



Πτυχιακή εργασία

Gaming Development with Unity 3D

Ονοματεπώνυμο : Σιάνος Γεώργιος

Αριθμός Μητρώου : Π2013040

Επιβλέπων Καθηγητής : Μυλωνάς Φοίβος

Η παρούσα πτυχιακή

υποβλήθηκε στο

Τμήμα Πληροφορικής Ιονίου Πανεπιστημίου

Σεπτέμβριος 2020

Πρόλογος – Ευχαριστίες

Αυτή η εργασία μου έδωσε την δυνατότητα να ασχοληθώ με ένα αντικείμενο που με ενδιαφέρει πραγματικά και που χρόνια πριν τις σπουδές μου κινούσε το ενδιαφέρον μου . Ήταν για μένα το πρώτο, ελπίζω από πολλά, βήμα στον κόσμο του Gaming Development.

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κύριο Μυλωνά Φοίβο, αφενός, γιατί μου έδωσε την ευκαιρία να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα, αφετέρου, για την επιμονή και την εμπιστοσύνη του όλο αυτό το διάστημα, αλλά και τα υπόλοιπα μέλη της επιτροπής, τον κύριο Κουρουθανάση Παναγιώτη και τον κύριο Στεφανιδάκη Μιχαήλ.

Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την συνεχή υποστήριξή τους.

Περίληψη

Στα πλαίσια της παρούσας διπλωματικής εργασίας παρουσιάζεται η σχεδίαση και η ανάπτυξη ενός παιχνιδιού εικονικής πραγματικότητας που ως κύριο σκοπό έχει την ψυχαγωγία, την διέγερση της φαντασίας και την ανάπτυξη υπαρχουσών ή μη δεξιοτήτων του χρήστη.

Για να πραγματοποιηθεί ο σχεδιασμός του παιχνιδιού χρειάστηκε να μελετηθεί τόσο η θεωρία και η επιστήμη που κρύβονται πίσω από την ανάπτυξη ηλεκτρονικών παιχνιδιών, όσο και οι ιδιαιτερότητες που προκύπτουν από ένα παιχνίδι ψυχαγωγικού περιεχομένου. Για τον λόγο αυτόν, η έρευνα επεκτάθηκε σε θεωρίες και μεθόδους ψυχαγωγίας καθώς έγινε και μελέτη αντίστοιχων ψυχαγωγικών παιχνιδιών, ιδίως παιχνιδιών βολής πρώτου προσώπου ή όπως αλλιώς είναι ευρέως γνωστά, First Person Shooter (FPS) games.

Το παιχνίδι απευθύνεται σε ηλικίες άνω των 13 ετών, καθώς το ζητούμενο είναι η επιβίωση του χρήστη σε ένα αφιλόξενο περιβάλλον με εχθρούς που ο χρήστης καλείται να εξοντώσει. Έχει δοθεί ιδιαίτερη έμφαση στη παροχή ερεθισμάτων και προκλήσεων με στόχο να εξαφθεί το ενδιαφέρον του χρήστη. Επιπλέον στοιχεία που στοχεύουν σε αυτό είναι η ελκυστική παρουσίαση του περιβάλλοντος και των φαινομένων, η επιβράβευση του παίχτη, καθώς και η σωστή και έγκυρη καθοδήγησή του κατά την διάρκεια του παιχνιδιού. Ο εκάστοτε χρήστης καλείται να αντιμετωπίσει μια σειρά προκλήσεων. Αυτό επιτυγχάνεται μέσω τεχνητών εμποδίων και διαφόρων αντικειμένων και στόχων που αποσκοπούν στην λήψη αποφάσεων από τον χρήστη σε πραγματικό χρόνο.

Η δημιουργία του εν λόγω παιχνιδιού έγινε με τη χρήση της Unity, μιας μηχανής ανάπτυξης παιχνιδιών, όχι και τόσο φιλικής προς τον χρήστη αλλά με πληθώρα δυνατοτήτων.

Λέξεις Κλειδιά: παιχνίδι, εικονική πραγματικότητα, ψυχαγωγία, παιχνίδι βολής πρώτου προσώπου, Unity.

Abstract

In the context of this dissertation, the design and development of a virtual reality game is presented, which has as its main purpose the entertainment, the stimulation of the imagination and the development of existing or non-existing skills of the user.

In order to design the game, both the theory and the science behind the development of electronic games and the peculiarities of an entertaining game had to be studied. For this reason, the research was extended to theories and methods of entertainment as well as the study of similar games in the field of entertainment and more specifically to first person shooter games.

The game is aimed at ages over 13 as its content has to do with the survival of the user in an inhospitable environment which includes enemies that the user must kill. Particular emphasis has been placed on providing stimuli and challenges to the user's interest. Additional factors that contribute to this are the attractive presentation of the environment and the phenomena, the rewarding of the player as well as his correct and valid guidance during the game. Each user is faced with a number of challenges. This takes place through the creation of artificial barriers, objects and targets that are intended by the user to make decisions in real time.

This game was created using Unity, a gaming development machine, not so user-friendly but with a wide range of features.

Keywords: game, virtual reality, entertainment, first-person shooter game, Unity.

Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1 : ΕΙΣΑΓΩΓΗ.....	12
1.1 Σύντομη Ιστορική Αναδρομή.....	12
1.2 Μηχανές Ανάπτυξης Βιντεοπαιχνιδιών.....	13
1.3 Οφέλη Βιντεοπαιχνιδιών.....	14
ΚΕΦΑΛΑΙΟ 2 : UNITY.....	15
2.1 Ιστορία της Unity.....	15
2.2 Παγκόσμια Επέκταση και Οικονομικά Στοιχεία.....	18
2.3 Multiplatform.....	19
2.4 Άλλες Χρήσεις.....	20
2.5 Παιχνίδια Φτιαγμένα με Unity.....	21
ΚΕΦΑΛΑΙΟ 3 : ΔΟΜΗ ΚΑΙ ΕΡΓΑΛΕΙΑ.....	25
3.1 Unity Editor.....	25
3.2 Παράθυρα του Editor.....	25
3.2.1 Scene View.....	26
3.2.2 Game View.....	27
3.2.3 Project.....	28
3.2.4 Hierarchy.....	28
3.2.5 Inspector.....	28
3.3 Γραμμή Μενού.....	29
3.3.1 File.....	29
3.3.2 Edit.....	29
3.3.3 Asset.....	29
3.3.4 GameObject.....	30
3.3.5 Component.....	31
3.3.6 Window.....	32
3.3.7 Build Settings.....	32
3.4 Asset Pipeline and Workflow.....	33

3.5 Asset Store.....	34
3.6 Λήψη της Unity και Εκδόσεις.....	35
ΚΕΦΑΛΑΙΟ 4 : ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ.....	37
4.1 Level Design.....	37
4.2 Creating FPS Player.....	40
4.3 Coding Player.....	42
4.3.1 Coding Player's Movement and Jump.....	42
4.3.2 Mouse Look and Mouse Controls.....	44
4.3.3 Sprint and Crouch.....	46
4.4 "WeaponHandler" και "Weapon Manager".....	50
4.5 Player Attack.....	54
4.5.1 "ArrowAndBow" Script.....	59
4.6 Creating Enemies.....	61
4.6.1 Detecting and Dealing Damage.....	68
4.7 Health.....	69
4.8 Player Stats UI.....	72
4.9 Creating Enemy Manager.....	74
4.10 Coins and Exit Door.....	78
4.11 Creating Menu.....	83
5. ΒΙΒΛΙΟΓΡΑΦΙΑ.....	85

Πίνακας Εικόνων

<i>Εικόνα 1 - Μηχανές ανάπτυξης βιντεοπαιχνιδιών.....</i>	<i>13</i>
<i>Εικόνα 1 - Το λογότυπο της Unity.....</i>	<i>15</i>
<i>Εικόνα 2 - Ο CEO της εταιρείας John Riccitiello.....</i>	<i>19</i>
<i>Εικόνα 3 - Το λογότυπο της Nintendo.....</i>	<i>20</i>
<i>Εικόνα 4 - Το παιχνίδι GooBall (2005).....</i>	<i>21</i>
<i>Εικόνα 5 - Το παιχνίδι Bad Piggies (2012).....</i>	<i>22</i>
<i>Εικόνα 6 - Το παιχνίδι Escape Plan (2012).....</i>	<i>22</i>
<i>Εικόνα 7 - Το παιχνίδι Pokemon Go (2016).....</i>	<i>24</i>
<i>Εικόνα 8 - Το παιχνίδι Hollow Knight (2017).....</i>	<i>24</i>
<i>Εικόνα 1 - Editor's interface.....</i>	<i>25</i>
<i>Εικόνα 2 - Scene View.....</i>	<i>26</i>
<i>Εικόνα 3 - Game View.....</i>	<i>27</i>
<i>Εικόνα 4 - Project.....</i>	<i>28</i>
<i>Εικόνα 5 - Build Settings.....</i>	<i>32</i>
<i>Εικόνα 6 - Asset Store.....</i>	<i>34</i>
<i>Εικόνα 7 - Pro edition plans.....</i>	<i>35</i>
<i>Εικόνα 8 - Pro edition plans 2.....</i>	<i>36</i>
<i>Εικόνα 1 - Terrain Editor.....</i>	<i>37</i>
<i>Εικόνα 2 - Τελικό Terrain.....</i>	<i>38</i>
<i>Εικόνα 3 - Περιβάλλον παιχνιδιού.....</i>	<i>39</i>
<i>Εικόνα 4 - Χωριό.....</i>	<i>39</i>
<i>Εικόνα 5 - Character Controller.....</i>	<i>41</i>
<i>Εικόνα 6 - FPS Player.....</i>	<i>41</i>
<i>Εικόνα 7 - Inspector του "Player" με τα script "Movement", "SprintAndCrouch".....</i>	<i>49</i>
<i>Εικόνα 8 - Inspector του "LookRoot" με το script "MouseLook".....</i>	<i>49</i>
<i>Εικόνα 9 - Το όπλο Assault Rifle.....</i>	<i>52</i>
<i>Εικόνα 10 - Το script "WeaponHandler" στο Assault Rifle.....</i>	<i>52</i>

Εικόνα 11 - Στόχος.....	54
Εικόνα 12 - Revolver χωρίς zoom.....	58
Εικόνα 13 - Revolver με zoom.....	58
Εικόνα 14 - Arrow και Spear spawning point.....	60
Εικόνα 15 - Εχθροί "Cannibal" και "Boar".....	61
Εικόνα 16 - Walkable terrain.....	62
Εικόνα 17 - Καρτέλα "Navigation".....	62
Εικόνα 18 - Attack Point.....	68
Εικόνα 19 - Player Stats.....	73
Εικόνα 20 - "Enemy Manager" Inspector.....	77
Εικόνα 21 - "Enemy Manager".....	77
Εικόνα 22 - Coins.....	78
Εικόνα 23 - Πόρτα τερματισμού μετά τη συλλογή των νομισμάτων.....	79
Εικόνα 24 - Πόρτα τερματισμού.....	79
Εικόνα 25 - Μήνυμα "You Win".....	82
Εικόνα 26 - Αρχικό Μενού.....	83
Εικόνα 27 - Pause Μενού.....	84

ΚΕΦΑΛΑΙΟ 1 : ΕΙΣΑΓΩΓΗ

1.1 Σύντομη Ιστορική Αναδρομή

Τα παιχνίδια αποτελούν αναπόσπαστο κομμάτι της κοινωνίας από τα προϊστορικά ακόμα χρόνια με τα πρώτα ευρήματα να χρονολογούνται κάπου στο 3500 π.Χ. και να είναι φτιαγμένα από πέτρες, ξύλα και οστά. Σαφώς από την περίοδο εκείνη μέχρι σήμερα τα παιχνίδια έχουν γνωρίσει μεγάλη εξέλιξη. Την δεκαετία του '50 εμφανίζονται τα πρώτα ηλεκτρονικά παιχνίδια για τα οποία όμως η μεγάλη άνθηση και η δημοτικότητα θα έρθει αργότερα με τη δημιουργία των πρώτων κονσολών.

Τότε είναι που ξεκινάει και η ιστορία των μηχανών ανάπτυξης παιχνιδιών, κοντά στις αρχές του '90 μετά το ξεκίνημα των 3D γραφικών, κάτι που φυσικά απογείωσε την βιομηχανία των ηλεκτρονικών παιχνιδιών. Κάθε εταιρεία τότε είχε και την δικιά της μηχανή ανάπτυξης και ακόμα και σήμερα πολλές είναι οι εταιρείες που ακολουθούν το πρότυπο αυτό. Βέβαια πλέον υπάρχουν πολλά διαθέσιμα δωρεάν προγράμματα ανάπτυξης παιχνιδιών όπως για παράδειγμα το Unreal Engine ή το Unity, καθώς και πολλά μαθησιακά βίντεο και ενημερωτικά φόρουμ όπου ο καθένας μπορεί να έχει πρόσβαση.

Στα μέσα της δεκαετίας αυτής λοιπόν έχουμε την καθιέρωση του όρου Game Engine και βλέπουμε να ξεκινάει η επανάσταση στο χώρο με την εμφάνιση των παιχνιδιών “Doom” και “Quake”. Η επιτυχία αυτή έκανε τους developers να δημιουργούν τα δικά τους παιχνίδια απλά αλλάάζοντας τα ήδη υπάρχοντα, προσθέτοντας διάφορα όπλα και κάποια γραφικά στοιχεία δηλαδή, και να τα πουλάνε ως ξεχωριστά video games. Σε αυτό βοήθησε σημαντικά η Unreal Engine που όμως μέχρι τότε παρέμενε άγνωστη για τους περισσότερους όπως και πολλές παρόμοιες μηχανές. Μερικές άλλες μηχανές ήταν η Pinball Construction Set (1983), η ASCII's War Game Construction Kit (1983), η Adventure Construction Set (1983), η Shooter – Up Construction Kit (1987), κ.α.

1.2 Μηχανές Ανάπτυξης Βιντεοπαιχνιδιών

Όπως εύκολα φαντάζεται κανείς, ο αριθμός μηχανών ανάπτυξης παιχνιδιών σήμερα έχει πολλαπλασιαστεί και θα μπορούσε κανείς να πει ότι υπάρχουν πλέον αμέτρητες μηχανές αυτού του είδους. Με την ταχύτατη ανάπτυξη της τεχνολογίας, ο αριθμός αυτός συνεχίζει να αυξάνεται, όπως όμως αυξάνεται και το πλήθος των δυνατοτήτων που οι μηχανές αυτές προσφέρουν. Βέβαια οι δυνατότητες αυτές διαφέρουν από μηχανή σε μηχανή, όπως και τα πλεονεκτήματα και μειονεκτήματά τους. Για παράδειγμα το Unity, με την οποία και θα ασχοληθούμε, θεωρείται η καλύτερη μηχανή για τη δημιουργία παιχνιδιών για smartphone και τάμπλετ και αυτό γιατί δεν έχει τόσο καλό rendering ώστε να κάνει ελκυστικά τα παιχνίδια για κονσόλες. Τα script όμως τρέχουν πιο γρήγορα απ' ότι στην UDK, η οποία αντιθέτως έχει το καλύτερο rendering και έτσι έχει κυριαρχήσει στην δημιουργία παιχνιδιών για κονσόλες και υπολογιστές. Οι πιο δημοφιλείς μηχανές ανάπτυξης σήμερα είναι η Unity, η Unreal Engine (ή αλλιώς UDK) και η Cry Engine.



Εικόνα 1 - Μηχανές ανάπτυξης βιντεοπαιχνιδιών

1.3 Οφέλη Βιντεοπαιχνιδιών

Μία από τις πιο διασκεδαστικές και χαλαρωτικές δραστηριότητες είναι με διαφορά τα ηλεκτρονικά παιχνίδια και αυτό ισχύει τόσο για τους μικρούς όσο και για τους μεγάλους. Εκτός από την διασκέδαση και την θετική επίδραση στη ψυχολογία όμως, μέσα από την ενασχόληση με βιντεοπαιχνίδια προκύπτουν κι άλλα οφέλη, όπως η βελτίωση της οπτικής επεξεργασίας πληροφοριών, η λήψη αποφάσεων και η ομαδικότητα. Πιο αναλυτικά το gaming παρέχει σημαντικά οφέλη στους παρακάτω τομείς.

- **Γνωστικές ικανότητες:** Πολλά είναι τα παιχνίδια που διεγείρουν τη μνήμη, την έκταση προσοχής και τη λογική σκέψη. Αυτές τις ικανότητες μπορούν ιδίως να ενισχύσουν τα παιχνίδια στρατηγικής (Strategy Games), αφού προκαλούν τον παίκτη να αναλύσει μια κατάσταση σε περιορισμένο χρόνο και να οργανώσει μία άμυνα ή επίθεση.
- **Λήψη αποφάσεων:** Τα παιχνίδια ρόλου (Role Playing Games – RPGs) βοηθούν σημαντικά την ανάπτυξη της τάχιστης λήψης αποφάσεων, καθώς απαιτούν από τον παίκτη να λάβει σε κλάσματα δευτερολέπτου αποφάσεις, οι οποίες επηρεάζουν και καθορίζουν τη πορεία του παιχνιδιού και ανάλογα με αυτές θα κριθεί αν ο τερματισμός θα γίνει με επιτυχία.
- **Επεξεργασία οπτικής πληροφορίας :** Τα First Person Shooter παιχνίδια (FPS), με τα οποία και θα ασχοληθούμε περισσότερο, απαιτούν από τον παίκτη συνεχή επαγρύπνηση και γρήγορη ανάλυση των στοιχείων της οθόνης, κάτι που τον κάνει συνεχώς να εστιάζει σε διάφορα σημεία της οθόνης με γρήγορε εναλλαγές μεταξύ αυτών. Με τον τρόπο αυτό, ο εγκέφαλος μαθαίνει να μοιράζει την προσοχή του και να βελτιώνει την επεξεργασία των εισερχόμενων οπτικών πληροφοριών.
- **Ομαδικότητα :** Αν όχι το σημαντικότερο, τότε σίγουρα από τα πλέον σημαντικά οφέλη. Πνεύμα ομαδικότητας προκύπτει όχι μόνο σε online παιχνίδια με πολλές ομάδες, αλλά ακόμα και σε offline παιχνίδια συνεργασίας με φίλους. Τα παιχνίδια αυτά απαιτούν από τους παίκτες να συνεργαστούν στενά σε μια ομάδα για την επίτευξη της νίκης.

ΚΕΦΑΛΑΙΟ 2 : UNITY

Η Unity είναι μία από τις πιο δημοφιλείς μηχανές ανάπτυξης παιχνιδιών σε πολλές και διαφορετικές πλατφόρμες. Ακόμη θεωρείται από πολλούς και η πιο ολοκληρωμένη λόγω του μεγάλου φόρουμ της, το οποίο αποτελείται από ένα μεγάλο πλήθος ανθρώπων πρόθυμους να βοηθήσουν στην επίλυση διάφορων προβλημάτων και να μοιραστούν ιδέες και γνώσεις.



Εικόνα 1 - Το λογότυπο της Unity

2.1 Ιστορία της Unity

Όλα ξεκίνησαν στις 21 Μαΐου του 2002 όταν ο Δανός προγραμματιστής Nicholas Francis δημοσίευσε στο Mac OpenGL φόρουμ ζητώντας βοήθεια σε σχέση με ένα shader που προσπαθούσε να εφαρμόσει στη δικιά του μηχανή ανάπτυξης παιχνιδιών. (Shader αποκαλείται το στοιχείο εκείνο που δίνει σε ένα τρισδιάστατο αντικείμενο την εμφάνισή και την αίσθηση του.) Στην δημοσίευση του Nicholas απάντησε έπειτα από λίγο ο Joachim Ante ο οποίος τότε ζούσε στη Γερμανία. Έτσι οι δύο προγραμματιστές συνεργάστηκαν για να φτιάξουν ένα shader σύστημα το οποίο θα δούλευε και στην μηχανή ανάπτυξης του Nicholas, αλλά και σε αυτή του Joachim. Μετά από λίγο όμως αποφάσισαν να συνεργαστούν πάνω σε κάτι άλλο. Αποφάσισαν να απορρίψουν τις μηχανές ανάπτυξης παιχνιδιών που είχαν κατασκευάσει και να δημιουργήσουν μία από κοινού. Η είδηση της ιδέας αυτής ενθουσίασε τον David Helgason, ο οποίος έτρεξε να προστεθεί ως ο τρίτος προγραμματιστής της παρέας.

Το εγχείρημα αυτό άρχισε να εξελίσσεται με την ενοικίαση ενός διαμερίσματος στην Κοπεγχάγη από τον Nicholas και τον Joachim. Ο David έμενε και αυτός εκεί κοντά και έτσι οι τρεις

τους ξεκίνησαν να αναπτύσσουν ένα από τα πιο σημαντικά εργαλεία στην βιομηχανία των βιντεοπαιχνιδιών. Ένας από τους σημαντικότερους λόγους που το Unity γνώρισε τόσο μεγάλη επιτυχία ήταν η χρήση της από ανεξάρτητους προγραμματιστές οι οποίοι δεν μπορούσαν ούτε να δημιουργήσουν την δική τους μηχανή ανάπτυξης, αλλά ούτε και να αποκτήσουν άδεια χρήσης άλλων μηχανών ώστε να εκμεταλλευτούν πλήρως τις διάφορες δυνατότητες που εκείνες παρείχαν. Δύο χρόνια αργότερα, το 2004, η τριάδα μας με κινητήριο δύναμη την δημιουργία μιας μηχανής ανάπτυξης τρισδιάστατου περιβάλλοντος, κατάφερε να λάβει σημαντικά δάνεια, άνοιξε το πρώτο της γραφείο και προσέλαβε μερικούς μηχανικούς.

Έτσι λοιπόν το 2004 ίδρυσαν την εταιρεία τους με την επωνυμία, Over the Edge Entertainment (OTEE) και διευθύνοντα σύμβουλο (CEO) τον Helgason καθώς ήταν ο πιο επικοινωνιακός απ' τους τρεις. Αμέσως άρχισαν να διαμορφώνουν το επιχειρηματικό τους σχέδιο και να μοντελοποιούν την εταιρεία τους με βάση την Βρετανική εταιρεία ανάπτυξης παιχνιδιών Criterion, η οποία είχε ξεκινήσει να γίνεται πολύ επιτυχημένη την περίοδο εκείνη. Το επιχειρηματικό σχέδιο της εταιρείας αυτής δεν στηριζόταν απλώς στην ανάπτυξη της υποκείμενης τεχνολογίας, αλλά και στην κτήση μεγάλων ονομάτων που θα χρησιμοποιούσαν την τεχνολογία αυτή. Το σκεπτικό ήταν ότι οι καταναλωτές δεν θα ήθελαν να αγοράσουν μία ακριβή μηχανή ανάπτυξης χωρίς να ξέρουν ότι μεγάλοι προγραμματιστές παιχνιδιών την χρησιμοποιούν.

Η Over the Edge Entertainment κατάλαβε ότι το περιστασιακό αλλά και το διαδικτυακό gaming επρόκειτο να αναπτυχθούν και αποφάσισαν να επικεντρώσουν τις προσπάθειες τους σε αυτή την αγορά. Μετά από 2 χρόνια προσπάθειας και έχοντας φτάσει κοντά στην κυκλοφορία του Unity στην αγορά, αποφάσισαν να δημιουργήσουν ένα εμπορικό παιχνίδι χρησιμοποιώντας την νέα τους μηχανή. Τα κίνητρα για τη δημιουργία του παιχνιδιού αυτού δεν ήταν απλά για να δοκιμάσουν τις δυνάμεις και τα όρια του Unity, αλλά και για να έχουν κάποια έσοδα τα οποία και χρειαζόταν οπωδήποτε εκείνη την περίοδο. “Κανείς δεν θυμάται πραγματικά πως επιζήσαμε εκείνη την περίοδο, εκτός από το ότι πιθανότατα δεν τρώγαμε πολύ” αναφέρει σε μία συνέντευξή του ο David Helgason. Οι επόμενοι 5 μήνες αφιερώθηκαν στη σχεδίαση του παιχνιδιού GooBall, το οποίο δημοσιεύθηκε από την Ambrosia Software. Η ομάδα του Unity είδε την δημιουργία του παιχνιδιού και ως ευκαιρία να δοκιμάσει τις αντοχές του Unity, να βρει σφάλματα, να αφαιρέσει ενοχλήσεις (bugs) και να διορθώσει τη διεπαφή με το χρήστη πριν την επίσημη κυκλοφορία της μηχανής. Ακόμη, με τα έσοδα που είχε από

το GooBall, η OTEE προσέλαβε ακόμη περισσότερους προγραμματιστές για να βελτιώσει περαιτέρω τη μηχανή της πριν την επίσημη κυκλοφορία της 1.0 τον Ιούνιο του 2005.

Εκείνη την περίοδο το μέγεθος της Unity ήταν το ακριβώς αντίθετο από το σημερινό. Βρισκόταν στα πρώτα της βήματα και η πλειοψηφία των χρηστών αποτελούταν κυρίως από ερασιτέχνες και ανεξάρτητους προγραμματιστές. Όμως, οι εταιρείες που χρησιμοποίησαν το Unity για τα παιχνίδια τους δεν πέτυχαν καλές πωλήσεις και έτσι άλλοι πιθανοί πελάτες δίστασαν να κάνουν χρήση της φοβούμενοι ανάλογες κακές πωλήσεις. Θα χρειαζόταν να περάσουν ακόμα 2 χρόνια ώστε η OTEE να αποδείξει ότι η Unity θα έχει διαρκή υποστήριξη και συνεχείς ενημερώσεις.

Η πρώτη διαθέσιμη για το κοινό έκδοση υποστήριζε μόνο project τα οποία θα τρέχανε σε Mac OS X. Αυτό άλλαξε με την έκδοση 1.1 όπου προστέθηκαν τα Microsoft Windows και τα προγράμματα περιήγησης ιστού (Web browsers). Επίσης η υποστήριξη για Web browsers ήρθε την κατάλληλη στιγμή καθώς οι προγραμματιστές που ήθελαν να δημιουργήσουν παιχνίδια για ιστό αντιμετώπιζαν πολλά προβλήματα και η Unity τους έδινε λύση. Στην έκδοση 1.1 προστέθηκε και υποστήριξη για εξωτερικά πρόσθετα C/C++, επιτρέποντας στους προγραμματιστές να χρησιμοποιούν υλικό και λογισμικό που δεν υποστηρίζονταν από τη Unity. Μετά την επιτυχία της έκδοσης αυτής οι τρεις ιδρυτές της OTEE αρχίζουν να νιώθουν ότι το προϊόν τους ξεκινούσε να ανταποκρίνεται σιγά σιγά στις αρχικές τους προσδοκίες και πέρασαν τους επόμενους τρεις μήνες ασχολούμενοι με τον εντοπισμό σφαλμάτων, τη βελτιστοποίηση και την εύρεση λύσεων για παλαιότερους υπολογιστές που αντιμετώπιζαν προβλήματα με τις κάρτες γραφικών. Οι λύσεις στα προβλήματα αυτά ήρθαν με την έκδοση 1.5.

Μετά από δύο χρόνια έγινε το μεγαλύτερο βήμα με την έκδοση 2.0 της Unity. Η έκδοση αυτή επικεντρωνόταν στο να ενισχύσει την υποστήριξη για Windows. Για την επίτευξη αυτού του στόχου, η ομάδα πρόσθεσε υποστήριξη Microsoft DirectX, κάτι το οποίο ήταν προεγκατεστημένο σε συστήματα Windows Vista σε αντίθεση με το OpenGL το οποίο έπρεπε να εγκατασταθεί ξεχωριστά. Άλλες σημαντικές προσθήκες της έκδοσης αυτής ήταν η προσθήκη σκιάων σε πραγματικό χρόνο, το σύστημα εδάφους, το Unity Asset Server και τέλος ένα νέο σύστημα GUI το οποίο βασιζόταν σε κώδικα. Η κυκλοφορία της Unity 2.0 έγινε το 2007 στο Unite Developer Conference.

Το 2008 βγήκε στην αγορά το πρώτο iPhone και άρχισε α λειτουργεί το App Store. Η γνωστή πλέον ως Unity Technologies αποφάσισε να αναπτύξει μία έκδοση της Unity η οποία θα

χρησιμοποιούνταν για παιχνίδια σε iPhone. Η έκδοση αυτή κυκλοφόρησε τον Δεκέμβριο του 2008 με όνομα Unity iPhone και ήταν διαθέσιμο σε δύο εκδόσεις, Basic και Pro.

Η επόμενη μεγάλη έκδοση είναι η 3.0 και δημοσιεύθηκε στις 27 Σεπτεμβρίου 2010. Το πιο σημαντικό στην έκδοση αυτή, πέρα από τις μεγάλες αναβαθμίσεις και τις νέες προσθήκες, ήταν ότι η ομάδα της Unity συγκέντρωσε όλες τις εκδόσεις που απαιτούσαν διαφορετικό εξωτερικό πρόγραμμα επεξεργασίας (π.χ. Unity iPhone, Wii) και τις ενσωμάτωσε μέσα σε μία έκδοση μαζί με τις υπάρχουσες επιλογές δημοσίευσης. Στην έκδοση αυτή η Unity μετρούσε πλέον 200.000 εγγεγραμμένους χρήστες και κατάφερε να γίνει, αφενός, η νούμερο ένα μηχανή για εκπαιδευτικούς σκοπούς, αφετέρου η καλύτερη επιλογή για έκδοση παιχνιδιών σε smartphone.

Τα χρόνια που ακολούθησαν δημοσιεύθηκαν 2 ακόμα νέες εκδόσεις, το Unity 4 τον Νοέμβριο 2012 και το Unity 5 τον Μάρτιο 2015, το οποίο έφερε μεγάλες αναβαθμίσεις σε ότι αφορά τους ήχους και τα γραφικά. Μέχρι το 2016 η εταιρεία ανέφερε περισσότερους από 5.5 εκατομμύρια εγγεγραμμένους χρήστες.

Σήμερα η Unity έχει δημοσιεύσει 6 Version με τις τελευταίες να ανανεώνονται συνεχώς λόγω της αυξημένης ζήτησης σε κινητά και τάμπλετ, καθώς διαθέτει και μια Pro έκδοση. Επίσης κυριαρχεί στον χώρο της ανάπτυξης mobile games, τόσο σε iOS όσο και σε Android, αφού σύμφωνα με έρευνες χρησιμοποιείται από ένα ποσοστό προγραμματιστών που ξεπερνάει το 50%.

2.2 Παγκόσμια Επέκταση και Οικονομικά Στοιχεία

Η Unity πλέον εδρεύει στο Σαν Φρανσίσκο της Καλιφόρνια με CEO τον John Riccitiello. Έχει επίσης επεκταθεί σε Καναδά, Κίνα, Κολομβία, Δανία, Φινλανδία, Σιγκαπούρη, Σουηδία, Ουκρανία και Ηνωμένο Βασίλειο. Περίπου το 20% των εσόδων τις προέρχεται από τις Ηνωμένες Πολιτείες, το 50% από τις υπόλοιπες χώρες και το 30% από εφαρμογές που δεν αφορούν στην ανάπτυξη παιχνιδιών. Η εταιρία έχει λάβει συνολικά 748,5 εκατομμύρια δολάρια χρηματοδότηση και αποτιμάται σε περίπου 3 δισεκατομμύρια δολάρια με κύκλο εργασιών πάνω από 300 εκατομμύρια δολάρια τον χρόνο. Επί του παρόντος, η Unity Technologies απασχολεί περισσότερους από 300 υπαλλήλους που εργάζονται σε 27 τοποθεσίες. Διαθέτει 2,5 εκατομμύρια εγγεγραμμένους προγραμματιστές, 500.000 μηνιαίους ενεργούς

χρήστες και 347 εκατομμύρια εγκαταστάσεις web player. Αυτά τα στατιστικά στοιχεία ενισχύουν τη θέση της Unity ως μια από τις ταχύτερα αναπτυσσόμενες και μεγαλύτερες κοινότητες προγραμματιστών στον κόσμο. Μεγαλύτερος ανταγωνιστής της Unity Technologies αυτή την στιγμή είναι η Improbable με επικεφαλής τον Herman Narula που είναι παράλληλα συνιδρυτής και διευθύνων σύμβουλος.



Εικόνα 2 - Ο CEO της εταιρείας John Riccitiello

2.3 Multiplatform

Ένας ακόμα λόγος που κάνει την Unity τόσο δημοφιλή και πετυχημένη είναι ότι μπορεί και υποστηρίζει πολλές διαφορετικές πλατφόρμες. Ο προγραμματιστής έχει την άνεση να εκδώσει το παιχνίδι του σε ότι συσκευή και λογισμικό επιθυμεί. Κάποιες από τις πιο γνωστές πλατφόρμες είναι οι εξής :

- Android
- iOS
- Windows
- WebGL
- Linux

- PlayStation 3, PlayStation 4
- Xbox 360, Xbox One
- Wii, Wii U

Επίσης αποτελεί το προκαθορισμένο λογισμικό πακέτο (Development Kit – SDK) για την κονσόλα “Wii U” της Nintendo. Παρέχεται επίσης για κάθε προγραμματιστή παιχνιδιών της κονσόλας μία ελεύθερη άδεια από την ίδια την Nintendo.



Εικόνα 3 - Το λογότυπο της Nintendo

2.4 Άλλες Χρήσεις

Πρόσφατα, μερικοί προγραμματιστές έχουν αρχίσει να συνειδητοποιούν ότι η Unity και γενικότερα οι μηχανές ανάπτυξης παιχνιδιών μπορούν να χρησιμοποιηθούν και γι' άλλους σκοπούς πέρα από την ανάπτυξη παιχνιδιών καθεαυτή. Για παράδειγμα μέσω αυτών αρχιτέκτονες μπορούν να δημιουργήσουν πρωτότυπες ιδέες, καλλιτέχνες να δημιουργήσουν διαδραστικές εγκαταστάσεις τέχνης ή ερευνητές να τις χρησιμοποιήσουν για οπτικοποίηση δεδομένων. Από μία δημοσκόπηση που έγινε σε χρήστες της Unity, προέκυψε ότι το ένα τρίτο αυτών δεν χρησιμοποιούσε το Unity για ανάπτυξη παιχνιδιών. Ο Francis δήλωσε σε μία συνέντευξή του πως το γεγονός αυτό σε συνδυασμό με την εύκολη χρήση και μάθηση της Unity, είναι και ο λόγος που έχει ακμάσει.

2.5 Παιχνίδια Φτιαγμένα με Unity

Όπως αναφέραμε και παραπάνω, το πρώτο παιχνίδι της Unity Technologies, ή Over the Edge Entertainment όπως ονομαζόταν η εταιρεία την περίοδο εκείνη, είναι το GooBall το οποίο δημοσιεύθηκε το 2005 από την Ambrosia Software και ήταν φτιαγμένο για Mac. Το παιχνίδι δεν γνώρισε την επιτυχία που ήθελαν οι δημιουργοί και αυτό οφειλόταν στο ότι ήταν πολύ δύσκολο να το παίξει κανείς όπως αναφέρει ο Joachim Ante σε μία συνέντευξή του.



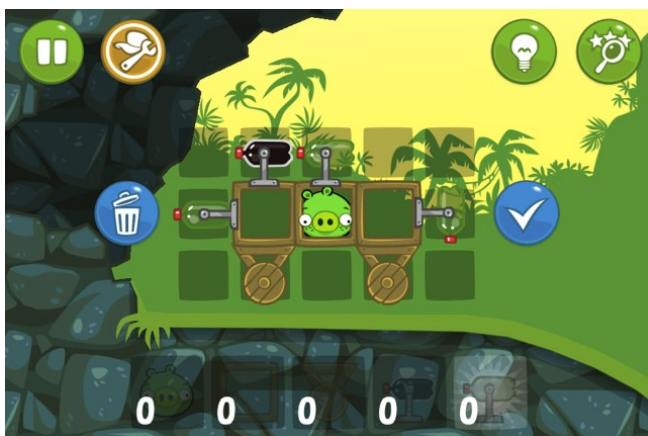
Εικόνα 4 - Το παιχνίδι GooBall (2005)

Μέχρι και το 2009 δημιουργήθηκαν άλλα 4 παιχνίδια με τη χρήση της Unity και από το 2010 η προτίμηση της μηχανής στους developer αυξήθηκε συστηματικά ώστε να δημιουργηθούν 5 νέα παιχνίδια την χρονιά αυτή. Το 2011 κυκλοφόρησαν ακόμα 11 παιχνίδια, φτιαγμένα με Unity και άλλα 21 το 2012. Παιχνίδια σταθμοί της τριετίας αυτής θεωρούνται τα εξής :

- **Battlestar Galactica Online (2011):** Massively multiplayer online game (MMO) βασισμένο στην τηλεοπτική σειρά “Battlestar Galactica”. Σχεδιάστηκε από την Bigpoint Games and Artplan και ξεπέρασε τους 2 εκατομμύρια εγγεγραμμένους χρήστες σε λιγότερο από 3 μήνες.
- **Temple Run (2011):** Endless running game, ένα από τα πιο γνωστά στο είδος του, το οποίο σχεδιάστηκε και κυκλοφόρησε από την Imangi Studios. Στην αρχή ήταν διαθέσιμο μόνο στο

App Store αλλά ένα χρόνο αργότερα κυκλοφόρησε και για Android. Το παιχνίδι ήταν μια εμπορική επιτυχία με πολλές θετικές κριτικές κάτι που οδήγησε σε συνεργασία με την Disney για το Temple Run: Treasure Hunters και Temple Run: Oz. Επίσης ενέπνευσε βιβλία δραστηριοτήτων, επιτραπέζια και έκανε την Warner Bros Studios να μιλάει για πιθανή προσαρμογή ταινιών.

- **Bad Piggies (2012):** Puzzle game που αναπτύχθηκε από την Rovio Entertainment και ήταν το πρώτο spin-off την εταιρείας Angry Birds. Κυκλοφόρησε σε iOS, Android, Windows και Mac το 2012, ενώ το 2013 κυκλοφόρησε για BlackBerry 10 και το 2014 για Windows Phone.
- **Escape Plan (2012)** Puzzle game που σχεδιάστηκε από τον Chris Millar, γνωστό για το προηγούμενο παιχνίδι του Far Princess, ενώ εκδόθηκε από την Fun Bits Interactive και προοριζόταν για PlayStation Vita. Το παιχνίδι φημίζεται για τα ευκρινή, ασπρόμαυρα οπτικά του, το χιουμοριστικό αλλά και φρικιαστικό του τόνο και τη διαισθητική χρήση των εισόδων του PlayStation Vita.



Εικόνα 5 - Το παιχνίδι Bad Piggies (2012)



Εικόνα 6 - Το παιχνίδι Escape Plan (2012)

Με την μεγάλη αναγνώριση και επιτυχία των παιχνιδιών αυτών αλλά και των υπολοίπων που δημιουργήθηκαν την περίοδο 2005 – 2012 η Unity γίνεται πλέον από τις πιο αναγνωρισμένες μηχανές στο χώρο. Από τη χρονιά 2013 μέχρι σήμερα η Unity έχει χρησιμοποιηθεί για την σχεδίαση και τον προγραμματισμό σε περισσότερα από 350 παιχνίδια. Κάποια από αυτά που αξίζει να σημειωθούν είναι :

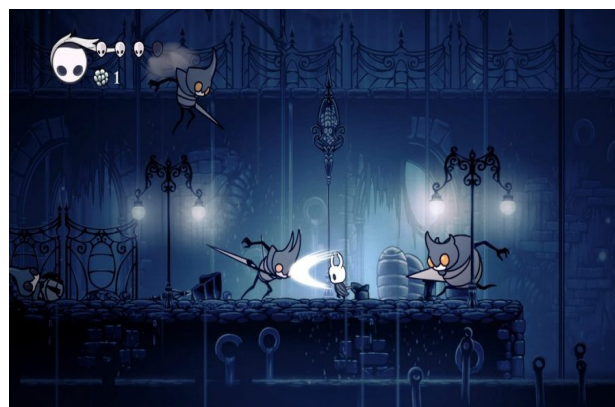
- **Device 6 (2013)** Text – based adventure game το οποίο δημιουργήθηκε από την εταιρεία Simogo και κυκλοφόρησε για iOS και τον Μάρτιο του 2014 είχε πουλήσει πάνω από 200.000 αντίτυπα. Το παιχνίδι έχει συγκεντρώσει βαθμολογία 92/100 στον ιστότοπο Metacritic και έχει κερδίσει πολλά βραβεία μερικά εκ των οποίων το “Excellence in Audio” το 2014 στο φεστιβάλ Independent Games, το “Best App Ever Award” από την Pocket Gamer και πολλά άλλα.
- **Max : The Curse of the Brotherhood (2013)** Puzzle game που αναπτύχθηκε από τη Press Play και κυκλοφόρησε για Xbox One και Xbox 360, Windows, Nintendo Switch και αργότερα για PlayStation 4. Το παιχνίδι έχει λάβει του πολύ καλές κριτικές για την εφευρετικότητα του, τον ισορροπημένο παζλ σχεδιασμό του, και τα πανέμορφα γραφικά του που προσφέρουν μία εξαιρετική εμπειρία στο χρήστη.
- **Ori and the Blind Forest (2015)** Platform – adventure παιχνίδι σχεδιασμένο από την Moon Studios και δημοσιευμένο από την Microsoft Studios. Έχει λάβει καθολική αναγνώριση και τρομερές κριτικές ακόμα και από τους πιο αυστηρούς κριτικούς καθώς φημίζεται για την ιστορία, τα γραφικά την μουσική αλλά και για τον περιβαλλοντικό του σχεδιασμό. Επίσης το παιχνίδι μετράει πολλαπλά βραβεία μεταξύ των οποίων το “Xbox Game of the Year”.
- **Firewatch (2016)** Adventure game το οποίο σχεδιάστηκε από την Campo Santo και δημοσιεύθηκε σε συνεργασία με την Panic. Είναι το πρώτο παιχνίδι της εταιρείας Campo Santo και κατάφερε να πετύχει περισσότερες από 500.000 πωλήσεις τον πρώτο μήνα και περισσότερες από 2.5 εκατομμύρια συνολικά σε όλες τις πλατφόρμες.
- **Pokemon Go (2016)** Ένα πολύ γνωστό παιχνίδι σε όλους καθώς είναι από τα πρώτα augmented reality (AR) mobile game. Σχεδιάστηκε από την Niantic σε συνεργασία με την The Pokemon

Company και κυκλοφόρησε για iOS και Android. Το παιχνίδι κατάφερε να σπάσει πολλά ρεκόρ λήψεων αφού μέσα στην πρώτη βδομάδα ξεπέρασε τις 10 εκατομμύρια λήψεις και μέχρι τις 12 Ιουλίου του 2012 έγινε το πιο ενεργό παιχνίδι για κινητά στις Ηνωμένες Πολιτείες με 21 εκατομμύρια ενεργούς χρήστες.

- **Hollow Knight (2017)** Action – adventure παιχνίδι το οποίο σχεδιάστηκε και δημοσιεύθηκε από την Team Cherry. Κυκλοφόρησε σε Windows, macOS το 2017 και για Nintendo Switch, PlayStation 4 και Xbox One το 2018. Απέσπασε πολύ καλές κριτικές και μέχρι τον Φεβρουάριο του 2019 ξεπέρασε τα 2,8 εκατομμύρια πωλήσεις.
- **Cuphead (2017)** Run and gun παιχνίδι σχεδιάστηκε και κυκλοφόρησε από την StudioMDHR. Κυκλοφόρησε για Windows, Xbox One, Nintendo Switch, macOS και αποτελεί ένα από τα πιο δημοφιλή παιχνίδια της Unity. Το παιχνίδι έχει συλλέξει πάνω από 25 βραβεία και συγκαταλέγεται σε ένα από τα καλύτερα παιχνίδια του 2017.
- **Getting Over It with Bennett Foddy (2017)** Platform video game το οποίο δημιουργήθηκε, σε αντίθεση από τα προηγούμενα, όχι από κάποια εταιρεία αλλά από έναν μόνο developer, τον Bennett Foddy. Έχει λάβει πολλές θετικές κριτικές για τον έξυπνο χειρισμό που διαθέτει καθώς και για την δυσκολία του. Έχει χαρακτηριστεί ως ένα από τα καλύτερα παιχνίδια του 2017 αλλά και ένα από τα πιο περίεργα. Τέλος το παιχνίδι έχει κατακτήσει το “Nuovo Award” το 2018.



Εικόνα 7 - Το παιχνίδι Pokemon Go (2016)



Εικόνα 8 - Το παιχνίδι Hollow Knight (2017)

ΚΕΦΑΛΑΙΟ 3 : ΔΟΜΗ ΚΑΙ ΕΡΓΑΛΕΙΑ

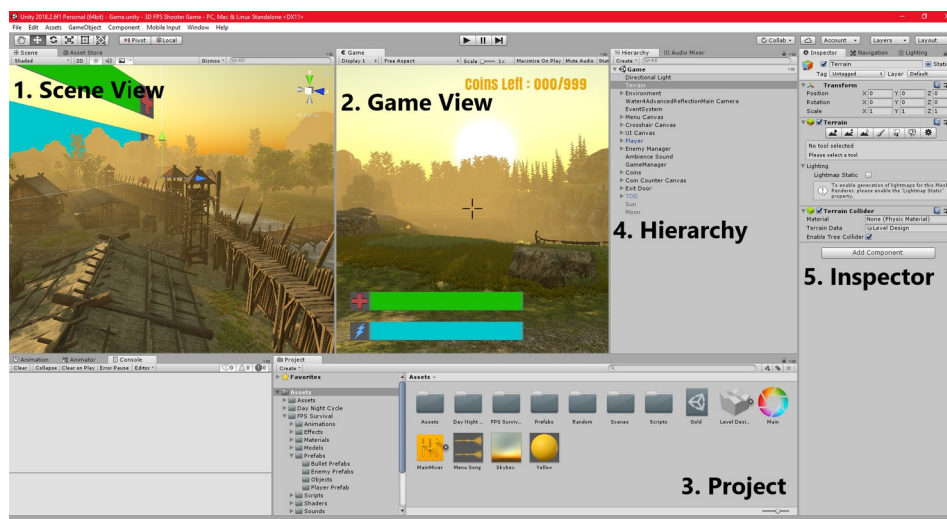
3.1 Unity Editor

Ο editor της Unity, αποτελεί το κύριο μέρος της μηχανής αφού σε αυτόν δημιουργούνται τα όλα τα project. Εδώ εισάγονται όλα τα στοιχεία (assets) του παιχνιδιού μας. Στον editor ουσιαστικά χτίζονται τα επίπεδα και τα σκηνικά του παιχνιδιού και επίσης εδώ είναι που πραγματοποιείται και η επίδραση των στοιχείων αυτών με τα scripts. Ας αναλύσουμε λοιπόν πως γίνονται όλα αυτά.

3.2 Παράθυρα του Editor

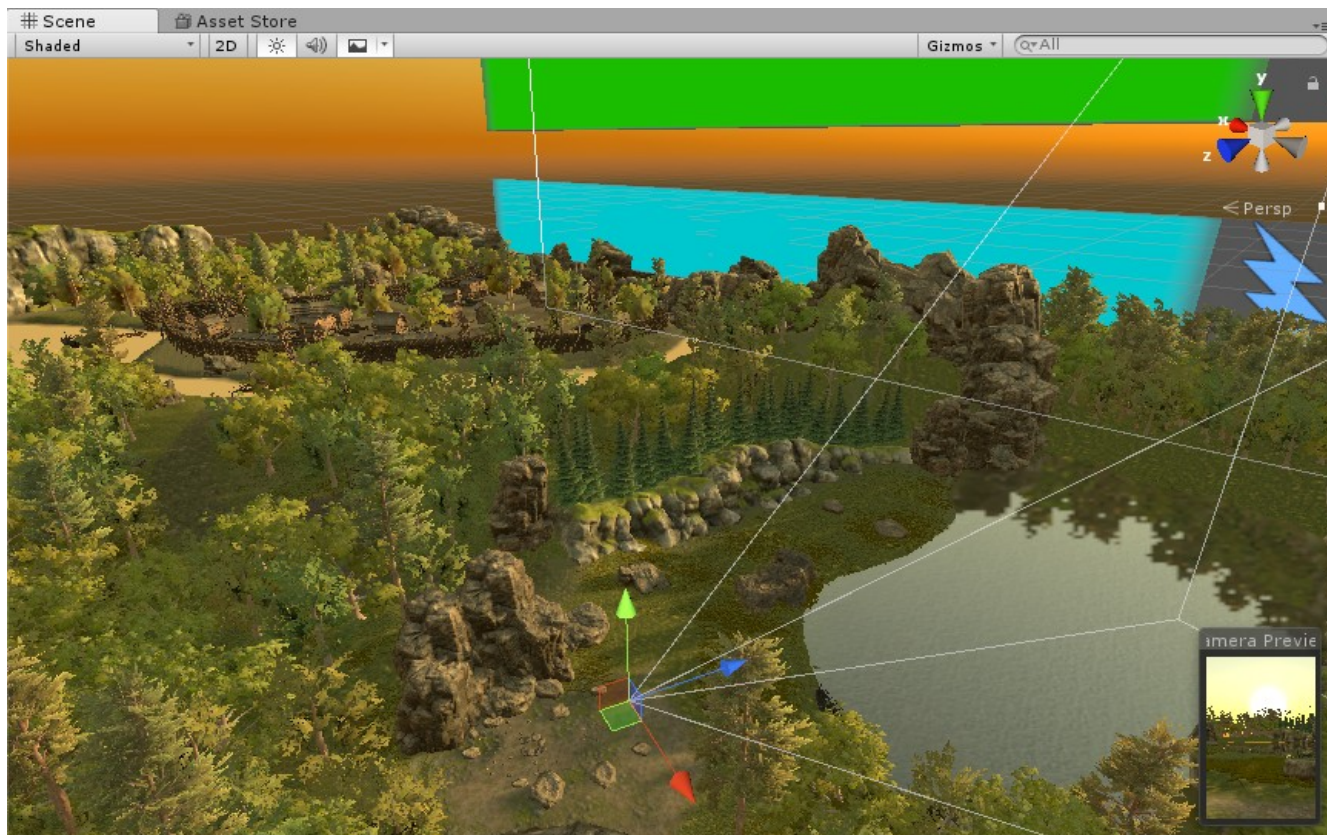
Ο editor αποτελείται από πολλά και διαφορετικά παράθυρα ή καλύτερα panel, τα οποία επιτρέπουν στον χρήστη να έχει μία οργανωμένη διαμόρφωση της κατασκευής των παιχνιδιών του. Τα πιο σημαντικά panel είναι τα εξής :

- Scene
- Game
- Project
- Hierarchy
- Inspector



Εικόνα 1 - Editor's interface

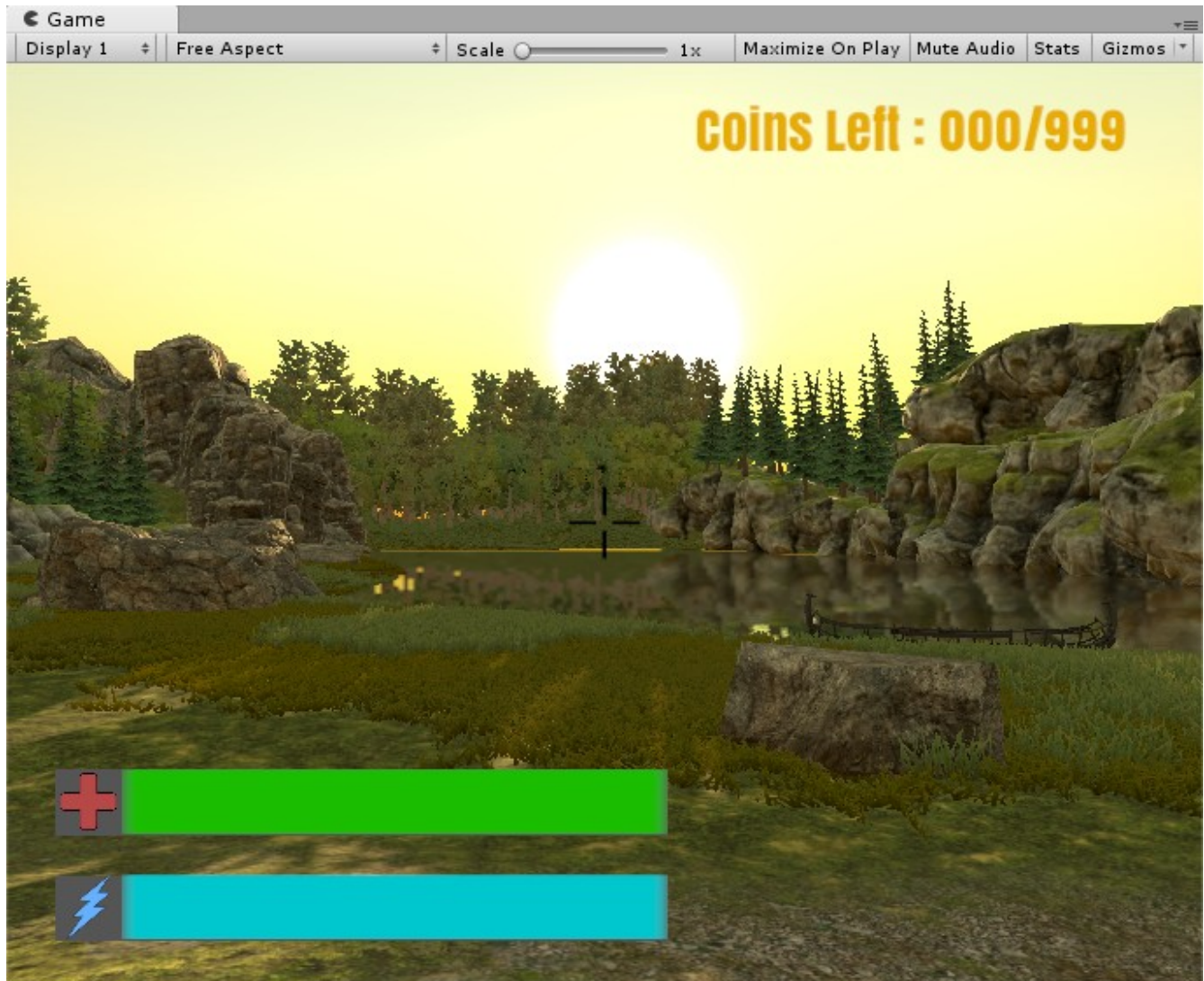
3.2.1 Scene View



Εικόνα 2 - Scene View

Το scene view panel είναι πιθανώς αυτό που χρησιμοποιεί ο προγραμματιστής πιο πολύ καθώς εδώ ουσιαστικά σχεδιάζεται όλο το παιχνίδι. Εδώ ο χρήστης μπορεί να εισάγει όλα τα αντικείμενα που επιθυμεί και να τα επεξεργαστεί με τα κατάλληλα εργαλεία ώστε να στήσει τη σκηνή του παιχνιδιού, η οποία μπορεί να είναι δισδιάστατη, αλλά και τρισδιάστατη. Ο χρήστης μπορεί να μεταβαίνει από το δισδιάστατο στο τρισδιάστατο περιβάλλον και αντίστροφα με το κουμπί “2D” το οποίο βρίσκεται συνήθως πάνω από το scene view. Επίσης έχει στη διάθεσή του εργαλεία για την επεξεργασία των αντικειμένων που χρησιμοποιεί, συγκεκριμένα το “Navigation Tool”, το οποίο χρησιμεύει στη πλοήγηση στο χώρο, το “Translate Tool” που χρησιμοποιείται για την μετακίνηση αντικειμένων στους 3 άξονες, το “Rotate Tool” που χρησιμεύει στην περιστροφή αντικειμένων και τέλος το “Scale Tool” για την ρύθμιση των διαστάσεων των αντικειμένων.

3.2.2 Game View

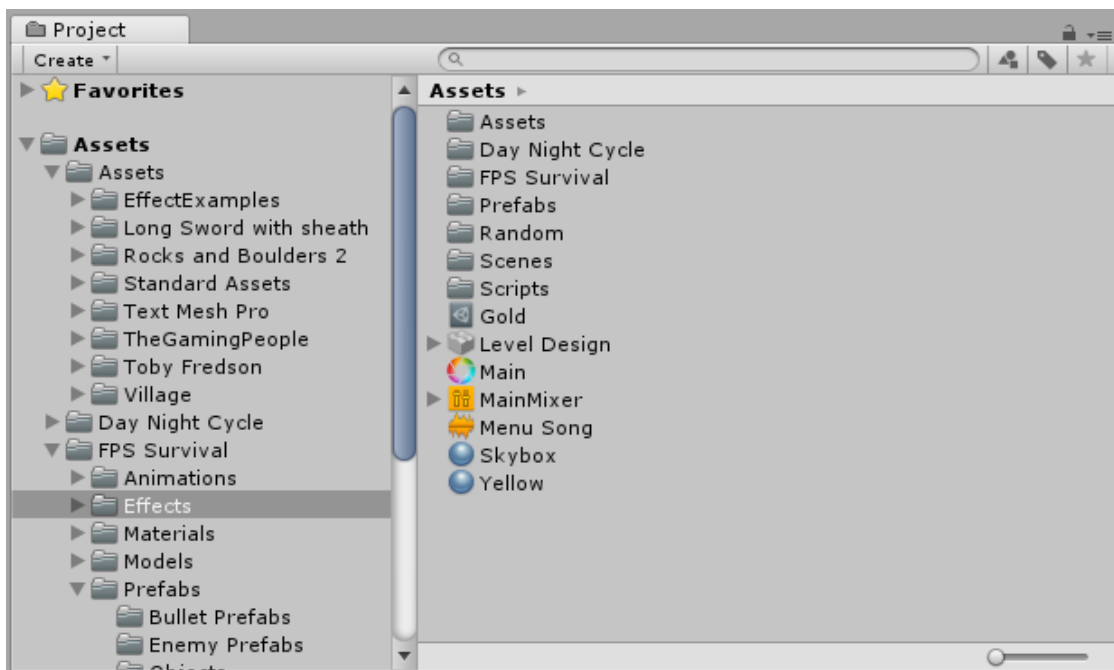


Εικόνα 3 - Game View

Το Game View θα μπορούσε να θεωρηθεί ως ένας προσομοιωτής των κονσολών για τις οποίες κυκλοφορούν τα παιχνίδια. Ο χρήστης πατώντας το κουμπί “Play” που βρίσκεται στο πάνω κεντρικό σημείο της μηχανής οδηγείται στο Game View όπου μπορεί να εξετάσει βήμα βήμα το παιχνίδι του και τις ρυθμίσεις του, τα γραφικά του, το πόσο καλά λειτουργούν τα script, ή οτιδήποτε επιθυμεί να τεστάρει στο παιχνίδι του την δεδομένη στιγμή.

3.2.3 Project

Στο παράθυρο αυτό τοποθετούνται όλα τα αρχεία που εισάγονται στη Unity και είναι διαθέσιμα για χρήση από τον προγραμματιστή.



Εικόνα 4 - Project

3.2.4 Hierarchy

Στο Hierarchy View περιέχονται όλα τα αντικείμενα που βρίσκονται στην τρέχουσα σκηνή. Αυτή η λίστα ενημερώνεται αυτόματα όταν ένα αντικείμενο μπαίνει στη σκηνή. Επίσης εδώ προγραμματιστής σέρνοντας τα αντικείμενα το ένα πάνω στο άλλο μπορεί να τα καταστήσει γονείς (parents) ή παιδιά (children).

3.2.5 Inspector

Τέλος, στο παράθυρο Inspector προβάλλονται οι λεπτομέρειες κάθε game object που μπορούν να τροποποιηθούν. Επίσης, εδώ εμφανίζονται κι όλα τα υπόλοιπα στοιχεία, τα οποία έχουν προστεθεί σε ένα game object, όπως για παράδειγμα τα scripts, colliders, sounds κ.α.

3.3 Γραμμή Μενού

Στην γραμμή μενού βρίσκονται όλες οι επιλογές που διαθέτει ο προγραμματιστής για το περιβάλλον ανάπτυξης. Οι καρτέλες που βρίσκονται στη γραμμή μενού είναι οι εξής :

- File
- Edit
- Assets
- GameObject
- Component
- Window
- Help

3.3.1 File

Στην καρτέλα File ξεκινάει όλο το project. Στην καρτέλα αυτή υπάρχει η επιλογή για δημιουργία καινούριου project ή δημιουργία σκηνής, οι επιλογές πλατφόρμας που θα κυκλοφορήσει το παιχνίδι, κάτι το οποίο θα αναλύσουμε παρακάτω, και επίσης η δυνατότητα να αποθηκευτεί το project οποιαδήποτε στιγμή το επιθυμεί ο χρήστης.

3.3.2 Edit

Η καρτέλα Edit προσφέρει στον χρήστη την δυνατότητα να αξιοποιεί κάποιες εντολές πληκτρολογίου, όπως για παράδειγμα Undo, Copy, Paste, Duplicate, Delete κ.α. Επίσης υπάρχουν επιλογές για τις προτιμήσεις του Project και του Editor.

3.3.3 Asset

Η καρτέλα Asset πέρα από τη γραμμή μενού βρίσκεται και στο παράθυρο Project. Σ' αυτή την καρτέλα παρέχεται η δυνατότητα εισαγωγής καινούριων αντικειμένων στο Project, καθώς επίσης και η εισαγωγή πακέτων με στοιχεία που παρέχονται άλλες φορές από την ίδια την Unity και άλλες φορές

από το Asset Store ή από διάφορες σελίδες στο διαδίκτυο. Τα πακέτα αυτά μπορεί να περιέχουν πολλά και διαφορετικά αντικείμενα ή στοιχεία, όπως για παράδειγμα δέντρα, σπίτια, φωτισμούς, λίμνες, πέτρες και πολλά άλλα.

3.3.4 GameObject

Η καρτέλα GameObject αποτελεί την κύρια βιβλιοθήκη που μπορεί να χρησιμοποιήσει ο προγραμματιστής για εισαγωγή αντικειμένων. Οι επιλογές που διαθέτει για εισαγωγή είναι :

- **Particle System** : Σύστημα σωματιδίων και χρησιμοποιείται για την σχεδίαση φυσικών φαινομένων όπως φωτιά, εκρήξεις, καπνός κλπ.
- **Camera** : Αντικείμενο ώστε να μπορεί να χρήστης να βλέπει τον εικονικό κόσμο.
- **UI** : Εδώ υπάρχουν στοιχεία όπως εικόνες, κείμενα, κλπ. τα οποία είναι για το σύστημα UI.
- **Light** : Στην ενότητα αυτή υπάρχουν διάφορα είδη φωτισμών που σκοπό έχουν να φωτίζουν ένα χώρο ή ένα σημείο ή ακόμα και ολόκληρο το επίπεδο του παιχνιδιού.
- **Cube / Sphere / Capsule / Cylinder / Plane / Quad** : Τα στοιχεία αυτά αποτελούν τρισδιάστατα αντικείμενα με τα οποία μπορεί να αλληλεπιδράσει ο χρήστης τοποθετώντας τα μέσα στις σκηνές του παιχνιδιού, αλλάζοντας τις διαστάσεις τους, το χρώμα τους, τις ιδιότητες τους και πολλά άλλα όπως ο ίδιος επιθυμεί.
- **Sprites** : Πρόκειται για δισδιάστατα (2D) αντικείμενα γραφικών που χρησιμοποιούνται για χαρακτήρες και άλλα αντικείμενα κυρίως δισδιάστατων παιχνιδιών.
- **Cloth** : Χρησιμοποιούνται για την εμφάνιση γραφικών πάνω σε αντικείμενα, όπως ρούχα, σημαίες κλπ.
- **Audio Reverb Zone** : Το αντικείμενο αυτό σχετίζεται με ήχους τους παιχνιδιού και προσφέρει τη δυνατότητα να υπάρχουν εφέ όπως ηχώ κλπ.
- **Terrain / Ragdoll / Tree / Wind Zone** : Αντικείμενα για εδάφη, μηχανισμούς κίνησης χαρακτήρων, δέντρα και ζώνες με αέρα για να αλληλεπιδρούν με τα υπόλοιπα αντικείμενα στη σκηνή.

3.3.5 Component

Η καρτέλα του Μενού “Component” περιέχει μια γκάμα στοιχείων τα οποία προστίθενται στα διάφορα αντικείμενα του παιχνιδιού. Τα στοιχεία αυτά, μαζί με τις ιδιότητές τους είναι τα εξής :

- **Mesh** : Είναι το πλέγμα που αποτελείται από πολλά τρίγωνα τοποθετημένα σε τρισδιάστατο χώρο και ρόλος τους είναι να δίνουν μια τρισδιάστατη στερεή όψη στα αντικείμενα που θέλουμε να χρησιμοποιήσουμε.
- **Effects** : Είναι τα εφέ που προστίθενται σε διάφορα αντικείμενα και μπορεί να είναι είτε συστήματα σωματιδίων (Particle Effects), είτε γραμμές που αφήνουν τα αντικείμενα κατά την κίνησή τους, για να δώσουν έναν πιο ρεαλιστικό τόνο στο παιχνίδι.
- **Physics / Physics 2D** : Ιδιότητες που παίρνει ένα αντικείμενο για να θεωρηθεί φυσικό σώμα και να μπορεί να κινείται και να αλληλεπιδρά με άλλα αντικείμενα του χώρου. Χωρίς αυτά οι χαρακτήρες του παιχνιδιού είναι στην ουσία άυλοι και δε μπορούν να υπάρξουν συγκρούσεις, καταστροφές και άλλα φαινόμενα στο παιχνίδι. Με τις ιδιότητες "colliders" ένα αντικείμενο μετατρέπεται σε υλικό για να μπορεί αυτό να συγκρούεται με άλλα αντικείμενα με την ίδια ιδιότητα. Η ιδιότητα “RigidBody” προσφέρει σε ένα αντικείμενο κίνηση μέσα στο χώρο, βαρύτητα και σύγκρουση με άλλα αντικείμενα. Σημειώνουμε ότι οι ιδιότητες της κατηγορίας “Physics 2D” σχετίζονται με δισδιάστατα στοιχεία του παιχνιδιού και μόνο.
- **Navigation** : Ο ρόλος του συστήματος αυτού είναι για να δείξει στον χρήστη ποια μέρη / χώροι στον κόσμο του παιχνιδιού είναι πιο προσβάσιμα καθώς κινείται ο κύριος χαρακτήρας του παιχνιδιού για να μην υπάρξουν προβλήματα γραφικών και κίνησης.
- **Audio** : Παρέχει τη δυνατότητα σε διάφορα αντικείμενα του παιχνιδιού να αναπαραγάγουν αρχεία μουσικής που εισάγουμε στο παιχνίδι.
- **Rendering** : Η επιλογή αυτή προσθέτει στοιχεία όπως camera, κείμενα και φώτα στο αντικείμενο που θέλουμε.
- **Miscellaneous** : Το κύριο μέρος αυτού του component είναι ότι παρέχει τη δυνατότητα στα αντικείμενα να αναπαραγάγουν animations που έχουν ήδη φτιαχτεί για οποιοδήποτε σκοπό.
- **Scripts** : Αυτή η επιλογή επιτρέπει την προσθήκη ενός script κώδικα σε κάποιο αντικείμενο με αποτέλεσμα αυτό να αποκτήσει κι άλλες ιδιότητες που δε μπορούμε να βάλουμε μέσω του Editor.

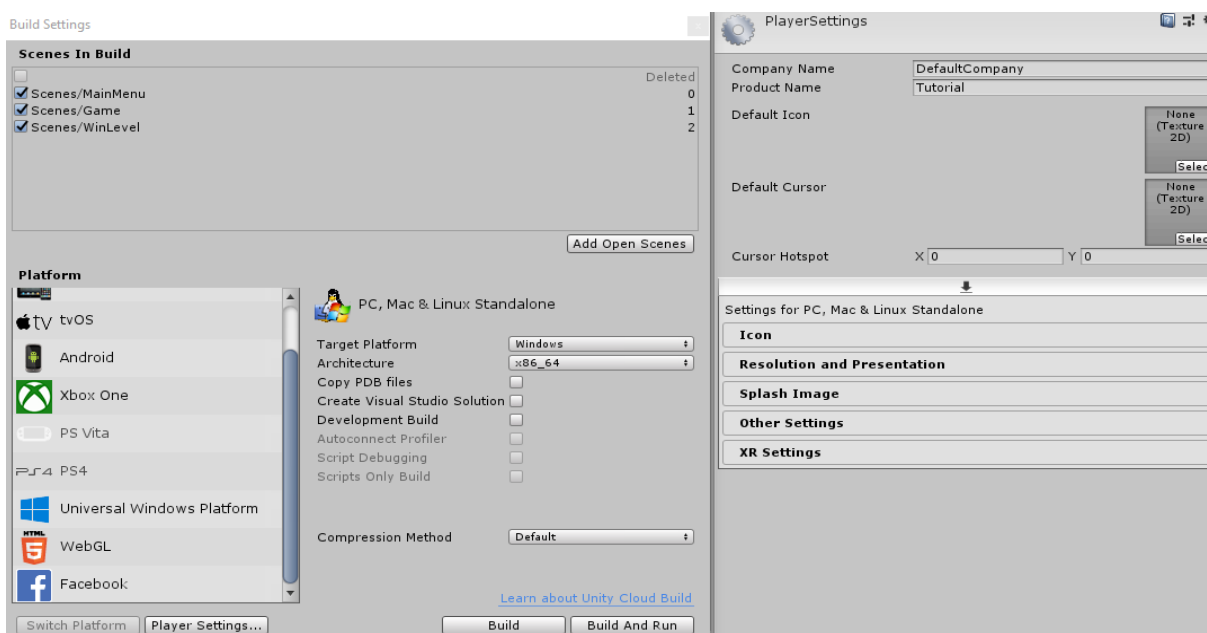
3.3.6 Window

Η καρτέλα Window περιλαμβάνει ως επιλογές όλα τα παράθυρα του Editor που αναφέραμε προηγουμένως όπως είναι το Hierarchy, Scene, Inspector, κ.α.

3.3.7 Build Settings

Όπως αναφέραμε και παραπάνω στην καρτέλα File υπάρχει επίσης η επιλογή Build Settings. Επιλέγεται μετά το τέλος της ανάπτυξης κάθε παιχνιδιού με σκοπό να γίνουν αλλαγές στις γενικές ρυθμίσεις και αποτελεί το τελευταίο βήμα πριν την κυκλοφορία του παιχνιδιού σε κάποια πλατφόρμα. Στο παράθυρο Build Setting υπάρχει ένας μικρός χώρος στον οποίο μπαίνουν οι σκηνές του παιχνιδιού είτε σέρνοντας με το ποντίκι τις σκηνές από το παράθυρο project σε αυτό, είτε πατώντας το κουμπί “Add Current” με τη σειρά που είναι επιθυμητή. Επίσης, εκεί ο developer μπορεί να επιλέξει ποια κονσόλα επιθυμεί να υιοθετήσει το παιχνίδι του. Οι επιλογές είναι: Pc, Mac & Linux Standalone, iOS, tvOS, Android, Xbox One, PS Vita, PS4, Universal Windows Platform, WebGL και Facebook.

Επίσης στην επιλογή Player Settings, ο developer έχει την δυνατότητα να προσθέσει ένα όνομα για το συγκεκριμένο παιχνίδι και την εταιρεία παραγωγής του καθώς και άλλες λεπτομέρειες ανάλογα με την επιλεγμένη πλατφόρμα και την έκδοση της Unity.



Εικόνα 5 - Build Settings

3.4 Asset Pipeline and Workflow

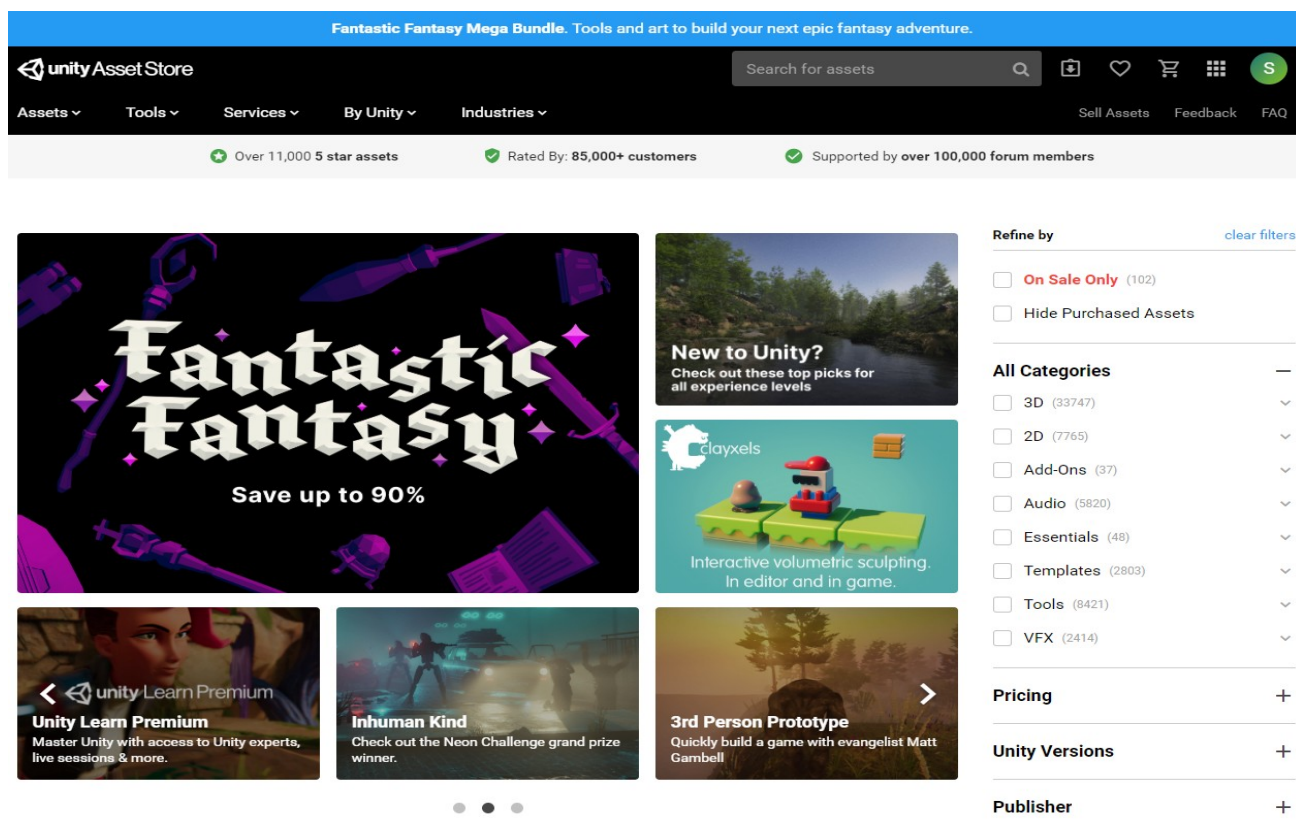
Ο αγωγός στοιχείων (asset pipeline) της Unity έφερε επανάσταση στον κλάδο του gaming από άποψη ευκολίας χρήσης και συμβατότητας με το τρέχον λογισμικό ανάπτυξης στοιχείων. Για να εισαγάγει ένα στοιχείο στο Unity, ο χρήστης απλά πρέπει να το σύρει στο παράθυρο του έργου και το Unity φροντίζει για τα υπόλοιπα. Κατά την επεξεργασία ενός στοιχείου, η Unity ενημερώνει αυτόματα το στοιχείο με τις πιο πρόσφατες τροποποιήσεις και το διαδίδει σε ολόκληρο το έργο.

«That's the real innovation behind Unity – I mean, the rest is really well done, but it's the workflow was just super-innovative. Nobody had figured out this concept of swallowing in whole assets from any application, using them, and updating them when they're saved. That was in super early, really early, and the product grew around the pipeline. It was in so early that we could import a multi-layer Photoshop file before we could even draw a mesh". - David Helgason

Το Unity υποστηρίζει την εισαγωγή στοιχείων από πολλές εφαρμογές, αλλά υποστηρίζει εγγενώς το Autodesk FBX και το COLLADA για μοντέλα με οστά ή κινούμενα σχέδια. Κατά την εισαγωγή των περισσότερων μοντέλων αυτού του τύπου, η Unity ανοίγει το αρχικό πρόγραμμα που έχει δημιουργήσει το μοντέλο και το εξάγει σε ένα αρχείο FBX που μπορεί να χρησιμοποιήσει η ίδια. Αυτό εκτελείται αυτόματα χωρίς να απαιτείται καμία άλλη ενέργεια από το χρήστη. Με αυτόν τον τρόπο εξοικονομείται ο χρόνος που άλλως θα χρειαζόταν οι προγραμματιστές για να μετατρέψουν τα στοιχεία στην σωστή μορφή. Εάν ο προγραμματιστής του παιχνιδιού χρειάζεται ακόμη πιο ακριβή έλεγχο στις ρυθμίσεις εισαγωγής, μπορεί να επεκτείνει τον editor ώστε να ταιριάζει στις ανάγκες του χρησιμοποιώντας το χώρο ονομάτων API UnityEditor. Η ροή εργασίας της Unity επιτρέπει επίσης στους προγραμματιστές, οι οποίοι στοχεύουν σε πολλές πλατφόρμες, να χρησιμοποιούν τα ίδια αρχεία προέλευσης επιτρέποντας την παράκαμψη των ρυθμίσεων εισαγωγής για κάθε στοχευμένη πλατφόρμα. Εάν ο προγραμματιστής εργάζεται σε ένα τεράστιο έργο με πολλά στοιχεία, ενδέχεται να έχει πρόβλημα να εντοπίσει το σωστό σε εύθετο χρόνο. Το παράθυρο του έργου επιτρέπει στο χρήστη να αναζητά και να εντοπίζει αντικείμενα πολύ αποτελεσματικά χρησιμοποιώντας πολλές παραμέτρους, συμπεριλαμβανομένων των καθορισμένων από τον χρήστη ετικετών.

3.5 Asset Store

Στις 10 Νοεμβρίου 2010, η Unity Technologies παρουσίασε μία third – party αγορά για τους προγραμματιστές η οποία ονομαζόταν Unity Asset Store. Η αγορά αυτή έδινε την δυνατότητα στους προγραμματιστές να αναζητούν και να αγοράζουν στοιχεία (assets) για την εφαρμογή τους και ήταν ενσωματωμένη στη μηχανή της Unity έτσι ώστε ο προγραμματιστής να μην χρειαστεί να εγκαταλείψει το έργο του όσο βρίσκεται στο Asset Store. Επίσης οι μεμονωμένοι προγραμματιστές μπορούσαν να δημιουργήσουν και να δημοσιεύσουν στοιχεία και πακέτα στο Asset Store κάτι το οποίο δημιούργησε στην κοινότητα της Unity μία εντελώς νέα πηγή εσόδων μέσα από τις δεξιότητες και τα talέντα τους. Η Unity Technologies κρατούσε το 30% των εσόδων από τις πωλήσεις των assets και ο σχεδιαστής το υπόλοιπο 70%. Επί του παρόντος υπάρχουν χιλιάδες assets διαθέσιμα για τους προγραμματιστές, πολλά εκ των οποίων δωρεάν, και ακόμα περισσότερα προστίθενται μέρα με τη μέρα.



Εικόνα 6 - Asset Store

3.6 Λήψη της Unity και Εκδόσεις

Η μηχανή της Unity είναι διαθέσιμη για κάποιον να την κατεβάσει από τον ιστόχωρο της Unity στην ενότητα “Get Started”, καθώς και να ενημερωθεί για τις απαιτήσεις συστήματος. Η Unity Technologies προσφέρει στο κοινό 2 εκδόσεις της μηχανής της. Η δωρεάν (Free) έκδοση είναι αυτή που χρησιμοποιήσαμε για την ανάπτυξη του παιχνιδιού μας και είναι επίσης αυτή που χρησιμοποιείται για εκμάθηση της πλατφόρμας και εξοικείωση των προγραμματιστών σε περιβάλλοντα 3D. Η άλλη έκδοση είναι η επαγγελματική, (Pro), η οποία είναι εύκολη και απλή στη χρήση όπως και η Free έκδοση, αλλά παρέχει μία πλήρη γκάμα επιπλέον δυνατοτήτων. Στην Pro έκδοση μπορούν να πραγματοποιηθούν πιο σύνθετα και εξελιγμένα παιχνίδια σε ό,τι αφορά τα γραφικά. Επίσης στην έκδοση αυτή αυξάνονται οι δυνατότητες και τα χαρακτηριστικά των παιχνιδιών που προορίζονται για κινητά τηλέφωνα. Τέλος στην Pro έκδοση είναι διαθέσιμα 3 διαφορετικά πλάνα πληρωμής ανάλογα με τις ανάγκες του κάθε πελάτη. Στις εικόνες 14 και 15, οι οποίες είναι από τον ιστόχωρο της Unity, φαίνονται τα 3 διαφορετικά πλάνα πληρωμής και οι διαφορές που υπάρχουν μεταξύ του κάθε πλάνου αλλά και της Free έκδοσης.

	Personal Free Get started	Plus \$40 /mo per seat Subscribe	Pro \$150 /mo per seat Subscribe	Enterprise \$200 /mo per seat Subscribe
① Financial eligibility	Eligible if revenue or funding is less than \$100K in the last 12 months	Eligible if revenue or funding is less than \$200K in the last 12 months	If revenue or funding is greater than \$200K in the last 12 months, you are required to use Pro or Enterprise	Minimum 20 seats. If revenue or funding is greater than \$200K in the last 12 months, you are required to use Pro or Enterprise
Create				
① Core Unity real-time development platform	✓	✓	✓	✓
① Dark theme UI	—	✓	✓	✓
① Splash screen customization	—	✓	✓	✓
① Integrations with collaboration tools	pick 1	✓	✓	✓
① Unity Teams Advanced (3 seats)	—	25GB storage with prepaid plan	✓	✓
① High-end art asset pack	—	—	✓	✓
① Source code access	—	—	+	+
① Industry-specific solution toolkits	—	—	—	+

Εικόνα 7 - Pro edition plans

Gaming Development with Unity 3D

Operate				
🔍 Advanced Cloud Diagnostics	—	✓	✓	✓
🔍 Core Analytics	—	✓	✓	✓
🔍 Analytics: Export 50GB/month of raw data	—	—	✓	✓
Monetize				
🔍 Unity Ads	✓	✓	✓	✓
🔍 In-App Purchase plugin	✓	✓	✓	✓
Support and learning				
🔍 Technical support	—	—	+	✓
🔍 Customer Success Manager	—	—	—	✓
🔍 Priority access to Unity Success Advisors	—	—	✓	✓
🔍 Priority queue for Customer Service	—	—	✓	✓
🔍 Premium learning resources	+	✓	✓	✓
🔍 Tailored learning plan	—	—	—	✓
🔍 Enterprise Learn Live sessions (4)	—	—	—	✓
🔍 Integrated Success Services	—	—	+	+
	Free	\$40 /mo per seat	\$150 /mo per seat	\$200 /mo per seat
	Get started	Subscribe	Subscribe	Subscribe
	Are you a student? Get the free Student plan			For large teams

Εικόνα 8 - Pro edition plans 2

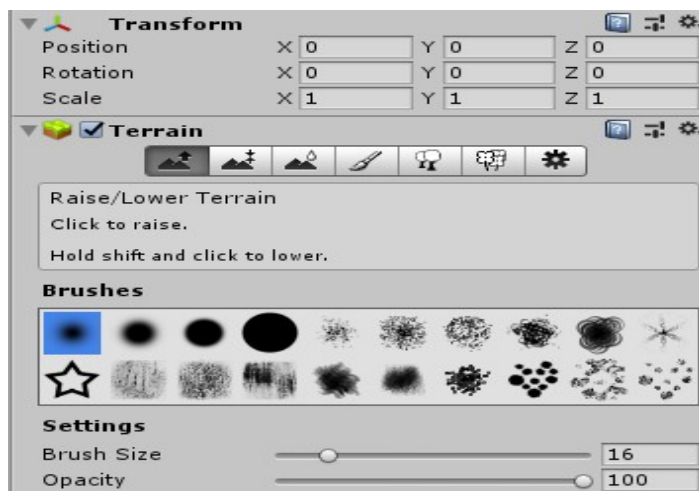
ΚΕΦΑΛΑΙΟ 4 : ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ

Το κεφάλαιο αυτό αποτελεί το πρακτικό μέρος της πτυχιακής εργασίας καθώς εδώ θα περιγραφεί η διαδικασία σχεδίασης και ανάπτυξης του παιχνιδιού με τη χρήση της Unity. Θα προσδιοριστεί ο σκοπός του παιχνιδιού, τα κριτήρια με τα οποία επιλέχθηκε το είδος του παιχνιδιού καθώς και οι παράμετροι που συντέλεσαν στον σχεδιασμό του. Στη συνέχεια θα γίνει η περιγραφή όλων των επιμέρους στοιχείων που απαρτίζουν το παιχνίδι, ενώ στο τέλος θα παρουσιαστούν κάποιες πληροφορίες σχετικές με την ανάπτυξη και την υλοποίηση του.

4.1 Level Design

Η πρώτη ενέργεια που έγινε ήταν η σχεδίαση του περιβάλλοντος του παιχνιδιού μας. Η Unity παρέχει πολλές δυνατότητες, που με απλές κινήσεις και εύκολα ο κάθε developer μπορεί να φτιάξει και να διακοσμήσει το δικό του περιβάλλον όπως αυτός επιθυμεί.

Ξεκινάμε λοιπόν με τη δημιουργία του terrain. Το terrain ουσιαστικά είναι το δάπεδο του παιχνιδιού, όπου με την βοήθεια διαφόρων εργαλείων που παρέχει η Unity μπορούμε να το προσαρμόσουμε όπως επιθυμούμε. Για παράδειγμα μπορούμε να αλλάξουμε το ύψους σε ορισμένα σημεία, ή να χρησιμοποιήσουμε διάφορα textures για προσθήκη γρασιδιού και δέντρων.



Εικόνα 1 - Terrain Editor



Εικόνα 2 - Τελικό Terrain

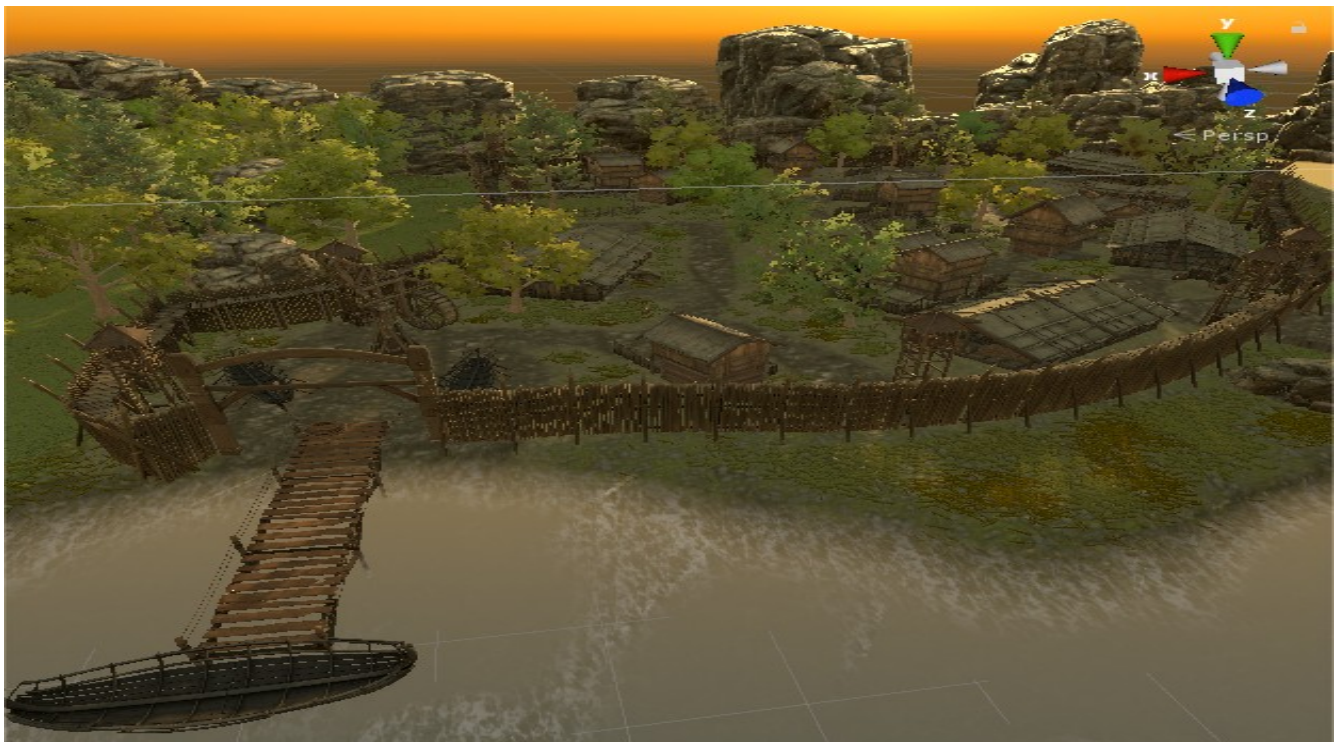
Στην συνέχεια αφού σχεδιάσαμε και και δημιουργήσαμε το terrain μας, με την βοήθεια κάποιων asset από το Asset Store, προσθέσαμε και τα πρώτα τρισδιάστατα αντικείμενα με σκοπό να το κάνουμε να φαίνεται πιο ελκυστικό και ρεαλιστικό προς τον χρήστη. Τα asset που χρησιμοποιήσαμε για τον σκοπό αυτόν ήταν τα εξής :

- Rocky Hills Environment
- Rock and Boulders 2
- Viking Village
- Long Sword with Sheath
- Shed, Tools, Bridge and Fences
- Standard Assets

Τα asset αυτά μας βοήθησαν να δημιουργήσουμε φυσικούς χώρους όπως για παράδειγμα βράχους, βουνά και λίμνες, αλλά και να σχεδιάσουμε και ένα χωριό στη μέση του terrain. Στις φωτογραφίες παρακάτω φαίνεται το τελικό αποτέλεσμα του περιβάλλοντος του παιχνιδιού.



Εικόνα 3 - Περιβάλλον παιχνιδιού



Εικόνα 4 - Χωριό

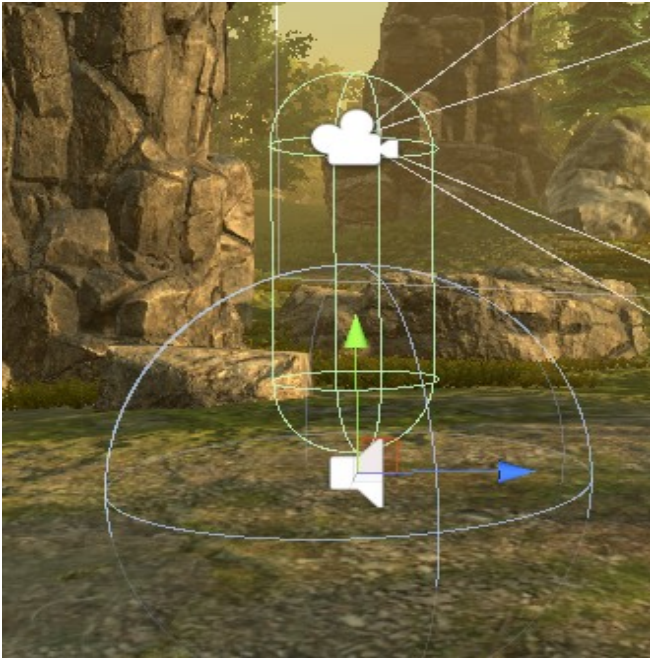
4.2 Creating FPS Player

Το αμέσως επόμενο βήμα ήταν η δημιουργία του χαρακτήρα του παιχνιδιού. Για να το πραγματοποιήσουμε αυτό δημιουργήσαμε στο hierarchy panel ένα empty object (άδειο αντικείμενο), το οποίο ονομάσαμε “Player” και έπειτα προσθέσαμε στο αντικείμενο αυτό το component “Character Controller”. Η λειτουργία του component αυτού είναι να μετακινεί το αντικείμενο σύμφωνα με το περιβάλλον. Δεν αποκρίνεται ούτε χρησιμοποιεί τη φυσική με οποιονδήποτε τρόπο. Η βασική ιδέα πίσω από το “Character Controller” είναι ότι παρέχει βασικές αποκρίσεις collider χωρίς φυσική. Επίσης δίνει την δυνατότητα στον προγραμματιστή να αλλάξει κάποια χαρακτηριστικά όπως :

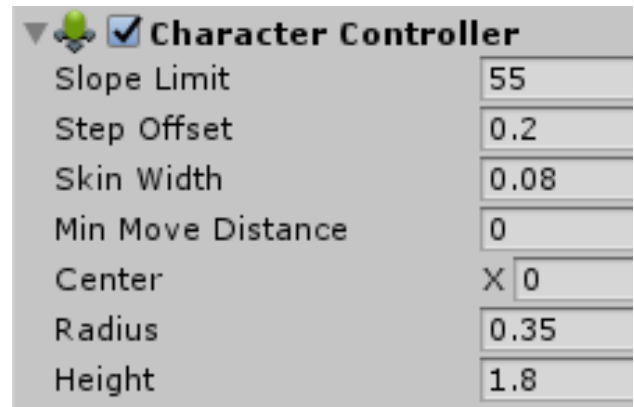
- **Slope Limit** (Οριοθετεί το collider για να ανεβαίνει μόνο πλαγιές που είναι λιγότερο απότομες (σε μοίρες) από την υποδεικνυόμενη τιμή).
- **Step Offset** (Επιτρέπει στον χαρακτήρα να ανεβεί μια σκάλα μόνο εάν είναι πιο κοντά στο έδαφος από την υποδεικνυόμενη τιμή. Αυτό δεν πρέπει να είναι μεγαλύτερο από το ύψος του ελεγκτή χαρακτήρων ή θα δημιουργήσει σφάλμα).
- **Minimum Move Distance** (Εάν ο χαρακτήρας προσπαθεί να κινηθεί κάτω από την υποδεικνυόμενη τιμή, δεν θα μετακινηθεί καθόλου. Αυτό μπορεί να χρησιμοποιηθεί για τη μείωση του jitter. Στις περισσότερες περιπτώσεις, αυτή η τιμή πρέπει να παραμείνει στο 0)
- **Radius** (Είναι το πλάτος του χαρακτήρα).
- **Height** (Το ύψος του χαρακτήρα. Η αλλαγή αυτή θα κλιμακώσει το collider κατά μήκος του άξονα Y τόσο σε θετικές όσο και σε αρνητικές κατευθύνσεις).

Στη συνέχεια δημιουργήσαμε ακόμα ένα empty object, μέσα στο αντικείμενο “Player” αυτή τη φορά, το οποίο μπορεί να χαρακτηριστεί και ως child του αντικειμένου “Player”, και το ονομάσαμε “Look Root”. Έπειτα ως child αυτού δημιουργήσαμε 2 κάμερες, την “Main Camera” και την “FP Camera”. Ο σκοπός της δεύτερης κάμερας είναι για να βλέπουμε τα όπλα μέσα από αυτή, τα οποία θα φτιάξουμε στην πορεία. Για να το πετύχουμε αυτό θα φτιάξαμε ένα καινούριο layer το οποίο ονομάσαμε “FirstPerson” (έπειτα θα το χρησιμοποιήσουμε ως layer και για τα όπλα) αλλάξαμε το

Culling Mask της (περιλαμβάνει ή παραλείπει επίπεδα αντικειμένων προς απόδοση από την κάμερα) σε “FirstPerson” και τέλος αλλάξαμε το “Clear Flags” σε Depth only (καθορίζει ποια μέρη της οθόνης θα διαγραφούν).



Εικόνα 6 - FPS Player



Εικόνα 5 - Character Controller

Τέλος δημιουργήσαμε ακόμα ένα child στο αντικείμενο “Player”, το οποίο ονομάσαμε “FP Weapons” μέσα στο οποίο θα τοποθετήσουμε όλα τα όπλα του παιχιδιού. Τα όπλα του παιχιδιού είναι τα εξής :

1. Axe
2. Revolver
3. Shotgun
4. Assault Rifle
5. Wooden Spear
6. Wooden Bow

4.3 Coding Player

4.3.1 Coding Player's Movement and Jump

Στη συνέχεια ξεκινήσαμε τον προγραμματισμό του χαρακτήρα. Για να το πετύχουμε αυτό δημιουργήσαμε ένα νέο C# script με όνομα “PlayerMovement” το οποίο προσθέσαμε στο αντικείμενο “Player”. Πριν ξεκινήσουμε όμως να δείχνουμε κομμάτια κώδικα από το παιχνίδι θα αναφερθούμε λίγο στα βασικά χαρακτηριστικά ενός script και συγκεκριμένο στις function του.

- **Awake** : Η συνάρτηση αυτή ξεκινάει όταν φορτώσει η σκηνή.
- **Start** : Η συνάρτηση αυτή ξεκινάει πριν ξεκινήσει να τρέχει το πρώτο frame.
- **Update** : Η βασικότερη συνάρτηση στο Unity αφού τρέχει κάθε frame.

Παρακάτω το script “PlayerMovement” :

```
public class PlayerMovement : MonoBehaviour
{
    private CharacterController character_Controller;

    private Vector3 move_Direction;

    public float speed = 5f;
    private float gravity = 20f;

    public float jump_Force = 10f;
    private float vertical_Velocity;

    private void Awake()
    {
        character_Controller = GetComponent<CharacterController>();
    }

    void Update ()
    {
        MoveThePlayer();
    }

    void MoveThePlayer()
    {
        move_Direction = new Vector3(Input.GetAxis(Axis.HORIZONTAL), 0f,
                                     Input.GetAxis(Axis.VERTICAL));
```

```
move_Direction = transform.TransformDirection(move_Direction);
move_Direction *= speed * Time.deltaTime;

ApplyGravity();
{
    if (character_Controller.isGrounded)
    {
        vertical_Velocity -= gravity * Time.deltaTime;

        //jump
        PlayerJump();
    }
    else
    {
        vertical_Velocity -= gravity * Time.deltaTime;
    }

    move_Direction.y = vertical_Velocity * Time.deltaTime;
}

character_Controller.Move(move_Direction);

} //Player Movement

void PlayerJump()
{
    if(character_Controller.isGrounded && Input.GetKeyDown(KeyCode.Space))
    {
        vertical_Velocity = jump_Force;
    }
}

void ApplyGravity()
{
}
}
```

Στην συνάρτηση `void MoveThePlayer()` ορίζουμε αρχικά έναν vector ώστε να γίνεται κάθε φορά το transform και ο παίχτης να κινείται αναλόγως και στη συνέχεια κάνουμε έλεγχο του gravity (βαρύτητας) ώστε να δούμε έναν ο παίχτης την δεδομένη χρονική στιγμή μπορεί να πηδήξει. Εάν βρίσκεται δηλαδή ή όχι στον αέρα. Τέλος γράφουμε τις συναρτήσεις `void PlayerJump()`, στην οποία ορίζουμε το Space ως το κουμπί για το άλμα του παίχτη, και την `void ApplyGravity()`.

4.3.2 Mouse Look and Mouse Controls

Ο παίχτης μας λοιπόν μπορεί να προχωρήσει μπροστά, πίσω, δεξιά και αριστερά μένει μόνο ακόμα να μπορεί και να κοιτάει δεξιά και αριστερά και όχι μόνο ευθεία. Για να πραγματοποιηθεί αυτό δημιουργήσαμε ένα ακόμα script, με όνομα “MouseLook” και το συνάψαμε στο αντικείμενο “Look Root”.

```
public class MouseLook : MonoBehaviour
{
    [SerializeField]
    private Transform playerRoot, lookRoot;

    [SerializeField]
    private bool invert;

    [SerializeField]
    private bool can_Unlock = true;

    [SerializeField]
    private float sensivity = 5f;

    [SerializeField]
    private int smooth_Steps = 10;

    [SerializeField]
    private float smooth_Weight = 0.4f;

    [SerializeField]
    private float roll_Angle = 10f;

    [SerializeField]
    private float roll_Speed = 3f;

    [SerializeField]
    private Vector2 default_Look_Limits = new Vector2(-70f, 80f);

    private Vector2 look_Angles;

    private Vector2 current_Mouse_Look;
    private Vector2 smooth_Move;

    private float current_Roll_Angle;

    private int last_Look_Frame;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }
}
```

```
void Update()
{
    LockAndUnlockCursor();

    if (Cursor.lockState == CursorLockMode.Locked)
    {
        LookAround();
    }
}

void LockAndUnlockCursor()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (Cursor.lockState == CursorLockMode.Locked)
        {
            Cursor.lockState = CursorLockMode.None;
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
        }
    }
} //Lock and Unlock

void LookAround()
{
    current_Mouse_Look = new Vector2(Input.GetAxis(MouseAxis.MOUSE_Y),
    Input.GetAxis(MouseAxis.MOUSE_X));

    look_Angles.x += current_Mouse_Look.x * sensitivity * (invert ? 1f : -1f);
    look_Angles.y += current_Mouse_Look.y * sensitivity;

    //look_Angles.y = look_Angles.y + current_Mouse_Look.y * sensitivity;

    look_Angles.x = Mathf.Clamp(look_Angles.x, default_Look_Limits.x,
    default_Look_Limits.y);

    current_Roll_Angle = Mathf.Lerp(current_Roll_Angle, Input.GetAxisRaw(MouseAxis.MOUSE_X)
    * roll_Angle, Time.deltaTime * roll_Speed);

    lookRoot.localRotation = Quaternion.Euler(look_Angles.x, 0f, current_Roll_Angle);
    playerRoot.localRotation = Quaternion.Euler(0f, look_Angles.y, 0f);
}
}
```

Στην αρχή του κώδικα αυτού χρησιμοποιούμε μια start function ώστε να κλειδώσουμε τον κέρσορα στην μέση της οθόνης. Έπειτα δημιουργούσε την συνάρτηση `void LockAndUnlockCursor()` μέσα στην οποία με τη χρήση του πλήκτρου Esc θα μπορούμε να κλειδώσουμε και να ξεκλειδώσουμε τον κέρσορα μας καθώς και να τον εμφανίζουμε και να τον εξαφανίζουμε.

Έπειτα φτιάχνουμε την συνάρτηση `void LookAround()` η οποία θα επιτρέπει στον παίχτη να μπορεί να κοιτάζει αριστερά και δεξιά χρησιμοποιώντας το ποντίκι. Η συνάρτηση βέβαια θα τρέχει στην update function μόνο εάν ο κέρσορας είναι κλειδωμένος όπως αναφέραμε και πιο πάνω.

Τι είναι το SerializeField;

Η χρήση του χαρακτηριστικού SerializeField κάνει το Unity να σειριοποιήσει οποιαδήποτε ιδιωτική μεταβλητή. Αυτό δεν ισχύει για στατικές μεταβλητές και ιδιότητες στο C#. Το SerializeField ουσιαστικά χρησιμοποιείται όταν χρειάζεται μια μεταβλητή να είναι ιδιωτική αλλά επίσης θέλουμε να εμφανίζεται στον Editor.

4.3.3 Sprint and Crouch

Τελευταίο βήμα για την ολοκλήρωση της κίνησης του παίχτη, μένει να προσθέσουμε την ικανότητα να τρέχει και να σκύβει. Για την υλοποίηση αυτού δημιουργήσαμε το script “PlayerSprintAndCrouch” το οποίο θα το προσθέσουμε και αυτό στο αντικείμενο “Player”. Επίσης ο αυτό το κομμάτι κώδικα είναι επίσης υπεύθυνο και για τον έλεγχο της ενέργειας του παίχτη (stamina), με το που η ενέργεια τελειώσει ο παίχτης δεν θα μπορεί να συνεχίσει να τρέχει μέχρις ότου η ενέργεια γεμίσει.

```
public class PlayerSprintAndCrouch : MonoBehaviour
{
    private PlayerMovement playerMovement;

    public float sprint_Speed = 10f;
    public float move_Speed = 5f;
    public float crouch_Speed = 2f;

    private Transform look_Root;
    private float stand_Height = 1.6f;
    private float crouch_Height = 1f;

    private bool is_Crouching;

    private PlayerFootsteps player_Footsteps;

    private float walk_Step_Distance = 0.4f;
    private float sprint_Step_Distance = 0.25f;
    private float crouch_Step_Distance = 0.5f;
```

```
private PlayerStats player_Stats;

private float sprint_Value = 100f;
public float sprint_Treshold = 10f;

void Awake ()
{
    playerMovement = GetComponent<PlayerMovement>();

    look_Root = transform.GetChild(0);

    player_Stats = GetComponent<PlayerStats>();
}

void Start()
{
}

void Update ()
{
    Sprint();
    Crouch();
}

void Sprint()
{
    //if we have stamina we can sprint
    if (sprint_Value > 0f)
    {
        if (Input.GetKeyDown(KeyCode.LeftShift) && !is_Crouching)
        {
            playerMovement.speed = sprint_Speed;
        }
    }

    if (Input.GetKeyUp(KeyCode.LeftShift) && !is_Crouching)
    {
        playerMovement.speed = move_Speed;
    }

    if(Input.GetKey(KeyCode.LeftShift) && !is_Crouching)
    {
        sprint_Value -= sprint_Treshold * Time.deltaTime;

        if(sprint_Value <= 0f)
        {
            sprint_Value = 0f;

            //reset the speed and sound
            playerMovement.speed = move_Speed;
        }

        player_Stats.Display_StaminaStats(sprint_Value);
    }
}
```

```
else
{
    if(sprint_Value != 100f)
    {
        sprint_Value += (sprint_Treshold / 2f) * Time.deltaTime;

        player_Stats.Display_StaminaStats(sprint_Value);

        if(sprint_Value > 100f)
        {
            sprint_Value = 100f;
        }
    }
}
} //sprint

void Crouch()
{
    if (Input.GetKeyDown(KeyCode.C))
    {
        if (is_Crouching) //if we are crouching then stand up
        {
            look_Root.localPosition = new Vector3(0f, stand_Height, 0f);
            playerMovement.speed = move_Speed;

            is_Crouching = false;
        }
        else //if we are not crouching then crouch
        {
            look_Root.localPosition = new Vector3(0f, crouch_Height, 0f);
            playerMovement.speed = crouch_Speed;

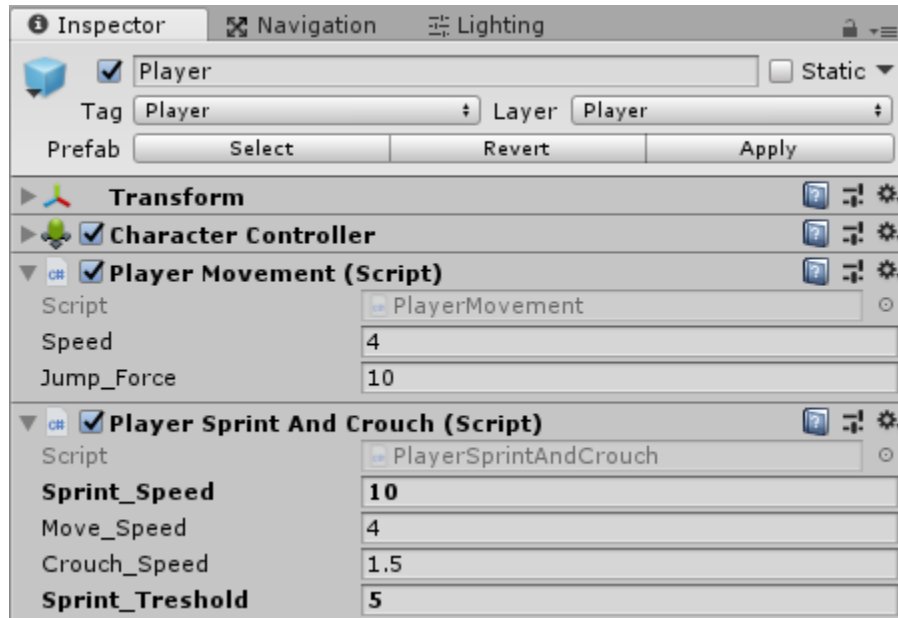
            is_Crouching = true;
        }
    }
} //if we press C

} //crouch

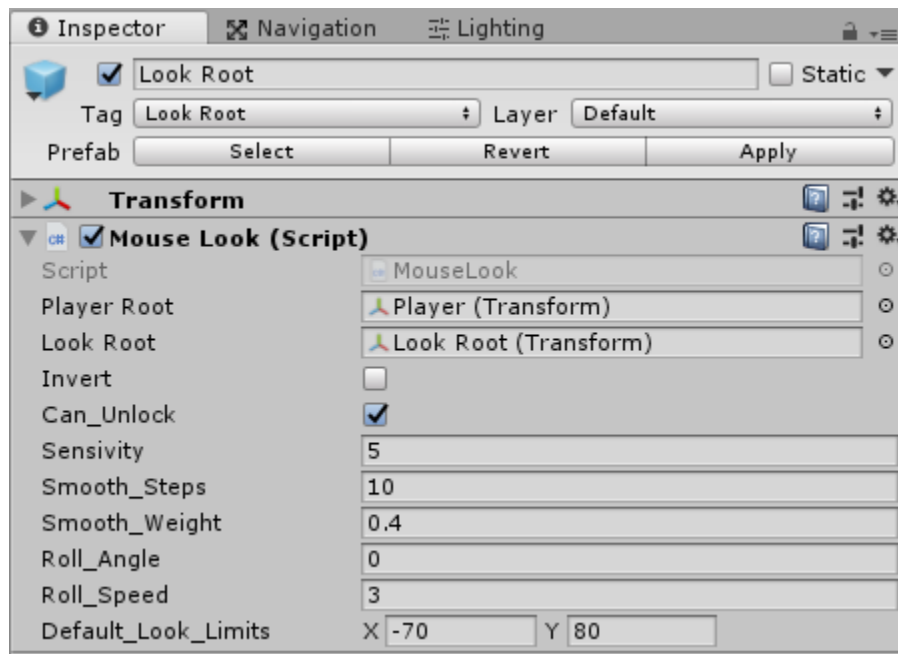
} //class
```

Οι βασικές συναρτήσεις του κώδικα αυτού είναι η `void Sprint()` και η `void Crouch()`. Στην `void Sprint()` με το πάτημα του αριστερού Shift, γίνεται έλεγχος αρχικά εάν η ενέργεια του παίχτη είναι μεγαλύτερη από 0. Εάν όχι ο παίχτης δεν μπορεί να τρέξει και η ενέργεια αυξάνεται με το χρόνο. Εάν ο παίχτης έχει ενέργεια τότε γίνεται έλεγχος για την κατάσταση του παίχτη εάν είναι σκυφτός ή όχι, αν είναι τότε εξακολουθεί να μην μπορεί να τρέξει αλλιώς η ταχύτητα του παίχτη αυξάνεται και η ενέργεια του μειώνεται.

Η `void Crouch()` είναι για το σκύψιμο του παίχτη. Με το πάτημα του πλήκτρου C μειώνεται το ύψος του παίχτη και η θέση της κάμερας καθώς και η ταχύτητα ελαττώνεται κάνοντας να φαίνεται ότι ο παίχτης είναι σκυμμένος.



Εικόνα 7 - Inspector του "Player" με τα script "Movement", "SprintAndCrouch"



Εικόνα 8 - Inspector του "LookRoot" με το script "MouseLook"

4.4 “WeaponHandler” και “Weapon Manager”

Έχοντας ολοκληρώσει τον χαρακτήρα του παιχνιδιού και μπορούμε πλέον να κινηθούμε μέσα στο χάρτη μας, το αμέσως επόμενο βήμα είναι η δημιουργία των όπλων που θα χειρίζεται ο χρήστης. Τα όπλα που θα διαθέτει ο χρήστης είναι 6 και τα έχουμε ήδη βάλει μέσα στη σκηνή μας, κάτι το οποίο αναφέρθηκε προηγουμένως. Μένει μόνο λοιπόν ο προγραμματισμός των όπλων αυτών, να είναι δηλαδή δυνατή η εναλλαγή των όπλων, το κάθε όπλο να διαθέτει δικό του ήχο και animation καθώς και να υπάρχει διαφορά στη ζημιά που τα όπλα αυτά κάνουν αλλά και ο τρόπος που γίνεται το reload των όπλων και ο τρόπος που πυροβολάνε. Πρώτο πράγμα που κάναμε για να πετύχουμε αυτό το αποτέλεσμα ήταν η δημιουργία ενός καινούριου script με όνομα “WeaponHandler”, το οποίο θα το προσθέσουμε και ως component σε κάθε όπλο ξεχωριστά.

```
public enum WeaponAim
{
    NONE,
    SELF_AIM,
    AIM,
}

public enum WeaponFireType
{
    SINGLE,
    MULTIPLE
}

public enum WeaponBulletType
{
    BULLET,
    ARROW,
    SPEAR,
    NONE
}

public class WeaponHandler : MonoBehaviour
{
    private Animator anim;

    public WeaponAim weapon_Aim;

    [SerializeField]
    private GameObject muzzleFlash;

    [SerializeField]
    private AudioSource shootSound, reload_Sound;

    public WeaponFireType fireType;
```

```
public WeaponBulletType bulletType;

public GameObject attack_Point;

void Awake()
{
    anim = GetComponent<Animator>();
}

public void ShootAnimation()
{
    anim.SetTrigger(AnimationTags.SHOOT_TRIGGER);
}

public void Aim(bool canAim)
{
    anim.SetBool(AnimationTags.AIM_PARAMETER, canAim);
}

void Turn_On_MuzzleFlash()
{
    muzzleFlash.SetActive(true);
}

void Turn_Off_MuzzleFlash()
{
    muzzleFlash.SetActive(false);
}

void Play_ShootSound()
{
    shootSound.Play();
}

void Play_ReloadSound()
{
    reload_Sound.Play();
}

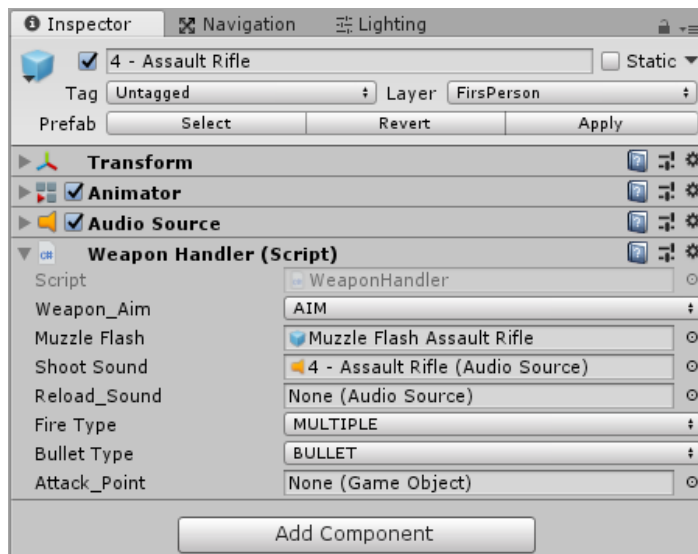
void Turn_On_AttackPoint()
{
    attack_Point.SetActive(true);
}

void Turn_Off_AttackPoint()
{
    if (attack_Point.activeInHierarchy)
    {
        attack_Point.SetActive(false);
    }
}
}
```

Ο παραπάνω κώδικας είναι υπεύθυνος για πολλές λειτουργίες μέσα στο παιχνίδι μας. Αρχικά με τις κλάσεις `public enum WeaponAim`, `public enum WeaponFireType`, `public enum WeaponBulletType` μπορούμε να ορίσουμε τον τρόπο με τον οποίο θα στοχεύουμε, το είδος του όπλου, καθώς και το είδος της σφαίρας που το κάθε όπλο θα παίρνει. Ακόμα ελέγχει τα animations των όπλων (`public void ShootAnimation()`), τους ήχους των όπλων (`void Play_ReloadSound()`, `void Play_ShootSound()`) και κάποια πράγματα ακόμη που θα εξηγήσουμε στη συνέχεια.



Εικόνα 9 - Το όπλο Assault Rifle



Εικόνα 10 - Το script “WeaponHandler” στο Assault Rifle

Αμέσως μετά δημιουργούμε ένα ακόμα script με όνομα “WeaponManager”, αυτή την φορά το προσθέτουμε στο αντικείμενο “Player” και όχι στο κάθε όπλο όπως κάναμε με το “WeaponHandler”. Ο κώδικας αυτός θα είναι υπεύθυνος για την εναλλαγή των όπλων. Ανάλογα λοιπόν με το ποιο κουμπί πατάει ο χρήστης, 1, 2, 3, 4, 5, ή 6, θα εμφανίζεται και το αντίστοιχο όπλο και θα εξαφανίζεται το προηγούμενο.

```
public class WeaponManager : MonoBehaviour
{
    [SerializeField]
    private WeaponHandler[] weapons;
    private int current_Weapon_Index;

    void Start ()
    {
        current_Weapon_Index = 0;
        weapons[current_Weapon_Index].gameObject.SetActive(true);
    }
}
```

```

void Update()
{
    if (Input.GetKeyDown(KeyCode.Alpha1))
    {
        TurnOnSelectedWeapon(0);
    }
    if (Input.GetKeyDown(KeyCode.Alpha2))
    {
        TurnOnSelectedWeapon(1);
    }
    if (Input.GetKeyDown(KeyCode.Alpha3))
    {
        TurnOnSelectedWeapon(2);
    }
    if (Input.GetKeyDown(KeyCode.Alpha4))
    {
        TurnOnSelectedWeapon(3);
    }
    if (Input.GetKeyDown(KeyCode.Alpha5))
    {
        TurnOnSelectedWeapon(4);
    }
    if (Input.GetKeyDown(KeyCode.Alpha6))
    {
        TurnOnSelectedWeapon(5);
    }
}
void TurnOnSelectedWeapon(int weaponIndex)
{
    //turn of the current weapon
    weapons[current_Weapon_Index].gameObject.SetActive(false);

    //turn on the selected weapon
    weapons[weaponIndex].gameObject.SetActive(true);

    //store the current selected weapon
    current_Weapon_Index = weaponIndex;
}
public WeaponHandler GetCurrentSelectedWeapon()
{
    return weapons[current_Weapon_Index];
}
}

```

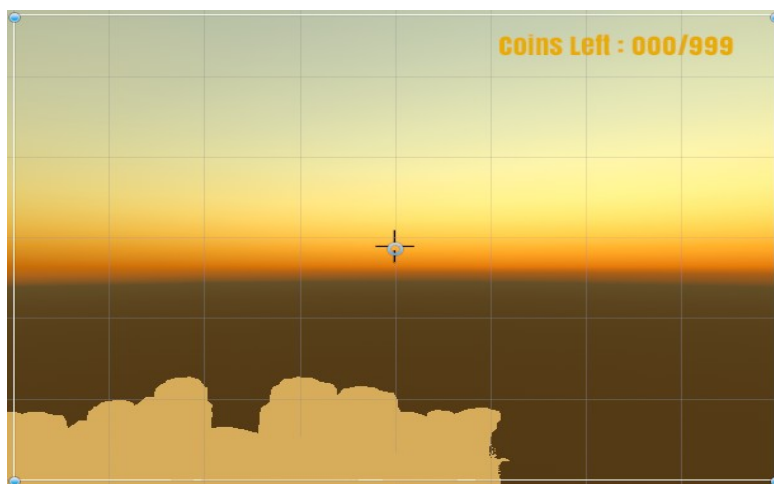
Tι είναι το enum (enumerator);

Ένα enum μπορεί, όπως και μια κλάση, να έχει χαρακτηριστικά και μεθόδους. Η μόνη διαφορά είναι ότι οι σταθερές enum είναι δημόσιες, στατικές και τελικές (αμετάβλητες - δεν μπορούν να παρακαμφθούν). Ένα enum δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία αντικειμένων και δεν μπορεί να επεκτείνει άλλες κλάσεις (αλλά μπορεί να εφαρμόσει διεπαφές).

4.5 Player Attack

Στη συνέχεια έχοντας τελειώσει και με τον προγραμματισμό των όπλων, δημιουργήσαμε ένα script με όνομα “PlayerAttack” και το προσθέσαμε και αυτό ως component στο αντικείμενο “Player”. Το script αυτό είναι υπεύθυνο για την εκφυροκρότηση των όπλων και είναι ίσως και το μεγαλύτερο που γράψαμε για το παιχνίδι αυτό.

Πριν όμως ξεκινήσουμε με το script φτιάξαμε ένα στόχο ώστε ο χρήστης να είναι σε θέση να καταλάβει που στοχεύει. Δημιουργήσαμε στο hierarchy λοιπόν ένα image το οποίο ονομάσαμε “Crosshair” και το τοποθετήσαμε στο κέντρο της οθόνης. Στη συνέχεια προσθέσαμε στο “Source Image” του αντικειμένου, μία εικόνα την οποία υλοποιήσαμε στο Photoshop και έπειτα προσθέσαμε στο Unity. Χρειάστηκε ακόμα να αλλάξαμε το “Texture Type” της εικόνας σε “Sprite (2D and UI).



Εικόνα 11 - Στόχος

```
public class PlayerAttack : MonoBehaviour
{
    private WeaponManager weapon_Manager;

    public float fireRate = 15f;
    private float nextTimeToFire;
    public float damage = 20f;

    private Animator zoomCameraAnim;
    private bool zoomed;

    private Camera mainCam;

    private GameObject crosshair;
```

```
private bool is_Aiming;

[SerializeField]
private GameObject arrow_Prefab, spear_Prefab;

[SerializeField]
private Transform arrow_Bow_StartPosition;

void Awake()
{
    weapon_Manager = GetComponent<WeaponManager>();

    zoomCameraAnim = transform.Find(Tags.LOOK_ROOT)
        .transform.Find(Tags.ZOOM_CAMERA).GetComponent<Animator>();

    crosshair = GameObject.FindWithTag(Tags.CROSSHAIR);

    mainCam = Camera.main;
}

void Update ()
{
    WeaponShoot();
    ZoomInAndOut();
}

void WeaponShoot()
{
    //assault rifle
    if(weapon_Manager.GetCurrentSelectedWeapon().fireType == WeaponFireType.MULTIPLE)
    {
        //if we press and hold left mouse click
        //and if Time is greater than the nextTimeToFire
        if(Input.GetMouseButton(0) && Time.time > nextTimeToFire)
        {
            nextTimeToFire = Time.time + 1f / fireRate;

            weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();

            BulletFired();
        }
    }

    //if regular weapon that shoots once
    else
    {
        if (Input.GetMouseButtonDown(0))
        {
            //handle axe
            if(weapon_Manager.GetCurrentSelectedWeapon().tag == Tags.AXE_TAG)
            {

```

```
        weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();
    }

    //handle shoot
    if(weapon_Manager.GetCurrentSelectedWeapon().bulletType ==
        WeaponBulletType.BULLET)
    {
        weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();

        BulletFired();
    }
    else
    {
        //we have arrow or spear
        if (is_Aiming)
        {
            weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();

            if (weapon_Manager.GetCurrentSelectedWeapon().bulletType ==
                WeaponBulletType.ARROW)
            {
                //throw arrow
                ThrowArrowOrSpear(true);
            } else if (weapon_Manager.GetCurrentSelectedWeapon().bulletType ==
                WeaponBulletType.SPEAR)
            {
                //throw spear
                ThrowArrowOrSpear(false);
            }
        }
    }
}

void ZoomInAndOut()
{
    //we are going to aim with our camera on the weapon
    if(weapon_Manager.GetCurrentSelectedWeapon().weapon_Aim == WeaponAim.AIM)
    {
        if (Input.GetMouseButtonDown(1))
        {
            zoomCameraAnim.Play(AnimationTags.ZOOM_IN_ANIM);

            crosshair.SetActive(false);
        }

        //when we release right mouse click
        if (Input.GetMouseButtonUp(1))
        {
            zoomCameraAnim.Play(AnimationTags.ZOOM_OUT_ANIM);

            crosshair.SetActive(true);
        }
    }
}
```

```
    }
} //if we need to zoom

if(weapon_Manager.GetCurrentSelectedWeapon().weapon_Aim == WeaponAim.SELF_AIM)
{
    if (Input.GetMouseButtonDown(1))
    {
        weapon_Manager.GetCurrentSelectedWeapon().Aim(true);

        is_Aiming = true;
    }

    if (Input.GetMouseButtonUp(1))
    {
        weapon_Manager.GetCurrentSelectedWeapon().Aim(false);

        is_Aiming = false;
    }

} //weapon with self aim
} //zoom in and out

void ThrowArrowOrSpear(bool throwArrow)
{
    if (throwArrow)
    {
        GameObject arrow = Instantiate(arrow_Prefab);
        arrow.transform.position = arrow_Bow_StartPosition.position;

        arrow.GetComponent<ArrowAndBowScript>().Launch(mainCam);
    }
    else
    {
        GameObject spear = Instantiate(spear_Prefab);
        spear.transform.position = arrow_Bow_StartPosition.position;

        spear.GetComponent<ArrowAndBowScript>().Launch(mainCam);
    }
}

void BulletFired()
{
    RaycastHit hit;

    if(Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out hit))
    {
        if(hit.transform.tag == Tags.ENEMY_TAG)
        {
            hit.transform.GetComponent<HealthScript>().ApplyDamage(damage);
        }
    }
}
}
```

Στον κώδικα αυτό όπως φαίνεται υπάρχουν 4 κλάσεις οι οποίες είναι :

1. `void` `WeaponShoot()` - Στην κλάση αυτή ελέγχεται ποιο όπλο χρησιμοποιεί ο χρήστης αν δηλαδή χρησιμοποιεί αυτόματο ή το τσεκούρι για παράδειγμα και καθορίζεται ο τρόπος που το όπλο αυτό θα λειτουργεί καθώς και η δύναμή του.
2. `void` `ZoomInAndOut()` - Με το πάτημα του δεξιού κλικ στο ποντίκι ο χρήστης έχει τη δυνατότητα να στοχεύσει μέσα από το όπλο κάτι το οποίο προσφέρει μεγαλύτερη ακρίβεια. Με το που ο χρήστης πατάει δεξί κλικ ο στόχος που έχουμε δημιουργήσει στο κέντρο της οθόνης εξαφανίζεται και το όπλο έρχεται πιο κοντά στην κάμερα.
3. `void` `BulletFired()` - Στην κλάση αυτή χρησιμοποιούμε Raycasting το οποίο είναι η διαδικασία λήψης μιας αόρατης ακτίνας από ένα σημείο, σε μια συγκεκριμένη κατεύθυνση για να ανιχνευθεί εάν υπάρχουν συγκρουόμενοι στο μονοπάτι της ακτίνας. Έτσι οι εχθροί που θα δημιουργήσουμε στη συνέχεια θα μπορούν να δεχθούν ζημιά από τα όπλα μας εάν βρίσκονται στην ευθεία της αόρατης ακτίνας.
4. `void` `ThrowArrowOrSpear()` - Θα μπορούσαμε να πούμε ότι είναι αντίστοιχη της `void` `BulletFired()` αλλά εδώ δεν χρησιμοποιούμε Raycasting. Οπότε για το `Spear` και το `Bow` χρησιμοποιούμε διαφορετικά πυρομαχικά τα οποία είναι `Spear` και `Arrow`. Στην κλάση αυτή επίσης καλείται το script “`ArrowAndBow`” το οποίο θα αναλύσουμε παρακάτω.



Εικόνα 12 - Revolver χωρίς zoom



Εικόνα 13 - Revolver με zoom

4.5.1 “ArrowAndBow” Script

```
public class ArrowAndBowScript : MonoBehaviour
{
    private Rigidbody myBody;

    public float speed = 30f;

    public float deactivate_Timer = 3f;

    public float damage = 30f;

    void Awake()
    {
        myBody = GetComponent<Rigidbody>();
    }

    void Start ()
    {
        Invoke("DeactivateGameObject", deactivate_Timer);
    }

    public void Launch(Camera mainCamera)
    {
        myBody.velocity = mainCamera.transform.forward * speed;

        transform.LookAt(transform.position + myBody.velocity);
    }

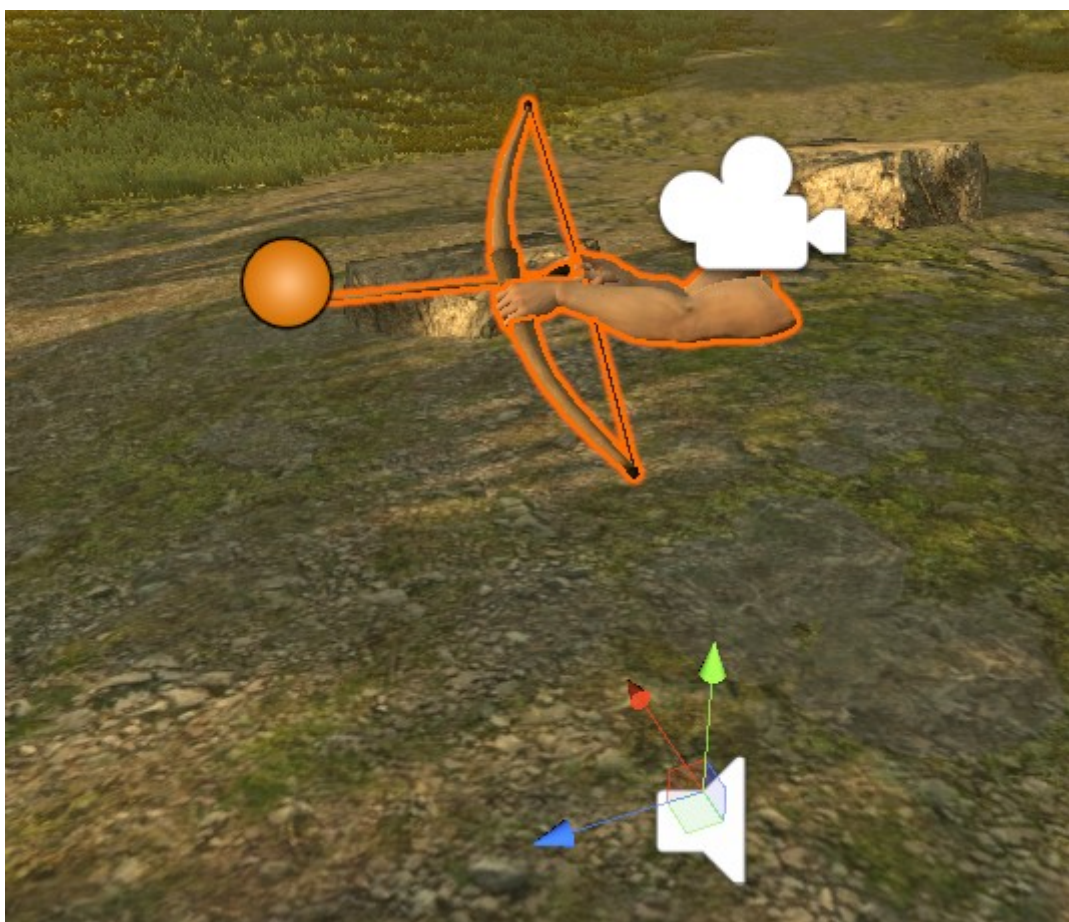
    void DeactivateGameObject()
    {
        if (gameObject.activeInHierarchy)
        {
            gameObject.SetActive(false);
        }
    }

    private void OnTriggerEnter(Collider target)
    {
        //after we touch enemy deactivate game object
        if(target.tag == Tags.ENEMY_TAG)
        {
            target.GetComponent<HealthScript>().ApplyDamage(damage);

            gameObject.SetActive(false);
        }
    }
}
```

Στο script αυτό ορίζουμε την δύναμη που θα έχει το arrow (βέλος) και το spear (ακόντιο) καθώς επίσης το πόσο γρήγορα ταξιδεύουν και το χρόνο μέχρι να εξαφανιστούνε. Για αρχή η `public void Launch(Camera mainCamera)` είναι η κλάση η οποία πυροβολεί το αντικείμενο. Στην κλάση `private void OnTriggerEnter(Collider target)` γίνεται έλεγχος αν κάποιο από τα δύο αυτά αντικείμενα έρθει σε επαφή με εχθρό. Εάν ναι ο εχθρός χάνει ζωή και το αντικείμενο εξαφανίζεται. Έχουμε ακόμα μία κλάση την `void DeactivateGameObject()` η οποία διαγράφει τα arrow και spear που δημιουργούνται στο hierarchy με κάθε επίθεση του χρήστη.

Ακόμη δημιουργούμε μπροστά στα όπλα Wooden Spear και Wooden Bow ένα empty object το οποίο ονομάζουμε “Arrow Bow Start Position” το οποίο αποτελεί το σημείο στο οποίο θα δημιουργούνται τα βέλη και τα ακόντια.



Εικόνα 14 - Arrow και Spear spawning point

4.6 Creating Enemies

Για την δημιουργία των εχθρών χρησιμοποιήσαμε δύο έτοιμα μοντέλα που βρήκαμε από το Asset Store. Αφού τα τοποθετήσαμε στη σκηνή μας προσθέσαμε στα αντικείμενα αυτά ως components “Box Collider”, “Nav Mesh Agent” και 2 script που γράψαμε στη συνέχεια. Πιο αναλυτικά τι είναι τα component που προσθέσαμε :

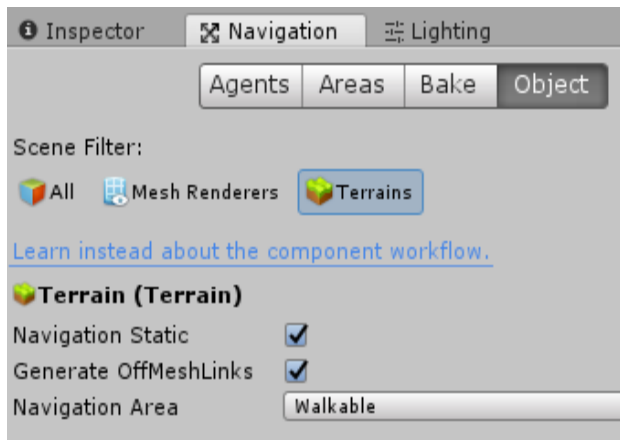
- **Box Collider** : - Το Box Collider μπορεί να αλλάζει το μέγεθος σε διάφορα σχήματα ορθογώνιων πρισμάτων. Οι Colliders συνεργάζονται με τα Rigidbodies για να φέρουν στη ζωή τη φυσική στην Unity. Ενώ τα Rigidbodies επιτρέπουν στα αντικείμενα να ελέγχονται από τη φυσική, τα Colliders επιτρέπουν στα αντικείμενα να συγκρούονται μεταξύ τους.
- **Nav Mesh Agent** : Τα στοιχεία NavMeshAgent βοηθούν να δημιουργήσουμε χαρακτήρες που αποφεύγουν ο ένας τον άλλον ενώ κινούνται προς τον στόχο τους. Οι πράκτορες έχουν λόγο για τον κόσμο του παιχνιδιού χρησιμοποιώντας το NavMesh.

NavMesh : Το NavMesh είναι μια τάξη που μπορεί να χρησιμοποιηθεί για την πραγματοποίηση χωρικών ερωτημάτων, όπως δοκιμές pathfinding και walkability, για τον καθορισμό του κόστους εύρεσης διαδρομών για συγκεκριμένους τύπους περιοχών και για την προσαρμογή της παγκόσμιας συμπεριφοράς του pathfinding και της αποφυγής.



Εικόνα 15 - Εχθροί "Cannibal" και "Boar"

Έπειτα στην καρτέλα “Navigation” ορίσαμε τη περιοχή που ο εχθρός θα αναγνωρίζει και θα μπορεί επάνω σε αυτή να κινηθεί.



Εικόνα 17 - Καρτέλα "Navigation"



Εικόνα 16 - Walkable terrain

Αφού ολοκληρώσαμε και αυτό το κομμάτι πρέπει τώρα να προγραμματίσουμε τους εχθρούς μας ώστε να κινούνται και να επιτίθενται. Το πρώτο script αφορά τα animation των εχθρών και ονομάζεται “EnemyAnimator”.

```
public class EnemyAnimator : MonoBehaviour
{
    private Animator anim;

    void Awake()
    {
        anim = GetComponent<Animator>();
    }

    public void Walk(bool walk)
    {
        anim.SetBool(AnimationTags.WALK_PARAMETER, walk);
    }

    public void Run(bool run)
    {
        anim.SetBool(AnimationTags.RUN_PARAMETER, run);
    }

    public void Attack()
    {
        anim.SetTrigger(AnimationTags.ATTACK_TRIGGER);
    }
}
```

```
public void Dead()
{
    anim.SetTrigger(AnimationTags.DEAD_TRIGGER);
}
}
```

Όπως φαίνεται αποτελείται από 4 κλάσεις μία για την κάθε κίνηση του. Η `public void Walk(bool walk)` ελέγχει το περπάτημα του εχθρού, η `public void Run(bool run)` το τρέξιμο του, η `public void Attack()` το πως επιτίθεται και τέλος η `public void Dead()` που πρόκειται για τον θάνατο του.

Το επόμενο script, “EnemyController”, είναι και αυτό ένα από τα μεγαλύτερα script που γράψαμε για το παιχνίδι και η λειτουργία του είναι να ελέγχει το ΑΙ του αντιπάλου και τον κάνει να κινείται στο χώρο καθώς και να επιτίθεται στον παίχτη.

```
public enum EnemyState
{
    PATROL,
    CHASE,
    ATTACK
}

public class EnemyController : MonoBehaviour
{
    private EnemyAnimator enemy_Anim;
    private NavMeshAgent navAgent;

    private EnemyState enemy_State;

    public float walk_Speed = 0.5f;
    public float run_Speed = 4f;

    public float chase_Distance = 7f;
    private float current_Chase_Distance;
    public float attack_Distance = 1.8f;
    public float chase_After_Attack_Distance = 2f;

    public float patrol_Radius_Min = 20f, patrol_Radius_Max = 60f;
    public float patrol_For_This_Time = 15f;
    private float patrol_Timer;

    public float wait_Before_Attack = 2f;
    private float attack_Timer;

    private Transform target;

    public GameObject attack_Point;

    private EnemyAudio enemy_Audio;
```

```
void Awake()
{
    enemy_Anim = GetComponent<EnemyAnimator>();
    navAgent = GetComponent<NavMeshAgent>();

    target = GameObject.FindWithTag(Tags.PLAYER_TAG).transform;

    enemy_Audio = GetComponentInChildren<EnemyAudio>();
}

void Start ()
{
    enemy_State = EnemyState.PATROL;

    patrol_Timer = patrol_For_This_Time;

    //when the enemy first gets to the player attack right away
    attack_Timer = wait_Before_Attack;

    //memorize the value of chase distance so we can put it back
    current_Chase_Distance = chase_Distance;
}

void Update ()
{
    if(enemy_State == EnemyState.PATROL)
    {
        Patrol();
    }
    if (enemy_State == EnemyState.CHASE)
    {
        Chase();
    }
    if (enemy_State == EnemyState.ATTACK)
    {
        Attack();
    }
}

void Patrol()
{
    //tell nav agent that he can move
    navAgent.isStopped = false;
    navAgent.speed = walk_Speed;

    patrol_Timer += Time.deltaTime;

    if(patrol_Timer > patrol_For_This_Time)
    {
        SetNewRandomDestination();

        patrol_Timer = 0f;
    }
}
```

```
if(navAgent.velocity.sqrMagnitude > 0)
{
    enemy_Anim.Walk(true);
}
else
{
    enemy_Anim.Walk(false);
}

//test the distance between the player and the enemy
if(Vector3.Distance(transform.position, target.position) <= chase_Distance)
{
    enemy_Anim.Walk(false);

    enemy_State = EnemyState.CHASE;

    //play spotted audio
    enemy_Audio.Play_ScreamSound();
}
}

void Chase()
{
    //enable the agent to move again
    navAgent.isStopped = false;
    navAgent.speed = run_Speed;

    //set the player's position as the desination bec we're chasing the player (running
    towards)
    navAgent.SetDestination(target.position);

    if (navAgent.velocity.sqrMagnitude > 0)
    {
        enemy_Anim.Run(true);
    }
    else
    {
        enemy_Anim.Run(false);
    }

    //if the distance between enemy and player is less than attack distance
    if(Vector3.Distance(transform.position, target.position) <= attack_Distance)
    {
        //stop the animations
        enemy_Anim.Run(false);
        enemy_Anim.Walk(false);
        enemy_State = EnemyState.ATTACK;

        //reset the chase distance to previous
        if(chase_Distance != current_Chase_Distance)
        {
            chase_Distance = current_Chase_Distance;
        }
    }
}
```

```
else if(Vector3.Distance(transform.position, target.position) > chase_Distance)
{
    //player run away from enemy

    //stop running
    enemy_Anim.Run(false);

    enemy_State = EnemyState.PATROL;

    //reset the patrol timer so that the fuction can calculate the new patrol
    destination right away
    patrol_Timer = patrol_For_This_Time;

    if(chase_Distance != current_Chase_Distance)
    {
        if (chase_Distance != current_Chase_Distance)
        {
            chase_Distance = current_Chase_Distance;
        }
    }
}

void Attack()
{
    navAgent.velocity = Vector3.zero;
    navAgent.isStopped = true;

    attack_Timer += Time.deltaTime;

    if(attack_Timer > wait_Before_Attack)
    {
        enemy_Anim.Attack();

        attack_Timer = 0f;

        //play attack sound
        enemy_Audio.Play_AttackSound();
    }

    if(Vector3.Distance(transform.position, target.position) > attack_Distance +
        chase_After_Attack_Distance)
    {
        enemy_State = EnemyState.CHASE;
    }
}

void SetNewRandomDestination()
{
    float rand_Radius = Random.Range(patrol_Radius_Min, patrol_Radius_Max);

    Vector3 randDir = Random.insideUnitSphere * rand_Radius;
    randDir += transform.position;

    NavMeshHit navHit;
```

```
NavMesh.SamplePosition(randDir, out navHit, rand_Radius, -1);
navAgent.SetDestination(navHit.position);
}

void Turn_On_AttackPoint()
{
    attack_Point.SetActive(true);
}

void Turn_Off_AttackPoint()
{
    if (attack_Point.activeInHierarchy)
    {
        attack_Point.SetActive(false);
    }
}

public EnemyState Enemy_State
{
    get; set;
}
}
```

Πιο αναλυτικά οι βασικές κλάσεις του script αυτού είναι οι εξής :

1. **void Patrol()** - Η κλάση αυτή είναι υπεύθυνη για την κίνηση του εχθρού στο χώρο. Ορίζεται η διαδρομή που θα ακολουθεί και ανανεώνεται ανά τακτά χρονικά διαστήματα έτσι ώστε να μην κινείται πάνω σε ευθεία συνέχεια. Ακόμη ελέγχεται και η απόσταση ανάμεσα από τον χρήστη και τον εχθρό έτσι ώστε εάν ο παίχτης πλησιάσει πολύ ο εχθρός να ξεκινάει να τον κυνηγάει και να τρέχει προς το μέρος του.
2. **void Chase()** - Εδώ γίνεται το κυνηγητό του παίχτη. Ο εχθρός ξεκινάει να τρέχει και να κινείται προς τον παίχτη καθώς και ετοιμάζεται για επίθεση όταν φτάσει σε αρκετά μικρή απόσταση. Αυτό συμβαίνει έως ότου ο παίχτης βγει από την εμβέλεια του εχθρού.
3. **void Attack()** - Στην κλάση αυτή ορίζεται η επίθεση του εχθρού στον παίχτη.
4. **void SetNewRandomDestination()** - Η κλάση αυτή βοηθάει στην εύρεση νέας τυχαίας διαδρομής του αντιπάλου.
5. **void Turn_On_AttackPoint()** και **void Turn_Off_AttackPoint()** - Πρόκειται για τα σημεία που θα δημιουργήσουμε αμέσως μετά που όταν έρχονται σε επαφή με τον χρήστη θα γίνεται αφαίρεση ζωής.

4.6.1 Detecting and Dealing Damage

Για να γίνεται ζημιά όμως από τους εχθρούς έπρεπε να ορίσουμε ένα σημείο το οποίο μόλις έρχεται σε επαφή με τον παίκτη θα γίνεται αφαίρεση ζωής. Επίσης το ίδιο κάναμε και για το όπλο Axe. Δημιουργήσαμε λοιπόν empty object ως childs στα αντικείμενα “Cannibal”, “Boar” και “Axe” και τα τοποθετήσαμε στην άκρη τους.



Εικόνα 18 - Attack Point

Τα “Attack Point” τέλος έχουν προσαρμοσμένο ως component το script “AttackScript”.

```
public class AttackScript : MonoBehaviour
{
    public float damage = 2f;
    public float radius = 1f;
    public LayerMask layerMask;

    void Update ()
    {
        Collider[] hits = Physics.OverlapSphere(transform.position, radius, layerMask);
```

```
        if(hits.Length > 0)
        {
            hits[0].gameObject.GetComponent<HealthScript>().ApplyDamage(damage);

            gameObject.SetActive(false);
        }
    }
}
```

4.7 Health

Αφού τελειώσαμε και με την δημιουργία των εχθρών στη συνέχεια ασχοληθήκαμε με την δημιουργία ενός script, το οποίο ονομάσαμε “HealthScript”, που θα είναι υπεύθυνο για την ζωή του παίχτη μας αλλά και των εχθρών. Το τοποθετήσαμε λοιπόν ως component και στα 3 αυτά αντικείμενα.

```
public class HealthScript : MonoBehaviour
{
    private EnemyAnimator enemy_Anim;
    private NavMeshAgent navAgent;
    private EnemyController enemy_Controller;

    public float health = 100f;

    public bool is_Player, is_Boar, is_Cannibal;
    private bool is_Dead;

    private EnemyAudio enemyAudio;

    private PlayerStats player_Stats;

    void Awake ()
    {
        if(is_Boar || is_Cannibal)
        {
            enemy_Anim = GetComponent<EnemyAnimator>();
            enemy_Controller = GetComponent<EnemyController>();
            navAgent = GetComponent<NavMeshAgent>();

            //get enemy audio
            enemyAudio = GetComponentInChildren<EnemyAudio>();
        }

        if (is_Player)
        {
            player_Stats = GetComponent<PlayerStats>();
        }
    }
}
```

```
public void ApplyDamage(float damage)
{
    //if we die don't execute the rest of the code
    if (is_Dead)
        return;

    health -= damage;

    if (is_Player)
    {
        //show the stats(display the health UI value)
        player_Stats.Display_HealthStats(health);
    }

    if(is_Boar || is_Cannibal)
    {
        if(enemy_Controller.Enemy_State == EnemyState.PATROL)
        {
            enemy_Controller.chase_Distance = 50f;
        }
    }

    if(health <= 0f)
    {
        PlayerDied();

        is_Dead = true;
    }
}

void PlayerDied()
{
    if (is_Cannibal)
    {
        GetComponent<Animator>().enabled = false;
        GetComponent<BoxCollider>().isTrigger = false;
        GetComponent<Rigidbody>().AddTorque(-transform.forward * 20f);

        enemy_Controller.enabled = false;
        navAgent.enabled = false;
        enemy_Anim.enabled = false;

        StartCoroutine(DeadSound());

        //EnemyManager spawn more enemies
        EnemyManager.instance.EnemyDied(true);
    }

    if (is_Boar)
    {
        navAgent.velocity = Vector3.zero;
        navAgent.isStopped = true;
        enemy_Controller.enabled = false;

        enemy_Anim.Dead();
    }
}
```

```

        StartCoroutine(DeadSound());

        //EnemyManager spawn more enemies
        EnemyManager.instance.EnemyDied(false);
    }

    if (is_Player)
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag(Tags.ENEMY_TAG);

        for(int i = 0; i < enemies.Length; i++)
        {
            enemies[i].GetComponent<EnemyController>().enabled = false;
        }

        //call enemy manager to stop spawning enemies
        EnemyManager.instance.StopSpawning();

        GetComponent<PlayerMovement>().enabled = false;
        GetComponent<PlayerAttack>().enabled = false;

        GetComponent<WeaponManager>().GetCurrentSelectedWeapon().GameObject.SetActive(false);
    }

    if(tag == Tags.PLAYER_TAG)
    {
        Invoke("RestartGame", 3f);
    }
    else
    {
        Invoke("TurnOffGameObject", 3f);
    }
}

//Player Died

void RestartGame()
{
    UnityEngine.SceneManagement.SceneManager.LoadScene("Game");
}

void TurnOffGameObject()
{
    gameObject.SetActive(false);
}

IEnumerator DeadSound()
{
    yield return new WaitForSeconds(0.3f);
    enemyAudio.Play_DeadSound();
}

}
}

```

Στον παραπάνω script οι ενέργειες που λαμβάνουν χώρα είναι οι εξής :

- `public void ApplyDamage(float damage)` : Στην κλάση αυτή γίνεται η εφαρμογή ζημιάς στην ζωή του παίχτη ή των εχθρών αντίστοιχα ανάλογα με τα χτυπήματα και ακόμη γίνεται έλεγχος εάν η ζωή είναι μεγαλύτερη ή όχι του μηδενός. Εάν ναι τότε αυτό σημαίνει ότι ο παίχτης ή ο εχθρός έχει πεθάνει.
- `void PlayerDied()` : Στην κλάση αυτή γίνεται η απενεργοποίηση όλων των component, ήχων, animations και navAgent καθώς το αντικείμενο είναι νεκρό. Εάν βέβαια το αντικείμενο είναι ο παίχτης μας τότε σε 3 δευτερόλεπτα το παιχνίδι πρόκειται να ξεκινήσει από την αρχή.
- `void RestartGame()` : Η κλάση αυτή ευθύνεται για την επανέναρξη του παιχνιδιού μετά τον θάνατο του παίχτη.

4.8 Player Stats UI

Για την καλύτερη εμπειρία του χρήστη δημιουργήσαμε ακόμα και ένα σύστημα UI στο οποίο θα εμφανίζονται η ζωή και η ενέργεια του παίχτη. Το Unity έχει ένα μεγάλο εύρος επιλογών το οποίο δίνει μεγάλη ελευθερία στον χρήστη και τον βοηθάει να δημιουργήσει ένα UI System όπως αυτός επιθυμεί.

Αρχικά δημιουργήσαμε ένα νέο canvas στο hierarchy και το ονομάσαμε “UI Canvas”. Στη συνέχεια ως child του αντικειμένου αυτού δημιουργήσαμε μία εικόνα την οποία τοποθετήσαμε μόνιμα στο κάτω αριστερά της οθόνης και αλλάξαμε το χρώμα της σε γκρι. Έπειτα προσθέσαμε ως child της προηγούμενης εικόνας, μία ακόμη εικόνα ενός σταυρού ο οποίος θα απεικονίζει τη ζωή του παίχτη και αλλάξαμε το χρώμα του σε κόκκινο. Τέλος προσθέσαμε και μία τρίτη εικόνα της οποίας αλλάξαμε το “Image Type” σε “Filled” και της δώσαμε το χρώμα πράσινο. Τέλος για την ενέργεια του παίχτη κάναμε ακριβώς το ίδιο με διαφορετικά χρώματα και σχέδια.

Έπειτα για να λειτουργήσει το “UI Canvas” γράψαμε ένα ακόμα script με όνομα “PlayerStats” και το προσθέσαμε ως component στον “Player” και ενημερώσαμε τα script “PlayerSprintAndCrouch” ώστε να γίνεται μείωση της ενέργειας όσο ο χρήστης έχει πατημένο το αριστερό Shift καθώς και το “HealthScript” ώστε να γίνεται αφαίρεση της χαμένης ζωής.

```
public class PlayerStats : MonoBehaviour
{
    [SerializeField]
    private Image health_Stats, stamina_Stats;

    public void Display_HealthStats(float healthValue)
    {
        healthValue /= 100f;

        health_Stats.fillAmount = healthValue;
    }

    public void Display_StaminaStats(float staminaValue)
    {
        staminaValue /= 100f;

        stamina_Stats.fillAmount = staminaValue;
    }
}
```



Εικόνα 19 - Player Stats

4.9 Creating Enemy Manager

Έπειτα και από την δημιουργία του συστήματος UI ένα ακόμη empty object με όνομα “Enemy Manager”. Ως child του αντικειμένου αυτού δημιουργήσαμε και άλλα τα οποία τοποθετήσαμε σε διάφορα τυχαία σημεία στον χάρτη μας τα οποία θα εξυπηρετούν ως spawn points των αντιπάλων. Από εκεί δηλαδή θα δημιουργούνται αντίπαλοι ανά τακτά χρονικά διαστήματα. Το script που γράψαμε για τη λειτουργία αυτή, “EnemyManagerScript” είναι το εξής :

```
public class EnemyManager : MonoBehaviour
{
    public static EnemyManager instance;

    [SerializeField]
    private GameObject boar_Prefab, cannibal_Prefab;

    public Transform[] cannibal_SpawnPoints, boar_SpawnPoints;

    [SerializeField]
    private int cannibal_Enemy_Count, boar_Enemy_Count;

    private int initial_Cannibal_Count, initial_Boar_Count;

    public float wait_Before_Spawn_Enemies_Time = 5f;

    void Awake ()
    {
        MakeInstance();
    }

    void Start()
    {
        initial_Cannibal_Count = cannibal_Enemy_Count;
        initial_Boar_Count = boar_Enemy_Count;

        SpawnEnemies();

        StartCoroutine("CheckToSpawnEnemies");
    }

    void SpawnEnemies()
    {
        SpawnCannibals();
        SpawnBoars();
    }
}
```

```
void SpawnCannibals()
{
    int index = 0;

    for (int i = 0; i < cannibal_Energy_Count; i++)
    {
        if(index >= cannibal_SpawnPoints.Length)
        {
            index = 0;
        }
        Instantiate(cannibal_Prefab, cannibal_SpawnPoints[index].position,
            Quaternion.identity);

        index++;
    }

    cannibal_Energy_Count = 0;
}

void SpawnBoars()
{
    int index = 0;

    for (int i = 0; i < boar_Energy_Count; i++)
    {
        if (index >= boar_SpawnPoints.Length)
        {
            index = 0;
        }
        Instantiate(boar_Prefab, boar_SpawnPoints[index].position, Quaternion.identity);

        index++;
    }

    boar_Energy_Count = 0;
}

void MakeInstance ()
{
    if(instance == null)
    {
        instance = this;
    }
}

IEnumerator CheckToSpawnEnemies()
{
    yield return new WaitForSeconds(wait_Before_Spawn_Enemies_Time);

    SpawnCannibals();

    SpawnBoars();

    StartCoroutine("CheckToSpawnEnemies");
}
```

```
public void EnemyDied(bool cannibal)
{
    if (cannibal)
    {
        cannibal_Enemy_Count++;

        if(cannibal_Enemy_Count > initial_Cannibal_Count)
        {
            cannibal_Enemy_Count = initial_Cannibal_Count;
        }
    }
    else
    {
        boar_Enemy_Count++;

        if(boar_Enemy_Count > initial_Boar_Count)
        {
            boar_Enemy_Count = initial_Boar_Count;
        }
    }
}

public void StopSpawning()
{
    StopCoroutine("CheckToSpawnEnemies");
}
}
```

Δημιουργούμε την αρχικά την κλάση `void MakeInstance ()` η οποία είναι singleton. Είναι ένας τρόπος ένας τρόπος δηλαδή για να διασφαλιστεί ότι μια τάξη διαθέτει μόνο μία μοναδική παρουσία διαθέσιμη ανά πάσα στιγμή. Συμπεριφέρεται σαν κανονική στατική τάξη αλλά με κάποια πλεονεκτήματα.

Έπειτα δημιουργούμε τις κλάσεις `void SpawnCannibals()` και `void SpawnBoars()` τις οποίες θα καλούμε μέσα στην κλάση `void SpawnEnemies()`. Οι κλάσεις αυτές ουσιαστικά θα δημιουργούν τους αντιπάλους στα διάφορα σημεία που έχουμε ορίσει στον χάρτη χωρίς όμως να ξεπερνούν έναν συγκεκριμένο αριθμό.

Στη συνέχεια στην κλάση `public void EnemyDied(bool cannibal)` θα γίνεται έλεγχος για τον θάνατο των εχθρών ώστε να γίνονται καινούρια spawn.

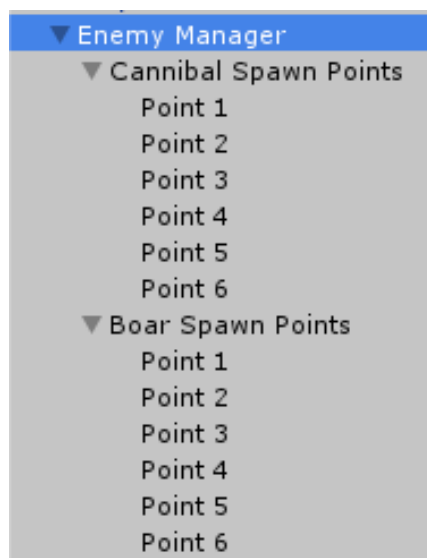
Τέλος δημιουργούμε την κλάση `IEnumerator CheckToSpawnEnemies()` η οποία θα είναι υπεύθυνη για το spawn των αντιπάλων η οποία θα τσεκάρει εάν χρειάζονται καινούρια spawn και μετά από κάποιο χρόνο θα δημιουργεί νέους εχθρούς.

Τι είναι το Coroutine;

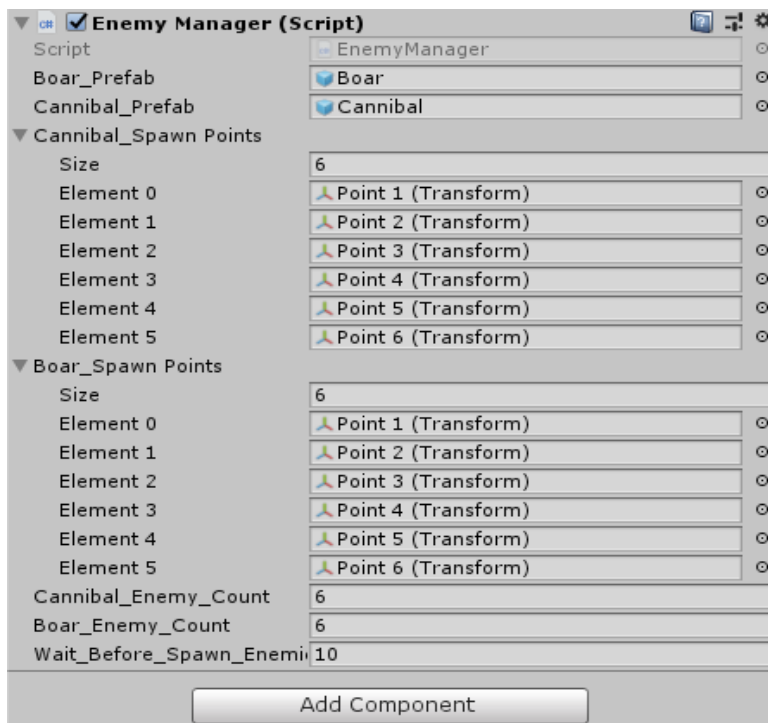
Το Coroutine είναι ένας τύπος Unity που χρησιμοποιείται για τη δημιουργία παράλληλων ενεργειών που επιστρέφουν έναν IEnumerator για να το κάνουν

Τι είναι το IEnumerator;

Το IEnumerator είναι μια διεπαφή, η οποία κατά την εφαρμογή μας επιτρέπει να επαναλάβουμε τη λίστα των στοιχείων ελέγχου. Για να το εφαρμόσουμε απαιτεί να παρέχουμε δύο μεθόδους - Reset για να επιστρέψουμε στην αρχή της λίστας και MoveNext για να προχωρήσουμε προς τα εμπρός και Current για να λάβουμε το τρέχον στοιχείο.



Εικόνα 21 - "Enemy Manager"



Εικόνα 20 - "Enemy Manager" Inspector

4.10 Coins and Exit Door

Έχοντας ολοκληρώσει τις βασικότερες λειτουργίες του παιχνιδιού μας απομένει ένας τρόπος ώστε ο παίχτης να μπορεί να τερματίσει το παιχνίδι και να γίνει έξοδος από τη σκηνή. Για τον λόγο αυτόν δημιουργήσαμε κάποιους “θησαυρούς” τους οποίους μόλις ο παίχτης συλλέξει θα μπορέσει να οδηγηθεί στην έξοδο. Δημιουργήσαμε λοιπόν ένα αντικείμενο το οποίο ονομάσαμε coin και το τοποθετήσαμε πολλαπλές φορές στον χάρτη.



Εικόνα 22 - Coins

Στη συνέχεια προσθέσαμε στο PlayerUI ένα μετρητή των νομισμάτων έτσι ώστε ο παίχτης να μπορεί να καταλάβει πόσα απομένουν να ανακαλύψει. Τέλος προσθέσαμε μία πολύ εμφανή κλειστή πόρτα η οποία με το που συλλεχθεί και το τελευταίο νόμισμα στον χάρτη ανοίγει και ο παίχτης μπορεί να περάσει μέσα από αυτήν και να οδηγηθεί στον τερματισμό.



Εικόνα 24 - Πόρτα τερματισμού



Εικόνα 23 - Πόρτα τερματισμού μετά τη συλλογή των νομισμάτων

Δημιουργήσαμε ακόμα 3 script. Το πρώτο ονομάζεται “CoinBehavior” και το τοποθετήσαμε ως component στο νόμισμα που φτιάξαμε.

```
public class CoinBehaviour : MonoBehaviour
{
    GameManagerScript GMS;
    private float rotateSpeed = 10;

    void Awake()
    {
        GMS = GameObject.Find("GameManager").GetComponent<GameManagerScript>();
        GMS.current_coins++;
    }

    void Update ()
    {
        transform.Rotate(Vector3.left * rotateSpeed);
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            Destroy(gameObject);
            GMS.current_coins--;
            //adding score...
            GMS.UpdateUI();
        }
    }
}
```

Το συγκεκριμένο script έχει δύο λειτουργίες. Η πρώτη είναι να κάνει το νόμισμα να περιστρέφεται ώστε να γίνεται πιο αληθοφανές και η δεύτερη λειτουργία του είναι να εξαφανίζει το κέρμα αυτό όταν γίνεται σύγκρουση με το αντικείμενο “Player”. Όταν αυτό γίνεται τα νομίσματα θα μειώνονται κατά 1.

Το δεύτερο script ονομάζεται “GameManagerScript” και είναι το εξής :

```
public class GameManagerScript : MonoBehaviour
{
    public Text coinsLeft;

    public int current_coins = 0;

    public int max_coins = 0;

    public GameObject Door;
```

```
void Start ()
{
    Door.SetActive(true);

    max_coins = current_coins;

    UpdateUI();
}

void Update ()
{
}

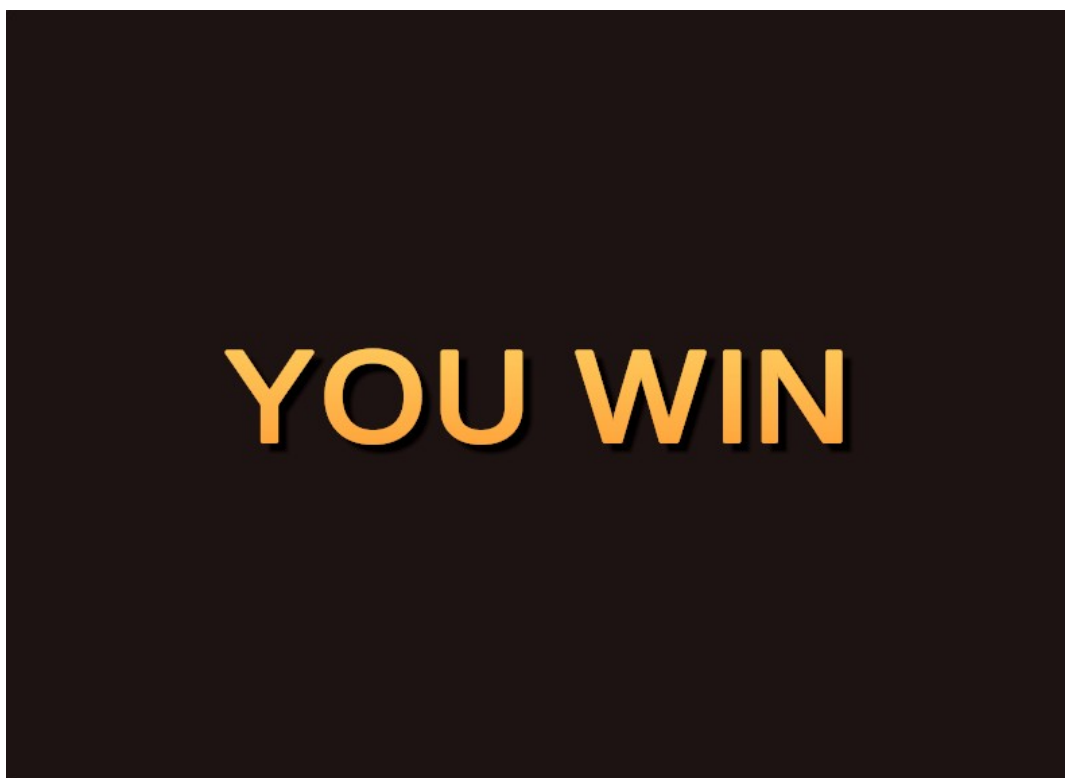
public void UpdateUI()
{
    if(current_coins > 0)
    {
        coinsLeft.text = "Coins Left: " + current_coins.ToString() + "/" +
            max_coins.ToString();
    }

    else if(current_coins <= 0)
    {
        Door.SetActive(false);
    }
}
}
```

Η λειτουργία του είναι να εμφανίζει την πόρτα στην αρχή και για όσο τα νομίσματα είναι μεγαλύτερα του 0 τότε να κρατάει την πόρτα αυτή ενεργή. Όταν τα νομίσματα φτάσουν μικρότερα ή ίσα του 0 τότε αυτή η πόρτα θα εξαφανιστεί. Όταν η πόρτα εξαφανιστεί λοιπόν θα εμφανιστεί ένα σημείο που βρίσκεται κρυμμένο ανάμεσα στην πόρτα στο οποίο θα προσθέσουμε ως component το επόμενο script “LoadLevel” . Το σημείο αυτό είναι αόρατο προς τον χρήστη και όταν ουσιαστικά επιχειρήσει να περάσει την πόρτα, θα γίνει σύγκρουση με το σημείο αυτό και θα οδηγηθεί στον τερματισμό του παιχνιδιού.

```
public class LoadLevel : MonoBehaviour
{
    public int levelToLoad;

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            Application.LoadLevel(levelToLoad);
        }
    }
}
```



Εικόνα 25 - Μήνυμα "You Win"

Η επόμενη σκηνή που θα φορτωθεί στο παιχνίδι είναι η “WinLevel” όπως την ονομάσαμε και βέβαια στην σκηνή αυτή ο χρήστης δεν θα μπορεί να κάνει κάποια ενέργεια. Θα εμφανίζεται το νικητήριο μήνυμα το οποίο θα σημειώνει και τη λήξη του παιχνιδιού. Έπειτα από ένα μικρό χρονικό διάστημα ο χρήστης θα επιστρέφει στο αρχικό μενού του παιχνιδιού. Αυτό θα γίνεται με ένα script που έχουμε προσθέσει στον canvas που αναγράφεται και το μήνυμα. Το script “LoadMainMenu” θα παίζει αμέσως αφού ο χρήστης εμφανίσει την σκηνή.

```
public class LoadMainMenu : MonoBehaviour
{
    public float delay = 40;
    public string NewLevel = "MainMenu";
    void Start()
    {
        StartCoroutine(LoadLevelAfterDelay(delay));
    }
    IEnumerator LoadLevelAfterDelay(float delay)
    {
        yield return new WaitForSeconds(delay);
        SceneManager.LoadScene(NewLevel);
    }
}
```

4.11 Creating Menu

Τέλος για την ολοκλήρωση του παιχνιδιού δημιουργήσαμε δύο μενού, ένα αρχικό μενού και άλλο ένα για την παύση του παιχνιδιού. Η προσθήκη των δύο αυτών, βοηθάει στην ομαλότερη λειτουργία του παιχνιδιού καθώς και το κάνει πιο εύχρηστο στον χρήστη. Στο αρχικό μενού η αλλιώς “Main Menu” όπως το ονομάσαμε, ο χρήστης συναντάει 3 κουμπιά, τα Start, Settings και Quit. Με το πρώτο το παιχνίδι ξεκινάει ενώ με το τελευταίο γίνεται έξοδος από το παιχνίδι. Το δεύτερο τον οδηγεί στις επιλογές Settings όπου έχει την δυνατότητα να αλλάξει την ανάλυση του παιχνιδιού αλλά και τα γραφικά του έτσι ώστε να ανταποκρίνεται καλύτερα στον εκάστοτε υπολογιστή.



Εικόνα 26 - Αρχικό Μενού

Το δεύτερο η αλλιώς “Pause Menu” είναι για την παύση του παιχνιδιού. Ο χρήστης με το πάτημα του κουμπιού “Esc” έχει την δυνατότητα να σταματήσει το παιχνίδι και να το συνεχίσει οποιαδήποτε στιγμή επιθυμεί. Οι επιλογές που θα εμφανιστούν στην οθόνη του αυτή τη φορά θα είναι

2, η επιλογή “Menu” η οποία θα τον οδηγεί δηλαδή στο αρχικό μενού καθώς και η επιλογή “Quit” η οποία θα τερματίσει το παιχνίδι και θα οδηγήσει τον χρήστη στην επιφάνεια εργασίας του. Εάν ο χρήστης επιθυμεί να συνεχίσει το παιχνίδι τότε αρκεί να πατήσει ξανά το κουμπί “Esc”.



Εικόνα 27 - Pause Menu

Ο κώδικας που δημιουργήθηκε για την υλοποίηση των μενού είναι :

Για το αρχικό μενού

```
public class MainMenu : MonoBehaviour
{
    public void PlayGame ()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void QuitGame ()
    {
        Debug.Log("QUIT");
        Application.Quit();
    }
}
```

Για το μενού των ρυθμίσεων

```
public class SettingsMenu : MonoBehaviour
{
    public AudioManager audioMixer;

    public Dropdown resolutionDropdown;

    Resolution[] resolutions;

    void Start ()
    {
        resolutions = Screen.resolutions;

        resolutionDropdown.ClearOptions();
        NewMethod();

        List<string> options = new List<string>();

        int currentResolutionIndex = 0;

        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + "x" + resolutions[i].height;
            options.Add(option);

            if (resolutions[i].width == Screen.currentResolution.width &&
                resolutions[i].height == Screen.currentResolution.height)
            {
                currentResolutionIndex = i;
            }
        }

        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();
    }

    private void NewMethod()
    {
        resolutionDropdown.ClearOptions();
    }

    public void SetResolution (int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
    }

    public void SetQuality (int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
    }
}
```

Και τέλος για το μενού παύσης

```
public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);
    }

    public void QuitGame()
    {
        Debug.Log("Quitting game...");
        Application.Quit();
    }
}
```

5. ΒΙΒΛΙΟΓΡΑΦΙΑ

- Αναγνώστου Κώστας, 2009. Βιντεοπαιχνίδια, βιομηχανία και ανάπτυξη, Κλειδάριθμος. Αθήνα
- Deitel J. Paul, Deitel M. Harvey, 2010. C Προγραμματισμός (+cd), Παρουσίαση c++ και αντικειμενοστραφούς προγραμματισμού με τη βιβλιοθήκη allegro c, Γκιούρδας Μ. Αθήνα

Διπλωματικές εργασίες

- Γεωργόπουλος Χρήστος, 2014. Δημιουργία mini παιχνιδιών με τη χρήση της μηχανής παιχνιδιών “Unity”, ΑΤΕΙ Θεσσαλίας, Σχολή Τεχνολογικών Εφαρμογών, Τμήμα Μηχανικών Πληροφορικής ΤΕ, Λάρισα
- Haas John, 2014. A History of the Unity Game Engine, An Interactive Qualifying Project, Worcester Polytechnic Institute. Massachusetts

Δικτυογραφία

- <https://blogs.unity3d.com>
- https://en.wikipedia.org/wiki/List_of_Unity_games
- [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))
- https://en.wikipedia.org/wiki/Unity_Technologies#See_also
- <https://www.globenewswire.com/news-release/2010/11/10/1251905/0/en/Unity-Technologies-Launches-3rd-Party-Marketplace-Unity-Asset-Store.html>

- <https://www.globenewswire.com/news-release/2011/07/21/1251946/0/en/Unity-Technologies-Lands-12-Million-in-Series-B-Funding-Led-by-WestSummit-Capital-and-iGlobe-Partners.html>
- <https://www.globenewswire.com/news-release/2012/06/18/1252006/0/en/The-Next-Generation-of-the-Unity-Game-Engine-Unveiled.html>
- <https://www.mcvuk.com/development-news/united-they-stand/116643/>
- <https://www.owler.com/company/unity3d>
- <https://sundaysundae.co/best-games-made-with-unity/>
- <https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/?guccounter=1>
- <https://unity.com>

Πηγές κώδικα

- https://www.youtube.com/watch?v=3khBcfXH__c
- <https://www.youtube.com/watch?v=JivuXdrIHK0>
- https://www.youtube.com/watch?v=zc8ac_qUXQY

(Τελευταία Επίσκεψη: 24/08/2020)