



ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ FIRST PERSON
SHOOTER (FPS) ΜΕ ΧΡΗΣΗ UNITY**

Του φοιτητή

Μάριος Ντόστη

Αρ. Μητρώου: 144281

Επιβλέπων

Ονοματεπώνυμο Ευκλείδης

Κεραμόπουλος

Καθηγητής

Ημερομηνία 06/09/2026

Τίτλος Δ.Ε.

Ανάπτυξη Παιχνιδιού First-Person Shooter (FPS) με χρήση Unity

Κωδικός Δ.Ε. 25258

Ονοματεπώνυμο φοιτητή/τών Μάριος Ντόστη

Ονοματεπώνυμο εισηγητή Ευκλείδης Κεραμόπουλος

Ημερομηνία ανάληψης Δ.Ε. 13-04-2025

Ημερομηνία περάτωσης Δ.Ε. 06-09-2026

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

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Μάριου Ντόστη που την εκπόνησε/αν. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραίτητα και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Στην οικογένεια και τους φίλους μου

Πρόλογος

Η ανάπτυξη βιντεοπαιχνιδιών είναι ένας από τους μεγαλύτερους κλάδους της πληροφορικής, συνδυάζοντας πολλά πράγματα όπως προγραμματισμό, σχεδίαση, αφήγηση και διαδραστική εμπειρία. Η επιλογή της υλοποίησης ενός 3D FPS (First-Person Shooter) παιχνιδιού με σκοπό την επιβίωση απέναντι σε zombie για την παρούσα πτυχιακή εργασία απορρέει από το ενδιαφέρον μου για τη μηχανή δημιουργίας παιχνιδιών Unity και την πρόκληση υλοποίησης συστημάτων πραγματικού χρόνου, όπως έλεγχος παίκτη, τεχνητή νοημοσύνη εχθρών, μηχανισμοί μάχης και διαχείριση καταστάσεων παιχνιδιού.

Περίληψη

Στην παρούσα πτυχιακή εργασία παρουσιάζεται η διαδικασία ανάπτυξης ενός τρισδιάστατου παιχνιδιού με χρήση του **Unity Engine** μιας από τις δημοφιλέστερες μηχανές ανάπτυξης παιχνιδιών.

Αρχικά γίνεται μια αναδρομή στην ιστορία των βιντεοπαιχνιδιών με ιδιαίτερη έμφαση στα FPS παιχνίδια και τα χαρακτηριστικά τους. Επιπλέον παρουσιάζεται η δομή των ομάδων ανάπτυξης παιχνιδιών και τα διαφορετικά ρόλων που απαιτούνται για την υλοποίηση ενός τέτοιου έργου. Ακολουθεί μια αναφορά στις γνωστές μηχανές ανάπτυξης παιχνιδιών που υπάρχουν σήμερα.

Στη συνέχεια, παρατίθεται μια εκτενής ανάλυση του **Unity Engine**, της πλατφόρμας που χρησιμοποιήθηκε για την υλοποίηση αυτής της εργασίας. Αναλύονται τα κύρια components, τα εργαλεία και οι δυνατότητες που προσφέρει.

Τέλος ακολουθεί λεπτομερής ανάλυση της δημιουργίας του παιχνιδιού, όπου παρακολουθείται ολόκληρη η διαδικασία ανάπτυξης και υλοποίησης.

Development of a First-Person Shooter (FPS) Game Using Unity

MARIOS DOSTI

Abstract

This thesis presents the development process of a three-dimensional game using the Unity Engine, one of the most popular game development platforms.

Initially, a historical overview of video games is provided, with particular emphasis on First-Person Shooter (FPS) games and their characteristics. Furthermore, the structure of game development teams and the different roles required for the implementation of such a project are presented. This is followed by an overview of the well-known game development engines available today.

Subsequently, an extensive analysis of the Unity Engine, the platform used for the implementation of this thesis, is provided. The main components, tools, and capabilities it offers are analyzed in detail.

Finally, a detailed analysis of the game's creation process is presented, covering the entire development and implementation workflow from initial design to final testing and optimization.

Ευχαριστίες

Στην οικογένεια και τους φίλους μου και στον καθηγητή Κ.Κεραμόπουλο μου έδωσε την ευκαιρία να ασχοληθούμε με αυτό το θέμα.

Περιεχόμενα

Πρόλογος.....	v
Περίληψη.....	vi
Abstract	vii
Ευχαριστίες	viii
Περιεχόμενα	ix
Κατάλογος Σχημάτων	xiii
Συντομογραφίες.....	xviii
Κεφάλαιο 1ο: Εισαγωγή	1
1.1 Γενικό υπόβαθρο και κίνητρα	1
Κεφάλαιο 2ο: Βιντεοπαιχνίδια.....	3
2.1 Εισαγωγή.....	3
2.2 Βιντεοπαιχνίδια	3
2.3 Ιστορική αναδρομή βιντεοπαιχνιδιών	4
2.3.1 Η Αρχή των Βιντεοπαιχνιδιών (1940s-1950s)	4
2.3.2 Arcade Era και Εμπορική Έκρηξη (1970s-1980s)	4
2.3.3 Τρισδιάστατα Γραφικά και Σύγχρονες Κονσόλες (1990s–Σήμερα).....	5
2.4 Κατηγορίες Βιντεοπαιχνιδιών	5
2.4.1 Παιχνίδια Δράσης – Action Games	6
2.4.2 Παιχνίδια Ρόλων Role Playing Games – RPG	7
2.4.3 Παιχνίδια Στρατηγικής Strategy Games.....	8
2.4.4 Παιχνίδια Προσομοίωσης Simulation Games	9
2.4.5 Παιχνίδια Αθλητισμού και Αγώνων Sports & Racing Games	9
2.5 Επίλογος.....	10
Κεφάλαιο 3ο: Θεωρητικό Υπόβαθρο.....	11
3.1 Εισαγωγή.....	11
3.2 Τεχνολογίες 3D Graphics & Physics.....	11
3.2.1 Τρισδιάστατη Γραφική και Rendering Pipeline	11
3.2.2 Physics Engines και Collision Detection.....	12
3.2.3 3D Animation Systems & Rigging.....	13
3.3 Τεχνητή νοημοσύνη -AI και βιντεοπαιχνίδια.....	14
3.3.1 Συμπεριφορική τεχνητή νοημοσύνη - Behavioral AI.....	14
3.3.2 Navigation AI	15

3.4	Μηχανές δημιουργίας βιντεοπαιχνιδιών – Game Engine.....	16
3.4.1	Τι είναι Game Engine	16
3.4.2	Κύριες Μηχανές Παιχνιδιών (Game Engines)	17
3.5	Ομάδα Ανάπτυξης Βιντεοπαιχνιδιών	18
3.5.1	Σχεδιαστής-Game Designer.....	18
3.5.2	Προγραμματιστή-Programmer	18
3.5.3	Καλλιτέχνες -Artist.....	18
3.5.4	Μηχανικός / Σχεδιαστής Ήχου-Sound Engineer	18
3.5.5	Διαχειριστής Έργου-Project Manager	19
3.5.6	Δοκιμαστής-Tester.....	19
3.6	Επίλογος.....	19
Κεφάλαιο 4ο:	Unity Engine – Κύρια παράθυρα και εργαλεία	21
4.1	Εισαγωγή.....	21
4.2	Hierarchy Window	21
4.3	Scene View	21
4.4	Game View.....	22
4.5	Inspector Window	22
4.6	Console Window	23
4.7	Lighting	23
4.8	Animator.....	24
4.9	Animation.....	25
4.10	Επίλογος.....	25
Κεφάλαιο 5ο:	Ανάλυση και σχεδίαση συστήματος	27
5.1	Εισαγωγή.....	27
5.2	Έγγραφο Σχεδιασμού Παιχνιδιού (Game Design Document - GDD).....	27
5.2.1	Περιγραφή παιχνιδιού.....	27
5.2.2	Στάδια παιχνιδιού-Gameplay Loop	27
5.2.3	Μηχανισμοί Παίκτη -Player Mechanics.....	29
5.2.4	Μηχανισμοί εχθρών – Enemy mechanics.....	29
5.2.5	Σύστημα Εμφάνισης (Spawning)	30
5.2.6	Αντικείμενα και Συλλογή (Pickups).....	30
5.2.7	Minimap & Radar System	31
5.3	Δομή Σκηνών-Scene Structure	31
5.3.1	Main menu scene	31
5.3.2	MainScene	31

5.4	Λειτουργικές & Μη Λειτουργικές Απαιτήσεις	32
5.4.1	Λειτουργικές Απαιτήσεις-Functional Requirements	32
5.4.2	Μη Λειτουργικές Απαιτήσεις (Non-Functional Requirements)	33
5.5	Τεχνικές Προδιαγραφές.....	33
5.5.1	Development Environment.....	33
5.5.2	Ελάχιστες λειτουργικές απαιτήσεις	34
5.6	Επίλογος.....	34
Κεφάλαιο 6ο:	Ανάπτυξη και Υλοποίηση.....	35
6.1	Εισαγωγή.....	35
6.2	Ξεκινώντας με Unity	35
6.3	Menus	36
6.3.1	Main Menu	37
6.3.2	Controller Menu	41
6.3.3	Pause Menu	42
6.4	Terrain	44
6.4.1	Δημιουργία και Διαμόρφωση Terrain	44
6.4.2	Paint Texture - Χρωματισμός του Τοπίου	46
6.4.3	Τοποθέτηση Δέντρων και Θάμνων στο Terrain	47
6.4.4	Αρχιτεκτονικά Στοιχεία Terrain	50
6.4.5	NavMesh.....	53
6.5	Φωτισμός & Ατμόσφαιρα	55
6.5.1	Directional Light.....	55
6.5.2	Skybox Panorama	55
6.5.3	Fog (Ομίχλη)	57
6.6	Player.....	57
6.6.1	Δημιουργία και Σχεδιασμός Χαρακτήρα Παίκτη	57
6.6.2	Σύστημα Κάμερας	59
6.6.3	Σύστημα Κίνησης Παίκτη	62
6.6.4	Animation-Animator	63
6.6.5	Σύστημα Υγείας & Kits Υγεία	67
6.7	Σύστημα Πυροβολισμού & Πυρομαχικών (Shooting & Ammo System)	71
6.7.1	Μηχανισμός Shooting:	71
6.7.2	Εφέ Αίματος (Particle System).....	72
6.7.3	Συγχρονισμός με το Animation – Animation Events	73
6.7.4	Σύστημα Πυρομαχικών και Reload.....	74

6.7.5	Raycast & Crosshair	75
6.7.6	Εμφάνιση και Απόκρυψη Όπλου.....	77
6.7.7	Σύστημα Ammo Kit.....	77
6.7.8	Σύστημα Ήχου (Audio System)	78
6.8	Σύστημα Εχθρών – Zombies (Enemy System)	79
6.8.1	Zombie FSM – Finite State Machine.....	79
6.8.2	Σύστημα Δημιουργίας Zombie (Zombie Spawning System)	82
6.8.3	Zombie Animations	83
6.8.4	Zombie Models.....	84
6.8.5	Zombie Sound.....	84
6.9	HUD	85
6.9.1	Health Bar.....	85
6.9.2	Ammo Counter	85
6.9.3	Kill Counter	86
6.9.4	Radar / Minimap System	86
6.10	Επίλογος.....	87
Κεφάλαιο 7ο:	Συμπεράσματα και Μελλοντικές Βελτιώσεις	89
7.1	Συμπεράσματα.....	89
7.2	Μελλοντικές Βελτιώσεις	89
BIBΛΙΟΓΡΑΦΙΑ.....		90

Κατάλογος Σχημάτων

Εικόνα 2-1 Βιντεοπαιχνίδια.....	Σελ. 3
Εικόνα 2-2 Tennis for Two.....	Σελ. 4
Εικόνα 2-3 Counter Strike.....	Σελ. 7
Εικόνα 2-4 World of Warcraft.....	Σελ. 8
Εικόνα 2-5 Age of Empire.....	Σελ. 8
Εικόνα 2-6 Microsoft Flight Simulator	Σελ.9
Εικόνα 2-7 FIFA	Σελ.10
Εικόνα 3-1 Shaders Collection	Σελ.12
Εικόνα 3-2 Textures.....	Σελ. 12
Εικόνα 3-3 εσωτερικός σκελετός	Σελ.13
Εικόνα 3-4 Components of FSM	Σελ.15
Εικόνα 3-5 Navigation Mesh όπως φαίνεται μέσα στο Unity	Σελ.16
Εικόνα 4-1 Παράθυρο Hierarchy	Σελ.21
Εικόνα 4-2 Παράθυρο Scene	Σελ.22
Εικόνα 4-3 Παράθυρο Game.....	Σελ. 22
Εικόνα 4-4 Παράθυρο Inspector.....	Σελ. 23
Εικόνα 4-5 Παράθυρο Console.....	Σελ. 23
Εικόνα 4-6 Παράθυρο Lighting.....	Σελ. 24
Εικόνα 4-7 Παράθυρο Animato.....	Σελ. 24
Εικόνα 4-8 Παράθυρο Animation.....	Σελ. 25
Εικόνα 5-1 διάγραμμα ροής αλληλεπίδρασης.....	Σελ. 28
Εικόνα 5-2 Zombie FSM διάγραμμά	Σελ.30

Εικόνα 6-1 Δημιουργία νέου project στο unity	Σελ.35
Εικόνα 6-2 Καθορισμός ονόματος και είδος παιχνιδιού.....	Σελ. 36
Εικόνα 6-3 Unity Editor	Σελ.36
Εικόνα 6-4 Δημιουργία Canvas.....	Σελ. 37
Εικόνα 6-5 Background Εικόνα.....	Σελ. 38
Εικόνα 6-6 Ρυθμίσεις Text.....	Σελ. 38
Εικόνα 6-7 Main Menu.....	Σελ. 39
Εικόνα 6-8 Build Settings.....	Σελ. 39
Εικόνα 6-9 Main Menu Script.....	Σελ. 40
Εικόνα 6-10 On Click() function.....	Σελ. 40
Εικόνα 6-11 Music Controller Settings.....	Σελ. 41
Εικόνα 6-12 Music Controller Script.....	Σελ. 41
Εικόνες 6-13 Controllers Scene.....	Σελ. 42
Εικόνα 6-14 Pause Menu Screen.....	Σελ. 43
Εικόνα 6-15 Απόσπασμα από το PauseMenu Pause() και Resume() function.....	Σελ. 44
Εικόνα 6-16 Δημιουργία Terrain.....	Σελ. 45
Εικόνα 6-17 Terrain Settings στο Terrain Tool	Σελ.45
Εικόνα 6-18 Διαστάσεις Terrain.....	Σελ. 45
Εικόνες 6-19 Paint Terarin και Brush Settings.....	Σελ. 46
Εικόνα 6-20 Paint Texture Setting και Texture.....	Σελ. 47
Εικόνα 6-21 Το τελικό terrain με όλα τα textures εφαρμοσμένα.....	Σελ. 47
Εικόνα 6-22 Paint Tree Settings.....	Σελ. 48
Εικόνα 6-23 Mass Place Tree Settings.....	Σελ.48

Εικόνα 6-24 Τα δέντρα όπως φαίνονται από το Scene.....	Σελ. 49
Εικόνα 6-25 Paint Details Setting.....	Σελ. 49
Εικόνα 6-26 Details Settings Window	Σελ.50
Εικόνα 6-27 Περιοχή του terrain με τοποθετημένα κτήρια.....	Σελ. 51
Εικόνα 6-28 Box Collider όπως φαίνεται εντός παιχνιδιού.....	Σελ. 51
Εικόνα 6-29 Άνω όψη του ολοκληρωμένου terrain με όλα τα στοιχεία.....	Σελ. 52
Εικόνα 6-30 Window - Package Manager	Σελ.53
Εικόνα 6-31 Εγκατάσταση AI Navigation package.....	Σελ. 53
Εικόνα 6-32 NavMeshSurface Component	Σελ.54
Εικόνα 6-33 Selected Agent Type Settings.....	Σελ. 54
Εικόνα 6-34 Directional Light Settings.....	Σελ. 55
Εικόνα 6-35 Custom Sky Box Scene View.....	Σελ. 56
Εικόνα 6-36 Environment Lighting ρυθμίσεις - Skybox.....	Σελ. 56
Εικόνα 6-37 Fog Settings	Σελ.57
Εικόνα 6-38 Steve Model.....	Σελ. 58
Εικόνα 6-39 Hierarchy δομή του παίκτη.....	Σελ. 58
Εικόνα 6-40 Ο χαρακτήρας Steve στη σκηνή με το όπλο.....	Σελ. 59
Εικόνα 6-41 Η κάμερα ως Child του FPC χαρακτήρα.....	Σελ. 59
Εικόνα 6-42 Ενσωμάτωση της κάμερας στο MainCharacterController script μέσω του Cam field	Σελ.60
Εικόνα 6-43 FixedUpdate()	Σελ.60
Εικόνα 6-44 ClampRotationAroundXAxis()	Σελ.61
Εικόνα 6-45 Cursor Lock Management - SetCursorLock(), UpdateCursorLock() και InternalLockUpdate() methods.....	Σελ. 61
Εικόνα 6-46 Sprint().....	Σελ. 62

Εικόνα 6-47 Jump().....	Σελ. 63
Εικόνα 6-48 Creating Animator Controller	Σελ.63
Εικόνα 6-49 Σύνδεση του Animator Controller με το MainCharacterController	Σελ.64
Εικόνα 6-50 FSM του MainCharacterController Animator - όλες οι πιθανές μεταβάσεις animations	Σελ. 66
Εικόνα 6-51 Επιλογή Add Event.....	Σελ. 67
Εικόνα 6-52 TakeHit()	Σελ.69
Εικόνα 6-53 Το playerBloodSplatter animation όπως φαίνεται εντός παιχνιδιού	Σελ.69
Εικόνα 6-54 StevePrefab.....	Σελ. 69
Εικόνα 6-55 Game Over screen	Σελ.70
Εικόνα 6-56 “MedKit” Tag	Σελ.70
Εικόνα 6-57 MedKit ()	Σελ.71
Εικόνα 6-58 Med kit model	Σελ.71
Εικόνα 6-59 Shooting()	Σελ.72
Εικόνα 6-60 Η μέθοδος ProcessZombieHit() και η υλοποίηση του Raycast.	Σελ.72
Εικόνα 6-61 Οι ρυθμίσεις του ZombieBloodSplatter Particle System.	Σελ. 73
Εικόνα 6-62: Οι συναρτήσεις Fire() και canShoot() στο script	Σελ.73
Εικόνα 6-63: Τοποθέτηση Animation Events στο Fire animation clip για τον συγχρονισμό του ήχου και του ρυθμού πυροβολισμού.	Σελ.74
Εικόνα 6-64 Reload()	Σελ.75
Εικόνα 6-65 Light Settings.....	Σελ.76
Εικόνα 6-66 Crosshair.....	Σελ. 77
Εικόνα 6-67 HideShowArm().....	Σελ. 77
Εικόνα 6-68 AmmoBox().....	Σελ. 78
Εικόνα 6-69 Ammo BOX.....	Σελ. 78

Εικόνα 6-70 FootSteps().....	Σελ. 79
Εικόνα 6-71 FSM States.....	Σελ. 79
Εικόνα 6-72 IDLE STATE.....	Σελ. 80
Εικόνα 6-73 WANDER STATE	Σελ.80
Εικόνα 6-74 CHASE STATE.....	Σελ. 81
Εικόνα 6-75 ATTACK STATE.....	Σελ. 81
Εικόνα 6-76 DEAD STATE.....	Σελ. 82
Εικόνα 6-77 ZombieSpawner Script	Σελ.83
Εικόνα 6-78 Το διάγραμμα του Animator Controller για τα ζόμπι, με τις πέντε βασικές καταστάσεις (Idle, Walking, Running, Attack, Death) και τις μεταβάσεις τους.	Σελ. 84
Εικόνα 6-79 Zombie configuration table	Σελ.84
Εικόνα 6-80 HUD From Game View.....	Σελ. 85
Εικόνα 6-81 Κώδικας υπολογισμού παίκτη	Σελ. 86
Εικόνα 6-82 Υπολογισμός αντικειμένων με βάση τον παίκτη.....	Σελ. 86
Εικόνα 6-83 Προσθήκη αντικειμένων στο radar.....	Σελ. 86

Συντομογραφίες

ΔΠΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
FPS	First Person Shooter
UI	User Interface
FSM	Finite State Machine
AI	Artificial Intelligence
GPU	Graphics Processing Unit
FSM	Finite State Machines

Κεφάλαιο 1ο: Εισαγωγή

1.1 Γενικό υπόβαθρο και κίνητρα

Στη σημερινή εποχή τα βιντεοπαιχνίδια δεν αποτελούν απλώς μια μορφή ψυχαγωγίας αλλά ένα αναπόσπαστο κομμάτι της καθημερινότητας για πολλούς ανθρώπους. Είναι ένα μέσο προσωπικής έκφρασης και μια μορφή τέχνης που συνδυάζει την τεχνολογία, τον σχεδιασμό και τη δημιουργικότητα. Όταν κάποιος κάθεται μπροστά σε μια οθόνη για να παίξει δεν συμμετέχει απλώς σε ένα παιχνίδι· βυθίζεται σε έναν άλλο κόσμο.

Χαρακτηριστικά το 2025 η βιομηχανία των βιντεοπαιχνιδιών αποτελεί έναν από τους μεγαλύτερους κλάδους ψυχαγωγίας παγκοσμίως αποφέροντας έσοδα άνω των 201,6 δισεκατομμυρίων δολαρίων και οι χρήστες ξεπερνάνε τα 3,57 δισεκατομμύρια [1]. Αυτό δεν είναι τυχαίο τα παιχνίδια έχουν εξελιχθεί σε μια παγκόσμια μορφή επικοινωνίας ένα μέσο που φέρνει κοντά ανθρώπους από διαφορετικούς πολιτισμούς και κοινωνικά υπόβαθρα.

Πώς όμως δημιουργείται στην πραγματικότητα ένα παιχνίδι; Πώς μπορεί ένας προγραμματιστής να χτίσει έναν ολόκληρο κόσμο από το μηδέν; Ποιες είναι τεχνολογίες που κρύβονται πίσω από κάθε action κάθε animation και κάθε αλληλεπίδραση; Αυτά ακριβώς τα ερωτήματα αποτέλεσαν την έμπνευση για την παρούσα πτυχιακή εργασία.

Τα παιχνίδια πρώτου προσώπου (**FPS**) αποτελούν μια ξεχωριστή κατηγορία παιχνιδιών που έχει επηρεάσει σημαντικά την εξέλιξη της βιομηχανίας. Από το θρυλικό Doom έως σύγχρονους τίτλους παιχνιδιών όπως το Call of Duty και το Valorant, το είδος εξελίσσεται συνεχώς εισάγοντας νέες τεχνικές και μηχανισμούς παιχνιδιού. Πίσω από κάθε πυροβολισμό και κάθε κίνηση του εχθρού κρύβεται ένα σύνθετο **UI** επιλογών και τεχνικών με στόχο τη δημιουργία μιας ολοκληρωμένης εμπειρίας παιχνιδιού.

Στόχος της πτυχιακής εργασίας είναι η εξερεύνηση των συστημάτων αυτών μέσω της ανάπτυξης ενός παιχνιδιού FPS με θέμα τα ζόμπι χρησιμοποιώντας τη μηχανή **Unity Engine** μία από τις πιο δημοφιλείς και ευέλικτες σύγχρονες πλατφόρμες ανάπτυξης παιχνιδιών. Η υλοποίηση του έργου περιλαμβάνει την επίλυση σύνθετων τεχνικών προκλήσεων με ιδιαίτερη έμφαση στη ρεαλιστική κίνηση των χαρακτήρων, τους μηχανισμούς μάχης καθώς και την ενσωμάτωση **Μηχανής Πεπερασμένων Καταστάσεων (FSM)** για την αυτόνομη λήψη αποφάσεων και τη δυναμική συμπεριφορά των εχθρών.

1.2 Δομή της Εργασίας

Η παρούσα πτυχιακή αποτελείται από επτά βασικά κεφάλαια που καλύπτουν όλες τις πτυχές της ανάπτυξης ενός FPS με θέμα τα ζόμπι. Κάθε κεφάλαιο έχει συγκεκριμένο στόχο και περιεχόμενο, σχεδιασμένα ώστε να διευκολύνουν την κατανόηση και την υλοποίηση του έργου.

Το **πρώτο κεφάλαιο** παρουσιάζει το γενικό υπόβαθρο του έργου, τα κίνητρα που οδήγησαν στη δημιουργία του, και τη σημασία της τεχνολογίας game development στη σύγχρονη ψυχαγωγία.

Στο **δεύτερο κεφάλαιο** γίνεται μια ιστορική αναδρομή στα βιντεοπαιχνίδια, με ιδιαίτερη έμφαση στα FPS παιχνίδια και τα χαρακτηριστικά τους. Παρουσιάζονται οι κατηγορίες βιντεοπαιχνιδιών και αναλύονται οι μηχανισμοί που τα διέπουν.

Στο **τρίτο κεφάλαιο**, αναλύεται το θεωρητικό υπόβαθρο των τεχνολογιών που χρησιμοποιούνται στην ανάπτυξη βιντεοπαιχνιδιών, όπως τα τρισδιάστατα γραφικά, η φυσική προσομοίωση, η τεχνητή

νοημοσύνη και οι μηχανές παιχνιδιών. Επίσης, παρουσιάζεται η δομή των ομάδων ανάπτυξης βιντεοπαιχνιδιών και οι ρόλοι που απαιτούνται.

Το **τέταρτο κεφάλαιο** επικεντρώνεται στη μηχανή Unity και τα βασικά εργαλεία της. Περιγράφονται τα κύρια παράθυρα και τα συστήματα που προσφέρει το Unity Editor, όπως το Scene View, το Game View, το Hierarchy, το Inspector και το σύστημα φωτισμού και animation.

Το **πέμπτο κεφάλαιο** παρουσιάζει την ανάλυση και σχεδίαση του συστήματος. Περιγράφεται το έγγραφο σχεδιασμού παιχνιδιού (GDD), οι μηχανισμοί του παίκτη και των εχθρών, το σύστημα εμφάνισης ζόμπι, τα αντικείμενα συλλογής, καθώς και οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος.

Στο **έκτο κεφάλαιο**, περιγράφεται αναλυτικά η διαδικασία ανάπτυξης και υλοποίησης του παιχνιδιού. Παρουσιάζονται βήμα προς βήμα όλα τα στάδια κατασκευής, από τη δημιουργία του έργου στο Unity και τα μενού πλοήγησης, μέχρι τη διαμόρφωση του terrain, την υλοποίηση του χαρακτήρα του παίκτη, το σύστημα μάχης, την τεχνητή νοημοσύνη των εχθρών, το σύστημα ήχου και τη διεπαφή χρήστη.

Στο **έβδομο και τελευταίο κεφάλαιο**, αναλύονται τα συμπεράσματα της εργασίας και παρουσιάζονται προτάσεις για μελλοντικές βελτιώσεις, τόσο σε τεχνικό όσο και σε σχεδιαστικό επίπεδο, με σκοπό τη βελτίωση της εμπειρίας του παίκτη και την επέκταση του παιχνιδιού.

Κεφάλαιο 2ο: Βιντεοπαιχνίδια

2.1 Εισαγωγή

Τα βιντεοπαιχνίδια αποτελούν σήμερα μία από τις πιο δημοφιλείς μορφές ψυχαγωγίας. Στο κεφάλαιο αυτό θα εξετάσουμε τι είναι βιντεοπαιχνίδι και την εξέλιξη του κάνοντας μια αναδρομή στην ιστορία τους από την αρχή τους και την εποχή των ηλεκτρονικών παιχνιδιών arcade έως την έλευση της οικιακής κονσόλας και την ανάπτυξη των τρισδιάστατων γραφικών. Στη συνέχεια γίνεται προσπάθεια ταξινόμησης των παιχνιδιών σε βασικές κατηγορίες όπως δράσης, ρόλων, στρατηγικής, προσομοίωσης και αθλητικά προσδιορίζοντας έτσι την έννοια του βιντεοπαιχνιδιού και τις βασικές κατηγορίες στις οποίες χωρίζεται.

2.2 Βιντεοπαιχνίδια

“Με τον όρο βιντεοπαιχνίδι (αγγλικά: videogame) εννοείται οποιοδήποτε παιχνίδι πραγματοποιείται με τη χρήση κάποιας ηλεκτρονικής συσκευής. Αυτή μπορεί να είναι ένας ηλεκτρονικός υπολογιστής, μια κονσόλα βιντεοπαιχνιδιών, ένα κινητό τηλέφωνο και άλλα.” [2]. Κάθε αλληλεπίδραση εντός βιντεοπαιχνιδιού βασίζεται σε δύο πυλώνες: τη λήψη εντολών που ορίζεται ως είσοδος και την παροχή ανατροφοδότησης που ορίζεται σαν έξοδος. Ο έλεγχος ασκείται μέσω διαφόρων μέσων (όπως πληκτρολόγιο, ποντίκι, μοχλού – βλ. εικόνα 2-1), ενώ η απόκριση του συστήματος μεταδίδεται στον παίκτη μέσω οπτικών, ακουστικών και απτικών σημάτων [3].



Εικόνα 2-1 Βιντεοπαιχνίδια

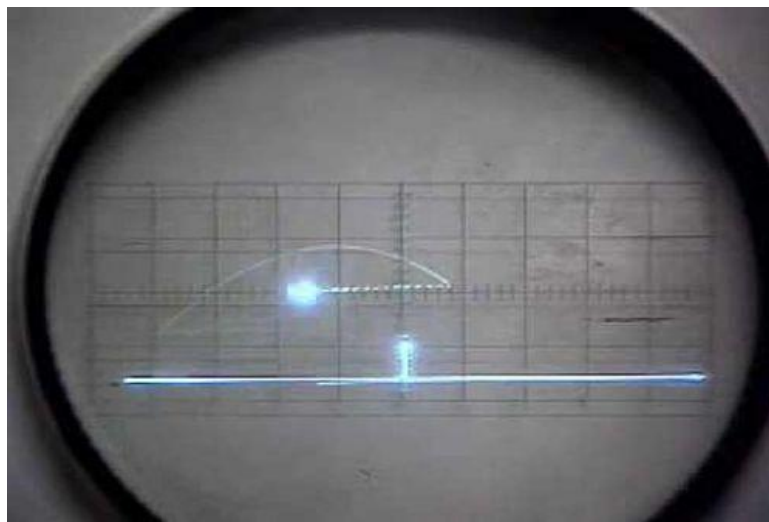
2.3 Ιστορική αναδρομή βιντεοπαιχνιδιών

Η ιστορία των βιντεοπαιχνιδιών ξεκίνησε στα τέλη της δεκαετίας του 1940 και εξελίχθηκε σταδιακά σε μια παγκόσμια βιομηχανία αξίας εκατοντάδων δισεκατομμυρίων δολαρίων [3]. Τα βιντεοπαιχνίδια πρωτοεμφανίστηκαν σε υπολογιστές πανεπιστημίων και στη συνέχεια επεκτάθηκαν σε ηλεκτρονικά παιχνίδια arcade, οικιακές κονσόλες και φορητές συσκευές, με κάθε εποχή να ανοίγει νέες τεχνολογικές δυνατότητες [4-5].

2.3.1 Η Αρχή των Βιντεοπαιχνιδιών (1940s-1950s)

Το 1958 ο William Higinbotham, μηχανικός στο Εθνικό Εργαστήριο Brookhaven δημιούργησε αυτό που θεωρείται ως το πρώτο παιχνίδι, το Tennis for Two στην οθόνη ενός παλμογράφου [3][5] (βλ. Εικόνα 2-2). Το Tennis for Two αποτελούσε μια απλή αλλά επαναστατική εξέλιξη στον χώρο της ψηφιακής ψυχαγωγίας. Ο τρόπος παιχνιδιού ήταν αρκετά απλός δύο παίκτες μπορούσαν να ελέγχουν την κίνηση μιας μπάλας χρησιμοποιώντας ειδικά χειριστήρια, προσπαθώντας να τη στείλουν πάνω από ένα δίχτυ που χώριζε την οθόνη [3]. Παρά τη σημασία του παρέμεινε κυρίως ένα επιστημονικό πείραμα και δεν κυκλοφόρησε ποτέ στο εμπόριο [5].

Δύο χρόνια αργότερα, το 1962 κάποιοι φοιτητές στο Τεχνολογικό Ινστιτούτο της Μασαχουσέτης (MIT) έφτιαξαν το Spacewar, ένα παιχνίδι διαστημικής μάχης που έτρεχε στον υπολογιστή PDP-1 [3][5]. Στο Spacewar δύο παίκτες χειρίζονταν διαστημόπλοια κάνοντας ελιγμούς στο διάστημα και προσπαθώντας να χτυπήσουν ο ένας τον άλλον με πυραύλους[5]. Το παιχνίδι αυτό διαδραμάτισε σημαντικό ρόλο, καθώς έθεσε τα θεμέλια για τα ψηφιακά παιχνίδια πολλαπλών παικτών και ήταν η αρχή του ανταγωνιστικού παιχνιδιού [3].



Εικόνα 2-2 Tennis for Two

2.3.2 Arcade Era και Εμπορική Έκρηξη (1970s-1980s)

Η δεκαετία του 1970 σηματοδότησε τη μετάβαση των βιντεοπαιχνιδιών από τα ερευνητικά εργαστήρια στη μαζική αγορά. Το πρώτο εμπορικό παιχνίδι για αίθουσες ηλεκτρονικών παιχνιδιών arcade ήταν το Computer Space του 1971, το οποίο διαδέχτηκε το Pong της Atari το 1972, ένας πρωτοποριακός τίτλος που σημείωσε παγκόσμια επιτυχία και έθεσε τα θεμέλια για τον εμπορικό τομέα της βιομηχανίας [5]. Παράλληλα, ο Ralph Baer, ο οποίος συχνά αποκαλείται «Πατέρας των Βιντεοπαιχνιδιών», ανέπτυξε το

πρωτότυπο Brown Box, οδηγώντας στην κυκλοφορία του Magnavox Odyssey το 1972 της πρώτης οικιακής κονσόλας βιντεοπαιχνιδιών [6].

Στα τέλη της δεκαετίας του 1970 η κυκλοφορία του Atari 2600 το 1977 καθιέρωσε τις κονσόλες με υποδοχή κασέτας, επιτρέποντας στους χρήστες να παίζουν διαφορετικά παιχνίδια σε μία και μόνο συσκευή [5][4]. Ωστόσο, ένας ανεξέλεγκτος καταγισμός παιχνιδιών χαμηλής ποιότητας και ο κορεσμός της αγοράς οδήγησαν σε σοβαρή κρίση τη βιομηχανία βιντεοπαιχνιδιών των ΗΠΑ την περίοδο 1983–1985 [5]. Από το 1985 και μετά, ο κλάδος ανέκαμψε σταδιακά χάρη στην είσοδο της Nintendo στην αμερικανική αγορά και την κυκλοφορία του Nintendo Entertainment System (NES), το οποίο εισήγαγε αυστηρότερα πρότυπα ποιότητας και έθεσε τα θεμέλια για το σύγχρονο μοντέλο παραγωγής. Από εκείνη την περίοδο και μετά, η βιομηχανία των βιντεοπαιχνιδιών γνωρίζει συνεχή αύξηση, με νέες εταιρείες να εισέρχονται στο εμπορικό παιχνίδι [5][4].

2.3.3 Τρισδιάστατα Γραφικά και Σύγχρονες Κονσόλες (1990s–Σήμερα)

Η δεκαετία του 1990 χαρακτηρίστηκε από τη μετάβαση από τα δισδιάστατα (2D) περιβάλλοντα στα τρισδιάστατα (3D) γραφικά, γεγονός που άλλαξε ριζικά τις προσεγγίσεις στον σχεδιασμό παιχνιδιών και στον τρόπο ανάπτυξής τους. Η είσοδος της Sony στην αγορά με την κονσόλα PlayStation το 1994 και η ταυτόχρονη κυκλοφορία του Nintendo 64 το 1996 καθιέρωσαν τα τρισδιάστατα γραφικά και τα οπτικά μέσα CD-ROM ως τις κυρίαρχες τεχνολογίες και τα βασικά μέσα αποθήκευσης [4-5]. Κατά τη δεκαετία του 2000, η βιομηχανία εισήλθε στην εποχή των τεχνολογιών δικτύωσης και της ψυχαγωγίας πολυμέσων. Κονσόλες όπως το PlayStation 2, το αρχικό Xbox και το Nintendo Wii εισήγαγαν υποδομές για διαδικτυακό παιχνίδι, συστήματα ελέγχου βασισμένα στην κίνηση και πλατφόρμες ψηφιακής διανομής [5].

Αυτή η μεταβολή μεταξύ των γενεών οδήγησε στις σύγχρονες εμπορικές πλατφόρμες, οι οποίες κατά την τελευταία δεκαετία έχουν επίσης συνδεθεί άμεσα με τις κινητές συσκευές [4]. Παράλληλα, η τεχνολογική εξέλιξη και ειδικότερα η επίδραση των ηλεκτρονικών υπολογιστών, οι οποίοι διαθέτουν διαφορετικά συστήματα ελέγχου (πληκτρολόγιο, ποντίκι) από εκείνα που διέθεταν οι κονσόλες εκείνη την περίοδο, συνέβαλε στον μετασχηματισμό των ειδών παιχνιδιού και της κουλτούρας του gaming [4].

Στους ηλεκτρονικούς υπολογιστές (PC), η ανάπτυξη εξειδικευμένων ειδών βιντεοπαιχνιδιών, όπως τα παιχνίδια πρώτου προσώπου (FPS), τα παιχνίδια στρατηγικής πραγματικού χρόνου (RTS) και τα μαζικά διαδικτυακά παιχνίδια ρόλων πολλαπλών παικτών (MMORPG), οδήγησε στην εμφάνιση εμβληματικών τίτλων, συμπεριλαμβανομένων των Quake, StarCraft και άλλων [4]. Από αυτό προέκυψε ένα ευρύ φάσμα ειδών που κυμαίνεται από παιχνίδια δράσης, βολών και στρατηγικής έως προσομοιώσεις, παιχνίδια ρόλων (RPG), προσομοιώσεις αθλημάτων και παιχνίδια περιπέτειας [4].

Σήμερα, τα βιντεοπαιχνίδια αποτελούν έναν από τους ισχυρότερους τομείς της παγκόσμιας οικονομίας της ψυχαγωγίας. Η τεχνολογική τους εξέλιξη τροφοδοτείται από τη χρήση προηγμένων τεχνολογιών, όπως η εικονική και η επαυξημένη πραγματικότητα (VR/AR), η φωτορεαλιστική απεικόνιση γραφικών σε πραγματικό χρόνο (ray tracing) και η ενσωμάτωση σύνθετων αλγορίθμων τεχνητής νοημοσύνης (AI) [4-5].

2.4 Κατηγορίες Βιντεοπαιχνιδιών

Η ταξινόμηση των βιντεοπαιχνιδιών σε είδη βασίζεται κυρίως στους μηχανισμούς του παιχνιδιού, στη φύση της αλληλεπίδρασης του παίκτη με το εικονικό περιβάλλον και στις βασικές προκλήσεις του

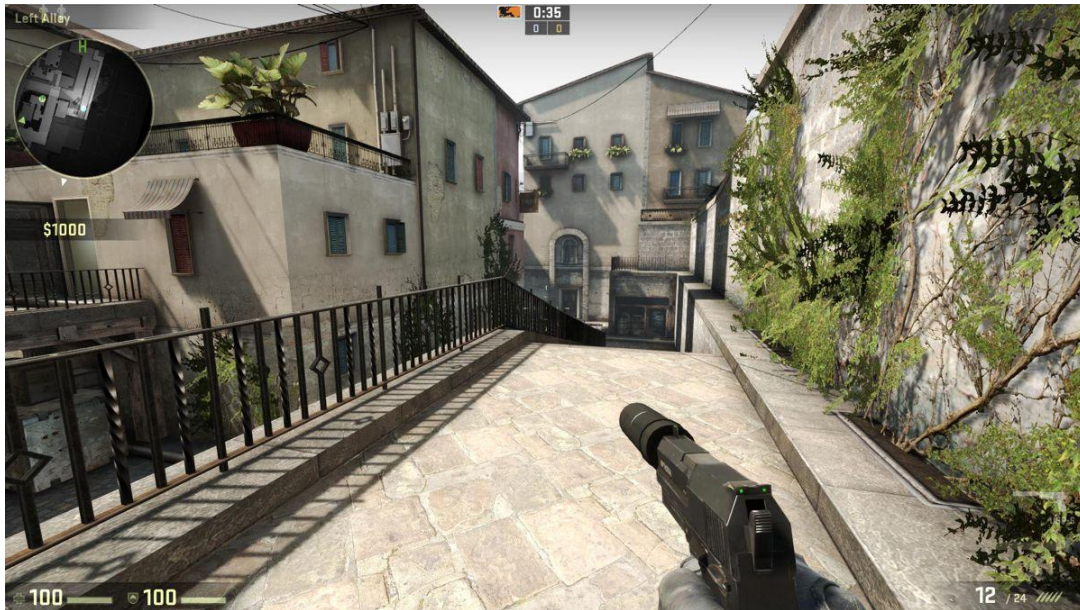
παιχνιδιού [7-8]. Αρχικά, μια θεμελιώδης και ιδιαίτερα σημαντική ταξινόμηση των βιντεοπαιχνιδιών βασίζεται στη διάσταση των γραφικών τους. Στις ευρέως αναγνωρισμένες κατηγορίες συγκαταλέγονται τα παιχνίδια δύο διαστάσεων (2D) και τριών διαστάσεων (3D). Τα τελευταία χρόνια, η πλειονότητα των σύγχρονων παιχνιδιών αναπτύσσεται σε 3D, καθώς προσφέρουν πολύ πιο εντυπωσιακά γραφικά και υψηλότερο βαθμό ρεαλισμού, ενώ παράλληλα προτιμώνται από τους χρήστες [8]. Με την τεχνολογική πρόοδο τα όρια μεταξύ των ειδών συχνά αμβλύνονται, οδηγώντας στη δημιουργία παιχνιδιών που δεν ανήκουν μόνο σε μία κατηγορία, αλλά περιέχουν στοιχεία από δύο ή περισσότερα είδη [9]. Παρ' όλα αυτά, μια τέτοια ταξινόμηση παραμένει σημαντική για τη μελέτη του σχεδιασμού παιχνιδιών και την κατανόηση της εμπειρίας του χρήστη [8].

Παρακάτω αναλύονται διεξοδικά οι κύριες κατηγορίες βιντεοπαιχνιδιών:

2.4.1 Παιχνίδια Δράσης – Action Games

Τα παιχνίδια δράσης συγκαταλέγονται στα παλαιότερα και πιο δημοφιλή είδη. Στα παιχνίδια αυτού του είδους, οι χρήστες απαιτείται να έχουν γρήγορα αντανακλαστικά και συντονισμό χεριού-ματιού και ακριβή χρονισμό [7]. Συνήθως, οι παίκτες έρχονται αντιμέτωποι με χρονικούς περιορισμούς, αντιπάλους ή επικίνδυνα εμπόδια εντός του περιβάλλοντος του παιχνιδιού, ή ακόμη και με άλλους χρήστες [8]. Το είδος περιλαμβάνει ένα φάσμα υποκατηγοριών, όπως παιχνίδια Fighting και Shooting games [7][9]. Η πρόοδος επιτυγχάνεται σταδιακά μέσω της αύξησης της δυσκολίας και της κατανόησης των μοτίβων συμπεριφοράς των αντιπάλων. Επιπλέον, πολλά παιχνίδια του είδους είναι ομαδικά και υποστηρίζουν διαδικτυακό παιχνίδι με άλλους παίκτες [8].

Στο πλαίσιο αυτό, τα παιχνίδια βολών (**Shooting**) αποτελούν μία από τις πιο δημοφιλείς και εμπορικά επιτυχημένες υποκατηγορίες των παιχνιδιών δράσης. Αν και βασίζονται στους βασικούς μηχανισμούς του είδους όπως οι γρήγορες αντιδράσεις, η απόκριση σε απειλές σε πραγματικό χρόνο και ο συντονισμός χεριού-ματιού διαφοροποιούνται από ένα βασικό χαρακτηριστικό: την υποκειμενική οπτική γωνία. Η πρώτη υποκατηγορία είναι η **First Person Perspective (FPS)**, όπου ο παίκτης βλέπει τον εικονικό κόσμο απευθείας από την οπτική γωνία του χαρακτήρα, βλέποντας συνήθως μόνο το όπλο ή τα χέρια του στο κάτω μέρος της οθόνης. Η δεύτερη είναι η **Third Person Perspective**, όπου η κάμερα τοποθετείται πίσω από τον χαρακτήρα, προσφέροντας ευρύτερη οπτική του χώρου, αλλά διατηρώντας διαφορετική δυναμική εμπειρίας. Η παρούσα πτυχιακή εργασία εστιάζει στην ανάπτυξη ενός 3D FPS παιχνιδιού, αξιοποιώντας τις προηγμένες δυνατότητες της μηχανής γραφικών Unity Engine για την υλοποίηση μηχανισμών σκόπευσης, κίνησης και αλληλεπίδρασης του παίκτη με το περιβάλλον. Κάποιοι από τους πιο γνωστούς τίτλους που αξίζει να αναφερθούν είναι οι εξής: Left 4 Dead, Call of Duty, αλλά και το θρυλικό Counter Strike (βλ. Εικόνα 2-3).



Εικόνα 2-3 Counter Strike

2.4.2 Παιχνίδια Ρόλων Role Playing Games – RPG

Τα παιχνίδια ρόλων βασίζονται στη σταδιακή ανάπτυξη των χαρακτήρων, στην αλληλεπίδραση με την πλοκή και στη λήψη αποφάσεων που επηρεάζουν την πορεία του παιχνιδιού. Ο παίκτης ελέγχει έναν ή περισσότερους πρωταγωνιστές και βελτιώνει τα χαρακτηριστικά τους όπως η μαχητική ισχύς, η άμυνα ή οι μαγικές ικανότητες συγκεντρώνοντας πόντους εμπειρίας, αγοράζοντας ή δημιουργώντας νέα αντικείμενα και ολοκληρώνοντας αποστολές (quests) που υπάρχουν εντός του εικονικού κόσμου [7-8]. Η κατηγορία διακρίνεται σε Action RPGs, Turn-Based RPGs και MMORPGs (Massively Multiplayer Online RPGs), στα οποία ένας μεγάλος αριθμός χρηστών συνυπάρχει και αλληλεπιδρά σε ένα κοινό ψηφιακό περιβάλλον [7][9]. Μερικά από τα πιο γνωστά παιχνίδια αυτής της κατηγορίας είναι το World of Warcraft (WoW) και το Lineage II (βλ. Εικόνα 2-4).



Εικόνα 2-4 World of Warcraft

2.4.3 Παιχνίδια Στρατηγικής Strategy Games

Σε αντίθεση με τα παιχνίδια που απαιτούν γρήγορα αντανακλαστικά, τα παιχνίδια στρατηγικής εστιάζουν σε πνευματικές προκλήσεις, στον μακροπρόθεσμο σχεδιασμό και στη σωστή διαχείριση πόρων [7-8]. Οι παίκτες καλούνται να αναπτύξουν υποδομές, να συγκεντρώσουν πόρους και να διοικήσουν στρατιωτικές μονάδες, προκειμένου να νικήσουν τους αντιπάλους τους. Το είδος αυτό χωρίζεται κυρίως σε παιχνίδια στρατηγικής πραγματικού χρόνου (**Real Time Strategy — RTS**), όπου οι ενέργειες εκτελούνται ταυτόχρονα, και σε παιχνίδια στρατηγικής με τη σειρά (**Turn Based Strategy — TBS**), όπου οι παίκτες παίζουν με τη σειρά τους [8-9]. Χαρακτηριστικά παραδείγματα αυτής της κατηγορίας αποτελούν οι τίτλοι Age of Empires και StarCraft II (βλ. Εικόνα 2-5).



Εικόνα 2-5 Age of Empires

2.4.4 Παιχνίδια Προσομοίωσης Simulation Games

Τα παιχνίδια αυτά βασίζονται στην όσο το δυνατόν πιο πιστή και ρεαλιστική απόδοση μιας συγκεκριμένης κατάστασης, έχοντας ως κοινό στοιχείο την προσομοίωση αληθινών ή υποθετικών κόσμων. Αναλύοντας την κατηγορία αυτή διακρίνονται οι εξής υποκατηγορίες: **Social Simulation Games**, **Vehicle Simulation** και **Construction and Management Simulation** [8][9]. Αρχικά στα **Social Simulation Games** εστίαση δίνεται κυρίως στις κοινωνικές αλληλεπιδράσεις του χαρακτήρα που ελέγχει ο παίκτης είτε με άλλους χρήστες σε διαδικτυακή μορφή, είτε με προκατασκευασμένους χαρακτήρες του παιχνιδιού εντός δικτύου ενώ σε ορισμένες περιπτώσεις ο χρήστης βιώνει μια εικονική ζωή λαμβάνοντας αποφάσεις και επιλέγοντας ελεύθερα τις ενέργειές του [7][8]. Στη δεύτερη περίπτωση, στα **Vehicle Simulation**, παρέχεται στον παίκτη η δυνατότητα να χειριστεί μέσα από το σπίτι του πράγματα που στην πραγματικότητα θα ήταν δύσκολα προσβάσιμα όπως για παράδειγμα ένα αεροπλάνο [8]. Τέλος στα παιχνίδια **Construction and Management Simulation**, ο παίκτης καλείται να σχεδιάσει να κατασκευάσει και να διαχειριστεί πόλεις ή κοινότητες. Χαρακτηριστικά παραδείγματα παιχνιδιών προσομοίωσης αποτελούν το The Sims, το Theme Park και το Microsoft Flight Simulator το οποίο συγκεντρώνει ιδιαίτερα μεγάλο ενδιαφέρον από την κοινότητα των βιντεοπαιχνιδιών (βλ. Εικόνα 2-6).

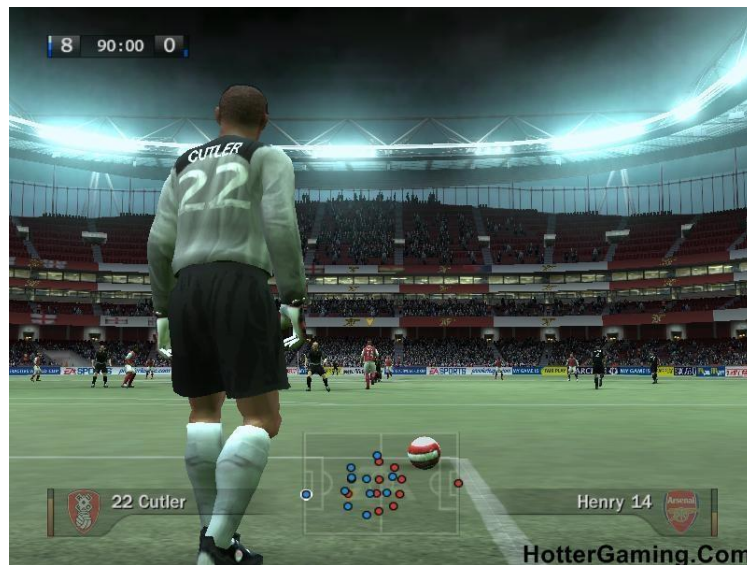


Εικόνα 2-6 Microsoft Flight Simulator

2.4.5 Παιχνίδια Αθλητισμού και Αγώνων Sports & Racing Games

Στην τελευταία κατηγορία ανήκουν τα αθλητικά παιχνίδια, τα οποία προσομοιώνουν διάφορα αθλήματα. Ο παίκτης μπορεί να αναμετρηθεί είτε με άλλους παίκτες είτε με αντιπάλους που ελέγχονται από το σύστημα. Ένα βασικό χαρακτηριστικό που τα διαφοροποιεί από άλλες κατηγορίες είναι το ρεαλιστικό και οικείο περιβάλλον τους, το οποίο περιέχει στοιχεία από τον πραγματικό κόσμο. Ο παίκτης αναλαμβάνει τον έλεγχο είτε ενός μεμονωμένου αθλητή είτε ολόκληρης της ομάδας, ενώ σε ορισμένους τίτλους υιοθετεί το ρόλο του προπονητή ή διοικητικού στελέχους και προσπαθεί να διαχειριστεί την απόδοση του συλλόγου. Ο αντικειμενικός σκοπός διαφοροποιείται ανάλογα με τους

κανόνες του εκάστοτε αθλήματος [7-8], με χαρακτηριστικά παραδείγματα τους τίτλους FIFA και NBA 2K (βλ. Εικόνα 2-7).



Εικόνα 2-7 Fifa

2.5 Επίλογος

Σε αυτό το κεφάλαιο εξετάσαμε την εξέλιξη των βιντεοπαιχνιδιών, πώς από απλά εργαστηριακά πειράματα έφτασαν σε μια παγκόσμια βιομηχανία ψυχαγωγίας. Η ιστορική πορεία της ανάπτυξής τους, από τα πρώτα πειραματικά στάδια έως τα σύγχρονα φωτορεαλιστικά τρισδιάστατα γραφικά, καταδεικνύει τη συνεχή τεχνολογική πρόοδο και τη δημιουργική καινοτομία.

Οι κατηγορίες βιντεοπαιχνιδιών καθορίζουν σε σημαντικό βαθμό τις βασικές λειτουργίες και τον τρόπο υλοποίησης τους. Τα παιχνίδια δράσης εστιάζουν στα αντανakλαστικά και απαιτούν γρήγορη ανταπόκριση, τα παιχνίδια ρόλων υλοποιούν συστήματα ανάπτυξης χαρακτήρων, τα παιχνίδια στρατηγικής απαιτούν μηχανισμούς διαχείρισης πόρων, τα παιχνίδια προσομοίωσης αναπαριστούν ρεαλιστικά σενάρια και τα αθλητικά παιχνίδια υλοποιούν αθλητικές μηχανικές. Ιδιαίτερα, τα παιχνίδια FPS, ένα από τα κυρίαρχα είδη, συνδυάζουν γρήγορο ρυθμό δράσης με περίπλοκα συστήματα σκόπευσης, κίνησης και αλληλεπίδρασης.

Κεφάλαιο 3ο: Θεωρητικό Υπόβαθρο

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα εξετάσουμε τις τεχνολογίες που χρησιμοποιούνται σήμερα για τη δημιουργία βιντεοπαιχνιδιών ξεκινώντας με τα τρισδιάστατα γραφικά και τη φυσική πίσω από αυτά που αποτελεί βάση για κάθε ρεαλιστική αναπαράσταση ενός παιχνιδιού. Εκεί δηλαδή όπου παίρνουν πνοή τα μοντέλα, τα χρώματα, οι σκιές και οι κινήσεις των αντικειμένων. Στη συνέχεια θα αναλύσουμε το πως η τεχνητή νοημοσύνη συνδέεται με το gaming εξηγώντας πώς οι χαρακτήρες που δεν ελέγχονται από τον παίκτη μπορούν να συμπεριφέρονται με αυτόνομο τρόπο που να φαίνεται φυσικός. Ακολουθεί παρουσίαση των πιο δημοφιλών μηχανών παιχνιδιών. Τέλος γίνεται μία αναφορά στις ομάδες ανάπτυξης βιντεοπαιχνιδιών καθώς και στους ρόλους και τις ειδικότητες που συνεργάζονται για την υλοποίηση αυτού.

3.2 Τεχνολογίες 3D Graphics & Physics

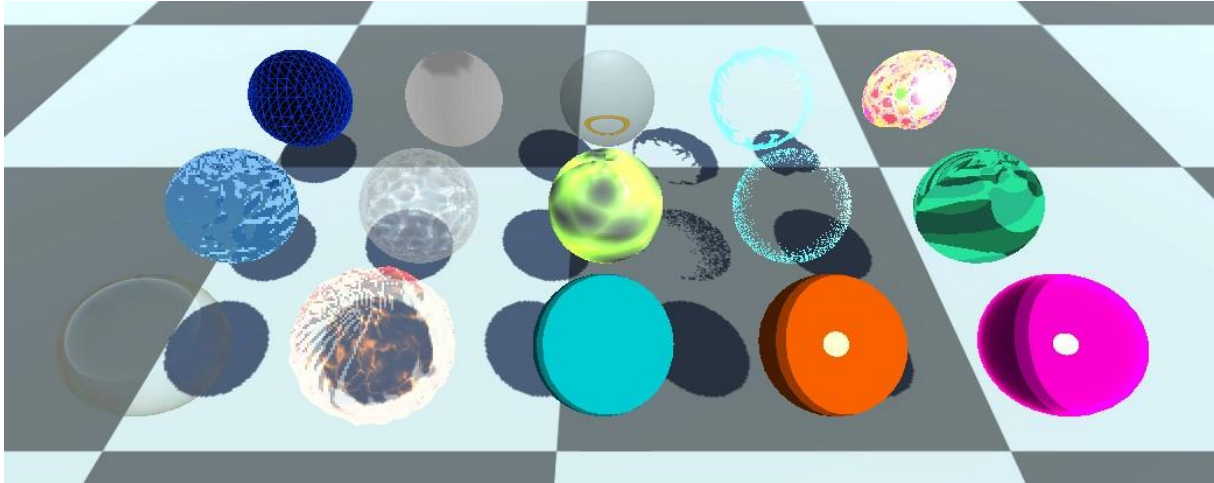
Οι τεχνολογίες τρισδιάστατης γραφικής και φυσικής προσομοίωσης διαθέτουν δύο κύρια και βασικά χαρακτηριστικά που προσδίδουν μια αίσθηση πραγματικότητας στα σύγχρονα παιχνίδια. Αυτές οι τεχνολογίες λειτουργούν συντονισμένα: τα γραφικά δείχνουν τον κόσμο στον παίκτη, ενώ η φυσική προσομοίωση διασφαλίζει ότι τα αντικείμενα συμπεριφέρονται με ρεαλιστικό τρόπο [13-15].

3.2.1 Τρισδιάστατη Γραφική και Rendering Pipeline

Τα τρισδιάστατα γραφικά δημιουργούνται μέσω μιας διαδικασίας που ονομάζεται **rendering pipeline**. Με απλά λόγια, αυτή η διαδικασία παίρνει τα τρισδιάστατα μοντέλα που δημιούργησαν οι καλλιτέχνες και τα μετατρέπει σε δισδιάστατες εικόνες που εμφανίζονται στην οθόνη [14]. Τα γεωμετρικά στοιχεία των 3D μοντέλων αποτελούνται από κορυφές (vertices) πιο απλά, σημεία στον χώρο οι οποίες συνδέονται για να σχηματίσουν τρίγωνα [14][16]. Κάθε κορυφή περιέχει πληροφορίες, όπως το χρώμα, τη κανονική διεύθυνση (η οποία καθορίζει πώς το φως αλληλοεπιδρά με την επιφάνεια) και τις UV συντεταγμένες (που χρησιμοποιούνται για την τοποθέτηση υφών πάνω στο μοντέλο) [16].

Οι **shaders** είναι ειδικά προγράμματα που εκτελούνται στη μονάδα επεξεργασίας γραφικών (**Graphics Processing Unit GPU**) και καθορίζουν πώς θα φαίνεται κάθε αντικείμενο [14-15]. Ελέγχουν πολλές οπτικές ιδιότητες, όπως το χρώμα, την υφή, την αντανάκλαση, τη διαφάνεια και τον τρόπο που τα αντικείμενα αντιδρούν στο φως. Υπάρχουν διαφορετικοί τύποι shaders, ο καθένας για διαφορετικό στάδιο της διαδικασίας: οι vertex shaders χειρίζονται τη γεωμετρία του μοντέλου, ενώ τα pixel shaders υπολογίζουν το τελικό χρώμα κάθε pixel [14].

Στην Εικόνα 3-1 παρουσιάζονται ενδεικτικά διάφοροι τύποι shaders και τα οπτικά αποτελέσματά τους.



Εικόνα -1 Shaders Collection

Η **διαδικασία της υφής texturing** είναι ουσιαστικά η εφαρμογή δισδιάστατων εικόνων πάνω στα τρισδιάστατα μοντέλα, προκειμένου να προσθέσουμε λεπτομέρεια. Αντί να δημιουργούμε εξαιρετικά πολύπλοκη γεωμετρία κάτι που θα ήταν υπολογιστικά ακριβό και θα επιβάρυνε τις κονσόλες ή τους υπολογιστές χωρίς ουσιαστικό όφελος — χρησιμοποιούμε υφές για να προσθέσουμε λεπτομέρεια και ρεαλισμό [14] (βλ. Εικόνα 3-2).



Εικόνα 3-2 Textures

3.2.2 Physics Engines και Collision Detection

Η **μηχανή φυσικής physics engine**, είναι ένα λογισμικό που προσομοιώνει τη φυσική συμπεριφορά των αντικειμένων, όπως τις δυνάμεις, την βαρύτητα, τις συγκρούσεις [13][15]. Οι μηχανές φυσικής είναι απαραίτητες στα παιχνίδια, γιατί χάρη σε αυτές οι κινήσεις και οι αλληλεπιδράσεις φαίνονται ρεαλιστικές [15].

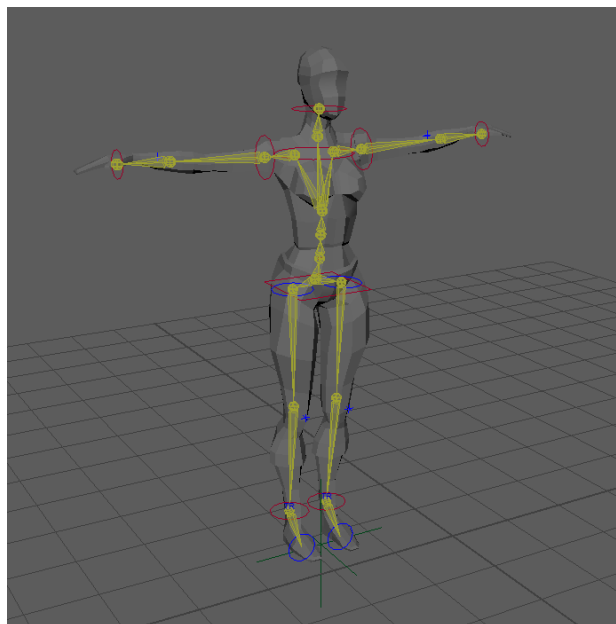
Ως **ανίχνευση συγκρούσεων collision detection**, ορίζεται η διαδικασία που καθορίζει πότε δύο αντικείμενα στον εικονικό κόσμο έρχονται σε επαφή [13][15]. Αυτό γίνεται σε δύο στάδια, στο πρώτο στάδιο η φάση που αποκαλείται **broadphase**, η μηχανή φυσικής πραγματοποιεί έναν γρήγορο και υπολογιστικά «φτηνό» έλεγχο, χρησιμοποιώντας απλά γεωμετρικά σχήματα που περιβάλλουν χονδρικά τα αντικείμενα. Με αυτόν τον τρόπο αποκλείονται γρήγορα όσα ζευγάρια αντικειμένων με σιγουριά δεν συμμετέχουν στη σύγκρουση, χωρίς να χρειαστεί να εξεταστεί η ακριβής γεωμετρία τους. Στη συνέχεια, η δεύτερη φάση, που ονομάζεται **narrowphase**, εκτελεί ακριβέστερους και πιο απαιτητικούς υπολογισμούς μόνο για τα ζευγάρια που πιθανώς συγκρούονται, προσδιορίζοντας με ακρίβεια το σημείο επαφής [13][15].

Η **δυναμική άκαμπτου σώματος rigid body dynamics**, προσομοιώνει τη συμπεριφορά σκληρών αντικειμένων που έχουμε ορίσει ότι δεν μπορούν να παραμορφωθούν [13][15]. Σε αυτήν τη διαδικασία, η μηχανή φυσικής παρακολουθεί τη θέση, την ταχύτητα και την περιστροφή κάθε αντικειμένου, εφαρμόζοντας φυσικές δυνάμεις όπως η βαρύτητα και υπολογίζοντας πώς κινούνται [15]. Υπάρχουν διαφορετικοί τύποι σωμάτων: τα **StaticBody** αντικείμενα δεν κινούνται, όπως οι τοίχοι, τα **RigidBody** αντικείμενα κινούνται και αντιδρούν στις δυνάμεις, ενώ τα **KinematicBody** αντικείμενα κινούνται από τη λογική του παιχνιδιού και όχι από τη φυσική [13][15]. Όταν συμβαίνει μια σύγκρουση, η μηχανή φυσικής εφαρμόζει μια ώθηση, ή *impulse*, μια ξαφνική αλλαγή στην ταχύτητα, για να χωρίσει τα αντικείμενα, όπως ακριβώς θα γινόταν και στον πραγματικό κόσμο.

3.2.3 3D Animation Systems & Rigging

Το τρισδιάστατο animation στα βιντεοπαιχνίδια είναι η διαδικασία με την οποία προσθέτουμε κίνηση σε τρισδιάστατα μοντέλα, δίνοντας ζωή στους χαρακτήρες και το περιβάλλον [19].

Rigging & Skeletal Animation: Πρόκειται για την τεχνική κατά την οποία δημιουργείται ένας εσωτερικός σκελετός (**bones/joints**) κάτω από το τρισδιάστατο πλέγμα (**mesh**) του μοντέλου. Όταν τα οστά κινούνται, το πλέγμα προσαρμόζεται ανάλογα, επιτρέποντας φυσικές κινήσεις στους χαρακτήρες [19] (βλ. Εικόνα 3-3).



Εικόνα 3-3 εσωτερικός σκελετός

Animator State Machines: Η διαχείριση των animations γίνεται μέσω μηχανών καταστάσεων Animation State Machines. Το σύστημα περνάει από το ένα animation clip στο άλλο (π.χ. από Idle σε Run ή Attack) ανάλογα με παραμέτρους του παιχνιδιού, όπως η ταχύτητα ή κάποιο γεγονός (trigger) [19].

Animation Events & Audio Sync: Είναι ένας μηχανισμός που επιτρέπει την εκτέλεση κώδικα ή την αναπαραγωγή ήχου σε συγκεκριμένα καρέ frames ενός animation. Για παράδειγμα μπορεί να αναπαράγει τον ήχο ενός βήματος τη στιγμή που το πόδι του χαρακτήρα αγγίζει το έδαφος [19].

3.3 Τεχνητή νοημοσύνη -AI και βιντεοπαιχνίδια

Η τεχνητή νοημοσύνη (AI) είναι ένα από τα πιο σημαντικά στοιχεία των σύγχρονων βιντεοπαιχνιδιών, καθώς είναι αυτή που δίνει αληθοφανή συμπεριφορά στους χαρακτήρες που δεν ελέγχει ο παίκτης όπως για παράδειγμα τους εχθρούς, τους συμμάχους και τους ουδέτερους χαρακτήρες, γνωστούς ως **Non-Playable Characters (NPC)**. Εξαιτίας του AI, οι χαρακτήρες δεν είναι απλά διακοσμητικά στοιχεία, αλλά αντιδρούν στις κινήσεις του παίκτη, τον ακολουθούν, του επιτίθενται ή συνεργάζονται μαζί του, δημιουργώντας την ψευδαίσθηση ότι είναι ζωντανοί.

Σε αυτήν την ενότητα θα εξετάσουμε δύο βασικές πτυχές της AI και ποιος είναι ο ρόλος τους στα σύγχρονα παιχνίδια. Η πρώτη είναι η **συμπεριφορική τεχνητή νοημοσύνη, ή Behavioral AI**, δηλαδή πώς οι χαρακτήρες λαμβάνουν αποφάσεις αυτόνομα κατά τη διάρκεια του παιχνιδιού. Η δεύτερη αφορά την **πλοήγηση** εντός του κόσμου του παιχνιδιού, δηλαδή πώς οι χαρακτήρες κινούνται στον χώρο, βρίσκοντας τον δρόμο τους χωρίς να προσκρούουν σε εμπόδια.

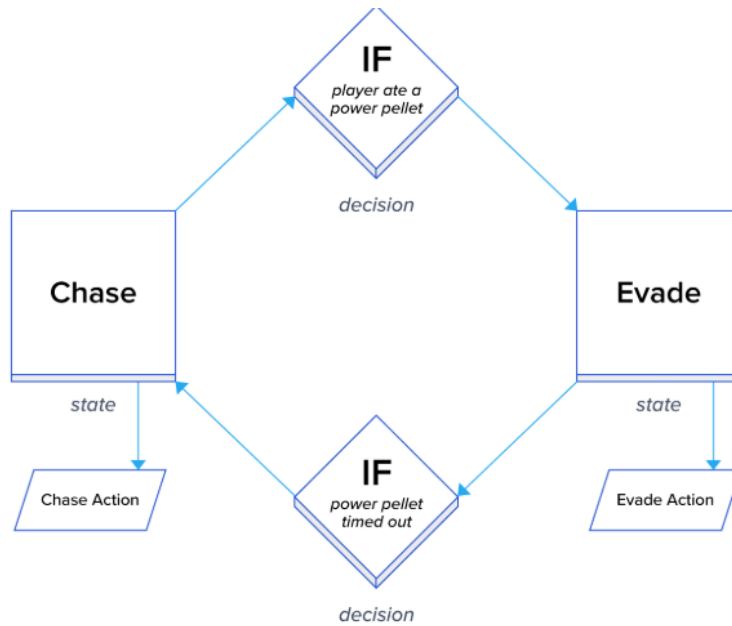
3.3.1 Συμπεριφορική τεχνητή νοημοσύνη - Behavioral AI

Η **συμπεριφορική τεχνητή νοημοσύνη, ή Behavioral AI**, είναι ένα στοιχείο που καθορίζει τις ενέργειες των χαρακτήρων που δεν ελέγχονται από παίκτης. Με άλλα λόγια, καθορίζει πώς αντιδρούν σε αυτά που συμβαίνουν γύρω τους — αν δηλαδή θα επιτεθούν, θα τραπούν σε φυγή, θα περιμένουν ή θα ακολουθήσουν κάποια άλλη πορεία δράσης.

Οι **μηχανές πεπερασμένων καταστάσεων (Finite State Machines — FSM)** αποτελούν μία από τις πιο διαδεδομένες και αποτελεσματικές μεθόδους για την υλοποίηση AI σε παιχνίδια [17]. Η λειτουργία τους γίνεται με βάση ένα σύνολο προκαθορισμένων καταστάσεων (states), όπως Idle, Chase ή Attack, όπου ο χαρακτήρας μπορεί να βρίσκεται σε μία μόνο κατάσταση κάθε φορά [17]. Οι μεταβάσεις μεταξύ των καταστάσεων πραγματοποιούνται όταν πληρούνται συγκεκριμένες συνθήκες (triggers/events), όπως για παράδειγμα η απόσταση του παίκτη ή η μείωση της υγείας του χαρακτήρα [17][19].

Τα **δέντρα αποφάσεων (decision trees)** αποτελούν μία ιεραρχική δομή όπου κάθε εσωτερικός κόμβος αντιπροσωπεύει μια δοκιμή ή συνθήκη (π.χ. «Είναι η υγεία < 20%;») και κάθε φύλλο αντιπροσωπεύει μια ενέργεια [17][18]. Σε αντίθεση με τις πεπερασμένες μηχανές καταστάσεων, η προσέγγιση αυτή προσφέρει μεγαλύτερη ευελιξία στην οργάνωση σύνθετων κανόνων συμπεριφοράς, καθώς επιτρέπει τη δημιουργία περισσότερων διακλαδώσεων και πιο λεπτομερών ελέγχων [17].

Στην Εικόνα 3-4 παρουσιάζονται τα βασικά συστατικά μιας μηχανής πεπερασμένων καταστάσεων (FSM) και οι μεταβάσεις μεταξύ των καταστάσεων.



Εικόνα 3-4 Components of FSM

3.3.2 Navigation AI

Η ικανότητα των εικονικών χαρακτήρων να κινούνται αυτόνομα στον χώρο, αποφεύγοντας εμπόδια και βρίσκοντας τη βέλτιστη διαδρομή, είναι κρίσιμη για την προσθήκη μιας ρεαλιστικής αίσθησης στο παιχνίδι. Αυτό επιτυγχάνεται χρησιμοποιώντας δύο βασικά εργαλεία: τον **αλγόριθμο A* (A-Star)** και το **Navigation Mesh**.

Ο **αλγόριθμος A* (A-Star)** αποτελεί τη βάση της πλοήγησης στα παιχνίδια, αναζητά τη συντομότερη διαδρομή ανάμεσα σε σημεία, συνδυάζοντας την πραγματική απόσταση που έχει ήδη διανύσει το NPC με μια εκτίμηση για την απόσταση που απομένει μέχρι τον στόχο [17][19].

Το **NavMesh** είναι μια τρισδιάστατη γεωμετρική δομή που τοποθετείται πάνω από το περιβάλλον του παιχνιδιού και καθορίζει ποιες περιοχές είναι προσβάσιμες τόσο για τους κύριους όσο και για τους δευτερεύοντες χαρακτήρες εντός του παιχνιδιού. Αντί να υπολογίζει διαδρομές σε μεμονωμένα σημεία, ο αλγόριθμος **pathfinding** κινείται πάνω στις επιφάνειες του NavMesh, με αποτέλεσμα οι υπολογισμοί να γίνονται γρήγορα και η κίνηση να είναι ομαλή και φυσική [17][19] (βλ. Εικόνα 3-5).



Εικόνα 3-5 Navigation Mesh όπως φαίνεται μέσα στο Unity

3.4 Μηχανές δημιουργίας βιντεοπαιχνιδιών – Game Engine

Η επιλογή της κατάλληλης μηχανής δημιουργίας παιχνιδιών είναι μία από τις πιο σημαντικές αποφάσεις που καλείται να πάρει οποιοσδήποτε θέλει να αναπτύξει ένα παιχνίδι. Συχνά απαιτείται λεπτομερής έρευνα για να διαπιστωθεί αν η μηχανή προσφέρει τις υπηρεσίες και τους πόρους που θα χρειαστούν κατά την ανάπτυξη. Σημαντικά κριτήρια επιλογής αποτελούν το είδος του παιχνιδιού που πρόκειται να δημιουργηθεί, καθώς και ο αριθμός των διαστάσεων (2D ή 3D) που θα περιέχει. Στις περιπτώσεις όπου η ομάδα ανάπτυξης δεν διαθέτει προηγούμενη εμπειρία, είναι σημαντικό να επιλεγεί μια σχετικά απλή μηχανή, η οποία θα προσφέρει παράλληλα ποικίλες γνώσεις πάνω στο αντικείμενο. Στη συνέχεια θα παρουσιαστούν οι πιο γνωστές μηχανές δημιουργίας παιχνιδιών, αναφέροντας για κάθε μία πλεονεκτήματα και μειονεκτήματα που είναι χρήσιμο να γνωρίζουν οι υποψήφιοι χρήστες. Συγκεκριμένα, θα εξεταστούν οι εξής μηχανές: Unreal Engine, Godot και Unity.

3.4.1 Τι είναι Game Engine

Μια μηχανή παιχνιδιού (Game Engine) είναι ένα ολοκληρωμένο πλαίσιο λογισμικού (software framework) σχεδιασμένο για τη δημιουργία και την ανάπτυξη βιντεοπαιχνιδιών [19]. Λειτουργεί ως το ενδιάμεσο στρώμα λογισμικού (middleware) που γεφυρώνει το υλικό (hardware) του υπολογιστή ή της κονσόλας με το περιεχόμενο του παιχνιδιού [19].

Αντί οι προγραμματιστές να δημιουργούν από το μηδέν βασικές λειτουργίες για κάθε νέο έργο, η μηχανή παιχνιδιού παρέχει ένα ευρύ σύνολο επαναχρησιμοποιούμενων δομικών στοιχείων και συστημάτων [19]. Τα κύρια υποσυστήματα που συνθέτουν μια σύγχρονη Game Engine είναι τα εξής:

Rendering Engine (Μηχανή Σχεδίασης): Υπεύθυνη για την παραγωγή 2D ή 3D γραφικών, τη διαχείριση φωτισμού, σκιών, υφών (textures) και εφέ στην οθόνη [19].

Physics Engine (Μηχανή Φυσικής): Υπολογίζει τις δυνάμεις, τη βαρύτητα, τις κινήσεις των σωμάτων και την ανίχνευση συγκρούσεων collision detection στο εικονικό περιβάλλον [19].

Audio Engine (Μηχανή Ήχου): Διαχειρίζεται την αναπαραγωγή μουσικής ηχητικών εφέ τρισδιάστατου χωρικού ήχου spatial audio και φωνητικών [19].

Animation System (Σύστημα Animation): Ελέγχει την κίνηση των 3D μοντέλων, τις μεταβάσεις μεταξύ animations και το rigging των σκελετών [19].

Scripting / Logic Interface: Πρόκειται για το υποσύστημα που επιτρέπει στους προγραμματιστές να ορίζουν τη συμπεριφορά των αντικειμένων, τους κανόνες του παιχνιδιού game logic, την αλληλεπίδραση του χρήστη και την τεχνητή νοημοσύνη. Οι σύγχρονες μηχανές παρέχουν υποστήριξη μέσω γλωσσών προγραμματισμού υψηλού επιπέδου όπως C# ή C++ ή συστημάτων οπτικού προγραμματισμού visual scripting [19].

Asset Management & Scene Editor: Αποτελεί το οπτικό περιβάλλον εργασίας της μηχανής, το οποίο επιτρέπει την εισαγωγή, την οργάνωση και τη διαχείριση των ψηφιακών πόρων assets, όπως 3D μοντέλα, υφές, ηχητικά αρχεία και animations [19]. Παράλληλα ο Scene Editor παρέχει εργαλεία χωρικής τοποθέτησης αντικειμένων, διαμόρφωσης terrain καθώς και ρύθμισης κάμερας και προσχεδιασμού φωτισμού [19]. Σημαντικό στοιχείο αποτελεί η χρήση επαναχρησιμοποιήσιμων προτύπων αντικειμένων (Prefabs), τα οποία επιτρέπουν τη δυναμική αναπαραγωγή (instantiation) στοιχείων κατά τη διάρκεια της εκτέλεσης [19].

3.4.2 Κύριες Μηχανές Παιχνιδιών (Game Engines)

Στη σύγχρονη βιομηχανία ανάπτυξης βιντεοπαιχνιδιών κυριαρχούν λίγες εξαιρετικά εξελιγμένες μηχανές παιχνιδιών, καθεμία από τις οποίες στοχεύει σε διαφορετικές ανάγκες παραγωγής, κλίμακες έργων indie έως AAA και τεχνικές απαιτήσεις. Οι πιο δημοφιλείς και ευρέως χρησιμοποιούμενες μηχανές σήμερα είναι οι εξής.

Unity: Είναι μια εξαιρετικά δημοφιλής μηχανή παιχνιδιών που αναπτύχθηκε από τη Unity Technologies και υποστηρίζει πολλές πλατφόρμες cross-platform. Ξεχωρίζει για την ευκολία χρήσης της, τη γλώσσα C# και την ικανότητά της να δημιουργεί εξίσου καλά 2D και 3D παιχνίδια για PC όσο και για κονσόλες ή κινητά ακόμη υποστηρίζει VR/AR. Διαθέτει ένα τεράστιο οικοσύστημα εργαλείων μέσω του Unity Asset Store και μια μεγάλη κοινότητα προγραμματιστών, γεγονός που την καθιστά την πρώτη επιλογή για ανεξάρτητους δημιουργούς indie developers αλλά και για μεσαία στούντιο [21].

Unreal Engine: Αναπτύχθηκε από την Epic Games και αποτελεί το βιομηχανικό πρότυπο για μεγάλες παραγωγές AAA [19]. Προσφέρει εξαιρετική οπτική ποιότητα, προηγμένα συστήματα φωτισμού σε πραγματικό χρόνο (Lumen, Nanite) και βασίζεται στη C++ καθώς και στο σύστημα οπτικού προγραμματισμού Blueprints [19]. Αν και είναι εξαιρετικά ισχυρή έχει πιο απότομη καμπύλη εκμάθησης και απαιτεί περισσότερους πόρους από τον υπολογιστή.

Godot Engine: Είναι μια ελεύθερη, ανοικτού κώδικα open-source μηχανή που έχει αποκτήσει μεγάλη δημοτικότητα τα τελευταία χρόνια. Είναι ελαφριά, διαθέτει δική της γλώσσα προγραμματισμού GDScript, παρόμοια με Python και υποστηρίζει και C#. Είναι ιδανική για 2D παιχνίδια και ελαφριά 3D έργα, προσφέροντας πλήρη ελευθερία σε indie developers και μικρές ομάδες που θέλουν ελευθερία χωρίς licensing περιορισμούς [22].

3.5 Ομάδα Ανάπτυξης Βιντεοπαιχνιδιών

Για να θεωρείται επιτυχημένη μία εταιρεία ανάπτυξης παιχνιδιών θα πρέπει να διαθέτει μία ολοκληρωμένη και έμπειρη ομάδα ανάπτυξης στην οποία κάθε μέλος συνεισφέρει βάσει της εξειδίκευσης και των δεξιοτήτων του [12]. Το μέγεθος δεν είναι πάντα το ίδιο υπάρχουν μικρές ανεξάρτητες ομάδες λίγων ατόμων έως μεγάλα στούντιο που αποτελούνται από εκατοντάδες ειδικούς τεχνικούς [11]. Ανεξάρτητα από το μέγεθος η βασική δομή είναι πάντα η ίδια και αποτελείται από τον: **παραγωγό, σχεδιαστή παιχνιδιού, τους προγραμματιστές, τους καλλιτέχνες, τους σχεδιαστές ήχου και τους δοκιμαστές διασφάλισης ποιότητας** [10][12]. Όλοι η ομάδα εργάζεται με στόχο το καλύτερο δυνατό αποτέλεσμα.

3.5.1 Σχεδιαστής-Game Designer

Ο σχεδιαστής παιχνιδιών διαμορφώνει τη συνολική δομή και το περιεχόμενο της εμπειρίας. Καθορίζει πώς αλληλοεπιδρούν οι παίκτες με το παιχνίδι, ποιες είναι οι συνθήκες νίκης ή ήττας, αλλά και πώς κλιμακώνεται σταδιακά η δυσκολία [10-12]. Στην πράξη, η δουλειά του είναι να μετατρέπει τις αρχικές ιδέες σε λεπτομερείς λειτουργικούς μηχανισμούς, ορίζοντας σαφείς στόχους, προκλήσεις μέσα στον ψηφιακό κόσμο και διαθέσιμους πόρους. Παράλληλα, θέτει τους βασικούς κανόνες που διέπουν τις ενέργειες του παίκτη και συνθέτει τα αφηγηματικά στοιχεία, δημιουργώντας το κατάλληλο πλαίσιο ώστε να προσφέρει στον χρήστη μια ολοκληρωμένη, πλήρης και μοναδική εμπειρία [11].

3.5.2 Προγραμματιστή-Programmer

Οι προγραμματιστές αποτελούν τον βασικό πυλώνα της ανάπτυξης ενός παιχνιδιού, καθώς μετατρέπουν τις ιδέες σε λειτουργικές και πραγματικές εμπειρίες [10][12]. Είναι υπεύθυνοι για την υλοποίηση των μηχανισμών του παιχνιδιού, των βασικών συστημάτων, του ελέγχου των χαρακτήρων και των στοιχείων αλληλεπίδρασης. Παράλληλα, έχουν υποχρέωση να φέρουν την ισορροπία ανάμεσα στην τεχνική αρτιότητα και τη βελτιστοποίηση της απόδοσης, διασφαλίζοντας έτσι την ομαλή ροή του παιχνιδιού. Για να επιτευχθεί το επιθυμητό αποτέλεσμα, εργάζονται σε στενή συνεργασία με τους σχεδιαστές, εξελίσσοντας και βελτιώνοντας όσο το δυνατόν περισσότερο την εμπειρία του παίκτη [11]. Τέλος, αξίζει να σημειωθεί ότι οι δύο πιο δημοφιλείς γλώσσες δημιουργίας παιχνιδιών είναι η C++ και η C#.

3.5.3 Καλλιτέχνες -Artist

Ο καλλιτέχνης είναι υπεύθυνος για την εικόνα και την ατμόσφαιρα του παιχνιδιού [10][12]. Στη σύγχρονη εποχή, χρησιμοποιούνται προγράμματα όπως το Blender και το Maya. Ο καλλιτέχνης παίρνει τις αρχικές ιδέες και τις μετατρέπει σε τρισδιάστατους χαρακτήρες, αντικείμενα και περιβάλλοντα. Στη συνέχεια, προσθέτει στα μοντέλα υφές και χρώματα, δίνοντάς τους λεπτομέρεια και μια πιο αληθοφανή αίσθηση. Τέλος, όλα αυτά ενσωματώνονται στο παιχνίδι, με αποτέλεσμα να συνθέτουν τον οπτικό κόσμο που βλέπει και βιώνει ο παίκτης.

3.5.4 Μηχανικός / Σχεδιαστής Ήχου-Sound Engineer

Ο σχεδιαστής ήχου δημιουργεί ηχητικά εφέ και ηχητικά περιβάλλοντα [11][12]. Επιλέγει στοιχεία από βιβλιοθήκες ήχου ή ηχογραφεί και συνθέτει μόνος του δικούς του ήχους, προκειμένου να διαμορφώσει την ηχητική ατμόσφαιρα μέσα στον ψηφιακό κόσμο. Συνδυάζει ήχους από το περιβάλλον, δυναμικούς ήχους χαρακτήρων και ηχητικά εφέ. Το αποτέλεσμα είναι μια πιο ρεαλιστική και αληθοφανής εμπειρία, ενώ παράλληλα μπορεί να προσφέρει στον παίκτη ανατροφοδότηση κατά τη διάρκεια του παιχνιδιού.

3.5.5 Διαχειριστής Έργου-Project Manager

Ο ρόλος του παραγωγού ή υπεύθυνου έργου είναι να συντονίζει και να επιβλέπει τη διαδικασία ανάπτυξης του παιχνιδιού [10-12]. Στις αρμοδιότητές του περιλαμβάνονται ο σχεδιασμός, η διαχείριση του χρόνου, η ομαλή συνεργασία, καθώς και ο συντονισμός των ομάδων. Αυτό επιτυγχάνεται χρησιμοποιώντας εργαλεία διαχείρισης έργων. Παρακολουθεί την πρόοδο των εργασιών, επιλύει τυχόν προβλήματα εντός των ομάδων και διασφαλίζει την τήρηση των προθεσμιών. Στόχος του είναι να επιτύχει μια ισορροπία ανάμεσα στο όραμα της ομάδας και τους πρακτικούς περιορισμούς, όπως ο προϋπολογισμός και τα χρονοδιαγράμματα του έργου [11].

3.5.6 Δοκιμαστής-Tester

Ένας ακόμη εξίσου σημαντικός ρόλος είναι του δοκιμαστή (tester), ο οποίος είναι υπεύθυνος για τη σταθερότητα και την ομαλή λειτουργία του παιχνιδιού [10-12]. Μέσω ενδελεχών και επαναλαμβανόμενων δοκιμών, στόχος του είναι να εντοπίζει τυχόν σφάλματα στο λογισμικό, δυσλειτουργίες ή λάθη στους μηχανισμούς και προβλήματα απόδοσης. Στη συνέχεια, ο ειδικός καταγράφει τα ευρήματα αυτά σε συστήματα παρακολούθησης σφαλμάτων, ώστε να μπορούν να επιλυθούν από τους προγραμματιστές. Έτσι, δημιουργείται μια διαδικασία ανατροφοδότησης, επιπλέον αξιολογεί τη συνολική εμπειρία του παίκτη και κατά πόσο αυτή είναι εύχρηστη, και επαληθεύει τη συμμόρφωση του παιχνιδιού με τις τεχνικές προδιαγραφές στις διάφορες πλατφόρμες (PC, κονσόλες, φορητές συσκευές), διασφαλίζοντας την υψηλή ποιότητα και την άρτια τελική μορφή του προϊόντος πριν από την κυκλοφορία του προς τους χρήστες.

Καταλαβαίνουμε λοιπόν, ότι η επιτυχία του έργου εξαρτάται από την άψογη συνεργασία μιας ισχυρής και ενθουσιώδους ομάδας, καθώς και από την αποτελεσματική επικοινωνία και τον σαφή συντονισμό μεταξύ των μελών της. Από τον σχεδιασμό και τη μηχανική ήχου έως τη διαχείριση του έργου και τον ποιοτικό έλεγχο, κάθε ρόλος αποτελεί ζωτικό στοιχείο της δημιουργικής διαδικασίας. Αν και η σύνθεση και το μέγεθος της ομάδας ποικίλλουν ανάλογα με τις ειδικές απαιτήσεις κάθε παιχνιδιού, η διαδικασία ανάπτυξης απαιτεί ένα ευρύ φάσμα δεξιοτήτων, προκειμένου να προσφερθεί στον χρήστη ένα ολοκληρωμένο, προσεγμένο και άρτια δομημένο τελικό αποτέλεσμα.

3.6 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν οι βασικές τεχνολογίες που χρησιμοποιούνται στην ανάπτυξη σύγχρονων βιντεοπαιχνιδιών. Ξεκινώντας από τα τρισδιάστατα γραφικά και τη φυσική προσομοίωση, εξετάστηκε ο τρόπος με τον οποίο το rendering pipeline, οι shaders, το texturing και ο φωτισμός συνθέτουν τον οπτικό κόσμο ενός παιχνιδιού, ενώ η μηχανή φυσικής και η ανίχνευση συγκρούσεων διασφαλίζουν τη ρεαλιστική αλληλεπίδραση των αντικειμένων. Στη συνέχεια, αναλύθηκε ο ρόλος της τεχνητής νοημοσύνης, με έμφαση στις πεπερασμένες μηχανές καταστάσεων και τα δέντρα αποφάσεων για τη συμπεριφορική AI, καθώς και στον αλγόριθμο A* και το NavMesh για την πλοήγηση των χαρακτήρων εντός ενός παιχνιδιού. Ακολούθως, παρουσιάστηκαν τα συστήματα animation, τονίζοντας τη σημασία του rigging, των state machines και των animation events για την κίνηση των μοντέλων.

Ακολούθως, το κεφάλαιο εστίασε στις μηχανές παιχνιδιών, περιγράφοντας τα βασικά υποσυστήματά τους και συγκρίνοντας τις τρεις επικρατέστερες πλατφόρμες — Unity, Unreal Engine και Godot — με βάση τα χαρακτηριστικά, τα πλεονεκτήματα και τους περιορισμούς τους. Τέλος, παρουσιάστηκε η δομή μιας ομάδας ανάπτυξης βιντεοπαιχνιδιών, αναδεικνύοντας τους ρόλους του σχεδιαστή, του

προγραμματιστή, του καλλιτέχνη, του σχεδιαστή ήχου, του παραγωγού και του δοκιμαστή, καθώς και τη σημασία της συνεργασίας τους για την επίτευξη ενός επιτυχημένου αποτελέσματος.

Το θεωρητικό και τεχνικό υπόβαθρο που τέθηκε στο κεφάλαιο αυτό αποτελεί το απαραίτητο σημείο αναφοράς για την κατανόηση των σχεδιαστικών αποφάσεων που ελήφθησαν κατά την υλοποίηση του παιχνιδιού. Οι τεχνολογίες που αναλύθηκαν, από τα γραφικά και τη φυσική έως την ΑΙ και τα συστήματα animation, βρίσκουν άμεση εφαρμογή στην πράξη, συνθέτοντας τα επιμέρους κομμάτια που απαρτίζουν μια ολοκληρωμένη εμπειρία παιχνιδιού. Η γνώση που αποκτήθηκε εδώ καθιστά δυνατή την πλήρη κατανόηση του τρόπου λειτουργίας των επιμέρους συστημάτων, της αλληλεπίδρασής τους και της συμβολής τους στη συνολική εμπειρία του παίκτη.

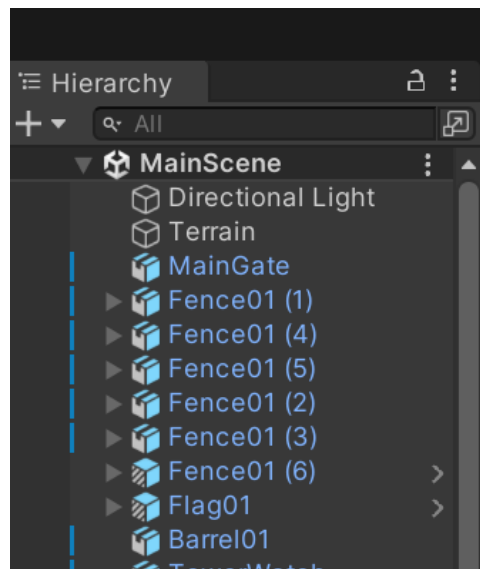
Κεφάλαιο 4ο: Unity Engine – Κύρια παράθυρα και εργαλεία

4.1 Εισαγωγή

Πριν ξεκινήσουμε την ανάπτυξη του παιχνιδιού είναι σημαντικό να γνωρίσουμε τα βασικά εργαλεία του Unity Engine είναι το περιβάλλον όπου χτίζουμε, ρυθμίζουμε και δοκιμάζουμε το παιχνίδι. Η εξοικείωση με αυτό είναι απαραίτητη για να μπορέσουμε να κατανοήσουμε την υλοποίηση του [21][22].

4.2 Hierarchy Window

Το **Hierarchy Window** (βλ. Εικόνα 4-1) εμφανίζει όλα τα αντικείμενα που υπάρχουν στην τρέχουσα σκηνή, σε μορφή λίστας [21][22]. Κάθε αντικείμενο μπορεί να έχει «παιδιά» (child objects) που εξαρτώνται από αυτό [23]. Για παράδειγμα, ένας χαρακτήρας μπορεί να έχει ως παιδί ένα φως ή μια κάμερα που τον ακολουθεί. Αυτή η ιεραρχική δομή βοηθά στην οργάνωση της σκηνής και είναι χρήσιμη για την επιλογή αντικειμένων, την παρατήρηση της συνολικής δομής και τη δημιουργία σχέσεων μεταξύ τους.



Εικόνα 4-1 Παράθυρο Hierarchy

4.3 Scene View

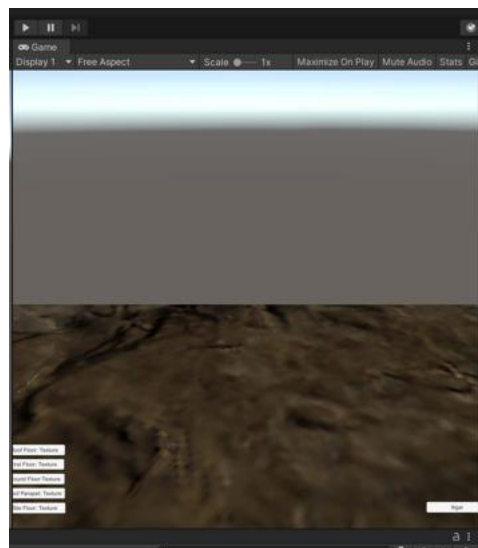
Το **Scene View** (βλ. Εικόνα 4-2) είναι ο βασικός χώρος εργασίας μας μέσα στο Unity. Είναι το παράθυρο όπου βλέπουμε και διαμορφώνουμε τον κόσμο του παιχνιδιού, τοποθετώντας αντικείμενα (GameObjects), όπως χαρακτήρες, κτίρια, φώτα και κάμερες. Το παράθυρο αυτό αποτελεί ίσως το σημαντικότερο εργαλείο μας, καθώς μπορούμε να δημιουργούμε νέα αντικείμενα, να τα μετακινούμε, να τα περιστρέφουμε και να αλλάζουμε το μέγεθός τους. Μπορούμε επίσης, να περιηγηθούμε στον χώρο με το ποντίκι, βλέποντας το περιβάλλον από κάθε γωνία. Αυτό μας δίνει απόλυτο έλεγχο στη διαμόρφωση του κόσμου μας [21][23].



Εικόνα 4-2 Παράθυρο Scene

4.4 Game View

Το **Game View** (βλ. Εικόνα 4-3) δείχνει το παιχνίδι ακριβώς όπως θα το δει ο παίκτης. Ενώ το Scene View είναι το περιβάλλον ανάπτυξης, το Game View είναι η εμπειρία του χρήστη. Όταν πατάμε το κουμπί Play, το παιχνίδι τρέχει σε πραγματικό χρόνο και βλέπουμε ακριβώς τι θα βλέπει ο παίκτης στην οθόνη του. Εδώ, ουσιαστικά, δοκιμάζουμε το gameplay, τους ελέγχους και τη συμπεριφορά του παιχνιδιού. Μπορούμε να παίζουμε σαν να ήμασταν ο τελικός χρήστης, να εντοπίσουμε προβλήματα και να δούμε αν όλα λειτουργούν σωστά [21][23].

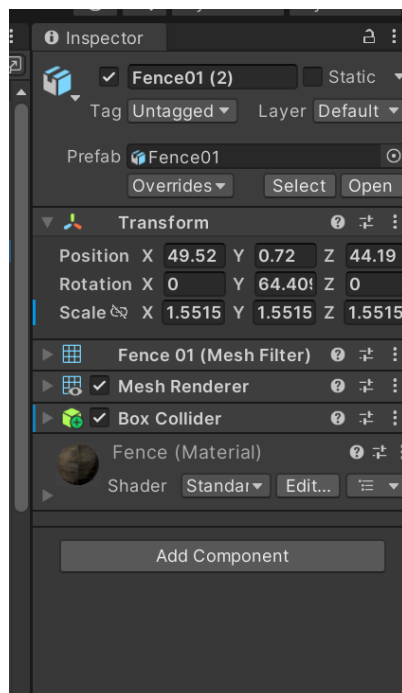


Εικόνα 4-3 Παράθυρο Game

4.5 Inspector Window

Στο παράθυρο του **Inspector** (βλ. Εικόνα 4-4) βλέπουμε όλες τις ιδιότητες και τα components ενός αντικειμένου που έχουμε επιλέξει. Όταν επιλέγουμε ένα αντικείμενο, βλέπουμε εδώ τη θέση του, την περιστροφή του, το μέγεθός του, αλλά και όλα τα components που έχει (π.χ. Rigidbody, Collider, Scripts) με τις ρυθμίσεις τους. Σε αυτό το παράθυρο μπορούμε να ρυθμίσουμε παραμέτρους χωρίς να

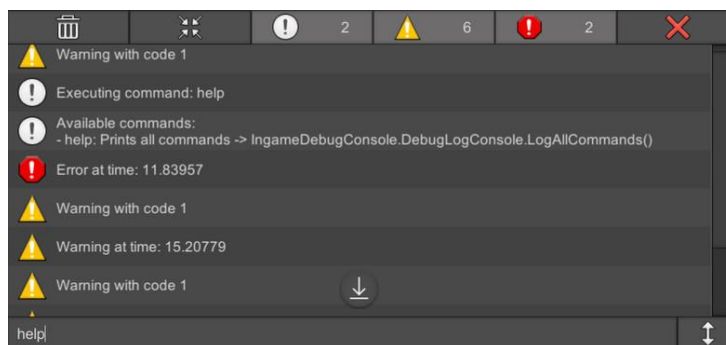
γράφουμε κώδικα. Εδώ αλλάζουμε ταχύτητες, δυνάμεις, χρώματα και συνδέουμε scripts με αντικείμενα [21][22].



Εικόνα 4-4 Παράθυρο Inspector

4.6 Console Window

Το Console Window (βλ. Εικόνα 4-5) είναι το εργαλείο αποσφαλμάτωσης (debugging) της Unity. Εδώ εμφανίζονται μηνύματα, προειδοποιήσεις και σφάλματα από το παιχνίδι μας. Όταν γράφουμε κώδικα, μπορούμε να χρησιμοποιήσουμε την εντολή `Debug.Log()` για να εκτυπώσουμε μηνύματα και να παρακολουθήσουμε τι συμβαίνει στο παιχνίδι. Η κονσόλα δείχνει τρεις τύπους μηνυμάτων: πληροφορίες (άσπρα), προειδοποιήσεις (κίτρινα) και σφάλματα (κόκκινα). Κάνοντας κλικ σε ένα σφάλμα, μας πηγαίνει απευθείας στο σημείο του κώδικα που το προκάλεσε [21][22].



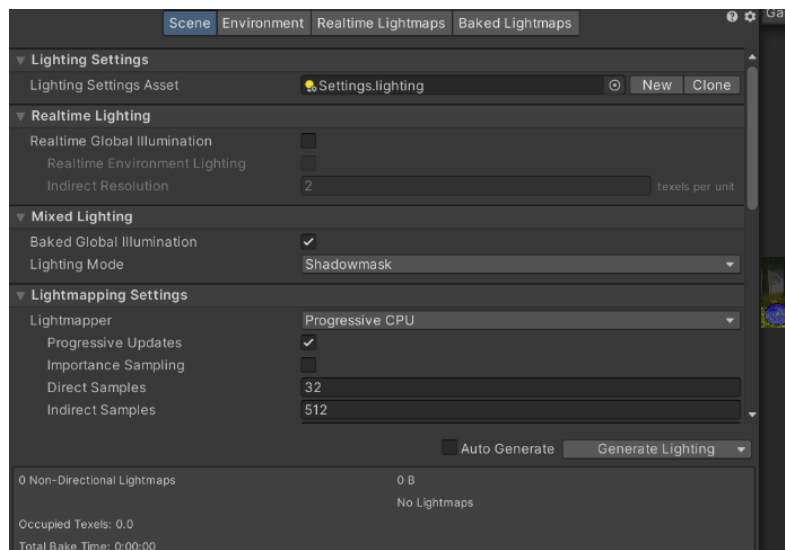
Εικόνα 4-5 Παράθυρο Console

4.7 Lighting

Το **Lighting Settings Window** (βλ. Εικόνα 4-6) είναι το παράθυρο όπου ρυθμίζουμε τον φωτισμό όλης της σκηνής. Εδώ ορίζουμε το γενικό φως (Ambient Light) που φωτίζει τις σκιερές περιοχές, ρυθμίζουμε

Κεφάλαιο 4

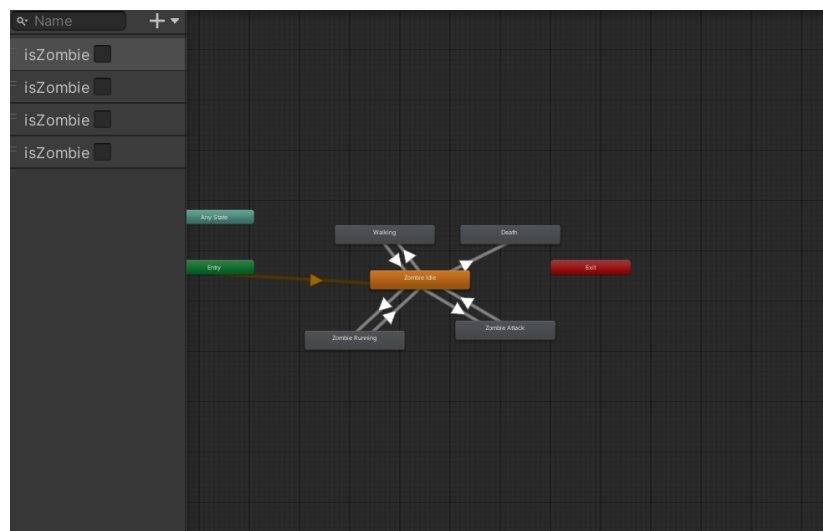
τις αντανakλάσεις και κάνουμε προ-υπολογισμό (baking) του φωτισμού για καλύτερη απόδοση [24][25].



Εικόνα 4-6 Παράθυρο Lighting

4.8 Animator

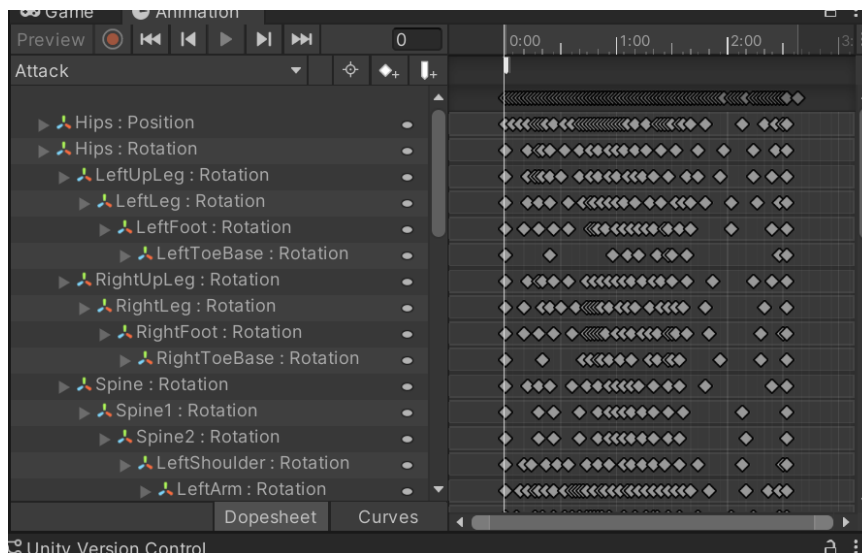
Το **Animator Window** (βλ. Εικόνα 4-7) αναλαμβάνει να ζωντανέψει τις κινήσεις των χαρακτήρων και των αντικειμένων. Λειτουργεί μέσω του Animator Controller, το οποίο ουσιαστικά είναι μια μηχανή καταστάσεων για animations. Εδώ μπορούμε να ορίσουμε διαφορετικές κινήσεις, να ορίσουμε τις μεταξύ τους σχέσεις, αλλά και τις συνθήκες μεταβάσεις που καθορίζουν πότε αλλάζει η κίνηση [26][27].



Εικόνα 4-7 Παράθυρο Animator

4.9 Animation

Το **Animation Window** (βλ. Εικόνα 4-8) είναι υπεύθυνο για την αναπαραγωγή των κινήσεων των χαρακτήρων ή των αντικειμένων. Το animation αποτελείται από μια σειρά καρτέ (frames) που δημιουργούν ομαλή κίνηση. Αντί να σχεδιάζουμε όλα τα frames μόνοι μας, χρησιμοποιούμε έτοιμα animation clips. Τα animation clips εισάγονται στο Unity και το Animator ελέγχει ποιο clip παίζεται κάθε φορά. Κάθε clip έχει χαρακτηριστικά, όπως διάρκεια (duration), επανάληψη (looping) και ταχύτητα.



Εικόνα 4-8 Παράθυρο Animation

4.10 Επίλογος

Η γνωριμία με τον Unity Editor και τα βασικά του εργαλεία αποτελεί το θεμέλιο για κάθε προσπάθεια ανάπτυξης. Η κατανόηση του τρόπου λειτουργίας του περιβάλλοντος είναι απαραίτητη. Τα εργαλεία που παρουσιάστηκαν από τα παράθυρα διαχείρισης αντικειμένων και σκηνών, μέχρι τα συστήματα φωτισμού και κίνησης αποτελούν την «εργαλειοθήκη» πάνω στην οποία χτίζεται κάθε παιχνίδι.

Κεφάλαιο 5ο: Ανάλυση και σχεδίαση συστήματος

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται ο σχεδιασμός και η αρχιτεκτονική του **Dead Horizon**, ενός 3D FPS παιχνιδιού επιβίωσης με ζόμπι. Βασιζόμενοι στα θεωρητικά εργαλεία που αναλύθηκαν στο προηγούμενο κεφάλαιο όπως η τρισδιάστατη γραφική, η φυσική, η τεχνητή νοημοσύνη και οι μηχανές παιχνιδιών θα εξετάσουμε πώς αυτές οι έννοιες μεταφέρονται στην πράξη.

Στόχος του κεφαλαίου είναι να παρουσιάσει τις αρχιτεκτονικές επιλογές, τις λειτουργικές και μη λειτουργικές απαιτήσεις, καθώς και τις τεχνικές προδιαγραφές που καθοδήγησαν την ανάπτυξη του παιχνιδιού. Κάθε σχεδιαστική απόφαση συνδέεται με τα θεωρητικά εργαλεία που είδαμε νωρίτερα τις πεπερασμένες μηχανές καταστάσεων για την AI, το NavMesh για την πλοήγηση και το rendering pipeline για τα γραφικά.

5.2 Έγγραφο Σχεδιασμού Παιχνιδιού (Game Design Document - GDD)

Το **Game Design Document (GDD)** είναι το έγγραφο που περιγράφει πώς θα είναι ένα παιχνίδι. Λειτουργεί σαν τα σχέδια ενός σπιτιού — καθορίζει κάθε λεπτομέρεια, από τους μηχανισμούς και τους χαρακτήρες μέχρι το περιβάλλον και την ιστορία. Στην ανάπτυξη παιχνιδιών το GDD βοηθάει όλη την ομάδα (σχεδιαστές, προγραμματιστές, καλλιτέχνες) να δουλεύουν προς τον ίδιο στόχο.

5.2.1 Περιγραφή παιχνιδιού

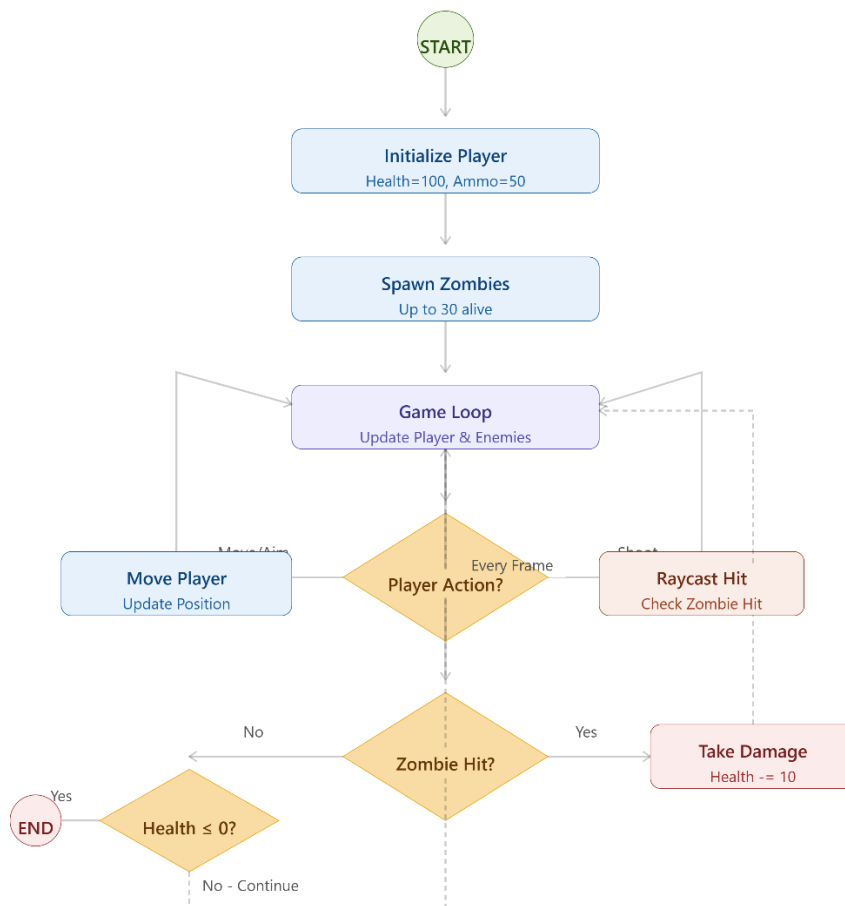
Το **Dead Horizon** είναι ένα FPS παιχνίδι όπου ο παίκτης βρίσκεται αντιμέτωπος με συνεχή κύματα ζόμπι σε ένα ανοιχτό περιβάλλον (open world). Στόχος του είναι να επιβιώσει για όσο το δυνατόν περισσότερο γίνεται ενώ παράλληλα πρέπει να σκοτώνει όσο το δυνατόν περισσότερους εχθρούς. Το παιχνίδι δεν διαθέτει ξεκάθαρο τέλος και η ροή του συνεχίζεται έως ότου η υγεία του παίκτη μηδενιστεί και ο χαρακτήρας του πεθάνει.

5.2.2 Στάδια παιχνιδιού-Gameplay Loop

Ο βασικός κύκλος του παιχνιδιού περιγράφει τον τρόπο με τον οποίο ο παίκτης αλληλεπιδρά με το σύστημα και χωρίζεται σε έξι βασικά στάδια (βλ. Εικόνα 5-1).

- **Εκκίνηση:** Ο παίκτης ξεκινά με 100 πόντους ζωής, 25 σφαίρες στον γεμιστήρα και 50 σφαίρες στην εφεδρεία.
- **Εξερεύνηση:** Ο παίκτης μπορεί να κινείται ελεύθερα στο περιβάλλον χρησιμοποιώντας τα πλήκτρα WASD για μετακίνηση και το ποντίκι για να κοιτάζει τριγύρω και έλεγχο της κάμερας. Αυτό του επιτρέπει να εξερευνήσει τον χάρτη να ψάξει για χρήσιμα αντικείμενα και να διαλέξει πώς θα τοποθετηθεί απέναντι στους εχθρούς.
- **Σύγκρουση:** Καθώς ο παίκτης περιπλανιέται μέσα στο terrain ζόμπι εμφανίζονται τυχαία σε διάφορες αποστάσεις γύρω του. Η εμφάνισή τους είναι ξαφνική και θέλει γρήγορη αντίδραση, δημιουργώντας έτσι μια συνεχή αίσθηση έντασης.

- **Μάχη:** Όταν ο παίκτης θέλει να επιτεθεί, πυροβολεί χρησιμοποιώντας ακτίνες (raycast). Κάθε βολή εκτελεί έναν έλεγχο που καθορίζει αν η ακτίνα χτυπά κάποιον εχθρό, σε περίπτωση επιτυχούς χτυπήματος το ζόμπι δέχεται ζημιά, πεθαίνει και προστίθεται ένας ακόμη πόντος στο σκορ του. Η διαδικασία αυτή απαιτεί ακρίβεια και γρήγορα αντανακλαστικά ειδικά όταν πολλοί εχθροί επιτίθενται ταυτόχρονα.
- **Συλλογή αντικειμένων:** Σε συγκεκριμένα σημεία του χάρτη εμφανίζονται κατά διαστήματα κιτ υγείας και πυρομαχικών. Ο παίκτης τα "σηκώνει" περνώντας από πάνω τους και έτσι αναπληρώνει τη ζωή του ή τα πυρομαχικά του. Η σωστή διαχείριση αντικειμένων είναι σημαντική για την επιβίωση ειδικά εντός του παιχνιδιού.
- **Θάνατος:** Όταν η υγεία του παίκτη φτάσει στο μηδέν το παιχνίδι τελειώνει στην συνέχεια εμφανίζεται η οθόνη Game Over, η οποία δείχνει τον συνολικό αριθμό ζόμπι που σκότωσε ο παίκτης. Αυτός ο αριθμός είναι το βασικό μέτρο επίδοσης, αφού το παιχνίδι δεν έχει τέλος και σκοπός είναι να φτάσει κανείς όσο πιο ψηλά γίνεται.



Εικόνα 5-1 διάγραμμα ροής αλληλεπίδρασης

5.2.3 Μηχανισμοί Παίκτη -Player Mechanics

Movement: Ο παίκτης κινείται με τα πλήκτρα WASD και πηδάει με το SPACE και περιστρέφει την κάμερα με το ποντίκι. Το παιχνίδι χρησιμοποιεί το σύστημα φυσικής της Unity, έτσι ο παίκτης δεν μπορεί να περάσει μέσα από τοίχους ή άλλα αντικείμενα.

Shooting: Με αριστερό κλικ ο παίκτης πυροβολεί. Κάθε βολή ελέγχει αν χτύπησε στόχο μέσω μιας ακτίνας (**raycast**). Υπάρχει μια μικρή καθυστέρηση 200ms μεταξύ των βολών, ώστε να ταιριάζει με την κίνηση του πυροβολισμού.

Ammo: Ο παίκτης έχει 25 σφαίρες στον γεμιστήρα και 50 σε εφεδρεία. Κάθε βολή αδειάζει μία σφαίρα. Όταν αδειάσει ο γεμιστήρας, ο παίκτης πατάει R για επαναφόρτωση. Η επαναφόρτωση διαρκεί όσο το animation της και κατά τη διάρκειά της ο παίκτης δεν μπορεί να πυροβολήσει.

Health: Ο παίκτης ξεκινά με 100 πόντους υγείας, κάθε χτύπημα από ζόμπι μειώνει την υγεία κατά 10 ή 20 πόντους ανάλογα τον τύπο του. Μπορεί να την ανακτήσει μαζεύοντας κιτ υγείας. Όταν τραυματίζεται εμφανίζεται μια κόκκινη κηλίδα στην οθόνη που λειτουργεί ως μηχανισμός ανατροφοδότησης.

5.2.4 Μηχανισμοί εχθρών – Enemy mechanics

Συμπεριφορά Ζόμπι: Κάθε ζόμπι λειτουργεί με μια μηχανή καταστάσεων (FSM) που έχει πέντε διαφορετικές καταστάσεις (βλ. Εικόνα 5-2):

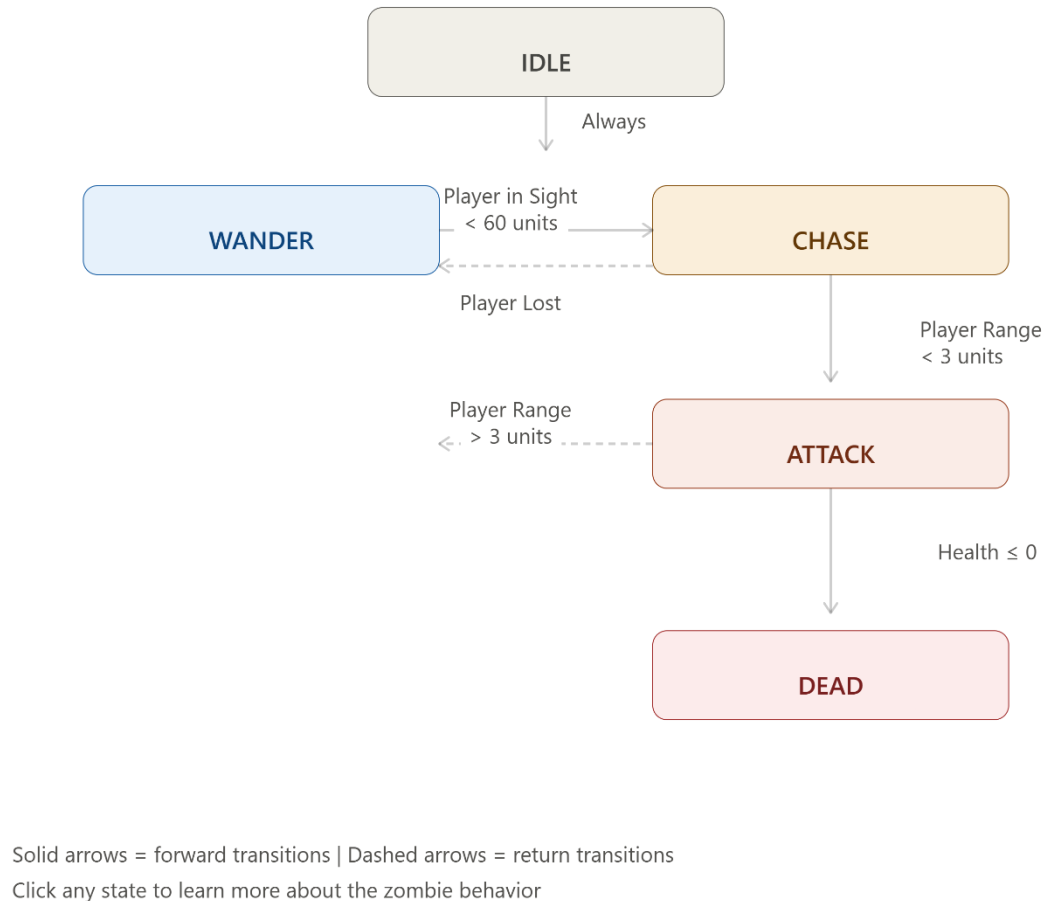
IDLE (Αδράνεια): Το ζόμπι στέκεται ακίνητο και περιμένει.

WANDER (Περιπλάνηση): Το ζόμπι περπατάει τυχαία μέσα στον χάρτη υστέρα διαλέγει ένα σημείο κοντά του μέχρι 5 μέτρα και πηγαίνει εκεί. Όταν φτάσει αυτή η διαδικασία επαναλαμβάνεται και διαλέγει ένα νέο σημείο.

CHASE (Καταδίωξη): Όταν το ζόμπι δει τον παίκτη από μακριά λιγότερο από 60 μέτρα, σταματά την περιπλάνηση και αρχίζει να τον κυνηγάει. Βρίσκει τον δρόμο του αξιοποιώντας το NavMesh και τον ακολουθεί όπου κι αν πάει ο παίκτης.

ATTACK (Επίθεση): Όταν το ζόμπι πλησιάσει πολύ τον παίκτη δηλαδή σε απόσταση λιγότερο από 3 μέτρα σταματά και αρχίζει να τον χτυπάει. Όσο η απόσταση μεταξύ του παίκτη και το zombie δεν αυξάνεται η διαδικασία αυτή επαναλαμβάνεται.

DEAD (Θάνατος): Όταν το ζόμπι χτυπηθεί από μια σφαίρα δηλαδή το raycast πεθαίνει στην συνέχεια σταματά να κινείται, το NavMeshAgent απενεργοποιείται και το ζόμπι αρχίζει να βυθίζεται στο έδαφος. Μετά από λίγα δευτερόλεπτα εξαφανίζεται.



Εικόνα 5-2 Zombie FSM διάγραμμα

5.2.5 Σύστημα Εμφάνισης (Spawning)

Τα ζόμπι εμφανίζονται σταδιακά στο παιχνίδι περίπου ένα νέο ζόμπι κάθε 5 δευτερόλεπτα μέχρι να φτάσουν τον μέγιστο αριθμό ζωντανών ζόμπι που έχει οριστεί. Κάθε ζόμπι τοποθετείται σε τυχαίο σημείο του χάρτη, με την προϋπόθεση ότι απέχει τουλάχιστον 15 μέτρα από τον παίκτη. Για να διασφαλιστεί ότι το σημείο είναι προσβάσιμο, το σύστημα ελέγχει αν το σημείο είναι εντός του NavMesh.

5.2.6 Αντικείμενα και Συλλογή (Pickups)

Health Kits: Εμφανίζονται σε τέσσερα συγκεκριμένα σημεία του χάρτη κάθε 120 δευτερόλεπτα. Κάθε κιτ δίνει 25 πόντους υγείας. Μόλις ο παίκτης το μαζέψει εξαφανίζεται και το χρονόμετρο ξεκινά από την αρχή για την επόμενη εμφάνιση.

Ammo Pickups: Λειτουργούν με τον ίδιο τρόπο εμφανίζονται στις ίδιες θέσεις κάθε 120 δευτερόλεπτα. Κάθε συλλογή προσθέτει 20 σφαίρες στο εφεδρικό απόθεμα του παίκτη.

5.2.7 Minimap & Radar System

Minimap & Radar: Στην κάτω δεξιά γωνία της οθόνης εμφανίζεται ένα μικρό παράθυρο (minimap/ραντάρ) που δείχνει μια άνωθεν άποψη του περιβάλλοντος. Το ραντάρ αυτό βοηθά τον παίκτη να έχει καλύτερη χωρική επίγνωση καθώς βλέπει τη θέση του, τη θέση των ζόμπι και των κιτ υγείας/πυρομαχικών γύρω του. Ο παίκτης αντιπροσωπεύεται από ένα βέλος στο κέντρο του mini map ενώ το σύστημα ενημερώνεται σε πραγματικό χρόνο και παρέχει συνεχείς πληροφορίες για όλα τα στοιχεία του κόσμου.

5.3 Δομή Σκηνών-Scene Structure

Το Dead Horizon χρησιμοποιεί δύο κύριες σκηνές, που ξεχωρίζουν τις διαφορετικές καταστάσεις του παιχνιδιού.

5.3.1 Main menu scene

MainMenu Scene είναι το πρώτο σημείο επαφής του παίκτη με το παιχνίδι περιέχει τρεις επιλογές:

- **Play** για έναρξη του παιχνιδιού οδηγεί τον παίκτη στο main scene.
- **Controls** περιέχει τις οδηγίες των πλήκτρων εντός παιχνιδιού.
- **Exit** το κουμπί αυτό τερματίζει το παιχνίδι και κλείνει την εφαρμογή.

Η σκηνή είναι απλή και εστιάζει σε μια καθαρή διεπαφή χρήστη.

5.3.2 MainScene

MainScene είναι η καρδιά του παιχνιδιού, όπου συμβαίνει όλη η δράση. Αποτελείται από τα εξής βασικά στοιχεία:

- **Terrain:** Ο ανοιχτός κόσμος με διάφορα εδαφικά στρώματα, που δίνει την αίσθηση ενός ζωντανού περιβάλλοντος.
- **Player GameObject:** Ο χαρακτήρας του παίκτη, που ελέγχεται από το script MainCharacterController.
- **Zombie Prefabs:** Τα αντικείμενα των εχθρών που δημιουργούνται δυναμικά κατά τη διάρκεια του παιχνιδιού. Κάθε zombie ελέγχεται από το script zombieController που υλοποιεί την συμπεριφορά FSM.
- **Spawn Points:** Προκαθορισμένες θέσεις όπου εμφανίζονται τα health kits και τα ammo pickups, εξασφαλίζοντας ότι οι πόροι κατανέμονται ομαλά.
- **UI Canvas:** Όλα τα στοιχεία της οθόνης το HUD υγεία, πυρομαχικά, σκορ, το minimap που δείχνει τη θέση των ζόμπι και των αντικειμένων γύρω από τον παίκτη, το μενού παύσης και η οθόνη Game Over.

- **Directional Light:** Ο ήλιος που φωτίζει το περιβάλλον και δημιουργεί σκιές, προσθέτοντας ατμόσφαιρα.

5.4 Λειτουργικές & Μη Λειτουργικές Απαιτήσεις

5.4.1 Λειτουργικές Απαιτήσεις-Functional Requirements

- **Player Movement:** Ο παίκτης μπορεί να κινείται χρησιμοποιώντας τα πλήκτρα WASD και να πηδάει με το SPACE.
- **Camera Control:** Το ποντίκι ελέγχει την περιστροφή της κάμερας, προσφέροντας οπτική ελευθερία.
- **Shooting:** Το αριστερό κλικ ενεργοποιεί τον πυροβολισμό με χρήση raycast. Υπάρχει καθυστέρηση 200ms μεταξύ των βολών για συγχρονισμό με το animation.
- **Reload:** Πατώντας το πλήκτρο R, ο παίκτης γεμίζει σφαίρες από την εφεδρεία στον γεμιστήρα.
- **Ammo Management:** Το σύστημα διαχειρίζεται ξεχωριστά τις σφαίρες του γεμιστήρα (clip) και της εφεδρείας (reserve).
- **Health System:** Ο παίκτης διαθέτει σύστημα υγείας, το οποίο μειώνεται όταν δέχεται χτύπημα από ζόμπι.
- **Zombie Spawning:** Τα ζόμπι εμφανίζονται περιοδικά σε τυχαίες θέσεις του χάρτη, με έλεγχο ελάχιστης απόστασης από τον παίκτη.
- **Zombie AI:** Τα ζόμπι λειτουργούν με πεπερασμένη μηχανή καταστάσεων (FSM) που περιλαμβάνει τις καταστάσεις IDLE, WANDER, CHASE, ATTACK και DEAD.
- **Pathfinding:** Τα ζόμπι χρησιμοποιούν NavMesh και τον αλγόριθμο A* για να εντοπίσουν και να ακολουθήσουν τον παίκτη.
- **Item Pickups:** Κιτ υγείας και πυρομαχικών εμφανίζονται περιοδικά σε σταθερές θέσεις και μπορούν να συλλεχθούν από τον παίκτη.
- **Game Over:** Όταν η υγεία του παίκτη μηδενιστεί, εμφανίζεται η οθόνη Game Over.
- **Kill Counter:** Ο αριθμός των νεκρών ζόμπι καταγράφεται και εμφανίζεται στην οθόνη σε πραγματικό χρόνο.
- **Pause Menu:** Πατώντας το πλήκτρο ESC, ανοίγει το μενού παύσης και ο χρόνος του παιχνιδιού σταματά.

- **Audio Feedback:** Το παιχνίδι παρέχει ηχητική ανατροφοδότηση για ενέργειες όπως πυροβολισμός, reload, χτύπημα ζόμπι και θάνατος.

5.4.2 Μη Λειτουργικές Απαιτήσεις (Non-Functional Requirements)

- **Frame Rate:** Το παιχνίδι πρέπει να τρέχει σταθερά στα 60 FPS σε Windows και Linux.
- **Max Zombies:** Το σύστημα υποστηρίζει έως έναν 30 ζωντανά ζόμπι ταυτόχρονα.
- **Intuitive Controls:** Οι έλεγχοι πρέπει να είναι διαισθητικοί WASD για κίνηση, ποντίκι για κάμερα, αριστερό κλικ για βολή, ESC για παύση κλπ.
- **Stability:** Το παιχνίδι δεν πρέπει να καταρρέει crash υπό κανονικές συνθήκες λειτουργίας.
- **Graphics Fidelity:** Το παιχνίδι χρησιμοποιεί 3D rendering με φωτισμό, σκιές και υφές.
- **Sound Design:** Το παιχνίδι διαθέτει διαφορετικά ηχητικά εφέ για κάθε ενέργεια (βολή, reload, χτύπημα, θάνατος).
- **Visual Feedback:** Το παιχνίδι παρέχει οπτική ανατροφοδότηση μέσω blood splatter, kill notifications και αλλαγών στο HUD.

5.5 Τεχνικές Προδιαγραφές

5.5.1 Development Environment

- **Unity 2022.3.62f1 LTS:** Μηχανή παιχνιδιού η κύρια πλατφόρμα ανάπτυξης. Η έκδοση LTS (Long Term Support) εξασφαλίζει σταθερότητα και μακροχρόνια υποστήριξη.
- **C#:** Γλώσσα προγραμματισμού χρησιμοποιήθηκε για τη συγγραφή όλων των scripts (κίνηση, AI, UI, κ.λπ.).
- **Windows / Linux:** Το παιχνίδι αναπτύχθηκε για να τρέχει και στα δύο λειτουργικά συστήματα.
- **Visual Studio:** Ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) χρησιμοποιήθηκε για την επεξεργασία του κώδικα.
- **Git Σύστημα ελέγχου έκδοσης:** Χρησιμοποιήθηκε για την παρακολούθηση των αλλαγών στον κώδικα και τη διαχείριση των εκδόσεων.

5.5.2 Ελάχιστες λειτουργικές απαιτήσεις

- **Λειτουργικό Σύστημα:** Windows 10/ Linux (Ubuntu 18.04+).
- **Επεξεργαστής:** Intel Core i5-6600 / AMD Ryzen 5 1600 ή ισοδύναμος.
- **Μνήμη (RAM):** 8 GB.
- **Κάρτα Γραφικών:** NVIDIA GeForce GTX 960 / AMD Radeon RX 470 με 2 GB VRAM.
- **Αποθηκευτικός Χώρος:** 1.5 GB ελεύθερος χώρος στον σκληρό δίσκο.
- **Ανάλυση Οθόνης:** 1280×720 (720p).
- **Στόχος Απόδοσης:** 30 FPS (σταθερό).

5.6 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε ο σχεδιασμός και η αρχιτεκτονική του Dead Horizon, μεταφέροντας τις θεωρητικές έννοιες των προηγούμενων κεφαλαίων σε πρακτικές λύσεις. Το **Game Design Document** καθόρισε τον βασικό στόχο του παιχνιδιού, όπου ο παίκτης αντιμετωπίζει συνεχή κύματα ζόμπι και προσπαθεί να επιβιώσει όσο περισσότερο γίνεται. Μέσα από την περιγραφή του gameplay loop, των μηχανισμών του παίκτη και της συμπεριφοράς των εχθρών, τέθηκαν οι βάσεις για την υλοποίηση.

Η αρχιτεκτονική του συστήματος έδειξε πώς τα επιμέρους υποσυστήματα — παίκτης, AI εχθρών, σύστημα εμφάνισης, φυσική, UI — επικοινωνούν μεταξύ τους, ενώ η χρήση δύο βασικών σκηνών (MainMenu για το μενού και MainScene για το gameplay) προσφέρει καθαρή οργάνωση και αποδοτική διαχείριση των πόρων. Οι λειτουργικές και μη λειτουργικές απαιτήσεις όρισαν με σαφήνεια τι πρέπει να κάνει το παιχνίδι και πώς πρέπει να το κάνει.

Οι τεχνικές προδιαγραφές παρουσίασαν τα εργαλεία, τις γλώσσες και τις απαιτήσεις συστήματος που υποστηρίζουν την ανάπτυξη — Unity 2022.3.62f1 LTS, C# και ελάχιστες προδιαγραφές hardware για ευρεία προσιμότητα.

Το κεφάλαιο αυτό λειτουργεί ως η γέφυρα ανάμεσα στα προηγούμενα δύο κεφάλαια (Κεφάλαιο 3 και Κεφάλαιο 4) και την υλοποίηση (Κεφάλαιο 5). Κάθε σχεδιαστική απόφαση συνδέεται με τα τεχνολογικά εργαλεία που αναλύθηκαν προηγουμένως — τις μηχανές καταστάσεων για την AI, το NavMesh για την πλοήγηση και το rendering pipeline για τα γραφικά. Οι αποφάσεις που παρουσιάστηκαν εδώ αποτέλεσαν τον οδηγό για την πρακτική ανάπτυξη του παιχνιδιού, η οποία περιγράφεται αναλυτικά στο επόμενο κεφάλαιο.

Κεφάλαιο 6ο: Ανάπτυξη και Υλοποίηση

6.1 Εισαγωγή

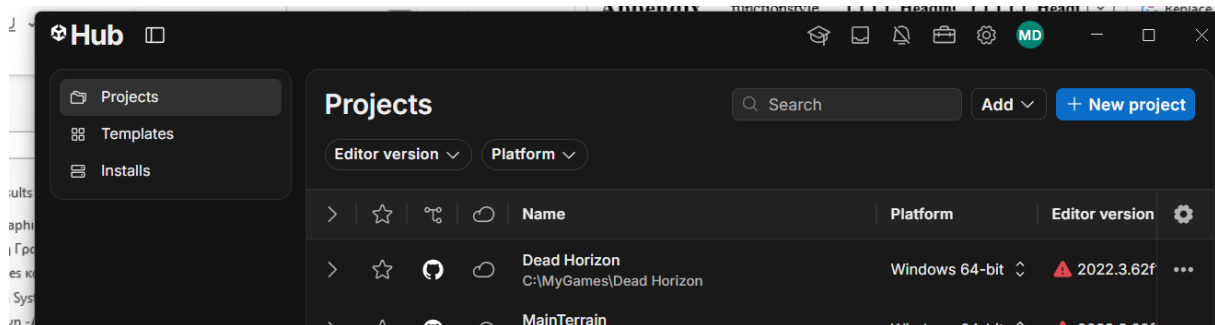
Στο κεφάλαιο αυτό ξεκινά η αναλυτική παρουσίαση της υλοποίησης του παιχνιδιού που αναπτύχθηκε στο πλαίσιο της πτυχιακής εργασίας. Περιγράφεται όλη η διαδικασία που ακολουθήθηκε κατά την ανάπτυξη, με στόχο να δοθεί μια ολοκληρωμένη τεχνική εικόνα της κατασκευής ενός τέτοιου συστήματος.

Το Dead Horizon είναι ένα FPS παιχνίδι όπου ο παίκτης ελέγχει έναν χαρακτήρα και, χρησιμοποιώντας τα διαθέσιμα πυρομαχικά και πόρους, προσπαθεί να επιβιώσει εξοντώνοντας συνεχόμενα κύματα ζόμπι. Η κάμερα βρίσκεται στο οπτικό πεδίο του χαρακτήρα, δημιουργώντας μια αίσθηση ρεαλισμού και βύθισης στον κόσμο του παιχνιδιού.

Στο παρόν κεφάλαιο παρουσιάζεται βήμα-βήμα η ανάπτυξη της εφαρμογής. Καταγράφονται τα εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν, οι ρυθμίσεις του περιβάλλοντος εργασίας και τα στάδια κατασκευής του παιχνιδιού.

6.2 Ξεκινώντας με Unity

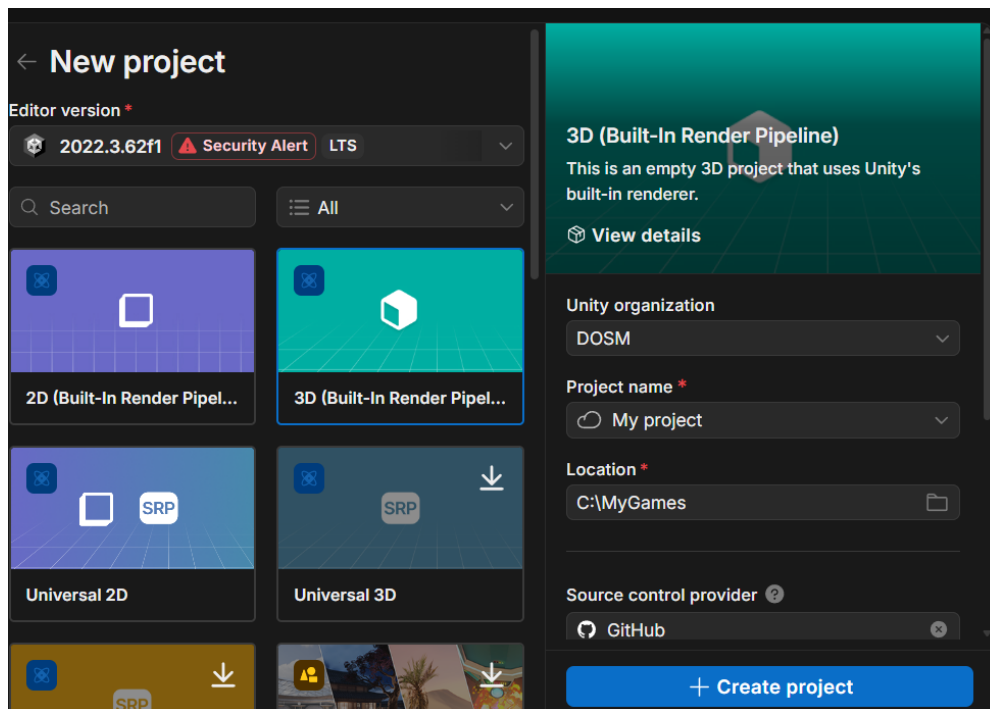
Η διαδικασία ανάπτυξης του Dead Horizon ξεκινά με τη δημιουργία ενός νέου project στο Unity. Ανοίγοντας το Unity Hub, ο χρήστης βρίσκεται στο κεντρικό παράθυρο διαχείρισης έργων. Στο πάνω αριστερό μέρος της οθόνης υπάρχει η επιλογή New Project, η οποία εκκινεί τη διαδικασία δημιουργίας ενός νέου έργου (βλ. Εικόνα 6-1).



Εικόνα 6-1 Δημιουργία νέου project στο unity

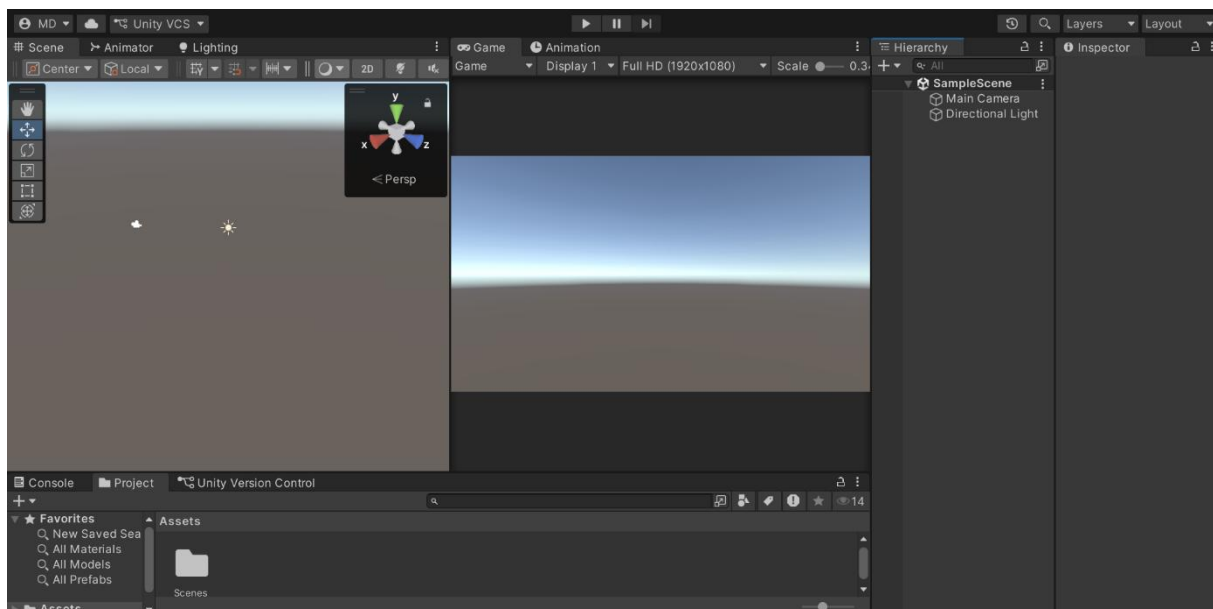
Με την επιλογή αυτή, εμφανίζεται ένα νέο παράθυρο στο οποίο ζητούνται ορισμένες βασικές ρυθμίσεις για το project. Αρχικά, ο χρήστης καλείται να επιλέξει τον τύπο του έργου ανάμεσα σε 2D, 3D, κ.ά. Για τις ανάγκες της πτυχιακής επιλέγεται το 3D.

Στη συνέχεια, ορίζεται το όνομα του project, το οποίο στην προκειμένη περίπτωση είναι Dead Horizon. Το επόμενο βήμα αφορά την επιλογή του φακέλου αποθήκευσης του έργου στον υπολογιστή, ώστε όλα τα αρχεία και τα assets να οργανώνονται σε μια συγκεκριμένη διαδρομή (βλ. Εικόνα 6-2).



Εικόνα 6-2 Καθορισμός ονόματος και είδος παιχνιδιού

Αφού συμπληρωθούν όλες οι απαραίτητες ρυθμίσεις, το Unity δημιουργεί το νέο project και ανοίγει τον Editor. Μόλις φορτώσει το περιβάλλον, εμφανίζεται το κεντρικό παράθυρο σχεδίασης της εφαρμογής (βλ. Εικόνα 6-3), όπου ξεκινά η διαδικασία δημιουργίας του παιχνιδιού. Η λειτουργία των παραθύρων έχει ήδη περιγραφεί αναλυτικά στο Κεφάλαιο 4.



Εικόνα 6-3 Unity Editor

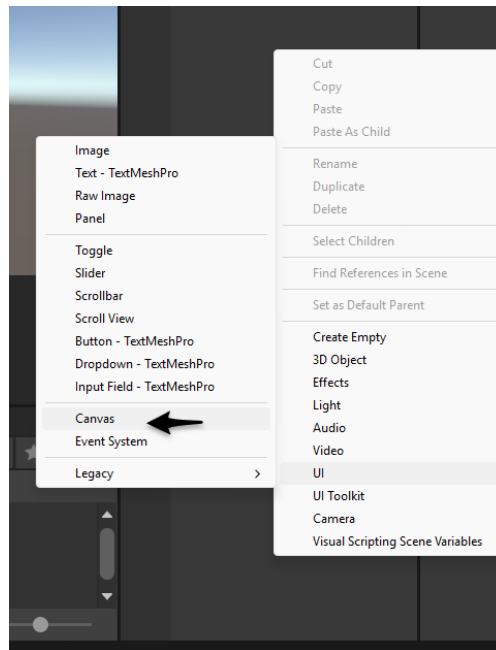
6.3 Menus

Τα συστήματα μενού αποτελούν δομικό στοιχείο για την αλληλεπίδραση του χρήστη με το παιχνίδι, καθώς διασφαλίζουν τη λειτουργική συνοχή και την ομαλή ροή της εφαρμογής. Λειτουργούν ως ο

κεντρικός μηχανισμός πλοήγησης, παρέχοντας στον παίκτη άμεση πρόσβαση σε πληροφορίες, λειτουργίες ελέγχου και παραμετροποίηση των επιλογών του συστήματος. Μέσω των διεπαφών, ο χρήστης μπορεί να ρυθμίσει κρίσιμες παραμέτρους και να διαχειριστεί την κατάσταση της συνεδρίας.

6.3.1 Main Menu

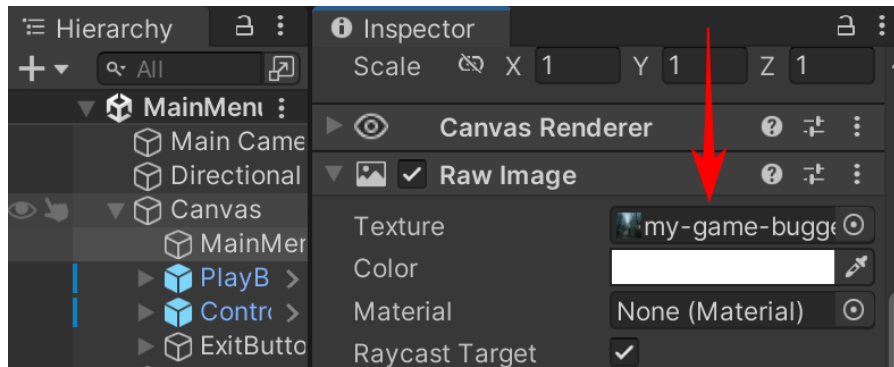
Το Main Menu είναι ένα από τα σημαντικότερα παράθυρα του project, καθώς είναι η πρώτη επαφή του χρήστη με την εφαρμογή και μπορεί να τον προδιαθέσει για το τι θα συναντήσει στο παιχνίδι. Για να δημιουργήσουμε το Main Menu, ξεκινάμε με ένα Canvas, το οποίο αποτελεί το βασικό στοιχείο για όλα τα UI elements στο Unity. Σε ένα άδειο Scene, πηγαίνουμε στο Hierarchy Window και κάνουμε δεξί κλικ, όπου εμφανίζεται ένα dropdown menu με διάφορες επιλογές. Επιλέγουμε UI → Canvas και αυτόματα δημιουργείται ένα νέο Canvas object στη σκηνή (βλ. Εικόνα 6-4).



Εικόνα 6-4 Δημιουργία Canvas

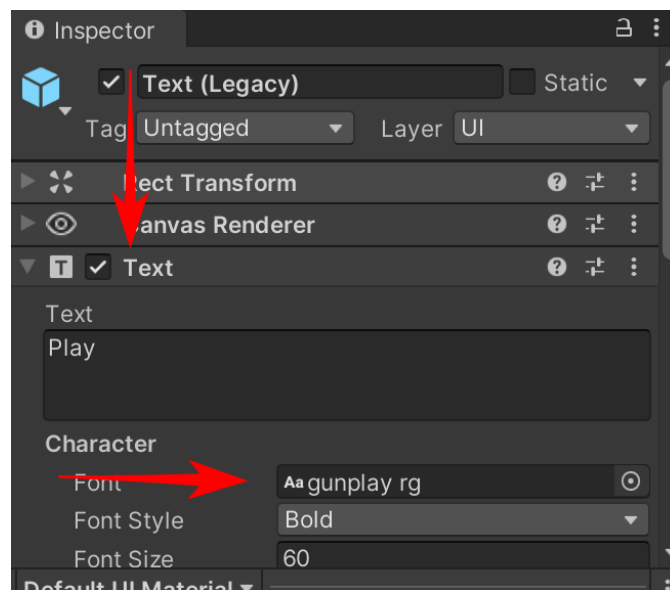
Το Canvas είναι ένας ειδικός τύπος GameObject που λειτουργεί ως ένας «καμβάς» στον οποίο ζωγραφίζονται όλα τα UI στοιχεία. Όταν δημιουργούμε ένα Canvas, το Unity δημιουργεί επίσης αυτόματα ένα EventSystem object, το οποίο είναι απαραίτητο για να λειτουργήσει σωστά το UI. Αφού δημιουργήσουμε το Canvas, εμφανίζεται μια λευκή ορθογώνια περιοχή στο Game View που αντιπροσωπεύει τα UI στοιχεία του Main Menu, ενώ στο Hierarchy βλέπουμε το Canvas ως ένα νέο GameObject με την αντίστοιχη δομή UI.

Για να τοποθετήσουμε μια εικόνα φόντου στο Main Menu, ξεκινάμε προσθέτοντας την εικόνα στα Assets του project. Στη συνέχεια, επιλέγουμε το Canvas στο Hierarchy και κάνουμε δεξί κλικ → UI → Raw Image, ώστε να δημιουργηθεί ένα νέο στοιχείο ως παιδί του Canvas. Στο Inspector του Raw Image, βρίσκουμε το πεδίο Texture και κάνουμε drag and drop την εικόνα από τα Assets μέσα σε αυτό (βλ. Εικόνα 6-5). Για να καλύπτει όλη την οθόνη, ρυθμίζουμε το Rect Transform ώστε να τεντώνεται σε όλο το Canvas. Με τον τρόπο αυτό, η εικόνα εμφανίζεται ως φόντο.



Εικόνα 6-5 Background Εικόνα

Στη συνέχεια, προσθέτουμε τα τρία κουμπιά επιλογών (Play, Game Controls, Quit Game) για το Main Menu. Η διαδικασία που ακολουθήθηκε είναι η εξής: δημιουργήσαμε ένα Button GameObject ως παιδί του Canvas. Κάνοντας δεξί κλικ στο Canvas και επιλέγοντας UI → Button, το Unity δημιουργεί αυτόματα ένα Button object με ένα child Text element. Στο παράθυρο του Inspector, στο πεδίο Text του Text component, γράψαμε το κείμενο που θέλαμε να εμφανίζεται στο κουμπί (π.χ. "Play") (βλ. Εικόνα 6-6).



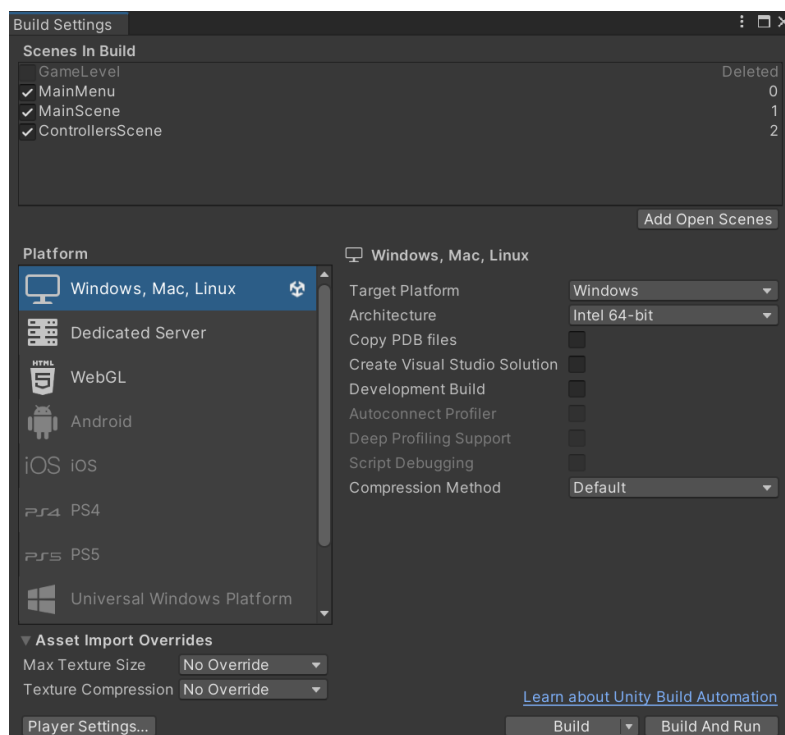
Εικόνα 6-6 Ρυθμίσεις Text

Στο Button component, ορίσαμε διαφορετικά χρώματα για τις καταστάσεις "Highlighted" και "Pressed", ώστε ο παίκτης να βλέπει μια άμεση οπτική ανταπόκριση όταν περνάει το ποντίκι πάνω από το κουμπί ή όταν το πατάει. Επαναλαμβάνουμε αυτή τη διαδικασία για καθένα από τα τρία κουμπιά του Main Menu (βλ. Εικόνα 6-7).



Εικόνα 6-7 Main Menu

Για να δώσουμε λειτουργικότητα σε όσα δημιουργήσαμε, προσθέσαμε ένα C# script στο Canvas του Main Menu (βλ. Εικόνα 6-9). Το script αυτό είναι υπεύθυνο για τη λειτουργικότητα των κουμπιών. Η μέθοδος για το PLAY αλλάζει σκηνή και μεταφέρει τον παίκτη από το Main Menu στο Game Scene, ενώ η μέθοδος για το GAME CONTROLS οδηγεί τον χρήστη σε μια ξεχωριστή σκηνή με τις οδηγίες των πλήκτρων. Για να λειτουργήσουν όλες αυτές οι αλλαγές, έπρεπε πρώτα να δηλώσουμε τις σκηνές στα Build Settings. Πήγαμε στο File → Build Settings και προσθέσαμε τις σκηνές μας με τη σειρά: Main Menu, Main Scene και Controls Scene (βλ. Εικόνα 6-8).



Εικόνα 6-8 Build Settings

```

public class MainMenu : MonoBehaviour
{
    public static MainMenu Instance;

    public void ExitGame()
    {
        Application.Quit();
        Destroy(this.gameObject);
    }

    public void StartGame()
    {
        SceneManager.LoadScene("MainScene");
    }

    public void Controller()
    {
        SceneManager.LoadScene("ControllersScene");
    }

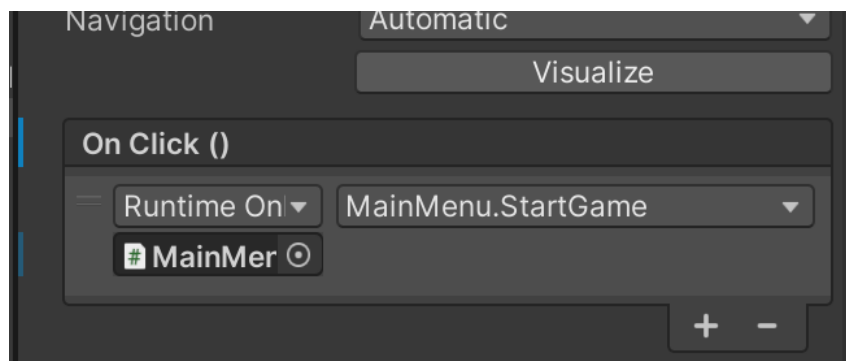
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }
}

```

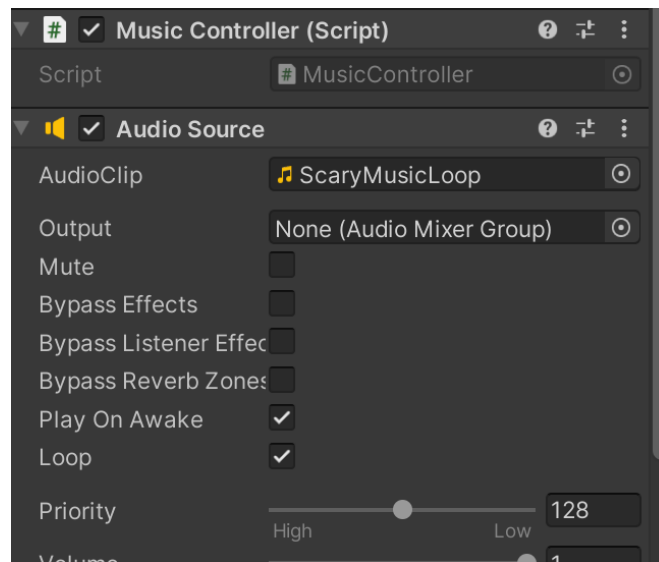
Εικόνα 6-9 Main Menu Script

Επίσης, δημιουργήσαμε μια μέθοδο QuitGame() που κλείνει την εφαρμογή και μια μέθοδο Back() που επιστρέφει στο Main Menu από το Controls Scene. Τέλος, συνδέσαμε κάθε κουμπί με την αντίστοιχη μέθοδό του μέσω του On Click event του Button component στο Inspector (βλ. Εικόνα 6-10).



Εικόνα 6-10 On Click() function

Για τη μουσική του Main Menu δημιουργήσαμε ένα ξεχωριστό GameObject, το οποίο ονομάσαμε MusicController. Προσθέσαμε ένα Audio Source component, το οποίο είναι υπεύθυνο για την αναπαραγωγή του ήχου. Στη συνέχεια, στο Audio Source τοποθετήσαμε το audio clip που επιλέξαμε για background music και ενεργοποιήσαμε δύο επιλογές: το "Play On Awake", ώστε η μουσική να ξεκινά αυτόματα κατά τη φόρτωση της σκηνής, και το "Loop", ώστε να παίζει συνεχώς σε επανάληψη (βλ. Εικόνα 6-11). Επιπλέον, προσθέσαμε ένα script το MusicController (βλ. Εικόνα 6-12) που ακολουθεί το Singleton pattern, διασφαλίζοντας έτσι ότι η μουσική θα συνεχίζει να παίζει ακόμα και κατά την αλλαγή σκηνής.



Εικόνα 6-11 Music Controller Settings

```

public class MusicController : MonoBehaviour
{
    2 references
    private static MusicController instance;

    0 references | @ Unity Message
    void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    // Start is called before the first frame update
    0 references | @ Unity Message
    void Start()
    {
    }
}

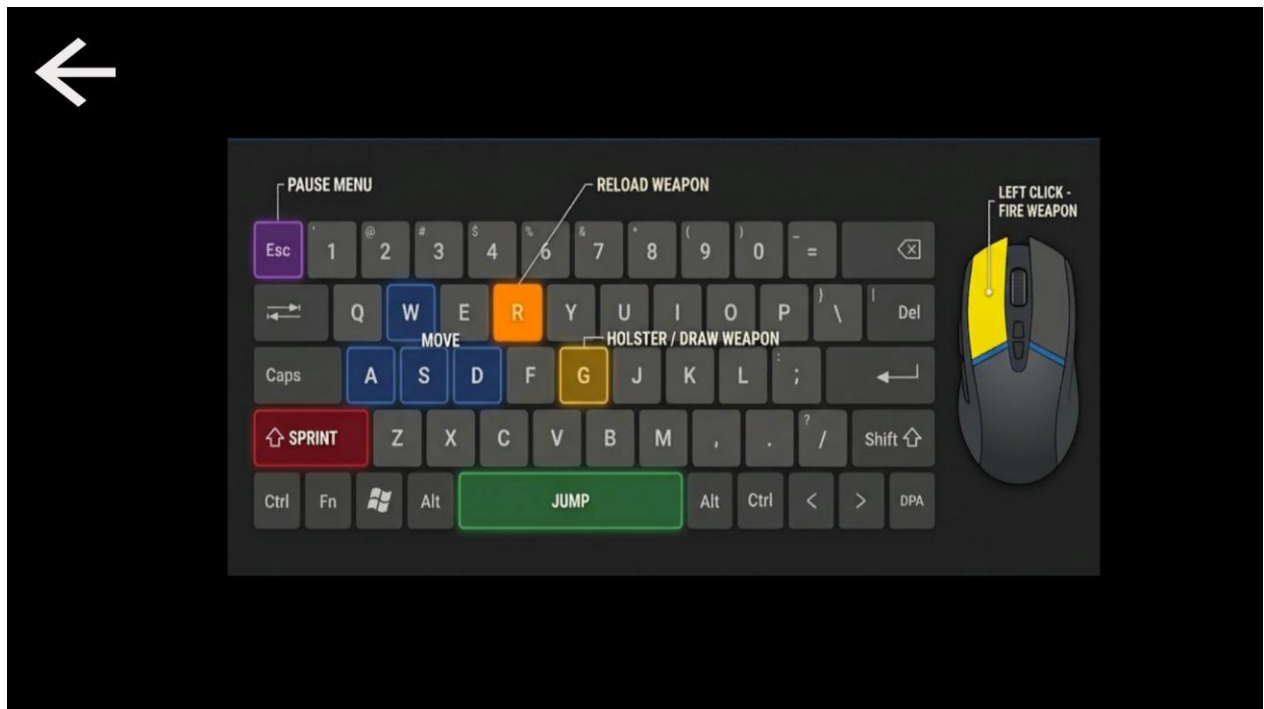
```

Εικόνα 6-12 Music Controller Script

6.3.2 Controller Menu

Για τη δημιουργία της σκηνής Game Controllers ακολουθήσαμε μια διαδικασία παρόμοια με αυτή του Main Menu. Συγκεκριμένα, δημιουργήσαμε ένα νέο Scene και το ονομάσαμε Controllers Scene. Στη σκηνή αυτή φτιάξαμε ένα Canvas και ορίσαμε το background του σε μαύρο. Στη συνέχεια, προσθέσαμε ένα Raw Image ως παιδί του Canvas, όπου τοποθετήσαμε την εικόνα που δείχνει τα πλήκτρα ελέγχου και τις εντολές του παιχνιδιού, κάνοντας drag and drop την εικόνα στο πεδίο Texture του Raw Image component, όπως ακριβώς κάναμε και με το φόντο του Main Menu (βλ. Εικόνα 6-13).

Τέλος, δημιουργήσαμε ένα Back button, το οποίο συνοδεύεται από μια εικόνα με ένα βέλος που δείχνει προς τα πίσω, ώστε να γίνεται άμεσα αντιληπτή η λειτουργία του. Το συνδέσαμε με το Main Menu script, ώστε όταν ο παίκτης το πατήσει να επιστρέφει στην κύρια σκηνή του μενού.



Εικόνες 6-13 Controllers Scene

6.3.3 Pause Menu

Για το Pause Menu, αντί να δημιουργήσουμε νέα σκηνή, το ενσωματώσαμε στο Canvas του MainScene. Έτσι, μπορούμε να το εμφανίζουμε και να το κρύβουμε εύκολα, χωρίς να αλλάζουμε σκηνή με το Main Scene να παραμένει ορατό στο background. Ξεκινήσαμε δημιουργώντας ένα Image GameObject που καλύπτει όλη την οθόνη, ώστε να λειτουργεί ως φόντο του μενού παύσης. Πάνω σε αυτό τοποθετήσαμε δύο κουμπιά: ένα "Resume" για να συνεχίζει το παιχνίδι και ένα "Exit to Main Menu" για να επιστρέφει στην αρχική οθόνη, με τον ίδιο τρόπο όπως στο Main Menu (βλ. Εικόνα 6-14).



Εικόνα 6-14 Pause Menu Screen

Η λειτουργία του ελέγχεται από το PauseMenu script (βλ. Εικόνα 6-15), το οποίο παρακολουθεί το πάτημα του πλήκτρου ESC. Όταν ο παίκτης πατάει ESC, το παιχνίδι παγώνει ($\text{Time.timeScale} = 0$), εμφανίζεται το μενού και ο κέρσορας γίνεται ορατός, ώστε ο χρήστης να μπορεί να επιλέξει κάποιο κουμπί. Το pause menu είναι αρχικά κρυφό (SetActive(false)) και εμφανίζεται μόνο όταν χρειάζεται. Πατώντας ξανά ESC, το μενού κρύβεται και το παιχνίδι συνεχίζεται.

```

1 reference
public void Resume()
{
    if (pauseMenuUI != null)
    {
        pauseMenuUI.SetActive(false);
    }

    Time.timeScale = 1f;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = true;
    isPaused = false;
}

1 reference
public void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;

    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;

    isPaused = true;
}

```

Εικόνα 6-15 Απόσπασμα από το PauseMenu Pause() και Resume() function

6.4 Terrain

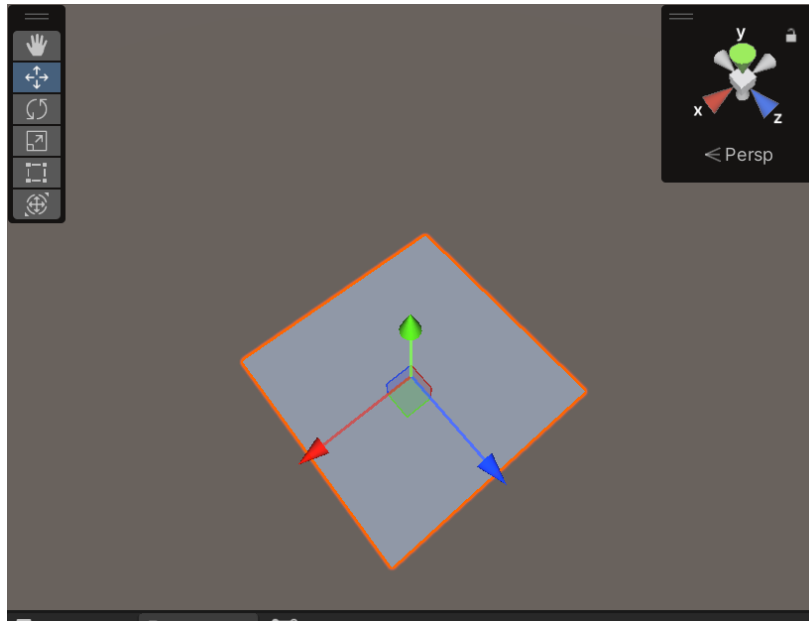
Το Terrain αποτελεί ένα από τα σημαντικότερα στοιχεία ενός παιχνιδιού, καθώς πάνω σε αυτό λαμβάνει χώρα όλη η δράση. Αναπαριστά έναν κόσμο μέσα στον οποίο ο παίκτης κινείται, συναντά εχθρούς και αναζητά πόρους. Ένα καλοσχεδιασμένο terrain δεν προσφέρει μόνο ένα ρεαλιστικό και ελκυστικό περιβάλλον, αλλά επηρεάζει ουσιαστικά και το ίδιο το gameplay — την κίνηση του παίκτη, τη συμπεριφορά των εχθρών, τον φωτισμό και τη συνολική αίσθηση του κόσμου. Το Unity παρέχει το Terrain Tool, ένα ισχυρό εργαλείο που επιτρέπει τη γρήγορη δημιουργία και διαμόρφωση τοπίων. Μέσω αυτού δημιουργήσαμε και σχεδιάσαμε το περιβάλλον του παιχνιδιού, με διαφορετικά υψόμετρα, υφές και δομές, κάνοντας το παιχνίδι πιο ενδιαφέρον και ευχάριστο για τον παίκτη.

Παρακάτω παρουσιάζονται αναλυτικά τα βήματα που ακολουθήθηκαν για τη δημιουργία του Terrain.

6.4.1 Δημιουργία και Διαμόρφωση Terrain

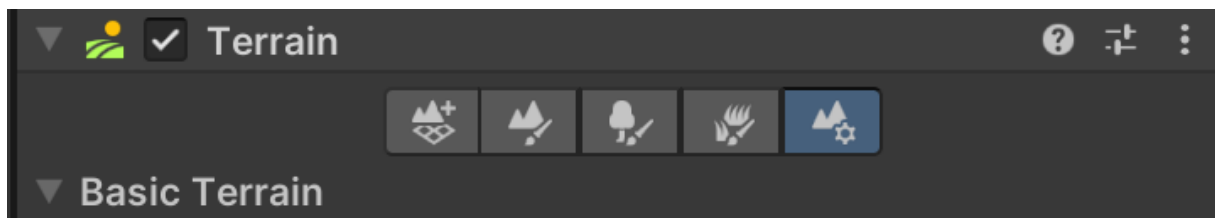
Ξεκινώντας τη δημιουργία του Terrain, φτιάξαμε πρώτα το κύριο Scene και το ονομάσαμε MainScene. Σε αυτό, κάνοντας δεξί κλικ στο Hierarchy Window και επιλέγοντας 3D Object → Terrain, το Unity

δημιούργησε αυτόματα ένα Terrain mesh μέσα στη σκηνή, το οποίο θα αποτελέσει τον κεντρικό κόσμο του παιχνιδιού (βλ. Εικόνα 6-16).

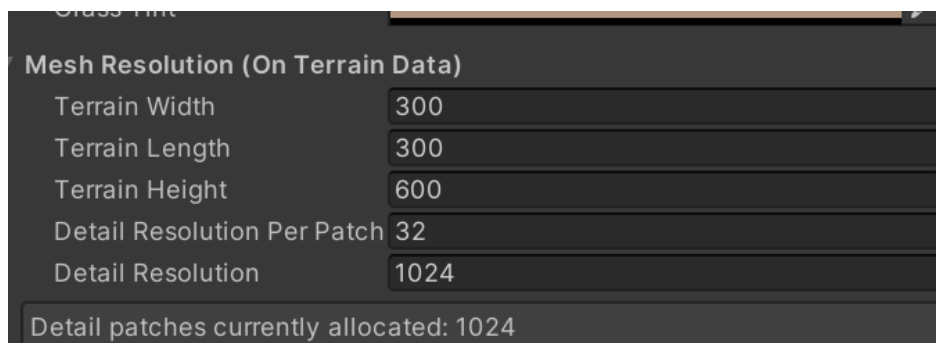


Εικόνα 6-16 Δημιουργία Terrain

Στη συνέχεια, πήγαμε στο Terrain Tool που εμφανίζεται στον Inspector και επιλέξαμε το Terrain Settings (βλ. Εικόνα 6-17), το οποίο περιέχει διάφορες επιλογές σχετικά με το Terrain. Μία από αυτές αφορά το μέγεθός του. Το προεπιλεγμένο μέγεθος ήταν 1000×1000, το οποίο κρίθηκε υπερβολικά μεγάλο για τις ανάγκες μας, έτσι το προσαρμόσαμε σε 300×300 μονάδες (βλ. Εικόνα 6-18).

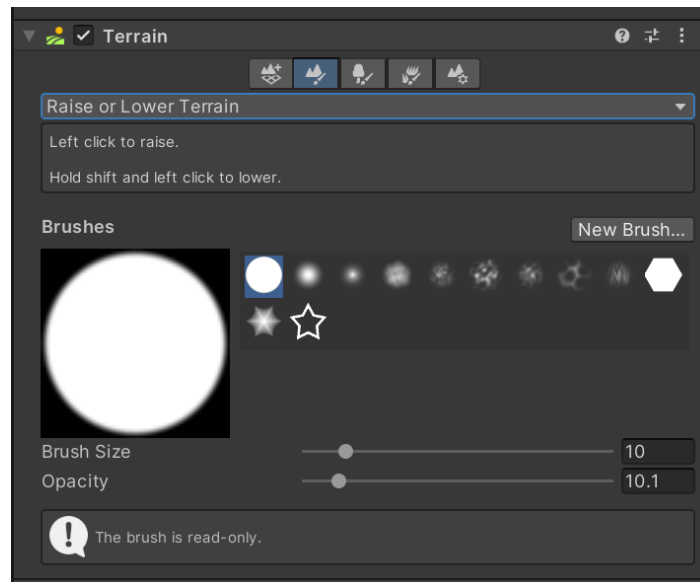


Εικόνα 6-17 Terrain Settings στο Terrain Tool



Εικόνα 6-18 Διαστάσεις Terrain

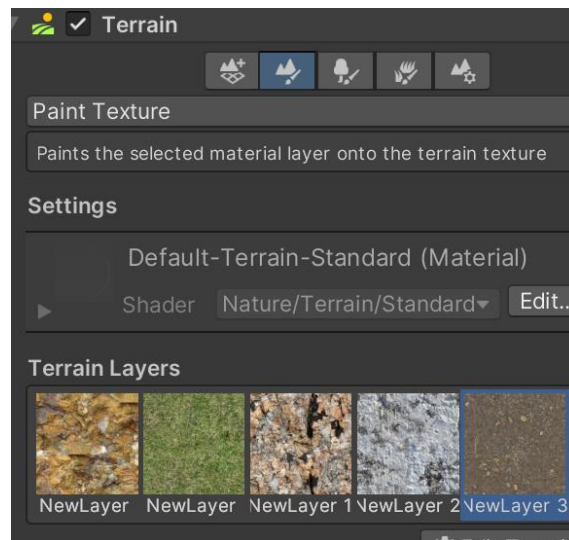
Μέσα από το Terrain Tool, επιλέξαμε την καρτέλα Paint Terrain, όπου υπάρχει ένα αναπτυσσόμενο μενού με διάφορες επιλογές διαμόρφωσης. Από αυτές, επιλέξαμε το εργαλείο Raise and Lower Terrain, το οποίο επιτρέπει την ανύψωση ή τη μείωση του ύψους του εδάφους. Το εργαλείο αυτό λειτουργεί με τη χρήση ενός brush, το οποίο διαθέτει τρεις βασικές παραμέτρους: το σχήμα του brush, το μέγεθος (brush size) που καθορίζει την επιφάνεια επίδρασης, και το opacity, που ρυθμίζει την ένταση της επίδρασης. Χρησιμοποιώντας αυτό το εργαλείο, σμιλεύουμε το Terrain, δημιουργώντας ένα ποικιλόμορφο και ενδιαφέρον τοπίο (βλ. Εικόνα 6-19).



Εικόνες 6-19 Paint Terrain και Brush Settings

6.4.2 Paint Texture - Χρωματισμός του Τοπίου

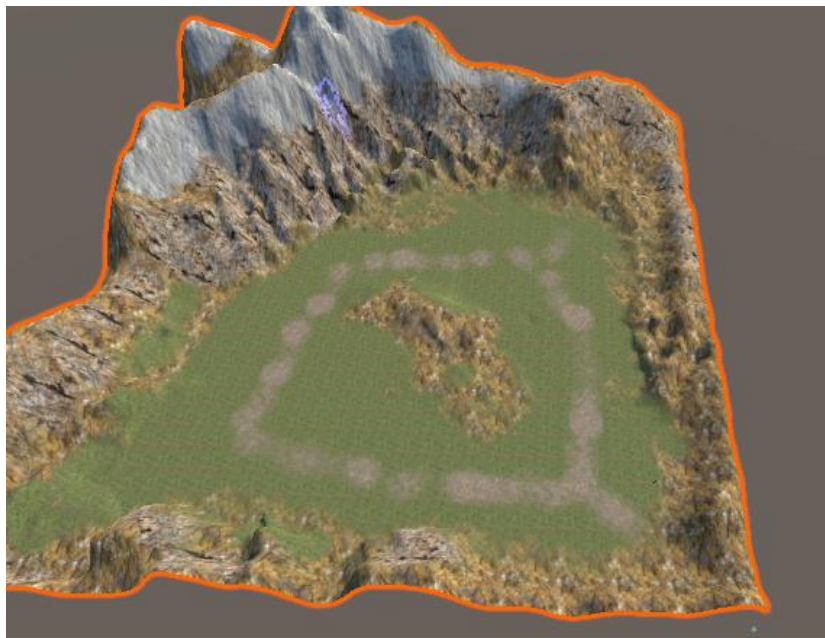
Στη συνέχεια, ήρθε η σειρά να περάσουμε τα textures στο Terrain. Στο Terrain Tool, από το dropdown menu, αντί για την επιλογή Raise and Lower Terrain, επιλέξαμε Paint Texture (βλ. Εικόνα 6-20). Αυτή η επιλογή μας επιτρέπει να εφαρμόσουμε 2D textures σε 3D χώρο, ακριβώς όπως αναφέραμε στο Κεφάλαιο 3.2.1 (Τρισδιάστατη Γραφική και το Rendering Pipeline). Στα assets του project προσθέσαμε διάφορα textures που χρησιμοποιήσαμε για αυτή τη διαδικασία. Για να έχουμε καλύτερο αποτέλεσμα, ρυθμίσαμε στα Tiling Settings όλων των textures την επιλογή 20×20, ώστε να φαίνονται καλύτερα στον χάρτη. Χρησιμοποιήσαμε το εργαλείο Paint Texture με το brush που αναφέρθηκε προηγουμένως, ρυθμίζοντας κάθε φορά το μέγεθος και την έντασή του για ακρίβεια, και εφαρμόσαμε κάθε texture σταδιακά στις διαφορετικές περιοχές του Terrain, δημιουργώντας έτσι ένα ποικίλο και ρεαλιστικό τοπίο.



Εικόνα 6-20 Paint Texture Setting και Texture

Textures που Χρησιμοποιήθηκαν (Με τη Σειρά Εφαρμογής):

1. **Atex_Rock 9** - Βάση του terrain, αίσθηση χώματος και φύσης.
2. **Atex_grass 3** - Γρασίδι στο κεντρικό κομμάτι του terrain.
3. **Attex_Rock 19** - Ψηλά σημεία και κορυφές βουνών.
4. **Atex_snowDirt 2** - Χιόνι και παγετός στις κορυφές των βουνών.
5. **Terrain Mud DryAlbedo** - Μονοπάτι λάσπης στο κεντρικό γρασίδι.

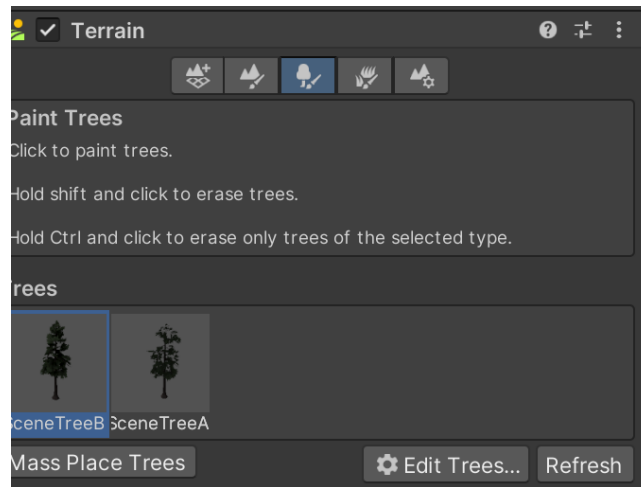


Εικόνα 6-21 Το τελικό terrain με όλα τα textures εφαρμοσμένα

6.4.3 Τοποθέτηση Δέντρων και Θάμνων στο Terrain

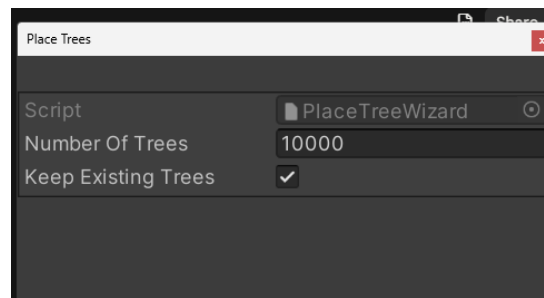
Για να δώσουμε μια ακόμη πιο ρεαλιστική αίσθηση δάσους, προσθέσαμε δέντρα και θάμνους μέσα στο Terrain. Ξεκινήσαμε με τα δέντρα, στο Terrain Tool επιλέξαμε την επιλογή Paint Trees που βρίσκεται

στο dropdown menu. Κάτω από αυτή την επιλογή υπάρχει ένα παράθυρο με την ονομασία Trees, όπου μπορούμε να προσθέσουμε ή να διαγράψουμε tree assets που θέλουμε να χρησιμοποιήσουμε. Προσθέσαμε τα δέντρα που επιθυμούσαμε από τα assets μας (βλ. Εικόνα 6-22).



Εικόνα 6-22 Paint Tree Settings

Το Unity παρέχει δύο διαφορετικές μεθόδους για την τοποθέτηση δέντρων. Η πρώτη επιλογή είναι η Mass Place Trees, η οποία τοποθετεί δέντρα αυτόματα σε τυχαία σημεία μέσα στο Terrain, και εσύ ορίζεις τον αριθμό των δέντρων που θέλεις να προστεθούν, χρησιμοποιώντας το ενσωματωμένο script Place Tree Wizard (βλ. Εικόνα 6-23). Η δεύτερη επιλογή είναι το manual placement με το εργαλείο brush, το οποίο σου επιτρέπει να τοποθετήσεις δέντρα ένα προς ένα ή να τα αφαιρέσεις από συγκεκριμένες θέσεις.



Εικόνα 6-23 Mass Place Tree Settings

Εμείς χρησιμοποιήσαμε και τις δύο μεθόδους για το καλύτερο αποτέλεσμα: πρώτα τοποθετήσαμε μαζικά δέντρα στο Terrain (βλ. Εικόνα 6-24) με το Place Tree Wizard και, στη συνέχεια, χρησιμοποιήσαμε το brush για να προσθέσουμε και να αφαιρέσουμε δέντρα σε διάφορα σημεία, δημιουργώντας έναν πιο φυσικό και ποικίλο δάσος με καλύτερη κατανομή της βλάστησης.

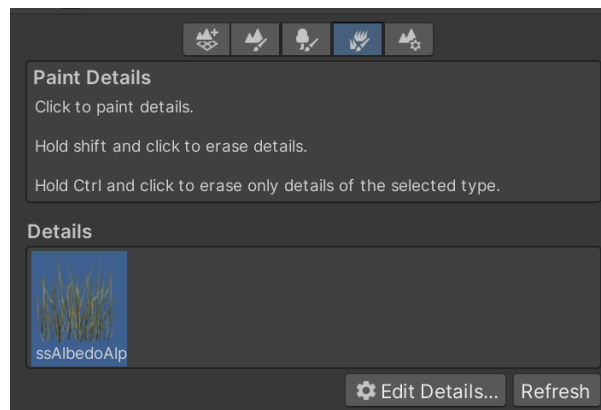
Asset trees που Χρησιμοποιήθηκαν:

1. SceneTreeB
2. SceneTreeA



Εικόνα 6-24 Τα δέντρα όπως φαίνονται από το Scene

Για τη δημιουργία θάμνων και λεπτομερειών του τοπίου, χρησιμοποιήσαμε την επιλογή Paint Details από το Terrain Tool (βλ. Εικόνα 6-25). Η διαδικασία είναι παρόμοια με αυτή των δέντρων, καθώς προσθέτουμε στη λίστα Details τα assets που θέλουμε να χρησιμοποιήσουμε. Επιλέξαμε και προσθέσαμε τα κατάλληλα detail assets από τα assets μας που αντιπροσωπεύουν θάμνους.

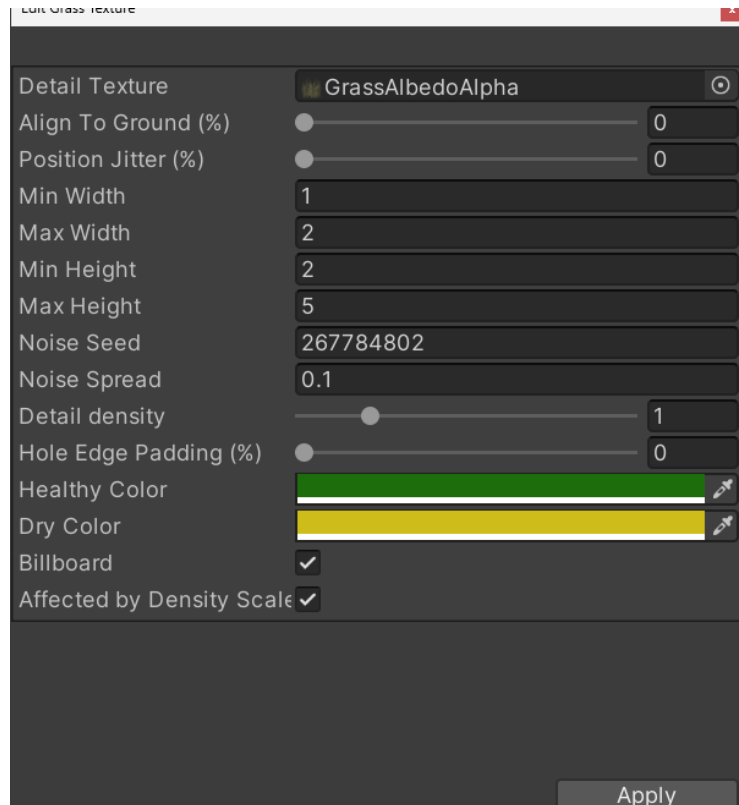


Εικόνα 6-25 Paint Details Setting

Η μόνη διαφορά με τα δέντρα είναι ότι για τα Details η μόνη διαθέσιμη επιλογή είναι το manual placement με το εργαλείο brush, χωρίς την επιλογή mass placement. Στο παράθυρο Edit Details (βλ. Εικόνα 6-26), μπορούμε να ρυθμίσουμε διάφορες παραμέτρους του επιλεγμένου detail asset, όπως το ύψος που θα εμφανίζονται οι θάμνοι, το χρώμα που θέλουμε να έχουν και άλλες ιδιότητες. Μόλις κάνουμε τις ρυθμίσεις στο Edit Details, αυτές εφαρμόζονται αυτόματα στο επιλεγμένο asset, επιτρέποντάς μας να δούμε αμέσως τις αλλαγές καθώς ζωγραφίζουμε τα details στο Terrain με το brush.

Details που Χρησιμοποιήθηκαν:

1. grassAlbedoAlpha

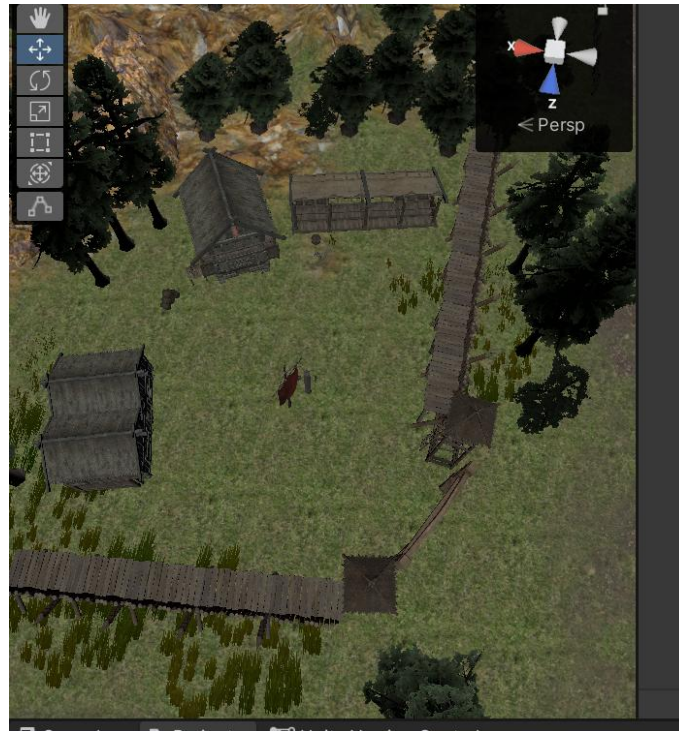


Εικόνα 6-26 Details Settings Window

6.4.4 Αρχιτεκτονικά Στοιχεία Terrain

Ένα επιπλέον σημαντικό στάδιο για τη διαμόρφωση ενός ρεαλιστικού περιβάλλοντος Terrain αποτελεί η προσθήκη κτιρίων και κατασκευών, τα οποία ενισχύουν την αληθοφάνεια του χάρτη. Για το σκοπό αυτό, αξιοποιήθηκαν έτοιμα ψηφιακά στοιχεία (assets) από εξωτερικές βιβλιοθήκες, τα οποία τοποθετήθηκαν μεμονωμένα ή σε κατάλληλους συνδυασμούς για τη δημιουργία σύνθετων δομών και δομημένων χώρων.

Η διαδικασία ενσωμάτωσής τους πραγματοποιήθηκε με τη μέθοδο drag and drop των αντικειμένων εντός της σκηνής. Η ακριβής τοποθέτηση και ο χωρικός προσανατολισμός τους υλοποιήθηκαν μέσω των εργαλείων του παραθύρου Scene (βλ. Εικόνα 6-27).



Εικόνα 6-27 Περιοχή του terrain με τοποθετημένα κτήρια

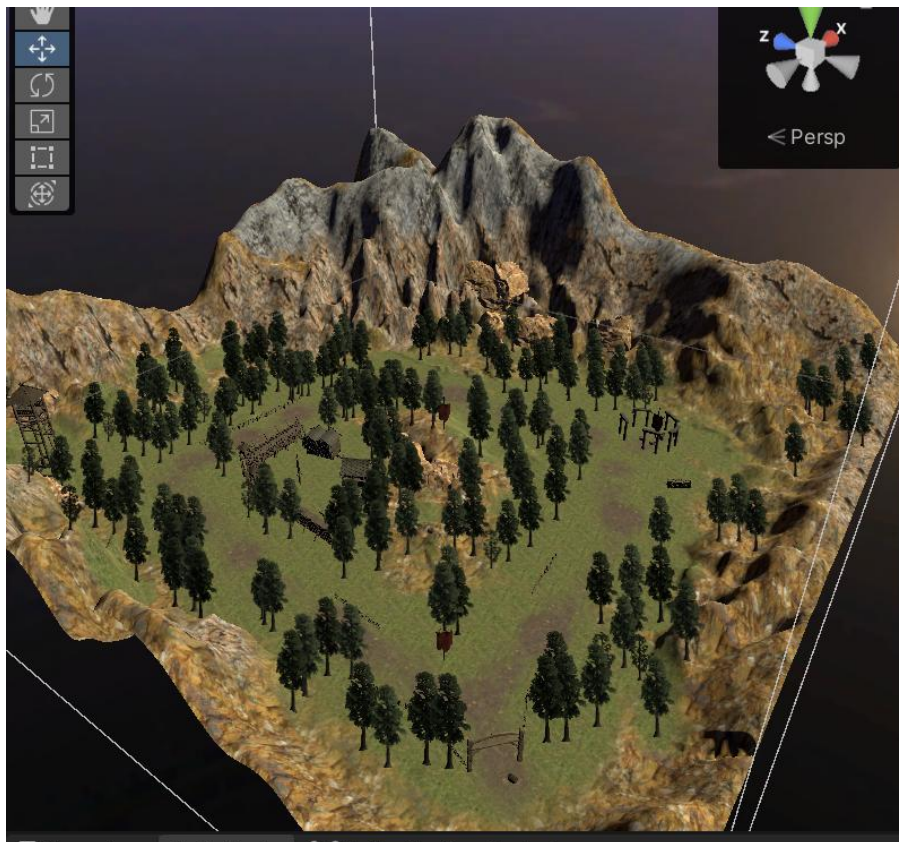
Για να εξασφαλίσουμε τη φυσική αλληλεπίδραση, επιλέξαμε κάθε asset μέσα στο Hierarchy Window και πήγαμε στο Inspector. Έπειτα, κάναμε κλικ στην επιλογή Add Component και προσθέσαμε Colliders (π.χ. Box Colliders) σε κάθε κατασκευή, προσαρμοσμένα στο σχήμα τους (βλ. Εικόνα 6-28). Με αυτόν τον τρόπο, ο παίκτης δεν μπορεί να περάσει μέσα από τα κτήρια, δημιουργώντας φυσικά εμπόδια που ενισχύουν τον ρεαλισμό και βελτιώνουν την εμπειρία του παιχνιδιού.



Εικόνα 6-28 Box Collider όπως φαίνεται εντός παιχνιδιού

Κτήρια και Κατασκευές που Χρησιμοποιήθηκαν:

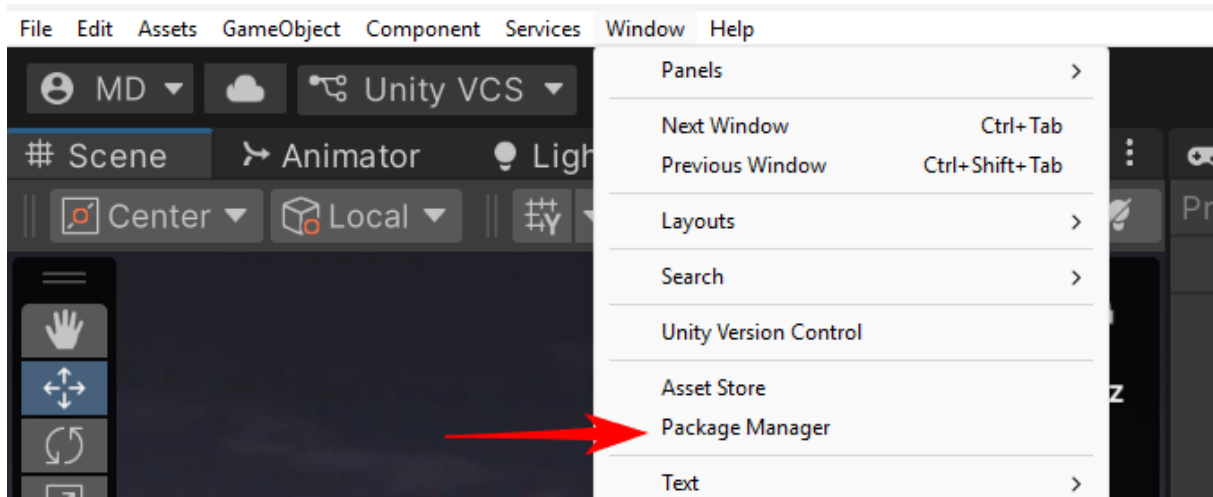
2. TowerWatch
3. Rock04
4. Altar
5. Shed02
6. Boulder01
7. Plank03
8. Pillar
9. Fence01
10. Barracks
11. HouseBig01
12. Flag01
13. MainGate
14. Barrel01
15. WallPanel01
16. Storage01
17. Gate
18. Tower
19. Shed01
20. Bucket01
21. Wheel02
22. Shovel01
23. Barrels02
24. Pillar01
25. Plank01



Εικόνα 6-29 Άνω όψη του ολοκληρωμένου terrain με όλα τα στοιχεία

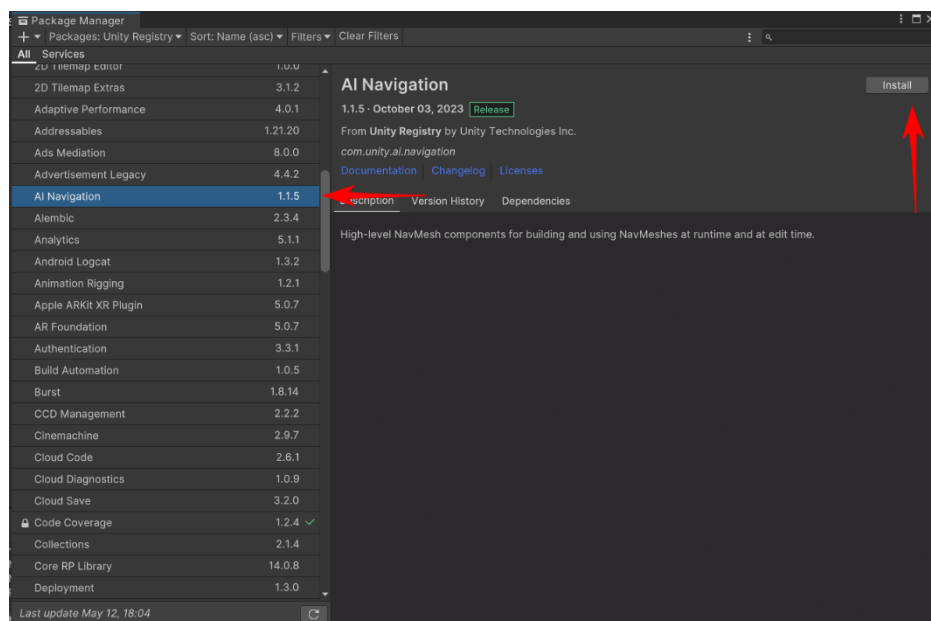
6.4.5 NavMesh

Το τελευταίο και πολύ σημαντικό κομμάτι του Terrain είναι η Navigation, ή αλλιώς NavMesh, όπως αναφέραμε στο Κεφάλαιο 3.3.2 για την Τεχνητή Νοημοσύνη και τα Συστήματα Πλοήγησης. Τα ζόμπι, για τη σωστή τους λειτουργία και το pathfinding, χρειάζονται το NavMesh, ώστε να λειτουργήσει σωστά ο αλγόριθμος A*. Αυτό το σύστημα επιτρέπει στα ζόμπι να κινούνται στο Terrain με ρεαλιστικό και αποδοτικό τρόπο. Το πρώτο βήμα για τη δημιουργία του NavMesh είναι να πάμε στις πάνω επιλογές του Unity και να επιλέξουμε Window → Package Manager (βλ. Εικόνα 6-30).



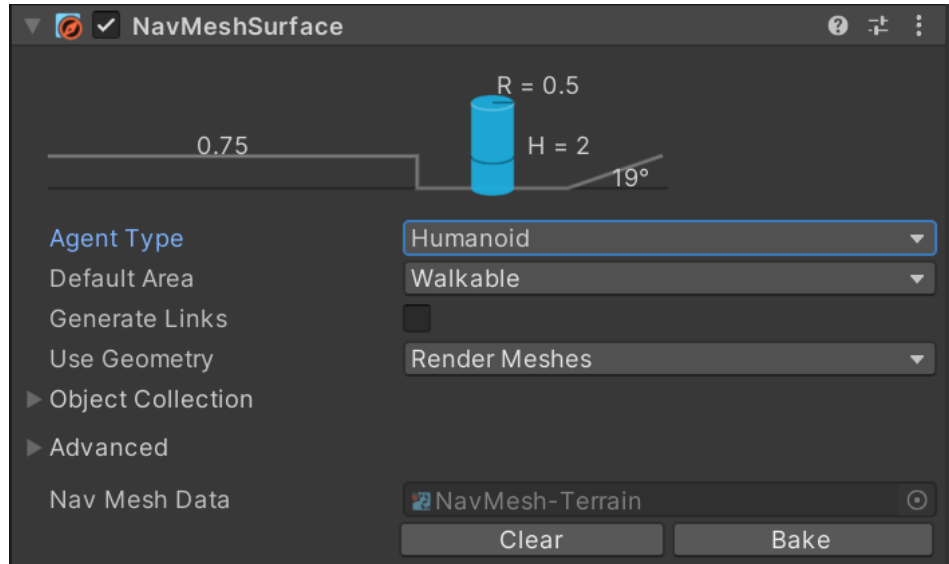
Εικόνα 6-30 Window - Package Manager

Ανοίγει ένα νέο παράθυρο αναζήτησης πακέτων. Στο search box του Package Manager πληκτρολογούμε "AI" και από τις επιλογές που εμφανίζονται επιλέγουμε AI Navigation, το οποίο είναι το απαραίτητο package για τη δημιουργία και τη διαχείριση του NavMesh στο Unity (βλ. Εικόνα 6-31).



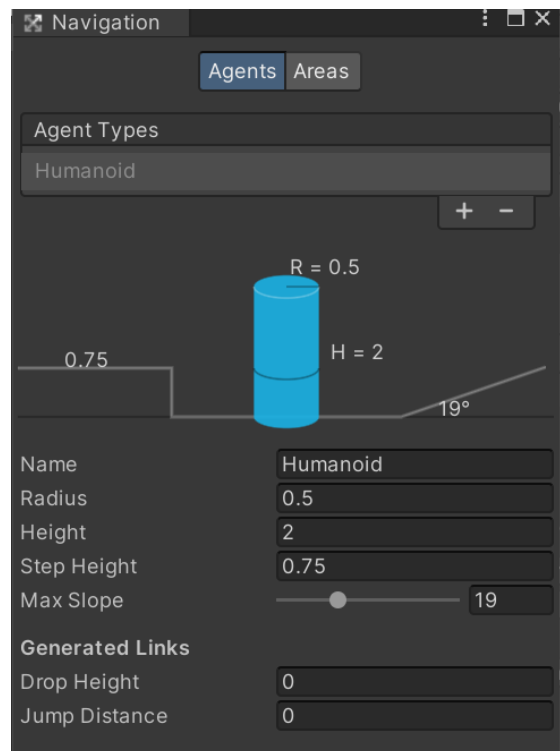
Εικόνα 6-31 Εγκατάσταση AI Navigation package

Μόλις ολοκληρωθεί η εγκατάσταση του AI Navigation package, επιλέγουμε το Terrain στο Hierarchy. Στη συνέχεια, στο Inspector πατάμε Add Component, αναζητούμε "NavMesh" και επιλέγουμε NavMeshSurface. Έτσι, προσθέτουμε το νέο component στο Terrain.



Εικόνα 6-32 NavMeshSurface Component

Εντός του NavMeshSurface υπάρχει η επιλογή Agent Type, όπου μπορούμε να ρυθμίσουμε παραμέτρους όπως η μέγιστη κλίση (Max Slope) που μπορούν να ανέβουν τα ζόμπι, το ύψος που χρειάζονται για να περάσουν κάτω από εμπόδια και άλλες ρυθμίσεις κίνησης. Εμείς ορίσαμε τις παρακάτω τιμές, όπως φαίνεται στην Εικόνα 6-33.



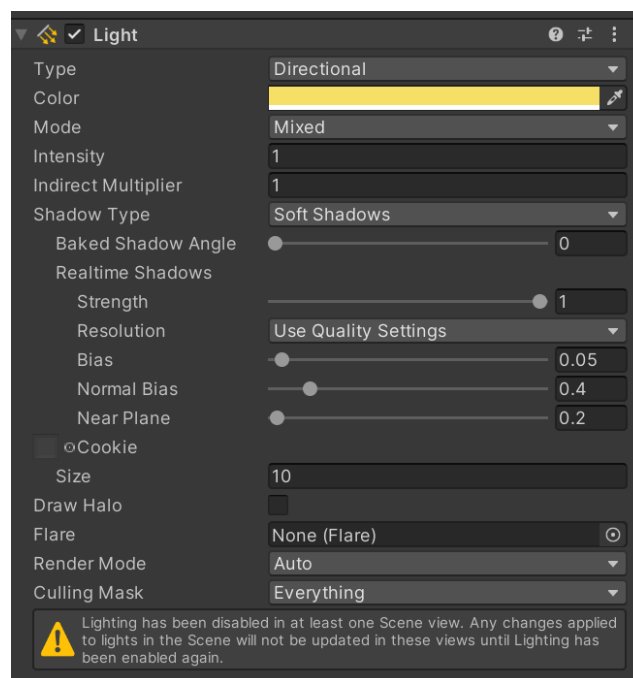
Εικόνα 6-33 Selected Agent Type Settings

Τέλος, επιλέγουμε το κουμπί Bake, το οποίο βρίσκεται κάτω δεξιά όπως φαίνεται στην Εικόνα 6-32, και δημιουργεί το NavMesh πάνω στο Terrain. Το Unity υπολογίζει και αποθηκεύει τον χάρτη πλοήγησης, τον οποίο θα χρησιμοποιήσουν τα ζόμπι για να βρίσκουν τον δρόμο τους μέσα στον κόσμο.

6.5 Φωτισμός & Ατμόσφαιρα

6.5.1 Directional Light

Το Directional Light αποτελεί την κύρια πηγή φωτισμού σε μια σκηνή, προσομοιώνοντας τον ήλιο και φωτίζοντας το περιβάλλον με παράλληλες ακτίνες από μια σταθερή κατεύθυνση. Στο Unity, κάθε νέα σκηνή δημιουργείται αυτόματα με ένα Directional Light. Επομένως, δεν χρειάστηκε να προσθέσουμε καινούργιο καθώς αξιοποιήσαμε το υπάρχον και προσαρμόσαμε τις παραμέτρους του στις ανάγκες του παιχνιδιού (βλ. Εικόνα 6-34). Η βασική αλλαγή που κάναμε ήταν στο χρώμα του φωτός, επιλέγοντας μια πιο ψυχρή κιτρινωπή απόχρωση, ώστε να δημιουργηθεί μια αίσθηση δύσης.



Εικόνα 6-34 Directional Light Settings

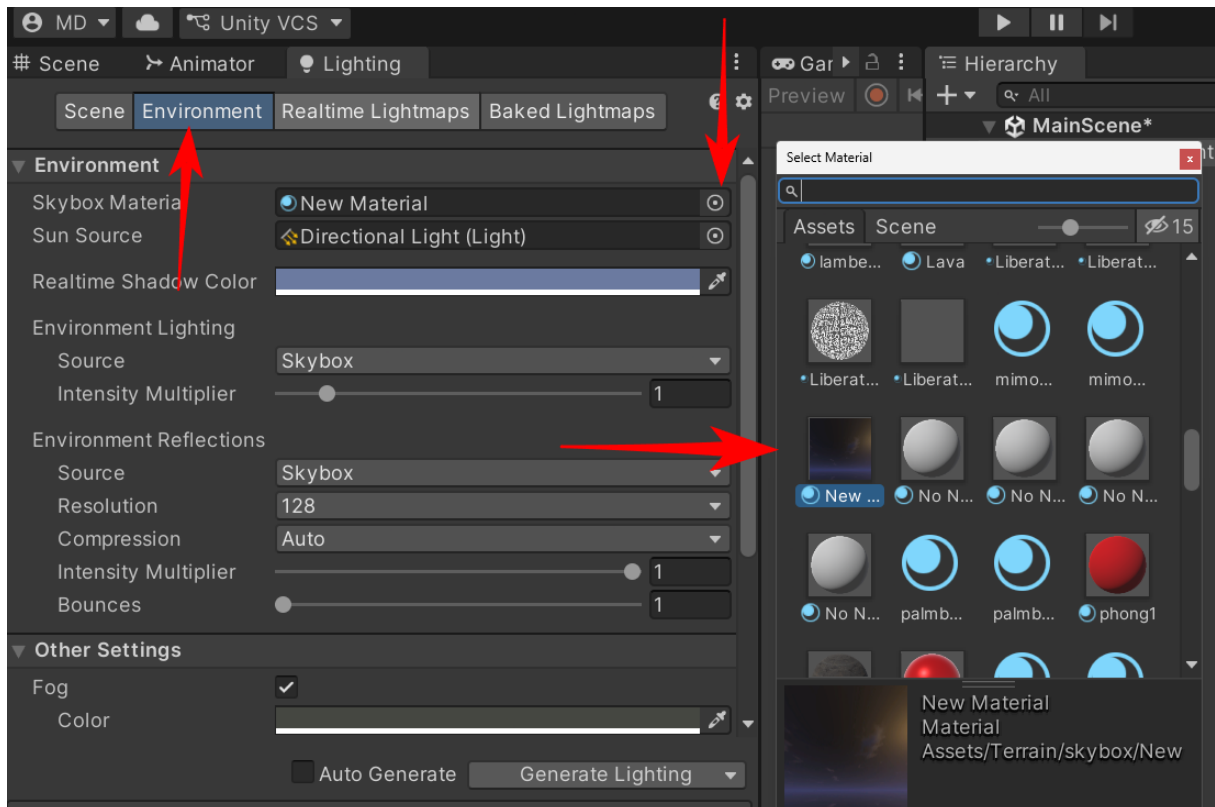
6.5.2 Skybox Panorama

Το Skybox Panorama είναι μια εικόνα ή μια σύνθεση εικόνων που χρησιμοποιείται για να αποδώσει τον ουρανό και το ευρύτερο περιβάλλον γύρω από τη σκηνή. Αντί να κρατήσουμε τον απλό μπλε ουρανό που προσφέρει το Unity από προεπιλογή, επιλέξαμε να δημιουργήσουμε ένα custom skybox, βασισμένο σε μια πανοραμική εικόνα, ώστε να δώσουμε περισσότερη ατμόσφαιρα στον κόσμο μας, όπως φαίνεται στην Εικόνα 6-35.



Εικόνα 6-35 Custom Sky Box Scene View

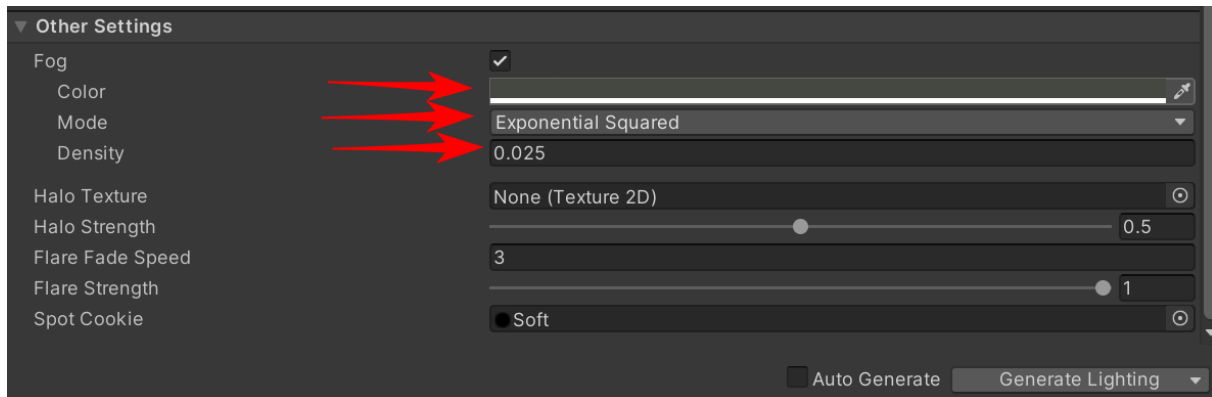
Για να το πετύχουμε αυτό, ανοίξαμε το παράθυρο Window → Rendering → Lighting και στις ρυθμίσεις περιβάλλοντος (Environment) αντικαταστήσαμε το προεπιλεγμένο υλικό του skybox με ένα δικό μας, φορτώνοντας την εικόνα panorama που είχαμε επιλέξει (βλ. Εικόνα 6-36).



Εικόνα 6-36 Environment Lighting ρυθμίσεις - Skybox

6.5.3 Fog (Ομίχλη)

Τέλος, για να ενισχύσουμε τη ρεαλιστική και απειλητική ατμόσφαιρα του τοπίου, προσθέσαμε ομίχλη. Στο παράθυρο Lighting Settings, στην ενότητα Environment, ενεργοποιήσαμε την επιλογή Fog. Αρχικά, επιλέξαμε ένα σκούρο γκρι χρώμα για το χρώμα της ομίχλης. Έπειτα, από τις διαθέσιμες επιλογές Mode, επιλέξαμε το Exponential Squared, το οποίο δημιουργεί μια πιο έντονη αίσθηση ομίχλης, και ορίσαμε την τιμή Density στο 0.025, η οποία ελέγχει το πόσο πυκνή θα εμφανίζεται η ομίχλη στον χώρο (βλ. Εικόνα 6-37).



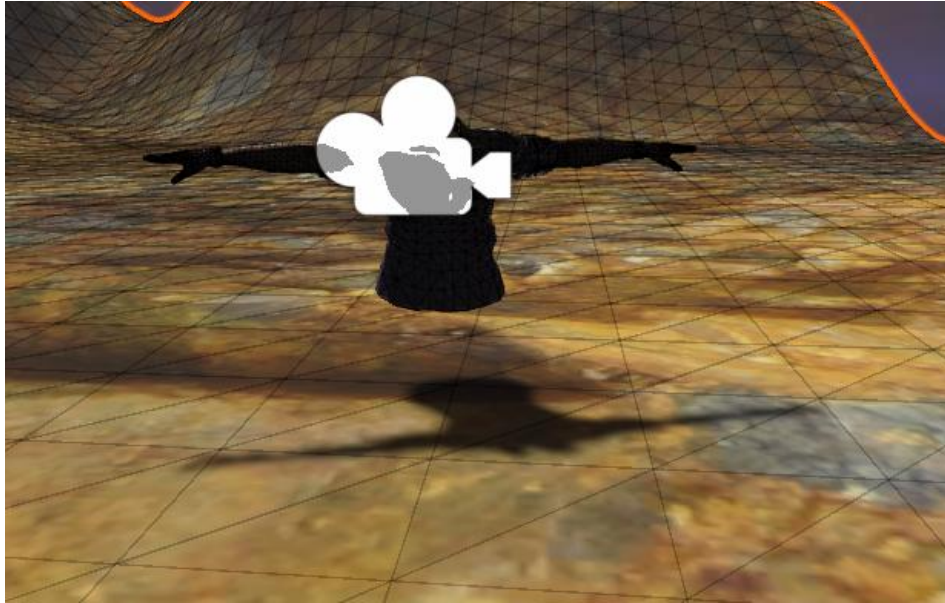
Εικόνα 6-37 Fog Settings

6.6 Player

6.6.1 Δημιουργία και Σχεδιασμός Χαρακτήρα Παίκτη

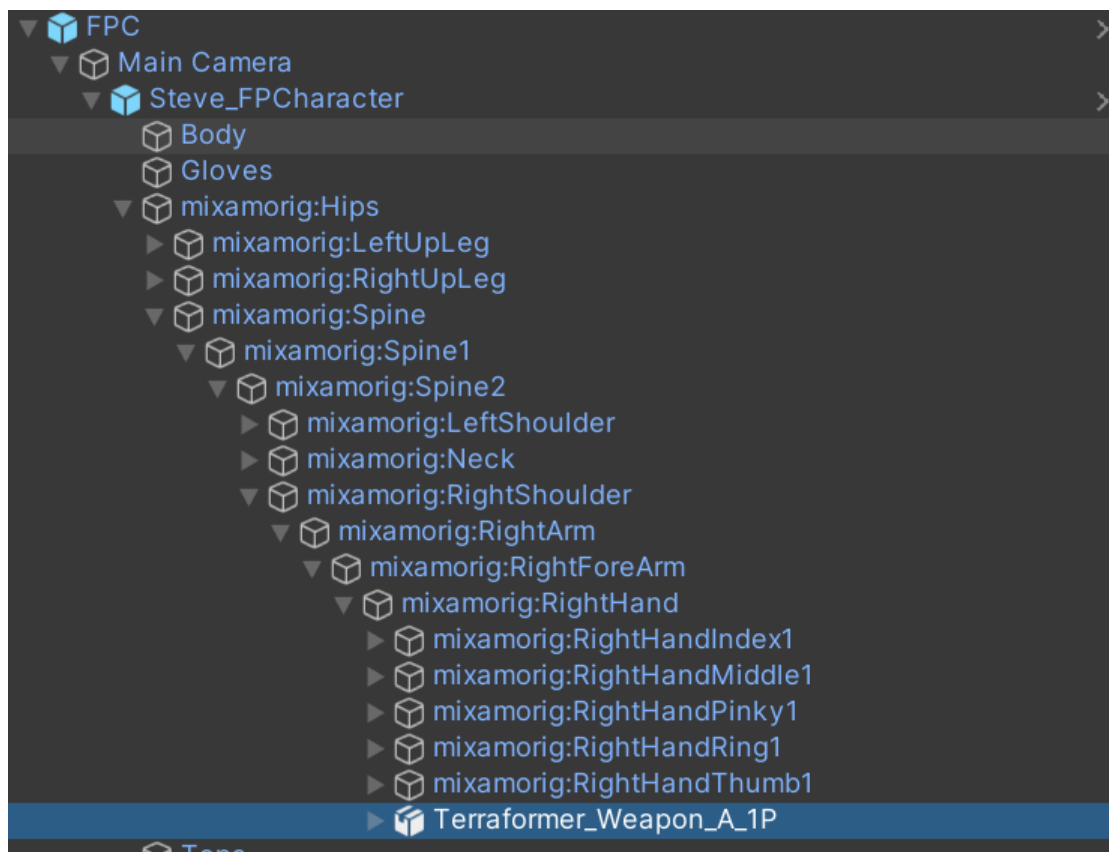
Η δημιουργία του χαρακτήρα ξεκίνησε με την τοποθέτηση ενός Capsule GameObject στη σκηνή, το οποίο θα αποτελέσει τον βασικό κορμό του παίκτη. Έπειτα, επιλέχθηκε το αντικείμενο αυτό από το Hierarchy και στο παράθυρο του Inspector προστέθηκαν τα απαραίτητα συστατικά φυσικής, σύμφωνα με όσα αναφέρθηκαν στην ενότητα 3.2.2 για τις Μηχανές Φυσικής και την Ανίχνευση Συγκρούσεων. Συγκεκριμένα, τοποθετήθηκε ένα Rigidbody component για την προσομοίωση φυσικών δυνάμεων και ένα Capsule Collider για τον ορισμό του όγκου σύγκρουσης.

Για την οπτική αναπαράσταση του παίκτη, χρησιμοποιήθηκε ένα τρισδιάστατο μοντέλο με το όνομα "Steve". Το μοντέλο ευθυγραμμίστηκε με τη θέση του Capsule και στη συνέχεια ορίστηκε ως child του. Με αυτόν τον τρόπο, το μοντέλο ακολουθεί πιστά τις κινήσεις του capsule και κληρονομεί όλες τις ιδιότητες που είχαν οριστεί προηγουμένως, όπως το Rigidbody και το Capsule Collider, εξασφαλίζοντας έτσι την ενιαία και συνεπή συμπεριφορά του χαρακτήρα στο φυσικό περιβάλλον του παιχνιδιού (βλ. Εικόνα 6-38).



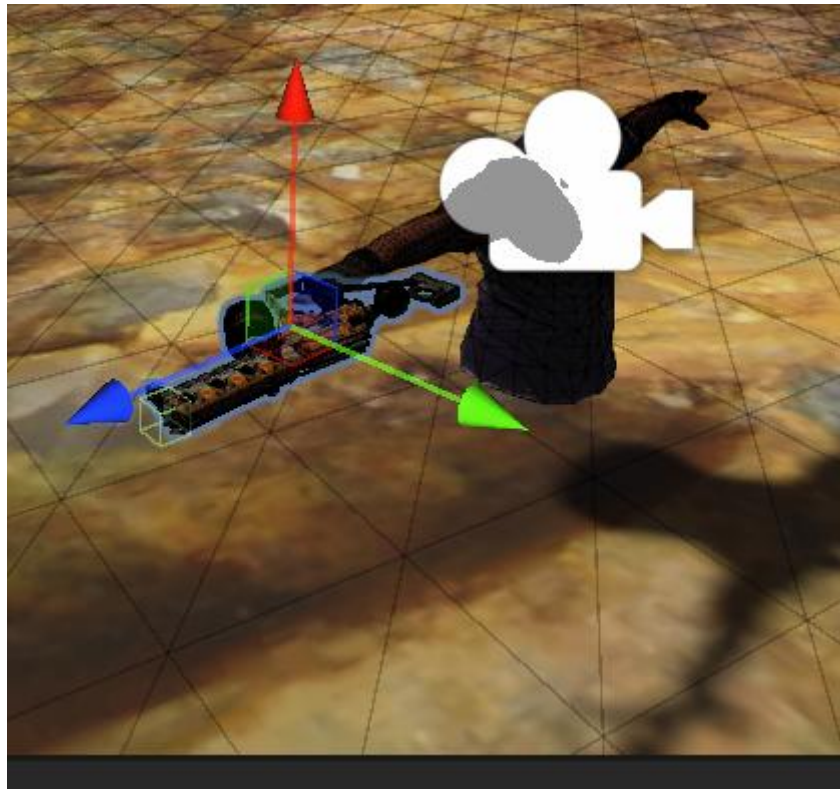
Εικόνα 6-38 Steve Model

Ακολουθως, πραγματοποιήθηκε η ενσωμάτωση του όπλου στον χαρακτήρα. Για την ακριβή τοποθέτησή του, αναπτύχθηκε η ιεραρχική δομή των οστών του μοντέλου μέχρι το δεξί άκρο (βλ. Εικόνα 6-39).



Εικόνα 6-39 Hierarchy δομή του παίκτη

Το αντικείμενο του όπλου τοποθετήθηκε ως εξάρτημα στο συγκεκριμένο σημείο και ευθυγραμμίστηκε προσεκτικά, ώστε να επιτευχθεί η φυσική συγκράτηση και η σωστή προβολή του κατά τη διάρκεια του παιχνιδιού (βλ. Εικόνα 6-40).



Εικόνα 6-40 Ο χαρακτήρας Steve στη σκηνή με το όπλο

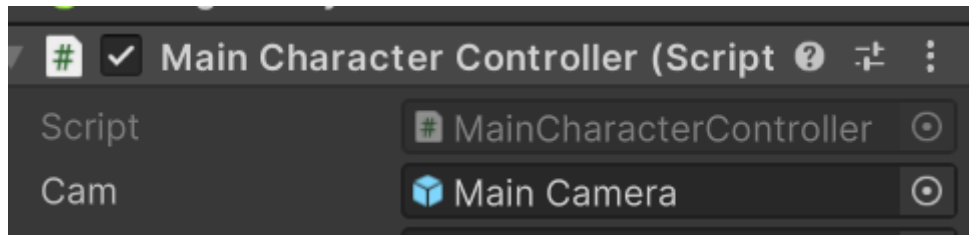
6.6.2 Σύστημα Κάμερας

Η κάμερα αποτελεί το σημαντικότερο στοιχείο ενός FPS παιχνιδιού, καθώς λειτουργεί ως τα «μάτια» του παίκτη, όπως αναφέραμε και στο Κεφάλαιο 2.4.1 για τα Παιχνίδια Δράσης – Action Games. Η κάμερα υλοποιήθηκε ως ξεχωριστό GameObject, το οποίο τοποθετήθηκε ως child του Capsule που αντιστοιχεί στον χαρακτήρα του παίκτη (βλ. Εικόνα 6-41).



Εικόνα 6-41 Η κάμερα ως Child του FPC χαρακτήρα

Μέσω του script `MainCharacterController`, η κάμερα δηλώνεται ως `public field`, επιτρέποντας την αντιστοίχισή της μέσα από το Inspector και παρέχοντας στο script τη δυνατότητα ελέγχου της περιστροφής της (βλ. Εικόνα 6-42).



Εικόνα 6-42 Ενσωμάτωση της κάμερας στο MainCharacterController script μέσω του Cam field

Η διαδικασία αυτή πραγματοποιείται στο `FixedUpdate()` (βλ. Εικόνα 6-43), όπου το script διαβάσει την κίνηση του ποντικιού χρησιμοποιώντας τις ενσωματωμένες μεθόδους της Unity, `Input.GetAxis("Mouse X")` και `Input.GetAxis("Mouse Y")`. Οι μέθοδοι αυτές επιστρέφουν την οριζόντια και κάθετη μετατόπιση του ποντικιού αντίστοιχα, οι οποίες πολλαπλασιάζονται με παραμέτρους ευαισθησίας (`Xsensitivity`, `Ysensitivity`), επιτρέποντας τη ρύθμιση της ταχύτητας περιστροφής. Η διαχείριση της περιστροφής βασίζεται σε δύο ξεχωριστά Quaternions: το ένα ελέγχει την περιστροφή του χαρακτήρα στον κάθετο άξονα και το άλλο την περιστροφή της κάμερας στον οριζόντιο άξονα. Αυτό επιτρέπει στον παίκτη να κινεί την κάμερα ανεξάρτητα από το σώμα, δημιουργώντας ένα φυσικό και ρεαλιστικό αποτέλεσμα.

```

void FixedUpdate()
{
    float yRot = Input.GetAxis("Mouse X") * Ysensitivity;
    float xRot = Input.GetAxis("Mouse Y") * Xsensitivity;

    cameraRot *= Quaternion.Euler(-xRot, 0, 0);
    characterRot *= Quaternion.Euler(0, yRot, 0);

    cameraRot = ClampRotationAroundXAxis(cameraRot);

    this.transform.localRotation = characterRot;
    cam.transform.localRotation = cameraRot;

    x = Input.GetAxis("Horizontal") * speed;
    z = Input.GetAxis("Vertical") * speed;

    transform.position += cam.transform.forward * z + cam.transform.right * x;

    UpdateCursorLock();
}

```

Εικόνα 6-43 FixedUpdate()

Για την αποφυγή ακραίων γωνιών, εφαρμόζεται clamping στον κατακόρυφο άξονα, περιορίζοντας την περιστροφή της κάμερας μεταξύ -90 και 90 μοιρών. Έτσι, ο παίκτης δεν μπορεί να γυρίσει το κεφάλι του προς τα πίσω, διατηρώντας μια φυσική και ρεαλιστική εμπειρία. Η διαδικασία ξεκινά με την κανονικοποίηση του Quaternion, διαιρώντας τις τιμές x, y, z με το w. Το Quaternion μετατρέπεται σε μια μορφή που διευκολύνει τον υπολογισμό της γωνίας. Στη συνέχεια, η γωνία του άξονα X υπολογίζεται μέσω της αντίστροφης εφαπτομένης (`Mathf.Atan`), χρησιμοποιώντας τον τύπο που συνδέει το Quaternion με τις γωνίες Euler (βλ. Εικόνα 6-44).

```

Quaternion ClampRotationAroundXAxis(Quaternion q)
{
    q.x /= q.w;
    q.y /= q.w;
    q.z /= q.w;
    q.w = 1.0f;

    float angleX = 2.0f * Mathf.Rad2Deg * Mathf.Atan(q.x);
    angleX = Mathf.Clamp(angleX, MinimumX, MaximumX);
    q.x = Mathf.Tan(0.5f * Mathf.Deg2Rad * angleX);

    return q;
}

```

Εικόνα 6-44 ClampRotationAroundXAxis()

Για την πλήρη εμπειρία FPS, το script διαχειρίζεται το κλείδωμα του κέρσορα. Κατά το gameplay, ο κέρσορας είναι κλειδωμένος στο κέντρο της οθόνης και αόρατος, επιτρέποντας την ελεύθερη περιστροφή της κάμερας. Η δημόσια μέθοδος SetCursorLock(bool value) καλείται από εξωτερικά scripts (π.χ. Pause Menu) για να ορίσει την κατάσταση κλειδώματος. Όταν η τιμή είναι false, ο κέρσορας γίνεται ορατός και ξεκλειδώνεται. Η UpdateCursorLock() καλείται κάθε frame και, αν το lockCursor είναι true, τρέχει την InternalLockUpdate(), η οποία διαχειρίζεται το πραγματικό κλείδωμα. Το πλήκτρο ESC ξεκλειδώνει τον κέρσορα (cursorIsLocked = false) για αλληλεπίδραση με το UI, ενώ το αριστερό κλικ τον κλειδώνει ξανά (cursorIsLocked = true), επαναφέροντας τον έλεγχο της κάμερας (βλ. Εικόνα 6-45).

```

0 references
public void SetCursorLock(bool value)
{
    lockCursor = value;
    if (!lockCursor)
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}

1 reference
public void UpdateCursorLock()
{
    if (lockCursor)
        InternalLockUpdate();
}

1 reference
public void InternalLockUpdate()
{
    if (Input.GetKeyUp(KeyCode.Escape))
        cursorIsLocked = false;
    else if (Input.GetMouseButtonUp(0))
        cursorIsLocked = true;

    if (cursorIsLocked)
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    else if (!cursorIsLocked)
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}

```

Εικόνα 6-45 Cursor Lock Management - SetCursorLock(), UpdateCursorLock() και InternalLockUpdate() methods

6.6.3 Σύστημα Κίνησης Παίκτη

Το σύστημα κίνησης χρησιμοποιεί δύο βασικά components: ένα Rigidbody component, το οποίο χειρίζεται τη φυσική του σώματος, και ένα CapsuleCollider component, το οποίο ορίζει τα όρια του παίκτη και επιτρέπει τη σωστή ανίχνευση συγκρούσεων με το περιβάλλον. Η κίνηση πραγματοποιείται μέσω των πλήκτρων WASD. Το script χρησιμοποιεί τις μεθόδους `Input.GetAxis("Horizontal")` για τα πλήκτρα A και D, και `Input.GetAxis("Vertical")` για τα πλήκτρα W και S. Οι μέθοδοι αυτές επιστρέφουν τιμές μεταξύ -1 και 1, οι οποίες πολλαπλασιάζονται με την παράμετρο `speed` για να ορίσουν την ταχύτητα κίνησης. Η ενημέρωση γίνεται εντός του `FixedUpdate()`, όπως φαίνεται στην Εικόνα 6-43.

Sprint: Στο `MainCharacterController` script, η λειτουργία του sprint υλοποιείται μέσω της συνάρτησης `Sprint()` (βλ. Εικόνα 6-46). Η συνάρτηση αυτή καλείται σε κάθε frame μέσα στο `Update()` και ελέγχει αν ο παίκτης κρατά πατημένο το πλήκτρο `LeftShift`. Όταν το πλήκτρο είναι πατημένο, η ταχύτητα κίνησης αυξάνεται στην τιμή `runningSpeed`, ενώ όταν αφήνεται, η ταχύτητα επανέρχεται στην κανονική τιμή `walkingSpeed`.

```

Reference
void Sprint()
{
    if (Input.GetKey(KeyCode.LeftShift))
    {
        speed = runningSpeed;
    }
    else
    {
        speed = walkingSpeed;
    }
}

```

Εικόνα 6-46 Sprint()

Jump: Η λειτουργία του άλματος υλοποιείται μέσω της μεθόδου `Jump()` (βλ. Εικόνα 6-47), η οποία καλείται σε κάθε frame μέσα στο `Update()`. Αρχικά, ελέγχεται αν ο παίκτης βρίσκεται σε επαφή με το έδαφος από τη συνάρτηση `IsGrounded()`. Εάν ο παίκτης είναι grounded και πατήσει το πλήκτρο `Space`, τότε εφαρμόζεται μια στιγμιαία κατακόρυφη δύναμη στο Rigidbody μέσω της `rb.AddForce(0, 300, 0)`, η οποία ωθεί τον παίκτη προς τα πάνω.

Παράλληλα, η μέθοδος ελέγχει αν ο παίκτης μόλις προσγειώθηκε. Αυτό γίνεται συγκρίνοντας την τρέχουσα κατάσταση `grounded` με την προηγούμενη (`previouslyGrounded`). Όταν ο παίκτης ήταν προηγουμένως στον αέρα και τώρα βρίσκεται στο έδαφος, η συνθήκη `!previouslyGrounded && grounded` είναι αληθής και το σύστημα καταγράφει την προσγείωση. Στο τέλος, η μεταβλητή `previouslyGrounded` ενημερώνεται με την τρέχουσα κατάσταση, ώστε να είναι διαθέσιμη για τον επόμενο έλεγχο.

```

void Jump()
{
    bool grounded = IsGrounded();
    if (Input.GetKeyDown(KeyCode.Space) && grounded)
    {
        rb.AddForce(0, 300, 0);
        jump.Play();
        if (anim.GetBool("walking"))
        {
            CancelInvoke("PlayFootStepAudio");
            playingWalking = false;
        }
    }
    else if (!previouslyGrounded && grounded)
    {
        land.Play();
    }

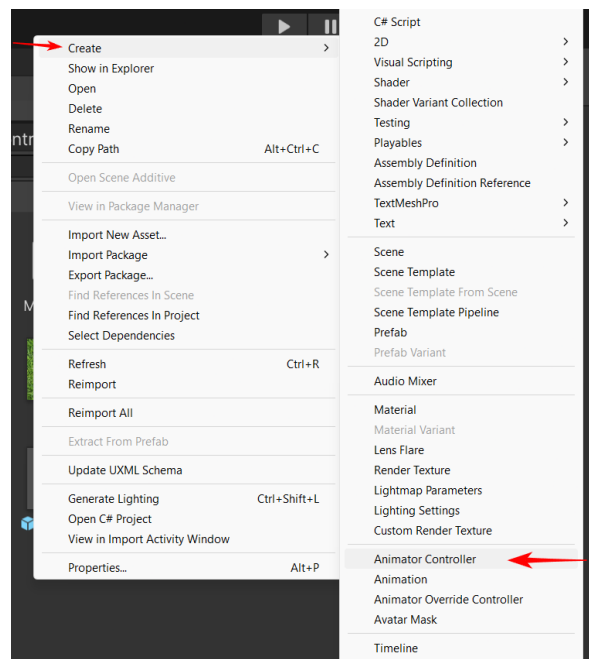
    previouslyGrounded = grounded;
}

```

Εικόνα 6-47 Jump()

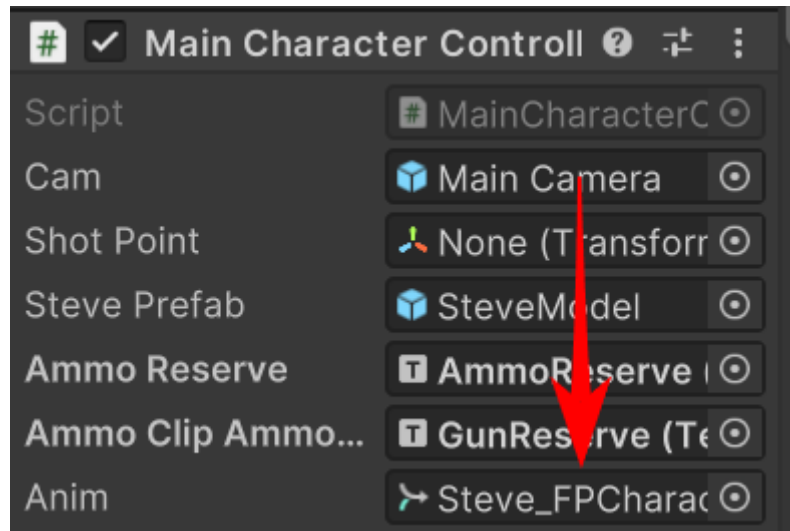
6.6.4 Animation-Animator

Στο σημείο αυτό θα περιγράψουμε τη διαδικασία ενσωμάτωσης κίνησης σε ένα 3D μοντέλο μέσω του Animator Controller της Unity, ακολουθώντας την ίδια λογική με τις Πεπερασμένες Μηχανές Καταστάσεων (FSM) που αναλύθηκαν στο Κεφάλαιο 3. Για τα animations του παίκτη χρησιμοποιήθηκαν 6 animation clips που υπήρχαν ήδη στο μοντέλο του χαρακτήρα Steve: Rifle Idle, Rifle Aiming Idle, Fire, Reloading, Walk With Rifle και Walk And Fire. Η δημιουργία του Animator Controller ξεκίνησε κάνοντας δεξί κλικ στο Project Window και επιλέγοντας Create → Animator Controller (βλ. Εικόνα 6-48).



Εικόνα 6-48 Creating Animator Controller

Αυτή η ενέργεια δημιούργησε ένα νέο Animator Controller asset, το οποίο χρησιμοποιείται για τη διαχείριση όλων των animations του χαρακτήρα. Στη συνέχεια, το Animator Controller συνδέθηκε με το MainCharacterController script μέσω της δήλωσης ενός δημόσιου πεδίου public Animator anim; και κάνοντας drag and drop το Animator Controller στο αντίστοιχο field του script στον Inspector (βλ. Εικόνα 6-49).



Εικόνα 6-49 Σύνδεση του Animator Controller με το MainCharacterController

Για τη δημιουργία του Animator State Machine, κάναμε διπλό κλικ στο Animator Controller που δημιουργήσαμε, ανοίγοντας το Animator Window. Σε αυτό το παράθυρο προσθέσαμε όλα τα 6 animation clips ως states (καταστάσεις). Κάθε state αντιστοιχεί σε ένα διαφορετικό animation clip, το οποίο θα εκτελεστεί όταν ο παίκτης βρίσκεται σε εκείνη την κατάσταση. Για να δημιουργήσουμε σχέσεις (transitions) μεταξύ των καταστάσεων, κάναμε δεξί κλικ πάνω σε ένα state και επιλέξαμε την επιλογή Make Transition. Αυτή η ενέργεια εμφάνισε ένα βέλος, το οποίο συνδέσαμε με το state προορισμού. Το βέλος αυτό αντιπροσωπεύει τη μετάβαση από το ένα animation στο άλλο.

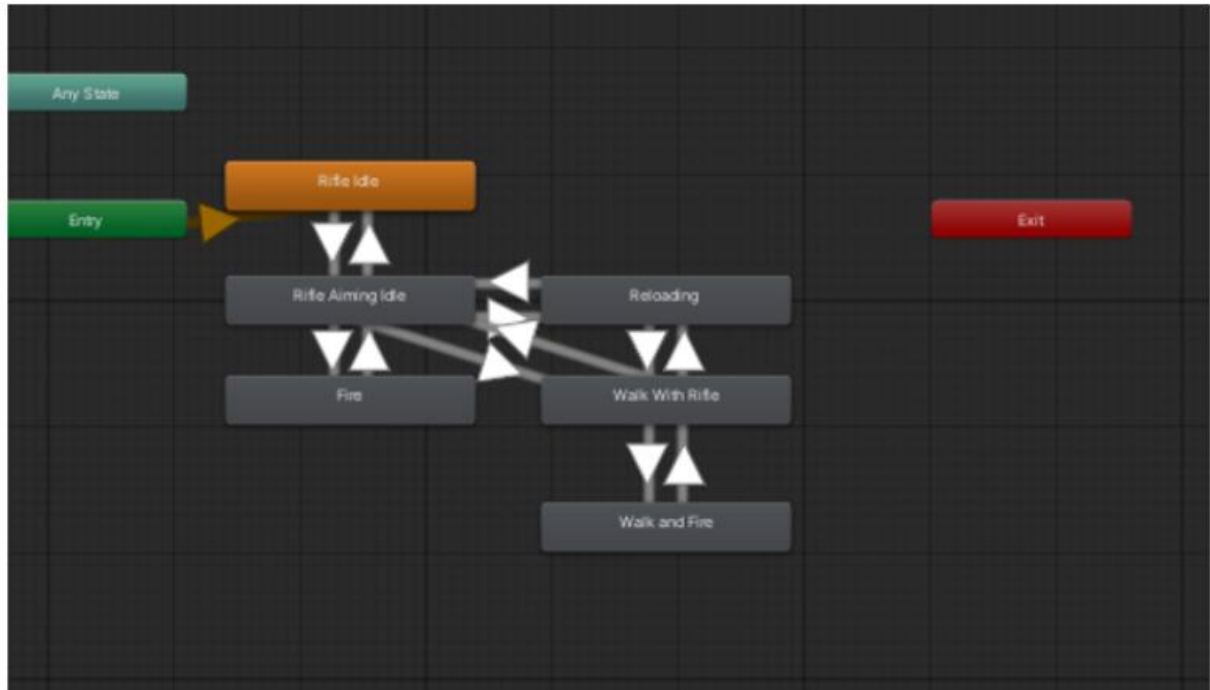
Όλες οι transitions ελέγχονται μέσω conditions (συνθηκών) που βασίζονται σε parameters. Για τον παίκτη δημιουργήσαμε συνολικά 4 parameters: δύο boolean parameters και δύο trigger parameters.

Boolean Parameters: Το πρώτο boolean parameter ονομάζεται arm και καθορίζει αν ο παίκτης έχει το όπλο στα χέρια του. Ο παίκτης μπορεί να το ενεργοποιήσει ή να το απενεργοποιήσει πατώντας το πλήκτρο F. Όταν το arm είναι true, ο παίκτης μπορεί να πυροβολήσει και να κάνει reload. Όταν είναι false, οι ενέργειες αυτές απενεργοποιούνται. Το δεύτερο boolean parameter ονομάζεται walking και καθορίζει αν ο παίκτης κινείται. Ενεργοποιείται όταν ο παίκτης πατάει WASD και απενεργοποιείται όταν σταματά, ελέγχοντας τα walking animations και τους ήχους βημάτων.

Trigger Parameters: Τα trigger parameters είναι παράμετροι που ενεργοποιούνται στιγμιαία και παραμένουν ενεργές μέχρι να καταναλωθούν από μια μετάβαση, επιτρέποντας την εκτέλεση μιας ενέργειας (π.χ. πυροβολισμός) χωρίς να παραμένουν ενεργές συνεχώς. Το πρώτο trigger ονομάζεται fire και ενεργοποιείται όταν ο παίκτης πατάει το αριστερό κουμπί του ποντικιού. Το δεύτερο trigger ονομάζεται reload και ενεργοποιείται όταν ο παίκτης πατάει το πλήκτρο R για να γεμίσει με σφαίρες το όπλο του.

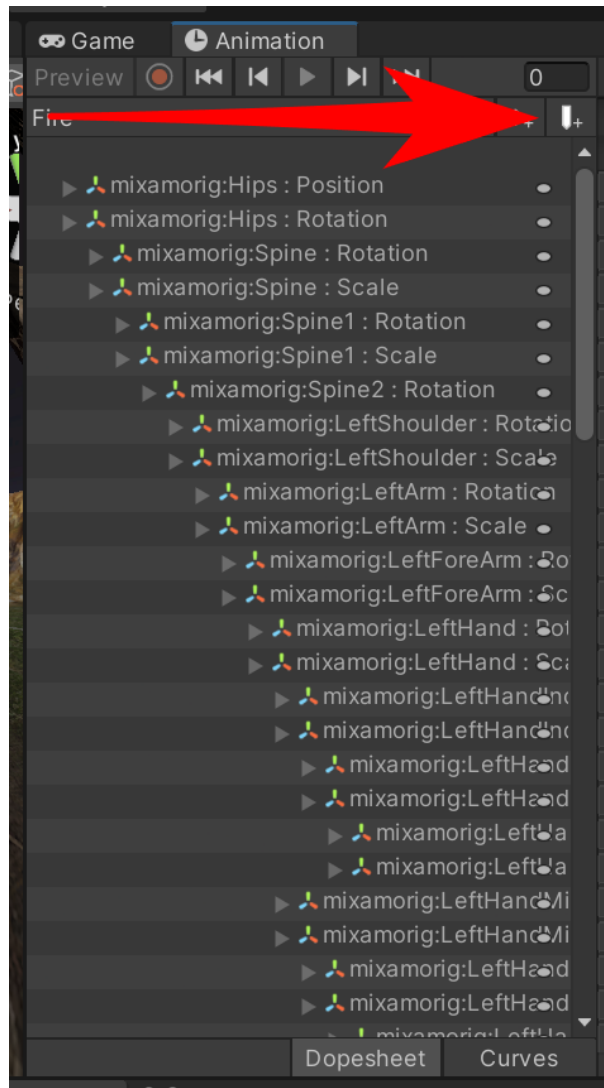
Αναλυτικές Σχέσεις & Transitions (βλ. Εικόνα 6-50):

- **Rifle Idle:** Αυτό είναι το entry state (κατάσταση εισόδου) του Animator. Όταν το `arm == true` και το `walking == false`, ο παίκτης παραμένει στο Rifle Idle, κρατώντας το όπλο ακίνητο.
- **Rifle Aiming Idle:** Υπάρχει μια αμφίδρομη σχέση (bidirectional transition) μεταξύ Rifle Idle και Rifle Aiming Idle. Αυτό σημαίνει ότι η μετάβαση μπορεί να γίνει και προς τις δύο κατευθύνσεις, επιτρέποντας στον παίκτη να εναλλάσσεται μεταξύ κανονικής στάσης και aiming stance χωρίς περιορισμούς, ανάλογα με τις συνθήκες που ορίζονται σε κάθε μετάβαση.
- **Fire State:** Όταν ο παίκτης ενεργοποιήσει το fire trigger (πατώντας το αριστερό κουμπί του ποντικιού), το Rifle Idle ή το Rifle Aiming Idle μεταβαίνουν στο Fire state. Αφού ολοκληρωθεί το shooting animation, ο παίκτης επιστρέφει αυτόματα στο Rifle Idle ή Rifle Aiming Idle, ανάλογα με την προηγούμενη κατάστασή του.
- **Reloading State:** Όταν ο παίκτης ενεργοποιήσει το reload trigger (πατώντας το R), από το Rifle Aiming Idle μεταβαίνει στο Reloading state. Μετά την ολοκλήρωση του reload animation, επιστρέφει στο Rifle Aiming Idle.
- **Walk With Rifle:** Όταν το `walking == true` και το `fire == false`, ο παίκτης μεταβαίνει από το Rifle Idle ή Rifle Aiming Idle στο Walk With Rifle. Αυτό το state αναπαράγει το walking animation με το όπλο προετοιμασμένο.
- **Walk And Fire:** Υπάρχει μια αμφίδρομη σχέση μεταξύ Walk With Rifle και Walk And Fire. Όταν ο παίκτης πυροβολεί ενώ κινείται (fire trigger ενεργό και `walking == true`), μεταβαίνει στο Walk And Fire state, που συνδυάζει τα δύο animations.
- **Επιστροφή σε Idle:** Όταν το `walking == false` (ο παίκτης σταματάει την κίνηση) από τα Walk With Rifle ή Walk And Fire states, ο παίκτης επιστρέφει αυτόματα στο Rifle Idle ή Rifle Aiming Idle, ανάλογα με την κατάσταση του πριν αρχίσει η κίνηση.



Εικόνα 6-50 FSM του MainCharacterController Animator - όλες οι πιθανές μεταβάσεις animations

Animation Events: Ένα ισχυρό χαρακτηριστικό του Animator είναι η δυνατότητα προσθήκης Animation Events. Τα events επιτρέπουν την εκτέλεση κώδικα απευθείας κατά τη διάρκεια της αναπαραγωγής ενός animation clip, προσφέροντας ακριβή συγχρονισμό μεταξύ των οπτικών κινήσεων και της λογικής του παιχνιδιού. Τα Animation Events προστίθενται απευθείας στο Animation Clip μέσω του Animation Window, επιλέγοντας ένα συγκεκριμένο frame και ορίζοντας τη συνάρτηση που θα κληθεί (βλ. Εικόνα 6-51).



Εικόνα 6-51 Επιλογή Add Event

6.6.5 Σύστημα Υγείας & Kits Υγεία

Το σύστημα υγείας αποτελεί σημαντικό καθώς διαχειρίζεται τη ζωή του παίκτη και καθορίζει το τέλος του παιχνιδιού ο παίκτης ξεκινά με 100 πόντους υγείας, οι οποίοι μειώνονται όταν δέχεται ζημιά από εχθρούς.

Σύστημα Υγείας: Η υγεία αποθηκεύεται στη μεταβλητή `public int health = 100`, με μέγιστο όριο `maxHealth = 100`. Ο έλεγχος της ζημιάς πραγματοποιείται μέσω της μεθόδου `TakeHit()` (βλ. Εικόνα 6-52), η οποία καλείται όταν ο παίκτης τραυματίζεται. Η `Mathf.Clamp()` εξασφαλίζει ότι η υγεία παραμένει εντός των ορίων (0–100).

```

public void TakeHit(float amount)
{
    health = (int)Mathf.Clamp(health - amount, 0, maxHealth);
    healthBar.value = health;

    GameObject screenBloodSplatter = Instantiate
    (playerBloodSplatterPrefab);
    screenBloodSplatter.transform.SetParent(canvas.transform);
    screenBloodSplatter.transform.position = new(Random.Range(0,
    canvasWidth), Random.Range(0, canvasHeight), 0);

    Destroy(screenBloodSplatter, 2.2f);
    if (health <= 0)
    {
        Vector3 spawnPos = new Vector3(
            this.transform.position.x,
            this.transform.position.y - 0.5f,
            this.transform.position.z);
        GameObject steve = Instantiate(stevePrefab, spawnPos, this.
        transform.rotation);
        steve.GetComponent<Animator>().SetTrigger("Death");
        GameStats.gameOver = true;
        Destroy(this.gameObject);

        GameObject gameOver = Instantiate(gameOverScreen);
        gameOver.transform.SetParent(canvas.transform);
        gameOver.transform.position = new Vector3(canvasWidth / 2,
        canvasHeight / 2, 0);

        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}

```

Εικόνα 6-52 TakeHit()

Κάθε φορά που καλείται η TakeHit(), η υγεία ενημερώνεται και ο παίκτης λαμβάνει άμεση οπτική ανατροφοδότηση. Ένα UI Slider (healthBar) ενημερώνεται αυτόματα για να απεικονίσει την τρέχουσα κατάσταση της υγείας, ενώ παράλληλα δημιουργείται ένα αιμάτινο overlay στην οθόνη μέσω του playerBloodSplatterPrefab.

Για τη δημιουργία του, προστέθηκε στο Canvas ένα νέο RawImage, στο πεδίο Texture του οποίου ορίστηκε η επιθυμητή εικόνα αίματος. Στη συνέχεια, δημιουργήθηκε ένα νέο Animation Clip μέσω του Animation Window, με το όνομα playerBloodSplatter (βλ. Εικόνα 6-53), και προστέθηκε η ιδιότητα Color του RawImage ως property. Η διάρκεια του animation αυξήθηκε στα 2.2 δευτερόλεπτα και στο τέλος του ορίστηκαν οι τιμές του χρώματος στο 0, επιτυγχάνοντας έτσι ένα εφέ fade out.



Εικόνα 6-53 Το playerBloodSplatter animation όπως φαίνεται εντός παιχνιδιού

Κατά την εκτέλεση της `TakeHit()`, το prefab αυτό δημιουργείται δυναμικά μέσω της `Instantiate(playerBloodSplatterPrefab)`, τοποθετείται ως παιδί του Canvas (`screenBloodSplatter.transform.SetParent(canvas.transform)`) και η θέση του ορίζεται τυχαία εντός των ορίων της οθόνης, χρησιμοποιώντας `Random.Range(0, canvasWidth)` και `Random.Range(0, canvasHeight)`. Το αντικείμενο παραμένει ορατό για 2.2 δευτερόλεπτα και στη συνέχεια καταστρέφεται με `Destroy(screenBloodSplatter, 2.2f)`, ενισχύοντας την ένταση του gameplay.



Εικόνα 6-54 StevePrefab

Σύστημα Θανάτου: Όταν η υγεία φτάσει στο μηδέν, ενεργοποιείται ο μηχανισμός θανάτου. Το script δημιουργεί ένα ragdoll GameObject από το `stevePrefab` (βλ. Εικόνα 6-54) στη θέση του παίκτη. Το συγκεκριμένο prefab διαθέτει δικό του Animator, ο οποίος ενεργοποιεί το death animation μέσω του `anim.SetTrigger("Death")`, καθώς και ενσωματωμένη κάμερα. Αυτό είναι απαραίτητο, γιατί το αρχικό αντικείμενο του παίκτη (FPC) καταστρέφεται με `Destroy(this.gameObject)`. Το ragdoll αναλαμβάνει πλέον το animation του θανάτου, ενώ η ενσωματωμένη κάμερα επιτρέπει τη θέαση της σκηνής από τη

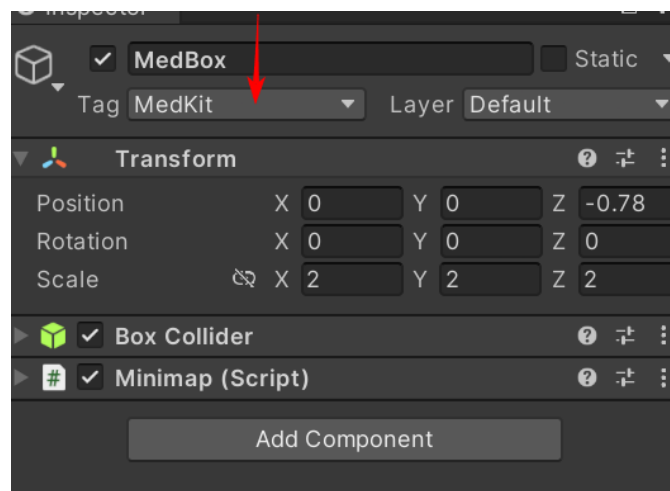
σκοπία του νεκρού χαρακτήρα. Ταυτόχρονα, παίζεται ο ήχος θανάτου (deathSound) και το αρχικό αντικείμενο του παίκτη καταστρέφεται.

Στη συνέχεια, εμφανίζεται η οθόνη Game Over (βλ. Εικόνα 6-55) στο κέντρο της οθόνης, ενώ η μεταβλητή GameStats.gameOver ορίζεται σε true, σταματώντας τη δημιουργία νέων εχθρών. Ο κέρσορας ξεκλειδώνεται (Cursor.lockState = CursorLockMode.None) και γίνεται ορατός (Cursor.visible = true), επιτρέποντας την αλληλεπίδραση με το μενού.



Εικόνα 6-55 Game Over screen

Σύστημα Pickup Υγείας: Η συλλογή αντικειμένων υγείας υλοποιείται μέσω του OnCollisionEnter() στο Unity. Τα Tags είναι μια λειτουργία που επιτρέπει την ταξινόμηση των GameObjects με μια λέξη-κλειδί, διευκολύνοντας την αναγνώρισή τους μέσα από τον κώδικα (βλ. Εικόνα 6-56).



Εικόνα 6-56 "MedKit" Tag

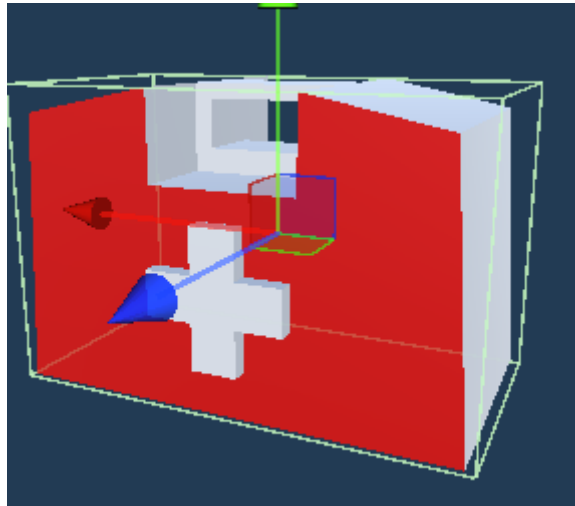
Όταν ο παίκτης συγκρούεται με ένα αντικείμενο που φέρει την ετικέτα "MedKit" και η υγεία του είναι μικρότερη του μέγιστου, η υγεία αυξάνεται κατά 25 πόντους με χρήση Mathf.Clamp για αποφυγή υπέρβασης του ορίου. Το UI slider ενημερώνεται, παίζεται ο αντίστοιχος ήχος (healthPickup.Play()) και το αντικείμενο καταστρέφεται (βλ. Εικόνα 6-57 και Εικόνα 6-58).

```

else if (col.gameObject.tag == "MedKit" && health < maxHealth)
{
    health = Mathf.Clamp(health + 25, 0, maxHealth);
    healthBar.value = health;
    Debug.Log("MedKit: " + health);
    Destroy(col.gameObject);
    healthPickup.Play();
}

```

Εικόνα 6-57 MedKit ()



Εικόνα 6-58 Med kit model

6.7 Σύστημα Πυροβολισμού & Πυρομαχικών (Shooting & Ammo System)

6.7.1 Μηχανισμός Shooting:

Η λειτουργία του πυροβολισμού υλοποιείται μέσω της μεθόδου Shooting() (βλ. Εικόνα 6-59), η οποία καλείται μέσα στην Update() όταν ο παίκτης πατάει το αριστερό κουμπί του ποντικιού (Input.GetMouseButtonDown(0)). Το script ελέγχει τρεις συνθήκες: αν ο παίκτης είναι οπλισμένος (anim.GetBool("arm")), αν δεν βρίσκεται ήδη σε κατάσταση πυροβολισμού (!anim.GetBool("fire")) και αν η βολή επιτρέπεται (GameStats.canShoot). Εάν όλες οι συνθήκες ικανοποιούνται, ενεργοποιείται το fire trigger του Animator, το οποίο αναπαράγει το animation Fire.

```

1 reference
void Shooting()
{
    if (Input.GetMouseButtonDown(0) && !anim.GetBool("fire") && anim.
        GetBool("arm") && GameStats.canShoot)
    {
        if (ammoClip > 0)
        {
            anim.SetTrigger("fire");
            ProcessZombieHit();

            ammoClip--;
            ammoClipAmount.text = ammoClip.ToString();

            GameStats.canShoot = false;
        }
        else
            triggerSound.Play();
    }
}

```

Εικόνα 6-59 Shooting()

Η ανίχνευση του στόχου πραγματοποιείται μέσω της μεθόδου `ProcessZombieHit()` (βλ. Εικόνα 6-60), η οποία εκτελεί ένα `Physics.Raycast` από τη θέση της κάμερας προς τα εμπρός, με μέγιστη εμβέλεια 300 μονάδων. Αν το raycast χτυπήσει αντικείμενο με ετικέτα "Zombie", δημιουργείται ένα εφέ αίματος (blood splatter) — το οποίο αναλύεται στην ενότητα 6.7.2 — στο σημείο της πρόσκρουσης, παρέχοντας άμεση οπτική ανατροφοδότηση.

```

1 reference
async void ProcessZombieHit()
{
    RaycastHit hitInfo;
    Vector3 rayOrigin = cam.transform.position + (cam.transform.forward *
        0.5f);
    if (Physics.Raycast(rayOrigin, cam.transform.forward, out hitInfo,
        300))
    {
        GameObject hitZombie = hitInfo.collider.gameObject;

        if (hitZombie.gameObject.tag == "Zombie")
        {
            await Task.Delay(200);
            hitZombie.GetComponent<zombieController>().DeadZombie();
            zombieBloodSplatterPrefab.transform.LookAt(this.transform.
                position);
            GameObject bloodSplatter = Instantiate
                (zombieBloodSplatterPrefab, hitInfo.point, Quaternion.
                    LookRotation(hitInfo.normal));
            Destroy(bloodSplatter, 0.5f);
        }
    }
}

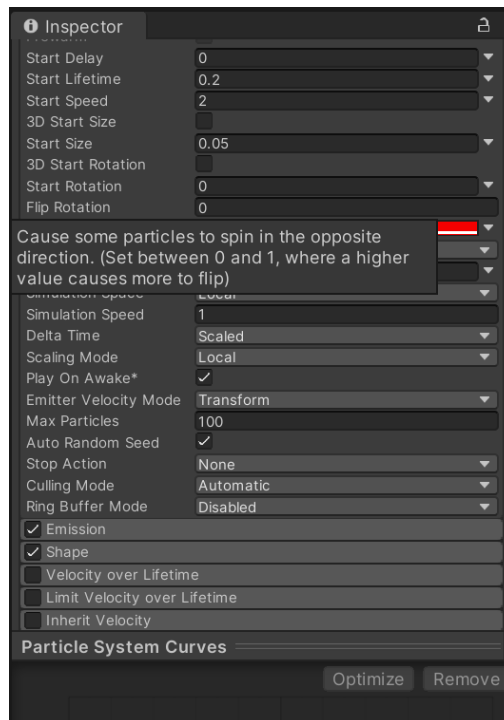
```

Εικόνα 6-60 Η μέθοδος `ProcessZombieHit()` και η υλοποίηση του Raycast.

6.7.2 Εφέ Αίματος (Particle System)

Για τη δημιουργία του εφέ αίματος αξιοποιήθηκε το Particle System της Unity. Το Particle System είναι ένα ισχυρό εργαλείο που επιτρέπει τη δημιουργία δυναμικών οπτικών εφέ, όπως πιτσιλιές αίματος, μέσω της εκπομπής και διαχείρισης πολλών μικρών σωματιδίων.

Κάνοντας δεξί κλικ στο Hierarchy, επιλέγοντας Effects → Particle System και στη συνέχεια μέσα από το Inspector, έγιναν οι εξής ρυθμίσεις: η Duration ορίστηκε σε 0.10 για στιγμιαία εκπομπή, απενεργοποιήθηκε το Looping, επιλέχθηκε κόκκινο χρώμα για τα σωματίδια και το Shape ορίστηκε σε Cone, ώστε να προσομοιώνει ρεαλιστικά την εκτόξευση σταγόνων αίματος (βλ. Εικόνα 6.61).



Εικόνα 6-61 Οι ρυθμίσεις του ZombieBloodSplatter Particle System.

Το σύστημα αποθηκεύτηκε ως Prefab και χρησιμοποιείται δυναμικά μέσω της Instantiate() κατά το χτύπημα ενός ζόμπι.

6.7.3 Συγχρονισμός με το Animation – Animation Events

Ένα κρίσιμο στοιχείο του συστήματος είναι η ασύγχρονη καθυστέρηση (async delay). Αντί να εφαρμόζεται η ζημιά αμέσως, ορίσαμε μια μικρή καθυστέρηση Task.Delay(200) πριν εφαρμοστεί η ζημιά. Αυτό επιτυγχάνει συγχρονισμό μεταξύ του Fire animation και του πραγματικού χτυπήματος. Η καθυστέρηση αυτή δεν μπλοκάρει το κύριο thread, επιτρέποντας στο παιχνίδι να συνεχίσει να τρέχει ομαλά. Για τον ακριβή συγχρονισμό του πυροβολισμού με τα οπτικά εφέ και τον ήχο, αξιοποιήθηκαν τα Animation Events της Unity. Μέσω αυτών δημιουργήθηκαν δύο συναρτήσεις: η Fire() και η canShoot() (βλ. Εικόνα 6.62).

```
0 references
public void Fire()
{
    shot.Play();
}

0 references
public void CanShoot()
{
    GameStats.canShoot = true;
}
```

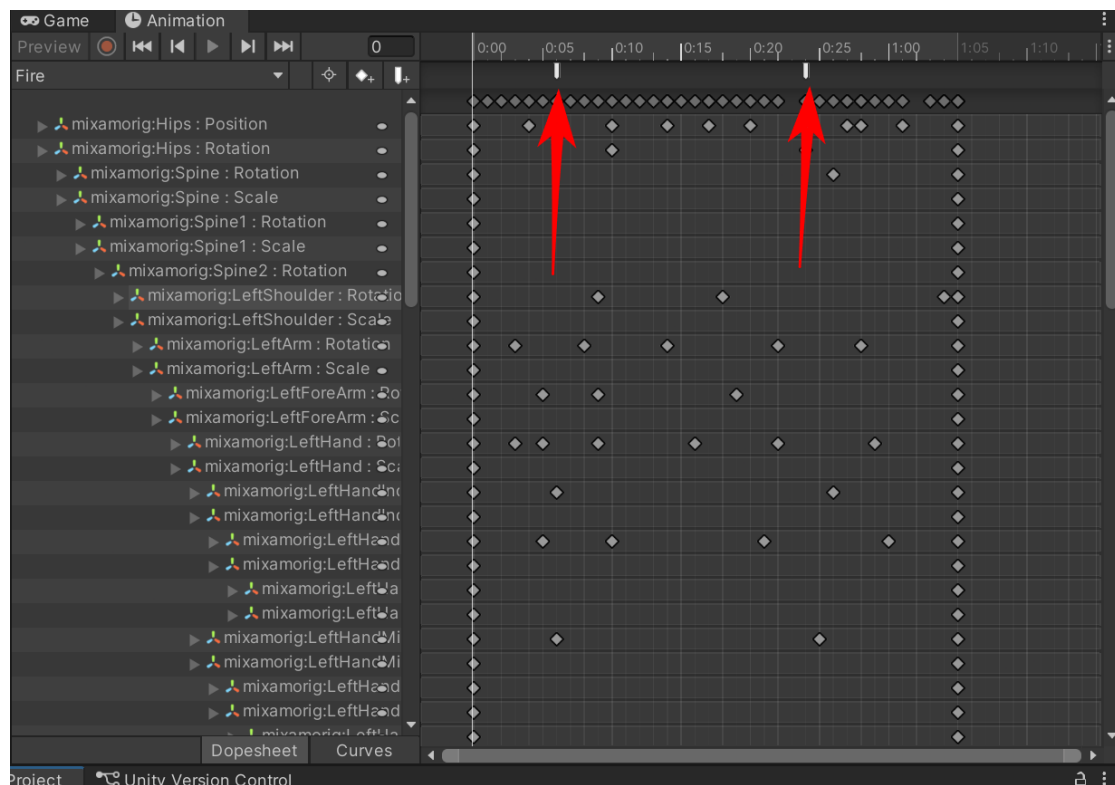
Εικόνα 6-62 Οι συναρτήσεις Fire() και canShoot() στο script.

Η ανάγκη για αυτές προέκυψε από ένα βασικό πρόβλημα: ο παίκτης μπορούσε να πατάει το κουμπί πυροβολισμού συνεχώς, ξοδεύοντας σφαίρες ακόμα και πριν ολοκληρωθεί το animation του προηγούμενου πυροβολισμού.

Η **Fire()** τοποθετήθηκε στην αρχή του animation, όπου ο χαρακτήρας «πατάει τη σκανδάλη». Όταν το animation φτάσει σε αυτό το frame, το event καλεί τη συνάρτηση, η οποία παίζει τον ήχο του πυροβολισμού.

Η **canShoot()** τοποθετήθηκε στο τέλος του Fire animation clip. Όταν το animation ολοκληρώνεται, το event καλεί τη συνάρτηση και θέτει το `GameStats.canShoot = true`, επιτρέποντας στον παίκτη να πυροβολήσει ξανά.

Στην Εικόνα 6.63 φαίνονται τα σημεία όπου τοποθετήθηκαν τα Animation Events μέσα στο Fire animation clip, με την **Fire()** να βρίσκεται στο frame της σκανδάλης και την **canShoot()** στο τέλος του animation.



Εικόνα 6-63 Τοποθέτηση Animation Events στο Fire animation clip για τον συγχρονισμό του ήχου και του ρυθμού πυροβολισμού.

6.7.4 Σύστημα Πυρομαχικών και Reload

Μηχανισμός Reload: Όταν ο παίκτης πατάει το πλήκτρο R και είναι οπλισμένος (`Input.GetKeyDown(KeyCode.R) && anim.GetBool("arm")`), ενεργοποιείται το animation του reload και παίζει ο αντίστοιχος ήχος. Στη συνέχεια, υπολογίζει πόσες σφαίρες χρειάζονται για να γεμίσει ο γεμιστήρας (`amountNeeded = ammoClipMax - ammoClip`) και πόσες είναι διαθέσιμες στην εφεδρεία (`ammoAvailable`). Οι σφαίρες μεταφέρονται από την εφεδρεία στον γεμιστήρα και τα UI texts ενημερώνονται ανάλογα (βλ. Εικόνα 6-64).

```

void Reload()
{
    if (Input.GetKeyDown(KeyCode.R) && anim.GetBool("arm"))
    {
        anim.SetTrigger("reload");
        reloadSound.Play();
        int amountNeeded = ammoClipMax - ammoClip;
        int ammoAvailable = amountNeeded < ammo ? amountNeeded : ammo;
        ammo -= ammoAvailable;
        ammoClip += ammoAvailable;
        ammoReserve.text = ammo.ToString();
        ammoClipAmount.text = ammoClip.ToString();
    }
}

```

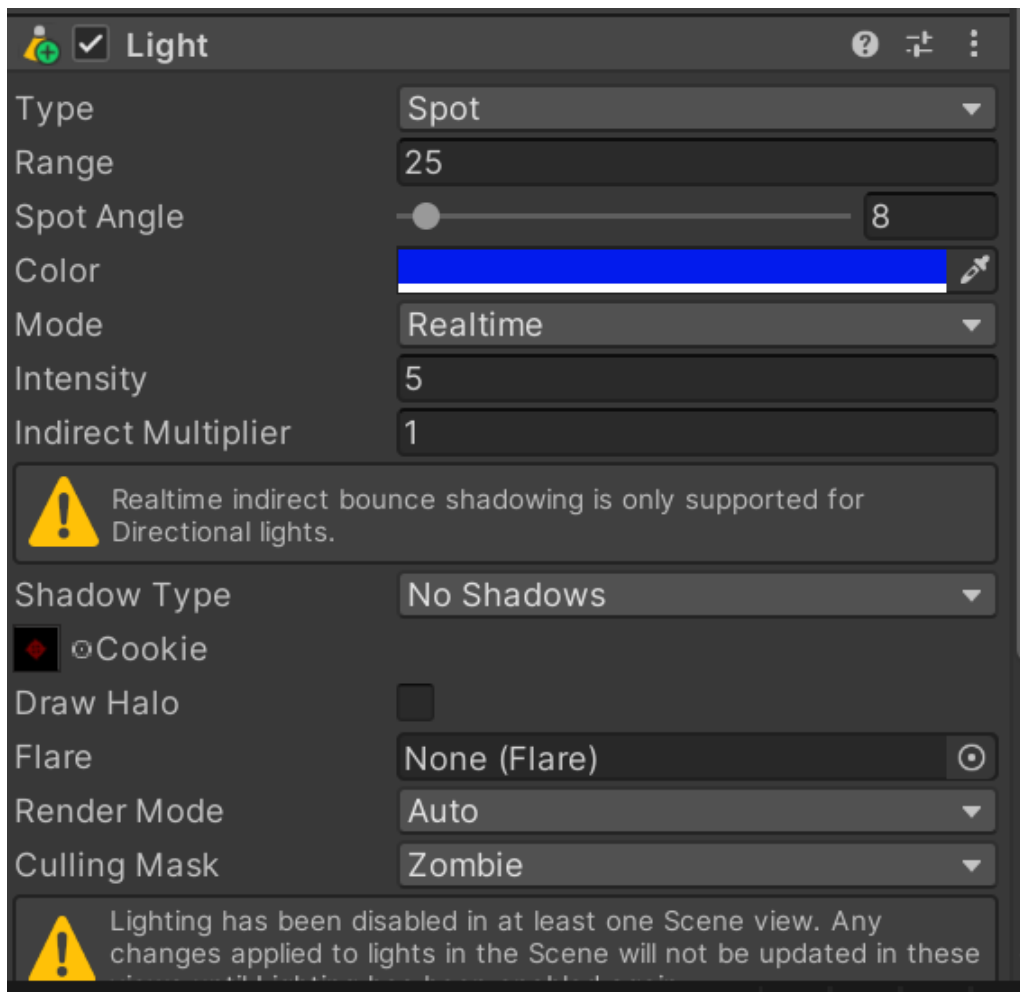
Εικόνα 6-64 Reload()

6.7.5 Raycast & Crosshair

Για τον μηχανισμό σκοποβολής χρησιμοποιήθηκε το Raycast της Unity. Το Raycast είναι μια μέθοδος που εκτοξεύει μια νοητή ακτίνα από ένα σημείο προς μια κατεύθυνση και ελέγχει αν αυτή τέμνει κάποιο αντικείμενο. Στο παιχνίδι, η ακτίνα εκτοξεύεται από την άκρη της κάνης του όπλου προς την κατεύθυνση που κοιτάζει ο παίκτης.

Για την ακριβή τοποθέτηση του σημείου εκτόξευσης, δημιουργήθηκε ένα Cube GameObject ως child του όπλου. Το Cube τοποθετήθηκε με ακρίβεια στην άκρη της κάνης και απενεργοποιήθηκε το Mesh Renderer του, ώστε να παραμείνει αόρατο. Στο MainCharacterController script, δημιουργήθηκε μια δημόσια μεταβλητή public Transform shotPoint; η οποία συνδέθηκε μέσω του Inspector με το Cube.

Για την παροχή οπτικής ανατροφοδότησης κατά τη σκόπευση, δημιουργήθηκε ένα Crosshair στο κέντρο της οθόνης. Παράλληλα, στο Cube που χρησιμοποιήθηκε ως σημείο εκτόξευσης, προστέθηκε ένα Light component. Στο Inspector του Light, πραγματοποιήθηκαν οι εξής ρυθμίσεις: Type: Spot, Range: 25, Spot Angle: 8, Intensity: 5, Shadow Type: No Shadows, Culling Mask: Zombie (βλ. Εικόνα 6-65).



Εικόνα 6-65 Light Settings

Η ρύθμιση Culling Mask καθορίζει ποια αντικείμενα θα επηρεάζονται από το φως — στην προκειμένη περίπτωση, ορίστηκε στο layer "Zombie", ώστε το φως να εμφανίζεται αποκλειστικά πάνω στους εχθρούς. Το Light αυτό λειτουργεί ως οπτικό feedback, επιβεβαιώνοντας ότι ο παίκτης «βλέπει» τους εχθρούς μέσα από το σκόπευτρο (βλ. Εικόνα 6-66).



Εικόνα 6-66 Crosshair

6.7.6 Εμφάνιση και Απόκρυψη Όπλου

Για την εμφάνιση και απόκρυψη του όπλου χρησιμοποιήθηκε το boolean parameter arm. Ο παίκτης μπορεί να πατήσει το πλήκτρο F, το οποίο εναλλάσσει την κατάσταση του arm μέσω της εντολής `anim.SetBool("arm", !anim.GetBool("arm"))`. Όταν το arm είναι true, το όπλο είναι ορατό, ενώ όταν είναι false, το όπλο κρύβεται (βλ. Εικόνα 6-67).

```
1 reference
void HideShowArm()
{
    if (Input.GetKeyDown(KeyCode.F))
    {
        anim.SetBool("arm", !anim.GetBool("arm"));
    }
}
```

Εικόνα 6-67 HideShowArm()

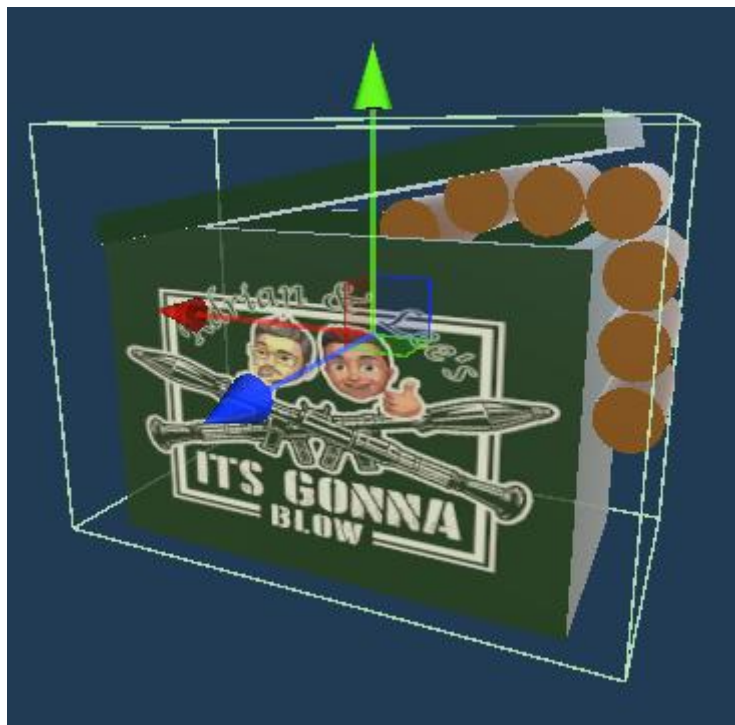
6.7.7 Σύστημα Ammo Kit

Η συλλογή πυρομαχικών υλοποιείται μέσω της μεθόδου `OnCollisionEnter()` στο `MainCharacterController` script. Όταν ο παίκτης συγκρούεται με ένα αντικείμενο που φέρει την ετικέτα "Ammo", το σύστημα αυξάνει την εφεδρεία πυρομαχικών (ammo) κατά 20 σφαίρες, με χρήση `Mathf.Clamp`. Το UI text της εφεδρείας ενημερώνεται, παίζεται ο αντίστοιχος ήχος (`ammoPickup.Play()`) και το αντικείμενο καταστρέφεται με `Destroy(col.gameObject)` (βλ. Εικόνα 6-68 και Εικόνα 6-69).

```
if (col.gameObject.tag == "Ammo" && ammo < maxAmmo)
{
    ammo = Mathf.Clamp(ammo + 20, 0, maxAmmo);
    ammoReserve.text = ammo.ToString();
    Debug.Log("Ammo: " + ammo);
    Destroy(col.gameObject);
    ammoPickup.Play();
}
```

Εικόνα 6-68 AmmoBox()

Η λειτουργία αυτή είναι πανομοιότυπη με αυτή του Health Kit, με τη διαφορά ότι αντί για υγεία αναπληρώνονται πυρομαχικά (βλ. Εικόνα 6-69).



Εικόνα 6-69 Ammo Box

6.7.8 Σύστημα Ήχου (Audio System)

Το MainCharacterController script διαχειρίζεται πολλαπλά AudioSource fields, τα οποία αντιστοιχούν σε διαφορετικές ενέργειες και events. Κάθε AudioSource αναφέρεται σε ένα συγκεκριμένο ηχητικό clip, το οποίο προέρχεται από εξωτερικά ηχητικά αρχεία (WAV, MP3 κ.λπ.).

- **Ήχοι Βημάτων (Footsteps):** Το script διαχειρίζεται ένα array από AudioSources (public AudioSource[] footsteps) με πολλαπλά διαφορετικά footstep sounds, για ρεαλιστική εμπειρία. Όταν ο παίκτης κινείται, ενεργοποιείται η InvokeRepeating("PlayFootStepAudio", 0, 0.4f), η οποία καλεί τη μέθοδο PlayFootStepAudio() κάθε 0.4 δευτερόλεπτα. Η μέθοδος επιλέγει τυχαία έναν ήχο και τον αναπαράγει, ενώ ανταλλάσσει τις θέσεις των audio sources στο array για ποικιλία (βλ. Εικόνα 6-70).

```

if (Mathf.Abs(x) > 0 || Mathf.Abs(z) > 0)
{
    if (!anim.GetBool("walking"))
    {
        anim.SetBool("walking", true);
        InvokeRepeating("PlayFootStepAudio", 0, 0.4f);
    }
}
else if (anim.GetBool("walking"))
{
    anim.SetBool("walking", false);
    CancelInvoke("PlayFootStepAudio");
    playingWalking = false;
}

```

Εικόνα 6-70 FootSteps()

- **Άλμα & Προσγείωση:** Το jump.Play() παίζεται όταν ο παίκτης πατάει Space, ενώ το land.Play() όταν προσγειώνεται, προσφέροντας άμεση ηχητική ανατροφοδότηση.
- **Πυροβολισμός & Reload:** Ο ήχος πυροβολισμού παίζεται μέσω της Fire() που καλείται από Animation Event, εξασφαλίζοντας απόλυτο συγχρονισμό με το animation. Το reloadSound.Play() παίζεται όταν ο παίκτης πατάει R για επαναφόρτωση.
- **Dry-Fire:** Όταν ο παίκτης προσπαθεί να πυροβολήσει χωρίς πυρομαχικά (ammoClip <= 0), παίζεται triggerSound () που υποδηλώνει ότι το όπλο είναι άδειο.
- **Pickup:** Κατά τη συλλογή αντικειμένων παίζεται ammoPickup.Play() για πυρομαχικά και healthPickup.Play() για κιτ υγείας, παρέχοντας θετική ανατροφοδότηση.
- **Θανάτου:** Όταν η υγεία του παίκτη μηδενιστεί, παίζεται deathSound.Play() δημιουργώντας ένα δραματικό ηχητικό εφέ.

6.8 Σύστημα Εχθρών – Zombies (Enemy System)

6.8.1 Zombie FSM – Finite State Machine

Το σύστημα AI των ζόμπι υλοποιείται μέσω του script zombieControler.cs. Το script διαχειρίζεται 5 κύριες καταστάσεις, οι οποίες ορίζονται σε μια enum (βλ. Εικόνα 6-71):

```
enum STATE { IDLE, WANDER, CHASE, ATTACK, DEAD }
```

Εικόνα 6-71 FSM States

IDLE: Αυτή είναι η αρχική κατάσταση, όπου το ζόμπι παραμένει ακίνητο. Σε αυτή την κατάσταση, το ζόμπι δεν κάνει καμία κίνηση και δεν αναπαράγει κανένα animation. Παραμένει σε αυτή την κατάσταση μέχρι κάποιο event να το ενεργοποιήσει, όπως η ανίχνευση του παίκτη. Από το IDLE state, το ζόμπι

μπορεί να μεταβεί σε WANDER, αν ο παίκτης δεν είναι κοντά, ή απευθείας σε CHASE, αν ο παίκτης μπει στο detection range (βλ. Εικόνα 6-72).

```
case STATE.IDLE:
    if (IsTargetInChaseRange())
    {
        currentState = STATE.CHASE;
    }
    else if (Random.Range(0, 5000) < 5)
    {
        currentState = STATE.WANDER;
    }
    break;
```

Εικόνα 6-72 IDLE STATE

WANDER: Σε αυτή την κατάσταση, το ζόμπι περιπλανιέται τυχαία στο Terrain. Κάθε φορά που δεν έχει path, το script υπολογίζει μια τυχαία θέση προορισμού χρησιμοποιώντας Random.Range(-5, 5) για τους άξονες X και Z. Για να εξασφαλίσει ότι η θέση είναι πάνω στο Terrain, χρησιμοποιεί Terrain.activeTerrain.SampleHeight() για να λάβει το σωστό Y coordinate. Το NavMeshAgent ορίζει αυτή τη θέση ως στόχο και το ζόμπι κινείται προς αυτή με walk speed. Το animation που αναπαράγεται σε αυτή τη φάση είναι το walking animation (isZombieWalking = true), το οποίο δίνει την αίσθηση ότι το ζόμπι περιφέρεται άσκοπα. Όταν φτάσει στον προορισμό, επιλέγει μια νέα τυχαία θέση. Αν ο παίκτης μπει στο detection range, το ζόμπι μεταβαίνει σε CHASE (βλ. Εικόνα 6-73).

```
case STATE.WANDER:
    if (!agent.hasPath)
    {
        float nexW = this.transform.position.x + Random.Range(-5, 5);
        float nextZ = this.transform.position.z + Random.Range(-5, 5);
        float nextY = Terrain.activeTerrain.SampleHeight(new Vector3(nexW, 0, nextZ));
        Vector3 nextDestination = new Vector3(nexW, nextY, nextZ);
        agent.SetDestination(nextDestination);
        agent.stoppingDistance = 0;
        TurnOffTriggers();
        agent.speed = walkSpeed;
        anim.SetBool("isZombieWalking", true);
    }

    if (IsTargetInChaseRange())
    {
        currentState = STATE.CHASE;
    }
    else if (Random.Range(0, 5000) < 5)
    {
        TurnOffTriggers();
        currentState = STATE.IDLE;
        agent.ResetPath();
    }
    break;
```

Εικόνα 6-73 WANDER STATE

CHASE: Όταν ο παίκτης μπει σε απόσταση μικρότερη των 60 μονάδων, το ζόμπι μεταβαίνει σε κατάσταση καταδίωξης. Το σύστημα NavMesh, αξιοποιώντας τον αλγόριθμο A* (A-star), αναλαμβάνει την εύρεση της βέλτιστης διαδρομής προς τη θέση του παίκτη, ενώ η ταχύτητα του ζόμπι αυξάνεται (run speed) για μεγαλύτερη ένταση. Ο αλγόριθμος A* συνδυάζει το πραγματικό κόστος μετακίνησης με μια εκτίμηση της υπολειπόμενης απόστασης, επιτρέποντας την ταχύτερη εύρεση της συντομότερης

διαδρομής. Το ζόμπι σταματά σε απόσταση 3 μονάδων από τον παίκτη, ώστε να μην περπατάει ακριβώς πάνω του. Αν ο παίκτης απομακρυνθεί πέρα από τις 100 μονάδες, το ζόμπι τον «ξεχνά» και επιστρέφει σε WANDER (βλ. Εικόνα 6-74).

```
case STATE.CHASE:
    if (GameStats.gameOver)
    {
        TurnOffTriggers();
        currentState = STATE.WANDER;
        return;
    }
    agent.SetDestination(target.transform.position);
    agent.stoppingDistance = 3;
    TurnOffTriggers();
    agent.speed = runSpeed;
    anim.SetBool("isZombieRunning", true);

    if (agent.remainingDistance <= agent.stoppingDistance && !agent.pathPending)
    {
        currentState = STATE.ATTACK;
    }

    if (ForgetPlayer())
    {
        currentState = STATE.WANDER;
        agent.ResetPath();
    }
    break;
```

Εικόνα 6-74 CHASE STATE

ATTACK: Όταν το ζόμπι φτάσει σε απόσταση μικρότερη των 3 μονάδων από τον παίκτη, μεταβαίνει σε κατάσταση επίθεσης. Σε αυτή τη φάση, το ζόμπι σταματά να κινείται, στρέφεται προς τον παίκτη και εκτελεί το animation επίθεσης. Στο κατάλληλο σημείο του animation, ενεργοποιείται ένα Animation Event που καλεί τη μέθοδο που προκαλεί ζημιά στον παίκτη. Παράλληλα, παίζεται ένας ήχος χτυπήματος για audio feedback (βλ. Εικόνα 6-75).

```
case STATE.ATTACK:
    if (GameStats.gameOver)
    {
        TurnOffTriggers();
        currentState = STATE.WANDER;
        break;
    }
    TurnOffTriggers();
    anim.SetBool("isZombieAttacking", true);
    this.transform.LookAt(target.transform.position);
    if (DistanceFromPlayer() > agent.stoppingDistance + 2)
    {
        currentState = STATE.CHASE;
    }
    break;
```

Εικόνα 6-75 ATTACK STATE

DEAD: Όταν το ζόμπι σκοτώνεται μέσω του raycast shooting, μεταβαίνει σε κατάσταση θανάτου. Το σύστημα πλοήγησης απενεργοποιείται, το ζόμπι σταματά κάθε κίνηση και αναπαράγει το animation θανάτου. Παράλληλα, ενεργοποιείται ένα ξεχωριστό script που βυθίζει σταδιακά το ζόμπι στο έδαφος, μέχρι να καταστραφεί οριστικά (βλ. Εικόνα 6-76).

```
case STATE.DEAD:
    Destroy(agent);
    this.GetComponent<Sink>().StartSink();
    break;
```

Εικόνα 6-76 DEAD STATE

6.8.2 Σύστημα Δημιουργίας Zombie (Zombie Spawning System)

Το σύστημα δημιουργίας των ζόμπι είναι υπεύθυνο για τη δυναμική εμφάνιση εχθρών κατά τη διάρκεια του gameplay. Αντί να τοποθετούμε τα ζόμπι χειροκίνητα στη σκηνή, χρησιμοποιούμε ένα Coroutine-based spawner που δημιουργεί ζόμπι περιοδικά σε τυχαίες θέσεις πάνω στο Terrain. Το script ZombieSpawner.cs ξεκινά ένα Coroutine (SpawnLoop) (βλ. Εικόνα 6-80) κατά την εκκίνηση του παιχνιδιού.

Κατά τη λειτουργία του, το σύστημα ελέγχει τρεις βασικές συνθήκες πριν δημιουργήσει ένα νέο ζόμπι:

- Έλεγχος κατάστασης παιχνιδιού: Αν το παιχνίδι έχει τελειώσει (GameStats.gameOver), σταματά η δημιουργία νέων ζόμπι.
- Όριο ζωντανών ζόμπι: Δημιουργούνται νέα ζόμπι μόνο αν ο αριθμός των ζωντανών είναι μικρότερος από το μέγιστο όριο (συνήθως 30).
- Χρονικό διάστημα αναμονής: Το σύστημα περιμένει ένα καθορισμένο χρονικό διάστημα (π.χ. 5 δευτερόλεπτα) μεταξύ κάθε προσπάθειας δημιουργίας.

Για την εύρεση θέσης εμφάνισης, ο αλγόριθμος υπολογίζει τυχαίες συντεταγμένες X και Z εντός των ορίων του Terrain, ενώ το ύψος (άξονας Y) προσδιορίζεται μέσω της Terrain.SampleHeight(). Πριν δημιουργηθεί το ζόμπι, ελέγχονται δύο ακόμη συνθήκες:

- Η θέση πρέπει να απέχει τουλάχιστον 15 μονάδες από τον παίκτη, ώστε να μην εμφανίζονται ζόμπι ακριβώς πάνω του.
- Η θέση πρέπει να είναι έγκυρη πάνω στο NavMesh, μέσω της NavMesh.SamplePosition(), εξασφαλίζοντας ότι το ζόμπι θα μπορεί να κινηθεί σωστά.

Όταν βρεθεί έγκυρη θέση, το ζόμπι δημιουργείται με Instantiate() και προστίθεται σε μια λίστα που παρακολουθεί τα ζωντανά ζόμπι. Με τον τρόπο αυτό, το σύστημα εξασφαλίζει μια συνεχή ροή εχθρών, διατηρώντας παράλληλα την απόδοση του παιχνιδιού σε σταθερά επίπεδα.

```

1 reference
IEnumerator SpawnLoop()
{
    while (true)
    {
        yield return new WaitForSeconds(spawnInterval);

        if (GameStats.gameOver) continue;

        spawnedZombies.RemoveAll(zombie => zombie == null);

        if (spawnedZombies.Count < maxZombiesAlive)
        {
            TrySpawnZombie();
        }
    }
}

1 reference
void TrySpawnZombie()
{
    Terrain terrain = Terrain.activeTerrain;
    Vector3 terrainSize = terrain.terrainData.size;
    Vector3 terrainPos = terrain.transform.position;

    You, last week • Big Reset Commit
    float marginX = terrainSize.x * 0.1f;
    float marginZ = terrainSize.z * 0.1f;

    for (int attempt = 0; attempt < 10; attempt++)
    {
        float x = terrainPos.x + marginX + Random.Range(0f, terrainSize.x - 2 * marginX);
        float z = terrainPos.z + marginZ + Random.Range(0f, terrainSize.z - 2 * marginZ);
        float y = terrainPos.y + terrain.SampleHeight(new Vector3(x, 0, z));
        Vector3 candidate = new Vector3(x, y, z);

        if (player != null && Vector3.Distance(candidate, player.transform.position) < minDistanceFromPlayer)
        {
            continue;
        }

        if (NavMesh.SamplePosition(candidate, out NavMeshHit hit, 5f, NavMesh.AllAreas))
        {
            GameObject zombie = Instantiate(zombiePrefab, hit.position, Quaternion.identity);
            spawnedZombies.Add(zombie);
            return;
        }
    }
}

```

Εικόνα 6-77 ZombieSpawner Script

6.8.3 Zombie Animations

Το σύστημα animation των ζόμπι βασίζεται σε τέσσερα boolean parameters, τα οποία ελέγχουν ποιο animation θα αναπαράγεται ανάλογα με την κατάσταση του ζόμπι. Κάθε parameter αντιστοιχεί σε μία συγκεκριμένη κίνηση:

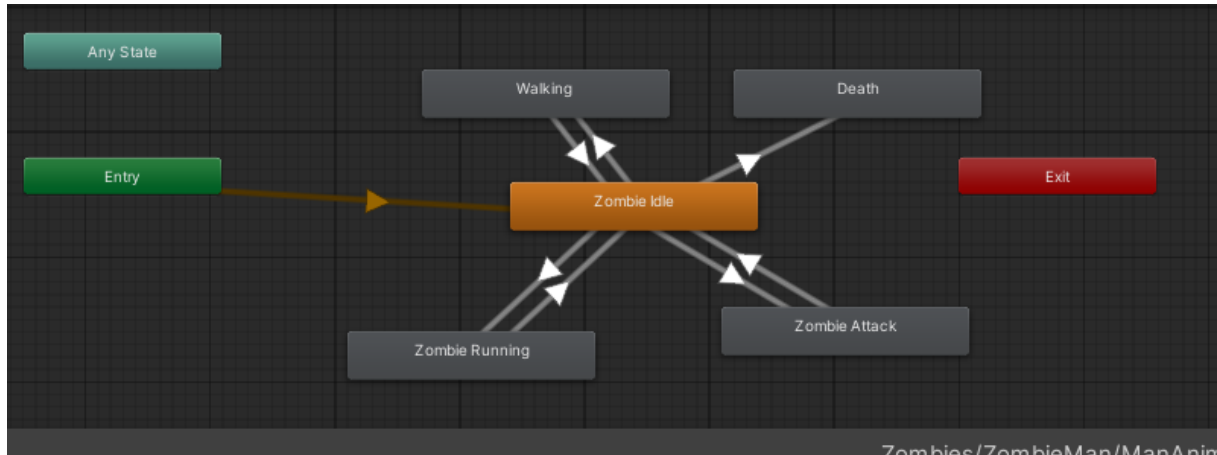
isZombieWalking: Ενεργοποιείται όταν το ζόμπι βρίσκεται στην κατάσταση WANDER, αναπαράγοντας το animation του περπατήματος.

isZombieRunning: Ενεργοποιείται στην κατάσταση CHASE, δίνοντας την αίσθηση του τρεξίματος και της καταδίωξης.

isZombieAttacking: Ενεργοποιείται κατά την επίθεση (ATTACK state), αναπαράγοντας το animation χτυπήματος.

Ενεργοποιείται στον θάνατο (DEAD state), αναπαράγοντας το animation θανάτου.

Για να αποφευχθεί η ταυτόχρονη αναπαραγωγή πολλαπλών animations χρησιμοποιείται η μέθοδος `TurnOffTriggers()`, η οποία απενεργοποιεί όλα τα boolean parameters πριν από κάθε μετάβαση σε νέα κατάσταση. Με τον τρόπο αυτό, εξασφαλίζεται ότι κάθε φορά παίζει μόνο ένα animation, προσφέροντας ομαλές και ρεαλιστικές μεταβάσεις στις κινήσεις των ζόμπι. Στην Εικόνα 6-82 παρουσιάζεται το διάγραμμα του Animator Controller, όπου φαίνονται οι καταστάσεις (Idle, Walking, Running, Attack, Death) και οι μεταβάσεις μεταξύ τους, όπως αυτές ορίστηκαν στο Unity.



Εικόνα 6-78 Το διάγραμμα του Animator Controller για τα ζόμπι, με τις πέντε βασικές καταστάσεις (Idle, Walking, Running, Attack, Death) και τις μεταβάσεις τους.

6.8.4 Zombie Models

Υλοποιήθηκαν τρεις διαφορετικοί τύποι ζόμπι, ο καθένας με διαφορετικά χαρακτηριστικά και επίπεδο απειλής. Οι διαφορές τους καθορίζονται από τρεις βασικές παραμέτρους: την ταχύτητα βαδίσματος (`walkSpeed`), την ταχύτητα καταδίωξης (`runSpeed`) και τη ζημιά που προκαλούν (`damageAmount`) (βλ. Εικόνα 6-83).

Τύπος Zombie	walkSpeed	runSpeed	damageAmount
Jill	2	4	10
ZombieMan	2.5	4.5	10
Boss	1.5	5	20

Εικόνα 6-79 Zombie configuration table

6.8.5 Zombie Sound

Το σύστημα ήχου των ζόμπι περιλαμβάνει:

Splash Sound: Παίζεται κάθε φορά που το ζόμπι χτυπάει τον παίκτη μέσω του AudioSource splash.

6.9 HUD

Το HUD (Heads-Up Display) είναι το σύστημα UI που παρουσιάζει κρίσιμες πληροφορίες στον παίκτη κατά τη διάρκεια του gameplay. Το σύστημα αποτελείται από τέσσερα κύρια στοιχεία: τη health bar, τον ammo counter, τον kill counter και το radar/minimap system. Όλα αυτά τα UI elements εμφανίζονται ταυτόχρονα στο Canvas του MainScene, ενημερώνοντας τον παίκτη σε πραγματικό χρόνο (βλ. Εικόνα 6-84).



Εικόνα 6-80 HUD From Game View

6.9.1 Health Bar

Τα βασικά HUD elements δημιουργούνται χρησιμοποιώντας τα built-in UI components της Unity. Το Health Bar είναι ένα Slider component που τοποθετήθηκε κεντρικά στο κάτω μέρος της οθόνης. Αναπαριστά την τρέχουσα υγεία του παίκτη ως ένα οπτικό bar που μειώνεται όταν ο παίκτης λαμβάνει ζημιά. Το script MainCharacterController έχει μια public μεταβλητή public Slider healthBar που αναφέρεται σε αυτό το component. Κάθε φορά που ο παίκτης λαμβάνει ζημιά και καλείται η TakeHit() μέθοδος, ο παίκτης ενημερώνει την τιμή του slider χρησιμοποιώντας healthBar.value = health, δημιουργώντας ένα ομαλό animation μείωσης.

6.9.2 Ammo Counter

Το Ammo Counter αποτελείται από δύο Text UI elements που εμφανίζουν τα πυρομαχικά του γεμιστήρα και τα πυρομαχικά της εφεδρείας. Ο πρώτος text δείχνει το ammoClip, το οποίο ξεκινά με τιμή 25, και ο δεύτερος δείχνει το ammo, το οποίο ξεκινά με τιμή 50. Όταν ο παίκτης πυροβολεί, τα πυρομαχικά του γεμιστήρα μειώνονται και ο text ενημερώνεται μέσω ammoClipAmmount.text =

ammoClip.ToString()). Όταν κάνει reload, τα πυρομαχικά μεταφέρονται από την εφεδρεία στον γεμιστήρα.

6.9.3 Kill Counter

Ο Kill Counter είναι ένα Text UI element που εμφανίζει τον αριθμό των ζόμπι που έχει σκοτώσει ο παίκτης. Η ενημέρωσή του γίνεται μέσω του script KillCounter.cs, το οποίο αξιοποιεί το Observer Pattern με C# events. Κατά την ενεργοποίησή του (OnEnable()), το σύστημα εγγράφεται στο event GameStats.OnZombieKilled, ώστε κάθε φορά που σκοτώνεται ένα ζόμπι να καλείται αυτόματα η μέθοδος UpdateKillCount(). Η UpdateKillCount() ενημερώνει το αντίστοιχο Text UI με τον τρέχοντα αριθμό θανάτων, ο οποίος λαμβάνεται από τη GameStats.zombieKills, προσφέροντας έτσι ανατροφοδότηση σε πραγματικό χρόνο στον παίκτη.

6.9.4 Radar / Minimap System

Το σύστημα υλοποιείται μέσω του script Radar.cs, το οποίο διαχειρίζεται μια στατική λίστα (radarObjects) που περιέχει όλα τα αντικείμενα που πρέπει να εμφανιστούν στο radar. Κάθε φορά που δημιουργείται ένα ζόμπι, καλείται η μέθοδος SubscribeRadarObject(), η οποία δημιουργεί ένα αντίγραφο του εικονιδίου (Image) και το προσθέτει στη λίστα μαζί με μια αναφορά στο αντικείμενο. Αντίστοιχα, όταν ένα ζόμπι πεθαίνει, καλείται η UnsubscribeRadarObject(), η οποία αφαιρεί το αντικείμενο από τη λίστα και καταστρέφει το εικονίδιο του.

Κατά την εκτέλεση του Update(), το script υπολογίζει για κάθε εγγεγραμμένο αντικείμενο τη σχετική του θέση ως προς τον παίκτη, σύμφωνα με τον ακόλουθο κώδικα (βλ. Εικόνα 6-81):

```
Vector3 radarPosition = radarObject.player.transform.position - playerPosition.position;
float distance = Vector3.Distance(playerPosition.position, radarObject.player.transform.position) *
mapScale;
```

Εικόνα 6-81 Κώδικας υπολογισμού παίκτη

Στη συνέχεια, υπολογίζει τη γωνία μεταξύ του παίκτη και του αντικειμένου, προσαρμόζοντάς την με βάση την κατεύθυνση που κοιτάζει ο παίκτης (βλ. Εικόνα 6-82):

```
float deltaY = Mathf.Atan2(radarPosition.x, radarPosition.z) * Mathf.Rad2Deg - 270 - playerPosition.
eulerAngles.y;
radarPosition.x = distance * Mathf.Cos(deltaY * Mathf.Deg2Rad) * -1;
radarPosition.z = distance * Mathf.Sin(deltaY * Mathf.Deg2Rad);
```

Εικόνα 6-82 Υπολογισμός αντικειμένων με βάση τον παίκτη

Τέλος, το εικονίδιο τοποθετείται στη σωστή θέση πάνω στο radar, λαμβάνοντας υπόψη το μέγεθος και τη θέση του UI panel (βλ. Εικόνα 6-83):

```
radarObject.radarIcon.transform.SetParent(this.transform);
RectTransform radarRectTransform = this.GetComponent<RectTransform>();
radarObject.radarIcon.transform.position = new Vector3(radarPosition.x + radarRectTransform.pivot.x,
radarPosition.z + radarRectTransform.pivot.y, 0) + this.transform.position;
```

Εικόνα 6-83 Προσθήκη αντικειμένων στο radar

6.10 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε αναλυτικά η διαδικασία ανάπτυξης και υλοποίησης της πτυχιακής εργασίας, ξεκινώντας από τη δημιουργία του έργου στο Unity και φτάνοντας μέχρι την ολοκλήρωση των επιμέρους συστημάτων του παιχνιδιού. Περιεγράφηκαν βήμα προς βήμα όλα τα στάδια κατασκευής — από τα μενού πλοήγησης και τη διαμόρφωση του Terrain, μέχρι την υλοποίηση του χαρακτήρα του παίκτη, του συστήματος μάχης, της τεχνητής νοημοσύνης των εχθρών και της διεπαφής χρήστη.

Ιδιαίτερη έμφαση δόθηκε στα επιμέρους συστήματα που συνθέτουν την εμπειρία του παιχνιδιού: το σύστημα κίνησης και κάμερας, τα animations μέσω του Animator Controller, το σύστημα υγείας και οι μηχανισμοί αναπλήρωσης πόρων, καθώς και το σύστημα πυροβολισμού με τον συγχρονισμό ήχου και animation μέσω των Animation Events. Παράλληλα, αναλύθηκε η λειτουργία της τεχνητής νοημοσύνης των ζόμπι μέσω της Πεπερασμένης Μηχανής Καταστάσεων (FSM), το σύστημα δημιουργίας εχθρών και η διαχείριση της πλοήγησής τους με το NavMesh και τον αλγόριθμο A*.

Κεφάλαιο 7ο: Συμπεράσματα και Μελλοντικές Βελτιώσεις

7.1 Συμπεράσματα

Η παρούσα πτυχιακή εργασία ανέδειξε τη δυνατότητα δημιουργίας ενός πλήρως λειτουργικού FPS παιχνιδιού επιβίωσης, αξιοποιώντας τη μηχανή Unity και τη γλώσσα προγραμματισμού C#. Το Dead Horizon αποτελεί ένα ολοκληρωμένο παράδειγμα εφαρμογής των θεωρητικών εννοιών που αναλύθηκαν στα προηγούμενα κεφάλαια, όπως η τρισδιάστατη γραφική, η φυσική προσομοίωση, η τεχνητή νοημοσύνη και οι μηχανές παιχνιδιών.

Ο τεχνολογικός συνδυασμός Unity και των υποσυστημάτων της (NavMesh, Physics, Animator, Particle System) παρείχε αξιόπιστη λειτουργία, ταχύτερη ανάπτυξη χάρη στα έτοιμα εργαλεία και τη δυνατότητα γρήγορου πρωτοτυποποίησης. Η αρχιτεκτονική που υλοποιήθηκε είναι καθαρή και επεκτάσιμη, επιτρέποντας την προσθήκη νέων χαρακτηριστικών με την ίδια λογική. Η χρήση της Πεπερασμένης Μηχανής Καταστάσεων (FSM) για την τεχνητή νοημοσύνη των ζόμπι απέδειξε ότι πρόκειται για μια απλή, αποτελεσματική και εύκολα συντηρήσιμη λύση, ενώ ο συνδυασμός της με το NavMeshAgent εξασφάλισε ρεαλιστική πλοήγηση των εχθρών στο τρισδιάστατο περιβάλλον.

Παράλληλα, αναδείχθηκαν πρακτικοί περιορισμοί, όπως η ανάγκη βελτιστοποίησης της απόδοσης για ομαλή λειτουργία σε μεγαλύτερο αριθμό ταυτόχρονων ζόμπι, η διαχείριση της μνήμης και η αποφυγή σφαλμάτων κατά τις μεταβάσεις μεταξύ των καταστάσεων της FSM. Οι περιορισμοί αυτοί δεν αναιρούν το αποτέλεσμα, αλλά κατευθύνουν συγκεκριμένες βελτιώσεις για ακόμη πιο ομαλή εμπειρία.

7.2 Μελλοντικές Βελτιώσεις

Για να βελτιωθεί ακόμη περισσότερο η εμπειρία του παιχνιδιού, προτείνονται οι παρακάτω στοχευμένες παρεμβάσεις, οι οποίες ενισχύουν την ευχρηστία, τη σταθερότητα σε διαφορετικές συσκευές και την εύκολη επέκταση σε νέα χαρακτηριστικά, χωρίς να αλλάξει ο βασικός κορμός της εφαρμογής.

- **Προσθήκη νέων όπλων και αντικειμένων:** Εισαγωγή διαφορετικών τύπων όπλων (πιστόλι, καραμπίνα, εκρηκτικά) με ξεχωριστά χαρακτηριστικά (ζημιά, ταχύτητα βολής, ακρίβεια), καθώς και νέων αντικειμένων όπως πανοπλία, αναβάθμιση ταχύτητας ή προσωρινά power-ups.
- **Σύστημα αποθήκευσης προόδου:** Δημιουργία συστήματος αποθήκευσης του σκορ (high scores) και των στατιστικών του παίκτη, ώστε να υπάρχει κίνητρο για βελτίωση και επανάληψη.
- **Προσθήκη ήχων περιβάλλοντος:** Εμπλουτισμός του ηχητικού τοπίου με ήχους άνεμου, πουλιών, βημάτων σε διαφορετικές επιφάνειες και άλλα ambient ηχητικά εφέ, για ακόμη μεγαλύτερη αίσθηση βύθισης.
- **Multiplayer:** Προσθήκη δυνατότητας συνεργατικού ή ανταγωνιστικού παιχνιδιού με άλλους παίκτες μέσω δικτύου, αυξάνοντας σημαντικά την επαναληπτικότητα και τη διάρκεια ζωής του παιχνιδιού.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] Levvel, "Video Game Industry Statistics," 2025. [Online]. Available: <https://levvel.com/el/statistics/video-game-industry/> [Accessed: 16 Αυγούστου 2026].
- [2] Wikipedia, "Βιντεοπαιχνίδι," [Online]. Available: <https://el.wikipedia.org/wiki/Βιντεοπαιχνίδι>. [Accessed: 16 Αυγούστου 2026].
- [3] Θωδωρής Λαϊνάς, "Ο νέος κόσμος των απτικών θαυμάτων," ΤΟ ΒΗΜΑ, 15 Φεβρουαρίου 2016. [Online]. [Accessed: 17 Αυγούστου 2026]. Available: <https://www.tovima.gr/2016/02/15/science/o-neos-kosmos-twn-aptikwn-thaymatwn/> [Accessed: 17 Αυγούστου 2026].
- [4] V4T, "Βιντεοπαιχνίδια," Nov 2017. [Online]. Available: https://v4t.pixel-online.org/manual_pdf/10/gr.pdf [Accessed: 17 Αυγούστου 2026].
- [5] EBSCO Research, "History of Video Games," 2021. [Online]. Available: <https://www.ebsco.com/research-starters/history/history-video-games> [Accessed: 17 Αυγούστου 2026].
- [6] Smithsonian Institution, "Video Game History," American History Museum. [Online]. Available: <https://www.si.edu/spotlight/the-father-of-the-video-game-the-ralph-baer-prototypes-and-electronic-games/video-game-history> [Accessed: 18 Αυγούστου 2026].
- [7] iD Tech, «Different types of video game genres», iD Tech Blog. [Online]. Available: <https://www.idtech.com/blog/different-types-of-video-game-genres> [Accessed: 18 Αυγούστου 2026].
- [8] HP Tech Takes, «Video game genres: The ultimate glossary», HP Official Site. [Online]. Available: <https://www.hp.com/us-en/tech-takes/gaming/glossary/video-game-genres.html> [Accessed: 18 Αυγούστου 2026].
- [9] Gameopedia, «The ultimate guide to video game genres», Gameopedia Blog. [Online]. Available: <https://gameopedia.com/blogs/the-ultimate-guide-to-video-game-genres> [Accessed: 18 Αυγούστου 2026].
- [10] Dmytro Kovtun, "Key roles in a game development team – 2024 edition," Pingle Studio, Jun 5, 2024. [Online]. Available: <https://pinglestudio.com/blog/key-roles-in-a-game-development-team-2024-edition/> [Accessed: 19 Αυγούστου 2026].
- [11] David Mullich, "Roles On A Game Development Team," David Mullich Blog, Feb 2, 2015. [Online]. Available: <https://davidmullich.com/2015/02/02/roles-on-a-game-development-team/> [Accessed: 20 Αυγούστου 2026].
- [12] Nikolay Minaiev, "Roles in game development: building a successful game development team," iLogos Game Studios Blog, Feb 10, 2025. [Online]. Available:

- <https://ilogos.biz/roles-in-game-development/> [Accessed: 20 Αυγούστου 2026].
- [13] "Rigid body physics," Godot Engine Game Development Projects, O'Reilly Media, 2018.
[Online]. Available: <https://www.oreilly.com/library/view/godot-engine-game/> [Accessed: 21 Αυγούστου 2026].
- [14] "3D Graphics and Rendering," Concept Art Empire, May 12, 2020. [Online].
Available: <https://conceptartempire.com/shaders/> [Accessed: 21 Αυγούστου 2026]
- [15] "Video Game Physics Tutorial - Part II: Collision Detection for Solid Objects,"
Toptal, May 11, 2026. [Online]. Available: <https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects> [Accessed: 22 Αυγούστου 2026]
- [16] "Rendering and Shading," Unity Learn. [Online]. Available:
<https://learn.unity.com/tutorial/rendering-and-shading> [Accessed: 22 Αυγούστου 2026]
- [17] J. Millington and J. Funge, Artificial Intelligence for Games, 3rd ed. Boca Raton, FL, USA: CRC Press, 2019.
- [18] S. Rabin, Ed., Game AI Pro 3: Collected Tales of AI Wisdom. Boca Raton, FL, USA: CRC Press, 2017.
- [19] J. Gregory, Game Engine Architecture, 3rd ed. Boca Raton, FL, USA: CRC Press, 2018.
- [20] M. McShaffry and D. Graham, Game Coding Complete, 4th ed. Boston, MA, USA: Course Technology, 2012.
- [21] E. Lengyel, Foundations of Game Engine Development, Vol. 1: Mathematics. Terathon Software LLC, 2016.
- [22] Unity Technologies, "Unity User Manual - Editor Windows," Unity Documentation.
[Online]. Available: <https://docs.unity3d.com/Manual/UsingTheSceneView.html> [Accessed: 23 Αυγούστου 2026]
- [23] Unity Technologies, "Getting Started with Unity," Unity Learn. [Online]. Available:
<https://learn.unity.com/> [Accessed: 23 Αυγούστου 2026]
- [24] Unity Technologies, "GameObjects and Components," Unity Manual. [Online]. Available:
<https://docs.unity3d.com/Manual/GameObjects.html> [Accessed: 24 Αυγούστου 2026]
- [25] Unity Technologies, "Lighting," Unity Manual. [Online]. Available:
<https://docs.unity3d.com/Manual/LightingOverview.html> [Accessed: 24 Αυγούστου 2026]
- [26] Unity Technologies, "Lighting Settings," Unity Manual. [Online]. Available:
<https://docs.unity3d.com/Manual/GlobalIllumination.html> [Accessed: 25 Αυγούστου 2026]
- [27] Unity Technologies, "Animator and Animation System," Unity Manual. [Online]. Available:
<https://docs.unity3d.com/Manual/AnimationOverview.html> [Accessed: 25 Αυγούστου 2026]

[28] Unity Technologies, "Animation Clips," Unity Manual. [Online].Available: <https://docs.unity3d.com/Manual/AnimationClips.html> [Accessed: 25Αυγούστου 2026]