για να αποδοθεί το καλύτερο δυνατό αποτέλεσμα. Ο προγραμματιστής μπορεί να αυξήσει ή να μειώσει τον βαθμό ρεαλισμού, ώστε να επιτύχει το επιθυμητό αποτέλεσμα σε σχέση με το είδος του παιχνιδιού που δημιουργεί. Όλες οι αλληλεπιδράσεις μεταξύ των οντοτήτων σε ένα παιχνίδι, η κίνηση των χαρακτήρων, η δυναμική των υγρών επεξεργάζονται από αυτό το σύστημα.

Η μηχανή ήχου χρησιμοποιείται για την εισαγωγή ήχων στο παιχνίδι. Ο σχεδιαστής εισάγει τους καταγεγραμμένους ήχους, δημιουργεί τα δεδομένα για το πως και πότε θα παιχτεί ο ήχος και η αναπαραγωγή γίνεται αυτόματα από τη μηχανή, σύμφωνα με τις καθορισμένες ρυθμίσεις.

Ο διαχειριστής εισόδου αναλαμβάνει την είσοδο που προέρχεται από το χρήστη. Ο χρήστης χρησιμοποιεί την οθόνη αφής και τα κουμπιά του κινητού για να παίξει το παιχνίδι. Ο διαχειριστής εισόδου δέχεται τις

εισόδους του χρήστη και τις μεταφράζει σε εντολές, τις οποίες καταλαβαίνει το παιχνίδι.

Τέλος, μια πλήρης μηχανή παιχνιδιών διαθέτει και άλλα συστήματα, για τη συγγραφή κώδικα (scripting), υποδομή δικτύου κτλ. Στη συνέχεια, θα παρουσιαστούν οι διασημότερες μηχανές παιχνιδιών για android, τα βασικά στοιχεία και χαρακτηριστικά τους, καθώς και οι διαφορές που έχουν με την τελική επιλογή, την unreal engine.

1.4 Άλλες Μηχανές Παιχνιδιών

1.4.1 Unity 5

Η Unity είναι μια από τις πιο διαδεδομένες μηχανές δισδιάστατων και τρισδιάστατων παιχνιδιών και η διασημότερη για την κατασκευή παιχνιδιών σε λειτουργικό android. Παρέχει την δυνατότητα δημιουργίας εκτελέσιμου σε διάφορες πλατφόρμες (iOS, Android, Mac, Windows, Web Browsers). Παρέχει δωρεάν την έκδοση για προσωπική χρήση, η οποία δίνει την δυνατότητα δημιουργία παιχνιδιών ακόμα και για εμπορική χρήση μέχρι το όριο κερδών των εκατό χιλιάδων δολαρίων.

Μέσα από τα εργαλεία της ο χρήστης έχει άμεση πρόσβαση στις λεπτομέρειες της γεωμετρίας, στην εισαγωγή στοιχείων όπως ήχους, σωματίδια, φωτισμούς κ.α. καθώς και σε έναν συντάκτη κώδικα scripting, για την υλοποίηση της λειτουργίας του παιχνιδιού. Υποστηρίζει σύνταξη κώδικα σε JavaScript, C# και Boo. Ο χρήστης

μπορεί να γράψει δικές του συμπεριφορές σε JavaScript, C# ή Boo χρησιμοποιώντας το ενσωματωμένο εργαλείο είτε άλλο εργαλείο συγγραφής κώδικα. Έχει ίσως την καλύτερη υποστήριξη με πολλά μέλη στην κοινότητα και πολύ χρήσιμο υλικό που διατίθεται τελείως δωρεάν.

Τέλος η Unity αποτελεί μια από τις καλύτερες μηχανές για την δημιουργία δισδιάστατων και τρισδιάστατων παιχνιδιών, για την εκμετάλλευση όμως των δυνατοτήτων της απαιτείται η συγγραφή κώδικα.

1.4.2 Construct 2

Η Construct 2 είναι η διασημότερη μηχανή δημιουργίας δισδιάστατων παιχνιδιών για διάφορα λειτουργικά συστήματα συμπεριλαμβανομένων των android συσκευών. Παρέχει την δωρεάν έκδοση για δοκιμαστικό σκοπό και έχει άλλες δύο εκδόσεις με πληρωμή για προσωπική και επαγγελματική χρήση .

Επίσης δεν απαιτείται συγγραφή κώδικα καθώς προσφέρει ανάλογα γραφικά εργαλεία. Ακόμα, αποτελεί μια ολοκληρωμένη και πολύ εύκολη λύση για δημιουργία όμως μόνο απλών δισδιάστατων παιχνιδιών. Τέλος απαιτεί αγορά μια εκ των δύο εκδόσεων με πληρωμή για τη δημιουργία ολοκληρωμένων παιχνιδιών.

1.4.3 Godot

Η Godot είναι μια δωρεάν μηχανή ανοικτής πηγής δισδιάστατων και τρισδιάστατων παιχνιδιών για πολλά λειτουργικά συμπεριλαμβανομένων των android συσκευών.

Έχει παρόμοιες λειτουργίες με την Unity αλλά η συγγραφή κώδικα γίνεται με C++ ή με τη δικιά του γλώσσα προγραμματισμού που λέγεται GDscript. Τέλος αποτελεί μια καλή και ολοκληρωμένη λύση όμως απαιτεί συγγραφή κώδικα και δεν υπάρχει αρκετή υποστήριξη και εκπαιδευτικό υλικό.

1.4.4 PlayCanvas

Η PlayCanvas, είναι μια ανοιχτού κώδικα μηχανή παιχνιδιών, με την οποία δημιουργούνται απλά 3D παιχνίδια που μπορούν να τρέχουν απ' ευθείας στους browsers μέσω WebGL. Μελλοντικά θα υποστηρίζει και πιο περίπλοκα στους browser μας, χωρίς να χρειάζεται να κατεβάσουμε τίποτα στον υπολογιστή μας και με επιδόσεις αντίστοιχες με τα

εγκατεστημένα παιχνίδια. Ο Playcanvas editor λειτουργεί αποκλειστικά σε browser επιτρέποντας έτσι εύκολη και γρήγορη πρόσβαση σε πραγματικό χρόνο σε πάνω από ένα σχεδιαστή. Τέλος παρέχει μία δωρεάν έκδοση με αρκετές λειτουργίες και δύο εκδόσεις με πληρωμή.

1.5 Unreal Engine 4

Η Unreal μηχανή δημιουργήθηκε από την Epic Games το 1998 για το παιχνίδι Unreal και πρόκειται για μια από τις καλύτερες, αν όχι η καλύτερη μηχανή παιχνιδιών αυτή τη στιγμή. Η μηχανή αυτή χρησιμοποιείται από τις μεγαλύτερα εταιρίες στον κόσμο, έχει χιλιάδες μέλη στην κοινότητα που παρέχουν βοήθεια και χαρακτηριστικά που δύσκολα βρίσκει κανείς σε άλλες μηχανές παιχνιδιών. Παρέχεται τελείως δωρεάν συμπεριλαμβανομένου και του πηγαίου κώδικα μέχρι το όριο των τριών χιλιάδων δολαρίων, όπου η Epic Games ζητάει ένα 5% των κερδών. Έχουν υπάρξει 3 μεταγενέστερες γενιές της μηχανής, κάθε μια προσφέρει ακόμα περισσότερες δυνατότητες με την 4^{η} γενιά και είναι μια από τις πιο σύγχρονες και ραγδαία αναπτυσσόμενες τεχνολογίες. Η 4^{η} γενιά της Unreal κυκλοφόρησε τον Μάιο του 2012 και η δημιουργία της ξεκίνησε το 2003 από τον ιδιοκτήτη της Epic, την εταιρία στην οποία ανήκει η unreal, Tim Sweeney.

Ο πηγαίος κώδικας της μηχανής είναι σε C++, πράγμα που την κάνει πολύ γρήγορη, και επίσης υποστηρίζει τη χρήση διαγραμμάτων uml,

γνωστά ως Blueprint Scripting για προ-σχεδιαστικούς σκοπούς μέσω τον οποίο μπορεί να δημιουργηθούν παιχνίδια χωρίς τη συγγραφή κώδικα. Η παρούσα πτυχιακή έχει υλοποιηθεί κυρίως με blueprints αλλά έχει γίνει και συγγραφή κώδικα . Η συγγραφή κώδικα από τον προγραμματιστή γίνεται σε C++ ή UnrealScript. Περιέχει όλα τα υπό-συστήματα που είναι απαραίτητα και είναι αρκετά απαιτητική από το σύστημα, λόγω του εντυπωσιακού αποτελέσματος.

Η εκμάθηση της μηχανής είναι δύσκολη, όμως επιλέχθηκε διότι διατίθεται δωρεάν και δίνει ατελείωτες δυνατότητες στην ανάπτυξη εφαρμογών τριών διαστάσεων. Επομένως θα εξετάσουμε τα πλεονεκτήματα και μειονεκτήματα της UE4 σαν μονάδα αλλά και συγκριτικά με άλλες μηχανές παιχνιδιών.

1.6 Πλεονεκτήματα – Μειονεκτήματα UE4

Σε αυτή τη κατηγορία εξετάζονται τα σημαντικότερα πλεονεκτήματα και μειονεκτήματα της Unreal Engine. Τα πολλά πλεονεκτήματα που προσφέρει ήταν ο κύριος λόγος επιλογής της UE4, ενώ τα μειονεκτήματα της είναι λίγα και σχετικά ασήμαντα.

Πλεονεκτήματα :

- Διατίθεται τελείως δωρεάν συμπεριλαμβανομένου και του πηγαίου κώδικα.
- Έχει εκπληκτικές δυνατότητες στο κομμάτι των γραφικών, με τις καλύτερες δυνατότητες δυναμικού φωτισμού και σκιών.
- Δεν απαιτεί συγγραφή κώδικα λόγω της καινοτομίας των διαγραμμάτων uml (Blueprint Scripting).
- Παρέχει καλή υποστήριξη και εκπαιδευτικό υλικό.
- Μεγάλη κοινότητα με πολλούς χρήστες.
- Παρέχει κάποια δωρεάν στοιχεία και αρχεία γνωστά ως game assets όπως materials, 3D μοντέλα, animations, κ.α.

- Μπορείς να δημιουργήσεις όλα τα είδη παιχνιδιών σε 2D, 2.5D ή 3D σε αντίθεση με άλλες μηχανές που υποστηρίζουν συγκεκριμένα είδη παιχνιδιών ή διαστάσεων.
- Είναι Cross-platform δηλαδή δίνει την δυνατότητα να δημιουργήσεις παιχνίδια που με μικρές αλλαγές θα παίζουν σε όλους τους υπολογιστές, κονσόλες και κινητά.
- Η κύρια γλώσσα για συγγραφή κώδικα είναι η C++, η οποία είναι παγκοσμίως μια από τις πιο διαδιδόμενες και χρησιμοποιημένες γλώσσες και έτσι δεν αναγκάζει τον προγραμματιστή να μάθει την δικιά του γλώσσα (UnrealScript) σε αντίθεση με άλλες μηχανές παιχνιδιών που η συγγραφή κώδικα γίνεται μόνο με τη δικιά τους γλώσσα.
- Έχει τη δυνατότητα του Level Streaming που σημαίνει όταν ο χαρακτήρας περάσει από ένα χώρο σε ένα άλλο και πλέον δεν υπάρχει ορατότητα, τότε η μηχανή σταματά την επεξεργασία του μη ορατού χώρου και εξοικονομείται έτσι επεξεργαστική ισχύς.

Μειονεκτήματα :

- Πολύ απαιτητικό από το σύστημα για να τρέξει καλά χωρίς να ‘κολλάει’.
- Πολύ λιγότερα δωρεάν στοιχεία και αρχεία σε σχέση με την Unity.
- Δύσκολη στην εκμάθηση.
- Ένα παιχνίδι που είναι φτιαγμένο μόνο με Blueprints μπορεί να είναι πολύ βαρύ για τους περισσότερους υπολογιστές, κονσόλες ή κινητά.

1.6 Λόγοι επιλογής της Unreal Engine 4

Υπάρχει μεγάλη πληθώρα μηχανών για την ανάπτυξη 3D παιχνιδιών. Η επιλογή του κατάλληλου εργαλείου αποτελεί σημαντικό βήμα ενός σχεδιαστή που θέλει να δημιουργήσει το δικό του παιχνίδι. Στην ενότητα 1.4 έγινε μια αναφορά στις διασημότερες αυτήν την εποχή μηχανές παιχνιδιών βέβαια υπάρχουν και πολλές ακόμη λιγότερο

γνωστές. Κανείς δεν μπορεί να πει αντικειμενικά ποια είναι η καλύτερη επιλογή, αφού αυτό εξαρτάται από πολλούς παράγοντες, καθώς και από τον ίδιο το σχεδιαστή. Η Unreal αν και δύσκολη στην εκμάθηση για κάποιον αρχάριο, υπήρχε μια εξοικείωση πράγμα που δεν συνέβη με άλλες μηχανές παιχνιδιών που δοκιμάστηκαν. Είναι ενδεικτικό πως υπάρχει μια πληθώρα παιχνιδιών, με γνωστούς τίτλους ανάμεσά τους, που έχουν δημιουργηθεί με την Unreal Engine. Επίσης τα πλεονεκτήματα που αναπτύχθηκαν στην ενότητα 1.6, επηρέασαν πολύ για την επιλογή της Unreal. Τέλος λόγω προ υπάρχουσας γνώσης της γλώσσας C++ δεν χρειαζόταν η εκμάθηση άλλης γλώσσας για τη συγγραφή κώδικα και έτσι υπήρξε και ευκολία στην χρήση των blueprints τα οποία είναι βασισμένα στη C++.

1.7 Ανάπτυξη παιχνιδιών

Κάθε ανάπτυξη ενός παιχνιδιού περνάει μέσα από πολλά στάδια πριν κυκλοφορήσει. Γενικώς υπάρχει ένα σχεδιαστικό πλάνο πριν την παραγωγή, το στάδιο της παραγωγής και το στάδιο μετά την παραγωγή. Όπως είναι φυσικό, ο περισσότερος χρόνος δαπανείται στο στάδιο της παραγωγής. Για την ανάπτυξη ενός παιχνιδιού χρειάζονται άτομα με ειδικότητα γραφίστας, 2D/3D σχεδιαστής, σχεδιαστής κινηματικών, σχεδιαστής ήχου και φυσικά προγραμματιστής.

Οι γραφίστες δημιουργούν όλα τα ορατά αντικείμενα του παιχνιδιού από το έδαφος μέχρι και κουμπιά του μενού. Κάποιοι γραφίστες ειδικεύονται σε 3D μοντέλα, ενώ άλλοι σε animation. Σκοπός τους να δημιουργήσουν όσο πιο ρεαλιστικά γραφικά και ακριβή μοντέλα. Οι εικόνες ή μοντέλα δημιουργούνται από τους γραφίστες με κάποιο λογισμικό όπως 3Ds Max, Blender, Maya, Photoshop και έπειτα προστίθενται στον Unreal Editor.

Οι σχεδιαστές κινηματικών είναι συνήθως εκπαιδευμένοι γραφίστες. Έχουν πολύ καλό μάτι και δημιουργικότητα για να δημιουργούν

σύντομες ταινίες. Χρησιμοποιούν είτε το sequencer editor ή το matinee tool για τη δημιουργία κινηματικών.

Οι σχεδιαστές ήχου έχουν καλή ακοή και είναι εκπαιδευμένοι στην μουσική. Δημιουργούν και ηχογραφούν τους ήχους του παιχνιδιού σε ειδικούς χώρους. Έπειτα εισάγουν τους ήχους στο παιχνίδι και τους τροποποιούν με το Sound Cue Editor.

Οι προγραμματιστές αποφασίζουν πιο λογισμικό θα χρησιμοποιηθεί για την δημιουργία του παιχνιδιού και επίσης τι άλλο λογισμικό χρειάζεται. Πρέπει επίσης να βεβαιώσουν ότι τα λογισμικά που θα χρησιμοποιηθούν είναι συμβατά μεταξύ τους. Έπειτα οι προγραμματιστές γράφουν κώδικα για τα αντικείμενα που δημιούργησαν οι υπόλοιποι σχεδιαστές. Μερικοί προγραμματιστές ασχολούνται με την δημιουργία εργαλείων και την ανάπτυξη καινούργιων εφαρμογών για το παιχνίδι.

1.8 Τεχνητή Νοημοσύνη

Τεχνητή νοημοσύνη (A.I) είναι η επιστήμη, που ασχολείται με τη σχεδίαση και την υλοποίηση υπολογιστικών συστημάτων, που μπορούν ουσιαστικά να αναπαράγουν την ανθρώπινη συμπεριφορά, αλλά να είναι και αυτόνομα. Η τεχνητή νοημοσύνη στα παιχνίδια δεν συγκρίνεται με την επιστήμη της τεχνητής νοημοσύνης καθώς έχει σχεδιαστεί για να λειτουργεί σε ένα καλά ελεγχόμενο εικονικό περιβάλλον. Αποτελείται κυρίως από ένα σετ από μόνιμους κανόνες που επιτρέπουν στους actors του παιχνιδιού να κάνουν τις σωστές πράξεις αναλόγως την περίσταση.

Οι χρήστες επιθυμούν μια ρεαλιστική και φανταστική εμπειρία από τα παιχνίδια. Η τεχνητή νοημοσύνη παίζει πολύ σημαντικό ρόλο στο να δημιουργήσει αυτή την εμπειρία για τους χρήστες. Ένας αντίπαλος/εχθρός του χρήστη είναι ο πιο συνηθισμένος και σημαντικός τρόπος χρήσης της τεχνητής νοημοσύνης. Επίσης τα διαφορετικά είδη τεχνητής νοημοσύνης όπως, πλοήγηση, μάχη, βοήθεια από το σύστημα, είναι ο καλύτερος τρόπος να κάνουμε τους αντιπάλους να μοιάζουν

ρεαλιστικοί. Πλέον τα μοντέρνα παιχνίδια εισάγουν ακόμα περισσότερα χαρακτηριστικά και δυνατότητες πράγμα όμως που δημιουργεί και προβλήματα καθώς οι παραπάνω δυνατότητες απαιτούν παραπάνω κανόνες τεχνητής νοημοσύνης.

1.9 Εμπειρία χρήστη

Το πιο σημαντικό κριτήριο για την επιτυχία ενός παιχνιδιού είναι η εμπειρία του χρήστη παίζοντας το παιχνίδι. Η περιήγηση του χρήστη στον εικονικό χώρο πρέπει να μοιάζει ρεαλιστική και να είναι αποδεκτή από τον χρήστη, έτσι είναι σημαντικό να εξετάσουμε τους παράγοντες που θα βοηθήσουν τον χρήστη να βιώσει την εικονική περιήγηση σαν να είναι πραγματικότητα.

- Παρουσία

Παρουσία ορίζεται ως η υποκειμενική εμπειρία του να είσαι εκεί και είναι ένα ψυχολογικό φαινόμενο που κατοικεί στις αντιλήψεις του χρήστη. Η αίσθηση της παρουσίας διευκολύνεται από την εικονική πραγματικότητα και οδηγεί τον χρήστη να νομίζει ότι κάθε τι που βιώνει είναι πραγματικότητα. Αυτό είναι σημαντικό επειδή οι άνθρωποι έχουν την τάση να θυμούνται πράγματα που βιώνουν από πρώτο χέρι πάνω από τα πράγματα που απλά διαβάζουν. Περισσότερες έρευνες έχουν δείξει ότι η παρουσία χωρίζεται σε προσωπική, περιβαλλοντική και κοινωνική. Η κοινωνική και η περιβαλλοντική παρουσία επικεντρώνονται στην αλληλεπίδραση με άλλους ανθρώπους και αντίστοιχα με το περιβάλλον, για την εικονική περιήγηση στον εικονικό χώρο, επιδίωξη είναι η προσωπική παρουσία, η οποία μπορεί να οριστεί ως η εμπειρία του χρήστη στο εσωτερικό του εικονικού κόσμου.

- Πλοήγηση

Ένας από τους καλύτερους τρόπους για έναν χρήστη να αισθανθεί την παρουσία είναι να μην χρειάζεται να ανησυχεί για ένα πολύπλοκο σύστημα πλοήγησης. Στα παιχνίδια πρώτου προσώπου για Android η πλοήγηση είναι πολύ εύκολη αφού ο χρήστης χρησιμοποιεί την οθόνη αφής και με απλές κινήσεις των χεριών του κατευθύνει την κάμερα και τον χαρακτήρα του.

- Απεικόνιση

Η ποιότητα του πως θα εμφανίζεται το περιβάλλον αποτελεί σημαντική συνεισφορά για το πως ένας χρήστης αισθάνεται την παρουσία. Από τεχνική άποψη, πρέπει η πλατφόρμα που χρησιμοποιεί ο χρήστης να είναι ικανή να τρέξει το εικονικό περιβάλλον ομαλά. Παιχνίδια φτιαγμένα με την unreal μπορούν να τρέξουν στις περισσότερες android συσκευές εφόσον υπάρχει μεγάλη γκάμα ρυθμίσεων για να εξυπηρετούνται πολλά μοντέλα συσκευών.

Κεφάλαιο 2
Εισαγωγή στην Unreal Engine 4

2.1 Εισαγωγή

Η δημιουργία ενός 3D παιχνιδιού με χρήση της Unreal αποτελείται από δύο κύριους μέρη: το κομμάτι του γραφικού σχεδιασμού και το κομμάτι του κώδικα ή/και των blueprints. Για το πρώτο μέρος, η Unreal προσφέρει μια πληθώρα εργαλείων, με τα οποία ο δημιουργός του παιχνιδιού, μπορεί να κατασκευάσει τους actors που επιθυμεί να περιλαμβάνει το παιχνίδι και να τους τοποθετήσει στο σημείο της σκηνής που επιθυμεί.

Σε αυτή την ενότητα θα εξετάσουμε τις βασικότερες έννοιες και εργαλεία της Unreal Engine τα οποία μαζί σχηματίζουν την δομή της Unreal Engine 4.

2.2 Βασικές Έννοιες της UE4

- Actor είναι ένα αντικείμενο που μπορεί να τοποθετηθεί σε ένα επίπεδο. Όλα τα αντικείμενα, τα φώτα, η κάμερες, κ.λπ. είναι actors. Οι Actors είναι γενικές κλάσεις που υποστηρίζουν 3D μετασχηματισμούς όπως περιστροφή, κλιμάκωση κ.α. Ένας actor

μπορεί να δημιουργηθεί ή να καταστραφεί μέσω κώδικα ή μέσω των blueprints.

- Ένα επίπεδο (Level), είναι μία περιοχή του παιχνιδιού καθορισμένη από τον σχεδιαστή. Τα Levels δημιουργούνται και επεξεργάζονται τοποθετώντας και μετασχηματίζοντας τις ιδιότητες των Actors. Είναι στην ουσία 3D περιβάλλοντα στα οποία τοποθετούνται αντικείμενα με γεωμετρικό τρόπο τα οποία καθορίζουν τον κόσμο που θα βιώσει ο χρήστης.
- Τα εξαρτήματα (Components), είναι κομμάτια λειτουργικότητας που μπορούν να προστεθούν σε actors. Ένα component δεν μπορεί να υπάρχει αυτόνομο όμως αν προστεθεί σε ένα actor, ο actor θα έχει πρόσβαση στην λειτουργικότητα του component.
- Τα πιόνια (Pawns) είναι υπό-κλάσεις του Actor και χρησιμεύουν ως προσωπικότητα εντός παιχνιδιού π.χ του χρήστη. Τα Pawns μπορούν να ελέγχονται είτε από τον χρήστη είτε από το παιχνίδι (NPC). Συνήθως χρησιμοποιούνται από το παιχνίδι καθώς αυτή η κλάση δεν περιέχει μεθόδους για τον χειρισμό του παίχτη από τον χρήστη.
- Ένας χαρακτήρας (Character), είναι υπό-κλάση του Pawn και χρησιμοποιείται ως ο χαρακτήρας του χρήστη. Χρησιμοποιείται κατά κύρια βάση μόνο για τον χαρακτήρα που θα ελέγχει ο χρήστης.
- Materials αποκαλούνται τα στοιχεία που εφαρμόζουμε σε ένα πλέγμα για να ελέγξουμε την οπτική παρουσίαση μιας σκηνής. Είναι στην ουσία η μπογιά ενός 3D αντικειμένου. Αποτελούνται από πολλές 2D στατικές εικόνες όπου η κάθε μια αναπαριστά μια ιδιότητα του material όπως χρώμα, μεταλλικότητα, αντανάκλαση στο φως.
- Animation είναι η προσομοίωση μιας κίνησης μέσα από μια σειρά από εικόνες ή πλαίσια.
- Static Mesh είναι ένα πολυγωνικό πλέγμα ενός 3d αντικειμένου το οποίο δημιουργείται μέσα από ένα πρόγραμμα 3d modelling και έπειτα εισάγεται στην unreal engine. Χρησιμοποιούνται για τη

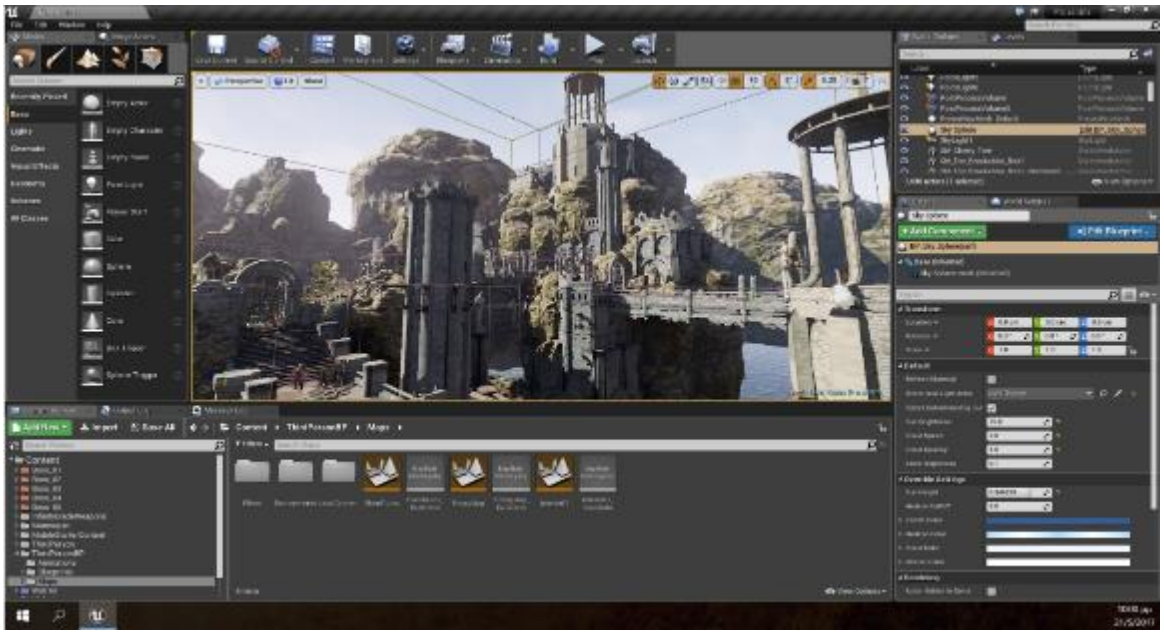
δημιουργία της γεωμετρίας εντός κάποιου επιπέδου. Αφού τοποθετηθούν σε ένα επίπεδο μπορούμε να τροποποιήσουμε το μέγεθος, τη περιστροφή και τη θέση τους.

- Τα σωματίδια (Particles) είναι τα οπτικά εφέ όπως φλόγα, χιόνι, ομίχλη κ.λπ. Χρησιμοποιούν γραφικά και 3d αντικείμενα για να προσομοιώσουν συγκεκριμένα φαινόμενα τα οποία είναι δύσκολο να προσομοιωθούν με άλλες τεχνικές λόγω αυξημένης υπολογιστικής ισχύος.

2.3 Εργαλεία του Unreal Editor

2.3.1 Level Editor

Είναι ο βασικός editor της Unreal που χρησιμοποιείται για την κατασκευή των επιπέδων του παιχνιδιού, σε αυτόν προσθέτουμε αντικείμενα, χαρακτήρες, material και γενικά ότι χρειάζεται το κάθε επίπεδο. Σε αυτόν τον editor δημιουργούνται τα επίπεδα δημιουργώντας και μετασχηματίζοντας actors. Μέσω του level editor τοποθετούμε γεωμετρία, διακοσμήσεις σε μορφή φωτισμού, οχήματα, αντικείμενα που θα χρησιμοποιήσει ο χρήστης κ.α.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο το viewport στο οποίο δημιουργούμε και προβάρουμε το παιχνίδι. Εδώ μπορούμε να τοποθετήσουμε με διάφορους τρόπους και εργαλεία όλα τα assets που έχουμε δημιουργήσει.

- Στα αριστερά το παράθυρο λειτουργιών το οποίο μας δίνει πολλές λειτουργίες ανάλογα με τις ανάγκες μας όπως τοποθέτηση actors, βάψιμο των actors, βάψιμο περιοχών, τροποποίηση της γεωμετρίας και δημιουργία φυσικού τοπίου.

- Πάνω δεξιά έχουμε το World Outliner το οποίο μας δείχνει όλους τους actors που έχουμε στο επίπεδό μας σε ιεραρχική μορφή.

- Κάτω δεξιά το παράθυρο με τις λεπτομέρειες του επιλεγμένου actor. Εδώ μπορούμε να προσθέσουμε components και blueprints στους actors.

- Κάτω έχουμε το περιεχόμενο μας που περιλαμβάνει όλα τα assets που θα χρησιμοποιήσουμε στο επίπεδο.

2.3.2 Blueprints Visual Scripting

Τα Blueprints είναι ένα πλήρες σύστημα ‘Οπτικού προγραμματισμού’ το οποίο καθιστά δυνατό την δημιουργία παιχνιδιών χωρίς συγγραφή κώδικα σε C++ ή UnrealScript. Όμως τα blueprints είναι βασισμένα στην c++ και υλοποιούν τις μεθόδους και τις λειτουργίες της γλώσσας άρα η προ υπάρχουσα γνώση της γλώσσας C++ βοηθάει αρκετά στην δημιουργία των Blueprints. Όπως και σε πολλές αντικειμενοστραφείς γλώσσες προγραμματισμού, τα blueprints χρησιμοποιούνται για να καθορίσουν αντικείμενα και κλάσεις εντός της μηχανής. Εντός της UE4 τα αντικείμενα που ορίζονται χρησιμοποιώντας blueprints αναφέρονται κοινώς ως <<Blueprints>>.

Αυτό το σύστημα είναι εξαιρετικά ευέλικτο και ισχυρό καθώς παρέχει τη δυνατότητα στους σχεδιαστές να χρησιμοποιούν σχεδόν το πλήρες φάσμα των εννοιών και εργαλείων γενικά διαθέσιμες μόνο σε έμπειρους προγραμματιστές. Στη βασική τους μορφή τα blueprints είναι uml διαγράμματα στα οποία ο σχεδιαστής συνδέει κόμβους, λειτουργίες και μεταβλητές με σύρματα δημιουργώντας έτσι πολύπλοκα στοιχεία της συμπεριφοράς του παιχνιδιού.

Τα blueprints χρησιμοποιούν γραφήματα για διάφορες λειτουργίες όπως κατασκευή αντικειμένου, επιμέρους λειτουργίες, δημιουργία γεγονότων του παιχνιδιού. Τα γραφήματα χωρίζονται σε γεγονότα (events), συναρτήσεις (functions), μακροεντολές (macros) και μεταβλητές (variables). Τα γεγονότα είναι οι κόκκινοι κόμβοι, οι συναρτήσεις οι μπλε και οι μακροεντολές οι γκρίζοι κόμβοι. Οι μεταβλητές χρωματίζονται ανάλογα με το είδος δηλαδή :

- Float – Πράσινος κόμβος
- Boolean – Κόκκινος κόμβος
- Integer – Γαλάζιος κόμβος
- String – Μωβ κόμβος
- 3D Vector – Πορτοκαλί κόμβος

Ωστόσο, ακόμη και αν τα Blueprints είναι ένα πολύ χρήσιμο εργαλείο, εξακολουθεί να είναι περιορισμένο από την άποψη του τι μπορεί να κάνει σε σ με την συγγραφή κώδικα σε c++. Συνεπώς, ο προγραμματισμός με c++ είναι πολύ πιο ευέλικτος και ευπροσάρμοστος από τα Blueprints. Η c++ είναι πολύ καλή στην υλοποίηση πολύπλοκων αλληλεπιδράσεων και μηχανισμών. Μια άλλη διαφορά μεταξύ των Blueprints και της C ++ είναι ότι τα Blueprints είναι πολύ πιο βαριά στο σύστημα που τρέχουν συγκριτικά με κώδικα C ++, αλλά αυτό είναι αισθητό, μόνο αν υπάρχουν πολλά Blueprints στο παιχνίδι.

Υπάρχουν δύο είδη blueprint, το level blueprint και το blueprint class. Κάθε επίπεδο (level) που δημιουργούμε έχει το δικό του level blueprint. Σε αυτό μπορούμε να ελέγξουμε ότι έχει να κάνει με το συγκεκριμένο

επίπεδο όπως να τροποποιούμε τις ιδιότητες των actors του επιπέδου ή να πυροδοτούμε ένα κινηματικό εφέ κ.λπ. Τα blueprint class απ' την άλλη είναι ειδικοί actors με πολλά components τα οποία περιλαμβάνουν στατικά 3D αντικείμενα, κάμερες, ήχο κ.α. και μπορούν να τοποθετηθούν σε οποιοδήποτε επίπεδο.

Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο τον γράφο στον οποίο δημιουργούμε κόμβους και αλληλουχίες. Ένας κόμβος μπορεί να είναι μια μεταβλητή, μια συνάρτηση, μια έκφραση και γενικώς οτιδήποτε στο οποίο συνδέουμε για να δημιουργήσουμε ένα blueprint. Κάθε κόμβος έχει ή εισόδους ή εξόδους ή και τα δύο. Σαν είσοδο μπορεί να πάρει κάποια τιμή ή έκφραση. Για να συνδέσουμε δύο κόμβους, συνδέουμε την καρφίτσα της εξόδου του πρώτου στην καρφίτσα της εισόδου του δεύτερου κόμβου. Τέλος, όσους λιγότερους κόμβους έχουμε τόσο πιο καλή θα είναι η απόδοση.

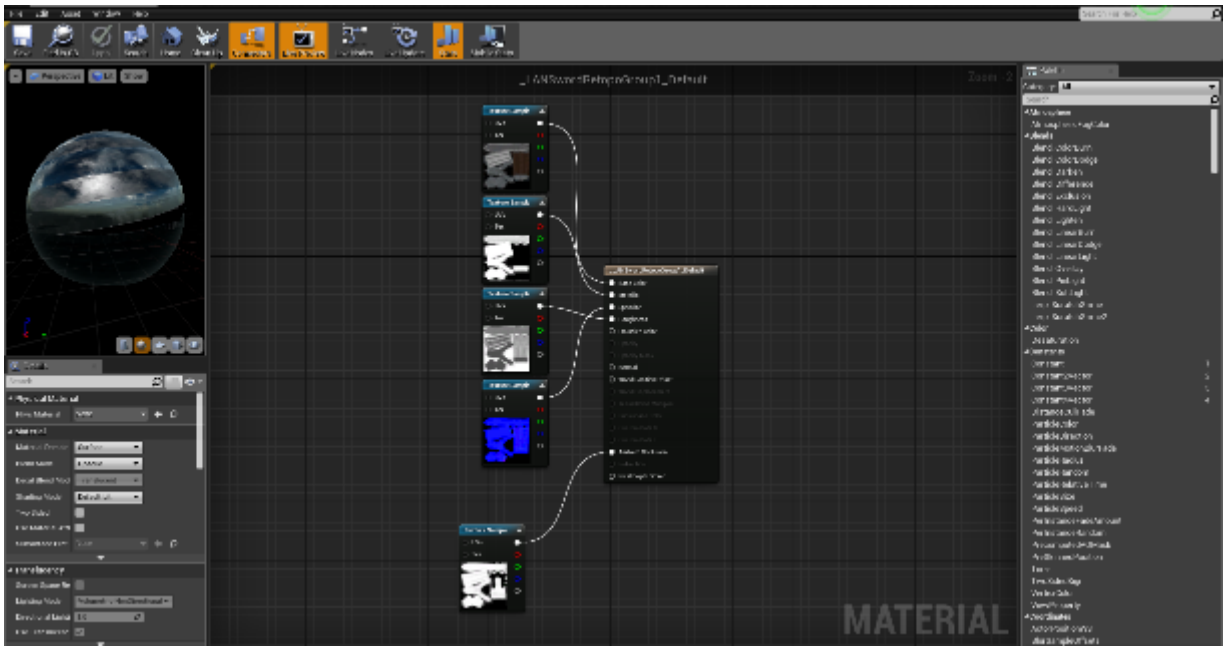
- Στα δεξιά το παράθυρο λεπτομερειών στο οποίο ρυθμίζονται οι ιδιότητες των διαφόρων κόμβων και των μεταβλητών. Διαθέτει ρυθμίσεις όπως ο τύπος της μεταβλητής, το όνομα κ.α.

- Κάτω το παράθυρο με τα αποτελέσματα του compile. Μας δίνει πληροφορίες όπως πόσο χρόνο έκανε το compile, πόσα σφάλματα εντόπισε κ.α.

- Στα αριστερά το παράθυρο My blueprint στο οποίο βλέπουμε μια λίστα με όλα τα συμβάντα και μεταβλητές που έχουμε δημιουργήσει. Επίσης εδώ μπορούμε να δημιουργήσουμε συναρτήσεις, μεταβλητές και εντολές macro.

2.3.3 Material Editor

Με τον Material Editor μπορούμε να δημιουργήσουμε ή να επεξεργαστούμε τα materials τα οποία αποτελούν μέρη που εφαρμόζονται σε 3D αντικείμενα για την ρύθμιση του οπτικού τους μέρους. Γίνεται επίσης να επεξεργαστούμε τα γραφικά που θα έχει ένα material κάτω από συγκεκριμένο φωτισμό ή αν καλύπτεται από σκιά κ.τ.λ., δημιουργώντας έτσι ρεαλιστικά γραφικά για τον χρήστη.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο τον γράφο στον οποίο θα δημιουργήσουμε τα materials μας. Ο γράφος του material editor λειτουργεί όπως και ο γράφος των blueprints.
  Επειδή τα περισσότερα κινητά δεν μπορούν να αναπαράγουν σωστά materials με πολλές ιδιότητες, γι' αυτό χρησιμοποιήθηκαν οι έξι βασικότερες ιδιότητες οι οποίες είναι : το βασικό χρώμα, η μεταλλικότητα, ο κατοπτρισμός, η τραχύτητα, η εκπομπή χρώματος και η αδιαφάνεια.

- Στα δεξιά το παράθυρο palette το οποίο περιέχει όλους τους κόμβους που μπορούν να χρησιμοποιηθούν στον material editor. Όλες οι λειτουργίες, εκφράσεις κ.α. είναι κατηγοριοποιημένα και ο σχεδιαστής μπορεί να χρησιμοποιήσει και φίλτρα για εξοικονόμηση χρόνου.

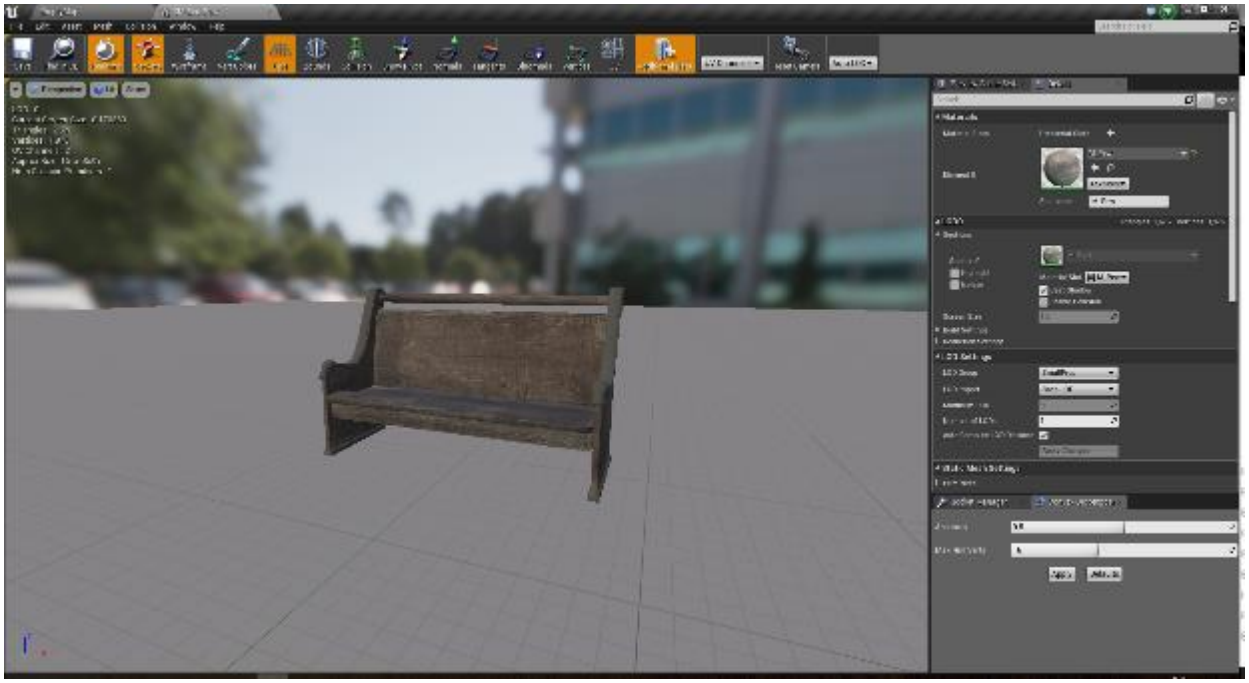
- Κάτω αριστερά το παράθυρο με τις λεπτομέρειες για το material στο οποίο επιλέγουμε τι είδος material δημιουργούμε.

- Πάνω Αριστερά το παράθυρο Viewport στο οποίο βλέπουμε μια προεπισκόπηση του πως θα είναι το material μας. Σε κάθε αλλαγή που κάνουμε ανανεώνεται αυτόματα η προεπισκόπηση. Το σχήμα της προεπισκόπησης από προεπιλογή είναι μια σφαίρα αλλά μπορούμε να το αλλάξουμε από τις ρυθμίσεις του viewport.

2.3.4 Static Mesh Editor

Με τον Static Mesh Editor μπορούμε να μπορούμε να τροποποιήσουμε την φυσική, τη σύγκρουση (collision) και την UV χαρτογράφηση ενός 3d μοντέλου. Τα αντικείμενα με τα οποία θέλουμε ο παίκτης μας είτε να πατάει πάνω σε αυτά (πχ πατώματα, σκάλες) ή να του εμποδίζουν την πρόσβαση (πχ τοίχοι, κάγκελα, πόρτες) πρέπει να τους προσθέσουμε ένα σύστημα σύγκρουσης. Αν το static mesh στο οποίο θέλουμε να

προσθέσουμε σύγκρουση είναι απλού σχεδιασμού τότε μπορούμε να προσθέσουμε το σύστημα σύγκρουσης μέσω του Static Mesh Editor χειροκίνητα ή αυτόματα. Για πολυσύνθετα μοντέλα πρέπει να χρησιμοποιηθεί ένα πρόγραμμα 3d σχεδιασμού.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο το viewport στο οποίο κάνουμε προεπισκόπηση το static mesh.

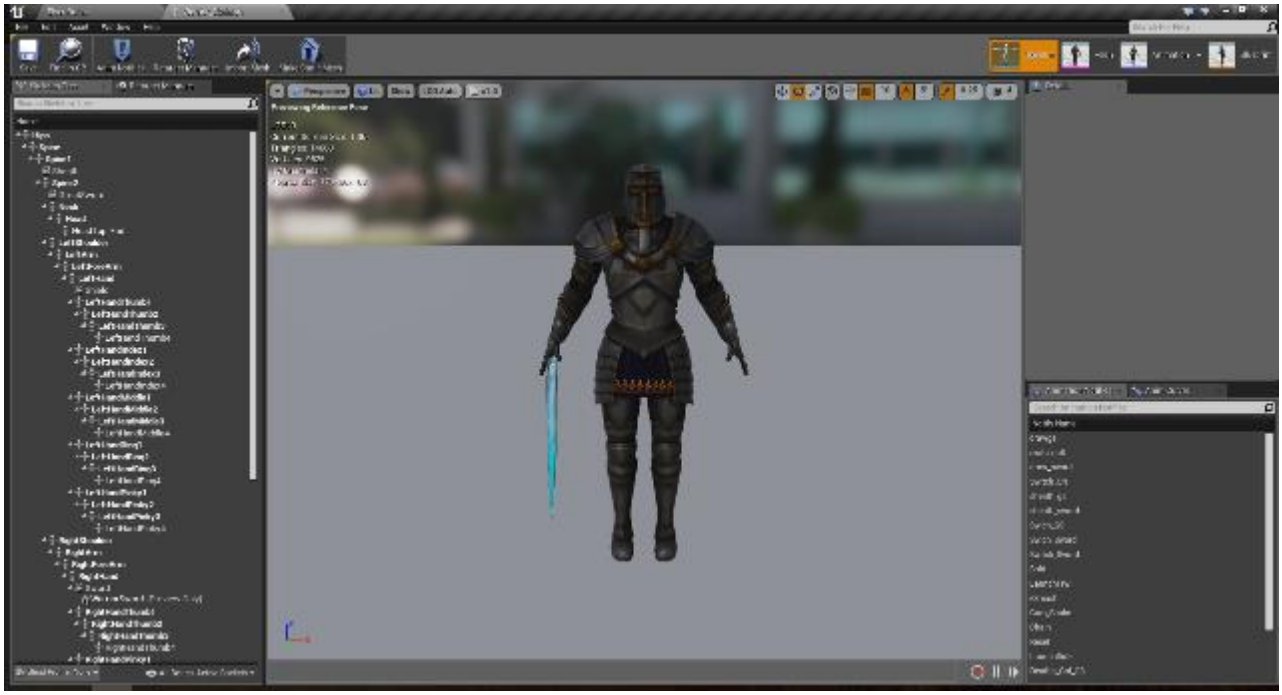
- Δεξιά το παράθυρο με τις λεπτομέρειες για το static mesh όπως ποιο material χρησιμοποιεί, ποιο σύστημα σύγκρουσης υλοποιεί κλπ.

- Πάνω τη γραμμή εργαλείων στην οποία υπάρχουν όλα τα εργαλεία που χρειαζόμαστε για να τροποποιήσουμε το static mesh.

2.3.5 Animation Editors

Για την δημιουργία και επεξεργασία των animations η unreal παρέχει 4 editors οι οποίοι είναι ο Skeleton Editor, ο Skeletal Mesh Editor, ο Animation Editor και ο Animation Blueprint Editor.

Ο Skeleton Editor χρησιμοποιείται για να τροποποιήσουμε το ‘σκελετό’ των actors αλλά και τα δομικά τους στοιχεία (assets). Σε αυτόν τον editor προσθέτουμε sockets στους ‘σκελετούς’ των actors. Τα sockets είναι εξωτερικά αντικείμενα τα οποία θέλουμε να προσθέσουμε σε ένα μοντέλο όπως π.χ. ένα σπαθί στο χέρι του παίχτη. Έπειτα μέσω του Skeleton Editor μπορούμε να τροποποιήσουμε το μέγεθος και τη θέση του socket.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

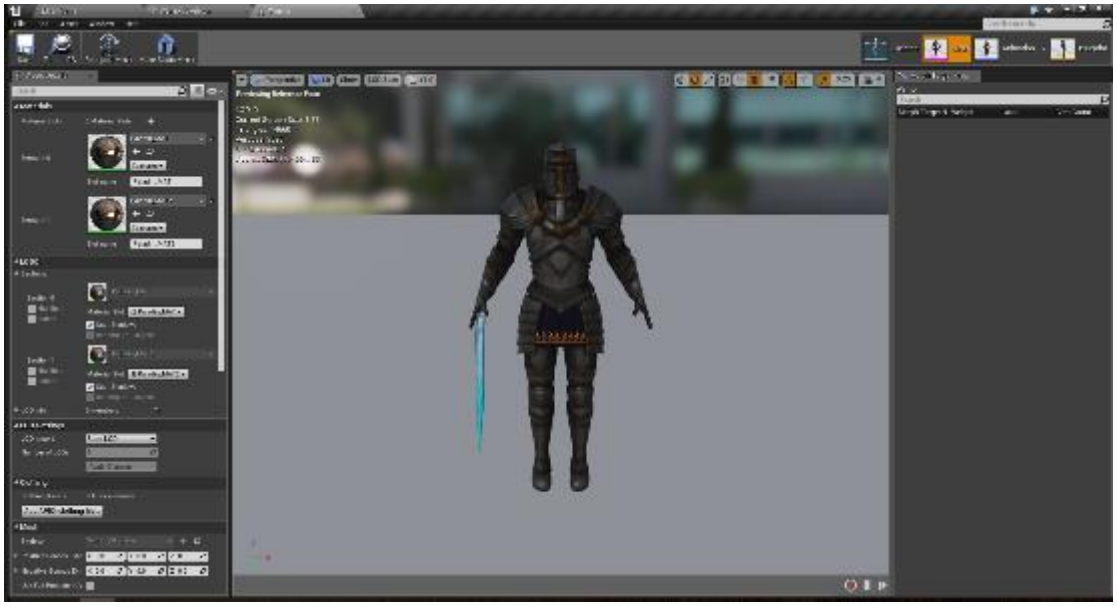
- Στο κέντρο το viewport το οποίο μας δίνει μια προεπισκόπηση του 3D μοντέλου μας με τις αλλαγές που του έχουμε κάνει. Εδώ επίσης τοποθετούμε και τροποποιούμε τα sockets.

- Στα αριστερά την ιεραρχία του ‘σκελετού’ μας. Εδώ επίσης μπορούμε να δημιουργήσουμε τα sockets.

- Πάνω δεξιά έχουμε το παράθυρο λεπτομερειών στο οποίο τροποποιούμε τις ρυθμίσεις των στοιχείων που έχουμε προσθέσει.

- Κάτω δεξιά το παράθυρο Anim Curves το οποίο μας δίνει μια προεπισκόπηση των καμπύλων που μπορούμε να εφαρμόσουμε στο τρέχον πλέγμα.

Μέσω του Skeletal Mesh Editor μπορούμε να αναθέσουμε τα materials που έχουμε φτιάξει στο ‘σκελετο’ των actors μας.



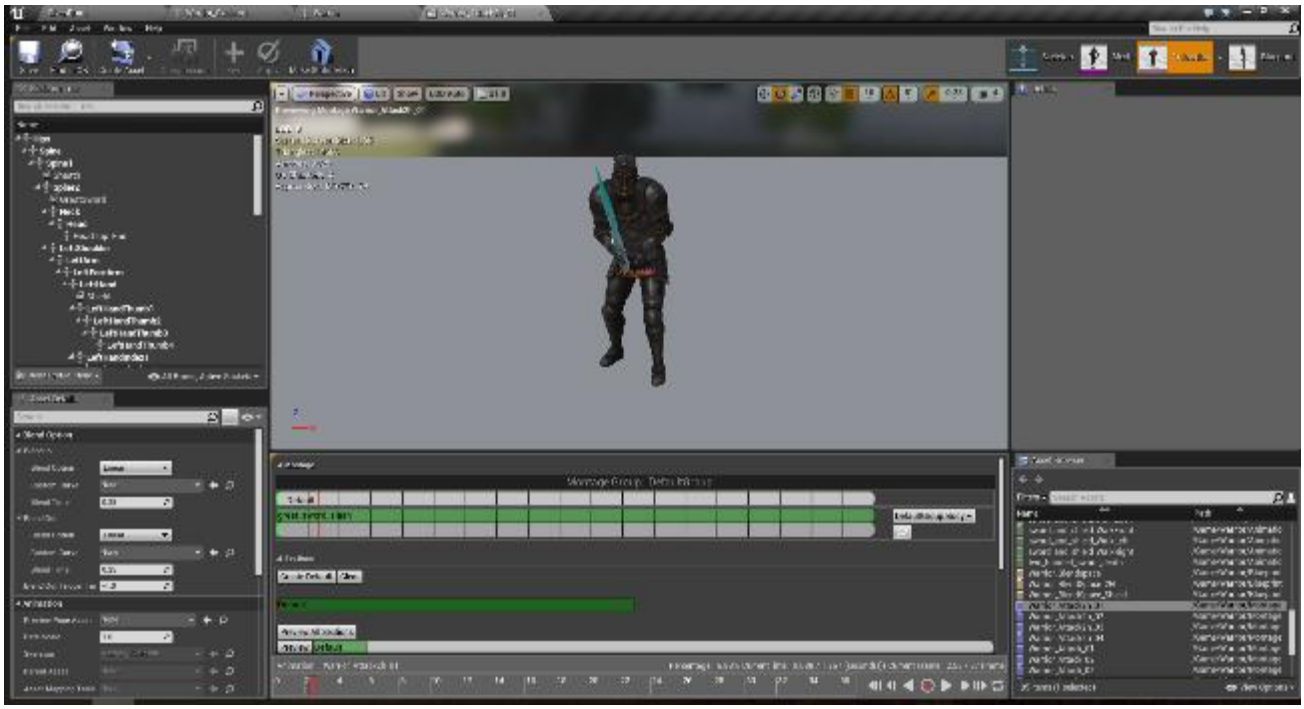
Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο το viewport το οποίο μας δίνει μια προεπισκόπηση των animation μας πάνω στο μοντέλο που θα εφαρμοστούν.

- Στα αριστερά είναι το παράθυρο των λεπτομερειών για τα στοιχεία μας. Δίνει πληροφορίες για τα material που έχουμε εφαρμόσει στο 3D μοντέλο μας.

- Στα δεξιά το παράθυρο Morph Targets μας δίνει μια προεπισκόπηση για αλλαγές που μπορούμε να κάνουμε σε μεμονομένα σημεία του 3D μοντέλου.

Με τον Animation Editor μπορούμε να δημιουργήσουμε animations. Δουλεύει με Animation Sequences τα οποία είναι κινούμενα στοιχεία (assets) και περιέχουν πληροφορίες για τη θέση, περιστροφή και κλίμακα ενός ‘σκελετού’. Έτσι όταν έχουμε αρκετά Animation Sequences δημιουργείται ένα animation.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο το viewport το οποίο μας αναπαράγει τα animation που έχουμε επιλέξει πάνω στο μοντέλο μας.

- Στα αριστερά το παράθυρο λεπτομερειών/δέντρου ιεραρχίας, τα οποία έχουν ίδια λειτουργικότητα με τα αντίστοιχα παράθυρα των Skeletal Mesh Editor και Skeleton Editor.

- Κάτω ο Asset Editor ο οποίος μεταβάλλεται ανάλογα με το είδος του animation που έχουμε ανοιχτό. Εδώ μπορούμε να τροποποιήσουμε τα animation sequences και γενικότερα οποιοδήποτε είδος animation.

- Πάνω δεξιά το παράθυρο λεπτομερειών το οποίο έχει ίδια λειτουργικότητα με το αντίστοιχο παράθυρο του level editor.

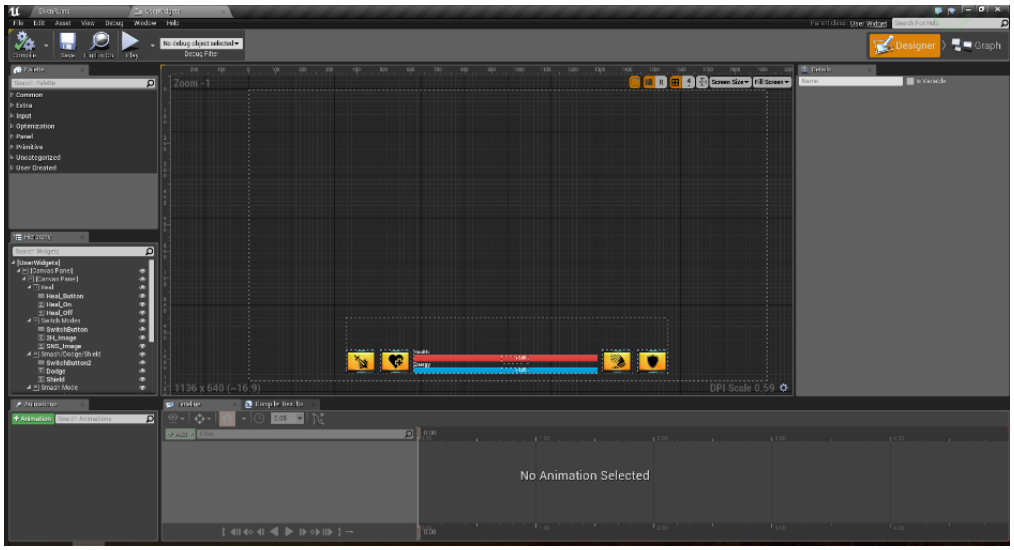
- Κάτω δεξιά είναι το παράθυρο των assets το οποίο μας δείχνει όλα τα animation στοιχεία που μπορούμε να χρησιμοποιήσουμε με το τρέχον επιλεγμένο.

Ο Animation Blueprint Editor χρησιμοποιείται για να ορίσουμε το πότε και το πως θα παιχτούν τα animations.

Η διεπαφή χρήστη όπως φαίνεται στην παραπάνω εικόνα είναι σχεδόν ίδια με αυτή του Blueprint editor.

2.3.6 UMG UI Editor

Με το UMG UI Editor μπορούμε να δημιουργήσουμε το σχεδιαστικό κομμάτι του παιχνιδιού (UI) όπως το κυρίως μενού, το HUD κ.α. Ο editor αυτός λειτουργεί με widgets τα οποία είναι μια σειρά από προ-σχεδιασμένες συναρτήσεις που μπορούν να χρησιμοποιηθούν για την κατασκευή της διασύνδεσης (UI). Τα Widgets επεξεργάζονται με ένα εξειδικευμένο Blueprint, το οποίο χρησιμοποιεί δύο καρτέλες για την κατασκευή των widget, την καρτέλα Designer η οποία επιτρέπει την οπτική διάταξη της διεπαφής και τις βασικές συναρτήσεις και την καρτέλα Graph η οποία παρέχει τη λειτουργικότητα πίσω από τα Widgets που χρησιμοποιήθηκαν.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Στο κέντρο τον γράφο στον οποίο τοποθετούμε εύκολα με drag and drop από το παράθυρο palette.

- Πάνω Αριστερά το παράθυρο palette το οποίο περιέχει όλα τα widgets που μπορούμε να τοποθετήσουμε στο γράφο. Όλες οι λειτουργίες, εκφράσεις κ.α. είναι κατηγοριοποιημένα και ο σχεδιαστής μπορεί να χρησιμοποιήσει και φίλτρα για εξοικονόμηση χρόνου.

- Αριστερά το παράθυρο ιεραρχίας το οποίο μας δείχνει τα widget που έχουμε τοποθετήσει στο γράφο μας με ιεραρχική σειρά.

- Κάτω αριστερά το παράθυρο animations το οποίο μας επιτρέπει να χρησιμοποιήσουμε κινηματικά εφέ που έχουν δημιουργηθεί για widgets.

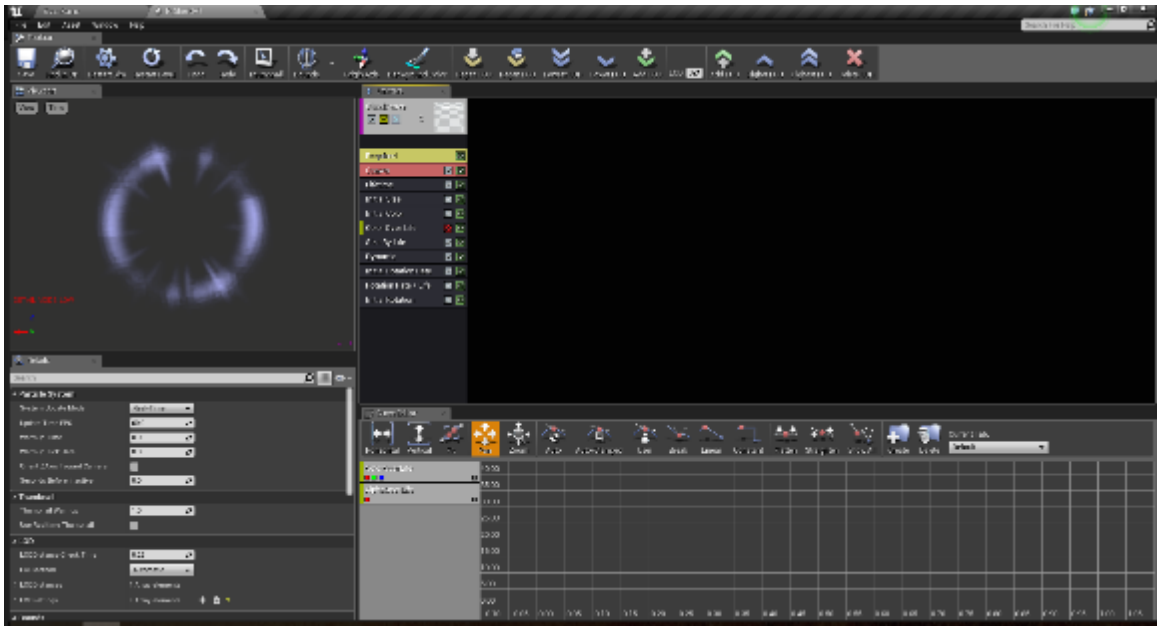
- Κάτω το παράθυρο timeline που μας δείχνει το χρονοδιάγραμμα των κινηματικών εφέ.

- Δεξιά το παράθυρο με τις λεπτομέρειες για το widget που έχουμε επιλέξει.

2.3.7 Cascade

Το Cascade είναι το εργαλείο για το σύστημα σωματιδίων το οποίο μας επιτρέπει να δημιουργήσουμε μοναδικά οπτικά εφέ όπως καπνό, σπίθες, φωτιά κ.α. Τα σωματίδια μπορούν να θεωρηθούν ως σημεία στο 3D χώρο με ορισμένες ιδιότητες οι οποίες κυβερνούν τις αλληλεπιδράσεις μεταξύ τους και του γύρου κόσμου. Το cascade έχει αρκετές λειτουργίες και παρέχει τη δυνατότητα να δημιουργήσει σχεδόν οποιοδήποτε σύστημα σωματιδίων. Τα συστήματα σωματιδίων

χρησιμοποιούνται σε μεγάλο βαθμό για την προσομοίωση φυσικών φαινομένων.



Στη διεπαφή χρήστη βλέπουμε τα εξής :

- Πάνω αριστερά το viewport που μας δείχνει μια προεπισκόπηση του τρέχοντος συστήματος σωματιδίων.

- Κάτω αριστερά το παράθυρο λεπτομερειών στο οποίο τροποποιούμε τις ρυθμίσεις του τρέχοντος συστήματος σωματιδίων.

- Πάνω δεξιά το παράθυρο μας δείχνει όλους τους πομπούς του τρέχοντος συστήματος σωματιδίων.

- Κάτω δεξιά είναι ο editor κυρτότητας ο οποίος μας δείχνει ιδιότητες που τροποποιούνται. Εδώ επίσης μπορούμε να προσαρμόσουμε το σύστημα σωματιδίων την ώρα που το αναπαράγουμε.

2.3.8 LightMass

Το LightMass είναι το σύστημα φωτισμού της unreal. Παρέχει υψηλής ποιότητας στατικό φωτισμό και χάρτες σκιών (shadow maps) με τεχνικές που είναι υπερβολικά πολύπλοκες για ενσωμάτωση στο τρέχον υπολογιστικό υλικό. Το Lightmass πρέπει επομένως, να εκτελεστεί χωρίς σύνδεση (offline), πριν από τη φόρτωση του επιπέδου. Η Unreal προσφέρει τέσσερα είδη φωτισμού :

- Το κατευθυνόμενο φως (Directional light) το οποίο είναι ο ιδανικός φωτισμός όταν υπάρχει μια υπαίθρια σκηνή και πρέπει γίνει μια προσομοίωση του φωτός του ήλιου, διότι αυτό το είδος προσομοιώνει το φως που προέρχεται από μια πηγή απείρως μακριά.

- Το φως ενός σημείου (point light) στο οποίο ένα σημείο προσομοιώνει το φως που προέρχεται από μία μόνο πηγή του φωτός και εκπέμπει ομοιόμορφα σε όλες τις κατευθύνσεις σαν μια λάμπα. Είναι το πιο χρησιμοποιημένο είδος φωτισμού.

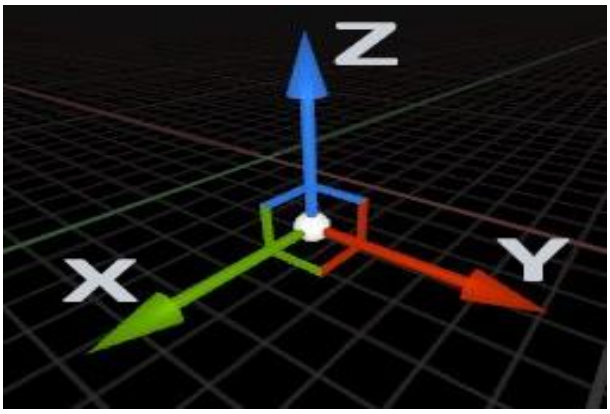
- Το ‘προβολέα’ (spotlight) που μοιάζει πολύ με το κατευθυνόμενο φωτισμό διότι και οι δύο εκπέμπουν φως από μία πηγή. Η διαφορά τους είναι ότι το κατευθυνόμενο φως εκπέμπει φως ομοιόμορφα σε όλες τις κατευθύνσεις, ενώ ο προβολέας εκπέμπει φως σε μια κατεύθυνση σε σχήμα κώνου.

- Το ‘φως του ουρανού’ (sky light) που προσομοιώνει το φως που αντανακλάται από την ατμόσφαιρα.

2.4 Συντεταγμένες & Μετασχηματισμοί

Η κατανόηση κάθε είδους 3D περιεχομένου απαιτεί την κατανόηση της χρήσης των τρισδιάστατων συντεταγμένων γνωστές ως καρτεσιανές συντεταγμένες. Καρτεσιανές συντεταγμένες είναι ένα σύστημα

υπολογισμών από το οποίο μπορούμε να αντλήσουμε πληροφορίες ή σημεία μέσα σε ένα δεδομένο χώρο. Όταν τοποθετούμε ένα σημείο σε ένα χώρο δύο διαστάσεων πρέπει να επιλέξουμε που θα τοποθετηθεί στον άξονα X και που στον άξονα Y. Έπειτα αν βρούμε που τα σημεία συναντιούνται έχουμε βρει τις στοχευμένες συντεταγμένες μέσα σε αυτό το χώρο. Η διαδικασία είναι ίδια για τρισδιάστατους χώρους με τη διαφορά ότι έχουν τρεις άξονες που ορίζουν που θα συναντιούνται τα σημεία. Η άξονες είναι ο Z που είναι ο άξονας από πάνω μέχρι κάτω, ο Y που είναι από τα αριστερά στα δεξιά και ο X ο οποίος είναι από τα μπροστά έως πίσω. Εντός του unreal editor κάθε actor μπορεί να τοποθετηθεί και να μεγεθυνθεί χρησιμοποιώντας τους άξονες.



Επίσης εντός του Unreal Editor υπάρχουν τρία είδη μετασχηματισμών των actors οι οποίοι είναι :

- Η μετακίνηση (move) που είναι ο πιο χρησιμοποιημένος μετασχηματισμός actor της UE4. Κάθε actor έχει μια συγκεκριμένη τοποθεσία στους άξονες X, Y και Z. Για να μετακινήσουμε έναν actor επιλέγουμε το εργαλείο της μετακίνησης από το παράθυρο λειτουργιών. Έπειτα μετακινούμε τον actor με απλές κινήσεις του ποντικιού.
- Ο μετασχηματισμός μεγέθους (scale) με τον οποίο μπορούμε να μεγαλώσουμε ή να μικρύνουμε οποιονδήποτε actor. Η προκαθορισμένη τιμή για κάθε actor είναι 1. Στο παράθυρο των λειτουργιών μπορούμε να βρούμε το μετασχηματισμό μεγέθους.

- Η περιστροφή των actors η οποία λειτουργεί με μοίρες όπως στους περισσότερους 3D editors και βρίσκεται στο παράθυρο των λειτουργιών μαζί με τους άλλους μετασχηματισμούς. Η περιστροφή ενός actor μπορεί να γίνει και στους τρεις άξονες.

Οι μετασχηματισμοί μπορούν να γίνουν με δύο τρόπους. Ο πρώτος είναι αυτός που αναλύθηκε παραπάνω δηλαδή η χρήση των εργαλείων μετακίνησης, μεγέθους και περιστροφής. Ο τρόπος αυτός ονομάζεται δια-δραστικός μετασχηματισμός και ενώ είναι εύχρηστος, δεν επιτρέπει ακριβείς αλλαγές. Παραδείγματος χάρη δεν μπορούμε να κάνουμε ένα actor να έχει ύψος ακριβώς 1.82 εκατοστά. Ο δεύτερος τρόπος ονομάζεται χειροκίνητος μετασχηματισμός και αφορά τη χρήση συγκεκριμένων τιμών στο παράθυρο λεπτομερειών για κάθε actor ξεχωριστά. Αν και έτσι θα έχουμε καλύτερο αποτέλεσμα θα χρειαστεί πολύς χρόνος για να γίνουν οι επιθυμητοί μετασχηματισμοί. Επομένως η λύση είναι η χρήση και τον δύο τρόπων για να έχουμε το επιθυμητό αποτέλεσμα σε σύντομο χρονικό διάστημα.

2.5 C++

Η C++ είναι η κύρια γλώσσα προγραμματισμού της Unreal η οποία δημιουργήθηκε το 1979 από τον Bjarne Stroustrup και αποτελεί μια επέκταση της γλώσσας C. Είναι αντικειμενοστραφής και θεωρείται μια από τις πιο διάσημες γλώσσες και χρησιμοποιείται κυρίως σε εφαρμογές/προγράμματα του συστήματος, drivers, εφαρμογές πελάτη-server και σε ενσωματωμένο firmware. Η κύρια έμφαση της C ++ είναι η συλλογή της από προκαθορισμένες κλάσεις, οι οποίες είναι τύποι δεδομένων που μπορούν να αρχικοποιούνται πολλές φορές. Επίσης η C++ περιλαμβάνει πολλούς τελεστές, όπως η σύγκριση, αριθμητική, τροποποίηση bit, λογικοί φορείς κ.λπ. Ένα από τα πιο ελκυστικά χαρακτηριστικά της C ++ είναι ότι επιτρέπει την υπερφόρτωση ορισμένων τελεστών. Μερικές από τις βασικές έννοιες της γλώσσας

προγραμματισμού C++ είναι ο πολυμορφισμός, οι συναρτήσεις virtual & friend, τα πρότυπα, τα namespaces και οι δείκτες.

Επιπλέον η Unreal δίνει πρόσβαση στον πηγαίο κώδικα της που είναι σε C++. Έτσι γίνεται να ενσωματωθεί εντός της μηχανής επιπλέον λογισμικό και επίσης γίνεται να τροποποιήσουμε τον πηγαίο κώδικα για να προσθέσουμε νέες λειτουργίες. Με αυτό τον τρόπο η Unreal κερδίζει ευελιξία.

Τέλος η συγγραφή κώδικα με c++ έχει περίπου δέκα φόρες πιο γρήγορα αποτελέσματα από τα Blueprints που σημαίνει ότι αν είχαμε ένα παιχνίδι με πολύπλοκα συστήματα και λειτουργίες υλοποιημένο μόνο με blueprints, τότε το παιχνίδι αυτό θα είχε πολύ κακή απόδοση ανεξαρτήτως της πλατφόρμας που θα παίζει. Επομένως αν και η χρήση της C++ δεν είναι υποχρεωτική παρόλα αυτά συνιστάται για την δημιουργία περίπλοκων παιχνιδιών.

2.6 Ενσωμάτωση σε Android

Σε αυτή την ενότητα θα εξετάσουμε ανά κατηγορίες ποιες λειτουργίες της unreal μπορούμε να εκμεταλλευτούμε επαρκώς, λόγω μειωμένων δυνατοτήτων των κινητών-tablets που χρησιμοποιούν android λογισμικό σε σχέση με άλλες συσκευές με καλύτερο hardware (π.χ. H/Y, playstation, xbox). Επίσης θα αναλυθούν τρόποι και τεχνικές για να υπάρχει παρά τους περιορισμούς το βέλτιστο αποτέλεσμα σε όλες τις συσκευές android.

2.6.1 Περιορισμοί γραφικών

Όσον αφορά τα γραφικά η μέγιστη ανάλυση που μπορεί να έχει ένα γραφικό στοιχείο είναι 2048px. Όμως επειδή για τις περισσότερες συσκευές αυτή η ανάλυση είναι μεγάλη, πρέπει να χρησιμοποιούνται πολύ μικρότερα μεγέθη. Επίσης το μέγεθος των γραφικών στοιχείων έχει μεγάλη σημασία οπότε πρέπει να χρησιμοποιούνται τεχνικές

συμπίεσης των εικόνων που χρησιμοποιούνται. Όπως προαναφέρθηκε και στην υπό-ενότητα 2.2.3 οι περισσότερες συσκευές android δεν μπορούν να αναπαράγουν σωστά materials με συγκεκριμένες ιδιότητες. Επιπλέον τα materials έχουν πάρα πολλές ιδιότητες και η πλατφόρμα android υποστηρίζει τις περισσότερες από αυτές. Όμως όσες περισσότερες ιδιότητες έχει ένα material τόσο απαιτεί παραπάνω επεξεργαστική ισχύ και γραφική ανάλυση πράγμα που δεν μπορούν να αναπαράγουν οι περισσότερες συσκευές android. Γι' αυτό το λόγο τα περισσότερα materials που χρησιμοποιήθηκαν είχαν τις βασικότερες ιδιότητες που είναι το βασικό χρώμα, ο κατοπτρισμός και η τραχύτητα. Επίσης οι κινητές συσκευές δεν υποστηρίζουν την λειτουργία PhysX Destruction που υλοποιεί η unreal και επιτρέπει δυναμική καταστροφή αντικειμένων εντός του παιχνιδιού.

2.6.2 Περιορισμοί Φωτισμού και Σκιών

Παρομοίως με τα γραφικά, ο φωτισμός της unreal για να τρέξει σωστά σε συσκευές android, υπάρχουν κάποιοι περιορισμοί. Τα δυναμικά αντικείμενα παραδείγματος χάρη δέχονται κανονικά φωτισμό αλλά δεν έχουν σκιές λόγω της μεγάλης υπολογιστικής ισχύος που χρειάζεται για να δημιουργούνται δυναμικές σκιές. Επίσης η αντανάκλαση που έχει ο φωτισμός απαιτεί μεγάλη υπολογιστική ισχύ, άρα πρέπει είτε να απενεργοποιούνται τελείως οι αντανακλάσεις ή να έχουν σαφώς μειωμένη ποιότητα.

Γι' αυτό το λόγο η unreal έχει δημιουργήσει τις κλιμακωτές σκιές (cascaded shadows). Αυτή η μέθοδος χωρίζει κλιμακωτά όλες τις σκιές που φαίνονται στη κάμερα, και όσο απομακρύνεται μια σκιά από την κάμερα, τόσο χαμηλότερη ανάλυση θα έχει. Αν και με αυτή τη μέθοδο μειώνεται η γραφική αναπαράσταση, έχουμε καλό αποτέλεσμα που μπορεί να τρέξει σωστά στις περισσότερες συσκευές android.

2.6.3 Άλλοι Περιορισμοί

- Το μέγεθος του παιχνιδιού θα είναι πάντα μεγάλο ακόμα και αν έχει φτιαχτεί ένα πολύ μικρό παιχνίδι. Υπάρχουν βέβαια αρκετές λειτουργίες τις Unreal για να μειωθεί σημαντικά το μέγεθος, παρόλα αυτά πάντα το τελικό μέγεθος θα είναι μεγάλο.
- Η Unreal επισήμως δεν υποστηρίζει ακόμα την προσθήκη κοινωνικών μέσων δικτύωσης σε παιχνίδια για συσκευές android. Ανεπίσημα βέβαια υπάρχουν τα κατάλληλα αρχεία προς αγορά στο ηλεκτρονικό κατάστημα της unreal από διάφορους σχεδιαστές, όμως οι περισσότερες μηχανές παιχνιδιών παρέχουν δωρεάν τις λειτουργίες για τη σύνδεση σε κοινωνικά μέσα δικτύωσης.
- Η Unreal δεν υποστηρίζει την εικονικοποίηση (virtualization), δηλαδή τον μηχανισμό να προσομοιώσει στην συγκεκριμένη περίπτωση ένα κινητό ή tablet με λογισμικό android στον Η/Υ. Αυτό σημαίνει ότι κάθε φορά που πρέπει να δοκιμαστεί πειραματικά το παιχνίδι έπρεπε αναγκαστικά να γίνει σε πραγματική συσκευή android, πράγμα το οποίο έπαιρνε πάρα πολύ χρόνο.

2.6.4 Device Profiles

Λόγω των πολλών εταιριών ανά τον κόσμο που κατασκευάζουν συσκευές με λογισμικό android, οι περισσότερες έχουν διαφορετικά χαρακτηριστικά και λειτουργίες. Γι’ αυτό το λόγο η unreal έχει ένα αρχείο τύπου .ini στο οποίο μπορούμε να θέσουμε ρυθμίσεις για όλους τους τύπους συσκευών android. Επιπλέον μπορούμε να δημιουργήσουμε προφίλ για συγκεκριμένα μοντέλα συσκευών ή μπορούμε να δημιουργήσουμε τρία προφίλ τα οποία θα είναι για ‘χαμηλού’, ‘μεσαίου’ και ‘μεγάλου’ επιπέδου συσκευές. Αυτά τα προφίλ αναφέρονται στην υπολογιστική ισχύ και τις γραφικές δυνατότητες της κάθε συσκευής ανάλογα με το προφίλ που ανήκει. Η unreal διαθέτει

επίσης ένα αρχείο που αναγνωρίζει αυτόματα τις δυνατότητες κάθε συσκευής και εφαρμόζει τις ρυθμίσεις μας. Οι ρυθμίσεις που μπορούμε να κάνουμε αφορούν την ποιότητα όλων των γραφικών, του φωτισμού και την ποιότητα των μακρινών αντικειμένων (LOD). Επίσης εδώ μπορούμε να θέσουμε το μέγεθος που θα πιάνει το παιχνίδι από την οθόνη της συσκευής.

Κεφάλαιο 3
Σχεδίαση & Διεπαφή

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναλυθεί ο σχεδιασμός και ο σκοπός του παιχνιδιού καθώς και το γραφικό περιβάλλον χρήστη. Επειδή η πλειοψηφία των συσκευών android λειτουργούν εξολοκλήρου με οθόνη αφής, έτσι πρέπει να υπάρχει μια γραφική διεπαφή με εύκολη καθοδήγηση για τον χρήστη και με κουμπιά αφής για τις επιπρόσθετες λειτουργίες.

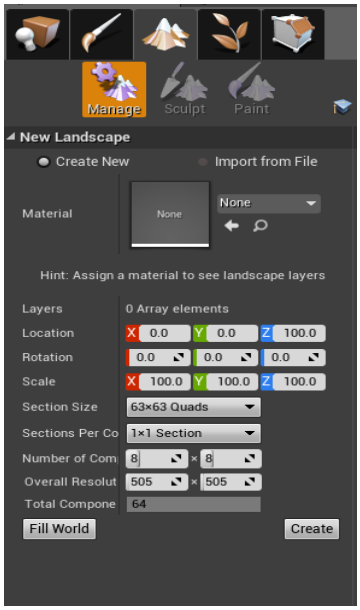
3.2 Σκοπός του Παιχνιδιού

Ο παίκτης που χρησιμοποιεί ο χρήστης είναι ο ‘πολεμιστής’ ο οποίος χρησιμοποιεί ένα σπαθί και μια ασπίδα ή ένα μεγάλο σπαθί που το χειρίζεται με τα δύο χέρια για να πολεμήσει. Σκοπός του παιχνιδιού είναι να νικήσει τον πέμπτο και δυνατότερο αντίπαλο και να πάρει την καλύτερη βαθμολογία νικώντας όλους του αντιπάλους. Για να φτάσει μέχρι τον τελευταίο αντίπαλο πρέπει να νικήσει τους πρώτους τρεις αντιπάλους όπου ο καθένας ξεκλειδώνει μια πόρτα για να προχωρήσει περαιτέρω στο επίπεδο. Καθώς ο πολεμιστής προχωράει περαιτέρω στο επίπεδο, οι αντίπαλοι είναι πιο δυνατοί και χρησιμοποιούν διαφορετικές επιθέσεις.

3.3 Σχεδιασμός Επιπέδων

Για να ξεκινήσουμε να κατασκευάζουμε επίπεδα και χάρτες θα χρειαστούν οντότητες και αντικείμενα και πιο συγκεκριμένα static meshes για να δημιουργηθεί η γεωμετρία του παιχνιδιού. Επίσης χρειαζόμαστε και σωματίδια (Particles) και materials για να προσδώσουν στο παιχνίδι την ανάλογη ατμόσφαιρα. Τα περισσότερα αντικείμενα που χρησιμοποιήθηκαν για την δημιουργία του επιπέδου διατίθενται από την unreal για ελεύθερη χρήση στο κοινό και τα υπόλοιπα δημιουργήθηκαν για τους σκοπούς της πτυχιακής εργασίας. Για την δημιουργία των αντικειμένων αυτών απαιτείται η σχεδίαση τους σε ένα πρόγραμμα μοντελοποίησης και ύστερα η δημιουργία κατάλληλων γραφικών με ένα πρόγραμμα επεξεργασίας εικόνων.

Επειδή το παιχνίδι έχει θέμα μεσαιωνικής εποχής χρησιμοποιήθηκαν τα ανάλογα αντικείμενα για την δημιουργία ενός επιπέδου με βουνά, θάλασσα και στο κέντρο ένα μεσαιωνικό κάστρο όπου πραγματοποιείται και το παιχνίδι. Για την δημιουργία του εδάφους και της θάλασσας χρησιμοποιήθηκε το εργαλείο δημιουργίας τοπίων (Landscape Tool) το οποίο δημιουργεί αυτόματα ένα τοπίο με τυχαίες γεωμετρικές αναλογίες. Έπειτα με τα εργαλεία ‘γλυπτικής’ του Landscape Tool μπορούμε να αλλάξουμε την γεωμετρία του τοπίου.



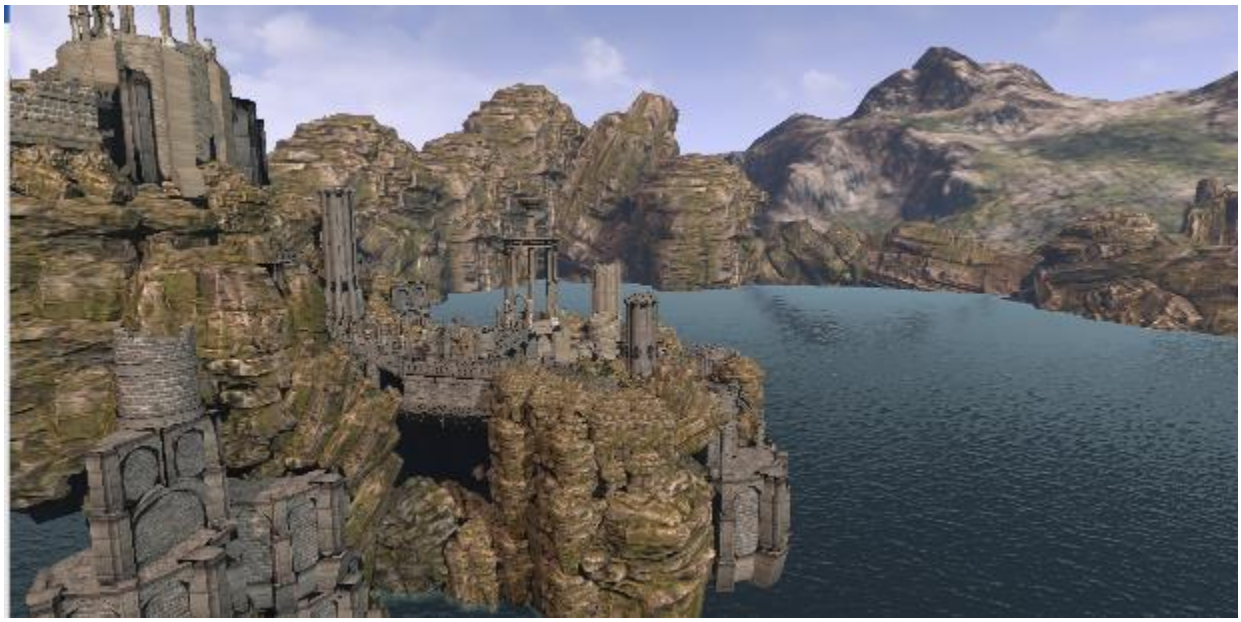
Για το επιθυμητό γραφικό αποτέλεσμα χρησιμοποιήθηκαν material που αναπαριστούν γρασίδι για τις χαμηλότερες και χώμα για τις ψηλότερες περιοχές του χάρτη. Επίσης προστέθηκαν και αντικείμενα που αναπαριστούν βράχους στις άκρες του παιχνιδιού έτσι ώστε να μην είναι ορατά τα όρια του χάρτη. Επιπρόσθετα προστέθηκε ένα sky light για την γραφική προσομοίωση του ουρανού και του ήλιου αλλά και για να προστεθεί φως στην πίστα.

Τρισδιάστατα μοντέλα αντικειμένων.

Έπειτα πραγματοποιήθηκε η δημιουργία του κάστρου προσθέτοντας αρχικά 2 όγκους Post Process Volume και Lightmass Importance Volume οι οποίοι δεν είναι ορατοί από τον χρήστη και καλύπτουν το κάστρο.

Χρησιμοποιούνται για να θέσουμε τα όρια που θέλουμε να υπάρχουν τα καλύτερα αποτελέσματα επεξεργασίας και φωτισμού αντίστοιχα. Έπειτα προστέθηκε ένας όγκος Nav Mesh Bounds Volume για να θέσουμε τα όρια στα οποία μπορεί ο παίκτης μας και οι αντίπαλοι να κινηθούν. Επίσης προστέθηκαν όγκοι Blocking Volumes οι οποίοι δεν επιτρέπουν στον παίκτη να φτάσει σε σημεία του χάρτη που δεν θέλουμε να έχει πρόσβαση ο χρήστης. Επίσης στην αρχή του κάστρου προστέθηκε ένα actor Player Start με το οποίο δηλώνουμε το σημείο του χάρτη που θα ξεκινήσει ο παίχτης μας αφού φορτώσει το επίπεδο. Ακόμη προστέθηκαν λίγα μέτρα πριν από κάθε αντίπαλο Trigger Boxes, τα οποία είναι αόρατα ‘κουτιά’, μέσω των οποίων όταν ο παίκτης περνάει μέσα από ένα trigger box ξεκινάει τη μάχη με τον κοντινότερο αντίπαλο.

Επιπρόσθετα προστέθηκαν σωματίδια, δέντρα και φυτά για να προσδώσουν στο επίπεδο την ανάλογη ατμόσφαιρα.



Πανοραμικές εικόνες του επιπέδου.

Τέλος δημιουργήθηκε άλλος ένας χάρτης ο οποίος θεωρείται ο εσωτερικός χώρος του κάστρου. Για αυτόν τον χάρτη χρησιμοποιήθηκε η λειτουργία Level Streaming μέσω της οποίας αφού εισέλθουμε στον νέο χάρτη ο οποίος είναι εντός του ίδιου επιπέδου που βρίσκεται και ο χάρτης με το εξωτερικό του κάστρου, ξεφορτώνει τον αρχικό χάρτη από τη μνήμη και φορτώνει μόνο το δεύτερο χάρτη που είναι το εσωτερικό του κάστρου εξοικονομώντας στην πορεία μνήμη και υπολογιστική ισχύ.

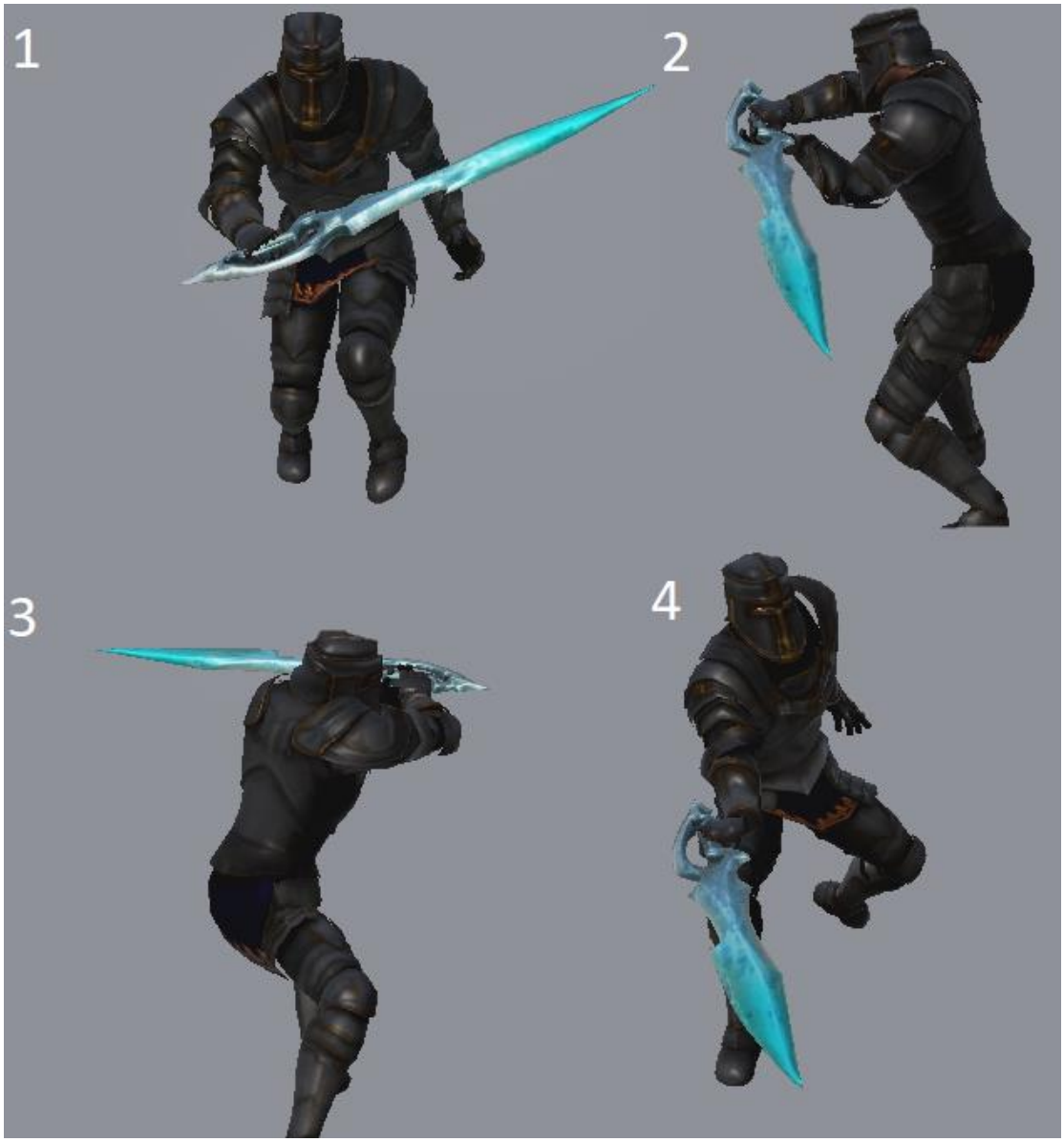
Το εσωτερικό του κάστρου.

3.4 Χαρακτήρας και Αντίπαλοι

Το τρισδιάστατο μοντέλο του χαρακτήρα

Το μοντέλο και τα animations του χαρακτήρα αποκτήθηκαν από την online πλατφόρμα Mixamo της Adobe η οποία παρέχει δωρεάν 3d μοντέλα και animations χαρακτήρων για προγραμματιστές και σχεδιαστές. Έπειτα τα μοντέλα και τα animations τροποποιήθηκαν με τα εργαλεία της unreal έτσι ώστε να υπάρχει μέγιστη συμβατότητα.

Στο μοντέλο του χαρακτήρα του χρήστη έχουν προστεθεί 26 animation τα οποία είναι οι κινήσεις του χαρακτήρα όπως τρέξιμο, στάσιμη κατάσταση, αλλαγή όπλων, έξι επιθέσεις με το σπαθί και την ασπίδα, οκτώ επιθέσεις με το μονό σπαθί, στάση θεραπείας, στάση θανάτου και κινήσεις αποφυγής επιθέσεων των αντιπάλων.



Γραφική αναπαράσταση ενός animation επίθεσης.

Μετά την εισαγωγή των animation στην unreal δημιουργούμε για κάθε animation, εκτός του τρεξίματος και της στάσιμης κατάστασης, animation montages τα οποία είναι εργαλεία που μας επιτρέπουν να θέσουμε πάνω στο χρονοδιάγραμμα των animation κάποια χρονικά σημεία στα οποία αργότερα μέσω blueprints θα πυροδοτούνται συγκεκριμένα γεγονότα όπως π.χ τη στιγμή που το σπαθί του παίχτη μας χτυπάει ένα αντίπαλο θα κουνιέται λίγο η κάμερα.

Τα μοντέλα και τα animation των αντιπάλων αποκτήθηκαν επίσης από την πλατφόρμα Mixamo.

Τα τρισδιάστατα μοντέλα των αντιπάλων.

Για τα animations των αντιπάλων ακολουθούμε την ίδια διαδικασία με αυτή του κύριου χαρακτήρα και θέτουμε πάνω στο χρονοδιάγραμμα των animation κάποια χρονικά σημεία στα οποία αργότερα μέσω των blueprints των αντιπάλων θα πυροδοτούνται συγκεκριμένα γεγονότα όπως π.χ τη στιγμή που ένας αντίπαλος χτυπάει τον παίχτη θα πυροδοτείται ένα σωματίδιο για να υποδεικνύει αν ήταν επιτυχές το

χτύπημα. Για τους αντιπάλους του παίκτη χρησιμοποιήθηκαν συνολικά 40 animations για να προσομοιώσουν κινήσεις παρόμοιες με αυτές του παίχτη.

Όσων αφορά τα animations του τρεξίματος και της στάσιμης κατάστασης για τον χαρακτήρα και τους αντιπάλους δημιουργούμε ξεχωριστά Blendspaces τα οποία μας επιτρέπουν να συνδυάσουμε δύο ή περισσότερα animations. Στην συγκεκριμένη περίπτωση θέλουμε να συνδυάσουμε τα animations του τρεξίματος και της στάσιμης κατάστασης έτσι ώστε όταν ο χαρακτήρας και οι αντίπαλοι είναι στάσιμοι να χρησιμοποιούν το ανάλογο animation και όταν αναπτύσσουν ταχύτητα να χρησιμοποιούν το animation του τρεξίματος.

Οι εναλλαγή από στάσιμη κατάσταση σε τρέξιμο.

3.5 Γραφική Διεπαφή Χρήστη

Σε αυτήν την ενότητα θα γίνει περιγραφή της γραφικής διεπαφής χρήστη (GUI) η οποία αποτελείται στο συγκεκριμένο παιχνίδι από το αρχικό μενού, την οθόνη φόρτωσης, το HUD, την τελική οθόνη του αποτελέσματος, τα ψηφιακά χειριστήρια καθοδήγησης και το HUD των αντιπάλων.

Το Κύριο Γραφικό περιβάλλον.

3.5.1 Αρχικό Μενού

Το αρχικό μενού είναι η αρχική οθόνη που εμφανίζει το παιχνίδι κατά την έναρξη. Αποτελείται από δύο κουμπιά αφής και μια στατική εικόνα. Ο χρήστης επιλέγει να ξεκινήσει να παίζει (Start Game) ή να κλείσει το παιχνίδι (Exit).

Το αρχικό Μενού.

3.5.2 Η Οθόνη Φόρτωσης

Η οθόνη φόρτωσης είναι μια απλή στατική εικόνα η οποία χρησιμοποιείται για να ενημερώσει τον παίχτη όταν το παιχνίδι βρίσκεται σε διαδικασία φόρτωσης ενός εκ των δύο χαρτών.

Η οθόνη φόρτωσης.

3.5.3 HUD

Το HUD (Heads-Up Display) είναι το κύριο γραφικό περιβάλλον ενός παιχνιδιού και χρησιμοποιείται για να ενημερώσει τον χρήστη για όλες τις πληροφορίες του χαρακτήρα που είναι άμεσα χρήσιμες. Συνήθως παρέχει μόνο ενημερωτικές πληροφορίες αλλά σε παιχνίδια για συσκευές με οθόνη αφής περιέχει και άλλες λειτουργίες όπως κουμπιά αφής. Ο Χρήστης μπορεί να επιτεθεί στους αντιπάλους πατώντας οπουδήποτε στην οθόνη εκτός από την περιοχή που βρίσκεται το HUD.

Το Heads-Up Display όταν ο παίκτης μάχεται με την ασπίδα και το σπαθί.

Στο συγκεκριμένο παιχνίδι το HUD προσφέρει τις παρακάτω λειτουργίες με σειρά από αριστερά στα δεξιά :

- Το πρώτο στοιχείο του HUD είναι ένα κουμπί αφής το οποίο αλλάζει τον τρόπο που μάχεται ο χαρακτήρας του παιχνιδιού. Στην συγκεκριμένη φάση αλλάζει τον τρόπο μάχης από ασπίδα και σπαθί στο μονό μεγάλο σπαθί.

- Το δεύτερο στοιχείο του HUD είναι ένα κουμπί αφής το οποίο στο πάτημα του επαναφέρει 50 πόντους ζωής στο παίχτη μας, δαπανώντας όμως 30 πόντους ενέργειας. Η συγκεκριμένη ιδιότητα μπορεί να χρησιμοποιηθεί μόνο κάθε 30 δευτερόλεπτα. Στην διάρκεια αυτών των δευτερολέπτων η εικόνα του κουμπιού αλλάζει ώστε να υποδείξει στον χρήστη ότι δεν έχει περάσει ακόμα ο εκτιμώμενος χρόνος.
- Στο κέντρο του HUD έχουμε δύο μπάρες που μας δείχνουν πόσο ζωή και ενέργεια έχει ο παίχτης μας. Οι μπάρες κατεβαίνουν αναλόγως με το ποσοστό ζημιάς που θα δεχτεί ο παίχτης μας.
- Το πέμπτο στοιχείο του HUD είναι ένα κουμπί αφής, το οποίο μπορεί να χρησιμοποιηθεί κατά την διάρκεια ή στο τελείωμα μιας κανονικής επίθεσης για να κάνει ο παίχτης μας μια ειδική επίθεση που κάνει μεγαλύτερη ζημιά. Η επιπλέον επίθεση όμως καταναλώνει περισσότερη ενέργεια από μια κανονική επίθεση.
- Το έκτο στοιχείο του HUD είναι ένα κουμπί αφής το οποίο σε συνεχόμενο πάτημα του χρήστη ενεργοποιεί τη λειτουργία ασπίδας κατά την οποία ο παίχτης μας όποτε δέχεται επίθεση από αντίπαλο δεν χάνει πόντους από τη ζωή του αλλά χάνει πόντους από την ενέργεια του. Όμως κατά τη διάρκεια της ενέργειας αυτής ο παίχτης μας έχει πολύ μειωμένη ταχύτητα. Η λειτουργία ασπίδας απενεργοποιείται όταν ο χρήστης σταματάει να πατάει το κουμπί αφής.

Επειδή όμως ο χαρακτήρας του παιχνιδιού χρησιμοποιεί μια ασπίδα και ένα σπαθί ή ένα μεγάλο σπαθί, το HUD αλλάζει ανάλογα με τον τρόπο που μάχεται ο χαρακτήρας.

Το Heads-Up Display όταν ο παίκτης μάχεται με το μεγάλο σπαθί

Όταν ο χαρακτήρας μάχεται με το μονό μεγάλο σπαθί γίνονται οι παρακάτω αλλαγές στο HUD :

- Στο κουμπί αφής το οποίο αλλάζει τον τρόπο που μάχεται ο χαρακτήρας του παιχνιδιού αλλάζει η εικόνα για να υποδείξει στον παίχτη ότι μάχεται με το μονό σπαθί και πατώντας το κουμπί θα μεταβεί πάλι στην ασπίδα και το σπαθί.
- Το κουμπί που ενεργοποιεί την λειτουργία ασπίδας αλλάζει εικόνα και λειτουργία εφόσον ο παίχτης μας δεν χρησιμοποιεί πλέον την ασπίδα του. Η καινούργια του λειτουργία είναι η αποφυγή επιθέσεων. Κάθε φορά που ο χρήστης πατάει το κουμπί ο παίχτης μας κάνει μια γρήγορη αποφυγή της επίθεσης του αντιπάλου προς την κατεύθυνση που του δείχνει ο χρήστης με το ψηφιακό χειριστήριο. Κάθε αποφυγή καταναλώνει πέντε πόντους ενέργειας.

3.5.4 Η Τελική Οθόνη

Κάθε φορά που ο χρήστης είτε κερδίσει είτε χάνει το παιχνίδι εμφανίζεται η παρακάτω οθόνη η οποία μεταβάλλεται ανάλογα του αποτελέσματος.

Η τελική οθόνη που μας δείχνει το τελικό αποτέλεσμα του παιχνιδιού.

Η παραπάνω οθόνη περιέχει μια μπάρα που δείχνει το σκορ του χρήστη ανάλογα με το πόσους αντιπάλους νίκησε. Ακολουθούν δύο κουμπιά αφής, ένα για να επιστρέψουμε στο κυρίως μενού (Main Menu) και ένα για να ξαναπαίξουμε από την αρχή το παιχνίδι (Retry).

3.5.5 Ψηφιακά Χειριστήρια

Τα Ψηφιακά Χειριστήρια.

Τα ψηφιακά χειριστήρια χρησιμοποιούνται για την ‘οδήγηση’ και την περιστροφή του παίχτη μας. Το αριστερό χειριστήριο χρησιμοποιείται για να ορίσουμε την φορά και την ταχύτητα που θα ταξιδεύει ο παίχτης μας. Το δεξί χειριστήριο χρησιμοποιείται για να ρυθμίσουμε την περιστροφή του παίχτη μας. Παρέχονται και υπάρχουν αυτόματα εντός της Unreal engine. Στο συγκεκριμένο παιχνίδι έχουν ρυθμιστεί και τα δύο χειριστήρια για να είναι πιο εύκολα στη χρήση και επίσης έχει απενεργοποιηθεί η περιστροφή πάνω και προς τα κάτω διότι δεν χρειαζόταν.

3.5.6 HUD Αντιπάλων

Όποτε ο παίχτης μας έρχεται αντιμέτωπος με έναν αντίπαλο τότε στο πάνω μέρος της οθόνης εμφανίζεται ένα HUD το οποίο χρησιμοποιείτε για να υποδείξει στον παίχτη πόσο ζωή έχει ακόμα ο αντίπαλος του.

Κεφάλαιο 4
Υλοποίηση

4.1 Εισαγωγή

Σε αυτό το κεφαλαίο θα εξετάσουμε πως θα προστεθεί λειτουργικότητα στον κύριο χαρακτήρα και στους αντιπάλους. Ο χαρακτήρας μας και οι αντίπαλοι για να λειτουργήσουν χρειάζονται ένα κανονικό blueprint και ένα animation blueprint για τον καθένα ξεχωριστά. Επίσης, θα αναλυθεί η λειτουργικότητα των επιπέδων και της διεπαφής χρήστη. Τα blueprint που θα αναλυθούν σε αυτό το κεφάλαιο είναι επεκτάσιμα και έχουν δημιουργηθεί με τέτοιο τρόπο ώστε να είναι εύκολη και απλή η προσθήκη νέων λειτουργιών.

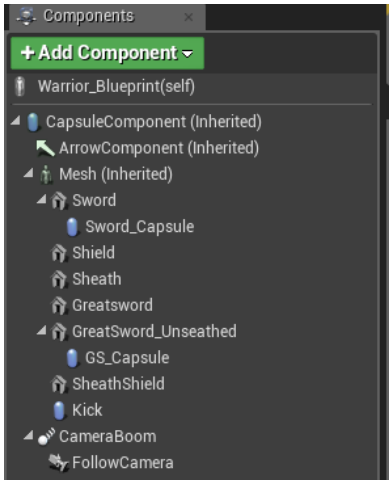
4.2 Λειτουργικότητα Κύριου Χαρακτήρα

Όπως αναφέρθηκε και παραπάνω ο χαρακτήρας μας θα χρειαστεί ένα blueprint το οποίο ονομάζουμε Warrior_Blueprint και με αυτό το blueprint θα προγραμματίσουμε την συμπεριφορά του ‘πολεμιστή’. Επίσης, χρειάζεται ένα animation blueprint το οποίο θα ελέγχει τα κινηματικά του σκελετού του πολεμιστή. Το animation blueprint του ‘πολεμιστή’ ονομάστηκε Warrior_Animation. Αρχικά θέλουμε ο χαρακτήρας μας να κρατάει στο δεξί χέρι ένα σπαθί και στο αριστερό μια ασπίδα και στην πλάτη του να έχει το μονό μεγάλο σπαθί.

Οπότε πρέπει να ανοίξουμε το σκελετό του πολεμιστή που δημιουργήθηκε, όταν προσθέσαμε το static mesh του πολεμιστή στην unreal.

Έπειτα προσθέτουμε ένα socket στο αριστερό χέρι του πολεμιστή, ένα στο δεξί χέρι, ένα στην πλάτη του πολεμιστή και ένα socket στο δεξί γοφό του πολεμιστή όπου εκεί θα ‘αποθηκεύει’ την ασπίδα και το σπαθί όταν χρησιμοποιεί το μονό σπαθί.

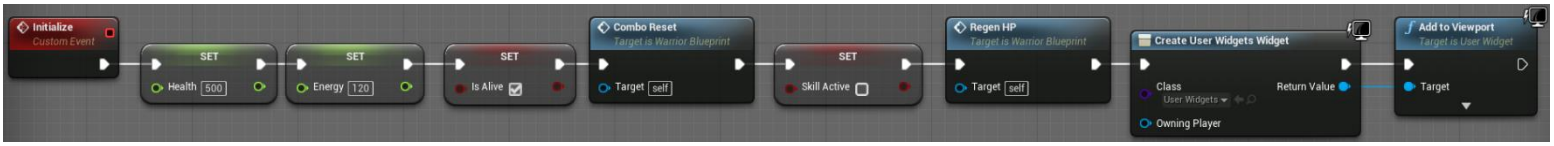
4.2.1 Το Blueprint του Κύριου Χαρακτήρα



Στη συνέχεια στο Warrior_Blueprint στην καρτέλα viewport δημιουργούμε τα ανάλογα components για τα socket που δημιουργήσαμε. Έπειτα για κάθε component επιλέγουμε τα static mesh που του αναλογούν όπως, για το sword component διαλέγουμε το static mesh του σπαθιού που έχουμε προσθέσει στην Unreal. Επίσης δημιουργούμε τα component που αφορούν την αποθήκευση των όπλων στα οποία βάζουμε τα static mesh των όπλων όπως θα θέλαμε να δείχνουν

αποθηκευμένα πάνω στον παίχτη. Στη συνέχεια προσθέτουμε μια κάμερα πίσω από τον πολεμιστή και την προσαρμόζουμε αναλόγως με τη θέση που θέλουμε να βλέπουμε τον παίχτη μας εντός του παιχνιδιού. Μετά πρέπει να προσθέσουμε κάψουλες στα components όπως στο σπαθί το οποίο χρησιμοποιεί ο παίχτης μας για να επιτεθεί στους αντιπάλους. Με βάση τα animations που χρησιμοποιήθηκαν ο πολεμιστής χρησιμοποιεί το κανονικό σπαθί, το μονό σπαθί και το δεξί του πόδι.

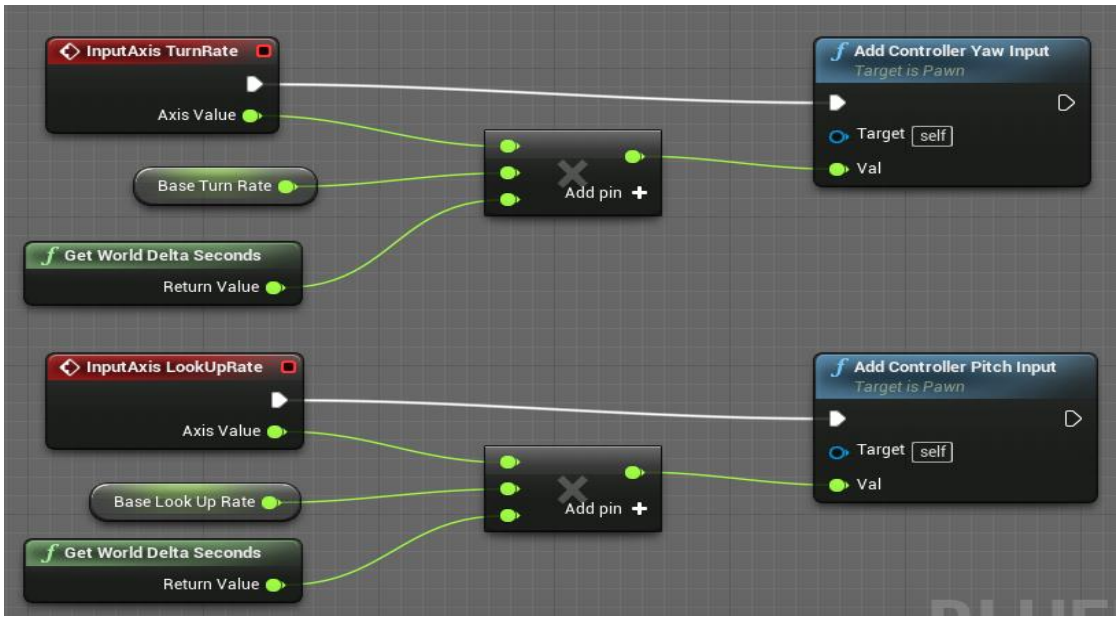
Στη συνέχεια μεταβαίνουμε στην καρτέλα event graph όπου εδώ θα προγραμματίσουμε τη συμπεριφορά του πολεμιστή. Αρχικά δημιουργούμε ένα δικό μας γεγονός με όνομα Initialize το οποίο θα εκτελείται όποτε ξεκινάει από την αρχή το παιχνίδι και θα αρχικοποιεί τα βασικά χαρακτηριστικά του πολεμιστή.



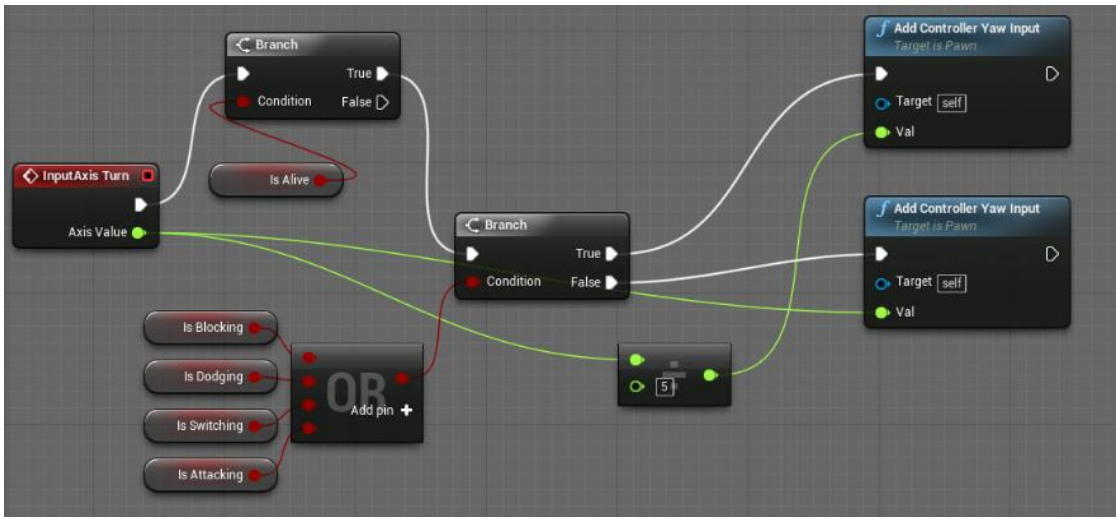
Οι float μεταβλητές Health και Energy αρχικοποιούν τη ζωή και την ενέργεια του πολεμιστή. Η Boolean μεταβλητή IsAlive αρχικοποιείται ως αληθής και χρησιμοποιείται αργότερα ως κριτήριο για κάθε ενέργεια του πολεμιστή. Η συνάρτηση Combo_Reset χρησιμοποιείται για να ακυρώσουμε όλες τις ενέργειες/επιθέσεις του πολεμιστή. Η μεταβλητή SkillActive αρχικοποιείται ως ψευδής και χρησιμοποιείται για να επαναφέρουμε την ιδιότητα του πολεμιστή, να αναπληρώσει πόντους ζωής στην αρχική της κατάσταση. Η συνάρτηση Regen_HP επαναφέρει μισό πόντο ζωής κάθε δέκα δευτερόλεπτα στον πολεμιστή, όπως γίνεται στα περισσότερα παιχνίδια του είδους. Οι συναρτήσεις Create User Widgets και Add to Viewport δημιουργούν την κύρια διεπαφή χρήστη και την προσθέτουν στην οθόνη.

Τα παρακάτω γραφήματα χρησιμοποιούνται για τον υπολογισμό των εισόδων κίνησης του χαρακτήρα και δημιουργούνται αυτόματα από την Unreal. Αφού ληφθούν οι είσοδοι από τον παίκτη, πολλαπλασιάζονται με τα χιλιοστά του δευτερολέπτου εντός του παιχνιδιού με την

συνάρτηση Get World Delta Seconds και με τις δύο Float μεταβλητές Base Turn Rate και Base Look Up Rate υπολογίζεται το ποσοστό κλίσης του μοχλού και της κάμερας.



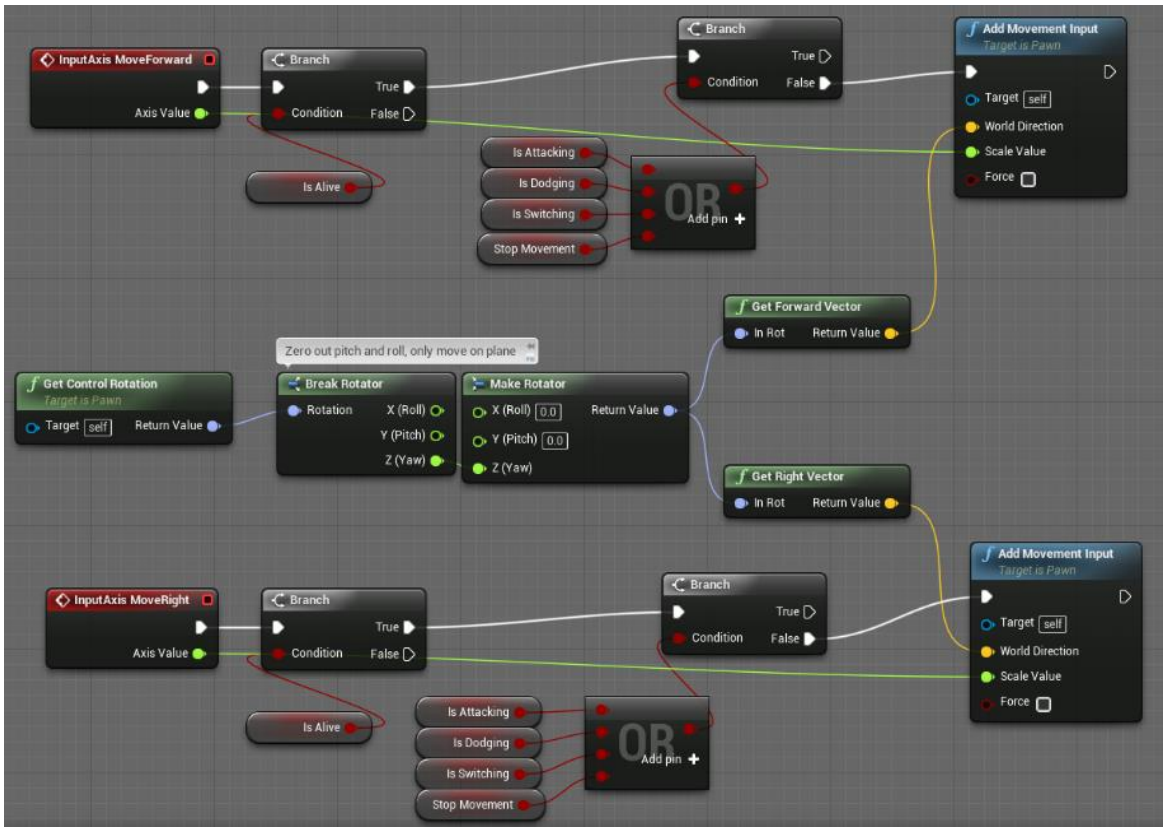
Στο παρακάτω γράφημα υπολογίζεται η δεξιόστροφη και αριστερόστροφη περιστροφή του παίχτη με βάση τα δεδομένα του παραπάνω γραφήματος.



Το γεγονός InputAxis Turn ενεργοποιείται όποτε ο χρήστης στρίβει την κάμερα με το δεξί χειριστήριο. Τα Branch είναι κοινώς στον προγραμματισμό η δομή επιλογής if-then-else. Στο πρώτο branch κάνουμε πρώτα έλεγχο αν ο παίχτης είναι ζωντανός με τη μεταβλητή

IsAlive τότε μπορεί ο χρήστης να περιστρέψει την κάμερα. Με το δεύτερο branch ελέγχουμε αν ο παίχτης την παρούσα στιγμή χρησιμοποιεί την ασπίδα για να μπλοκάρει(isBlocking) ή να κάνει αποφυγή επιθέσεων (IsDodging) ή να αλλάξει όπλα (IsSwitching) ή να επιτεθεί (IsAttacking) τότε η ταχύτητα περιστροφής διαιρείται με το πέντε και έτσι η κάμερα περιστρέφεται πολύ αργά για ευκολία του χρήστη. Αλλιώς η κάμερα περιστρέφεται με την προκαθορισμένη τιμή που δίνεται από την Unreal. Η συνάρτηση Add Controller Yaw Input δίνει μια περιστροφή στον άξονα yaw της κάμερας του χαρακτήρα ανάλογα με την τιμή που δέχεται.

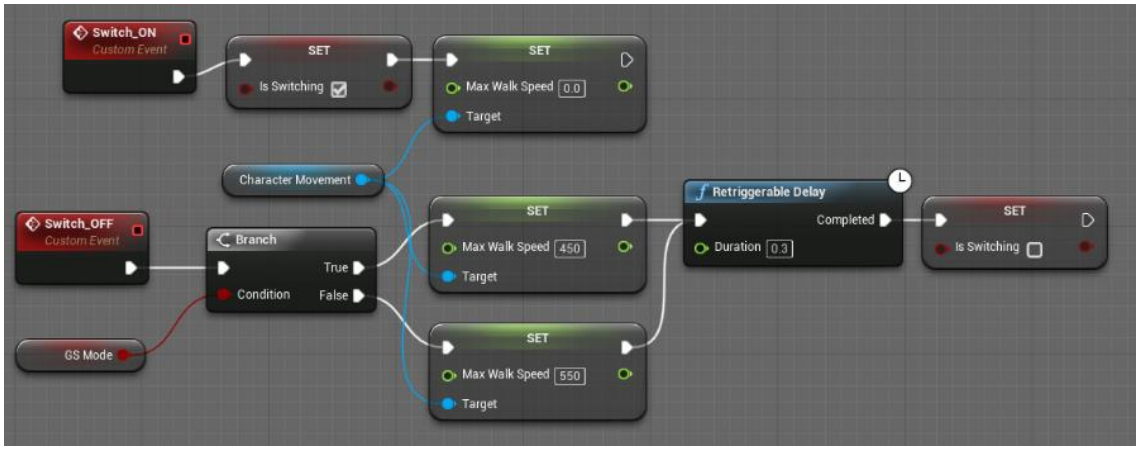
Στο επόμενο γράφημα γίνεται ο υπολογισμός της κίνησης του παίχτη :



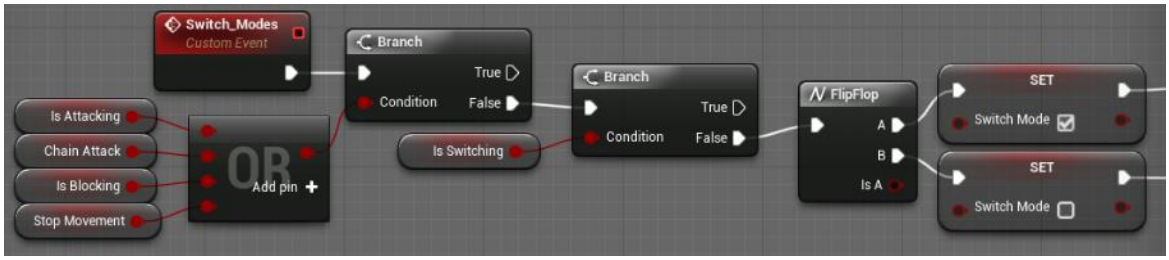
Το γεγονός InputAxis MoveForward ενεργοποιείται όποτε ο χρήστης μετακινεί μπροστά ή πίσω το αριστερό χειριστήριο ενώ το γεγονός InputAxis MoveRight ενεργοποιείται όποτε ο χρήστης το μετακινεί αριστερά και δεξιά. Αρχικά και για τα δύο γεγονότα γίνεται ένας έλεγχος αν ο παίχτης είναι ζωντανός για να ενεργοποιηθεί η κίνηση του. Έπειτα γίνεται ακόμα ένας έλεγχος με κάποια από τα ίδια κριτήρια των

προηγούμενων γραφημάτων. Η boolean μεταβλητή StopMovement χρησιμοποιείται για να σταματήσουμε την κίνηση του παίχτη όταν εκτελείται το animation για όταν ο παίκτης χρησιμοποιεί την ιδιότητα του να επαναφέρει πόντους ζωής. Η συνάρτηση Get Control Rotation δέχεται το διάνυσμα κίνησης του χαρακτήρα. Οι συναρτήσεις Break Rot και Make Rot δέχονται το διάνυσμα κίνησης του χαρακτήρα και το διαιρούν στους τρεις άξονες κίνησης. Έπειτα οι συναρτήσεις AddMovement Input δέχονται τα διανύσματα κίνησης στον άξονα Y για κίνηση μπρος-πίσω και στον άξονα X για κίνηση αριστερά-δεξιά.

Στη συνέχεια θα αναλυθεί η δημιουργία του συστήματος αλλαγής όπλων.

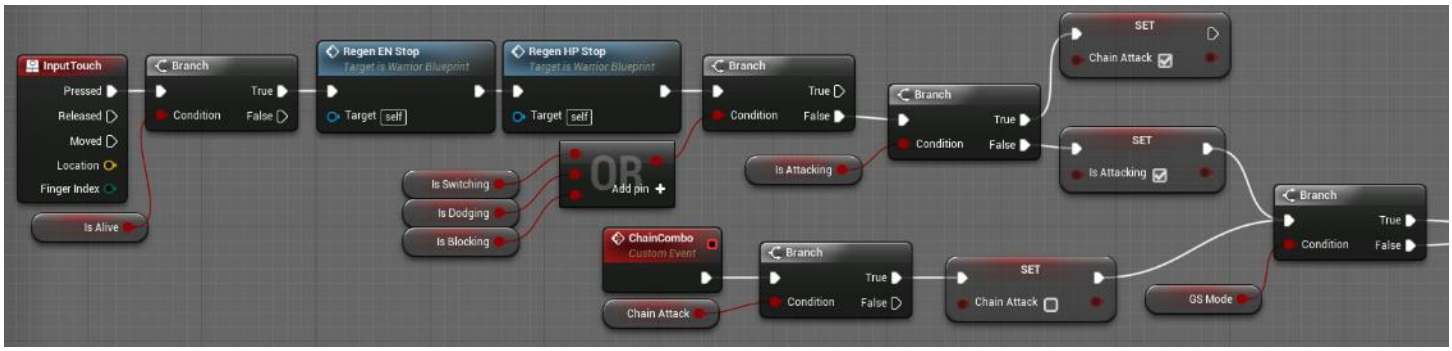


Αρχικά δημιουργήθηκαν δύο γεγονότα, ένα για όταν η αλλαγή γίνεται την παρούσα χρονική στιγμή και ένα για όταν έχει πραγματοποιηθεί η αλλαγή. Σε περίπτωση που γίνεται αλλαγή όπλων η μέγιστη ταχύτητα του παίχτη μηδενίζεται. Αν η αλλαγή έχει πραγματοποιηθεί και ο χρήστης χρησιμοποιεί την ασπίδα και το σπαθί τότε έχει την κανονική μέγιστη ταχύτητα τρεξίματος. Αλλιώς αν ο χρήστης χρησιμοποιεί το μονό μεγάλο σπαθί (GS_Mode->True) τότε η μέγιστη ταχύτητα τρεξίματος μειώνεται για να δώσει την εντύπωση στο χρήστη ότι το μονό σπαθί είναι πιο δύσκολο στη χρήση.

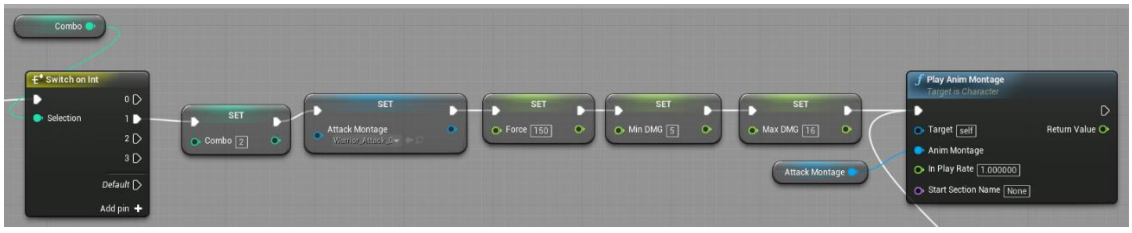


Συνεχίζοντας με το σύστημα αλλαγής όπλων, δημιουργήθηκε ένα ακόμα γεγονός (Switch_Modes) το οποίο θα εκτελείται όταν ο χρήστης πατήσει το κουμπί αλλαγής όπλων. Σε περίπτωση που ο παίκτης δεν επιτίθεται, χρησιμοποιεί την ασπίδα του ή αλλάζει όπλα την παρούσα χρονική στιγμή τότε πραγματοποιείται η αλλαγή όπλων.

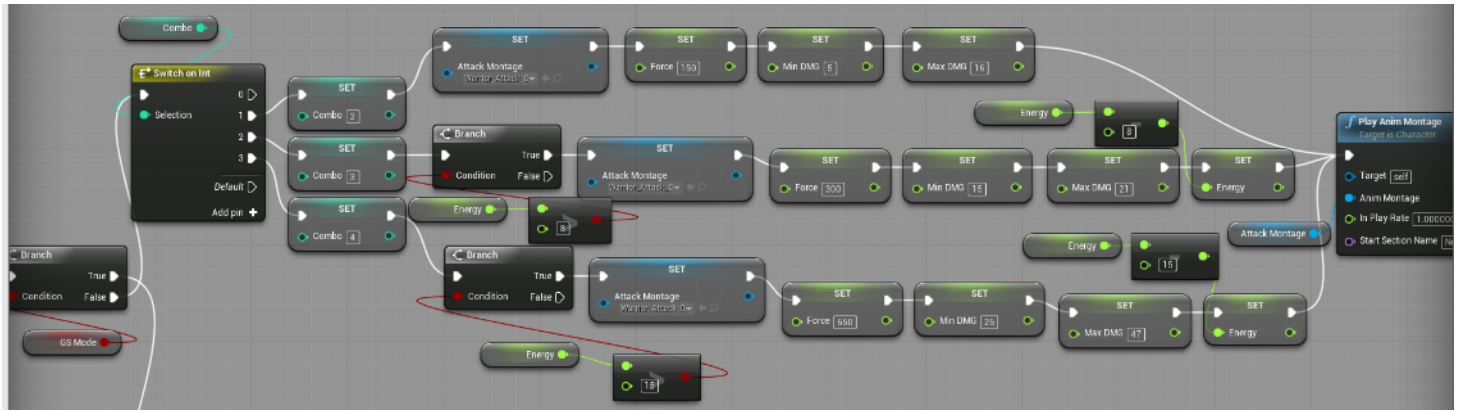
Επομένως, δημιουργούμε το σύστημα επίθεσης χωρίς όμως να γίνεται ακόμα ο υπολογισμός ζημιάς στον αντίπαλο.



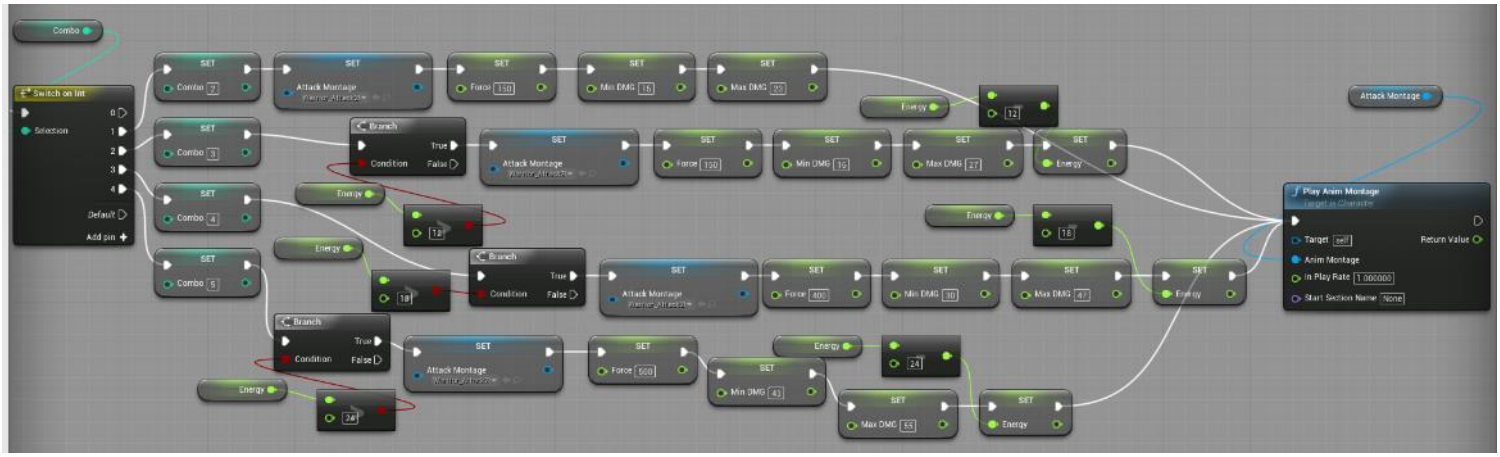
Για το σύστημα επιθέσεων δημιουργήθηκε ένα σύστημα combo δηλαδή ο παίκτης θα πραγματοποιεί διαδοχικές επιθέσεις μέχρι να σταματήσει τις επιθέσεις ο χρήστης ή μέχρι να μην είναι αρκετή η ενέργεια που έχει περισσέψει στον παίχτη. Την ώρα που ο παίχτης επιτίθεται σταματά η επαναφορά ζωής και ενέργειας και ξεκινά το σύστημα combo εκτός και αν είναι ήδη σε ισχύ. Αν ο χρήστης χρησιμοποιεί την ασπίδα και το σπαθί τότε ο παίχτης θα μπορεί να κάνει τρεις επιθέσεις σε διαδοχική σειρά. Στο παρακάτω γράφημα φαίνονται οι ρυθμίσεις της πρώτης επίθεσης :



Αν έχουν πραγματοποιηθεί τα προηγούμενα κριτήρια τότε αυξάνεται το combo, ώστε την επόμενη φορά να συνεχίσει με την επόμενη επίθεση. Έπειτα θέτουμε το Animation Montage της πρώτης επίθεσης. Μετά με το Force ρυθμίζουμε πόσο θα εκτοξευτεί ευθεία ο παίχτης όταν κάνει ένα βήμα την ώρα που εκτελείται το animation της επίθεσης. Μετά ρυθμίζουμε το εύρος ζημιάς που μπορεί να κάνει η συγκεκριμένη επίθεση. Τέλος, καλούμε μια συνάρτηση η οποία θα παίξει το animation montage που επιλέξαμε. Παρομοίως με την πρώτη επίθεση δημιουργήθηκαν και οι υπόλοιπες με τη διαφορά ότι η δεύτερη και η τρίτη επίθεση χρειάζονται και καταναλώνουν ενέργεια για να χρησιμοποιηθούν.



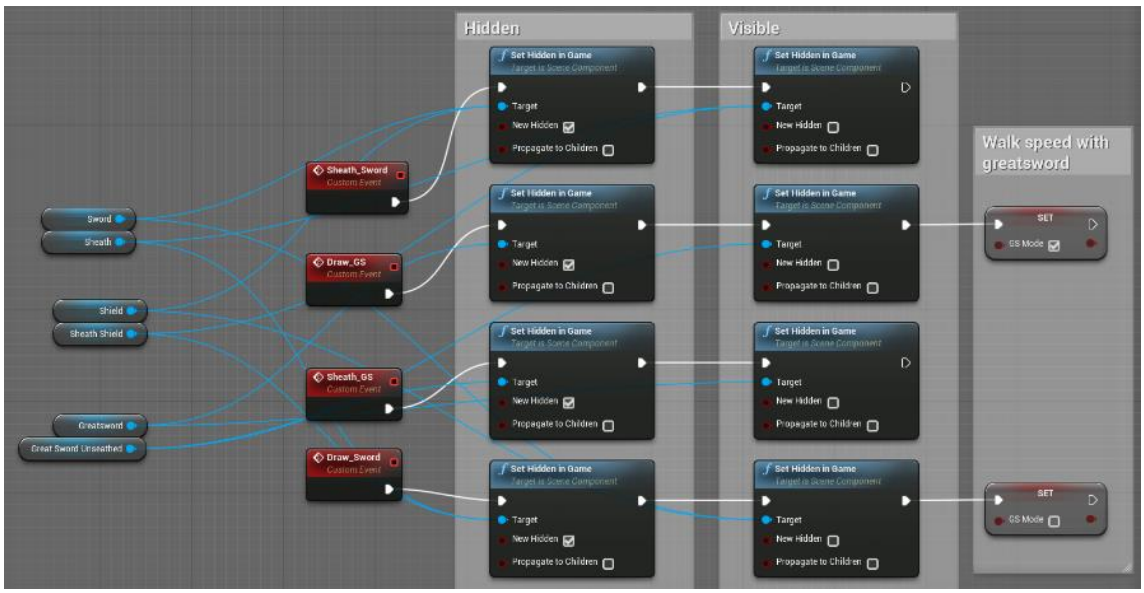
Παρομοίως δημιουργήθηκαν και οι τέσσερις επιθέσεις για το μονό σπαθί:



Σε περίπτωση που ο παίχτης δεν έχει αρκετή ενέργεια να εκτελέσει κάποια επίθεση τότε μπορεί να εκτελέσει μέχρι και την προηγούμενη

επίθεση. Αν ο παίκτης δεν έχει καθόλου ενέργεια τότε μπορεί να εκτελέσει μόνο την πρώτη επίθεση.

Όταν ο χαρακτήρας μας αλλάζει όπλα από ασπίδα και σπαθί σε μονό σπαθί θέλουμε να κρύβονται η ασπίδα και το σπαθί στα χέρια του και να εμφανίζονται το σπαθί και το σπαθί που έχουμε τοποθετήσει στο γοφό του. Παράλληλα, θέλουμε να κρύβεται το μεγάλο σπαθί που βρίσκεται στην πλάτη του και να εμφανίζεται αυτό που έχουμε τοποθετήσει στο χέρι του. Επίσης όταν ο χαρακτήρας μας αλλάζει όπλα από το μεγάλο σπαθί σε ασπίδα και σπαθί, θέλουμε να κρύβεται το μεγάλο σπαθί στο χέρι του και να εμφανίζεται το μεγάλο σπαθί που έχουμε τοποθετήσει στην πλάτη του. Παράλληλα, πρέπει να κρύβεται το σπαθί και η ασπίδα που βρίσκονται στο γοφό του και να εμφανίζονται το σπαθί στο δεξί του χέρι και η ασπίδα στο αριστερό του χέρι. Το παρακάτω γράφημα εκτελεί αυτές τις λειτουργίες.

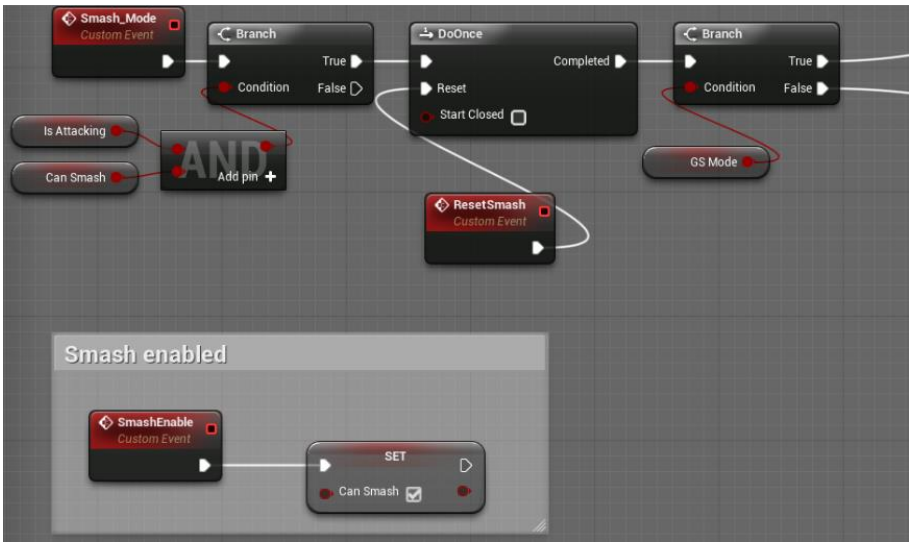


Τα γεγονότα που έχουν δημιουργηθεί θα καλούνται εντός του animation blueprint γιατί πρέπει σε κάθε animation που αφορά την αλλαγή όπλων να επιλεγούν τα σωστά γεγονότα με βάση το animation.

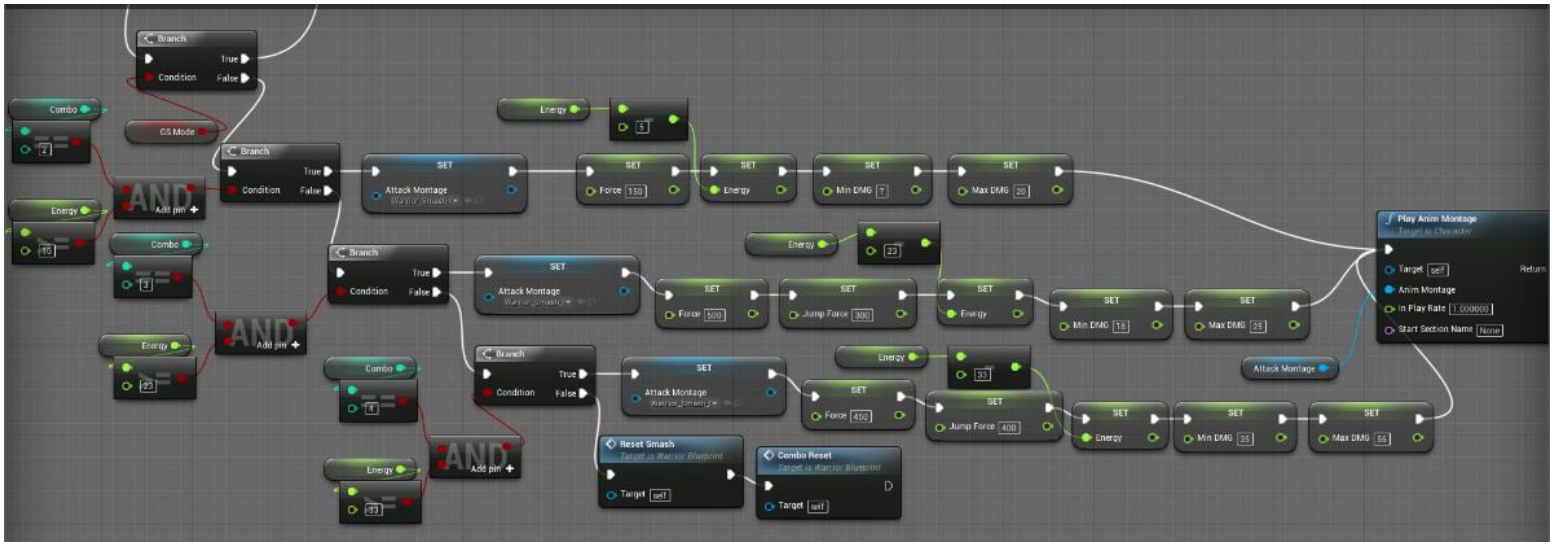
Κατόπιν δημιουργήθηκε ένα γεγονός το οποίο ‘ακυρώνει’ κάποιες ενέργειες του χρήστη όπως : combo, επίθεση, ειδικές επιθέσεις, δύναμη άλματος, κινήσεις αποφυγής επιθέσεων και έπειτα ξεκινάει πάλι τις

συναρτήσεις που επαναφέρουν πόντους ζωής και ενέργειας. Ένα τέτοιο γεγονός χρειάζεται σε πολλά σημεία του blueprint.

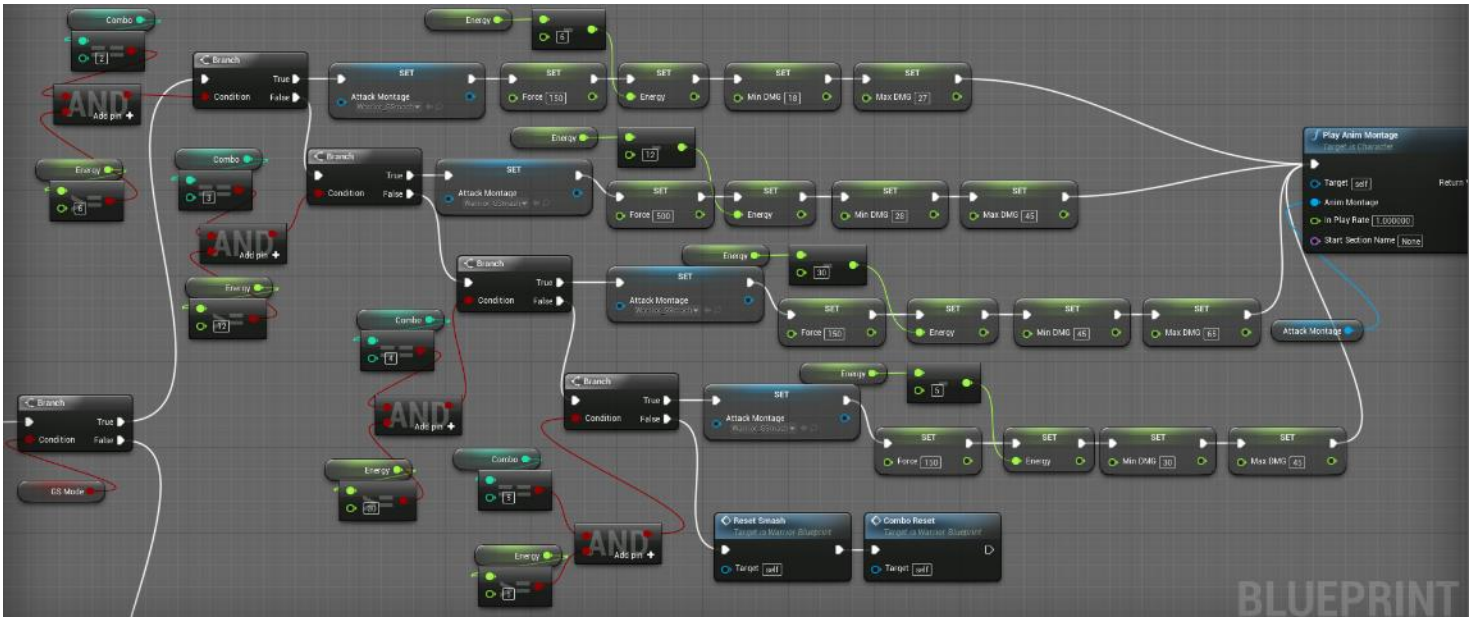
Έπειτα δημιουργήθηκε το σύστημα ειδικών επιθέσεων στο οποίο αν πραγματοποιούνται κάποιες προϋποθέσεις που θα αναλυθούν και ο παίχτης μας επιτίθεται τότε, όταν ο χρήστης πατήσει το κουμπί ειδικής επίθεσης κατά τη διάρκεια μιας κανονικής επίθεσης θέλουμε να εκτελείτε μια συγκεκριμένη επίθεση η οποία κάνει μεγαλύτερη ζημιά στον αντίπαλο αλλά ακυρώνει το σύστημα combo. Οπότε, για κάθε κανονική επίθεση του παίχτη υπάρχει και μια ειδική επίθεση, άρα όταν ο πολεμιστής χρησιμοποιεί την ασπίδα και το σπαθί θα μπορεί να χρησιμοποιήσει τρεις ειδικές επιθέσεις και όταν χρησιμοποιεί το μεγάλο σπαθί θα μπορεί να χρησιμοποιήσει τέσσερις ειδικές επιθέσεις. Αρχικά δημιουργούνται δύο γεγονότα, ένα (SmashEnable) που θα καλείται εντός του animation blueprint διότι θέλουμε για κάθε animation επίθεσης ο παίχτης μας να μπορεί σε συγκεκριμένα χρονικά σημεία να εκτελέσει μια ειδική επίθεση έτσι ώστε να υπάρχει μια ωραία οπτική μετάβαση από ένα animation επίθεσης στο αντίστοιχο animation ειδικής επίθεσης και ένα γεγονός (Smash_Mode) το οποίο θα ενεργοποιεί το σύστημα ειδικής επίθεσης όταν ο χρήστης πατήσει το κουμπί ειδικής επίθεσης.



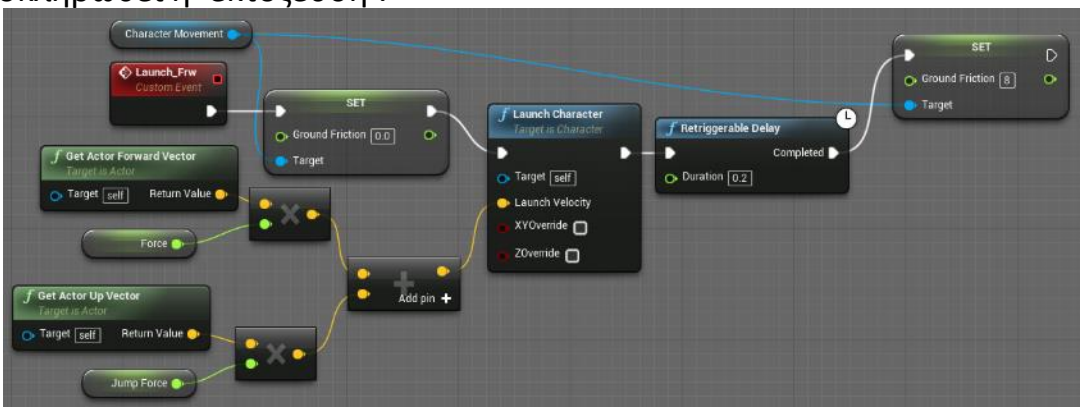
Η συνέχεια του συστήματος ειδικών επιθέσεων μοιάζει αρκετά με το σύστημα απλών επιθέσεων δηλαδή, για κάθε ειδική επίθεση θέλουμε να εκτελούνται τα εξής : να καλούμε ένα διαφορετικό animation montage για κάθε ειδική επίθεση, να θέτουμε πόσο θα ‘εκτοξευτεί’ ευθεία ο παίκτης μας σε κάθε animation, να αφαιρείται η ενέργεια που χρειάζεται να καταναλώσει η επίθεση από την συνολική ενέργεια, την ελάχιστη και τη μέγιστη ζημιά που μπορεί να κάνει και τελικά να παίζει το animation montage. Επίσης, για κάθε ειδική επίθεση θέλουμε να υπάρχει ένας έλεγχος αν ο παίχτης έχει αρκετή ενέργεια για να εκτελέσει την επίθεση και σε πια επίθεση ή combo βρίσκεται ο παίχτης έτσι ώστε να εκτελέσει την κατάλληλη επίθεση ανάλογα με την τιμή του combo. Στο τέλος του γραφήματος πρέπει να προστεθούν και τα κατάλληλα γεγονότα έτσι ώστε όταν ολοκληρώνεται η ειδική επίθεση να ακυρώνεται και το combo του παίχτη.



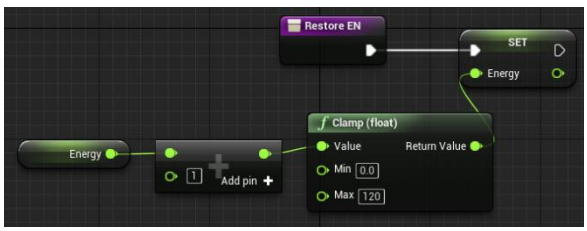
Το παραπάνω γράφημα είναι το σύστημα ειδικών επιθέσεων για την ασπίδα και το σπαθί. Σε αυτό το γράφημα επειδή σε κάποια animation ο παίχτης κάνει ένα άλμα έχει προστεθεί και μια μεταβλητή που θέτει πόσο ψηλό θα είναι το άλμα του παίχτη μας. Στο παρακάτω γράφημα είναι το σύστημα ειδικών επιθέσεων για το μεγάλο σπαθί.

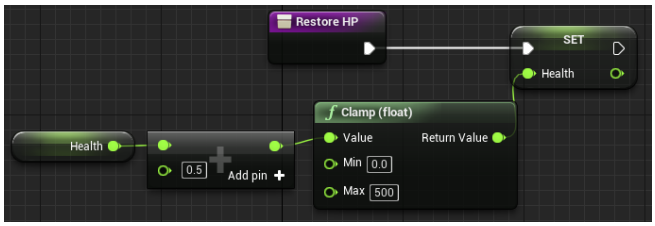


Ύστερα δημιουργήθηκε ένα γεγονός (Launch_Frw), το οποίο θα καλείται εντός του animation blueprint του πολεμιστή, διότι θέλουμε για κάθε animation επίθεσης ο παίχτης μας να μπορεί σε συγκεκριμένα χρονικά σημεία να ‘εκτοξεύεται’ μπροστά ή και να κάνει ένα άλμα για κάθε βήμα που κάνει εντός των animation. Έχουμε ήδη θέσει το πόσο μπροστά θα εκτοξεύεται ή και θα κάνει ένα άλμα ο παίχτης μας για κάθε animation εντός των συστημάτων επίθεσης και ειδικής επίθεσης. Αρχικά παίρνουμε το διάνυσμα μπροστινής κίνησης (Get Actor Forward Vector) και το διάνυσμα πάνω κίνησης του παίχτη μας (Get Actor Up Vector) και τα πολλαπλασιάζουμε με τις τιμές που έχουμε θέσει για την ‘εκτόξευση’ και το άλμα αντίστοιχα εντός των συστημάτων επίθεσης. Στην συνέχεια τα προσθέτουμε και περνάμε το τελικό διάνυσμα κίνησης στη συνάρτηση Launch Character, η οποία ‘εκτοξεύει’ τον παίχτη. Τέλος, προσθέτουμε τριβή στο διάνυσμα κίνησης του παίχτη για να ολοκληρωθεί η ‘εκτόξευση’.

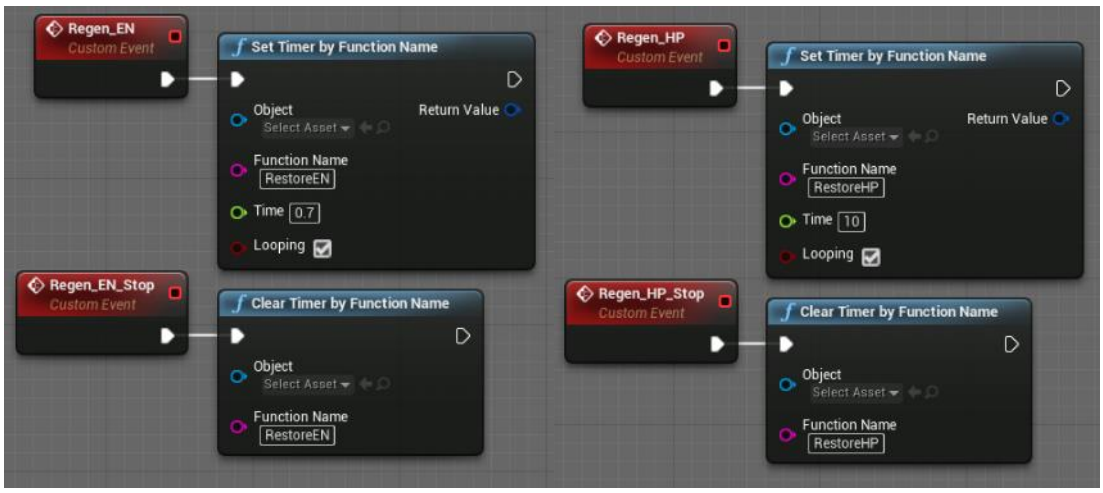


Έπειτα δημιουργούμε δύο συναρτήσεις οι οποίες θα επαναφέρουν πόντους ζωής και ενέργειας στο παίχτη ανά τακτά χρονικά διαστήματα που ορίζουμε εμείς. Εντός των συναρτήσεων θέτουμε την ελάχιστη και τη μέγιστη τιμή που μπορεί να έχει η ζωή και η ενέργεια. Έπειτα προσθέτουμε ένα πόντο ενέργειας και μισό πόντο ζωής και επιστρέφουμε το τελικό αποτέλεσμα στους συνολικούς πόντους ζωής ή ενέργειας.

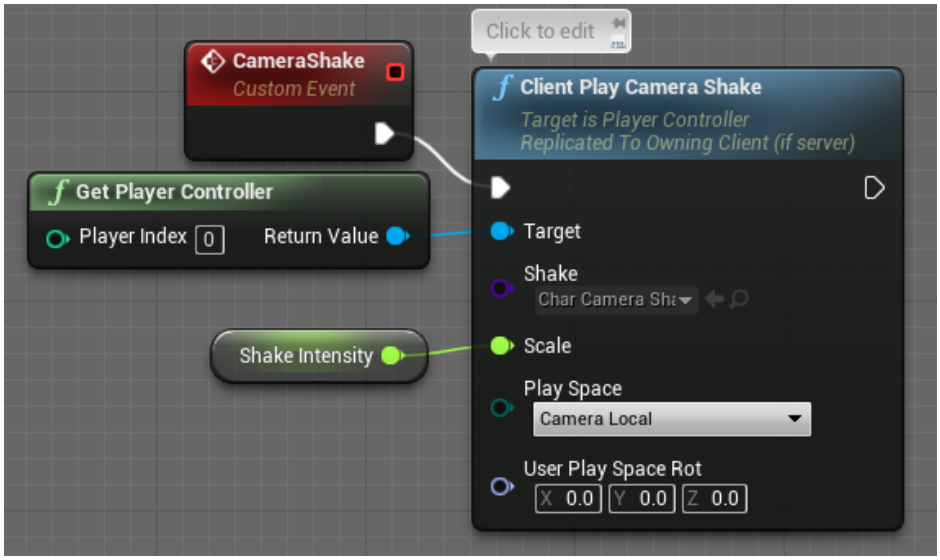




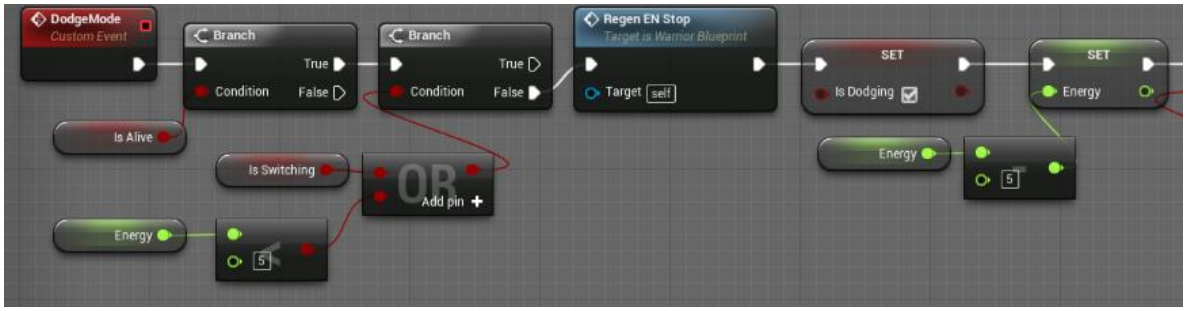
Μετά στο warrior_blueprint δημιουργούμε τέσσερα γεγονότα για να ξεκινάνε και να σταματάνε την επαναφορά ζωής και ενέργειας. Τα γεγονότα αυτά χρησιμοποιούνται σε αρκετά σημεία του warrior_blueprint. Έπειτα καλούμε τις συναρτήσεις που δημιουργήσαμε μέσα σε συναρτήσεις που τις εκτελούν ανά τακτά χρονικά διαστήματα που ορίζουμε (Set Timer By Function Name) ή σταματάνε να εκτελούν τις συναρτήσεις επαναφοράς ενέργειας/ζωής (Clear Time By Function Name).



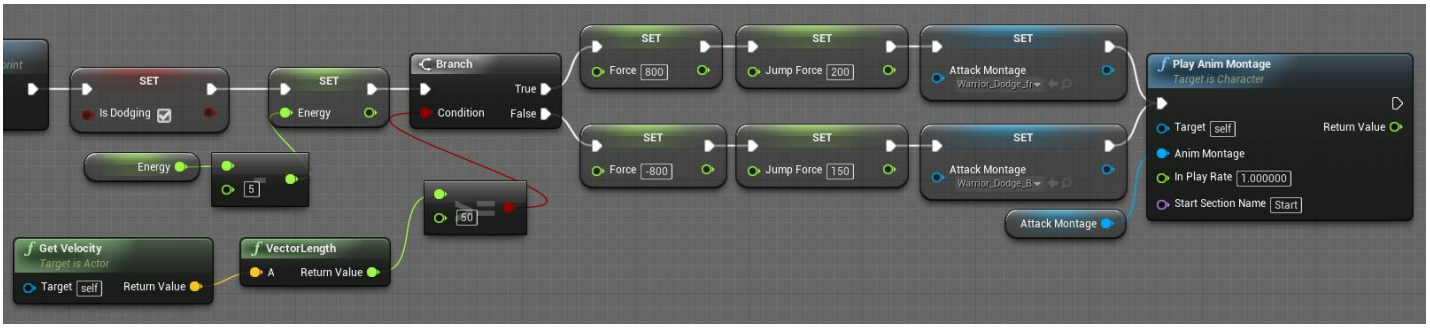
Στη συνέχεια δημιουργήθηκε ένα γεγονός (CameraShake), το οποίο θα καλείται εντός του animation blueprint του πολεμιστή διότι θέλουμε για κάθε animation επίθεσης ο παίχτης μας όποτε χτυπάει με επιτυχία τον αντίπαλο να κουνιέται ελάχιστα η κάμερα ώστε να υποδεικνύει στο χρήστη ότι η επίθεση ήταν επιτυχής. Αυτό γίνεται πολύ εύκολα με την συνάρτηση Client Play Camera Shake, η οποία δέχεται τη συνάρτηση που διαχειρίζεται μια από τις ριζικές κλάσεις του πολεμιστή, δηλαδή την κλάση Player Controller και έπειτα με μια δικιά μας μεταβλητή (Shake Intensity) καθορίζεται το πόσο θα κουνηθεί η κάμερα.



Στην παρακάτω εικόνα είναι η υλοποίηση του συστήματος αποφυγής επιθέσεων. Για το σύστημα αποφυγής επιθέσεων χρειάζεται αρχικά να δημιουργήσουμε ένα δικό μας γεγονός (DodgeMode), το οποίο θα πραγματοποιείται όταν ο χρήστης πατήσει το κουμπί για την αποφυγή επιθέσεων. Μετά γίνεται ένας έλεγχος αν ο παίχτης μας είναι ζωντανός. Επίσης, πρέπει να μην κάνει αλλαγή όπλων και να έχει αρκετή ενέργεια για να πραγματοποιήσει μια αποφυγή. Εφόσον ο παίκτης μας θα πραγματοποιεί μια λειτουργία που καταναλώνει ενέργεια, πρέπει για όσο κρατάει η λειτουργία να μην επαναφέρεται η ενέργεια του. Έπειτα με τη μεταβλητή IsDodging δηλώνουμε ότι ο παίχτης μας την παρούσα στιγμή αποφεύγει μια επίθεση.



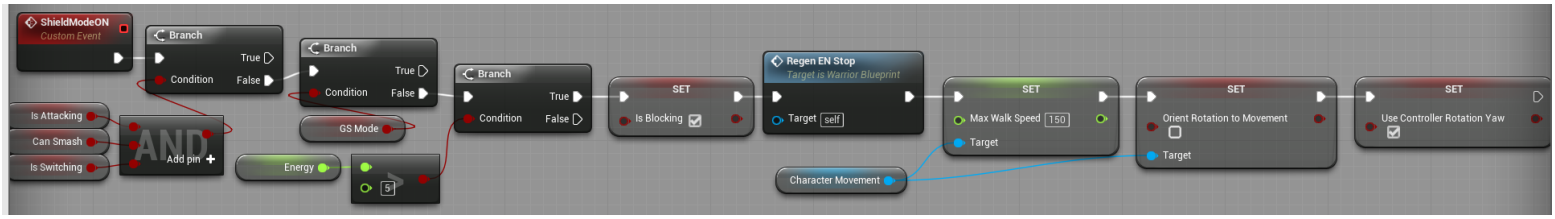
Επομένως, πρέπει να θέσουμε ποια θα είναι η προκαθορισμένη κατεύθυνση που ο παίχτης θα κάνει την αποφυγή όταν είναι στάσιμος ή κινείται με πολύ μικρή ταχύτητα. Αυτό επιτυγχάνεται παίρνοντας την ταχύτητα του παίκτη (Get Velocity). Τέλος, ρυθμίζουμε πόσο θέλουμε ο παίχτης μας να εκτοξεύεται μπροστά ή πίσω, προσθέτουμε ένα μικρό άλμα και έπειτα καλούμε τα κατάλληλα animation για τη μπροστινή και πίσω κίνηση και τα αναπαράγουμε.



Το αποτέλεσμα είναι ότι όποτε ο παίχτης μας είναι στάσιμος ή τρέχει με πού χαμηλή ταχύτητα τότε θα κάνει μια αποφυγή προς τα πίσω. Αλλιώς θα κάνει μια αποφυγή ευθεία από την κατεύθυνση που θα βρίσκεται ο παίχτης μας.

Έπειτα θα αναλυθεί το γράφημα για όταν ο παίχτης μας χρησιμοποιεί την ασπίδα του για να αμυνθεί. Αρχικά δημιουργούμε δυο γεγονότα, ένα για όταν ο χρήστης ενεργοποιεί την ασπίδα πατώντας συνεχόμενα το κουμπί της ασπίδας (ShieldModeON) και ένα όταν ο χρήστης σταματήσει να πατάει το κουμπί της ασπίδας και απενεργοποιείται η ασπίδα του παίχτη (ShieldModeOff). Για όταν ο χρήστης ενεργοποιεί την ασπίδα πρέπει αρχικά ο παίχτης να μην επιτίθεται, να μην κάνει αλλαγή όπλων και να μην χρησιμοποιεί το μεγάλο σπαθί. Επίσης πρέπει να έχει

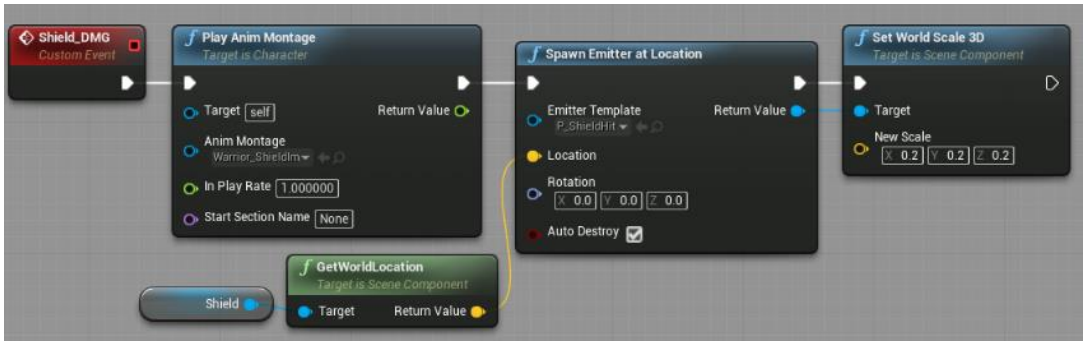
αρκετή ενέργεια ώστε να χρησιμοποιήσει την ασπίδα του. Έπειτα με την μεταβλητή IsBlocking δηλώνουμε ότι ο παίχτης μας χρησιμοποιεί την ασπίδα του. Εφόσον πλέον ο παίκτης μας θα χρησιμοποιεί την ασπίδα του πρέπει για όσο την κρατάει να μην επαναφέρεται η ενέργεια του. Επίσης θέλουμε για όσο ο παίχτης χρησιμοποιεί την ασπίδα να περπατάει οπότε μειώνουμε τη μέγιστη ταχύτητα τρεξίματος του παίχτη. Τέλος, θέτοντας τις μεταβλητές Orient Rotation to Movement σε ψευδή και Use Controller Rotation Yaw σε αληθή ο παίκτης μας θα περιστρέφεται μαζί με την κάμερα όταν χρησιμοποιεί την ασπίδα.



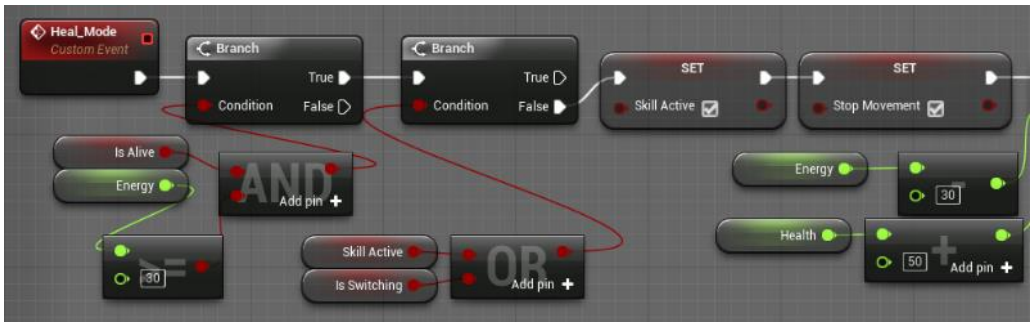
Στο γεγονός που απενεργοποιεί την ασπίδα θέλουμε να επιστρέφει η κανονική μέγιστη ταχύτητα του παίχτη, η περιστροφή της κάμερας να είναι πάλι ανεξάρτητη του παίκτη και τέλος να ξεκινά πάλι η επαναφορά ενέργειας.

Ο υπολογισμός του πόσο ζημιά θα απορροφάει η ασπίδα όταν επιτίθενται οι αντίπαλοι στον παίχτη πρέπει θα γίνει στο γράφημα που αφορά την συνολική ζημιά που μπορούν να κάνουν οι αντίπαλοι στον παίχτη.

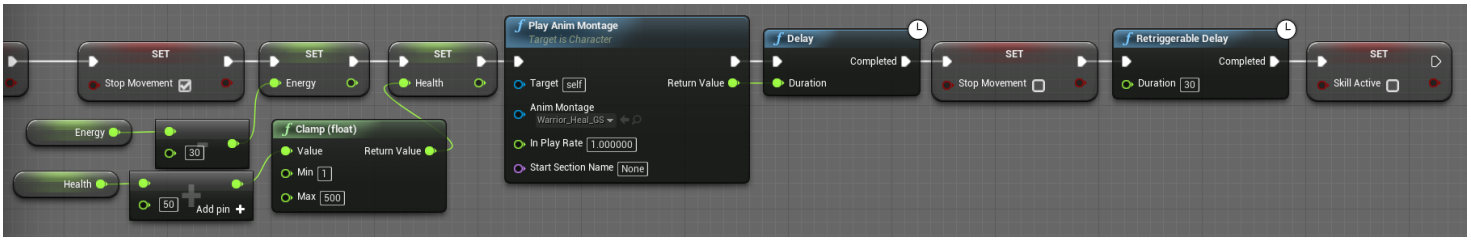
Στο επόμενο διάγραμμα δημιουργείται ένα νέο γεγονός (Shield_DMG), το οποίο θα ενεργοποιείται όποτε ο παίχτης μας δέχεται επίθεση ενώ χρησιμοποιεί την ασπίδα του. Για το συγκεκριμένο γεγονός θέλουμε να παίζει κάθε φόρα ένα animation που δείχνει τον παίχτη μας να αποκρούει την επίθεση του αντιπάλου. Έπειτα στη θέση που βρίσκεται η ασπίδα θέλουμε να δημιουργείται ένα σύντομο σύστημα σωματιδίων, το οποίο θα υποδεικνύει στο χρήστη ότι η επίθεση αποκρούστηκε επιτυχώς. Έπειτα τροποποιούμε το μέγεθος που θα έχει το σωματίδιο.



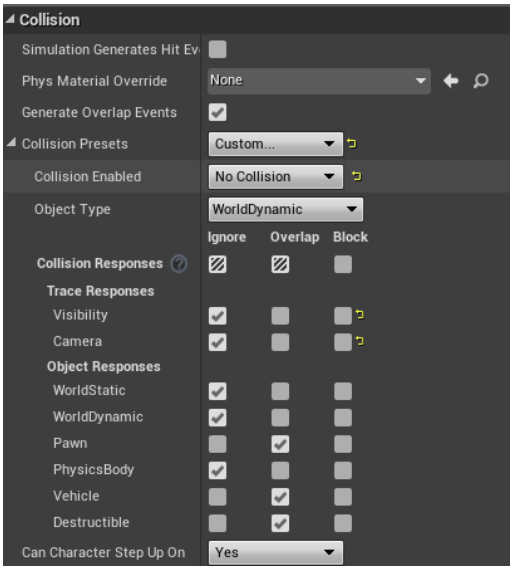
Στη συνέχεια δημιουργήθηκε το σύστημα θεραπείας του παίχτη. Για να ενεργοποιηθεί το σύστημα ο παίχτης μας πρέπει να είναι ζωντανός και να έχει αρκετή ενέργεια για να θεραπευτεί. Επίσης πρέπει να μην αλλάζει όπλα και να έχει περάσει ο χρόνος των 30 δευτερολέπτων που χρειάζεται για να επαναχρησιμοποιηθεί η ιδιότητα. Έπειτα με την μεταβλητή Skill_Active δηλώνουμε ότι ο παίχτης χρησιμοποιεί την ιδιότητα του να γιατρευτεί και με τη μεταβλητή Stop_Movement σταματάει η κίνηση του παίχτη για τη διάρκεια της ιδιότητας.



Κατόπιν αφαιρούνται 30 πόντοι ενέργειας και προστίθενται 50 πόντοι ζωής. Έπειτα καλούμε το κατάλληλο animation montage να παίξει, ενεργοποιούμε και πάλι την κίνηση του παίχτη και δημιουργούμε μια καθυστέρηση των 30 δευτερολέπτων μέχρι να μπορεί η ιδιότητα να χρησιμοποιηθεί.

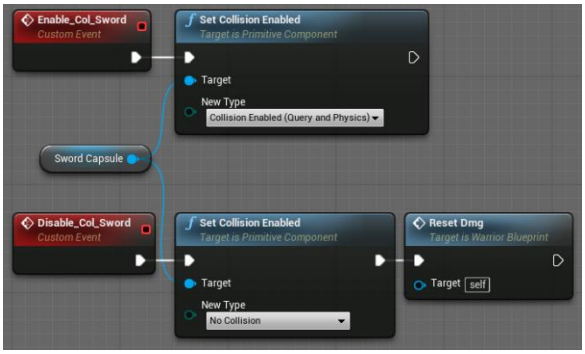


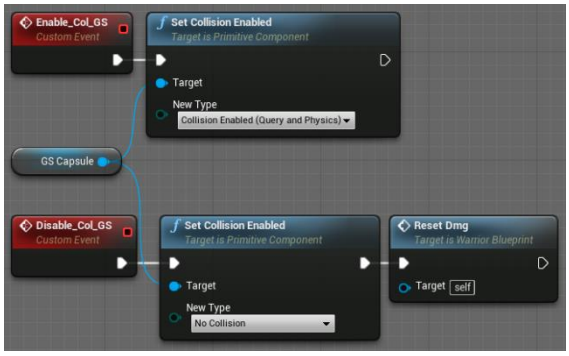
Το μόνο που έμεινε για να ολοκληρωθεί το warrior_blueprint είναι ο υπολογισμός ζημιάς που κάνει και δέχεται ο παίχτης μας. Για να υπολογίσουμε τη ζημιά που κάνει ο παίχτης θα χρησιμοποιήσουμε τις κάψουλες που προσθέσαμε στην αρχή της υπό ενότητας. Με αυτό τον τρόπο όποτε ο παίχτης μας πραγματοποιεί μια επίθεση και μία από τις κάψουλες που τοποθετήσαμε ακουμπάει τον αντίπαλο τότε θα ενεργοποιείται το σύστημα σύγκρουσης (Collision) της συγκεκριμένης κάψουλας. Όταν ο παίχτης μας δεν πραγματοποιεί μια επίθεση το σύστημα σύγκρουσης της κάψουλας θα απενεργοποιείται.

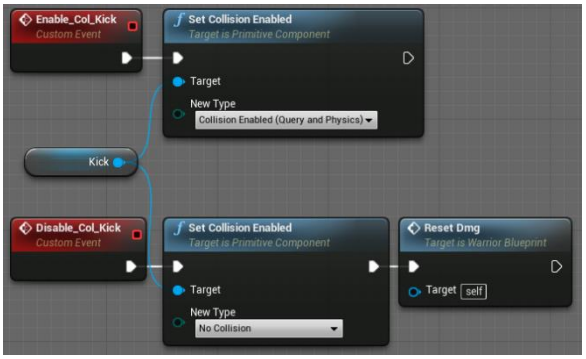


Αρχικά πρέπει να αλλάξουμε τις ρυθμίσεις σύγκρουσης και στις τρεις κάψουλες ώστε να απενεργοποιείται το σύστημα σύγκρουσης εκτός και αν έρχονται σε επαφή με ένα αντικείμενο της κλάσης Pawn, δηλαδή τις ριζικής κλάσης όλων των αντιπάλων.

Έπειτα πρέπει να δημιουργήσουμε 2 γεγονότα για κάθε κάψουλα τα οποία θα ενεργοποιούν ή απενεργοποιούν αντίστοιχα το σύστημα σύγκρουσης.

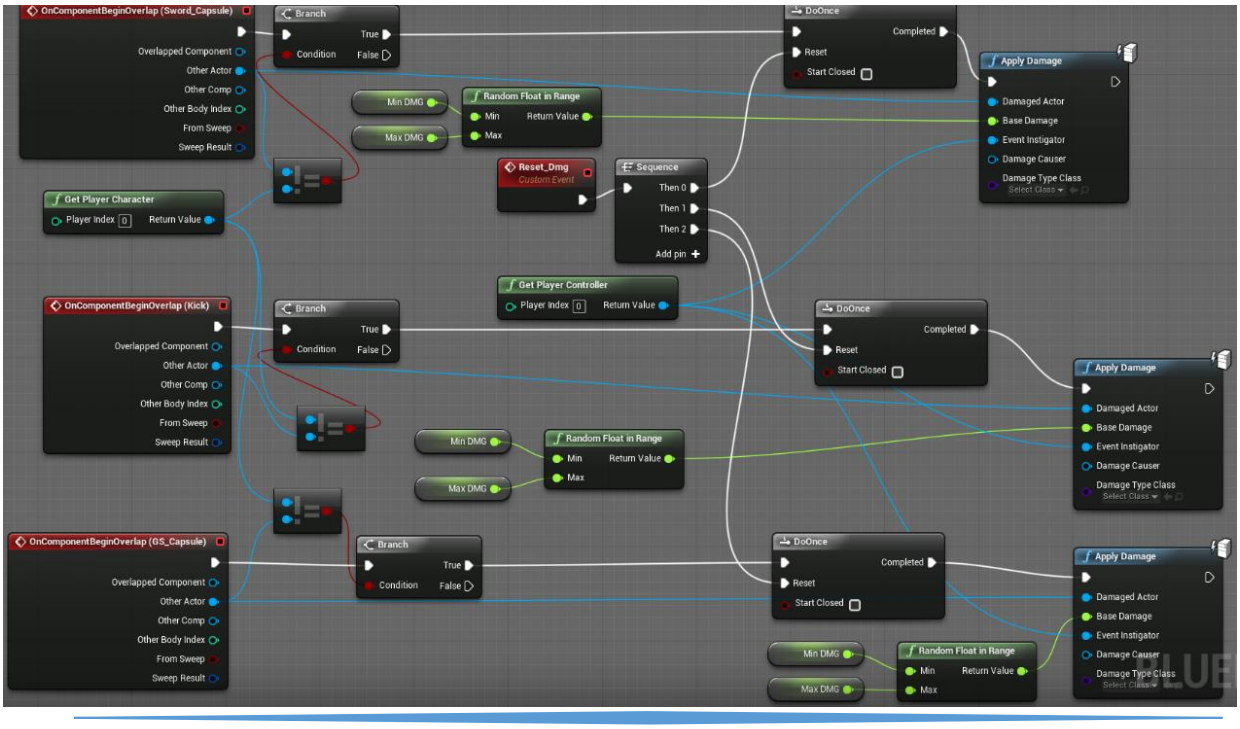




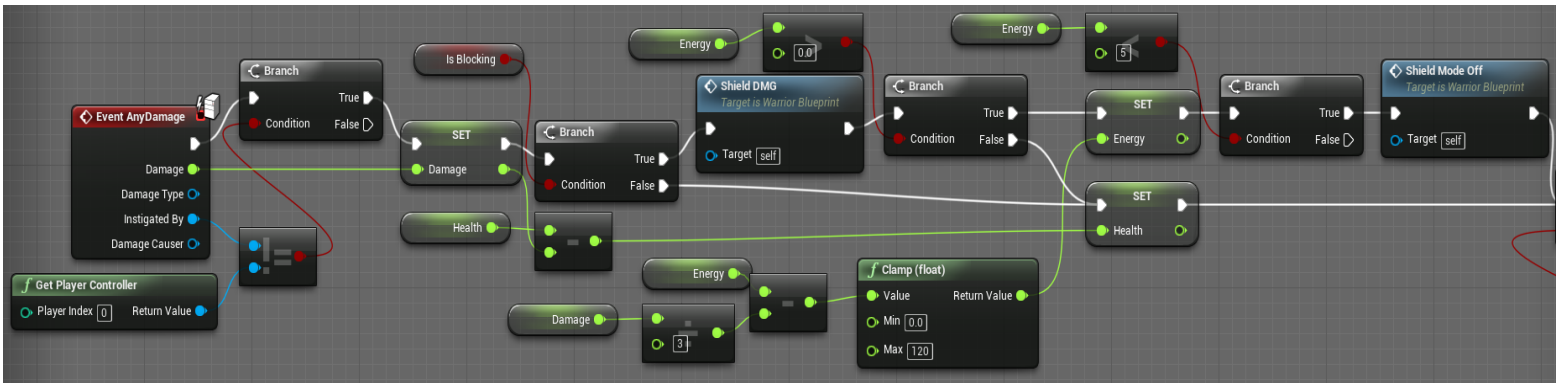


Τα γεγονότα αυτά θα καλούνται εντός του animation blueprint του πολεμιστή επειδή θέλουμε σε κάθε animation επίθεσης του παίχτη μας να ενεργοποιείται το σύστημα σύγκρουσης σε συγκεκριμένα χρονικά σημεία για κάθε animation. Αλλιώς μπορεί ο παίχτης μας σε κάποια animation να ακουμπάει πολλαπλές φορές τον αντίπαλο και έτσι να του προκαλείται ζημιά πολλές φορές ενώ θέλουμε σε κάθε animation να του προκαλείται μόνο μια φορά ζημιά.

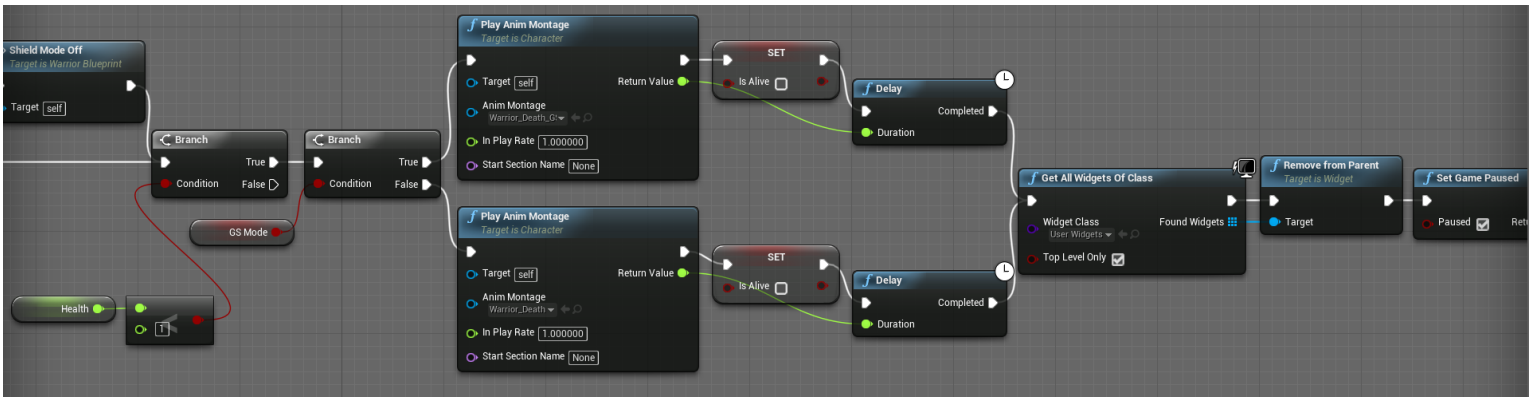
Επομένως δημιουργήθηκε το γράφημα υπολογισμού ζημιάς που προκαλεί ο παίκτης στον αντίπαλο. Για τον υπολογισμό χρησιμοποιήθηκαν τρία γεγονότα OnComponentBeginOverlap για την κάθε κάψουλα τα οποία ενεργοποιούνται όταν υπάρχει επαφή ενός actor με μια από τις κάψουλες. Αρχικά γίνεται ένας έλεγχος ώστε ο actor να μην είναι ο παίχτης μας για λόγους σωστής πρακτικής. Έπειτα χρησιμοποιήθηκε η μακροεντολή DoOnce έτσι ώστε ο παίχτης μας να προκαλεί μόνο μια φορά ζημιά στον αντίπαλο κάθε φορά που πραγματοποιεί μια επίθεση. Στη συνέχεια χρησιμοποιείται η συνάρτηση ApplyDamage για να προκαλέσουμε τη ζημιά στον actor που θα βρίσκεται μπροστά στον παίχτη. Η ελάχιστη και μέγιστη ζημιά που μπορεί να προκαλέσει κάθε επίθεση έχει οριστεί στα γραφήματα επιθέσεων και ειδικών επιθέσεων.



Κατόπιν δημιουργήθηκε το διάγραμμα υπολογισμού για τη ζημιά που δέχεται ο παίχτης. Χρησιμοποιώντας το γεγονός Event AnyDamage μπορούμε να πάρουμε τη ζημιά που δέχτηκε ο παίχτης μας και έπειτα να την αποθηκεύσουμε σε μια μεταβλητή Damage. Παράλληλα γίνεται ένας έλεγχος για το αν ο actor που δέχεται ζημιά είναι ο παίχτης μας. Στη συνέχεια γίνεται ένας έλεγχος για το αν ο παίχτης μας χρησιμοποιεί την ασπίδα του. Αν τη χρησιμοποιεί τότε δεν χάνει πόντους ζωής αλλά χάνει το $\frac{1}{3}$ της ζημιάς που δέχτηκε σε πόντους ενέργειας. Αν δεν χρησιμοποιεί την ασπίδα του τότε χάνει τόσους πόντους ζωής όσο ήταν η ζημιά που δέχτηκε. Επίσης στην περίπτωση που ο παίχτης χρησιμοποιεί την ασπίδα του τότε γίνεται ακόμα ένας έλεγχος για το αν έχει αρκετή ενέργεια για να μπορεί να αποκρούσει μια επίθεση. Αν η διαθέσιμη ενέργεια δεν είναι επαρκής για να αποκρούσει την επίθεση τότε αυτόματα ο παίχτης σταματά να χρησιμοποιεί την ασπίδα του.

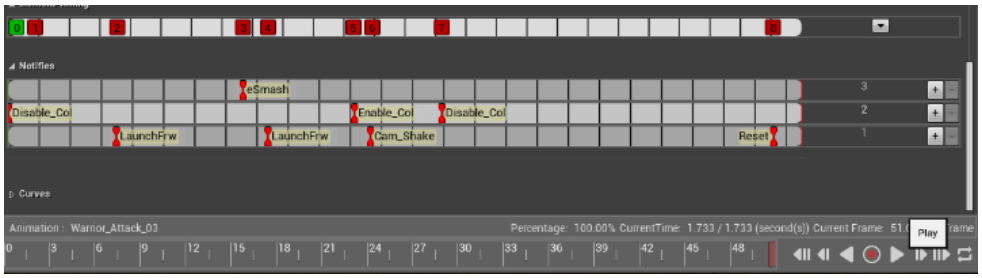


Στη συνέχεια ελέγχουμε αν οι πόντοι ζωής του παίχτη μας έχουν μηδενιστεί έτσι ώστε να μπουν σε λειτουργία τα animation θανάτου του παίχτη. Έπειτα γίνεται ακόμα ένας έλεγχος για το αν ο παίχτης χρησιμοποιεί ασπίδα και σπαθί ή το μονό σπαθί έτσι ώστε να παιχτεί το ανάλογο animation montage θανάτου. Κατά την διάρκεια που παίζει ένα εκ των δύο animation θανάτου θέτουμε τη μεταβλητή IsAlive σε ψευδή έτσι ώστε να σταματήσει η κίνηση και οποιαδήποτε άλλη λειτουργία του παίχτη. Κατόπιν καλούμε το HUD του παίχτη και το αφαιρούμε από την οθόνη. Τέλος καλούμε και ενεργοποιούμε τη συνάρτηση Set Game Paused η οποία παγώνει τελείως το παιχνίδι.



4.2.2 Το Animation Blueprint του Κύριου Χαρακτήρα

Πριν ξεκινήσουν οι αλλαγές στο animation blueprint του παίχτη πρέπει πρώτα όπως αναφέρθηκε και στις υπό-ενότητες 3.4 και 4.2.1 να προσθέσουμε κάποια σημεία ή ειδοποιήσεις στο χρονοδιάγραμμα ορισμένων animation montage του χαρακτήρα. Κάθε φορά που ο παίχτης μας εκτελεί ένα animation και φτάνει σε ένα σημείο του χρόνου όπου υπάρχει μια ειδοποίηση τότε θα εκτελείται το ανάλογο γεγονός που έχουμε δημιουργήσει εντός του κανονικού blueprint του παίχτη. Το animation blueprint χρησιμοποιείται για να κάνει τη σύνδεση μεταξύ των ειδοποιήσεων και των γεγονότων.

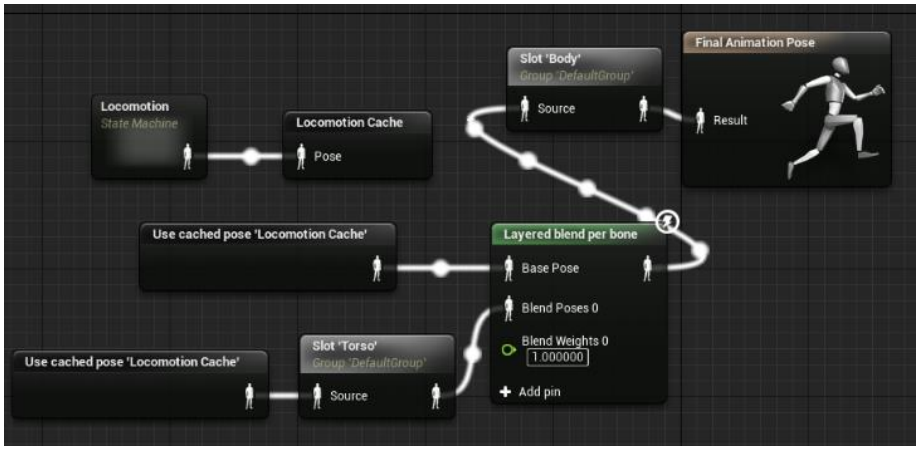


Χρονοδιάγραμμα animation με ειδοποιήσεις.

Εφόσον έχουν δημιουργηθεί και τοποθετηθεί στα κατάλληλα σημεία οι ειδοποιήσεις, μπορούμε να τις καλέσουμε στο animation blueprint και να τις συνδέσουμε με τα κατάλληλα γεγονότα από το warrior_blueprint.

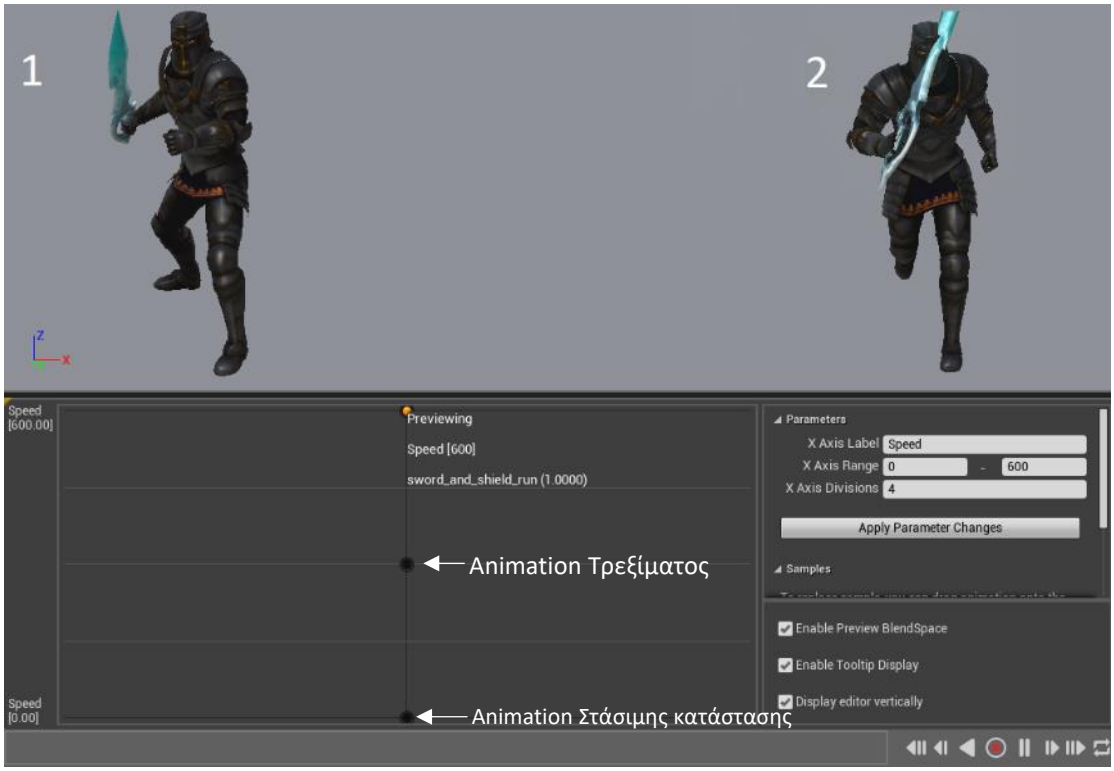
Στη συνεχεία μεταβαίνουμε στο animation blueprint του παίχτη και στο γράφο AnimGraph. Σε αυτόν τον γράφο δημιουργούνται οι στάσεις που

θα χρησιμοποιεί ο παίχτης δηλαδή, η στάσιμη στάση με ασπίδα και σπαθί και η μετάβαση σε τρέξιμο, η στάσιμη στάση με το μονό σπαθί με τη μετάβαση σε τρέξιμο και η στάση όταν χρησιμοποιεί την ασπίδα. Οι παραπάνω μεταβάσεις θα γίνουν εντός της μηχανής κατάστασης του παίχτη (Locomotion). Τη μηχανή κατάστασης την αποθηκεύουμε σε μια προσωρινή μνήμη (Locomotion Cache) για να μπορεί να χρησιμοποιηθεί σε διαφορετικά σημεία του γράφου. Στη συνέχεια δημιουργούμε δύο θυρίδες (Slot) οι οποίες κρατάνε animation montage και χρησιμοποιούνται ώστε να μπορεί ο παίχτης μας την ώρα του παιχνιδιού να μεταβεί από ένα ή περισσότερα montage που χρησιμοποιούν μια θυρίδα σε ένα διαφορετικό montage που χρησιμοποιεί διαφορετική θυρίδα. Η ανάθεση των θυρίδων γίνεται εντός των animation montage. Στη συγκεκριμένη περίπτωση η θυρίδα body κρατάει όλα τα animation montage πλην του animation montage που δείχνει τον παίχτη μας να αποκρούει μια επίθεση του αντιπάλου με την ασπίδα του. Με αυτό τον τρόπο όταν ο παίχτης χρησιμοποιεί την ασπίδα του αν δεχτεί επίθεση από αντίπαλο τότε ο παίχτης θα σταματήσει το animation της θυρίδας Body θα μεταβεί στο animation της θυρίδας Torso και μετά πάλι στο animation της θυρίδας body. Η παραπάνω μετάβαση πραγματοποιείται με την συνάρτηση Layered Blend per Bone.



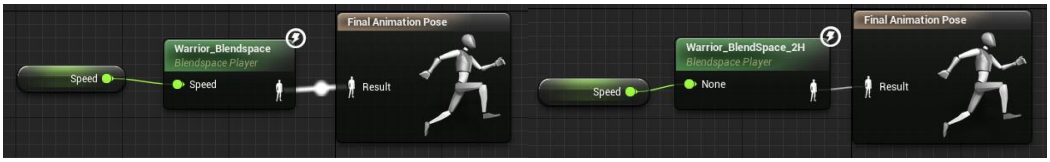
Πριν ξεκινήσει η δημιουργία του γράφου για τη μηχανή κατάστασης πρέπει πρώτα να δημιουργηθούν τα Blendspaces που θα χρησιμοποιεί ο παίχτης για να τρέχει με την ασπίδα και το σπαθί, το μονό σπαθί και

για να περπατάει όταν χρησιμοποιεί την ασπίδα του. Όπως προαναφέρθηκε και στην υπό-ενότητα 3.4 τα Blendspaces μας επιτρέπουν να συνδυάσουμε δύο ή περισσότερα animations. Στην συγκεκριμένη περίπτωση τα animations θα διαδέχονται το ένα το άλλο ανάλογα με την ταχύτητα του παίχτη. Στην παρακάτω εικόνα είναι το blendspace το οποίο περιέχει τη μετάβαση του χαρακτήρα από στάσιμη κατάσταση σε τρέξιμο.

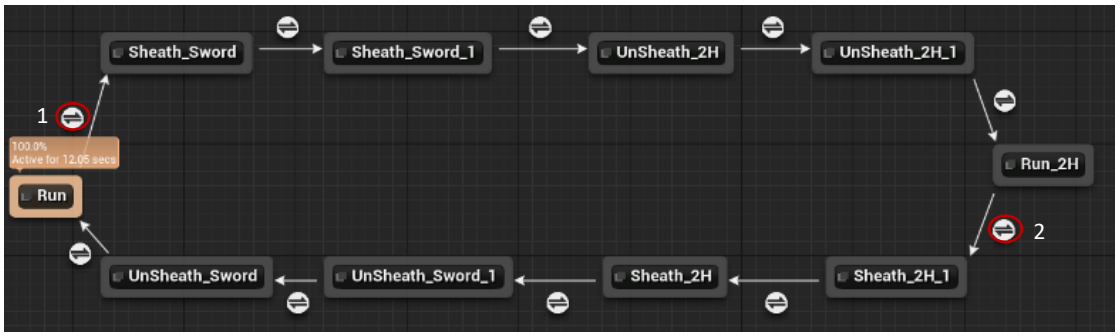


Με την ίδια διαδικασία δημιουργήθηκαν και τα Blendspace για το μονό σπαθί και την ασπίδα.

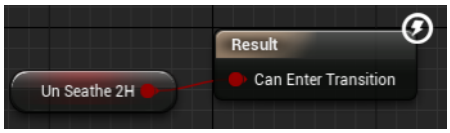
Στη συνέχεια θα αναλυθεί ο γράφος της μηχανής κατάστασης Locomotion. Αρχικά δημιουργούμε τις καταστάσεις τρέξιμο με ασπίδα και σπαθί (Run) και τρέξιμο με το μονό σπαθί (Run_2H). Έπειτα εντός των καταστάσεων καλούμε τα ανάλογα blendspace και δημιουργούμε μια μεταβλητή η οποία περιέχει την ταχύτητα του παίχτη.



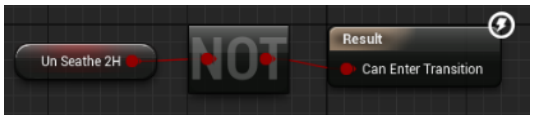
Στη συνέχεια με drag and drop προσθέτουμε τα animation που χρησιμοποιεί ο παίχτης για την αλλαγή όπλων με σωστή σειρά από τον asset browser στο γράφο της μηχανής κατάστασης.



Τα βελάκια μεταξύ των animation αντιπροσωπεύουν τη μετάβαση μεταξύ των ομώνυμων animation. Στις μεταβάσεις οι οποίες είναι μαρκαρισμένες με κόκκινο θα γίνει η αλλαγή όπλων. Για την αλλαγή όπλων δημιουργούμε μια μεταβλητή (UnSeathe_2H), η οποία αν είναι αληθής τότε γίνεται η αλλαγή σε μονό σπαθί.

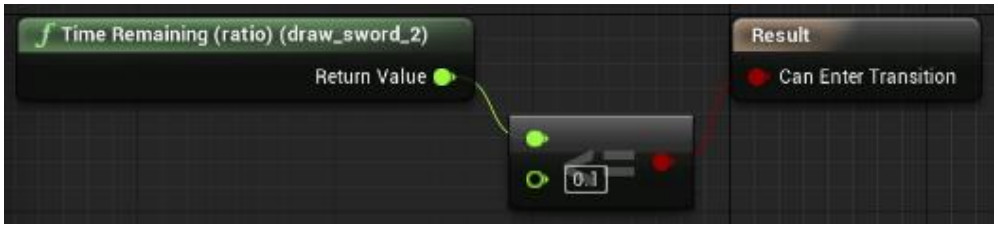


Άρα εσωτερικά τις πρώτης μετάβασης απλώς καλούμε την μεταβλητή και την επιστρέφουμε.

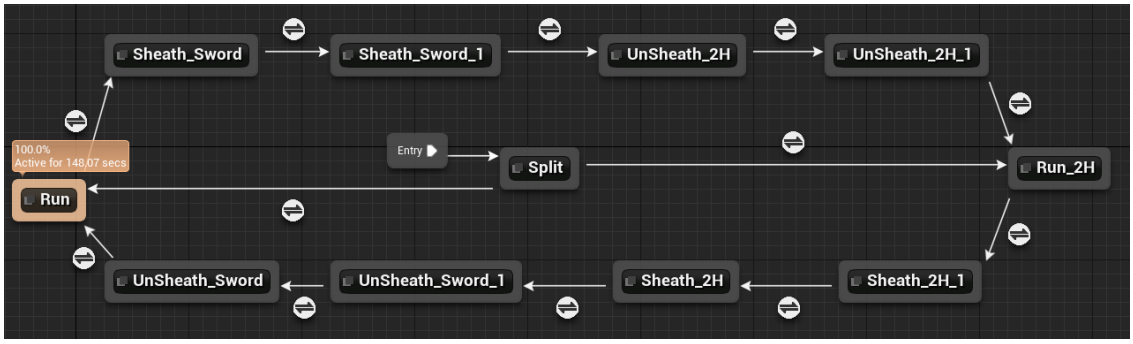


Ενώ η δεύτερη μετάβαση θέλουμε να γίνεται μόνο αν η μεταβλητή είναι αρνητική.

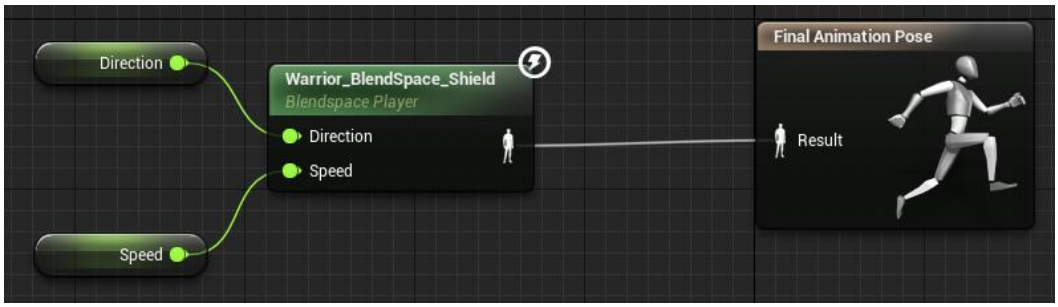
Για τις υπόλοιπες μεταβάσεις θέλουμε απλά να γίνεται ένας έλεγχος αν το animation βρίσκεται ακριβώς πριν το τέλος του για να παίξει στη συνέχεια το επόμενο animation. Αυτό πραγματοποιείται με τη συνάρτηση Time Remaining (ratio).



Στη συνέχεια προσθέτουμε την είσοδο του γράφου (Entry) και ένα κενό animation (Split) ανάμεσα στα δύο τρεξίματα για να ολοκληρωθεί η αλλαγή των όπλων. Χρησιμοποιούμε τις ίδιες μεταβάσεις με τη μεταβλητή UnSeathe_2H όπως και παραπάνω.

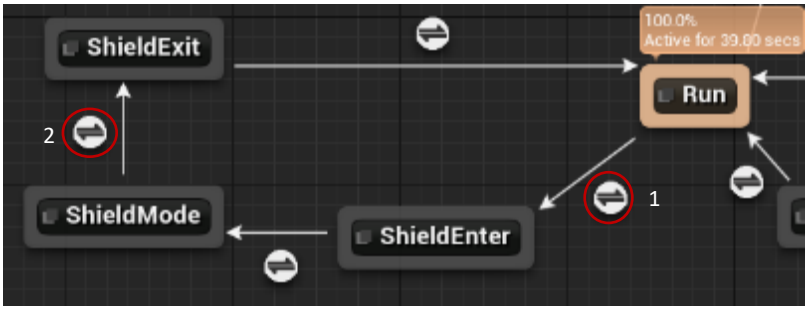


Για να ολοκληρωθεί η μηχανή κατάστασης πρέπει να προσθέσουμε και τρεις ακόμη καταστάσεις που αφορούν την ασπίδα του παίχτη. Οπότε προσθέτουμε από τον asset browser το animation που δείχνει τον παίχτη να εισέρχεται στη λειτουργία ασπίδας (ShieldEnter) και το animation που δείχνει τον παίχτη να εξέρχεται στη λειτουργία ασπίδας (ShieldExit). Έπειτα δημιουργούμε την κατάσταση στην οποία ο παίχτης χρησιμοποιεί την ασπίδα (ShieldMode). Μετά εντός της κατάστασης καλούμε το ανάλογο blendspace.



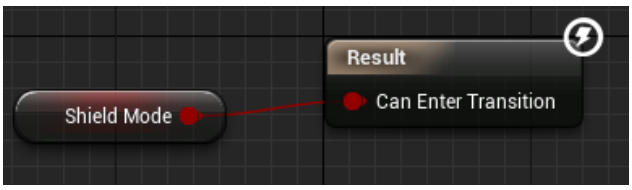
Επειδή το blendspace της λειτουργίας ασπίδας περιέχει πάνω από δύο animation είναι τρισδιάστατο οπότε δέχεται δύο μεταβλητές, την ταχύτητα που δημιουργήσαμε νωρίτερα και μια καινούργια μεταβλητή η οποία είναι η κατεύθυνση του παίχτη.

Εφόσον η λειτουργία ασπίδας χρησιμοποιείται μόνο κατά τη διάρκεια της κατάστασης τρεξίματος με ασπίδα και σπαθί, η κατάσταση αυτή θα είναι η είσοδος και η έξοδος της λειτουργίας ασπίδας.

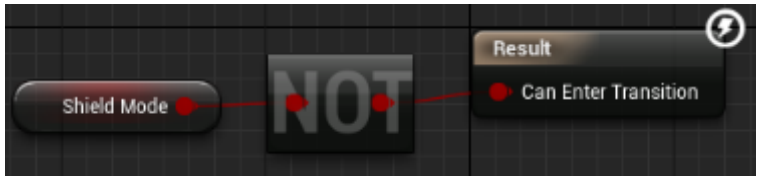


Το μόνο που απομένει είναι ο υπολογισμός των μεταβάσεων. Στις μεταβάσεις οι οποίες είναι μαρκαρισμένες με κόκκινο θα γίνει η είσοδος και έξοδος από τη λειτουργία ασπίδας αντίστοιχα. Για τη λειτουργία ασπίδας δημιουργούμε μια μεταβλητή (ShieldMode), η οποία αν είναι αληθής τότε ο παίχτης ξεκινά να χρησιμοποιεί την ασπίδα του.

Άρα εσωτερικά της πρώτης μετάβασης απλώς καλούμε τη μεταβλητή και την επιστρέφουμε.



Ενώ η δεύτερη μετάβαση θέλουμε να συμβαίνει μόνο αν η μεταβλητή είναι αρνητική.

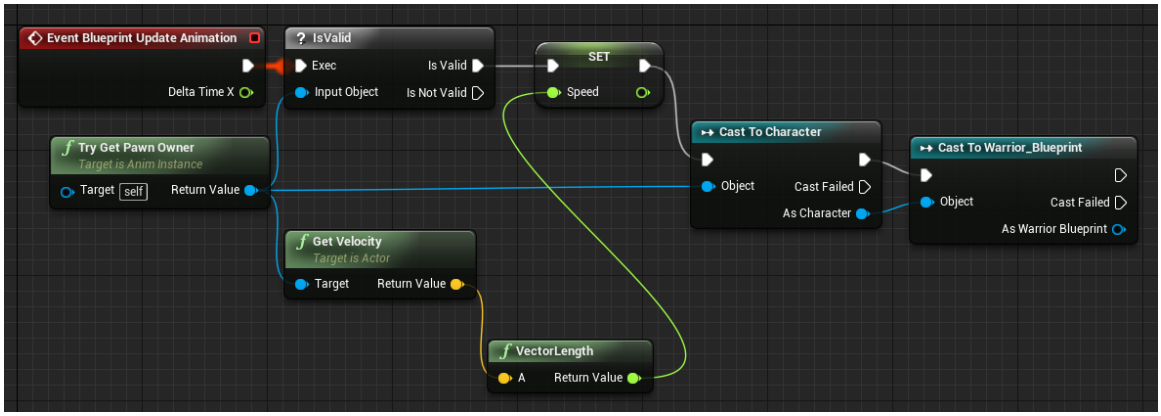


Για τις υπόλοιπες μεταβάσεις θέλουμε απλά να γίνεται ένας έλεγχος αν το animation βρίσκεται ακριβώς πριν το τέλος του για να παίξει στη συνέχεια το επόμενο animation, όπως κάναμε και προηγουμένως.

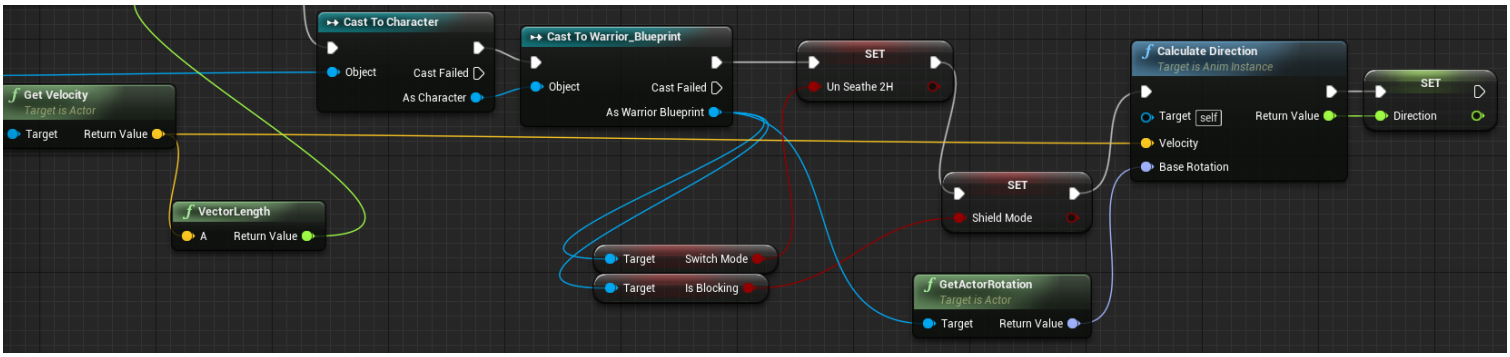
Εφόσον ολοκληρώθηκε πλήρως ο γράφος AnimGraph μεταβαίνουμε στο γράφο των γεγονότων ώστε να γίνει η σύνδεση των animation με τα ανάλογα γεγονότα από το κανονικό blueprint.

Αρχικά χρειαζόμαστε ένα γεγονός Blueprint Update animation, το οποίο εκτελείται κάθε που πραγματοποιείται ένα animation εντός του

παιχνιδιού. Στη συνέχεια κάνουμε έναν έλεγχο αν ο actor που χρησιμοποίησε κάποιο animation είναι ο παίχτης μας. Παράλληλα παίρνουν την τιμή κίνησης του διανύσματος του παίχτη και την θέτουμε στην ταχύτητα του παίχτη. Στη συνέχεια χρησιμοποιούμε τους ειδικούς κόμβους Cast To για να αποκτήσουμε πρόσβαση σε όλες τις συναρτήσεις και τα γεγονότα που υπάρχουν στο warrior_blueprint. Επομένως για να καλέσουμε τα γεγονότα που χρειαζόμαστε από το warrior blueprint πρέπει κάθε γεγονός να είναι συνδεδεμένο με την έξοδο As Warrior Blueprint του κόμβου Cast to Warrior_Blueprint. Οι αλλαγές φαίνονται στο παρακάτω γράφημα.



Έπειτα καλούμε από το κανονικό blueprint τα γεγονότα που πραγματοποιούν την αλλαγή όπλων και τη λειτουργία ασπίδας και τα συνδέουμε με τις ανάλογες μεταβλητές που δημιουργήσαμε εντός του animation blueprint. Μετά για να υπολογίσουμε την κατεύθυνση του παίχτη όταν χρησιμοποιεί την ασπίδα χρησιμοποιούμε την συνάρτηση Calculate direction.



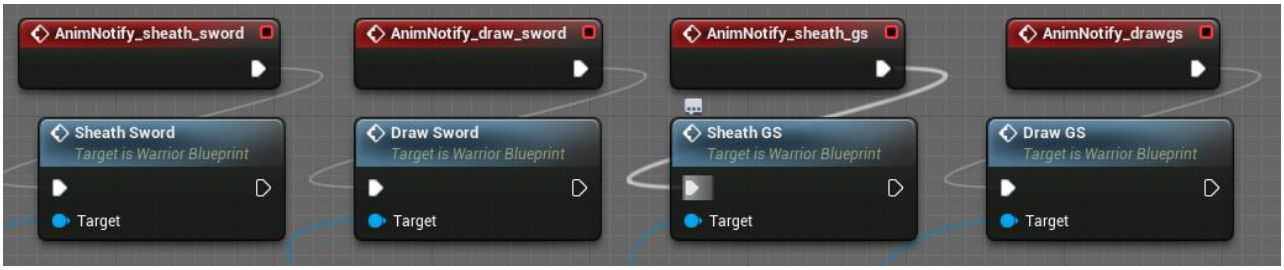
Το μόνο που έμεινε για να ολοκληρωθεί το animation blueprint είναι να συνδέσουμε τις ειδοποιήσεις που προσθέσαμε στα animation με τα κατάλληλα γεγονότα από το warrior blueprint.

- Η ειδοποίηση Chain χρησιμοποιείται από τα animation απλών επιθέσεων του παίχτη για να συνδέσει διαδοχικά τις επιθέσεις δημιουργώντας έτσι το σύστημα combo.
- Η ειδοποίηση Reset χρησιμοποιείται στις τελικές επιθέσεις και στις ειδικές επιθέσεις του παίχτη για να σταματήσει το σύστημα combo.

- Η ειδοποίηση LaunchFrw χρησιμοποιείται από τα animation αποφυγής επιθέσεων και απλών/ειδικών επιθέσεων για να ‘εκτοξεύσει’ τον παίχτη μπροστά.

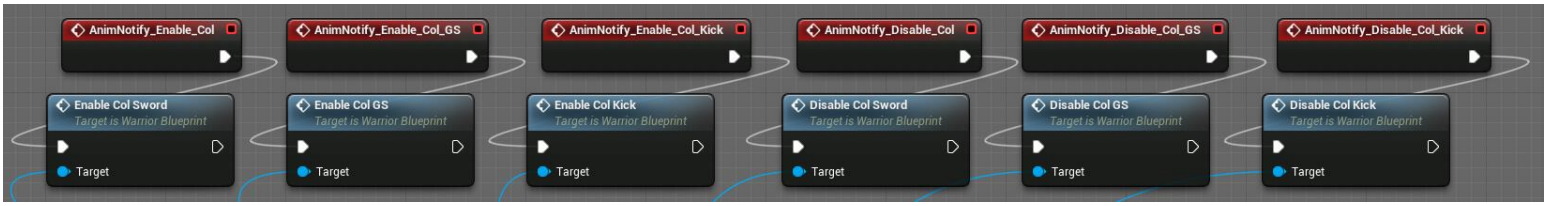
- Η ειδοποίηση Cam_Shake χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων του παίχτη για να κουνηθεί

ελάχιστα η κάμερα όταν ο παίχτης χτυπά επιτυχώς έναν αντίπαλο.



- Η ειδοποίηση sheath_sword χρησιμοποιείται από τα animation αλλαγής όπλων για να κρύψει την ασπίδα και το σπαθί που βρίσκονται στα χέρια του παίχτη και να εμφανίσει την ασπίδα και το σπαθί που βρίσκονται στο γοφό του.
- Η ειδοποίηση draw_sword χρησιμοποιείται από τα animation αλλαγής όπλων για να κρύψει την ασπίδα και το σπαθί που βρίσκονται στο γοφό του παίχτη και να εμφανίσει την ασπίδα και το σπαθί που βρίσκονται στα χέρια του.
- Η ειδοποίηση sheath_gs χρησιμοποιείται από τα animation αλλαγής όπλων για να κρύψει το μονό σπαθί που βρίσκεται στα χέρια του παίχτη και να εμφανίσει το μονό σπαθί που βρίσκεται στην πλάτη του.
- Η ειδοποίηση drawgs χρησιμοποιείται από τα animation αλλαγής όπλων για να κρύψει το μονό σπαθί που βρίσκεται στην πλάτη του παίχτη και να εμφανίσει το μονό σπαθί που βρίσκεται στα χέρια του.

- Η ειδοποίηση Switch_ON χρησιμοποιείται από τα animation αλλαγής όπλων για να πραγματοποιήσει ο παίχτης την αλλαγή όπλων από ασπίδα και σπαθί, σε μονό σπαθί.
- Η ειδοποίηση Switch_off χρησιμοποιείται από τα animation αλλαγής όπλων για να πραγματοποιήσει ο παίχτης την αλλαγή όπλων από μονό σπαθί, σε ασπίδα και σπαθί.



- Η ειδοποίηση Enable_Col χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων όπου ο παίχτης χρησιμοποιεί την ασπίδα και το σπαθί. Χρησιμοποιείται για να ενεργοποιηθεί το σύστημα σύγκρουσης του σπαθιού του παίχτη.
- Η ειδοποίηση Enable_Col_GS χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων όπου ο παίχτης χρησιμοποιεί το μονό σπαθί. Χρησιμοποιείται για να ενεργοποιηθεί το σύστημα σύγκρουσης του μονού σπαθιού του παίχτη.
- Η ειδοποίηση Enable_Col_Kick χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων όπου ο παίχτης χρησιμοποιεί την ασπίδα και το σπαθί. Χρησιμοποιείται για να ενεργοποιηθεί το σύστημα σύγκρουσης του δεξιού ποδιού του παίχτη.
- Η ειδοποίηση Disable_Col χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων όπου ο παίχτης χρησιμοποιεί την ασπίδα και το σπαθί. Χρησιμοποιείται για να απενεργοποιηθεί το σύστημα σύγκρουσης του σπαθιού του παίχτη.
- Η ειδοποίηση Disable_Col_GS χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων όπου ο παίχτης χρησιμοποιεί το μονό σπαθί. Χρησιμοποιείται για να απενεργοποιηθεί το σύστημα σύγκρουσης του μονού σπαθιού του παίχτη.

- Η ειδοποίηση Disable_Col_Kick Χρησιμοποιείται από τα animation απλών και ειδικών επιθέσεων όπου ο παίχτης χρησιμοποιεί την ασπίδα και το σπαθί. Χρησιμοποιείται για να απενεργοποιηθεί το σύστημα σύγκρουσης του δεξιού ποδιού του παίχτη.

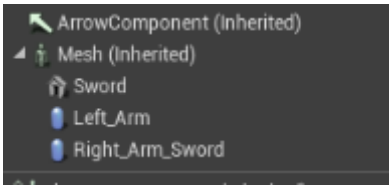
4.3 Λειτουργικότητα Αντιπάλων

Για τους αντιπάλους του παίχτη πρέπει να δημιουργήσουμε τρία blueprints τα οποία είναι το κανονικό blueprint, το animation blueprint και το blueprint το οποίο θα ανοίγει την πόρτα που βρίσκεται πίσω από τους τρεις πρώτους αντιπάλους. Επειδή όλοι οι αντίπαλοι του παίχτη χρησιμοποιούν τα ίδια blueprints θα αναλυθούν μόνο τα blueprints του πρώτου αντιπάλου. Οι διαφορές των αντιπάλων είναι οι πόντοι ζωής, ο αριθμός των επιθέσεων, η ζημιά που κάνουν οι επιθέσεις και όλοι οι αντίπαλοι χρησιμοποιούν διαφορετικά animations, τα οποία είναι προσαρμοσμένα στο σκελετό του κάθε αντιπάλου. Επίσης για όλους τους αντίπαλους χρησιμοποιήθηκε το γεγονός ‘εκτόξευσης’ (Launch_Frw), όπως είχε δημιουργηθεί στο blueprint του πολεμιστή.

Για τους αντιπάλους οι οποίοι χρησιμοποιούν όπλα πρέπει να δημιουργηθούν τα ανάλογα socket στο σκελετό του κάθε αντιπάλου όπως κάναμε και για τον παίχτη.

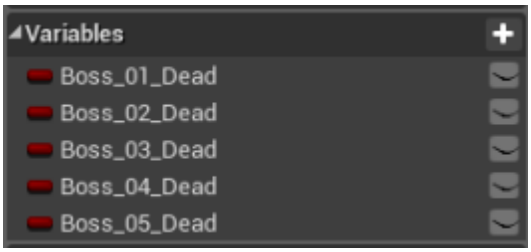
Αφού τοποθετηθούν τα sockets πρέπει εντός του blueprint να τοποθετηθούν και να προσαρμοστούν τα όπλα. Στη συνέχεια στην καρτέλα viewport του blueprint πρέπει να προστεθούν κάψουλες στα όπλα ή στα μέλη του σώματος με τα οποία επιτίθεται ο κάθε αντίπαλος.

4.3.1 Το Blueprint των Αντιπάλων

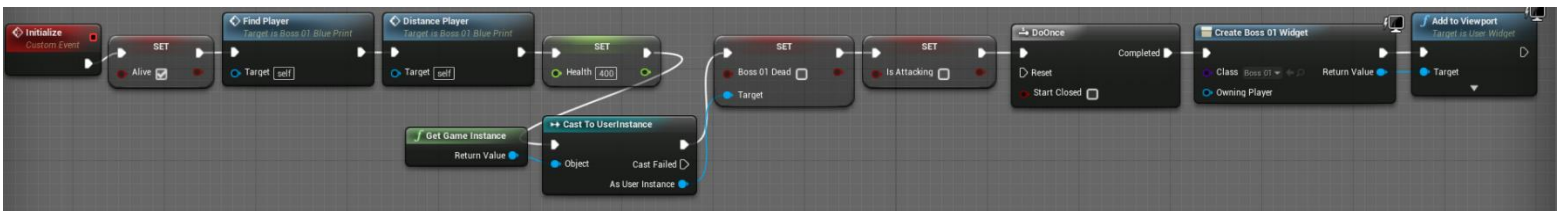


Ο πρώτος αντίπαλος του παίχτη χρησιμοποιεί ένα σπαθί στο δεξί του χέρι και το αριστερό του χέρι.

Επειδή ο παίχτης μας θα αλλάζει επίπεδο όταν πρόκειται να αντιμετωπίσει τον τελευταίο αντίπαλο, κάθε γεγονός που συμβαίνει στο προηγούμενο επίπεδο θα διαγράφεται από τη μνήμη. Άρα πρέπει με κάποιο τρόπο να κρατήσουμε στη μνήμη του παιχνιδιού ποιους αντιπάλους έχει νικήσει ο παίχτης μας έτσι ώστε να μετρήσουν στο score. Αυτό επιτυγχάνεται με Game Instances. Οπότε δημιουργούμε ένα Game Instance Blueprint με όνομα UserInstance και το μόνο που χρειάζεται να κάνουμε είναι να προσθέσουμε πέντε μεταβλητές, οι οποίες θα υποδεικνύουν ποιους αντιπάλους έχει νικήσει ο παίχτης. Οι ακόλουθες μεταβλητές θα αλλάζουν εντός των blueprints του κάθε αντιπάλου.



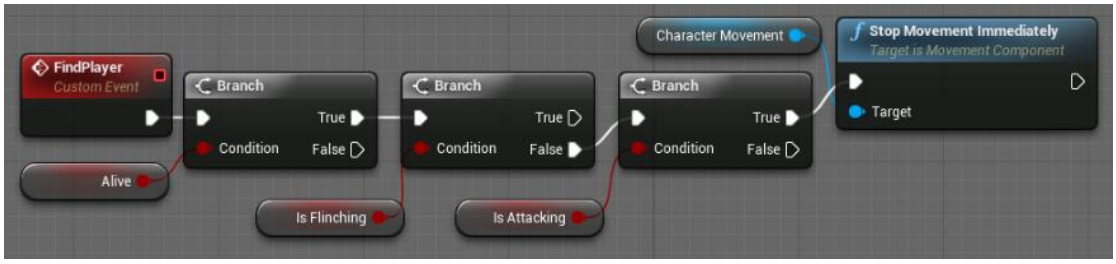
Στη συνέχεια μεταβαίνουμε στην καρτέλα γεγονότων του blueprint του πρώτου αντιπάλου όπου εδώ θα προγραμματίσουμε τη συμπεριφορά του αντιπάλου. Αρχικά δημιουργούμε ένα δικό μας γεγονός με όνομα Initialize, το οποίο θα εκτελείται όποτε ο παίχτης μας περνάει ένα trigger box που βρίσκεται λίγα μέτρα πριν από κάθε αντίπαλο. Τη



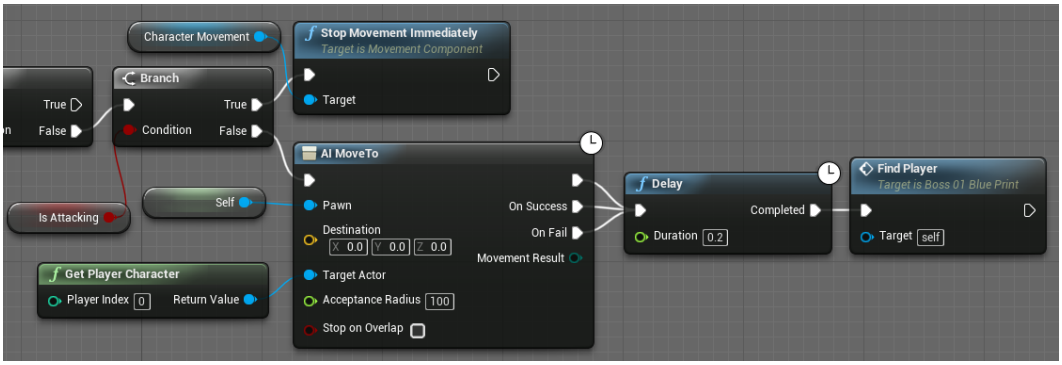
στιγμή που ενεργοποιηθεί το Initialize θα ξεκινάει και η λειτουργία του αντιπάλου.

Εντός της initialize αρχικά δηλώνουμε ότι ο αντίπαλος είναι ζωντανός. Έπειτα καλούμε δύο συναρτήσεις, οι οποίες χρησιμοποιούνται η μεν πρώτη για να βρει ο αντίπαλος τον παίχτη μας, η δε δεύτερη για να αρχίσει να επιτίθεται ο αντίπαλος στον παίχτη μας αν είναι σε κοντινή απόσταση. Στη συνέχεια ορίζουμε τους αρχικούς πόντους ζωής του αντιπάλου. Μετά καλούμε τη μεταβλητή που δημιουργήσαμε εντός του Game Instance για το αν ο πρώτος αντίπαλος είναι νεκρός και την θέτουμε ψευδή. Τέλος, θέτουμε ο αντίπαλος να μην επιτίθεται με το που ξεκινάει η λειτουργία του και δημιουργούμε το widget που εμφανίζει τους πόντους ζωής του αντιπάλου.

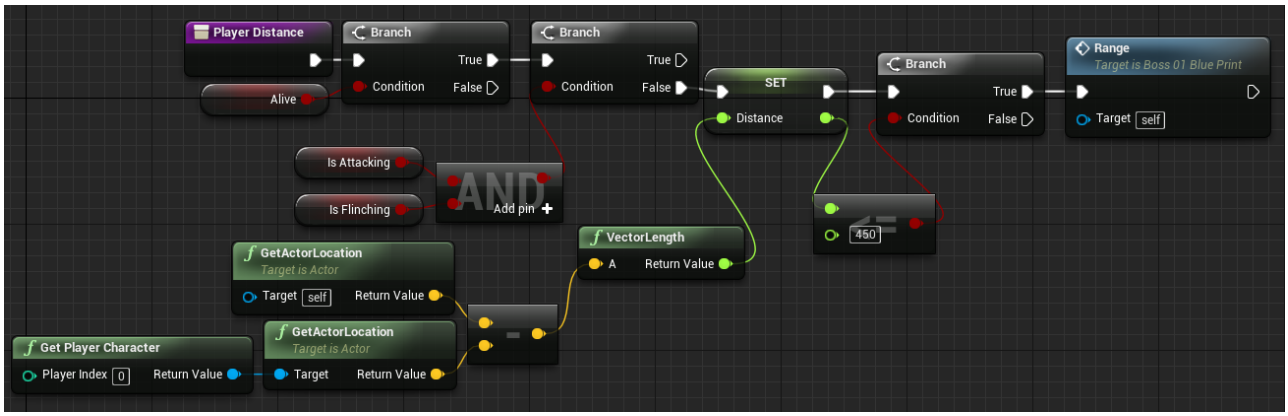
Στη συνέχεια δημιουργούμε το επαναλαμβανόμενο γεγονός, με το οποίο ο αντίπαλος θα βρει τον παίχτη μας μέσα στο επίπεδο και μετά θα ξεκινήσει να τον ακολουθεί (FindPlayer). Έπειτα κάνουμε δύο ελέγχους για το αν ο αντίπαλος είναι ζωντανός και αν δεν έχει προσωρινά ακινητοποιηθεί από μια δυνατή επίθεση του παίχτη (IsFlinching). Έπειτα αν ο αντίπαλος επιτίθεται, τότε θα σταματήσει να κινείται.



Αν δεν επιτίθεται τότε με την συνάρτηση Ai MoveTo θα ξεκινήσει να οδεύει προς τον παίχτη μας. Έπειτα προσθέτουμε στο τέλος το γεγονός που δημιουργήσαμε, για να είναι το γεγονός επαναλαμβανόμενο.

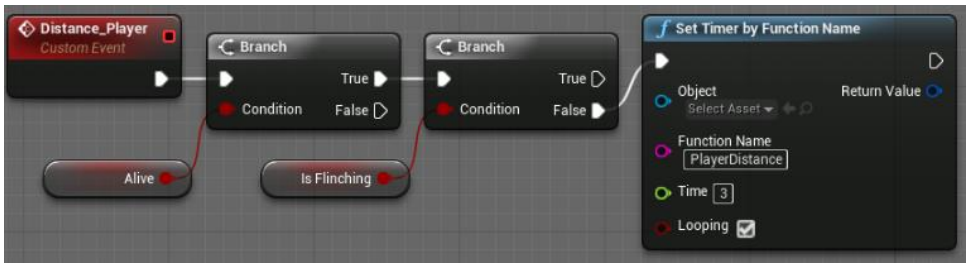


Επομένως, δημιουργούμε μια συνάρτηση με την οποία ο αντίπαλος θα αρχίσει να επιτίθεται στον παίχτης μας, όταν βρίσκεται σε κοντινή απόσταση. Αρχικά, εσωτερικά της συνάρτησης κάνουμε έναν έλεγχο αν ο αντίπαλος είναι ζωντανός. Έπειτα πραγματοποιούνται ακόμα δύο έλεγχοι, για το αν ο αντίπαλος δεν έχει προσωρινά ακινητοποιηθεί από μια δυνατή επίθεση του παίχτη και αν δεν επιτίθεται στον παίχτη. Μετά παίρνουμε την τοποθεσία του παίχτη και την τοποθεσία του αντίπαλου μέσα στο επίπεδο και τις αφαιρούμε. Το αποτέλεσμα αποθηκεύεται σε μια μεταβλητή Distance. Τέλος, κάνουμε ένα έλεγχο αν ο παίχτης βρίσκεται σε κοντινή απόσταση από τον αντίπαλο. Αν βρίσκεται σε κοντινή απόσταση τότε καλείται το γεγονός Range στο οποίο θα γίνονται οι επιθέσεις του αντιπάλου.



Στη συνέχεια μεταβαίνουμε πάλι στο γράφο γεγονότων και δημιουργούμε ένα γεγονός Distance_Player, το οποίο θα καλεί τη συνάρτηση που υπολογίζει την απόσταση. Αρχικά ελέγχουμε αν ο αντίπαλος είναι ζωντανός και αν δεν έχει προσωρινά ακινητοποιηθεί από μια δυνατή επίθεση. Έπειτα με την συνάρτηση Set Timer By

Function Name καλούμε τη συνάρτηση Player_distance να εκτελείται κάθε τρία δευτερόλεπτα.

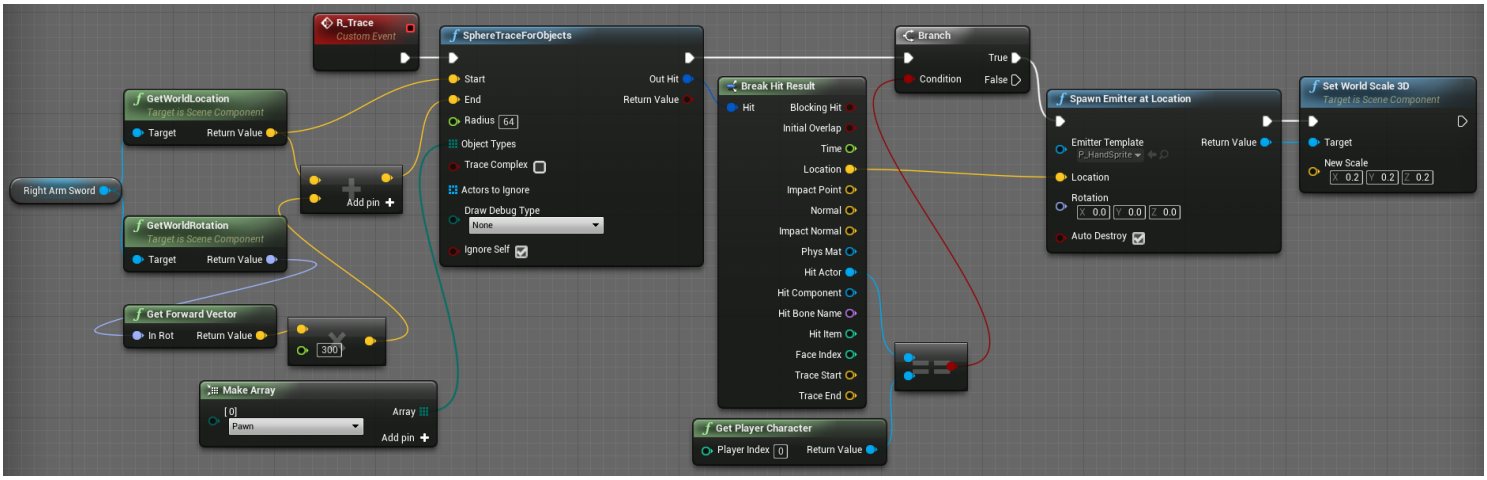


Έπειτα δημιουργούμε το γεγονός Range, στο οποίο ο αντίπαλος θα επιτίθεται στον παίχτη μας. Αρχικά θέτουμε ότι ο αντίπαλος επιτίθεται. Μετά με τις συναρτήσεις Random Integer In Range και Switch on Int ο αντίπαλος θα πραγματοποιεί στην τύχη μια από τις τέσσερις επιθέσεις που θα του αναθέσουμε. Στη συνέχεια για κάθε επίθεση καλούμε ένα διαφορετικό animation montage, θέτουμε πόσο θα 'εκτοξευτεί' ευθεία ο αντίπαλος σε κάθε animation, θέτουμε την ελάχιστη και τη μέγιστη ζημιά που μπορεί να κάνει η επίθεση και τελικά καλούμε την ανάλογη συνάρτηση για να παίξουν τα animation montage.

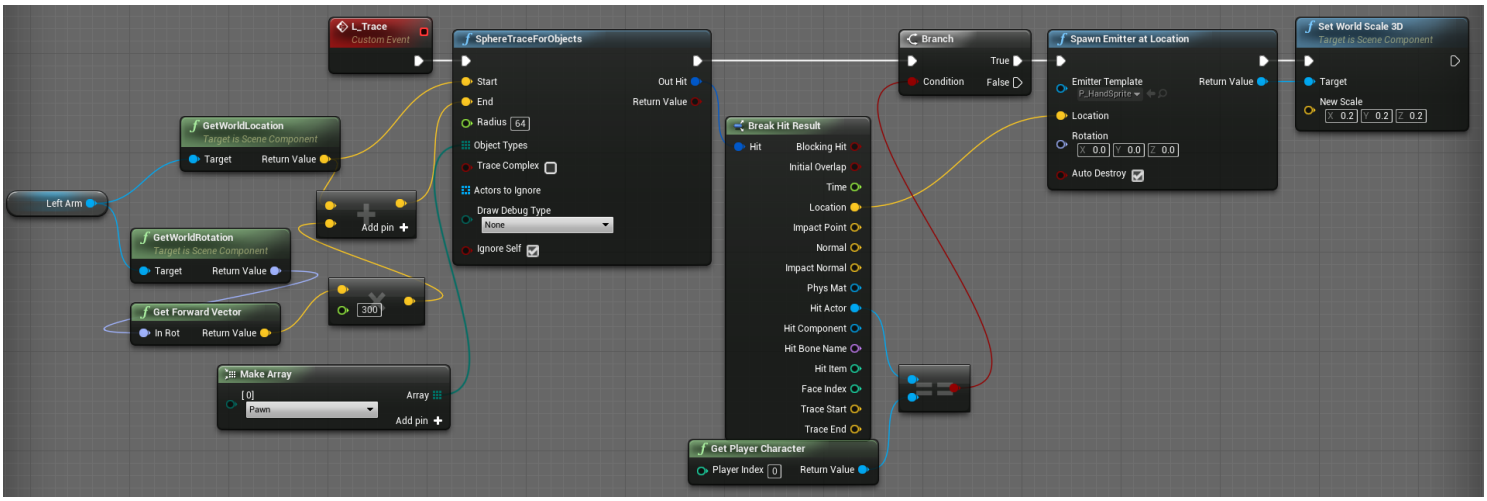
Εφόσον ο αντίπαλος πραγματοποιήσει στην τύχη μια επίθεση σταματά να επιτίθεται και ξεκινά να αναζητεί και πάλι τον παίχτη μας. Τέλος, αν ο παίχτης βρίσκεται σε κοντινή απόσταση με τον αντίπαλο τότε καλείται το γεγονός Face_Player με το οποίο ο αντίπαλος περιστρέφεται έτσι ώστε να αντικρύζει τον παίχτη μας.

Το επόμενο γράφημα είναι η υλοποίηση του γεγονότος FacePlayer. Χρησιμοποιούμε τη συνάρτηση Find Look at Rotation ώστε να πάρουμε τις μοίρες που πρέπει να στρίψει ο αντίπαλος από τον παίχτη μας. Έπειτα επιστρέφουμε τις μοίρες στην συνάρτηση SetActorRotation ώστε να πραγματοποιηθεί η περιστροφή του αντιπάλου. Το γεγονός Faceplayer θα χρησιμοποιηθεί και εντός του animation blueprint του αντιπάλου ώστε ο αντίπαλος να αντικρύζει τον παίχτη μας την ώρα που πραγματοποιεί μια επίθεση.

Στη συνέχεια δημιουργήθηκαν δύο γεγονότα με τα οποία όποτε ο αντίπαλος πετυχαίνει τον παίχτη μας σε μία επίθεση θα δημιουργείται ένα σύστημα σωματιδίων, στη θέση που ο αντίπαλος πέτυχε τον παίχτη μας ώστε, να υποδείξει στο χρήστη ότι το χτύπημα του αντιπάλου ήταν επιτυχές. Εφόσον ο πρώτος αντίπαλος έχει δύο κάψουλες πρέπει να δημιουργηθεί ένα γεγονός για την κάθε κάψουλα. Για την κάψουλα που βρίσκεται στο σπαθί του αντιπάλου παίρνουμε την τοποθεσία και την περιστροφή της κάψουλας και τις περνάμε στη συνάρτηση SphereTraceForObjects. Η συνάρτηση αυτή θα ψάξει το επίπεδο για actors της κλάσης Pawn όπως είναι και οι αντίπαλοι του παίχτη. Έπειτα γίνεται ένας έλεγχος αν η κάψουλα του σπαθιού ακούμπησε τον παίχτη μας. Αν τον έχει ακουμπήσει τότε θα δημιουργηθεί το σύστημα σωματιδίων στην θέση που υπήρξε η επαφή. Τέλος, ορίζουμε το μέγεθος που θα έχει το σύστημα σωματιδίων.



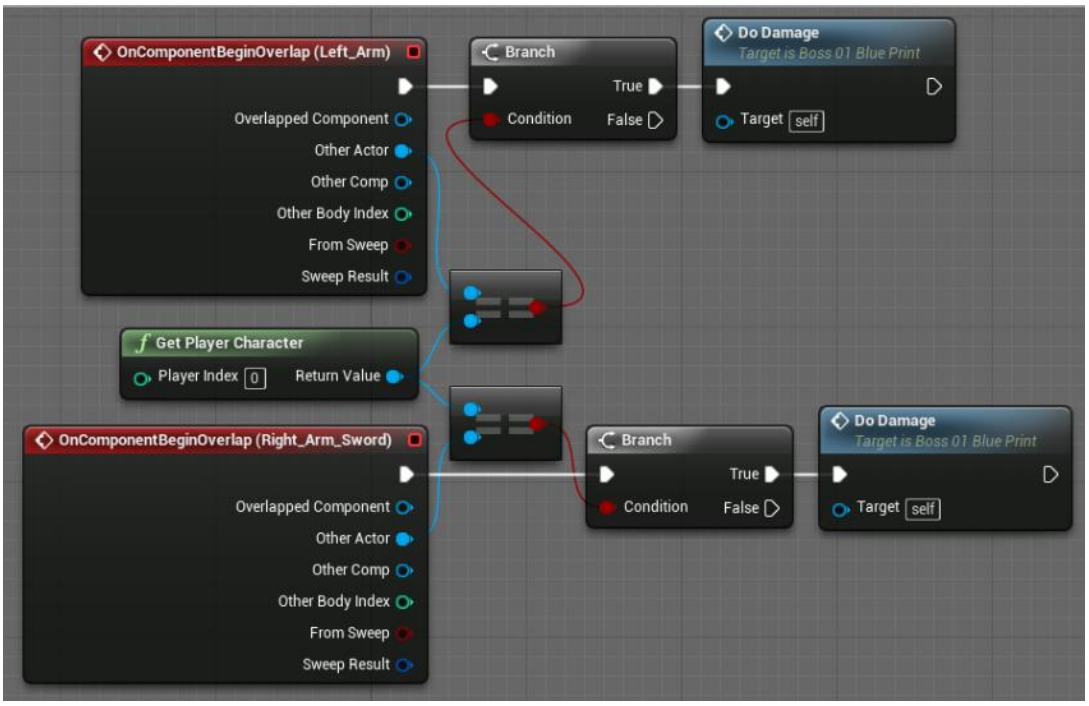
Παρομοίως δημιουργούμε και το γεγονός για την κάψουλα του αριστερού χεριού :



Τα γεγονότα αυτά θα καλούνται εντός του animation blueprint γιατί πρέπει σε κάθε animation επίθεσης των αντιπάλων να επιλεγούν τα σωστά σημεία που θα παίζεται το σύστημα σωματιδίων με βάση το animation.

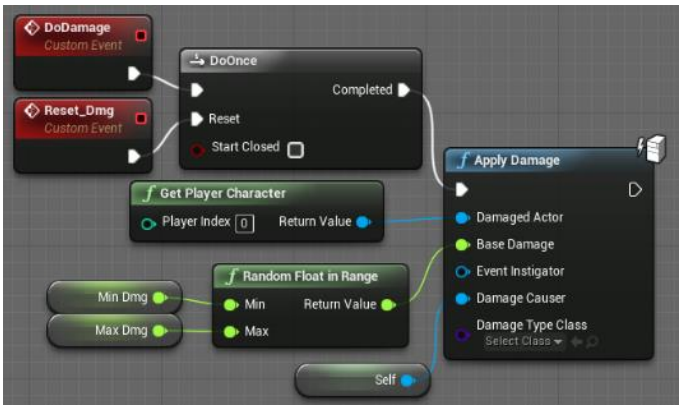
Επομένως, για να υπολογίσουμε τη ζημιά που κάνει ο αντίπαλος στον παίχτη θα χρησιμοποιήσουμε τις κάψουλες του αντιπάλου όπως πράξαμε και με τον παίχτη. Η διαδικασία που είναι ίδια με αυτή του παίχτη. Αφού αλλάξουμε τις ρυθμίσεις σύγκρουσης των καψουλών του αντιπάλου δημιουργούμε τα κατάλληλα γεγονότα που θα χρησιμοποιηθούν στο animation blueprint.

Έπειτα δημιουργήθηκε το γράφημα υπολογισμού ζημιάς, που προκαλεί ο αντίπαλος στον παίχτη. Για τον υπολογισμό χρησιμοποιήθηκαν δύο γεγονότα OnComponentBeginOverlap για την κάθε κάψουλα, τα οποία ενεργοποιούνται όταν υπάρχει επαφή ενός actor με μια από τις κάψουλες. Αρχικά γίνεται ένας έλεγχος εάν ο actor που δέχεται την επίθεση είναι ο παίχτης. Στη συνέχεια καλείται το γεγονός DoDamage, όπου συνεχίζεται ο υπολογισμός της ζημιάς.

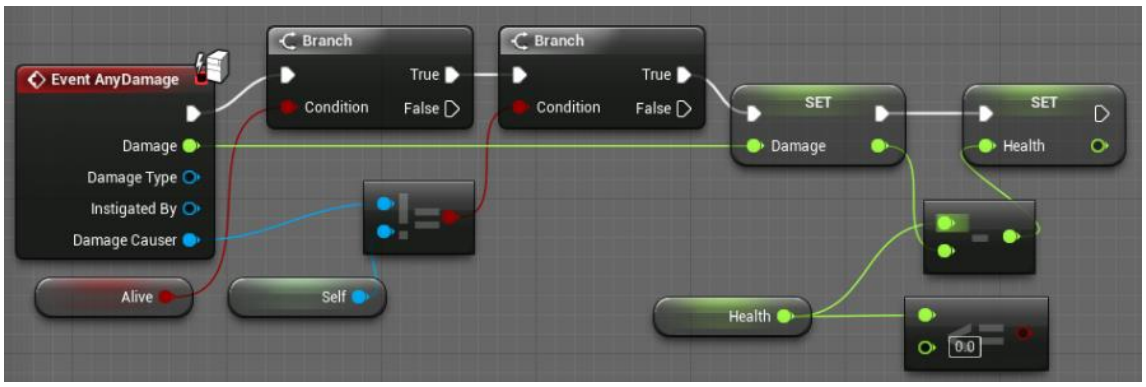


Στο γεγονός DoDamage χρησιμοποιήθηκε η μακροεντολή DoOnce έτσι ώστε ο παίχτης μας να προκαλεί μόνο μια φόρα ζημιά στον αντίπαλο, κάθε φορά που πραγματοποιεί μια επίθεση. Στη συνέχεια χρησιμοποιείται η συνάρτηση ApplyDamage για να προκαλέσει ζημιά ο αντίπαλος στον παίχτη. Η ελάχιστη και μέγιστη ζημιά που μπορεί να

προκαλέσει κάθε επίθεση έχει οριστεί στο γεγονός Range. Όταν ολοκληρωθεί η DoOnce τότε επαναφέρεται στην αρχική της κατάσταση με το γεγονός Reset_Dmg.



Για να ολοκληρωθεί το blueprint του αντιπάλου πρέπει να γίνει και ο υπολογισμός ζημιάς που δέχεται ο αντίπαλος από τον παίχτη. Χρησιμοποιώντας το γεγονός Event AnyDamage μπορούμε να πάρουμε τη ζημιά που δέχτηκε ο αντίπαλος και έπειτα να την αποθηκεύσουμε σε μια μεταβλητή Damage. Στη συνέχεια κάνουμε δύο ελέγχους, ο μεν πρώτος αν ο αντίπαλος είναι ζωντανός, ο δε δεύτερος αν η ζημιά που προκλήθηκε στον αντίπαλο δεν είναι από τον ίδιο τον αντίπαλο. Έπειτα αφαιρούμε τη ζημιά που δέχτηκε ο αντίπαλος από τους πόντους ζωής του.

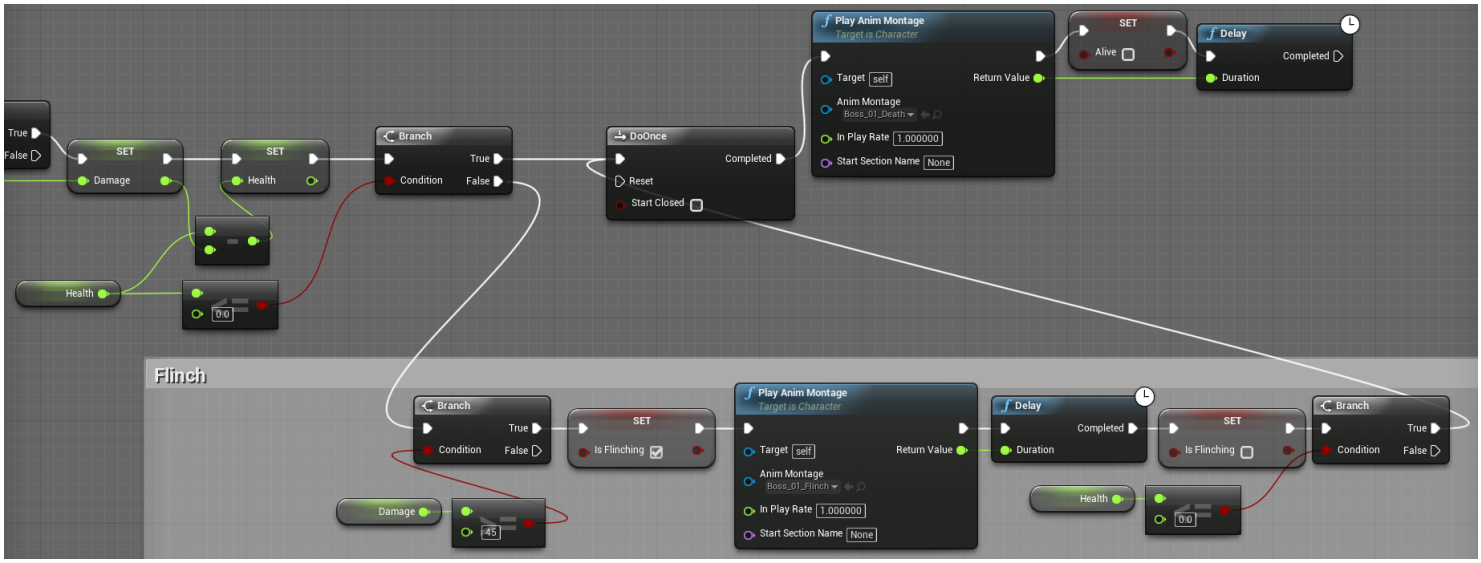


Μετά κάνουμε έναν έλεγχο αν οι πόντοι ζωής του αντιπάλου είναι μικρότεροι ή ίσοι του μηδενός.

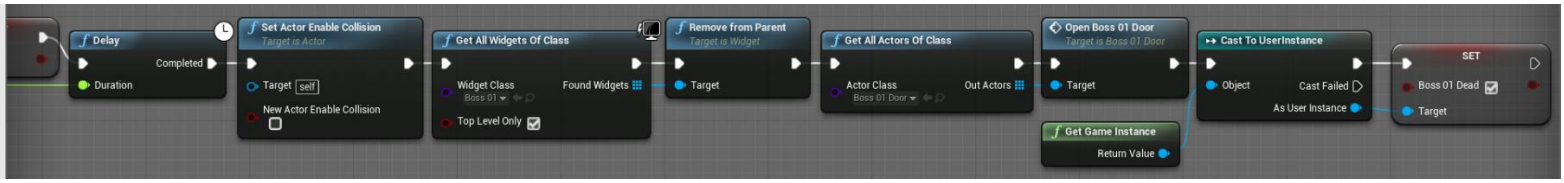
- Αν είναι μεγαλύτεροι του μηδενός τότε γίνεται ένας έλεγχος αν η ζημιά που προκλήθηκε στον αντίπαλο ήταν μεγάλη (>45). Αν ήταν

τότε δηλώνουμε ότι ο αντίπαλος είναι προσωρινά ακινητοποιημένος και παίζουμε το ανάλογο animation. Μετά από λίγα δευτερόλεπτα που πρόκειται να ολοκληρωθεί το animation δηλώνουμε ότι ο αντίπαλος δεν είναι πλέον ακινητοποιημένος. Έπειτα γίνεται ακόμα ένας έλεγχος αν ο αντίπαλος είναι ζωντανός. Αν είναι τότε συνεχίζει να επιτίθεται. Αλλιώς ακολουθεί την πορεία της επόμενης κουκίδας.

- Αν είναι μικρότεροι ή ίσοι του μηδενός τότε καλούμε τη μακροεντολή DoOnce ώστε να παιχτεί μόνο μια φορά το animation θανάτου του αντιπάλου. Έπειτα θέτουμε ότι ο αντίπαλος είναι νεκρός.



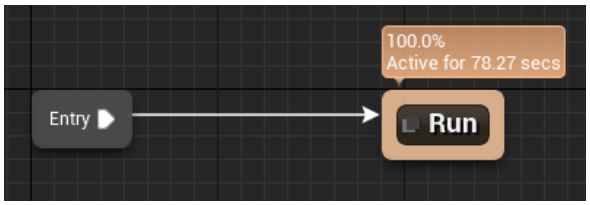
Συνεχίζοντας από τη δεύτερη κουκίδα καλούμε τη συνάρτηση SetActorEnableCollision για να απενεργοποιήσουμε το σύστημα σύγκρουσης του αντιπάλου έτσι ώστε να μπορεί ο παίχτης μας να περάσει μέσα από τον αντίπαλο αντί να κολλήσει πάνω του. Έπειτα επιλέγουμε το widget του αντιπάλου και το βγάζουμε από την οθόνη. Μετά καλούμε τη συνάρτηση Boss_01_Door από το blueprint που ανοίγει την πόρτα πίσω από το αφεντικό. Τέλος, καλούμε την μεταβλητή που δημιουργήσαμε εντός του Game Instance για το αν ο πρώτος αντίπαλος είναι νεκρός και την θέτουμε αληθή.



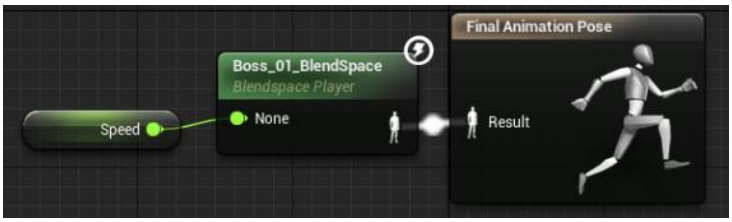
4.3.2 Το Animation Blueprint των Αντιπάλων

Πριν ξεκινήσουν οι αλλαγές στο animation blueprint του παίχτη πρέπει πρώτα να προσθέσουμε ειδοποιήσεις στο χρονοδιάγραμμα ορισμένων animation montage κάθε αντιπάλου.

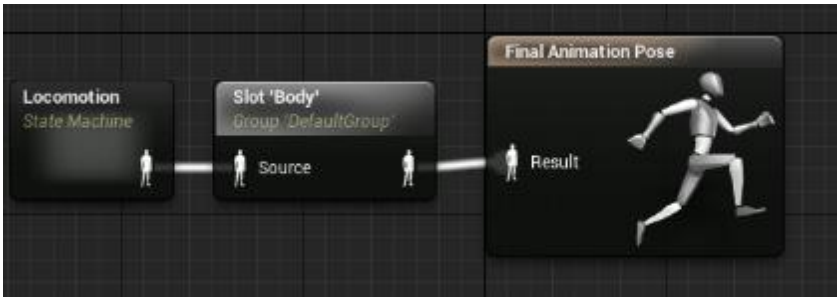
Στη συνεχεία μεταβαίνουμε στο animation blueprint του πρώτου αντιπάλου και στο γράφο AnimGraph. Δημιουργούμε μια θυρίδα (Slot) η οποία κρατάει όλα τα animation montage του αντιπάλου. Η ανάθεση των θυρίδων γίνεται εντός των animation montage. Έπειτα δημιουργούμε τη μηχανή κατάστασης του αντιπάλου.



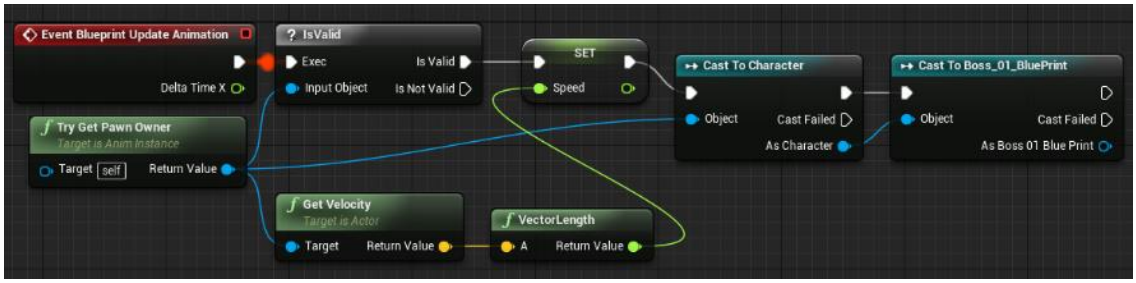
Εσωτερικά της μηχανής κατάστασης χρειαζόμαστε μόνο την είσοδο του γράφου και μια μηχανή κατάστασης για το τρέξιμο του αντιπάλου.



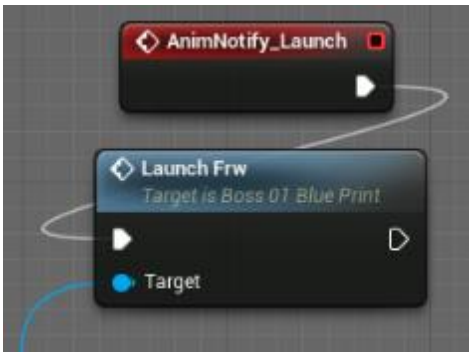
Έπειτα εντός της κατάστασης του τρεξίματος καλούμε το ανάλογο blendspace και δημιουργούμε μια μεταβλητή, η οποία περιέχει την ταχύτητα του παίχτη.



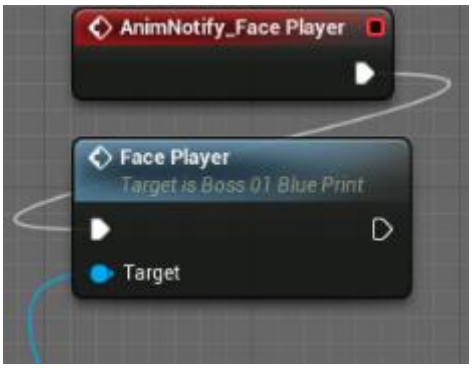
Εφόσον ολοκληρώθηκε πλήρως ο γράφος AnimGraph μεταβαίνουμε στο γράφο των γεγονότων ώστε να γίνει η σύνδεση των animation με τα ανάλογα γεγονότα από το blueprint του αντιπάλου. Η αρχή του γράφου είναι ίδια με αυτή του παίχτη με τη διαφορά ότι χρησιμοποιούμε τον ειδικός κόμβος Cast To Boss_01_Blueprint για να αποκτήσουμε πρόσβαση σε όλες τις συναρτήσεις και τα γεγονότα που υπάρχουν στο blueprint του πρώτου αφεντικού. Επομένως κάθε γεγονός πρέπει να είναι συνδεδεμένο με την έξοδο As Boss 01 Blueprint του κόμβου Cast to Boss_01_Blueprint. Οι αλλαγές φαίνονται στο παρακάτω γράφημα.



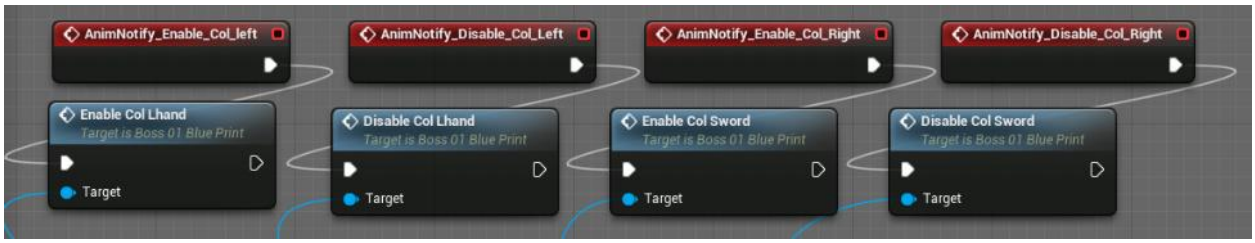
Επομένως το μόνο που μένει για να ολοκληρωθεί το animation blueprint είναι να συνδέσουμε τις ειδοποιήσεις που προσθέσαμε στα animation με τα κατάλληλα γεγονότα από το blueprint του αντιπάλου.



- Η ειδοποίηση Launch χρησιμοποιείται από τα animation επίθεσης για να ‘εκτοξεύσει’ τον αντίπαλο μπροστά.



- Η ειδοποίηση Face_player χρησιμοποιείται από τα animation επίθεσης για να στρίψει τον αντίπαλο ώστε να αντικρίζει τον παίχτη.

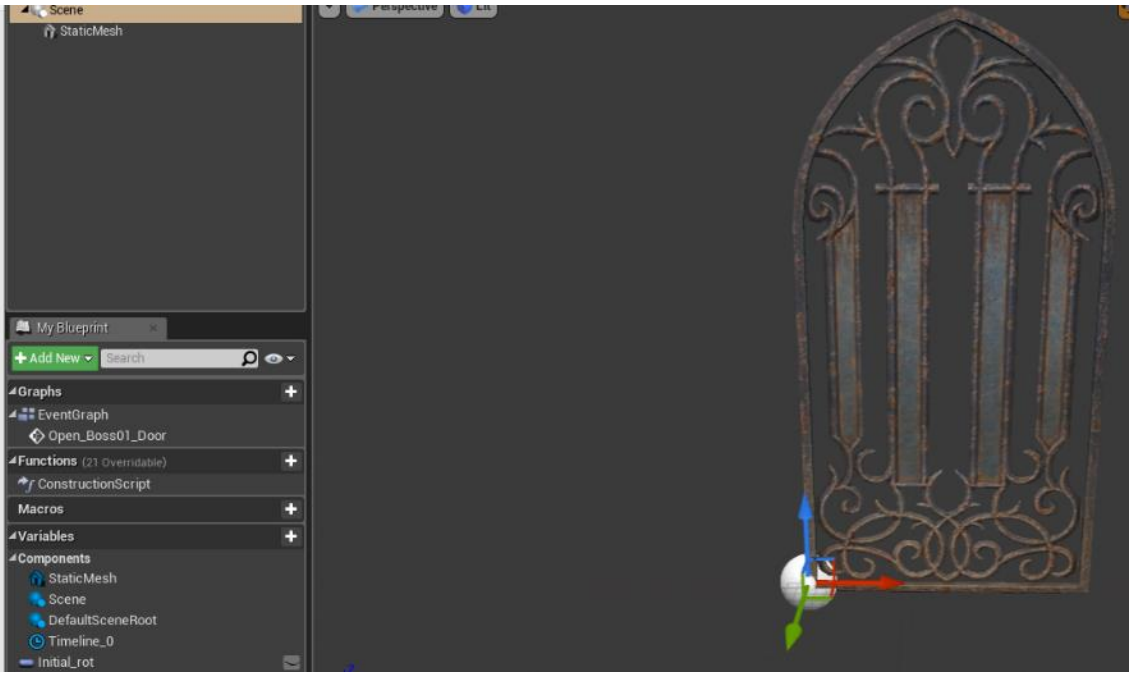


- Η ειδοποίηση Enable_Col_left χρησιμοποιείται από τα animation επίθεσης του αντιπάλου για να ενεργοποιήσει το σύστημα σύγκρουσης του αριστερού χεριού του αντιπάλου.
- Η ειδοποίηση Disable_Col_left χρησιμοποιείται από τα animation επίθεσης του αντιπάλου για να απενεργοποιήσει το σύστημα σύγκρουσης του αριστερού χεριού του αντιπάλου.
- Η ειδοποίηση Enable_Col_Right χρησιμοποιείται από τα animation επίθεσης του αντιπάλου για να ενεργοποιήσει το σύστημα σύγκρουσης του σπαθιού του αντιπάλου.
- Η ειδοποίηση Disable_Col_ Right χρησιμοποιείται από τα animation επίθεσης του αντιπάλου για να απενεργοποιήσει το σύστημα σύγκρουσης του σπαθιού του αντιπάλου.

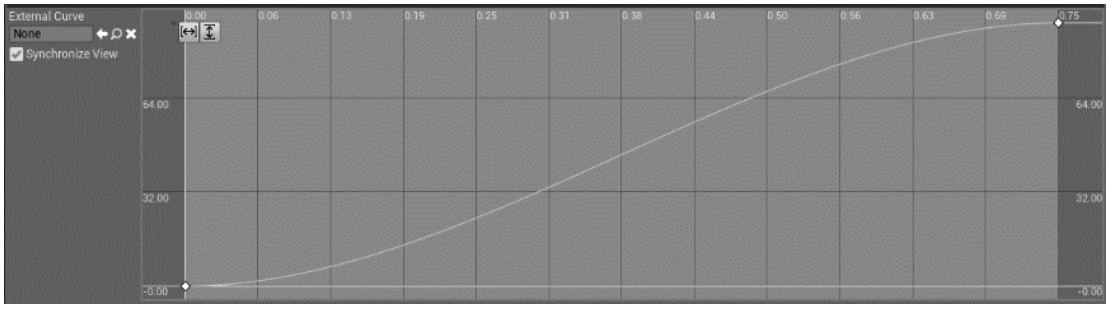
- Η ειδοποίηση Left_Trace χρησιμοποιείται από τα animation επίθεσης του αντιπάλου για να δημιουργεί το ανάλογο σύστημα σωματιδίων όταν ο αντίπαλος επιτίθεται επιτυχώς στον παίχτη με το αριστερό του χέρι.
- Η ειδοποίηση Right_Trace χρησιμοποιείται από τα animation επίθεσης του αντιπάλου για να δημιουργεί το ανάλογο σύστημα σωματιδίων όταν ο αντίπαλος επιτίθεται επιτυχώς στον παίχτη με το σπαθί του.

4.3.3 Το Blueprint της Πόρτας των Αντιπάλων

Όπως έχει είδη αναφερθεί οι πρώτοι τρεις αντίπαλοι θα έχουν μια κλειστή πόρτα πίσω τους έτσι ώστε ο χρήστης θα πρέπει αναγκαστικά να τους νικήσει για να προχωρήσει στο επίπεδο. Για να γίνει αυτό εφικτό δημιουργούμε τρία καινούργια blueprint για κάθε αντίπαλο. Τα blueprint εσωτερικά είναι ίδια αλλά έχουμε δημιουργήσει τρία με διαφορετικά ονόματα ώστε ο κάθε αντίπαλος να ξεκλειδώνει διαφορετική πόρτα. Αρχικά στην καρτέλα viewport προσθέτουμε ένα Scene component το οποίο έχει ίδια λειτουργία με ένα κανονικό component αλλά δέχεται και μετασχηματισμούς μεγέθους και περιστροφής. Εσωτερικά του Scene component τοποθετούμε το static mesh μιας πόρτας. Έπειτα προσαρμόζουμε το Scene component να βρίσκεται στην κάτω αριστερή γωνία της πόρτας.

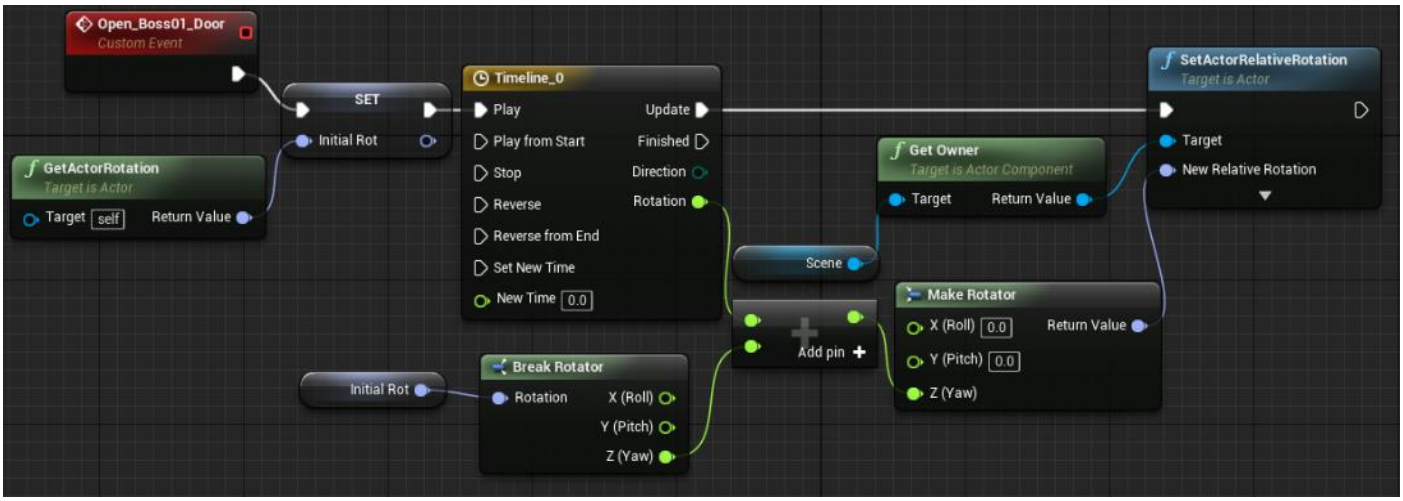


Έπειτα στο γράφο γεγονότων του blueprint θα κάνουμε το Scene component να περιστρέφεται 90^{o} μοίρες και επειδή η πόρτα είναι παιδί του Scene component θα περιστρέφεται και αυτή μαζί του με φυσικό τρόπο. Στη συνέχεια δημιουργούμε το γεγονός που θα καλείται από το animation των αντιπάλων για να ανοίξει η πόρτα. Στη συνέχεια παίρνουμε τη αρχική περιστροφή που έχει το Scene component και την αποθηκεύουμε σε μια μεταβλητή Initial_rot. Μετά δημιουργούμε ένα χρονοδιάγραμμα έτσι ώστε η πόρτα να μην ανοίγει στιγμιαία αλλά να ανοίγει με φυσικό τρόπο.



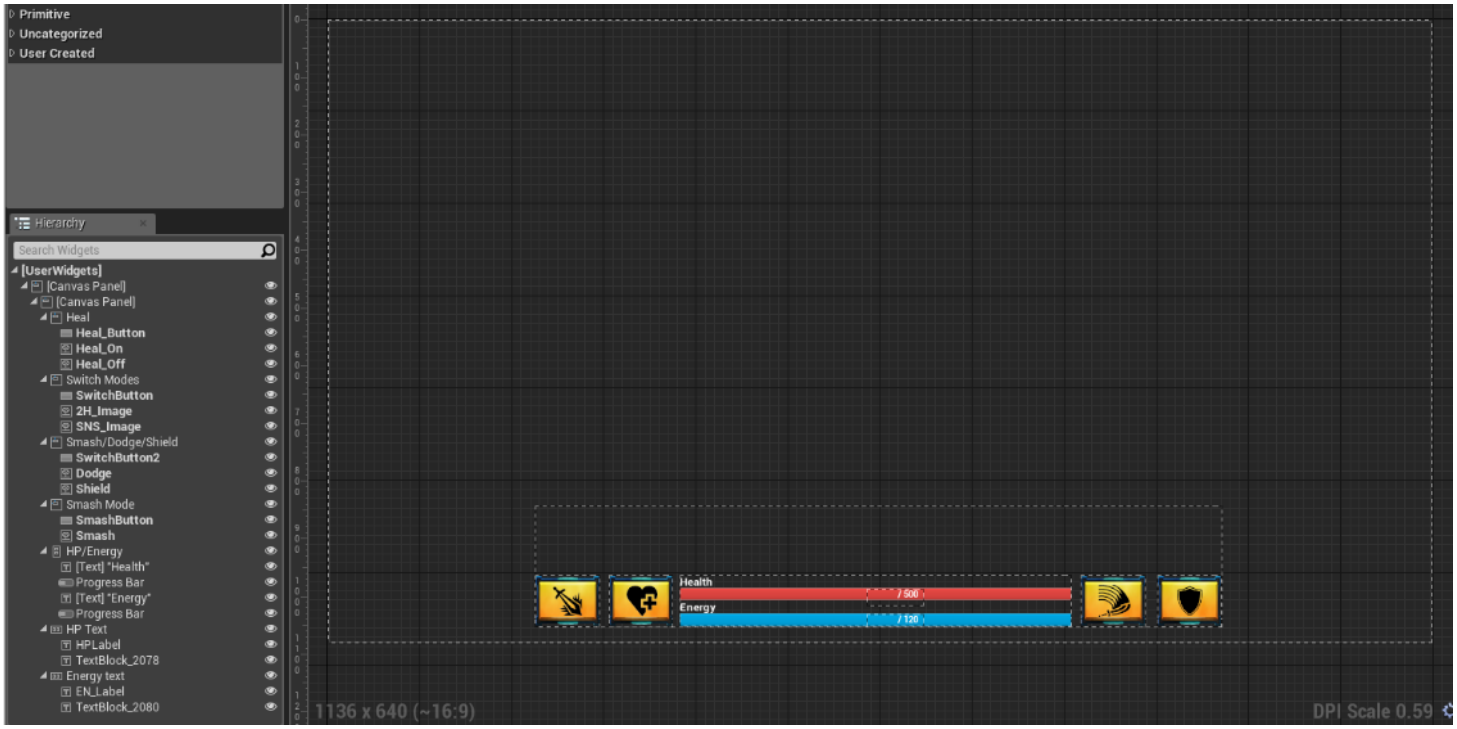
Εντός του χρονοδιαγράμματος ρυθμίσαμε το Scene component να παίρνει μια περιστροφή των 90^{o} σε λίγο παραπάνω από ένα δευτερόλεπτο. Στην συνέχεια προσθέτουμε την αρχική περιστροφή με

την περιστροφή του χρονοδιαγράμματος και τη ρυθμίζουμε να πραγματοποιείται στο άξονα yaw. Τέλος, με την συνάρτηση SetActorRelativeRotation εφαρμόζουμε την περιστροφή στο Scene component.



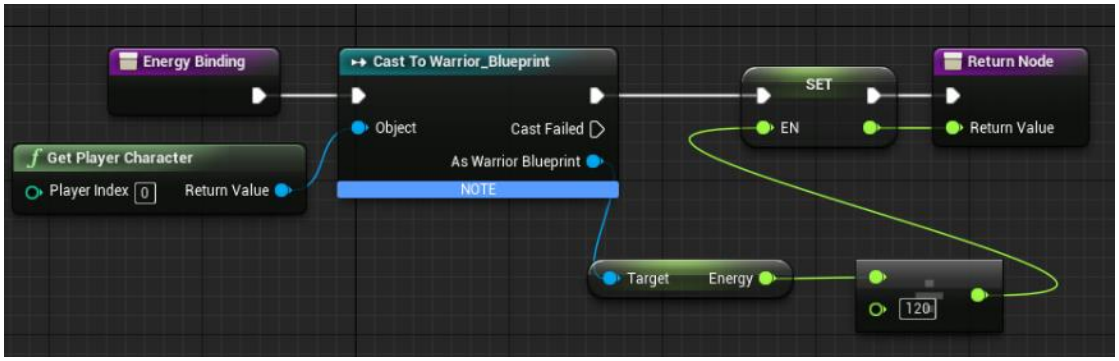
4.4 Λειτουργικότητα HUD

Για την δημιουργία του HUD δημιουργήθηκε ένα νέο blueprint κλάσης Widget Blueprint με όνομα UserWidgets, το οποίο χρησιμοποιείται για την υλοποίηση στοιχείων διεπαφής χρήστη. Έπειτα το καινούργιο blueprint ανοίγει αυτόματα με τον UMG UI Editor. Στη καρτέλα σχεδιαστή τοποθετούμε αρχικά δύο progress bar. Για να έχουμε το επιθυμητό αποτέλεσμα στα κουμπιά αφής, πρώτα τοποθετούμε στις θέσεις που θέλουμε να βρίσκονται τα κουμπιά, τις εικόνες που θα εμφανίζονται όταν ο παίχτης χρησιμοποιεί το μονό σπαθί. Έπειτα τοποθετούμε πάνω στις προηγούμενες εικόνες αυτές που θα εμφανίζονται όταν ο χρήστης χρησιμοποιεί την ασπίδα και το σπαθί. Έπειτα πάνω από τις εικόνες τοποθετούμε τα κουμπιά και στις ρυθμίσεις των κουμπιών τα οποία κάνουμε διαφανή. Για την καλύτερη διαρρύθμιση του χώρου οι εικόνες και τα κουμπιά τοποθετήθηκαν μέσα σε containers. Στην παρακάτω εικόνα στην καρτέλα ιεραρχίας φαίνεται η τελική κατάταξη των widgets.

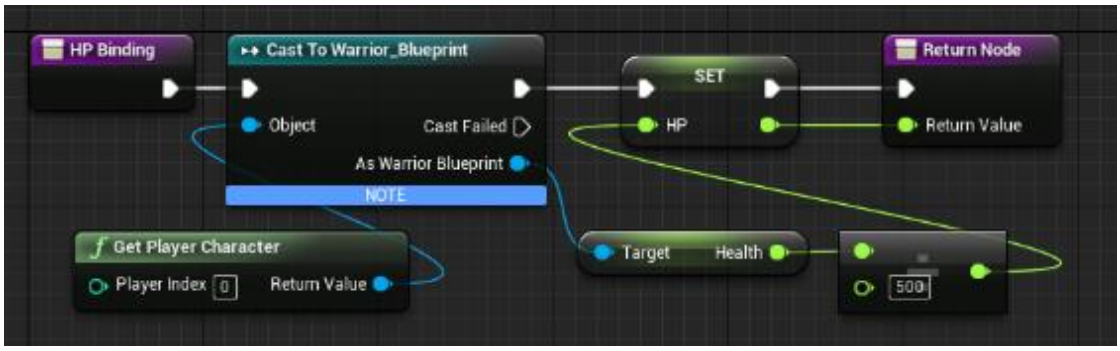


Έτσι όταν ο χρήστης πατήσει το κουμπί αλλαγής όπλων η εικόνα του μονού σπαθιού και η εικόνα της ασπίδας θα γίνουν διαφανείς και θα φαίνονται οι εικόνες που είναι τοποθετημένες από κάτω δηλαδή, η εικόνα της ασπίδας και του σπαθιού και η εικόνα της αποφυγής επιθέσεων. Επίσης όταν ο χρήστης πατήσει το κουμπί για να αναπληρώσει τη ζωή του, τότε η εικόνα θα γίνει επίσης αδιαφανής για να εμφανιστεί η εικόνα που υποδεικνύει στον παίχτη ότι δεν έχει περάσει ακόμα ο χρόνος των 30 δευτερολέπτων.

Για τις μπάρες της ζωής και της ενέργειας δημιουργούμε δύο συναρτήσεις η οποίες θα χειρίζονται το ποσοστό που θα έχει η κάθε μπάρα ανάλογα με την ζωή/ενέργεια του παίχτη. Εντός της συνάρτησης της ενέργειας χρησιμοποιούμε τον ειδικός κόμβος Cast To για να πάρουμε την ενέργεια του παίχτη. Έπειδή οι μπάρες λειτουργούν με ποσοστό επί τις 100 διαιρούμε την ενέργεια μας με την αρχική της τιμή και θέτουμε το αποτέλεσμα σε μια νέα μεταβλητή EN. Έπειτα επιστρέφουμε τη μεταβλητή EN.



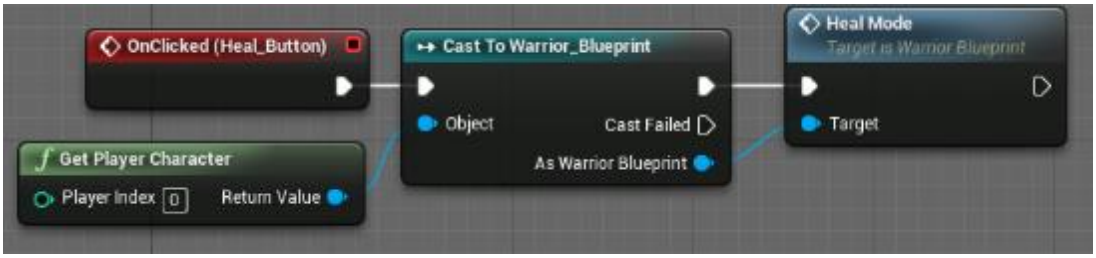
Η ίδια διαδικασία πραγματοποιήθηκε και για τη μπάρα της ζωής :



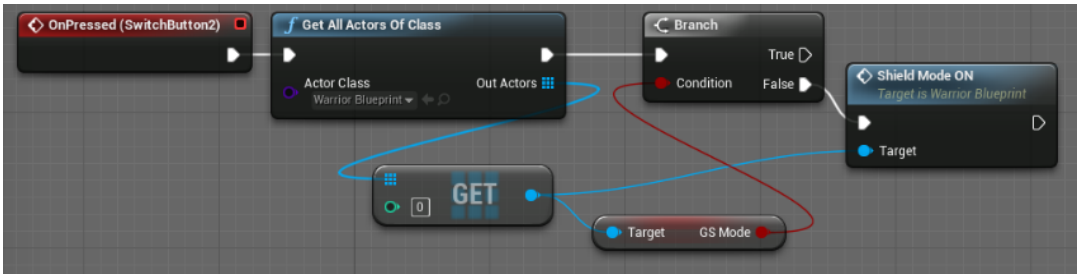
Στη συνέχεια μεταβαίνουμε στο γράφο του UserWidgets για να προσθέσουμε λειτουργικότητα στα κουμπιά του HUD.

Για το κουμπί αλλαγής όπλων προστέθηκε ένα γεγονός OnClicked το οποίο ενεργοποιείται όταν ο χρήστης πατήσει το κουμπί. Έπειτα με το κόμβο cast το καλούμε το γεγονός αλλαγής όπλων από το blueprint του πολεμιστή.

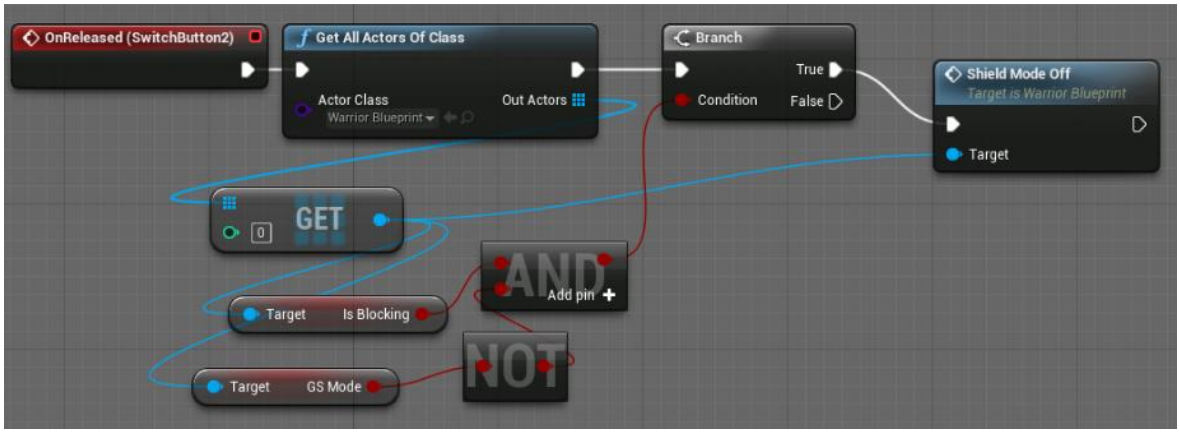
Για τα κουμπιά αναπλήρωσης ζωής και ειδικών επιθέσεων πραγματοποιήθηκε η ίδια διαδικασία:



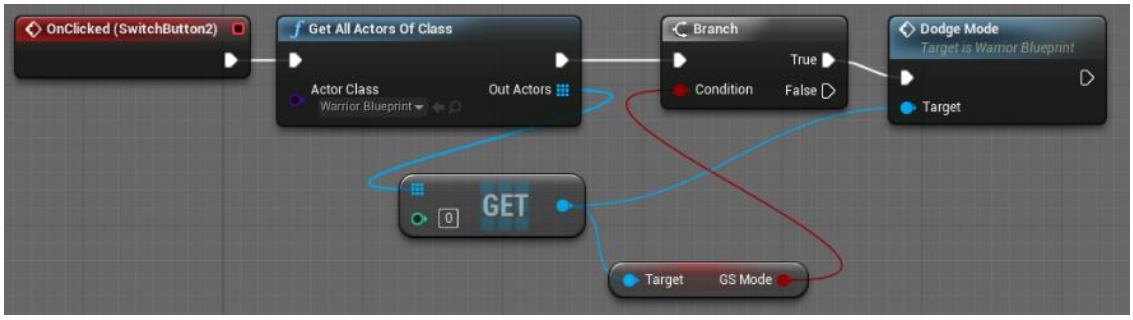
Για το κουμπί της λειτουργίας ασπίδας/αποφυγής επιθέσεων χρησιμοποιήθηκε αρχικά το γεγονός OnPressed το οποίο ενεργοποιείται στο συνεχόμενο πάτημα του κουμπιού από τον χρήστη. Έπειτα χρησιμοποιήθηκε η συνάρτηση Get All Actors of class με την οποία μπορούμε να πάρουμε γεγονότα, συναρτήσεις και μεταβλητές από ένα άλλο blueprint. Μετά κάνουμε ένα έλεγχο αν ο χρήστης χρησιμοποιεί το διπλό σπαθί. Αν δεν το χρησιμοποιεί ενεργοποιείται η λειτουργία ασπίδας.



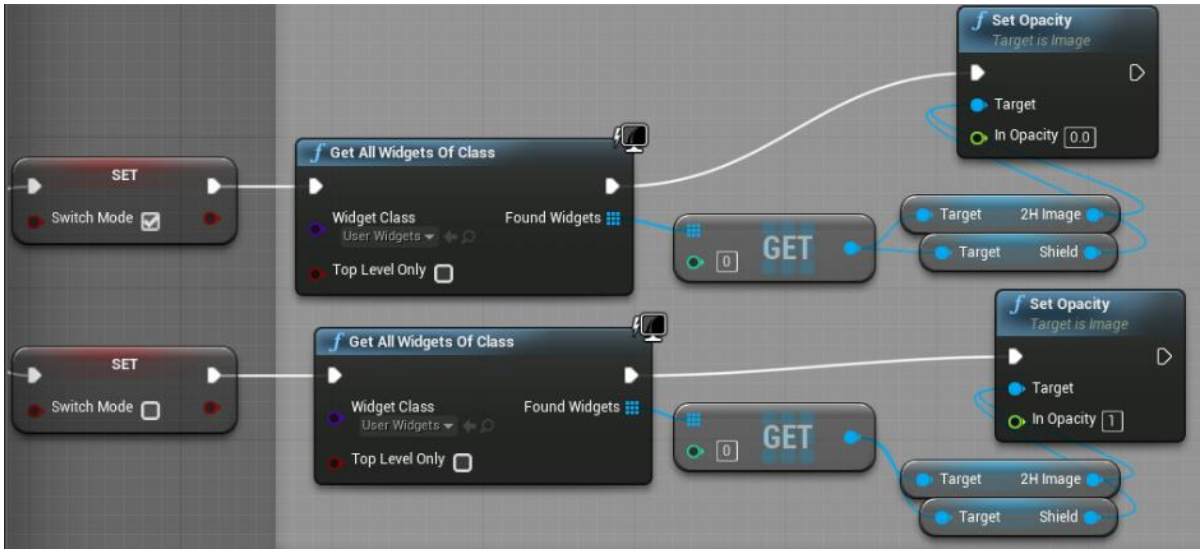
Επίσης δημιουργούμε ένα γεγονός OnReleased το οποίο ενεργοποιείται όταν ο χρήστης σταματήσει να πατάει το κουμπί λειτουργίας ασπίδας. Έπειτα κάνουμε έναν έλεγχο αν ο χρήστης χρησιμοποιεί την ασπίδα και αν δεν χρησιμοποιεί το διπλό σπαθί. Τέλος, απενεργοποιείται η λειτουργία ασπίδας.



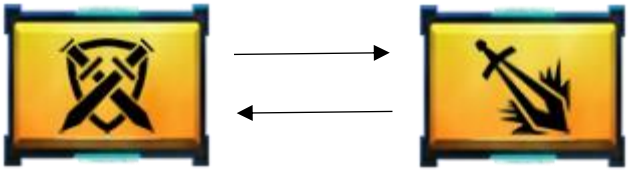
Για την αποφυγή επιθέσεων χρησιμοποιήθηκε ένα γεγονός OnClicked. Έπειτα πραγματοποιείται ένας έλεγχος αν ο χρήστης χρησιμοποιεί το μονό σπαθί. Αν το χρησιμοποιεί ενεργοποιείται η αποφυγή επιθέσεων.



Για την αλλαγή των εικόνων κατά την αλλαγή όπλων πρέπει εντός του warrior_blueprint στο τέλος του γεγονότος switch_modes να προστεθούν οι παρακάτω αλλαγές :

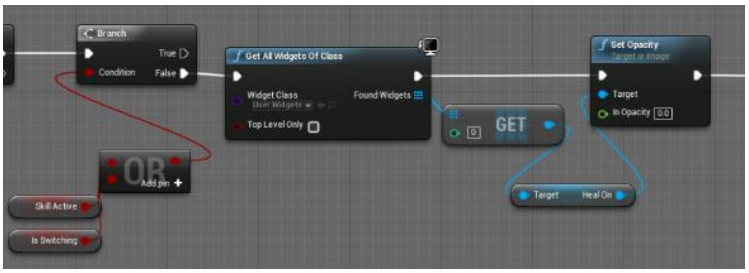


Με τις αλλαγές αυτές αλλάζουμε την διαφάνεια των εικόνων ανάλογα με ποια όπλα χρησιμοποιούμε. Το αποτέλεσμα είναι οι παρακάτω εικόνες.

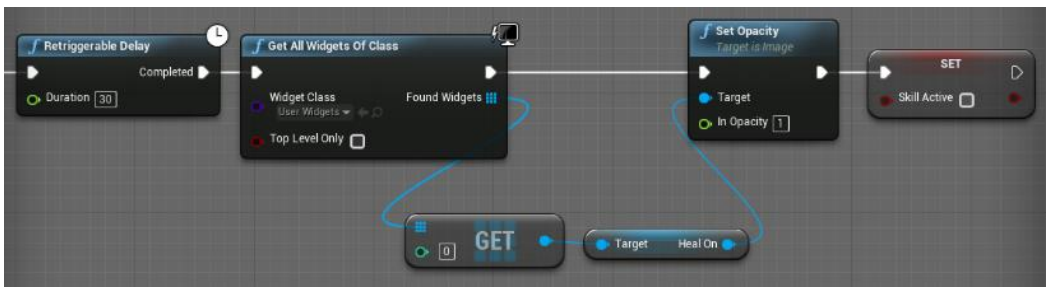


Για την αλλαγή των εικόνων κατά την αναπλήρωση ζωής πρέπει εντός του warrior_blueprint στο τέλος του γεγονότος Heal_Mode να προστεθούν οι παρακάτω αλλαγές.

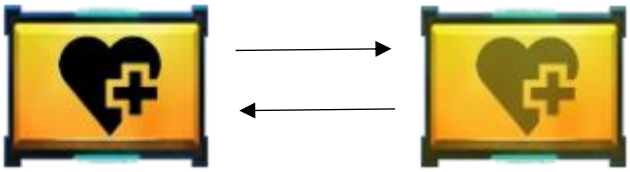
- Όταν ξεκινήσει η λειτουργία της αναπλήρωσης ζωής :



- Αφού περάσουν τα 30 δευτερόλεπτα :

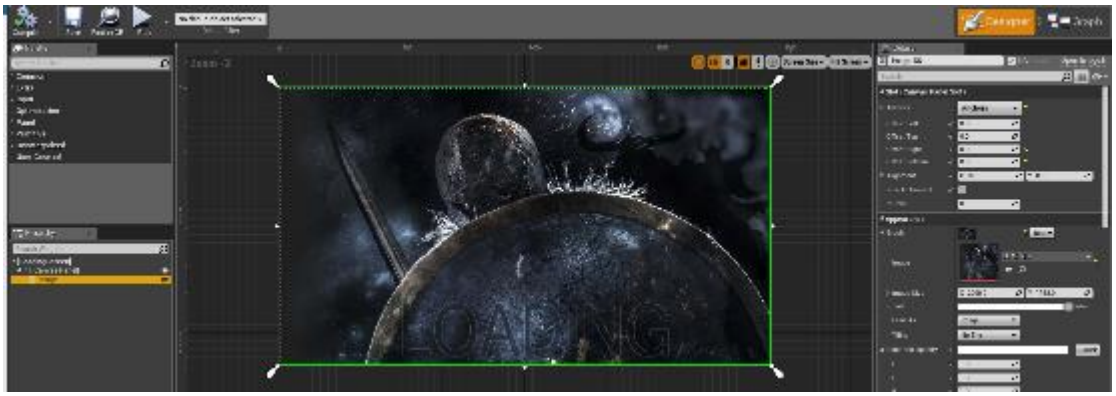


Το αποτέλεσμα είναι οι παρακάτω εικόνες :



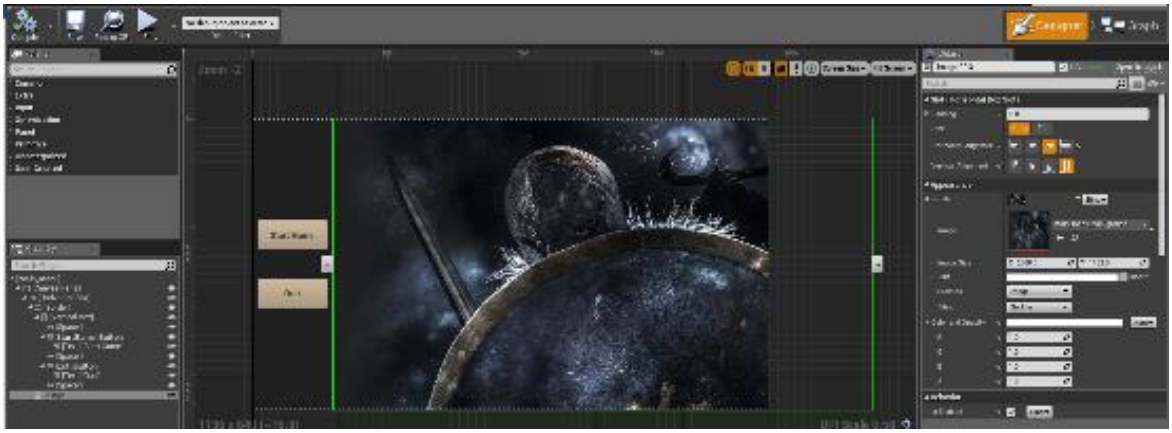
4.5 Κύριο Μενού, Οθόνη Φόρτωσης & Γενικές Ρυθμίσεις

Αρχικά θα δημιουργήσουμε την αρχική οθόνη που βλέπει ο χρήστης και έπειτα την οθόνη φόρτωσης. Αρχικά δημιουργούμε δύο widget blueprint. Το πρώτο widget blueprint ονομάζεται LoadingScreen και περιέχει την οθόνη φόρτωσης και το δεύτερο ονομάζεται Main_Menu και περιέχει το κεντρικό μενού. Στο blueprint LoadingScreen τοποθετούμε απλώς ένα Image widget, το οποίο περιέχει την εικόνα που θέλουμε να εμφανίζεται όποτε φορτώνει το παιχνίδι. Έπειτα μεγαλώνουμε το widget για να καλύπτει ολόκληρη την οθόνη. Το συγκεκριμένο Blueprint δεν θα περιέχει λειτουργικότητα εφόσον θέλουμε απλά να εμφανίζεται κατά την φόρτωση κάποιου επιπέδου.



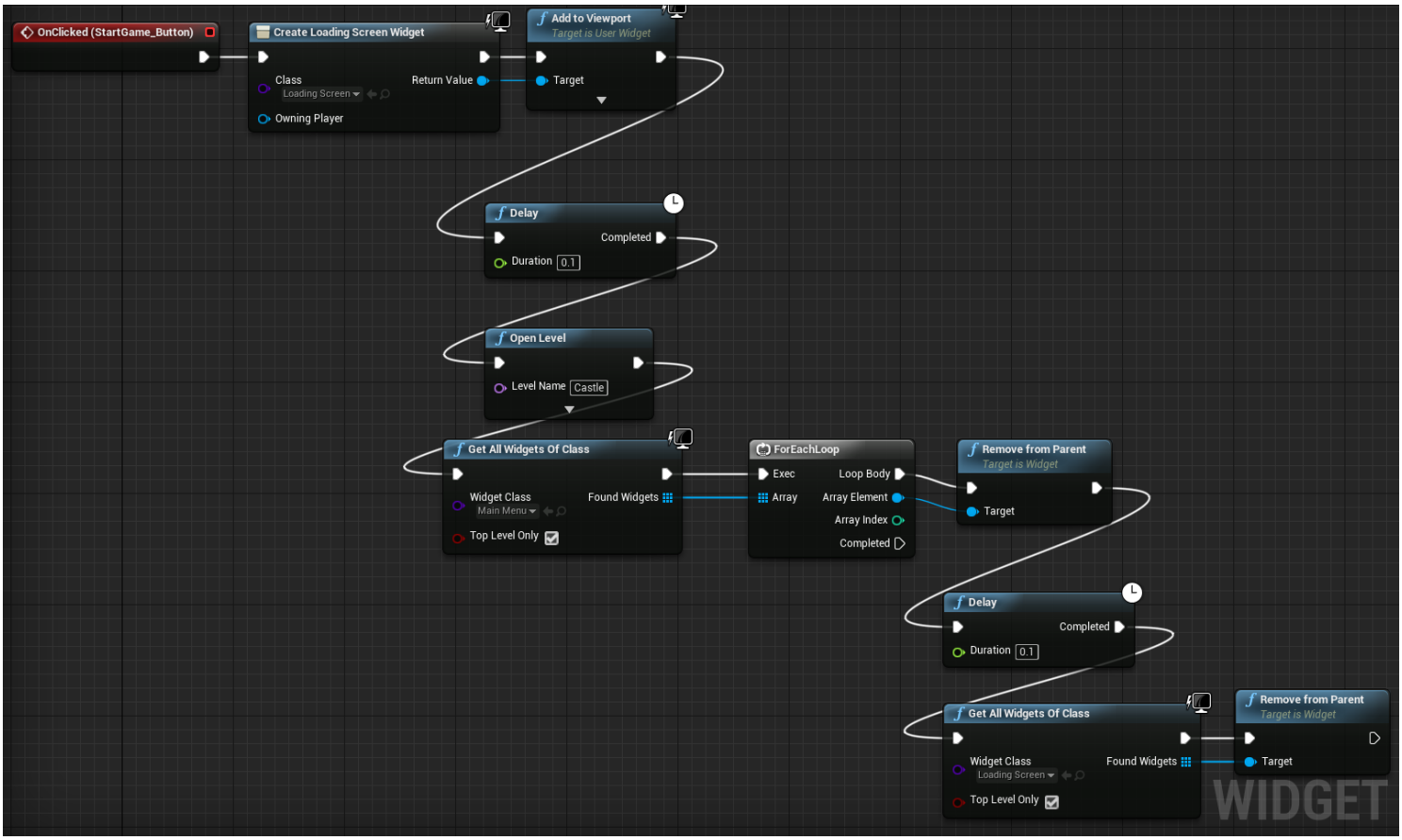
Η Σχεδίαση της οθόνης φόρτωσης.

Έπειτα στο Main_Menu Blueprint δημιουργούμε δύο widget κουμπιών και ένα Image widget. Έπειτα στο παράθυρο λεπτομερειών αλλάζουμε κάποιες ιδιότητες των widget που έχουμε τοποθετήσει όπως χρώμα, μέγεθος και το κείμενο των κουμπιών. Το κουμπί με το οποίο θα ξεκινάει το παιχνίδι ονομάστηκε StartGame_Button και το κουμπί με το οποίο θα κλείνει το παιχνίδι ονομάστηκε Exit_Button.

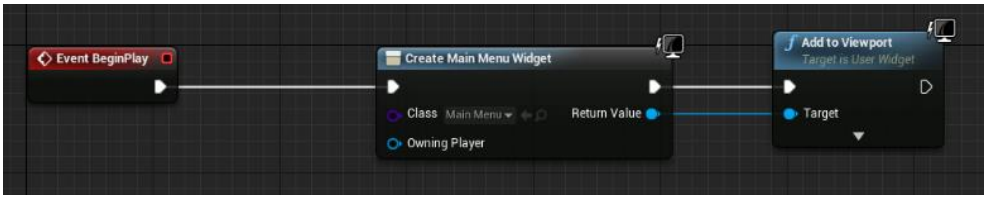


Εφόσον ολοκληρώθηκε το σχεδιαστικό κομμάτι πρέπει να προστεθεί η ανάλογη λειτουργικότητα στα δύο κουμπιά. Γυρνώντας στην καρτέλα Graph δημιουργούμε δύο γεγονότα OnClicked για το κάθε κουμπί. Έπειτα για το κουμπί Exit_Button καλούμε τη συνάρτηση Quit Game, η οποία κλείνει αυτόματα το παιχνίδι.

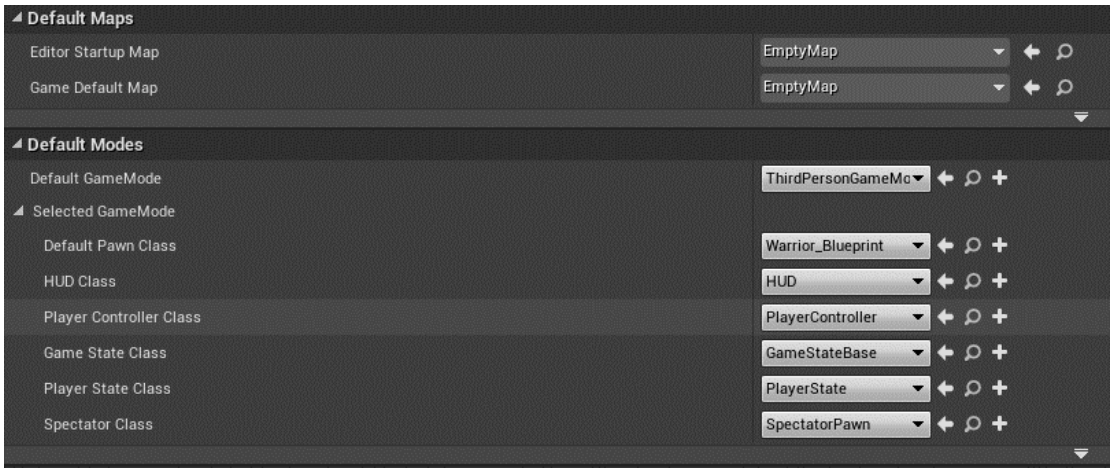
Για το κουμπί StartGame_button καλούμε τις παρακάτω συναρτήσεις οι οποίες κάθε φορά που καλούνται κάνουν τα εξής με διαδοχική σειρά, δημιουργούν τα widget της οθόνης φόρτωσης, τα προσθέτουν στο παιχνίδι, φορτώνει το επίπεδο Castle που είναι ο εξωτερικός χάρτης του κάστρου, επιλέγει τα widget της κύριας οθόνης, αφαιρεί τα widget της κύριας οθόνης, επιλέγει τα widget της οθόνης φόρτωσης και τέλος αφαιρεί τα widget της οθόνης φόρτωσης.



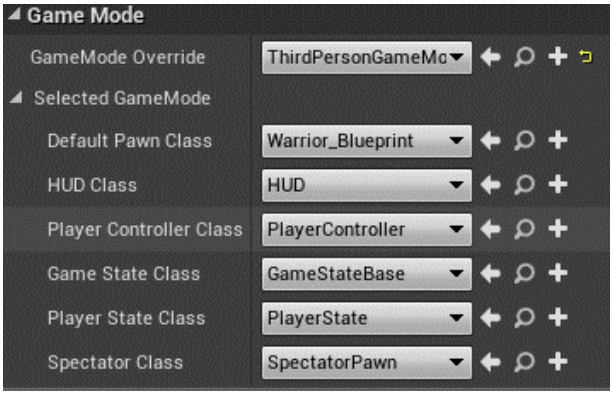
Έπειτα δημιουργούμε ένα κενό επίπεδο με όνομα EmptyMap, που θα είναι ο αρχικός χάρτης του παιχνιδιού και σε αυτό θα υπάρχει το αρχικό μενού. Μετά δημιουργούμε ένα νέο ειδικό blueprint κλάσης HUD το οποίο θα δημιουργεί το αρχικό μενού κάθε φορά που φορτώνεται ο άδειος χάρτης.



Το μόνο που έμεινε είναι να αλλάξουμε στις ρυθμίσεις της Unreal και στις ρυθμίσεις του χάρτη Castle τις βασικές κλάσεις που θα χρησιμοποιούν.



Γενικές ρυθμίσεις της Unreal

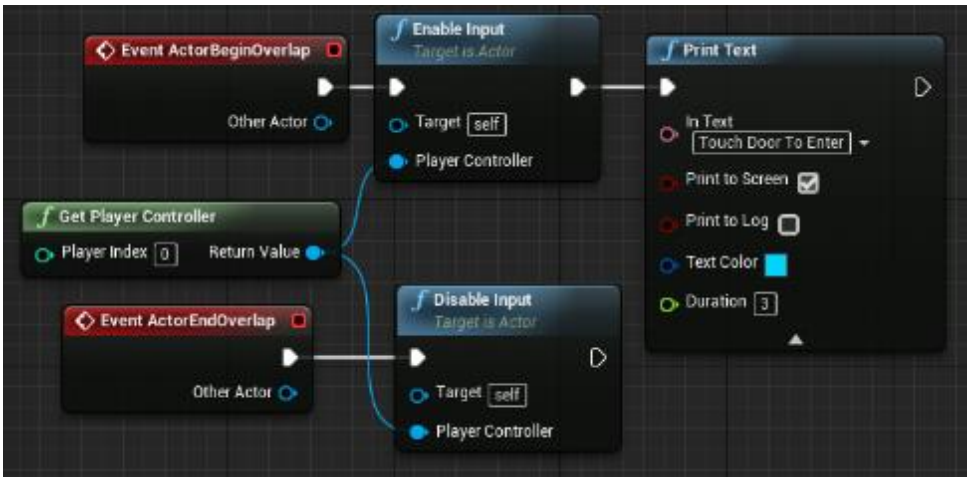


Όλες οι κλάσεις πλην της Warrior_Blueprint είναι οι προκαθορισμένες κλάσεις τις Unreal και χρησιμοποιήθηκαν διότι έχουν χρησιμοποιηθεί εντός του Warrior_Blueprint.

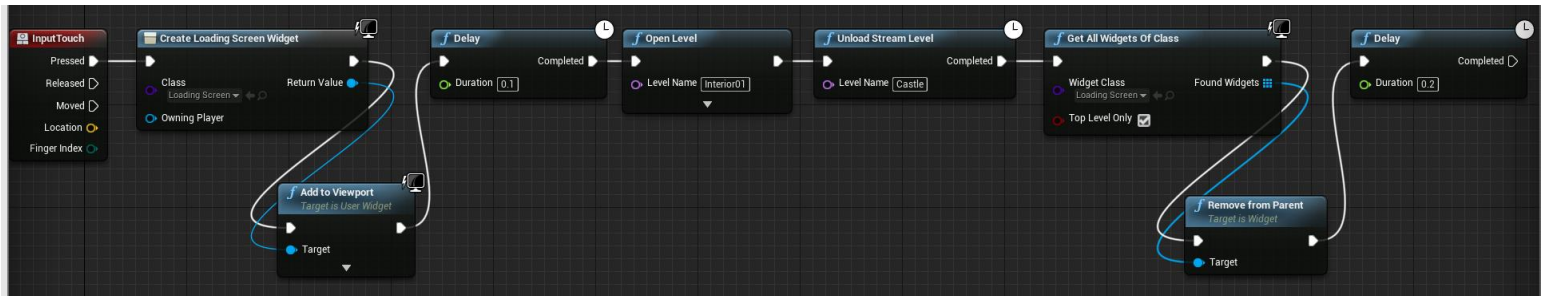
Οι ρυθμίσεις του Επιπέδου Castle

Στη συνέχεια δημιουργήθηκε το blueprint το οποίο θα είναι υπεύθυνο για τη μεταφορά του παίχτη από το εξωτερικό στο εσωτερικό του κάστρου. Στην καρτέλα viewport του νέου blueprint προσθέτουμε απλώς ένα κουτί το οποίο δεν είναι ορατό στο χρήστη. Έπειτα στο γράφο γεγονότων καλούμε τα γεγονότα ActorbeginOverlap και ActorEndOverlap. Με το πρώτο γεγονός όταν ο παίχτης μας εισέρχεται

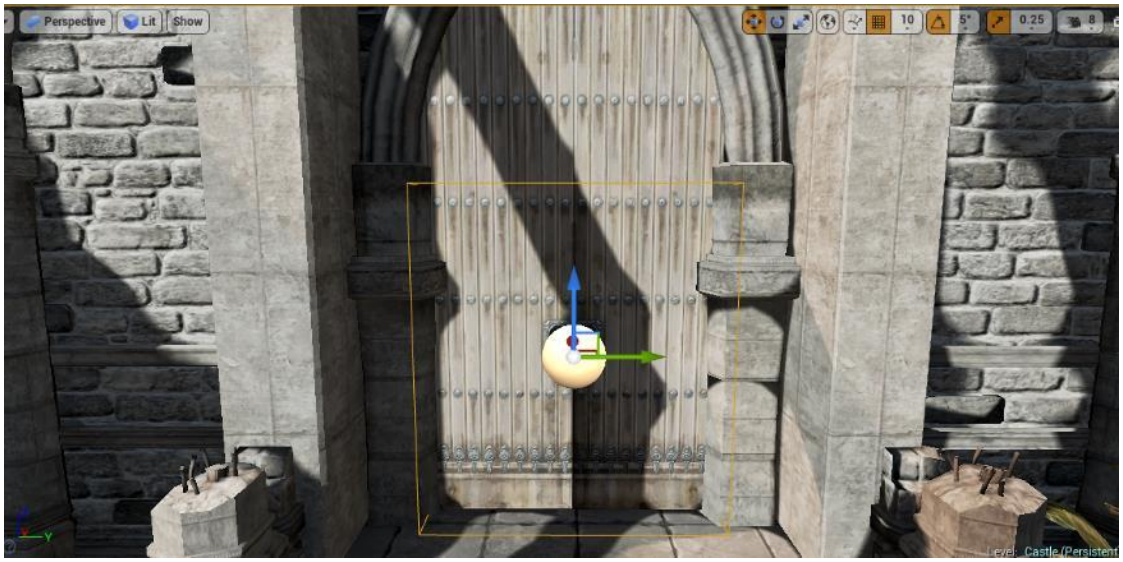
στο κουτί θα ενεργοποιείται όποιο κουμπί εισόδου χρησιμοποιήσουμε εντός του blueprint παρακάμπτοντας άλλα blueprints που χρησιμοποιούν τις ίδιες εισόδους, ενώ με το δεύτερο γεγονός απενεργοποιείται η είσοδος. Έπειτα εμφανίζεται στην οθόνη ένα μήνυμα ότι ο παίχτης μπορεί να εισέλθει στο εσωτερικό του κάστρου.



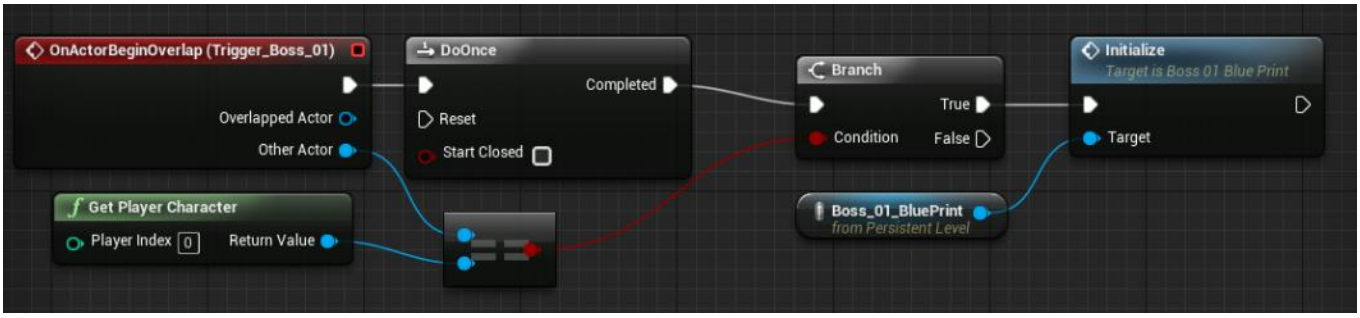
Στη συνέχεια χρησιμοποιήθηκε για είσοδος το απλό πάτημα στην οθόνη από τον χρήστη. Έπειτα καλούμε την οθόνη φόρτωσης και φορτώνουμε το εσωτερικό του κάστρου. Μετά ξεφορτώνουμε από τη μνήμη το εσωτερικό του κάστρου.



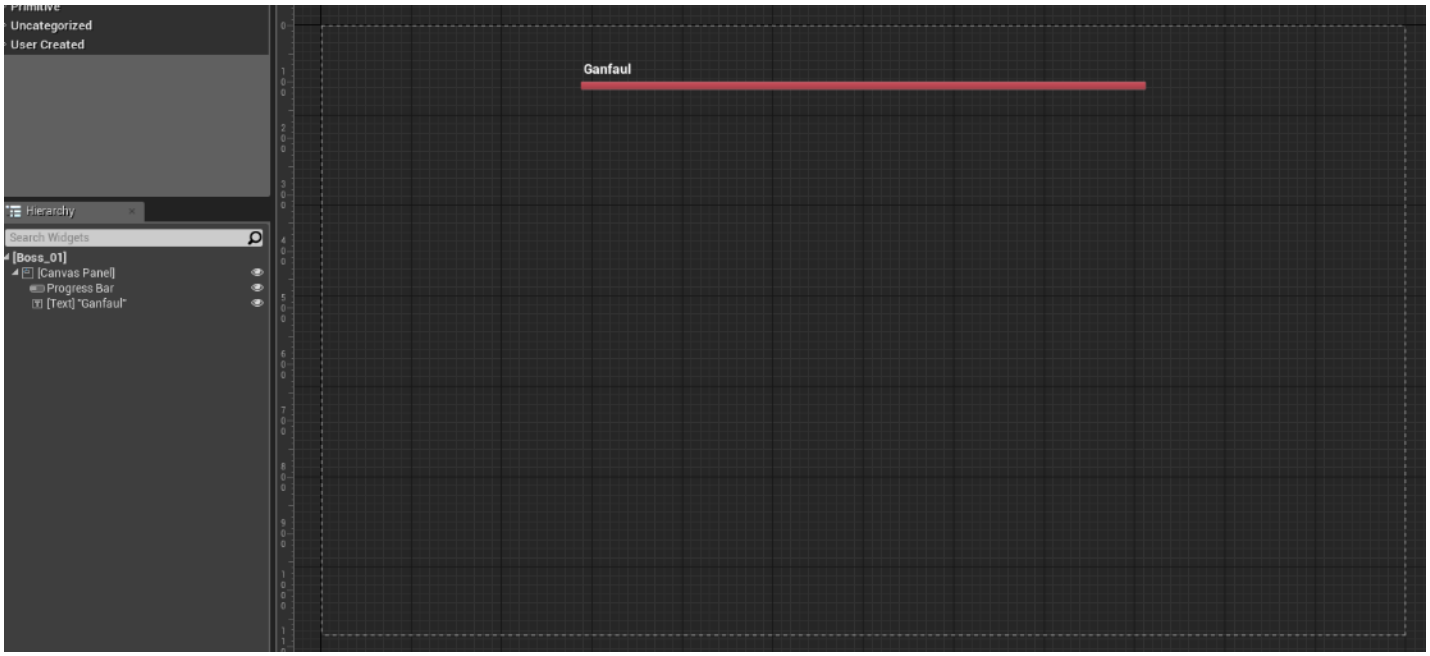
Επομένως τοποθετούμε το blueprint εντός του επιπέδου και το προσαρμόζουμε έτσι ώστε να καλύπτει το μεγαλύτερο μέρος της πόρτας που οδηγεί στο εσωτερικό του κάστρου.



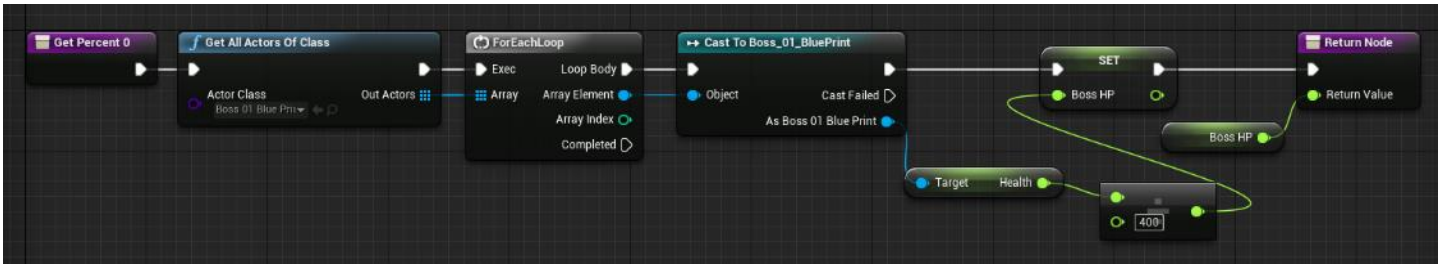
Τέλος, για να ξεκινήσει η λειτουργικότητα των αντιπάλων πρέπει σε κάθε Trigger Box που βρίσκεται λίγα μέτρα πριν από κάθε αντίπαλο να προσθέσουμε λειτουργικότητα. Ανοίγουμε το Level Blueprint του εξωτερικού του κάστρου, το οποίο δημιουργείται αυτόματα από την Unreal. Έπειτα για κάθε Trigger Box χρησιμοποιούμε το γεγονός OnActorBeginOverlap το οποίο ενεργοποιείται όταν ο παίχτης μας εισέλθει στο Trigger Box. Μετά χρησιμοποιούμε την μακροεντολή DoOnce έτσι ώστε η λειτουργικότητα των αντιπάλων να ξεκινήσει μόνο μια φορά. Έπειτα ελέγχουμε αν ο actor που εισήλθε στο Trigger box είναι ο παίχτης μας. Τέλος, καλούμε το γεγονός Initialize για να ενεργοποιηθεί ο αντίπαλος. Η παραπάνω διαδικασία είναι ίδια για όλους τους αντιπάλους.



Όσον αφορά το HUD των αντιπάλων δημιουργούμε ένα καινούργιο Widget blueprint για κάθε αντίπαλο. Έπειτα τοποθετούμε ένα progress bar και το προσαρμόζουμε ανάλογα στο πάνω μέρος της οθόνης.

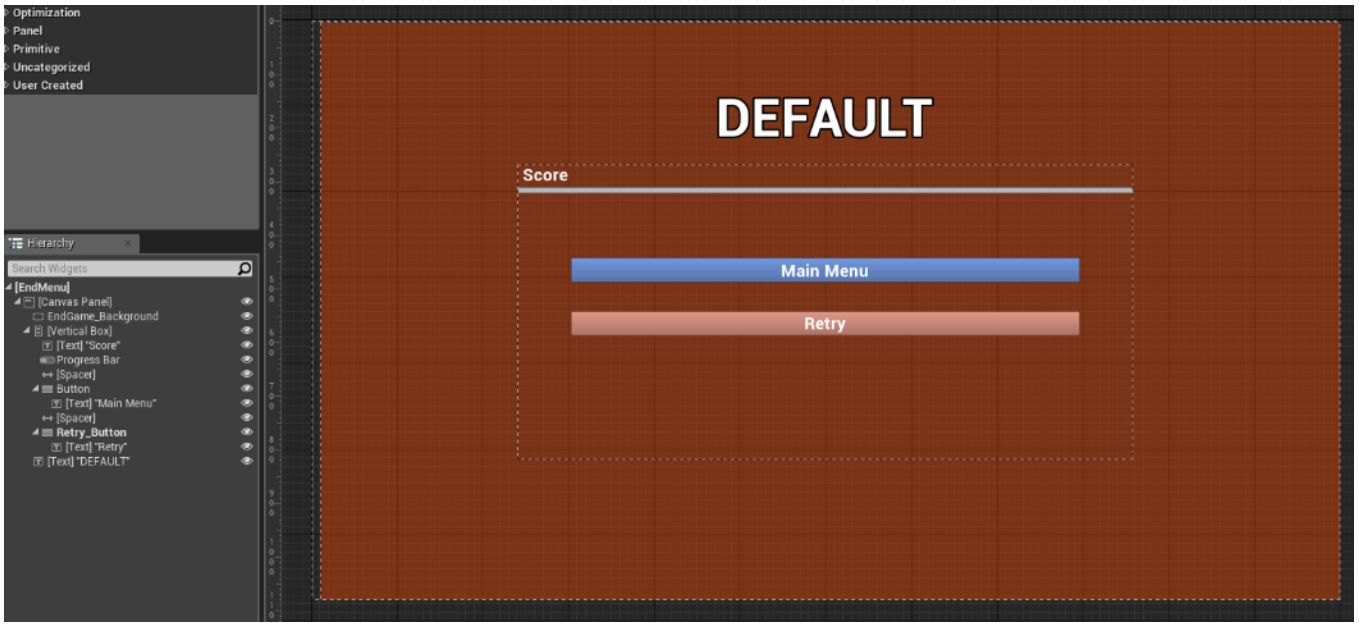


Έπειτα στο γράφο του blueprint δημιουργούμε μια συνάρτηση, η οποία θα υπολογίζει το ποσοστό της μπάρας. Εντός της συνάρτησης για να χρησιμοποιήσουμε μεταβλητές και γεγονότα από το blueprint ενός αντιπάλου πρέπει να χρησιμοποιήσουμε μια επανάληψη ForEachloop. Στη συνέχεια διαιρούμε τη ζωή του αντιπάλου με την αρχική της τιμή και θέτουμε το αποτέλεσμα σε μια νέα μεταβλητή Boss HP. Έπειτα επιστρέφουμε τη μεταβλητή Boss Hp.

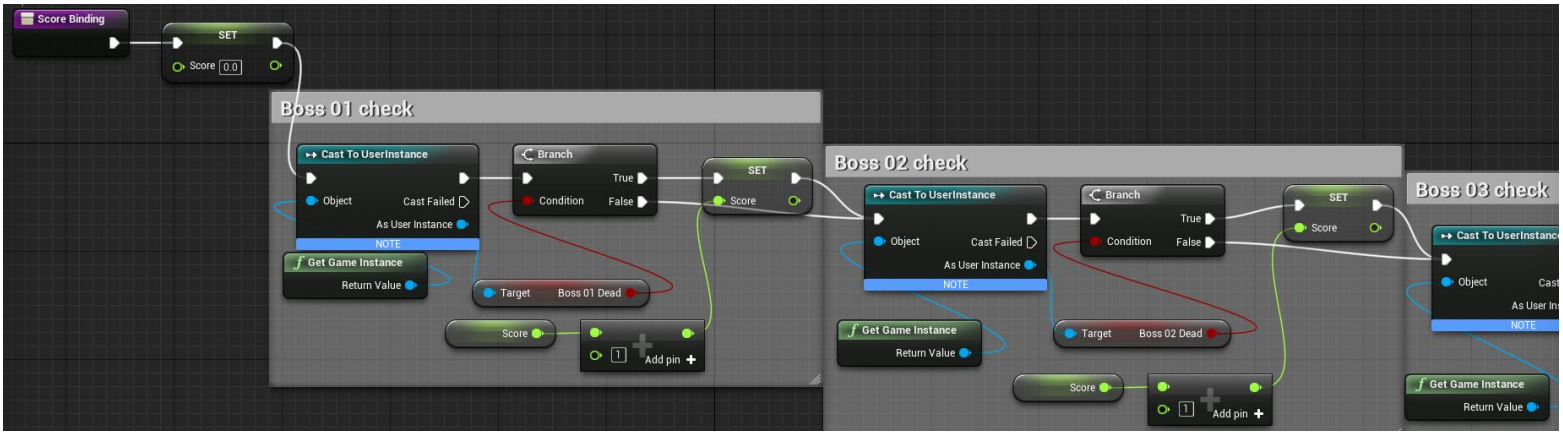


4.6 Λειτουργικότητα Τελικής Οθόνης

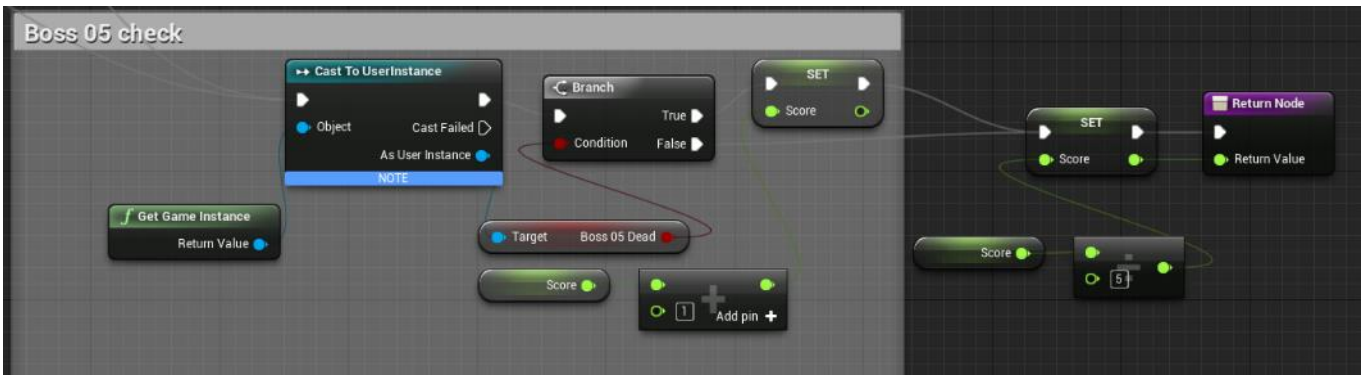
Για την τελική οθόνη δημιουργούμε ένα καινούργιο Widget Blueprint με όνομα EndMenu. Στη συνέχεια προσθέτουμε ένα border widget και το προσαρμόζουμε να καλύπτει όλη την οθόνη. Έπειτα προσθέτουμε ένα text box, μια μπάρα και δύο κουμπιά και τα προσαρμόζουμε όπως θέλουμε. Η τελική διαρρύθμιση φαίνεται στη παρακάτω εικόνα.



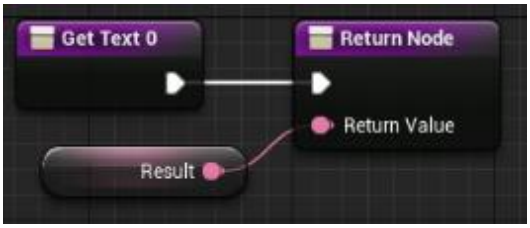
Στη συνέχεια δημιουργούμε μια συνάρτηση, η οποία θα είναι υπεύθυνη για τον υπολογισμό του score. Αρχικά δημιουργούμε μια μεταβλητή Score και τη μηδενίζουμε. Έπειτα θέλουμε για κάθε αντίπαλο να ελέγχουμε αν η μεταβλητή που βρίσκεται στο UserInstance είναι αληθής ή ψευδής. Αν είναι αληθής προσθέτουμε μια μονάδα στο score αλλιώς αν είναι ψευδής συνεχίζουμε στον έλεγχο για τον επόμενο αντίπαλο.



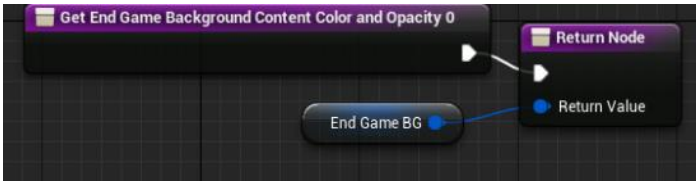
Τέλος, διαιρούμε το score με το 5 διότι οι μπάρες λειτουργούν με ποσοστό επί τις 100.



Έπειτα για να αλλάξουμε το κείμενο του Text Box που προσθέσαμε ανάλογα με το αποτέλεσμα δημιουργούμε μια συνάρτηση μέσα στην οποία δημιουργούμε μια μεταβλητή Result και την επιστρέφουμε.

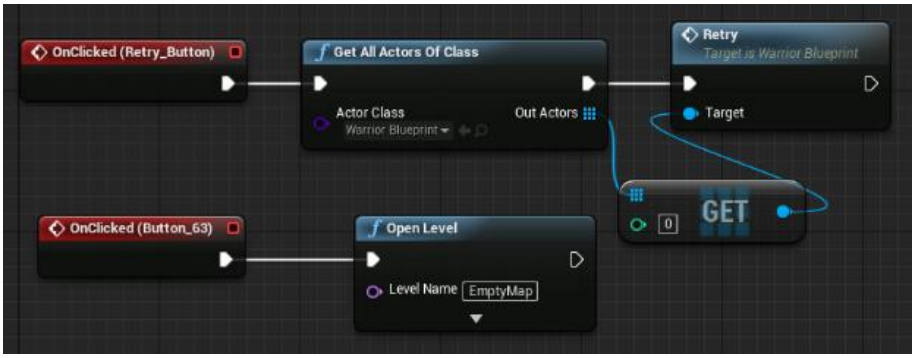


Επίσης για να αλλάξουμε το χρώμα του border ακολουθούμε την ίδια διαδικασία.



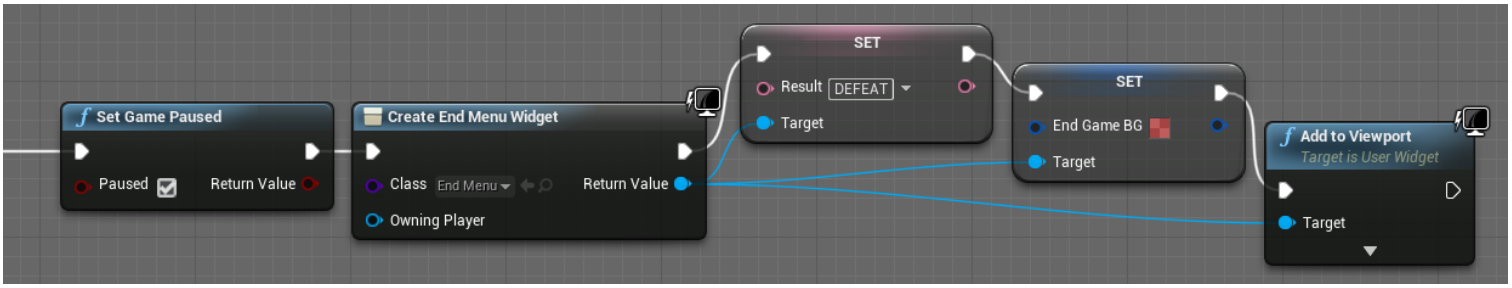
Στην συνέχεια μεταβαίνουμε στο γράφο του EndMenu για να προσθέσουμε λειτουργικότητα στα κουμπιά που προσθέσαμε. Για το κουμπί Retry καλούμε ένα γεγονός που θα δημιουργήσουμε εντός του warrior_blueprint.

Για το κουμπί που επιστρέφει το χρήστη στο αρχικό μενού το μόνο που χρειάζεται να κάνουμε είναι να φορτώσουμε τον κενό χάρτη που περιέχει το αρχικό μενού.

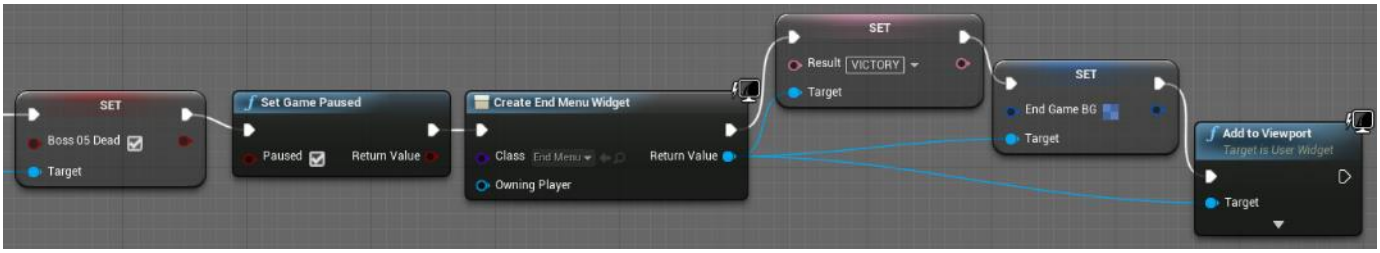


Επομένως μεταβαίνουμε στο warrior_blueprint και δημιουργούμε το γεγονός Retry. Έπειτα πρέπει να διακόψουμε την παύση του παιχνιδιού που συμβαίνει όταν ο παίχτης νικήσει ή χάσει το παιχνίδι. Μετά επιλέγουμε και βγάζουμε το EndMenu από την οθόνη. Τέλος, φορτώνουμε το επίπεδο Castle.

Έπειτα εντός του Warrior_Blueprint όταν ο παίχτης μας ηττηθεί πραγματοποιούμε τις ανάλογες αλλαγές στο τελικό μενού. Οπότε στο τέλος του γραφήματος που ο παίχτης μας δέχεται ζημιά πρέπει να αλλάξουμε το κείμενο του Text box από “Default” σε “Defeat”. Έπειτα αλλάζουμε το χρώμα του border σε κόκκινο και προσθέτουμε το EndMenu στην οθόνη.



Εφόσον ο μόνος τρόπος για να πραγματοποιηθεί επιτυχώς το παιχνίδι είναι να ηττηθεί ο τελευταίος αντίπαλος πρέπει στο γράφημα που δέχεται ζημιά ο τελευταίος αντίπαλος να πραγματοποιήσουμε τις ανάλογες αλλαγές στο τελικό μενού. Αρχικά καλούμε και ενεργοποιούμε τη συνάρτηση Set Game Paused για να παγώσει τελείως το παιχνίδι. Έπειτα αλλάζουμε το κείμενο του Text box από “Default” σε “Victory”. Έπειτα αλλάζουμε το χρώμα του border σε μπλε και προσθέτουμε το EndMenu στην οθόνη.



Κεφάλαιο 5

Απόδοση, Επεκτασιμότητα & Επίλογος

5.1 Ρυθμίσεις Απόδοσης

Όπως έχει ήδη αναφερθεί τα παιχνίδια που είναι φτιαγμένα με Unreal Engine λόγω των πολύ καλών γραφικών είναι πολύ βαριά για τις περισσότερες συσκευές Android. Σε αυτήν την ενότητα θα εξετάσουμε τεχνικές, οι οποίες καλυτερεύουν την απόδοση του παιχνιδιού ή μειώνουν την ποιότητα των γραφικών.

Όπως αναφέραμε στην ενότητα 2.1 τα static meshes είναι πολυγωνικά πλέγματα 3d αντικειμένων. Αυτό σημαίνει ότι οι πλευρές ενός static mesh αποτελούνται από τρίγωνα, τετράπλευρα ή κυρτά πολύγωνα. Ανάλογα με το πόσα πολύγωνα έχει κάθε πλευρά ενός static mesh γίνεται όλο και πιο δύσκολο η αναπαράσταση του από μια συσκευή android. Οπότε πριν την εισαγωγή ενός πολύπλοκου γεωμετρικά static mesh έγινε μια τροποποίηση του σε πρόγραμμα 3d modelling ώστε να μειωθεί ο αριθμός των πολυγώνων του αντικειμένου.

Μια από της καλύτερες τεχνικές για να κερδίσει απόδοση το παιχνίδι μας είναι η απενεργοποίηση του συστήματος σύγκρουσης (collision) από όλα τα αντικείμενα που δεν έρχονται σε επαφή με τον παίχτη μας. Το σύστημα σύγκρουσης λειτουργεί παρόμοια με τα static meshes αφού υλοποιεί ίδιες τεχνικές για την δημιουργία πολύπλοκων

συστημάτων σύγκρουσης. Οπότε για τα περισσότερα αντικείμενα του παιχνιδιού που έχουν σύστημα σύγκρουσης χρησιμοποιήθηκαν πολύ απλά γεωμετρικά σχήματα.

Μια από τις πιο απαιτητικές λειτουργίες της Unreal είναι η αναπαραγωγή συστημάτων σωματιδίων. Αν και σε συσκευές android τα συστήματα σωματιδίων είναι πολύ περιορισμένα μπορούν παρόλα αυτά τα προκαλέσουν μεγάλη πτώση στην απόδοση. Οπότε στο παιχνίδι έχουν υλοποιηθεί λίγα συστήματα σωματιδίων τα οποία έχουν απλά εφέ.

Τέλος στην υπό-ενότητα 2.6.4 αναφέρθηκε η χρησιμότητα των Device Profiles. Παρακάτω είναι οι ρυθμίσεις που χρησιμοποιήθηκαν για την καλύτερη δυνατή απόδοση του παιχνιδιού :

- r.mobileContentScaleFactor = 0.25, Είναι ο συντελεστής κλίμακας για να κλιμακώσουμε την ανάλυση του παιχνιδιού ώστε να ταιριάζει καλύτερα στην ανάλυση κινητού.
- r.BloomQuality = 0, Το bloom χρησιμοποιείται κυρίως σε υπολογιστές για να προσθέσει ρεαλισμό στο φωτισμό. Όμως είναι αρκετά απαιτητικό οπότε έχει απενεργοποιηθεί.
- r.depthOfField = 0, Χρησιμοποιείται για να βελτιώσει την ποιότητα γραφικών αντικειμένων που βρίσκονται στο βάθος ενός επιπέδου. Έχει απενεργοποιηθεί επειδή είναι απαιτητικό από το σύστημα.
- r.LightShaftQuality = 0, Χρησιμοποιείται από directional lights για να δημιουργήσει ατμοσφαιρικές σκιές. Έχει απενεργοποιηθεί επειδή είναι άκρως απαιτητικό από το σύστημα.
- r.refractionQuality = 0, Χρησιμοποιείται για τη διάθλαση όταν το φως έρχεται σε επαφή με επιφάνειες, όπως νερό και γρασίδι. Έχει απενεργοποιηθεί επειδή είναι άκρως απαιτητικό από το σύστημα
- r.shadowQuality = 1, Είναι η ποιότητα γραφικών των σκιών. Έχει ρυθμιστεί για μέτρια γραφικά.

- r.materialQualitylevel = 0, Είναι η ποιότητα γραφικών των materials. Έχει ρυθμιστεί για χαμηλά γραφικά.

5.2 Επεκτασιμότητα

Το παιχνίδι που δημιουργήθηκε μπορεί να είναι απλό και σύντομο, αλλά είναι έτσι κατασκευασμένο που μπορεί να επεκταθεί πολύ εύκολα. Η προσθήκη νέων λειτουργιών στον παίχτη μας, όπως καινούργιες επιθέσεις, μπορούν να προστεθούν στις ήδη υπάρχουσες με μεγάλη ευκολία. Επίσης μπορούν να προστεθούν καινούργιοι εχθροί με βάση τους δημιουργημένους. Επίσης το παιχνίδι μπορεί να βελτιωθεί δημιουργώντας περισσότερα επίπεδα με μεγαλύτερες προκλήσεις για τον χρήστη. Επιπρόσθετα, θα μπορούσαν να προστεθούν επίπεδα στο παίχτη μας και στους αντιπάλους με ένα σύστημα experience για να ανεβαίνει ο παίχτης μας επίπεδο.

Ακόμα θα μπορούσε να προστεθεί διάλογος και ένα σύστημα αποστολών με επιλογές και ανταμοιβές, όπως συνηθίζεται σε παρόμοια παιχνίδια.

Τέλος, ολόκληρη η υλοποίηση της λειτουργικότητας του παιχνιδιού θα μπορούσε να μετατραπεί σε C++ έτσι ώστε το παιχνίδι να έχει την βέλτιστη απόδοση.

5.3 Επίλογος

Παρόλο που η εργασία έφτασε στο τέλος της υπάρχουν σκέψεις για την μελλοντική δημιουργία ενός παρόμοιου παιχνιδιού. Η διαδικασία κατασκευής ενός παιχνιδιού είναι αρκετά διασκεδαστική αλλά και συγχρόνως πολύ χρονοβόρα. Παρόλα αυτά μέσα σε ένα πολύ σύντομο χρονικό διάστημα χωρίς προηγούμενη γνώση στον τομέα κατασκευής παιχνιδιών δημιούργησα ένα ολοκληρωμένο τρισδιάστατο παιχνίδι χρησιμοποιώντας τα ίδια λογισμικά με αυτά που χρησιμοποιούν οι μεγαλύτερες εταιρείες στον κόσμο. Αν και δεν έχει τις ίδιες δυνατότητες

με τα παιχνίδια της εποχής πλέον ξέρω τι χρειάζεται για να γίνει μια τέτοια κατασκευή.

Βιβλιογραφία

Εισαγωγή στην C++	Kris Jamsa, 2007, Εκδόσεις ''Κλειδάριθμος''
C++ προγραμματισμός	Deitel Paul J., Deitel Harvey M., 2010, Εκδόσεις ''Γκιούρδας''
Τεχνητή Νοημοσύνη – Μια Σύγχρονη προσέγγιση	Stuart Russel, Peter Norvig, 2005, Εκδόσεις ''Κλειδάριθμος''
Learning C++ by Creating Games with UE4	William Sherif, 2015, Publisher ''PACKT''
Learning Unreal Engine Android Game Development	Nitish Misra, 2015, Publisher ''PACKT''
Master the Art of Unreal Engine 4 - Blueprints	Ryan Shah, 2015, Kindle Edition
Τεχνολογίες Πολυμέσων	Φώτης Λαζαρίνης, 2007,Εκδόσεις ''Κλειδάριθμος''
Adobe Photoshop CC Βήμα προς Βήμα	Adobe Creative Team, 2013, Εκδόσεις "Γκιούρδας Μ."
Χρήσιμα Έγγραφα της Unreal Engine 4	https://docs.unrealengine.com
Μηχανή παιχνιδιού (Game engine)	http://en.wikipedia.org/wiki/Game_engine
Βικιπαίδεια της Unity	http://wiki.unity3d.com/index.php/Main_Page
Εγχειρίδιο Construct 2	https://www.scirra.com/manual/1/construct-2
Χρήσιμα Έγγραφα της Godot	http://docs.godotengine.org/en/stable/
Action - Adventure	https://en.wikipedia.org/wiki/Action-adventure_game
Παιχνίδια τρίτου προσώπου	https://en.wikipedia.org/wiki/Third-person_shooter
Ηλεκτρονικό κατάστημα της Unreal	https://www.unrealengine.com/marketplace
Χρήσιμα Έγγραφα - Blender	https://docs.blender.org/manual/en/dev/
Χρήσιμα Έγγραφα - 3ds Max	http://docs.autodesk.com/3DSMAX/15/ENU/3ds-Max-Help/
Καρτεσιανό σύστημα συντεταγμένων	https://el.wikipedia.org/wiki/Καρτεσιανό_σύστημα_συντεταγμένων