ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΙΚΟΝΙΚΟΥ ΠΕΡΙΒΑΛΛΟΝΤΟΣ ΣΕ ΜΗΧΑΝΗ ID TECH 3

ΓΕΩΡΓΙΟΣ ΓΕΛΑΡΔΟΣ (ΑΜ 2096)
ΜΙΧΑΗΛ ΧΑΤΖΗΙΩΑΝΝΟΥ (ΑΜ 2258)
(πρώην Τμήματος ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ, ΤΕΙ ΔΥΤ. ΕΛΛΑΔΑΣ)

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: ΑΘΑΝΑΣΙΟΣ ΚΟΥΤΡΑΣ

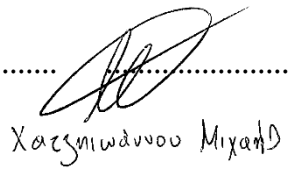
ΠΑΤΡΑ, 2022

[Αυτή η σελίδα είναι κενή]

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΜΗ ΛΟΓΟΚΛΟΠΗΣ

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Ακόμα δηλώνω ότι αυτή η γραπτή εργασία προετοιμάστηκε από εμένα προσωπικά και αποκλειστικά και ειδικά για την συγκεκριμένη πτυχιακή εργασία και ότι θα αναλάβω πλήρως τις συνέπειες εάν η εργασία αυτή αποδειχθεί ότι δεν μου ανήκει.

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ 1	**ΑΜ**	**ΥΠΟΓΡΑΦΗ**
ΓΕΛΑΡΔΟΣ ΓΕΩΡΓΙΟΣ	2096	
.....	.....	.....

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ 2 *(σε περίπτωση που είναι απαραίτητο)*	**ΑΜ**	**ΥΠΟΓΡΑΦΗ**
ΧΑΤΖΗΙΩΑΝΝΟΥ ΜΙΧΑΗΛ	2258	
.....	.....	.....

[Αυτή η σελίδα είναι κενή]

ΕΥΧΑΡΙΣΤΙΕΣ

Ολοκληρώνοντας την παρούσα πτυχιακή σε ένα ιδιαίτερο και ενδιαφέρον αντικείμενο, το level design, θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή μας κ. Αθανάσιο Κούτρα για την αποδοχή του θέματος και τη συνεργασία του για την επίλυσή του.

Επίσης, θα θέλαμε να ευχαριστήσουμε τους MarcusG, pcj27 και djole22lt, μέλη της κοινότητας του παιχνιδιού για το χρόνο που αφιέρωσαν δοκιμάζοντας την πίστα μας, δίνοντας ιδέες για βελτιώσεις.

[Αυτή η σελίδα είναι κενή]

ABSTRACT

This thesis is a study on the video game Level Design using the toolchain “Radiant” as well as its representation in id Tech 3 engine. Since the 90s, a variety of game engines not only have given the developers the ability to develop 3D worlds but also fully functional games. The thesis contains a short historical review of first person shooter games and popular game engines with a particular emphasis on id Tech as well as Radiant. The objective of this thesis is the development of a 3D level, with the combination of code writing, which will be playable in the game *Call of Duty* (2003).

ΠΕΡΙΛΗΨΗ

Η συγκεκριμένη πτυχιακή εργασία ασχολείται με τη σχεδίαση και ανάπτυξη τρισδιάστατου κόσμου (Level Design) με τη χρήση του προγράμματος Radiant καθώς και την προβολή της στη μηχανή id Tech 3. Από τη δεκαετία του 1990, διαφορετικές μηχανές παιχνιδιών (Game Engines) έχουν δώσει την ικανότητα στους προγραμματιστές να σχεδιάζουν εικονικούς κόσμους αλλά και ολόκληρα παιχνίδια. Στην παρούσα πτυχιακή γίνεται μια ιστορική αναδρομή των παιχνιδιών πρώτου προσώπου και γίνεται αναφορά σε διάφορα Game Engines με ιδιαίτερη βαρύτητα στην id Tech καθώς και το Radiant. Στόχος της πτυχιακής είναι η ανάπτυξη μιας, σε όλες τις πτυχές, ολοκληρωμένης τρισδιάστατης πίστας σε συνδυασμό με σύνταξη κώδικα, την οποία θα διαβάζει το παιχνίδι *Call of Duty* (2003).

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

id Tech, Game Engine, Radiant, Level Design

[Αυτή η σελίδα είναι κενή]

ΠΕΡΙΕΧΟΜΕΝΑ

[Αυτή η σελίδα είναι κενή]

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

[Αυτή η σελίδα είναι κενή]

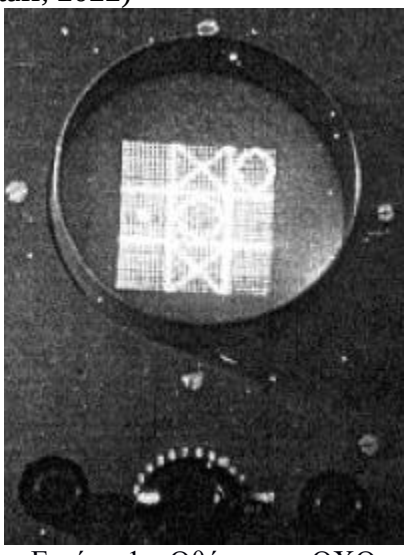
1. ΕΙΣΑΓΩΓΗ ΣΤΑ VIDEO GAMES

1.1. Ιστορική Αναδρομή

1.1.1. Τα πρώτα παιχνίδια

Το παλαιότερο γνωστό παιχνίδι είναι η Συσκευή Διασκέδασης Καθοδικού Σωλήνα (Cathode Ray Tube Amusement Device) των Thomas T. Goldsmith Jr. και Estle Ray Mann, όπου δημιουργήθηκε το 1947. Ο χρήστης, με ένα μοχλό, μετακινούσε μια ακτίνα καθοδικού σωλήνα – κουκίδα στην οθόνη – ώστε να την τοποθετήσει στις προκαθορισμένες συντεταγμένες ενός αεροπλάνου, το οποίο ήταν ζωγραφισμένο πάνω στην οθόνη CRT, εντός χρονικού ορίου και να το πυροβολήσει πατώντας ένα κουμπί. (Norman, 2022)

Το 1952 στο Πανεπιστήμιο του Cambridge ο Alexander Shafto Douglas δημιούργησε το OXO, γνωστό σε εμάς ως «Τρίλιζα», στον EDSAC υπολογιστή του Πανεπιστημίου, ο οποίος είχε χωρητικότητα περίπου 2 kilobytes. Ο χρήστης έπαιζε τρίλιζα με αντίπαλο τον EDSAC και τα πρωτόγνωρα γραφικά εμφανίζονταν σε οθόνη CRT. (History-Computer-Staff, 2022)

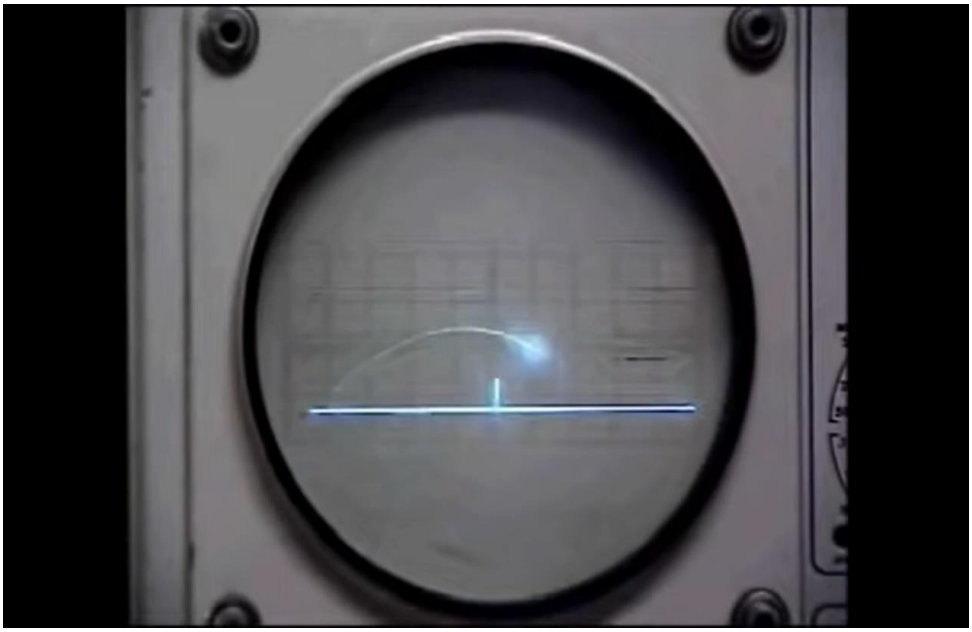


Εικόνα 1 - Οθόνη του OXO

Ακολούθησε το γνωστό “Tennis for Two” από τον William Higinbotham σε συνεργασία με τον Robert Dvorak το 1958, στο Εθνικό Εργαστήριο του Brookhaven στην Αμερική. Το παιχνίδι λειτουργούσε στον αναλογικό υπολογιστή Donner Model 30, όπου μια από τις λειτουργίες του αλλά και βασική έμπευση του Higinbotham ήταν ο υπολογισμός της τροχιάς βαλλιστικών πυράβλων.

Ο χρήστης έβλεπε στην οθόνη του παλμοσκοπίου ένα γήπεδο τένις από το πλάι, και συνδεδεμένα στον υπολογιστή ήταν δυο χειριστήρια αλουμινίου με ένα κουμπί για το χτύπημα της μπάλας και ένα περιστροφικό κουμπί για τον έλεγχο της τροχιάς. Επιπλέον, ο Higinbotham σχεδίασε το παιχνίδι έτσι ώστε αν η μπάλα δεν περάσει ή βρει στο δίχτυ, ο αντίστοιχος παίχτης να χάσει τον πόντο.

Μεγάλη καινοτομία στον σχεδιασμό ήταν η χρήση νέων πυκνωτών που επέτρεψαν στο κύκλωμα να δέχεται τρεις εισόδους από τον υπολογιστή και να τις εμφανίζει στο παλμοσκόπιο με ταχύτητα 36 Hertz – την μπάλα, το δίχτυ και το γήπεδο, με αποτέλεσμα να φαίνεται σαν μια εικόνα στο ανθρώπινο μάτι. (History-Computer-Staff, The Complete History of Tennis for Two, 2021)



Εικόνα 2 - Η οθόνη του Tennis for two

Εικόνα 3 - Χειριστήριο του Tennis for two

Με τα χρόνια δημιουργήθηκαν κι άλλα παιχνίδια – ορόσημα όπως το “Odyssey” το 1972, η πρώτη κονσόλα παιχνιδιών οικιακής χρήσης, το θρυλικό Space Invaders το 1978 την εποχή των arcades και το γνωστό σε όλους Tetris το 1984. Κατά την επόμενη δεκαετία δημιουργήθηκε ένα νέο είδος παιχνιδιών – τα First Person Shooters.

1.1.2. Ιστορία First Person Shooters (FPS) Games

Ο τίτλος του πρώτου First Person Shooter έχει δοθεί στο Wolfenstein 3D της id Software το 1992. Το παιχνίδι καινοτόμησε με το γρήγορο και γεμάτο δράση gameplay του, την πλούσια παλέτα 256 χρωμάτων και τοίχους με «υφή» (textures). Σε ένα χρόνο πωλήθηκαν 200,000 αντίγραφα, και άλλους 6 μήνες μετά κυκλοφόρησε μια μεγαλύτερη επιτυχία της εταιρείας.

Το 1993 κυκλοφόρησε το Doom, με την νέα μηχανή id Tech 1. Η id Software μπορούσε να ορίσει textures και στο πάτωμα και ταβάνι σε αντίθεση με την id Tech 0, μεταβαλλόμενο φωτισμό και ύψος εδάφους όπως σκαλιά και ράμπες.

Οι καινοτομίες της id Software συνέχισαν με το Quake το 1996, το πρώτο παιχνίδι με πραγματικά τρισδιάστατο περιβάλλον και πολυγωνικά μοντέλα. Η μεγάλη επιτυχία του ώθησε και άλλες εταιρείες να ασχοληθούν με τα First Person Shooters.

Η εταιρεία Rare δημιούργησε το GoldenEye 007 το 1997, παιχνίδι βασισμένο στην αντίστοιχη ταινία James Bond, και κυκλοφόρησε για την κονσόλα Nintendo 64 με μεγάλη επιτυχία χάρη στην ιστορία της καμπάνιας και τον σχεδιασμό των πιστών για multiplayer.

Ένα χρόνο αργότερα κυκλοφόρησε το Half-Life της Valve, βασισμένο σε παρόμοια μηχανή με το Quake. Το παιχνίδι κατέληξε να είναι ένα από τα σημαντικότερα και καλύτερα παιχνίδια που έχουν κυκλοφορήσει χάρη στον τρόπο αφήγησης της ιστορίας· δεν υπήρχαν σκηνές, και ο χρήστης παρέμενε σε πρώτο πρόσωπο καθ' όλη τη διάρκεια του παιχνιδιού. Επίσης πρόσφερε μεγάλη διαδραστικότητα με το λεπτομερώς σχεδιασμένο περιβάλλον του, και τους έξυπνους χαρακτήρες NPC.

Το 1999 κυκλοφόρησε το Quake III Arena, το Unreal Tournament που χρησιμοποιούσε την πασίγνωστη μηχανή Unreal Engine, καθώς και το Counter-Strike (τροποποίηση του Half-Life) τα οποία ανέβασαν τον πήχη του παιχνιδιού με πολλαπλούς παίχτες χάρη στις ισορροπημένες και όμορφες πίστες που είχαν σχεδιαστεί.

1.1.3. Εξέλιξη παιχνιδιών

Πολλές βασικές αρχές για ένα καλά σχεδιασμένο First Person Shooter είχαν ήδη οριστικοποιηθεί πριν το 2000, ωστόσο οι εξελίξεις συνέχισαν με τους γνωστούς τίτλους όπως το Counter-Strike: Source (2004), το πιο γνωστό παιχνίδι FPS πολλαπλών παιχτών, τη σειρά Halo που συνέχισε τα First Person Shooters και στις κονσόλες, τα Battlefield & Call of Duty που πρόσφεραν εμβυθιστικές ιστορίες για single-player και online multiplayer. Αυτές οι εξελίξεις ωστόσο δυσκολεύουν ολοένα και περισσότερο τον σχεδιασμό πιστών· οι περισσότεροι παίχτες και η αυξημένη ζήτηση στο τομέα απαιτούν εκτεταμένη ανάλυση κατά τον σχεδιασμό έτσι ώστε το τελικό αποτέλεσμα να είναι μια ισορροπημένη, ενδιαφέρουσα και διαδραστική πίστα.

1. Μηχανές Παιχνιδιών

1.1. Τι είναι μια μηχανή παιχνιδιού

Μια μηχανή παιχνιδιών (Game engine) είναι ένα περιβάλλον υλοποίησης λογισμικού που προσφέρει επιλογές και ρυθμίσεις για την εύκολη και βέλτιστη υλοποίηση ενός παιχνιδιού. Επίσης μπορεί να παρέχει επιπρόσθετες λειτουργίες όπως μηχανισμούς για εξαγωγή γραφικών 2D ή 3D, ηχητικών εφέ, κινουμένων σχεδίων.

Οι μηχανές παιχνιδιών γρήγορα έγιναν απαραίτητες διότι οι δημιουργοί μπορούσαν να αναπτύξουν ένα νέο παιχνίδι ή να το προσφέρουν σε μια νέα πλατφόρμα χωρίς να σπαταλούν χρόνο για την υλοποίηση βασικών λειτουργιών όπως οι νόμοι της φύσης του παιχνιδιού.

1.2. Οι διασημότερες μηχανές παιχνιδιών

Με την αυξημένη ζήτηση στο τομέα υπάρχει και μεγάλη ζήτηση για τα κατάλληλα εργαλεία. Παρακάτω θα αναφερθούν ορισμένες διάσημες μηχανές παιχνιδιών, καθώς και η id Tech που χρησιμοποιήθηκε για την παρούσα πτυχιακή εργασία.

1.2.1. Unreal Engine

Ίσως η διασημότερη μηχανή, δημιουργήθηκε το 1998 για το παιχνίδι Unreal από τον Tim Sweeney και χρειάστηκε 3,5 χρόνια. Με κύριο ανταγωνιστή την id Software, η Unreal Engine καινοτόμησε με τα γραφικά της που παρείχαν παλέτα χρωμάτων 16bit (65536 διαθέσιμα χρώματα), ρεαλιστικές πηγές φωτός και καπνού. Γνωστοί τίτλοι όπως Mass Effect, Bioshock, Rocket League, Fortnite χρησιμοποιούν εκδόσεις της Unreal Engine. Η 5^{η} έκδοση, Unreal Engine 5, έγινε διαθέσιμη στους δημιουργούς στις 5 Απριλίου 2022 και παρέχει την δυνατότητα φωτογραμμετρίας, με την οποία εξάγονται δεδομένα 3D από φωτογραφίες, με αποτέλεσμα την δημιουργία λεπτομερούς περιβάλλοντος σε σύντομο χρονικό διάστημα.

Εικόνα 4 - Περιβάλλον σε demo της Unreal Engine 5

1.2.2.Unity

Ανακοινώθηκε το 2005 από την εταιρεία Unity Technologies με σκοπό την εύκολη πρόσβαση σε αυτή από τους δημιουργούς. Συμβατή με τις περισσότερες πλατφόρμες στην αγορά όπως iOS, Android, Windows, η μηχανή χρησιμοποιήθηκε και σε τομείς εκτός των παιχνιδιών όπως στον κινηματογράφο, αρχιτεκτονική και εξομοιώσεις. Σύμφωνα με την Unity, το 50% των παιχνιδιών σε κινητά τηλέφωνα είναι φτιαγμένα με την μηχανή της.

Οι μεγαλύτερες διαφορές μεταξύ Unity και Unreal είναι ότι η Unreal προσφέρει καλύτερες επιδόσεις και γραφικά, ωστόσο η Unity κυριαρχεί διότι δεν απαιτεί δικαιώματα από τους δημιουργούς, παρέχει δωρεάν έκδοση αλλά και πλήρης επι πληρωμή μιας φοράς, και παρέχει περισσότερες υφές (textures) και αντικείμενα (props) μέσω του Asset Store. (Trainor-Fogleman, 2021)

1.2.3.Id Tech

Η id Tech είναι μια σειρά από μηχανές που ειδικεύονται στην προσομοίωση τρισδιάστατων περιβαλλόντων για χρήση σε βιντεοπαιχνίδια πρώτου προσώπου. Η πρώτη κύρια έκδοση της μηχανής κατασκευάστηκε το 1993 από τον John Carmack, ένα από τα τέσσερα μέλη της εταιρείας id Software. Η πιο πρόσφατη έκδοση της μηχανής (id Tech 7) δημοσιεύτηκε το 2018. Παρακάτω θα δούμε με μεγαλύτερη λεπτομέρεια την εξέλιξη της id Tech και τις βελτιώσεις που δέχτηκε με τα χρόνια.

Πριν την πρώτη επίσημη έκδοση της μηχανής, η id Software κυκλοφόρησε το παιχνίδι πρώτου προσώπου με όνομα Wolfenstein 3D (1992). Μιας και η μηχανή βρισκόταν ακόμα σε πολύ πρώιμα στάδια, το περιβάλλον και οι εχθροί του Wolfenstein δεν ήταν 3D, αλλά 2.5D. Δεν υπήρχε δυνατότητα εφαρμογής υφών (textures) στο δάπεδο και στο ταβάνι του εικονικού κόσμου. Άλλο χαρακτηριστικό είναι ότι η κάμερα του παίκτη κινείται με αργή ταχύτητα με αποτέλεσμα η μηχανή να μπορεί εύκολα να φορτώσει τα διαφορετικά δωμάτια του κόσμου χωρίς να επιβραδύνει το σύστημα στο οποίο "τρέχει".

Εικόνα 5 - Wolfenstein 3D (1992)

1.2.3.1.1. id Tech 1

Η id Tech 1 υποστήριξε τα παιχνίδια πρώτου προσώπου Doom (1993) και Doom II (1994), γι' αυτό και ονομάστηκε Doom Engine.

Όπως και το Wolfenstein 3D, έτσι και τα Doom δεν έιναι 3D, αλλά ψεύδο-3D (ή 2.5D). Βέβαια, η μηχανή είχε δεχτεί πολλές βελτιώσεις από τον Carmack. Πλέον οι προγραμματιστές θα μπορούσαν να δημιουργήσουν πιο σύνθετους κόσμους με μεγαλύτερη ποικιλία υφών καθώς και η προσθήκη πηγών φωτός βοήθησε στο να κάνει τους κόσμους πιο ατμοσφαιρικούς.

Εικόνα 6 - Το πρώτο επίπεδο του Doom (1993)

Κατά την κατασκευή του Doom όμως, ο Carmack αντιμετώπισε ένα πολύ σοβαρό πρόβλημα. Εκτός από το γεγονός ότι οι κόσμοι του Doom ήταν πολύ πιο σύνθετοι, οι προγραμματιστές του παιχνιδιού αποφάσισαν η κάμερα του παίκτη να κινείται με πολύ μεγαλύτερη ταχύτητα σε σύγκριση με αυτή του Wolfenstein 3D. Αυτό είχε ως αποτέλεσμα η μηχανή να επιβραδύνει αρκετά την ομαλότητα κίνησης της κάμερας αφού για κάθε κόσμο που φόρτωνε, διάβαζε απευθείας όλα τα δεδομένα του.

Έτσι χρησιμοποίησε τη μέθοδο της δυαδικής κατάτμησης χώρου (Binary Space Partitioning). Κατά την διαδικασία μετατροπής του αρχείου επεξεργασίας του εικονικού κόσμου σε αρχείο που διαβάζει και προσομοιώνει η μηχανή, ο κόσμος διαιρείται σε δυαδικά δέντρα δηλαδή σε επιμέρους "κελιά".

Η διαδικασία είναι η εξής:

⇒ Δημιουργείται ο αρχικός κόμβος Α που περιλαμβάνει ολόκληρο τον εικονικό κόσμο.

⇒ Μέσω ενός αλγορίθμου, χρησιμοποιείται μια διαχωριστική γραμμή που διαιρεί τον κόσμο σε δύο περιοχές που ονομάζονται κόμβοι-παιδιά Β και C, με προτεραιότητα τον κόμβο που βρίσκεται πιο κοντά στην κάμερα του παίκτη. Αντίστοιχα, οι περιοχή του κόμβου Β χωρίζεται σε επιμέρους περιοχές μέσω μιας νέας διαχωριστικής γραμμής.

Με αυτή τη μέθοδο, σχεδιάζεται μόνο η γεωμετρία του κόσμου που έχει πρόσβαση ο παίκτης και έτσι μπορούν να σχεδιαστούν μεγαλύτεροι εικονικοί κόσμοι χωρίς να επιβραδύνεται η ταχύτητα του παιχνιδιού. Γι' αυτό και τα αρχεία των εικονικών κόσμων που διαβάζει η μηχανή έχουν την κατάληξη *BSP*. (History, 2019)

Παρά τις βελτιώσεις όμως, η μηχανή διατήρησε περιορισμούς. Ένας από τους κυριότερους περιορισμούς της μηχανής Doom είναι ότι στους κόσμους του παιχνιδιού δεν μπορούν να κατασκευαστούν δύο ή περισσότερα δωμάτια ("κελιά") το ένα πάνω από το άλλο (room-over-room).

Άλλοι περιορισμοί είναι ότι ο χρήστης μπορεί να περιστρέφει την κάμερα μόνο σε οριζόντιο άξονα και όχι σε κάθετο. Επίσης, οι εχθροί που κινούνται στο εικονικό περιβάλλον συντίθενται από μια δισδιάστατη εικόνα (sprite) η οποία είναι προγραμματισμένη να αντικρίζει πάντα την κάμερα του χρήστη.

1.2.3.1.2. id Tech 2

Δύο χρόνια μετά, ο Carmack βελτίωσε τη μηχανή και δημοσίευσε τη δεύτερη έκδοσή της ώστε να υποστηρίξει το παιχνίδι Quake (1996) της id Software.

Στο Quake χρησιμοποιείται ένα σύστημα στο μέρος του level design που προ-καταχωρεί δεδομένα στο αρχείο του εικονικού κόσμου αυξάνοντας την ταχύτητα που τον διαβάζει η μηχανή. Το αρχείο που επεξεργάζονται οι προγραμματιστές κόσμων, έχει τη μορφή *MAP*. Χρησιμοποιούν τρισδιάστατα γεωμετρικά σχήματα (brushes) για να δημιουργήσουν ένα κλειστό και ογκομετρικό περιβάλλον στο οποίο ο παίκτης θα μπορει να κινείται.

Εικόνα 7 - Gameplay του Quake (1996)

Αφού ολοκληρωθεί ένας εικονικός κόσμος, μετατρέπεται σε αρχείο *BSP* μέσω ενός προγράμματος (renderer). Το πρόγραμμα αυτό έχει τη δυνατότητα να εντοπίσει σε ποια διαστήματα του κόσμου ο παίκτης μπορεί να έχει πρόσβαση και σε ποια όχι. Αφού γίνει αυτό, το renderer αφαιρεί τις πλευρές των γεωμετρικών σχημάτων (brushes) στα οποία ο παίκτης δεν μπορεί να δει, μειώνοντας έτσι την πολυπλοκότητα του τρισδιάστατου κόσμου.

Προ-καταχωρημένα στα αρχεία *BSP* είναι πλέον και τα *lightmaps* καθώς και οι πηγές φωτός, μειώνοντας έτσι το υπολογιστικό κόστος της μηχανής αφού το rendering των *BSP* αρχείων γίνεται εκτός αυτής. Με τον όρο *lightmaps* εννοούμε μία δομή δεδομένων κατά την οποία η φωτεινότητα κάθε πλευράς των σταθερών γεωμετρικών

σχημάτων, από τα οποία αποτελείται ο εικονικός κόσμος (τοίχοι, κουτιά κλπ.) των προγραμματιστών, εξαρτάται από τις πηγές φωτός που οι ίδιοι τοποθετούν σε αυτόν και είναι προκαθορισμένη από τον compiler. (Houska, 2006)

Οι νέες προσθήκες είχαν ως αποτέλεσμα οι παίχτες του παιχνιδιού Quake να βυθιστούν για μεγαλύτερα χρονικά διαστήματα στο εικονικό περιβάλλον του.

1.2.3.1.3. id Tech 3

Καθώς οι υπολογιστές γίνονται όλο και πιο δυνατοί, ακολούθησαν δύο βελτιώσεις της μηχανής από τον Carmack που στόχευσαν στην βελτίωση του φωτισμού που δημιουργούν οι πηγές φωτός στον τρισδιάστατο κόσμο αλλά και βελτιώσεις στα μοντέλα του κόσμου, όπως μοντέλα εχθρών και όπλων. Προστέθηκαν περισσότερα γεωμετρικά δεδομένα (triangles) στα μοντέλα και αυξήθηκε η λεπτομέρεια κίνησής τους (αυξημένο καρέ-ανά-καρέ). (Henry, 2004)

Επίσης, τα όρια της μηχανής αυξήθηκαν δίνοντας τη δυνατότητα στους level designers να κατασκευάζουν μεγαλύτερους κόσμους, με μεγαλύτερη λεπτομέρεια όπως καμπυλωτές επιφάνειες (curved terrain). Ακόμη, παρουσιάστηκαν τα *shaders*, ένας έξυπνος τρόπος για να αυξηθεί ο ρεαλισμός του εικονικού περιβάλλοντος.

Τα *shaders*, όπως και τα textures, εφαρμόζονται σε πλευρές των γεωμετρικών σχημάτων του κόσμου. Η διαφορά είναι ότι τα *shaders* αποτελούν ένα σύνολο υφών που τροποποιούν τις ιδιότητες των pixel τους. (Jaquays, 2000)

Μετά την κυκλοφορία του παιχνιδιού Quake III (1999), προγραμματιστές άλλων εταιρειών άρχισαν να χρησιμοποιούν την id Tech 3 για να κατασκευάσουν τα δικά τους παιχνίδια. Χαρακτηριστικά παιχνίδια είναι τα "Return To Castle Wolfenstein" και "Call of Duty".

Εικόνα 8 - Καμπυλωτές επιφάνειες στο Call of Duty (2003)

1.2.3.1.4. id Tech 4

Το 2003, ο Carmack αναβάθμισε τη μηχανή και πρόσθεσε νέες τεχνολογίες στο γραφικό της κομμάτι που θα μπορούσαν να δώσουν έναν ακόμα μεγαλύτερο ρεαλισμό στον τρισδιάστατο κόσμο των παιχνιδιών. Η σημαντικότερη προσθήκη είναι η τεχνολογία *MegaTexture rendering*.

Στις αρχές του 2000, για τους προγραμματιστές κόσμων ήταν δύσκολο να εμπλουτίσουν τους εξωτερικούς χώρους, που κατασκεύαζαν, με λεπτομέρεια. Με την τεχνολογία *MegaTexture rendering* όμως, μπορούσε να δημιουργηθεί μια υφή (texture) σε πολύ μεγάλη ανάλυση, συγκεκριμένα 32768x32768. Η τεράστια αυτή υφή αποτελείται από ένα σύνολο άλλων υφών υψηλής ποιότητας και χωρίζεται σε εικονικά "πλακάκια", ανάλυσης συνήθως 128x128 το κάθε ένα. Οι προγραμματιστές την εφαρμόζουν σε επιφάνεια που καλύπτει μεγάλη έκταση στον κόσμο τους. Κατά τη διάρκεια του παιχνιδιού, η μηχανή σχεδιάζει μόνο τα πλακάκια που είναι ορατά στην κάμερα του παίκτη και αποκρύπτει τα υπόλοιπα. (Dammertz, 2012)

Εικόνα 9 - MegaTexture Rendering στο παιχνίδι Rage

Η τεχνολογία αποτέλεσε μία έξυπνη τεχνική που χρησιμοποιείται ακόμα και σήμερα σε πολλά παιχνίδια της id Tech για τη δημιουργία πλούσιων τρισδιάστατων κόσμων.

1.2.3.1.5. id Tech 5-7

Η μηχανή συνέχισε να αναβαθμίζεται με τα χρόνια για να καλύψει τις απαιτήσεις των εταιρειών που κατασκευάζουν βιντεοπαιχνίδια. Στις επόμενες εκδόσεις, βελτιώθηκε η τεχνολογία MegaTexture rendering, αυξάνοντας το μέγιστο όριο ανάλυσης των textures. Επίσης, βελτιώθηκε ακόμη περισσότερο το εφέ σκιάς στον κόσμο, οι φυσικές των αντικειμένων που αλληλοεπιδρούν με τον κόσμο και η τεχνητή νοημοσύνη των εχθρών.

Προστέθηκαν και νέες τεχνικές γραφικών με σκοπό να κάνουν τον κόσμο που κινείται ο παίκτης ακόμα πιο ρεαλιστικό. Ορισμένες από αυτές είναι η "Αντανάκλαση Χώρου Οθόνης" (Screen Space Reflection) που εφαρμόζεται σε σημεία του κόσμου που υπάρχει αντανάκλαση, όπως νερό και καθρέφτες, και τα εφέ "Θόλωση Κίνησης" (Motion Blur) και "Χρωματική Εκτροπή" (Chromatic Aberration).

Εικόνα 10 - Screen Space Reflection στο παιχνίδι Cyberpunk 2077 (2020)

Στην Εικόνα 10, βλέπουμε ότι το έδαφος αντικαθρεπτίζει τον κόσμο που κοιτάει ο παίκτης. Όσο ο παίκτης κοιτάει πιο χαμηλά, ο κόσμος που χάνεται από το πάνω μέρος της οθόνης, χάνεται και από το έδαφος. Ο αλγόριθμος Screen Space Reflection χρησιμοποιείται συχνά από μηχανές παιχνιδιών στην προσπάθεια να αυξήσουν τον ρεαλισμό του κόσμου. (Anthony Paul Beug, 2020)

2. ΠΡΟΓΡΑΜΜΑ ΕΠΕΞΕΡΓΑΣΙΑΣ ΚΟΣΜΩΝ ΓΙΑ ΤΗ ΜΗΧΑΝΗ ΠΑΙΧΝΙΔΙΩΝ id Tech 3

2.1. Radiant

2.1.1. Εισαγωγή στο Radiant

Το *Radiant* αποτελεί το επίσημο πρόγραμμα δημιουργίας και επεξεργασίας τρισδιάστατων κόσμων των παιχνιδιών που βασίζονται στη μηχανή id Tech. Για κάθε παιχνίδι οι δημιουργοί χρησιμοποιούν παραλλαγές του Radiant, ο θεμελιώδης σκελετός του προγράμματος όμως παραμένει ο ίδιος.

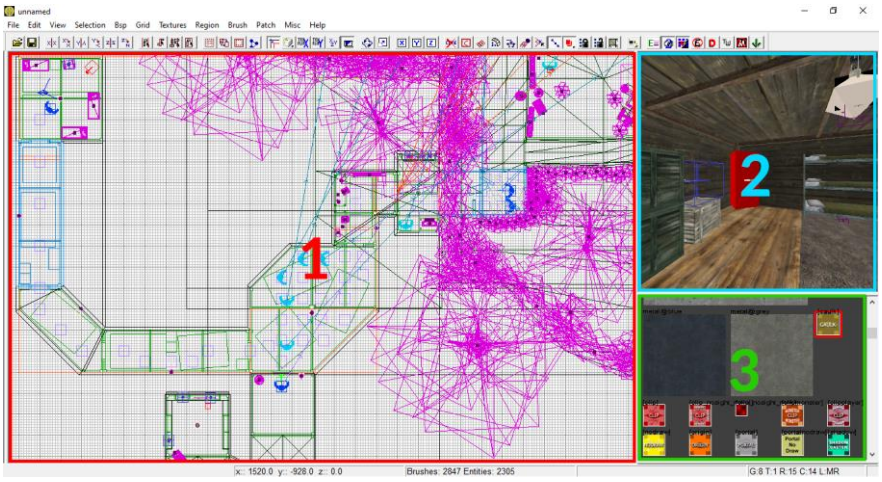
Η μηχανή χωρίζει κάθε κόσμο σε δύο κατηγορίες. Στη γεωμετρία του κόσμου (**World Geometry**) και τις οντότητες (**Entities**). (Jaquays, 2000)

⇒ Η γεωμετρία του κόσμου αντιπροσωπεύει τα γεωμετρικά σχήματα (Brushes) και σχήματα με υπολογισμένες καμπύλες (Patches) που σχεδιάζει ο προγραμματιστής στο Radiant για να χτίσει τον κόσμο. Ένα brush αποτελεί ένα συμπαγή ορθογώνιο όγκο που ορίζεται από τις συντεταγμένες των γωνιακών κορυφών του.

⇒ Οι οντότητες περιλαμβάνουν "pick-ups", όπως όπλα και σφαίρες που μπορεί να σηκώσει ο παίκτης, πηγές φωτός, σημεία από τα οποία ξεκινούν ο παίκτης και A.I. (Spawners) και σημεία που θα συμβεί ένα οπτικό ή ηχητικό εφέ.

2.1.2. Περιβάλλον του Radiant

Όπως φαίνεται και στην παρακάτω εικόνα, το περιβάλλον του Radiant αποτελείται από τρία παράθυρα και μία γραμμή εργαλείων.



Εικόνα 11 - Προεπιλεγμένο περιβάλλον του Radiant

⇒ Το παράθυρο (*1*) αποτελεί το περιβάλλον σχεδίασης σε 2D. Εκεί, σχεδιάζοντας γεωμετρικά σχήματα και τοποθετώντας τρισδιάστατα μοντέλα, δημιουργούμε τον κόσμο.

⇒ Στο παράθυρο (*2*) βλέπουμε τον κόσμο μας σε 3D και μπορούμε να κινηθούμε σε αυτόν.

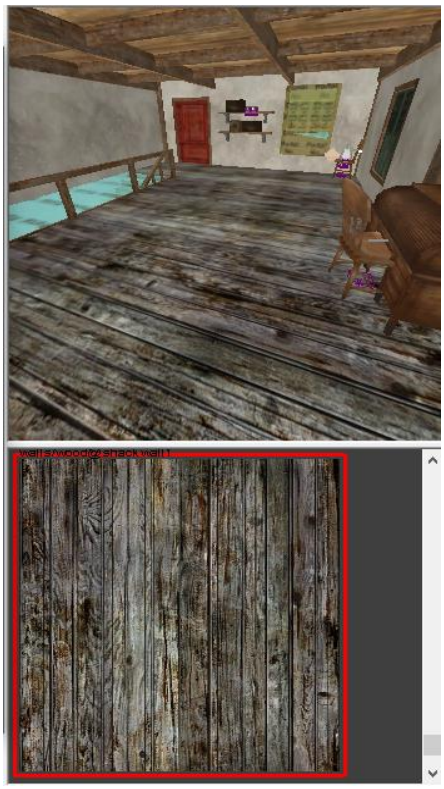
⇒ Το παράθυρο (*3*) περιλαμβάνει τις υφές που έχουμε εισάγει ή θέλουμε να εισάγουμε στον κόσμο μας.

Η γραμμή εργαλείων βρίσκεται στην κορυφή του Radiant και αποτελεί εργαλεία όπως το εργαλείο περιστροφής σχημάτων και οντοτήτων, δυνατότητα επιλογής πολλών στοιχείων του κόσμου ταυτόχρονα, εισαγωγή υφών, αλλαγή οπτικής του δισδιάστατου κόσμου και πολλά άλλα.

2.1.3. Υφές

Στις πλευρές των γεωμετρικών σχημάτων, που φτιάχνει ο προγραμματιστής, πρέπει να εφαρμοστούν υφές (Textures). Οι υφές αποτελούν την τέχνη στο περιβάλλον του κόσμου. Εφαρμόζονται σε τοίχους, έδαφος, ουρανό, ταβάνι. Ο προγραμματιστής μπορεί να χρησιμοποιήσει υφές που υπάρχουν ήδη στο παιχνίδι ή να προσθέσει τις δικές του. Οι υφές έχουν τη μορφή (*.dds*), (*.tga*) και (*.jpg*).

Κάθε texture έχει ένα ξεχωριστό όνομα· μπροστά από κάθε όνομα αναγράφεται το είδος της κάθε υφής. Για παράδειγμα, αν το πάτωμα ενός δωματίου θέλουμε να είναι ξύλινο, δεν επαρκεί μόνο να έχει την υφή αλλά και τον ήχο του ξύλου όταν ο παίκτης περπατάει πάνω του καθώς και το εικονικό εφέ της πρόσκρουσης όταν το πυροβολεί. Στην περίπτωση αυτή, το όνομα της υφής μιας ξύλινης επιφάνειας ξεκινά με "*wood@*".



Εικόνα 12 - Παράδειγμα δαπέδου με ξύλινη υφή

Άλλα είδη υφών είναι:

- *concrete@*
- *grass@*
- *dirt@*
- *metal@*
- *glass@*
- *foliage@*

Η πρώτη και κυριότερη υφή που πάντα πρέπει να εφαρμόζουμε στις πλευρές των γεωμετρικών σχημάτων, λέγεται *Caulk*. Η υφή αυτή προβάλλεται με κίτρινο χρώμα στο Radiant αλλά είναι διαφανής στο παιχνίδι. Κάθε πλευρά που ο παίκτης δεν μπορεί να δει πρέπει να έχει την υφή *Caulk*. Στις υπόλοιπες πλευρές μπορούμε να αλλάξουμε την υφή σε μία της προτίμησής μας. Με τον τρόπο αυτό αποφεύγονται πιθανά σφάλματα που μπορεί να εμφανίσει ο compiler καθώς ελαχιστοποιείται σημαντικά ο χρόνος που μετατρέπει την πίστα μας σε *BSP* και μειώνεται το συνολικό μέγεθος του αρχείου. (Jaquays, 2000)

Εικόνα 13 - Το texture *Caulk*

Πιο νωρίς, κάναμε αναφορά στα *shaders*. Τα *shaders* είναι αρχεία, ξεχωριστά από τις υφές, που περιλαμβάνουν μικρά κομμάτια κώδικα, που χρησιμοποιεί η μηχανή παιχνιδιού για να κάνει περαιτέρω προσαρμογές στην εμφάνιση ή τη λειτουργία της υφής. Από τα *shaders*, δημιουργούνται υφές όπως ο ουρανός (skybox) και το κυματιστό νερό σε μία πίστα.

2.1.4. Εισαγωγή μοντέλων

Πέρα από το να κατασκευάζουμε σχήματα για να φτιάξουμε τον κόσμο μας, μπορούμε να εισάγουμε τρισδιάστατα μοντέλα που διαθέτει το παιχνίδι όπως έπιπλα και δέντρα ώστε να τον "ντύσουμε" και να δώσουμε ζωή στο περιβάλλον. Τα μοντέλα μπορούν να είναι σταθερές οντότητες αλλά και μεταβλητές που σημαίνει ότι στο μέλλον μπορούμε να τις προγραμματίσουμε ανάλογα με τις ανάγκες μας.

Εικόνα 14 – Τρισδιάστατα μοντέλα στο Call of Duty (2003)

2.1.5. Μετατροπή του αρχείου MAP σε BSP (Compiling)

Η πίστα που διαβάζει το Radiant έχει τη μορφή *MAP*. Όταν ολοκληρώσουμε τη γεωμετρία της πίστας μας και θέλουμε να την τρέξουμε στη μηχανή παιχνιδιού (in-game), πρέπει να τη μετατρέψουμε σε αρχείο *BSP*. Για τη μετατροπή θα χρειαστεί να την περάσουμε σε έναν compiler που λέγεται *Q3MAP*. Ο compiler διαβάζει τη γεωμετρία που έχουμε κατασκευάσει και τη διαβιβάζει σε νέο αρχείο *BSP*, χωρίς όμως φωτισμό. Για να προστεθεί στο *BSP* ο φωτισμός της πίστας που έχουμε ορίσει στο Radiant, θα χρειαστεί να περάσουμε το αρχείο *BSP* σε ένα δεύτερο πρόγραμμα του λέγεται *FLARE*.

Αφού οι compiler μας επιστρέψουν με επιτυχία το αρχείο *BSP*, μπορούμε να το σύρουμε στον κατάλληλο προορισμό του παιχνιδιού και να το τρέξουμε στη μηχανή.

2.1.6. Συνήθη λάθη κατά τη διάρκεια δημιουργίας του κόσμου

Υπάρχουν ορισμένα λάθη που συνήθως κάνουν οι αρχάριοι προγραμματιστές και που πρέπει να αποφεύγουμε όταν σχεδιάζουμε έναν κόσμο, αφού έχουν ως αποτέλεσμα να αυξάνουν το μέγεθος αρχείου *BSP* και να επιβραδύνουν σημαντικά την απόδοση της μηχανής στην προσπάθειά της να διαβάσει την πίστα. Τα λάθη αυτά μπορούν ακόμα να προκαλέσουν τη μηχανή να εμφανίσει σφάλμα και να μην τρέξει καθόλου την πίστα. Τα βασικότερα είναι τα εξής:

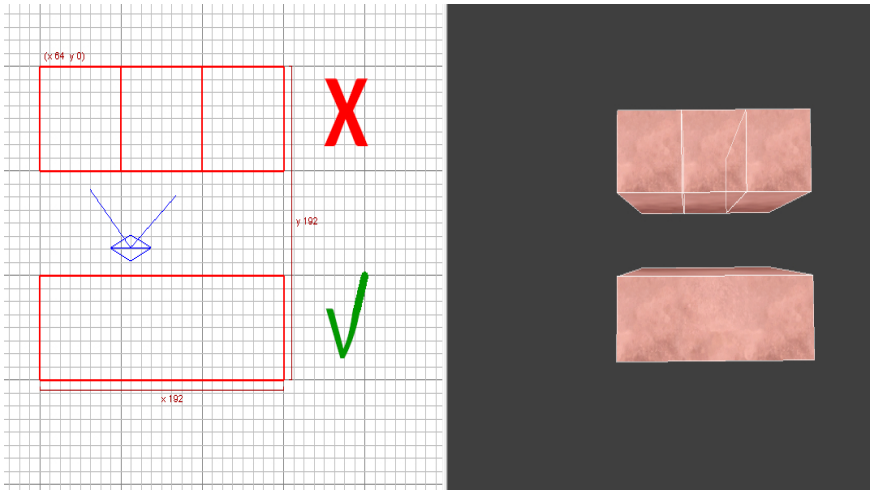
⇒ Δύο πλευρές γεωμετρικών σχημάτων με διαφορετικές υφές δεν πρέπει να μοιράζονται τον ίδιο χώρο. Αποτελεσματική κατασκευή κόσμων σημαίνει ότι όλα τα σχήματα συνδέονται μεταξύ τους, αλλά ποτέ δεν τέμνονται.

⇒ Στα σχήματα που σχεδιάζουμε με υπολογισμένες καμπύλες, οι γωνίες της κάθε πλευράς τους πρέπει να αγγίζει τις αντίστοιχες γωνίες άλλων γεωμετρικών σχημάτων και όχι να τις διαπερνά.

⇒ Όπως προαναφέραμε, οι πλευρές των γεωμετρικών σχημάτων που δεν μπορεί να δει ο παίκτης πρέπει πάντα να έχουν την υφή *Caulk*. Με αυτό τον τρόπο ο compiler δεν υπολογίζει τις πλευρές αυτές και δε χρησιμοποιεί πόρους για να τις σχεδιάσει. In-game, η συγκεκριμένη υφή είναι διαφανής.

Εικόνα 15 - Παράδειγμα χρήσης *Caulk* στο Radiant και σε μηχανή παιχνιδιού

⇒ Χρήση όσο το δυνατό λιγότερων γεωμετρικών σχημάτων. Για παράδειγμα, μπορεί να σχεδιάσουμε τρία σχήματα το ένα ακριβώς δίπλα στο άλλο, με ίδιες διαστάσεις και υφές. Αυτό σημαίνει ότι ο compiler θα υπολογίσει τρεις φορές περισσότερες πλευρές κατά τη διάρκεια μετατροπής της πίστας. Στην περίπτωση αυτή μπορούμε εύκολα να αντικαταστήσουμε τα τρία σχήματα με **ένα** ώστε ο compiler να υπολογίσει μόνο το 1/3 της συνολικής γεωμετρίας του κόσμου. Κάνοντας το συγκεκριμένο λάθος επανειλημμένα σε μία πίστα, θα αυξηθεί δραστικά ο χρόνος μετατροπής της σε *BSP*, το μέγεθος του καταληκτικού αρχείου και θα επηρεαστεί αρνητικά η απόδοση της μηχανής παιχνιδιού στην προσπάθεια να διαβάσει την πίστα. (Jaquays, 2000)



Εικόνα 16 - Υπόδειγμα βελτιστοποίησης της πίστας χρησιμοποιώντας ένα γεωμετρικό σχήμα αντί για τρία

3. ΥΛΟΠΟΙΗΣΗ ΤΡΙΣΔΙΑΣΤΑΤΟΥ ΚΟΣΜΟΥ ΜΕ ΤΗ ΧΡΗΣΗ RADIANT

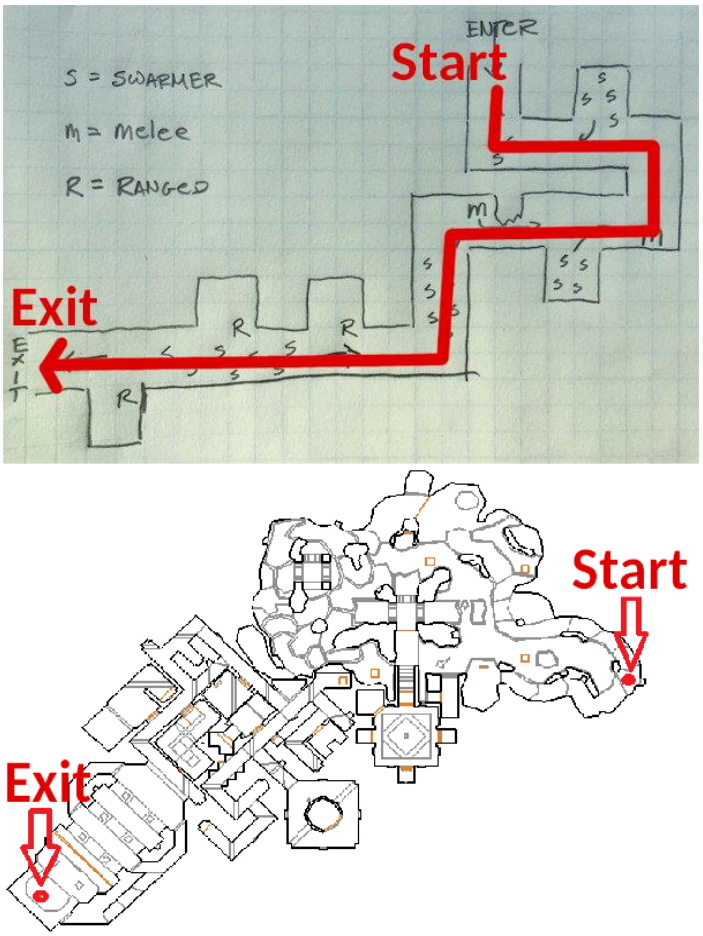
3.1. Εισαγωγή στο παιχνίδι “Call of Duty”

Το παιχνίδι που θα τρέχει η πίστα μας λέγεται "***Call of Duty***", κατασκευασμένο από την Infinity Ward το 2003. Είναι ένα παιχνίδι πρώτου προσώπου που προσομοιώνει τον πόλεμο του Β' Παγκοσμίου Πολέμου και είναι βασισμένο στη μηχανή id Tech 3. Ο παίκτης έχει το ρόλο ενός Συμμάχου στρατιώτη και παίρνει μέρος σε διάφορα ευρωπαϊκά μέτωπα.

Το Call of Duty χωρίζεται σε δύο είδη gameplay:

⇒ Στο "*Single Player*", ο παίκτης παίζει μόνος του σε πίστα με εχθρούς και συμμάχους A.I. και πρέπει να ακολουθήσει μια σειρά από στόχους (objectives) για να ολοκληρώσει το επίπεδο. Στις πίστες, το pacing μπορεί να είναι *linear* ή *non-linear*.

- *Linear* σημαίνει ότι ο παίκτης ακολουθεί ένα μόνο μονοπάτι που τον οδηγεί στην ολοκλήρωση του επιπέδου.
- Στην περίπτωση που το επίπεδο είναι *Non Linear*, ο παίκτης έχει την ευκαιρία να ακολουθήσει διαδρομές της επιλογής του για να φτάσει στο τέρμα, δίνοντάς του μεγαλύτερη ελευθερία κίνησης στο χώρο.

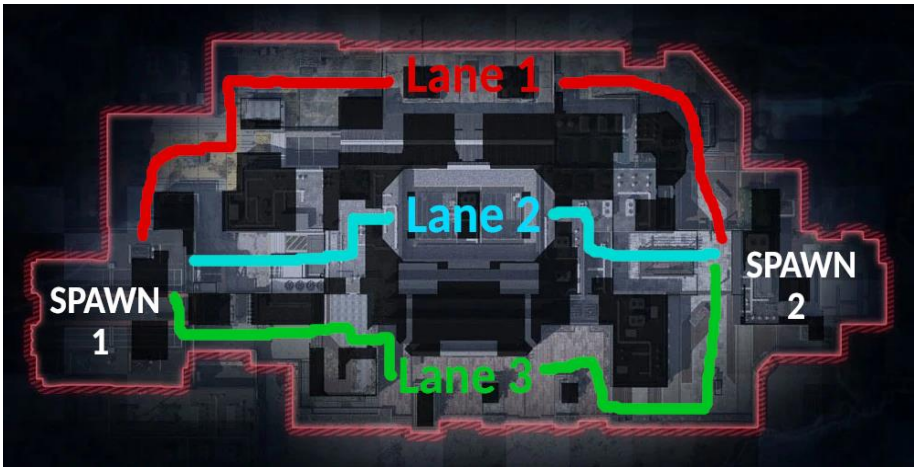


Εικόνα 17 – Παράδειγμα πίστας *Linear* vs. *Non linear*

Πίσω από τον τρισδιάστατο κόσμο "κρύβονται" πολλές γραμμές κώδικα για την επίτευξη συμβάντων (scripted events) όπως εκρήξεις, ηχητικά και οπτικά εφέ, κίνηση των A.I σε μονοπάτια κλπ.

⇒ Στο "*Multiplayer*", δεν υπάρχει δυνατότητα εισαγωγής A.I., αφού στο είδος αυτό παίζουν ανταγωνιστικά πολλοί παίκτες σε πίστες μικρότερες συνήθως από αυτές που υποστηρίζει το "*Single Player*". Το pacing χωρίζεται σε δύο κατηγορίες.

- 3-lane maps. Η πίστα χωρίζεται σε τρεις διαδρόμους που ενώνουν τις δύο περιοχές που ξεκινά κάθε ομάδα. Μεταξύ των διαδρόμων μπορεί να υπάρχει πρόσβαση αλλά όχι ορατότητα. Το είδος αυτό προτιμάται κυρίως από πιο ανταγωνιστικούς παίκτες αφού παίζει πολύ σημαντικό ρόλο η σωστή θέση του παίκτη στο χώρο που θα οδηγήσει στη νίκη αυτού ή της ομάδας του.



Εικόνα 18 – 3-Lane πίστα “Arsenal”, Call of Duty: Black Ops 4 (2018)

- Η δεύτερη κατηγορία δεν έχει συγκεκριμένη ονομασία, αλλά είναι το αντίθετο από τις 3-lane πίστες. Οι πίστες αυτές είναι πιο σύνθετες, έχουν πιο ανοιχτές περιοχές, κάνοντας τον παίκτη πιο εκτεθειμένο στον αντίπαλο με αποτέλεσμα το gameplay να είναι πιο χαοτικό αφού ο παίκτης δε νιώθει ποτέ ασφαλής.


Εικόνα 19 – Multiplayer πίστα “Crossroads”, Call of Duty: Black Ops Cold War (2020)

3.2. Μελέτη και υλοποίηση της πίστας Mlawa

Η πίστα μας αποτελεί μέρος του "*Single Player*". Της δώσαμε το όνομα "***Mlawa***" (πολωνικά: *Mława*, ελληνικά: *Μιάβα* ή *Μουάβα*), αφού είναι βασισμένη σε μία από τις πρώτες μάχες του Β' Παγκοσμίου πολέμου μεταξύ των Γερμανών και των Πολωνών. Η μάχη πραγματοποιήθηκε λίγα χιλιόμετρα έξω από την πόλη της Μουάβα μεταξύ 1 με 3 Σεπτεμβρίου 1939. Ο παίκτης έχει τον ρόλο ενός πολωνού στρατιώτη.

3.2.1. Μελέτη της πίστας

Κύριος σκοπός μας ήταν να φτιάξουμε έναν ρεαλιστικό κόσμο σε συνδυασμό με game play που θα δώσει την ευκαιρία στους παίκτες να βυθιστούν σε αυτόν και να παραμείνουν σε εγρήγορση καθ' όλη τη διάρκεια του παιχνιδιού. Θέλαμε η πίστα μας να είναι απαιτητική για τους παίκτες σε θέμα δυσκολίας, αλλά ταυτόχρονα και ενδιαφέρουσα.

Πριν ξεκινήσουμε να φτιάχνουμε τον κόσμο μας, διαβάσαμε άρθρα που περιγράφουν τη μάχη ώστε να καταλάβουμε όσο το δυνατόν περισσότερο πως εξελίχθηκε αυτή μεταξύ των δύο λαών, το περιβάλλον και τις καιρικές συνθήκες. Αμέσως καταλάβαμε ότι το βασικό gameplay που θα εφαρμόσουμε πρέπει να είναι αμυντικό, δηλαδή ο παίκτης πρέπει να αποκρούσει την επίθεση των γερμανικών δυνάμεων. Ο κόσμος πρέπει να αποτελείται από υπαίθριο περιβάλλον. Πρέπει δηλαδή να κυριαρχεί η βλάστηση, όπως δέντρα και θάμνοι. Το έδαφος πρέπει να αποτελείται κυρίως από πεδιάδες και μικρούς λόφους. Στην πραγματική μάχη ο καιρός ήταν άστατος γι' αυτό κι έπρεπε να κάνουμε το περιβάλλον συννεφιασμένο και μουντό. Αξίζει να αναφέρουμε ότι για το σχεδιασμό της πίστας πήραμε και ιδέες από το τραγούδι "*40:1*" του σουηδικού συγκροτήματος, *Sabaton*.

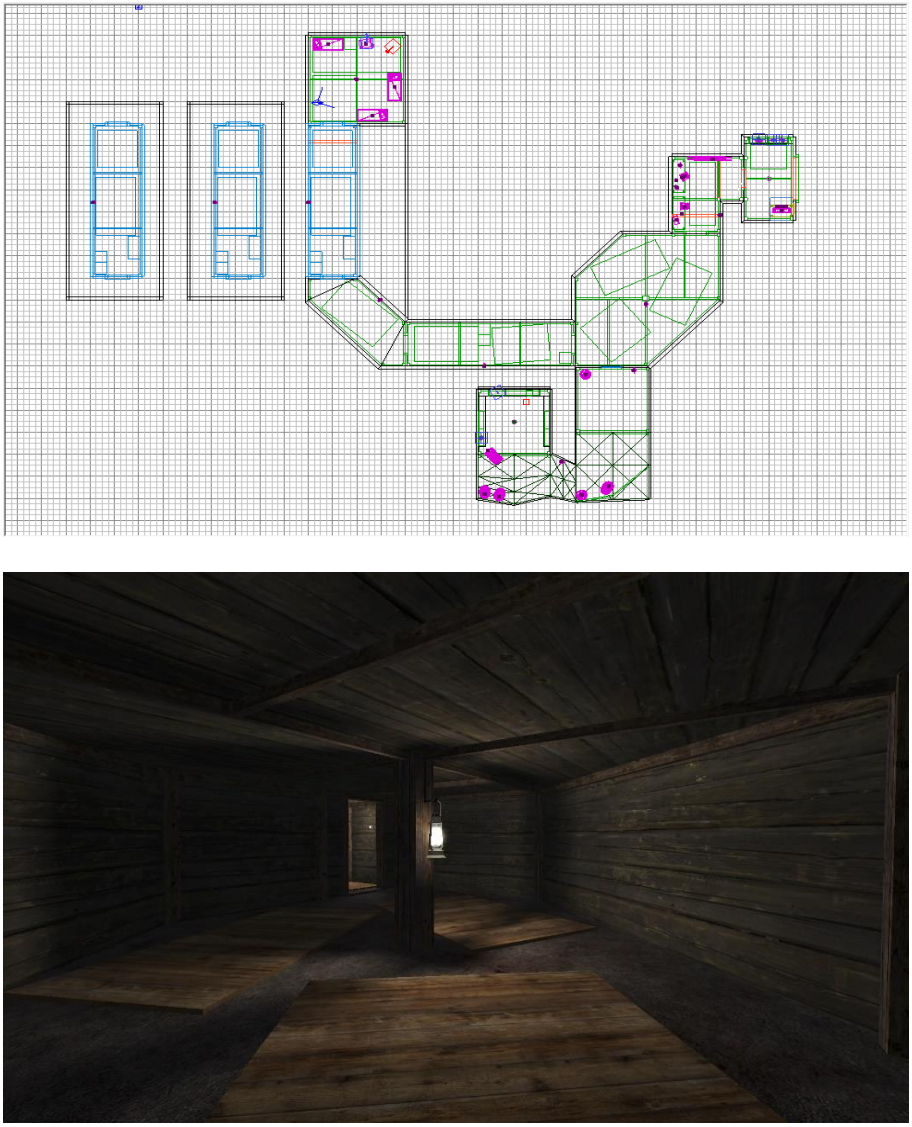
Εικόνα 20 – Τσιμεντένιο πολωνικό οχυρό έξω από την πόλη της Mlawa (1939)

Ως αρχική εικόνα είχαμε τον παίκτη να ξεκινήσει από το εσωτερικό ενός οχυρού και να βγει στο πεδίο της μάχης κατά την έναρξή της. Στο πεδίο της μάχης θα προσπαθήσει να αποκρούσει επιθέσεις των εχθρών, πεζικό και άρματα μάχης. Ήταν δύσκολο να καταλάβουμε από την αρχή πως θα εξελιχθεί η πίστα αφού ό,τι ιδέα είχαμε προς το παρόν, ήταν μόνο βασισμένη στις πληροφορήσεις μας από τις περιγραφές της πραγματικής μάχης.

3.2.2. Υλοποίηση της πίστας

3.2.2.1. Σημείο έναρξης της πίστας / Οχυρό

Ξεκινήσαμε να φτιάχνουμε το εσωτερικό του υπογείου οχυρού. Το οχυρό μας είχε αρκετούς διαδρόμους και δωμάτια ώστε να το κάνουμε να φαίνεται πιο σύνθετο. Γρήγορα όμως καταλάβαμε ότι ο παίκτης πρέπει γρήγορα να βρει την έξοδο ώστε να πάρει μέρος στη μάχη, μιας και αυτός είναι ο κύριος στόχος. Έτσι περιοριστήκαμε σε ένα μόνο διάδρομο που οδηγεί στην έξοδο. Ως καθολικό σχεδιασμό της πίστας λοιπόν επιλέξαμε το *linear pacing*. Φτιάξαμε το χώρο του οχυρού που θα κινηθεί ο παίκτης με όλες τις λεπτομέρειές του, όπως φωτισμό και τρισδιάστατα μοντέλα και τον επικαλύψαμε με τις κατάλληλες υφές.



Εικόνα 21 – Το τελικό σχέδιο του οχυρού σε δισδιάστατο περιβάλλον σχεδίασης και σε μηχανή παιχνιδιού

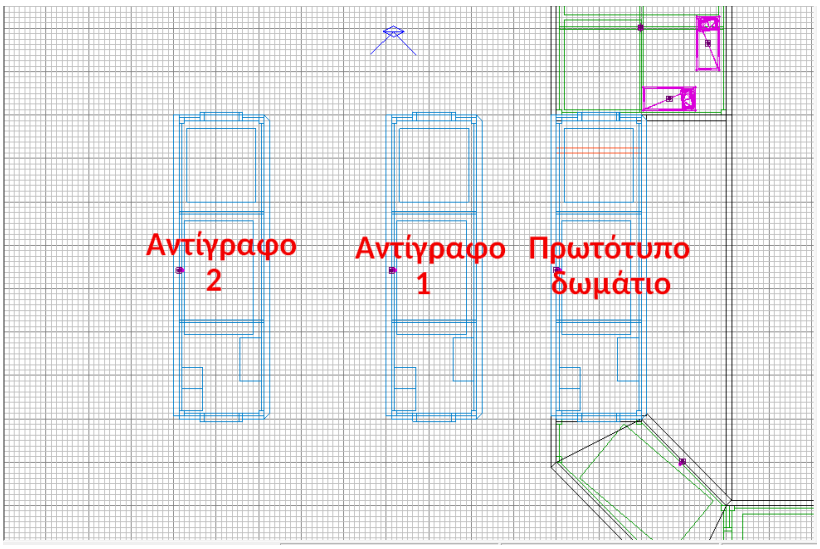
Πριν προχωρήσουμε στο σχεδιασμό του πεδίου της μάχης, ξεκινήσαμε ήδη να δημιουργούμε στοιχεία gameplay στο εσωτερικό του οχυρού. Σύμφωνα με την ιστορία, οι Γερμανοί πριν ξεκινήσουν την επίθεσή τους, χρησιμοποιούσαν ολμοβόλα βομβαρδίζοντας την περιοχή. Όσο λοιπόν ο παίκτης βρίσκεται στο οχυρό, προσθέσαμε **μονάχα** ηχητικά εφέ έκρηξης δίνοντας την αίσθηση ότι γίνονται εκρήξεις εξωτερικά στην οροφή του οχυρού. Για κάθε ηχητικό εφέ έκρηξης προσθέσαμε και ένα οπτικό εφέ "σεισμού" κάνοντας την οθόνη του παίκτη να ταρακουνιέται. Τα εφέ αυτά προστέθηκαν φτιάχνοντας ένα αρχείο *GSC* συνδεδεμένο με την πίστα, στο οποίο συντάξαμε κώδικα σε γλώσσα προγραμματισμού **C**.

Έχοντας πάντα στο μυαλό μας να κρατάμε τον παίκτη σε επαγρύπνηση, σε ένα διάδρομο/δωμάτιο του οχυρού που διαβαίνει, θέλαμε με μία έκρηξη που προκαλεί ένα ολμοβόλο, να αναβοσβήσει το φως. Αυτό αποτέλεσε μία πρόκληση για εμάς αφού η id Tech 3 δεν υποστηρίζει δυναμικό φωτισμό, δηλαδή ο φωτισμός που θα εφαρμόσουμε στην πίστα είναι προ-καταχωρημένος από τον compiler και δεν μπορεί να αλλάξει κατά τη διάρκεια του παιχνιδιού.

Για να πετύχουμε το στόχο μας, φτιάξαμε δύο αντίγραφα δωμάτια με διαφορετική ένταση φωτός το καθένα και τα τοποθετήσαμε δίπλα από το πρωτότυπο σε σημείο που δε μπορεί να δει ο παίκτης. Τα προγραμματίσαμε έτσι ώστε όταν ο παίκτης εισέρχεται στο πρωτότυπο δωμάτιο, ακούγεται ένα ηχητικό εφέ έκρηξης και τα τρία δωμάτια εναλλάσσονται, παίρνοντας το ένα τη θέση του άλλου με πολύ γρήγορη ταχύτητα χωρίς να μπορεί να το αντιληφθεί ο παίκτης.

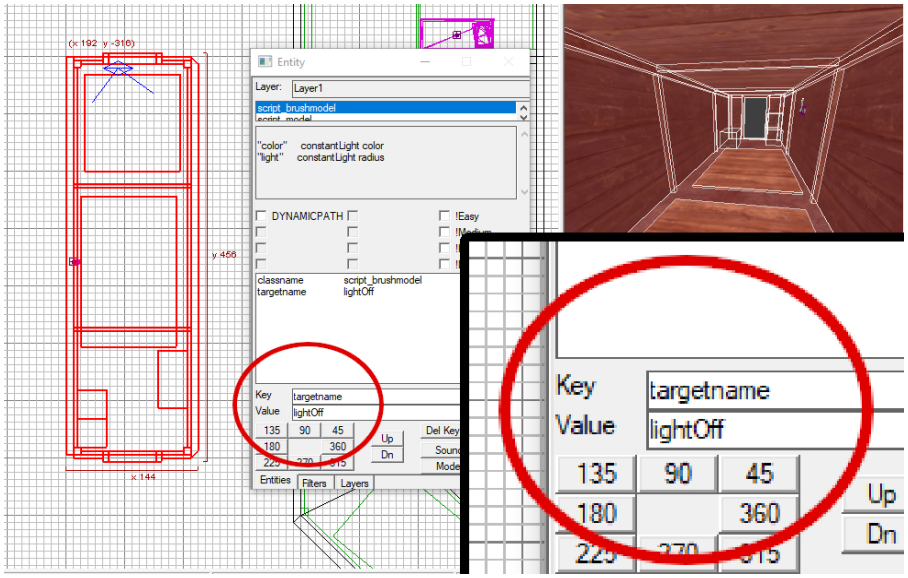
Συγκεκριμένα, ακολουθήσαμε την εξής διαδικασία:

1. Τα brushes που φτιάχνουμε στον κόσμο έχουν τη μορφή "*worldspawn*" που σημαίνει ότι είναι σταθερά σχήματα που δεν μπορούν να επηρεαστούν από κανένα παράγοντα και δεν μπορούν να προγραμματιστούν. Επιλέξαμε όλα τα brushes του δωματίου, του οποίου θέλουμε να αλλάξουμε τον φωτισμό, και τα μετατρέψαμε σε *brushmodels*. Το ίδιο επαναλάβαμε με το μοντέλο της λάμπας μετατρέποντάς το σε *script_model*. Πλέον μπορούμε να προγραμματίσουμε τα *brushmodels* και *script_models* μέσω του αρχείου *GSC*.
2. Δημιουργήσαμε δύο αντίγραφα του δωματίου. Στο πρώτο αντίγραφο αφαιρέσαμε την πηγή φωτός, ενώ στο δεύτερο ελαττώσαμε την ένταση στο μισό.



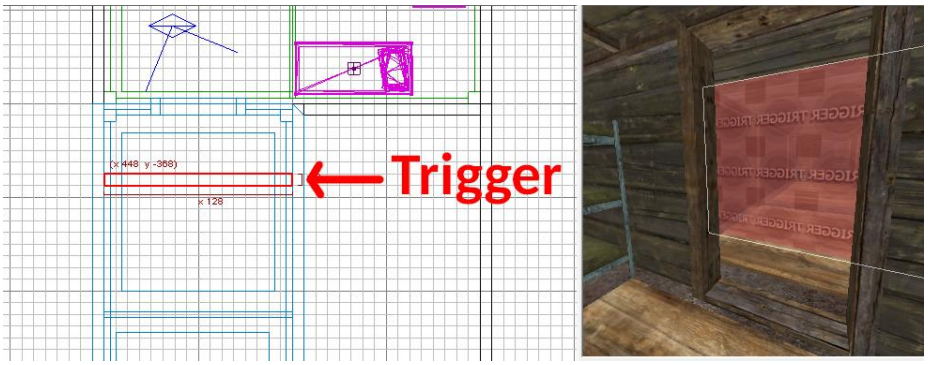
Εικόνα 22

3. Στη συνέχεια, για να προγραμματίσουμε τα στοιχεία των δωματίων, πρέπει να τους δώσουμε όνομα. Για να επιτευχθεί αυτό, επιλέξαμε όλα τα γεωμετρικά σχήματα που αποτελείται το δωμάτιο και τους δώσαμε το ίδιο όνομα (*targetname*). Τα ονόματα που δώσαμε στο πρωτότυπο δωμάτιο και τα δύο αντίγραφα είναι "lightOn", "lightOff" και "lightLow". Αντίστοιχα, ονομάσαμε τα script_model, δηλαδή τα μοντέλα της λάμπας ως "lightModelOn", "lightModelOff" και "lightModelLow".



Εικόνα 23 – Μενού επεξεργασίας οντότητας στο Radiant

4. Φτιάξαμε ένα γεωμετρικό σχήμα που καλύπτει την είσοδο του δωματίου, στο οποίο δώσαμε την υφή "trigger", το μετατρέψαμε σε τύπο "*trigger_multiple*" και το ονομάσαμε "lightsOut". Τοποθετήσαμε το “trigger” ώστε να αλλάξει ο φωτισμός μόνο όταν ο παίκτης περάσει μέσα από αυτό.



Εικόνα 24

5. Τέλος, χρησιμοποιώντας κώδικα, προγραμματίσαμε τα προαναφερόμενα στοιχεία αναλόγως ώστε το ένα δωμάτιο να αποκτά τη θέση του άλλου. Ακολουθούν εικόνες με τον κώδικα που συντάξαμε.

```
light_room()
{
    lighton = getent("lighton", "targetname");
    lightoff = getent("lightoff", "targetname");
    lightlow = getent("lightlow", "targetname");
    on = getent("lightmodelon", "targetname");
    off = getent("lightmodeloff", "targetname");
    low = getent("lightmodellow", "targetname");
    on setmodel("xmodel/light_cagelight_white_on");
    off setmodel("xmodel/light_cagelight_white_off");
    low setmodel("xmodel/light_cagelight_white_off");
    trig = getent("lightsout", "targetname");

    trig waittill("trigger");

    thread mortarblast_lightsout((520, -768, 128));
    wait(0.25);
```

Αρχικά δηλώνουμε τα targetname που ορίσαμε προηγουμένως στο Radiant.

Η συνάρτηση περιμένει τον παίκτη να περάσει μέσα από το *trigger* που έχει τοποθετηθεί στην είσοδο του δωματίου.

Καλείται άλλη συνάρτηση ώστε να ακουστεί το ηχητικό εφέ της έκρηξης στις συντεταγμένες x,y,z που έχουμε ορίσει.

```
//OFF
    lighton moveX(-248, .0001);
    lightoff moveX(248, .0001);
    on moveX(-248, .0001);
    off moveX(248, .0001);
    wait(.55);
//LOW
    lightlow moveX(568, .0001);
    lightoff moveX(-248, .0001);
    low moveX(568, .0001);
    off moveX(-248, .0001);
    wait(.4);
//ON
    lighton moveX(248, .0001);
    lightlow moveX(-568, .0001);
    on moveX(248, .0001);
    low moveX(-568, .0001);
    wait(.25);
//LOW
    lighton moveX(-248, .0001);
    lightlow moveX(568, .0001);
    on moveX(-248, .0001);
    low moveX(568, .0001);
    wait(.15);
//ON
    lighton moveX(248, .0001);
    lightlow moveX(-568, .0001);
    on moveX(248, .0001);
    low moveX(-568, .0001);

    wait(1);
    thread mortars_while_in_bunker();
}
```

Ακολουθεί η εναλλαγή των δωματίων.

> Το δωμάτιο (Αντίγραφο 1) με το σβηστό φως παίρνει τη θέση του πρωτότυπου.

> Το δωμάτιο (Αντίγραφο 2) με το χαμηλό φως παίρνει τη θέση του Αντίγραφου 1.

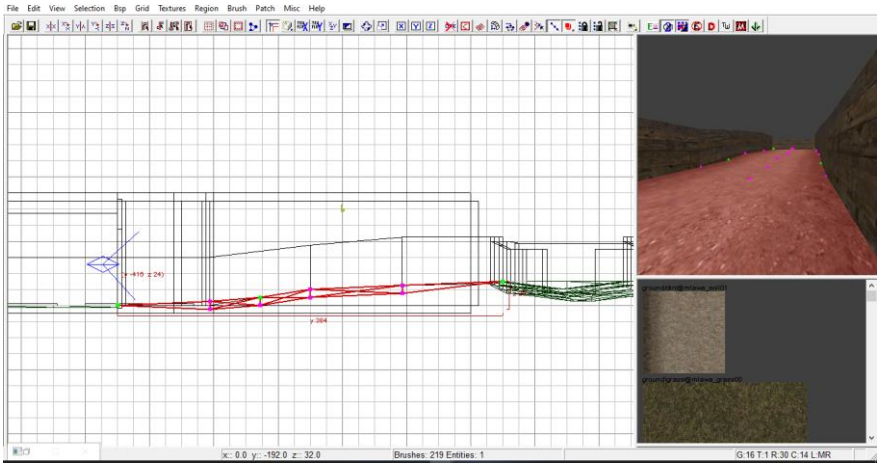
> Το πρωτότυπο δωμάτιο παίρνει τη θέση του Αντίγραφου 2, κ.ο.κ.

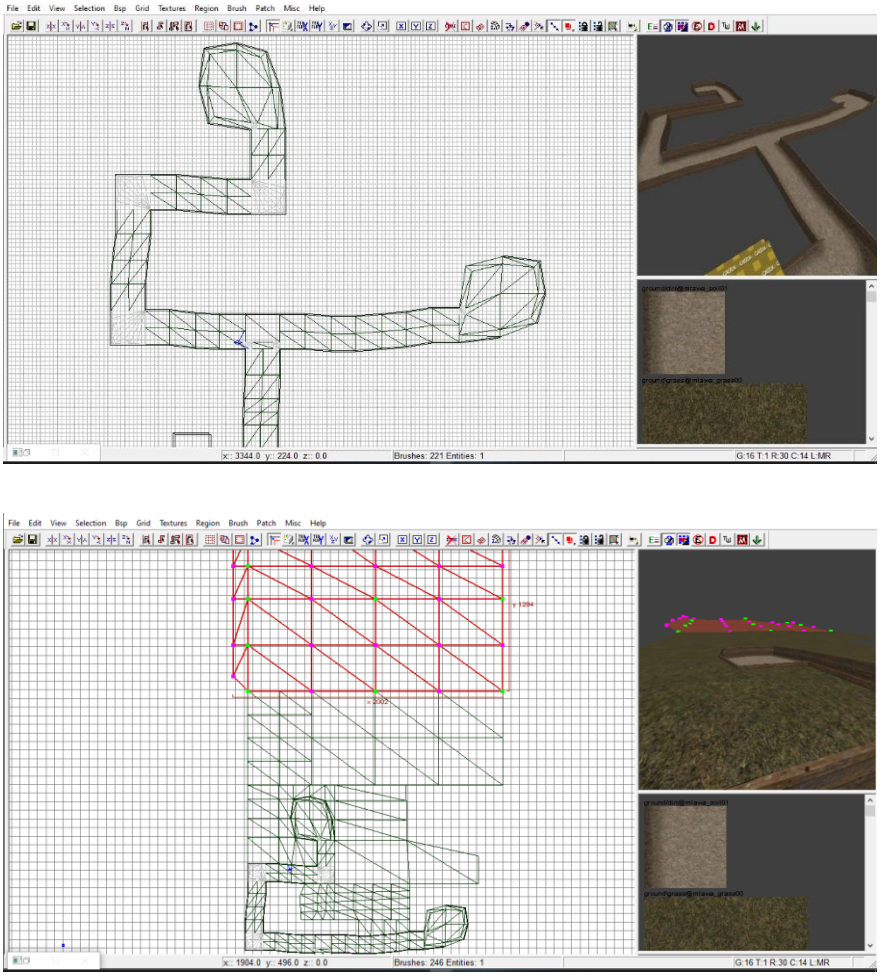
Πριν κλείσουμε τη συνάρτηση, καλούμε τη συνάρτηση *mortars_while_in_bunker()* με την οποία δημιουργούμε μελλοντικά ηχητικά εφέ έκρηξης όσο ο παίκτης παραμένει στο εσωτερικό του οχυρού.

Αξίζει να αναφέρουμε ότι η εντολή **moveX(*units, time*)** σημαίνει ότι το brushmodel που έχουμε ορίσει κινείται στον άξονα X στον αριθμό μονάδων (units) που έχουμε ορίσει και σε χρόνο (time) 0.0001 ώστε η κίνηση να γίνει τόσο γρήγορα ώστε ο παίκτης να μην μπορεί να την αντιληφθεί.

3.2.2.2. Πρώτο πεδίο μάχης

Στη συνέχεια, δημιουργήσαμε τα χαρακώματα που θα βρίσκονται οι πολωνοί στρατιώτες καθώς και τη γύρω περιοχή. Χρησιμοποιώντας *patch meshes*, κάναμε τα χαρακώματα ανισόπεδα ώστε να αυξήσουμε τον ρεαλισμό του κόσμου και, βασισμένοι στην πραγματική μάχη, κάναμε το έδαφος ανηφορικό αφού οι εχθροί θα επιτίθενται από την κορυφή του.





Εικόνα 25 – Σχεδιασμός ανισόπεδου εδάφους στο Radiant

Εικόνα 26 – Αρχικός σχεδιασμός του πρώτου πεδίου μάχης

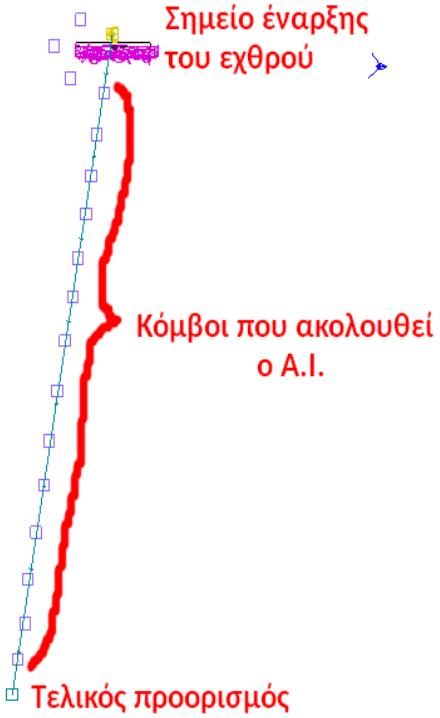
Το χρώμα που υπερισχύει στο έδαφος είναι το πράσινο. Θέλαμε να "τσακίσουμε" αυτή τη μονοτονία σχεδιάζοντας ένα ποτάμι που χωρίζει το έδαφος και μία γέφυρα που ενοποιεί τις δύο ακροποταμιές. Αυτή η προσθήκη μας έδωσε και νέες ιδέες για gameplay.

Εικόνα 27

Οι υφές που χρησιμοποιήσαμε αρχικά ήταν ζωηρές και "ζωντανές", κάτι που έπρεπε τροποποιήσουμε αφού θέλαμε να κάνουμε το περιβάλλον πιο άφωτο και καταθλιπτικό. Αλλάξαμε το χρώμα του εδάφους όσο και του ουρανού σε υφές που επεξεργαστήκαμε στο πρόγραμμα επεξεργασίας εικόνων, *GIMP*.

Αφού ολοκληρώσαμε τη πρωταρχική γεωμετρία, ξεκινήσαμε να προσθέτουμε περισσότερα τρισδιάστατα μοντέλα, όπως θάμνους και δέντρα γύρω από την περιοχή που κινείται ο παίκτης.

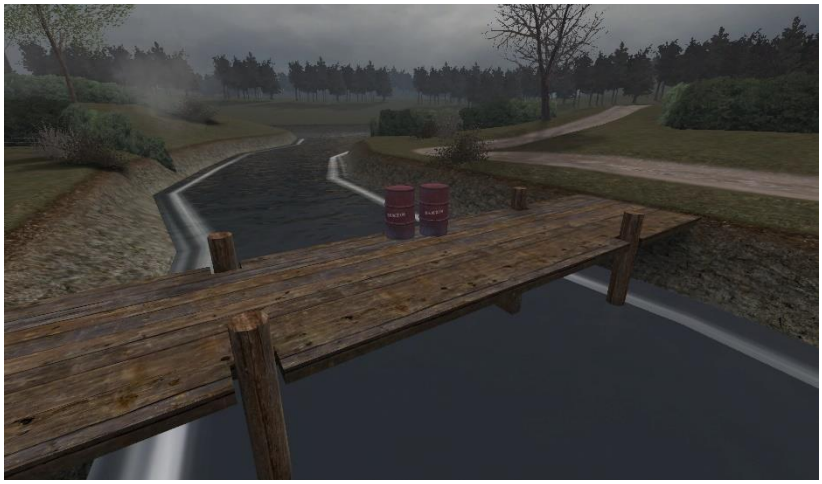
Όταν μείναμε ικανοποιημένοι με το τμήμα της γεωμετρίας, ξεκινήσαμε να προσαρμόζουμε gameplay. Αρχικά, προσθέσαμε μερικά "κύματα" εχθρών που εμφανίζονται από την κορυφή του λόφου αλλά και από την έναντι όχθη. Για κάθε αντίπαλο αλλά και σύμμαχο A.I. πρέπει να ορίσουμε το σημείο από το οποίο θα εμφανίζεται αλλά και τη διαδρομή που θα ακολουθήσει, τοποθετώντας κόμβους (pathnodes) ώστε να φτιάξουμε μονοπάτια.



Εικόνα 28 – Παράδειγμα κίνησης Α.Ι.

Μία δεύτερη πρόκληση που αντιμετωπίσαμε έχει να κάνει με το κομμάτι της γέφυρας που ενώνει τις δύο πλευρές του εδάφους. Στη γέφυρα τοποθετήσαμε βαρέλια και τα προγραμματίσαμε ώστε όταν ο παίκτης τα πυροβολήσει, αυτά "εκρήγνυνται" και η γέφυρα κόβεται στα δύο. Θέλαμε από την καταστροφή της γέφυρας να επωφεληθεί ο παίκτης. Τοποθετήσαμε τρεις εχθρούς οι οποίοι κάνουν την απόπειρα να διασχίσουν τη γέφυρα και να πλησιάσουν τον παίκτη. Αν η γέφυρα εκραγεί τη στιγμή που οι εχθροί τη διασχίζουν, η έκρηξη θα τους σκοτώσει.

Για να το πετύχουμε αυτό, φτιάξαμε **δύο** γέφυρες, μία ολοκληρωμένη και μία κατεστραμμένη, τις τοποθετήσαμε τη μία μέσα στην άλλη και τις μετατρέψαμε σε brushmodels ώστε να μπορούμε να τις προγραμματίσουμε. Με κατάλληλο προγραμματισμό, πριν τα βαρέλια εκραγούν, κρύβουμε την κατεστραμμένη έκδοση της γέφυρας. Όταν ο παίκτης τα πυροβολήσει, εμφανίζεται για δύο δευτερόλεπτα ένα εφέ φωτιάς πάνω τους, ακολουθεί ηχητικό εφέ έκρηξης και ταυτόχρονα φανερώνουμε την κατεστραμμένη έκδοση της γέφυρας και εξαφανίζουμε την αρχική.



Εικόνα 29 – Διαδικασία έκρηξης της γέφυρας

Η πραγματική όμως δυσκολία εμφανίστηκε στην περίπτωση που η γέφυρα έχει καταστραφεί πριν ξεκινήσουν οι εχθροί να τη διαβούν. Όπως είπαμε, για κάθε A.I. πρέπει να ορίσουμε το μονοπάτι που θα ακολουθήσει καθώς και τον προορισμό του. Ως τελικό προορισμό είχαμε ορίσει ένα σημείο στην απέναντι όχθη. Το πρόβλημα ήταν ότι, αν η γέφυρα είχε ήδη γκρεμιστεί πριν αυτοί παρουσιαστούν, οι εχθροί την παράβλεπαν και διέσχιζαν το ποτάμι με αποτέλεσμα να χάνεται ο ρεαλισμός. Με κατάλληλο προγραμματισμό ο προορισμός των τριών εχθρών να αλλάζει βάσει του αν έχει καταστραφεί η γέφυρα.

Συγκεκριμένα ακολουθήσαμε την εξής διαδικασία.

1. Αρχικά, στην *main()* δηλώνουμε δύο flag που μας υποδεικνύουν αν η γέφυρα έχει καταστραφεί ή όχι καθώς και για το αν οι τρεις εχθροί έχουν ήδη διασχίσει τη γέφυρα και έχουν φτάσει την απέναντι ακροποταμιά.

   level.bridge_flag = true;

   level.other_side = false;

2. Φτιάχνουμε τη συνάρτηση *bridge_break()* η οποία περιλαμβάνει τον κώδικα για την εναλλαγή της αδιαίρετης/κατεστραμμένης γέφυρας καθώς και τα κατάλληλα εφέ ανατίναξης. Αν η γέφυρα καταστραφεί, το flag *level.bridge_flag* γίνεται **FALSE**.

```
bridge_break()
{
    trig = getent("benzin_boom", "targetname");
    explo = getent("explo_sound", "targetname");
    explo2 = getent("explo_sound2", "targetname");
    benzin1 = getent("benzin", "targetname");
    benzin2 = getent("benzin2", "targetname");
    benzin1 setmodel("xmodel/barrel_benzin");
    benzin2 setmodel("xmodel/barrel_benzin");
    bridge = getent("bridge", "targetname");
    bridge_d = getent("bridge_d", "targetname");

    bridge_d hide();

    trig waittill("trigger");

    for(i=1;i<2;i++)
    {
        playfx (level._effect["fire1"], ((3616, 500, 75)));
        wait(1);
    }
    level.bridge_flag = false;
    explo playsound ("explo_metal_rand");
    playfx (level._effect["explosion1"], ((3616, 500, 60)));
    playfx (level._effect["explosion2"], ((3616, 500, 60)));
    earthquake(0.35, 3, ((3380, 448, 50)), 3000);
    bridge_d show();
    bridge delete();
    benzin1 delete();
    benzin2 delete();
    maps\mp\_fx::loopfx("smoke1", (3600, 412, -24), 0.6);
    radiusDamage((3616, 448, 40), 280, 400, 250);
}
```

3. Φτιάχνουμε δύο νέες συναρτήσεις. Η πρώτη έχει το όνομα *bridge_guys_alt_goal()*. Ο παρακάτω κώδικας μας εξηγεί ότι αν η γέφυρα έχει καταστραφεί πριν παρουσιαστούν οι τρεις εχθροί, δεν επιδιώκουν να τη διασχίσουν και παίρνουν θέση στην όχθη που βρίσκονται.

```
bridge_guys_alt_goal()
{
    alt = getnode("altnode","targetname");
    thread bridge_guys_def_goal();
    thread randomspawn(30,"bridge_guys","script_noteworthy");
    wait(.2);
    bridge_guys = getentarray("bridge_guys", "groupname");
    if(level.bridge_flag == false)
    {
        for(i=0;i<bridge_guys.size;i++)
        {
            if (isSentient(bridge_guys[i]))
            {
                bridge_guys[i] setgoalpos (alt.origin);
                bridge_guys[i].goalradius = 16;
            }
        }
    }
```

Αν όμως η γέφυρα καταστρέφεται όταν οι εχθροί έχουν ήδη εμφανιστεί και έχουν πλησιάσει σε αυτή αλλά η έκρηξη **δεν** τους σκοτώσει, τότε επανέρχονται πίσω στην όχθη τους.

```
else
{
    while(level.other_side == false)
    {
        if((level.bridge_flag == false) && (level.other_side == false))
        {
            wait(1.25);
            if(level.other_side == false)
            {
                for(i=0;i<bridge_guys.size;i++)
                {
                    if (isSentient(bridge_guys[i]))
                    {
                        bridge_guys[i] setgoalpos (alt.origin);
                        bridge_guys[i].goalradius = 16;
                    }
                }
            }
        }
        wait(.2);
    }
}
```

4. Η δεύτερη συνάρτηση έχει το όνομα *bridge_guys_def_goal()*, και ελέγχει μέσω ενός trigger που έχουμε τοποθετήσει στην απέναντι ακροποταμιά αν οι εχθροί είναι ζωντανοί και έχουν διασχίσει τη γέφυρα. Αν αυτό ισχύει, το flag *level.other_side* γίνεται **TRUE**, και οι εχθροί αποκτούν τη θέση που έχουμε προσδιορίσει στην απέναντι όχθη.

```
bridge_guys_def_goal()
{
    def = getnode("def_node","targetname");
    trig = getent("other_side_trig","targetname");
    trig waittill ("trigger");
    level.other_side = true;
    bridge_guys = getentarray("bridge_guys", "groupname");
    for(i=0;i<bridge_guys.size;i++)
    {
        if (isSentient(bridge_guys[i]))
        {
            bridge_guys[i] setgoalpos (def.origin);
            bridge_guys[i].goalradius = 16;
        }
    }
}
```

Ενδιάμεσα στα “κύματα” εχθρών, συντάσσοντας κώδικα εμπλουτίσαμε την εμπειρία του παίκτη με σκοπό να διαφυλάξουμε το ενδιαφέρον του. Προσθέσαμε αεροπλάνα stuka να διασχίζουν τον ουρανό, εκρήξεις που δημιουργούν οπές στο έδαφος συνθέτοντας νέα μονοπάτια για τον παίκτη καθώς και pick-ups όπως ζωή και χειροβομβίδες που μπορεί να σηκώσει από το έδαφος. Όπως βλέπουμε στην παρακάτω εικόνα, μία έκρηξη σπάει ένα κομμάτι από τα ξύλινα χαρακώματα δίνοντας το δικαίωμα στον παίκτη να αποκτήσει πρόσβαση στον υπόλοιπο χώρο.

Εικόνα 30 – Στιγμιότυπο από τη μηχανή παιχνιδιού

Αφού ο παίκτης αποκρούσει ορισμένες επιθέσεις εχθρών, εμφανίζονται τρία εχθρικά τανκ που καλείται να καταστρέψει με τη χρήση ενός αντιαρματικού όπλου. Τα τανκ εμφανίζονται ένα προς ένα κάθε 30 δευτερόλεπτα. Όσο ο παίκτης αργοπορεί να εξοντώσει ένα τανκ, τα πράγματα γίνονται όλο και πιο ζόρικα γι' αυτόν.

Μετά την καταστροφή των τριών τανκ, η πρώτη μάχη φτάνει στο τέλος της.

Εικόνα 31 – Στιγμιότυπο από τον τελικό σχεδιασμό της πρώτης μάχης

3.2.2.3. Μονοπάτι

Αφού ολοκληρωθεί η πρώτη μάχη, φτιάξαμε ένα μακρύ μονοπάτι που θα διασχίσει ο παίκτης για να φτάσει στο δεύτερο πεδίο μάχης. Το μονοπάτι αποτελεί ειρηνική ζώνη αφού ο παίκτης δεν κινδυνεύει από κανένα παράγοντα. Ένας σημαντικός λόγος που φτιάξαμε το μονοπάτι μεταξύ των δύο πεδίων είναι ώστε να επιτευχθεί η καλύτερη απόδοση της πίστας κάτι που θα εξετάσουμε περισσότερο στη συνέχεια.

Στο μονοπάτι προσθέσαμε διάφορες λεπτομέρειες. Αρχικά, όταν ο παίκτης ξεκινά την πορεία του, πραγματώσαμε ένα εφέ βροχής στην οθόνη.

Στη μέση της διαδρομής, σχεδιάσαμε ένα μικρό σπίτι που μπορεί να εξερευνηθεί. Αναλυτικά, δημιουργήσαμε ένα απλό παζλ στο εσωτερικό του σπιτιού που ο παίκτης έχει την επιλογή να λύσει ώστε να αποκτήσει ένα νέο όπλο, χειροβομβίδες και να ανακτήσει ζωή.

Εικόνα 32 – Καταληκτικός σχεδιασμός του μονοπατιού

Έξω από το μονοπάτι σχεδιάσαμε έναν ανεμόμυλο βασισμένο σε αληθινό σχέδιο πολωνικού ανεμόμυλου της εποχής.

Εικόνα 33 –

Αριστερά: Αναπαράσταση στη μηχανή παιχνιδιού

Δεξιά: Πραγματική φωτογραφία του ανεμόμυλου

Με τις παραπάνω λεπτομέρειες, σκοπεύσαμε να ξεχαστεί για λίγο η σκληρότητα του πολέμου προβάλλοντας ένα διαφορετικό σκηνικό.

3.2.2.4. Δεύτερο πεδίο μάχης

Στο δεύτερο πεδίο μάχης, όπως και στο πρώτο, ο παίκτης τίθεται να αντικρούσει την επίθεση των εχθρών. Θα πρέπει όμως να διατηρήσουμε το ενδιαφέρον του παίκτη φτιάχνοντας κάτι ξεχωριστό. Ξεκινήσαμε να φτιάχνουμε τη γεωμετρία της νέας περιοχής. Χρησιμοποιώντας πάλι *patch meshes*, κάναμε αυτή τη φορά το έδαφος κατηφορικό, σε αντίθεση με πριν. Στην κορυφή της φτιάξαμε ένα τσιμεντένιο οχύρωμα βασισμένο στο πραγματικό σχέδιο που είχαν τα πολωνικά οχυρά της εποχής όπως φαίνεται στην παρακάτω εικόνα.

Εικόνα 34 – Σύγκριση γνήσιου οχυρού με αναπαράσταση στη μηχανή παιχνιδιού

Πιο χαμηλά σχεδιάσαμε χαρακώματα, αυτή τη φορά με αλλιώτικο σχέδιο και κλίση από προηγουμένως. Τα χαρακώματα περικλείουν το οχυρό δίνοντας τη δυνατότητα του παίκτη για κάλυψη.

Στο πιο χαμηλό μέρος της περιοχής, τοποθετήσαμε χαμόδεντρα από τα οποία θα εμφανίζονται οι εχθροί που θα τρέχουν προς το οχύρωμα και τανκ που ο παίκτης πρέπει να εξοντώσει με διαφορετικό, αυτή τη φορά, τρόπο από πριν.

Εικόνα 35 – Τα κυκλικά χαρακώματα

Στη συνέχεια, φτιάξαμε δύο μικρούς λόφους πίσω από τους οποίους τοποθετήσαμε *spawners* που θα εμφανίζονται σύμμαχοι A.I. κατά τη διάρκεια της μάχης. Όταν ένας σύμμαχος πέσει, ο *spawner* θα "γεννάει" ένα νέο A.I. που θα τρέχει στο μέτωπο. Με αυτό τον τρόπο, όση ώρα και να περάσει κατά τη διάρκεια της μάχης, ο παίκτης δε θα μείνει μόνος του να πολεμάει αν όλοι του οι σύμμαχοι πέσουν. Γενικώς, οι *spawners* δεν μπορούν να τοποθετηθούν σε σημεία που μπορεί να δει ο παίκτης αφού η μηχανή αποτρέπει νέα A.I να “γεννηθούν”. Γι' αυτό και τους τοποθετήσαμε στο πίσω μέρος των λόφων, που ο παίκτης δεν έχει ορατότητα. Να σημειωθεί επιπλέον ότι ο παίκτης δεν έχει πρόσβαση την κορυφή των λόφων αυτών αφού σχεδιάστηκαν μόνο ως *spawners* αλλά και για να εμπλουτίσουν τον κόσμο.

Εικόνα 36 – Ένας από τους δύο λόφους της δεύτερης περιοχής

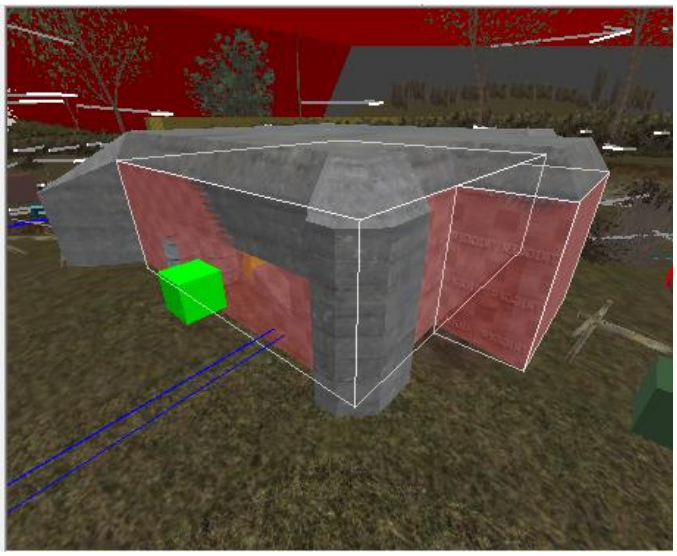
Αφού ολοκληρώσαμε τη γεωμετρία της δεύτερης περιοχής, επεκταθήκαμε στο gameplay.

Πριν την έναρξη της μάχης, οι εχθροί βομβαρδίζουν την περιοχή γύρω από το οχύρωμα και ο παίκτης καλείται να παραμείνει στο εσωτερικό του. Για να το κατορθώσομε αυτό, τοποθετήσαμε σε διάφορες θέσεις του χώρου μία σειρά από οντότητες που μετατρέψαμε σε *script_origin* και τις δώσαμε το όνομα *bunker_mortar*.

```
bunker_mortars()
{
    mortar = getentarray("bunker_mortar","targetname");
    for(y=0;y<3;y++)
    {
        for(i=0;i<mortar.size;i++)
        {
            origin = mortar[i] getorigin();
            mortar[i] playsound ("shell_incoming1");
            wait randomfloat(0.6);
            mortar[i] playsound ("shell_explosion1");
            playfx(level.mortar, origin);
            mortar[i] playsound("mortar_explosion2");
            earthquake(0.35, 3.25, origin, 1150);
            radiusDamage(origin, 150, 75, 75);
            wait randomfloat(1);
        }
    }
}
```

Όπως παρατηρούμε στην συνάρτηση *bunker_mortars*, αρχικά δηλώνουμε τα script_origin σε πίνακα και στη συνέχεια στη θέση του κάθε ενός εφαρμόζουμε οπτικά και ηχητικά εφέ έκρηξης σε τυχαίο χρόνο. Η έκρηξη σε κάθε θέση επαναλαμβάνεται τρεις φορές.

Αρχικά, καλύψαμε το οχυρό με ένα γεωμετρικό σχήμα που του παραχωρήσαμε την υφή *trigger* και το ονοματίσαμε *safe_zone*, όπως φαίνεται παρακάτω.



Εικόνα 37

```
safe_zone()
{
    zone = getent ("safe_zone","targetname");
    zone_2 = getent ("safe_zone_2","targetname");

    while ((level.player IsTouching(zone) == false)
            && (level.player IsTouching(zone_2) == false)
            && (level.mortar_rain == true))
    {
        level.player playsound ("shell_incoming1");
        wait(.2);
        level.player playsound ("shell_explosion1");
        playfx(level.mortar, level.player.origin);
        level.player playsound("mortar_explosion2");
        earthquake(0.35, 3.25, level.player.origin, 1150);
        radiusDamage(level.player.origin, 50, 100, 100);
        wait(3.5);
    }
    wait(.2);
    thread safe_zone();
}
```

Με τη συνάρτηση *safe_zone*, δηλώνουμε το *trigger* που φτιάξαμε στο Radiant. Με την εντολή **while**, ελέγχουμε αν ο παίκτης αγγίζει το *trigger*. Αν δεν το αγγίζει, τότε κάθε 3,5 δευτερόλεπτα στη θέση του παίκτη εφαρμόσαμε εφέ έκρηξης, ίδιο με το εφέ της προηγούμενης συνάρτησης, το οποίο κάνει ζημιά στον παίκτη και του αφαιρεί ένα μέρος της ζωής. Με τον τρόπο αυτό, αν ο παίκτης βρίσκεται εκτός του οχυρού, ακόμα και σε θέση που δεν έχουμε ορίσει να γίνεται έκρηξη, δέχεται ζημιά που του υποδεικνύει να μπει στο οχύρωμα.

Εικόνα 38 – Το εσωτερικό του οχυρού

Στη συνέχεια, δημιουργήσαμε πάλι μερικά "κύματα" εχθρών και ενδιάμεσά τους δύο τανκ που ο παίκτης πρέπει να καταστρέψει. Στην περίπτωση αυτή, το αντιαρματικό όπλο είναι ανώφελο απέναντι στα τανκ και ο παίκτης τίθεται να προσεγγίσει το κάθε ένα τοποθετώντας του δυναμίτη. Αλλάζει έτσι ο τρόπος που ο παίκτης θα χρησιμοποιήσει τον κόσμο που δημιουργήσαμε αφού πρέπει να προσαρμοστεί σε αυτόν αναλόγως.

Εικόνα 39 – Τανκ που έχει καεί από τον παίκτη κατά τη διάρκεια της δεύτερης μάχης

3.2.2.5. Feedback και αλλαγές

Αφού ολοκληρώσαμε τη γεωμετρία και το gameplay σε όλη την έκταση της πίστας, μοιράσαμε την πίστα σε τρεις χρήστες της κοινότητας του παιχνιδιού για **feedback**. Όλοι εξέφρασαν τον ενθουσιασμό τους για τον κόσμο που δημιουργήσαμε αλλά ανέφεραν δυσκολία να καταλάβουν το δρόμο που πρέπει να ακολουθήσουν, κατόπιν της πρώτης μάχης καθώς και ότι η πίστα ολοκληρώνεται απρόοπτα χωρίς κάποιο βροντερό περιστατικό. Επίσης, παρατήρησαν ότι ο παίκτης έχει το ρόλο ενός απλού στρατιώτη και δεν μπορεί να κινείται μόνος του στο χώρο και να παίρνει δικές του αποφάσεις. Μας συμβούλεψαν ότι χρειάζεται ένας στρατηγός A.I. που θα δίνει εντολές στον παίκτη καθ' όλη τη διάρκεια της πίστας.

Τοποθετήσαμε ένα A.I. που βρίσκεται στο πλευρό του παίκτη από την αρχή κιόλας της πίστας και τον διατάζει να τον ακολουθήσει. Τον προγραμματίσαμε ώστε αν ο παίκτης απομακρυνθεί από αυτόν, να στέκεται και να τον καλεί. Με αυτόν τον τρόπο όχι μόνο παρείχαμε μεγαλύτερο ρεαλισμό στην πίστα, αλλά ο παίκτης δεν μπορεί πλέον να χαθεί στον κόσμο.

Εικόνα 40 – Ο στρατηγός που διατάζει τον παίκτη

```
if(distance(sgt.origin, level.player.origin)>400)
{
    sgt setgoalpos (sgt.origin);
    sgt.goalradius = 32;
}
else if(distance(sgt.origin, level.player.origin)<400)
{
    sgt setgoalpos (node.origin);
    sgt.goalradius = 32;
}
```

Ακόμα, η προσθήκη του στρατηγού μας έδωσε και ιδέα για το πώς θα ολοκληρώσουμε την πίστα με πιο εκρηκτικό τρόπο. Συγκεκριμένα, μετά τη δεύτερη μάχη, ο στρατηγός σκοτώνεται και ο παίκτης πρέπει μόνος του να υπερασπιστεί το οχυρό έως ότου φτάσουν ενισχύσεις. Οι ενισχύσεις εξολοθρεύουν όλους του εχθρούς και η πίστα λήγει.

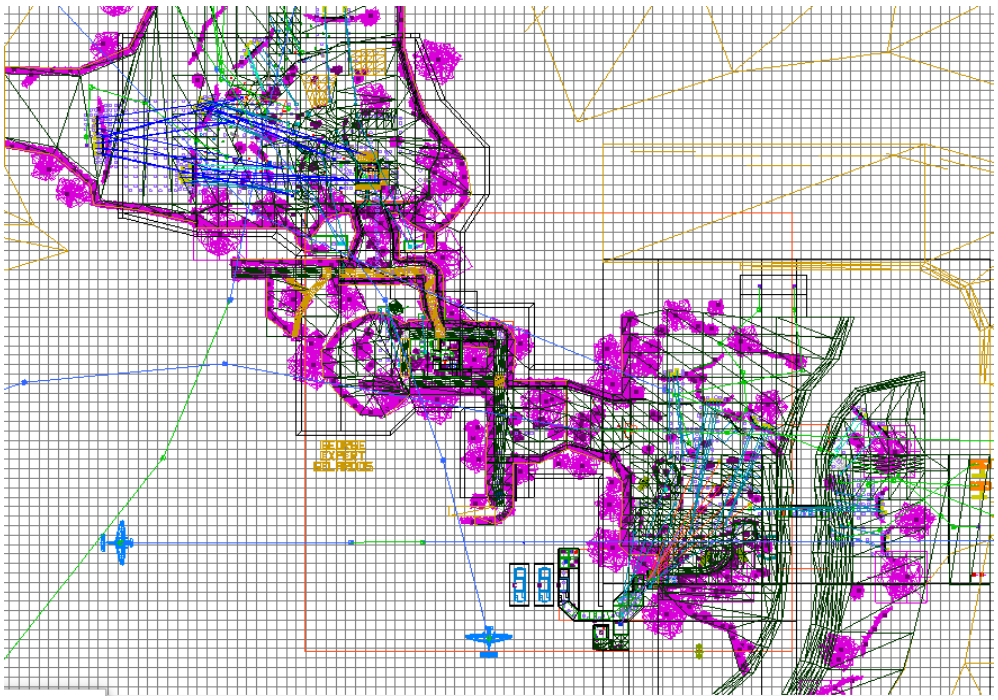
Από την αρχή της πίστας έχουμε δώσει στον στρατηγό απεριόριστη ζωή μέσω του flag *level.sgt_is_god = true* και ορίζοντας τη ζωή του σε “90.000”. Όσο το flag είναι **TRUE**, αν η ζωή του στρατηγού λιγοστέψει κάτω από “90.000” πόντους, επανέρχεται στο ίδιο νούμερο.

```
sgt_is_god()
{
    level.sarge = getent("sgt","script_noteworthy");
    while(level.sgt_is_god == true)
    {
        if(level.sarge.health != 90000)
        {
            level.sarge.health = 90000;
        }
        wait(1);
    }
}
```

Για να κάνουμε τον στρατηγό να ξεψυχήσει τη στιγμή που επιθυμούμε, τοποθετήσαμε μία οντότητα *script_origin* στο βάθος της δεύτερης περιοχής, της δώσαμε το όνομα *sfx* και την προγραμματίσαμε ώστε να ηχήσει ένα ηχητικό εφέ πυρός. Ταυτόχρονα, μεταβάλλομε το flag *level.sgt_is_god* σε **FALSE** και ελαττώνουμε τη ζωή του στρατηγού στο μηδέν, κάνοντάς του ζημιά πάνω από "90.000" πόντους, όπως φαίνεται στον παρακάτω κώδικα.

```
level.sgt_is_god = false;
wait(4);
sfx playsound("kar98_sfx");
wait(.15);
sgt DoDamage( 1000000, (0,0,0) );
```

Ύστερα από πολλούς πειραματισμούς και διορθώσεις τόσο στο κομμάτι του σχεδιασμού του κόσμου αλλά και στο κομμάτι του κώδικα, ολοκληρώσαμε την πίστα σε γεωμετρία και gameplay και περάσαμε στο κομμάτι της μουσικής.



Εικόνα 41 – Καταληχτικός σχεδιασμός της Mlawa στο δισδιάστατο περιβάλλον του Radiant

3.2.2.6. Μουσική

Η μουσική σε ένα παιχνίδι ή, στην περίπτωσή μας, σε μία πίστα, παίζει πολύ σημαντικό ρόλο στην τελική εμπειρία του χρήστη. Χωρίς αυτή, ο κόσμος μας, όσο όμορφος και εμπλουτισμένος με λεπτομέρειες και να είναι, φαίνεται κενός και μίζερος.

Για μουσική, χρησιμοποιήσαμε κομμάτια τόσο του ίδιου του παιχνιδιού, όσο και από άλλα πολεμικά παιχνίδια, όπως τα *Call of Duty: WWII* και *Battlefield V*. Κάθε κομμάτι εκπέμπει ένα ξεχωριστό δυναμισμό και έχει εφαρμοστεί στο κατάλληλο σημείο της πίστας, δίνοντας στο παίκτη διαφορετικά συναισθήματα.

⇒ Στην εισαγωγή, η μουσική είναι σκοτεινή και τρομακτική, σαν κάτι επικίνδυνο να περιμένει τον παίκτη έξω από το οχυρό.

⇒ Κατά τη διάρκεια τη μάχης, η μουσική αποκτά ένταση, είναι πιο γοργή και σε συνδυασμό με το gameplay θέτει τον παίκτη στο επίκεντρο της δράσης.

⇒ Όσο ο παίκτης διασχίζει το μονοπάτι, η μουσική παύει και ηχεί το απαλό άκουσμα της βροχής. Ο παίκτης άμεσα αντιλαμβάνεται ότι πλέον δεν κινδυνεύει.

⇒ Όταν φτάσουν οι ενισχύσεις των συμμάχων και εξολοθρεύσουν τους εχθρούς, η μουσική δίνει μία ελπίδα στον παίκτη ότι ίσως και να κατάφερε να επιβιώσει κάτι που, μέχρι τότε, φαινόταν αναπότρεπτο.

3.2.2.7. Level Optimization

Υπολείπεται να εφαρμοστεί ένα πολύ σημαντικό στοιχείο και αυτό είναι η βελτίωση της απόδοσης της πίστας μας. Όπως προαναφέραμε, η id Tech 3 έχει περιορισμούς. Ένας από αυτούς είναι ότι δεν μπορεί να διαβάζει συγχρόνως πολλές λεπτομέρειες αφού η μηχανή κουράζεται και μειώνεται η ομαλότητα της εμπειρίας του παίκτη. Αν θέλουμε να πετύχουμε βελτιωμένη απόδοση στην πίστα μας, απαιτείται αρχικά να τη σχεδιάσουμε ορθά και στη συνέχεια να ακολουθήσουμε τις διαδικασίες *Portaling* και *VClog*.

Ορθός σχεδιασμός της πίστας σημαίνει η πίστα να διασπάται σε μικρές περιοχές οι οποίες μπορούν να ενώνονται μεταξύ τους αλλά χωρίς να υφίσταται άμεση ορατότητα από τη μία περιοχή στην άλλη. Αυτός είναι και ο πρωταρχικός λόγος που τοποθετήσαμε το μονοπάτι ανάμεσα στα δύο πεδία μάχης. Το μονοπάτι είναι σχεδιασμένο ώστε όταν ο παίκτης βρίσκεται σε αυτό, δεν έχει σε καμία περίπτωση ορατότητα συγχρόνως και στις δύο περιοχές, αλλά μονάχα σε μία ή καμία, αναλόγως το σημείο που βρίσκεται.

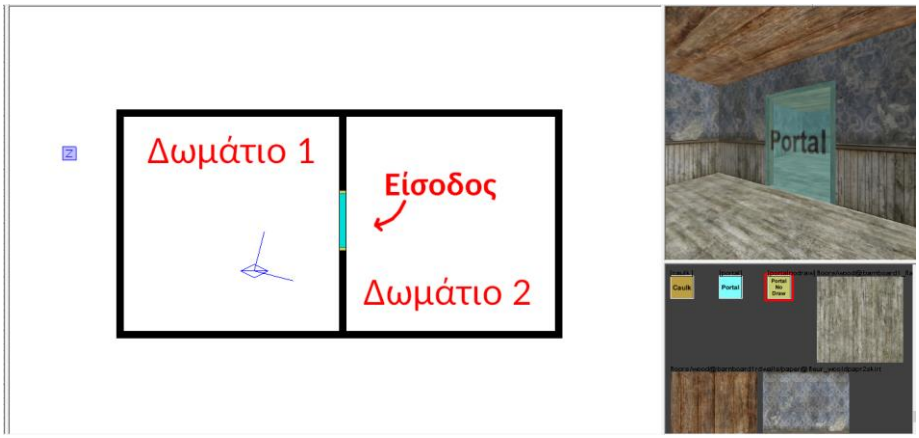
Εικόνα 42

3.2.2.7.1.1. Portals

Στη συνέχεια, περνάμε στη διαδικασία του *portaling*. Η πίστα πρέπει να χωριστεί σε κελιά. Κάθε περιοχή αποτελεί ένα κελί.

Για να χωρίσουμε την πίστα σε κελιά πρέπει να προσθέσουμε **portals**. Αυτό γίνεται εφαρμόζοντας τις υφές *portal* και *portalNoDraw* σε γεωμετρικά σχήματα με τα οποία θα περιφράξουμε κάθε κελί της πίστας μας.

⇒ Όταν μιλάμε για **εσωτερικούς χώρους**, η διαδικασία είναι απλή. Αν για παράδειγμα έχουμε ένα χώρο χωρισμένο σε δύο δωμάτια, το μόνο που χρειάζεται να κάνουμε είναι να "κλείσουμε" τις εισόδους μεταξύ των δωματίων με portals αφού τα δωμάτια ούτως ή άλλως περιβάλλονται από τοίχους και ταβάνι. Για να τοποθετήσουμε αποτελεσματικά ένα portal σε μία είσοδο, εφαρμόζουμε την υφή *portalNoDraw* στο γεωμετρικό σχήμα που καλύπτει την είσοδο εκτός της πλευράς που κοιτάει το επόμενο κελί, εφαρμόζοντάς της την υφή *portal*. Τα portals συνδέουν δύο κελιά.



Εικόνα 43 – Παράδειγμα portaling #1

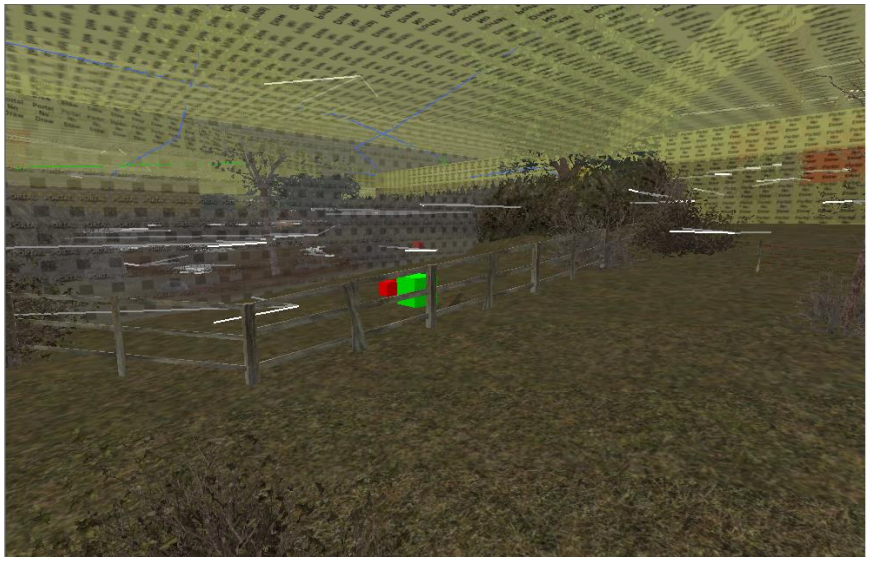
⇒ Η διαδικασία του *portaling* γίνεται πιο σύνθετη και απαιτητική όταν πρέπει να εφαρμοστεί σε **εξωτερικούς χώρους** αφού δεν υπάρχουν τοίχοι και επιφάνειες

που να κρύβουν την ορατότητα του παίκτη. Πρέπει να σχεδιαστούν γεωμετρικά σχήματα (*portals*) που να περιφράζουν κάθε περιοχή. Η πλευρά κάθε γεωμετρικού σχήματος πρέπει να αγγίζει την αντίστοιχη πλευρά του άλλου. Χρειάζεται ιδιαίτερη προσοχή αφού αν υπάρχει κενό μεταξύ τους ή τα portals μοιράζονται τον ίδιο χώρο, το *portaling* δε γίνεται δεκτό από τον compiler εμφανίζοντας σφάλμα.

Η Mlawa είναι χωρισμένη σε τέσσερα βασικά κελιά, όπως φαίνεται στην εικόνα 3.2.2.20:

- Κελί **Οχυρό**
- Κελί **Περιοχή 1**
- Κελί **Μονοπάτι**
- Κελί **Περιοχή 2**

Εικόνα 44 – Τα κελιά της Mlawa που δημιουργήθηκαν μέσω portals



Εικόνα 45 – Παράδειγμα portaling #2

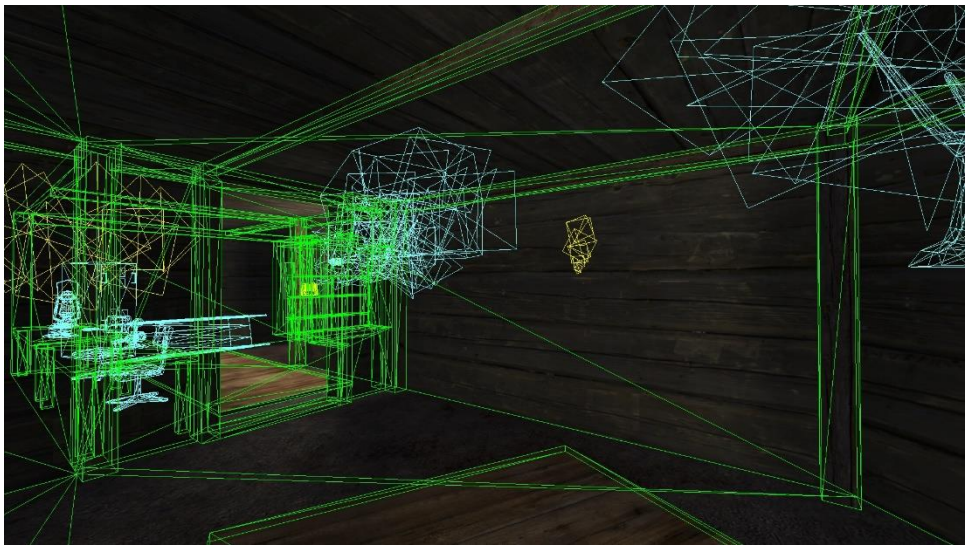
Στην Εικόνα 45, βλέπουμε ότι η κάμερα βρίσκεται στο κελί “Περιοχή 1” και κοιτάει το κελί “Μονοπάτι”. Κάθε κελί είναι σφραγισμένο με portals.

Πως όμως τα portals βοηθούν στην αύξηση της απόδοσης της πίστας;

Ας υποθέσουμε ότι ο παίκτης βρίσκεται στο αρχικό οχυρό της πίστας. Όσο βρίσκεται στο κελί “Οχυρό”, η μηχανή κρύβει τα υπόλοιπα κελιά με αποτέλεσμα να μη χρειάζεται να διαβάζει λεπτομέρειες που ο παίκτης δε μπορεί να δει από τη θέση που βρίσκεται. Όταν πλησιάσει και στραφεί στην έξοδο του οχυρού και όσο έχει ορατότητα στο κελί “Περιοχή 1”, η μηχανή αυτομάτως φανερώνει το νέο κελί και πλέον διαβάζει δύο κελιά. Όταν ο παίκτης βγει από το οχυρό, το κελί “Οχυρό” κρύβεται.

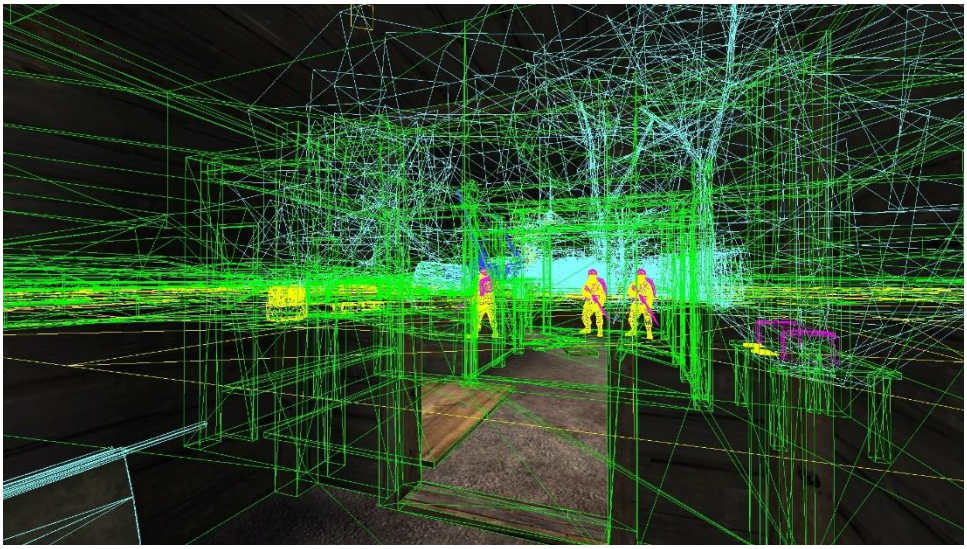
Ενεργοποιώντας την εντολή *r_showtris* στη μηχανή παιχνιδιού, μπορούμε να παρατηρήσουμε μέσα από τοίχους τη γεωμετρία που διαβάζει η μηχανή από τη θέση που βρίσκεται ο παίκτης.

Όπως παρατηρούμε στις παρακάτω εικόνες, ο παίκτης βρίσκεται το κελί “Οχυρό” και αρχικά κοιτάει προς το κελί “Περιοχή 1”. Ο τοίχος εμποδίζει την ορατότητά του και η μηχανή **δε** διαβάζει τη γεωμετρία του επόμενου κελιού



Εικόνα 46 – Παράδειγμα portaling #3

Στη συνέχεια, παίκτης βρίσκεται στην **έξοδο** του κελιού “Οχυρό” και κοιτάει προς το κελί “Περιοχή 1” έχοντας ορατότητα σε αυτό. Η μηχανή διαβάζει όλες τις λεπτομέρειες του επόμενου κελιού.

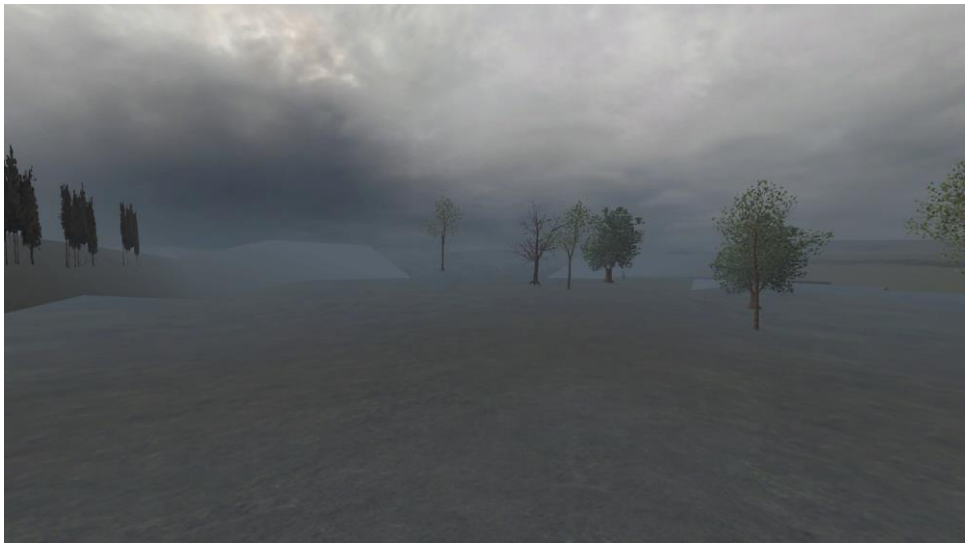


Εικόνα 47 - Παράδειγμα portaling #4

Αξίζει να αναφέρουμε πως κάθε περιοχή αλλά και το μονοπάτι περιβάλλονται από μοντέλα θάμνων. Πέρα από το γεγονός ότι οι θάμνοι εμπλουτίζουν τον κόσμο και ταιριάζουν με το περιβάλλον που φτιάξαμε, βοηθούν ώστε ο παίκτης να μην μπορεί να δει άλλα κελιά. Οι θάμνοι εμποδίζουν τον παίκτη να δει στοιχεία της πίστας που δεν πρέπει.

Εικόνα 48

Από τη στιγμή που έχουμε προσθέσει portals με επιτυχία στην πίστα, αν ο παίκτης μπορούσε να δει τα άλλα κελιά, στο σημείο του κάθε κελιού θα έβλεπε ένα κενό αφού η μηχανή το έχει αποκρύψει, αλλοιώνοντας τον ρεαλισμό. Οι θάμνοι εμποδίζουν τον παίκτη στο να παρατηρήσει πέρα από το κελί στο οποίο βρίσκεται.



Εικόνα 49

Το τοπίο που θα έβλεπε ο παίκτης αν δεν υπήρχαν θάμνοι. Το κελί “Περιοχή 1” έχει κρυφτεί από τη μηχανή λόγω των portals που εφαρμόσαμε

Το *portaling*, λοιπόν, αποτελεί μία πολύ σημαντική προσθήκη στην πίστα μας αφού από αυτή κρίνεται η εμπειρία του τελικού χρήστη. Η ορθή και αποτελεσματική εφαρμογή των portal κρίνει τον επαγγελματικό σχεδιασμό μιας πίστας.

3.2.2.7.1.2. VClog

Ένα μικρό πρόβλημα που εντοπίζουμε όταν φτιάχνουμε μεγάλες, σε έκταση, πίστες και επηρεάζει την εμπειρία του παίκτη είναι όταν ο παίκτης περνάει από το ένα κελί στο άλλο, η μηχανή "παγώνει" για λιγότερο από ένα δευτερόλεπτο επιχειρώντας να διαβάσει τις λεπτομέρειες του νέου κελιού. Συγκεκριμένα, οι οντότητες, όπως τρισδιάστατα μοντέλα και A.I., φωτίζονται από πηγές φωτός της πίστας. Σε σύνθετους κόσμους, ο υπολογισμός των πηγών φωτός που επηρεάζουν τις οντότητες απαιτεί από τη μηχανή πολλή δουλειά.

Για να αποτρέψουμε τη μηχανή να το κάνει αυτό σε πραγματικό χρόνο, δημιουργούμε ένα αρχείο που λέγεται *VClog*, φορτώνοντας την πίστα μας στη μηχανή παιχνιδιού με μία ειδική εντολή. Στη συνέχεια, κινούμαστε προσπαθώντας να καλύψουμε όσο το δυνατόν περισσότερο χώρο "τρώγοντας" τις τελείες που εμφανίζονται στην οθόνη, σαν να παίζουμε «Pac-man».

Εικόνα 50 – Διαδικασία δημιουργίας VClog στη μηχανή παιχνιδιού

Όσο κινούμαστε, στο αρχείο καταχωρούνται στοιχεία. Όταν ολοκληρώσουμε τη διαδικασία αυτή, κάνουμε έξοδο από τη μηχανή και οι πληροφορίες που συλλέχθηκαν καταχωρούνται στο αρχείο *BSP* της πίστας.

Αυτή η διαδικασία έχει ως αποτέλεσμα η πίστα να χρειάζεται περισσότερα δευτερόλεπτα να φορτώσει, αφού διαβάζει από την αρχή όλες τις λεπτομέρειες των κελιών αλλά κατά τη διάρκεια του παιχνιδιού το "πάγωμα" της οθόνης απαλείφεται.

3.3. Πακετάρισμα

Συνήθως, τα παιχνίδια που βασίζονται στην id Tech περιλαμβάνουν τα αρχεία τους σε συμπιεσμένα αρχεία *PK3*. Αφού ολοκληρώσαμε την βελτιστοποίηση της απόδοσης της πίστας, είμαστε έτοιμοι να αντιγράψουμε όλα τα απαραίτητα αρχεία που σχετίζονται με αυτή, όπως υφές που δημιουργήσαμε, τα μουσικά κομμάτια, τα αρχεία *BSP*, *GSC* και *SHADER* σε ένα φάκελο, τον οποίο στη συνέχεια θα συμπιέσουμε σε αρχείο *PK3*, μέσω ενός προγράμματος συμπίεσης αρχείων.

Με τον τρόπο αυτό, μπορούμε εύκολα να μοιραστούμε την πίστα μας με όποιο μέλος της κοινότητας του παιχνιδιού θέλει να την δοκιμάσει.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Καταληκτικά, δουλεύοντας στο Radiant, παρατηρούμε ότι η ανάπτυξη ενός τρισδιάστατου κόσμου και γενικότερα η δημιουργία ενός βιντεοπαιχνιδιού είναι μία σύνθετη και χρονοβόρα διαδικασία. Χρειάζεται μεγάλο ανθρώπινο δυναμικό για την επίτευξη του τελικού αποτελέσματος, αφού απαιτούνται συγκεκριμένες γνώσεις για κάθε μέρος της παραγωγής, όπως γνώσεις coding, level design, sound design και environmental art.

Ακόμη και μία πίστα σαν τη Mlawa, χρειάστηκε πολλές ώρες δουλειάς αφού προϋποθέτει πολλές διαφορετικές γνώσεις για τη δημιουργία της, δηλαδή από τον ορθό σχεδιασμό του κόσμου, τη σωστή επιλογή υφών, το ελκυστικό pacing και τον κώδικα έως το optimization και το portaling.

Παρόλα αυτά, το Radiant, όσο χρονοβόρο και να είναι, μας δίνει τη δυνατότητα να κάνουμε πραγματικότητα ό,τι έχουμε φανταστεί στο κομμάτι των τρισδιάστατων κόσμων χωρίς περιορισμούς.

Σχεδιάζοντας τη Mlawa, μάθαμε πως να σχεδιάζουμε αποτελεσματικά έναν τρισδιάστατο κόσμο που να διαβάζει μια μηχανή παιχνιδιών ώστε να διατηρήσουμε το ενδιαφέρον του παίκτη καθ' όλη τη διάρκεια χρόνου του στην πίστα αυτή, μέσω όλων των οπτικών και ηχητικών εφέ που προσθέσαμε γράφοντας κώδικα.

Βιβλιογραφία

Anthony Paul Beug, R. S. (2020). *Screen Space Reflection Techniques*. Ανάκτηση από https://ourspace.uregina.ca/bitstream/handle/10294/9245/Beug_Anthony_MSC_CS_Spring2020.pdf?sequence=1&isAllowed=y

Dammertz, H. (2012). *Sparse Virtual Texturing*. Ανάκτηση από http://holger.dammertz.org/stuff/notes_VirtualTexturing.html

Henry, D. (2004). *MD2 File Format*. Ανάκτηση από http://tfc.duke.free.fr/coding/md2-specs-en.html

History, T.-B. (2019). *How Much of a Genius-Level Move Was Using Binary Space Partitioning in Doom?* Ανάκτηση από https://twobithistory.org/2019/11/06/doom-bsp.html

History-Computer-Staff. (2021). *The Complete History of Tennis for Two*. Ανάκτηση από https://history-computer.com/tennis-for-two-complete-history/

History-Computer-Staff. (2022). *OXO Game Guide*. Ανάκτηση από https://history-computer.com/oxo-game-guide/

Houska, P. (2006). *Directional Lightmaps*. Ανάκτηση από http://www.decew.net/OSS/References/sem_ss06_07-Independant%20Explanation.pdf

Jaquays, P. (2000). *Q3Radiant Editor Manual*. Ανάκτηση από https://www.asc.ohio-state.edu/lewis.239/Gauge/q3rmanhtml.htm

Norman, J. (2022). *The Cathode Ray Tube Amusement Device*. Ανάκτηση από https://www.historyofinformation.com/detail.php?entryid=3573

Trainor-Fogleman, E. (2021). *Unity vs Unreal Engine: Game engine comparison guide for 2021*. Ανάκτηση από https://www.evercast.us/blog/unity-vs-unreal-engine