ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και υλοποίηση παιχνιδιού 2d platform fighter με αυξανόμενα επίπεδα δυσκολίας και τεχνητή νοημοσύνη

Γιαλαμπουκίδης Καρυστινός Αδαμάντιος
Α.Μ. 71347685

Εισηγητής: Τρούσσας Χρήστος, Επίκουρος Καθηγητής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ 2D PLATFORM FIGHTER ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΑΥΞΑΝΟΜΕΝΑ ΕΠΙΠΕΔΑ ΔΥΣΚΟΛΙΑΣ ΚΑΙ ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΝΗ

Γιαλαμπουκίδης Καρυστινός Αδαμάντιος
Α.Μ. 71347685

Εισηγητής: Τρούσσας Χρήστος, Επίκουρος Καθηγητής

Εξεταστική Επιτροπή:

Ονοματεπώνυμο	Ιδιότητα	Ψηφιακή Υπογραφή
Χρήστος Τρούσσας	**Επίκουρος Καθηγητής**	Christos Troussas Digitally signed by Christos Troussas Date: 2024.09.30 15:52:00 +03'00'
Φοίβος Μυλωνάς	**Αναπληρωτής Καθηγητής**	Φοίβος-Απόστολος Μυλωνάς - Ψηφιακά υπογεγραμμένο 2024.09.30 17:21:16 +03'00'
Ακριβή Κρούσκα	**Μέλος ΕΔΙΠ**	Akrivi Krouska Digitally signed by Akrivi Krouska Date: 2024.09.30 17:39:29 +03'00'

Ημερομηνία εξέτασης: Σεπτέμβριος 2024

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

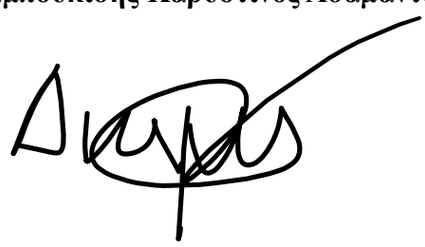
Ο κάτωθι υπογεγραμμένος Γιαλαμπουκίδης Καρυστινός Αδαμάντιος του Μάριου, με αριθμό μητρώου 71347685 φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

«Βεβαιώνω ότι είμαι συγγραφέας της παρούσας διπλωματικής εργασίας και ότι έχω αναφέρει ή παραπέμψει σε αυτή, ρητά και συγκεκριμένα, όλες τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, προτάσεων ή λέξεων, είτε αυτές μεταφέρονται επακριβώς (στο πρωτότυπο ή μεταφρασμένες) είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για την συγκεκριμένη διπλωματική εργασία»

Ο/Η Δηλών/ούσα

Γιαλαμπουκίδης Καρυστινός Αδαμάντιος



ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διπλωματική εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό της επεξεργασίας κειμένου. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, τον οποίο θα ήθελα να ευχαριστήσω.

Ακόμα θα ήθελα να ευχαριστήσω την οικογένειά μου για τη συμπαράσταση κατά τη διάρκεια των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία ασχολείται με τη δημιουργία ενός ολοκληρωμένου παιχνιδιού 2d platform fighter παιχνιδιού με τη χρήση της μηχανής σχεδιασμού UE5 και της γλώσσας C++. Αρχικά αναφέρεται στο θεωρητικό κομμάτι που πρέπει να γνωρίζει κανείς για τον σχεδιασμό στην γλώσσα C++ και τα πλεονεκτήματα μιας τέτοιας γλώσσας υψηλού επιπέδου, καθώς διευκολύνει τους προγραμματιστές στο έργο τους και είναι από τις πιο δημοφιλής γλώσσες για σχεδιασμό παιχνιδιών. Ακόμα γίνεται μία σύντομη περιγραφή της UE5 και όλων των σύγχρονων και εύκολων στην χρήση εργαλείων που προσφέρει μία τέτοια μηχανή για την απλοποίηση της διαδικασίας του σχεδιασμού σε επίπεδο σχεδίασης γραφικών, φωτισμού, σχεδίασης διεπαφών, βίντεο, φυσικής, ρεαλισμού αλλά και προγραμματισμού. Η UE5 διαθέτει πολλά αυτοματοποιημένα εργαλεία που όπως το ray tracing και πολλούς φιλικούς προς τον χρήστη editor για κάθε τι χρειαστεί ένας προγραμματιστής. Στη συνέχεια δίνεται μεθοδολογία για την ανάπτυξη ενός παιχνιδιού για νέους σχεδιαστές που επιθυμούν να μάθουν πως μπορεί κανείς να υλοποιήσει ένα τέτοιο πρότζεκτ βήμα – βήμα. Μετέπειτα γίνεται αναλυτική επεξήγηση του σχεδιασμού του παιχνιδιού ανά στάδιο δημιουργίας και η αξιολόγησή του από παίχτες – δοκιμαστές. Τέλος παρουσιάζονται τα συμπεράσματα που αποκομίστηκαν από το σύνολο της εργασίας από την οπτική γωνία του δημιουργού της.

ABSTRACT

This thesis focuses on the creation of a complete 2D platform fighter game using the Unreal Engine 5 (UE5) and the C++ programming language. Initially, it discusses the theoretical aspects that one needs to understand regarding design in C++ and the advantages of such a high-level language, as it facilitates developers in their work and is one of the most popular languages for game design. It also provides a brief description of UE5 and all the modern and user-friendly tools that this engine offers to simplify the design process at the level of graphics design, lighting, interface design, video, physics, realism, and programming. UE5 has many automated tools like ray tracing and numerous user-friendly editors for everything a developer might need. The methodology for developing a game is then provided for new designers who wish to learn how to implement such a project step-by-step. Subsequently, there is a detailed explanation of the game's design process at each stage of creation and its evaluation by playtesters. Finally, the conclusions drawn from the entirety of the work are presented from the creator's perspective.

ΕΠΙΣΤΗΜΟΝΙΗ ΠΕΡΙΟΧΗ: Σχεδιασμός γραφικών ,προγραμματισμός υπολογιστών, τεχνητή νοημοσύνη, αλληλεπίδραση ανθρώπου υπολογιστή, έξυπνη εφαρμογή, σχεδιασμός παιχνιδιών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:2D platformer, AI, Level progression, αυξανόμενη δυσκολία, UE5,game development, animations, editor, flipbook, κλάση, συνάρτηση.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΛΕΞΙΛΟΓΙΟ

1. Unreal Engine 5 (UE5):
   Η Unreal Engine 5 είναι ένας σύγχρονος και ισχυρός κινητήρας ανάπτυξης παιχνιδιών που προσφέρει εργαλεία για τη δημιουργία 3D και 2D περιβαλλόντων υψηλής ποιότητας.

2. Σχεδιασμός Παιχνιδιών (Game Design):
   Η διαδικασία σχεδιασμού των κανόνων, των μηχανισμών και της αισθητικής ενός παιχνιδιού, με σκοπό τη δημιουργία μιας διασκεδαστικής και ισορροπημένης εμπειρίας για τον παίκτη.

3. Προγραμματιστής (Developer):
   Το άτομο που είναι υπεύθυνο για την ανάπτυξη και την υλοποίηση του κώδικα και των μηχανισμών ενός παιχνιδιού.

4. Σφάλμα (Bug):
   Μια ανεπιθύμητη ή απροσδόκητη συμπεριφορά σε ένα πρόγραμμα ή παιχνίδι που προκαλεί τη μη σωστή λειτουργία του.

5. Κινούμενη Εικόνα (Animation):
   Η διαδικασία δημιουργίας της αίσθησης της κίνησης μέσω μιας σειράς στατικών εικόνων ή καρέ.

6. Flipbook:
   Τεχνική δημιουργίας κινούμενων εικόνων στην Unreal Engine, χρησιμοποιώντας μια ακολουθία εικόνων για να δημιουργηθεί η ψευδαίσθηση της κίνησης.

7. Επεξεργαστής (Editor):
   Το εργαλείο εντός της Unreal Engine που χρησιμοποιείται για τη δημιουργία και την τροποποίηση του περιεχομένου του παιχνιδιού.

8. Platformer:
   Είδος παιχνιδιού όπου ο παίκτης πρέπει να κινηθεί σε μια πλατφόρμα, συχνά με άλματα, αποφεύγοντας εμπόδια και εχθρούς.

9. 2D:
   Αναφέρεται σε παιχνίδια που έχουν δισδιάστατη (2D) γραφική απεικόνιση, με την κίνηση να γίνεται σε δύο άξονες (οριζόντια και κάθετα).

10. Τεχνητή Νοημοσύνη (AI):
    Η προγραμματισμένη συμπεριφορά των μη-παικτικών χαρακτήρων (NPC) που επιτρέπει την αυτόνομη δράση τους στο παιχνίδι.

11. Χειριστήριο (Controller):
    Μια μονάδα ή αντικείμενο που επιτρέπει τον έλεγχο ενός χαρακτήρα ή μιας λειτουργίας στο παιχνίδι. Μπορεί να αναφέρεται και στον προγραμματιστικό κώδικα που ελέγχει τη συμπεριφορά των αντικειμένων.

12. Μη-Παικτικός Χαρακτήρας (NPC):
    Χαρακτήρες στο παιχνίδι που δεν ελέγχονται από τον παίκτη αλλά από το παιχνίδι (AI).

13. Ολοκληρωμένο Περιβάλλον Ανάπτυξης (IDE):
    Ένα λογισμικό που παρέχει εργαλεία προγραμματισμού, όπως επεξεργαστή κώδικα, μεταγλωττιστή, και debugger, όλα σε μια ενιαία διεπαφή.

14. Εικόνα (Sprite):
    Δισδιάστατη εικόνα ή γραφικό που χρησιμοποιείται για την αναπαράσταση χαρακτήρων, αντικειμένων και άλλων στοιχείων σε ένα 2D παιχνίδι.

15. Κλάση (Class):
    Στη C++, μια κλάση είναι ένα πρότυπο που καθορίζει τη δομή και τη συμπεριφορά των αντικειμένων που δημιουργούνται από αυτήν.

16. Διάνυσμα (Vector):
Στην ανάπτυξη παιχνιδιών, ένα διάνυσμα είναι ένα μαθηματικό αντικείμενο που αναπαριστά ένα μέγεθος και μια κατεύθυνση, συχνά χρησιμοποιούμενο για τον καθορισμό θέσης και κίνησης.

17. Κατάσταση Μηχανής (State Machine):
Ένα λογικό μοντέλο που καθορίζει τα διαφορετικά στάδια ή καταστάσεις μιας οντότητας στο παιχνίδι και τις μεταβάσεις μεταξύ αυτών των καταστάσεων

Ενότητα 1η

1. Εισαγωγή

1.1 Εισαγωγή στον κόσμο των δισδιάστατων(2d) παιχνιδιών πλατφόρμας(platformer)

Το platformer (ονομάζεται επίσης παιχνίδι πλατφόρμας και μερικές φορές παιχνίδι jump 'n' run) είναι ένα υπο-είδος βιντεοπαιχνιδιών δράσης στο οποίο ο βασικός στόχος είναι να μετακινηθεί ο χαρακτήρας του παίκτη μεταξύ διαφόρων σημείων σε ένα περιβάλλον. Τα παιχνίδια πλατφόρμας χαρακτηρίζονται από επίπεδα με ανώμαλο έδαφος και αναρτημένες πλατφόρμες διαφορετικού ύψους που απαιτούν άλματα και αναρρίχηση για να τα διασχίσουν. Άλλοι ακροβατικοί ελιγμοί μπορεί να επηρεάσουν το παιχνίδι, όπως η αιώρηση από τα κλήματα ή τα άγκιστρα, το άλμα από τοίχους, γλιστρώντας στον αέρα ή η αναπήδηση από εφαλτήρια ή τραμπολίνα. Το είδος ξεκίνησε με το arcade βιντεοπαιχνίδι του 1980 Space Panic, το οποίο έχει σκάλες αλλά όχι άλματα. Το Donkey Kong, που κυκλοφόρησε το 1981, δημιούργησε ένα πρότυπο για αυτά που αρχικά ονομάζονταν "παιχνίδια αναρρίχησης". Το Donkey Kong ενέπνευσε πολλούς κλώνους και παιχνίδια με παρόμοια στοιχεία, όπως το Miner 2049er (1982) και το Kangaroo (1982), ενώ το Arcade παιχνίδι Sega Congo Bongo (1983) προσθέτει μια τρίτη διάσταση μέσω ισομετρικών γραφικών. Ένα άλλο δημοφιλές παιχνίδι εκείνης της περιόδου, το Pitfall! (1982), επιτρέπει την κίνηση αριστερά και δεξιά μέσα από μια σειρά οθονών χωρίς κύλιση, επεκτείνοντας την περιοχή παιχνιδιού. Το κορυφαίο Super Mario Bros. (1985) της Nintendo ήταν ένα καθοριστικό παιχνίδι για το εκκολαπτόμενο είδος, με επίπεδα οριζόντιας κύλισης και τον παίκτη να ελέγχει έναν επώνυμο χαρακτήρα—τον Mario, ο οποίος έγινε μασκότ της εταιρείας. Ο όρος παιχνίδι πλατφόρμας απέκτησε έλξη στα τέλη της δεκαετίας του 1980, όπως και το εναλλακτικό παιχνίδι πλατφόρμας. Κατά τη διάρκεια της κορύφωσης της δημοτικότητάς τους, τα platformers εκτιμήθηκε ότι αποτελούν από το ένα τέταρτο έως το ένα τρίτο όλων των παιχνιδιών κονσόλας. Μέχρι το 2006, το είδος είχε υποστεί πτώση στις πωλήσεις, αντιπροσωπεύοντας μερίδιο αγοράς 2% σε σύγκριση με 15% το 1998. Παρόλα αυτά, τα platformers εξακολουθούν να κυκλοφορούν εμπορικά κάθε χρόνο, συμπεριλαμβανομένων ορισμένων που έχουν πουλήσει εκατομμύρια αντίτυπα.

Year	Event
1950s	The Birth of Video Games
1962	Spacewar! and the Birth of Modern Gaming
1970s-1980s	The Rise of Home Gaming and the Golden Arcade Age
1983-1985	Gaming Crisis and the Legend of E.T.
1985-1995	NES Revolution and Iconic Franchises
1990s	SNES and PlayStation Era
1996-2000	N64 and PlayStation 2
Early 2000s	Microsoft Enters and World of Warcraft
2006	The Wii and Motion-Based Gaming
Present -	The Era of Diversity and Cross-Platform Play

Εικόνα1: gamedesigning.com- Χρονοδιάγραμμα της εξέλιξης των ηλεκτρονικών παιχνιδιών

Γιατί είναι το 2D gaming τόσο σημαντικό ακόμα και στην σύγχρονη εποχή; Οι λόγοι είναι πολύ και ο κυρίως λόγος είναι η προσβασιμότητα που προσφέρουν. Οι χρήστες(gamers) των παιχνιδιών αυτών μαθαίνουν πιο εύκολα να χειρίζονται τα παιχνίδια αυτά, καθώς αφαιρείται η πολυπλοκότητα χειρισμού που έχουν τα τρισδιάστατα παιχνίδια. Έτσι τα πρώτα καθίστανται πιο φιλικά προς τον μέσο πελάτη, που μπορεί να μην έχει μεγάλη εμπειρία στον κόσμο των βιντεοπαιχνιδιών. Επίσης τα δισδιάστατα παιχνίδια είναι πιο εύκολο να οπτικοποιηθούν και να υλοποιηθούν και από τους προγραμματιστές(developers), δηλαδή από τους δημιουργούς τους. Το περιβάλλον και τα γραφικά στοιχεία των δισδιάστατων παιχνιδιών δεν απαιτούν τόσους προγραμματιστικούς πόρους και εργαλεία όσο ένα τρισδιάστατο(3D) παιχνίδι. Αυτό φυσικά δεν σημαίνει πως ένα 2d παιχνίδι είναι ευκολότερο ή πιο φθηνό να δημιουργηθεί σε σχέση με ένα 3d.

1.2 Εισαγωγή στην τεχνητή νοημοσύνη(AI) στα παιχνίδια

Η τεχνητή νοημοσύνη μπορεί να βελτιώσει τα παιχνίδια με πολλούς τρόπους απλά παίζοντας τα. Η βιομηχανία παιχνιδιών συνήθως λαμβάνει επαίνους για την τεχνητή νοημοσύνη των παιχνιδιών τους—ιδίως για τον μη παίκτη ή τον αντίπαλο τεχνητής νοημοσύνης (AI).Το AI του παιχνιδιού προσθέτει στην εμπορική αξία του παιχνιδιού, συμβάλλει σε καλύτερες κριτικές παιχνιδιών και βελτιώνει την εμπειρία του παίκτη. Είτε το AI βασίζεται σε ένα απλό δέντρο συμπεριφοράς(behavior tree), ένα AI βασισμένο σε βοηθητικά προγράμματα ή εναλλακτικά σε ένα εκλεπτυσμένο χειριστήριο που αντιδρά με βάση την μηχανική μάθηση(machine learned reactive controller), είναι μικρής σημασίας εφόσον εξυπηρετεί τους προαναφερθέντες σκοπούς. Μία αντισυμβατική και αποτελεσματική λύση σε μια εργασία που έχει δοθεί από τον από τον υπολογιστή (NPC-Non player character – μία οντότητα η οποία εμφανίζει κάποιες συμπεριφορές παίκτη, αναλόγως με τον προγραμματισμό της, αλλά στην πραγματικότητα η λειτουργία της ελέγχεται από ένα υπολογιστικό πρόγραμμά. Δηλαδή κώδικα που καταλαβαίνουν οι υπολογιστικές μηχανές.) προς τον παίκτη μπορεί συχνά να είναι ένας κρίσιμος παράγοντας που διαμορφώνει τις στρατηγικές διαχείρισης, μάρκετινγκ και δημιουργίας εσόδων κατά τη διάρκεια και μετά την παραγωγή ενός παιχνιδιού. Η τεχνητή νοημοσύνη παίζει παιχνίδια έχοντας κατά νου δύο βασικούς στόχους: παίξε καλά και/ή παίξε πιστευτά (ή σαν άνθρωπο, ή με ενδιαφέρον τρόπο). Περαιτέρω το AI(τεχνητή νοημοσύνη) μπορεί να ελέγξει είτε τον χαρακτήρα του παίκτη είτε έναν εικονικό χαρακτήρα του παιχνιδιού που δεν ελέγχεται από παίκτη. Ένα AI που παίζει καλά ως ένας χαρακτήρας που δεν είναι αληθινός παίκτης εστιάζει στη βελτιστοποίηση της απόδοσης του παιχνιδιού—η απόδοση μετριέται ως αποκλειστικά ο βαθμός στον οποίο ένας παίκτης εκπληρώνει τους στόχους του παιχνιδιού. Μια τέτοια τεχνητή νοημοσύνη μπορεί να είναι τεράστιας σημασίας για τον έλεγχο και την αξιολόγηση του σχεδιασμού ενός αυτόματου παιχνιδιού στο σύνολό του. Ένα AI που παίζει καλά ως χαρακτήρας που δεν ελέγχεται από έναν παίκτη, μπορεί να ενισχύσει τη δυναμική προσαρμογή δυσκολίας και τους αυτόματους μηχανισμούς εξισορρόπησης παιχνιδιών που με τη σειρά τους θα εξατομικεύσουν και θα βελτιώσουν την εμπειρία για τον παίκτη. Εάν η εστίαση της τεχνητής νοημοσύνης μετατοπιστεί στον έλεγχο χαρακτήρων που δεν ελέγχονται από αληθινούς παίκτες, που παίζουν πιστευτά σαν αληθινό άνθρωπο, τότε το AI μπορεί να χρησιμεύσει ως μέσο για τον εντοπισμό σφαλμάτων εμπειρίας παίκτη ή για την επίδειξη ενός ρεαλιστικού παιχνιδιού για σχεδιαστικούς σκοπούς. Τέλος, ένα παιχνίδι που διαθέτει πλούσια αλληλεπίδραση με τα NPC(μη αληθινούς παίκτες που έχουν δημιουργηθεί από πρόγραμμα τεχνικής νοημοσύνης) μπορεί να επωφεληθεί από την τεχνητή νοημοσύνη εμφανίζοντας εκφραστικές και πιστευτές συμπεριφορές που μοιάζουν με ανθρώπινες.

1.3 Χρησιμότητα της τεχνητής νοημοσύνη στα δισδιάστατα παιχνίδια

1.3.1 Επαύξηση εμπειρίας παιχνιδιού

Η τεχνητή νοημοσύνη σε παιχνίδια 2D μπορεί να βελτιώσει σημαντικά τις εμπειρίες παιχνιδιού δημιουργώντας πιο δυναμικά και ανταποκρινόμενα περιβάλλοντα παιχνιδιού. Ακολουθούν μερικοί τρόποι με τους οποίους το AI επιτυγχάνει αυτό:

I. Δυναμική συμπεριφορά εχθρού:

- Προσαρμοστική τεχνητή νοημοσύνη (Adaptive AI): Το AI μπορεί να σχεδιαστεί για να μαθαίνει και να προσαρμόζεται στις στρατηγικές των παικτών σε πραγματικό χρόνο. Αυτό δημιουργεί εχθρούς που μπορούν να ανταποκριθούν έξυπνα στις ενέργειες του παίκτη, παρέχοντας μια προκλητική και συναρπαστική εμπειρία. Για παράδειγμα, σε ένα 2D platformer, οι εχθροί μπορεί να μάθουν να αποφεύγουν τα συχνά χρησιμοποιούμενα μοτίβα επίθεσης ή να βάζουν παγίδες με βάση την κίνηση του παίκτη.

-Αλγόριθμος εύρεσης μονοπατιών (Pathfinding): Η τεχνητή νοημοσύνη μπορεί να χρησιμοποιήσει αλγόριθμους εύρεσης μονοπατιών για την πλοήγηση σε πολύπλοκα 2D περιβάλλοντα. Αυτό κάνει τις κινήσεις του εχθρού πιο απρόβλεπτες και προκλητικές, καθώς μπορούν να βρουν τα βέλτιστα μονοπάτια για να κυνηγήσουν ή να αποφύγουν τον παίκτη.

II. Περιβαλλοντικές αλληλεπιδράσεις:

-Συστήματα διαδικαστικής τεχνητής νοημοσύνης: Η τεχνητή νοημοσύνη μπορεί να ελέγχει περιβαλλοντικά στοιχεία που αντιδρούν στις ενέργειες των παικτών. Για παράδειγμα, οι πλατφόρμες ή τα εμπόδια μπορεί να αλλάξουν θέση ανάλογα με το πού βρίσκεται ο παίκτης, κάνοντας τα επίπεδα να αισθάνονται πιο ζωντανά και να ανταποκρίνονται.

-Προσαρμογή σε πραγματικό χρόνο: Το AI μπορεί να προσαρμόσει τη δυσκολία του παιχνιδιού σε πραγματικό χρόνο παρακολουθώντας την απόδοση του παίκτη. Εάν ένας παίκτης δυσκολεύεται, η τεχνητή νοημοσύνη μπορεί να μειώσει τον αριθμό των εχθρών ή να κάνει πιο εύκολη την πρόσβαση στις πλατφόρμες, διασφαλίζοντας μια ισορροπημένη εμπειρία παιχνιδιού.

1.3.2 Διαδικαστική Δημιουργία Περιεχομένου

Το AI μπορεί να χρησιμοποιηθεί για τη δημιουργία διαδικαστικού περιεχομένου για τη δημιουργία ποικίλων και ενδιαφέροντων επιπέδων και σεναρίων παιχνιδιών. Παρακάτω αναφέρεται ο τρόπος λειτουργίας του:

I. Δημιουργία επιπέδου:

-Τυχαίες διατάξεις: Η τεχνητή νοημοσύνη μπορεί να δημιουργήσει επίπεδα χρησιμοποιώντας αλγόριθμους που τοποθετούν τυχαία στοιχεία εντός προκαθορισμένων χώρων. Αυτό διασφαλίζει ότι κάθε πίστα προσφέρει μια μοναδική εμπειρία.

-Θεματική συνέπεια: Η τεχνητή νοημοσύνη μπορεί να διασφαλίσει ότι τα επίπεδα που δημιουργούνται διαδικαστικά διατηρούν μια συνεπή καμπύλη θεμάτων και δυσκολίας, αποφεύγοντας τη δημιουργία αδύνατων ή παράλογων διατάξεων. Αυτή η ισορροπία διατηρεί το παιχνίδι ελκυστικό και δίκαιο.

II. Ποικιλία περιεχομένου:

-Τοποθέτηση αντικειμένου και εχθρού: Το AI μπορεί να τοποθετεί τυχαία αντικείμενα και εχθρούς σε διαφορετικές τοποθεσίες κάθε φορά που φορτώνεται ένα επίπεδο. Αυτό προσθέτει ένα στοιχείο έκπληξης και απαιτεί από τους παίκτες να προσαρμόζουν συνεχώς τις στρατηγικές τους.

-Αφηγηματικά στοιχεία: Η τεχνητή νοημοσύνη μπορεί να δημιουργήσει διαδικαστικά στοιχεία ιστορίας ή αποστολές, παρέχοντας στους παίκτες μοναδικές αποστολές και προκλήσεις κάθε φορά που παίζουν. Αυτό μπορεί να βελτιώσει σημαντικά τη δυνατότητα αναπαραγωγής και τη βύθιση στον κόσμο του παιχνιδιού.

III. Εξισορρόπηση και δοκιμή:

-Αυτοματοποιημένη δοκιμή αναπαραγωγής: Η τεχνητή νοημοσύνη μπορεί να προσομοιώσει χιλιάδες δοκιμές παιχνιδιού για να εντοπίσει πιθανά ζητήματα σε περιεχόμενο που δημιουργείται διαδικαστικά, όπως αθέμιτες αιχμές δυσκολίας ή μη προσβάσιμες περιοχές. Αυτό βοηθά τους προγραμματιστές να τελειοποιήσουν το παιχνίδι πριν από την κυκλοφορία.

-Σχόλια σε πραγματικό χρόνο: Κατά τη διάρκεια του παιχνιδιού, η τεχνητή νοημοσύνη μπορεί να συλλέξει δεδομένα σχετικά με τις αλληλεπιδράσεις των παικτών με περιεχόμενο που δημιουργείται διαδικαστικά και να κάνει προσαρμογές επί τόπου. Αυτό διασφαλίζει ότι το παιχνίδι παραμένει ισορροπημένο και απολαυστικό σε διαφορετικά στυλ παιχνιδιού.

1.4 Παραδείγματα γνωστών δισδιάστατων παιχνιδιών με εφαρμογή τεχνητής νοημοσύνης

Αξιοσημείωτα παραδείγματα AI σε παιχνίδια 2D

1. "Hollow Knight" (https://www.hollowknight.com/)

- Εφαρμογή AI: Η τεχνητή νοημοσύνη στο "Hollow Knight" είναι αξιοσημείωτο για την έξυπνη εχθρική συμπεριφορά του. Οι εχθροί στο παιχνίδι έχουν μοναδικά μοτίβα και προσαρμόζονται στις κινήσεις του παίκτη, παρέχοντας ποικίλα και προκλητικά σενάρια μάχης. Το Boss AI, συγκεκριμένα, έχει σχεδιαστεί για να δημιουργεί μια ελκυστική και δυναμική πρόκληση προσαρμόζοντας τα μοτίβα επίθεσης με βάση τις ενέργειες του παίκτη.
- Συμβολή στην επιτυχία: Η προηγμένη τεχνητή νοημοσύνη συμβάλλει σημαντικά στην αναγνώριση και τη δημοτικότητα του παιχνιδιού από τους κριτικούς. Η προκλητική αλλά δίκαιη συμπεριφορά του εχθρού ενισχύει τη συνολική εμπειρία του παίκτη, ενθαρρύνοντας την ανάπτυξη δεξιοτήτων και τη στρατηγική σκέψη.

Εικόνα 2: Πηγή pockettactics.com - 2d παιχνίδι hollow knight

2. "Dead Cells" (https://dead-cells.com/)

- Εφαρμογή AI: Το "Dead Cells" χρησιμοποιεί διαδικαστική δημιουργία σε συνδυασμό με προσαρμοστική τεχνητή νοημοσύνη για τη δημιουργία μοναδικών διατάξεων επιπέδου και τοποθετήσεων εχθρών κάθε φορά που ένας παίκτης ξεκινά μια νέα διαδρομή. Το AI ελέγχει τις συμπεριφορές του εχθρού για να διασφαλίσει ότι παραμένουν απρόβλεπτες και προκλητικές, προσαρμόζοντας τις στρατηγικές τους με βάση τις ενέργειες και την πρόοδο του παίκτη.

- Συμβολή στην επιτυχία: Αυτή η εφαρμογή της τεχνητής νοημοσύνης ήταν καθοριστικής σημασίας για να γίνει το «Dead Cells» ένα ξεχωριστό παιχνίδι απατεώνων. Τα συνεχώς μεταβαλλόμενα περιβάλλοντα και οι έξυπνοι εχθροί διατηρούν το παιχνίδι φρέσκο και ελκυστικό, οδηγώντας σε υψηλή δυνατότητα αναπαραγωγής και διαρκές ενδιαφέρον των παικτών.

3. "Terraria (https://terraria.org/)

- Εφαρμογή AI: Το "Terraria" χρησιμοποιεί τεχνητή νοημοσύνη για τον έλεγχο μιας ποικιλίας συμπεριφορών NPC, από εχθρούς που καταδιώκουν τον παίκτη με μοναδικούς τρόπους έως NPC που αλληλεπιδρούν με το περιβάλλον και τον παίκτη με ουσιαστικούς τρόπους. Η δημιουργία διαδικαστικού κόσμου, καθοδηγούμενη από αλγόριθμους τεχνητής νοημοσύνης, διασφαλίζει ότι κάθε κόσμος παιχνιδιού είναι μοναδικός, με διαφορετικό έδαφος, βίωμα και πόρους.

- Συμβολή στην επιτυχία: Η τεχνητή νοημοσύνη και οι συμπεριφορές NPC έχουν κάνει το "Terraria" ένα αγαπημένο παιχνίδι sandbox(τύπος παιχνιδιού χωρίς αυστηρά προσδιορισμένους κανόνες και στόχους). Η ατελείωτη ποικιλία και οι δυναμικές αλληλεπιδράσεις παρέχουν στους παίκτες μια αίσθηση ανακάλυψης και περιπέτειας, συμβάλλοντας στη μακροπρόθεσμη επιτυχία και δημοτικότητά του.

Ανάλυση θετικών συνεπειών της χρήσης της τεχνητής νοημοσύνης στα παραπάνω παιχνίδια

1.Ενίσχυση της αφοσίωσης των παικτών

-Δυναμικές προκλήσεις: Η τεχνητή νοημοσύνη σε παιχνίδια όπως το "Hollow Knight" (https://www.hollowknight.com/) και το "Dead Cells" (https://dead-cells.com/) διασφαλίζει ότι οι παίκτες αντιμετωπίζουν δυναμικές και προσαρμοστικές προκλήσεις, αποτρέποντας το μονότονο του παιχνιδιού. Αυτή η προσαρμοστικότητα διατηρεί το ενδιαφέρον των παικτών και παρέχει μια ικανοποιητική εμπειρία καθώς οι παίκτες μαθαίνουν και ξεπερνούν όλο και πιο περίπλοκα εμπόδια.

2.Αυξημένη δυνατότητα αναπαραγωγής του παιχνιδιού

-Διαδικαστική δημιουργία: Η δημιουργία διαδικαστικού περιεχομένου με γνώμονα την τεχνητή νοημοσύνη, όπως φαίνεται στα "Dead Cells" και "Terraria" (https://terraria.org/), ενισχύει σημαντικά τη δυνατότητα επανάληψης, προσφέροντας μοναδικές εμπειρίες σε κάθε playthrough(παιχνίδι). Οι παίκτες παρακινούνται να επαναλάβουν το παιχνίδι για να εξερευνήσουν νέα περιβάλλοντα και συναντήσεις, ενισχύοντας τη μακροζωία του παιχνιδιού.

3.Ισορροπημένη Δυσκολία

-Προσαρμοστική δυσκολία: Η τεχνητή νοημοσύνη μπορεί να εξισορροπήσει τη δυσκολία του παιχνιδιού σε πραγματικό χρόνο με βάση την απόδοση του παίκτη. Αυτό διασφαλίζει ότι το παιχνίδι παραμένει προσβάσιμο σε νέους παίκτες, ενώ εξακολουθεί να αποτελεί πρόκληση για έμπειρους παίκτες. Η προσαρμοστική δυσκολία βοηθά στη διατήρηση μιας ευρύτερης βάσης παικτών καλύπτοντας διαφορετικά επίπεδα δεξιοτήτων.

4.Κοινότητα και Δημιουργία Περιεχομένου

-Περιεχόμενο που δημιουργείται από τον χρήστη: Παιχνίδια όπως το "Terraria" επωφελούνται από τη δημιουργία διαδικασιών βάσει τεχνητής νοημοσύνης που υποστηρίζει εκτεταμένες τροποποιήσεις και περιεχόμενο που δημιουργείται από τον χρήστη. Αυτό ενθαρρύνει μια ισχυρή κοινότητα γύρω από το παιχνίδι,

με τους παίκτες να συνεισφέρουν νέο περιεχόμενο και εμπειρίες, επεκτείνοντας περαιτέρω τη διάρκεια ζωής και την ελκυστικότητα του παιχνιδιού.

Συμπέρασμα

Η τεχνητή νοημοσύνη έχει διαδραματίσει καθοριστικό ρόλο στη βελτίωση των εμπειριών παιχνιδιού αξιοσημείωτων παιχνιδιών 2D, δημιουργώντας δυναμικά, ανταποκρινόμενα και ποικίλα περιβάλλοντα παιχνιδιού. Η επιτυχής εφαρμογή της τεχνητής νοημοσύνης σε αυτά τα παιχνίδια όχι μόνο συνέβαλε στην κριτική και εμπορική επιτυχία τους, αλλά επίσης βελτίωσε σημαντικά την αφοσίωση και την ικανοποίηση των παικτών.

1.5 Στόχος πτυχιακής εργασίας και δομή

1.5.1 Επισκόπηση της Πτυχιακής εργασίας

Στο εξελισσόμενο πεδίο της ανάπτυξης βιντεοπαιχνιδιών, η ενσωμάτωση της τεχνητής νοημοσύνης (AI) στη μηχανική του παιχνιδιού έχει γίνει ένας κρίσιμος τομέας καινοτομίας. Αυτή η πτυχιακή διερευνά τον σχεδιασμό και την υλοποίηση ενός παιχνιδιού πλατφόρμας 2D που αξιοποιεί την τεχνητή νοημοσύνη για να βελτιώσει την εμπειρία των παικτών μέσω δυναμικών εχθρικών συμπεριφορών και προοδευτικών επιπέδων δυσκολίας. Ο πρωταρχικός στόχος είναι η επίδειξη πώς η τεχνητή νοημοσύνη μπορεί να χρησιμοποιηθεί για τη δημιουργία πιο ελκυστικού, προκλητικού και προσαρμοστικού παιχνιδιού σε περιβάλλοντα 2D.

1.5.2 Πλαίσιο έρευνας

Η αναβίωση των παιχνιδιών 2D platformer στη σκηνή του indie gaming υπογραμμίζει τη διαρκή ελκυστικότητά τους. Παιχνίδια όπως το "Hollow Knight" και το "Celeste" (https://www.celestegame.com/) έχουν αποδείξει ότι τα παιχνίδια 2D μπορούν να προσφέρουν βαθιές, εμπλουτισμένες εμπειρίες που συναγωνίζονται την πολυπλοκότητα των σύγχρονων 3D τίτλων. Ωστόσο, μία από τις σημαντικές προκλήσεις στον σχεδιασμό δισδιάστατων παιχνιδιών είναι η διατήρηση της αφοσίωσης των παικτών σε εκτεταμένες περιόδους παιχνιδιού. Οι στατικές εχθρικές συμπεριφορές και η προβλέψιμη εξέλιξη της δυσκολίας μπορούν γρήγορα να οδηγήσουν σε κόπωση του παίκτη και μειωμένο ενδιαφέρον. Με την ενσωμάτωση της τεχνητής νοημοσύνης, μπορούμε να αντιμετωπίσουμε αυτές τις προκλήσεις, δημιουργώντας ένα πιο δυναμικό και ανταποκρινόμενο περιβάλλον παιχνιδιού.

1.5.3 Στόχοι της Πτυχιακής

Η παρούσα διπλωματική εργασία στοχεύει στην επίτευξη πολλών βασικών στόχων. Αρχικά στοχεύει στην ανάπτυξη έξυπνων εχθρικών NPC (χαρακτήρες χωρίς παίκτες) που προσαρμόζονται στις ενέργειες των παικτών, παρέχοντας μια προκλητική και απρόβλεπτη εμπειρία. Αυτό περιλαμβάνει τον σχεδιασμό αλγορίθμων AI που επιτρέπουν στους εχθρούς να μάθουν από τη συμπεριφορά των παικτών, να προσαρμόσουν τις στρατηγικές τους και να αντιδράσουν δυναμικά μέσα στο περιβάλλον του παιχνιδιού. Στην συνέχεια στόχος είναι η εφαρμογή προοδευτικής δυσκολίας, δηλαδή η δημιουργία ενός συστήματος προοδευτικής δυσκολίας που προσαρμόζεται σε πραγματικό χρόνο με βάση την απόδοση του παίκτη. Αυτό το σύστημα διασφαλίζει ότι το παιχνίδι παραμένει ανταγωνιστικό αλλά δίκαιο, αυξάνοντας σταδιακά τη δυσκολία του καθώς ο παίκτης βελτιώνεται, διατηρώντας έτσι μια ισορροπημένη και συναρπαστική εμπειρία. Ένας ακόμα στόχος είναι η δημιουργία διαδραστικού και δυναμικού περιβάλλοντος στο χώρο του παιχνιδιού. Η χρησιμοποίηση έξυπνων τεχνικών για τη δημιουργία παγίδων και αντικειμένων αλληλεπίδρασης μεταξύ περιβάλλοντος και παίκτη για την δημιουργία ποικίλων και

ενδιαφερόντων επιπέδων παιχνιδιού. Αυτό περιλαμβάνει δημιουργία διατάξεων επιπέδου, τοποθετήσεων εχθρών και διανομών αντικειμένων, διασφαλίζοντας ότι κάθε επίπεδο προσφέρει μια μοναδική εμπειρία. Στη συνέχεια γίνεται η ανάλυση της εμπειρίας του παίχτη, δηλαδή η πραγματοποίηση ανάλυσης του τρόπου με τον οποίο τα χαρακτηριστικά που βασίζονται στο AI επηρεάζουν την αφοσίωση, την ικανοποίηση και τη διατήρηση των παικτών. Αυτό περιλαμβάνει τη συλλογή και την αξιολόγηση των σχολίων των παικτών και των δεδομένων παιχνιδιού για την αξιολόγηση της αποτελεσματικότητας των υλοποιήσεων AI. Για την επίτευξη αυτών των στόχων, η πτυχιακή υιοθετεί μια πολύπλευρη προσέγγιση, που συνδυάζει τη θεωρητική έρευνα με την πρακτική εφαρμογή. Η διαδικασία ανάπτυξης περιλαμβάνει τα ακόλουθα βήματα. Πραγματοποιήθηκε μια εκτενής ανασκόπηση των υπαρχόντων ερευνών και μελετών σχετικά με την τεχνητή νοημοσύνη στα παιχνίδια, εστιάζοντας στην τεχνητή νοημοσύνη του εχθρού, στο επίπεδο δυσκολίας του εχθρού, το επίπεδο δυσκολίας του περιβάλλοντος , καθώς και στην διαδραστικότητά του. Αυτό παρέχει μια σταθερή βάση για τη διαδικασία ανάπτυξης και εντοπίζει βέλτιστες πρακτικές και κοινές παγίδες. Στη συνέχεια δημιουργήθηκε το παιχνίδι 2D action platformer χρησιμοποιώντας μια σύγχρονη μηχανή ανάπτυξης παιχνιδιών, την unreal engine 5, που ενσωματώνει χαρακτηριστικά AI για συμπεριφορά εχθρού και εξέλιξη δυσκολίας. Η διαδικασία ανάπτυξης περιλαμβάνει επαναληπτικές δοκιμές και βελτίωση για να διασφαλιστεί ότι η τεχνητή νοημοσύνη συμπεριφέρεται όπως προβλέπεται και συμβάλλει θετικά στην εμπειρία παιχνιδιού. Εφαρμόστηκε μια δοκιμαστική φάση όπου οι παίκτες κλήθηκαν να παίξουν το παιχνίδι και να δώσουν σχόλια. Αυτή η ανατροφοδότηση είναι ζωτικής σημασίας για τον εντοπισμό περιοχών για βελτίωση και την επικύρωση της αποτελεσματικότητας των χαρακτηριστικών τεχνητής νοημοσύνης. Εν κατακλείδι δημιουργώντας πιο δυναμικές και ανταποκρινόμενες εχθρικές συμπεριφορές, το παιχνίδι στοχεύει να κρατήσει τους παίκτες αφοσιωμένους για μεγαλύτερα χρονικά διαστήματα, μειώνοντας την πιθανότητα κόπωσης του παίκτη και αυξάνοντας τη συνολική απόλαυση. Η δημιουργία δυναμικού περιεχομένου διασφαλίζει ότι κάθε καινούργιο παιχνίδι είναι διαδραστικό, ενθαρρύνοντας τους παίκτες να επιστρέψουν στο παιχνίδι πολλές φορές για να βιώσουν νέες προκλήσεις και περιβάλλοντα. Το σύστημα προοδευτικής δυσκολίας παρέχει συμβάλλει για την δημιουργία ισορροπημένων και δίκαννων εμπειριών παιχνιδιού που προσαρμόζονται στα επίπεδα δεξιοτήτων των παικτών. Τα ευρήματα και οι μεθοδολογίες που παρουσιάζονται σε αυτή τη διπλωματική μπορούν να χρησιμεύσουν ως αναφορά για άλλους προγραμματιστές παιχνιδιών που θέλουν να ενσωματώσουν την τεχνητή νοημοσύνη στα έργα τους, συμβάλλοντας στην ευρύτερη γνωσιακή βάση των εφαρμογών AI στο gaming.

Ενότητα 2η

2. Επισκόπηση του πεδίου, προσδιορισμός του προβλήματος και διεθνής υλικό

2.1 Σύντομη εισαγωγή στη σχεδίαση 2d παιχνιδιών πλατφόρμας και C++

Η σχεδίαση ενός 2d παιχνιδιού είναι εκ των πραγμάτων η γραφική σχεδίαση ενός παιχνιδιού εντός μόνο δύο διαστάσεων. Με αυτό συνεπάγεται πως και η κίνηση του πραγματικού ανθρώπου-παίκτη και τον διαφόρων άλλων npc - παικτών που η λειτουργία της ελέγχεται από ένα state machine ή έναν αλγόριθμο με δένδρο αποφάσεων ή ακόμα και από ένα machine learning αλγόριθμο που θα αναφερθούν και εξηγηθούν παρακάτω. Παρόλο που σε κάποιες περιπτώσεις μπορεί ο χρήστης(gamer) να εξαπατηθεί από τον σχεδιασμό των γραφικών, τα οποία πολλές φορές προσδίδουν ένα βάθος στο γραφικό περιβάλλον του παιχνιδιού, στην πραγματικότητα τα παιχνίδια αυτά κινούνται μόνο σε δύο διαστάσεις, τον άξονα x και τον άξονα y. Δηλαδή εκτείνονται σε μήκος και σε ύψος, αλλά ποτέ σε βάθος. Αν και τα σύγχρονα χρόνια έχει αρχίσει να γίνεται όλο και πιο δημοφιλής η δημιουργία hybrid 2d και 3d παιχνιδιών, δηλαδή 2d παιχνίδια με 3d γραφικά στοιχεία. Πιο συγκεκριμένα τα παιχνίδια 2d πλατφόρμας ή αλλιώς platformer games, όπως θα αναφέρονται από εδώ και στο εξής στην εργασία. Τα παιχνίδια αυτά συνδυάζουν στοιχεία δράσης με μηχανισμούς platformer game. Σε αυτά τα παιχνίδια η οπτική γωνία του χειριστή είναι από το πλάι και ο ίδιος καλείται να ελέγχει τον χαρακτήρα του μέσα στο παιχνίδι και να εξερευνά διάφορα επίπεδα γεμάτα με εμπόδια, εχθρούς και διάφορες παγίδες. Ο κύριος σκοπός είναι ο παίκτης να νικήσει ένα επίπεδο, τον σπουδαιότερο αντίπαλο(boss) ή να βρει κάποιο αντικείμενο.

2.2 Γλώσσα προγραμματισμού C++

Η δημιουργία αυτού του παιχνιδιού που είναι και το θέμα της διπλωματικής εργασίας επιτεύχθηκε με την χρήση της γλώσσας προγραμματισμού υψηλού επιπέδου C++, χρησιμοποιώντας τον αντικειμενοστραφή μέθοδο προγραμματισμού. Γιατί επιλέχθηκε η C++ για την δουλειά αυτή και για ποιο λόγο η C++ θεωρείτε πολύ καλή επιλογή για την σχεδίαση βιντεοπαιχνιδιών θα αναλυθεί παρακάτω:

-Απόδοση: Η C++ επιτρέπει τον ακριβή έλεγχο πάνω στους πόρους ενός υπολογιστικού συστήματος που είναι πολύ σημαντικό για την απόδοση του συστήματος σε παιχνίδια με υψηλές απαιτήσεις λογισμικού.

Το C++ AMP (C++ Accelerated Massive Parallelism) επιταχύνει την εκτέλεση κώδικα C++ εκμεταλλευόμενο το υλικό παράλληλης επεξεργασίας δεδομένων, το οποίο συνήθως είναι παρόν ως μονάδα επεξεργασίας γραφικών (GPU) σε μια ξεχωριστή κάρτα γραφικών. Το C++ Accelerated Massive Parallelism (C++ AMP) είναι ένα εγγενές μοντέλο προγραμματισμού που περιέχει στοιχεία που εκτείνονται στη γλώσσα προγραμματισμού C++ και στη βιβλιοθήκη χρόνου εκτέλεσης της. Το C++ AMP είναι μια βιβλιοθήκη που υλοποιείται στο DirectX 11 και είναι μια ανοικτή προδιαγραφή από τη Microsoft για την υλοποίηση παράλληλης επεξεργασίας δεδομένων απευθείας σε C++. Αυτή η γλώσσα είναι πιο εύκολη στη χρήση και περιέχει πολλές βιβλιοθήκες για την ανάπτυξη εφαρμογών παράλληλης επεξεργασίας δεδομένων. Για να φανεί πιο ξεκάθαρα πώς να χρησιμοποιήσετε το C++ AMP για την επίλυση ενός προβλήματος παράλληλης επεξεργασίας δεδομένων, θα παρουσιαστεί το παρακάτω παράδειγμα. Το πρόβλημα είναι απλό και έχει παράλληλη φύση, η πολλαπλασιασμός πινάκων. Ας πάρουμε μια μαθηματική ή οικονομική εφαρμογή όπου μέρος της διαδικασίας είναι ο πολλαπλασιασμός πινάκων, αλλά τα πιθανά σενάρια είναι ο πολλαπλασιασμός πινάκων μικρού μεγέθους και η πράξη πολλαπλασιασμού δεν θα εκτελεστεί πολλές φορές, τότε ο σειριακός κώδικας για αυτήν την πράξη δεν θα χρειαζόταν μεγάλο χρόνο εκτέλεσης και ο παράλληλος κώδικας θα ήταν υπερβολικός. Αλλά, φανταστείτε ένα σενάριο με 200 πίνακες με 40000 στοιχεία ο καθένας. Από εδώ έχουμε 100 πράξεις πολλαπλασιασμού και ο σειριακός κώδικας θα έπαιρνε μεγάλο χρόνο εκτέλεσης, ενώ αν οι 100 πράξεις εκτελούνταν παράλληλα, αξιοποιώντας τις δυνατότητες που παρέχει η GPU, ο χρόνος εκτέλεσης θα ήταν μικρότερος[1]. Παρακάτω παρατίθεται μια απλή μη-παράλληλη συνάρτηση σε C++ για τον πολλαπλασιασμό δύο πινάκων:

1Παραπομπή: [Mil1] Gregory, Kate, and Ade Miller. C++ AMP: Accelerated Massive Parallelism with Microsoft® Visual C++®. " O'Reilly Media, Inc.", 2012

```
multiplication(vector<vector<int>>& T1,
vector<vector<int>>& T2, vector<vector<int>>&
T3, const int n, const int m, const int k)
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<k; j++)
        {
            int sum = 0;
            for(int z=0; z<m; z++)

                sum += T1[i][z]*T2[z][j];
            T3[i][j] = sum;
        }
    }
}
```

Συμπερασματικά εάν η κύρια προϋπόθεση του προτεινόμενου αλγορίθμου για ένα πρόβλημα είναι η ταχύτητα εκτέλεσης, τότε ο παράλληλος προγραμματισμός γίνεται σημαντικός. Ο προγραμματισμός GPGPU είναι μια νέα και απαιτητική τεχνική που χρησιμοποιείται για την επίλυση προβλημάτων με φύση παράλληλης επεξεργασίας δεδομένων. Ορισμένοι τομείς όπου μπορούμε να βρούμε αυτού του είδους τα προβλήματα είναι: συστήματα ελέγχου σε πραγματικό χρόνο, επιστημονική μοντελοποίηση και προσομοίωση, gaming, χρηματοοικονομική προσομοίωση, επεξεργασία εικόνας, κ.λπ. Για προβλήματα που ταιριάζουν καλά με την αρχιτεκτονική της GPU, είναι σύνηθες να επιτυγχάνεται εύκολα μια επιτάχυνση 2× ή και περισσότερη σε σύγκριση με την υλοποίηση του ίδιου προβλήματος σε CPU, ενώ οι βελτιστοποιημένες υλοποιήσεις μπορούν να ξεπεράσουν την CPU κατά 10 έως 100 φορές. Η GPU είναι μια προγραμματιζόμενη και ισχυρή υπολογιστική συσκευή από μόνη της. Η GPU έχει μια αρχιτεκτονική μαζικής παράλληλης επεξεργασίας που αποτελείται από χιλιάδες μικρότερους, πιο αποδοτικούς πυρήνες σχεδιασμένους για την εκτέλεση πολλαπλών εργασιών ταυτόχρονα. Το εύρος ζώνης μνήμης μιας CPU είναι περίπου 20 GB/s, σε σύγκριση με τα 150 GB/s της GPU. Ένα ισχυρό σημείο των GPUs είναι η κατανάλωση ενέργειας. Μια GPU μπορεί να αποδώσει 10 GFLOPS/watt, ενώ μια CPU μπορεί να αποδώσει 1 GIGAFLOPS/watt. Εάν θέλουμε να αναπτύξουμε κώδικα που θα εκτελεστεί παράλληλα, η GPGPU είναι ένας καλός υποψήφιος ως νέα τεχνολογία και το C++ AMP είναι μια γλώσσα που παρέχει ευκολίες κατά τον προγραμματισμό και τις απαραίτητες βιβλιοθήκες[2].

-Διαχείριση μνήμης:

Η διαχείριση μνήμης που προσφέρει η C++ είναι ένα μεγάλο πλεονέκτημα για την σχεδίαση παιχνιδιών, καθώς έχει άμεση επιρροή πάνω στην απόδοση, τη σταθερότητα και την αποτελεσματικότητα των προγραμμάτων. Η C++ παρέχει διάφορα εργαλεία και τεχνικές για την διαχείριση της μνήμης διευκολύνοντας έτσι τους σχεδιαστές παιχνιδιών(Game developers-devs). Πιο συγκεκριμένα η C++ προσφέρει στατική και δυναμική διαχείριση μνήμης. Η Global και static(παγκόσμιες και στατικές) μεταβλητές αποθηκεύονται καθ' όλης της διάρκειας ζωής του προγράμματος, ενώ μεταβλητές που δηλώνονται μέσα σε κάποιο πεδίο(scope) αποθηκεύονται στην στοίβα και διαγράφονται αυτόματα όταν η ροή του προγράμματος φτάνει στο τέλος αυτού του πεδίου. Επιπρόσθετα η δυναμική διαχείριση μνήμης επιτρέπει στον προγραμματιστή να δεσμεύσει δυναμικά μέρος της μνήμης από τη μνήμη σωρού(heap) ή να την αποδεσμεύσει, όπως εκείνος επιθυμεί. Επιπλέον υπάρχουν οι έξυπνοι δείκτες(smart pointers) που είναι κλάσεις προτύπων στην Τυπική βιβλιοθήκη που διαχειρίζονται τον κύκλο ζωής των δυναμικά εκχωρημένων αντικειμένων. Στην ανάπτυξη παιχνιδιών, η αποτελεσματική διαχείριση μνήμης είναι ζωτικής σημασίας λόγω των υψηλών απαιτήσεων απόδοσης των παιχνιδιών. Η C++ χρησιμοποιείται ευρέως στη βιομηχανία επειδή παρέχει στους προγραμματιστές ακριβή έλεγχο των πόρων του συστήματος, επιτρέποντας τη βελτιστοποίηση της χρήσης μνήμης και της ταχύτητας επεξεργασίας. Αποτελεσματικές τεχνικές διαχείρισης μνήμης στην C++, όπως η χειροκίνητη κατανομή και αποδέσμευση μνήμης, μαζί με τη χρήση έξυπνων δεικτών

2Παραπομπή:[Gas2] B. R. Gaster and L. Howes, "Can GPGPU Programming Be Liberated from the Data-Parallel Bottleneck," Computer, vol. 45, no. 8, pp. 42–52, Aug. 2012

(smart pointers), βοηθούν στην αποτροπή διαρροών μνήμης και κατακερματισμού, που είναι κρίσιμες για τη διατήρηση της απόδοσης σε σύνθετα περιβάλλοντα παιχνιδιών[3].

-Αντικειμενοστραφής προγραμματισμός (oop):

Η C++ είναι μία από τις γλώσσες υψηλού επιπέδου που υποστηρίζει τις τεχνικές του αντικειμενοστραφή προγραμματισμού, με τον οποίο επιτυγχάνεται η ανάπτυξη λογισμικού γύρω από δεδομένα ή αντικείμενα έναντι συναρτήσεων και απλής λογικής. Ο αντικειμενοστραφής προγραμματισμός περιέχει κάποιες βασικές έννοιες οι οποίες θα αναλυθούν παρακάτω για να γίνει κατανοητός. Πρώτα από όλα πρέπει να αναφερθεί η έννοια της κλάσης (class), καθώς είναι μία πολύ βασική λειτουργία του oop. Μία κλάση είναι ένα προσχέδιο για την δημιουργία αντικειμένων. Καθορίζει έναν τύπο δεδομένων συνδυάζοντας δεδομένα – μεταβλητές και συναρτήσεις ή αλλιώς μεθόδους που λειτουργούν με αυτά τα δεδομένα σε μία ενιαία μονάδα. Ένα αντικείμενο είναι ένα παράδειγμα μίας κλάσης. Είναι μία οντότητα η οποία έχει χαρακτηριστικά και συμπεριφορές που ορίζονται από την κλάση του. Πολλά αντικείμενα μπορούν να προκύψουν από μία κλάση, άλλα όλα αυτά τα αντικείμενα εφόσον προέρχονται από την ίδια κλάση θα είναι του ίδιου τύπου αντικείμενα ή αλλιώς τις ίδιας ομάδας. Έπειτα υπάρχει η έννοια της ενθυλάκωσης (encapsulation). Η ενθυλάκωση είναι η ομαδοποίηση δεδομένων και μεθόδων που υπάρχουν σε μία κλάση με σκοπό τον περιορισμό της πρόσβασης αυτών από άλλα μέλη του προγράμματος ή χρήστες που δεν έχουν δικαιοδοσία. Αυτό σημαίνει ότι και τα αντικείμενα που προκύπτουν από μία κλάση θα έχουν περιορισμούς πρόσβασης σε κάποια από τα δεδομένα τους. Η ενθυλάκωση επιτυγχάνεται με χρήση λέξεων κλειδιών που περιορίζουν την πρόσβαση, όπως οι λέξεις: Ιδιωτικός (private), προστατευόμενος (protected) και δημόσιος (public). Εν συνεχεία υπάρχει η έννοια της κληρονομικότητας, η οποία επιτρέπει σε μία νέα κλάση να κληρονομεί χαρακτηριστικά και μεθόδους από μια υπάρχουσα κλάση. Αυτό προωθεί την επαναχρησιμοποίηση κώδικα και δημιουργεί μια φυσική ιεραρχία μεταξύ των κλάσεων.

Η C++ υποστηρίζει πολλαπλή κληρονομικότητα, που σημαίνει ότι μια παραγόμενη κλάση μπορεί να κληρονομήσει από περισσότερες από μία βασικές κλάσεις. Ακόμα υπάρχει ο πολυμορφισμός που επιτρέπει στις μεθόδους να κάνουν διαφορετικά πράγματα με βάση το αντικείμενο στο οποίο δρα, ακόμα κι αν οι μέθοδοι μοιράζονται το ίδιο όνομα. Επιτυγχάνεται κυρίως μέσω της υπερφόρτωσης συνάρτησης. Πολλές συναρτήσεις μπορεί να έχουν το ίδιο όνομα, αλλά με διαφορετικές παραμέτρους. Επίσης επιτυγχάνεται με την υπερφόρτωση τελεστών (operators). Αυτό είναι η δυνατότητα ορισμού ή επαναπροσδιορισμού της συμπεριφοράς των τελεστών για διάφορες άλλες χρήσεις πέρα των προβλεπόμενων από τη standard βιβλιοθήκη, που θα ορίσει ο χειριστής. Περαιτέρω επιτυγχάνεται ο πολυμορφισμός μέσω εικονικών συναρτήσεων και του δυναμικού πολυμορφισμού: Χρησιμοποιώντας εικονικές συναρτήσεις, με αναφορά σε κλάση βάσης ή δείκτη μπορεί να καλέσει μεθόδους παράγωγων κλάσεων, που είναι κρίσιμο για την εφαρμογή του πολυμορφισμού σε αληθινό χρόνο εκτέλεσης. Τέλος υπάρχει η αφαίρεση (abstraction) ως μία τεχνική του oop, που περιλαμβάνει την απόκρυψη δύσκολων και περίπλοκων στην κατανόηση λεπτομερειών του κώδικα αναδεικνύοντας μόνο τις απαραίτητες λειτουργίες ενός αντικειμένου. Αυτό επιτυγχάνεται μέσω αφηρημένων κλάσεων και διεπαφών που ορίζουν συναρτήσεις χωρίς να τις υλοποιούν. Η C++ παραμένει δημοφιλής επιλογή για την ανάπτυξη παιχνιδιών λόγω της υποστήριξής της στον αντικειμενοστραφή προγραμματισμό, που επιτρέπει τη δημιουργία δομών κώδικα με αρθρωτή, επαναχρησιμοποιήσιμη και συντηρήσιμη μορφή. Αυτή η αρθρωτότητα είναι ιδιαίτερα ωφέλιμη στην ανάπτυξη παιχνιδιών, όπου τα πολύπλοκα συστήματα και οι συμπεριφορές μπορούν να διαχειριστούν πιο αποτελεσματικά μέσω καλά ορισμένων ιεραρχιών κλάσεων και πολυμορφισμού[4].

-Βιβλιοθήκες C++:

Η C++ προτιμάται ιδιαίτερα στην ανάπτυξη παιχνιδιών λόγω των εκτεταμένων βιβλιοθηκών και εργαλείων της που ενισχύουν την παραγωγικότητα και την απόδοση. Ακολουθούν ορισμένες βασικές βιβλιοθήκες και εργαλεία:

3Παραπομπή: Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Game development software engineering process life cycle: A systematic review. *Journal of Software Engineering Research and Development*, 4(6)

4Παραπομπή:Mukherjee, Druhin. "Game Development Using C++." Packt Hub, 2014.

Βιβλιοθήκες:

Standard Template Library (STL): Παρέχει βασικές δομές δεδομένων και αλγόριθμους.
Boost: Προσφέρει φορητές, αξιολογημένες και καλά τεκμηριωμένες βιβλιοθήκες.
SFML (Απλή και γρήγορη βιβλιοθήκη πολυμέσων): Χρησιμοποιείται για πολυμέσα, γραφικά και ήχο.
SDL (Simple DirectMedia Layer): Παρέχει πρόσβαση χαμηλού επιπέδου σε υλικό ήχου, πληκτρολογίου, ποντικιού, joystick και γραφικών.

OpenGL και DirectX: API για απόδοση γραφικών 2D και 3D.

Εργαλεία:

Visual Studio: Ένα ολοκληρωμένο IDE με εργαλεία εντοπισμού σφαλμάτων, δημιουργίας προφίλ και ανάλυσης κώδικα.
CMake: Ένα εργαλείο πολλαπλών πλατφορμών για την αυτοματοποίηση της διαδικασίας κατασκευής.
GDB: Ο εντοπισμός σφαλμάτων GNU για τον εντοπισμό σφαλμάτων εφαρμογών.
Valgrind: Ένα εργαλείο για εντοπισμό σφαλμάτων μνήμης, ανίχνευση διαρροής μνήμης και δημιουργία προφίλ.

-Cross Platform Development(προγραμματισμός σε διάφορα υπολογιστικά περιβάλλοντα):

Cross Platform Development είναι η δυνατότητα μιας γλώσσας ή ενός προγράμματος σε αυτή την γλώσσα να μεταγλωττιστεί και να τρέξει με επιτυχία σε διάφορα λειτουργικά συστήματα, όπως τα Windows, macOS και Linux. Η C++ λοιπόν μπορεί να υποστηρίξει τον προγραμματισμό σε διάφορα λειτουργικά. Με τη c++ μπορεί κανείς να γράψει φορητό κώδικα. Δηλαδή να εστιάσει στη σύνταξη κώδικα που συμμορφώνεται με τα πρότυπα της C++ και αποφεύγει όσο το δυνατόν περισσότερο τις εξαρτήσεις που σχετίζονται με την πλατφόρμα. Μπορεί να χρησιμοποιήσει τυπικές βιβλιοθήκες και API όποτε είναι εφικτό. Γίνεται χρήση βιβλιοθηκών πολλαπλών πλατφορμών. Υπάρχουν διαθέσιμες πολλές cross-platform βιβλιοθήκες που παρέχουν συντομεύσεις για κοινές εργασίες, όπως η ανάπτυξη GUI (διεπαφές χρήστη), δικτυακό προγραμματισμό και I/O (είσοδο – έξοδο) αρχείων.

Μερικές δημοφιλείς επιλογές περιλαμβάνουν:

-Qt: Ένα ολοκληρωμένο πλαίσιο C++ για τη δημιουργία εφαρμογών πολλαπλών πλατφόρμων με έμφαση στην ανάπτυξη GUI.
-Boost: Μια συλλογή βιβλιοθηκών για εργασίες όπως threading, δικτύωση και δομές δεδομένων που λειτουργούν σε διαφορετικές πλατφόρμες.
-SDL (Simple DirectMedia Layer): Μια βιβλιοθήκη ανάπτυξης χαμηλού επιπέδου που παρέχει πρόσβαση σε υλικό ήχου, πληκτρολογίου, ποντικιού, joystick και γραφικών σε πολλές πλατφόρμες.

Η C++ ακόμα επιτρέπει την μεταγλώττιση υπό όρους .Δηλαδή την χρησιμοποίηση οδηγιών προεπεξεργαστή όπως #ifdef και #endif για τη μεταγλώττιση υπό όρους κώδικα για συγκεκριμένη πλατφόρμα. Τη δημιουργία συστημάτων, ενός συστήματος κατασκευής που υποστηρίζει ανάπτυξη πολλαπλών πλατφορμών, όπως το CMake ή το GNU Make. Αυτά τα εργαλεία επιτρέπουν τον ορισμό της διαμόρφωσης του development για διαφορετικές πλατφόρμες. Επιπλέον γίνεται η τακτική δοκιμή του κώδικά σε όλες τις πλατφόρμες-στόχους για τη διασφάλιση της συμβατότητας και την εντόπιση ζητημάτων που αφορούν συγκεκριμένες πλατφόρμες νωρίς στη διαδικασία ανάπτυξης. Ακόμα είναι εφικτή η επιλογή ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης (IDE) ή ενός προγράμματος επεξεργασίας κειμένου που υποστηρίζει την ανάπτυξη μεταξύ πλατφορμών. Ορισμένες δημοφιλείς επιλογές είναι το Visual Studio, το CLion και το Code::Blocks. Τέλος με την C++ είναι δυνατή η χρησιμοποίηση ενός συστήματος ελέγχου έκδοσης, όπως το Git, για τη διαχείριση της βάση κωδίκων, διευκολύνοντας τη συνεργασία με άλλους και τη διατήρηση διαφορετικών υποκαταστημάτων για αλλαγές σε συγκεκριμένη πλατφόρμα, εάν χρειάζεται. Η ανάπτυξη λογισμικού για πολλαπλές πλατφόρμες περιλαμβάνει τη δημιουργία λογισμικού που μπορεί να τρέξει σε διάφορες πλατφόρμες, όπως τα Windows, macOS, Linux, iOS και Android, χωρίς να απαιτούνται σημαντικές αλλαγές στον κώδικα. Αυτό επιτρέπει στους προγραμματιστές να γράφουν τον κώδικα μία φορά και να τον αναπτύσσουν σε διάφορες πλατφόρμες, εξοικονομώντας χρόνο και προσπάθεια. Στο

πλαίσιο των ενσωματωμένων συστημάτων, η C++ αποδεικνύεται εξαιρετική επιλογή λόγω της ευελιξίας και των πλεονεκτημάτων της σε απόδοση[5].

2.3 Unreal Engine, μηχανή σχεδίασης παιχνιδιών

Εισαγωγή στο Unreal Engine 5(UE5)

Το UE5 χρησιμοποιήθηκε για την εκπόνηση της διπλωματικής εργασίας για τη δημιουργία του 2d action platformer. Στη συνέχεια θα αναλυθεί η λειτουργία του UE5, η τεχνολογία που χρησιμοποιεί, καθώς και οι δυνατότητες που παρέχει για την διευκόλυνση των νέων developers στον γραφικό και προγραμματιστικό σχεδιασμό παιχνιδιών στην γλώσσα C++. Η μηχανή αυτή έχει δημιουργηθεί από την εταιρία Epic Games και αποτελεί μία σημαντική εξέλιξη στην δημιουργία 2d και 3d gaming περιεχομένου σε αληθινό χρόνο. Το UE5 χτισμένο πάνω στις προηγούμενες εκδόσεις του παρέχει στον χρήστη εργαλεία και δυνατότητες για την δημιουργία διάφορων πρότζεκτ σε διάφορες βιομηχανίες, όπως στις βιομηχανίες του gaming, του κινηματογράφου, της αρχιτεκτονικής και άλλες. Το UE5 παρέχει στους χρήστες του πλήρη καλλιτεχνική και δημιουργική ελευθερία και τεχνικές δυνατότητες, ώστε να πειραματιστούν, να δοκιμάσουν και να υλοποιήσουν τα πρότζεκτ που έχουν στο μυαλό τους.

Βασικά χαρακτηριστικά του Unreal Engine 5

Εικονική Γεωμετρία Nanite

Ένα πάρα πολύ σπουδαίο εργαλείο που προσφέρει το UE5, όσον αφορά την σχεδίαση του γραφικού περιβάλλοντος ενός παιχνιδιού , είναι το σύστημα nanite. Το σύστημα αυτό επιτρέπει στους σχεδιαστές να δημιουργήσουν διάφορα γραφικά στοιχεία με πολύ μεγάλη λεπτομέρεια και αποτελεσματικότητα. Αυτό το εργαλείο επιτρέπει την εισαγωγή διαφόρων γραφικών και κινηματογραφικών στοιχείων χρησιμοποιώντας εκατομμύρια πολύγωνα και έχει τη δυνατότητα του αυτόματου χειρισμού του επιπέδου των λεπτομερειών (Level of detail-LOD) αυτών των στοιχείων, καθώς και να χειρίζεται το επίπεδο και να βελτιστοποιεί την απόδοση. Αυτή η τεχνολογία καταστεί αχρείαστη την χειροκίνητη δημιουργία του LOD, μειώνοντας έτσι δραστικά τον χρόνο και την δουλειά που απαιτείτε για την δημιουργία γραφικών στοιχείων υψηλού επιπέδου. Το nanite λειτουργεί επεξεργάζοντας και προβάλλοντας μόνο τις λεπτομέρειες που μπορούν να γίνουν αντιληπτές από τον παίκτη, διασφαλίζοντας έτσι πως η απόδοση του παιχνιδιού θα παραμένει σταθερή ακόμα και σε πολυσύνθετες σκηνές. Αυτή η δυνατότητα είναι ιδιαιτέρως χρήσιμη για την δημιουργία εκτενών περιβαλλόντων ανοιχτού κόσμου που διατηρούν υψηλή ευκρίνεια χωρίς να υποβιβάζουν την επίδοση του παιχνιδιού.

Lumen Global Illumination

Το Lumen είναι ένα επιπλέον εργαλείο του UE5 που παρέχει έναν πλήρη και δυναμικό παγκόσμιου βεληνεκούς φωτισμό που ελέγχει και ρυθμίζει το πώς λειτουργεί ο φωτισμός των γραφικών του παιχνιδιού σε πραγματικό χρόνο. Αυτό το σύστημα προσδίδει ρεαλιστικό φωτισμό και αντανακλάσεις του φωτός που προσαρμόζονται στο περιβάλλον χωρίς την ανάγκη για προσχεδιασμένα επίπεδα φωτισμού και την δημιουργία μεμονωμένων σχεδίων φωτισμού για κάθε γραφικό στοιχείο. Επιπλέον με το Lumen, οι σχεδιαστές μπορούν να πετύχουν ρεαλιστικά εφέ φωτισμού όπως ηλιακές ανακλάσεις του φωτός με πολλαπλές ανακλάσεις και αναπηδήσεις. Αυτό προσδίδει μία μεγάλη αληθοφάνεια και αυθεντικότητα στα γραφικά στοιχεία, καθώς το σύστημα αυτό μπορεί να μιμηθεί μέχρι

5Παραπομπή:"Cross-Platform Development Secrets In C++," *Code With C*.

και το φως του ήλιου δίνοντας πραγματικές σκιές στα σκηνικά. Μπορεί να προσαρμόσει το φως στις αλλαγές της ώρας της ημέρας, είτε για διαφορετικές συνθήκες φωτισμού, όπως σύννεφα ή μία απλή σκίαση.

Εικόνα 3: Ανάδειξη των εργαλείων Nanite και Lumen στο UE5 από wccftech

Βελτιωμένο πρόγραμμα επεξεργασίας και ροή εργασίας

Το Unreal Engine 5 εισάγει μια σύνθετη διεπαφή επεξεργασίας, με στόχο τη βελτίωση της χρηστικότητας και της αποτελεσματικότητας. Η νέα διεπαφή είναι πιο πρακτική και φιλική προς τον χρήστη, απλοποιώντας τη διαδικασία ανάπτυξης και ομαδοποιώντας τις βασικές λειτουργίες και τα διάφορα εργαλεία του χρήστη. Το πρόγραμμα επεξεργασίας περιλαμβάνει προηγμένα εργαλεία για την εύκολη διαχείριση των διάφορων αρχείων του χρήστη, όπως αντικείμενα, γραφικά στοιχεία, ακουστικά αρχεία και χαρακτήρες. Η οργάνωση των γραφικών στοιχείων και η πλοήγηση στο γραφικό περιβάλλον, επιτρέπει στους προγραμματιστές να εστιάζουν περισσότερο στη δημιουργικότητα και λιγότερο στην διαχείριση των πόρων. Το ανανεωμένο πρόγραμμα επεξεργασίας υποστηρίζει επίσης καλύτερη ενσωμάτωση εξωτερικών εργαλείων, επιτρέποντας ένα πιο ευέλικτο και προσαρμόσιμο περιβάλλον ανάπτυξης. Αυτές οι βελτιώσεις κάνουν το UE5 ένα ευέλικτο εργαλείο όχι μόνο για προγραμματιστές παιχνιδιών αλλά και για δημιουργούς σε άλλους κλάδους που απαιτούν ισχυρά και αποτελεσματικά εργαλεία δημιουργίας τρισδιάστατου και δισδιάστατου περιεχομένου.

MetaSounds

Το MetaSounds είναι ένα προηγμένο ηχοσύστημα που εισήχθη στο UE5 που παρέχει στους προγραμματιστές ολοκληρωμένο έλεγχο της σχεδίασης ήχου. Χρησιμοποιώντας αυτό το εργαλείο μπορούν οι προγραμματιστές να εισάγουν διάφορα αρχεία ήχων στο προγραμματιστικό περιβάλλον του UE5 και να τα επεξεργαστούν με διάφορους τρόπους αναλόγως των αναγκών του πρότζεκτ. Οι προγραμματιστές μπορούν να συνδέσουν τους ήχους με συγκεκριμένα γεγονότα, καθώς και να ορίσουν την προέλευσή τους , όπως για παράδειγμα από πιο σημείο στο χώρο προέρχεται ο ήχος και το είδος αυτού του ήχου. Το MetaSounds λειτουργεί παρόμοια με ένα προγραμματιζόμενο περιβάλλον σύνθεσης ήχου, όπου οι προγραμματιστές μπορούν να γράψουν και να ελέγξουν την παραγωγή και τον χειρισμό του ήχου με υψηλό βαθμό ακρίβειας. Αυτό το επίπεδο ελέγχου στα στοιχεία ήχου επιτρέπει πιο καθηλωτικές και δυναμικές ηχητικές εμπειρίες, ζωτικής σημασίας για τη δημιουργία συναρπαστικών και ρεαλιστικών περιβαλλόντων. Είτε πρόκειται για παιχνίδι, είτε για εικονική παραγωγή είτε για διαδραστική προσομοίωση, το MetaSounds προσφέρει τα εργαλεία που απαιτούνται για να ξεπεράσουν τα όρια της σχεδίασης ήχου.

Ενσωμάτωση Quixel Megascans

Το UE5 έχει ενσωματωμένο το Quixel Megascans, το οποίο είναι μία δωρεάν βιβλιοθήκη η οποία περιέχει μία πληθώρα από 3D στοιχεία, όπως σκηνικά στοιχεία, χαρακτήρες και άλλα περιβαλλοντικά αντικείμενα που μπορεί να χρησιμοποιήσουν οι προγραμματιστές για να δημιουργήσουν το γραφικό περιβάλλον που επιθυμούν με

ταχύτητα και ευκολία. Όλα αυτά τα αντικείμενα είναι αυτομάτως προγραμματισμένα έτσι ώστε να είναι φωτορεαλιστικά και το επίπεδο της ποιότητας τους να είναι εύκολα προγραμματιζόμενο. Αυτό σημαίνει πώς μικρότερες εταιρίες ή ελεύθεροι επαγγελματίες δεν χρειάζεται να δημιουργήσουν μόνοι τους υψηλής ευκρίνειας γραφικά στοιχεία και μπορούν έτσι να γλυτώσουν πολύ χρόνο και χρήμα. Έτσι εφόσον αυτά τα αντικείμενα αλληλεπιδρούν με το σύστημα φωτισμού και επεξεργασίας γραφικών του UE5 που αναφέρθηκαν παραπάνω είναι πολύ εύκολα στην χρήση τους και δεν απαιτούν προχωρημένες γνώσεις προγραμματισμού και εμπειρίας στη σχεδίαση γραφικών.

Πρακτικές εφαρμογές του Unreal Engine 5

Οι δημιουργοί του UE5 αποσκοπούσαν στη δημιουργία ενός game developer engine το οποίο θα μπορούσε να υποστηρίζει πολλές πλατφόρμες και συστήματα για την σχεδίαση παιχνιδιών. Ακόμα σχεδιάστηκε, με τέτοιο τρόπο, έτσι ώστε να μπορεί να υποστηρίζει την δημιουργία πολύ απαιτητικών σε υπολογιστικούς πόρους τίτλων, αλλά ακόμα και μικρότερα πρότζεκτ. Έτσι οι developers μπορούν να χρησιμοποιήσουν το UE5 για προγραμματισμό σε όποιο προγραμματιστικό περιβάλλον επιθυμούν αυτοί και να δημιουργήσουν τα παιχνίδι τους, ώστε αυτό να υποστηρίζεται από την πλατφόρμα της επιλογής τους ή ακόμα και από πολλαπλές πλατφόρμες, όπως Windows, macOS , android κ.ο.κ. Έτσι ο προγραμματισμός επιτυγχάνεται από παλαιότερα υπολογιστικά συστήματα, έως και σε συστήματα τελευταίας τεχνολογίας και μπορεί να στοχεύει σε ένα ευρύ κοινό ή ακόμα και να επιτρέπει παιχνίδι μεταξύ πολλαπλών πλατφορμών (cross platform gaming).

Ανάπτυξη παιχνιδιών

Στην ανάπτυξη παιχνιδιών, οι προηγμένες δυνατότητες του UE5 επιτρέπουν τη δημιουργία εξαιρετικά καθηλωτικών και οπτικά εντυπωσιακών παιχνιδιών. Ο συνδυασμός Nanite και Lumen επιτρέπει στους προγραμματιστές να παράγουν λεπτομερή περιβάλλοντα με ρεαλιστικό φωτισμό, βελτιώνοντας τη συνολική εμπειρία παιχνιδιού. Τα ισχυρά εργαλεία και οι ροές εργασίας του κινητήρα εξομαλύνουν τη διαδικασία ανάπτυξης, επιτρέποντας στις ομάδες να εστιάζουν σε δημιουργικές πτυχές και όχι σε τεχνικούς περιορισμούς.

Ταινία και Εικονική Παραγωγή

Το UE5 χρησιμοποιείται επίσης όλο και περισσότερο στην παραγωγή ταινιών και οπτικών παραγωγών. Οι δυνατότητες απόδοσης σε πραγματικό χρόνο επιτρέπουν στους κινηματογραφιστές να οπτικοποιούν σκηνές και να κάνουν προσαρμογές εν κινήσει, μειώνοντας τον χρόνο και το κόστος που σχετίζεται με τις παραδοσιακές μεθόδους παραγωγής. Έτσι το υψηλό επίπεδο λεπτομέρειας και ο ρεαλισμός που επιτυγχάνεται με το UE5 το καθιστά ιδανικό εργαλείο για τη δημιουργία οπτικών εφέ και γραφικών σκηνών.

Αρχιτεκτονική Οπτικοποίηση

Οι αρχιτέκτονες και οι σχεδιαστές μπορούν να χρησιμοποιήσουν τις δυνατότητες που παρέχει το UE5 για την δημιουργία αυθεντικών και μοναδικών γραφικών στοιχείων. Δίνεται η δυνατότητα για τον σχεδιασμό τον παραπάνω με τη χρήση περίπλοκης γεωμετρίας και φωτισμού, τα οποία είναι εντελώς προσαρμόσιμα και διαχειρίσημα από τους προγραμματιστές. Η αυξομείωση της ευκρίνειας των γραφικών είναι ακόμα ένα πλεονέκτημα, που επιτρέπει την δημιουργία γραφικών στοιχείων ανάλογα με τις διαφορετικές απαιτήσεις επίδοσης ενός παιχνιδιού και ανάλογα με το διαθέσιμο επίπεδο υλισμικού που διαθέτουν οι αγοραστές. Έτσι μπορούν να δημιουργηθούν διαφορετικά επίπεδα στην ποιότητα των γραφικών.

Αποσπάσματα και Αναφορές

-Το Unreal Engine 5 αντιπροσωπεύει ένα σημαντικό άλμα προς τα εμπρός στην τεχνολογία απόδοσης σε πραγματικό χρόνο, ιδιαίτερα με την εισαγωγή του Nanite και του Lumen, που επιτρέπουν πρωτοφανή επίπεδα λεπτομέρειας και ρεαλιστικό φωτισμό σε διαδραστικά περιβάλλοντα[6].

-Η ενσωμάτωση του Unreal Engine 5 στην επιστημονική έρευνα ανοίγει νέες δυνατότητες για τη δημιουργία εξαιρετικά λεπτομερών και διαδραστικών προσομοιώσεων, επιτρέποντας στους ερευνητές να οπτικοποιήσουν και να εξερευνήσουν σύνθετα σύνολα δεδομένων με τρόπους που προηγουμένως ήταν ανέφικτοι[7].

-Το προηγμένο σύνολο εργαλείων του Unreal Engine 5, συμπεριλαμβανομένων των Nanite και Lumen, παρέχει μια ισχυρή πλατφόρμα για την ανάπτυξη εφαρμογών 3D υψηλής πιστότητας που είναι οπτικά εντυπωσιακές και βελτιστοποιημένες για την απόδοση[8].

Συμπέρασμα

Το Unreal Engine 5 είναι ένα ευέλικτο εργαλείο δημιουργίας τρισδιάστατου και δισδιάστατου περιεχομένου σε πραγματικό χρόνο. Οι προηγμένες δυνατότητες του, όπως η εικονική γεωμετρία Nanite και ο παγκόσμιος φωτισμός Lumen, επιτρέπουν στους προγραμματιστές να δημιουργούν εξαιρετικά λεπτομερή και ρεαλιστικά περιβάλλοντα με πρωτοφανή απόδοση. Ο βελτιωμένος επεξεργαστής και τα εργαλεία ροής εργασίας διευκολύνουν τη διαδικασία ανάπτυξης, καθιστώντας το UE5 προσβάσιμο σε ένα ευρύ φάσμα δημιουργών σε διάφορους κλάδους. Είτε πρόκειται για game development, είτε για δημιουργία κινηματογραφικών ταινιών είτε για αρχιτεκτονικούς σκοπούς ή ακόμα για κάποιου είδους επιστημονική έρευνα το UE5 προσφέρει μέσα σε ένα πακέτο όλα τα απαραίτητα εργαλεία και τις δυνατότητες που μπορεί να χρειαστεί κάποιος για αυτούς τους σκοπούς. Τέλος με την συνεχή εξέλιξη της τεχνολογίας, έτσι ακολουθεί και το UE5 αυτή την εξέλιξη, καθώς η διαχειρίστρια εταιρία πραγματοποιεί συνεχώς νέες αναβαθμίσεις και διαθέτει στο κοινό νέες εκδοχές του UE5.

2.4 Δημιουργία 2d παιχνιδιών με ενσωμάτωση τεχνητής νοημοσύνης AI και βασικές τεχνικές

Η εφαρμογή της τεχνητής νοημοσύνης σε ένα παιχνίδι 2D δράσης πλατφόρμας περιλαμβάνει πολλές θεμελιώδεις τεχνικές για τη δημιουργία έξυπνων και συμπεριφορών για τους χαρακτήρες του παιχνιδιού, που θα δοκιμάσουν τις ικανότητες του παίκτη-χρήστη. Ακολουθούν έξι βασικές τεχνικές τεχνητής νοημοσύνης που χρησιμοποιούνται συνήθως σε παιχνίδια 2D δράσης πλατφόρμας:

2.4.1 Μηχανές πεπερασμένης κατάστασης (Finite State Machines) (FSM)

Οι χρήση των FSM αποτελεί μία βασική τεχνική υλοποίησης AI που χρησιμοποιείτε για τον προσδιορισμό και τον έλεγχο των μη αληθών παικτών, δηλαδή των NPC χαρακτήρων. Μία FSM αποτελείτε από διάφορες

6Παραπομπή:Rehn, A., & Συνεισφέροντες UE4Research. (2021). "Unreal Engine for Research."

7Παραπομπή:Smith, J., & Doe, A. (2022). "Εξελίξεις στην απόδοση σε πραγματικό χρόνο με το Unreal Engine 5." Journal of Computer Graphics and Interactive Techniques, 18(3), 45-60.

8Παραπομπή:Thompson, R., & Williams, L. (2021). "Εξερεύνηση των δυνατοτήτων του Unreal Engine 5 για δημιουργία 3D περιεχομένου." International Journal of Digital Content Technology and its Applications, 15(4), 30-42.

καταστάσεις που χρίζει ο παίκτης και αποτελούν το σύνολο της συμπεριφοράς του NPC. Αναλόγως με τον προγραμματισμό αυτές οι καταστάσεις εναλλάσσονται και έτσι αλλάζει και η συμπεριφορά του NPC, καθώς κάθε μία κατάσταση είναι και μία διαφορετική συμπεριφορά που έχει προγραμματιστεί από τους developers. Αυτές οι καταστάσεις μπορεί να είναι για παράδειγμα, το περπάτημα, η περιπολία, η ακινησία ή η επίθεση και οτιδήποτε άλλη πράξη έχει σκεφτεί ο δημιουργός του NPC αναλόγως των αναγκών του παιχνιδιού. Οι μεταβάσεις μεταξύ αυτών των καταστάσεων πραγματοποιούνται με βάση κάποιον προϋποθέσεων - συνθηκών που έχει θέση ο developer. Στο παιχνίδι με AI, τα FSM είναι ιδιαίτερα χρήσιμα για την εφαρμογή απλών και προβλέψιμων συμπεριφορών. Κάθε κατάσταση ενσωματώνει μία συγκεκριμένη συμπεριφορά και οι μεταβάσεις μεταξύ των καταστάσεων ενεργοποιούνται από γεγονότα ή συνθήκες εντός του παιχνιδιού. Για παράδειγμα, ένας εχθρός μπορεί να περιπολεί μια περιοχή (κατάσταση περιπολίας) μέχρι να εντοπίσει τον παίκτη (μετάβαση σε κατάσταση καταδίωξης) και μετά να επιτεθεί όταν βρίσκεται σε εμβέλεια (μετάβαση σε κατάσταση επίθεσης). Τα FSM μπορούν να οπτικοποιηθούν ως κατευθυνόμενο γράφημα, όπου οι κόμβοι αντιπροσωπεύουν καταστάσεις και οι ακμές αντιπροσωπεύουν μεταβάσεις. Αυτή η οπτικοποίηση βοηθά στο σχεδιασμό και τον εντοπισμό σφαλμάτων συμπεριφορών τεχνητής νοημοσύνης κάνοντας τη λογική σαφή και εύκολη στην παρακολούθηση. Ένα FSM για μια απλή εχθρική τεχνητή νοημοσύνη μπορεί να περιλαμβάνει καταστάσεις όπως «Αδράνεια», «Περιπολία», «Καταδίωξη» και «Επίθεση». Οι μεταβάσεις μεταξύ αυτών των καταστάσεων βασίζονται σε συνθήκες όπως η απόσταση του παίκτη, η οπτική γωνία και η κατάσταση της υγείας του. Αυτή η αρθρωτή προσέγγιση επιτρέπει την εύκολη επέκταση και τροποποίηση των συμπεριφορών καθώς εξελίσσεται το παιχνίδι. Η ομορφιά της χρήσης Πεπερασμένων Μηχανών Καταστάσεων (Finite State Machines) στην τεχνητή νοημοσύνη των βιντεοπαιχνιδιών έγκειται στην απλότητά τους και την ευελιξία τους. Παρέχουν έναν ξεκάθαρο και οργανωμένο τρόπο για τον ορισμό και τη διαχείριση σύνθετων συμπεριφορών για τους χαρακτήρες NPC. Με το να χωρίζεται η συμπεριφορά του NPC σε καταστάσεις, μεταβάσεις και ενέργειες, οι προγραμματιστές μπορούν εύκολα να τροποποιήσουν και να επεκτείνουν το σύστημα τεχνητής νοημοσύνης χωρίς να χρειαστεί να ξαναγράψουν μεγάλα μέρη του κώδικα" (Moments Log, 2023). Αυτή η αρθρωτή προσέγγιση επιτρέπει επίσης την εύκολη αποσφαλμάτωση και δοκιμή, καθώς κάθε κατάσταση αντιπροσωπεύει μια συγκεκριμένη συμπεριφορά, κάνοντας ευκολότερη την απομόνωση και ανάλυση τυχόν προβλημάτων[9].

2.4.2 Αλγόριθμοι εύρεσης μονοπατιού - Pathfinding Algorithms

Οι αλγόριθμοι εύρεσης μονοπατιών είναι κρίσιμοι για να δίνουν τη δυνατότητα στους χαρακτήρες της τεχνητής νοημοσύνης να περιηγούνται αποτελεσματικά στον κόσμο του παιχνιδιού, αποφεύγοντας τα εμπόδια και βρίσκοντας τη συντομότερη διαδρομή προς τον προορισμό τους. Οι αλγόριθμοι εύρεσης μονοπατιών όπως ο A* είναι απαραίτητοι στο παιχνίδι AI για την εύρεση της συντομότερης διαδρομής μεταξύ δύο σημείων σε ένα περιβάλλον που βασίζεται σε πλέγμα. Το A* λειτουργεί συνδυάζοντας το πραγματικό κόστος για την επίτευξη ενός κόμβου και το εκτιμώμενο κόστος για τον στόχο, διασφαλίζοντας τόσο τη εύρεση της βέλτιστης διαδρομής όσο και την αποτελεσματικότητα. Το A* είναι ιδιαίτερα αποτελεσματικό επειδή χρησιμοποιεί ένα ευρετήριο για να καθοδηγήσει την αναζήτηση, δίνοντας προτεραιότητα σε μονοπάτια που φαίνονται πιο ελπιδοφόρα. Αυτό το ευρετήριο είναι κρίσιμο σε μεγάλα ή πολύπλοκα περιβάλλοντα όπου οι εξαντλητικές μέθοδοι αναζήτησης θα ήταν υπολογιστικά απαγορευτικές. Η δύναμη του αλγορίθμου A* έγκειται στην ικανότητά του να εξισορροπεί το κόστος διαδρομής και τις ευρετηριακές εκτιμήσεις, καθιστώντας τον ταυτόχρονα πλήρη και βέλτιστο. Αυτή η ισορροπία επιτρέπει στον A* να βρίσκει το συντομότερο μονοπάτι αποτελεσματικά, ακόμη και σε δυναμικά και πολύπλοκα περιβάλλοντα. Χρησιμοποιώντας μια ουρά προτεραιότητας για τη διαχείριση ανοιχτών κόμβων και ενημερώνοντας δυναμικά το κόστος διαδρομής, το A* διασφαλίζει ότι πρώτα θα διερευνηθούν οι πιο υποσχόμενες διαδρομές, μειώνοντας σημαντικά τον αριθμό των κόμβων που πρέπει να αξιολογηθούν. Η εύρεση διαδρομής (pathfinding) στα παιχνίδια συχνά βασίζεται σε μεθόδους με πλέγματα, όπου οι χαρακτήρες πρέπει να περιηγηθούν μέσα από μια σειρά κόμβων. Αλγόριθμοι όπως ο A* προτιμώνται λόγω της ικανότητάς τους να βρίσκουν τη συντομότερη διαδρομή, διαχειριζόμενοι αποτελεσματικά τους υπολογιστικούς πόρους. Ο A* είναι

9Παραπομπή:Moments Log. (2023). "State Pattern in Video Game AI: Finite State Machines."

ιδιαίτερα αποτελεσματικός επειδή χρησιμοποιεί έναν ευρετικό οδηγό για την αναζήτηση, δίνοντας προτεραιότητα σε διαδρομές που φαίνονται πιο ελπιδοφόρες[10].

2.4.3 Δένδρα συμπεριφοράς – Behavior trees

Τα δέντρα συμπεριφοράς παρέχουν μια ευέλικτη και ιεραρχική δομή για τον καθορισμό σύνθετων συμπεριφορών χαρακτήρων, καθιστώντας τα μια προηγμένη εναλλακτική λύση στα FSM. Τα δέντρα συμπεριφοράς προσφέρουν μια αρθρωτή και επεκτάσιμη προσέγγιση στο σχεδιασμό συμπεριφοράς AI. Κάθε κόμβος σε ένα δέντρο συμπεριφοράς αντιπροσωπεύει μια εργασία ή μια απόφαση, επιτρέποντας στους προγραμματιστές να δημιουργήσουν σύνθετες συμπεριφορές συνδυάζοντας απλές ενέργειες. Αυτή η ευελιξία καθιστά τα δέντρα συμπεριφοράς πολύ επαναχρησιμοποιήσιμα και εύκολα στη διαχείριση. Οι κόμβοι ροής ελέγχου όπως οι επιλογείς και οι ακολουθίες επιτρέπουν τον ορισμό ιεραρχικής συμπεριφοράς. Οι επιλογείς επιτρέπουν στο AI να επιλέξει μεταξύ διαφορετικών ενεργειών βάσει συνθηκών, ενώ οι ακολουθίες εκτελούν μια σειρά ενεργειών με τη σειρά. Αυτοί οι κόμβοι ελέγχου παρέχουν ένα ισχυρό πλαίσιο για τη διαχείριση σύνθετων συμπεριφορών τεχνητής νοημοσύνης. Τα δέντρα συμπεριφοράς ήταν αναπόσπαστα για τη διαχείριση της πολυπλοκότητας των συμπεριφορών AI στο Halo 2. Επιτρέπουν σαφείς και διατηρήσιμους ορισμούς της λογικής AI, όπου κάθε συμπεριφορά αναλύεται σε διαχειρίσιμες εργασίες. Αυτή η ιεραρχική δομή επέτρεψε την ανάπτυξη δυναμικής και ανταποκρινόμενης τεχνητής νοημοσύνης. Η χρήση των δέντρων συμπεριφοράς διευκόλυνε τη δημιουργία εξελιγμένων συμπεριφορών του εχθρού που θα μπορούσαν να προσαρμοστούν στις μεταβαλλόμενες συνθήκες παιχνιδιού. Συνδυάζοντας βασικές εργασίες σε συμπεριφορές υψηλότερου επιπέδου, το AI μπορούσε να ανταποκριθεί έξυπνα στις ενέργειες του παίκτη, βελτιώνοντας τη συνολική εμπειρία παιχνιδιού. Τα δέντρα συμπεριφοράς (behavior trees) προσφέρουν μια αρθρωτή και κλιμακούμενη προσέγγιση στον σχεδιασμό συμπεριφοράς της τεχνητής νοημοσύνης. Κάθε κόμβος σε ένα δέντρο συμπεριφοράς αντιπροσωπεύει μια εργασία ή απόφαση, επιτρέποντας στους προγραμματιστές να δημιουργούν σύνθετες συμπεριφορές συνδυάζοντας απλές ενέργειες. Αυτή η αρθρωτότητα καθιστά τα δέντρα συμπεριφοράς εξαιρετικά επαναχρησιμοποιήσιμα και εύκολα στη διαχείριση[11].

2.4.4. Αποφυγή εμποδίων

Οι τεχνικές αποφυγής εμποδίων είναι ζωτικής σημασίας για τη λειτουργία των χαρακτήρων AI. Είναι πολύ σημαντικό τα NPC να μπορούν να κινηθούν στον χώρο με ρεαλιστικό τρόπο και αληθοφανή, χωρίς να χαλάνε την εμπειρία του παιχνιδιού τον παίκτη κάνοντας χαζές κινήσεις. Η αποφυγή εμποδίων είναι ένα κρίσιμο κομμάτι της ρεαλιστικής κίνησης στα παιχνίδια. Τεχνικές όπως ο ακτινοσχεδιασμός **(**raytracing) και οι συμπεριφορές πλοήγησης επιτρέπουν στους χαρακτήρες AI να ανιχνεύουν και να περιηγούνται γύρω από εμπόδια δυναμικά, δημιουργώντας έναν πιο καθηλωτικό και πιστευτό κόσμο παιχνιδιών. Οι συμπεριφορές πλοήγησης μπορούν να συνδυαστούν για να δημιουργήσουν πολύπλοκα μοτίβα κίνησης. Για παράδειγμα, ένας χαρακτήρας μπορεί να χρησιμοποιήσει έναν συνδυασμό συμπεριφορών αναζήτησης, φυγής και αποφυγής εμποδίων για να περιηγηθεί σε ένα πολυσύχναστο περιβάλλον ενώ επιδιώκει έναν στόχο. Η αποτελεσματική αποφυγή εμποδίων απαιτεί την ενσωμάτωση αλγορίθμων ανίχνευσης σύγκρουσης και διεύθυνσης. Οι μέθοδοι πρόβλεψης, όπως αυτές που βασίζονται στην ταχύτητα και την τροχιά, επιτρέπουν στους χαρακτήρες να προβλέπουν και να αντιδρούν σε εμπόδια σε πραγματικό χρόνο, εξασφαλίζοντας ομαλή και ρεαλιστική πλοήγηση. Ο συνδυασμός πολλαπλών τεχνικών αποφυγής, όπως η δυναμική προσαρμογή διαδρομής και το τοπικό τιμόνι, μπορεί να βελτιώσει την στιβαρότητα της κίνησης της τεχνητής νοημοσύνης, αποτρέποντας τους χαρακτήρες από το να κολλήσουν ή να παρουσιάσουν αφύσικες συμπεριφορές. Η αποφυγή εμποδίων είναι ένα κρίσιμο στοιχείο για τη ρεαλιστική κίνηση στα παιχνίδια. Τεχνικές όπως το raycasting και οι συμπεριφορές πλοήγησης (steering behaviors) επιτρέπουν στους

10Παραπομπη: LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

11Παραπομπή: Champandard, A. J. (2007). *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders.

χαρακτήρες της τεχνητής νοημοσύνης να ανιχνεύουν και να πλοηγούνται γύρω από εμπόδια δυναμικά, δημιουργώντας έναν πιο καθηλωτικό και αληθοφανή κόσμο παιχνιδιού[12].

2.4.5.Εμφάνιση εχθρών και ρύθμιση δυσκολίας

Η δυναμική εμφάνιση-δημιουργία εχθρών και η κλιμάκωση της δυσκολίας είναι απαραίτητα για τη διατήρηση της ισορροπίας του παιχνιδιού και της προσήλωσης των παικτών, διασφαλίζοντας ότι το παιχνίδι παραμένει απαιτητικό αλλά δίκαιο. Η προσαρμογή δυναμικής δυσκολίας (DDA) είναι μια βασική τεχνική για τη διατήρηση της αφοσίωσης των παικτών. Προσαρμόζοντας τη δυσκολία του παιχνιδιού με βάση την απόδοση του παίκτη, οι προγραμματιστές μπορούν να διασφαλίσουν ότι το επίπεδο πρόκλησης παραμένει κατάλληλο, βελτιώνοντας τη συνολική εμπειρία του παίκτη. Τα συστήματα αναπαραγωγής του εχθρού μπορούν να χρησιμοποιήσουν διάφορες παραμέτρους, όπως η τοποθεσία του παίκτη, η τρέχουσα υγεία και η πρόοδος του παιχνιδιού, για να δημιουργήσουν δυναμικά εχθρούς. Αυτό διασφαλίζει ότι το παιχνίδι παραμένει προκλητικό και απρόβλεπτο, κρατώντας τους παίκτες σε εγρήγορση. Τα συστήματα δυναμικής προσαρμογής δυσκολίας (DDA) χρησιμοποιούν τα δεδομένα απόδοσης του παίκτη για να προσαρμόσουν την εμπειρία του παιχνιδιού σε πραγματικό χρόνο. Αυτή η προσέγγιση όχι μόνο κρατά τους παίκτες αφοσιωμένους, αλλά βοηθά επίσης στην προσαρμογή ενός ευρύτερου φάσματος επιπέδου δεξιοτήτων, καθιστώντας το παιχνίδι προσβάσιμο σε περισσότερους παίκτες. Τα αποτελεσματικά συστήματα DDA παρακολουθούν διάφορες μετρήσεις, όπως η υγεία των παικτών, τα ποσοστά επιτυχίας και ο χρόνος που απαιτείται για την ολοκλήρωση των εργασιών, για να προσαρμόσουν δυναμικά τη δυσκολία του παιχνιδιού. Αυτός ο βρόχος ανάδρασης σε πραγματικό χρόνο βοηθά στη διατήρηση μιας ελκυστικής και ισορροπημένης εμπειρίας παιχνιδιού. Η δυναμική προσαρμογή δυσκολίας (Dynamic Difficulty Adjustment - DDA) βοηθά στη διατήρηση του ενδιαφέροντος των παικτών, προσαρμόζοντας την πρόκληση του παιχνιδιού με βάση την απόδοσή τους. Αυτό μπορεί να περιλαμβάνει την τροποποίηση του ρυθμού εμφάνισης των εχθρών, της υγείας τους ή των μοτίβων επίθεσης σε πραγματικό χρόνο[13].

Συμπέρασμα

Συμπερασματικά, η ανάπτυξη ενός 2D action platformer με τη χρήση του Unreal Engine 5 και με την αξιοποίηση της C++ είναι μια στρατηγική επιλογή που ευθυγραμμίζεται τόσο με τις βραχυπρόθεσμες ανάγκες του έργου όσο και με τη μακροπρόθεσμη ανάπτυξη και εξέλιξή του. Ο συνδυασμός προσφέρει μια ισχυρή εργαλειοθήκη για τη δημιουργία παιχνιδιών υψηλής ποιότητας, αποτελεσματικότητας και με πολλές δυνατότητες αναβάθμισης. Η ευελιξία στη χρήση τόσο της C++ όσο και των Blueprintsτης UE5, η υποστήριξη για γραφικά υψηλής ποιότητας και η δυνατότητα ανάπτυξης για πολλαπλές πλατφόρμες είναι μερικά μόνο από τα πολλά πλεονεκτήματα που κάνουν τα UE5 και C++ την ιδανική επιλογή για την ανάπτυξη παιχνιδιών. Αυτή η προσέγγιση όχι μόνο διασφαλίζει μια ομαλή και γρήγορη πορεία ανάπτυξης, αλλά επίσης δίνει την δυνατότητα δημιουργίας ενός ετοιμοπαράδοτου παιχνιδιού στην αγορά εργασίας.

12Παραπομπή: Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. ACM SIGGRAPH Computer Graphics.

13Παραπομπή: Adams, E., & Rollings, A. (2006). *Fundamentals of Game Design*. Prentice Hall.

Ενότητα 3η

3. Μεθοδολογία δημιουργίας 2d action platformer

Η δημιουργία ενός 2D action platformer περιλαμβάνει έναν συνδυασμό δημιουργικού σχεδιασμού, δεξιοτήτων προγραμματισμού και κατανόησης των εργαλείων ανάπτυξης παιχνιδιών. Το Unreal Engine 5, σε συνδυασμό με τη δύναμη της C++, παρέχει ένα ισχυρό πλαίσιο για την ανάπτυξη τέτοιων παιχνιδιών. Στις παρακάτω υποενότητες που θα ακολουθήσουν θα αναφερθεί βήμα βήμα η μεθοδολογία που ακολουθήθηκε για την δημιουργία του παιχνιδιού 2d action platformer με την ενσωμάτωση λειτουργιών τεχνητής νοημοσύνης.

3.1. Σύλληψη και σχεδιασμός παιχνιδιού

Το αρχικό στάδιο στη δημιουργία κάθε παιχνιδιού φυσικά είναι η σύλληψη του παιχνιδιού στο μυαλό του σχεδιαστή και η αποτύπωση του γενικού πλάνου ή αλλιώς της γενικής κατεύθυνσης που θέλει να ακολουθήσει ο σχεδιαστής στο μυαλό του, προφορικά ή γραπτά. Η φάση ανάπτυξης της ιδέας είναι κρίσιμη καθώς θέτει τα θεμέλια για ολόκληρο το έργο. Περιλαμβάνει δημιουργική σκέψη, εμπειρία πάνω στον κόσμο των βιντεοπαιχνιδιών, κατανόηση της ζήτησης της αφοράς του gaming και καθορισμό των βασικών μηχανισμών και χαρακτηριστικών που θα κάνουν το παιχνίδι μοναδικό και συναρπαστικό. Σε αυτή την περίπτωση ο σκοπός της διπλωματικής είναι η ανάδειξη λειτουργιών τεχνητής νοημοσύνης, έξυπνης αλληλεπίδρασης με το περιβάλλον και προσαρμοστικής δυσκολίας. Έτσι δίνοντας έμφαση σε αυτές τις πτυχές δημιουργήθηκε ένα 2d action platformer παιχνίδι, όπου ο ήρωας πολεμάει και αλληλεπιδρά με εχθρικές NPC μονάδες AI, αλλά και με το άμεσο περιβάλλον του. Στο περιβάλλον υπάρχουν παγίδες αλλά και σκάλες και πλατφόρμες τις οποίες μπορεί να αξιοποιήσει ο παίκτης για να επιτύχει τους στόχους του. Το παιχνίδι αποτελείτε από 3 διαφορετικά επίπεδα δυσκολίας, στα οποία οι εχθροί γίνονται προοδευτικά δυνατότεροι.

3.2 Εργαλεία Ανάπτυξης και Περιβάλλον

3.2.1 Επισκόπηση Unreal Engine 5

Το επόμενο στάδιο είναι η επιλογή του προγραμματιστικού περιβάλλοντος πάνω στο οποίο θα πραγματοποιηθεί η υλοποίηση του παιχνιδιού. Όπως έχει αναφερθεί και στην προηγούμενη ενότητα για την εκπόνηση της διπλωματικής αυτής επιλέχθηκε ως εργαλείο το UE5 , καθώς προσφέρει μία πληθώρα συστημάτων που βοηθούν πάρα πολύ τους σχεδιαστές, παρέχοντας πολλά δωρεάν εργαλεία δημιουργίας γραφικών, ήχου και φωτισμού. Το Unreal Engine 5 προσφέρει υπερσύγχρονα εργαλεία και δυνατότητες που βελτιώνουν σημαντικά τη διαδικασία ανάπτυξης του παιχνιδιού. Οι νέες τεχνολογίες Nanite και Lumen επιτρέπουν πρωτοφανή επίπεδα λεπτομέρειας και ρεαλισμού φωτισμού, καθιστώντας το ιδανική επιλογή τόσο για παιχνίδια 3D όσο και για 2D. Ένα ακόμα ατού αυτής την μηχανής είναι η χρήση της C++ ως γλώσσας αναφοράς που ενδείκνυται για τον σχεδιασμό παιχνιδιών, όπως επίσης έχει αναφερθεί στην ενότητα 2. Η εξοικείωση με το προγραμματιστικό περιβάλλον ήταν ένα από τα πιο δοκιμαστικά κομμάτια της εργασίας και απαίτησε μεγάλο χρονικό διάστημα για να πραγματοποιηθεί. Η εξοικείωση έγινε μέσο των συνεχών δοκιμών που πραγματοποιήθηκαν πάνω στο περιβάλλον του UE5, ώστε να αποκτηθεί ένα καλό επίπεδο γνώσης χειρισμού του editor, των blueprints και όλων των εργαλείων του UE5. Ήταν απαραίτητη η εκμάθηση βασικών γνώσεων σχεδίασης γραφικών υπολογιστή αλλά και η εκμάθηση όλων των κλάσεων blueprints και των αντικειμένων τους που είναι ζωτικής σημασίας για την υλοποίηση των συμπεριφορών του παίχτη.

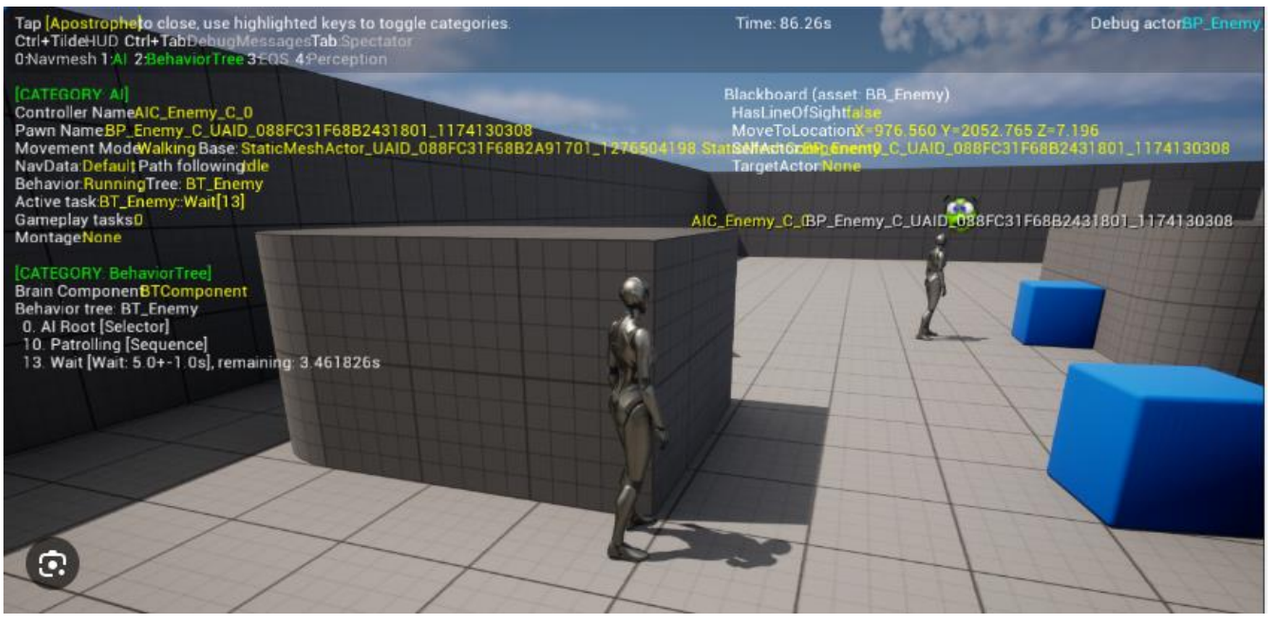
3.2.2 Ολοκληρωμένο Αναπτυξιακό Περιβάλλον (IDE)

Το Visual Studio χρησιμοποιήθηκε ως ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για προγραμματισμό και εντοπισμό σφαλμάτων. Τα χαρακτηριστικά του και η εύκολη ενσωμάτωσή του με το UE5 το καθιστούν μια προτιμώμενη επιλογή για τους προγραμματιστές παιχνιδιών. Το Visual Studio παρέχει ολοκληρωμένα εργαλεία για προγραμματισμό, εντοπισμό σφαλμάτων και έλεγχο έκδοσης, τα οποία είναι απαραίτητα για μια βελτιωμένη διαδικασία ανάπτυξης. Η ενσωμάτωσή του με την Unreal Engine διευκολύνει την αποτελεσματική ανάπτυξη παιχνιδιών.

3.3 Διαδικασία ανάπτυξης παιχνιδιών

3.3.1 Πρωτοτυποποίηση

Η διαδικασία ανάπτυξης ξεκίνησε με τη δημιουργία ενός βασικού πρωτοτύπου για τη δοκιμή των βασικών μηχανισμών του παιχνιδιού. Αυτό περιλάμβανε βασική κίνηση του παίκτη, άλματα και αλληλεπίδραση με το περιβάλλον. Φυσικά το περιβάλλον που χρησιμοποιήθηκε για τις δοκιμές δεν ήταν το πραγματικό επίπεδο του παιχνιδιού, καθώς αυτό δεν είχε υλοποιηθεί ακόμα, άλλα ένα δοκιμαστικό που παρέχεται αυτόματα από το UE5. Επίσης ούτε ο ίδιος ο παίκτης είχε μοντελοποιηθεί και χρησιμοποιήθηκε ένας ήδη υπάρχων χαρακτήρας της μηχανής. Η δημιουργία πρωτοτύπων είναι μια επαναληπτική διαδικασία που επιτρέπει στους προγραμματιστές να δοκιμάσουν και να βελτιώσουν τη μηχανική του παιχνιδιού νωρίς στον κύκλο ανάπτυξης. Βοηθά στον εντοπισμό πιθανών προβλημάτων και στη βελτίωση του παιχνιδιού πριν ξεκινήσει η ανάπτυξη πλήρους κλίμακας.



Εικόνα 4: Επίπεδο debugging και testing στο UE5

3.3.2 Σχεδιασμός χαρακτήρων(Sprites) και κινησιολογίας (flipbooks)

Τα sprites, όπως ονομάζονται οι εικόνες των χαρακτήρων δημιουργήθηκαν χρησιμοποιώντας το Adobe Photoshop και οι κινησιολογία τους υλοποιήθηκαν χρησιμοποιώντας τα εργαλεία κίνησης του UE5. Ο στόχος ήταν η δημιουργία ρευστών και ανταποκρινόμενων κινήσεων χαρακτήρων. Το Photoshop είναι ένα ισχυρό εργαλείο για τη δημιουργία γραφικών στοιχείων παιχνιδιών, ιδιαίτερα για 2D sprites. Επιτρέπει στους σχεδιαστές να σχεδιάζουν γραφικά με πολλές λεπτομέρειες και να προσδίδουν κινήσεις που ζωντανεύουν τους χαρακτήρες. Τα εργαλεία κινούμενων εικόνων του UE5 παρέχουν ένα ισχυρό πλαίσιο για τη δημιουργία πολύπλοκων κινούμενων

εικόνων χαρακτήρων. Προσφέρουν μια σειρά από λειτουργίες για ανάμειξη, δημιουργία αλληλουχίας και βελτίωσης κινούμενων εικόνων για την επίτευξη ομαλών και ρεαλιστικών κινήσεων.

3.3.3 Σχεδίαση επιπέδων

Τα επίπεδα σχεδιάστηκαν χρησιμοποιώντας τον επεξεργαστή επιπέδου του UE5, εστιάζοντας στη δημιουργία απαιτητικών και συναρπαστικών επιπέδων πλατφόρμας. Ο σχεδιασμός περιλάμβανε διάφορα εμπόδια, πλατφόρμες και τοποθετήσεις εχθρών. Η σχεδίαση του επιπέδου απαιτούσε αρκετή διαχείριση όλων των γραφικών στοιχείων, όπως η σωστή δημιουργία των διαστάσεων, η τοποθέτησή τους στον χώρο και η αποτύπωση της υλικής υπόστασης(collision) πάνω σε αυτά. Ο σχεδιασμός του επιπέδου είναι μια κρίσιμη πτυχή της ανάπτυξης παιχνιδιών που συνδυάζει την τέχνη, την αρχιτεκτονική και το παιχνίδι. Περιλαμβάνει τη δημιουργία χώρων που δεν είναι μόνο οπτικά ελκυστικοί αλλά παρέχουν επίσης μια διασκεδαστική και προκλητική εμπειρία για τους παίκτες. Η Διαδικαστική Δημιουργία Περιεχομένου (PCG) είναι μια σημαντική μέθοδος που χρησιμοποιείται για το σχεδιασμό επιπέδων για 2D platformers. Περιλαμβάνει τη δημιουργία περιεχομένου παιχνιδιού αλγοριθμικά με ελάχιστη ανθρώπινη παρέμβαση. Το PCG είναι ιδιαίτερα χρήσιμο για τη δημιουργία διαφορετικών και μοναδικών επιπέδων, τη μείωση του χρόνου ανάπτυξης και τη βελτίωση της δυνατότητας αναπαραγωγής. Η πρόκληση έγκειται στη δημιουργία αλγορίθμων που παράγουν επίπεδα που όχι μόνο μπορούν να αξιοποιηθούν από τους παίχτες αλλά και συναρπαστικά και ποικίλα. Μια προσέγγιση για αποτελεσματικό PCG σε 2D platformers είναι η ενσωμάτωση του ρυθμού (rhythm) και της επανάληψης, διασφαλίζοντας ότι τα παραγόμενα επίπεδα διατηρούν μια ισορροπία μεταξύ προβλεψιμότητας και καινοτομίας για να διατηρηθεί το παιχνίδι ενδιαφέρον. Ο ρυθμός στην σχεδίαση επιπέδων ενός παιχνιδιού αναφέρετε στον χρόνο και την απόσταση που πρέπει να διανύσει ένας παίκτης εντός παιχνιδιού για να συναντήσει την επόμενη ανταμοιβή ή δυσκολία. Ένας καλός ρυθμός σε ένα 2D action platforrmer ορίζετε αυτός που προσδίδει μια ομαλότητα στον ρυθμό που ένας παίκτης αντιμετωπίζει δυσκολίες στην πίστα παιχνιδιού και στα μεταξύ διαλείμματα που μπορεί να επαναπαυτεί και να εισπράξει ανταμοιβές ή χρόνο ξεκούρασης. Αυτό μειώνει τον εκνευρισμό του παίχτη και δεν τον κουράζει νοητικά με αποτέλεσμα το παιχνίδι να γίνεται πιο απολαυστικό και λιγότερο εκνευριστικό. Έρευνες δείχνουν πως οι σχεδιάσεις παιχνιδιών που είναι βασισμένες στον ρυθμό του παιχνιδιού, που αναλύουν τις κινήσεις το παίχτη και ενσωματώνουν ρυθμικά μοτίβα βελτιώνουν την εμπειρία παιχνιδιού. Στα 2D platformers, η καθαρή οπτική σχεδίαση είναι ζωτικής σημασίας. Τα οπτικά στοιχεία θα πρέπει να καθοδηγούν τον παίκτη διαισθητικά στο επίπεδο. Οι πλατφόρμες, οι κίνδυνοι και τα συλλεκτικά αντικείμενα πρέπει να διακρίνονται εύκολα. Τα οπτικά εφέ μπορούν επίσης να ενσωματωθούν για να βελτιώσουν το παιχνίδι παρέχοντας σχόλια και βελτιώνοντας τη συνολική αισθητική εμφάνιση. Για παράδειγμα, διαφορετικά χρώματα και σχήματα για διαφορετικούς τύπους πλατφορμών και εχθρών μπορούν να βοηθήσουν τους παίκτες να κατανοήσουν γρήγορα το περιβάλλον του παιχνιδιού και να αντιδράσουν ανάλογα. Ένα καλά σχεδιασμένο επίπεδο θεωρείτε αυτό που εμφανίζει μια προοδευτική δυσκολία, που εξομαλύνει την περίοδο εκμάθησης του παίχτη, όπως έγινε σε αυτή την εργασία. Τα πρώτα επίπεδα θα πρέπει να μαθαίνουν στον παίχτη τους βασικούς μηχανισμούς του παιχνιδιού και σταδιακά να ενσωματώνουν πιο σύνθετες προκλήσεις, καθώς ο παίκτης προχωρά. Αυτή η προσέγγιση διασφαλίζει ότι οι παίκτες δεν θα καταπονηθούν αρχικά και θα έχουν την ευκαιρία να αναπτύξουν τις δεξιότητές τους με προοδευτικό τρόπο. Η εξισορρόπηση της δυσκολίας του παιχνιδιού περιλαμβάνει επίσης, τη στρατηγική τοποθέτηση ειδικών σημείων επαναφοράς παιχνιδιού (checkpoints) για τη μείωση της απογοήτευσης των παικτών από επαναλαμβανόμενες αποτυχίες.

3.4 Ενσωμάτωση της τεχνητής νοημοσύνης

3.4.1Finite State Machines (FSM)

Τα FSM χρησιμοποιήθηκαν για τον έλεγχο της συμπεριφοράς των εχθρών. Κάθε εχθρός έχει ορισμένες συμπεριφορές, όπως την αδράνεια, περιπολία, καταδίωξη και επίθεση, με μεταβάσεις μεταξύ αυτών που βασίζονται στις ενέργειες των παικτών και στα περιβαλλοντικά ερεθίσματα. Οι μηχανές πεπερασμένης

κατάστασης είναι αποτελεσματικές για την εφαρμογή προβλέψιμων και διαχειρίσιμων συμπεριφορών τεχνητής νοημοσύνης. Ορίζοντας καταστάσεις και μεταβάσεις, οι προγραμματιστές μπορούν να δημιουργήσουν δυναμικά NPC, που έχουν την ικανότητα της ανταπόκρισης αναλόγως των κινήσεων του παίκτη. Επιπλέον τα FSM παρέχουν μια σαφή δομή για την ανάπτυξη AI παιχνιδιών, επιτρέποντας την εύκολη διόρθωση σφαλμάτων και τροποποίηση. Κάθε κατάσταση αντιπροσωπεύει μια ξεχωριστή συμπεριφορά, απλοποιώντας τη διαδικασία ανάπτυξης.

3.4.2 Τεχνικές εύρεσης μονοπατιού

Οι τεχνικές εύρεσης μονοπατιών, εφαρμόστηκαν για να επιτρέψουν στους εχθρούς να περιηγηθούν στο περιβάλλον και να κυνηγήσουν τον παίκτη, συγκεκριμένα εφαρμόστηκε μία ray-tracing τεχνική. Το raytracing ενσωματώνει τεχνικές ανίχνευσης ακτινών στη διαδικασία εύρεσης μονοπατιών για να βελτιώσει την αποτελεσματικότητα και τον ρεαλισμό της πλοήγησης. Η βασική ιδέα είναι η χρήση ακτινών για την ανίχνευση εμποδίων και τον υπολογισμό των μονοπατιών σε πραγματικό χρόνο, επιτρέποντας τη δυναμική και ανταποκρινόμενη εύρεση μονοπατιών. Οι ακτίνες εκπέμπονται από την κινούμενη οντότητα (π.χ. έναν παίκτη ή έναν χαρακτήρα τεχνητής νοημοσύνης) για τον εντοπισμό εμποδίων και τον προσδιορισμό του ελεύθερου χώρου στο περιβάλλον. Ανιχνεύοντας ακτίνες και ανιχνεύοντας συγκρούσεις, ο αλγόριθμος μπορεί να χαρτογραφήσει δυναμικά την περιοχή πλοήγησης και να ενημερώσει τις διαδρομές ανάλογα. Αυτή η προσέγγιση επιτρέπει στις οντότητες να προσαρμόζονται σε αλλαγές στο περιβάλλον σε πραγματικό χρόνο, βελτιώνοντας τις δυνατότητες πλοήγησής τους. Επίσης στην εργασία αυτή χρησιμοποιήθηκε και ανίχνευση σύγκρουσης με την εκτέλεση δοκιμών τομής μεταξύ των ακτινών και των αντικειμένων στο περιβάλλον. Γίνετε καταγραφή των σημείων χτυπήματος όπου οι ακτίνες τέμνονται με αντικείμενα. Αυτά τα σημεία υποδεικνύουν τα όρια των προσβάσιμων.

3.4.3 Δένδρα αποφάσεων

Τα δέντρα συμπεριφοράς είναι ένα μοντέλο που χρησιμοποιείται για τον έλεγχο της διαδικασίας λήψης αποφάσεων στο AI. Προέρχονται από τη ρομποτική και έχουν υιοθετηθεί ευρέως στη βιομηχανία παιχνιδιών για την αποτελεσματικότητά τους στη διαχείριση σύνθετων συμπεριφορών. Τα δένδρα αποφάσεων χρησιμοποιήθηκαν στη εργασία για τον προσδιορισμό των κινήσεων που υλοποιεί το AI. Τα δέντρα συμπεριφοράς προσφέρουν ένα επεκτάσιμο και ευέλικτο πλαίσιο για τον καθορισμό πολύπλοκων συμπεριφορών τεχνητής νοημοσύνης. Επιτρέπουν στους προγραμματιστές να δημιουργούν διακλαδιζόμενες και επαναχρησιμοποιήσιμες ενέργειες, ενισχύοντας την προσαρμοστικότητα και την ανταπόκριση του AI. Το UE5 διαθέτει ένα έτοιμο πρόγραμμα διαχείρισης δένδρων αποφάσεων και υποστηριζόμενες κλάσης με προ-προγραμματισμένες μεθόδους και αντικείμενα. Έτσι τα δένδρα αποφάσεων μπορούν να διαχειριστούν διάφορες καταστάσεις και μεταβάσεις μεταξύ αυτών. Για παράδειγμα, η κίνηση ενός χαρακτήρα μπορεί να σχεδιαστεί ως η μετάβαση από την ακινησία στην κίνηση με το που αλλάζει η τιμή της επιτάχυνσης και τις ταχύτητας σε έναν άξονα. Επιπλέον αναλόγως με τον άξονα κίνησης μπορεί να μεταβληθεί η κίνηση του παίχτη σε εναέρια πτήση ή άλμα, ενώ με την επίδραση της βαρύτητας ο χαρακτήρας να γειωθεί ξανά ή να κάνει κάποιο γλίστρημα. Όλες αυτές οι μεταβάσεις πραγματοποιούνται από τα δένδρα αποφάσεων.

3.4.4 Αποφυγή εμποδίων

Το UE5 παρέχει διάφορα εργαλεία για την υλοποίηση της αποφυγής εμποδίων. Το σύστημα χρησιμοποιεί πλέγματα πλοήγησης (navmeshes) για την αναπαράσταση περιοχών που μπορούν να προσπελαστούν σε ένα επίπεδο. Η ανίχνευση σύγκρουσης στο UE5 περιλαμβάνει τη χρήση προκαθορισμένων σχημάτων σύγκρουσης (π.χ. κουτιά, σφαίρες, κάψουλες), που μπορούν να βρεθούν στο μενού με το γραφικό περιβάλλον του UE5, για την αναπαράσταση των φυσικών ορίων των αντικειμένων. Αυτά τα σχήματα χρησιμοποιούνται για την ανίχνευση συγκρούσεων (collisions) και την ενεργοποίηση συγκεκριμένων γεγονότων όταν συμβαίνουν αυτές.

Αποτελεσματικοί αλγόριθμοι ανίχνευσης σύγκρουσης διασφαλίζουν ότι αυτοί οι έλεγχοι εκτελούνται γρήγορα και με ακρίβεια, αποτρέποντας την επιβράδυνση του παιχνιδιού. Στην εργασία χρησιμοποιήθηκαν τεχνικές αποφυγής εμποδίων, όπως το raycasting και αλγόριθμοι συμπεριφορών διεύθυνσης (steering) για να διασφαλιστεί η ομαλή και ρεαλιστική πλοήγηση του εχθρού. Οι συμπεριφορές Raycasting και steering είναι απαραίτητες για τη δυναμική αποφυγή εμποδίων, επιτρέποντας στους χαρακτήρες AI να περιηγούνται στα εμπόδια με ευκολία και φυσικότητα και να διατηρούν ρεαλιστικά μοτίβα κίνησης. Η αποτελεσματική αποφυγή εμποδίων ενσωματώνει την ανίχνευση σύγκρουσης με προγνωστικούς αλγόριθμους διεύθυνσης, επιτρέποντας στους χαρακτήρες AI να προβλέπουν και να ανταποκρίνονται σε εμπόδια σε πραγματικό χρόνο.

3.4.5 Σταδιακά αυξανόμενη δυσκολία επιπέδων και εχθρών

Η συνεχώς μεταβαλλόμενη δυσκολία των εχθρών ήταν ένα πολύ σημαντικό στάδιο στην ανάπτυξη του παιχνιδιού. Η αλλαγή στον βαθμό της δυσκολίας είναι απαραίτητη για την διασφάλιση της διασκέδασης του παίχτη και την καταπολέμηση της επανάληψης και της ρουτίνας εντός του παιχνιδιού. Το παιχνίδι θα πρέπει αρχικά να δίνει τον απαραίτητο χρόνο στον παίχτη, ώστε αυτός να προσαρμοστεί στον τρόπο και τον χειρισμό του χαρακτήρα του. Αφού ο παίχτης εξοικειωθεί με τους ελέγχους του παιχνιδιού, τότε με τη χρήση διαφόρων παραμέτρων και δεδομένων εντοπίζεται το επίπεδο ικανότητας του παίχτη, ώστε το παιχνίδι να μην είναι ούτε πολύ εύκολο και βαρετό , αλλά ούτε και υπερβολικά απαιτητικό, πέραν των δυνατοτήτων του. Οι τεχνικές δυναμικής αυξομείωσης της δυσκολίας παιχνιδιού ονομάζονται Dynamic Difficulty Adjustment (DDA). Η αυξομείωση της δυσκολίας επιτυγχάνεται μέσω τις ρύθμισης διαφόρων χαρακτηριστικών των εχθρών, όπως η ζωή τους, η ζημιά που προκαλούν στον παίχτη, η ταχύτητά τους και άλλες παράμετροι του παιχνιδιού ,όπως ο ρυθμός αναγέννησης των εχθρών και η επιθετικότητά τους. Συμπερασματικά η αποτελεσματική αυξομείωση της δυσκολίας των εχθρών γίνεται με τη συλλογή δεδομένων του παίχτη, με βάση την απόδοσή του. Με την χρήση εργαλείων, όπως των δένδρων αποφάσεων του UE5 και των στιγμιότυπων (blueprints) οι προγραμματιστές μπορούν να δημιουργήσουν προσαρμοστικά συστήματα AI και χαρακτήρες NPC που εξασφαλίζουν ένα ισορροπημένο, αλλά και ανταγωνιστικό τρόπο παιχνιδιού. Αυτό μπορεί να γίνει εφικτό με τις συνεχής δοκιμές του παιχνιδιού και πραγματοποιώντας μερικές ανεπαίσθητες αλλαγές σε αυτό για την διατήρηση του επιθυμητού επιπέδου δυσκολίας και για της συνεχής προσοχής των παιχτών, χωρίς την πρόκληση εκνευρισμού.

3.5 Δοκιμές και βελτιστοποίηση

3.5.1 Δοκιμή παιχνιδιού

Αυτό το στάδιο είναι απαραίτητο για τη τελειοποίηση του παιχνιδιού και την διασφάλιση της ποιότητας του. Είναι αδιανόητα για ένα παιχνίδι να κυκλοφορήσει στην αγορά, χωρίς αυτό να έχει δοκιμαστεί για διάφορους παράγοντες, όπως είναι η ομαλή λειτουργία του, για τυχών σφάλματα που μπορεί να προκύψουν κατά το παιχνίδι ή την φόρτωση του παιχνιδιού και για την πιστοποίηση της εμπειρίας χρηστών. Πρέπει αν ελεγχθεί αν όλα λειτουργούν όπως ήταν προβλεπόμενο και δεν υπάρχουν σφάλματα στον κώδικα και στην λογική του παιχνιδιού(bugs and glitches). Σε αυτό το στάδιο, εφόσον το παιχνίδι είναι σε φάση να λειτουργήσει ομαλά, διανέμεται σε εγκεκριμένους παίχτες – play testers οι οποίοι θα αξιολογήσουν το παιχνίδι από την οπτική γωνία του πελάτη gamer και θα τροφοδοτήσουν τους προγραμμάτισες με διάφορες πληροφορίες, όπως για τυχών σφάλματα ή βελτιώσεις που μπορούν να υλοποιηθούν στην λογική του παιχνιδιού ή κάποια νέα λογική που μπορεί να ενσωματωθεί στον κώδικα για την αύξηση της ψυχαγωγίας του παιχνιδιού ή της προσβασιμότητας και άλλα πολλά. Για αυτούς του λόγους το playtesting είναι ένα κρίσιμο στοιχείο της διαδικασίας ανάπτυξης του παιχνιδιού. Επιτρέπει στους προγραμματιστές να συλλέγουν πολύτιμα σχόλια από τους παίκτες, να εντοπίζουν πιθανά προβλήματα και να βελτιώνουν τη λογική και τη σχεδίαση του παιχνιδιού για να βελτιώσουν τη συνολική εμπειρία του παίκτη. Τέλος μέσω του playtesting, οι προγραμματιστές μπορούν να παρατηρήσουν πώς οι παίκτες αλληλοεπιδρούν με το παιχνίδι, να κατανοήσουν τη συμπεριφορά τους και να κάνουν τις απαραίτητες

προσαρμογές. Αυτή η επαναληπτική διαδικασία βοηθά στη ρύθμιση της δυσκολίας, του ρυθμού και της συνολικής ισορροπίας του παιχνιδιού.

3.5.2 Τεχνικές Βελτιστοποίησης

Στη συνέχεια αφού επιτεύχθηκε το παραπάνω στάδιο της ανατροφοδότησης, έγιναν οι απαραίτητες αλλαγές στο παιχνίδι για την βελτίωση της εμπειρίας των παιχτών. Η βελτιστοποίηση είναι ένα ουσιαστικό μέρος της διαδικασίας ανάπτυξης, ιδιαίτερα στην ανάπτυξη παιχνιδιών όπου η απόδοση είναι κρίσιμη. Η αποτελεσματική βελτιστοποίηση περιλαμβάνει τον εντοπισμό και την εξάλειψη των σημείων που μειώνεται η απόδοση, τη βελτιστοποίηση του κώδικα και την αποτελεσματική χρήση των πόρων του συστήματος. Για την επίτευξη βέλτιστης απόδοσης, είναι μεγάλης σημασίας η εφαρμογή αποτελεσματικών αλγορίθμων, η μείωση της χρήσης μνήμης και η βελτιστοποίηση των διαδικασιών απόδοσης. Αυτό διασφαλίζει ότι το παιχνίδι εκτελείται ομαλά για διάφορες κατανομές υλισμικού για διάφορες υπολογιστικές μηχανές. Η βελτιστοποίηση του παιχνιδιού μπορεί να επιτευχθεί σε διάφορα σημεία του παιχνιδιού. Οι τεχνικές βάθους λεπτομερειών (LOD) χρησιμοποιούνται για τη μείωση της πολυπλοκότητας των μοντέλων καθώς απομακρύνονται από την κάμερα, βελτιώνοντας έτσι την απόδοση. Το Frustum culling διασφαλίζει την επεξεργασία και απόδοση μόνο των αντικειμένων εντός της οπτικής της κάμερας, γεγονός που μπορεί να μειώσει σημαντικά τον αριθμό των κλήσεων κώδικα και να βελτιώσει την απόδοση. Η βελτιστοποίηση των shader περιλαμβάνει τη μείωση της πολυπλοκότητας των υπολογισμών shader και τον προυπολογισμό δεδομένων όπου είναι δυνατόν, για τη βελτίωση της απόδοσης. Επιπλέον οι αλγόριθμοι ανίχνευσης σύγκρουσης ευρείας φάσης, όπως η χωρική κατάτμηση και οι ιεραρχίες όγκου οριοθέτησης, συμβάλλουν στην ελαχιστοποίηση του αριθμού των ελέγχων σύγκρουσης που απαιτούνται, βελτιώνοντας έτσι την απόδοση. Το Multithreading μπορεί να βελτιώσει την απόδοση κατανέμοντας εργασίες σε πολλούς πυρήνες, αλλά απαιτεί προσεκτική διαχείριση της κοινής χρήσης δεδομένων και του συγχρονισμού για την αποφυγή συνθηκών αγώνα και αδιέξοδων. Τέλος η βελτιστοποίηση για συγκεκριμένο υλικό υπολογιστή περιλαμβάνει την κατανόηση της μοναδικής αρχιτεκτονικής και των δυνατοτήτων κάθε πλατφόρμας για την καλύτερη χρήση των διαθέσιμων πόρων.

3.6 Λεπτομέρειες εφαρμογής

3.6.1 Scripting και προγραμματισμός

Η λογική του παιχνιδιού και οι συμπεριφορές AI υλοποιήθηκαν χρησιμοποιώντας το σύστημα Blueprint της Unreal Engine και προγραμματισμό με C++. Αυτός ο συνδυασμός παρείχε ευελιξία και έλεγχο στους μηχανισμούς του παιχνιδιού και στις συμπεριφορές AI. Το σύστημα Blueprint της Unreal Engine προσφέρει ένα οπτικό περιβάλλον σκριπτ που επιτρέπει στους προγραμματιστές να δημιουργούν πολύπλοκη λογική παιχνιδιού χωρίς να γράφουν κώδικα. Για πιο προηγμένες λειτουργίες, ο προγραμματισμός σκριπτ σε C++ παρέχει την απαραίτητη ισχύ και ευελιξία. Ο συνδυασμός Blueprints και C++ επιτρέπει στους προγραμματιστές να αξιοποιήσουν την ευκολία του οπτικού σκριπτ διατηρώντας παράλληλα την απόδοση και τον έλεγχο που παρέχει η C++. Αυτή η υβριδική προσέγγιση είναι ιδιαίτερα αποτελεσματική στη διαχείριση πολύπλοκων συμπεριφορών AI και λειτουργιών παιχνιδιών.

3.6.2 Κινησιολογία και αλληλεπίδραση χαρακτήρων με το περιβάλλον

Σε αυτό το βήμα υλοποιήθηκε όλοι η σχετική διεργασία για την δημιουργία του κυρίως χαρακτήρα του παιχνιδιού, αλλά και των NPC. Αυτό έγινε με την βοήθεια ενός εξωτερικού εργαλείου το οποίο είναι μία ενσωμάτωση (plugin) , το paperzd του UE5 και παρέχει τα εργαλεία για τη δημιουργία όλων των κινήσεων με τη δημιουργία

flipbook σε animation source files. Όλα αυτά θα εξηγηθούν με μεγάλη λεπτομέρεια στην επόμενη ενότητα. Έτσι σε αυτό το στάδιο δημιουργούνται τα στιγμιότυπα για κάθε animation των χαρακτήρων και ορίζεται η ταχύτητα αναπαραγωγής τους και άλλες λεπτομέρειες όπως η ανάλυση που θα έχουν ή κάποιο φίλτρο που τυχών μπορεί να χρησιμοποιηθεί. Συμπερασματικά τα εργαλεία κινουμένων σχεδίων της Unreal Engine παρέχουν ένα ολοκληρωμένο σύνολο λειτουργιών για τη δημιουργία και τη διαχείριση κινούμενων εικόνων χαρακτήρων. Αυτά τα εργαλεία επιτρέπουν στους προγραμματιστές να συνδυάζουν, να ακολουθούν και να τελειοποιούν κινούμενα σχέδια για να επιτύχουν ομαλές και ρεαλιστικές κινήσεις. Επιπρόσθετα οι προσομοιώσεις αλληλεπιδράσεων περιβάλλοντος είναι ζωτικής σημασίας για τη δημιουργία ρεαλιστικών αλληλεπιδράσεων στα παιχνίδια. Προσομοιώνοντας με ακρίβεια δυνάμεις, συγκρούσεις και άλλα φυσικά φαινόμενα, οι προγραμματιστές μπορούν να βελτιώσουν τη βύθιση και τον ρεαλισμό του κόσμου του παιχνιδιού.

3.6.3 Ενσωμάτωση ηχητικών εφέ

Η σχεδίαση ήχου παίζει ζωτικό ρόλο στη δημιουργία μιας διαδραστικής εμπειρίας παιχνιδιού. Ενσωματώνοντας ηχητικά εφέ και μουσική υψηλής ποιότητας, οι προγραμματιστές μπορούν να βελτιώσουν σημαντικά τη συναισθηματική επίδραση του παιχνιδιού στους παίκτες και τη συνολική ατμόσφαιρα. Η αποτελεσματική ενσωμάτωση ήχου περιλαμβάνει προσεκτικό συγχρονισμό ηχητικών εφέ με ενέργειες παιχνιδιού, καθώς και τη χρήση ήχων και μουσικής περιβάλλοντος για τη δημιουργία ενός συνεκτικού και καθηλωτικού ηχητικού τοπίου. Με τα εργαλεία που προσφέρει το UE5 μπορεί να γίνει η ενσωμάτωση εξωτερικών αρχείων ήχου στην μηχανή, είτε η αναπαραγωγή τέτοιων ήχων που παρέχονται ήδη από αυτή. Έπειτα γίνεται η επεξεργασία τους και η ενσωμάτωση τους στο παιχνίδι. Αυτοί οι ήχοι μπορούν να προστεθούν είτε μέσο ειδικών scripts στο κομμάτι του κώδικα, είτε να ενσωματωθούν στο animation source file που παρέχεται από το paperzd plugin σε συγκεκριμένα καρέ, όταν γίνεται η αναπαραγωγή των animation.

3.6.4 Σχεδίαση διεπαφών χρηστών – user interface

Το επόμενο στάδιο στην ανάπτυξη ενός παιχνιδιού είναι ο σχεδιασμός της διεπαφής χρήστη (UI), που είναι κρίσιμος στο gaming, καθώς επηρεάζει άμεσα την εμπειρία και την αφοσίωση του παίκτη. Ένα καλά σχεδιασμένο περιβάλλον χρήστη παρέχει σαφή, διαισθητική πλοήγηση, βελτιώνοντας το παιχνίδι και διασφαλίζοντας ότι οι παίκτες μπορούν να έχουν εύκολη πρόσβαση στις λειτουργίες και τις πληροφορίες του παιχνιδιού χωρίς σύγχυση. Μια καλή διεπαφή χρήστη είναι ζωτικής σημασίας για να επιτρέπει στους παίκτες να πλοηγούνται στον κόσμο του παιχνιδιού, να έχουν πρόσβαση στις επιλογές του παιχνιδιού και να κατανοούν τους μηχανισμούς του παιχνιδιού. Θα πρέπει να είναι σαφές, διαισθητικό και προσβάσιμο σε όλους τους παίκτες. Τα στοιχεία διεπαφής χρήστη παρέχουν ουσιαστική ανατροφοδότηση στους παίκτες, ενημερώνοντάς τους για την κατάσταση, την πρόοδό τους και τις ενέργειές τους μέσα στο παιχνίδι. Αυτή η ανατροφοδότηση είναι ζωτικής σημασίας για τη διατήρηση της εμβάπτισης και τη διασφάλιση ότι οι παίκτες αισθάνονται συνδεδεμένοι με τον κόσμο του παιχνιδιού. Μια ελκυστική και αισθητικά ευχάριστη διεπαφή χρήστη μπορεί να αυξήσει σημαντικά τη διατήρηση και την ικανοποίηση των παικτών. Δεν πρόκειται μόνο για τη λειτουργικότητα αλλά και για τη δημιουργία μιας οπτικά ελκυστικής και συνεκτικής εμπειρίας που βελτιώνει τη συνολική σχεδίαση του παιχνιδιού.

3.7 Μετά την ανάπτυξη

3.7.1 Δοκιμή beta – τελικό στάδιο δοκιμής

Μετά από την συνολική ανάπτυξη του παιχνιδιού ακολουθεί το τελευταίο δοκιμαστικό στάδιο, όπου το παιχνίδι στην βελτιστοποιημένη έκδοση του, είτε βγαίνει στην αγορά είτε κατανέμεται σε μία μερίδα τυχαίων ή επιλεγμένων παικτών και την τελευταία δοκιμαστική περίοδο. Εάν βγει στην αγορά, τότε λέμε ότι βρίσκεται ακόμα στο στάδιο Beta, δηλαδή στην πιο πρώιμη έκδοση του προτού πάρει η διαχειρίστρια εταιρία την πρώτη ανατροφοδότηση από αληθινούς παίκτες. Η δοκιμή beta είναι μια κρίσιμη φάση στην ανάπτυξη του παιχνιδιού, παρέχοντας πολύτιμες πληροφορίες για το πώς αλληλοεπιδρούν οι πραγματικοί παίκτες με το παιχνίδι. Αυτά τα σχόλια βοηθούν τους προγραμματιστές να εντοπίσουν και να διορθώσουν σφάλματα, να εξισορροπήσουν το παιχνίδι και να βελτιώσουν τη συνολική εμπειρία χρήστη. Με τη συμμετοχή των παικτών στη φάση δοκιμών beta, οι προγραμματιστές μπορούν να συγκεντρώσουν πλήθος ανατροφοδοτήσεων και δεδομένων, τα οποία είναι απαραίτητα για τη εξέλιξη και τη βελτιστοποίηση του παιχνιδιού πριν από την τελική του κυκλοφορία.

3.7.2 Τελικές τροποποιήσεις και ρύθμιση λεπτομερειών

Αυτό είναι και το τελικό στάδιο της μεθοδολογίας που ακολουθήθηκε για τη δημιουργία του παιχνιδιού. Σε αυτό το στάδιο οι προγραμματιστές μαζεύουν όλα τα δεδομένα που έχουν στην διάθεση τους από την ανατροφοδότηση των παικτών και επιλέγουν ποιες αλλαγές θα ήθελαν και μπορούν να πραγματοποιήσουν πάνω στο παιχνίδι. Αυτό επιτυγχάνει την δύο πράγματα, την εξέλιξη και επαναεπισκεψημότητα του παιχνιδιού, αλλά και την ικανοποίηση της αγοράς. Είναι πολύ σημαντικό οι πελάτες να νιώθουν πως τα αιτήματά τους εισακούονται και πως η εταιρία που έχει τα δικαιώματα του παιχνιδιού ακούει τα παράπονα και τις προτάσεις των πελατών της. Έτσι τα τελικά στάδια ανάπτυξης περιλαμβάνουν προσαρμογές και ρυθμίσεις του παιχνιδιού για να διασφαλιστεί ένα προϊόν υψηλής ποιότητας. Αυτό περιλαμβάνει τη διόρθωση τυχόν εναπομεινάντων σφαλμάτων, τη βελτιστοποίηση της απόδοσης και τη βελτίωση των μηχανισμών του παιχνιδιού. Αυτό περιλαμβάνει τη βελτίωση των οπτικών εφέ, την εξισορρόπηση του παιχνιδιού και τη διασφάλιση ομαλής απόδοσης σε όλες τις πλατφόρμες.

Συμπέρασμα

Η μεθοδολογία για την ανάπτυξη ενός παιχνιδιού πλατφόρμας 2D δράσης με εχθρούς AI χρησιμοποιώντας το Unreal Engine 5 περιλαμβάνει μια ολοκληρωμένη και επαναληπτική διαδικασία, συμπεριλαμβανομένου του σχεδιασμού, του προγραμματισμού, της ανάπτυξης, της δοκιμής και της βελτιστοποίησης. Αξιοποιώντας τα ισχυρά χαρακτηριστικά του UE5 και χρησιμοποιώντας ισχυρές τεχνικές AI, έγινε δυνατή η δημιουργία μιας συναρπαστικής και ανταγωνιστικής εμπειρίας παιχνιδιού. Ο λεπτομερής σχεδιασμός και οι αυστηρές δοκιμές εξασφάλισαν ένα ακονισμένο τελικό προϊόν που ανταποκρίνεται στις προσδοκίες των παικτών.

Ενότητα 4η

4. Σχεδιασμός και Παρουσίαση της εφαρμογής

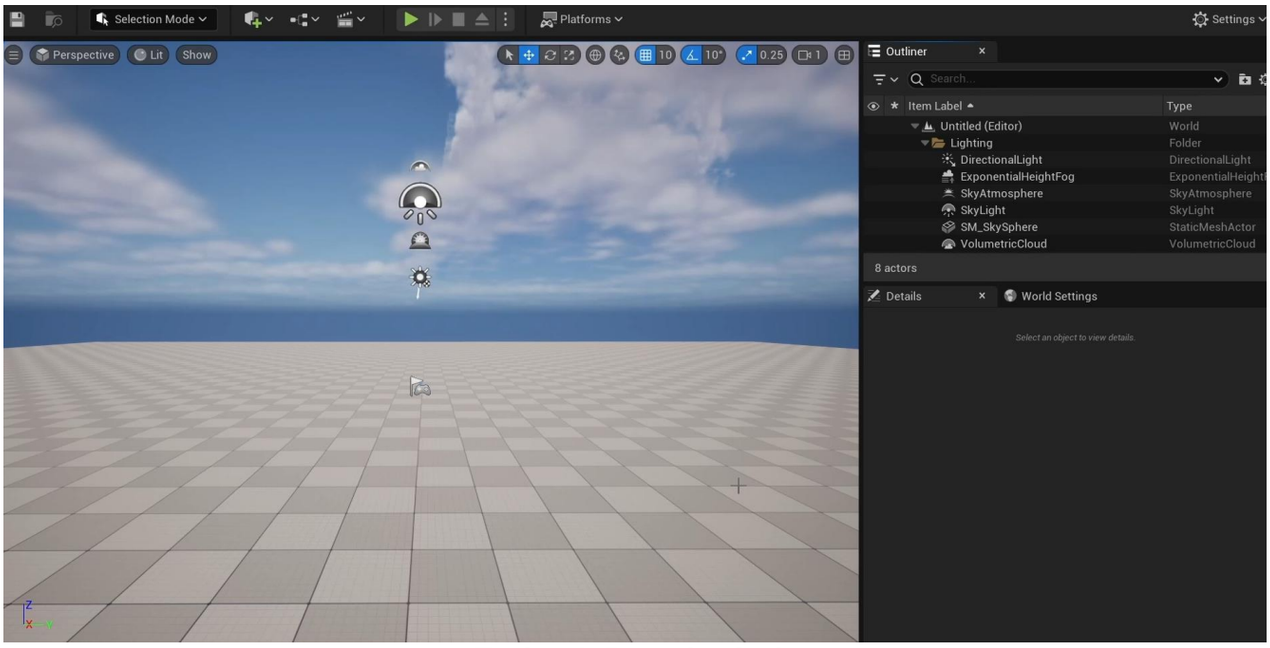
4.1.1 Αρχικό στάδιο προετοιμασίας, ρυθμίσεις και εργαλεία που χρησιμοποιήθηκαν

Το πρώτο στάδιο για την εκπόνηση της εργασίας ήταν η εγκατάσταση της μηχανής ανάπτυξης παιχνιδιών UE5 που προσφέρετε δωρεάν από την εταιρία epic games. Αφού πραγματοποιήθηκε η εγκατάσταση της εφαρμογής epic games από εκεί έγινε και η εγκατάσταση της μηχανής UE5. Επίσης υποχρεωτικό για την εγκατάσταση ήταν η δημιουργία ενός λογαριασμού epic games ή η σύνδεση της εφαρμογής με έναν ηλεκτρονικό λογαριασμό facebook, google, xbox games, steam και άλλες πλατφόρμες δικτύωσης. Η επιλογή της έκδοσης 5.3.2 πιο συγκεκριμένα ήταν η έκδοση που επιλέχθηκε λόγο προηγούμενης εμπειρίας με αυτή. Πιο αναλυτικά ο πρώτος αριθμός της έκδοσης συμβολίζει την κύρια έκδοση της μηχανής όπου εκεί παρατηρούνται και οι μεγαλύτερες διαφοροποιήσεις μεταξύ τους. Όσο μεγαλύτερος είναι ο πρώτος αριθμός τόσο πιο σύγχρονη είναι η έκδοση της μηχανής και πιο προηγμένες οι λειτουργίες της. Επίσης συνήθως όσο πιο καινούργια είναι η έκδοση συνήθως διορθώνονται και πιο πολλά σφάλματα (bugs) που είχαν οι παλαιότερες εκδόσεις. Ο δεύτερος αριθμός της έκδοσης είναι η υποέκδοση. Ενώ παρατηρούνται μεγάλες διαφορές μεταξύ τον κανονικών εκδόσεων στις υποεκδόσεις διατηρείτε η κυρίως γραμμή που ακολουθεί η βασική έκδοση και δεν αλλάζει η λειτουργικότητα της μηχανής. Τέλος ο τελευταίος αριθμός συμβολίζει πολύ μικρές διορθώσεις πάνω στην υποέκδοση. Για αυτό το λόγο επιλέχθηκε η έκδοση 5, ενώ η έκδοση 6 που μόλις έκανε την εμφάνιση της στην αγορά ήταν ακόμα σε πολύ πρώιμο στάδιο εξέλιξης και για αυτό δεν επιλέχθηκε, για την αποφυγή τυχών σφαλμάτων και τον χρόνο επανεκπαίδευσης που θα απαιτούσε ο χειρισμός της.

Εικόνα 5: Εγκατάσταση Unreal Engine στο epic games launcher

Το αμέσως επόμενο βήμα μετά την εγκατάσταση της μηχανής ήταν η δημιουργία του πρότζεκτ της διπλωματικής, δηλαδή η δημιουργία ενός πρότζεκτ που επιτρέπει την ανάπτυξη ενός 2d action platformer game. Οπότε αρχικά πραγματοποιείτε η εκκίνηση της μηχανής UE5 και το μενού επιλογής πρότζεκτ επιλέχθηκε η καρτέλα με αναγραφή παιχνίδι. Έπειτα έγινε η επιλογή του τύπου παιχνιδιού που θέλουμε να φτιάξουμε. Το UE5 παρέχει πολλά έτοιμα καλούπια (templates) που δίνουν αμέσως προβάδισα στον προγραμματιστή επιλέγοντας το template που απεικονίζει τον τύπο παιχνιδιού που επιθυμεί να δημιουργήσει. Δυστυχώς δεν υποστηρίζεται κάποιο template για δημιουργία 2d παιχνιδιών, οπότε σε αυτή την φάση επιλέχθηκε το blank template, δηλαδή το κενό template. Στη συνέχεια δόθηκε ένα όνομα στο πρότζεκτ το οποίο αρχικοποιήθηκε και αποθηκεύτηκε στην προκαθορισμένη τοποθεσία, σε έναν φάκελο που αποθηκεύονται όλα τα πρότζεκτ του UE5. Αφού δημιουργήθηκε το πρότζεκτ το πρόγραμμα επεξεργασίας που ονομάζεται editor εμφανίζει ένα μεγάλο 3d ανάγλυφο χάρτη (map) ή αλλιώς επίπεδο που είναι άχρηστο για ένα 2d game. Έτσι εφόσον ένα 2d game χρειάζεται επίπεδο έδαφος η πρώτη πράξη που πραγματοποιήθηκε είναι από το μενού επιλογών η επιλογή ενός νέου βασικού επιπέδου για τα αρχικά στάδια της δημιουργίας του παιχνιδιού που αργότερα θα αντικατασταθεί από το κανονικό επίπεδο παιχνιδιού που θα δημιουργηθεί σε αυτή την εργασία. Το αρχικό επίπεδο αυτό παρέχεται αυτόματα από την

μηχανή και δεν δημιουργήθηκε από τον προγραμματιστή. Στην εικόνα 6 φαίνεται πως είναι το επίπεδο αυτό. Στη συνέχεια δημιουργήθηκε ένας νέος φάκελος εντός των αρχείων του πρότζεκτ με το όνομα χάρτες. Στη συνέχεια έγινε η αποθήκευση του βασικού επιπέδου μέσα σε αυτό τον φάκελο με το όνομα debug. Έπειτα στις ρυθμίσεις του πρότζεκτ στην κατηγορία maps and modes ορίστηκε αυτό το επίπεδο ως το βασικό επίπεδο εκκίνησης του editor και του πρότζεκτ.

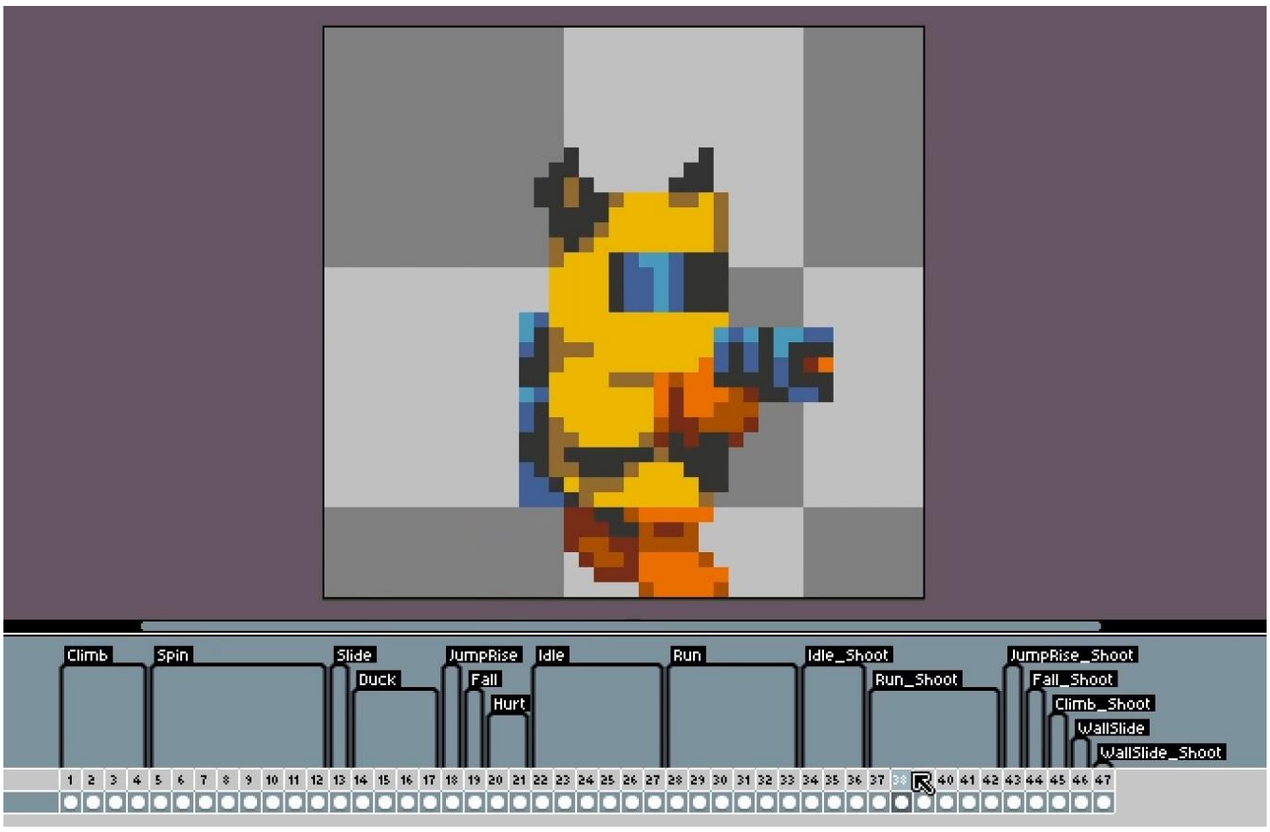


Εικόνα 6: Βασικό επίπεδο

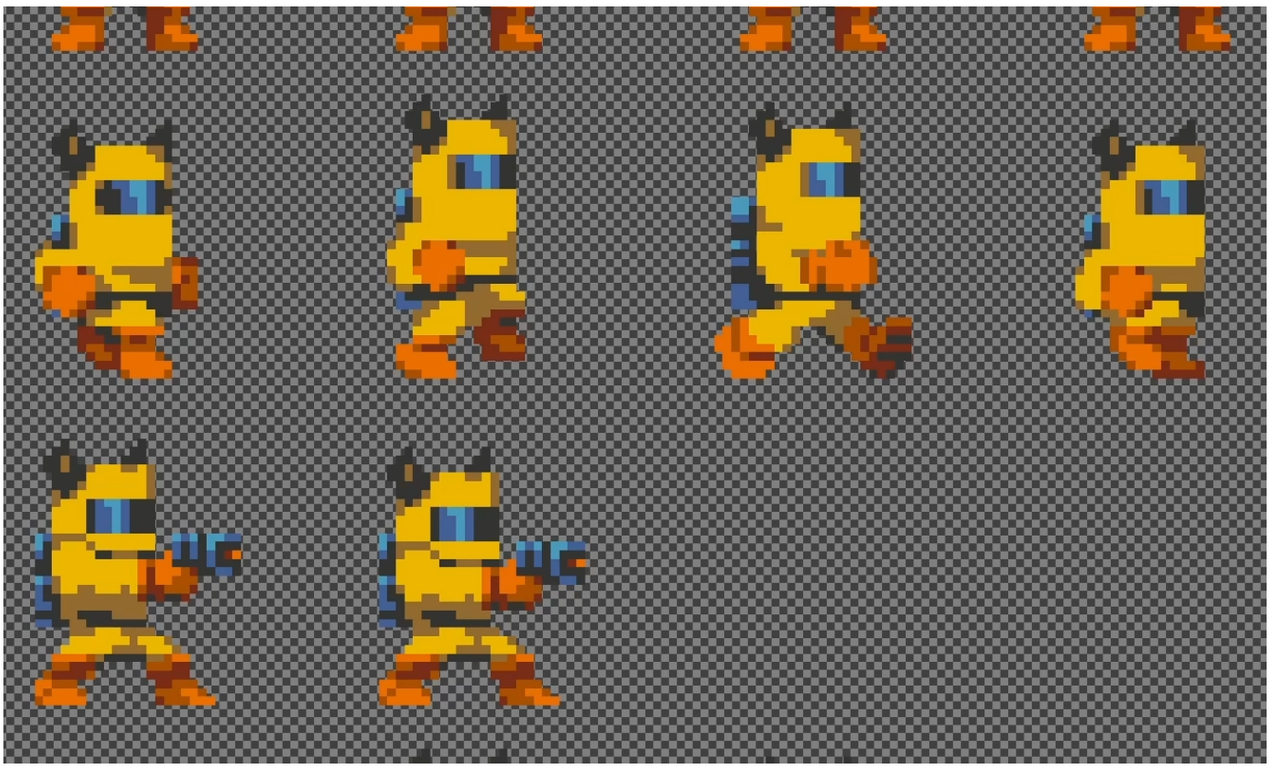
Ένα ακόμα σημαντικό βήμα για την αρχικοποίηση του πρότζεκτ είναι η επιλογή συγκεκριμένων ρυθμίσεων που θα χρειαστούν για το δισδιάστατο game, καθώς οι προεπιλεγμένες ρυθμίσεις ευνοούν πιο πολύ τα τρισδιάστατα games και θα επηρεάσουν αρνητικά τα γραφικά του παιχνιδιού εάν δεν αλλαχθούν. Όλες οι παρακάτω ρυθμίσεις βρίσκονται ήδη εντός του προγραμματιστικού περιβάλλοντος του UE5 και ο προγραμματιστής δεν χρειάζεται να τις εκτελέσει χειροκίνητα, αλλά μόνο να τις αλλάξει από το μενού ρυθμίσεων. Η πρώτη ρύθμιση που χρειάστηκε αλλαγή είναι αυτή της εξομάλυνσης των ακμών. Πρόκειται για μια τεχνική που χρησιμοποιείται στην ψηφιακή επεξεργασία εικόνας για να μειώσει ή να εξαλείψει την εμφάνιση των "σκαλοπατιών" (γνωστά ως "aliasing") στις γραμμές και τις ακμές των ψηφιακών εικόνων. Έτσι έγινε η επιλογή του FXAA(Fast approximate – anti aliasing). Αν και είναι ταχύτερο, το FXAA μπορεί να είναι λιγότερο ακριβές και να θολώνει λεπτομέρειες της εικόνας σε σύγκριση με άλλες μεθόδους anti-aliasing. Οπότε με λίγα λόγια επιλέχθηκε η έμφαση στην απόδοση του παιχνιδιού και όχι στις λεπτομέρειες των γραφικών, καθώς έτσι και αλλιώς στην εργασία αυτή γίνεται χρήση 2d pixelated art, ενός στυλ ρετρό γραφικών που δεν απαιτούν ευκρίνεια. Στη συνέχεια γίνεται η απενεργοποίηση της ρύθμισης θαμπάδας κίνησης (motion blur), όπου προσδίδει μία θαμπάδα την κάμερα του παίκτη για να δοθεί η ψευδαίσθηση της κίνησης. Όμως στα 2d games ο παίκτης έχει μία εξωτερική οπτική γωνία λες και παρατηρεί τον χαρακτήρα του από το πλάι, οπότε η θαμπάδα στην οθόνη είναι εντελώς αχρείαστη. Η επόμενη ρύθμιση είναι η προεπιλογή των αριθμών των πίξελ (pixel) στην οθόνη από τα οποία αποτελούνται τα γραφικά αντικείμενα. Ένα Unreal engine unit αντιστοιχεί σε ένα εκατοστό και από προεπιλογή σε ένα Pixel. Σε αυτή τη ρύθμιση γίνεται η επιλογή 0,25 από 1 έτσι ώστε να επιτευχθεί η επιθυμητή ανάλυση και ευκρίνεια στην οθόνη. Ο καθορισμός της τιμής "Default Pixels Per Unit" (PPU) σε 0,25 στο Unreal Engine σημαίνει ότι κάθε μονάδα του κόσμου (world unit) αντιστοιχεί σε 0,25 pixels. Αυτό έχει άμεσο αντίκτυπο στην κλίμακα και την απεικόνιση των γραφικών assets στον κόσμο του παιχνιδιού ή της εφαρμογής. Δηλαδή αν η τιμή PPU είναι 0.25, τότε για να γεμίσει μία μονάδα του κόσμου με pixels, θα χρειαστούν 4 pixels (δηλαδή, 1 μονάδα του κόσμου = 4 pixels). Αυτό πρακτικά σημαίνει ότι τα γραφικά assets θα εμφανίζονται 4 φορές μεγαλύτερα από ότι αν η τιμή PPU ήταν 1. Ακόμα για την προετοιμασία του παιχνιδιού εγκαταστάθηκε στο UE5 μία δωρεάν επέκταση, το paperZD που όπως έχει αναφερθεί σε παραπάνω ενότητα είναι πολύ σημαντικό για την διαχείριση των γραφικών στοιχείων.

4.1.2 Προετοιμασία γραφικών στοιχείων (sprites)

Για την απόκτηση όλων των γραφικών στοιχείων της διπλωματικής εργασίας έγινε εγκατάσταση και ενσωμάτωση έτοιμων αρχείων τα οποία είναι τύπου cc0. Αυτό σημαίνει πως τα στοιχεία αυτά μπορούν να χρησιμοποιηθούν ελεύθερα είτε για προσωπική είτε για εμπορική χρήση. Επιτρέπετε η τροποποίηση τους και η αναδιανομή προς όλα τα νομικά πρόσωπα. Δεν χρειάζεται να ζητηθεί άδεια χρήσης ούτε να γίνει ονομασία της προέλευσης. Τα αρχεία αυτά περιλαμβάνουν έναν φάκελο χαρακτήρων με τα sprite των εχθρών και του παίκτη και κάθε φάκελος περιλαμβάνει ένα αρχείο τύπου .png και ένα αρχείο τύπου .json. Επιπλέον υπάρχει ένας φάκελος με αρχεία png που περιλαμβάνει τα αντικείμενα που θα χρησιμοποιηθούν για την δημιουργία του επιπέδου και των σκηνικών. Τα στοιχεία αυτά ονομάζονται tile sets. Τέλος υπάρχει ένας φάκελος με εικόνες από διάφορα εφέ που θα χρησιμοποιηθούν. Για την ενσωμάτωση όλων αυτών των αρχείων δημιουργήθηκε ένας φάκελος με όνομα paper assets. Εντός αυτού του φάκελου δημιουργήθηκε ένας φάκελος με όνομα characters (χαρακτήρες) και εντός αυτού ένας με όνομα Player. Τα json αρχεία δημιουργήθηκαν με τη βοήθεια του εργαλείου aspritre που βοηθάει στην απόσπαση πληροφοριών από μία ακολουθία sprites και την διαχώριση των animations (κινήσεων των χαρακτήρων) σε διάφορες κατηγορίες, όπως θα φανεί στην εικόνα 7. Για την δημιουργία των sprites χρησιμοποιήθηκαν τα αρχεία json για την απλοποίηση της διαδικασίας, αλλά για το πρώτο sprite θα χρησιμοποιηθεί το αρχείο png για την ανάδειξη της διαδικασίας δημιουργίας ενός sprite χαρακτήρα από αρχείο png. Στο επόμενο βήμα γίνεται η εισαγωγή του png αρχείου με τα sprite του παίχτη στον φάκελο player. Αρχικά το αρχείο δεν φαίνεται ως ένα αρχείο με pixel art, οπότε για να δοθεί η κατάλληλη υφή στο αρχείο, πηγαίνοντας στο content drawer editor επιλέγεται το sprite actions και έπειτα apply paper 2d texture settings. Αυτό δίνει την επιθυμητή όψη στα γραφικά. Στη συνέχει από τις ρυθμίσεις γίνεται αλλαγή του φίλτρου της εικόνας στην επιλογή nearest. Αυτή η επιλογή γίνεται εμπειρικά, καθώς δοκιμάζοντας τα διάφορα φίλτρα εικόνας μόνο αυτό έχει τα επιθυμητά αποτελέσματα που ταιριάζουν στο pixel art της εργασίας. Έπειτα καθώς όλα τα sprite είναι ενοποιημένα στην εικόνα, κάνοντας δεξί κλικ στο αρχείο με τα sprite και έπειτα επιλέγοντας extract sprites το UE5 εντοπίζει και διαχωρίζει αυτόματα τα animation ανά κατηγορίες. Στις παρακάτω εικόνες θα αναδειχθεί το πριν και το μετά της εξαγωγής των sprite.



Εικόνα 7: Asprite διαχωρισμός animation



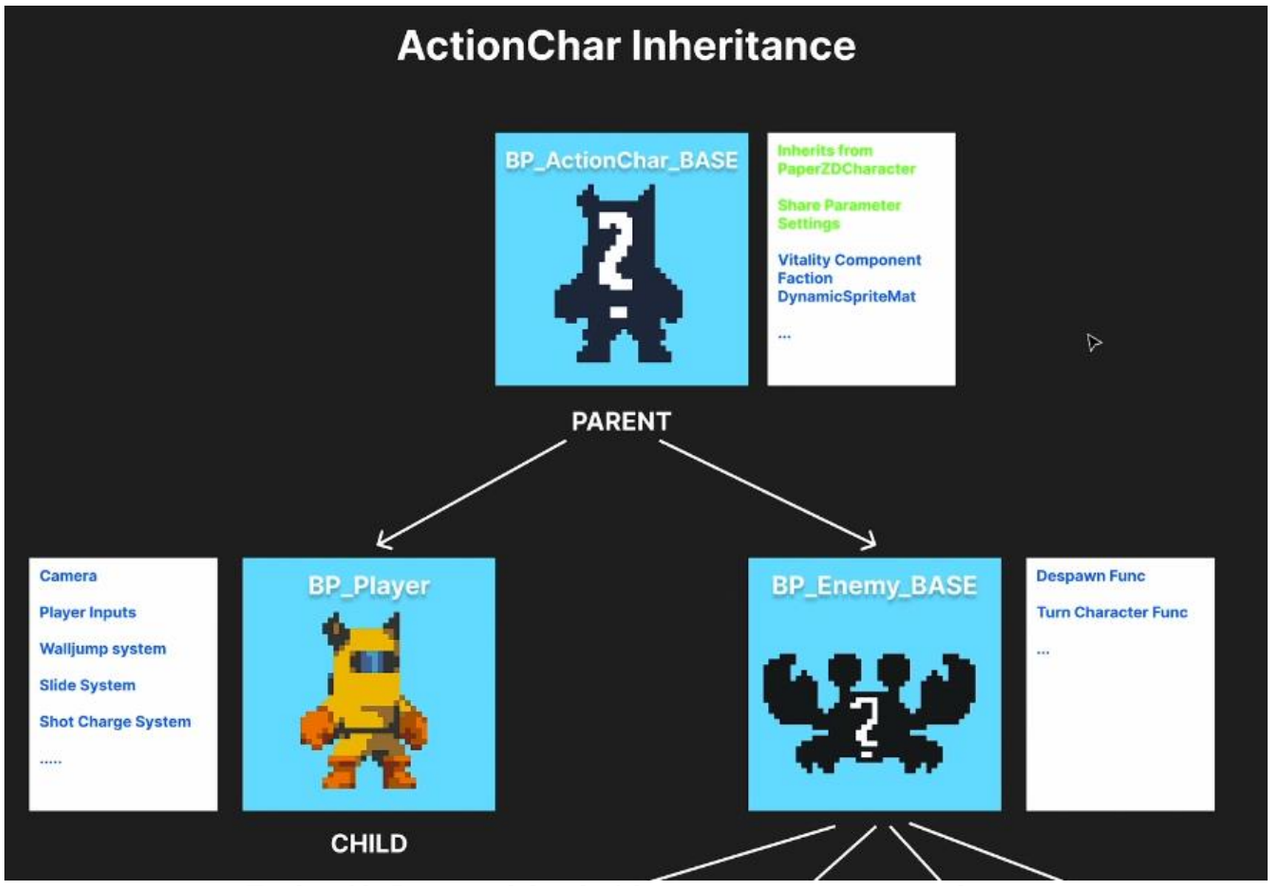
Εικόνα 8: Πριν την εξαγωγή

Επειδή είναι ακόμα ενοποιημένα εντός ενός πορτοκαλί κουτιού έγινε διαχωρισμός της εικόνας σε 6 στήλες και 14 σειρές από τις ρυθμίσεις. Μετά πατώντας το κουμπί extract, πλέον απομονώνονται όλα τα animation σε ξεχωριστά sprites. Στη συνέχεια επιλέγοντας όλα τα sprite τις κάθε κατηγορίας, όπως π.χ. τρέξιμο ή πυροβολισμός και πατώντας δεξί κλικ create flipbook, τα sprite αυτά ενοποιούνται και παίζουν σε σειρά δημιουργώντας έτσι τα ολοκληρωμένα animations. Πλέον έχουν δημιουργηθεί τα animations του παίχτη. Ένα flipbook είναι μία συλλογή από sprite που τρέχουν σε σειρά με συγκεκριμένο ρυθμό(frames per second). Έτσι δημιουργούνται τα flipbooks από ένα αρχείο png. Επειδή όμως έχουν δημιουργηθεί ήδη αρχεία json που είναι πολύ πιο αποτελεσματικά, τώρα θα αναδειχθεί ο πραγματικός τρόπος που υλοποιήθηκε η δημιουργία των animation στην εργασία. Αρχικά έγινε η εισαγωγή του json αρχείου στον φάκελο player. Με την πραγματοποίηση της εισαγωγής το UE5 αναγνώρισε αυτόματα το json file και δημιούργησε 3 αρχεία, έναν φάκελο με όνομα frames, ένα φάκελο textures και ένα sprite sheet. Στον φάκελο frames δημιουργούνται όλα τα sprites που προηγουμένως είχαν δημιουργηθεί χειροκίνητα με αυτόματο τρόπο. Στον φάκελο textures βρίσκεται η εικόνα 9 η οποία δημιουργήθηκε επίσης αυτόματα με τις σωστές ρυθμίσεις. Κάνοντας δεξί κλικ στο αρχείο sprite sheet και επιλέγοντας create flipbooks δημιουργούνται αυτόματα όλα τα flipbook που προηγουμένως είχαν δημιουργηθεί αυτόματα εξοικονομώντας πάρα πολύ χρόνο. Για αυτό και προτιμάτε η δημιουργία και η χρήση των αρχείων json. Στη συνέχεια επιλέγοντας όλα τα animations flipbooks – asset actions – edit selection in property matrix έγινε η ρύθμιση της ταχύτητας αναπαραγωγής των animation από 15 καρέ το δευτερόλεπτο (frames per second - fps) σε 7, για να φαίνονται οι κινήσεις πιο φυσικές. Έπειτα για την εισαγωγή τον εχθρών δημιουργήθηκε εντός του φακέλου characters ένας νέος φάκελος με όνομα enemies, όπου εκεί τοποθετήθηκαν τα αρχεία json των εχθρών. Ακριβός η ίδια διαδικασία επαναλήφθηκε για την δημιουργία των animations των εχθρών. Έπειτα στον φάκελο paper assets δημιουργήθηκε ένας νέος φάκελος με όνομα Fx, όπου εντός αυτού εισάχθηκαν τα αρχεία με τα ειδικά εφέ (Fx). Πάλι ακολουθήθηκε η ίδια διαδικασία για την δημιουργία των flipbooks. Τέλος δημιουργήθηκε ένας επιπλέον φάκελος με όνομα environment, όπου τοποθετήθηκαν τα αρχεία με τα tile sets. Αυτά τα αρχεία δεν είναι ίδιου τύπου με τα προηγούμενα και δεν αξιοποιήθηκαν ακόμα σε αυτό το σημείο της εργασίας. Μόνο πειράζοντας τις ρυθμίσεις έγινε ρύθμιση των γραφικών σε 2d paper texture, όπως και προηγουμένως. Τέλος ρυθμίστηκε το φίλτρο για αυτά τα αρχεία στην επιλογή nearest.

Εικόνα 9: Μετά την εξαγωγή

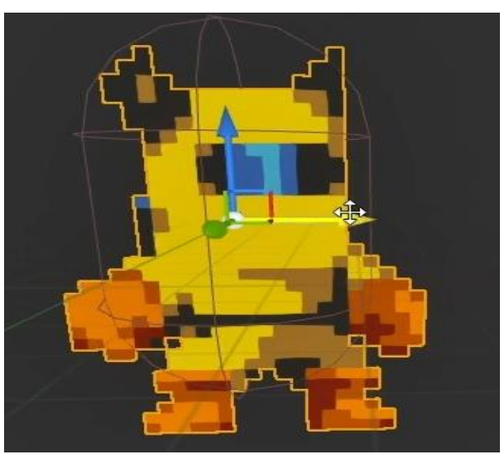
4.1.3 Δημιουργία blueprint βασικού χαρακτήρα και κληρονομικότητας

Σε αυτό το στάδιο γίνεται η αρχικοποίηση του βασικού χαρακτήρα, όπου εντός των φακέλων του πρότζεκτ θα δημιουργηθεί ένας νέος φάκελος με όνομα blueprints, όπου εκεί θα αποθηκεύονται όλα τα αντικείμενα που θα χρησιμοποιηθούν για την εργασία. Εντός του φακέλου blueprints θα δημιουργηθεί ένας νέος φάκελος με όνομα characters, όπου εκεί θα τοποθετούνται τα blueprints των χαρακτήρων. Εντός του φακέλου αυτού θα δημιουργήσουμε μία κλάση τύπου paperZD character η οποία είναι παιδί των κλάσεων object- actor-pawn-character- papercharacter- paperzd character. Έτσι η κλάση paperZD θα κληρονομήσει όλα τα αντικείμενα και τις μεθόδους όλων των προαναφερθέντων κλάσεων γονέων με αποτέλεσμα να αυτοματοποιηθούν και απλοποιηθούν πολλές λειτουργίες του παίχτη. Όλες αυτές οι κλάσεις παρέχονται από το UE5. Μερικά χαρακτηριστικά της κλάσης αυτής είναι: Ο χαρακτήρας αποτελείται από ένα UPaperSpriteComponent, το οποίο χρησιμοποιείται για την εμφάνιση και τη διαχείριση των γραφικών του χαρακτήρα. Ενσωματώνει ένα CharacterMovementComponent που παρέχει τις δυνατότητες κίνησης και φυσικής, επιτρέποντας στον χαρακτήρα να περπατά, να τρέχει, να πηδά, κλπ. Μπορεί να υποστηρίξει 2D animations μέσω flipbooks (UPaperFlipbook), επιτρέποντας την αναπαραγωγή ακολουθιών εικόνων για τη δημιουργία κινούμενων χαρακτήρων. Διαθέτει ένα CapsuleComponent για τη διαχείριση της σύγκρουσης και της φυσικής του χαρακτήρα, το οποίο μπορεί να προσαρμοστεί ανάλογα με τις ανάγκες του παιχνιδιού. Μπορεί να γίνει χρήση των state machines και blend spaces για τον έλεγχο και τη μετάβαση μεταξύ διαφορετικών καταστάσεων και κινήσεων του χαρακτήρα. Τέλος υποστηρίζει ενσωματωμένη διαχείριση εισόδων, επιτρέποντας τον εύκολο χειρισμό κινήσεων και ενεργειών του χαρακτήρα μέσω πληκτρολογίου, gamepad ή άλλων συσκευών εισόδου. Έτσι δημιουργήθηκε η πρώτη κλάση με όνομα BP_ActionChar_Base. Από κάτω θα παρουσιαστεί το σχέδιο κληρονομικότητας που θα εφαρμοστεί έτσι ώστε να απλοποιηθεί η υλοποίηση του πρότζεκτ, καθώς πολλές κλάσεις χαρακτήρων θα μοιράζονται αρκετά κοινά χαρακτηριστικά. Οπότε για να μειωθεί η επανάληψη ενσωμάτωσης κάποιων μεθόδων και αντικειμένων θα χρησιμοποιηθεί η παρακάτω δομή της εικόνας 10 και 11.

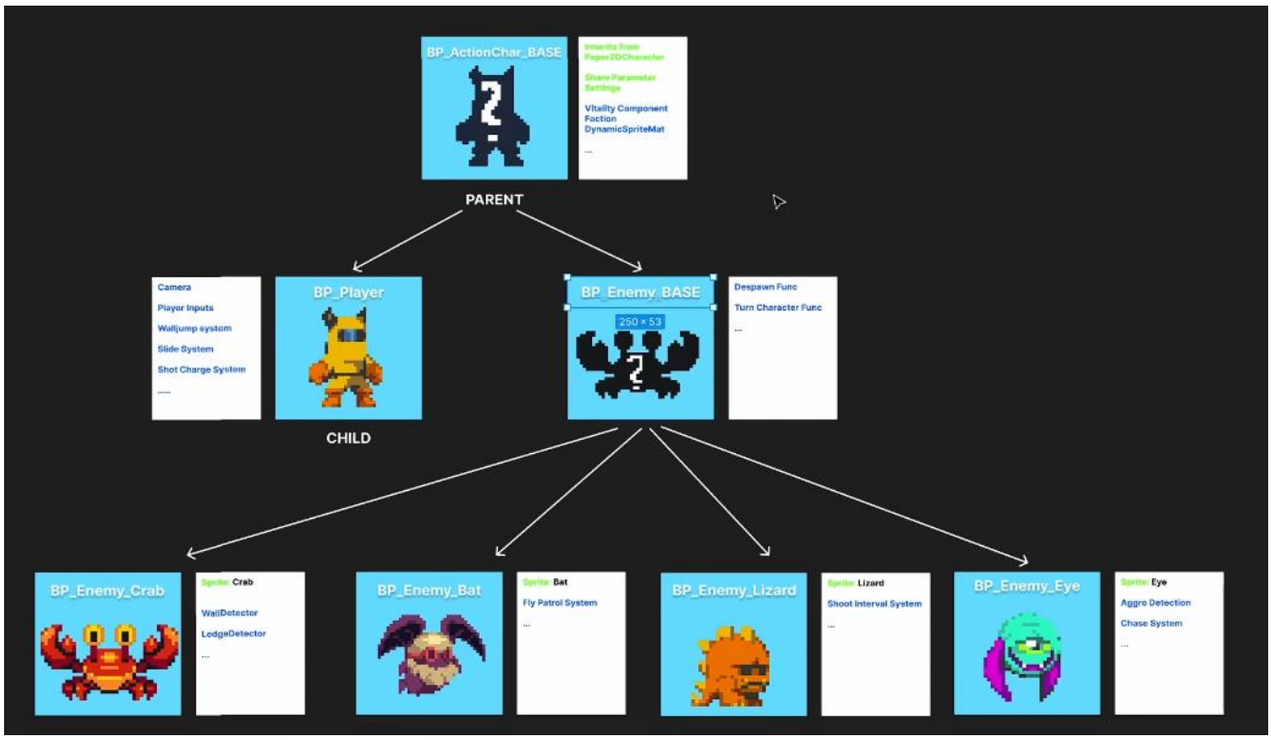


Εικόνα 10: Ιεραρχική δομή κλάσεων και χαρακτηριστικά τους

Η βασική μας κλάση κληρονομεί από την κλάση paperZD character του UE5 plugin. Θα έχει ως αντικείμενα ένα vitality component, faction και DynamicSpriteMat. Τα; components είναι βασικά δομικά στοιχεία που μπορούν να προστεθούν σε actors (ηθοποιοί) για να τους δώσουν συγκεκριμένες ιδιότητες και λειτουργικότητες. Σε αυτή την περίπτωση το component vitality θα χειρίζεται όλες τις λειτουργίες που έχουν να κάνουν με την ζωή του παίχτη. Η μεταβλητή faction θα χρησιμοποιηθεί για τον διαχωρισμό τον φιλικών με των εχθρικών μονάδων. Στη συνέχεια θα δημιουργηθούν δύο κλάσεις παιδιά, μία για τον χαρακτήρα του παίχτη και μία για τους εχθρούς με ονόματα BP_Player και BP_Enemy_Base. Στην συνέχεια όλοι οι διαφορετικοί τύποι εχθροί θα είναι παιδιά της κλάσης enemy base. Τα κληρονομημένα αντικείμενα και οι μέθοδοι της κάθε κλάσης, καθώς και τα δικά τους μοναδικά χαρακτηριστικά φαίνονται στις εικόνες και θα επεξηγηθούν στην παρακάτω σε αυτό το κεφάλαιο, καθώς θα δημιουργηθούν. Στη συνέχεια εντός την κλάσης BP_actionchar_base θα ορίσουμε το sprite για την κλάση αυτή το animation player idle. Έπειτα θα ρυθμιστεί το capsule component έτσι ώστε η κάψουλα να περικλείει τον χαρακτήρα με ακρίβεια καθώς αυτή θα καθορίσει τις φυσικές διαστάσεις του.



Εικόνα 11: player capsule component

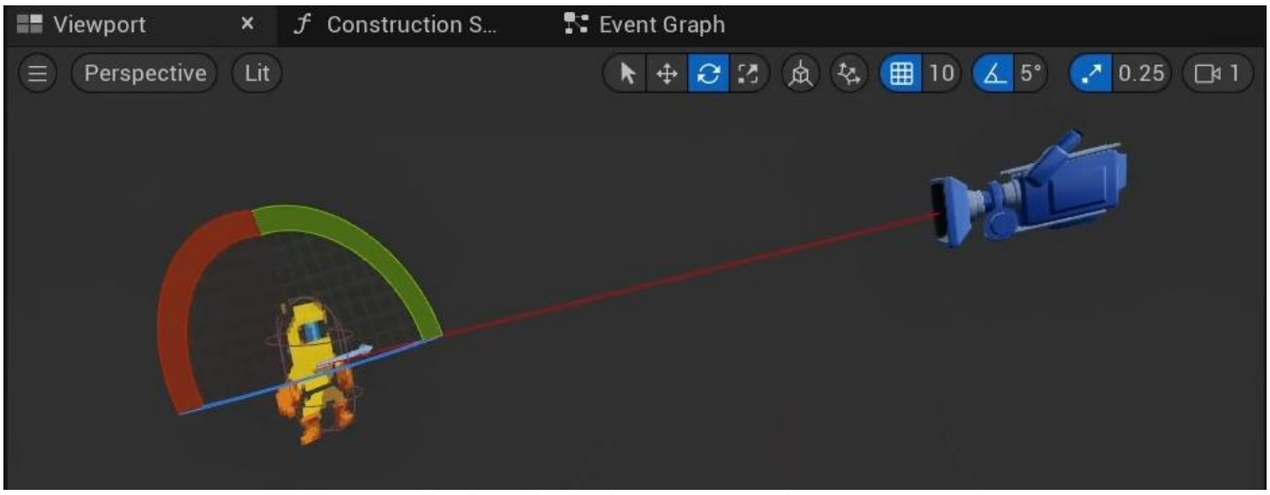


Εικόνα 12: Πλήρης δομή κληρονομικότητας χαρακτήρων

Στη συνέχεια είναι πολύ σημαντικό να οριστεί σε ποια θα βρίσκεται ο παίχτης στον χώρο, οπότε στον άξονα y που είναι το βάθος είναι πολύ σημαντικό να τον ορίσουμε στην θέση 0. Όπως και θα οριστεί για κάθε αντικείμενο στον χώρο, αφού το παιχνίδι μας είναι δισδιάστατο και όχι τρισδιάστατο. Έπειτα τραβώντας με το ποντίκι τον χαρακτήρα του παίχτη εντός του αρχείου με το επίπεδο, ο χαρακτήρας θα πάρει για πρώτη φορά υπόσταση. Όμως επειδή έχει μία ελαφριά αιώρηση που δεν θα έπρεπε να υπάρχει γίνεται μία μικρή ρύθμιση στον άξονα z εντός το action char base στη θέση 13,9 έτσι ώστε να ακουμπάει ο χαρακτήρας στο έδαφος. Τελικά οι συντεταγμένες που ορίστηκαν είναι x: -5, y:0, z:13,9. Έπειτα στον φάκελο blueprrints- characters θα δημιουργηθεί ένας νέος φάκελος με ονομασία player, και μία νέα κλάση τύπου paperZd character με όνομα BP_Player και θα οριστεί ως παιδί της κλάσης action char base.

Το επόμενο βήμα είναι η απόκτηση του ελέγχου του παίχτη από τον χειριστή. Για να επιτευχθεί αυτό θα πρέπει πρώτα να δημιουργηθεί ένα game mode class το οποίο θα έχει τους βασικούς ελέγχους και παραμέτρους του παιχνιδιού. Η κλάση Game Mode στην Unreal Engine 5 (UE5) είναι μια από τις πιο βασικές κλάσεις που ρυθμίζει τη ροή του παιχνιδιού και τις βασικές του παραμέτρους. Ουσιαστικά, η Game Mode κλάση καθορίζει το πώς θα λειτουργεί το παιχνίδι, ορίζοντας διάφορες πτυχές του gameplay και παρέχοντας κανόνες και λογική που διέπουν την εμπειρία του παιχνιδιού. Έτσι εντός του φακέλου blueprints έγινε η δημιουργία μίας κλάσης τύπου game mode με όνομα GM_Action. Μερικές από τις λειτουργίες που θα χρησιμοποιηθούν είναι ο καθορισμός ποιας κλάσης θα χρησιμοποιηθεί ως ο προεπιλεγμένος χαρακτήρας του παίκτη. Κάθε φορά που ένας παίκτης γεννιέται στον κόσμο, θα χρησιμοποιηθεί αυτός ο χαρακτήρας και ο καθορισμός ποιας κλάση θα χρησιμοποιηθεί για τον έλεγχο του παίκτη. Το Player Controller διαχειρίζεται τις εισόδους του παίκτη και αλληλοεπιδρά με τον κόσμο του παιχνιδιού. Εντός του gm action θα ορίσουμε ως default pawn class την κλάση bp player, δηλαδή ως βασικό χαρακτήρα παίχτη. Έπειτα για να γίνει η ενεργοποίηση του gm action σαν την κλάση ελέγχου του παιχνιδιού πηγαίνοντας στις ρυθμίσεις του πρότζεκτ, στην κατηγορία maps and modes ορίζεται ως προεπιλεγμένο πρόγραμμα παιχνιδιού η κλάση gm action. Για να εμφανιστεί ο χαρακτήρας στο επίπεδο πρέπει να οριστεί ένα player start position, δηλαδή το σημείο από το οποίο θα ξεκινήσει ο παίχτης. Αυτό παρέχεται αυτόματα από το UE5, οπότε το μόνο που έχει να κάνει ο προγραμματιστής είναι να το τοποθετήσει σε σημείο αρεσκείας του εντός του επιπέδου. Πλέον με την εκκίνηση του παιχνιδιού ο παίχτης ξεκινά εντός του χαρακτήρα του, αλλά δεν μπορεί να κάνει καμία κίνηση, αφού δεν έχει οριστεί ακόμα η λειτουργικότητα του χαρακτήρα. Για τη δημιουργία των διάφορων λειτουργιών πρέπει πρώτα να δημιουργηθεί ένα χειριστήριο παίκτη (player controller) το οποίο θα μπορεί να δέχεται inputs από το πληκτρολόγιο. Πριν όμως γίνει αυτό υπάρχει ένα ακόμη πρόβλημα. Κατά τον έλεγχο του χαρακτήρα από τον παίχτη, ο δεύτερος έχει μία περίεργη οπτική γωνία σαν να βρίσκεται εντός του χαρακτήρα. Φυσικά αυτή δεν είναι η επιθυμητή οπτική γωνία για τον παίχτη, καθώς αυτός πρέπει να βλέπει τον

χαρακτήρα του από το πλάι. Οπότε στη συνέχεια θα γίνει η ρύθμιση της οπτικής γωνίας του παίχτη. Εντός της κλάσης bp player στο viewport editor, όπως ονομάζεται μπορεί ο προγραμματιστής να διαχειριστεί τα γραφικά στοιχεία και τα components του blueprint. Έτσι στο capsule component θα τοποθετηθεί ένα spring arm και πάνω στο spring arm θα τοποθετηθεί μία κάμερα. Το spring arm είναι μία αόρατη προέκταση από την κάψουλα του παίχτη προς το πλάι. Έτσι πραγματοποιείτε περιστροφή του spring arm κατά -90 μοίρες, ώστε να έρθει από την πίσω όψη στην πλαϊνή. Έπειτα έγινε περιστροφή κατά -15 μοίρες στον άξονα y, ώστε να δοθεί η αίσθηση ότι γίνεται η παρακολούθηση του χαρακτήρα από ένα πιο ψηλό σημείο. Τέλος πρέπει να αφαιρεθεί η ικανότητα σύγκρουσης (collision) του spring arm με άλλα αντικείμενα, οπότε στις ρυθμίσεις do collision test απενεργοποιήθηκε αυτή η ρύθμιση.



Εικόνα 13: Spring arm και κάμερα

Τώρα ήρθε η στιγμή για την δημιουργία των βασικών κινήσεων του χαρακτήρα και γίνεται η δημιουργία ενός νέου φακέλου με το όνομα inputs και εντός αυτού ενός άλλου με το όνομα actions. Η είσοδος εντολών γίνεται στο UE5 με τη χρήση της enhanced Input Actions class, που θα χρησιμοποιηθεί σε λίγο. Στον φάκελο actions δημιουργούνται κλάσεις για κάθε είδος δράσης που θα κάνει ο παίχτης. Δημιουργήθηκαν έτσι οι δράσεις (actions), IA_Jump (αναπήδηση) τύπου digital boolean, IA_Move (κίνηση) τύπου axis 1d(float), κίνησης σε μία διάσταση. Αφού δημιουργήθηκαν οι δύο πιο βασικές δράσεις κίνησης τώρα πρέπει να γίνει η σύνδεσή τους με το πληκτρολόγιο. Οπότε στον φάκελο input δημιουργήθηκε μία κλάση τύπου input mapping context με όνομα IMC_Action με σκοπό την διαχείριση της εισόδου εντολών από το πληκτρολόγιο. Οπότε η AI Jump δέθηκε με το πλήκτρο space και k. Η IA Move δέθηκε με τα πλήκτρα d (κίνηση προς τα δεξιά στον θετικό άξονα x), με το A (κίνηση προς τα αριστερά με negate modifier που αντιστρέφει το πρόσημο της κίνησης για “αρνητική” κίνηση στον άξονα x). Για το επόμενο στάδιο πρέπει να δοθεί η εντολή στον παίχτη να χρησιμοποιήσει την κλάση imc action, ώστε να καταλάβει η κλάση bp player από που θα δέχεται εντολές.

```
// Συνάρτηση για αρχικοποίηση του game mode
ΣΥΝΑΡΤΗΣΗ InitializeGameMode()
    // Παίρνουμε τον Player Controller (για single-player)
    PlayerController = ΠάρεPlayerController(αντίγραφο αυτού του game mode, Παίκτης 0)
    // Έλεγχος αν ο PlayerController είναι έγκυρος
    AN PlayerController υπάρχει ΤΟΤΕ
        // Παίρνουμε το Pawn που ελέγχεται από τον PlayerController
        ControlledPawn = PlayerController.ΠάρεΕλεγχόμενοPawn()
        // Έλεγχος αν το Pawn υπάρχει
```

```
        ΑΝ ControlledPawn υπάρχει ΤΟΤΕ
            // Κάνουμε cast το Pawn στην κλάση του συγκεκριμένου χαρακτήρα μας
            PlayerCharacter = ΚάνεCastΣεYourGameCharacter(ControlledPawn)
            // Αν το cast είναι επιτυχές και ο PlayerCharacter είναι έγκυρος
            ΑΝ PlayerCharacter υπάρχει ΤΟΤΕ
                // Αλληλεπιδράστε με τον χαρακτήρα του παίκτη (π.χ., κλήση κάποιας συνάρτησης)
                PlayerCharacter.ΕκτέλεσηΚάποιαΣυνάρτησης()
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για τη ρύθμιση του input του παίκτη
ΣΥΝΑΡΤΗΣΗ SetupPlayerInput()
    // Δέσμευση της ενέργειας "Jump" στο πλήκτρο spacebar
    ΔέσμευσηInputΕνέργειας("Jump", ΌτανΠατηθείΠλήκτρο, Jump)
    // Δέσμευση της ενέργειας "StopJumping" όταν το πλήκτρο αφήνεται
    ΔέσμευσηInputΕνέργειας("Jump", ΌτανΑφεθείΠλήκτρο, StopJumping)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για την ενέργεια Jump
ΣΥΝΑΡΤΗΣΗ Jump()
    // Κλήση της βασικής συνάρτησης για Jump
    ΚλήσηΒασικήςΣυνάρτησηςCharacterJump()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για τη διακοπή της αναπήδησης
ΣΥΝΑΡΤΗΣΗ StopJumping()
    // Κλήση της βασικής συνάρτησης για διακοπή αναπήδησης
    ΚλήσηΒασικήςΣυνάρτησηςCharacterStopJumping()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

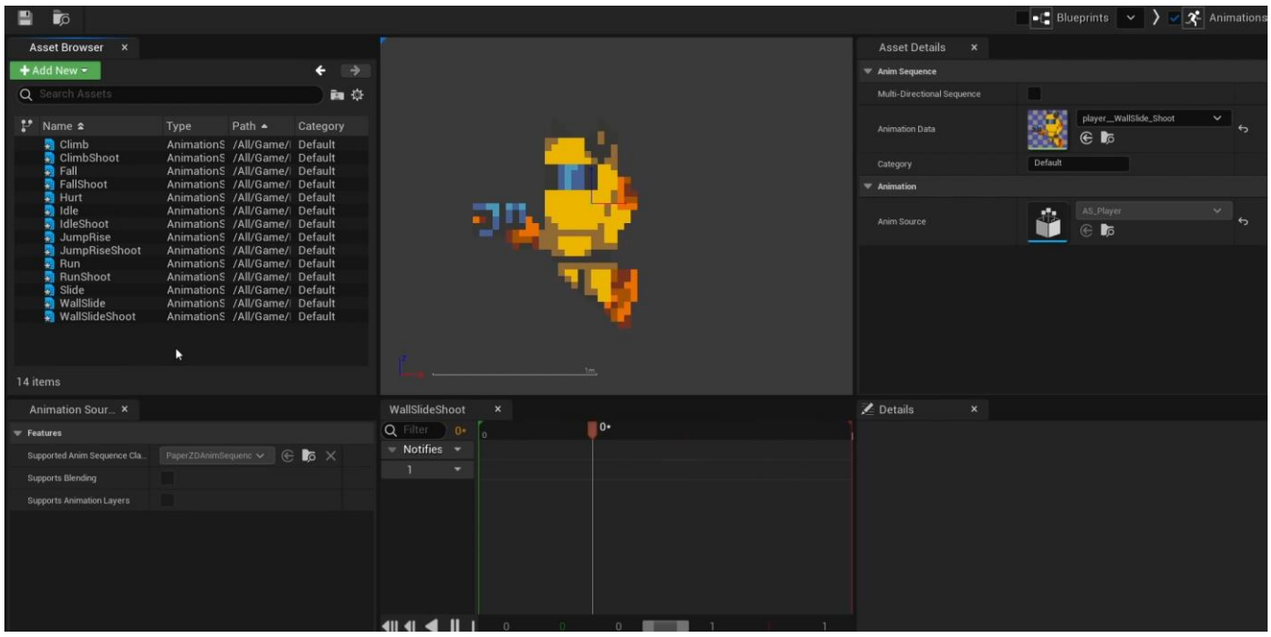
Μετέπειτα θα γίνει η εισαγωγή τον input actions που δημιουργήθηκαν προηγουμένως. Με την εκκίνηση του γεγονότος δηλαδή με το πάτημα του κουμπιού ο χαρακτήρας θα πραγματοποιεί αναπήδηση. Είναι πολύ σημαντικό να οριστεί μετά το πέρας την πράξης ο χαρακτήρας να σταματήσει την αναπήδηση, οπότε πρέπει να προγραμματιστεί και αυτό.

```
// YourCharacter.cpp
ΣΥΝΑΡΤΗΣΗ SetupPlayerInputComponent(PlayerInputComponent)
    ΚάλεσεΤηΒασικήΣυνάρτησηSetupPlayerInputComponent(PlayerInputComponent)
    // Δέσμευση της ενέργειας "Jump" στο πλήκτρο Spacebar
```

```
    PlayerInputComponent.ΔέσμευσηΕνέργειας("Jump", ΌτανΠατηθείΠλήκτρο, Jump)
    PlayerInputComponent.ΔέσμευσηΕνέργειας("Jump", ΌτανΑφεθείΠλήκτρο, StopJumping)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

ΣΥΝΑΡΤΗΣΗ Jump()
    // Κλήση της βασικής συνάρτησης Jump
    ΚλήσηΒασικήςΣυνάρτησηςJump()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

ΣΥΝΑΡΤΗΣΗ StopJumping()
    // Κλήση της βασικής συνάρτησης StopJumping
    ΚλήσηΒασικήςΣυνάρτησηςStopJumping()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Στη συνέχεια επαναλαμβάνεται η ίδια διεργασία για την κλάση IA Move, όμως σε αυτή την περίπτωση δεν είναι επιθυμητό να πραγματοποιείται η κίνηση με το πάτημα του κουμπιού, αλλά με το κράτημα και να σταματάει με την αποδέσμευση του πλήκτρου. Επίσης στο character movement component που παρέχεται αυτόματα από την μηχανή στην κίνηση του παίχτη πρέπει να οριστεί η τιμή με την οποία περπατάει ο παίχτης δηλαδή η μονάδα της απόστασης και η κατεύθυνση. Έτσι ορίστηκε ως scale value για το περπάτημα η τιμή 1 που είναι και η αυτόματη τιμή και ως κατεύθυνση κίνησης ορίστηκε η τιμή 1 στον άξονα x και 0 για τους άξονες y και z, εφόσον η κίνηση πραγματοποιείται μόνο σε έναν άξονα στα 2d games πέρα από την αναπήδηση. Τώρα όμως παρόλο που ο χαρακτήρας μπορεί να περπατήσει δεν αλλάζει κατεύθυνση προς το σημείο που περπατάει, έτσι προστίθεται μία συνθήκη στο scale value. Εάν αυτό είναι μεγαλύτερο ή ίσο του 0, τότε κάνοντας αναφορά στην κλάση player controller πραγματοποιείται περιστροφή του παίχτη προς τα δεξιά, δηλαδή ορίζεται στον άξονα z η τιμή 0, ενώ εάν η τιμή είναι μικρότερη του 0 τότε ορίζεται στον άξονα z η τιμή 180 και γίνεται περιστροφή του sprite προς τα αριστερά. Το πρόβλημα που δημιουργείτε όμως είναι ότι αφού το spring arm με την κάμερα είναι συνδεδεμένα στον παίχτη κάνουν και αυτά περιστροφή μαζί του και έτσι δημιουργείτε ένα πού περίεργο εφέ. Έτσι πηγαίνοντας στις ρυθμίσεις του spring arm γίνεται αλλαγή της ρύθμισης περιστροφής από relative σε world και πλέον η κάμερα δεν περιστρέφεται μαζί με τον χαρακτήρα, προσδίδοντας έτσι αληθοφάνεια στην κίνησή του.

Σε αυτό το στάδιο θα ενσωματωθούν τα animation του χαρακτήρα που χρειάζονται στον κώδικα, καθώς μέχρι τώρα ο χαρακτήρας ενώ είχε κίνηση δεν αποτυπωνόταν αυτή στην πράξη. Εντός του φακέλου Player θα δημιουργηθεί ένα animation source class από το paperZD plugin με όνομα AS Player. Σε αυτό το editor θα δημιουργηθούν τα animation για τον παίχτη και ως δεδομένα animation θα επιλεγούν τα αντίστοιχα flipbooks που είχαν δημιουργηθεί προηγουμένως. Δηλαδή στο idle animation θα προστεθεί το idle flipbook. Παρακάτω υπάρχει η εικόνα που φαίνεται ακριβώς πως λειτουργεί ο editor με όλα τα animation του παίχτη πλέον ενσωματωμένα στην κλάση AS Player. Με τη δημιουργία των animations δημιουργήθηκε αυτόματα και ένας φάκελος με όνομα anime sequences στον φάκελο player που περιέχει όλα τα animations που δημιουργήθηκαν στην εικόνα 14. Στη συνέχεια στον φάκελο player δημιουργείτε μία κλάση τύπου paperZD animBP και επιλέγεται ως γονέας η κλάση AS Payer, ώστε η πρώτη να έχει πρόσβαση σε όλα τα animation της δεύτερης. Αυτή η κλάση επιτρέπει την δημιουργία animation trees και δηλαδή δένδρα αποφάσεων με καταστάσεις μετάβασης από το ένα animation στο άλλο. Όμως για να γίνει το παραπάνω πρέπει η κλάση αυτή να αποκτήσει πρόσβαση στα αντικείμενα της κλάσης BP Player από όπου θα χρειαστεί πληροφορίες για την δημιουργία των καταστάσεων μετάβασης.

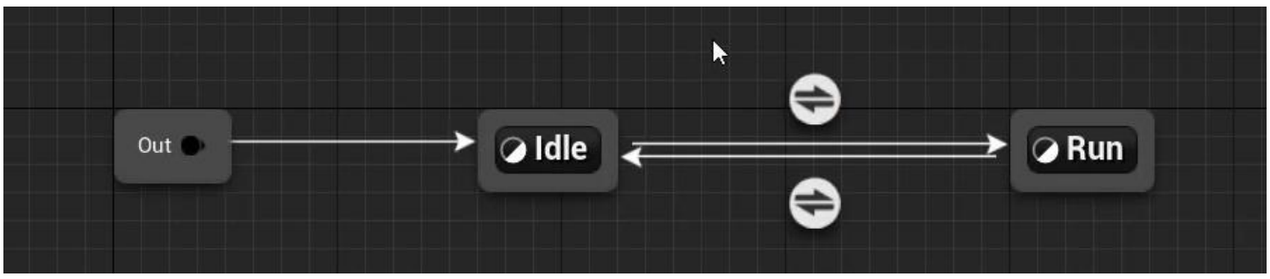


Εικόνα 14: AS Player editor - class animations

Κάπως έτσι θα πραγματοποιηθεί το παραπάνω βήμα:

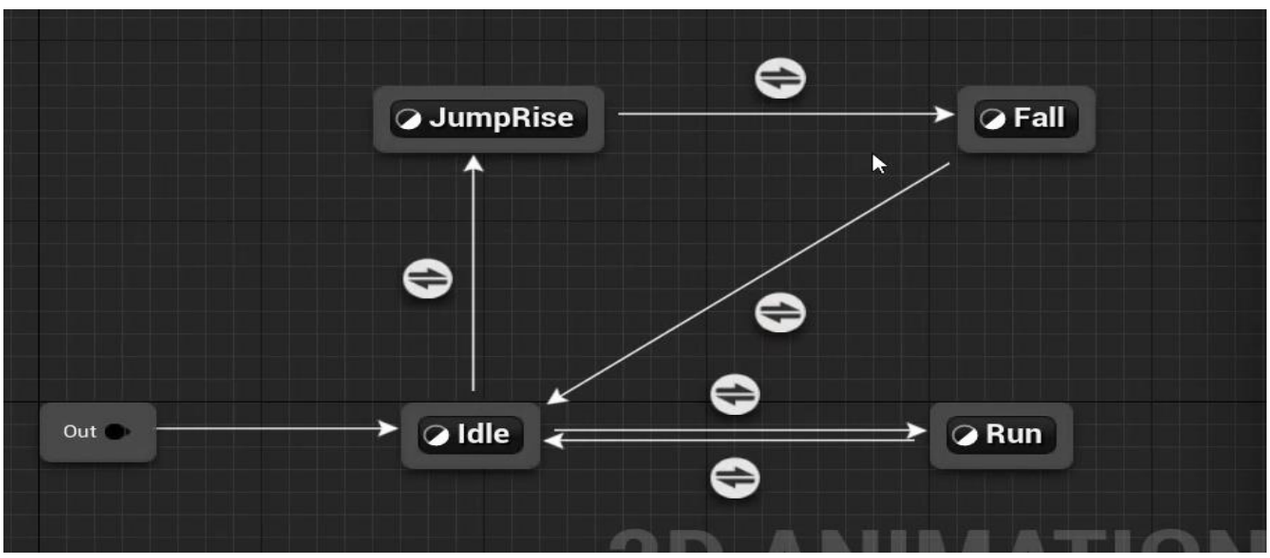
```
// YourFunction
ΣΥΝΑΡΤΗΣΗ YourFunction()
    // Παίρνουμε τον πρώτο Player Controller
    PlayerController = ΠάρεPlayerController(αυτή η κλάση, Παίκτης 0)
    ΑΝ PlayerController υπάρχει ΤΟΤΕ
        // Παίρνουμε το Pawn που ελέγχει ο Player Controller
        ControlledPawn = PlayerController.ΠάρεΕλεγχόμενοPawn()
        ΑΝ ControlledPawn υπάρχει ΤΟΤΕ
            // Κάνουμε cast το Pawn στην κλάση του χαρακτήρα μας
            PlayerCharacter = ΚάνεCastΣεYourPlayerCharacter(ControlledPawn)
            ΑΝ PlayerCharacter υπάρχει ΤΟΤΕ
                // Παίρνουμε το Anim Instance του χαρακτήρα
                AnimInstance = PlayerCharacter.ΠάρεMesh().ΠάρεAnimInstance()
                ΑΝ AnimInstance υπάρχει ΤΟΤΕ
                    // Κάνουμε cast το Anim Instance στην κλάση μας του AnimBP
                    YourAnimBP = ΚάνεCastΣεYourPaperZDAnimBPClass(AnimInstance)
                    ΑΝ YourAnimBP υπάρχει ΤΟΤΕ
                        // Τώρα μπορούμε να αλληλεπιδράσουμε με τις λειτουργίες του AnimBP
                        YourAnimBP.ΕκτέλεσηΚάποιαΣυνάρτησης()
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Επιπρόσθετα εντός της κλάσης BP Player στο animation component στις ρυθμίσεις πρέπει να οριστεί σαν anim instance class η κλάση ABP Player. Ουσιαστικά αυτή η ρύθμιση λέει στην κλάση του χαρακτήρα να πραγματοποιεί τις κινήσεις του με βάση το δένδρο αποφάσεων και τα animation της κλάσης ABP_Player. Οπότε τώρα εντός της ABP Player ορίζεται ως βασική κίνηση η αδράνεια και ως μεταβατική κίνηση το τρέξιμο. Όμως πρέπει να οριστεί και μία παράμετρος μετάβασης από τη μία συμπεριφορά στην άλλη, έτσι έχοντας κάνει ήδη αναφορά στην κλάση του παίχτη θα πάρουμε το velocity. Η ταχύτητα (velocity) σε ένα animation component στην Unreal Engine 5 είναι συνήθως μια μεταβλητή τύπου FVector. Ο τύπος FVector είναι μια δομή που αντιπροσωπεύει ένα διάνυσμα στον τρισδιάστατο χώρο και περιέχει τρεις συνιστώσες: X, Y και Z, οι οποίες αντιπροσωπεύουν τις ταχύτητες κατά μήκος των αντίστοιχων αξόνων. Οπότε μετά θα αναζητηθεί το μέγεθος του vector και με βάση την συνθήκη εάν το μήκος είναι μεγαλύτερο του 0, τότε θα ενεργοποιηθεί το run animation, αλλιώς στην αντίθετη περίπτωση με η τιμή του μήκους είναι μικρότερη ή ίση με το 0, τότε θα παίξει η κίνηση της αδράνειας. (εικόνα 15)



Εικόνα 15: Από αδράνεια σε κίνηση και αντιστρόφως

Από το idle μετά πάμε στο jump animation που είναι η κίνηση αναπήδησης. Ως παράμετρος κοιτάμε αν η παράμετρος is falling του movement component είναι αληθής, καθώς έχουμε ορίσει πως το η αναπήδηση είναι ένα action τύπου boolean. Εάν το is falling είναι αληθές τότε μεταβαίνει ο χαρακτήρας στην κατάσταση αναπήδησης, όπου μετά μεταβαίνει στην κατάσταση fall. Η συνθήκη μετάβασης εδώ είναι αν η τιμή του Z από το velocity που είναι τύπου Fvector είναι μικρότερη ή ίση με το μηδέν που σημαίνει ότι ο παίχτης κινείτε αρνητικά στον άξονα Z, δηλαδή πέφτει προς τα κάτω. Τελικά από το animation fall ο παίχτης μεταβαίνει στην κατάσταση idle υπό την συνθήκη ότι η μεταβλητή isfalling δεν είναι αληθής.(εικόνα 16). Επιπλέον δεν έχει δημιουργηθεί ακόμα μετάβαση από την κατάσταση τρέξιμο στο animation της αναπήδησης, οπότε αυτό είναι το επόμενο βήμα. Η συνθήκη μετάβασης είναι ακριβώς η ίδια με αυτή από την αδράνεια προς την αναπήδηση, δηλαδή αν ο bp player isfalling είναι αληθές τότε αυτό σημαίνει ότι βρίσκεται στον αέρα και περνάει στην κατάσταση αναπήδησης.

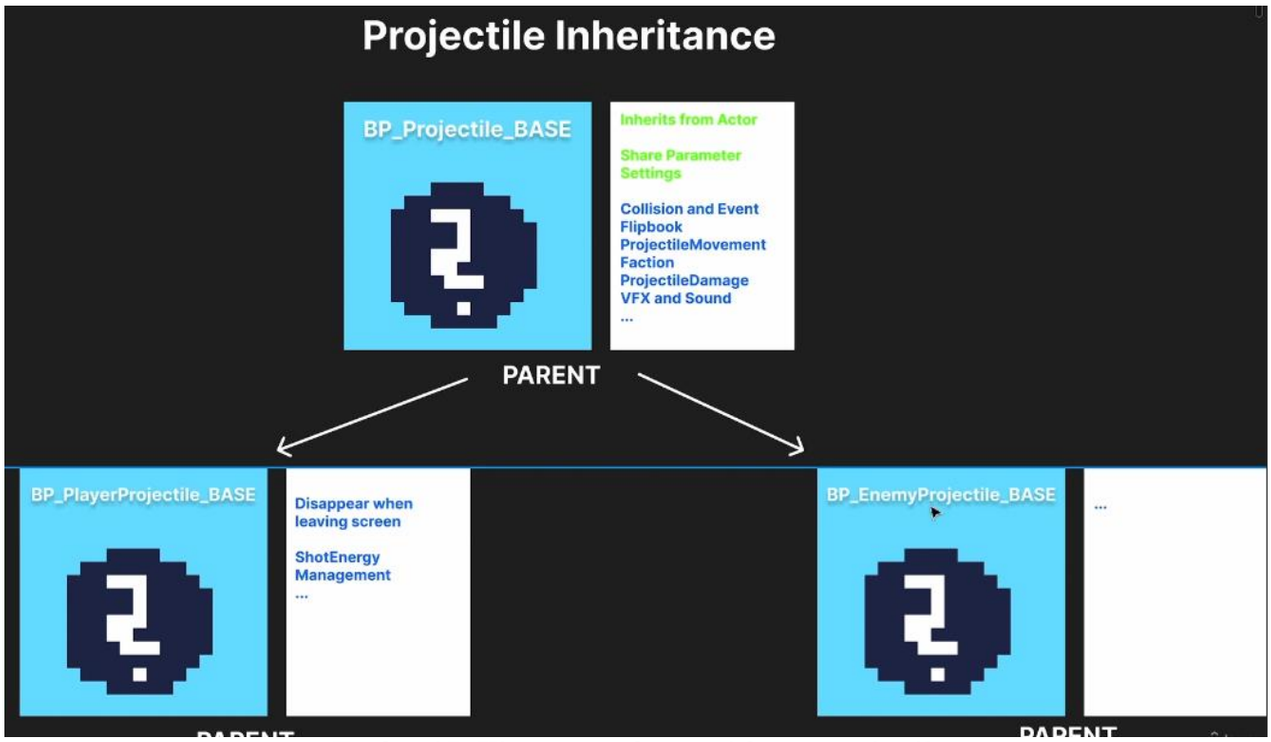


Εικόνα 16: Idle, walk, jump, fall animations

Ένα ακόμα βήμα για την βελτίωση της αλληλεπίδρασης του χαρακτήρα με το περιβάλλον είναι η διόρθωση των μηχανικών όσον αφορά τη σύγκρουση με άλλα αντικείμενα. Καθώς πρόκειται για ένα 2d παιχνίδι είναι πολύ σημαντικό ο χαρακτήρας μετά από κάποια αλληλεπίδραση με ένα αντικείμενο ή εχθρό να μην μπορεί να εκτοπιστεί εκτός της προκαθορισμένης διαδρομής του στον άξονα y. Όλο το περιβάλλον είναι χτισμένο στην ίδια "διαδρομή", οπότε αν ο ήρωας εκτοπιζόταν δεν θα μπορούσε πλέον να αλληλεπιδράσει με τα υπόλοιπα αντικείμενα. Εφόσον θα ήταν το πιο ωφέλιμο να υλοποιηθούν ταυτόχρονα αυτές οι αλλαγές σε όλους τους χαρακτήρες, αυτές θα γίνουν εντός της κλάσης bp actionchar base. Έπειτα στο character movement component στην ρύθμιση constrain to plane έγινε η μετατροπή της ρύθμισης αυτής 1 μόνο για τον άξονα y. Αυτό σημαίνει πως πλέον δεν είναι δυνατό για κανένα χαρακτήρα- παιδί αυτής της κλάσης να μετακινηθεί στον άξονα y. Επιπλέον μία ακόμη διόρθωση που έγινε ήταν πως λόγω της ύπαρξης του capsule component που ορίζει τα φυσικά όρια του παίχτη, αυτός δεν έπεφτε ομαλά στους κενούς χώρους του επιπέδου, αλλά κατά κάποιον τρόπο κόλλαγε λίγο στο πλάι του εδάφους και γλίστραγε περίεργα προς τα κάτω λόγω του περίεργου σχήματος της κάψουλας που τον περικλείει. Έτσι στο character movement component της actionchar base, όπως και πριν θα γίνει η αλλαγή μιας ρύθμισης για όλους έτσι ώστε να πέφτουν ομαλά όλοι οι χαρακτήρες από τα ύψη του επιπέδου. Στην ρύθμιση use flat base for floor checks επιλέγετε η αληθής τιμή. Έπειτα για την βελτίωση των μηχανισμών της αναπήδησης έγινε η αλλαγή κάποιων ακόμα ρυθμίσεων. Αυξήθηκε η ρύθμιση gravity scale στο 3(τύπου float - πολλαπλασιάζει το global gravity * 3), δηλαδή αυξήθηκε η έλξη της βαρύτητας και το jump z velocity ρυθμίστηκε στα 600cm/s, δηλαδή η δύναμη της αναπήδησης. Έπειτα στην ρύθμιση air control έγινε η επιλογή τιμής 1 , ώστε να έχει ο παίχτης περισσότερο έλεγχο της κίνησης στον αέρα. Η Air Control είναι μια τιμή μεταξύ 0 και 1. Όταν ένας χαρακτήρας πηδάει ή πέφτει και βρίσκεται στον αέρα, η κίνησή του επηρεάζεται κυρίως από τη βαρύτητα και οποιεσδήποτε αρχικές δυνάμεις που τον ώθησαν στον αέρα. Η μεταβλητή Air Control καθορίζει πόσο μπορεί ο παίκτης να αλλάξει την κατεύθυνση και την ταχύτητα του χαρακτήρα του ενώ βρίσκεται στον αέρα. Σε αυτό το σημείο θα γίνει εφαρμογή ενός μηχανισμού αναπήδησης όπου ο παίχτης θα επιλέγει αυτός πόσο ψηλά θέλει να αναπηδήσει ο χαρακτήρας αναλόγως του πόσο παρέμεινε πατημένο το πλήκτρο space. Στην κλάση bp player καθώς μόνο αυτή θα επηρεάσει ο νέος μηχανισμός γίνεται αλλαγή της μεταβλητής max hold time στην τιμή 0,4. Η μεταβλητή Max Hold Time στο Unreal Engine 5 (UE5) συνήθως αναφέρεται στη μέγιστη διάρκεια που μπορεί να κρατηθεί κάποιο input ή ενέργεια πριν αυτή ενεργοποιηθεί ή απελευθερωθεί. Έπειτα γίνεται υλοποίηση ώστε η συνάρτηση stop jumping να καλείτε και στην απελευθέρωση του πλήκτρου, αλλά και όταν ολοκληρωθεί το jump. Τέλος γίνετε η ρύθμιση του lateral friction στην τιμή 50. Η μεταβλητή Lateral Friction στο Unreal Engine 5 (UE5) αναφέρεται στην τριβή που εφαρμόζεται στις πλευρικές κινήσεις ενός χαρακτήρα ή αντικειμένου. Αυτή η τριβή επηρεάζει το πώς ο χαρακτήρας ή το αντικείμενο επιβραδύνεται όταν κινείται πλευρικά, όπως κατά την ολίσθηση ή την περιστροφή. Πλέον ο χαρακτήρας μπορεί να κινείτε σχεδόν ελεύθερα όταν βρίσκεται στον αέρα που δίνει στον παίχτη πολλές επιλογές και ελευθερία κινήσεων.

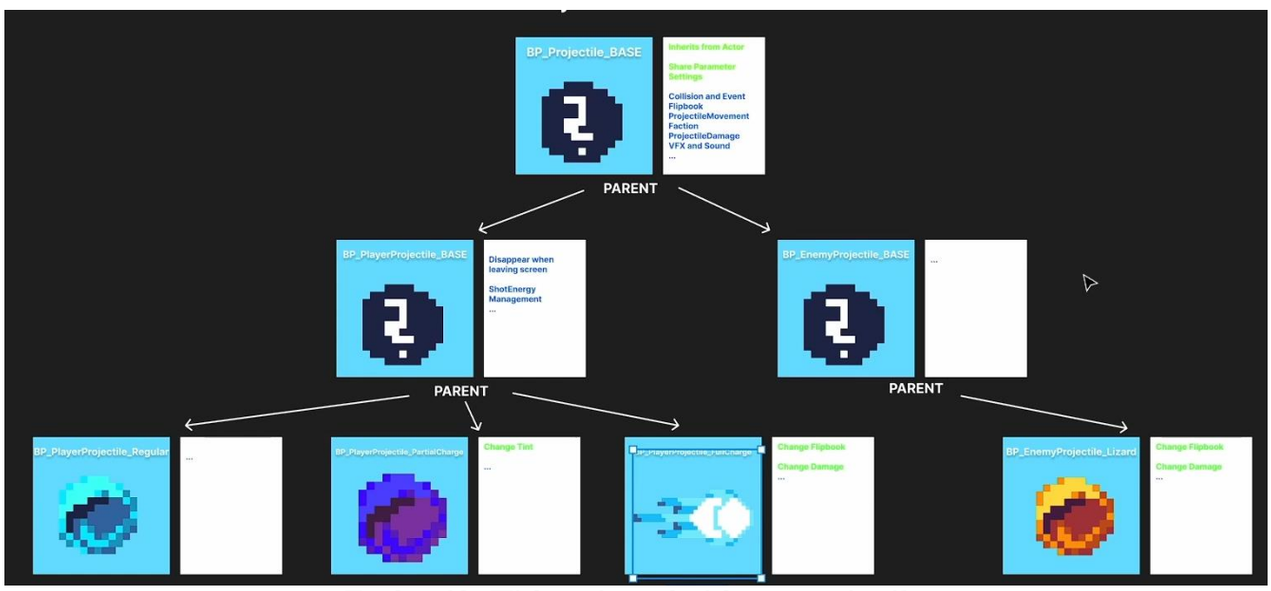
4.2 Δημιουργία animation και μηχανισμού πυροβολισμού

Φυσικά το πρώτο πράγμα για την δημιουργία των εφέ πυροβολισμού και την σφαιρών είναι η εισαγωγή των spite και η δημιουργία animation για τις σφαίρες που πυροβολεί ο κύριος χαρακτήρας, αλλά και ένας από τους εχθρούς. Με τον ίδιο τρόπο που αναφέρθηκε στην παραπάνω υποενότητα και δεν χρειάζεται να επαναληφθεί ξανά, έγινε η εισαγωγή των sprite των σφαιριδίων (projectiles) και η δημιουργία των flipbook. Το πρώτο βήμα είναι η ρύθμιση της ταχύτητας αναπαραγωγής των flipbook. Το πρώτο animation είναι αυτό της πρόσκρουσης με εχθρούς ή αντικείμενα και ορίστηκε στα 13 καρέ ανά δευτερόλεπτο (frames per second). Έπειτα το animations για την φόρτωση βολής του παίχτη, το οποίο θα είναι ένας μηχανισμός όπου ο παίχτης μπορεί να ενδυναμώσει την δύναμη των βολών του, ρυθμίστηκε και αυτό στα 13fps. Το εφέ της πορείας της βολής ρυθμίστηκε στα 9fps και η μπάλα φωτιάς του εχθρού στα 9 επίσης. Τέλος η έκρηξη που θα χρησιμοποιηθεί ως εφέ θανάτου για τους εχθρούς ρυθμίστηκε στα 13fps. Αφού έληξε η διαδικασία δημιουργίας των εφέ, το επόμενο βήμα είναι η δημιουργία της λογικής με την οποία θα λειτουργούν οι σφαίρες και για αυτό το σκοπό θα γίνει χρήση πάλι της κληρονομικότητας. Από κάτω στην εικόνα 17 δίνετε η λογική η οποία θα ακολουθηθεί για την δημιουργία των κλάσεων των projectiles, εχθρικών και φιλικών.



Εικόνα 17: Κληρονομικότητα κλάσεων projectile

Η αρχική κλάση θα κληρονομεί από την κλάση Actor του UE5 και θα έχει στοιχεία, όπως τα animation για την σύγκρουση και άλλα γεγονότα, το projectile movement component, όπου θα γίνεται η ρύθμιση της ταχύτητας, την μεταβλητή faction που θα ορίζει εάν πρόκειται για εχθρικά ή φιλικά πυρά, την ζημία που προκαλεί η σφαίρα, τον ήχο και άλλα. Θα έχει δύο κλάσεις παιδιά, την player projectile base και την enemy projectile base, για τις φιλικές βολές και τις εχθρικές αντίστοιχα, όπου εκεί θα προστεθούν επιπλέον αντικείμενα, όπως μια μεταβλητή για το όριο βολών του παίχτη, μία μέθοδο για την αναπλήρωσή τους και μία μέθοδο για την εξαφάνιση των βολών όταν αυτές βγαίνουν από τα όρια της οθόνης. Όλα αυτά βοηθούν και στην μείωση των υπολογιστικών πόρων που χρησιμοποιεί το παιχνίδι, καθώς και για την εξισορρόπηση της δυσκολίας. Τέλος θα χρησιμοποιηθούν άλλες 4 κλάσεις παιδιά, όπως φαίνονται στην εικόνα 18, τρεις από τις οποίες θα είναι για τις βολές του παίχτη και μία για τους εχθρούς που θα αναλυθούν παρακάτω.



Εικόνα 18: Πλήρης λογική κλάσεων projectiles

Επομένως δημιουργήθηκαν οι φάκελοι blueprints → projectiles → player , enemy , bp_projectile_base (Κλάση εντός του φακέλου projectiles). Έπειτα δημιουργήθηκαν οι κλάσεις παιδιά bp playerprojectile base στον φάκελο player και η κλάση bp playerprojectile regular. Στο επόμενα βήμα θα γίνει η ρύθμιση της κλάσης bp_projectile_base που θα είναι ο γονέας όλων των υπολοίπων και θα περιέχει όλες τις βασικές ιδιότητες που θα μοιράζονται οι κλάσεις παιδιά. Καταρχάς θα γίνει η προσθήκη του projectilemovement component στην κατηγορία των component της κλάσης, το οποίο είναι ένα component που παρέχεται έτοιμο από την μηχανή, καθώς είναι ένα πάρα πολύ συνηθισμένο component που υπάρχει σε όλα τα παιχνίδια βολών. Για να δουλέψει όμως χρειάζεται όπως όλα τα αντικείμενα να έχει κάποια φυσικά όρια, οπότε στον viewport editor θα γίνει η προσθήκη ενός sphere collision, ένα είδος ανιχνευτή (collision) που χρησιμοποιείται για να εντοπίσει τις συγκρούσεις μεταξύ αντικειμένων στον κόσμο του παιχνιδιού με σχήμα σφαίρας. Έπειτα αφού δημιουργήθηκε το καλούπι της σφαίρας στη συνέχεια πρέπει να γίνει η προσθήκη των animation, οπότε γίνεται η προσθήκη ενός flipbook στην σφαίρα από τον editor, συγκεκριμένα το player shot loop flipbook. Αφού το sphere component θα είναι υπεύθυνο για όλους τους μηχανισμούς σύγκρουσης γίνεται ορισμός της ρύθμισης collision στο flipbook από μπλοκάρισμα στα πάντα, στην επιλογή μηδενικής σύγκρουσης. Έπειτα γίνεται ρύθμιση της περιμέτρου του σφαιριδίου έτσι ώστε αυτό να έχει το ίδιο μέγεθος με το sprite της σφαίρας, αλλιώς η περιοχή σύγκρουσης δεν θα ήταν ίδια με τη περίμετρο του sprite και δεν θα έβγαζε κανένα νόημα. Στη συνέχεια πρέπει να γίνει η αφαίρεση της βαρύτητας από το projectile movement component γιατί η σφαίρα θα πέφτει πολύ γρήγορα προς τα κάτω μετά την απελευθέρωσή της από το όπλο που δεν είναι επιθυμητό. Έτσι γίνεται ρύθμιση της βαρύτητας στην τιμή 0. Επίσης για να μπορεί η σφαίρα να ταξιδεύει σε ευθεία πορεία στον άξονα x γίνεται ο ορισμός της τιμής initial speed (αρχικής ταχύτητας) σε 1000 και του max speed (μέγιστης ταχύτητας) σε 1000. Επιπλέον για να καταστρέφεται η βολή με το που συναντήσει έναν τοίχο χρειάζεται η δημιουργία ενός φίλτρου συγκρούσεων. Όλοι οι τοίχοι θα έχουν ρύθμιση σύγκρουσης ορισμένη σε block all dynamic, δηλαδή μπλοκάρισμα όλων των αντικειμένων οπότε δεν χρειάζεται να πειραχτούν οι ρυθμίσεις αυτές. Οπότε στις ρυθμίσεις τις σφαίρας γίνεται επιλογή της ρύθμισης block all dynamic. Στη συνέχεια γίνεται η χειροκίνητη δημιουργία ενός γεγονότος που ξεκινάει με την σύγκρουση της σφαίρας με άλλο αντικείμενο που έχει ικανότητα σύγκρουσης. Με την εκκίνηση του γεγονότος έπειτα καλείται η συνάρτηση destroy actor που καταστρέφει το αντικείμενο της κλάσης από τον κόσμο του παιχνιδιού, δηλαδή την σφαίρα.

// Συνάρτηση για την σύγκρουση

// OnOverlapBegin

ΣΥΝΑΡΤΗΣΗ OnOverlapBegin(OverlappedComp, OtherActor, OtherComp, OtherBodyIndex, bFromSweep, SweepResult)

ΑΝ OtherActor υπάρχει ΚΑΙ (OtherActor ΔΕΝ είναι αυτό το αντικείμενο) ΤΟΤΕ

// Καταστροφή αυτού του αντικειμένου

Καταστροφή()

ΤΕΛΟΣ ΑΝ

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

Στην συνέχεια θα γίνει η δημιουργία του flipbook της έκρηξης σφαίρας κατά την σύγκρουση με άλλο αντικείμενο. Αυτό θα πραγματοποιηθεί με την δημιουργία ενός νέου blueprint το οποίο θα παίζει ένα νέο flipbook και όταν το flibook σταματήσει να παίζει θα αυτοκαταστραφεί, δηλαδή μόλις χτυπήσει με κάποιο αντικείμενο θα παιχθεί το animation καταστροφής και η σφαίρα θα εξαφανιστεί από τον χώρο του παιχνιδιού. Γίνεται η δημιουργία Blueprints → Vfx folder → new blueprint class BP_VFX_Base. Μετά την δημιουργία της κλάσης γίνεται προσθήκη του play shoot hit flipbook. Εφόσον πρόκειται για ένα εφέ γίνεται η απενεργοποίηση όλων των συγκρούσεων με τον ίδιο τρόπο που έχει αναφερθεί προηγουμένως με τη ρύθμιση no collision στο fipbook. Στη συνέχεια γίνεται στο γεγονός begin play, θα δημιουργηθεί η συνάρτηση set looping η οποία ορίζει την τιμή Looping του blueprint σε ψευδή. Αυτό θα σταματήσει την συνεχή επανάληψη του animation, καθώς αυτό πρέπει να πραγματοποιείτε μόνο μία φορά κατά την καταστροφή της σφαίρας. Έπειτα γίνεται η δημιουργία ενός νέου γεγονότος που εκτελείτε με την λήξη του animation του flipbook, και δημιουργείτε η συνάρτηση destroy actor που καταστρέφει την σφαίρα με το που γίνει το animation. Στη συνέχεια γίνεται η δημιουργία του παιδιού της κλάσης vfx_base με όνομα BP_VFX_Shothit. Ώστε να γίνει η χρήση αυτού στον κόσμο γιατί η βασική κλάση

μπορεί να ξανά αλλάξει στην πορεία. Εφόσον δημιουργήθηκε η κλάση καταστροφής της σφαίρας, στη συνέχεια θα γίνει η κλήση της από την κλάση projectile base που χειρίζεται όλες τις βασικές λειτουργίες. Οπότε στο γεγονός on component hit που έχει ήδη δημιουργηθεί καλείτε η συνάρτηση spawn actor που εμφανίζει την bp vfx shot hit στη θέση που ήταν το projectile της κλάσης base projectile με παραμέτρους την τοποθεσία της bp projectile base και το rotation, δηλαδή την φορά της κατεύθυνσής της. Τέλος ορίζεται Always spawn, ignore collisions (να εμφανίζεται πάντα και να αγνοεί τις συγκρούσεις) στο collision handling override για να εξασφαλιστεί η ομαλή λειτουργία.

```
// ReplaceProjectile
ΣΥΝΑΡΤΗΣΗ ReplaceProjectile(OldProjectile, NewProjectileClass)
    ΑΝ OldProjectile ΔΕΝ υπάρχει Ή NewProjectileClass ΔΕΝ υπάρχει ΤΟΤΕ
        ΕΠΙΣΤΡΟΦΗ
    ΤΕΛΟΣ ΑΝ

    // Λήψη της θέσης και περιστροφής του παλιού projectile
    OldLocation = OldProjectile.ΠάρεΘέση()
    OldRotation = OldProjectile.ΠάρεΠεριστροφή()

    // Δημιουργία του νέου projectile
    SpawnParams.ΧειρισμόςΣύγκρουσης = ΠάνταΔημιουργία
    NewProjectile = ΠάρεΚόσμο().ΔημιούργησεActor(NewProjectileClass, OldLocation, OldRotation, SpawnParams)

    // Αντικατάσταση του παλιού projectile
    ΑΝ NewProjectile υπάρχει ΤΟΤΕ
        OldProjectile.Καταστροφή()
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Για το επόμενα βήμα τις εργασίας υλοποιήθηκε ο μηχανισμός του όπλου. Προηγουμένως δημιουργήθηκαν όλα τα κατάλληλα animation για τις σφαίρες και την μηχανική πρόσκρουσης με αντικείμενα και εχθρούς, αλλά ακόμα δεν έχει υλοποιηθεί η είσοδος του πληκτρολογίου για ανάθεση πλήκτρου στην λειτουργία του πυροβολισμού, καθώς δεν υπάρχει ακόμα κάποιο σύστημα πρόκλησης ζημιάς. Ακόμα παραμένουν να προγραμματιστούν και η ειδική ενέργεια που θα μπορεί να πραγματοποιήσει ο παίχτης με τη επιφορτισμένη βολή. Άρα για αρχή πρέπει να δεθεί ένα πλήκτρο με την κίνηση, οπότε στον φάκελο input → actions θα δημιουργηθεί μία νέα κλάση τύπου input action με όνομα IA_Shoot για τον πυροβολισμό, τύπου digital boolean. Οπότε έπειτα στο IMC_ Action όπου γίνονται όλες οι συνδέσεις των πλήκτρων, όπως έγινε και προηγουμένως για τις κινήσεις jump και move, τώρα κάνουμε Input ένα νέο key mapping (ανάθεση πλήκτρων) και εισάγεται η κίνηση IA_Shoot με ανάθεση στο πλήκτρο j. Έπειτα στην κλάση bp_player του βασικού χαρακτήρα προστίθεσαι το enhancedinputAction IA_Shoot . Μετά χρησιμοποιήθηκε η συνάρτηση spawn actor from class, η οποία επιτρέπει την εμφάνιση ενός στιγμιότυπου οποιασδήποτε άλλης κλάσης έχει δημιουργηθεί και έτσι εμφανίζεται το βασικό projectile πυροβολισμού , που είναι το απλό animation της σφαίρας. Όπως και πριν αυτή η συνάρτηση έχει παραμέτρους την τοποθεσία, την περιστροφή και το μέγεθος του αντικειμένου της κλάσης που θα εμφανιστεί. Δεν θα αλλαχθεί το scale, προφανώς για την τοποθεσία εμφάνισής τις σφαίρας επιλέχθηκε η τοποθεσία του παίχτη το οποίο δεν είναι απολύτως σωστό, καθώς η σφαίρα πρέπει μα εμφανίζεται από το όπλο. Ένα πρόβλημα που θα αντιμετωπιστεί αργότερα. Επίσης για την φορά της σφαίρας επιλέχθηκε να είναι ίδια με την φορά του παίχτη τη στιγμή που αυτός πυροβόλησε. Μετά από αυτήν την υλοποίηση κατά τη δοκιμή του πυροβολισμού υπήρχε το πρόβλημα πως ο

παίχτης μπορούσε να πυροβολήσει τον εαυτό του και μετατοπίζεται από την θέση του όταν συμβαίνει αυτό. Οπότε για να λυθεί το πρόβλημα τις μετατόπισης του παίχτη από τις σφαίρες, εντός της κλάσης bp_projectile_base στις ρυθμίσεις σύγκρουσης γίνεται η δημιουργία μίας χειροκίνητης λίστας ρυθμίσεων που είναι μία από τις λειτουργίες που προσφέρει το UE5 για την διευκόλυνση των προγραμματιστών. Οπότε στο νέο πακέτο ρυθμίσεων που δημιουργήθηκε μπήκαν οι ρυθμίσεις, οι σφαίρες να μπορούν να περνάνε μέσα από άλλες σφαίρες αλλά και από τον παίχτη και να συγκρούονται με όλα τα άλλα στοιχεία του περιβάλλοντος. Σε αυτό το σημείο επειδή ακόμα δεν έχει υλοποιηθεί η μετάβαση στο animation του πυροβολισμού για τον κυρίως παίχτη, αυτό θα είναι το επόμενο βήμα της εργασίας. Για υπενθύμιση προς το παρών έχουν δημιουργηθεί μόνο οι καταστάσεις της αδράνειας, του τρεξίματος, της αναπήδησης και της πτώσης. Οπότε πάλι στην κλάση bp player θα δημιουργηθεί μία νέα μεταβλητή με όνομα is shooting τύπου boolean με προεπιλεγμένη τιμή την ψευδή, η οποία με το που ενεργοποιηθεί το input action σύστημα που δημιουργήθηκε προηγουμένως θα αλλάξει η τιμή της σε αληθής, δηλαδή ο παίχτης πυροβολεί. Στην συνέχεια στο ABP_Player που είναι η κλάση με το δένδρο καταστάσεων που είχε δημιουργηθεί προηγουμένως για τον παίχτη θα προστεθεί στην κατάσταση idle η συνθήκη: Αν is shooting είναι true, τότε παίξε το animation του πυροβολισμού, αλλιώς παίξε το idle. Ακριβώς η ίδια συνθήκη θα προστεθεί στην κατάσταση run αντιστοίχως και στην κατάσταση jump, με την διαφορά ότι στην κατάσταση jump θα παίζει το jump shoot animation και όχι το απλό shoot animation που είναι διαφορετικά μεταξύ τους. Το πρόβλημα όμως είναι ότι δεν ορίστηκε το is shooting σε ψευδές αφού ο παίχτης σταμάτησε να πυροβολεί, οπότε πάλι στην κλάση του παίχτη στο shoot input action θα οριστεί η συνθήκη πως όταν τελειώσει αυτή η πράξη η μεταβλητή is shooting θα γίνει ψευδής. Εδώ θα δημιουργηθεί μία συνάρτηση με retriggerable delay η οποία αφού θα τελειώνει το input action μετά από 0,4 δεύτερα θα γίνεται η αλλαγή του is shooting, ώστε να μην είναι άμεση η αλλαγή των animation από πυροβολισμό σε άλλη κατάσταση και να είναι πιο ευχάριστο οπτικά για τον παίχτη. Έχει τη δυνατότητα επανεκκίνησης της καθυστέρησης αν ο κόμβος ενεργοποιηθεί ξανά πριν λήξει η προηγούμενη καθυστέρηση. Αυτό είναι ιδιαίτερα χρήσιμο σε περιπτώσεις όπου χρειάζεται να γίνει επαναφορά του χρονομέτρου κάθε φορά που πυροβολεί ο παίχτης. Η μεταβλητή float της χρονικής καθυστέρησης ονομάστηκε Shoot anim duration. Το επόμενο πρόβλημα που πρέπει να λυθεί είναι η εμφάνιση της σφαίρας στο σωστό σημείο πάνω στο όπλο του παίχτη. Πηγαίνοντας στο bp player class το UE5 επιτρέπει την ένταξη scene components που είναι ένα βασικό συστατικό που μπορεί να προσδιορίσει έναν μετασχηματισμό (θέση, περιστροφή και κλίμακα) στον κόσμο του παιχνιδιού και θα μετονομαστεί σε ShootPos. Το scene αυτό θα τοποθετηθεί ακριβώς πάνω στο όπλο του παίχτη και η τοποθεσία αυτού του scene θα χρησιμοποιηθεί για την εμφάνιση της σφαίρας στην συνάρτηση spawn actor και θα αντικαταστήσει την προηγούμενη παράμετρο που είχε χρησιμοποιηθεί η γενική θέση του παίχτη ως σημείο αναφοράς. Έτσι πλέον ο παίχτης πυροβολεί με το όπλο του επιτυχώς.

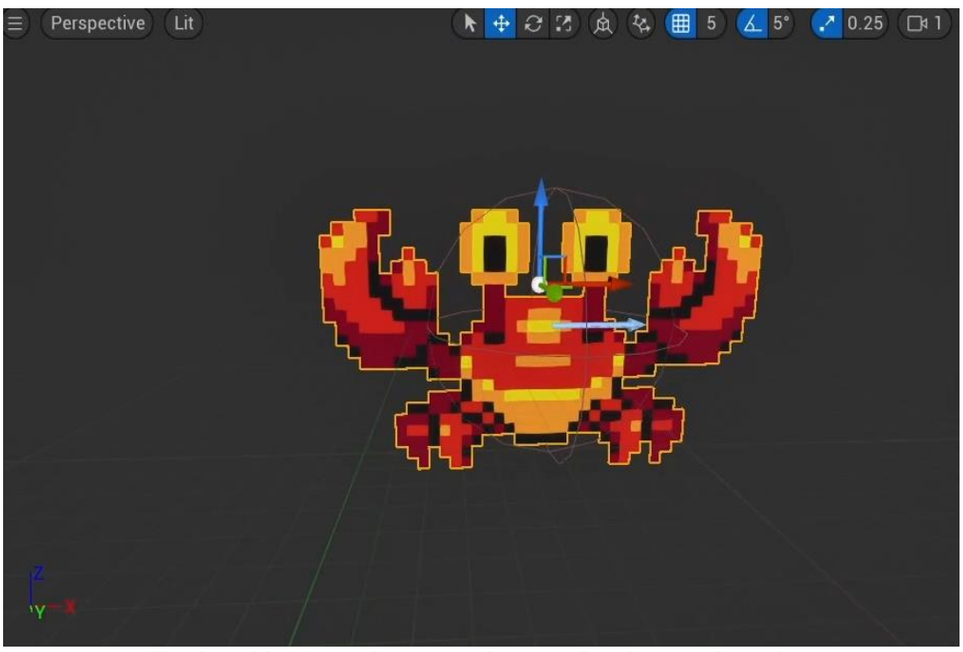
Τελικά θα ενσωματωθεί και μία ακόμα λειτουργία στον μηχανισμό πυροβολισμού του παίχτη με σκοπό να αυξηθεί η δυσκολία του παιχνιδιού. Η λειτουργία αυτή θα απαγορεύει στον παίχτη να μπορεί να ρίξει πάνω από τρεις ενεργές βολές συγχρόνως, αλλά θα διαλύει τις βολές που ξεφεύγουν από το όριο της οθόνης μειώνοντας έτσι το βελεηνεκές τους. Επομένως στην κλάση bp_playerprojectile_base θα εντάξουμε την λειτουργία στο πρόγραμμα να ελέγχει σε κάθε καρέ εάν οι βολές έχουν ξεφύγει από τα όρια της οθόνης και τις καταστρέφει. Οπότε στο γεγονός on tick που δίνεται αυτόματα από τις βιβλιοθήκες της UE5 χρησιμοποιώντας την συνάρτηση get player controller γίνεται αναφορά στην κλάση αυτή και από αυτή την κλάση με την συνάρτησή της convert world location to screen location γίνεται η μετατροπή μιας θέσης στον κόσμο σε θέση στην οθόνη. Αυτή η συνάρτηση παίρνει μια θέση στον κόσμο και την μετατρέπει σε συντεταγμένες της οθόνης. Ως επιπλέον παράμετρος πρέπει να δοθεί και η τοποθεσία του αντικείμενου player projectile στον κόσμο. Δίνονται σαν έξοδος η συντεταγμένη του άξονα x της σφαίρας στην οθόνη και υπό την συνθήκη πως αν είναι μικρότερη ή ίση με το 0, τότε καταστρέφουμε το αντικείμενο με την συνάρτηση destroy actor που έχει χρησιμοποιηθεί και νωρίτερα. Αυτό καλύπτει την αριστερή πλευρά της οθόνης , όμως για την δεξιά πλευρά πρέπει να εντοπιστεί το μέγεθος του viewport editor, καθώς αυτό μπορεί να συρρικνωθεί ή να μεγαλώσει αναλόγως των προτιμήσεων του παίχτη. Έτσι γίνεται ταυτόχρονα και ο έλεγχος αν η συντεταγμένη x είναι μεγαλύτερη ή ίση της τιμής x του viewport editor, όπου αν είμαι αληθές πάλι καταστρέφεται η σφαίρα. Έπειτα για να μην εμφανίζεται το εφέ καταστροφής της σφαίρας όταν φεύγει από τα όρια της οθόνης, όπως αν χτύπαγε έναν εχθρό, αλλά απλός να εξαφανίζεται αρκεί να γίνει αλλαγή στην σειρά του κώδικα εντός της κλάσης projectile_base. Έτσι κατά την σύγκρουση αντί να καταστρέφεται η σφαίρα και μετά να παίζει το animation, τώρα πρώτα θα παίζει το animations και μετά θα καταστρέφεται ο ηθοποιός. Εφόσον ποτέ δεν γίνεται σύγκρουση δεν θα παιχτεί ποτέ animation καταστροφής , ενώ πριν τον animation της καταστροφής ήταν συνδεδεμένο με την καταστροφή της σφαίρας η οποία συνέβαινε και κατά την έξοδό της από την οθόνη.

Ο τελευταίος μηχανισμός που θα δημιουργηθεί είναι ο παίχτης να έχει έως τρεις ενεργές βολές κάθε φορά ή αντί αυτού να μπορεί να ρίξει μία μεγάλη βολή ξοδεύοντας όλη την ενέργειά του. Έτσι οι βολές του παίχτη θα αντιπροσωπεύονται από την μεταβλητή της ενέργειας που του απομένει. Έτσι στο Bp player γίνεται δημιουργία της μεταβλητής ShotEnergy τύπου integer με τιμή 3. Άρα στον μηχανισμό του πυροβολισμού στην αρχή θα προστεθεί η συνθήκη πως ο παίχτης για να πυροβολήσει θα πρέπει το ShotEnergy να είναι μεγαλύτερο του μηδενός. Εφόσον αυτό είναι αληθές τότε μειώνεται η τιμή του ShotENergy κατά 1 που σημαίνει πως υπάρχει μία βολή ενεργή και τίθεται το is shooting σε true και συνεχίζει ο παλιός κώδικας που έχει ήδη αναλυθεί. Επιπρόσθετα θα δημιουργηθεί μία συνάρτηση για την αναπλήρωση της ενέργειας του παίχτη με όνομα RestoreShotEnergy, όπου απλός όταν καλείτε θα αυξάνεται η τιμή του shot energy κατά 1. Οπότε μετά πηγαίνοντας στην κλάση playerprojectile στο σημείο του κώδικα όπου καταστρέφεται η σφαίρα γίνεται αναφορά στην κλάση bp player και καλείται η συνάρτηση restoreshotenergy με αποτέλεσμα να αναπληρώνει ενέργεια ο παίχτης.

4.3 Δημιουργία συστήματος ζωής και λήψης ζημιάς

4.3.1 Δημιουργία βασικής κλάσης εχθρών

Σε αυτό το κεφάλαιο θα πραγματοποιηθεί η δημιουργία της βάσης για όλες τις κλάσης των εχθρών και ο πιο απλός αντίπαλος. Οπότε για το πρώτο βήμα θα γίνει η δημιουργία μίας κλάσης παιδί με όνομα bp_enemy_base η οποία θα κληρονομήσει όλα τα χαρακτηριστικά της από την bp_actiochar_base βάση της λογικής που έχει ήδη αναφερθεί. Στη συνέχεια θα γίνει από την bp_enemy_base η δημιουργία της κλάσης παιδί bp_enemy_crab και αμέσως θα προστεθεί το flipbook που περιέχει την κίνηση αδράνειας του πρώτου εχθρού του παιχνιδιού. Όπως γίνεται και σε κάθε κλάση χαρακτήρα προσαρμόζεται το component της κάψουλας ώστε να περικλείει επαρκώς τον χαρακτήρα. Ως υπενθύμιση η συνιστώσα κάψουλας (Capsule Component) στην Unreal Engine 5 είναι ένα βασικό στοιχείο που χρησιμοποιείται κυρίως για την αναπαράσταση απλών σχήματος σύγκρουσης (collision shapes) και για την κίνηση χαρακτήρων. Οι συνιστώσες κάψουλας είναι αποτελεσματικές και χρησιμοποιούνται συχνά για να ορίσουν τα όρια των χαρακτήρων ή αντικειμένων που χρειάζονται απλά και αποτελεσματικά σχήματα σύγκρουσης.



Εικόνα 19: Απεικόνιση πρώτου εχθρού και κάψουλας

Πολύ σημαντικό είναι να οριστεί, όπως και όλοι οι χαρακτήρες η θέση του στο σημείο μηδέν του άξονα z για λόγους που έχουν ήδη αναφερθεί και με τη κάνοντας τεστ να βρεθούν οι κατάλληλες συνταγμένες, ώστε ο χαρακτήρας να ακουμπάει στο έδαφος με ρεαλιστικό τρόπο. Ένα από τα προβλήματα που εμφανίστηκαν είναι πως όταν ο παίχτης βρίσκεται στο ίδιο σημείο με τον εχθρό αυτό, το παιχνίδι δεν μπορεί να αναγνωρίσει ποιος έχει προτεραιότητα για να φανεί στο προσκήνιο. Έτσι ένας εύκολος τρόπος αντιμετώπισης ατού του προβλήματος για

να είναι ο παίχτης πάντα αυτός που φαίνεται στο προσκήνιο κατά την αλληλεπίδραση με εχθρούς επιτυγχάνεται με την ελάχιστη μετακίνησή του στον άξονα z κατά 0,05 μονάδες. Έτσι βρίσκεται ελαφρώς πιο μπροστά από τους εχθρούς, αλλά παρόλα αυτά μπορεί ακόμα να αλληλοεπιδράει επιτυχώς με αυτούς. Προς το παρών οι σφαίρες του παίχτη δεν μπορούν να χτυπήσουν τον εχθρό και περνούν από μέσα του. Έτσι στην κλάση projectile_base γίνεται δημιουργία ενός event που ενεργοποιείτε κατά την διάτρηση ενός στόχου. Παρακάτω θα γίνει η επεξήγηση των events στο UE5. Στην Unreal Engine 5, τα events (συμβάντα) είναι ένας τρόπος επικοινωνίας μεταξύ των αντικειμένων. Επιτρέπουν την ενεργοποίηση λειτουργιών σε απόκριση συγκεκριμένων συμβάντων, όπως η αλληλεπίδραση του χρήστη, οι φυσικές συγκρούσεις, και άλλες αλλαγές κατάστασης. Τα events μπορούν να δημιουργηθούν και να χρησιμοποιηθούν με διάφορους τρόπους, τόσο σε Blueprints όσο και σε C++. Στην Unreal Engine 5 (UE5), τα events δεν είναι ακριβώς συναρτήσεις, αλλά είναι στενά συνδεδεμένα με συναρτήσεις. Τα events λειτουργούν σαν μηχανισμοί ειδοποίησης που μπορούν να ενεργοποιήσουν συναρτήσεις ή μεθόδους (callbacks) όταν συμβαίνουν συγκεκριμένα γεγονότα. Παρακάτω θα εξηγηθεί πώς λειτουργούν τα events και πώς συνδέονται με συναρτήσεις. Τα events στην UE5 υλοποιούνται χρησιμοποιώντας delegates. Ένας delegate είναι ένα αντικείμενο που κρατάει αναφορές σε μία ή περισσότερες συναρτήσεις. Οι delegates μπορούν να εκπέμπουν σήματα (broadcast) που καλούν όλες τις συνδεδεμένες συναρτήσεις (callbacks).

Παράδειγμα Χρήσης Events και Συναρτήσεων:

Δημιουργία ενός Event στο Actor:

```
// YourActor.h
ΚΛΑΣΗ YourActor ΚΛΗΡΟΝΟΜΕΙ από Actor
    ΣΥΝΑΡΤΗΣΗ YourActor()
    // Δημιουργία του event χρησιμοποιώντας delegate
    ΔΗΛΩΣΗ_EVENT(YourActor, FCustomEvent)
    FCustomEvent& ΠάρεCustomEvent() { επιστροφή CustomEvent }

    ΣΥΝΑΡΤΗΣΗ TriggerEvent() // Συνάρτηση που θα ενεργοποιήσει το event

ΙΔΙΩΤΙΚΑ:
    FCustomEvent CustomEvent
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Υλοποίηση του Event και της Συνάρτησης:
// YourActor.cpp
ΣΥΝΑΡΤΗΣΗ YourActor()
    // Οποιαδήποτε αρχική ρύθμιση μπορεί να γίνει εδώ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

ΣΥΝΑΡΤΗΣΗ TriggerEvent()
    // Κάποιες ενέργειες
    // Ενεργοποίηση του event
    CustomEvent.Μετάδοση()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Οπότε θα δημιουργηθεί ένα event που εκκινείτε τη στιγμή που δύο αντικείμενα κλάσεων κάνουν Overlap. Το "overlap" αναφέρεται σε ένα είδος σύγκρουσης όπου δύο ή περισσότερα αντικείμενα καταλαμβάνουν τον ίδιο χώρο χωρίς να προκαλούν αλληλεπίδραση που να αλλάζει την πορεία ή τη θέση τους. Το overlap χρησιμοποιείται συχνά για να εντοπίσουμε αν ένα αντικείμενο έχει εισέλθει σε μια συγκεκριμένη περιοχή ή έχει έρθει σε επαφή με ένα άλλο αντικείμενο, χωρίς να εφαρμοστούν δυνάμεις ή μετατοπίσεις. Οπότε γίνεται ο προγραμματισμός έτσι ώστε όταν συμβαίνει αυτό το γεγονός να καταστρέφονται όλοι οι χαρακτήρες τύπου action_char με τους οποίους γίνετε το overlap. Το πρόβλημα εδώ είναι όμως πως έτσι μπορεί να αυτοκαταστρέφεται και ο ίδιος ο παίχτης με τις σφαίρες του, οπότε θα δημιουργηθεί ένας επιπλέον έλεγχος ομάδας με την χρήση της μεταβλητής Faction. Οπότε στην κλάση actionchar θα γίνει η δημιουργία της μεταβλητής Faction τύπου enumarator. Η τιμή enemy θα συμβολίζει τις εχθρικές μονάδες, ενώ η τιμή player θα απευθύνεται στον παίχτη. Για την υλοποίηση αυτής της λογικής γίνεται χρήση ενός enumerator. Σε έναν νέο φάκελο με όνομα Enums θα δημιουργηθεί ένα blueprint τύπου enum. Θα δημιουργηθούν δύο enumarators με ονόματα player και enemy. Έτσι θα τεθεί ως η αρχική τιμή του Faction ως player για τις κλάσεις bp_actionchar και bp_player και enemy για την bp_enemy_base. Στην κλάση bp_projectile_base θα δημιουργηθεί άλλη μία μεταβλητή faction τύπου enumerator με τιμή enemy. Στην κλάση playerprojectile θα οριστεί η μεταβλητή faction σε player. Γίνετε υπενθύμιση πως αυτή η κλάση είναι παιδί της projectile_base που έχει τιμή faction enemy και για αυτό πρέπει να οριστεί αυτή η αλλαγή. Οπότε πίσω στην κλάση projectile_base στο overlap event γίνετε σύγκριση μεταξύ του faction της σφαίρας και το faction της κλάσης του εχθρού και εφαρμόζεται η συνθήκη, αν το faction της σφαίρας είναι ίδιο με του εχθρού να μην συμβεί τίποτα αλλιώς να καταστραφεί ο εχθρός. Επίσης για την βελτιστοποίηση της φυσικότητας του παιχνιδιού μετά την καταστροφή του παίχτη γίνεται και καταστροφή του της σφαίρας, όμως δεν παίζει το εφέ καταστροφής της σφαίρας σε αυτή την περίπτωση, οπότε απλός με επανάληψη προηγούμενου κώδικα ρυθμίζεται πριν την καταστροφή της σφαίρας να παίζει και το κατάλληλο εφέ καταστροφής. Τέλος θα προστεθεί ένα νέο για την καταστροφή των εχθρών με τη δημιουργία της κλάσης BP_vfx_enemydestroyed και θα προστεθεί το flipbook της έκρηξης του εχθρού. Έπειτα στην κλάση enemy_base έχει γίνει σύνδεση της συνάρτησης της καταστροφής του εχθρού σε ένα event με όνομα destroyed. Στη συνέχεια γίνεται χρήση του event με όνομα destroyed και με κώδικα που έχει ήδη αναφερθεί με την συνάρτηση spawn actor θα γίνει η εμφάνιση του εφέ καταστροφής, δηλαδή με του που καταστραφεί ο εχθρός και γίνει κλήση η αυτού του event.

4.3.2 Ενσωμάτωση συστήματος ζωής

Προς το παρών όλοι οι εχθροί καταστρέφονται αμέσως με μονάχα μία σφαίρα, οπότε θα γίνει η δημιουργία ενός health component, δηλαδή ενός συστήματος ζωής που θα χρησιμοποιούν όλοι οι χαρακτήρες. Προς υπενθύμιση τα Components στην Unreal Engine 5 είναι βασικά δομικά στοιχεία που προστίθενται στους Actors για να επεκτείνουν τη λειτουργικότητά τους. Κάθε Actor μπορεί να αποτελείται από πολλαπλά Components, καθένα από τα οποία προσθέτει συγκεκριμένες ιδιότητες ή λειτουργίες στον Actor. Τα Components μπορούν να είναι οπτικά (όπως Static Mesh Components), φυσικά (όπως Collision Components), ή να παρέχουν άλλες δυνατότητες (όπως Audio Components). Θα γίνει δημιουργία ενός φάκελου με όνομα components και μέσα αυτόν θα δημιουργηθεί μια κλάση τύπου actor component με όνομα bpc_vitality. Στην κλάση actionchar_base θα γίνει προσθήκη αυτού του component στον editor μαζί με τα υπόλοιπα components της κλάσης. Οπότε σε αυτό το component θα δημιουργηθούν 3 μεταβλητές, δύο μεταβλητές MaxHealth και CurrentHealth που αντιπροσωπεύουν την μέγιστη και την παροντική ζωή του παίχτη και μία μεταβλητή IsDefeated τύπου Boolean που θα καθορίζει εάν ο χαρακτήρας είναι νεκρός ή όχι. Η μεταβλητή MaxHealth θα αρχικοποιηθεί με την τιμή δέκα. Έπειτα στο event έναρξης του παιχνιδιού για την κλάση αυτή που παρέχεται από την Ue5 θα οριστεί η μεταβλητή CurentHealth να πάρει την τιμή της μεταβλητής MaxHealth. Έτσι εάν χρειαστεί να γίνει κάποια τροποποίηση στην ζωή των χαρακτήρων, το μόνο που απαιτείται είναι η αλλαγή της μεταβλητής MaxHealth. Έτσι στην κλάση του εχθρού Bp_enemy_crab θα γίνει αλλαγή της τιμής MaxHealth σε 3 για αυτόν τον χαρακτήρα. Στη συνέχεια στην κλάση bp_projectile_base θα δημιουργηθεί μια συνάρτηση apply damage η οποία διαχειρίζεται την ζημία που προκαλούν οι σφαίρες. Έτσι γίνεται αντικατάσταση της συνάρτησης καταστροφής αντικειμένου στον κώδικα της κλάσης και χρησιμοποιείται η apply damage.

Παράδειγμα apply damage:

// Κάπου στον κώδικα, π.χ. σε μια συνάρτηση

ΣΥΝΑΡΤΗΣΗ CauseDamage()

```
    // Αναφορά στον στόχο (damageable actor)
    Target = ... // actor στόχος

    ΑΝ Target υπάρχει ΤΟΤΕ
        // Εφαρμογή 20 μονάδων ζημιάς στον στόχο
        ΕφαρμογήΖημιάς(Target, 20.0, καμία, αυτό, καμία)
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Η apply damage προγραμματίστηκε έτσι ώστε να προκαλείται μία μονάδας ζημιάς ανά σφαίρα. Επίσης δημιουργήθηκε η μεταβλητή τύπου float με όνομα projectileDamage που διαχειρίζεται τη ζημιά των βολών. Όμως τώρα πρέπει να δημιουργηθεί μία συνάρτηση που θα καταστρέφει τους εχθρούς εάν δεχθούν πολύ ζημιά, καθώς πλέον οι εχθροί δεν καταστρέφονται. Στην κλάση bp_actionChar_base γίνεται η δημιουργία ενός event με όνομα event any damage το οποίο καλείται με την εκτέλεση της συνάρτησης apply damage. Έτσι στο event αυτό θα δημιουργηθεί μία συνάρτηση με όνομα receive damage που δέχεται μία μεταβλητή με όνομα damage ως παράμετρο και αφαιρεί την τιμή του damage από την τιμή της μεταβλητής currentHealth. Έτσι πλέον μειώνεται η ζωή των αντιπάλων. Επιπλέον για να μην μπορεί να μειωθεί η ζωή ενός χαρακτήρα υπό του μηδενός στην κλάση bpc_vitality στην receive damage εφαρμόζεται και η συνθήκη πως το current health δεν μπορεί να μειωθεί παραπάνω εφόσον φτάσει στο μηδέν με την χρήση return max(damage,0); και πως δεν μπορεί μα ξεπεράσει την μέγιστη ζωή. Στη συνέχεια εφαρμόζεται η λογική ελέγχου εάν ο εχθρός έχει ηττηθεί ή όχι. Εάν η ζωή του εχθρού είναι μικρότερη ή ίση με το 0 μετά την αφαίρεση της ζημιάς, τότε η μεταβλητή isDefeated γίνεται αληθής. Έπειτα στο receive damage event γίνεται ο έλεγχος αν η μεταβλητή isdefeated είναι αληθής τότε καλείται η συνάρτηση destroy actor και καταστρέφεται ο χαρακτήρας. Πλέον οι εχθρικοί χαρακτήρες μπορούν να δεχθούν ζημία και να καταστραφούν όταν ξεμείνουν από ζωή.

4.3.3 Υλοποίηση εφέ λήψης ζημιάς χαρακτήρων

Η επόμενη εργασία που θα γίνει είναι η υλοποίηση του εφέ που θα συμβαίνει όταν ένας παίχτης δέχεται ζημιά. Το animation αυτό έχει ήδη εισαχθεί μαζί με όλα τα υπόλοιπα σε προηγούμενο βήμα, οπότε απομένει η υλοποίηση της λογικής. Το UE5 προσφέρει από μόνο του ένα είδος εφέ το οποίο μπορεί να χρησιμοποιηθεί ως μία ένδειξη όταν ο παίχτης δεχθεί ζημιά. Έτσι έγινε αντιγραφή της κλάσης defaultLitSprite material και της defaultSpriteMaterial , που αντιπροσωπεύουν το υλικό από το οποίο αποτελούνται οι χαρακτήρες, εντός ενός φακέλου με όνομα materials. Οι κλάσεις αυτές μετονομάστηκαν σε costumlit και costumunlit sprite material. Ξεκινώντας από την 1η γίνεται προσθήκη χρώματος προσθέτοντας ένα vector parameter με όνομα flashColor και συνδέοντάς στον στην είσοδο emissive color το οποίο αντιπροσωπεύει μία απόχρωση χρώματος που θα εκπέμπεται από αυτό το υλικό. Για το πρώτο υλικό γίνετε επιλογή ενός κίτρινου χρώματος. Όμως εάν χρησιμοποιηθεί αυτό το υλικό ως υλικό για τον παίχτη τότε ο παίχτης θα έχει μονίμως ένα κίτρινο χρώμα. Το επιθυμητό αποτέλεσμα είναι αυτό το χρώμα να εναλλάσσεται, δηλαδή να αναβοσβήνει πάνω στον παίχτη. Για να συμβεί αυτό γίνεται ο πολλαπλασιασμός του χρωματικού vector με μία μεταβλητή τύπου float με όνομα flashmultiplier. Η ίδια διαδικασία γίνετε και για το costumunlit sprite material. Στη συνέχεια από αυτά τα δύο material blueprints θα δημιουργηθούν δύο material instances με ονόματα mi_ costumunlitsprite_material και mi_ costumlitsprite_material. Στην Unreal Engine 5 (UE5), ένα "Material Instance" αναφέρεται σε μια παραλλαγή ενός βασικού υλικού που επιτρέπει την τροποποίηση των παραμέτρων του χωρίς να χρειάζεται να δημιουργηθεί ένα νέο υλικό από την αρχή. Με τα "Παράγωγα Υλικά" (Material Instances), οι χρήστες μπορούν να αλλάξουν παραμέτρους όπως χρώματα, υφές, διαφάνειες και άλλες ιδιότητες, παρέχοντας ευελιξία και εξοικονόμηση χρόνου κατά τη διαδικασία ανάπτυξης. Αυτό γίνεται διατηρώντας τον βασικό ορισμό του υλικού ανέπαφο και επιτρέποντας τροποποιήσεις μέσω των παραμέτρων που ορίζονται στο βασικό υλικό. Δηλαδή σαν δύο στιγμιότυπα

των κλάσεων των materials. Οπότε στην κλάση bp_actionchar_base θα γίνει χρήση στις ρυθμίσεις υλικού του sprite το mi_costumlitspritematerial. Έπειτα στο γεγονός εκκίνησης του παιχνιδιού που υπάρχει για όλες τις κλάσεις θα δημιουργηθεί η συνάρτηση με όνομα create dynamic material instance. Η παραπάνω συνάρτηση και η συνολική κλάση δημιουργούν και εφαρμόζουν ένα δυναμικό παράγωγο υλικό σε ένα Static Mesh Component και αλλάζουν δυναμικά το χρώμα του εκπεμπόμενου φωτός σε κάθε καρέ. Η κλάση ADynamicMaterialActor κληρονομεί από την AActor, που είναι η βασική κλάση για όλα τα αντικείμενα στον κόσμο του παιχνιδιού. Στον κατασκευαστή της κλάσης (ADynamicMaterialActor::ADynamicMaterialActor), δημιουργείται και αρχικοποιείται ένα StaticMeshComponent. Αυτό το εξάρτημα θα χρησιμοποιηθεί για να εμφανίσει το υλικό στο επίπεδο. Στη συνάρτηση BeginPlay, ελέγχεται αν υπάρχει ένα BaseMaterial (δηλαδή ένα υλικό που έχει οριστεί στον Editor). Αν υπάρχει, δημιουργείται ένα UMaterialInstanceDynamic από αυτό το υλικό και εφαρμόζεται στο StaticMeshComponent. Στη συνάρτηση Tick, που καλείται σε κάθε καρέ, αλλάζει το χρώμα του εκπεμπόμενου φωτός χρησιμοποιώντας τη συνάρτηση SetVectorParameterValue.Το νέο χρώμα καθορίζεται από τη συνάρτηση FLinearColor::MakeRandomColor(), που δημιουργεί ένα τυχαίο χρώμα.

Παράδειγμα:

Κατασκευαστής:

```
// Κατασκευαστής ADynamicMaterialActor
ΣΥΝΑΡΤΗΣΗ ADynamicMaterialActor()
    PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές
    MeshComponent = ΔημιουργίαDefaultSubobject("MeshComponent")
    RootComponent = MeshComponent
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// BeginPlay
ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()

    ΑΝ BaseMaterial υπάρχει ΤΟΤΕ
        DynamicMaterialInstance = ΔημιουργίαMaterialInstanceDynamic(BaseMaterial, αυτό)
        ΑΝ DynamicMaterialInstance υπάρχει ΤΟΤΕ
            MeshComponent.ΡύθμισεΥλικό(0, DynamicMaterialInstance)
        ΤΕΛΟΣ ΑΝ
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Tick
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)

    ΑΝ DynamicMaterialInstance υπάρχει ΤΟΤΕ
        NewColor = ΔημιουργίαΤυχαίουΧρώματος()
```

DynamicMaterialInstance.ΡύθμισεΤιμήΠαραμέτρου("EmissiveColor", NewColor)

ΤΕΛΟΣ ΑΝ

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

Η έξοδος αυτής της λειτουργίας είναι η δημιουργία και η εφαρμογή ενός δυναμικού παράγωγου υλικού σε ένα αντικείμενο (συγκεκριμένα σε ένα Static Mesh Component) και παράγει ένα αντικείμενο (object) με όνομα DynamicSpriteMat. Έπειτα θα δημιουργηθεί ένα event με όνομα triggerflash με μία συνάρτηση η οποία θα αλλάζει την τιμή της flashmultiplier και μία που θα αλλάζει την τιμή της flashcolor που είχαν δημιουργηθεί προηγουμένως για την ρύθμιση της έντασης και του χρώματος της κλάσης lit και Unlit material. Οπότε πίσω στο event begin play μετά την συνάρτηση create dynamic material instance θα δημιουργηθεί μία συνάρτηση με όνομα delay η οποία θα δίνει μία καθυστέρηση 1 δευτερολέπτων και έπειτα μία συνάρτηση με όνομα triggerflash η οποία ενεργοποιεί το event triggerflash που δημιουργήθηκε προηγουμένως και περνάει ως παραμέτρους τις τιμές 1 για το flashmultiplier και το χρώμα κόκκινο στον vector flashcolor. Όμως επειδή πρέπει ο χαρακτήρας να επιστρέψει στη συνέχεια στο αρχικό του χρώμα εντός του event trigger flash στο τέλος προστίθεται ένα timeline. Στην Unreal Engine 5 (UE5), το Timeline είναι ένα ισχυρό εργαλείο που επιτρέπει στους προγραμματιστές και στους σχεδιαστές να δημιουργούν σύνθετες κινούμενες εικόνες (animations) και να ελέγχουν τις παραμέτρους με βάση τον χρόνο. Το Timeline μπορεί να χρησιμοποιηθεί για να δημιουργήσει διακριτές κινήσεις, εφέ, και αλλαγές κατά τη διάρκεια μιας χρονικής περιόδου. Μετά το timeline καλείτε πάλι η συνάρτηση scalar parameter που είναι υπεύθυνη για τον φωτισμό του υλικού του παίχτη και παίρνει τιμές από μία συνάρτηση lerp. Η συνάρτηση Lerp (Linear Interpolation) χρησιμοποιείται για να υπολογίσει μια ενδιάμεση τιμή μεταξύ δύο τιμών με βάση ένα ποσοστό (Alpha). Η τιμή που επιστρέφεται είναι μια γραμμική μετάβαση από την αρχική τιμή προς την τελική τιμή, ανάλογα με το ποσοστό που καθορίζεται. Στην Unreal Engine 5, η συνάρτηση Lerp είναι συχνά χρησιμοποιούμενη για την ομαλή μετάβαση τιμών όπως θέσεις, χρώματα, ή άλλες παραμέτρους κατά τη διάρκεια του χρόνου. Ο τύπος της Lerp μπορεί να εφαρμοστεί σε διάφορες μορφές δεδομένων, όπως float, FVector, FLinearColor, κ.λπ.

Παράδειγμα lerp:

float StartValue = 0.0f;

float EndValue = 100.0f;

float Alpha = 0.5f; // 50% μεταξύ StartValue και EndValue

float InterpolatedValue = FMath::Lerp(StartValue, EndValue, Alpha);

Η τιμή alpha της lerp θα παίρνει τιμές από το timeline το οποίο θα ξεκινάει από το 0 με αρχική τιμή το 0 και τελειώνει μετά από ένα δεύτερο στην τιμή 1. Αυτό σημαίνει ότι μέσα σε ένα δευτερόλεπτο η έξοδος του timeline προοδευτικά με τον χρόνο θα παίρνει τιμές από το 0 μέχρι και το 1 και θα τις περνάει σαν τιμή alpha στην συνάρτηση lerp. Όπου το A value του lerp θα είναι η μεταβλητή flash multiplier και το B value θα είναι το 0. Που σημαίνει ότι η flashmultiplier θα μειώνεται και σε ένα δεύτερο θα γίνει ίση με το 0 με αποτέλεσμα να ξεθωριάσει ο φωτισμός του υλικού του παίχτη μέχρις ότου εξαφανιστεί δίνοντας έτσι ένα εφέ σαν να αναβοσβήνει κάθε φορά που δέχεται ζημιά. Έπειτα εφαρμόζεται μία συνάρτηση με όνομα play rate πριν από το timeline που δέχεται μία μεταβλητή με όνομα fadespriteflash που περιέχει έναν πολλαπλασιαστή, για να γίνει η επιτάχυνση του timeline και να αυξηθεί η ταχύτητα αναπαραγωγής κατά 5 φορές παραπάνω. Τέλος διαγράφεται ο κώδικας της συνάρτησης delay και της event trigger flash από το begin play, γιατί δεν θέλουμε να αναβοσβήνει ο παίχτης στην αρχή του παιχνιδιού, αλλά όταν δέχεται ζημιά. Οπότε στην κλάση actionchar_base στο γεγονός με όνομα anydamage που έχει δημιουργηθεί νωρίτερα στη συνθήκη όπου γίνεται ο έλεγχος της μεταβλητής isdefeated αν το isdefeated είναι αληθές υλοποιείτε η συνάρτηση destroy actor, όμως σε περίπτωση που είναι ψευδής θα προστεθεί εδώ η συνάρτηση triggerflash που ενεργοποιεί το event trigger flash και πραγματοποιείτε το εφέ του παίχτη. Με είσοδο το κίτρινο χρώμα και flash multiplier 10.

4.3.4 Σύστημα λήψης ζημιάς παίχτη

Αφού υλοποιήθηκε ο μηχανισμός λήψης ζημιάς από τον παίχτη, τώρα πρέπει να γίνει η υλοποίηση να μπορεί να δεχθεί και ο παίχτης ζημιά αντιστοίχως. Προς το παρών, όταν ο παίχτης έρχεται σε επαφή με έναν εχθρό τότε μπλοκάρει ο ένας τον άλλον και δεν δέχεται ζημιά ο παίχτης. Θα υλοποιηθεί ένα σύστημα κατά το οποίο ο παίχτης δεν μπλοκάρεται πλέον από τον εχθρό και δέχεται ζημιά κάθε φορά που έρχονται σε επαφή. Στην κλάση bp_actionchar_base στις ρυθμίσεις collision της κάψουλας, όπως έχει γίνει και πρωτύτερα, γίνεται αλλαγή της ρύθμισης pawn collision από block σε overlap. Αυτό λύνει το πρόβλημα της σύγκρουσης, όμως ακόμα ο παίχτης δεν μπορεί να δεχθεί ζημιά οπότε θα δημιουργηθεί ένα γεγονός που θα ενεργοποιείται όταν δύο χαρακτήρες έρχονται σε επαφή. Αυτό όμως δεν θα δημιουργηθεί στην γενική κλάση actionchar αλλά στη κλάση του παίχτη bp_player. Οπότε στο γεγονός αυτό γίνεται cast στην κλάση bp_enemybase, καθώς θέλουμε να δέχεται ο παίχτης ζημία μόνο από τους εχθρούς. Όταν γίνεται cast (ρίψη) σε έναν άλλο χαρακτήρα στην Unreal Engine 5 χρησιμοποιώντας C++, αυτό σημαίνει ότι γίνεται προσπάθεια μετατροπής ενός δείκτη ή μιας αναφοράς από έναν τύπο δεδομένων σε έναν άλλο, συνήθως για την απόκτηση πρόσβασης σε συγκεκριμένες λειτουργίες ή ιδιότητες του νέου τύπου δεδομένων. Αρχικά, υπάρχει ένας δείκτη ή μια αναφορά σε έναν βασικό τύπο δεδομένων (π.χ., Aactor). Χρησιμοποιείτε η συνάρτηση Cast<> της Unreal Engine για την μετατροπή του δείκτη στον επιθυμητό τύπο. Η Cast<> είναι ασφαλής κατά το χρόνο εκτέλεσης και επιστρέφει nullptr εάν η ρίψη αποτύχει. Η συνάρτηση Cast<> ελέγχει αν ο αρχικός δείκτης (π.χ., MyActor) μπορεί να μετατραπεί στον επιθυμητό τύπο (π.χ., AMyCharacter). Αυτός ο έλεγχος βασίζεται στις πληροφορίες τύπων που είναι διαθέσιμες κατά το χρόνο εκτέλεσης. Εάν η ρίψη είναι επιτυχής, επιστρέφεται ένας νέος δείκτης του επιθυμητού τύπου, ο οποίος επιτρέπει την πρόσβαση σε συγκεκριμένες μεθόδους και ιδιότητες του νέου τύπου. Εάν η ρίψη αποτύχει (δηλαδή, ο αρχικός δείκτης δεν είναι συμβατός με τον επιθυμητό τύπο), η συνάρτηση Cast<> επιστρέφει nullptr. Χρησιμοποιείται το Cast<AEnemyCharacter>(OtherActor) για να γίνει η μετατροπή του ηθοποιού που έγινε overlap σε τύπο AenemyCharacter. Αν το OtherActor είναι πράγματι ένας AEnemyCharacter, το casting θα είναι επιτυχές και η μεταβλητή Enemy θα περιέχει έναν έγκυρο δείκτη στον εχθρό. Αν το OtherActor δεν είναι AEnemyCharacter, το casting θα επιστρέψει nullptr. Αν το casting είναι επιτυχές (δηλαδή Enemy != nullptr), τότε καλείται η συνάρτηση ApplyDamage στον παίχτη. Θα δημιουργηθεί μία μεταβλητή με όνομα baseDamage που είναι η τιμή της ζημιάς που δέχεται ο παίχτης από τον εχθρό και τίθεται η τιμή σε 1.

Παράδειγμα cast:

```
// SomeFunction
ΣΥΝΑΡΤΗΣΗ SomeFunction()
    // Αρχικά δημιουργούμε ή βρίσκουμε έναν AActor
    MyActor = ΠάρεΚόσμο().ΔημιούργησεActor(AActor.ΣτατικήΚλάση())

    // Προσπαθούμε να κάνουμε cast σε AMyCharacter
    MyCharacter = ΚάνεCastΣεAMyCharacter(MyActor)

    // Έλεγχος αν η ρίψη ήταν επιτυχής
    ΑΝ MyCharacter υπάρχει ΤΟΤΕ
        // Η ρίψη ήταν επιτυχής, καλούμε μεθόδους του AMyCharacter
        MyCharacter.ΕκτέλεσηΕιδικήςΕνέργειας()
    ΑΛΛΙΩΣ
        // Η ρίψη απέτυχε, ενέργειες χειρισμού σφάλματος
        ΚαταγραφήΜηνύματος(Warning, "Casting failed, the actor is not of type AMyCharacter.")
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Σε αυτό το σημείο όμως πρέπει να υλοποιηθεί η λογική πως ο παίχτης δεν θα δέχεται ζημιά απλός κάνοντας overlap με τον εχθρό αλλά με το capsule component του εχθρού. Οπότε στο event overlap αφού έχει ήδη γίνει cast στην κλάση του εχθρού και υπάρχει δείκτης στον εχθρό, υπάρχει πρόσβαση στο αντικείμενο capsule component και εφαρμόζεται η συνθήκη ότι ο παίχτης θα δεχθεί ζημιά μόνο όταν κάνει overlap με ένα component που θα είναι capsule component της κλάσης enemybase, αλλιώς δεν θα συμβεί τίποτα. Επιπλέον πρέπει να δημιουργηθεί ένας μηχανισμός που θα κάνει τον παίχτη άτρωτο για λίγα δεύτερα αφού λάβει ζημιά ώστε το παιχνίδι να είναι πιο ισορροπημένο και να δίνεται επαρκής χρόνος αντίδρασης στον παίχτη για να αποφύγει τον εχθρό. Έτσι στην bp_player θα δημιουργηθεί ένα νέο event με όνομα triggerInvincibillity το οποίο θα αλλάζει τη ρύθμιση του capsule component του παίχτη από overlap σε ignore. Έπειτα θα προστεθεί μία συνάρτηση retriggerable delay στο τέλος που έχει ήδη επεξηγηθεί και την μεταβλτή invincibillityDuration που θα ορίζει την τιμή του delay σε 0,5. Έπειτα θα γίνει πάλι αλλαγή της ρύθμισης capsule σε overlap. Έπειτα θα γίνει προσθήκη του event any damage το οποίο θα καλεί το event anydamge του γονέα πρώτα και μετά θα γίνεται κλήση του event triggerInvincibillity. Πλέον ο παίχτης μπορεί να δεχθεί ζημιά και να γίνει άτρωτος για λίγα δεύτερα, αλλά πρέπει να δημιουργηθεί στη συνέχεια και το κατάλληλο animation. Έτσι γίνεται δημιουργία της συνάρτησης togglespriteflicker όπου αλλάζει την μεταβλητή visibility τύπου boolean η οποία υπάρχει από μόνη της στης ρυθμίσεις spirte στο UE5 και ελέγχει εάν το sprite θα φαίνεται ή όχι στον παίχτη. Οπότε εφαρμόζεται η λογική εάν το visibility είναι true κάνε το false και το αντίστροφο. Οπότε πίσω στο γεγονός triggerinvicibillity πριν το delay εφαρμόζεται η συνάρτηση αυτή με μία άλλη συνάρτηση με όνομα set timer η οποία θα κάνει loop την συνάρτηση togglespriteflicker με βάση της μεταβλητής spriteflickerinterval που ορίζει τον χρόνο που θα γίνεται το loop και ορίστηκε σε 0,12 δεύτερα. Αυτή η συνάρτηση θα λειτουργεί μέχρι το τέλος του retriggerabledelay, ώστε όταν τελειώσει ο χρόνος της αθανασίας του παίχτη αυτός να σταματήσει να αναβοσβήνει και γίνεται πάλι η ρύθμιση του visibility σε αληθές, ώστε να μην υπάρχει περίπτωση μετά το τέλος της συνάρτησης toggle sprite flicker ο παίχτης να μείνει αόρατος για πάντα. Έπειτα έγινε ρύθμιση της μεταβλητής invincibility Duration σε 1,5 από 0,5.

4.3.5 Εφαρμογή μηχανισμού μετατόπισης παίχτη και σαστίσματος κατά τη λήψη ζημιάς

Πρώτα θα γίνει εφαρμογή της μετατόπισης του παίχτη προς την αντίθετη κατεύθυνση όταν αυτός δέχεται ζημιά ως ένα ωραίο εφέ που θα βελτιώσει την εμπειρία του παιχνιδιού. Οπότε στο event any damage εντός της κλάσης bp_player θα εφαρμοστεί μια συνάρτηση launch character. Πριν από αυτό όμως πρέπει να γίνει εντοπισμός του εχθρού σε σχέση με τον παίχτη ώστε να γίνει η εκτόξευση προς την αντίθετη πλευρά από αυτή που βρίσκεται ο εχθρός. Γίνεται η χρήση μιας μεθόδου με όνομα GetActorLocation(), η οποία επιτρέπει την απόκτηση της τρέχουσας θέσης ενός αντικειμένου (bp_player) στον κόσμο του παιχνιδιού. Έτσι αποθηκεύεται η τοποθεσία του αντικειμένου σε μια μεταβλητή FVector. Αυτή η συνάρτηση χρησιμοποιείται και στην αναφορά του αντικειμένου bp_enemychar που έχει γίνει νωρίτερα για τον εντοπισμό της θέσης του εχθρού.

Παράδειγμα(GetActorLocation):

Πρέπει να υπάρχει στην κεφαλίδα header που περιέχει τον ορισμό της κλάσης AActor. Αυτό γίνεται με την εντολή: #include "GameFramework/Actor.h"

// Εισάγουμε την απαραίτητη κεφαλίδα για την κλάση Aactor

// Συνάρτηση για την απόκτηση της τοποθεσίας ενός αντικειμένου

ΣΥΝΑΡΤΗΣΗ PrintActorLocation()

// Λήψη της τοποθεσίας του αντικειμένου

ActorLocation = ΠάρεΤοποθεσίαΑντικειμένου()

// Εκτύπωση της τοποθεσίας στην κονσόλα

ΚαταγραφήΜηνύματος(Warning, "Η τοποθεσία του αντικειμένου είναι: %s", ActorLocation.ΜετατροπήΣεΚείμενο())

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται όταν αρχίζει το παιχνίδι ή όταν το αντικείμενο δημιουργείται

ΣΥΝΑΡΤΗΣΗ BeginPlay()

ΚάλεσεSuperBeginPlay()

// Κλήση της συνάρτησης για εκτύπωση της τοποθεσίας

PrintActorLocation()

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

Έπειτα χρησιμοποιείται η συνάρτηση με όνομα find rotation που χρησιμοποιείται για να υπολογίσει την περιστροφή που απαιτείται ώστε ένα αντικείμενο (actor) να κοιτάξει προς μια συγκεκριμένη θέση (target position). Αυτή η συνάρτηση είναι ιδιαίτερα χρήσιμη σε παιχνίδια και εφαρμογές όπου χρειάζεται να κατευθύνουμε ένα αντικείμενο προς κάποιο στόχο. Γίνεται εισαγωγή των κεφαλίδων Kismet/KismetMathLibrary.h για τις μαθηματικές λειτουργίες και GameFramework/Actor.h για την κλάση AActor. Δημιουργείτε μια συνάρτηση που λαμβάνει ως παράμετρο την τοποθεσία του στόχου (TargetLocation). Λαμβάνει την τρέχουσα τοποθεσία του αντικειμένου χρησιμοποιώντας τη GetActorLocation και υπολογίζει την απαιτούμενη περιστροφή για να κοιτάξει το αντικείμενο προς τον στόχο χρησιμοποιώντας τη FindLookAtRotation. Τέλος εφαρμόζει την υπολογισμένη περιστροφή στο αντικείμενο με τη SetActorRotation.

Παράδειγμα:

// Εισάγουμε τις απαραίτητες κεφαλίδες

// Συνάρτηση για τον υπολογισμό και την εφαρμογή της περιστροφής προς ένα στόχο

ΣΥΝΑΡΤΗΣΗ RotateTowardsTarget(TargetLocation)

// Λήψη της τρέχουσας τοποθεσίας του αντικειμένου

ActorLocation = ΠάρεΤοποθεσίαΑντικειμένου()

// Υπολογισμός της περιστροφής για να κοιτάξει το αντικείμενο προς τον στόχο

LookAtRotation = ΥπολόγισεΠεριστροφήΓιαΚοίταγμα(ActorLocation, TargetLocation)

// Εφαρμογή της περιστροφής στο αντικείμενο

ΡύθμισηΠεριστροφήςΑντικειμένου(LookAtRotation)

// Εκτύπωση της περιστροφής στην κονσόλα

ΚαταγραφήΜηνύματος(Warning, "Η περιστροφή του αντικειμένου είναι: %s", LookAtRotation.ΜετατροπήΣεΚείμενο())

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

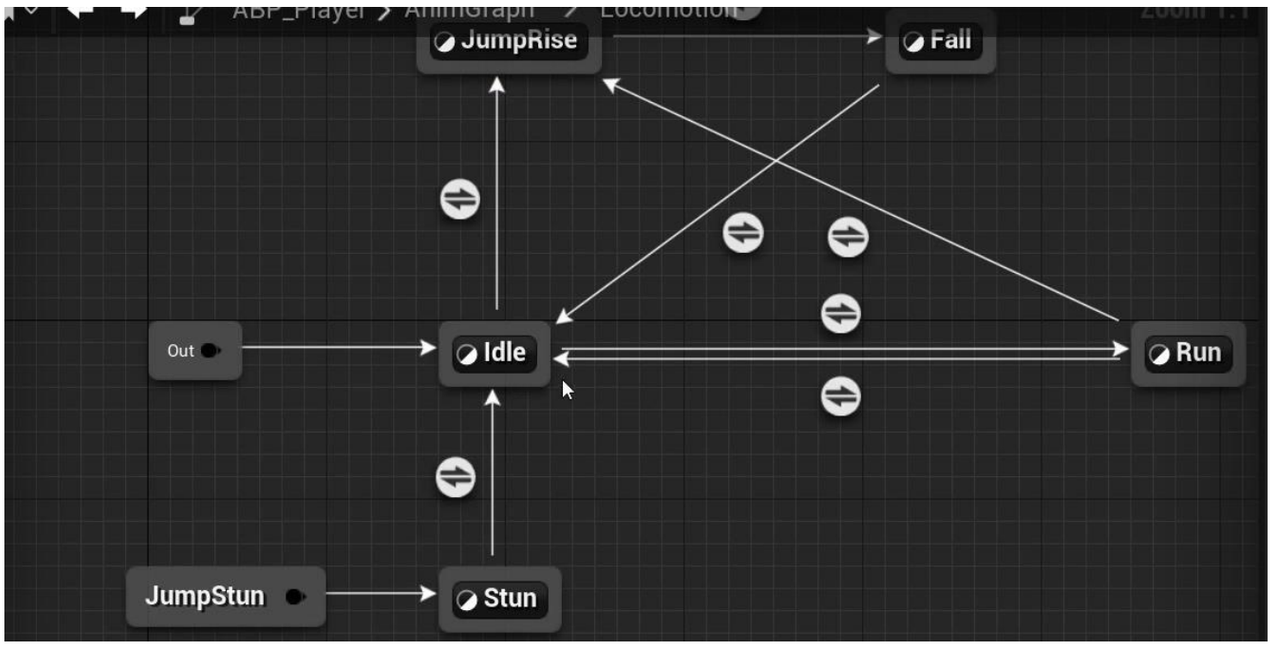
// Συνάρτηση που καλείται όταν αρχίζει το παιχνίδι ή όταν το αντικείμενο δημιουργείται

ΣΥΝΑΡΤΗΣΗ BeginPlay()

```
    ΚάλεσεSuperBeginPlay()

    // Καθορισμός τοποθεσίας του στόχου
    TargetLocation = ΔημιουργίαVector(1000.0, 1000.0, 0.0)

    // Κλήση συνάρτησης περιστροφής προς τον στόχο
    RotateTowardsTarget(TargetLocation)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση εκτόξευσης του χαρακτήρα βάσει συνθήκης
ΣΥΝΑΡΤΗΣΗ LaunchCharacterBasedOnCondition(bCondition, Value1, Value2, Multiplier)
    // Επιλογή τιμής βάσει της συνθήκης
    SelectedValue = AN bCondition ΤΟΤΕ Value1 ΑΛΛΙΩΣ Value2

    // Πολλαπλασιασμός της επιλεγμένης τιμής με τον παράγοντα πολλαπλασιασμού
    LaunchValue = SelectedValue * Multiplier

    // Δημιουργία κατεύθυνσης εκτόξευσης με βάση τον άξονα X
    LaunchDirection = ΔημιουργίαVector(LaunchValue, 0, 0)

    // Εκτόξευση του χαρακτήρα χρησιμοποιώντας την μέθοδο LaunchCharacter
    ΕκτόξευσηΧαρακτήρα(LaunchDirection, αληθές, αληθές)

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Έτσι γίνεται η εκτόξευση του χαρακτήρα προς τα πάνω στη σωστή κατεύθυνση, όμως αυτό δεν είναι αρκετά ικανοποιητικό από μόνο του, οπότε θα γίνει αλλαγή τον ρυθμίσεων του charactermovement component στις ρυθμίσεις της αναπήδησης του παίχτη. Έτσι ακριβώς πριν την υλοποίηση της προηγούμενης λογικής θα ρυθμιστεί η μεταβλητή falling lateral friction του component σε 0. Επίσης να αναφερθεί πως για τη συνάρτηση launch character είχε δημιουργηθεί μία μεταβλητή με όνομα knockbackpower hor και μία Knockbackpower vert που καθορίζουν την μετατόπιση στους δύο άξονες. Μετά τη συνάρτηση launch character θα χρησιμοποιηθεί η συνάρτηση retriggerable delay που έχει ήδη αναφερθεί με διάρκεια ίδια με την μεταβλητή invisibillity duration ώστε να ρυθμιστεί το lateral friction στην αρχική τιμή 50, αλλιώς αυτό θα επηρέαζε τον μηχανισμό της αναπήδησης του παίχτη αν παρέμενε στην τιμή 0. Έπειτα θα γίνει η δημιουργία του stun animation. Στην κλάση AS_Player έχει ήδη γίνει προσθήκη του animation αυτού μαζί με όλα τα υπόλοιπα όπως έχει αναλυθεί στο κεφάλαιο 4.1. Οπότε στην κλάση ABP_Player που εφαρμόζεται το δένδρο καταστάσεων του παίχτη θα εισαχθεί μία νέα κατάσταση με όνομα stun που θα παίζει το animation stun. Το πρόβλημα είναι ότι ο παίχτης μπορεί να βρεθεί σε αυτή την κατάσταση από οποιαδήποτε άλλη, για αυτό το λόγο θα χρησιμοποιηθεί μία λειτουργία του paperzd plugin που λέγεται animation jump. Στην ουσία δημιουργείται ένας ¨”κόμβος” στον οποίο γίνεται κλήση απευθείας από ένα άλλο σημείο του προγράμματος. Οπότε θα συνδεθεί το animation αυτό με ένα node με όνομα JumpStun. Οπότε μετά την συνάρτηση launch character και πριν την retriggerable delay θα προστεθεί μία λογική η οποία στην ουσία θα καλεί την συνάρτηση Jump node που παίζει το animation. Έπειτα γίνεται ρύθμιση στο PapaerZD της μεταβλητής loop animation σε ψευδής για να παίξει το εφέ μόνο μία φορά και να μην επαναλαμβάνεται. Έπειτα πρέπει να εφαρμοστεί μία λογική έτσι ώστε να επιστρέφει πίσω στην κατάσταση

αδράνειας και το υπόλοιπο δένδρο. Οπότε σαν συνθήκη πολύ απλά ορίστηκε πως όταν τελειώσει η διάρκεια αυτού του εφέ θα μεταβεί ο παίχτης πίσω στην κατάσταση αδράνειας.



Εικόνα 20: Ανανεωμένο δένδρο καταστάσεων

Ένα πρόβλημα που προκύπτει όμως είναι ότι ο παίχτης μπορεί ακόμα να πυροβολεί ενώ είναι χτυπημένος οπότε χρησιμοποιώντας την μεταβλητή isStunned που ήδη υπάρχει, αφού παιχτεί το εφέ ορίζεται σε αληθής. Έπειτα στο input action shoot στην αρχή του γεγονότος πριν από την υπόλοιπη εκτέλεση κώδικα τίθεται η συνθήκη πως θα εκτελεστεί μόνο όταν η μεταβλητή isStunned είναι ψευδής. Το ίδιο θα υλοποιηθεί και για το move input. Τέλος γίνεται η επαναφορά της μεταβλητής isStunned που ορίζει την ακινητοποίηση του παίχτη μετά από 0,4 δεύτερα από όταν ορίστηκε σε αληθής με μία συνάρτηση retriggerable delay. Το 0,4 αποθηκεύτηκε σε μία μεταβλητή με όνομα stun duration. Οπότε τώρα γίνεται η αλλαγή της συνθήκης από stunned σε αδρανής με την νέα συνθήκη όταν η μεταβλητή isstunned είναι ψευδής και όχι με την χρονική διάρκεια του εφέ.

4.4 Επιπλέον ικανότητες παίχτη

4.4.1 Φορτισμένη βολή παίχτη

Σε αυτό το στάδιο θα προστεθεί η ικανότητα στον παίχτη αντί της κανονικής του βολής να μπορεί να κρατήσει την βολή του και να την απελευθερώσει μετά από μία χρονική περίοδο, ώστε να προκαλέσει περισσότερη ζημιά στον αντίπαλο. Στην κλάση IMC_Action γίνεται προσθήκη στην εντολή shoot ενός trigger με όνομα released, το οποίο ουσιαστικά δίνει την εντολή στην input action shoot να υπολογίζει επίσης τον χρόνο που το πλήκτρο j παραμένει πατημένο όταν ο παίχτης πυροβολεί. Στην enhanced input shoot γίνεται η διαχείριση της από το UenhancedInputComponent που υπάρχει στο UE5 που δέχεται την είσοδο της εντολής shoot από το πληκτρολόγιο. Για αυτή την νέα βολή θα δημιουργηθούν δύο ακόμα κλάσεις παιδιά της bp_playerprojectile_base, μία κλάση για την ημιφορτισμένη βολή και μία κλάση για την πλήρης φορτισμένη βολή με ονόματα bp_playerprojectile_partialcharge και bp_playerprojectile_fullcharge. Στην πρώτη η οποία κληρονομεί το sprite της κανονικής βολής θα γίνει η αλλαγή του χρώματός της σε μοβ από τις ρυθμίσεις του sprite. Αυτή η βολή θα κάνει και αυτή μόνο μία μονάδα ζημιάς. Στην δεύτερη κλάση που θα είναι η πλήρης φορτισμένη βολή θα γίνει αλλαγή της μεταβλητής projectile damage από 1 σε 3. Έπειτα θα γίνει αλλαγή του flipbook από την βασική βολή στο εφέ της επιφορτισμένης βολής. Στη συνέχεια θα γίνει αλλαγή των initial speed και max speed στην τιμή 1300. Αυτές οι μεταβλητές κληρονομούνται από την κλάση γονέα. Έπειτα θα δημιουργηθεί ένα γεγονός με όνομα triggershoot animation στο οποίο όποτε καλείται θα προστεθεί σε αυτό ο κώδικας που υπάρχει ήδη με την λογική ότι η μεταβλητή ishooting αλλάζει από αληθής σε ψευδής με ένα retriggerable delay. Αυτό γίνεται για συντομία ώστε να μην χρειάζεται να επαναλαμβάνεται η λογική αλλά απλός να γίνεται κλήση του event. Επιπλέον στο σημείο του κώδικα που αφαιρούμε την ενέργεια με κάθε βολή για την αποφυγή σφαλμάτων υλοποιείται η λογική

με χρήση της συνάρτησης max, πως κατά την αφαίρεση ενέργειας δεν είναι δυνατόν η τιμή της ενέργειας να είναι μικρότερη από 0 και στην συνάρτηση restore energy που έχει δημιουργηθεί προηγουμένως δεν μπορεί να πάρει τιμή μεγαλύτερη από 3. Οπότε η συνάρτηση restore energy επειδή καλείτε από την κλάση projectile base και κληρονομήθηκε από την playerprojectile_fullcharge θα γίνει αλλαγή, ώστε να προσθέτει συν 3 ενέργεια αντί για 1, αφού η πλήρης φορτισμένη βολή ξοδεύει 3 ενέργεια. Οπότε στην συνέχεια στην bp_player εφαρμόζεται η λογική ελέγχου πως ο παίχτης πρέπει να ελέγχει εάν έχει αρκετή ενέργεια ώστε να μπορεί να πυροβολήσει αναλόγως της βολής που θέλει να κάνει. Τέλος εφόσον υπάρχει ήδη η ολοκληρωμένη λογική για το πως μπορεί να πυροβολήσει ο παίχτης και τι συμβαίνει αφού πυροβολήσει εφαρμόζεται η συνθήκη πως εάν κρατηθεί το πλήκτρο j άνω των 0,8 δευτερολέπτων και κάτω των 3 δευτερολέπτων θα γίνει κλήση του εφέ της κλάσης bp_ playerprojectile_partialcharge αλλιώς αν το κρατήσει για άνω των τρία ή ίσα με τρία δευτερόλεπτα, τότε θα παίξει το εφέ της κλάσης bp_playerprojectile_fullcharge. Στην πρώτη περίπτωση θα χάσει μία ενέργεια κατά την βολή και θα αναπληρώσει 1 μετά, ενώ στην δεύτερη θα χάσει 3 και θα αναπληρώσει 3 με την καταστροφή της σφαίρας. Δημιουργήθηκαν για αυτό τον σκοπό οι μεταβλητές fullchargetime και partialchargetime. Τέλος πρέπει αν εφαρμοστεί η λογική πως εάν ο παίχτης δεχθεί ζημιά κατά την περίοδο φόρτισης της βολής τότε θα ακυρωθεί η βολή αυτή. Για αυτό το λόγο γίνεται δημιουργία της μεταβλητής washitduringcharge τύπου boolean και μετά την κλήση του γεγονότος anydamage της κλάσης bp_player, αφού δεχθεί ζημιά ο παίχτης τίθεται η μεταβλητή washitduringcharge σε αληθής. Οπότε στο action shoot μετά την λογική πως ο παίχτης μπορεί να πυροβολήσει μόνο αν δεν είναι σαστισμένος που έχει ήδη υλοποιηθεί προστίθεται η λογική πως η μεταβλητή washitduringcharge αλλάζει πάλι σε ψευδής. Οπότε κάθε φορά που πυροβολεί ο παίχτης θα γίνεται επαναφορά της μεταβλητής. Έπειτα στο σημείο του κώδικα που εκτελείτε η φορτισμένη βολή μετά από τρία δεύτερα πατήματος προστίθεται επιπλέον η συνθήκη πως η μεταβλητή δεν πρέπει να είναι washitduringcharge δεν είναι αληθής ή με άλλα λόγια να είναι ψευδής. Εάν δεν είναι ψευδής αυτό σημαίνει πως ο παίχτης δέχθηκε ζημιά και έτσι δεν μπορεί να πυροβολήσει, οπότε δεν συμβαίνει τίποτα. Στη συνέχεια θα πρέπει να υλοποιηθεί το κατάλληλο animations το οποίο θα υποδηλώνει ότι ο παίχτης πραγματοποιεί αυτή τη φορτισμένη βολή ώστε να είναι εμφανές στον παίχτη πότε αρχίζει η διαδικασία φόρτισης αλλά και για την ικανοποίηση του αισθητικού κομματιού του παιχνιδιού. Οπότε στην κλάση BP_player θα δημιουργηθεί μια νέα συνάρτηση με όνομα handle charge flash που θα είναι υπεύθυνη για το animation. Αυτή η συνάρτηση θα εκτελείτε κατά την αρχή του IA_Shoot action και δέχεται ως παράμετρο τα δευτερόλεπτα για τα οποία είναι ενεργή η πράξη του πυροβολισμού. Όμως το animation θα αλλάζει για κάθε διαφορετική φάση της φορτισμένης βολής. Οπότε γίνεται έλεγχος στην συνάρτηση εάν ο χρόνος της βολής ή αλλιώς elapsed time, όπως ονομάστηκε η βολή είναι μικρότερος ή ίσος με την μεταβλητή partial charged shot 0,8 seconds, τότε δεν γίνεται τίποτα, αλλιώς θα δημιουργηθεί ένα χρονόμετρο κατά τη διάρκεια του οποίου θα παίζει το animation και στην συνέχεια θα γίνεται επαναφορά του χρονομέτρου. Δημιουργείται μία μεταβλητή με όνομα charge flash timer τύπου float στην οποία μετά τα 0,8 δευτερόλεπτα θα αρχίσουν να προστίθενται τα δευτερόλεπτα μετά το τελευταίο καρέ με τη χρήση της συνάρτησης get world delta seconds, καθώς η UE5 έχει μία λειτουργία η οποία αποθηκεύει τα δευτερόλεπτα από την εκκίνηση του παιχνιδιού στην μεταβλητή delta seconds που μπορεί να χρησιμοποιηθεί και από τους προγραμματιστές για την καταμέτρηση συγκεκριμένων χρονικών περιόδων. Στην περίπτωση που δεν έχουν περάσει 0,8 δεύτερα από την εκκίνηση του πυροβολισμού, τότε γίνεται επαναφορά του χρονομέτρου και τίθεται η μεταβλητή στην τιμή 0. Έπειτα στην περίπτωση που ξεπεράστηκαν τα 0,8 δεύτερα και άρχισε να μετράει το χρονόμετρο τότε γίνεται επιπλέον έλεγχος για κάθε 0,2 δευτερόλεπτα που περνούν και εάν έχουν περάσει τότε γίνεται επαναφορά του χρονομέτρου στα 0 δεύτερα εκτελείται η συνάρτηση trigger flash που έχει ήδη υλοποιηθεί και προσθέτει το animation της λάμψης στον παίχτη. Στο τέλος για τον προσδιορισμό του χρώματος της λάμψης θα προστεθεί η τελική συνθήκη εάν η μεταβλητή elapsed time είναι μεγαλύτερη ή ίση της full charge time τότε το χρώμα θα τεθεί σε μπλε αλλιώς θα τεθεί σε άσπρο χρώμα, ώστε ο παίχτης να μπορεί να διαχωρίσει την πλήρως φορτισμένη βολή από την μη πλήρως φορτισμένη. Τέλος σε περίπτωση που ο παίχτης δεχθεί ζημιά κατά την φόρτωση της βολής σταματάει την διαδικασία του πυροβολισμού όπως είχε υλοποιηθεί προηγουμένως ορθά, αλλά δεν σταματάει το animation. Έτσι στην αρχή του ελέγχου της συνάρτησης handle charge flash θα γίνει και ο έλεγχος πως η μεταβλητή was hit during charge που είχε δημιουργηθεί πρωτύτερα να είναι μη ψευδής. Έτσι ολοκληρώνεται ο μηχανισμός της φορτισμένης βολής.

4.4.2 Σύστημα αναρρίχησης τοίχου

Ένας επιπλέον μηχανισμός του παίχτη θα είναι η ικανότητα να μπορεί να γραπώνεται από τους τοίχους σαν να σκαρφαλώνει και να μπορεί να μεταπηδήσει από τον ένα τοίχο στον άλλο, με την σωστή χρήση συγκεκριμένων πλήκτρων, ώστε να του δίνεται η ικανότητα ουσιαστικά να σκαρφαλώνει σε επιφάνειες. Όταν ο παίχτης γραπώνεται από έναν τοίχο τότε αυτός θα γλιστράει με πολύ αργό ρυθμό προς τα κάτω εκτός αν επιλέξει να αποδεσμευτεί από το γράπωμα όπου θα πέσει κάτω ακαριαία και θα μπορεί κάνοντας άλμα να πιαστεί από άλλον τοίχο. Οπότε για αρχή πρέπει ο παίχτης να μπορεί να εντοπίσει τους κοντινούς τοίχους ώστε να μπορεί να τους σκαρφαλώσει δηλαδή πρέπει να μπορεί να αναγνωρίζει τις κοντινές επιφάνειες στις οποίες μπορεί να γραπωθεί. Αυτό θα επιτευχθεί με τη χρήση του line trace. Το line trace γνωστό και ως raycast μπορεί να χρησιμοποιηθεί για τον εντοπισμό αντικειμένων στον κόσμο του παιχνιδιού. Για την δημιουργία του πρέπει να ενσωματωθούν τα κατάλληλα headers, να γίνει ορισμός του σημείου αρχής και τέλους του line trace, να γίνει κλήση της μεθόδου που εκτελεί το Line trace και να γίνει επεξεργασία του αποτελέσματος.

Παράδειγμα:

```
// Συνάρτηση εκτέλεσης Line Trace
ΣΥΝΑΡΤΗΣΗ PerformLineTrace()
    // Προσδιορισμός αρχικού σημείου και τελικού σημείου
    Start = ΠάρεΤοποθεσίαΑντικειμένου()
    ForwardVector = ΠάρεΔιεύθυνσηΜπροστάΑντικειμένου()
    End = Start + (ForwardVector * 1000.0) // 1000 μονάδες μπροστά από τον ηθοποιό

    // Προσδιορισμός παραμέτρων σύγκρουσης
    CollisionParams.ΠρόσθεσεΠαράβλεψηΑντικειμένου(αυτό)

    // Εφαρμογή line trace
    HitResult = ΕφαρμογήLineTraceSingleByChannel(Start, End, ECC_Visibility, CollisionParams)

    // Έλεγχος αν υπήρξε σύγκρουση
    ΑΝ HitResult.bBlockingHit ΤΟΤΕ
        // Δημιουργία γραμμής debug για οπτικοποίηση του line trace
        ΣχεδίασηDebugΓραμμής(ΠάρεΚόσμο(), Start, End, Κόκκινο, ψευδές, 1, 0, 1)

// Εκτύπωση του ονόματος του ηθοποιού που εντοπίστηκε
ΑΝ HitResult.ΠάρεΑντικείμενο() ΤΟΤΕ
ΚαταγραφήΜηνύματος(Warning, "Hit Actor: %s", ΠάρεΌνομαΑντικειμένου(HitResult.ΠάρεΑντικείμενο()))
ΑΛΛΙΩΣ
        // Σχεδίαση πράσινης γραμμής για οπτικοποίηση του line trace
        ΣχεδίασηDebugΓραμμής(ΠάρεΚόσμο(), Start, End, Πράσινο, ψευδές, 1, 0, 1)
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Οπότε στην κλάση bp_player για κάθε καρέ θα γίνει κλήση της μεθόδου line trace για αντικείμενα, η μέθοδος θα δέχεται ως παραμέτρους την αρχή και το τέλος του Line trace , καθώς και τους τύπους αντικείμενων για τα οποία θα "ψάχνει". Οπότε στην παράμετρο των αντικειμένων θα δοθεί σαν τύπος καναλιού το world static , δηλαδή τα αντικείμενα του κόσμου τα οποία δεν έχουν ικανότητα κίνησης, όπως τοίχοι πατώματα και άλλα στοιχεία. Για την αρχή του line trace θα οριστεί η θέση του blueprint του παίχτη και έπειτα για τον ορισμό της τελικής θέσης θα γίνει η χρήση της διανυσματικής κατεύθυνσης του ηθοποιού προς τα εμπρός που είναι αποθηκευμένη στο movement component σε ένα vector με όνομα forward vector που ουσιαστικά έχει τις συντεταγμένες του ηθοποιού στον άξονα x. Έπειτα θα πολλαπλασιαστεί αυτή η τιμή επί 200, δηλαδή θα προστεθούν σε αυτή την κατεύθυνση άλλες 200 μονάδες προς τα εμπρός και στη συνέχεια θα προστεθεί αυτή η τιμή στις συντεταγμένες του ηθοποιού. Αυτό σημαίνει πως το line trace θα λειτουργεί από την θέση του ηθοποιού έως και 200 μονάδες εμπρός από αυτή την θέση προς την πλευρά που κοιτάει και κατευθύνεται ο παίχτης/ ηθοποιός. Έπειτα επειδή είναι επιθυμητό να μπορεί να γραπωθεί ο παίχτης από τον τοίχο μόνο εάν βρίσκεται πολύ κοντά σε αυτόν θα μειωθούν οι μονάδες στης 40, καθώς η κάψουλα του παίχτη είναι 34 μονάδες και πειραματικά βρέθηκε πως οι 40 μονάδες είναι ιδανικές. Αυτές οι μονάδες θα αποθηκευτούν σε μία μεταβλητή με όνομα wall trace length. Έπειτα μετά το line trace θα γίνεται έλεγχος με τη δημιουργία μιας νέας μεταβλητής is wall sliding τύπου boolean. Αν οι έξοδος του line trace είναι αληθής τότε αυτό σημαίνει πως ο παίχτης βρίσκεται δίπλα σε τοίχο και έτσι τίθεται η παραπάνω μεταβλητή σε true αλλιώς παραμένει σε κατάσταση false. Στην περίπτωση που είναι αληθής η έξοδος γίνεται η αλλαγή του Velocity του παίχτη μέσο της κλάσης του Character Movement component, δηλαδή την ταχύτητα κίνησης του χαρακτήρα. Αυτό που πρέπει να αλλάξει είναι η κίνηση στον κάθετο άξονα εφόσον ο παίχτης βρίσκεται κοντά σε τοίχο ώστε ουσιαστικά να μην πέφτει προς τα κάτω δίνοντας την ψευδαίσθηση ότι κρατιέται από τον τοίχο. Έτσι θα μεταβληθεί μόνο η ταχύτητα στον άξονα z ενώ στον x και y θα παραμείνουν το ίδιο. Οπότε με την χρήση της max από τη βιβλιοθήκη της c++ θα τεθεί πως η μέγιστη τιμή στον άξονα y θα είναι η -20, οπότε ο παίχτης εφόσον βρίσκεται κοντά σε τοίχο δεν θα μπορεί να κινηθεί στον άξονα y με ταχύτητα μεγαλύτερη από -20 που είναι μία πολύ αργή κίνηση προς τα κάτω. Όμως τώρα παρατηρείται ότι ο παίχτης μπορεί να γραπωθεί από έναν τοίχο ενώ ταξιδεύει προς τα πάνω, πχ μετά από ένα άλμα σκαρφαλώνοντας προς στιγμήν στο τοίχο με ανοδική πορεία αντί για καθοδική, έτσι θα εφαρμοστεί στην παραπάνω λογικής στην αρχή ένας επιπλέον έλεγχος. Ο παίχτης θα μπορεί να γραπωθεί από τον τοίχο μόνο εάν οι ταχύτητά του στον άξονα z είναι μεγαλύτερη ή ίση από την τιμή – 40, δηλαδή μόνο εάν βρίσκεται σε σίγουρη καθοδική πορεία. Επιπλέον ένας ακόμα μηχανισμός που θα δημιουργηθεί είναι πως ο παίχτης δεν θα μπορεί να γραπωθεί από έναν τοίχο παρά μόνο εάν είναι πατημένο το αντίστοιχο πλήκτρο με κατεύθυνση προς την πλευρά του τοίχου. Αυτό θα δίνει την ικανότητα στον παίχτη να επιλέγει εκείνος εάν επιθυμεί να γραπωθεί ή όχι και να επιλέγει επίσης εάν θέλει να τερματίσει νωρίτερα αυτή τη κίνηση αποδεσμεύοντας το πλήκτρο, καθώς πριν η αναρρίχηση ήταν εξαναγκασμένη εφόσον ο παίχτης βρισκόταν κοντά στον τοίχο. Οπότε θα γίνει έλεγχος εάν η κατεύθυνση της εισόδου που δίνει ο χρήστης είναι ίδια με αυτή που κοιτάει ο παίχτης κατά την διαδικασία της αναρρίχησης. Οπότε παίρνοντας τις τιμές του forward vector του παίχτη και την τιμή της πράξης εισόδου move που έχουμε ήδη δημιουργήσει, δηλαδή την έξοδο που μπορεί να είναι -1,0 ή 1 αναλόγως εάν δίνεται κάποια είσοδος και ποια είσοδος δίνεται και συγκρίνοντας εάν ο πολλαπλασιασμός τους επιστρέφει θετικό ή αρνητικό πρόσημο. Εάν το αποτέλεσμα είναι μεγαλύτερο του 0 τότε η είσοδος κίνησης έχει την ίδια κατεύθυνση με την μπροστινή πορεία του παίχτη αλλιώς είναι αντίθετες. Ενσωματώνοντας αυτή τη λογική στην αρχή του ελέγχου εξασφαλίζει πως ο παίχτης μπορεί να δεσμευτεί ή να αποδεσμευτεί από την πράξη της αναρρίχησης όποτε αυτός επιθυμεί.

4.4.3 Δημιουργία animation αναρρίχησης και ικανότητας αναπήδησης τοίχου

Ο παίχτης πλέον μπορεί μα αναρριχηθεί στους τοίχους αλλά δεν έχει γίνει ακόμα η εφαρμογή της κινησιολογίας που προσδίδει αυτή την ικανότητα. Έτσι στην γνωστή κλάση ABP_Player με το δένδρο συμπεριφοράς του παίχτη πρέπει να συμπεριληφθεί η νέα κατάσταση μετάβασης προς και από το νέο animation αναρρίχησης. Ήδη είναι γνωστό από την εφαρμογή της λογικής του προηγούμενου κεφαλαίου πως ο παίχτης μπορεί να μεταβεί στην κατάσταση αναρρίχησης μόνο όταν βρίσκεται σε καθοδική πορεία, δηλαδή όταν πέφτει από μία επιφάνεια, έτσι η κατάσταση της αναρρίχησης θα προστεθεί μετά την κατάσταση fall. Στην κατάσταση wall slide όπως ονομάστηκε η αναρρίχηση προστίθεται το αντίστοιχο animation, οπότε όταν ο παίχτης μεταβαίνει σε αυτή την κατάσταση πλέον θα φαίνεται αυτό και στην πράξη. Όμως σε περίπτωση που ο παίχτης πυροβολεί την ώρα της αναρρίχησης θα προστεθεί η συνθήκη εάν η μεταβλητή is shooting είναι αληθής τότε θα παίζεται το animation wall slide shoot, αλλιώς θα παίζεται το απλό wall slide. Τέλος πρέπει να οριστεί η κατάσταση μετάβασης

από την κατάσταση fall. Η συνθήκη μετάβασης πολύ απλά είναι εάν η μεταβλητή is wall sliding που έχει ήδη δημιουργηθεί είναι αληθής τότε ο παίχτης μεταβαίνει από την κατάσταση fall στην wall slide. Δηλαδή ο παίχτης ενώ πέφτει είναι κοντά σε τοίχο και η είσοδος πλήκτρου αντιστοιχεί στην κατεύθυνση που κοιτάει. Έπειτα για να τελειώσει η κατάσταση wall slide και να μεταβεί πίσω στην fall η συνθήκη είναι η μεταβλητή is wall sliding να είναι ψευδής. Έτσι ο παίχτης επιστρέφει στην κατάσταση fall και από εκεί σε οποιαδήποτε άλλη κατάσταση χρειαστεί. Παρόλα αυτά πρέπει να γίνει μία επιπλέον ρύθμιση στο animation του wall slide, καθώς ο παίχτης δεν εφάπτεται με τον τοίχο αλλά βρίσκεται στον αέρα, κάτι που φυσικά δεν είναι επιθυμητό. Οπότε στο player wallslide animation class θα γίνει ρύθμιση της μεταβλητής custom pivot point σε 17 μονάδες, ώστε όταν αρχίζει το animation να αλλάζει ελαφρώς η θέση εκκίνησης του, ώστε πλέον ο παίχτης να εφάπτεται με τον τοίχο. Το ίδιο γίνεται και για το wall slide shoot animation. Ένα άλλο πρόβλημα που προκύπτει είναι ότι τώρα ο παίχτης όταν βρίσκεται σε αυτή την κατάσταση και πυροβολεί, δεν ξέρει προς τα που να πυροβολήσει και πολλές φορές πυροβολεί μέσα στον τοίχο αφού τεχνικά ο παίχτης κοιτάει προς τα εκείνη την κατεύθυνση. Οπότε στο shootpos component που έχει ήδη δημιουργηθεί για την κανονική θέση πυροβολισμού του παίχτη θα γίνει αντιγραφή του και θα δημιουργηθεί ένας νέο component με όνομα wall slide shoot pos, δηλαδή θέση πυροβολισμού κατά τη διάρκεια της αναρρίχησης. Έπειτα θα τοποθετηθεί αυτό το component γραφικά στο σημείο της κάνης του όπλου του wall slide shoot animation. Οπότε στην συνάρτηση spawn projectile, η οποία διαχειρίζεται την εμφάνιση της σφαίρας κατά τη διάρκεια του πυροβολισμού θα προστεθεί μία νέα συνθήκη. Πριν την συνάρτηση spawn actor που εμφανίζει την σφαίρα, γίνεται έλεγχος εάν η μεταβλητή is wall sliding είναι αληθής. Στην περίπτωση που είναι τότε ορίζεται ως θέση πυροβολισμού το shoot pos wall slide, αλλιώς η θέση πυροβολισμού παραμένει η προηγούμενη. Αφού πλέον υλοποιήθηκε πλήρως ο μηχανισμός της αναρρίχησης πρέπει τώρα να δημιουργηθεί η ικανότητα ο παίχτης να μπορεί να αναπηδήσει όσο βρίσκεται πάνω στους τοίχους. Όμως η συνθήκη για την αναπήδηση του παίχτη που έχει δημιουργηθεί προηγουμένως είναι πως ο παίχτης πρέπει να βρίσκεται στο έδαφος για να αναπηδήσει, δηλαδή να μην είναι στον αέρα, όμως το παιχνίδι θεωρεί την αναρρίχηση ως μία εναέρια κίνηση, καθώς ο παίχτης τεχνικά πέφτει σε αργή κίνηση άρα είναι στον αέρα και έτσι δεν μπορεί να αναπηδήσει από αυτή την κατάσταση. Αυτό το πρόβλημα θα αντιμετωπιστεί παρακάτω. Οπότε αντί για την συνάρτηση jump θα χρησιμοποιηθεί μία άλλη συνάρτηση η οποία έχει ήδη υλοποιηθεί για την εφαρμογή της εκτόπισης του παίχτη όταν δέχεται ζημιά, την συνάρτηση launch character. Μόνο που αυτή τη φορά θα μεταβληθεί η λειτουργία της, ώστε να λειτουργεί σαν την συνάρτηση αναπήδησης του παίχτη. Οπότε στην Input action Jump πριν την εφαρμογή της συνάρτησης jump θα γίνει έλεγχος της μεταβλητής is wall sliding, εάν είναι ψευδής τότε με την είσοδο του πλήκτρου spacebar ο παίχτης θα αναπηδά κανονικά όπως και πριν, εάν όμως είναι αληθής τότε η λειτουργία του πλήκτρου spacebar θα αλλάζει. Στη δεύτερη περίπτωση καλείται η συνάρτηση launch character με velocity z, δηλαδή ταχύτητα στον κάθετο άξονα 1500(μεταβλητή Wall jump power vertical) μονάδες και velocity x στον οριζόντιο άξονα προς την αντίθετη κατεύθυνση από αυτή που βρίσκεται ο παίχτης. Για να δοθεί φορά προς την αντίθετη κατεύθυνση του άξονα x, γίνεται χρήση των συντεταγμένων του forward vector του παίχτη πάλι και γίνεται ο πολλαπλασιασμός με -1, ώστε να αλλάξει η κατεύθυνση και κατά 1000(μεταβλητή wall jump power horizontal) μονάδες ώστε να αυξηθεί η “δύναμη” του άλματος. Επίσης θα οριστεί η τιμή του lateral falling friction στην τιμή 0, ώστε ο παίχτης να μην δέχεται αντιστάσεις την ώρα της αναπήδησης όσο βρίσκεται στον αέρα πριν την εφαρμογή του launch character. Στη συνέχεια μετά την Launch character θα προστεθεί η γνωστή συνάρτηση retriggerable delay για 0.2(μεταβλητή wall jump lock) δεύτερα και γίνεται επαναφορά της τιμής του falling lateral friction πίσω στην αρχική της τιμή.

4.4.4 Υλοποίηση μηχανισμού γλιστρήματος εδάφους

Για την ρευστότητα της κίνησης του παίχτη αλλά και για να δοθούν στον παίχτη επιπλέον επιλογές ελιγμού και αποφυγής των εχθρών έγινε η προσθήκη ενός επιπλέον μηχανισμού που επιτρέπει σε αυτόν να κινείτε με μεγαλύτερη ταχύτητα στο έδαφος γλιστρώντας πάνω σε αυτό. Για τη δημιουργία αυτής της κίνησης χρησιμοποιήθηκε ένα plugin με όνομα rootmovement5.3. Αυτή η προσθήκη περιέχει κάποιες βιβλιοθήκες C++ που παρέχουν συναρτήσεις που θα χρησιμοποιηθούν στο πρότζεκτ απευθείας από εκεί για ευκολία. Οπότε από αυτή την βιβλιοθήκη γίνεται χρήση της συνάρτησης apply root motion constant force with callbacks. Με τη χρήση αυτής της συνάρτησης θα επιτευχθεί η δημιουργία της κίνηση του γλιστρήματος, όμως για να δουλέψει αυτή η συνάρτηση πρέπει να περαστούν κάποιες παράμετροι. Οι παράμετροι που περνούν στην συνάρτηση είναι το character movement component και το forward vector του παίχτη, έπειτα πρέπει να περαστούν οι μεταβλητές strength (1000) και duration(0,3) που είναι η δύναμη της κίνησης και η διάρκεια αντιστοίχως. Οπότε για το

επόμενο βήμα θα πρέπει να δημιουργηθεί μία νέα είσοδος πληκτρολογίου για αυτήν τη νέα κίνηση. Η νέα είσοδος θα ενεργοποιείται με την χρήση του πλήκτρου space και του s, δηλαδή κάνοντας αναπήδηση αλλά με κατεύθυνση προς τα κάτω. Έτσι στην κλάση IA_Move θα γίνει μετατροπή της ρύθμισης value type από axis1D(vector 1D) σε axis2D(Vector 2D), καθώς σε αργότερα στάδια ανάπτυξης θα προστεθεί κίνηση και στον άξονα z, δηλαδή κίνηση σε 2 διαστάσεις, ενώ μέχρι τώρα υπήρχε μόνο σε μία. Στην κλάση IMC_Action στην IA_Move στις ρυθμίσεις key mappings θα προστεθεί νέο πλήκτρο w με modifier swizzle input axis values και αντιστοίχως πλήκτρο s με negated swizzle input axis values για κίνηση προς τα κάτω. Οπότε στο input action jump στην κλάση bp_player που έχει υλοποιηθεί η λογική αναπήδησης από έδαφος και από τοίχους στον πρώτο έλεγχο του is wall sliding αν είναι ψευδής προστίθεται ένας επιπλέον έλεγχος εάν η τιμή του y από την κίνηση IA_Move είναι μικρότερη του 0, αυτό σημαίνει πως ο χρήστης πατάει το πλήκτρο s. Εάν δεν είναι μικρότερη του 0 τότε ο παίχτης πραγματοποιεί κανονική αναπήδηση, αλλιώς θα εφαρμοστεί η συνάρτηση apply root motion constant force with callbacks και ο παίχτης θα πραγματοποιήσει γλίστρημα. Όμως τώρα όταν ο παίχτης πατάει το πλήκτρο s ο χαρακτήρας αλλάζει κατεύθυνση που δεν είναι το επιθυμητό αποτέλεσμα. Έτσι πηγαίνοντας στην συνάρτηση update rotation στην αρχή γίνεται προσθήκη ελέγχου που κοιτάει αν η κατεύθυνση του παίχτη είναι διάφορη του 0. Έτσι τώρα δεν αλλάζει ο παίχτης κατεύθυνση όταν γίνεται χρήση του πλήκτρου s. Πίσω στην IA_Jump όταν καλείτε η συνάρτηση για το γλίστρημα γίνεται δημιουργία μίας μεταβλητής με όνομα isGroundSliding και τίθεται σε αληθής και κατά το τέλος της συνάρτησης αλλάζει πάλι σε ψευδής. Έτσι γίνεται γνωστό πότε βρίσκεται σε αυτή την κατάσταση ο χαρακτήρας. Εν συνεχεία στο δένδρο καταστάσεων της κλάσης ABP_Player πρέπει να γίνει προσθήκη της νέας κατάστασης ground sliding. Οπότε από την κατάσταση idle θα προστεθεί η κατάσταση γλιστρήματος. Σε αυτή την κατάσταση δεν θα μπορεί να πυροβολήσει ο παίχτης για να μην είναι πολύ εύκολο το παιχνίδι οπότε προστίθεται μόνο το animation slide. Η κατάσταση μετάβασης είναι αν η μεταβλητή isGroundSliding είναι αληθής από idle σε κατάσταση slide και αντιστρόφως αν είναι ψευδής από slide σε idle(αδράνεια). Επίσης στην κατάσταση slide μπορεί να εισέλθει ο παίχτης και από την κατάσταση run με την ίδια συνθήκη, όμως δεν χρειάζεται να οριστεί συνθήκη επιστροφής στην κατάσταση run αφού ο παίχτης μπορεί να επιστρέψει στην κατάσταση idle και από εκεί να μεταβεί σε οποιαδήποτε άλλη κατάσταση όπως η run. Σε αυτό το σημείο έχει εφαρμοστεί πλήρως η λειτουργίας του γλιστρήματος, όμως ακόμα πρέπει να γίνουν κάποιες αλλαγές ώστε να βελτιστοποιηθεί. Αρχικά πρέπει ο χαρακτήρας να μην μπορεί να πυροβολήσει όσο πραγματοποιεί το γλίστρημα και να μην μπορεί να αλλάξει κατεύθυνση την ώρα της πράξης γιατί αυτό θα ήταν πολύ αφύσικο. Επίσης πρέπει να προστεθεί η επιλογή ο παίχτης να μπορεί να ακυρώσει την διαδικασία γλιστρήματος πατώντας το πλήκτρο της αναπήδησης, να μπορεί να ακυρώνεται το γλίστρημα όταν ο παίχτης πέφτει από κάποιο ύψος και όταν ο παίχτης βρίσκεται στον αέρα. Οπότε για την απενεργοποίηση της ικανότητας του παίχτη να πυροβολεί όσο γλιστράει στην κλάση Bp_player στην εκτέλεση της συνάρτησης Ia_shoot στην αρχή του ελέγχου ως επιπλέον έλεγχος για να πυροβολήσει ο παίχτης τίθεται πως η μεταβλητή isGroundSliding πρέπει να μην είναι αληθής μαζί με την μεταβλητή IsStunned δηλαδή ο παίχτης να μην είναι σαστισμένος το οποίο έχει ήδη υλοποιηθεί. Έπειτα ακριβώς το ίδιο θα πρέπει να εφαρμοστεί και στο σημείο ελέγχου για την φορτισμένη βολή. Έτσι πολύ απλά πλέον ο παίχτης δεν μπορεί να πυροβολήσει όσο γλιστράει στο έδαφος. Στη συνέχεια για να μην μπορεί να αλλάξει κατεύθυνση ο παίχτης όσο γλιστράει στην συνάρτηση updateRotation που χειρίζεται να την μεταβολή κατεύθυνσης του παίχτη προστίθεται ένας επιπλέον έλεγχος ότι η μεταβλητή isGroundSliding είναι ψευδής. Οπότε ως υπενθύμιση της λειτουργίας αυτής της συνάρτησης εάν ο παίχτης γλιστράει ή κινείται προς τα πάνω ή προς τα κάτω δεν μπορεί να αλλάξει κατεύθυνση κίνησης με τα πλήκτρα a και d. Επιπρόσθετα για την απενεργοποίηση της λειτουργίας του παίχτη να μπορεί να γλιστρήσει στον αέρα στην συνάρτηση IA_jump στο σημείο που γίνεται ο έλεγχος εάν ο παίχτης πατάει μαζί με το space και το πλήκτρο s για την πραγματοποίηση του γλιστρήματος θα γίνει έλεγχος και από το character movement component με την συνάρτηση isMovingOnGround για να διαπιστωθεί εάν ο χαρακτήρας όντως βρίσκεται στο έδαφος και μόνο εάν η επιστροφή αυτής της συνάρτησης είναι αληθής και δηλαδή ο χαρακτήρας είνια πράγματι στο έδαφος θα μπορεί να πραγματοποιήσει το γλίστρημα. Βασικές έννοιες της κίνησης χαρακτήρα:

-Επιτάχυνση (Acceleration): Ο ρυθμός αλλαγής της ταχύτητας ενός χαρακτήρα. Στο UE5, η επιτάχυνση μπορεί να καθοριστεί μέσω των παραμέτρων επιτάχυνσης και μέγιστης επιτάχυνσης.

-Τριβή εδάφους (Ground Friction): Ο συντελεστής τριβής που επηρεάζει την ολίσθηση του χαρακτήρα στο έδαφος. Η τριβή εδάφους επηρεάζει την επιτάχυνση και επιβράδυνση του χαρακτήρα.

-Ταχύτητα (Velocity): Η ταχύτητα με την οποία κινείται ο χαρακτήρας σε μια δεδομένη στιγμή.

-Χρόνος (Delta Time): Το χρονικό διάστημα που περνά μεταξύ δύο πλαισίων (frames) του παιχνιδιού.

Εξίσωση κίνησης

Η εξίσωση που περιγράφει την κίνηση του χαρακτήρα στο έδαφος μπορεί να συνοψιστεί ως εξής:

Velocity=Velocity+(Acceleration−Friction)×Delta Time\text{Velocity} = \text{Velocity} + (\text{Acceleration} - \text{Friction}) \times \text{Delta Time}Velocity=Velocity+(Acceleration−Friction)×Delta Time

Όπου:

Velocity: Η τρέχουσα ταχύτητα του χαρακτήρα.
Acceleration: Η επιτάχυνση που εφαρμόζεται στον χαρακτήρα.
Friction: Η δύναμη τριβής που εφαρμόζεται στον χαρακτήρα.
Delta Time: Το χρονικό διάστημα μεταξύ των ενημερώσεων της κίνησης.

Υλοποίηση σε C++

Παρακάτω είναι ένα παράδειγμα κώδικα C++ για την υλοποίηση της κίνησης σε χαρακτήρα στο UE5 χρησιμοποιώντας την παράμετρο κίνησης χαρακτήρα.

```
// Συνάρτηση φυσικής περπατήματος του χαρακτήρα

ΣΥΝΑΡΤΗΣΗ PhysWalking(DeltaTime, Iterations)

    ΑΝ DeltaTime < ΕΛΑΧΙΣΤΟΣ_ΧΡΟΝΟΣ_ΕΝΗΜΕΡΩΣΗΣ ΤΟΤΕ

        ΕΠΙΣΤΡΟΦΗ

    ΤΕΛΟΣ ΑΝ


    // Υπολογισμός της επιτάχυνσης και της τριβής

    Acceleration = ΠάρεΤρέχουσαΕπιτάχυνση()

    Friction = ΠάρεΤριβήΕδάφους()


    // Ενημέρωση της ταχύτητας του χαρακτήρα

    Velocity = Velocity + (Acceleration - Friction) * DeltaTime


    // Περιορισμός της ταχύτητας στη μέγιστη επιτρεπόμενη ταχύτητα

    Velocity = Velocity.ΠεριορισμόςΣεΜέγιστηΤαχύτητα(ΠάρεΜέγιστηΤαχύτητα())


    // Εφαρμογή της νέας θέσης του χαρακτήρα

    Delta = Velocity * DeltaTime

    ΜετακίνησεΧαρακτήρα(Delta)


    // Ενημέρωση της θέσης και της ταχύτητας του χαρακτήρα

    ΕνημέρωσηΤαχύτηταςΣτοιχείου()

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Το επόμενο βήμα είναι η ακύρωση του γλιστρήματος με την ικανότητα αναπήδησης του παίχτη. Οπότε στη συνάρτηση apply root motion constant force with callbacks θα δημιουργηθεί μία νέα συνάρτηση με όνομα cancel η οποία θα ακυρώνει την λειτουργία της προηγούμενης σε περίπτωση που πατηθεί το πλήκτρο space και ο χαρακτήρας θα αναπηδάει. Η συνάρτηση Cancel σε μια ασύγχρονη δράση (async action) είναι σημαντική για τον έλεγχο και τη διαχείριση των ασύγχρονων διεργασιών, ειδικά όταν χρειάζεται να ακυρωθεί μια διεργασία που βρίσκεται σε εξέλιξη.

-Δημιουργία και Εγγραφή της Δράσης: Η μέθοδος RunAsyncAction δημιουργεί και εγγράφει την ασύγχρονη δράση με το παιχνίδι.

-Εκκίνηση της Ασύγχρονης Εργασίας: Η μέθοδος Activate χρησιμοποιεί το Async για να ξεκινήσει την εργασία σε ένα νήμα από το ThreadPool.

-Έλεγχος για Ακύρωση: Μέσα στον βρόχο της ασύγχρονης εργασίας, η μεταβλητή bIsCancelled ελέγχεται συνεχώς για να δει αν η εργασία πρέπει να σταματήσει.

-Ακύρωση της Δράσης: Η μέθοδος Cancel θέτει τη μεταβλητή bIsCancelled σε true, εκπέμπει το σήμα ακύρωσης και καλεί τη μέθοδο SetReadyToDestroy για να καθαρίσει τη δράση.

Οπότε μετά την ακύρωση του γλιστρήματος στο έδαφος τίθεται η μεταβλητή isGroundSliding σε ψευδής έτσι ώστε να σταματήσει να παίζει το animation του γλιστρήματος ενώ ο παίχτης δεν γλιστράει. Επίσης πρέπει να δημιουργηθεί και η λειτουργία ο χαρακτήρας να σταματάει το γλίστρημα όταν δέχεται ζημιά από εχθρούς. Έτσι στο γεγονός any damage που έχει δημιουργηθεί που ορίζει τη θα συμβεί όταν ο παίχτης δέχεται ζημιά και την διαδικασία σαστίσματος, πρώτα θα υλοποιηθεί η συνάρτηση cancel slide, ώστε ο παίχτης σε περίπτωση που πραγματοποιεί γλίστρημα πρώτα να ακυρώνεται το γλίστρημα και μετά να υλοποιείται η υπόλοιπη λογική, ώστε να μην υπάρχουν προβλήματα – bugs. Η λογική της cancel έχει ήδη υλοποιηθεί και απλός γίνεται η επανάληψή της και σε αυτό το σημείο του κώδικα. Στη συνέχει πρέπει να εφαρμοστεί η λειτουργία της ακύρωσης του γλιστρήματος όταν ο χαρακτήρας πέφτει από κάποιο υψόμετρο. Έτσι θα γίνει δημιουργία μίας λογικής όπου θα ανιχνεύει όταν ο χαρακτήρας βρίσκεται κοντά σε κάποια άκρη και πάει να πέσει και στη συνέχεια θα εκτελείται πάλι η συνάρτηση cancel η οποία ακυρώνει το γλίστρημα. Η λογική παρατίθεται από κάτω:

Για να γίνει επεξήγηση της λειτουργίας της κίνησης ενός χαρακτήρα κατά το περπάτημα στην άκρη (ledge walking) αυτά είναι τα βασικά βήματα και οι έννοιες που εμπλέκονται.

Βασικές Έννοιες Κίνησης στην Άκρη

1. Έλεγχος Άκρης (Ledge Detection): Προσδιορισμός αν ο χαρακτήρας βρίσκεται κοντά στην άκρη μιας επιφάνειας.
2. Κίνηση κατά την Άκρη (Ledge Movement): Προσαρμογή της κίνησης του χαρακτήρα ώστε να μπορεί να περπατάει κατά μήκος της άκρης.
3. Αντιστάθμιση Θέσης και Προσανατολισμού: Διασφάλιση ότι ο χαρακτήρας παραμένει σωστά ευθυγραμμισμένος κατά την κίνηση στην άκρη.

Επεξήγηση του Κώδικα

1. Ανίχνευση Άκρης (Ledge Detection): Η μέθοδος CanGrabLedge χρησιμοποιεί ένα LineTrace για να ανιχνεύσει αν υπάρχει άκρη μπροστά από τον χαρακτήρα.
2. Διαχείριση Κίνησης στην Άκρη (HandleLedgeWalking): Αυτή η μέθοδος χειρίζεται την πραγματική κίνηση του χαρακτήρα κατά μήκος της άκρης, προσαρμόζοντας τη θέση του χαρακτήρα με βάση τον χρόνο (DeltaTime).
3. Έναρξη και Διακοπή Κίνησης στην Άκρη (StartLedgeWalking, StopLedgeWalking): Αυτές οι μέθοδοι ξεκινούν και σταματούν την κίνηση του χαρακτήρα κατά μήκος της άκρης, αντίστοιχα.

Έτσι ο πλέον ο χαρακτήρας δεν κάνει γλίστρημα όταν πέφτει από κάποια άκρη, αλλά επιστρέφει στο κανονικό falling animation(κινησιολογία πτώσης) και εφαρμόζονται οι γνωστές παράμετροι φυσικής. Επιπλέον θα γίνει προσθήκη της λειτουργίας ο παίχτης να μπορεί να ακυρώσει το γλίστρημα αν πολύ απλά πατήσει το πλήκτρο για κίνηση προς την αντίθετη κατεύθυνση από αυτή που βρίσκεται ήδη ο παίχτης. Στην συνάρτηση TryWallSlide που έχει ήδη δημιουργηθεί έχει ήδη εφαρμοστεί ένα μέρος της λογικής όπου ο παίχτης μπορούσε να γραπωθεί στον τοίχο εάν η κατεύθυνση που κοίταζε ο χαρακτήρας ήταν η ίδια με την κατεύθυνση της εισόδου από το πληκτρολόγιο στον άξονα x. Οπότε με μία μικρή παραλλαγή θα γίνει έλεγχος εάν ο πολλαπλασιασμός των στοιχείων είναι μικρότερος του 0 αντί για μεγαλύτερος δηλαδή εάν η είσοδος είναι η ανάποδη της κατεύθυνσης.

Οπότε θα δημιουργηθεί νέα συνάρτηση η οποία θα καλείται ανά καρέ και εάν η μεταβλητή isGroundSliding είναι αληθής τότε σε συνδυασμό και με τον παραπάνω έλεγχο εάν ο παίχτης προσπαθήσει να κινηθεί προς την αντίθετη κατεύθυνση θα εφαρμόζεται η συνάρτηση cancel και θα ακυρώνεται το γλίστρημα. Ένα ακόμα ζήτημα που δημιουργείται είναι πως προς το παρών ο παίχτης μπορεί να πραγματοποιήσει διαδοχικά απεριόριστα γλιστρήματα, που δεν είναι επιθυμητό, έτσι στην λογική του γλιστρήματος στο σημείο ελέγχου του πλήκτρου s μία επιπλέον προϋπόθεση για το γλίστρημα είναι πως η μεταβλητή isGroundSliding θα πρέπει να είναι ψευδής. Δηλαδή ο παίχτης δεν θα μπορεί να πραγματοποιήσει επιπλέον γλίστρημα στο έδαφος εάν ήδη το κάνει.

4.5 Δημιουργία σημείων ελέγχου και επαναφοράς παίχτη

4.5.1 Επαναφορά παίχτη

Σε αυτό το σημείο της εργασίας ο χαρακτήρας που ελέγχει ο παίχτης μπορεί να καταστραφεί εάν δεχθεί υπερβολική ζημιά από τους εχθρούς, αλλά δεν έχει οριστεί κάποια λειτουργία για το τι θα συμβεί μετά. Επειδή καλείτε η συνάρτηση destroy actor, όταν ο χαρακτήρας καταστρέφεται η κλάση player controller δεν ξέρει τι να κάνει καθώς δεν υπάρχει πλέον χρησιμοποιήσιμος χαρακτήρας και έτσι το πρόγραμμα βγάζει πολλά λάθη και πρέπει να γίνει επανεκκίνηση του παιχνιδιού. Οπότε είναι πολύ σημαντικό σε αυτό το στάδιο να οριστεί η λειτουργία επαναφοράς του παίχτη στον κόσμο του παιχνιδιού, όχι μόνο για να μην καταρρεύσει το πρόγραμμα, αλλά και για να μπορεί ο χρήστης να συνεχίσει να παίζει χωρίς να χρειαστεί να κλείσει χειροκίνητα το πρόγραμμα και να το κάνει επανεκκίνηση κάθε φορά που χάνει. Οπότε για αρχή κάθε φορά που καταστρέφεται το στιγμιότυπο της κλάσης bp_player θα πρέπει να δημιουργείται ένα νέο στιγμιότυπο της κλάσης στην αρχική θέση PlayerStart που έχει οριστεί και να οριστεί μεταβιβαστεί ο έλεγχος από το παλιό χαρακτήρα στον καινούργιο. Θα μπορούσε να χρησιμοποιηθεί ως μέθοδος επαναφοράς του παίχτη η χρήση του ήδη υπάρχοντος ηττημένου χαρακτήρα και η μεταφορά του στην αρχική θέση με ανανεωμένη ζωή, αλλά η μέθοδος που θα χρησιμοποιηθεί εδώ είναι η πλήρης καταστροφή του παλιού χαρακτήρα και η δημιουργία ενός καινούργιου στον οποίο θα ανατεθεί ο έλεγχος από την κλάση player controller. Αυτή η μέθοδος είναι γενικά πιο ξεκάθαρη και εύκολη στην χρήση, καθώς στην συνέχεια θα χρησιμοποιηθεί και για την δημιουργία των σημείων επαναφοράς (checkpoints) από όπου ο παίχτης θα μπορεί να αναβιώνει εφόσον έχει καταφέρει να τα φτάσει. Οπότε δεν χρειάζεται να γίνεται κάθε φορά επαναφορά της ζωής και όλων των καταστάσεων που βρισκόταν ο παίχτης την ώρα του θανάτου του. Οπότε στην κλάση gm_action που είναι η κλάση ελέγχου των παραμέτρων του παιχνιδιού θα δημιουργηθεί μία συνάρτηση με όνομα respawnPlayer και εκεί θα υλοποιηθεί η λογική της συνάρτησης του παίχτη. Η συνάρτηση SpawnActor ανήκει στην κλάση UWorld και δημιουργεί ένα νέο αντικείμενο του τύπου AActor (ή παράγωγης κλάσης) στον κόσμο του παιχνιδιού. Η γενική μορφή της συνάρτησης είναι η εξής: template<class T>

T* SpawnActor(UClass* Class, const FTransform& Transform, const FActorSpawnParameters& SpawnParameters = FactorSpawnParameters());

Παράμετροι:

-*UClass Class**: Ο τύπος της κλάσης του ηθοποιού που θέλετε να δημιουργήσετε.

-FTransform& Transform: Η θέση, περιστροφή και κλίμακα που θέλετε να έχει ο νέος ηθοποιός.

-FActorSpawnParameters& SpawnParameters: Πρόσθετες παράμετροι για τον έλεγχο της διαδικασίας δημιουργίας.

// Συνάρτηση για την επανεμφάνιση του παίκτη

ΣΥΝΑΡΤΗΣΗ RespawnPlayer()

  ΑΝ PlayerCharacterClass != κενό ΤΟΤΕ

    // Ορισμός των συντεταγμένων για τη θέση επανεμφάνισης

    SpawnLocation = ΔημιουργίαVector(0.0, 0.0, 100.0)

    SpawnRotation = ΜηδενικήΠεριστροφή()

SpawnParameters.Κάτοχος = αυτό

SpawnParameters.Instigator = ΠάρεInstigator()

// Δημιουργία νέου παίκτη στη θέση επανεμφάνισης

NewPlayer = ΠάρεΚόσμο().ΔημιούργησεActor(PlayerCharacterClass, SpawnLocation, SpawnRotation, SpawnParameters)

ΑΝ NewPlayer υπάρχει ΤΟΤΕ

// Οποιαδήποτε επιπλέον ρύθμιση για τον νέο παίκτη

ΤΕΛΟΣ ΑΝ

ΤΕΛΟΣ ΑΝ

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

Εξήγηση του Κώδικα:

- FVector SpawnLocation: Ορίζεται η θέση επανεμφάνισης του παίκτη.
- FRotator SpawnRotation: Ορίζεται η αρχική περιστροφή του παίκτη.
- FActorSpawnParameters SpawnParameters: Ρυθμίσεις για την επανεμφάνιση, όπως ο ιδιοκτήτης και ο υποκινητής.
- SpawnActor: Δημιουργεί ένα νέο αντικείμενο της κλάσης AYourPlayerCharacter στη θέση και με τις ρυθμίσεις που ορίστηκαν.

Έτσι ως παράμετροι για την συνάρτηση αυτή θα εισαχθούν το deafult pawn class που είναι η κλάση bp_player ως το αντικείμενο που θα εμφανισθεί και επιλέγονται προς το παρών ως συντεταγμένες οι αρχικές. Έπειτα θα χρησιμοποιηθούν οι συναρτήσεις get player controller και posses ώστε η κλάση Player controller να αποκτήσει των έλεγχο του νέου χαρακτήρα. Η συνάρτηση GetPlayerController χρησιμοποιείται για να αποκτήσετε τον player controller από τον κόσμο του παιχνιδιού. Συνήθως καλείται από την κλάση του GameMode ή από την κλάση του χαρακτήρα. Η συνάρτηση Possess καλείται από τον player controller για να αποκτήσει τον έλεγχο ενός χαρακτήρα (character). Αυτή η συνάρτηση ενημερώνει τον εσωτερικό μηχανισμό του Unreal Engine για να συνδέσει τον controller με τον νέο χαρακτήρα.

// Συνάρτηση για την επανεμφάνιση του παίκτη

ΣΥΝΑΡΤΗΣΗ RespawnPlayer()

ΑΝ PlayerCharacterClass != κενό ΤΟΤΕ

// Ορισμός των συντεταγμένων για τη θέση επανεμφάνισης

SpawnLocation = ΔημιουργίαVector(0.0, 0.0, 100.0)

SpawnRotation = ΜηδενικήΠεριστροφή()

SpawnParameters.Κάτοχος = αυτό

SpawnParameters.Instigator = ΠάρεInstigator()

// Δημιουργία νέου παίκτη στη θέση επανεμφάνισης

NewPlayer = ΠάρεΚόσμο().ΔημιούργησεActor(PlayerCharacterClass, SpawnLocation, SpawnRotation, SpawnParameters)

ΑΝ NewPlayer υπάρχει ΤΟΤΕ

```
        // Απόκτηση του player controller
        PlayerController = ΠάρεΚόσμο().ΠάρεΠρώτοPlayerController()
        ΑΝ PlayerController υπάρχει ΤΟΤΕ
            // Απόκτηση του ελέγχου του νέου χαρακτήρα
            PlayerController.Έλεγξε(NewPlayer)
        ΤΕΛΟΣ ΑΝ
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

- Spawn Location and Rotation: Ορίζει τη θέση και την περιστροφή του νέου χαρακτήρα.
- SpawnActor: Δημιουργεί ένα νέο αντικείμενο της κλάσης AYourPlayerCharacter.
- GetFirstPlayerController: Αποκτά τον πρώτο player controller στον κόσμο του παιχνιδιού. Εναλλακτικά, μπορείτε να χρησιμοποιήσετε το UGameplayStatics::GetPlayerController για να αποκτήσετε τον controller συγκεκριμένου παίκτη.
- Possess: Ο player controller αποκτά τον έλεγχο του νέου χαρακτήρα.

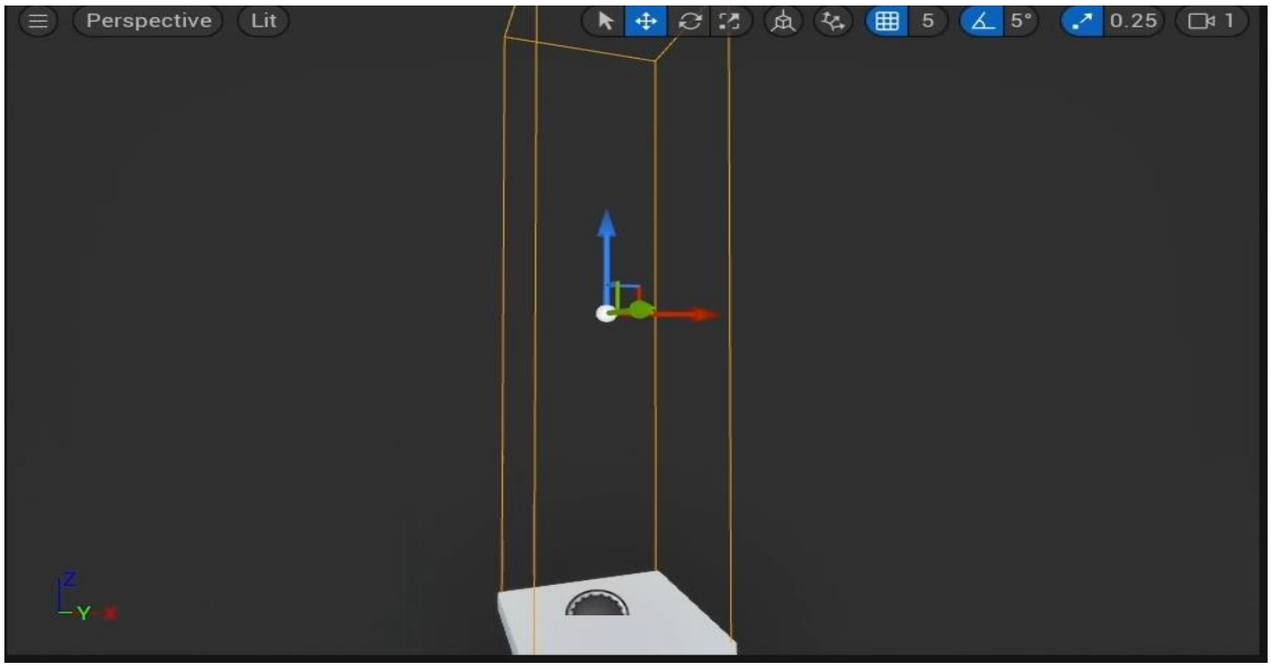
Τώρα που δημιουργήθηκε αυτή η συνάρτηση πρέπει να γίνει η κλήση της από το σωστό σημείο του κώδικα, δηλαδή από το σημείο ελέγχου που βλέπει εάν ο χαρακτήρας ηττήθηκε ή όχι. Αυτή η λογική έχει εφαρμοστεί στην κλάση bp_actionChar_base, όπου διαχειρίζεται την καταστροφή των εχθρών αλλά και του χαρακτήρα του παίχτη. Όμως μόνο ο παίχτης είναι επιθυμητό να μπορεί να αναγεννηθεί όταν πεθαίνει. Οπότε θα γίνει αντιγραφή της λογικής καταστροφής από την κλάση bp_actionChar_base στην κλάση bp_enemy_base η οποία ελέγχει μόνο τις εχθρικές μονάδες, έτσι όλοι οι εχθροί καταστρέφονται κατευθείαν και γίνεται διαχωρισμός μεταξύ της λειτουργίας καταστροφής του παίχτη και τω εχθρών. Έπειτα θα γίνει αντιγραφή της λογικής καταστροφής και στην κλάση του παίχτη bp_player, όμως τώρα θα χρησιμοποιηθούν κάποια επιμέρους παράμετροι ώστε να μπορεί ο παίχτης να αναγεννιέται. Οπότε θα γίνει κλήση της κλάσης gm_action με αναφορά με την μέθοδο casting για να έχει πρόσβαση η κλάση bp_player στην συνάρτηση της επαναφοράς παίχτη αυτής της κλάσης. Οπότε μετά την συνάρτηση destroy actor θα γίνει κλήση της συνάρτησης respawn player και ο παίχτης πλέον όποτε καταστρέφεται γίνεται η επαναφορά του στην αρχή του κόσμου του παιχνιδιού. Κάτι που πρέπει να διορθωθεί είναι πως ο παίχτης θα μπορεί να ηττηθεί μόνο μία φορά και όχι πολλαπλές φορές, οπότε στην κλάση bp_actionChar_base πριν την λογική που ο παίχτης δέχεται ζημιά θα προστεθεί ο έλεγχος πως η μεταβλητή isDefeated του bpc vitality component θα είναι ψευδής, δηλαδή ο παίχτης δεν θα είναι ηττημένος. Εάν ο παίχτης ηττηθεί δηλαδή η ζωή του είναι μικρότερη ή ίση του μηδενός τότε ο παίχτης θα σταματήσει να δέχεται ζημιά και δεν θα επαναφέρει απεριόριστους παίχτες. Όμως ο παίχτης που καταστρέφεται συνεχίζει να υπάρχει στον κόσμο του παιχνιδιού και αυτό πρέπει να διορθωθεί. Οπότε στην κλάση Bp_player στην λογική που εφαρμόζεται η λήψη ζημιάς στον παίχτη στο event any damage θα γίνει έλεγχος πρώτα για την μεταβλητή isDefeated εάν είναι ψευδής, εφόσον ο παίχτης είναι ηττημένος τότε θα σταματήσουν να εκτελούνται όλες οι καταστάσεις λήψης ζημιάς μετά τον θάνατό του που προηγουμένως εκτελούνταν απεριόριστα. Τώρα σταματούν να παίζουν όλα τα εφέ λήψης ζημιάς αλλά ο ηττημένος παίχτης συνεχίζει να υπάρχει στον κόσμο του παιχνιδιού. Στην κλάση bp_player μετά την συνάρτηση respawn player θα προστεθεί η συνάρτηση destroy actor για τον παλιό χαρακτήρα και έτσι πλέον λύθηκε και το τελευταίο πρόβλημα. Πριν την κλήση της respawn player θα τεθεί στο capsule component η ρύθμιση σύγκρουσης σε μηδέν για να μην μπορεί καμία οντότητα να αλληλεπιδράσει με τον χαρακτήρα αφού ηττηθεί. Έτσι μόλις ο παίχτης καταστρέφεται αρχίζει και πέφτει μέσα από το πάτωμα στο κενό. Έτσι στη συνέχεια ορίζεται η ρύθμιση στο spring arm component που έχει την κάμερα του παίχτη η ρύθμιση της τοποθεσίας και της περιστροφής σε keep world, δηλαδή να παραμείνουν στην θέση του ς και να μην ακολουθήσουν τον παίχτη μέσα από το πάτωμα. Έπειτα στην any damage, δηλαδή την συνάρτηση που ελέγχει τη λειτουργία αφού δεχθεί ζημιά ο παίχτης, στον έλεγχο εάν ο παίχτης είναι ηττημένος στην περίπτωση που έχει ηττηθεί θα προστεθεί η συνάρτηση kockback που ήδη υπάρχει και σε αυτό το σημείο του κώδικα, για να προστεθεί ένα παραπάνω εφέ κατά την καταστροφή του παίχτη. Έπειτα πριν την συνάρτηση respawn θα προστεθεί μία συνάρτηση delay 2 δευτερολέπτων έτσι ώστε να μην επαναφέρεται

ο παίχτης άμεσα που βοηθάει και αισθητικά , αλλά και πρακτικά δίνοντας χρόνο στις διεργασίες που χρησιμοποιούσαν τον παλιό παίχτη να τελειώσουν και έτσι να γίνει αποφυγή διάφορων σφαλμάτων. Έπειτα πριν το delay για τον εμπλουτισμό του εφέ θανάτου θα χρησιμοποιηθεί και μία συνάρτηση camera fade η οποία θα σκουραίνει τον φωτισμό της οθόνης, καθώς ο παίχτης πεθαίνει.

```
// Συνάρτηση για την έναρξη του camera fade
ΣΥΝΑΡΤΗΣΗ StartCameraFade()
    // Fade out (μαύρισμα)
    ΞεκίναCameraFade(0.0, 1.0, 1.0, ΜαύροΧρώμα, ψευδές, αληθές)

    // Αναμονή για 1 δευτερόλεπτο πριν το fade in
    TimerHandle = ΔημιούργησεTimer()
    ΠάρεΚόσμο().ΠάρεΔιαχειριστήΧρονομέτρου().ΘέσεTimer(TimerHandle, αυτό, Κάλεσε(FadeIn), 1.0, ψευδές)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για το fade in (επαναφορά)
ΣΥΝΑΡΤΗΣΗ FadeIn()
    // Fade in (επαναφορά)
    ΞεκίναCameraFade(1.0, 0.0, 1.0, ΜαύροΧρώμα, ψευδές, ψευδές)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

4.5.2 Δημιουργία σημείων ελέγχου- επαναφοράς (checkpoints)

Τώρα θα πραγματοποιηθεί η δημιουργία σημείων ελέγχου γνωστά και ως checkpoints στα οποία μόλις τα φτάνει ο παίχτης θα αποθηκεύεται η πρόοδός του ώστε όταν πεθαίνει να μπορεί να αναβιώνει από το τελευταίο σημείο ελέγχου που έχει φτάσει και να συνεχίζει το παιχνίδι του από εκεί. Οπότε θα γίνει η δημιουργία ενός νέου φακέλου με όνομα props και θα δημιουργηθεί μία νέα κλάση τύπου actor με όνομα bp_checkpoint. Σε αυτή την κλάση θα δημιουργηθεί ένα box collision component με όνομα trigger. Να σημειωθεί πως όλα αυτά συμβαίνουν στον editor γραφικών του UE5, όπως φαίνεται στην εικόνα 21. Το checkpoint δεν θα είναι ορατό για τον παίχτη και δεν θα μπορεί ο παίχτης ή καμία άλλη οντότητα να αλληλεπιδράσει μαζί του, οπότε ορίζονται όλες οι ρυθμίσεις σύγκρουσης σε αγνόηση, εκτός από την ρύθμιση Pawn collision δηλαδή η σύγκρουση με τον παίχτη που τίθεται στην επιλογή overlap, δηλαδή να επιτρέπεται η αλληλεπίδραση. Έπειτα θα δημιουργηθεί ένα Overlap event όπως έχει γίνει και προηγουμένως στην εργασία και πραγματοποιείται cast στην κλάση Bp_player, όπως φαίνεται παρακάτω.



Εικόνα 21: Γραφική ανάδειξη του checkpoint

// Κλάση Checkpoint

ΚΛΑΣΗ ACheckpoint ΚΛΗΡΟΝΟΜΕΙ από AActor

ΔΗΜΟΣΙΑ:

// Ορίζει τις προεπιλεγμένες τιμές για αυτόν τον actor

ΣΥΝΑΡΤΗΣΗ ACheckpoint()

ΠΡΟΣΤΑΤΕΥΜΕΝΑ:

// Καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο actor

ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

ΔΗΜΟΣΙΑ:

// Καλείται κάθε frame

ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ

// Συστατικό για ανίχνευση σύγκρουσης

ΙΔΙΟΤΗΤΑ BoxComponent ΟρατήΠαντού, ΜόνοΓιαΑνάγνωση, Κατηγορία "Components"

// Συνάρτηση για τη διαχείριση των γεγονότων επικαλύψεων

ΣΥΝΑΡΤΗΣΗ OnOverlapBegin(OverlappedComp, OtherActor, OtherComp, OtherBodyIndex, bFromSweep, SweepResult)

ΤΕΛΟΣ ΚΛΑΣΗΣ

```
// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές
ΣΥΝΑΡΤΗΣΗ ACheckpoint()
    PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές
    BoxComponent = ΔημιουργίαDefaultSubobject("BoxComponent")
    RootComponent = BoxComponent

    // Ρύθμιση του γεγονότος επικαλύψεων
    BoxComponent.ΌτανΑρχίζειΕπικάλυψη.ΠρόσθεσεΔυναμική(αυτό, Κάλεσε(OnOverlapBegin))
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται όταν ξεκινάει το παιχνίδι ή γεννιέται ο actor
ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για τη διαχείριση των γεγονότων επικαλύψεων
ΣΥΝΑΡΤΗΣΗ OnOverlapBegin(OverlappedComp, OtherActor, OtherComp, OtherBodyIndex, bFromSweep, SweepResult)
    // Μετατροπή του OtherActor σε BP_Player
    Player = ΚάνεCastΣεBP_Player(OtherActor)
    ΑΝ Player υπάρχει ΤΟΤΕ
        // Αν το cast είναι επιτυχές, κάνε ό,τι χρειάζεται για το checkpoint
        ΚαταγραφήΜηνύματος(Warning, "Player reached checkpoint!")
        // Παράδειγμα: Αποθήκευση προόδου ή ενεργοποίηση λογικής
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Η παραπάνω λειτουργία θα υλοποιηθεί μόνο για την πρώτη φορά που θα περάσει ο παίχτης από το σημείο και δεν θα ενεργοποιείται τις υπόλοιπες φορές. Στη συνέχεια θα γίνει αναφορά στην κλάση gm_action με μέθοδο casting της bp_checkpoint και θα περαστεί στην κλάση gm game mode με αναφορά το τρέχον checkpoint.

```
// Κλάση Checkpoint
ΚΛΑΣΗ ACheckpoint ΚΛΗΡΟΝΟΜΕΙ από AActor
    ΔΗΜΟΣΙΑ:
        // Ορίζει τις προεπιλεγμένες τιμές για αυτόν τον actor
```

ΣΥΝΑΡΤΗΣΗ ACheckpoint()

ΠΡΟΣΤΑΤΕΥΜΕΝΑ:

// Καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο actor

ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

ΔΗΜΟΣΙΑ:

// Καλείται κάθε frame

ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ

// Συστατικό για ανίχνευση σύγκρουσης

ΙΔΙΟΤΗΤΑ BoxComponent ΟρατήΠαντού, ΜόνοΓιαΑνάγνωση, Κατηγορία "Components"

// Συνάρτηση για διαχείριση γεγονότων επικαλύψεων

ΣΥΝΑΡΤΗΣΗ OnOverlapBegin(OverlappedComp, OtherActor, OtherComp, OtherBodyIndex, bFromSweep, SweepResult)

ΙΔΙΩΤΙΚΑ:

// Boolean για έλεγχο αν η ενέργεια έχει εκτελεστεί

ΙΔΙΟΤΗΤΑ bHasActionBeenPerformed

// Συνάρτηση για εκτέλεση ενέργειας μία φορά

ΣΥΝΑΡΤΗΣΗ PerformActionOnce()

ΤΕΛΟΣ ΚΛΑΣΗΣ

// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές

ΣΥΝΑΡΤΗΣΗ ACheckpoint()

PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές

BoxComponent = ΔημιουργίαDefaultSubobject("BoxComponent")

RootComponent = BoxComponent

// Ρύθμιση του γεγονότος επικαλύψεων

BoxComponent.ΌτανΑρχίζειΕπικάλυψη.ΠρόσθεσεΔυναμική(αυτό, Κάλεσε(OnOverlapBegin))

// Αρχικοποίηση του boolean σε ψευδές

bHasActionBeenPerformed = ψευδές

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

```
// Συνάρτηση που καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο actor
ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για διαχείριση γεγονότων επικαλύψεων
ΣΥΝΑΡΤΗΣΗ OnOverlapBegin(OverlappedComp, OtherActor, OtherComp, OtherBodyIndex, bFromSweep, SweepResult)
    // Μετατροπή του OtherActor σε BP_Player
    Player = ΚάνεCastΣεBP_Player(OtherActor)
    ΑΝ Player υπάρχει ΤΟΤΕ
        ΕκτέλεσεPerformActionOnce()
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για εκτέλεση ενέργειας μία φορά
ΣΥΝΑΡΤΗΣΗ PerformActionOnce()
    // Έλεγχος αν η ενέργεια έχει ήδη εκτελεστεί
    ΑΝ bHasActionBeenPerformed == ψευδές ΤΟΤΕ
        // Εκτέλεση της ενέργειας εδώ
        ΚαταγραφήΜηνύματος(Warning, "Checkpoint reached!")

        // Αναφορά στο GameMode
        GameMode = ΚάνεCastΣεYourGameMode(ΠάρεGameMode(αυτό))
        ΑΝ GameMode υπάρχει ΤΟΤΕ
            GameMode.ΘέσεCurrentCheckpoint(αυτό)
        ΤΕΛΟΣ ΑΝ

        // Θέσε το boolean σε αληθές για να αποτρέψεις μελλοντικές εκτελέσεις
        bHasActionBeenPerformed = αληθές
    ΤΕΛΟΣ ΑΝ
```

```
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ


// Κλάση GameMode
ΚΛΑΣΗ AYourGameMode ΚΛΗΡΟΝΟΜΕΙ από AGameModeBase
  ΔΗΜΟΣΙΑ:
    // Συνάρτηση για ρύθμιση του τρέχοντος checkpoint
    ΣΥΝΑΡΤΗΣΗ ΘέσεCurrentCheckpoint(NewCheckpoint)

  ΙΔΙΩΤΙΚΑ:
    // Το τρέχον checkpoint
    ΙΔΙΟΤΗΤΑ CurrentCheckpoint
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Συνάρτηση για ρύθμιση του τρέχοντος checkpoint
ΣΥΝΑΡΤΗΣΗ ΘέσεCurrentCheckpoint(NewCheckpoint)
  ΑΝ NewCheckpoint υπάρχει ΤΟΤΕ
    CurrentCheckpoint = NewCheckpoint
    ΚαταγραφήΜηνύματος(Warning, "Current checkpoint set!")
  ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

- ACheckpoint::PerformActionOnce(): Ελέγχει αν η ενέργεια έχει ήδη εκτελεστεί χρησιμοποιώντας τη boolean μεταβλητή bHasActionBeenPerformed. Αν όχι, εκτελεί την ενέργεια και θέτει την μεταβλητή σε true.
- ACheckpoint::OnOverlapBegin(): Καλεί τη PerformActionOnce() όταν ο παίκτης εισέλθει στην περιοχή του BoxComponent.
- AYourGameMode::SetCurrentCheckpoint(): Ορίζει το τρέχον checkpoint στο GameMode.
- Περάστε Αναφορά στο Checkpoint: Η συνάρτηση PerformActionOnce() καλεί τη
- SetCurrentCheckpoint() του GameMode για να περάσει την αναφορά του τρέχοντος checkpoint.

Οπότε μετά εντός της κλάσης gm_gamemode στην συνάρτηση respan player πρέπει να οριστεί σε ποιο σημείο θα γίνει η επαναφορά του παίχτη. Εάν ο παίχτης θα αναγεννηθεί στην αρχική θέση ή σε κάποιο checkpoint. Οπότε στην αρχή της λογικής γίνεται έλεγχος σε περίπτωση που υπάρχει κάποιο active checkpoint , δηλαδή αν έχει περάσει κάποιο checkpoint με αναφορά στην κλάση. Εάν όχι τότε θα κάνει respawn στην αρχή αλλιώς θα κάνει επαναφορά στο τελευταίο checkpoint. Οπότε γίνεται αντιγραφή της συνάρτησης respawn player με τις παραμέτρους συντεταγμένων του active checkpoint έναντι των συντεταγμένων της αρχής. Το πρόβλημα που δημιουργείται μετά είναι πως πρέπει να οριστεί το κατάλληλο ύψος στις συντεταγμένες ώστε ο παίχτης να μην κάνει respawn στον αέρα ή μέσα στο πάτωμα. Αυτό διορθώνεται εύκολα κάνοντας απλός κάποιες πειραματικές δοκιμές στις συντεταγμένες. Οπότε προστίθεται εντός του κουτιού ένα scene component στο κατάλληλο ύψος και στην κατάλληλη φορά και χρησιμοποιούνται οι συντεταγμένες του στον χώρο ως αναφορά για το σημείο και την κατεύθυνση επαναφοράς του παίχτη. Τέλος πρέπει να αντιμετωπιστεί το πρόβλημα πως ο παίχτης δεν μπορεί να

αναγεννηθεί στην περίπτωση που συμβεί κάποιο σφάλμα και πέσει εκτός του κόσμου παιχνιδιού. Σε περίπτωση που συνέβαινε αυτό θα έπρεπε ο παίχτης να κάνει ολική επανεκκίνηση του παιχνιδιού. Για αυτό το λόγο θα εφαρμοστεί λογική που εντοπίζει εάν ο παίχτης πέφτει ασταμάτητα και προκαλεί αυτόματα την καταστροφή του. Οπότε στον φάκελο props θα δημιουργηθεί μία νέα κλάση τύπου actor με όνομα fallDetector. Σε αυτή θα γίνει η δημιουργία ενός box collision component όπως και πριν με το checkpoint με μοναδική ρύθμιση σύγκρουσης το Overlap με τον παίχτη. Έπειτα θα δημιουργηθεί ένα overlap event και θα γίνει cast στην κλάση bp_player , ώστε να ενεργοποιείται μόνο όταν εντοπίζει αντικείμενο της κλάσης του παίχτη, όπως έγινε και με το checkpoint. Έπειτα γίνεται κλάση της συνάρτησης apply damage που υπάρχει ήδη με παράμετρο ζημιάς μία πολύ μεγάλη τιμή που σκοτώνει απευθείας τον παίχτη. Έτσι γίνεται τοποθέτησή αυτού του γραφικού όπου είναι επιθυμητό ο παίχτης να δέχεται μεγάλη ζημιά πτώσης, καθώς και κάτω από τον κόσμο του παιχνιδιού , έτσι ώστε ο να μπορεί να γίνεται η επαναφορά του σε κοντινά σημεία ή να πεθαίνει όταν πέφτει σε σημεία με παγίδες.

4.6 Δημιουργία πρώτου εχθρού με συμπεριφορά τεχνητής νοημοσύνης

4.6.1 Δημιουργία πρώτου εχθρού - καβούρι

Σε αυτή την ενότητα θα γίνει η δημιουργία των εχθρών και η ενσωμάτωση συμπεριφοράς σε αυτούς. Στο UE5 για να επιτευχθεί η δημιουργία της συμπεριφοράς των εχθρών χρειάζεται η δημιουργία μία κλάσης τύπου AI_Controller η οποία θα χειρίζεται τις εχθρικές μονάδες. Έτσι θα δημιουργηθεί η κλάση AIC_Base και το παιδί της AIC_Crab. Η πρώτη είναι μία γενική κλάση γονέας η οποία χειρίζεται όλα τα κοινά χαρακτηριστικά των εχθρών, ενώ η δεύτερη είναι η πρώτη κλάση που αντιπροσωπεύει την εχθρική μονάδα του καβουριού. Η λειτουργία του πρώτου εχθρού θα είναι πολύ απλή. Ο εχθρός αυτός θα κινείτε είτε δεξιά είτε αριστερά προκαλώντας ζημιά κάθε φορά που έρχεται σε επαφή με τον παίχτη και θα έχει την ικανότητα να εντοπίζει τοίχους αλλά και γκρεμούς ώστε να μπορεί να αλλάζει κατεύθυνση και να συνεχίζει την πορεία του. Οπότε αρχικά στην κλάση AIC_Crab θα γίνεται έλεγχος εάν το AI controller του UE5 ελέγχει την κλάση με εχθρού με όνομα BP_enemy_crab. Ακολουθεί παράδειγμα:

```
// Κλάση EnemyAIController
ΚΛΑΣΗ AEnemyAIController ΚΛΗΡΟΝΟΜΕΙ από AAIController
    ΠΡΟΣΤΑΤΕΥΜΕΝΑ:
        // Καλείται όταν το AI Controller παίρνει τον έλεγχο ενός Pawn
        ΣΥΝΑΡΤΗΣΗ OnPossess(InPawn) ΥΠΕΡΚΑΛΥΠΤΕΙ
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Συνάρτηση OnPossess που καλείται όταν το AI Controller παίρνει τον έλεγχο ενός Pawn
ΣΥΝΑΡΤΗΣΗ OnPossess(InPawn)
    ΚάλεσεSuperOnPossess(InPawn)

    // Έλεγχος αν το Pawn είναι της κλάσης εχθρού
    EnemyCharacter = ΚάνεCastΣεYourEnemyCharacter(InPawn)
    ΑΝ EnemyCharacter υπάρχει ΤΟΤΕ
        // Εκτέλεση λογικής όταν το AI Controller παίρνει τον έλεγχο του εχθρού
```

ΚαταγραφήΜηνύματος(Warning, "AI Controller has possessed the enemy: %s", ΠάρεΌνομα(EnemyCharacter))

// Παράδειγμα: Ξεκινήστε μια συμπεριφορά όπως περιπολία ή επίθεση

// EnemyCharacter.ΞεκίνησεBehaviorTree()

ΤΕΛΟΣ ΑΝ

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

Στην κλάση bp_enemy_crab, στις ρυθμίσεις AI Controller Class, θα οριστεί η AIC_Crab. Έτσι, η συγκεκριμένη κλάση θα έχει πλέον τον έλεγχο του πρώτου εχθρού. Στη συνέχεια, θα οριστεί η ρύθμιση Auto Possess AI σε "Placed in World or Spawned", ώστε να διασφαλιστεί ότι η κλάση AIC_Crab θα έχει άμεσο έλεγχο του καβουριού-εχθρού σε κάθε περίπτωση. Η ίδια αλλαγή θα γίνει και την κλάση BP_enemy_base ώστε να οριστεί αυτή η ρύθμιση για όλες τις κλάσεις εχθρών. Οπότε στην κλάση bp_enemy_crab θα γίνει η προσθήκη κίνησης. Έτσι με μία συνάρτηση με όνομα add movement input και που δέχεται ως παράμετρο το forward vector της enemy_crab το καβούρι πλέον μπορεί να κινείτε ευθεία. Γίνεται χρήση της getActorForwardVector για να βρεθεί η κατεύθυνση προς τα εμπρός του εχθρού και AddMovementInput για να δοθεί η κίνηση.

// Κλάση YourCharacter

ΚΛΑΣΗ AYourCharacter ΚΛΗΡΟΝΟΜΕΙ από ACharacter

ΔΗΜΟΣΙΑ:

// Ορίζει προεπιλεγμένες τιμές για τις ιδιότητες του χαρακτήρα

ΣΥΝΑΡΤΗΣΗ AYourCharacter()

ΠΡΟΣΤΑΤΕΥΜΕΝΑ:

// Καλείται όταν ξεκινά το παιχνίδι ή όταν γεννιέται ο χαρακτήρας

ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

ΔΗΜΟΣΙΑ:

// Καλείται κάθε frame

ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ

// Καλείται για να συνδέσει τη λειτουργία με το input

ΣΥΝΑΡΤΗΣΗ SetupPlayerInputComponent(PlayerInputComponent) ΥΠΕΡΚΑΛΥΠΤΕΙ

// Προσαρμοσμένη συνάρτηση για κίνηση προς τα εμπρός

ΣΥΝΑΡΤΗΣΗ WalkForward(ScaleValue)

ΤΕΛΟΣ ΚΛΑΣΗΣ

// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές

ΣΥΝΑΡΤΗΣΗ AYourCharacter()

PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές

```
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται για να συνδέσει λειτουργικότητα με το input
ΣΥΝΑΡΤΗΣΗ SetupPlayerInputComponent(PlayerInputComponent)
    ΚάλεσεSuperSetupPlayerInputComponent(PlayerInputComponent)

    // Σύνδεση της συνάρτησης WalkForward με το input
    PlayerInputComponent.ΔέσμευσηΆξονα("MoveForward", αυτό, Κάλεσε(WalkForward))
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για την κίνηση προς τα εμπρός
ΣΥΝΑΡΤΗΣΗ WalkForward(ScaleValue)
    ΑΝ Controller != κενό ΚΑΙ ScaleValue != 0.0 ΤΟΤΕ
        // Πάρε τον διάνυσμα κατεύθυνσης προς τα εμπρός
        Direction = ΠάρεΔιάνυσμαΕμπρός()

        // Πρόσθεσε κίνηση προς την κατεύθυνση
        ΠρόσθεσεΚίνηση(Direction, ScaleValue)
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

AYourCharacter: Κλάση που κληρονομεί από την ACharacter, η οποία παρέχει βασική λειτουργικότητα για χαρακτήρες με κίνηση.

WalkForward(float ScaleValue): Συνάρτηση που δέχεται ως παράμετρο την τιμή κλίμακας (ScaleValue) και χρησιμοποιείται για να προσθέσει την κίνηση προς τα εμπρός.
GetActorForwardVector(): Παίρνει τον διανυσματικό προσανατολισμό του ηθοποιού (actor) προς τα εμπρός.
AddMovementInput(FVector Direction, float ScaleValue): Προσθέτει την κίνηση προς την κατεύθυνση του διανύσματος Direction με την κλίμακα ScaleValue.
BindAxis("MoveForward", this, &AYourCharacter::WalkForward): Συνδέει την είσοδο του πληκτρολογίου (ή άλλου input) με τη συνάρτηση WalkForward.

Οπότε αυτές οι συναρτήσεις θα χρησιμοποιηθούν στην κλάση AIC_Crab για τον προσδιορισμό της κίνησης του καβουριού και θα τρέχουν για κάθε καρέ. Στο character movement component της κλάσης bp_enemy_crab θα οριστεί το max walk speed σε 120cm/s. Στην ίδια κλάση θα δημιουργηθεί μία νέα συνάρτηση που θα καλείται ανά καρέ με όνομα UpdateAnim. Οπότε εδώ θα οριστεί μία λογική παρόμοια με αυτή στις κινήσεις του παίχτη για εναλλαγή μεταξύ των καταστάσεων της κίνησης του καβουριού. Όμως επειδή σε αυτή την περίπτωση το καβούρι έχει μόνο δύο κινήσεις μία απλή συνάρτηση είναι αρκετή για αυτό τον έλεγχο. Οι κινήσεις αυτές είναι είτε το περπάτημα είτε η στάση. Οπότε θα γίνεται έλεγχος εάν η μεταβλητή velocity έχει τιμή >0, τότε θα παίζει το animation του περπατήματος, αλλιώς θα παίζει το animation της στάσης.

```
// Κλάση YourCharacter

ΚΛΑΣΗ AYourCharacter ΚΛΗΡΟΝΟΜΕΙ από ACharacter

    ΔΗΜΟΣΙΑ:

        // Ορίζει προεπιλεγμένες τιμές για τον χαρακτήρα

        ΣΥΝΑΡΤΗΣΗ AYourCharacter()


    ΠΡΟΣΤΑΤΕΥΜΕΝΑ:

        // Καλείται όταν ξεκινά το παιχνίδι ή όταν γεννιέται ο χαρακτήρας

        ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ


    ΔΗΜΟΣΙΑ:

        // Καλείται κάθε frame

        ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ


        // Καλείται για να συνδέσει τη λειτουργικότητα με το input

        ΣΥΝΑΡΤΗΣΗ SetupPlayerInputComponent(PlayerInputComponent) ΥΠΕΡΚΑΛΥΠΤΕΙ


        // Συνάρτηση για ενημέρωση της κίνησης

        ΣΥΝΑΡΤΗΣΗ UpdateAnimation()

ΤΕΛΟΣ ΚΛΑΣΗΣ


// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές

ΣΥΝΑΡΤΗΣΗ AYourCharacter()

    PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές

    Sprite = ΔημιουργίαDefaultSubobject("Sprite")

    Sprite.ΣύνδεσηΜε(RootComponent)

ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ


// Συνάρτηση που καλείται όταν ξεκινά το παιχνίδι ή όταν γεννιέται ο χαρακτήρας

ΣΥΝΑΡΤΗΣΗ BeginPlay()

    ΚάλεσεSuperBeginPlay()
```

```
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)
    ΕνημέρωσηΚίνησης()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται για να συνδέσει λειτουργικότητα με το input
ΣΥΝΑΡΤΗΣΗ SetupPlayerInputComponent(PlayerInputComponent)
    ΚάλεσεSuperSetupPlayerInputComponent(PlayerInputComponent)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για την ενημέρωση της κίνησης
ΣΥΝΑΡΤΗΣΗ UpdateAnimation()
    // Λήψη ταχύτητας του χαρακτήρα
    Velocity = ΠάρεΤαχύτητα()

    // Υπολογισμός της ταχύτητας (μήκος του διανύσματος ταχύτητας)
    Speed = Ταχύτητα.Μέγεθος()

    // Προσδιορισμός κατάλληλου animation βάσει της ταχύτητας
    DesiredAnimation = ΑΝ Speed > 0.0 ΤΟΤΕ RunAnimation ΑΛΛΙΩΣ IdleAnimation

    // Ορισμός του animation
    ΑΝ Sprite ΚΑΙ DesiredAnimation ΚΑΙ Sprite.ΠάρεFlipbook() != DesiredAnimation ΤΟΤΕ
        Sprite.ΟρισμόςFlipbook(DesiredAnimation)
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Για τον εντοπισμό εμποδίων ή τοίχων ώστε να αλλάζει το καβούρι κατεύθυνση περπατήματος θα χρησιμοποιηθεί ένα box collision. Οπότε όπως και προηγουμένως θα δημιουργηθεί γραφικά ένα κουτί το οποίο θα τοποθετηθεί μπροστά από το καβούρι και θα εντοπίζει με τη σύγκρουση πότε πρέπει να αλλάξει κατεύθυνση. Οπότε στην ρυθμίσεις collision θα ενεργοποιηθεί μόνο η world static για ακίνητα αντικείμενα και θα δημιουργηθεί συνάρτηση turn character για τη σύγκρουση με άλλα αντικείμενα στο box collision component. Οπότε αυτή η συνάρτηση θα ελέγχει το πρόσημο της κατεύθυνσης του εχθρού και θα το αντιστρέφει για να αλλάξει κατεύθυνση. Έπειτα θα γίνει αλλαγή της ρύθμισης σύγκρουσης και για world dynamic, ώστε να αλλάζει κατεύθυνση το καβούρι και όταν έρχεται σε επαφή με τον παίχτη ή άλλον εχθρό. Παρόμοια με το box collision θα δημιουργηθεί ένα ledgeDetector γραφικά ως ένα αόρατο κουτί που βρίσκεται μπροστά και κάτω από το καβούρι το οποίο θα έχει collision μόνο

με world static. Εδώ θα χρησιμοποιηθεί η αντίστροφη λογική και θα καλείται η συνάρτηση turn character όποτε το ledgeDetector δεν συγκρούεται με κάτι, δηλαδή όταν δεν υπάρχει έδαφος κάτω από το καβούρι.

4.6.2 Δημιουργία εχθρού – σαύρας

Σε αυτό το στάδιο της εργασίας θα γίνει η υλοποίηση του εχθρού της σαύρας. Αυτός ο εχθρός αν και είναι στατικός και δεν θα έχει τη δυνατότητα κίνησης θα μπορεί να εκτοξεύει βολές σε μία ευθεία τις οποίες ο παίχτης θα πρέπει να αποφεύγει για να μην δεχθεί ζημιά. Όπως και με τον πρώτο εχθρό η κλάση της σαύρας bp_enemy_lizard θα είναι παιδί της κλάσης bp_enemy_base. Οπότε αρχικά θα γίνει η ένταξη του sprite της σαύρας και θα γίνει προσαρμογή του capsule component ώστε να ταιριάζει στις διαστάσεις του sprite. Εφόσον έχει δημιουργηθεί το vitality component στην βασική κλάση του εχθρού και η σαύρα θα έχει πρόσβαση σε αυτό το component και ορίζεται η μέγιστη ζωή της στην τιμή 3. Οπότε στη συνέχεια θα γίνει η δημιουργία της κλάσης της βολής της σαύρας. Έτσι γίνεται η δημιουργία της κλάσης bp_enemy_projectile_base που θα είναι παιδί της projectile_base που υπάρχει ήδη και μετέπειτα θα δημιουργηθεί η κλάση bp_projectile_lizard ως παιδί της πρώτης. Στην τελευταία θα προστεθεί το flipbook με το animation της σφαίρας που θα εκτοξεύει η σαύρα και θα γίνει ρύθμιση του μεγέθους του sphere component που κληρονομήθηκε ώστε να ταιριάζει με αυτό της σφαίρας. Στη συνέχεια στην κλάση της σαύρας θα δημιουργηθεί συνάρτηση με όνομα shootFireball η οποία θα εμφανίζει την σφαίρα καλώντας ένα στιγμιότυπο της κλάσης projectile_lizard με την συνάρτηση spawnActor που έχει χρησιμοποιηθεί πολλές φορές προηγουμένως. Ως παραμέτρους για το σημείο εμφάνισης θα οριστούν η τοποθεσία και η κατεύθυνση που βρίσκεται σε εκείνο το καρέ η σαύρα που έχουν αναλυθεί ξανά σε παραπάνω ενότητα και δεν θα επαναληφθεί η εξήγηση της λειτουργία τους. Έπειτα θα γίνει η δημιουργία της κλάσης AIC_Lizard ως παιδί της AIC_Base που θα είναι η κλάση ελέγχου της σαύρας, ακριβώς όπως έγινε και με τον εχθρό καβούρι και όπως θα γίνει και με τους υπόλοιπους εχθρούς , καθώς όλες οι κλάσεις χαρακτήρων χρειάζονται και μία κλάση ελέγχου. Οπότε στην κλάση AIC_Lizard θα γίνει cast της κλάσης bp_enemy_lizard , ώστε η κλάση ελέγχου να έχει πρόσβαση στης λειτουργίες της πρώτης. Αυτό επιτυγχάνεται με τον παρακάτω τρόπο:

```
// Κλάση MyAIController

ΚΛΑΣΗ AMyAIController ΚΛΗΡΟΝΟΜΕΙ από AAIController

  ΔΗΜΟΣΙΑ:

    // Υπερκαλύπτεται όταν ο AI Controller αποκτά τον έλεγχο ενός Pawn

    ΣΥΝΑΡΤΗΣΗ OnPossess(InPawn) ΥΠΕΡΚΑΛΥΠΤΕΙ


  ΙΔΙΩΤΙΚΑ:

    // Μεταβλητή για τον εχθρικό ηθοποιό

    ΙΔΙΟΤΗΤΑ ControlledEnemy

ΤΕΛΟΣ ΚΛΑΣΗΣ


// Συνάρτηση OnPossess που καλείται όταν ο AI Controller αποκτά τον έλεγχο ενός Pawn

ΣΥΝΑΡΤΗΣΗ OnPossess(InPawn)

  ΚάλεσεSuperOnPossess(InPawn)


  // Προσπάθεια εκχώρησης του ηθοποιού στον τύπο AMyEnemyActor

  ControlledEnemy = ΚάνεCastΣεMyEnemyActor(InPawn)
```

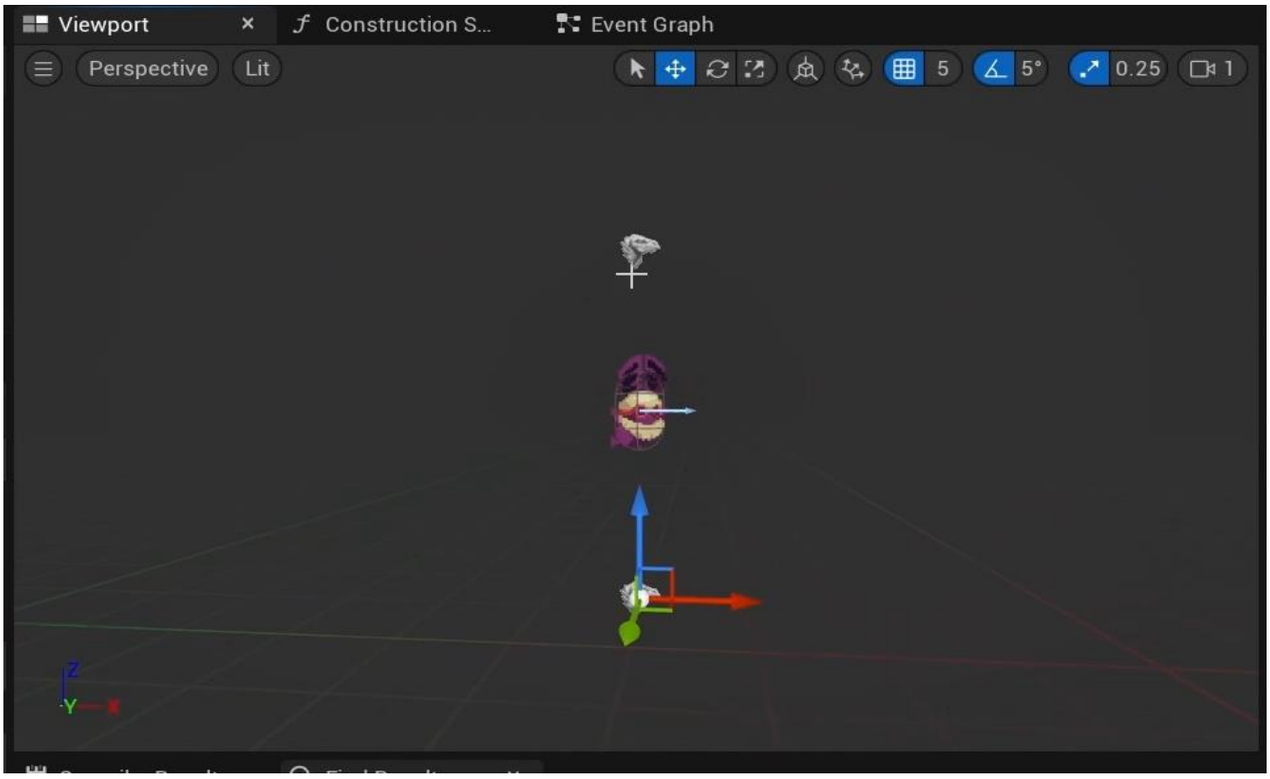
```
    AN ControlledEnemy υπάρχει ΤΟΤΕ
        // Επιτυχής εκχώρηση
        ΚαταγραφήΜηνύματος(Warning, "Successfully possessed: %s", ΠάρεΌνομα(ControlledEnemy))

        // Πρόσβαση στις λειτουργίες του AMyEnemyActor μέσω της ControlledEnemy
        // ControlledEnemy.ΚάλεσεMyEnemyFunction()
    ΑΛΛΙΩΣ
        // Αποτυχία εκχώρησης
        ΚαταγραφήΜηνύματος(Error, "Failed to possess enemy actor.")
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Έτσι αφού η κλάση AIC_Lizard έχει πρόσβαση στις λειτουργίες τις bp_enemy_lizard μπορεί να χρησιμοποιήσει την συνάρτηση shootFireball και γίνεται και δημιουργία ενός επαναλαμβανόμενου χρονομέτρου ώστε η σαύρα να μπορεί να εκτοξεύει βολές ανά συγκεκριμένα χρονικά διαστήματα, δηλαδή η συνάρτηση να καλείται ξανά και ξανά ανά την τιμή της μεταβλητής shootInterval που είναι 3 δευτερόλεπτα. Οπότε ορίζεται στις ρυθμίσεις της bp_enemy_lizard η AIC_Lizard ως κλάση ελέγχου στις ρυθμίσεις και πλέον η σαύρα μπορεί να εκτοξεύει βολές κάθε 3 δεύτερα μέχρι να καταστραφεί. Τώρα όμως πρέπει να γίνει κατάλληλη προσθήκη ενός animation που θα υποδεικνύει ότι η σαύρα όντως πυροβολεί κατά τη στιγμή της βολής. Στη θα δημιουργηθεί ένα animation source με τη χρήση του PaperZD pluggin με όνομα AS_Enemy_Lizard. Οπότε ουσιαστικά σε αυτό το editor θα γίνει προσθήκη των δύο animation της σαύρας και θα γίνει η δημιουργία της κλάσης ABP_enemy_lizard και θα γίνει cast της κλάσης bp_enemy_lizard. Δηλαδή ακριβώς ότι έγινε και για τον βασικό παίχτη του παιχνιδιού για τη διαχείριση των συμπεριφορών του. Οπότε θα δημιουργηθεί ένα state machine με πρώτη κατάσταση την idle που θα παίζει το idle animation και δεύτερη κατάσταση shoot σαν ένα Override slot που θα παίζει το animation του πυροβολισμού. Έτσι όποτε καλείται η συνάρτηση shootFireball θα γίνεται override της κατάστασης idle και θα παίζει το animation shoot. Στη συνέχεια στο τέταρτο καρέ στο Animation source της σαύρας θα προστεθεί ένα notify spit, όπως ονομάζεται το οποίο δίνει σαν σημείο αναφοράς το 4ο καρέ από το animation εκτόξευσης βολής της σαύρας. Οπότε στην κλάση ABP_Enemy_Lizard θα οριστεί αυτό ως σημείο εκκίνησης της συνάρτησης spawnProjectile, οπότε η σφαίρα θα εμφανίζεται ακριβώς σε εκείνο το καρέ. Τέλος θα βρεθούν οι ακριβείς συντεταγμένες του στόματος της σαύρας σε εκείνο το animation και θα αντικαταστήσουν τις παλιές στη συνάρτηση spawnActor ώστε η σφαίρα να εκτοξεύεται ακριβώς από το στόμα της σαύρας. Έτσι τελειοποιήθηκε και οπτικά το animation της βολής της σαύρας.

4.6.3 Δημιουργία ιπτάμενου εχθρού – νυχτερίδα

Όπως και κάθε φορά για την δημιουργία του εχθρού αυτού θα δημιουργηθεί μία νέα κλάση με όνομα bp_enemy_bat ως παιδί της bp_enemy_base και θα γίνει προσθήκη του sprite της νυχτερίδας, καθώς θα προσαρμοστεί και το capsule component. Επειδή αυτός ο εχθρός είναι ιπτάμενος θα πρέπει να γίνει αυτή η μετατροπή στις ρυθμίσεις, καθώς το UE5 αντιλαμβάνεται από μόνο του όλους τους εχθρούς ως όντα κινούνται επί του εδάφους. Όλες οι κλάσεις τύπου Pawn έχουν ενσωματωμένο αυτόματα το character movement component που διαχειρίζεται ότι έχει να κάνει με την κίνηση ενός χαρακτήρα και έτσι πολύ απλά στις ρυθμίσεις αυτού θα γίνει η μετατροπή στο default land movement mode από walking(περπάτημα) σε flying(πτήση). Αυτός ο εχθρός θα μπορεί να πετάει και πηγαίνοντας στον άξονα y από πάνω προς τα κάτω θα εμποδίζει την διάβαση του παίχτη. Επειδή η λειτουργία αυτή είναι πολύ απλή δεν απαιτείτε η δημιουργία μίας AI controller κλάσης όπως έγινε με τους προηγούμενος εχθρούς και η λειτουργία αυτή μπορεί να προστεθεί άμεσα στην κλάση της νυχτερίδας. Οπότε στον editor του capsule component θα προστεθούν αλλά δύο scene components τα οποία απλός είναι δύο σημεία πάνω στον χώρο. Το ένα θα είναι η άνω τοποθεσία και το άλλο η κάτω. Οι συντεταγμένες αυτών των δύο

component θα χρησιμοποιηθούν ως σημείο αναφοράς για την κίνηση της νυχτερίδας από τις συντεταγμένες του ανώτερου σημείο προς το κατώτερο και αντιστρόφως. Έτσι θα επιτευχθεί η λειτουργία της νυχτερίδας να πετάει από πάνω προς τα κάτω. Παρακάτω γίνεται η απεικόνιση της νυχτερίδας και των δύο σημείων στον editor.



Εικόνα 22: Γραφική αναπαράσταση 2 scene components και νυχτερίδας

Οπότε για την επίτευξη της λειτουργίας πτήσης της νυχτερίδας θα δημιουργηθεί ένα timeline το οποίο θα τρέχει συνεχώς και θα ανανεώνει την τοποθεσία της νυχτερίδας με τις κατάλληλες συντεταγμένες αναλόγως του χρόνου που έχει περάσει. Για την επιλογή της τοποθεσίας στην οποία θα μετακινείτε η νυχτερίδα θα δημιουργηθεί ένα lerp vector το οποίο θα παίρνει ως όρισμα Α τις συντεταγμένες της άνω τοποθεσίας και ως Β τις συντεταγμένες της κάτω τοποθεσίας. Το Timeline θα οριστεί για 1 δευτερόλεπτο διάρκεια και θα ξεκινάει από 0 sec με τιμή 0 και θα τελειώνει στο σημείο 1 sec με τιμή 1. Οπότε από εκεί θα προκύπτει η τιμή alpha η οποία θα περάσει και αυτή στον lerp vector, ώστε να ξέρει ο vector πως να εναλλάσσεται μεταξύ των τιμών Α και Β. Αυτή η λογική θα εφαρμοστεί ώστε να τρέχει σε κάθε καρέ. Έτσι η νυχτερίδα πλέον εναλλάσσεται μεταξύ των σημείων Α και Β

ομαλά και αυτό δίνει την αίσθηση πως πετάει. Παρακάτω δίνεται επεξηγηματικός κώδικας.

Βήματα Υλοποίησης

1. Δημιουργία Κλάσης Εχθρικού Ηθοποιού (Enemy Actor Class)

Αρχικά, δημιουργείται η κλάση του εχθρικού ηθοποιού. Αυτός ο ηθοποιός θα περιλαμβάνει τα συστατικά (scene components) για τις θέσεις και τον timeline.

2. Δημιουργία Scene Components για Θέσεις

Θα δημιουργηθούν δύο scene components για την αποθήκευση των κορυφαίων και χαμηλότερων θέσεων.

3. Δημιουργία Timeline και Χρήση του για τη Μετακίνηση

Θα δημιουργηθεί ένα timeline για τον έλεγχο της κίνησης από την κορυφαία θέση στη χαμηλότερη θέση.

#pragma once

// Κλάση FlyingEnemy

ΚΛΑΣΗ AFlyingEnemy ΚΛΗΡΟΝΟΜΕΙ από AActor

```
    ΔΗΜΟΣΙΑ:
        // Κατασκευαστής για προεπιλεγμένες τιμές
        ΣΥΝΑΡΤΗΣΗ AFlyingEnemy()

    ΠΡΟΣΤΑΤΕΥΜΕΝΑ:
        // Καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
        ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

    ΔΗΜΟΣΙΑ:
        // Καλείται κάθε frame
        ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ

    ΙΔΙΩΤΙΚΑ:
        // Συστατικά και μεταβλητές
        ΙΔΙΟΤΗΤΑ Root
        ΙΔΙΟΤΗΤΑ TopPosition
        ΙΔΙΟΤΗΤΑ BottomPosition
        ΙΔΙΟΤΗΤΑ MovementTimeline
        ΙΔΙΟΤΗΤΑ MovementCurve
        ΙΔΙΟΤΗΤΑ StartLocation
        ΙΔΙΟΤΗΤΑ EndLocation

        // Συνάρτηση για την πρόοδο κίνησης
        ΣΥΝΑΡΤΗΣΗ HandleMovementProgress(Value)
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές
ΣΥΝΑΡΤΗΣΗ AFlyingEnemy()
    PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές
    Root = ΔημιουργίαDefaultSubobject("Root")
    RootComponent = Root
    TopPosition = ΔημιουργίαDefaultSubobject("TopPosition")
    TopPosition.ΣύνδεσηΜε(Root)
    BottomPosition = ΔημιουργίαDefaultSubobject("BottomPosition")
    BottomPosition.ΣύνδεσηΜε(Root)
    MovementTimeline = ΔημιουργίαDefaultSubobject("MovementTimeline")
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

```
// Συνάρτηση που καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()

    ΑΝ MovementCurve υπάρχει ΤΟΤΕ
        ProgressFunction.ΣύνδεσηΛειτουργίας(this, "HandleMovementProgress")
        MovementTimeline.ΠροσθήκηInterpFloat(MovementCurve, ProgressFunction)
        StartLocation = TopPosition.ΠάρεΤοποθεσίαΣυστατικού()
        EndLocation = BottomPosition.ΠάρεΤοποθεσίαΣυστατικού()
        MovementTimeline.ΡύθμισηΕπανάληψης(ψευδές)
        MovementTimeline.ΡύθμισηΡυθμούΑναπαραγωγής(1.0)
        MovementTimeline.ΠαίξεΑπόΤηνΑρχή()
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για την πρόοδο κίνησης
ΣΥΝΑΡΤΗΣΗ HandleMovementProgress(Value)
    NewLocation = FMath.Lerp(StartLocation, EndLocation, Value)
    ΟρισμόςΤοποθεσίαςΗθοποιού(NewLocation)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Εξήγηση:

1.Δημιουργία και Αρχικοποίηση Συστατικών:

Δημιουργούνται τα USceneComponent για τις θέσεις TopPosition και BottomPosition.
Δημιουργείται το UTimelineComponent για την κίνηση.

2.Ρύθμιση και Έναρξη του Timeline:

Στο BeginPlay, ελέγχουμε αν το MovementCurve είναι έγκυρο.
Δημιουργείται και συνδέεται η συνάρτηση που θα καλείται κατά τη διάρκεια της κίνησης (ProgressFunction).
Καθορίζονται οι αρχικές και τελικές θέσεις.
Ρυθμίζεται το timeline να μην κάνει loop και να ξεκινά από την αρχή.

3.Ενημέρωση Θέσης Ηθοποιού:

Η συνάρτηση HandleMovementProgress καλείται κατά τη διάρκεια της κίνησης και ενημερώνει τη θέση του ηθοποιού χρησιμοποιώντας την FMath::Lerp για τη γραμμική παρεμβολή μεταξύ των θέσεων. Τέλος γίνεται ρύθμιση της μεταβλητής Play rate σε 0.8, ώστε να οριστεί η ταχύτητα με την οποία θα κινείται η νυχτερίδα. Αυτό θα συμβάλλει στην ευκολότερη αυξομείωση της δυσκολίας αργότερα.

4.6.4 Δημιουργία τελικού εχθρού – ιπτάμενο τέρας

Αυτός θα είναι ο τελικός εχθρός του παιχνιδιού και πιθανός ο πιο δύσκολος, καθώς όχι μόνο θα είναι ένας ιπτάμενος εχθρός αλλά μόλις ο παίχτης εισέλθει στην εμβέλεια εντοπισμού του, ο πρώτος θα τον αντιλαμβάνεται και θα τον κυνηγάει μέχρι ένας από τους δύο να πεθάνει. Οπότε για την δημιουργία αυτού του εχθρού θα δημιουργηθούν οι κλάσεις bp_enemy_eye , καθώς και η κλάση ελέγχου που θα χρειαστεί AIC_Eye. Οπότε όπως συνήθως στην κλάση AIC_Eye θα γίνει cast στην κλάση bp_nemey_eye, ώστε να περάσει ένα reference της κλάσης αυτής στην AIC_Eye και να έχει πρόσβαση σε κάποια στοιχεία της. Ακόμα γίνεται reference και στην κλάση player pawn που είναι η κλάση γονέας του παίχτη, καθώς θα χρειαστεί να υπάρχει πρόσβαση και στην κλάση το παίχτη. Έπειτα θα οριστεί στις ρυθμίσεις η AIC_Eye ως το AI controller class της bp_enemy_eye. Επίσης θα οριστεί το default movement mode πάλι σε πτήση. Στη συνέχεια στην AIC_Eye θα υλοποιηθεί η λογική εντοπισμού του παίχτη. Αρχικά με την συνάρτηση get actor location βρίσκονται οι συντεταγμένες του εχθρού σε κάθε καρέ και με τον ίδιο τρόπο και οι συντεταγμένες του παίχτη. Χρησιμοποιώντας την συνάρτηση find look at rotation με ορίσματα τις συντεταγμένες του εχθρού και του παίχτη υπολογίζεται η περιστροφή που απαιτείται για να κοιτάξει από τη θέση του "Owning Eye" προς τη θέση του "Target Pawn" και με την συνάρτηση get forward vector ο εχθρός παίρνει τον forward vector από την περιστροφή που υπολογίστηκε. Πλέον μπορεί να παρακολουθεί ακριβός τις κινήσεις του παίχτη, δηλαδή που βρίσκεται και προς ποια κατεύθυνση κινείται.

```
// Κλάση FlyingEnemy
ΚΛΑΣΗ AFlyingEnemy ΚΛΗΡΟΝΟΜΕΙ από AActor
  ΔΗΜΟΣΙΑ:
    // Κατασκευαστής για προεπιλεγμένες τιμές
    ΣΥΝΑΡΤΗΣΗ AFlyingEnemy()

  ΠΡΟΣΤΑΤΕΥΜΕΝΑ:
    // Καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
    ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

  ΔΗΜΟΣΙΑ:
    // Καλείται κάθε frame
    ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ

    // Συνάρτηση για να πάρουμε τον forward vector προς τον στόχο
    ΣΥΝΑΡΤΗΣΗ GetForwardVectorTowardsTarget(OwningEye, TargetPawn)
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές
```

```
ΣΥΝΑΡΤΗΣΗ AFlyingEnemy()
    PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)

    // Παράδειγμα χρήσης για να πάρουμε τον forward vector
    OwningEye = ΠάρεΙδιοκτήτη()
    TargetPawn = κενό // Βάλε εδώ τον πραγματικό στόχο

    ΑΝ OwningEye ΚΑΙ TargetPawn ΤΟΤΕ
        ForwardVector = ΠάρεForwardVectorΠροςΣτόχο(OwningEye, TargetPawn)
        // Χρησιμοποίησε τον ForwardVector όπως χρειάζεται
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για να πάρουμε τον forward vector προς τον στόχο
ΣΥΝΑΡΤΗΣΗ GetForwardVectorTowardsTarget(OwningEye, TargetPawn)
    ΑΝ OwningEye == κενό Ή TargetPawn == κενό ΤΟΤΕ
        ΕΠΙΣΤΡΟΦΗ FVector::ZeroVector
    ΤΕΛΟΣ ΑΝ

    OwningEyeLocation = OwningEye.ΠάρεΤοποθεσίαΗθοποιού()
    TargetPawnLocation = TargetPawn.ΠάρεΤοποθεσίαΗθοποιού()
    LookAtRotation = ΒρεςΠεριστροφήΓιαΚοίταγμα(OwningEyeLocation, TargetPawnLocation)
    ForwardVector = ΠάρεForwardVector(LookAtRotation)

    ΕΠΙΣΤΡΟΦΗ ForwardVector
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Μετέπειτα για να πραγματοποιηθεί η κίνηση θα δημιουργηθεί μία λογική παρόμοια με αυτή που έγινε για τον εχθρό καβούρι. Στην κλάση bp_enemy_eye θα γίνει χρήση της συνάρτησης addMovementInput όπου ο εχθρός θα κινείται με τιμή 1, δηλαδή την κανονική του ταχύτητα προς το forward vector του παίχτη, δηλαδή συνεχώς προς τις συντεταγμένες του παίχτη.

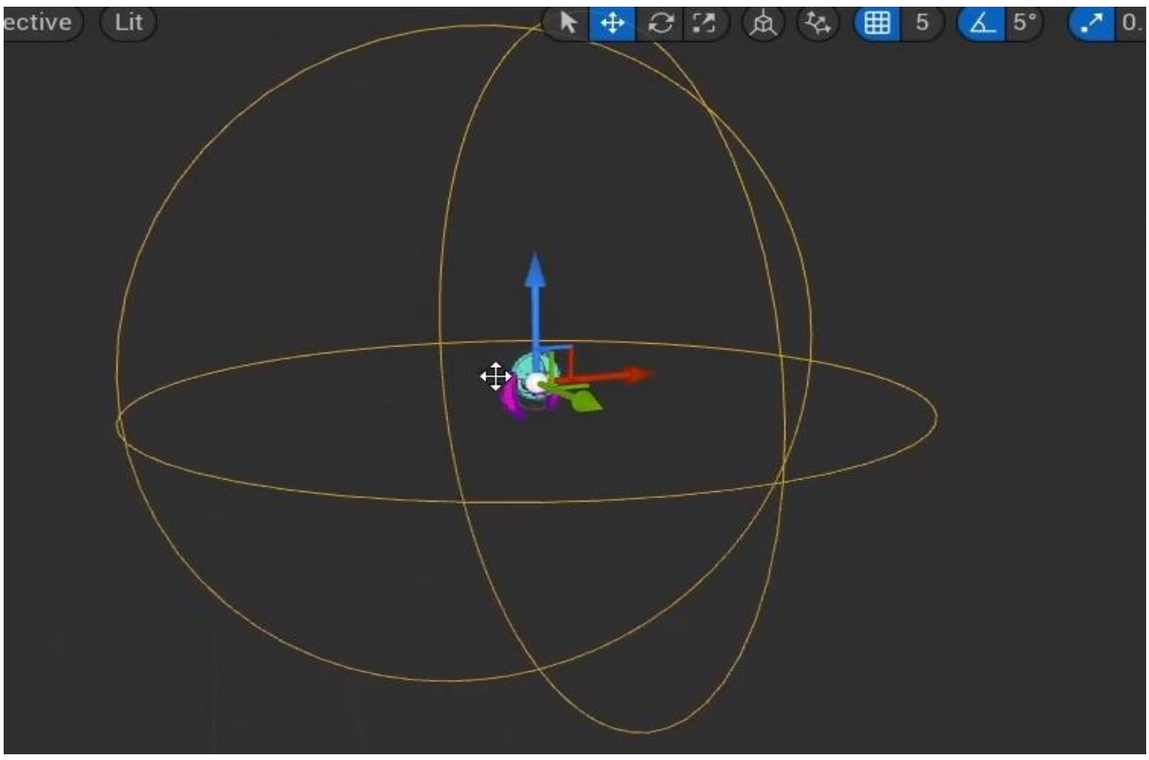
```
// Κλάση FlyingEnemy
ΚΛΑΣΗ AFlyingEnemy ΚΛΗΡΟΝΟΜΕΙ από AActor
  ΔΗΜΟΣΙΑ:
    // Κατασκευαστής για προεπιλεγμένες τιμές
    ΣΥΝΑΡΤΗΣΗ AFlyingEnemy()

  ΠΡΟΣΤΑΤΕΥΜΕΝΑ:
    // Καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
    ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

  ΔΗΜΟΣΙΑ:
    // Καλείται κάθε frame
    ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime) ΥΠΕΡΚΑΛΥΠΤΕΙ

  ΙΔΙΩΤΙΚΑ:
    // Συνάρτηση για να πάρουμε τον forward vector προς τον στόχο
    ΣΥΝΑΡΤΗΣΗ GetForwardVectorTowardsTarget(OwningEye, TargetPawn)

    // Συνάρτηση για να μετακινήσουμε το OwningEye προς τον στόχο
    ΣΥΝΑΡΤΗΣΗ MoveOwningEyeTowardsTarget(OwningEye, TargetPawn)
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Κατασκευαστής που ορίζει προεπιλεγμένες τιμές
ΣΥΝΑΡΤΗΣΗ AFlyingEnemy()
  PrimaryActorTick.ΜπορείΝαΚάνειTick = αληθές
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται όταν ξεκινάει το παιχνίδι ή όταν γεννιέται ο χαρακτήρας
ΣΥΝΑΡΤΗΣΗ BeginPlay()
  ΚάλεσεSuperBeginPlay()
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται κάθε frame
```

```
ΣΥΝΑΡΤΗΣΗ Tick(DeltaTime)
    ΚάλεσεSuperTick(DeltaTime)

    // Παράδειγμα χρήσης για να μετακινήσουμε το Owning Eye προς τον στόχο
    OwningEye = ΚάνεCastΣεCharacter(ΠάρεΙδιοκτήτη())
    TargetPawn = κενό // Βάλε εδώ τον πραγματικό στόχο

    ΑΝ OwningEye ΚΑΙ TargetPawn ΤΟΤΕ
        ΜετακίνησεOwningEyeΠροςΣτόχο(OwningEye, TargetPawn)
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για να πάρουμε τον forward vector προς τον στόχο
ΣΥΝΑΡΤΗΣΗ GetForwardVectorTowardsTarget(OwningEye, TargetPawn)
    ΑΝ OwningEye == κενό Ή TargetPawn == κενό ΤΟΤΕ
        ΕΠΙΣΤΡΟΦΗ FVector::ZeroVector
    ΤΕΛΟΣ ΑΝ

    OwningEyeLocation = OwningEye.ΠάρεΤοποθεσίαΗθοποιού()
    TargetPawnLocation = TargetPawn.ΠάρεΤοποθεσίαΗθοποιού()
    LookAtRotation = ΒρεςΠεριστροφήΓιαΚοίταγμα(OwningEyeLocation, TargetPawnLocation)
    ForwardVector = ΠάρεForwardVector(LookAtRotation)

    ΕΠΙΣΤΡΟΦΗ ForwardVector
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για να μετακινήσουμε το Owning Eye προς τον στόχο
ΣΥΝΑΡΤΗΣΗ MoveOwningEyeTowardsTarget(OwningEye, TargetPawn)
    ForwardVector = ΠάρεForwardVectorΠροςΣτόχο(OwningEye, TargetPawn)
    OwningEye.ΠρόσθεσεΕίσοδοΚίνησης(ForwardVector, 1.0)
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

MoveOwningEyeTowardsTarget: Αυτή η συνάρτηση παίρνει τον χαρακτήρα (OwningEye) και τον στόχο (TargetPawn), υπολογίζει τον forward vector και χρησιμοποιεί τη συνάρτηση AddMovementInput για να προσθέσει κίνηση προς την κατεύθυνση του στόχου με scale value 1.
Tick: Στην μέθοδο Tick, γίνεται ο έλεγχος αν το OwningEye και το TargetPawn είναι έγκυρα και στη συνέχεια καλείται η συνάρτηση MoveOwningEyeTowardsTarget για να εφαρμοστεί η κίνηση.

Ορίζεται η μέγιστη ζωή σε 5 και η μέγιστη ταχύτητα πτήσης σε 400. Για να κοιτάει ο εχθρός προς την κατεύθυνση του παίχτη ακόμα και όταν αυτός αλλάζει κατεύθυνση από το forward vector του παίχτη γίνεται έλεγχος της

συντεταγμένης X εάν είναι αρνητική ή θετική, έτσι αναλόγως της τιμής της γνωρίζει ο εχθρός εάν κινείται αριστερά ή δεξιά και με την συνάρτηση set actor rotation ο εχθρός γυρνάει προς την ίδια κατεύθυνση. Για το επόμενο βήμα θα γίνει η δημιουργία ενός sphere collision component με όνομα aggro range, το οποίο είναι μία σφαίρα η οποία περικλείει τον εχθρό με σχετικά μεγάλο εύρος. Αυτή η σφαίρα θα αποτελέσει την περιοχή στην οποία μπορεί να γίνει ο εντοπισμός του παίχτη, δηλαδή μόλις ο παίχτης κάνει Overlap (έρθει σε επαφή) με αυτή την αόρατη σφαίρα τότε ο εχθρός θα ενεργοποιηθεί και θα αρχίσει μα τον κυνηγάει. Οπότε όπως έχει ήδη γίνει πολλαπλές φορές η σφαίρα θα ρυθμιστεί να μπορεί να αλληλεπιδρά μόνο με αντικείμενα της κλάσης Pawn, δηλαδή τον παίχτη και θα προστεθεί ένα event begin on overlap όπου γίνεται cast στον παίχτη και υλοποιείται η λογική του κυνηγητού του παίχτη.

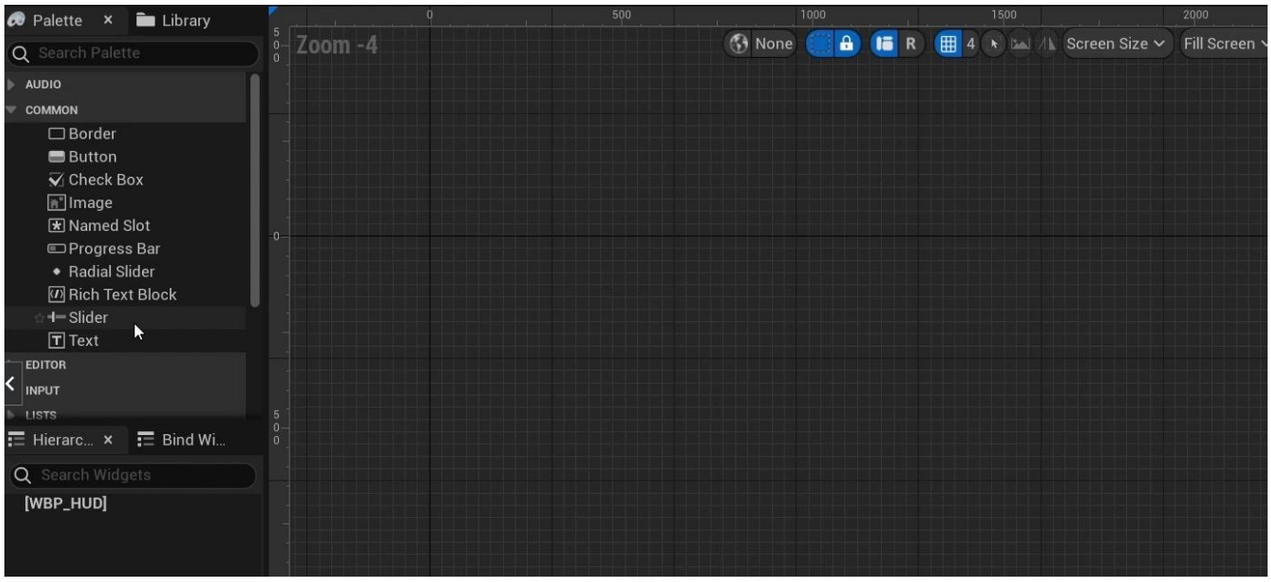


Εικόνα 23: Περιοχή εντοπισμού παίχτη

4.7 Υλοποίηση διεπαφής χρήστη – user interface (UI)

4.7.1 Εισαγωγή στο UI

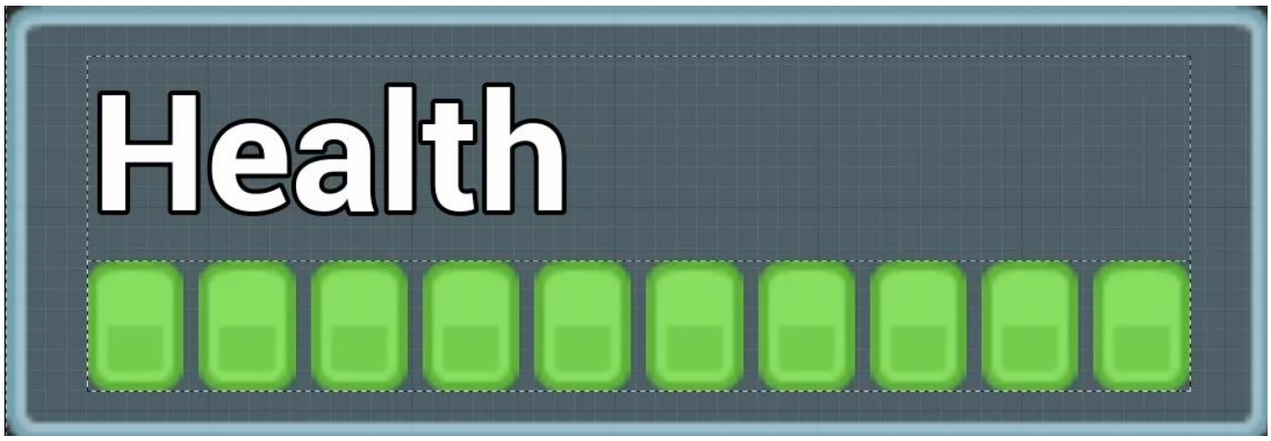
Για την εκπόνηση αυτής της διπλωματικής εργασίας η δημιουργία πολύπλοκων διεπαφής χρηστών ή αλλιώς UI για συντομία δεν ήταν ο κύριος σκοπός και έτσι τα UI είναι πολύ απλά, αλλά πολύ σημαντικά για την ομαλή εμπειρία του παίχτη και την ομαλή ροή του παιχνιδιού. Στο UE5 όλα τα στοιχεία UI ονομάζονται widgets και έτσι δημιουργήθηκε ένας φάκελος με όνομα widgets όπου θα εναποτίθενται όλα τα widget εργασίας. Για την δημιουργία ενός widget πρέπει να δημιουργηθεί ένα widget blueprint. Σε αυτό το blueprint υπάρχει ένας editor όπου στα αριστερά διαθέτει ένα designer για την γραφική δημιουργία των στοιχείων και το graph όπου γίνεται η υλοποίηση του κώδικα που είναι συνδεδεμένος με ένα widget. Εντός του designer editor μπορούν να προστεθούν διάφορα γραφικά στοιχεία όπως πλαίσια, σχήματα, μενού και κείμενο όπως και πολλά άλλα.



Εικόνα 24: Widget editor

4.7.2 Σχεδιασμός του UI της ζωής του παίχτη

Κάθε παιχνίδι το οποίο λειτουργεί με σύστημα ζημιάς και ζωής πρέπει να έχει ένα UI με την μπάρα ζωής του παίχτη, ώστε αυτή να γίνεται αντιληπτή με ευκολία και ευκρίνεια από τον παίχτη. Το πακέτο με τα σχήματα που χρησιμοποιήθηκε για τον σχεδιασμό είναι το UI Pack του Kenney και είναι τύπου CC0, οπότε δεν τίθεται θέμα πνευματικών δικαιωμάτων και χρήσης/ διαμοιρασμού αυτού του πακέτου. Οπότε από αυτό το πακέτο θα χρησιμοποιηθούν τρεις εικόνες. Μία γυάλινου χρώματος για το πλαίσιο της μπάρας ζωής, μία πράσινη για την ζωή και μία εικόνα γκρι που θα συμβολίζει την απώλεια της ζωής. Δημιουργείτε ένας νέος φάκελος με όνομα UI, όπου γίνεται προσθήκη των παραπάνω εικονιδίων και γίνεται αλλαγή κάποιων ρυθμίσεων για την βελτιστοποίηση τους, με ποιο σημαντική την ρύθμιση texture group σε UI, ώστε να αντιλαμβάνεται το UE5 ότι αυτές οι εικόνες πρόκειται να χρησιμοποιηθούν ως γραφικά στοιχεία UI. Στη συνέχεια θα γίνει η δημιουργία του widget blueprint με όνομα WBP_Health_Bar για την ζωή του παίχτη. Αρχικά θα προστεθεί ένα γραφικό στοιχείο border που είναι ένα κουτί. Μέσα σε αυτό θα προστεθεί η εικόνα glass panel με ρύθμιση draw as box ακριβώς πάνω στο border. Επιπλέον θα προστεθεί ένα vertical box μέσα στο border στο οποίο μπορούν να προστεθούν κείμενο και εικόνες. Η εικόνα που θα προστεθεί είναι το πράσινο κουτί για την ζωή και το κείμενο θα λέει Health. Φυσικά θα γίνει αλλαγή των συντεταγμένων και μεγεθών των διαφόρων στοιχείων, ώστε να είναι τα επιθυμητά. Έπειτα θα γίνει αντιγραφή του πράσινου κουτιού άλλες 10 φορές έτσι ώστε κάθε πράσινο κουτί μα απεικονίζει και μία μονάδα ζωής.



Εικόνα 25: UI μπάρας ζωής του παίχτη

Έπειτα στην κλάση GM_Action που είναι η κλάση ελέγχου του παιχνιδιού θα δημιουργηθεί συνάρτηση με όνομα create widget, η οποία δημιουργεί ένα νέο widget βασισμένο σε μια κλάση widget blueprint, συγκεκριμένα την WBP HUD και η συνάρτηση add to viewport που προσθέτει το widget που δημιουργήθηκε στο viewport, δηλαδή το εμφανίζει στην οθόνη του χρήστη.

```
// Κλάση MyHUD

ΣΥΝΑΡΤΗΣΗ BeginPlay()
    ΚάλεσεSuperBeginPlay()


    ΑΝ WBP_HUD_Class υπάρχει ΤΟΤΕ
        PlayerController = ΠάρεΚόσμο().ΠάρεΠρώτοPlayerController()


        ΑΝ PlayerController υπάρχει ΤΟΤΕ
            // Δημιουργία του widget
            CurrentWidget = ΔημιούργησεWidget(PlayerController, WBP_HUD_Class)


            ΑΝ CurrentWidget υπάρχει ΤΟΤΕ
                // Προσθήκη του widget στο viewport
                CurrentWidget.ΠρόσθεσεΣτοViewport()
            ΤΕΛΟΣ ΑΝ
        ΤΕΛΟΣ ΑΝ
    ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

4.7.3 Προσθήκη λειτουργικότητας στο UI

Σε αυτή τη φάση το UI έχει δημιουργηθεί με επιτυχία αλλά δεν έχει ακόμα καμία λειτουργία. Η λειτουργία αυτή θα προστεθεί εντός της κλάσης WBP_Health_Bar και κάθε φορά που ο παίχτης δέχεται ζημιά θα αφαιρείται μία πράσινη εικόνα από την μπάρα ζωής και θα γίνεται αντικατάστασή της από μία ίδια εικόνα γκρι χρώματος. Στο UE5 υπάρχει η δυνατότητα να περαστεί ως μεταβλητή κάποιο γραφικό στοιχείο από το widget designer και έτσι θα περαστεί ως μεταβλητή το κουτί που περικλείει τα πράσινα κουτιά ως μεταβλητή με όνομα health box. Όλα τα στοιχεία που βρίσκονται εντός της μεταβλητής HealthBox θεωρούνται τα "παιδιά" της και έτσι μέσο αυτής έχουμε πρόσβαση στης 10 εικόνες που βρίσκονται εντός του HealthBox που είναι τα πράσινα κουτιά.

```
// Κλάση MyWidget
ΚΛΑΣΗ UMyWidget ΚΛΗΡΟΝΟΜΕΙ από UUserWidget
   ΠΡΟΣΤΑΤΕΥΜΕΝΑ:
      // Καλείται όταν το widget κατασκευάζεται
      ΣΥΝΑΡΤΗΣΗ NativeConstruct() ΥΠΕΡΚΑΛΥΠΤΕΙ

      // Αναφορά στο HealthBox από το UI
      ΙΔΙΟΤΗΤΑ HealthBox

      // Αναφορά στην υφή που θα χρησιμοποιηθεί στο UI
      ΙΔΙΟΤΗΤΑ NewTexture
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Συνάρτηση που καλείται όταν το widget κατασκευάζεται
ΣΥΝΑΡΤΗΣΗ NativeConstruct()
   ΚάλεσεSuperNativeConstruct()

   ΑΝ HealthBox ΚΑΙ NewTexture υπάρχουν ΤΟΤΕ
      // Παίρνουμε όλα τα children του HealthBox
      ChildrenCount = HealthBox.ΠάρεΑριθμόChildren()

      ΓΙΑ i = 0 ΜΕΧΡΙ ChildrenCount ΚΑΝΕ
         // Παίρνουμε το child στο index i
         ChildWidget = HealthBox.ΠάρεChildΣτο(i)

         // Προσπάθεια cast σε UImage
         ImageWidget = ΚάνεCastΣεImage(ChildWidget)

         ΑΝ ImageWidget υπάρχει ΤΟΤΕ
```

```
            // Ορισμός του brush από την υφή
            ImageWidget.ΟρισμόςBrushΑπόΥφή(NewTexture, αληθές)
        ΤΕΛΟΣ ΑΝ
    ΤΕΛΟΣ ΓΙΑ
ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

NativeConstruct: Αυτή η συνάρτηση εκτελείται όταν το widget δημιουργείται.

HealthBox: Το HealthBox είναι ένας πίνακας (panel) widget που περιέχει όλα τα children.

GetChildrenCount: Παίρνει τον αριθμό των children στο HealthBox.

For Loop: Με ένα for loop, διατρέχονται όλα τα children του HealthBox.

Cast<UImage>: Κάθε child γίνεται προσπάθεια να γίνει cast σε τύπο Uimage.

SetBrushFromTexture: Αν το cast είναι επιτυχές, χρησιμοποιείται η SetBrushFromTexture για να οριστεί η νέα υφή στο UImage.

Πριν την χρήση της SetBrushFromTexture όλα τα παιδιά αφού μετατραπούν σε μορφή εικόνας θα προστεθούν σε έναν πίνακα εικόνων με όνομα HealthImages. Οπότε στη συνέχεια εντός της κλάσης BP_Player στο σημείο του κώδικα που γίνεται η λήψη ζημίας του παίχτη ReceiveDamage γίνεται αναφορά στο WBP_Health_Bar μέσω της κλάσης GM_Action και κάθε φορά που ο παίχτης δέχεται ζημία ενημερώνεται το HealthBar και με την SetBrushFromTexture γίνεται σύγκριση της ζωής του παίχτη με τον δείχτη του πίνακα των εικόνων -1 . Οπότε θα γίνει πέρασμα από τον πίνακα ανάλογα με τη ζωή του παίχτη, μέχρι η ζωή να είναι ίση με τον δείχτη του πίνακα πλην 1. Και για όλες τις τιμές που είναι μεγαλύτερες από αυτή τότε ο πίνακας θα αλλάζει τα υπόλοιπα στοιχεία με την συνάρτηση SetBrushFromTexture από πράσινες σε γκρι εικόνες. Για παράδειγμα αν ο παίχτης έχει 3 ζωή, τότε για δείχτη πίνακα μεγαλύτερο του 3 - 1 όλα τα πράσινα κουτάκια θα γίνονται γκρι. Ο δείχτης + 2 είναι η τρίτη θέση το πίνακα, οπότε μόνο 3 κουτιά θα μένουν πράσινα. Και έτσι ολοκληρώνεται η υλοποίηση του UI.

4.8 Προσθήκη εφέ ήχου

4.8.1 Ήχοι παιχνιδιού

Σχεδόν όλες οι λειτουργίες του παιχνιδιού έχουν ολοκληρωθεί, όμως το παιχνίδι παραμένει βουβό, καθώς δεν έχουν ενσωματωθεί ακόμα ήχοι για το περιβάλλον του παιχνιδιού, για τις διάφορες κινήσεις των παιχτών και άλλοι ήχοι που θα προστεθούν στη συνέχεια. Για τους ήχους χρησιμοποιήθηκε ένα έτοιμο πακέτο ήχων με όνομα The ultimate 2017 16bit mini pack από τον phoenix1291 με άδεια χρήσης για κάθε σκοπό. Υπάρχουν τρεις κατηγορίες ήχων, οι ήχοι μίας φοράς που ενεργοποιούνται σε συγκεκριμένες περιστάσεις όπως η αναπήδηση, το γλίστρημα, ο πυροβολισμός και άλλα. Υπάρχουν οι ήχοι που είναι συνδεδεμένοι με κάποιο animation όπως το περπάτημα του παίχτη και των εχθρών και επαναλαμβανόμενοι ήχοι, όπως στη φόρτωση της βολής του παίχτη. Οπότε για αρχή θα περαστούν όλα τα αρχεία ήχου στο UE5 στον φάκελο sounds. Έπειτα στις ρυθμίσεις θα επιλεγεί για όλους τους ήχους η ρύθμιση create multiple cues, ώστε να γίνει αλλαγή στον τύπο αρχείου, καθώς σε ένα sound cue αρχείο υπάρχει η δυνατότητα από το UE5 μέσο ενός editor ήχου να γίνει επεξεργασία αυτών των ήχων. Έτσι με τη βοήθεια της συνάρτησης play sound 2d εισάγονται οι ήχοι μίας φοράς, όπως ο ήχος της αναπήδησης. Ας υποθέσουμε ότι θέλουμε να παίξουμε έναν ήχο που ονομάζεται Jump_Player_SoundCue όταν ο παίκτης πηδάει. Ο κώδικας μπορεί να μοιάζει με τον εξής:

```
// Κλάση MyCharacter
```

```
ΚΛΑΣΗ AMyCharacter ΚΛΗΡΟΝΟΜΕΙ από ACharacter
  ΔΗΜΟΣΙΑ:
    // Κατασκευαστής του χαρακτήρα
    ΣΥΝΑΡΤΗΣΗ AMyCharacter()

  ΠΡΟΣΤΑΤΕΥΜΕΝΑ:
    // Καλείται όταν ξεκινά το παιχνίδι
    ΣΥΝΑΡΤΗΣΗ BeginPlay() ΥΠΕΡΚΑΛΥΠΤΕΙ

    // Ήχος πηδήματος
    ΙΔΙΟΤΗΤΑ JumpSound

    // Συνάρτηση για αναπαραγωγή του ήχου πηδήματος
    ΣΥΝΑΡΤΗΣΗ PlayJumpSound()
ΤΕΛΟΣ ΚΛΑΣΗΣ

// Κατασκευαστής που ορίζει αρχικές τιμές
ΣΥΝΑΡΤΗΣΗ AMyCharacter()
  // Ορισμός αρχικών τιμών
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση που καλείται όταν ξεκινά το παιχνίδι
ΣΥΝΑΡΤΗΣΗ BeginPlay()
  ΚάλεσεSuperBeginPlay()
  // Μπορείτε να καλέσετε τη συνάρτηση PlayJumpSound σε ένα συμβάν όπως το πηδάλισμα
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ

// Συνάρτηση για αναπαραγωγή του ήχου πηδήματος
ΣΥΝΑΡΤΗΣΗ PlayJumpSound()
  ΑΝ JumpSound υπάρχει ΤΟΤΕ
    ΠαίξεΉχο2D(αυτό, JumpSound)
  ΤΕΛΟΣ ΑΝ
ΤΕΛΟΣ ΣΥΝΑΡΤΗΣΗΣ
```

Δημιουργείται μια μεταβλητή τύπου USoundBase* για την αποθήκευση του ήχου που θέλουμε να παίξουμε. Η μεταβλητή αυτή μπορεί να ρυθμιστεί στον Unreal Editor (με το EditAnywhere, BlueprintReadWrite). Αυτή η συνάρτηση χρησιμοποιεί τη UGameplayStatics::PlaySound2D για να παίξει τον ήχο. Παίρνει ως παράμετρο το this, που αναφέρεται στον ηθοποιό (character) που καλεί τη συνάρτηση, και τον ήχο JumpSound που έχει οριστεί. Η συνάρτηση PlayJumpSound μπορεί να κληθεί όποτε χρειάζεται να παιχτεί ο ήχος, για παράδειγμα όταν ο παίκτης πηδάει. Έτσι η συνάρτηση αυτή εφαρμόζεται στην κλάση bp_player μετά την πραγματοποίηση του

κώδικα αναπήδησης. Το ίδιο θα γίνει και την κίνηση αναπήδησης σε τοίχους. Ακριβώς η ίδια διαδικασία θα επαναληφθεί για τις κινήσεις: Προσγείωση του παίχτη, γλίστρημα, ανάβαση σκάλας(θα υλοποιηθεί αργότερα), λήψη ζημιάς παίχτη, θάνατος παίχτη, επαναφορά παίχτη, πυροβολισμός, επιφορτισμένη βολή, πυροβολισμός σαύρας, καταστροφή σφαίρας, θάνατος εχθρών, εντοπισμός παίχτη από ιπτάμενο εχθρό. Έτσι γίνεται η υλοποίηση όλων των ήχων που ενεργοποιούνται μία φορά.

4.8.2 Υλοποίηση ήχων συνδεδεμένων με animation

Για αυτούς τους ήχους πρέπει να υλοποιηθεί μία διαφορετική λογική, καθώς πρέπει να παίζουν συνεχώς για συγκεκριμένα καρέ. Στην κλάση AS_Player που έχει τον editor με όλα τα animation του παίχτη, στο run animation κάθε φορά που το πόδι του παίχτη ακουμπάει το έδαφος θα παίζει ο ήχος του περπατήματος. Οπότε για τα καρέ 1 και 4 που πατάει το έδαφος ο παίχτης θα γίνει προσθήκη ενός sound notify, δηλαδή ενός γεγονότος που θα ενεργοποιείται σε αυτά τα καρέ μόνο και θα παίζει τον επιθυμητό ήχο. Για αυτό το UE5 είναι πάρα πολύ φιλικό προς τους προγραμματιστές και έχει απλουστευμένα editors με τα οποία μπορούν να γίνουν τέτοιες λειτουργίες αυτόματα και δεν χρειάζεται οι προγραμματιστές να σπαζοκεφαλιάζουν με τεχνικά θέματα όπως ο ήχος που δεν είναι και το πεδίο ασχολίας τους. Οπότε τόσο απλά τώρα όταν ο παίχτης περπατάει ακούγεται και ο αντίστοιχος ήχος. Το ίδιο ακριβώς θα γίνει και για το run shoot animation. Για τον εχθρό καβούρι που δεν έχει animation source κλάση, η προσθήκη του ήχου θα πρέπει να γίνει χειροκίνητα. Οπότε θα υλοποιηθεί λογική, όπου ελέγχοντας από το flipbook του καβουριού εφόσον παίζει το animation του περπατήματος και όχι της αδράνειας και έπειτα εφόσον βρίσκεται στα καρέ 2 ή 4 που το πόδι του ακουμπάει στο έδαφος, τότε χρησιμοποιείται η συνάρτηση Playsound η οποία παίζει τον ήχο του περπατήματος.

4.8.3 Υλοποίηση επαναλαμβανόμενων ήχων

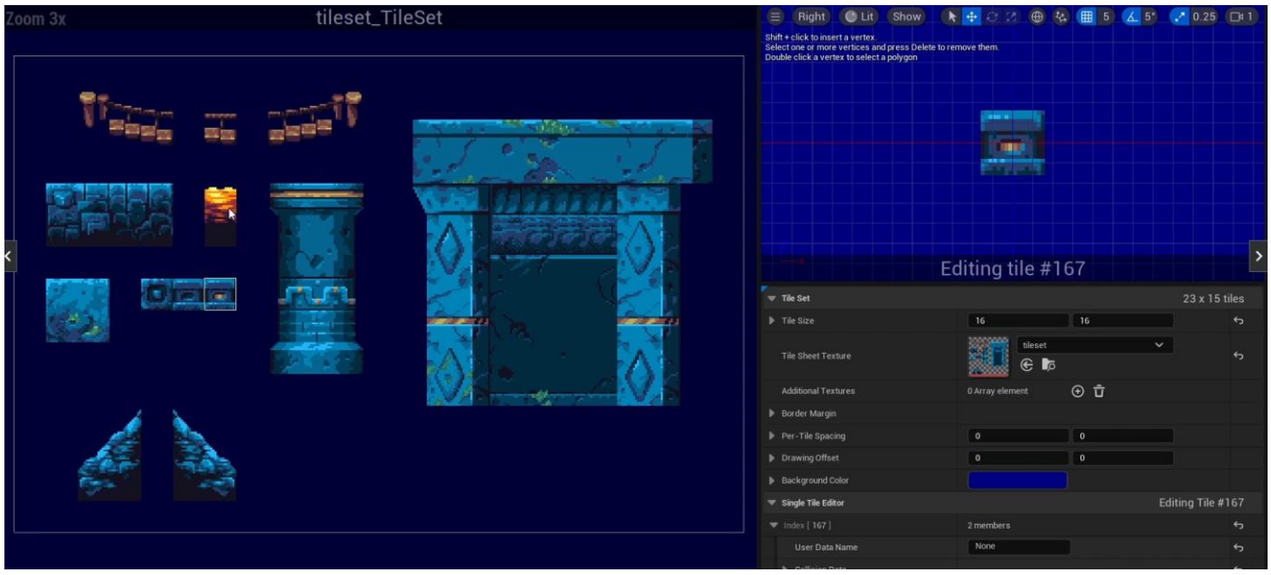
Τέλος για την υλοποίηση του επαναλαμβανόμενου ήχου της επιφορτισμένης βολής του παίχτη κατά την περίοδο της φόρτωσής της θα χρησιμοποιηθεί μία διαφορετική τεχνική. Στην κλάση BP_Player στον editor θα γίνει προσθήκη ενός νέου component ήχου με όνομα chargeSound και σε αυτό προστίθεται ο ήχος της φόρτισης βολής. Οπότε στο σημείο του κώδικα όπου πραγματοποιείται η φορτισμένη βολή και αναβοσβήνει ο παίχτης θα προστεθεί κώδικας που ενεργοποιεί τον ήχο από το chargeSound component. Όμως ο ήχος συνεχίζει να παίζει και μετά το πέρας της περιόδου φόρτωσης. Οπότε στον κώδικα όπου πυροβολεί ο παίχτης στο σημείο που αφήνει το πλήκτρο βολής και ελευθερώνεται η βολή γίνεται απενεργοποίηση του component, οπότε με το που απελευθερώνεται η βολή από την φόρτιση πλέον σταματάει και ο ήχος. Όμως τώρα σε περίπτωση που ο παίχτης δεχθεί ζημιά όσο φορτώνει την βολή και πάθει σάστισμα ο ήχος συνεχίζει να παίζει ασταμάτητα. Οπότε στον κώδικα που δέχεται ζημιά ο παίχτης γίνεται επίσης απενεργοποίηση του component σε περίπτωση που είναι ενεργό και έτσι όταν δέχεται ζημιά ο παίχτης κατά την φόρτιση πάλι σταματάει ο ήχος. Έτσι ολοκληρώθηκε και αυτό το κομμάτι.

4.9 Δημιουργία επιπέδων παιχνιδιού

4.9.1 Δημιουργία tileset και tilemap

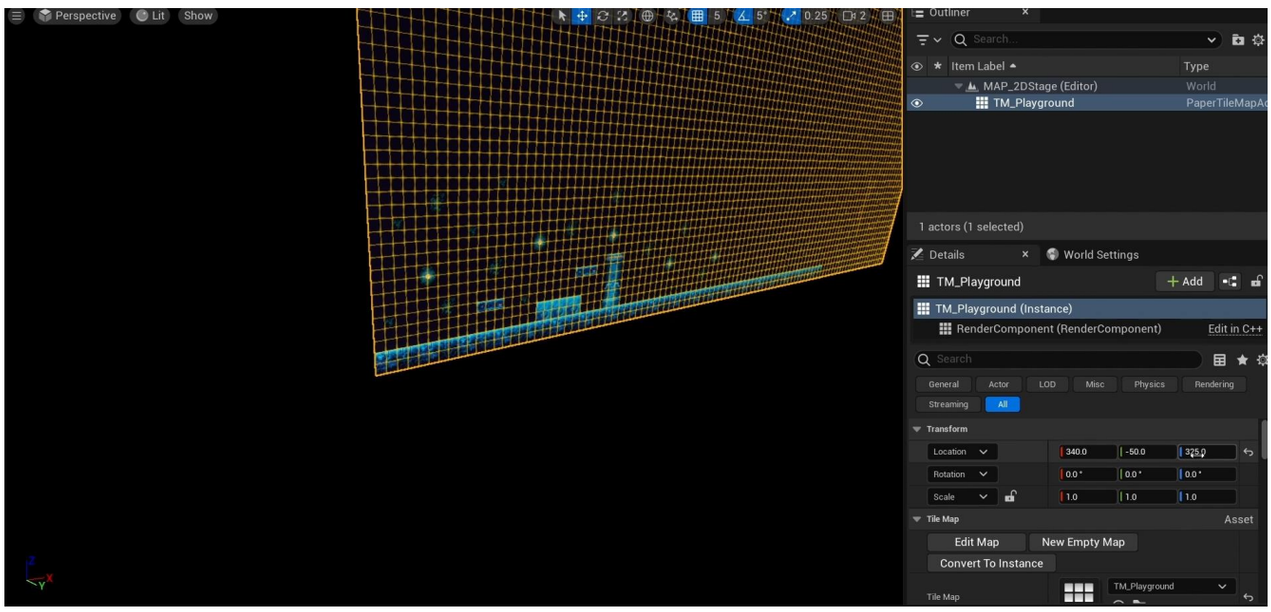
Στο αρχικό στάδιο δημιουργίας του επιπέδου πρέπει πρώτα να γίνει η προετοιμασία των στοιχείων του επιπέδου , δηλαδή των εικόνων που θα αποτελέσουν τα γραφικά στοιχεία του επιπέδου. Αυτά τα στοιχεία όπως αναφέρεται στην αρχή της ενότητας έχουν ενταχθεί ήδη στο UE5 από πριν μαζί με τα animation των χαρακτήρων. Οπότε για όλα τα στοιχεία θα γίνει η μετατροπή τους σε μορφή tilesets μέσο των ρυθμίσεων. Στο UE5 τα γραφικά

στοιχεία του επιπέδου ονομάζονται tiles και ο editor για την διαμόρφωση του επιπέδου ονομάζεται tileset editor. Ο κύριος λόγος του ύπαρξης του tileset editor είναι η ρύθμιση του collision στα στοιχεία αυτά. Τα στοιχεία η αλλιώς assets χωρίζονται σε 2 κατηγορίες. Στα asset υπόβαθρου ή αλλιώς background τα οποία δεν έχουν collision και στα σκηνικά με τα οποία αλληλεπιδρά ο παίχτης και έχουν collision, που ονομάζονται foreground. Οπότε σε όλα τα foreground αντικείμενα θα προστεθούν box collision. Έτσι ο παίχτης μπορεί να αλληλεπιδρά με το περιβάλλον. Τώρα όλα αυτά τα Tileset που δημιουργήθηκαν θα χρησιμοποιηθούν ως στοιχεία για την δημιουργία του επιπέδου στο tilemap editor, όπου στο tilemap γίνεται ο σχεδιασμός του τελικού επιπέδου.



Εικόνα 26: Tileset editor

Στο tilemap editor προστίθενται όλα τα στοιχεία του επιπέδου όπως επιθυμεί ο δημιουργός και έτσι διαμορφώνεται το επίπεδο. Έπειτα μόλις δημιουργηθεί το tilemap αυτό τοποθετείτε στο επίπεδο, δηλαδή στο Map έχει δημιουργηθεί από την αρχή και πλέον είναι ενεργό το επίπεδο του παιχνιδιού. Το επίπεδο στο οποίο θα τοποθετηθεί το tilemap θα είναι ένα κενό επίπεδο, ώστε το tilemap να είναι το μοναδικό στοιχείο που διαθέτει. Το tilemap είναι ένα 2d επίπεδο το οποίο ουσιαστικά τοποθετείτε στον τρισδιάστατο χώρο του UE5.



Εικόνα 27: Προσθήκη tilemap στο επίπεδο

Έπειτα ρυθμίζεται ο φωτισμός των χαρακτήρων στο costumunlitspritematerial, δηλαδή σε φωτισμό χωρίς σκιές αφού πρόκειται για ένα 2d game και γίνεται απενεργοποίηση των σκιών. Έπειτα ρυθμίζεται το auto exposure σε χειροκίνητο και με τιμή 0, ώστε να μειωθεί περαιτέρω η φωτεινότητα του επιπέδου. Στη συνέχεια ρυθμίζεται το

post processing level να είναι το ίδιο για όλο το επίπεδο. Ακόμα τίθεται η ρύθμιση global illumination σε μη υπαρκτή, ώστε το εργαλείο lumen για το φωτισμό του επιπέδου να μην σπαταλάει υπολογιστικούς πόρους.

4.9.2 Δημιουργία ορθογραφικής κάμερας

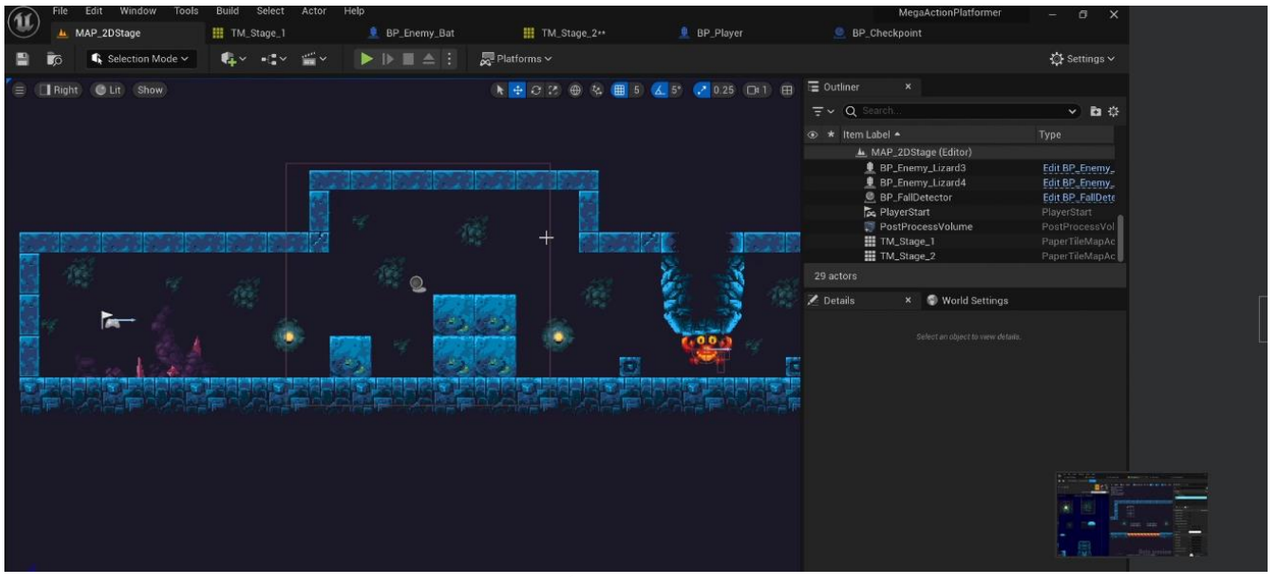
Η Unreal Engine 5 (UE5) υποστηρίζει δύο βασικούς τύπους κάμερας: την ορθογραφική κάμερα και την προοπτική κάμερα. Αυτές οι δύο κάμερες χρησιμοποιούνται για διαφορετικούς σκοπούς και έχουν διαφορετικά χαρακτηριστικά. Η ορθογραφική κάμερα παρουσιάζει το περιβάλλον χωρίς προοπτική. Αυτό σημαίνει ότι τα αντικείμενα δεν μειώνονται σε μέγεθος καθώς απομακρύνονται από την κάμερα. Όλα τα αντικείμενα φαίνονται στο ίδιο μέγεθος, ανεξάρτητα από την απόστασή τους. Αυτή η προσέγγιση είναι χρήσιμη για δισδιάστατες (2D) προβολές, όπως χάρτες ή UI (user interface) στοιχεία, καθώς και σε παιχνίδια ή εφαρμογές όπου η προοπτική δεν είναι απαραίτητη. Η προοπτική κάμερα, από την άλλη πλευρά, δημιουργεί μια εικόνα που είναι πιο κοντά στον τρόπο που βλέπουμε τον κόσμο με τα μάτια μας. Σε αυτή την κάμερα, τα αντικείμενα φαίνονται να μικραίνουν καθώς απομακρύνονται από την κάμερα, δημιουργώντας την αίσθηση του βάθους. Αυτή η κάμερα χρησιμοποιείται κυρίως για τρισδιάστατες (3D) προβολές και είναι ιδανική για παιχνίδια ή εφαρμογές όπου η ρεαλιστική αναπαράσταση του χώρου και του βάθους είναι σημαντική. Οπότε θα γίνει αλλαγή του camera component του παίχτη που έχει ήδη δημιουργηθεί σε ορθογραφική κάμερα.

4.9.3 Δημιουργία δυναμικής 2D κάμερας

Σε κάποια σημεία του επιπέδου όπου το περιβάλλον γίνεται πολύ επικίνδυνο για τον παίχτη, είτε επειδή απαιτεί καλή ορατότητα για δύσκολα άλματα, είτε επειδή εμπεριέχει πολλές παγίδες, είτε επειδή υπάρχουν πολλοί εχθροί για τους οποίους πρέπει να γνωρίζει ο παίχτης θα δημιουργηθεί μία κάμερα η οποία θα τοποθετείτε σε συγκεκριμένα σημεία του επιπέδου και θα εστιάζει σε συγκεκριμένα σημεία που είναι επιθυμητά. Έτσι θα αλλάζει η οπτική γωνία από την κάμερα του παίχτη σε αυτή τη νέα κάμερα. Έτσι θα δημιουργηθεί νέα κλάση τύπου actor με όνομα bp_Camera_Restrictor και θα προστεθεί ένα box collision component με ρύθμιση collision overlap μόνο με τον παίχτη. Οπότε με την αρχή αλλά και το τέλος του overlap γίνεται cast στην κλάση bp_player. Έτσι η κλάση μπορεί να αναγνωρίσει την είσοδο και έξοδο του παίχτη από τα όρια του κουτιού. Οπότε με την είσοδο του παίχτη αλλάζει η μεταβλητή overlapping restrictor που ανήκει στην κλάση camera του bp_player σε αληθής και με την έξοδο του παίχτη σε ψευδής. Μετέπειτα στην κλάση bp_player γίνεται έλεγχος για κάθε καρέ αν η μεταβλητή overlapping restrictor είναι αληθής. Οπότε στην περίπτωση που είναι αληθής θα τεθεί ως νέα θέση της κάμερας του παίχτη η θέση του camera restrictor με το οποίο έκανε Overlap ο παίχτης. Στην περίπτωση που είναι ψευδής τότε τίθεται πάλι η κάμερα να είναι η ίδια με την τοποθεσία του παίχτη. Έτσι όταν ο παίχτης δεν κάνει πλέον overlap με το κάμερα restrictor η κάμερα γυρνάει πίσω στην κανονική της θέση και ακολουθεί τον παίχτη.

4.9.4 Σχεδίαση τριών επιπέδων

Σε αυτό το σημείο τοποθετούνται όλα τα σκηνικά του προσκήνιου και του παρασκηνίου, οι διάφοροι εχθροί, οι τοποθεσίες εκκίνησης του παίχτη, τα κουτιά εντοπισμού πτώσης, τα σημεία επαναφοράς του παίχτη και οι χώροι ενεργοποίησης της δυναμικής κάμερας, ώστε όλα αυτά μαζί να δέσουν και να δημιουργηθούν τα επίπεδα. Το κάθε επίπεδο από σχεδιαστική πλευρά στοχεύει να δυσκολέψει περισσότερο τον παίκτη από ότι το προηγούμενο χρησιμοποιώντας περιβαλλοντικά στοιχεία ώστε να δυσκολέψει το μονοπάτι. Αυξάνονται οι εχθροί, καθώς και οι παγίδες αλλά και οι δυσκολία των αλμάτων. Αυτό αποσκοπεί ώστε ο παίχτης να νιώσει αισθητά την διαφορά στην δυσκολία του κάθε επιπέδου. Ακόμα κάθε πίστα σχεδιάζεται αρκετά διαφορετικά, ώστε ο παίχτης να μην νιώθει συνεχώς παγιδευμένος στο ίδιο σημείο και για αυτό γίνεται εναλλαγή σκηνικών.



Εικόνα 28: Επίδειξη ενός σημείου του επιπέδου

4.9.5 Σχεδιασμός συνθήκης νίκης

Στα επίπεδα ακόμα δεν έχει προστεθεί κάποιος μηχανισμός που σηματοδοτεί την νίκη του παίχτη, έτσι θα γίνει υλοποίηση ενός τέτοιου μηχανισμού σε αυτή την ενότητα. Η νίκη του παίχτη θα επιτυγχάνεται όταν ο παίχτης φτάνει στο τέλος του σχεδιασμένου επιπέδου και συναντάει μία πύλη. Σε αυτή την πύλη θα υπάρχει ένα αόρατο box collision το οποίο θα αλληλεπιδρά μόνο με τον παίχτη θα βλέπει την ζωή του και εάν είναι μεγαλύτερη της τιμής 5 θα τον μεταφέρει στο επόμενο επίπεδο αλλιώς ο παίχτης θα μεταφέρεται στην αρχή του ίδιου επιπέδου. Ακόμα θα εμφανίζεται μία διεπαφή στην οθόνη του παίχτη η οποία θα τον ενημερώνει σε πιο επίπεδο μεταφέρεται. Έτσι γίνεται η δημιουργία μιας κλάσης με όνομα win condition στην οποία προστίθεται ένα box collision component που κάνει Overlap μόνο με τον παίχτη. Έτσι με το begin overlap event θα γίνεται cast στην κλάση του παίχτη και έπειτα θα χρησιμοποιείται η συνάρτηση open level by name ώστε να ανοίξει το κατάλληλο επίπεδο ανάλογα με τον έλεγχο της ζωής του παίχτη. Στη συνέχεια θα δημιουργηθεί μία νέα κλάση widget που είναι τα Interface του UE5 η οποία θα εμφανίζεται με τη συνάρτηση Create widget και add to viewport για μία φορά όταν ο παίχτης εισέρχεται στην ζώνη νίκης και τον ειδοποιεί ότι νίκησε το επίπεδο. Έπειτα με την open level by name θα ανοίγει το επόμενο επίπεδο μετά από μία καθυστέρηση 2 δευτερολέπτων.

```
// Εκτέλεση κώδικα μόνο μία φορά
AN bHasRunOnce == ψευδές ΤΟΤΕ
    bHasRunOnce = αληθές

    // Δημιουργία widget
    Widget = ΔημιούργησεWidget(ΠάρεΚόσμο(), UYourWidgetClass::ΣτατικήΚλάση())

    // Ενσωμάτωση widget στην οθόνη
    AN Widget υπάρχει ΤΟΤΕ
        Widget.ΠρόσθεσεΣτοViewport()
    ΤΕΛΟΣ ΑΝ
```

```
    // Συνάρτηση καθυστέρησης
    TimerHandle = ΔημιούργησεTimerHandle()
    ΠάρεΚόσμο().ΠάρεΔιαχειριστήΧρονομέτρου().ΘέσεTimer(TimerHandle, [=]()
    {
        // Άνοιγμα νέου επιπέδου με βάση το όνομά του
        ΆνοιξεΕπίπεδο(ΠάρεΚόσμο(), FName("LevelName1"))
    }, 5.0, ψευδές) // Καθυστέρηση 5 δευτερολέπτων
ΤΕΛΟΣ ΑΝ
```

4.9.6 Ενσωμάτωση λειτουργίας ανάβασης σκάλας

Για το τελικό στάδιο του σχεδιασμού θα υλοποιηθεί λειτουργία, όπου ο παίχτης όταν συναντάει μία σκάλα στην διαδρομή του θα μπορεί να την ανεβαίνει. Με το που έρχεται σε επαφή ε την σκάλα θα μηδενίζεται η βαρύτητα για την περίοδο της επαφής και ο παίχτης θα μπορεί να πετάει προς τα επάνω παίζοντας το animation ανάβασης της σκάλας δίνοντας έτσι την ψευδαίσθηση ότι ο παίχτης σκαρφαλώνει την σκάλα. Για αυτό τον σκοπό θα δημιουργηθεί μία νέα κλάση τύπου actor bp_laddervolume με ένα box collision component που θα κάνει overlap μόνο με τον παίχτη. Οπότε όπως πάντα γίνεται cast στην Bp_player κατά την αρχή και το τέλος του overlap, ώστε η σκάλα να αλληλεπιδρά μόνο με την κλάση του παίχτη. Η ανάβαση της σκάλας θα γίνεται με την χρήση του w πλήκτρου που βρίσκεται στην bp_player στο IA_move event. Οπότε σε εκείνο το σημείο θα γίνει έλεγχος εάν το IA_move το Y value που είναι η κίνηση του παίχτη στον άξονα y είναι μεγαλύτερο του 0 που σημαίνει ότι το πλήκτρο w είναι πατημένο. Στη συνέχεια γίνεται έλεγχος εάν υπάρχει κάποιο αντικείμενο bp_laddervolume και τότε γίνεται αλλαγή του Movement mode σε flying, δηλαδή ο παίχτης να πετάει και όχι να περπατάει, το οποίο είναι μία ενσωματωμένη λειτουργία στο UE5. Επίσης επειδή στο UE5 υπάρχει και η μεταβλητή της ορμής του χαρακτήρα επειδή δεν είναι επιθυμητό ο παίχτης να ανεβεί την σκάλα με την προηγούμενη ορμή από το περπάτημα μηδενίζεται η μεταβλητή velocity. Όπως είχε γίνει προηγουμένως για την κίνηση του παίχτη στον x άξονα έτσι θα υλοποιηθεί η συνάρτηση add movement input με τιμή στο z 1 και για τον y άξονα έτσι ώστε ο παίχτης να μπορεί να μετακινείτε πάνω και κάτω στην σκάλα. Για να μην μπορεί όμως ο παίχτης να κινηθεί δεξιά και αριστερά όσο ανεβαίνει την σκάλα θα δημιουργηθεί μία μεταβλητή boolean isClimbing που θα γίνεται αληθής όταν ο παίχτης πατάει το w. Και θα οριστεί λογική στον κώδικα με την κίνηση του παίχτη δεξιά και αριστερά πως μπορεί να επιτευχθεί μόνο εάν η μεταβλητή isClimbing είναι ψευδής. Οπότε στη συνέχεια πρέπει να προστεθεί το animation, έτσι στην κλάση ABP_Player θα γίνει ένταξη της συμπεριφοράς climb ως ένα jump node που έχει επεξηγηθεί ήδη και εάν η μεταβλητή isShooting είναι αληθής τότε θα παίζει το climpshoot animation (πυροβολισμού και ανάβασης ταυτόχρονα) αλλιώς θα παίζει το climb animation. Αυτό το animation μετά θα περνάει στο idle state με λογική μετάβασης εάν η μεταβλητή isClimbing είναι ψευδής. Οπότε με την συνάρτηση jump to node όταν ο παίχτης πατάει το w και εκτελεί ανάβαση παίζει και το κατάλληλο animation. Η επόμενος σχεδιασμός που πρέπει να γίνει είναι η δημιουργία λογικής που θα επιτρέπει στον παίχτη να αφήνεται από την σκάλα αν το επιθυμεί. Οπότε αρχικά η μεταβλητή isclimbing θα αλλάζει σε ψευδής εάν είναι αληθής και θα οριστεί το Movement mode του παίχτη από πτήση σε πτώση. Οπότε στην bp_laddervolume όταν η σκάλα σταματάει να κάνει overlap με τον παίχτη καλείτε η παραπάνω λογική και έτσι όταν ο παίχτης φτάνει στην κορυφή της σκάλας πέφτει πάλι κάτω. Έπειτα στην λογική της bp_player στην IA_Jump κίνηση εάν η μεταβλητή Isclimbing είναι αληθής τότε εφαρμόζεται ο κώδικας της λογικής αναπήδησης σε τοίχο και έτσι πλέον ο παίχτης μπορεί να αναπηδήσει από την σκάλα όσο σκαρφαλώνει προς όποια κατεύθυνση επιθυμεί. Έτσι τελειώνει το πρακτικό κομμάτι αυτής της εργασίας και το παιχνίδι είναι πλέον ολοκληρωμένο.

Ενότητα 5η

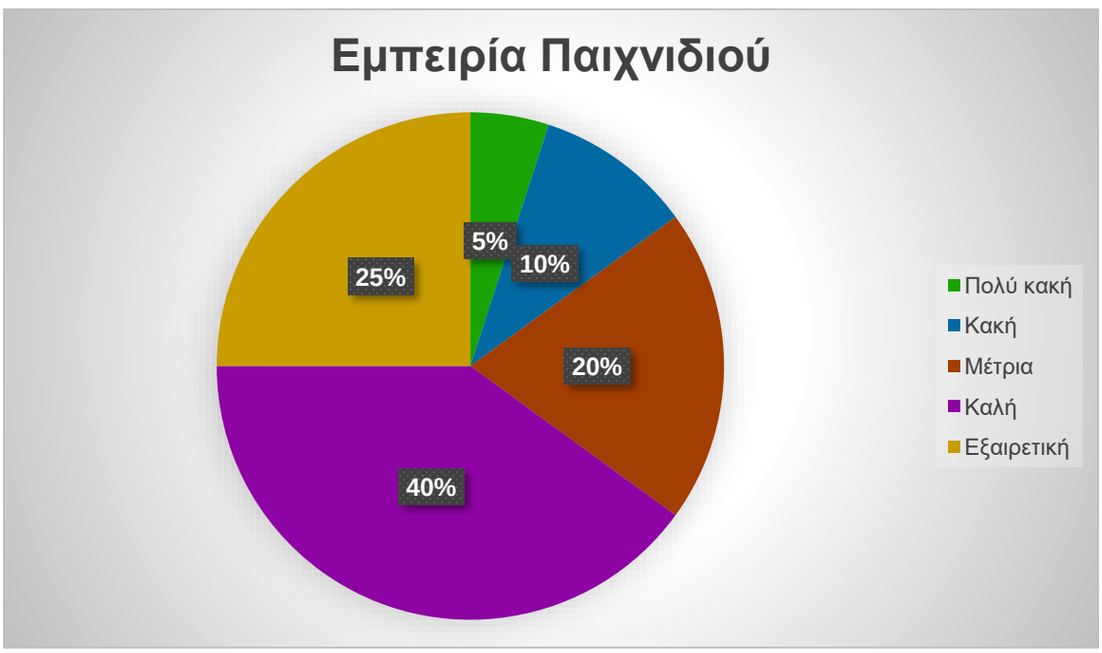
5. Ερωτηματολόγιο και αξιολόγηση παιχνιδιού από τυχαίους χρήστες

5.1 Ποιοι συμμετείχαν στην αξιολόγηση του παιχνιδιού

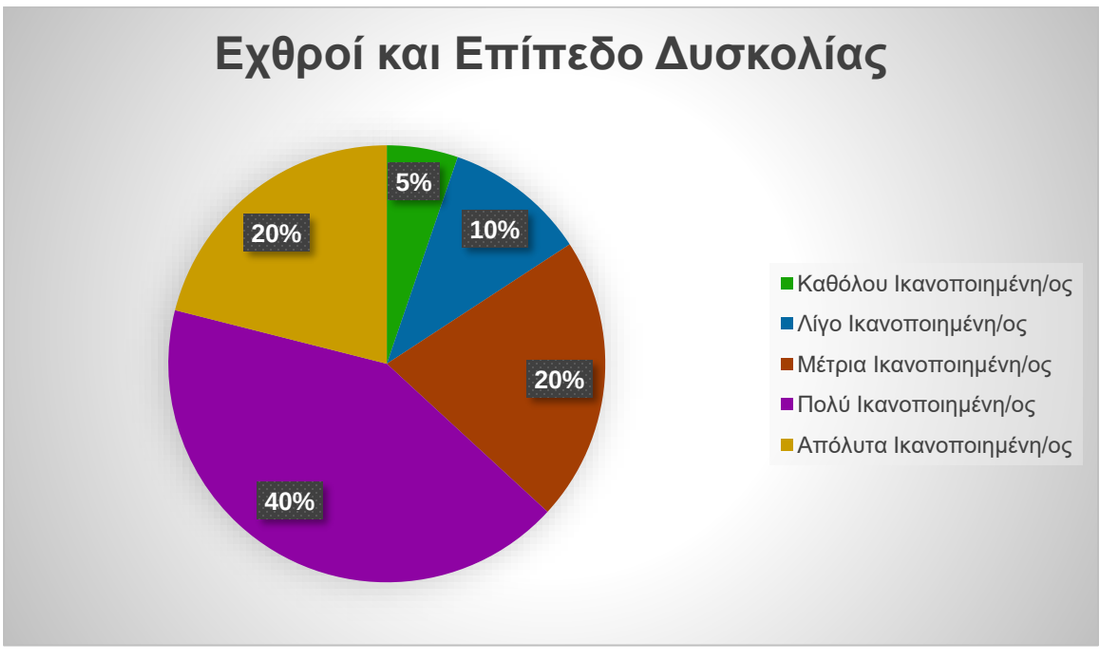
Μετά την ολοκλήρωση του σχεδιαστικού σταδίου του παιχνιδιού έγινε η αξιολόγησή του από τυχαίους παίχτες οι οποίοι αφού έπαιξαν το παιχνίδι κλήθηκαν να απαντήσουν σε ένα ερωτηματολόγιο με 10 ερωτήσεις για να διαπιστωθεί εάν η εμπειρία τους με το παιχνίδι ήταν θετική και για την εύρεση τυχών σφαλμάτων αλλά και σημείων βελτίωσης. Η χρήστες ενημερώθηκαν πως το παιχνίδι πρόκειται για ένα πρότζεκτ αρχάριου επιπέδου στο κομμάτι του σχεδιασμού παιχνιδιού και πως προορίζεται ως διπλωματική εργασία για εκπαιδευτικούς και ερευνητικούς σκοπούς και όχι σαν ένα επαγγελματικό και εμπορικό παιχνίδι που σαφώς θα είχε πολύ περισσότερες λειτουργίες και στοιχεία. Με το παραπάνω κατά νου οι παίχτες – δοκιμαστές κλήθηκαν να απαντήσουν σε 10 ερωτήσεις με βαθμολογία ξεκινώντας από 1 που είναι η χειρότερη βαθμολογία έως 5 που είναι η υψηλότερη. Η επιλογή των συμμετεχόντων έγινε τυχαία από μία γκάμα ανθρώπων ανδρών και γυναικών διαφόρων κλάδων επαγγελματικού προσανατολισμού και ηλικίας μεταξύ 20 και 30 ετών. Οι συμμετέχοντες είχαν διάφορα επίπεδα εκπαίδευσης από φοιτητές μέχρι και κατόχους διπλωμάτων επιπέδου master. Επιπλέον οι εμπειρία των χρηστών πάνω στο κόσμο των βιντεοπαιχνιδιών κάλυπτε ένα ευρύ πλαίσιο από συχνούς και έμπειρους παίκτες με πολλά χρόνια ενασχόλησης με τα βιντεοπαιχνίδια, αλλά και από πιο καθημερινούς χρήστες οι οποίοι παίζουν με πολύ μικρή συχνότητα και δεν είναι τόσο μυημένοι στον κόσμο αυτό. Έτσι επιτεύχθηκε πιο ρεαλιστική προσομοίωση της κοινότητας ή αλλιώς της βάσης παικτών που έχει ένα βιντεοπαιχνίδι και αποκομίστηκαν πολύ χρήσιμες πληροφορίες από διάφορες οπτικές γωνίες. Η διαφορετικότητα μεταξύ των ερωτηθέντων παρείχε πληροφορίες και προτάσεις από διάφορα επίπεδα παικτών, οι οποίοι είχαν διαφορετικές προσδοκίες και στόχους για το παιχνίδι αυτό και έτσι επιτεύχθηκε η βέλτιστη ανατροφοδότηση.

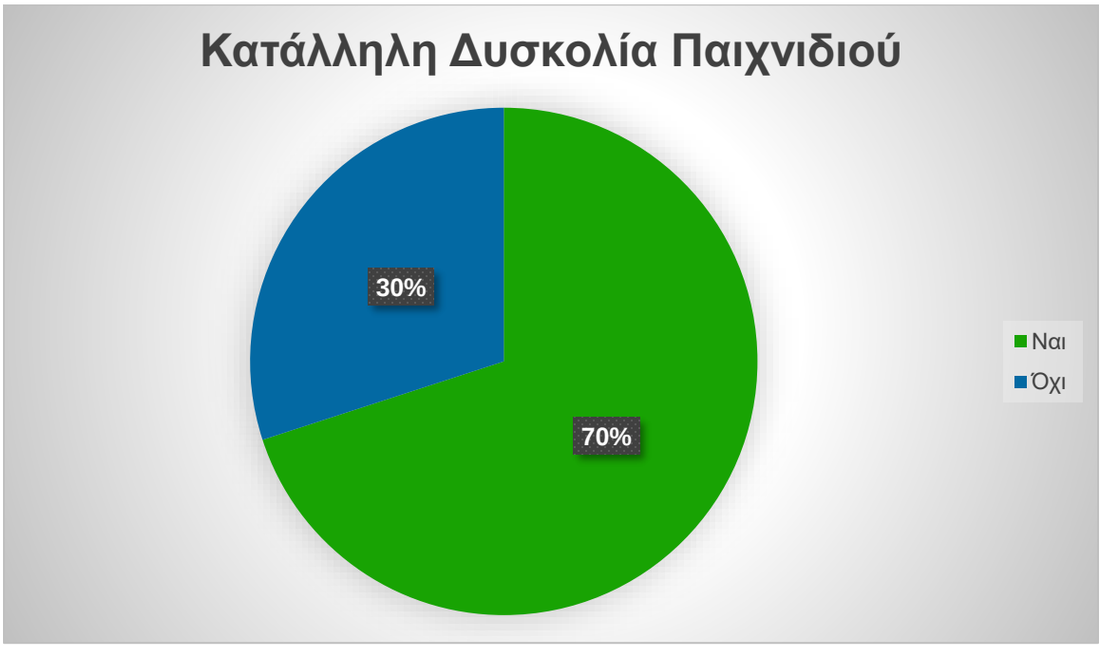
5.2 Αποτελέσματα αξιολόγησης

Παρακάτω παρατίθενται γραφικά με μορφή ποσοστών οι απαντήσεις που δόθηκαν από τους ερωτηθέντες για την κάθε ερώτηση, καθώς και το περιεχόμενο των ερωτήσεων.

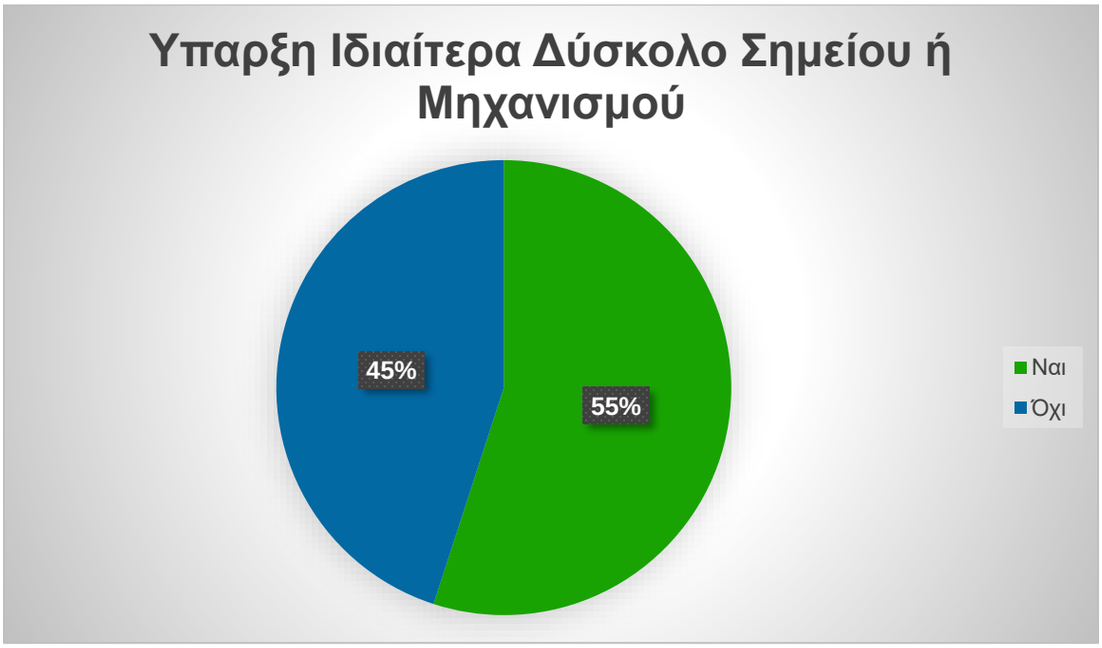


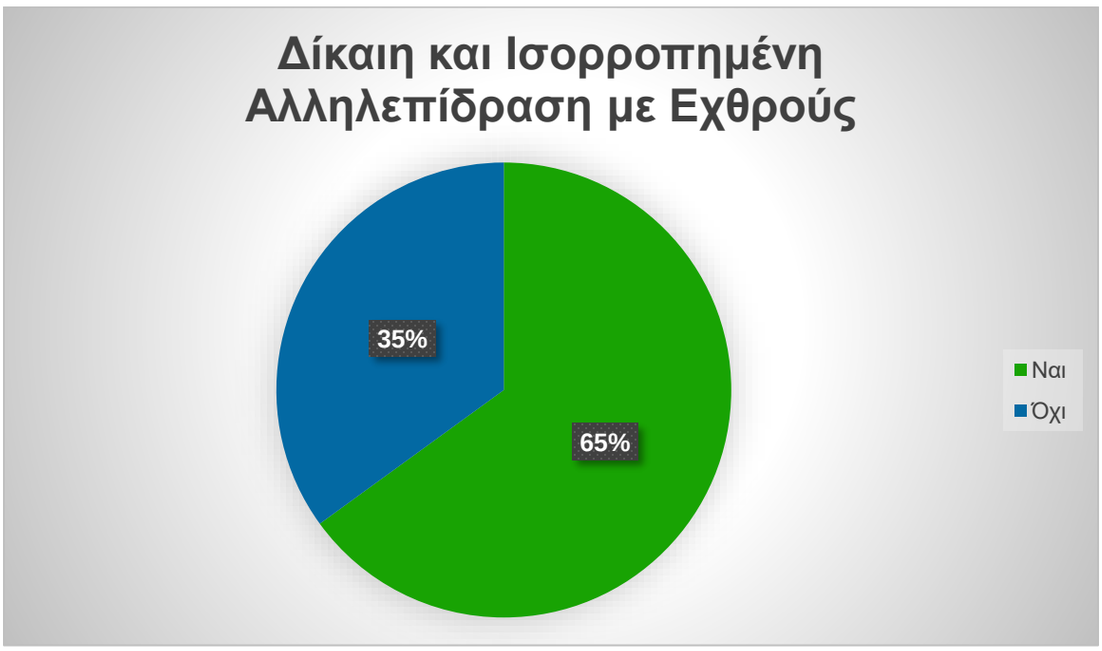

Μηχανισμοί Παιχνιδιού
10%
10%
30%
35%
15%
Πολύ κακή
Κακή
Μέτρια
Καλή
Εξαιρετική

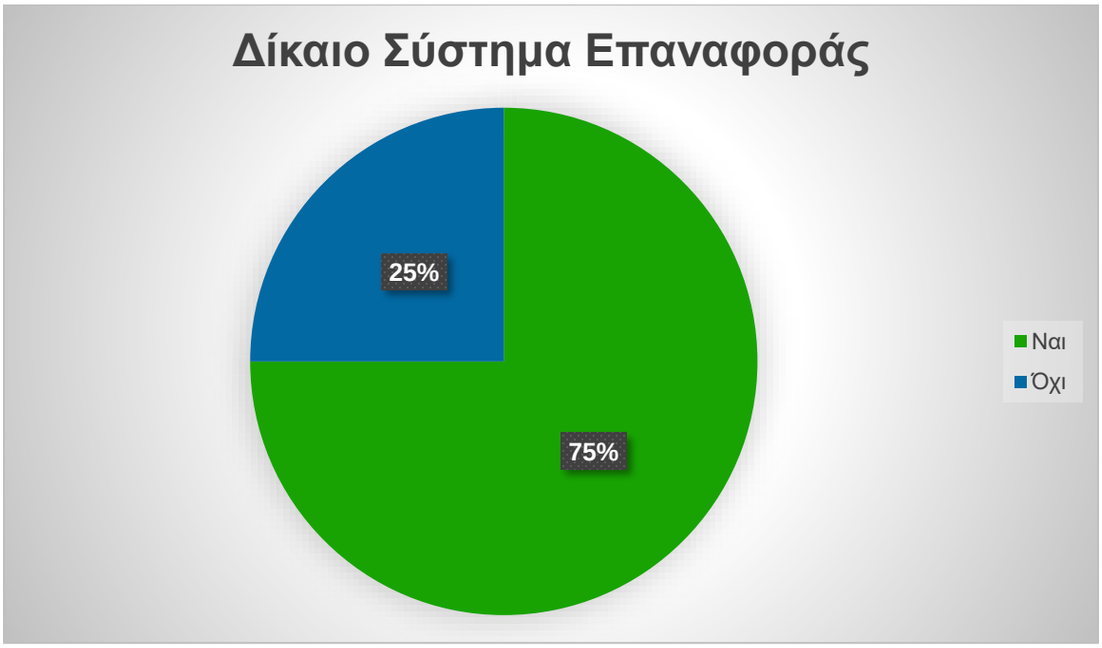

Εχθροί και Επίπεδο Δυσκολίας
5%
10%
20%
40%
20%
Καθόλου Ικανοποιημένη/ος
Λίγο Ικανοποιημένη/ος
Μέτρια Ικανοποιημένη/ος
Πολύ Ικανοποιημένη/ος
Απόλυτα Ικανοποιημένη/ος


Κατάλληλη Δυσκολία Παιχνιδιού
30%
70%
Ναι
Όχι

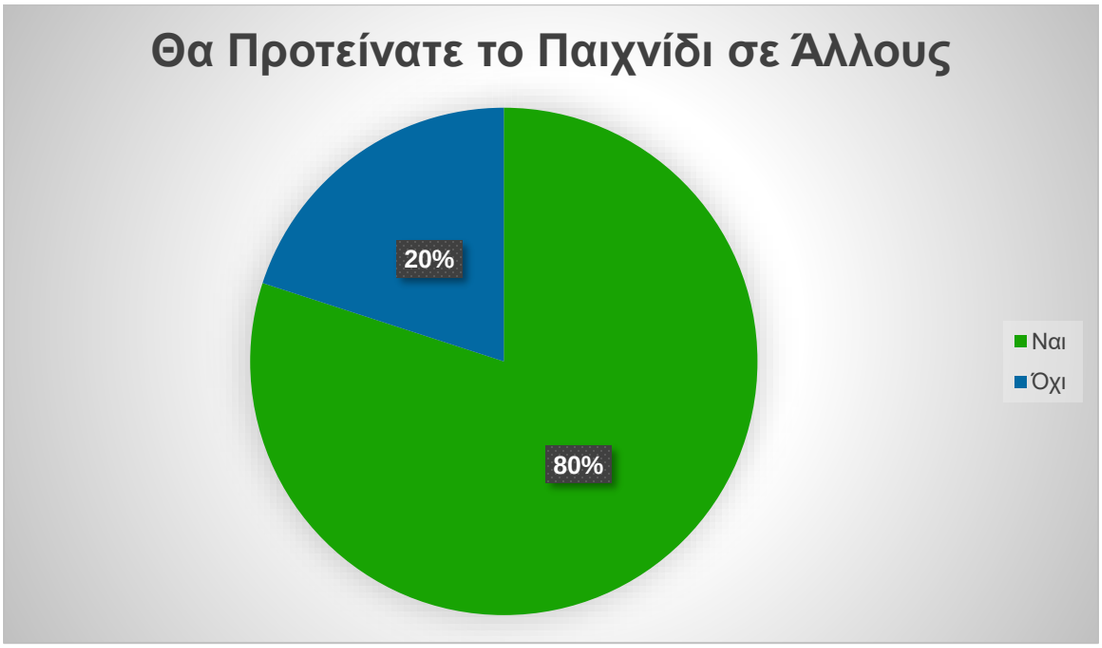

Πίστες Παιχνιδιού
5%
10%
20%
40%
25%
Καθόλου Διασκεδαστικές
Λίγο Διασκεδαστικές
Μέτρια Διασκεδαστικές
Πολύ Διασκεδαστικές
Απόλυτα Διασκεδαστικές


Υπαρξη Ιδιαίτερα Δύσκολο Σημείου ή Μηχανισμού
45%
55%
Ναι
Όχι


Δίκαιη και Ισορροπημένη Αλληλεπίδραση με Εχθρούς
35%
65%
Ναι
Όχι


Δίκαιο Σύστημα Επαναφοράς
25%
75%
Ναι
Όχι


Αξιολόγηση Γραφικών Και Σχεδιασμός Επιπέδων
5%
15%
35%
45%
Πολύ κακά
Κακά
Μέτρια
Καλά
Εξαιρετικά



5.3 Ανάλυση των αποτελεσμάτων

Το ερωτηματολόγιο που απευθύνθηκε σε 20 συμμετέχοντες είχε σκοπό την αξιολόγηση του παιχνιδιού και παρείχε πολύτιμες πληροφορίες σχετικά με την εμπειρία των παικτών, τους μηχανισμούς παιχνιδιού, τη δυσκολία, και την γενική αίσθηση του σχεδιασμού. Τα αποτελέσματα που συλλέχθηκαν αποκαλύπτουν μια γενικά θετική στάση απέναντι στο παιχνίδι, αν και με ορισμένες διαφοροποιήσεις που αντικατοπτρίζουν την ποικιλομορφία των αντιλήψεων και των προσδοκιών των παικτών. Παρακάτω ακολουθεί μια αναλυτική επισκόπηση των αποτελεσμάτων και του τι σημαίνουν για τη συνολική εμπειρία και σχεδιασμό του παιχνιδιού. Το 40% των συμμετεχόντων αξιολόγησαν την εμπειρία τους ως καλή, με επιπλέον 25% να δηλώνουν ότι βρήκαν την εμπειρία εξαιρετική. Το γεγονός ότι η μέση βαθμολογία κυμαίνεται στο 3.8 υποδεικνύει ότι το παιχνίδι παρείχε στους περισσότερους παίκτες μια ευχάριστη εμπειρία, αν και υπήρχαν ορισμένοι που εξέφρασαν επιφυλάξεις (10% κακή, 5% πολύ κακή). Αυτό μπορεί να αποδοθεί σε διάφορους παράγοντες, όπως το επίπεδο πρόκλησης ή συγκεκριμένα προβλήματα που αντιμετώπισαν ορισμένοι παίκτες με τους μηχανισμούς του παιχνιδιού. Οι μηχανισμοί του παιχνιδιού όπως το άλμα, η ολίσθηση, η αναρρίχηση τοίχου και σκάλας βρέθηκαν να είναι μέτριοι έως καλοί, με 35% των παικτών να τους χαρακτηρίζουν "καλούς" και 15% "εξαιρετικούς". Ωστόσο, 20% των συμμετεχόντων βρήκαν τους μηχανισμούς κακούς ή πολύ κακούς. Αυτό υποδηλώνει ότι ενώ το παιχνίδι κατάφερε να ικανοποιήσει την πλειονότητα των παικτών, υπήρξαν ορισμένες αδυναμίες στην υλοποίηση των μηχανισμών που ενδέχεται να δυσκόλεψαν ορισμένους χρήστες. Χρήστες οι οποίοι δεν ήταν εξοικειωμένοι με αυτό το στυλ παιχνιδιού σίγουρα συνάντησαν μεγαλύτερη δυσκολία στην εκμάθησή του από άλλους οι οποίοι είχαν δοκιμάσει παρόμοια παιχνίδια στο παρελθόν. Η ποικιλία των εχθρών και το επίπεδο δυσκολίας βρέθηκαν επίσης σε θετικό επίπεδο, με το 40% των παικτών να δηλώνουν ότι είναι πολύ ικανοποιημένοι και ένα 20% απόλυτα ικανοποιημένοι. Ωστόσο, ένα μικρό ποσοστό (5%) δήλωσε ότι δεν ήταν καθόλου ικανοποιημένο, υποδηλώνοντας ότι υπήρξαν κάποιοι παίκτες που θεωρούσαν ότι η πρόκληση ήταν υπερβολικά χαμηλή ή πολύ περιορισμένη. Αυτό είναι ένα αναμενόμενο εύρημα, καθώς η αίσθηση της δυσκολίας μπορεί να ποικίλλει σημαντικά ανάλογα με τις δεξιότητες και τις προσδοκίες κάθε παίκτη. Το 70% των συμμετεχόντων βρήκαν τη δυσκολία του παιχνιδιού κατάλληλη, ενώ το 30% εξέφρασαν αντίθετη άποψη. Αυτή η διαίρεση είναι τυπική σε παιχνίδια που περιλαμβάνουν διαφορετικά επίπεδα πρόκλησης, καθώς κάθε παίκτης έχει διαφορετική αίσθηση δυσκολίας και ικανότητες. Ενδεχομένως, οι πιο έμπειροι παίκτες να βρήκαν το παιχνίδι εύκολο, ενώ οι λιγότερο εξοικειωμένοι να το βρήκαν υπερβολικά δύσκολο. Το 40% των παικτών θεώρησαν τις πίστες πολύ διασκεδαστικές, ενώ το 25% τις χαρακτήρισε εξαιρετικά διασκεδαστικές. Αυτό είναι ένα θετικό αποτέλεσμα που δείχνει ότι οι πίστες, παρά τις προκλήσεις τους, ήταν καλά σχεδιασμένες και προσέφεραν ευχαρίστηση στους περισσότερους παίκτες. Ωστόσο, η παρουσία ενός μικρού ποσοστού (5%) που βρήκε τις πίστες καθόλου διασκεδαστικές δείχνει ότι μπορεί να υπάρχουν σημεία που χρειάζονται βελτίωση. Περίπου 55% των παικτών ανέφεραν ότι βρήκαν κάποιο σημείο ή

μηχανισμό δύσκολο ή εκνευριστικό, με το υπόλοιπο 45% να μην αντιμετώπισε ιδιαίτερες δυσκολίες. Αυτή η κατανομή είναι αναμενόμενη, καθώς κάθε παίκτης έχει διαφορετικές δεξιότητες και προτιμήσεις ως προς τη διαχείριση μηχανισμών. Συχνά σε παιχνίδια τύπου platformer, η δυσκολία σε συγκεκριμένα σημεία είναι αναπόφευκτη, καθώς αποτελεί μέρος της δομής του είδους. Το 65% των συμμετεχόντων θεώρησε την αλληλεπίδραση με τους εχθρούς δίκαιη και ισορροπημένη, ενώ το 35% βρήκε τα στοιχεία αυτά μη ικανοποιητικά. Αυτά τα αποτελέσματα δείχνουν ότι, γενικά, οι παίκτες ένιωσαν ότι η δυσκολία των εχθρών ήταν καλά ρυθμισμένη, παρόλο που υπήρχαν περιθώρια βελτίωσης για να επιτευχθεί μεγαλύτερη ισορροπία μεταξύ των επιπέδων δυσκολίας των εχθρών. Το 75% των παικτών θεωρεί ότι το σύστημα επανεκκίνησης ήταν δίκαιο και σωστά τοποθετημένο, κάτι που δείχνει ότι οι παίκτες εκτίμησαν την τοποθέτηση σημείων επανεκκίνησης με τέτοιον τρόπο ώστε να μην καθιστούν το παιχνίδι υπερβολικά δύσκολο ή αποθαρρυντικό. Τα γραφικά και ο σχεδιασμός των επιπέδων έλαβαν μια από τις υψηλότερες βαθμολογίες, με το 45% να τα χαρακτηρίζει καλά και το 35% εξαιρετικά. Αυτά τα αποτελέσματα δείχνουν ότι η οπτική απεικόνιση και η δομή των επιπέδων ήταν ένα από τα πιο δυνατά σημεία του παιχνιδιού. Το 80% των παικτών δήλωσαν ότι θα πρότειναν το παιχνίδι σε άλλους, γεγονός που αποτελεί έναν ισχυρό δείκτη επιτυχίας για το παιχνίδι. Το γεγονός ότι το 20% δεν θα το πρότεινε υποδηλώνει ότι υπάρχουν κάποιες αδυναμίες που ενδεχομένως χρειάζονται προσοχή σε μελλοντικές βελτιώσεις. Τα αποτελέσματα του ερωτηματολογίου δείχνουν ότι το παιχνίδι ήταν κατά κύριο λόγο επιτυχές, προσφέροντας μια διασκεδαστική και προκλητική εμπειρία στους παίκτες. Η ομαλή κατανομή των απαντήσεων και η ύπαρξη μιας σχεδόν κανονικής κατανομής των αξιολογήσεων είναι αναμενόμενη για ένα παιχνίδι που περιλαμβάνει ποικίλα επίπεδα πρόκλησης και μηχανισμούς που απαιτούν εξοικείωση. Οι διαφοροποιήσεις μεταξύ των παικτών μπορούν να αποδοθούν στην ποικιλομορφία των εμπειριών τους και των προσδοκιών τους από ένα παιχνίδι τύπου action platformer. Σε γενικές γραμμές, οι παίκτες βρήκαν το παιχνίδι καλά ισορροπημένο και διασκεδαστικό, αν και ορισμένοι μηχανισμοί και επίπεδα δυσκολίας θα μπορούσαν να βελτιωθούν περαιτέρω. Η θετική αποδοχή των γραφικών και του σχεδιασμού επιπέδων αποτελεί μια ισχυρή βάση για μελλοντικές αναβαθμίσεις και επιβεβαιώνει ότι η κατεύθυνση ήταν κατάλληλη για την επίτευξη των στόχων της εργασίας.

Ενότητα 6η

6. Συμπεράσματα

Συμπέρασμα: Σχεδιασμός και Ανάπτυξη Παιχνιδιών στην UE5 για ένα 2D Action Platformer

Η ενασχόληση με την ανάπτυξη παιχνιδιών, ειδικά στην Unreal Engine 5 (UE5) χρησιμοποιώντας C++, προσέφερε μια πλούσια και σημαντική εμπειρία. Αυτό το έργο, που επικεντρώθηκε στη δημιουργία ενός 2D action platformer εμπνευσμένου από κλασικά παιχνίδια, παρείχε μια βαθιά κατανόηση των λεπτομερειών του σχεδιασμού και της ανάπτυξης παιχνιδιών. Η διαδικασία περιλάμβανε την ενσωμάτωση βασικών μηχανισμών παιχνιδιού όπως ο πυροβολισμός, το άλμα, η ολίσθηση, η αναρρίχηση και οι αλληλεπιδράσεις με τους εχθρούς, ενώ ταυτόχρονα έγινε προσπάθεια ισορρόπησης της δυσκολίας σε πολλά επίπεδα. Μέσα από αυτή τη διπλωματική εργασία, τις προκλήσεις που αντιμετώπισα και τις γνώσεις που απέκτησα από αυτό το έργο, δεν ήταν μόνο απαραίτητες για την δημιουργία του 2d platformer παιχνιδιού, αλλά συνέβαλαν επίσης σε μια ευρύτερη κατανόηση των αρχών σχεδιασμού παιχνιδιών και των μοναδικών δυνατοτήτων της UE5 και της C++. Ο σχεδιασμός παιχνιδιών είναι μια πολύπλευρη επιστήμη που περιλαμβάνει διάφορα στοιχεία όπως οι μηχανισμοί παιχνιδιού, η αφήγηση, η αισθητική και η αλληλεπίδραση του παίκτη. Για την ανάπτυξη ενός επιτυχημένου παιχνιδιού, απαιτείται μια ισορροπημένη προσέγγιση που να λαμβάνει υπόψη τόσο την τεχνική όσο και την καλλιτεχνική πλευρά του σχεδιασμού.

Στην περίπτωση ενός 2D action platformer, ο σχεδιασμός επικεντρώνεται κυρίως στους μηχανισμούς παιχνιδιού και στην αίσθηση της πρόκλησης που παρέχει το παιχνίδι. Ένας από τους κύριους στόχους του σχεδιασμού αυτού του παιχνιδιού ήταν να επιτευχθεί η σωστή ισορροπία ανάμεσα στη δυσκολία και τη διασκέδαση. Οι μηχανισμοί όπως το άλμα, η ολίσθηση και η αναρρίχηση τοίχου έπρεπε να είναι άμεσοι και ικανοποιητικοί, χωρίς όμως να καθιστούν το παιχνίδι υπερβολικά δύσκολο ή εύκολο. Η εμπειρία μου έδειξε ότι η απόδοση αυτών των μηχανισμών σε ένα 2D περιβάλλον απαιτεί προσεκτική ρύθμιση των παραμέτρων για να επιτευχθεί η ιδανική αίσθηση για τον παίκτη. Παράλληλα, η αλληλεπίδραση με τους εχθρούς ήταν κρίσιμη για την απόδοση του επιπέδου πρόκλησης που επιθυμούσα να πετύχω. Οι τέσσερις τύποι εχθρών που σχεδιάστηκαν για αυτό το παιχνίδι — οι πεζοί εχθροί, οι εχθροί που πυροβολούν, οι ιπτάμενοι που επιτίθενται στον παίκτη και οι ιπτάμενοι που απλώς κινούνται κατακόρυφα — προσέφεραν διαφορετικά επίπεδα πρόκλησης και ανάγκασαν τον παίκτη να προσαρμόσει τη στρατηγική του ανάλογα με την κατάσταση. Αυτό το στοιχείο προσέφερε βάθος στο παιχνίδι, διατηρώντας το ενδιαφέρον του παίκτη σε κάθε επίπεδο. Η χρήση της Unreal Engine 5 για την ανάπτυξη ενός 2D action platformer παρουσίασε μια σειρά από προκλήσεις, αλλά και σημαντικές ευκαιρίες για μάθηση. Η UE5 είναι γνωστή για τις εξαιρετικές δυνατότητες που παρέχει σε 3D περιβάλλοντα, αλλά η χρήση της για 2D ανάπτυξη απαιτεί μια πιο προσαρμοσμένη προσέγγιση. Μια από τις κύριες προκλήσεις ήταν η αποτελεσματική χρήση της φυσικής και των μηχανισμών κίνησης σε ένα 2D πλαίσιο. Η UE5 προσφέρει έναν ισχυρό φυσικό κινητήρα που όμως είναι βελτιστοποιημένος για 3D παιχνίδια. Για την επίτευξη του επιθυμητού βαθμού ελέγχου και ακριβείας στους μηχανισμούς κίνησης, χρειάστηκε να προσαρμοστούν πολλές από τις προεπιλογές του κινητήρα. Αυτό περιλάμβανε την προσαρμογή της συμπεριφοράς του χαρακτήρα κατά το άλμα, τη διαχείριση των συγκρούσεων με τα αντικείμενα και τους εχθρούς, καθώς και την αλληλεπίδραση με τις επιφάνειες στις οποίες ο παίκτης μπορεί να αναρριχηθεί.

Ένα άλλο σημαντικό μάθημα ήταν η διαχείριση της απόδοσης του παιχνιδιού. Η UE5 είναι ένας εξαιρετικά ισχυρός κινητήρας, αλλά η διατήρηση της απόδοσης σε υψηλά επίπεδα σε ένα 2D παιχνίδι απαιτεί προσεκτική διαχείριση των πόρων. Η βελτιστοποίηση των γραφικών και η σωστή διαχείριση των assets ήταν κρίσιμες για να διασφαλιστεί ότι το παιχνίδι τρέχει ομαλά, ακόμα και όταν ο αριθμός των αντικειμένων στην οθόνη είναι υψηλός. Η ανάπτυξη με την C++ στην UE5 ήταν μια εμπειρία που προσέφερε βαθιά κατανόηση της λειτουργίας του κινητήρα και των δυνατοτήτων που παρέχει. Η C++ είναι η κύρια γλώσσα προγραμματισμού της UE5 και προσφέρει εξαιρετική ευελιξία και έλεγχο στον προγραμματιστή, επιτρέποντας τη δημιουργία προσαρμοσμένων μηχανισμών και λογικής παιχνιδιού. Η χρήση της C++ για την ανάπτυξη των μηχανισμών του παιχνιδιού μου επέτρεψε να έχω πλήρη έλεγχο σε κάθε πτυχή του παιχνιδιού, από τις κινήσεις του χαρακτήρα μέχρι τις αλληλεπιδράσεις με τους εχθρούς και τα αντικείμενα του περιβάλλοντος. Ωστόσο, αυτό απαιτούσε καλή κατανόηση της δομής της UE5 και των αρχών προγραμματισμού σε C++, καθώς και την ικανότητα να διαχειρίζομαι την πολυπλοκότητα του κώδικα καθώς το έργο εξελισσόταν. Ένα από τα βασικά πλεονεκτήματα

της χρήσης της C++ ήταν η δυνατότητα βελτιστοποίησης της απόδοσης του παιχνιδιού. Με την C++, ήταν δυνατή η γραφή κώδικα που εκτελείται πιο γρήγορα και είναι πιο αποδοτικός από ό,τι θα μπορούσε να γίνει με άλλα εργαλεία προγραμματισμού που προσφέρει η UE5, όπως το Blueprint. Αυτό ήταν ιδιαίτερα σημαντικό για τη διατήρηση του ρυθμού ανανέωσης και της ομαλής λειτουργίας του παιχνιδιού, ακόμα και σε πιο απαιτητικές καταστάσεις, όπως η διαχείριση πολλών εχθρών ταυτόχρονα. Συμπερασματικά η ανάπτυξη ενός 2D action platformer στην Unreal Engine 5 χρησιμοποιώντας C++ ήταν μια πρόκληση που προσέφερε σημαντικά διδάγματα για τον σχεδιασμό και την ανάπτυξη παιχνιδιών. Ο σχεδιασμός ενός τέτοιου παιχνιδιού απαιτεί προσεκτική ισορροπία ανάμεσα στην πρόκληση και τη διασκέδαση, ενώ η υλοποίηση των μηχανισμών απαιτεί λεπτομερή γνώση του εργαλείου ανάπτυξης και των προγραμματιστικών δεξιοτήτων. Η εμπειρία αυτή με δίδαξε τη σημασία της λεπτομέρειας στο σχεδιασμό των μηχανισμών, την αναγκαιότητα για βελτιστοποίηση της απόδοσης και την αξία της προσαρμογής στις δυνατότητες του κινητήρα ανάπτυξης.

Η χρήση της C++ στην UE5 προσέφερε τον απαραίτητο έλεγχο και την ευελιξία για την υλοποίηση ενός παιχνιδιού που δεν θα ήταν δυνατό με άλλες μεθόδους. Αυτό το έργο, αν και μικρό σε κλίμακα, αποτελεί ένα αντιπροσωπευτικό παράδειγμα των δεξιοτήτων και της γνώσης που απαιτούνται για την ανάπτυξη παιχνιδιών και θέτει τις βάσεις για μελλοντικές, πιο πολύπλοκες προκλήσεις στον τομέα της ανάπτυξης παιχνιδιών. Η Unreal Engine 5 και η C++ αποδεικνύονται ισχυρά εργαλεία για την υλοποίηση οποιουδήποτε σχεδίου, και η κατανόηση και η εμπειρία που αποκόμισα από αυτό το έργο θα είναι ανεκτίμητες στην μελλοντική μου πορεία στον τομέα της ανάπτυξης παιχνιδιών.

Βιβλιογραφία

1. "A Brief History of Video Games" by Richard Stanton
2. "Artificial Intelligence and Games" by Georgios N. Yannakakis and Julian Togelius
3. "Game Programming Patterns" by Robert Nystrom
4. "C++ Programming for Unreal Game Development" by Richard M. Reese
5. "Programming 2D Games" by Charles Kelly
6. "C++ for Game Programmers" by Noel Llopis
7. [Mil1] Gregory, Kate, and Ade Miller. C++ AMP: Accelerated Massive Parallelism with Microsoft® Visual C++®. " O'Reilly Media, Inc.", 2012
8. [Gas2] B. R. Gaster and L. Howes, "Can GPGPU Programming Be Liberated from the Data-Parallel
9. Bottleneck," Computer, vol. 45, no. 8, pp. 42–52, Aug. 2012
10. Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Game development software engineering process life cycle: A systematic review. Journal of Software Engineering Research and Development, 4(6)
11. Mukherjee, Druhin. "Game Development Using C++." Packt Hub, 2014.
12. "Cross-Platform Development Secrets In C++," Code With C.
13. FSM:Millington, I., & Funge, J. (2009). Artificial Intelligence for Games. Morgan Kaufmann.
14. Buckland, M. (2005). Programming Game AI by Example. Jones & Bartlett Learning.
15. Pathfinding:LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press. Dechter, R., & Pearl, J. (1985).
16. Generalized Best-First Search Strategies and the Optimality of A.* Journal of the ACM, 32(3), 505-536.
17. Behavior trees: Champandard, A. J. (2007). AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders. Isla, D. (2005). Handling Complexity in the Halo 2 AI. Proceedings of the Game Developers Conference.
18. Obstacle Avoidance:Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. ACM SIGGRAPH Computer Graphics. Van Den Bergen, G. (2004). Collision Detection in Interactive 3D Environments. Morgan Kaufmann.
19. Enemy Spawning and Difficulty Scaling:Adams, E., & Rollings, A. (2006). Fundamentals of Game Design. Prentice Hall. Hunicke, R. (2005). The Case for Dynamic Difficulty Adjustment in Games. Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology.
20. Moments Log. (2023). "State Pattern in Video Game AI: Finite State Machines." Retrieved from Moments Log.
21. LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.
22. Champandard, A. J. (2007). AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders.
23. Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. ACM SIGGRAPH Computer Graphics.
24. Adams, E., & Rollings, A. (2006). Fundamentals of Game Design. Prentice Hall.

25. Troussas, C., Virvou, M., Espinosa, K.J.: Using visualization algorithms for discovering patterns in groups of users for tutoring multiple languages through social networking. J. Networks 10, 668–674 (2015).

26. Troussas, C., Chrysafiadi, K., Virvou, M. (2018). Machine Learning and Fuzzy Logic Techniques for Personalized Tutoring of Foreign Languages. In: Penstein Rosé, C., et al. Artificial Intelligence in Education. AIED 2018. Lecture Notes in Computer Science(), vol 10948. Springer, Cham. https://doi.org/10.1007/978-3-319-93846-2_67

27. C. Troussas, A. Krouska and M. Virvou, "Integrating an Adjusted Conversational Agent into a Mobile-Assisted Language Learning Application," 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), Boston, MA, USA, 2017, pp. 1153-1157, doi: 10.1109/ICTAI.2017.00176.

28. Krouska, A., Troussas, C., Sgouropoulou, C. (2020). Applying Genetic Algorithms for Recommending Adequate Competitors in Mobile Game-Based Learning Environments. In: Kumar, V., Troussas, C. (eds) Intelligent Tutoring Systems. ITS 2020. Lecture Notes in Computer Science(), vol 12149. Springer, Cham. https://doi.org/10.1007/978-3-030-49663-0_23

Παράρτημα

Ερωτηματολόγιο Αξιολόγησης Παιχνιδιού (1: Πολύ κακή, 5: Εξαιρετική)

1.Πώς αξιολογείτε τη γενική εμπειρία παιχνιδιού σας; (1: Πολύ κακοί, 5: Εξαιρετικοί)

- Μέσος Όρος: 3.8
- Οι παίκτες γενικά βρήκαν το παιχνίδι καλό, με ορισμένους να το θεωρούν εξαιρετικό, αλλά υπήρξαν και μερικοί που είχαν προβλήματα.

2.Πώς σας φάνηκαν οι μηχανισμοί του παιχνιδιού (άλμα, πυροβολισμός, ολίσθηση, αναρρίχηση τοίχου, αναρρίχηση σκάλας); (1: Καθόλου ικανοποιημένος, 5: Απόλυτα ικανοποιημένος)

- Μέσος Όρος: 3.5
- Οι παίκτες βρήκαν τους μηχανισμούς αποδεκτούς, με ορισμένες βελτιώσεις να είναι απαραίτητες, ειδικά στον τομέα της αναρρίχησης και του άλματος.

3.Πόσο ικανοποιημένοι είστε με την ποικιλία των εχθρών και το επίπεδο δυσκολίας που παρέχουν; (Ναι / Όχι - Εξηγήστε)

- Μέσος Όρος: 3.7
- Η ποικιλία των εχθρών και η δυσκολία θεωρήθηκαν γενικά καλές, με μερικούς παίκτες να προτείνουν περισσότερη ποικιλία ή αυξημένη πρόκληση.

4.Η δυσκολία του παιχνιδιού ήταν κατάλληλη για εσάς; (1: Καθόλου διασκεδαστικές, 5: Πολύ διασκεδαστικές)

- Ναι: 70%
- Όχι: 30%
- Χαρακτηριστικές Απαντήσεις:
  - "Ναι, η δυσκολία ήταν ακριβώς όπως έπρεπε για μένα, ούτε πολύ εύκολη ούτε πολύ δύσκολη."
  - "Όχι, το παιχνίδι ήταν υπερβολικά δύσκολο και με αποθάρρυνε από το να συνεχίσω."

5.Πόσο διασκεδαστικές βρήκατε τις τρεις πίστες του παιχνιδιού; (Ναι / Όχι - Αν ναι, παρακαλώ περιγράψτε)

- Μέσος Όρος: 3.9
- Οι περισσότερες πίστες θεωρήθηκαν διασκεδαστικές, με τους παίκτες να αναφέρουν ότι η δυσκολία ανέβαινε με ικανοποιητικό τρόπο.

6.Υπήρχε κάποιο σημείο ή μηχανισμός που βρήκατε ιδιαίτερα δύσκολο ή εκνευριστικό; (Ναι / Όχι - Εξηγήστε)

- Ναι: 55%
- Όχι: 45%
- Χαρακτηριστικές Απαντήσεις:
  - "Ναι, το άλμα ήταν δύσκολο να ελεγχθεί, ιδιαίτερα σε στενά σημεία."

- "Όχι, βρήκα τους μηχανισμούς εύχρηστους και διασκεδαστικούς."

7.Η αλληλεπίδραση με τους εχθρούς ήταν δίκαιη και ισορροπημένη;

- Ναι: 65%
- Όχι: 35%
- Χαρακτηριστικές Απαντήσεις:
  - "Ναι, η αλληλεπίδραση με τους εχθρούς ήταν δίκαιη και οι μάχες ήταν ισορροπημένες."
  - "Όχι, κάποιοι εχθροί ήταν υπερβολικά δυνατοί και δεν μπορούσα να τους αντιμετωπίσω αποτελεσματικά."

8.Το σύστημα των σημείων επανεκκίνησης (respawn points) ήταν αρκετά δίκαιο και κατάλληλα τοποθετημένο; (Ναι / Όχι - Εξηγήστε)

- Ναι: 75%
- Όχι: 25%
- Χαρακτηριστικές Απαντήσεις:
  - "Ναι, τα σημεία επανεκκίνησης ήταν καλά τοποθετημένα και δίκαια."
  - "Όχι, τα σημεία επανεκκίνησης ήταν πολύ αραιά και με δυσκόλευαν να προχωρήσω."

9.Πώς θα αξιολογούσατε τα γραφικά και τον σχεδιασμό των επιπέδων; (1: Πολύ κακά, 5: Εξαιρετικά)

- Μέσος Όρος: 4.1
- Τα γραφικά και ο σχεδιασμός των επιπέδων εκτιμήθηκαν ιδιαίτερα, με αρκετούς παίκτες να τα θεωρούν εξαιρετικά για τον σκοπό του παιχνιδιού.

10.Θα προτείνατε αυτό το παιχνίδι σε κάποιον άλλον; (Ναι / Όχι - Εξηγήστε)

- Ναι: 80%
- Όχι: 20%
- Χαρακτηριστικές Απαντήσεις:
  - "Ναι, σίγουρα θα το πρότεινα σε άλλους, είναι πολύ διασκεδαστικό!"
  - "Όχι, θα χρειαστεί βελτιώσεις πριν το προτείνω."