ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Πτυχιακή εργασία

Σχεδίαση και δημιουργία παιχνιδιού με το Unity

Ηλιόπουλος Μάριος
2022201700053

Επιβλέπων:

Πλατής Νικόλαος
Επίκ. Καθηγητής

Τρίπολη, Απρίλιος 2022

Περίληψη

Η παρούσα πτυχιακή εργασία έχει αντικείμενο το σχεδιασμό και τη δημιουργία ενός παιχνιδιού με τη μηχανή παιχνιδιού Unity 3D και με τη γλώσσα προγραμματισμού C#. Το παιχνίδι που αναπτύχθηκε είναι ένα 2.5D tower defense game.

Αρχικά, γίνεται περιληπτικά μια αναφορά στις πιο γνωστές μηχανές παιχνιδιών και εξηγείται γιατί έγινε η επιλογή του Unity 3D. Στη συνέχεια αναφέρουμε μερικούς από τους τύπους παιχνιδιών και παραδείγματα αυτών. Έπειτα, παρουσιάζουμε το tower defense game, screenshot από το παιχνίδι και κώδικες-scripts που αναπτύχθηκαν. Τέλος, συνοψίζουμε την αναφορά και αναφέρουμε πιθανές επεκτάσεις του παιχνιδιού. Δίνουμε επίσης έναν οδηγό εγκατάστασης για το Unity 3D καθώς και άλλα προγράμματα που χρησιμοποιήθηκαν.

Abstract

This text is a reference for the present essay on designing and creating a game with the game engine *Unity 3D* and the *C#* programming language. The game that developed is a 2.5D tower defense game.

First, a brief overview of the most popular gaming machines and why the Unity 3D was chosen. Next, we will present some of the most popular types of video games and examples of them. Then, we will describe the tower defense game, and represent screenshots of the game and codes-scripts used. Finally, we summarize the report and mention possible extensions of the game. We also provide an installation guide for the Unity 3D.

Περιεχόμενα

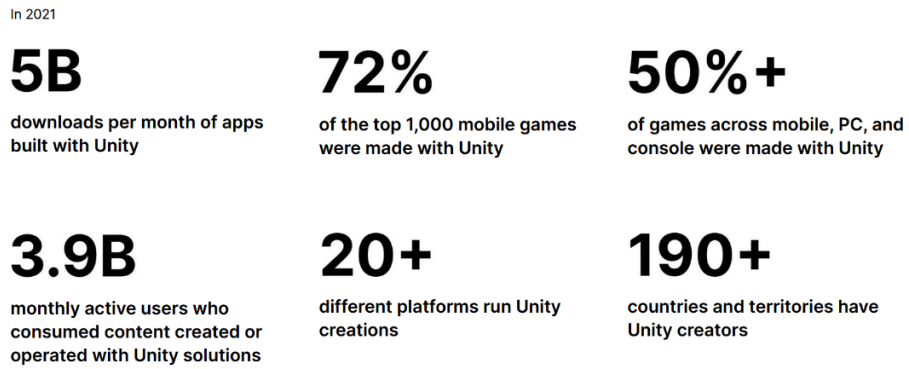
1 Game Engines

Η βιομηχανία ηλεκτρονικών παιχνιδιών αποτελεί ένα μεγάλο κομμάτι στην οικονομία ολόκληρου του πλανήτη. Η βιομηχανία περιλαμβάνει την παραγωγή και το marketing ενός video game. Δίνεται η δυνατότητα σε πολλές θέσεις εργασίας, όπως για παράδειγμα σχεδιαστές, καλλιτέχνες, προγραμματιστές και πολλά ακόμη επαγγέλματα. Για να φτιαχτεί όμως ένα παιχνίδι χρησιμοποιούνται προγράμματα που ειδικεύονται για αυτόν τον τομέα. Τα προγράμματα αυτά αναλαμβάνουν λειτουργίες αναγκαίες για την δημιουργία ενός παιχνιδιού. Η πιο σημαντική λειτουργία είναι το rendering ή αλλιώς απεικόνιση γραφικών. Επιπλέον εργασίες, που μπορούν να κάνουν τα game engines, εξίσου σημαντικές με το rendering είναι η παραγωγή και επεξεργασία ήχων, δίνει τη δυνατότητα ρεαλιστικών ήχων με την καλύτερη εμπειρία μέσω των παιχνιδιών. Επίσης, χρησιμοποιούνται σε εξαιρετικό βαθμό οι φυσικοί νόμοι, για παράδειγμα η βαρύτητα, τριβή και γενικά οι δυνάμεις. Όλα αυτά είναι βασικά χαρακτηριστικά ενός game engine στην δημιουργία ενός παιχνιδιού, από το πιο απλό μέχρι το πιο σύνθετο. Στην επόμενη ενότητα παρουσιάζονται μερικά από τα πιο δημοφιλή προγράμματα δημιουργίας ηλεκτρονικών παιχνιδιών.

1.1 Unity 3D

Το Unity 3D είναι ένα από τα πιο δημοφιλή προγράμματα σχεδίασης και δημιουργίας ηλεκτρονικών παιχνιδιών. Η εταιρεία Unity Technologies ανακοίνωσε το Unity στο *World Developers Conference* που διοργανώθηκε από την Apple τον Ιούνιο του 2005. Αρχικά προοριζόταν αποκλειστικά για λειτουργία σε λειτουργικό σύστημα Mac OS, πλέον όμως υποστηρίζει και πολλά άλλα λειτουργικά συστήματα, όπως Windows, Linux, καθώς και Android, iOS για κινητές συσκευές. Επίσης, το Unity υποστηρίζει και τις πιο δημοφιλείς παιχνιδομηχανές όπως PlayStation, Xbox, Nintendo και Stadia. Το Unity προσφέρει τη δυνατότητα δημιουργίας παιχνιδιού με drag-and- drop λειτουργία, παρ' όλα αυτά τα περισσότερα project απαιτούν τη γραφή κώδικα. Οι χρήστες μπορούν να γράψουν σε γλώσσες *C#, JavaScript* και *Boo* (σύνταξη παρόμοια της Python), αν και στις πιο πρόσφατες εκδόσεις του Unity η Boo δεν υποστηρίζεται καθόλου και η Javascript πρόκειται να αφαιρεθεί. Το περιβάλλον ανάπτυξης τρέχει χρησιμοποιώντας το *Mono,* μια εφαρμογή ανοιχτού κώδικα του *.NET Framework.* Ωστόσο, το Unity έχει φτιαχτεί με τη γλώσσα *C++* [1][7].

Σύμφωνα με στατιστικά του 2021 από το site του Unity [2]:



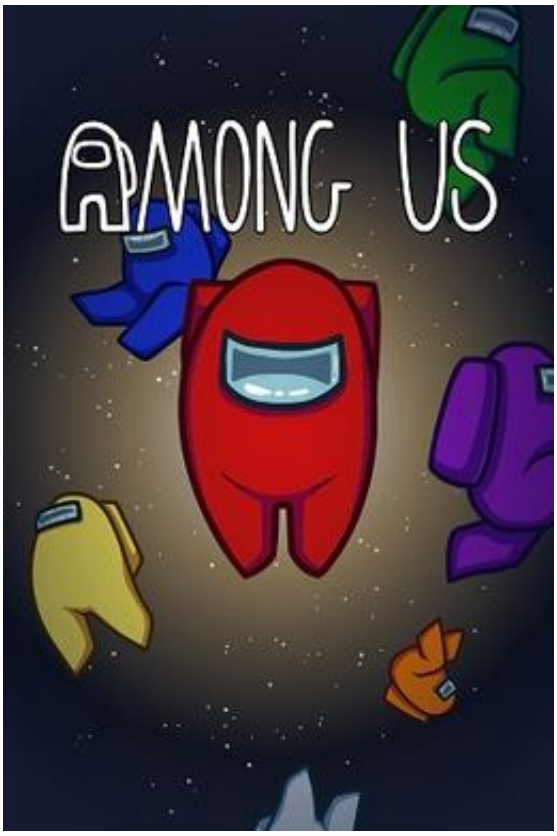
Εικόνα 1 - Στατιστικά από το Unity, 2021

Δημοφιλή παιχνίδια που φτιάχτηκαν με το Unity [3]:

Εικόνα 2 – παιχνίδι: Hearthstone, εταιρεία προγραμματισμού: Blizzard Entertainment, εκδότης: Blizzard Entertainment, είδος: Card game, χρονολογία: 2014, δημιουργήθηκε με το Unity

Εικόνα 3 – παιχνίδι: Temple Run, εταιρεία προγραμματισμού: Imangi Studios, εκδότης: Imangi, είδος: Endless runner, χρονολογία: 2011

Εικόνα 4 – παιχνίδι: Pokémon Go, εταιρεία προγραμματισμού: Niantic, εκδότης: Niantic, είδος: Augmented Reality, χρονολογία: 2016



Εικόνα 5 – παιχνίδι: Among Us, εταιρεία προγραμματισμού: Innersloth, εκδότης: Innersloth, είδος: Party/Social Deduction, χρονολογία: 2018

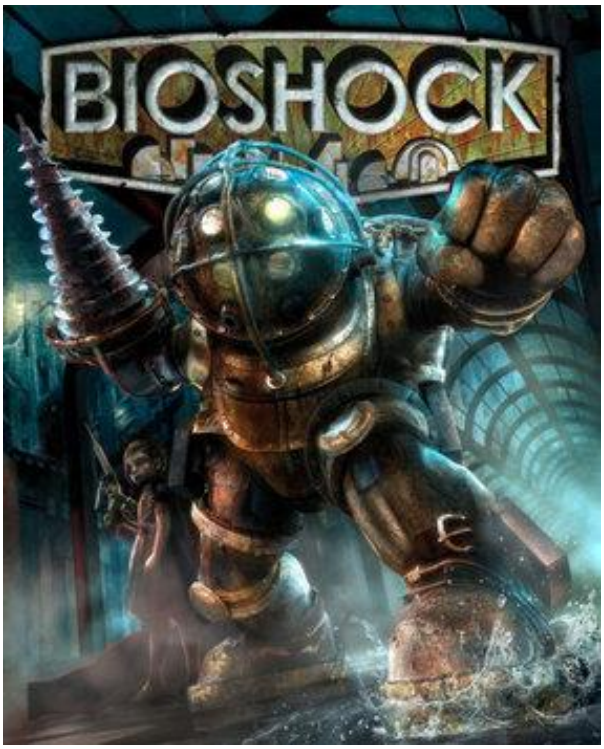
1.2 Unreal Engine

Η Unreal Engine είναι μια μηχανή δημιουργίας παιχνιδιών, προερχόμενη από την εταιρεία *Epic Games*, που πρωτοεμφανίστηκε το 1998 σε ένα first-person shooter παιχνίδι που ονομάζεται *Unreal*. Αρχικά η μηχανή είχε σχεδιαστεί για την παραγωγή first-person shooter games, όμως αργότερα ξεκίνησε να χρησιμοποιείται και για άλλους τύπους 3D παιχνιδιών. Η εταιρεία έχει συνολικά πέντε (5) εκδόσεις, First Generation ή αλλιώς Unreal Engine 1, Unreal Engine 2, Unreal Engine 3, Unreal Engine 4 και τέλος η πιο καινούρια έκδοση Unreal Engine 5 [4].

Η Unreal Engine διαθέτει εργαλεία με τα οποία κάποιος μπορεί να δημιουργήσει ρεαλιστικά 3D γραφικά, τα οποία υποστηρίζονται από υπολογιστές (Windows, Linux, MacOS), κινητά (Android, iOS) και κονσόλες (Xbox, PlayStation). Επίσης, η τεχνολογία της συγκεκριμένης μηχανής έχει χρησιμοποιηθεί στον κινηματογράφο και στην τηλεοπτική βιομηχανία. Οι χρήστες μπορούν να γράψουν σε *C++* γλώσσα προγραμματισμού [5].

Παιχνίδια που φτιάχτηκαν με την Unreal Engine [4]:

Εικόνα 6 – παιχνίδι: Unreal Tournament, εταιρεία προγραμματισμού: Epic Games/Digital Extremes, εκδότης: GT Interactive Software (PC), είδος: First Person Shooter, χρονολογία: 1999, δημιουργήθηκε με το Unreal Engine 1



Εικόνας 7 – παιχνίδι: Bioshock, εταιρεία προγραμματισμού: 2K Games, εκδότης: 2K Games, είδος: First Person Shooter, χρονολογία: 2007

Εικόνα 8 – παιχνίδι: Batman Arkham Asylum, εταιρεία προγραμματισμού: Rocksteady Studios, εκδότης: Eidos Interactive/Warner Bros Interactive Entertainment, είδος: Action-Adventure, χρονολογία: 2009

Εικόνα 9 – παιχνίδι: Tekken 7, εταιρεία προγραμματισμού: Bandai Namco Studios, εκδότης: Bandai Namco Entertainment, είδος: Fighting, χρονολογία: 2015

1.3 Επιλογή game engine

Για την παρούσα πτυχιακή εργασία επιλέχθηκε η μηχανή Unity. Οι βασικοί λόγοι για την επιλογή ήταν (α) η ύπαρξη πολλών tutorials στο YouTube στα αγγλικά αλλά και στα ελληνικά, και (β) το γεγονός ότι είναι αρκετά πιο ελαφρύ πρόγραμμα από το Unreal Engine, χρησιμοποιώντας λιγότερους πόρους του υπολογιστή και προσφέροντας καλύτερη εμπειρία στον χρήστη. Παρότι η γλώσσα προγραμματισμού C# που απαιτείται για την ανάπτυξη με Unity δεν ήταν γνώριμη, αφού δεν έχει διδαχθεί από το Πανεπιστήμιο, δεν υπήρξε εμπόδιο για τη χρήση του εργαλείου, καθώς είναι μια αντικειμενοστρεφής γλώσσα με πολλά κοινά χαρακτηριστικά με την Java που έχει διδαχθεί.

Στο τέλος του εγγράφου υπάρχει παράρτημα με έναν οδηγό εγκατάστασης του προγράμματος.

2 Τύποι ηλεκτρονικών παιχνιδιών

Στην προηγούμενη ενότητα έγινε παρουσίαση των προγραμμάτων για τη δημιουργία ηλεκτρονικών παιχνιδιών. Στην παρούσα ενότητα θα γίνει επίδειξη μερικών από τους γνωστούς τύπους ηλεκτρονικών παιχνιδιών, μαζί με παραδείγματα από δημοφιλή παιχνίδια του κάθε τύπου [6].

2.1 Action Games

Τα action games είναι ο πιο δημοφιλής τύπος παιχνιδιού. Τέτοια παιχνίδια κατά πλειοψηφία περιέχουν σκηνές μάχης και αγωνίας, με τον παίχτη να βρίσκεται στο επίκεντρο της δράσης και με κατάλληλες δεξιότητες να μπορεί να ξεπεράσει τις δυσκολίες του παιχνιδιού. Αυτός ο τύπος παιχνιδιού περιέχει πολλές υποκατηγορίες.

2.1.1 Shooter

Στην κατηγορία αυτή, ο παίχτης πρέπει να εξοντώσει τους εχθρούς του με τη χρήση όπλων. Είναι μια κατηγορία που απευθύνεται κυρίως σε ενήλικο κοινό καθώς προβάλλει όπλα, πράγμα ακατάλληλο προς τους κάτω των δεκαοχτώ (18) ετών. Επίσης, η υποκατηγορία Shooter χωρίζεται σε First-person shooter και Third-person shooter, τα οποία ξεχωρίζουν ανάλογα με την τοποθέτηση της κάμερας και την προοπτική του παίχτη.

Εικόνα 10 - Αριστερά – παιχνίδι: Call Of Duty, First-person shooter – Δεξιά – παιχνίδι: Gears Of War 3, Third-person shooter

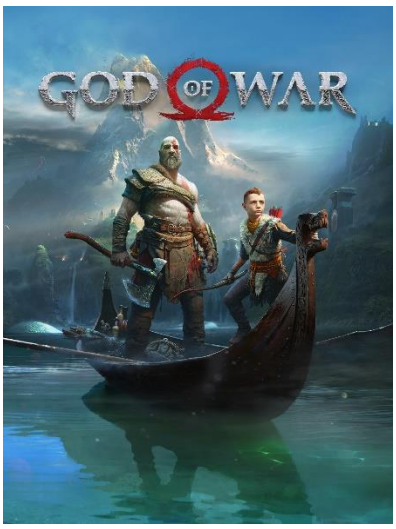
2.1.2 Fighting

Παιχνίδια αυτού του τύπου περιλαμβάνουν μάχες σώμα με σώμα χρησιμοποιώντας πολεμικές τέχνες. Ο παίχτης κερδίζει όταν καταφέρει να μηδενίσει τους πόντους ζωής του αντιπάλου του.

Εικόνα 11 – παιχνίδι: Street Fighter V

2.1.3 Hack and Slash

Το Hack and Slash μοιάζει με το Fighting, όμως ο παίχτης αντί να αντιμετωπίζει έναν εχθρό, έχει μπροστά του ένα κύμα εχθρών. Με κατάλληλες ενέργειες και δεξιότητες μπορεί να κατατροπώσει τους εχθρούς του.



Εικόνα 12 – παιχνίδι: God Of War

2.2 Adventure Games

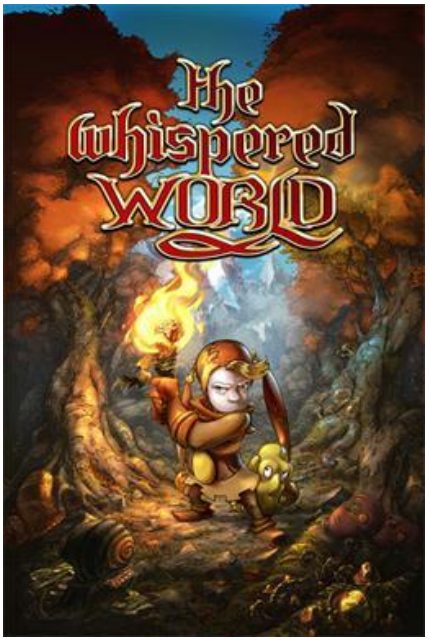
Συνήθως τέτοια παιχνίδια περιγράφουν μια ιστορία και ο παίχτης είναι ο πρωταγωνιστής και ελέγχει τον βασικό χαρακτήρα της ιστορίας. Περιέχει puzzle και δυσκολίες, που στέκονται εμπόδια για την εξέλιξη της πλοκής.

2.2.1 Text adventure games

Ήταν από τα πρώτα adventure games και δεν χρησιμοποιούνται γραφικά υπολογιστή, παρά μόνο ένα κείμενο το οποίο περιγράφει την πλοκή του παιχνιδιού. Από τα πρώτα παιχνίδια αυτής της κατηγορίας είναι το *Colossal Cave Adventure*.

2.2.2 Graphic adventure games

Χρησιμοποιούνται γραφικά για να απεικονιστεί το περιβάλλον στον παίχτη. Μια υποκατηγορία είναι τα Point and Click. Στη διάθεσή του ο παίχτης έχει μόνο το ποντίκι του υπολογιστή και κάνοντας click στα κατάλληλα σημεία κάνει πρόοδο στην ιστορία του παιχνιδιού.



Εικόνα 13 – παιχνίδι: The Whispered World

2.2.3 Interactive Movie

Παιχνίδια αυτού του τύπου χρησιμοποιούν είτε γραφικά είτε ηθοποιούς. Ο παίχτης με τις επιλογές του επηρεάζει την πλοκή της ταινίας. Επειδή τα παιχνίδια αυτά συνήθως είναι περιορισμένα στα γραφικά ή στην ηθοποιία, ο παίχτης έχει ελάχιστες ενέργειες και οι λάθος επιλογές του μπορεί να οδηγήσουν σε πρόωρο τέλος το παιχνίδι.

Εικόνα 14 – παιχνίδι: Under a Killing Moon

2.3 Role-Playing Games (RPG)

Ακόμα μια δημοφιλής κατηγορία ηλεκτρονικών παιχνιδιών. Κατά πλειοψηφία έχουν μεσαιωνικό ύφος και στυλ. Όμως, δεν παραμένουν μόνο στο μεσαιωνικό στυλ, παρατηρούνται και παιχνίδια με διαφορετικό φανταστικό ύφος, π.χ. στο διάστημα κλπ. Αυτή η κατηγορία παιχνιδιών προέρχεται από το γνωστό επιτραπέζιο *Dungeons and Dragons*, το οποίο παίζεται χρησιμοποιώντας χαρτί και μολύβι. Πλέον υπάρχει στη διάθεση του κόσμου ο ηλεκτρονικός υπολογιστής και λόγω αυτού, παιχνίδια αυτού του τύπου αποκτούν γραφικά. Υπάρχουν υποκατηγορίες όπως για παράδειγμα ***Action RPG***, ***MMORPG, Roguelike, Tactical RPG.*** Στην γενική κατηγορία, ο παίχτης μπορεί να διαλέξει ανάμεσα σε πολλούς χαρακτήρες του παιχνιδιού και μπορεί να κάνει πρόοδο στη ιστορία ολοκληρώνοντας αποστολές. Στη διάθεσή του ο παίχτης έχει μόνο το ποντίκι του υπολογιστή και κάνοντας click στα κατάλληλα σημεία κάνει πρόοδο στην ιστορία του παιχνιδιού.

Εικόνα 15 – παιχνίδι: Diablo Eternal Collection

2.4 Strategy

Τα παιχνίδια στρατηγικής στοχεύουν στις δεξιότητες διαχείρισης πόρων των παιχτών. Για να καταφέρουν να ολοκληρώσουν τους στόχους του παιχνιδιού πρέπει να βρουν και να εφαρμόσουν μια στρατηγική κατάλληλη, η οποία θα χρησιμοποιεί με σύνεση τους πόρους που δίνονται στους παίχτες.

2.4.1 Real-time strategy (RTS)

Τα παιχνίδια στρατηγικής σε πραγματικό χρόνο απαιτούν από τον παίχτη να συλλέγει και να διατηρεί πόρους, όπως βάσεις, ενώ προχωρά και αναπτύσσει τόσο πόρους όσο και μονάδες μάχης.

Εικόνα 16 – παιχνίδι: StarCraft 2

2.4.2 Multiplayer Online Battle Arena (MOBA)

Η κατηγορία αυτή συνδυάζει παιχνίδια δράσης, παιχνίδια ρόλων και παιχνίδια στρατηγικής σε πραγματικό χρόνο. Τα παιχνίδια αυτής της υποκατηγορίας είναι ομαδικά παιχνίδια και παίχτες από όλο το κόσμο συνεργάζονται μαζί για να αντιμετωπίσουν την αντίπαλη ομάδα. Δεν δημιουργούν βάσεις ή μονάδες μάχης, αντίθετα, οι παίχτες ελέγχουν έναν χαρακτήρα.

Εικόνα 17 – παιχνίδι: League of Legends

2.4.3 Tower Defense

Στα παιχνίδια άμυνας πύργων (Tower Defense), οι παίχτες για να κερδίσουν πρέπει να αντιμετωπίσουν τους εχθρούς που ελέγχονται από τον υπολογιστή. Οι δυνατότητες των πύργων και οι κινήσεις των εχθρών διαφέρουν από παιχνίδι σε παιχνίδι. Συνήθως, οι πύργοι διαθέτουν διαφορετικές δυνατότητες, όπως για παράδειγμα να επιβραδύνουν τους εχθρούς ή να προκαλούν ζημιά μεγάλης εμβέλειας. Κάθε φορά που ένας εχθρός εξοντώνεται, ο παίχτης κερδίζει χρήματα για να μπορεί να αγοράσει περισσότερους πύργους ή να μπορέσει να αναβαθμίσει τους ήδη υπάρχοντες.

Εικόνα 18 – παιχνίδι: Plants vs Zombies

2.5 Simulation Games

Τα παιχνίδια προσομοίωσης είναι σχεδιασμένα να μιμούνται τη πραγματικότητα, ή και να προσομοιώνουν μια πραγματική κατάσταση. Παιχνίδια αυτής της κατηγορίας είναι παιχνίδια οδήγησης και πτήσης αεροπλάνων.

Εικόνα 19 – παιχνίδι: Flight Simulator

2.6 Sports Game

Τα αθλητικά παιχνίδια προσομοιώνουν αθλήματα όπως το γκολφ, ποδόσφαιρο, το μπάσκετ και το baseball. Οι αντίπαλοι παίχτες συνήθως ελέγχονται από τον υπολογιστή όμως μπορούν να έχουν χειρισμό από άνθρωπο.

Εικόνα 20 – παιχνίδι: FIFA 22

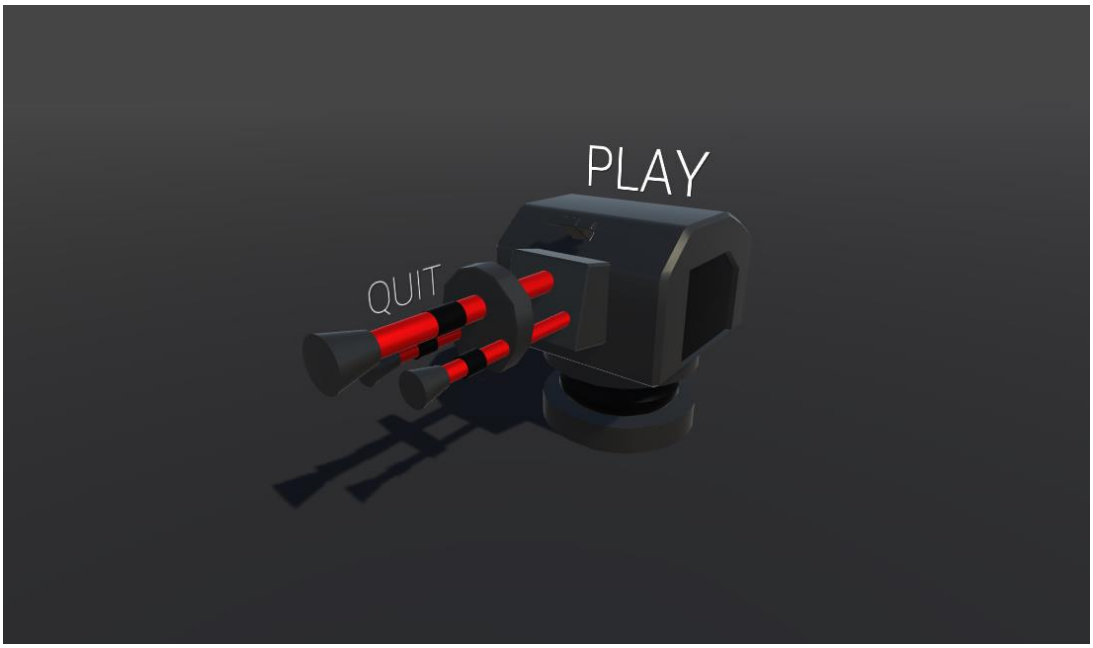
Αναφέρθηκαν μερικές από τις πιο μεγάλες κατηγορίες ηλεκτρονικών παιχνιδιών. Παρ’ όλα αυτά υπάρχουν αμέτρητες κατηγορίες και υποκατηγορίες ή συνδυασμοί αυτών, που δεν έχει νόημα να αναφερθούν αναλυτικά στο παρόν έγγραφο

3 Tower Defense

Στο κεφάλαιο αυτό γίνεται αναλυτική περιγραφή του παιχνιδιού της πτυχιακής εργασίας. Όνομα του παιχνιδιού είναι *«Tower Defense»*, προφανώς βασίζεται στον τύπο του παιχνιδιού. Πρόκειται για ένα απλό παιχνίδι χωρίς πολλά και σύνθετα γραφικά, καθώς κύριος σκοπός ήταν η εξοικείωση με τη μηχανή δημιουργίας παιχνιδιών Unity και την αντικειμενοστρεφή γλώσσα C#.

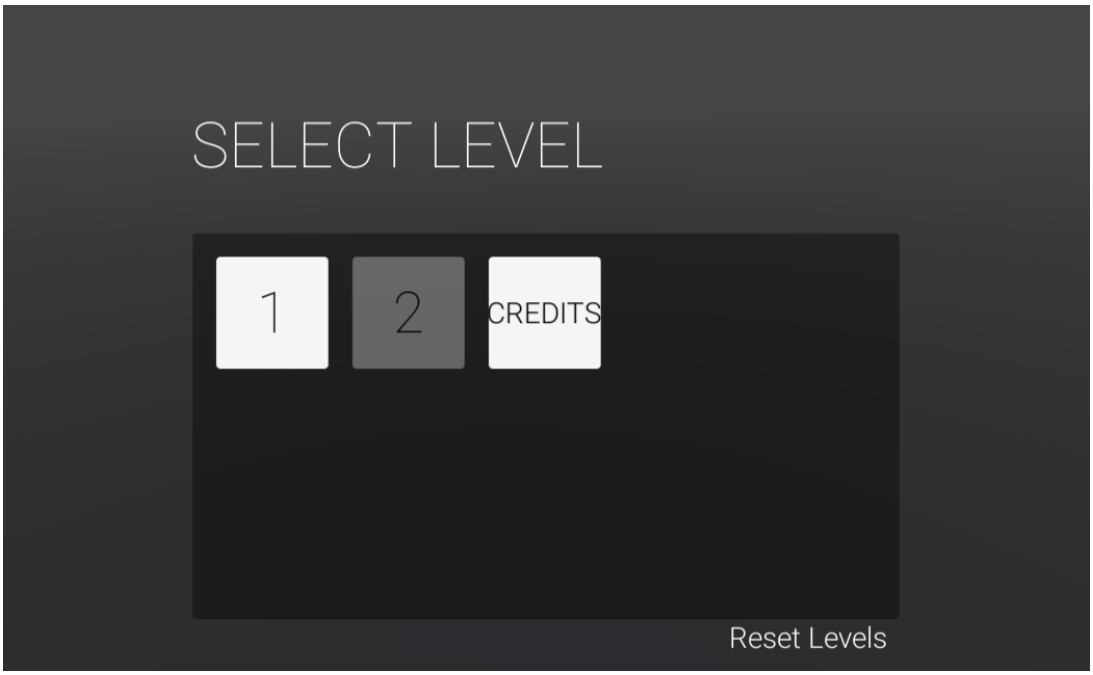
3.1 Περιγραφή

Ο χρήστης μπαίνοντας στο παιχνίδι βλέπει την αρχική οθόνη (Εικόνα 21). Στην αρχική οθόνη φαίνεται η επιλογή να ξεκινήσει και η επιλογή να κλείσει το παιχνίδι.



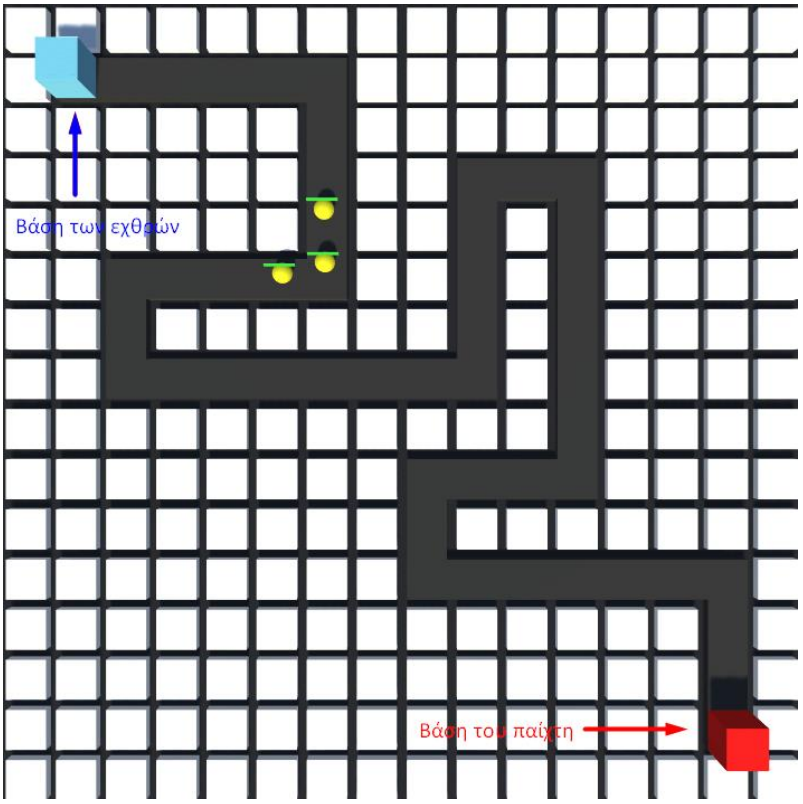
Εικόνα 21 – Αρχική σκηνή – Play και Quit

Ξεκινώντας λοιπόν, περνάει στην επόμενη οθόνη (Εικόνα 22), η οποία είναι το menu με τα επίπεδα του παιχνιδιού. Στο menu αυτό μπορεί να δει δύο επίπεδα που έχουν υλοποιηθεί, εκ των οποίων το 2ο είναι κλειδωμένο και ανοίγει τελειώνοντας το 1ο επίπεδο. Επίσης, ο παίχτης έχει τη δυνατότητα να κάνει Reset αυτά που έχει κάνει, δηλαδή αν έχει ξεκλειδώσει το 2ο επίπεδο μπορεί να το κάνει Reset και να είναι πάλι κλειδωμένο.



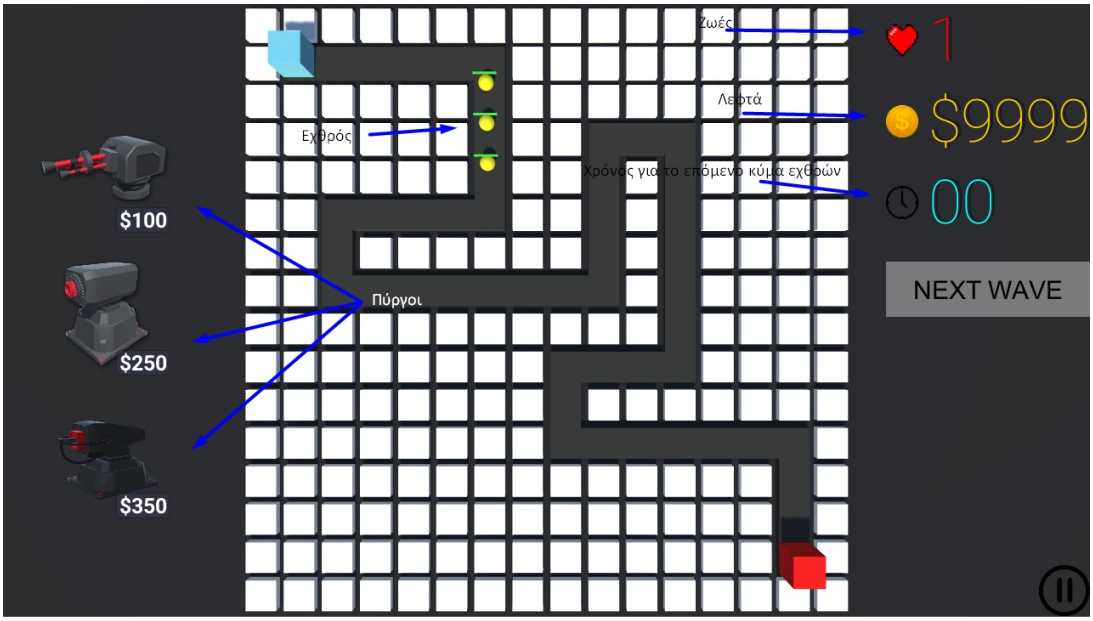
Εικόνα 22 – Select Level – Επίπεδο 1, Επίπεδο 2 (κλειδωμένο), Credits, Reset Levels

Επιλέγοντας τώρα το 1ο επίπεδο βλέπει μια διαδρομή και τα σημεία (μπλε και κόκκινο), ένα για κάθε άκρη της διαδρομής. Η βάση του παίχτη είναι το κόκκινο σημείο στο χάρτη, την οποία πρέπει να προστατέψει από τους εχθρούς που έρχονται από το μπλε σημείο (Εικόνα 23).



Εικόνα 23 – Φωτογραφία από το παιχνίδι, φαίνονται τα δύο σημεία

Για να καταφέρει να αντιμετωπίσει τους εχθρούς έχει στη διάθεσή του πύργους με πολυβόλα όπλα, πυραύλους και ακτίνες laser.

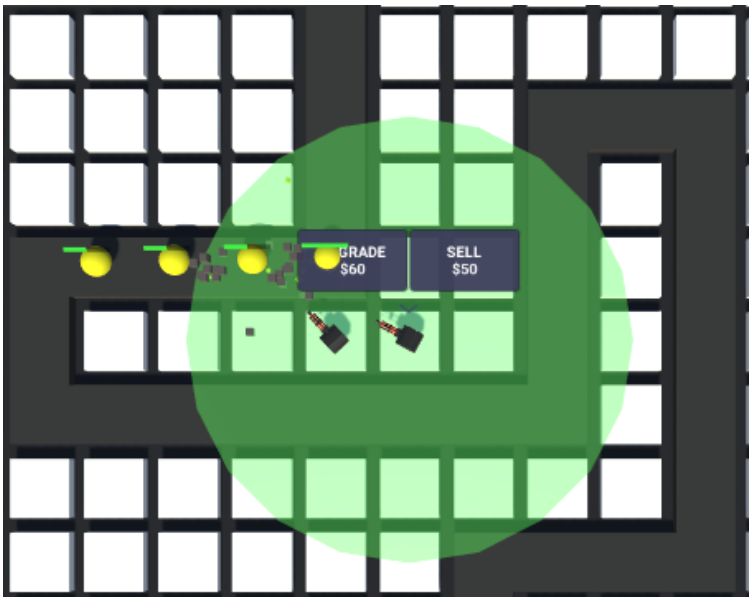


Εικόνα 24 – Φωτογραφία από το παιχνίδι, φαίνονται οι πύργοι, οι εχθροί, οι ζωές, τα λεφτά και ο χρόνος για το επόμενο κύμα εχθρών

Κάθε πύργος έχει διαφορετικό κόστος κατασκευής, το οποίο αφαιρείται από το «πορτοφόλι» του παίχτη. Τα άσπρα τετράγωνα είναι οι θέσεις που μπορεί να τοποθετήσει ο παίκτης τους πύργους. Επιλέγει έναν πύργο και, αν διαθέτει τα χρήματα για να τον χτίσει, το τετράγωνο που σημαδεύει αποκτά χρώμα πράσινο, ενώ σε αντίθετη περίπτωση, δηλαδή αν δεν έχει τα χρήματα, τότε το τετράγωνο γίνεται κόκκινο. Στην οθόνη (Εικόνα 24) μπορούμε να δούμε τους πύργους στα αριστερά, ενώ στα δεξιά βλέπουμε τα στατιστικά του παίχτη και το χρόνο για το επόμενο κύμα. Οι ζωές του παίχτη (η καρδιά με τον κόκκινο αριθμό δίπλα), αρχικά είναι 20 και μειώνονται με κάθε εχθρό που περνάει στη βάση του παίχτη. Το «πορτοφόλι» του παίχτη (το χρυσό νόμισμα με τους χρυσούς αριθμούς δίπλα του), τα χρήματα αυξάνονται με κάθε εξόντωση εχθρού. Ο χρόνος για το επόμενο κύμα εχθρών (το ρολόι με τους γαλάζιους αριθμούς δίπλα του), και τέλος, το κουμπί “NEXT WAVE”, για να επιταχύνεται η αντίστροφη μέτρηση του επόμενου κύματος.

3.2 Πύργοι

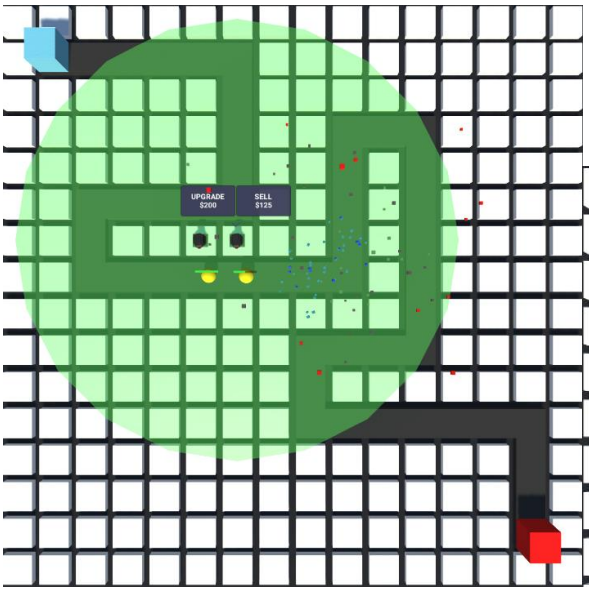
Οι πύργοι στοχεύουν με την κάνη τους προς τους εχθρούς, σημαδεύοντας τον κοντινότερο εχθρό προς αυτούς. Όταν οι εχθροί βρεθούν μέσα στην εμβέλεια των πύργων (Εικόνα 25), οι πύργοι πυροβολούν προς τους εχθρούς αυτόματα, με ταχύτητα και δύναμη που διαφέρει για κάθε τύπο πύργου.



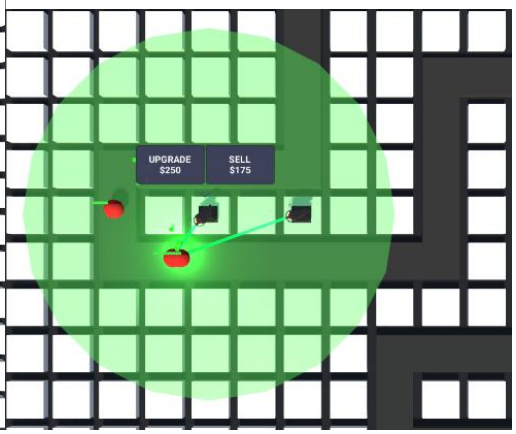
Εικόνα 25 – Πύργος πολυβόλο - Εμβέλεια του πύργου (πράσινος κύκλος)

Για να εμφανιστεί η εμβέλεια του πύργου, κάνουμε click πάνω στον πύργο και φαίνεται ο κύκλος με το πράσινο χρώμα. Κάθε πύργος διαθέτει μία αναβάθμιση, με την αναβάθμιση αυξάνεται η ταχύτητα βολής του πύργου καθώς και η ισχύς του. Με κάποιο χρηματικό ποσό που αναγράφεται πάνω στο κουμπί που λέει “Upgrade”, ο παίχτης μπορεί να αναβαθμίσει τον πύργο που έχει επιλέξει. Αναβαθμίζοντας τον πύργο παρατηρείται αλλαγή στα χρώματα του πύργου. Εκτός από το κουμπί του “Upgrade”, βλέπουμε και το κουμπί “Sell”, με αυτό το κουμπί ο παίχτης μπορεί να πουλήσει τον συγκεκριμένο πύργο και να του επιστραφεί ένα ποσό χρημάτων που προστίθεται στο πορτοφόλι του.

Υπάρχουν τρεις τύποι πύργων. Ο πρώτος πύργος, με τα πολυβόλα όπλα (Εικόνα 25), πυροβολεί μία σφαίρα το δευτερόλεπτο με ισχύ 50 πόντους ζημιάς (οι πόντοι ζημιάς αφαιρούνται από τη ζωή των εχθρών κατά το χτύπημά τους) και εμβέλεια 15 πόντους. Ο δεύτερος πύργος, με τους πυραύλους (Εικόνα 26), εκτοξεύει πυραύλους κάθε τέσσερα δευτερόλεπτα. Είναι από τους πιο αργούς πύργους, αλλά οι πύραυλοί του έχουν ένα ιδιαίτερο χαρακτηριστικό, δηλαδή κατά την πρόσκρουσή τους προκαλούν ζημιά και στους τριγύρω εχθρούς με ισχύ 50 πόντους ζημιάς. Διαθέτει 30 πόντους εμβέλειας. Τέλος, ο πύργος με το laser (Εικόνα 27), το laser κάνει ζημιά ανά δευτερόλεπτο με 20 πόντους ζημιάς, μπορεί όμως να επιβραδύνει την ταχύτητα του εχθρού κατά 50% της αρχικής του ταχύτητας. Διαθέτει 20 πόντους εμβέλειας.

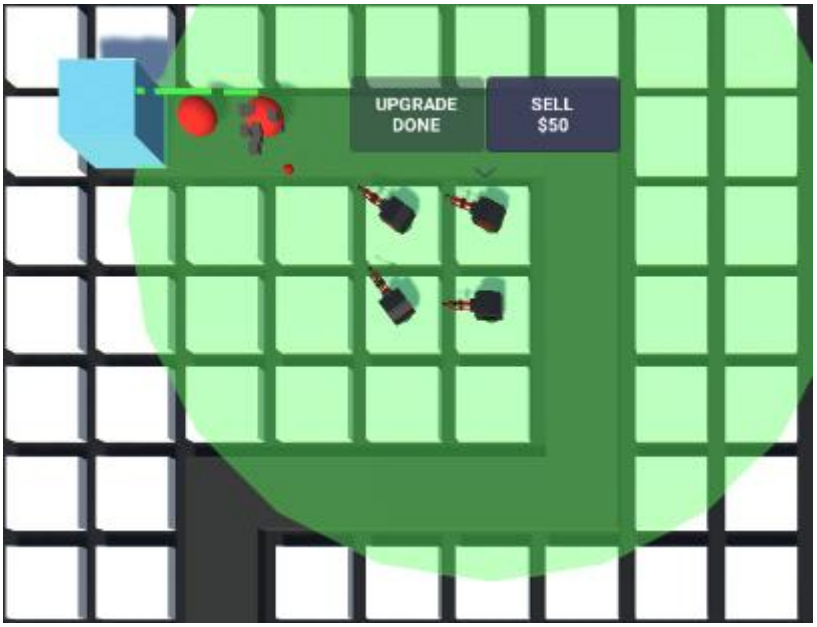


Εικόνα 26 – πύργος πυραύλων

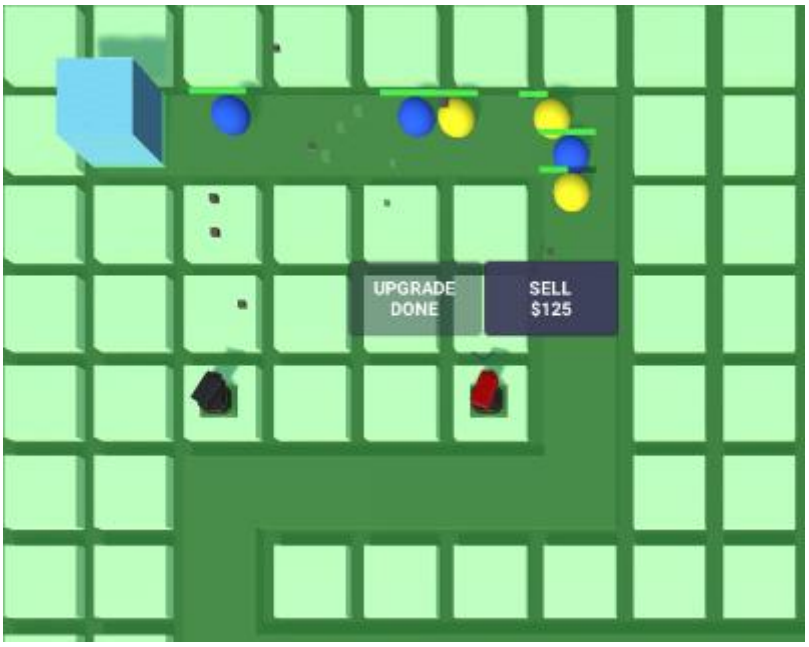


Εικόνα 27 – πύργος laser

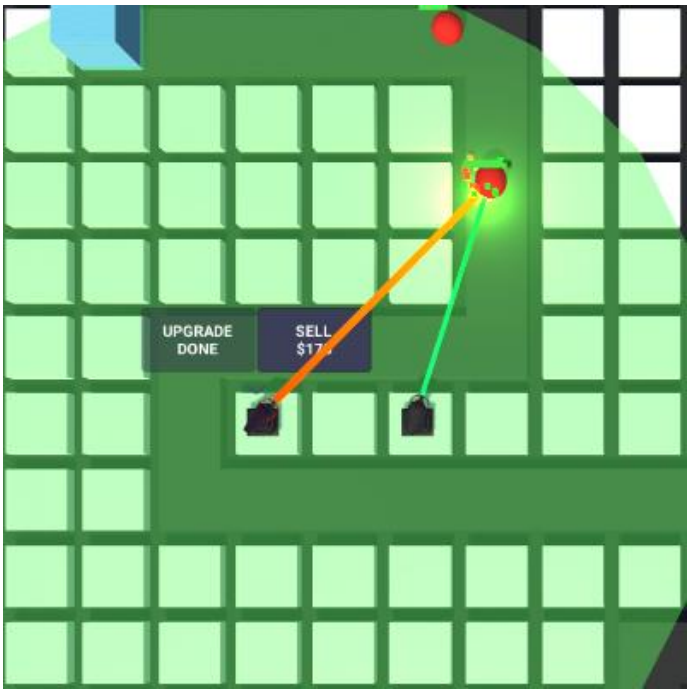
Όπως ήδη αναφέρθηκε, κάθε πύργος έχει από μία αναβάθμιση, με την οποία αυξάνεται η ταχύτητα βολής, η ισχύς και η εμβέλεια. Για τον πύργο με τα πολυβόλα (Εικόνα 25), η εμβέλεια αυξάνεται από 15 πόντους σε 20, η ταχύτητα βολής από 1 σφαίρα το δευτερόλεπτο κατά προσέγγιση 2 σφαίρες το δευτερόλεπτο οι και πόντοι ζημιάς από 50 σε 70. Επίσης, αλλάζει το χρώμα στο πλάι του πύργου σε κόκκινη απόχρωση. Για τον πύργο με τους πυραύλους (Εικόνα 26), εμβέλεια από 30 σε 40, ταχύτητα βολής από κάθε 4 δευτερόλεπτα σε κάθε 2 δευτερόλεπτα, ενώ οι πόντοι ζημιάς παραμένουν οι ίδιοι. Παρατηρούμε ότι η κεφαλή του πύργου αλλάζει σε κόκκινο χρώμα. Τέλος, για τους πύργους laser (Εικόνα 27), εμβέλεια από 20 σε 30, ζημιά ανά δευτερόλεπτο από 20 σε 40 και επιβράδυνση από 50% σε 80%. Επίσης αλλάζει το χρώμα του.



Εικόνα 28 – Αναβάθμιση πύργου με τα πολυβόλα



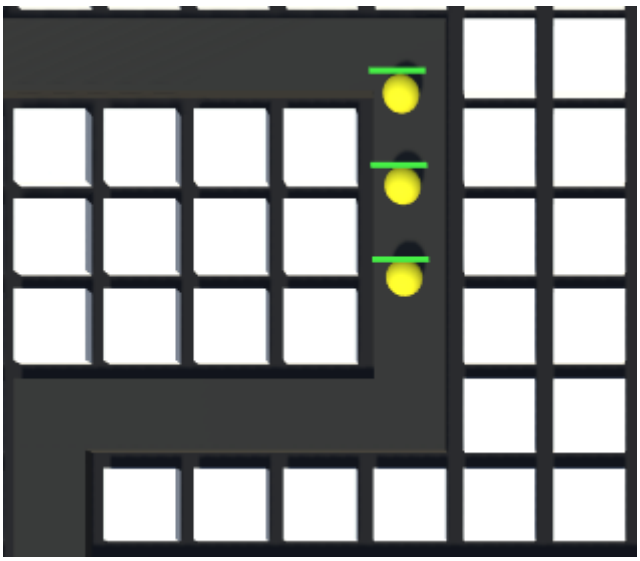
Εικόνα 29 – Αναβάθμιση πύργου με πυραύλους



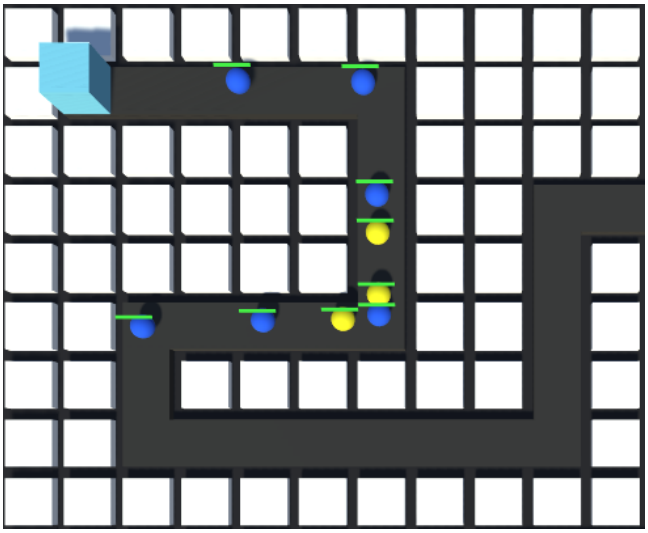
Εικόνα 30 – Αναβάθμιση πύργου laser

3.3 Εχθροί

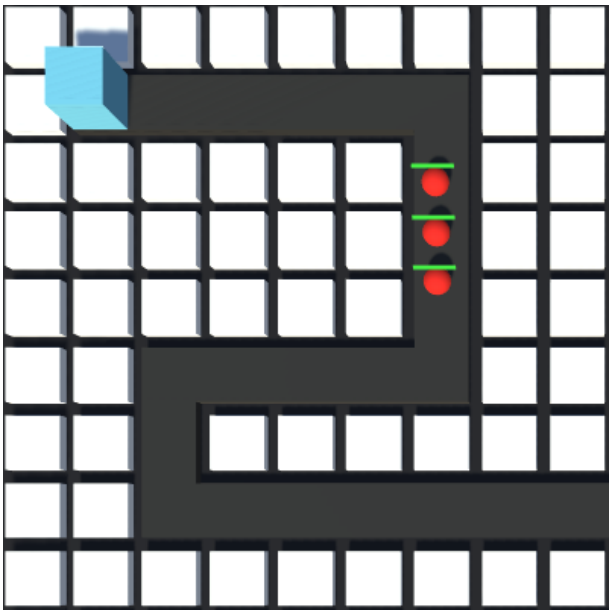
Το παιχνίδι διαθέτει τρεις τύπους εχθρών. Κάθε εχθρός διαφέρει σε χρώμα, ταχύτητα, πόντους ζωής. Όταν εξοντώνεται ένας εχθρός, ένα ποσό χρημάτων (διαφορετικό για κάθε εχθρό) προστίθεται στο «πορτοφόλι» του παίχτη. Ο κίτρινος (Εικόνα 31), έχει 10 πόντους ταχύτητας, 100 πόντους ζωής και 50 χρηματική αξία. Ο μπλε είναι πιο γρήγορος αλλά με λιγότερη ζωή (Εικόνα 32). Έχει 20 πόντους ταχύτητας, 30 πόντους ζωής και 25 χρηματική αξία. Τέλος, υπάρχει και ο κόκκινος που είναι ο πιο αργός σε σχέση με τους υπόλοιπους αλλά με τη μεγαλύτερη ζωή (Εικόνα 33). Έχει 7 πόντους ταχύτητας, 350 πόντους ζωής και 150 χρηματική αξία.



Εικόνα 31 – Κίτρινος εχθρός



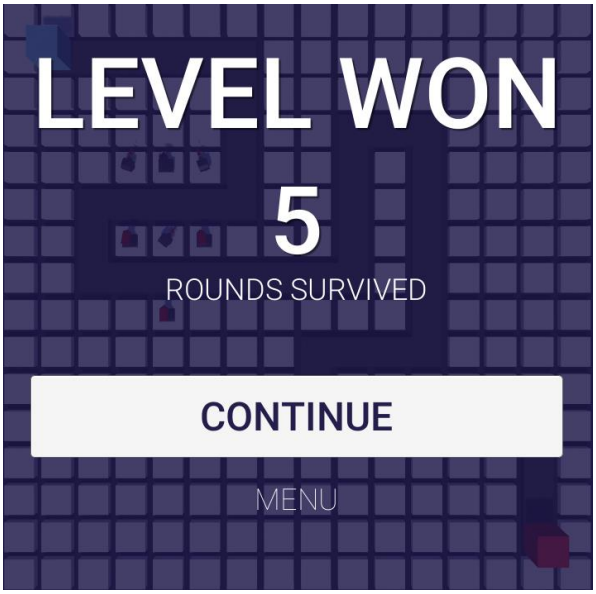
Εικόνα 32 – Μπλε εχθρός



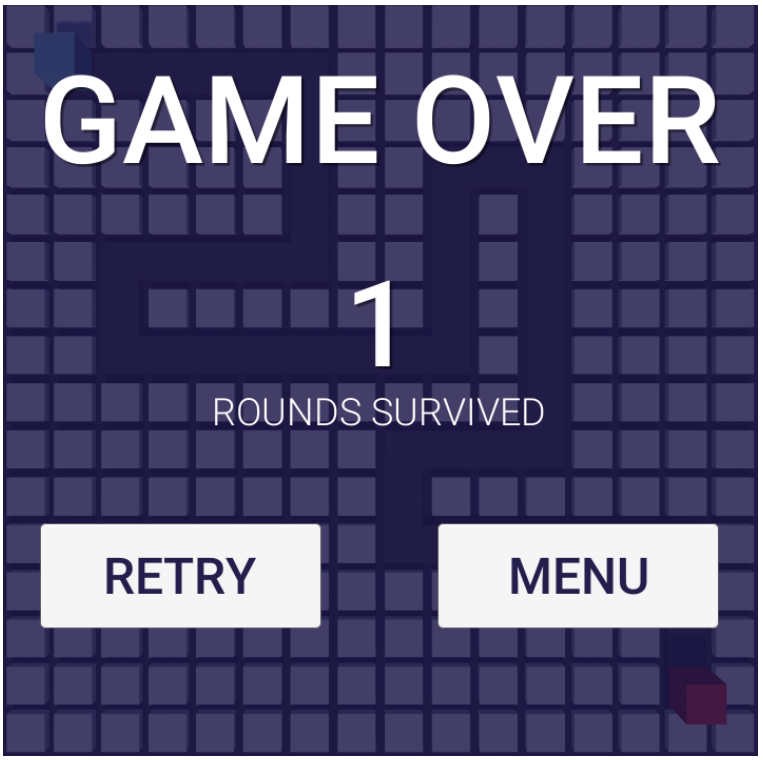
Εικόνα 33 – Κόκκινος εχθρός

3.4 Τέλος παιχνιδιού

Αν ο παίχτης καταφέρει να αντιμετωπίσει όλους τους εχθρούς με επιτυχία τότε περνάει στο επόμενο επίπεδο (ή μπορεί να επιστρέψει στο αρχικό μενού τελειώνοντας το παιχνίδι) (Εικόνα 34). Σε περίπτωση όμως που δεν καταφέρει να τους εξοντώσει και φτάσουν στη βάση, τότε ο παίχτης χάνει μια ζωή (φαίνονται ως καρδιές στην Εικόνα 24) για κάθε εχθρό που περνάει. Όταν τελειώσουν οι ζωές του χάνει και έχει τη δυνατότητα να παίξει πάλι το ίδιο επίπεδο από την αρχή ή να αποχωρήσει στο αρχικό menu (Εικόνα 35).



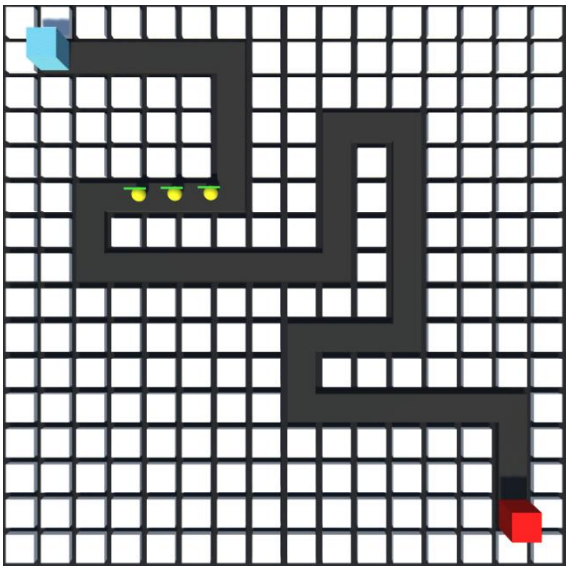
Εικόνα 34 – Ο παίχτης κέρδισε και επιβίωσε 5 γύρους, Continue για να συνεχίσει στο επόμενο επίπεδο ή Menu για να πάει στο αρχικό menu



Εικόνα 35 – Ο παίχτης έχασε και επιβίωσε μόνο 1 γύρο, Retry για να δοκιμάσει από την αρχή το επίπεδο, Menu για να πάει στο αρχικό menu

3.5 Επίπεδα

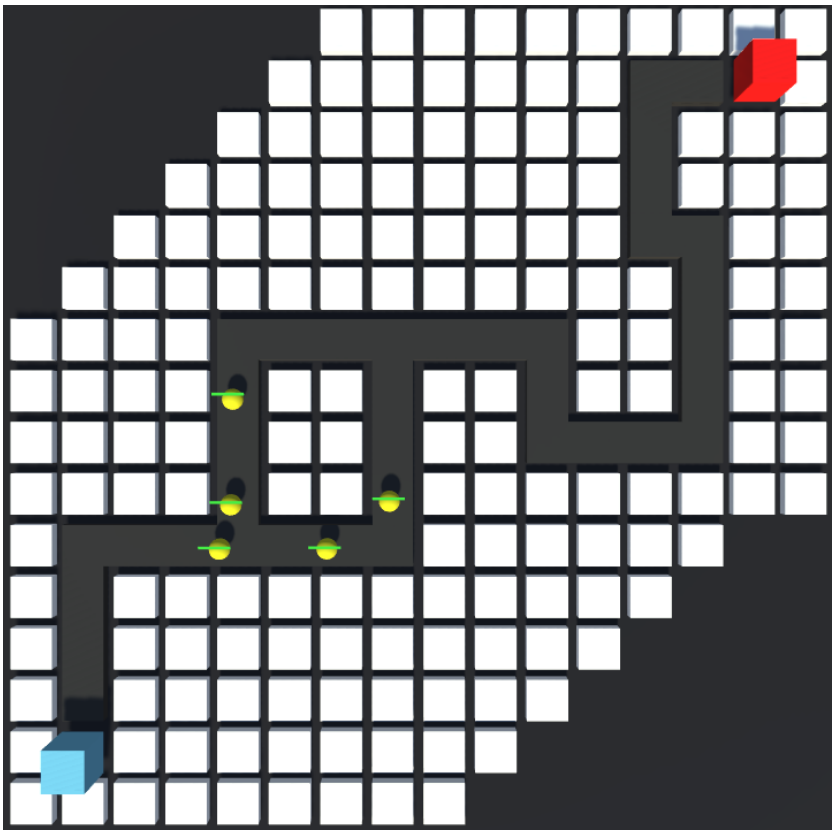
Στο μενού του παιχνιδιού βλέπουμε δύο επίπεδα. Στο πρώτο επίπεδο υπάρχει μια μονάχα διαδρομή από την βάση των εχθρών προς τη βάση του παίχτη (Εικόνα 36).



Εικόνα 36 – 1ο επίπεδο

Αν ο παίχτης καταφέρει να κερδίσει το 1ο επίπεδο μπορεί να συνεχίσει στο 2ο (Εικόνα 34). Συνεχίζοντας, πάλι ξεκινάει το επίπεδο με το ίδιο ποσό χρημάτων όπως και στο πρώτο επίπεδο, δηλαδή 400 χρυσά, καθώς και τις ίδιες ζωές, 20.

Στο 2ο επίπεδο (Εικόνα 37) υπάρχει μια ιδιαιτερότητα, η διαδρομή έχει μια διακλάδωση και οι εχθροί χωρίζονται (τυχαία) στην διακλάδωση και τέλος κατευθύνονται πάλι προς τη βάση του παίχτη.

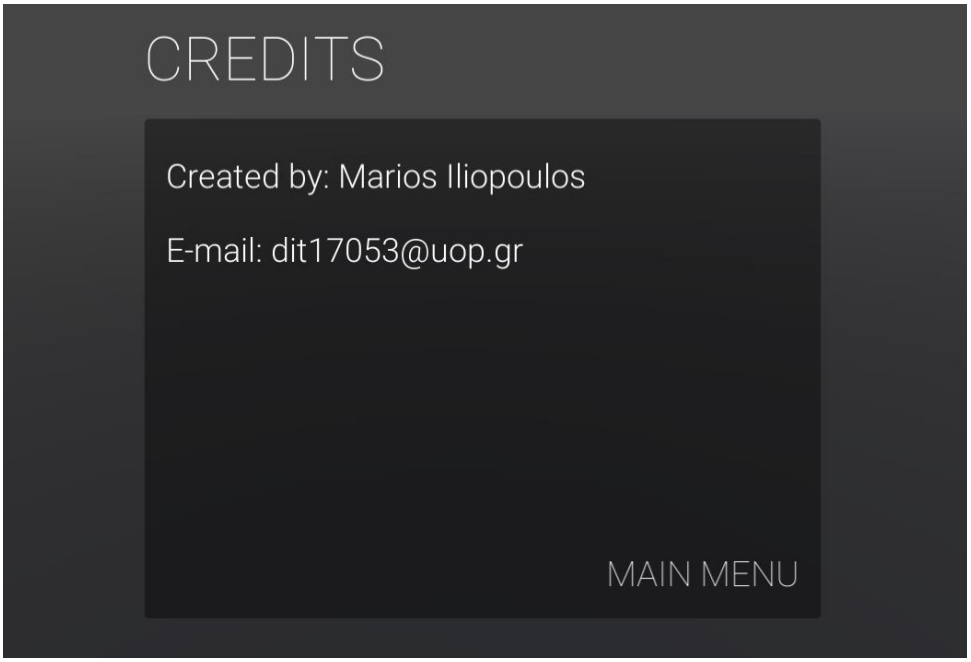


Εικόνα 37 – 2ο επίπεδο με διακλάδωση

Όταν καταφέρει να νικήσει και το 2ο επίπεδο, εμφανίζονται στην οθόνη οι γύροι που επιβίωσε καθώς και οι επιλογές να δει τα Credits, δηλαδή ποιος εργάστηκε για το παιχνίδι, και να επιστρέψει στο αρχικό μενού (Εικόνα 38).

Εικόνα 38 – Τερματισμός 2ου επιπέδου, Credits και Menu

Αν πατήσει το κουμπί "Credits", μεταφέρεται σε επόμενη οθόνη και βλέπει ποιος δημιούργησε το παιχνίδι (Εικόνα 39).



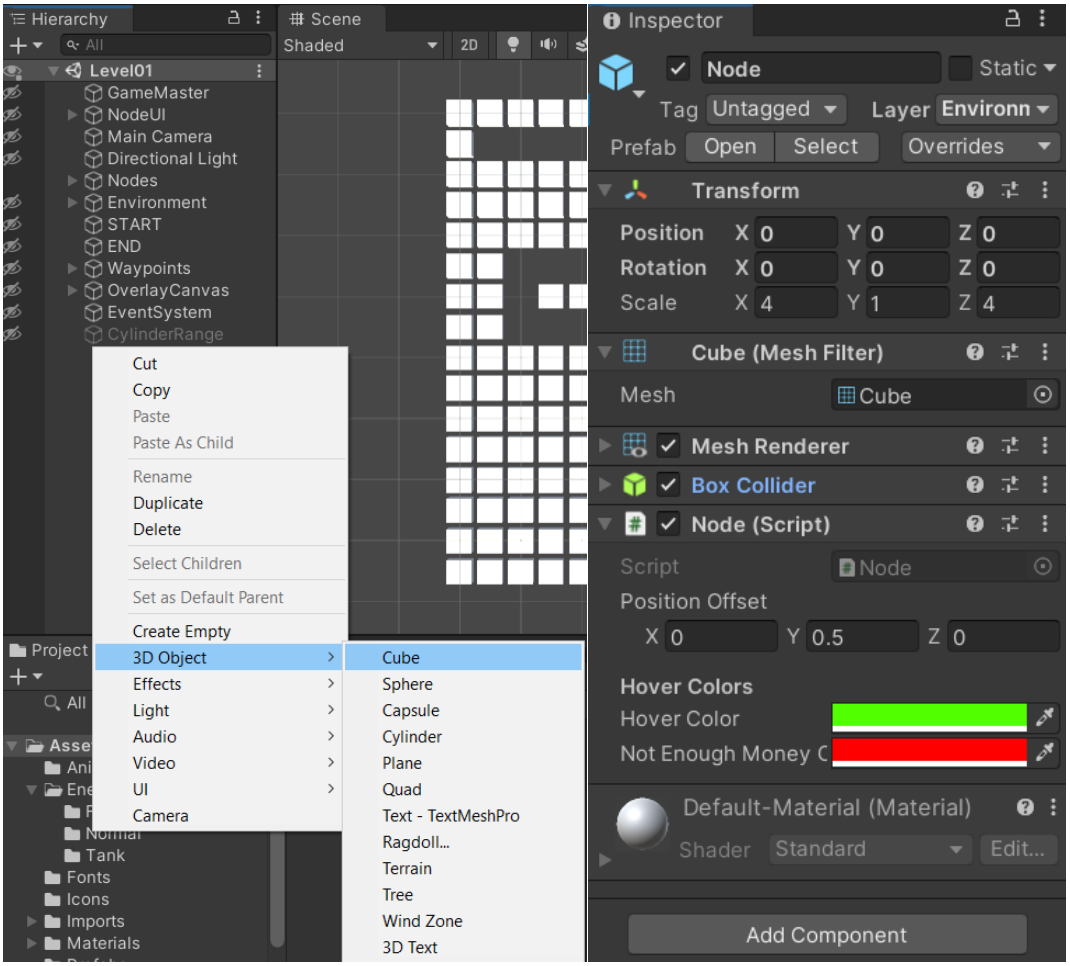
Εικόνα 39 - Credits

4 Δημιουργία του παιχνιδιού

Στην ενότητα αυτή γίνεται αναφορά στα περισσότερα από τα αντικείμενα που χρησιμοποιούν scripts και κώδικες για την λειτουργία του παιχνιδιού. Τέτοια αντικείμενα είναι το έδαφος που τοποθετούνται οι πύργοι, οι πύργοι, και οι εχθροί.

4.1 Έδαφος-Περιβάλλον

Για τα άσπρα κουτάκια, όπου πάνω σε αυτά τοποθετούνται οι πύργοι, (Εικόνα 40) στο Unity στο πεδίο που λέει ***Hierarchy*** βλέπει κανείς το αντικείμενο *Nodes*. Είναι μια ομάδα αντικειμένων που αποτελούνται από κύβους και ονομάζονται Node. Είναι ένας κύβος με κλίμακα στους άξονες (x, y, z) = (4, 1, 4). Αρχικά φτιάχνουμε ένα τέτοιο τετράγωνο και κάνουμε copy-paste όσα θέλουμε για να φτιάξουμε το επίπεδο. Για το πρώτο επίπεδο χρειαστήκαμε 256 τετράγωνα. Για να φτιαχτεί η διαδρομή χρησιμοποιούνται τα ίδια τετράγωνα, με τη διαφορά όμως ότι αλλάζει το scale, δηλαδή το μέγεθος. Είναι προτιμότερο να χρησιμοποιηθεί ένα μόνο παραλληλόγραμμο κατάλληλου μεγέθους για τη δημιουργία του κάθε δρόμου (οριζόντιου ή κατακόρυφου), από το να χρησιμοποιούνται πολλά τετράγωνα μικρότερου μεγέθους. Επίσης, έχει αλλάξει και το χρώμα, σε μια γκρι-μαύρη απόχρωση. Τέλος, φτιάξαμε ένα μεγάλο παραλληλόγραμμο, τεράστιων διαστάσεων για να καλύψουμε το background, κι εδώ υπάρχει η ίδια απόχρωση με την διαδρομή.



Εικόνα 40 – Αριστερά φαίνεται το Hierarchy και δεξιά το Inspector του αντικειμένου που έχουμε επιλέξει

Το Script που έχει το Node είναι και το πιο βασικό καθώς αναλαμβάνει την ευθύνη για την τοποθέτηση του πύργου (συναρτήσεις GetBuildPosition, OnMouseDown, BuildTurret), την πώληση (κουμπί Sell όταν πατάμε πάνω στον πύργο) (συνάρτηση SellTurret), την αναβάθμιση (κουμπί Upgrade) (συνάρτηση UpgradeTurret) και την ένδειξη με μορφή χρωμάτων (κόκκινο ή πράσινο) αν ο παίχτης έχει λεφτά να αγοράσει και να τοποθετήσει τον πύργο ή όχι (συναρτήσεις OnMouseEnter, OnMouseExit).

```
using UnityEngine;
using UnityEngine.EventSystems;
public class Node : MonoBehaviour
{

    public Vector3 positionOffset;

    [Header("Hover Colors")]
    public Color hoverColor;
    public Color notEnoughMoneyColor;

    [HideInInspector]
    public GameObject turret;
    [HideInInspector]
    public TurretBlueprint turretBlueprint;
    [HideInInspector]
    public bool isUpgraded = false;

    private Renderer rend;
    private Color startColor;

    BuildManager buildManager;

    private void Start()
    {
        rend = GetComponent<Renderer>();
        startColor = rend.material.color;

        buildManager = BuildManager.instance;
    }

    public Vector3 GetBuildPosition()
    {
        return transform.position + positionOffset;
    }

    private void OnMouseDown()
    {
        if (EventSystem.current.IsPointerOverGameObject())
        {
            return;
        }

        if (turret == null)
        {
            buildManager.DeselectNode();
        }
```

```
        if (turret != null)
        {
            buildManager.SelectNode(this);
            return;
        }

        if (!buildManager.CanBuild)
        {
            return;
        }

        BuildTurret(buildManager.GetTurretToBuild());
    }

    void BuildTurret(TurretBlueprint blueprint)
    {
        if (PlayerStats.Money < blueprint.cost)
        {
            Debug.Log("Not enough money");
            return;
        }

        PlayerStats.Money -= blueprint.cost;

        GameObject _turret = (GameObject)Instantiate(blueprint.prefab,
GetBuildPosition(), Quaternion.identity);
        turret = _turret;

        turretBlueprint = blueprint;

        GameObject effect = (GameObject)Instantiate(buildManager.buildEffect,
GetBuildPosition(), Quaternion.identity);
        Destroy(effect, 5f);

        Debug.Log("Turret built!");

        buildManager.DeselectTurret();
    }

    public void UpgradeTurret()
    {
        if (PlayerStats.Money < turretBlueprint.upgradeCost)
        {
            Debug.Log("Not enough money to upgrade");
            return;
        }

        PlayerStats.Money -= turretBlueprint.upgradeCost;

        //Destroy old turret.
        Destroy(turret);

        //Build a new one upgraded.
        GameObject _turret =
(GameObject)Instantiate(turretBlueprint.upgradedPrefab, GetBuildPosition(),
Quaternion.identity);
        turret = _turret;

        GameObject effect = (GameObject)Instantiate(buildManager.buildEffect,
GetBuildPosition(), Quaternion.identity);
        Destroy(effect, 5f);
```

```
        isUpgraded = true;

        Debug.Log("Turret upgraded!");
    }

    public void SellTurret()
    {
        PlayerStats.Money += turretBlueprint.GetSellAmount();
        Destroy(turret);
        turretBlueprint = null;
    }

    private void OnMouseEnter()
    {
        if (EventSystem.current.IsPointerOverGameObject())
        {
            return;
        }

        if (!buildManager.CanBuild)
        {
            return;
        }

        if (buildManager.HasMoney)
        {
            rend.material.color = hoverColor;
        }
        else
        {
            rend.material.color = notEnoughMoneyColor;
        }

    }

    private void OnMouseExit()
    {
        rend.material.color = startColor;
    }
}
```

Script 1 – Node.cs

4.2 Waypoints

Τα waypoints (Εικόνα 42) είναι αντικείμενα που δεν φαίνονται στο παιχνίδι, είναι βοηθητικά αντικείμενα για την κίνηση των εχθρών μέσα στη διαδρομή του επιπέδου. Τοποθετούνται σε κάθε γωνία της διαδρομής. Βρίσκονται ως κατευθυντήρια σημεία πάνω στο δρόμο για τους εχθρούς. Αποτελούν έναν πίνακα, ο οποίος χρησιμοποιείται από τους εχθρούς όπως θα δούμε και στην αντίστοιχη ενότητα. Παρακάτω παρουσιάζεται το script για τη δημιουργία του πίνακα.

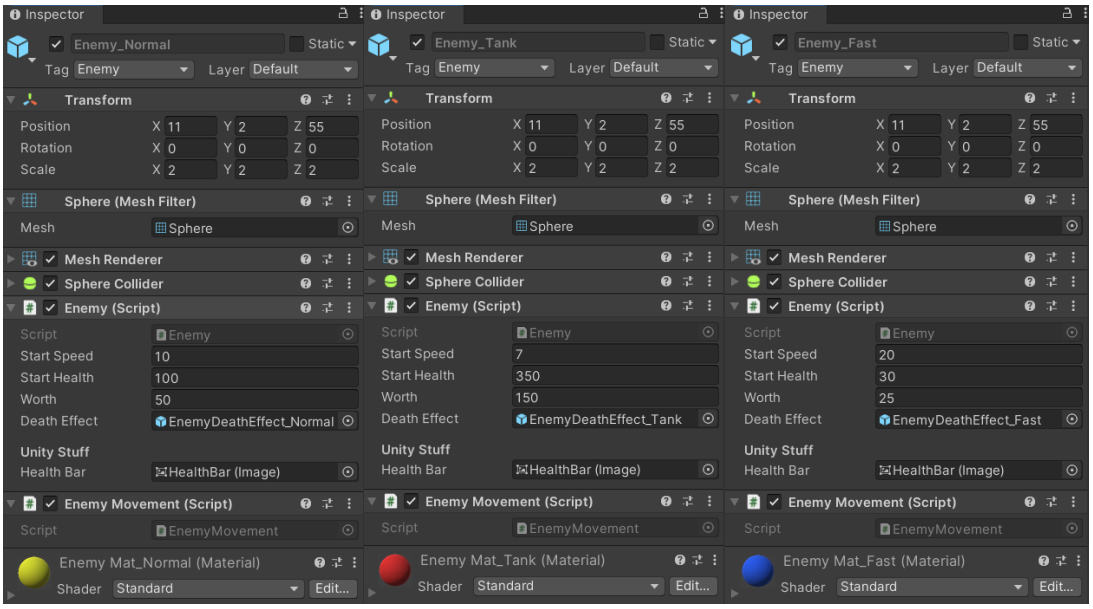
```
using UnityEngine;

public class Waypoints : MonoBehaviour
{
```

```
    public static Transform[] points;

    //public int branchPosition = 0;
    public int[] branchAmount;

    private void Awake()
    {
        points = new Transform[transform.childCount];
        for (int i = 0; i < points.Length; i++)
        {
            points[i] = transform.GetChild(i);
        }
    }
}
```

Script 2 – Waypoints.cs

4.3 Εχθροί

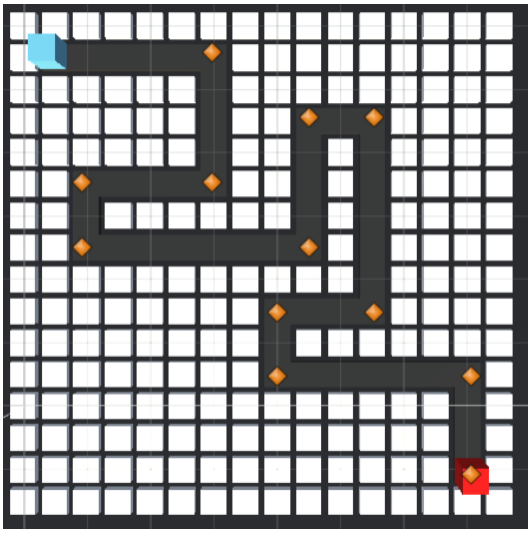
Το παιχνίδι διαθέτει τρεις διαφορετικούς τύπους εχθρών. Είναι σχήματα σφαίρας με διαφορετικά χρώματα ο καθένας. Όπως αναφέρθηκε, κάθε τύπος εχθρού έχει διαφορετική ταχύτητα, ζωή και αξία.



Εικόνα 41 – Εχθρός κίτρινος Normal, Εχθρός κόκκινος Tank, Εχθρός μπλε Fast

Οι εχθροί έχουν δύο script για να λειτουργήσουν, το script *Enemy* το οποίο έχει τα στατιστικά του, δηλαδή τη ζωή, την ταχύτητα και την αξία αλλά είναι υπεύθυνο και για τον υπολογισμό της ζημιάς ώστε να αφαιρείται από τη ζωή και τον έλεγχο αν έχει πεθάνει. Το δεύτερο script *Enemy Movement* είναι για την κίνησή του πάνω στη διαδρομή. Πάνω στη διαδρομή έχουν τοποθετηθεί σημεία, *Waypoints* (Εικόνα 42), και τα ακολουθεί ο εχθρός μέχρι το τέλος. Όταν φτάσει στο

τέλος, δηλαδή στη βάση του παίχτη, ο εχθρός «εξαφανίζεται» από το παιχνίδι και μειώνεται μια ζωή από τον παίχτη.



Εικόνα 42 – Τα πορτοκαλί σημεία είναι τα Waypoints

Στο script Enemy έχουμε τις πληροφορίες του εχθρού, δηλαδή τη ζωή, την ταχύτητα και τη χρηματική αξία. Ελέγχουμε την ζημιά που δέχεται από τους πύργους την οποία αφαιρούμε από την ζωή του (συνάρτηση TakeDamage), την επιβράδυνση από τον πύργο laser (συνάρτηση Slow). Τέλος, κάνουμε έλεγχο αν έχει εξοντωθεί με σκοπό να επιστρέφεται το ανάλογο ποσό χρημάτων στον παίχτη και η αφαίρεση του εχθρού από το παιχνίδι (συνάρτηση Die).

```
using UnityEngine;
using UnityEngine.UI;

public class Enemy : MonoBehaviour
{
    public float startSpeed = 10f;

    [HideInInspector]
    public float speed;

    public float startHealth = 100;
    private float health;

    public int worth = 50;

    public GameObject deathEffect;

    [Header("Unity Stuff")]
    public Image healthBar;

    private bool isDead = false;

    private void Start()
    {
        speed = startSpeed;
        health = startHealth;
    }
```

```
    public void TakeDamage(float amount)
    {
        health -= amount;
        healthBar.fillAmount = health / startHealth;
        if(health <= 0 && !isDead)
        {
            Die();
        }
    }

    public void Slow(float percent)
    {
        speed = startSpeed * (1f - percent);
    }

    void Die()
    {
        isDead = true;

        PlayerStats.Money += worth;

        GameObject effect = (GameObject)Instantiate(deathEffect,
transform.position, Quaternion.identity);
        Destroy(effect, 5f);

        Destroy(gameObject);
        WaveSpawner.EnemiesAlive--;

    }
}
```

Script 2 – Enemy.cs

Το script EnemyMovement είναι για την κίνηση του εχθρού πάνω στη διαδρομή του επιπέδου. Ο εχθρός έχει στόχο το waypoint στο οποίο πρέπει να κινηθεί, κάθε φορά που θα φτάνει στο συγκεκριμένο waypoint θα έχει επόμενο στόχο το επόμενο waypoint (συναρτήσεις Update, GetNextWaypoint). Στην περίπτωση της διακλάδωσης έχουμε τον παράγοντα της τύχης και ο εχθρός κινείται ανάλογα. Όταν φτάσει στο τέλος της διαδρομής, δηλαδή στη βάση του παίχτη, αφαιρείται μια ζωή από τον παίχτη και ο εχθρός εξαφανίζεται από το παιχνίδι (συνάρτηση EndPath).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

[RequireComponent(typeof(Enemy))]
public class EnemyMovement : MonoBehaviour
{
    private Waypoints waypointsPos;



    private Transform target;
    private int waypointIndex = 0;

    private Enemy enemy;
```

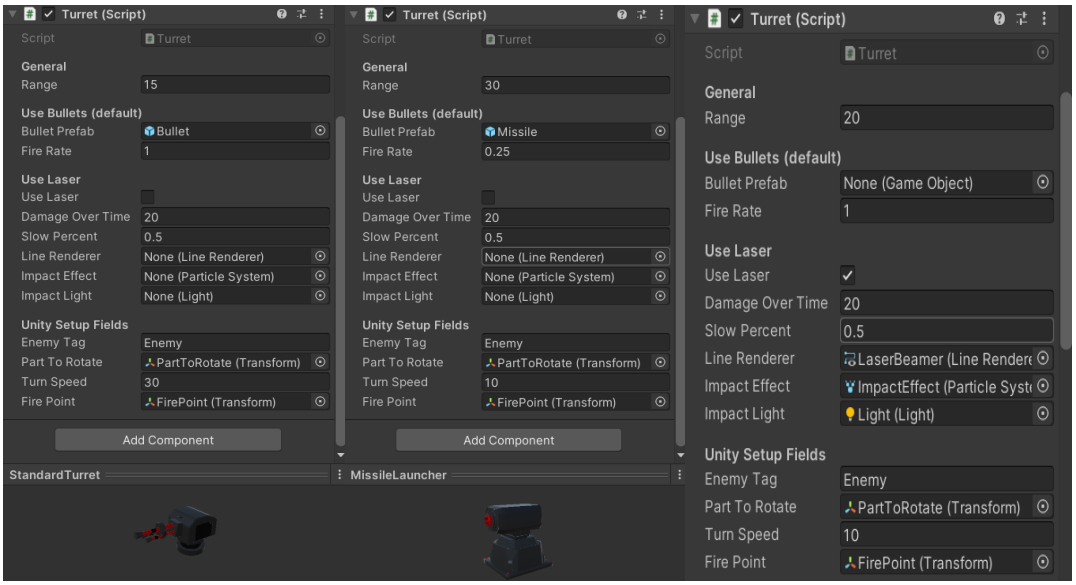
```
    private int counterBranch = 0;

    private void Start()
    {
        enemy = GetComponent<Enemy>();

        waypointsPos = FindObjectOfType<Waypoints>();

        target = Waypoints.points[0];
    }

    private void Update()
    {
        Vector3 dir = target.position - transform.position;
        transform.Translate(dir.normalized * enemy.speed * Time.deltaTime,
Space.World);

        if (Vector3.Distance(transform.position, target.position) <= 0.4f)
        {
            GetNextWaypoint();
        }

        enemy.speed = enemy.startSpeed;
    }

    void GetNextWaypoint()
    {
        if (waypointIndex >= Waypoints.points.Length - 1)
        {
            EndPath();
            return;
        }
        else
        {

            if (waypointsPos.branchAmount.Length!=0)
            {
                if (waypointIndex == waypointsPos.branchAmount[counterBranch])
                {
                    int x = Random.Range(0, 2);
                    if (x == 0)
                    {
                        waypointIndex++;
                        target = Waypoints.points[waypointIndex];
                        waypointIndex++;
                    }
                    else if (x == 1)
                    {
                        waypointIndex = waypointIndex + 2;
                        target = Waypoints.points[waypointIndex];
                    }
                    if (counterBranch < waypointsPos.branchAmount.Length-1)
                    {
                        counterBranch++;
                    }
                }
                else
                {
                    waypointIndex++;
                    target = Waypoints.points[waypointIndex];
                }
            }
```

```
            else
            {
                waypointIndex++;
                target = Waypoints.points[waypointIndex];
            }
        }
    }

    void EndPath()
    {
        PlayerStats.Lives--;
        PlayerStats.Lives = Mathf.Clamp(PlayerStats.Lives, 0, PlayerStats.Lives);

        Destroy(gameObject);
        WaveSpawner.EnemiesAlive--;
    }

}
```

Script 3 – EnemyMovement.cs

4.4 Πύργοι

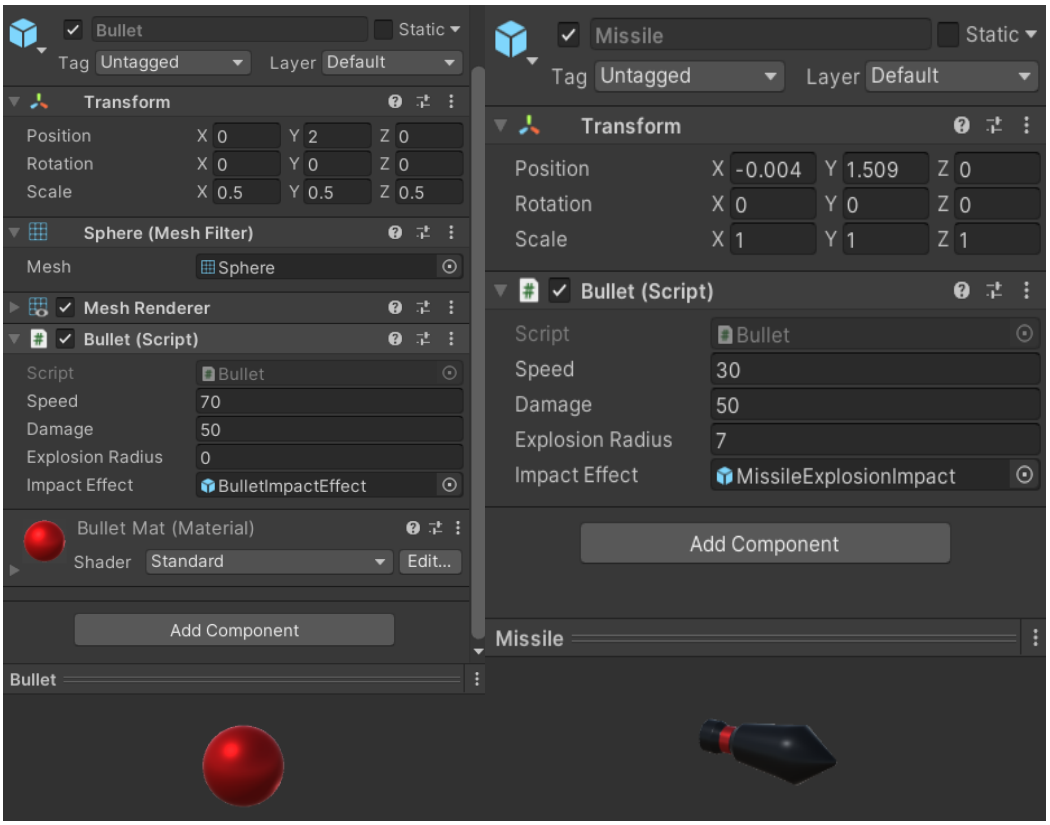
Υπάρχουν τρεις πύργοι, πύργος με πολυβόλα, Standard Turret. Ο πύργος τους με πυραύλους, Missile Launcher Turret. Και τέλος πύργος με ακτίνα laser, Laser Beamer Turret. Κάθε πύργος διαθέτει και μια αναβάθμιση που αυξάνει τα στατιστικά του. Η αναβάθμιση έχει ορατό αποτέλεσμα καθώς κάθε πύργος έχει και διαφορετικά χρώματα από ότι ήταν πριν. Τα μοντέλα των πύργων δημιουργήθηκαν με την εφαρμογή Blender, μια εφαρμογή που ειδικεύεται στη δημιουργία μοντέλων και γραφικών με κύρια χρήση τα ηλεκτρονικά παιχνίδια [8].



Εικόνα 43 – Standard Turret, Missile Launcher Turret, Laser Beamer Turret

Τα turrets χρησιμοποιούν το script *Turret* για να έχουν στατιστικά όπως την εμβέλεια, την ταχύτητα περιστροφής και την ταχύτητα βολής. Κάθε πύργος κλειδώνει πάνω στον κοντινότερο

εχθρό (συνάρτηση UpdateTarget, LockOnTarget) και σε λίγα δευτερόλεπτα πυροβολεί χρησιμοποιώντας την δικιά του σφαίρα (κάθε πύργος έχει διαφορετική σφαίρα) (συνάρτηση Shoot, Laser), οι σφαίρες έχουν δικό τους script *Bullet* και διαθέτουν ξεχωριστό ποσοστό ζημιάς.



Εικόνα 44 – Bullet για το standard turret και Missile για το Missile Launcher Turret

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Turret : MonoBehaviour
{
    private Transform target;
    private Enemy targetEnemy;


    [Header("General")]

    public float range = 15f;

    [Header("Use Bullets (default)")]
    public GameObject bulletPrefab;
    public float fireRate = 1f;
    private float fireCountdown = 0f;

    [Header("Use Laser")]
    public bool useLaser = false;

    public int damageOverTime = 20;
    public float slowPercent = .5f;
```

```
    public LineRenderer lineRenderer;
    public ParticleSystem impactEffect;
    public Light impactLight;

    [Header("Unity Setup Fields")]

    public string enemyTag = "Enemy";

    public Transform partToRotate;
    public float turnSpeed = 10f;


    public Transform firePoint;

    // Start is called before the first frame update
    void Start()
    {
        InvokeRepeating("UpdateTarget", 0f, 0.5f);
    }

    void UpdateTarget()
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);
        float shortestDistance = Mathf.Infinity;
        GameObject nearestEnemy = null;
        foreach (GameObject enemy in enemies)
        {
            float distanceToEnemy = Vector3.Distance(transform.position,
enemy.transform.position);
            if (distanceToEnemy < shortestDistance)
            {
                shortestDistance = distanceToEnemy;
                nearestEnemy = enemy;
            }
        }

        if (nearestEnemy != null && shortestDistance <= range)
        {
            target = nearestEnemy.transform;
            targetEnemy = nearestEnemy.GetComponent<Enemy>();
        }
        else
        {
            target = null;
        }

    }

    // Update is called once per frame
    void Update()
    {
        if (target == null)
        {
            if (useLaser)
            {
                if (lineRenderer.enabled)
                {
                    lineRenderer.enabled = false;
                    impactEffect.Stop();
                    impactLight.enabled = false;
                }
```

```
            }

            return;
        }
        //target
        LockOnTarget();

        if (useLaser)
        {
            Laser();
        }
        else
        {
            if (fireCountdown <= 0f)
            {
                Shoot();
                fireCountdown = 1f / fireRate;
            }

            fireCountdown -= Time.deltaTime;
        }


    }

    void LockOnTarget()
    {
        Vector3 dir = target.position - transform.position;
        Quaternion lookRotation = Quaternion.LookRotation(dir);
        Vector3 rotation = Quaternion.Lerp(partToRotate.rotation, lookRotation,
Time.deltaTime * turnSpeed).eulerAngles;
        partToRotate.rotation = Quaternion.Euler(0f, rotation.y, 0f);
    }

    void Laser()
    {
        targetEnemy.TakeDamage(damageOverTime * Time.deltaTime);
        targetEnemy.Slow(slowPercent);


        if (!lineRenderer.enabled)
        {
            lineRenderer.enabled = true;
            impactEffect.Play();
            impactLight.enabled = true;
        }

        lineRenderer.SetPosition(0, firePoint.position);
        lineRenderer.SetPosition(1, target.position);

        Vector3 dir = firePoint.position - target.position;

        impactEffect.transform.position = target.position + dir.normalized;

        impactEffect.transform.rotation = Quaternion.LookRotation(dir);


    }

    void Shoot()
    {
```

```
        GameObject bulletGO = (GameObject)Instantiate(bulletPrefab,
firePoint.position, firePoint.rotation);
        Bullet bullet = bulletGO.GetComponent<Bullet>();

        if (bullet != null)
        {
            bullet.Seek(target);
        }
    }

    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, range);
    }

}
```

Script 4 – Turret.cs

Το script Bullet για τις σφαίρες, χρησιμοποιείται για την αρχικοποίηση των πληροφοριών της κάθε σφαίρας. Η σφαίρα έχει σαν στόχο τον εχθρό και τον ακολουθεί μέχρι, είτε να τον χτυπήσει είτε να εξαφανιστεί ο εχθρός (συνάρτηση Update). Αν η σφαίρα χτυπήσει τον εχθρό τότε του προκαλεί ζημιά (συνάρτηση HitTarget, Damage). Ο πύραυλος έχει το ιδιαίτερο χαρακτηριστικό της έκρηξης, όταν πετύχει έναν εχθρό προκαλεί ζημιά και στους τριγύρω εχθρούς (συνάρτηση Explode).

```
using UnityEngine;

public class Bullet : MonoBehaviour
{
    private Transform target;

    public float speed = 70f;

    public int damage = 50;

    public float explosionRadius = 0f;
    public GameObject impactEffect;

    public void Seek(Transform _target)
    {
        target = _target;
    }

    // Update is called once per frame
    void Update()
    {
        if(target == null)
        {
            Destroy(gameObject);
            return;
        }

        Vector3 dir = target.position - transform.position;
        float distanceThisFrame = speed * Time.deltaTime;

        if (dir.magnitude <= distanceThisFrame)
        {
            HitTarget();
            return;
```

```
        }

        transform.Translate(dir.normalized * distanceThisFrame, Space.World);
        transform.LookAt(target);

    }

    void HitTarget()
    {
        GameObject effectsIns = (GameObject)Instantiate(impactEffect,
transform.position, transform.rotation);
        Destroy(effectsIns, 5f);

        if (explosionRadius > 0f)
        {
            Explode();
        } else
        {
            Damage(target);
        }



        Destroy(gameObject);
    }

    void Explode()
    {
        Collider[] colliders = Physics.OverlapSphere(transform.position,
explosionRadius);
        foreach (Collider collider in colliders)
        {
            if (collider.tag == "Enemy")
            {
                Damage(collider.transform);
            }
        }
    }

    void Damage(Transform enemy)
    {
        Enemy e = enemy.GetComponent<Enemy>();
        if (e != null)
        {
            e.TakeDamage(damage);
        }



    }

    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, explosionRadius);
    }

}
```

Script 5 – Bullet.cs

5 Επίλογος

Στην παρούσα αναφορά παρουσιάστηκε το παιχνίδι «Tower Defense». Αρχικά, αναφέρθηκαν οι μηχανές παραγωγής ηλεκτρονικών παιχνιδιών (Unity και Unreal Engine) καθώς και παιχνίδια που φτιάχτηκαν από αυτά τα game engines. Στη συνέχεια παρουσιάστηκαν οι κυριότεροι τύποι των ηλεκτρονικών παιχνιδιών και παραδείγματα αυτών. Κατόπιν, περιγράφηκε αναλυτικά το παιχνίδι *«Tower Defense»,* και δόθηκαν λεπτομέρειες για την ανάπτυξή του καθώς και οι πιο βασικοί κώδικες που δημιουργήθηκαν για τη λειτουργία του παιχνιδιού.

Χαρακτηριστικά που θα μπορούσαν να προστεθούν στο παιχνίδι αλλά λόγω χρόνου ή και γνώσεων δεν υλοποιήθηκαν.

- Στα ηλεκτρονικά παιχνίδια χρησιμοποιούνται ήχοι και μουσικές, που είναι απαραίτητα για την ολοκληρωμένη εμπειρία του χρήστη. Ένα καλό soundtrack βοηθάει τον παίχτη να συγκεντρωθεί στο παιχνίδι. Παρ’ όλα αυτά, για τη δημιουργία ενός soundtrack ή ακόμα και εφέ για του πύργους την ώρα βολής χρειάζεται κάποιο εξωτερικό πρόγραμμα, το οποίο δεν είναι εύκολο στη χρήση και κατά πλειοψηφία χρησιμοποιείται από ειδικούς του αντίστοιχου επαγγέλματος.

- Θα μπορούσε να υπάρχει είναι η δυνατότητα στον παίχτη να φτιάξει τη δικιά του πίστα, δεν υλοποιήθηκε λόγω χρόνου.

- Υπάρχουν μόνο δύο πίστες, καθώς χρησιμοποιούνται τα περισσότερα από τα χαρακτηριστικά που θα μπορούσαν να έχουν οι πίστες.

- Επιπλέον αναβαθμίσεις στους πύργους ή και ακόμα περισσότεροι τύποι πύργων.

6 Πηγές

[1] https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

[2] https://unity.com/our-company

[3] https://unity.com/madewith

[4] https://en.wikipedia.org/wiki/Unreal_Engine

[5] https://www.unrealengine.com/en-US/

[6] https://www.idtech.com/blog/different-types-of-video-game-genres

[7] https://en.wikipedia.org/wiki/Unity_(game_engine)

[Εικόνα 2] https://gr.ign.com/hearthstone-heroes-of-warcraft/978/review/hearthstone-heroes-of-warcraft-review

[Εικόνα 3] https://en.wikipedia.org/wiki/Temple_Run

[Εικόνα 4] https://en.wikipedia.org/wiki/Pok%C3%A9mon_Go

[Εικόνα 5] https://en.wikipedia.org/wiki/Among_Us

[Εικόνα 6] https://en.wikipedia.org/wiki/Unreal_Tournament

[Εικόνα 7] https://en.wikipedia.org/wiki/BioShock

[Εικόνα 8] https://en.wikipedia.org/wiki/Batman:_Arkham_Asylum

[Εικόνα 9] https://en.wikipedia.org/wiki/Tekken_7

[Εικόνα 10] https://en.wikipedia.org/wiki/Call_of_Duty:_Black_Ops
https://en.wikipedia.org/wiki/Gears_of_War_3

[Εικόνα 11] https://streetfighter.com/en/

[Εικόνα 12] https://store.epicgames.com/en-US/p/god-of-war

[Εικόνα 13] https://en.wikipedia.org/wiki/The_Whispered_World

[Εικόνα 14] https://en.wikipedia.org/wiki/Under_a_Killing_Moon

[Εικόνα 15] https://www.playstation.com/el-gr/games/diablo-iii-reaper-of-souls-ultimate-evil-edition/
[Εικόνα 16] https://starcraft2.com/en-us/game

[Εικόνα 17] https://www.leagueoflegends.com/el-gr/

[Εικόνα 18] https://gr.ign.com/plants-vs-zombies/

[Εικόνα 19] https://store.steampowered.com/app/1250410/Microsoft_Flight_Simulator_Game_of_the_Year_Edition/

[Εικόνα 20] https://store.steampowered.com/app/1506830/FIFA_22/

[8] Οι πύργοι είναι δωρεάν και βρίσκονται στην ιστοσελίδα https://devassets.com/assets/tower-defense-assets/
Δημιουργήθηκαν με το εργαλείο Blender.

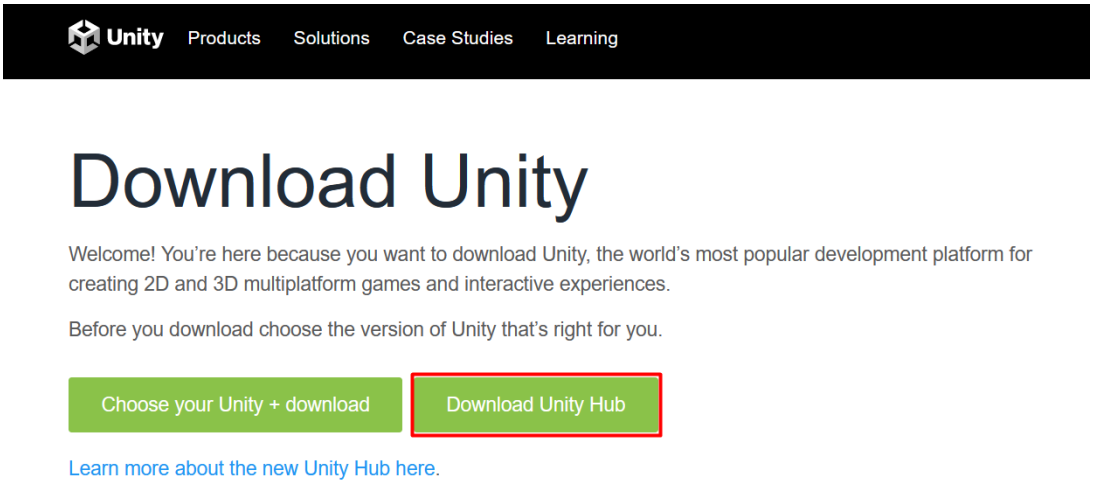
Tutorials και βοήθεια από YouTube από το κανάλι https://www.youtube.com/c/Brackeys
Και από την επίσημη ιστοσελίδα του Unity https://docs.unity3d.com/Manual/index.html

Επίσης χρησιμοποιήθηκε το εργαλείο GitHub για την αποθήκευση του project https://github.com/vroumsvroums/TowerDefence

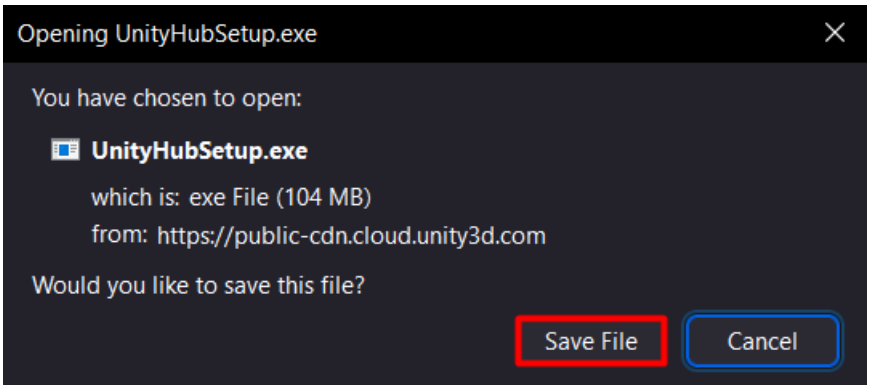
Παράρτημα: Οδηγός εγκατάστασης Unity

Παρακάτω παρουσιάζονται όλα τα βήματα για την εγκατάσταση του προγράμματος Unity.

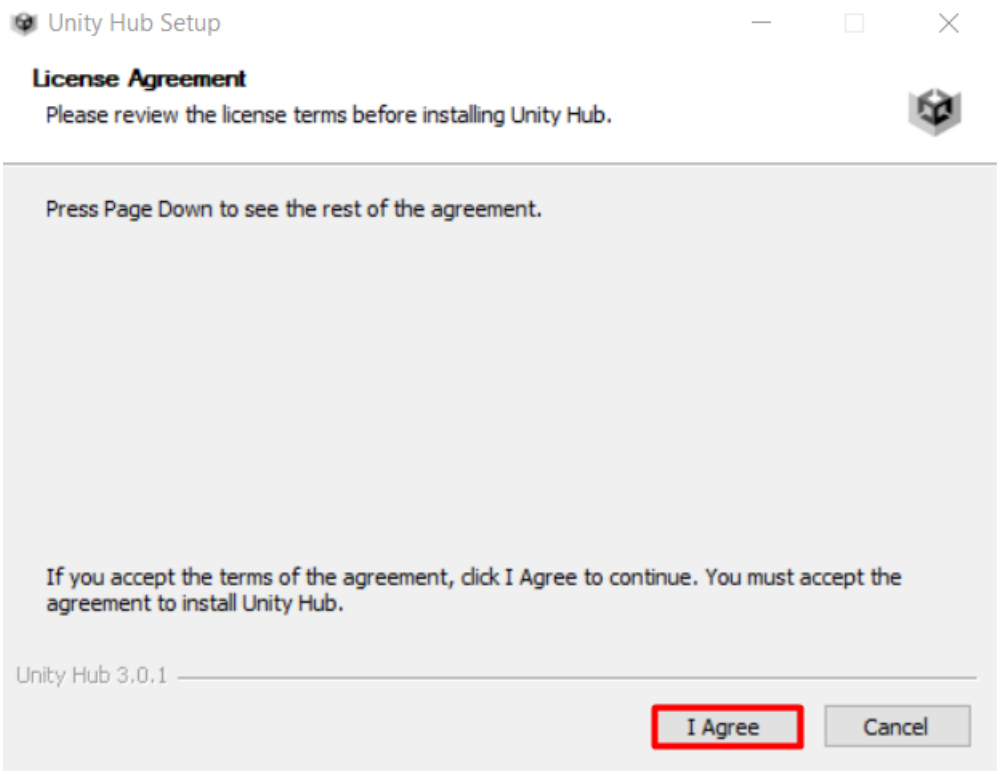
Αρχικά κάνουμε αναζήτηση στο Internet τη σελίδα για να κατεβάσουμε το Unity, https://unity3d.com/get-unity/download



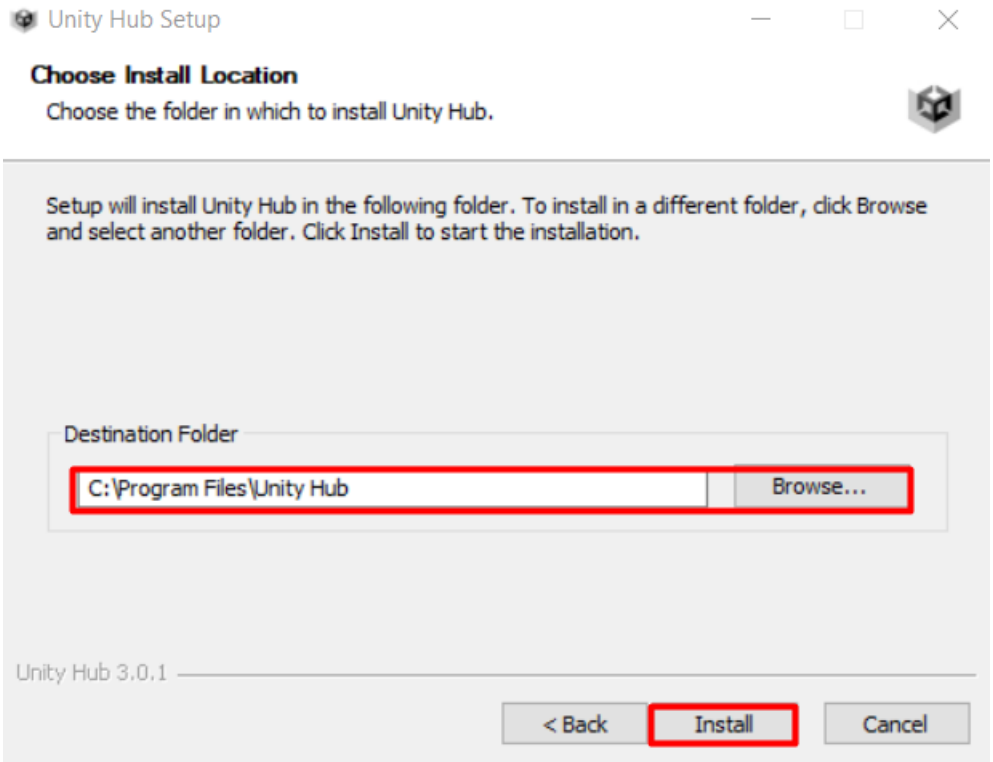
Επιλέγουμε την επιλογή *"Download Unity Hub"* και κατεβάζουμε το αρχείο



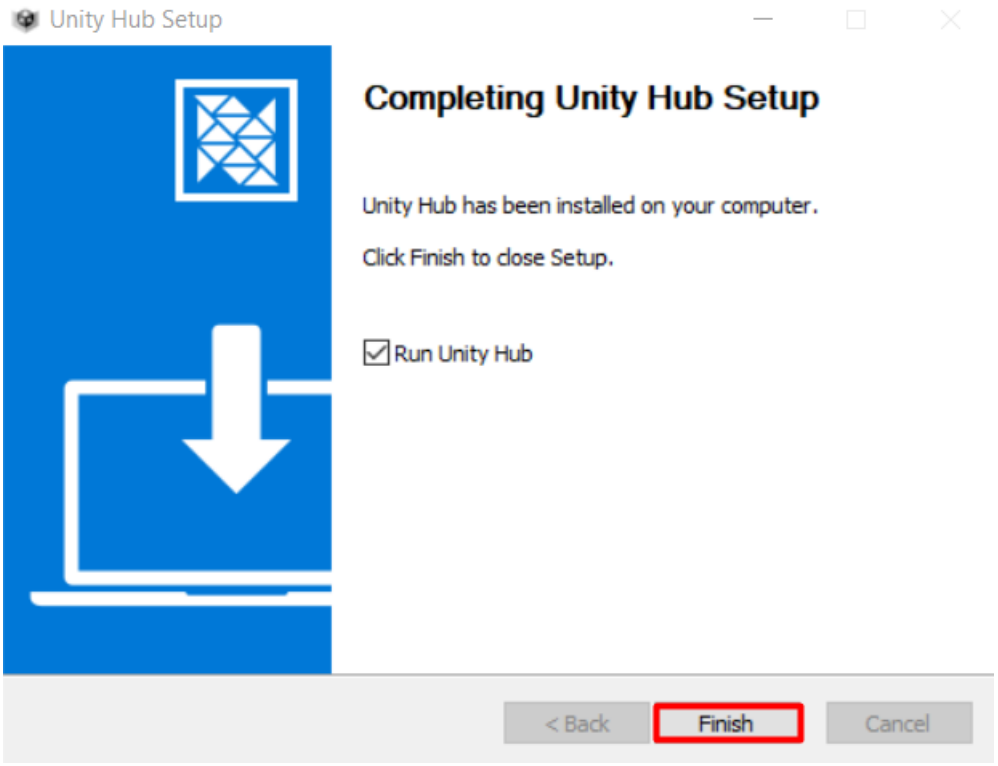
Το αποθηκεύουμε και μόλις ολοκληρωθεί η λήψη ανοίγουμε το exe.



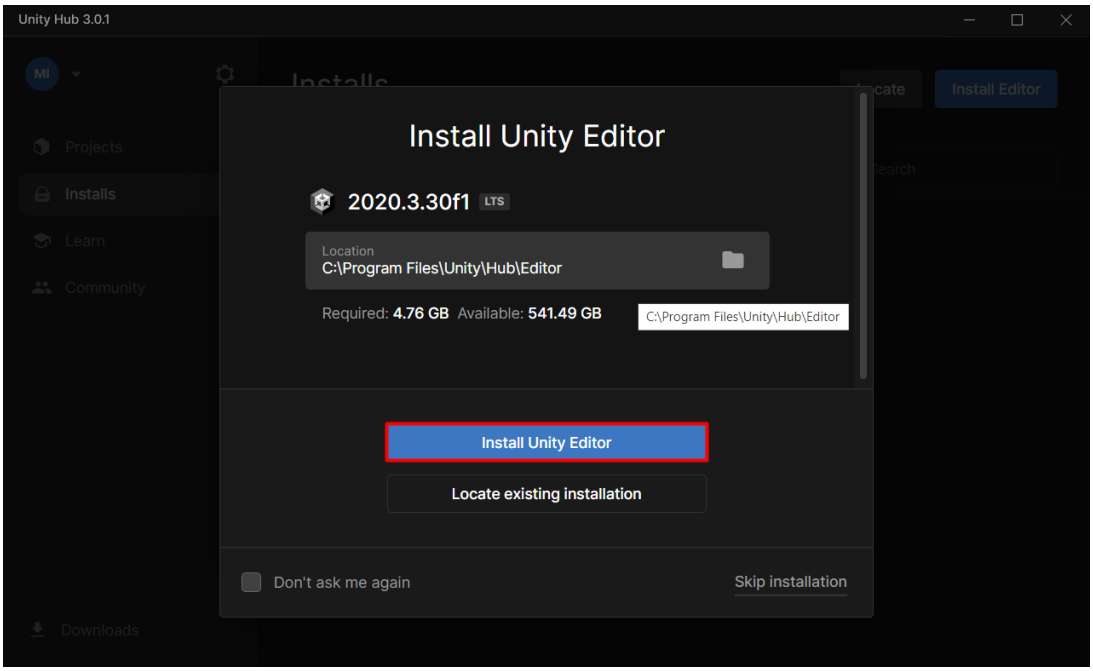
Πατάμε *"I Agree"* και συνεχίζουμε,



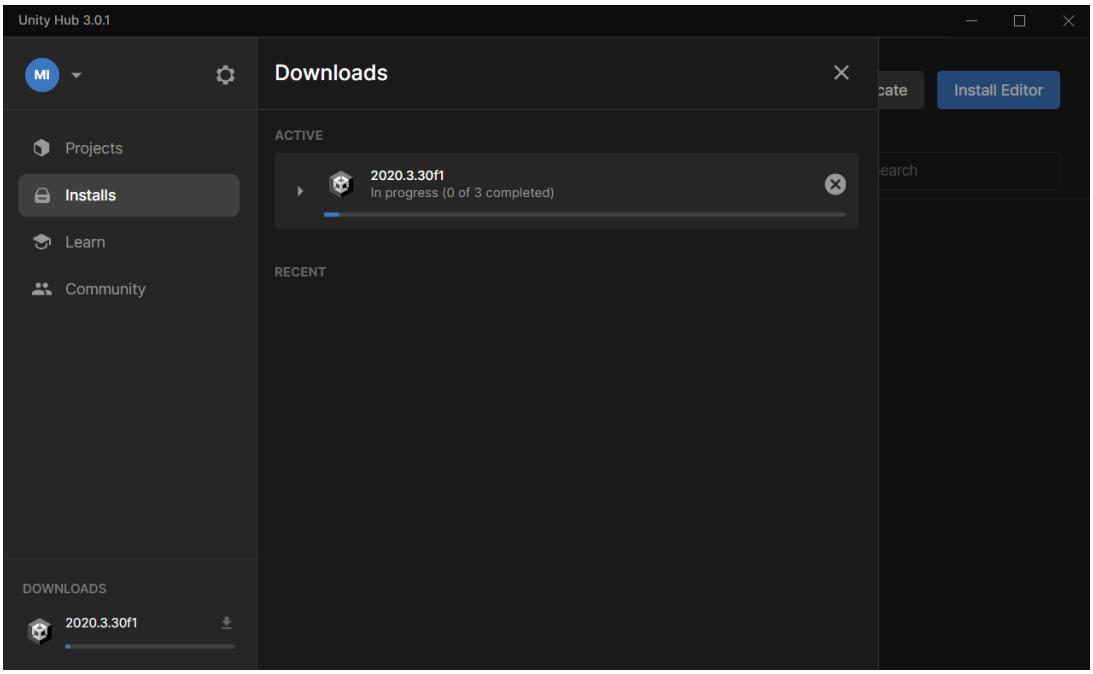
Διαλέγουμε που θέλουμε να εγκαταστήσουμε το Unity και πατάμε *"Install"*,



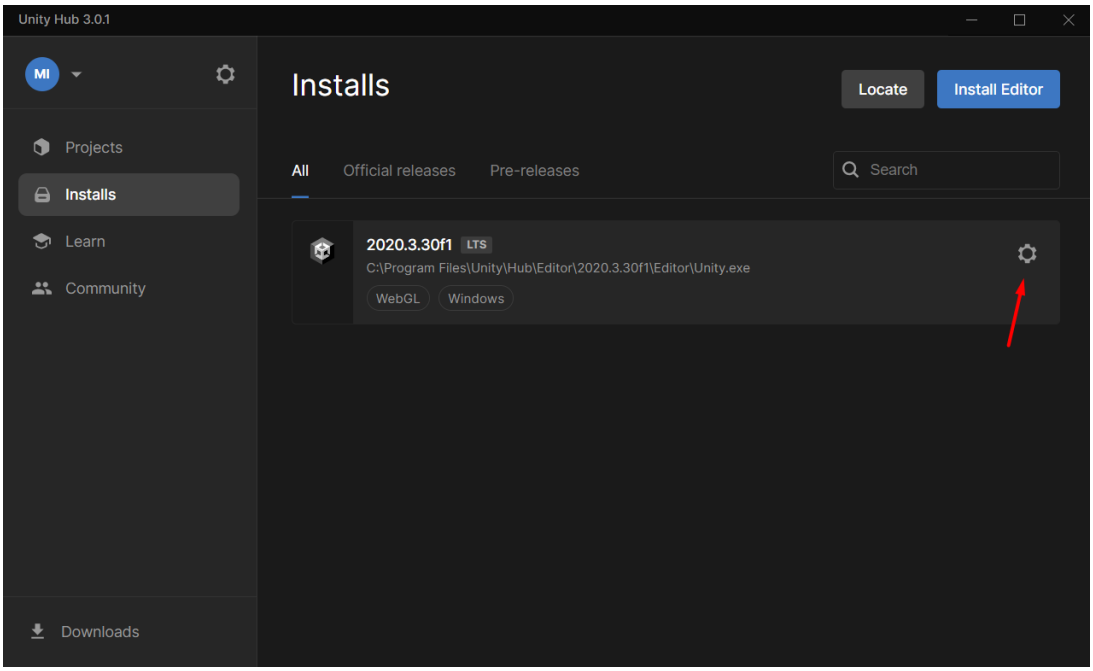
Συνεχίζουμε πατώντας *"Finish"* και ανοίγουμε το Unity Hub,



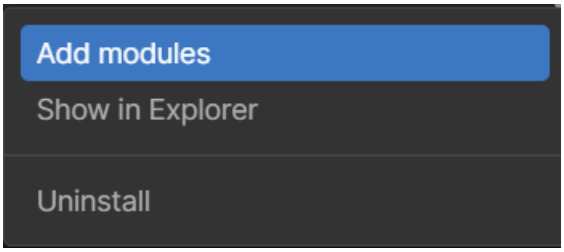
Αφήνουμε όπως έχει το path για την εγκατάσταση και πατάμε *"Install Unity Editor"*,

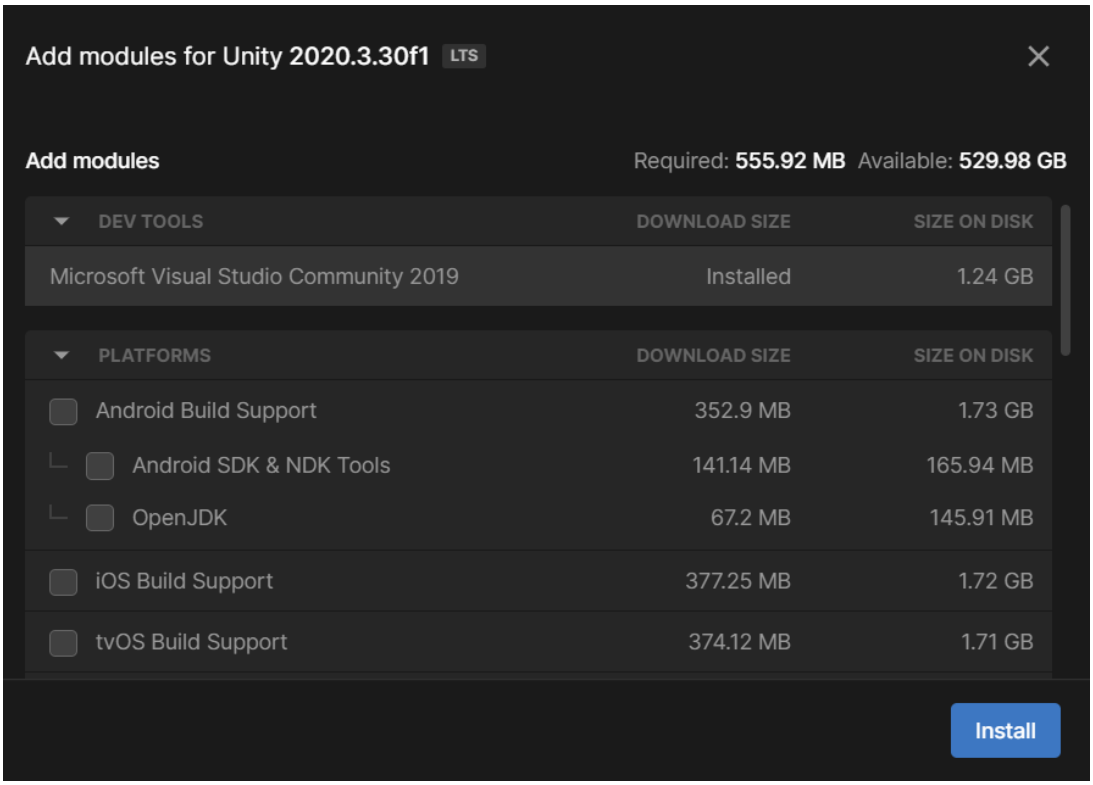


Κάνει τις απαραίτητες λήψεις για τη σωστή λειτουργία και περιμένουμε,

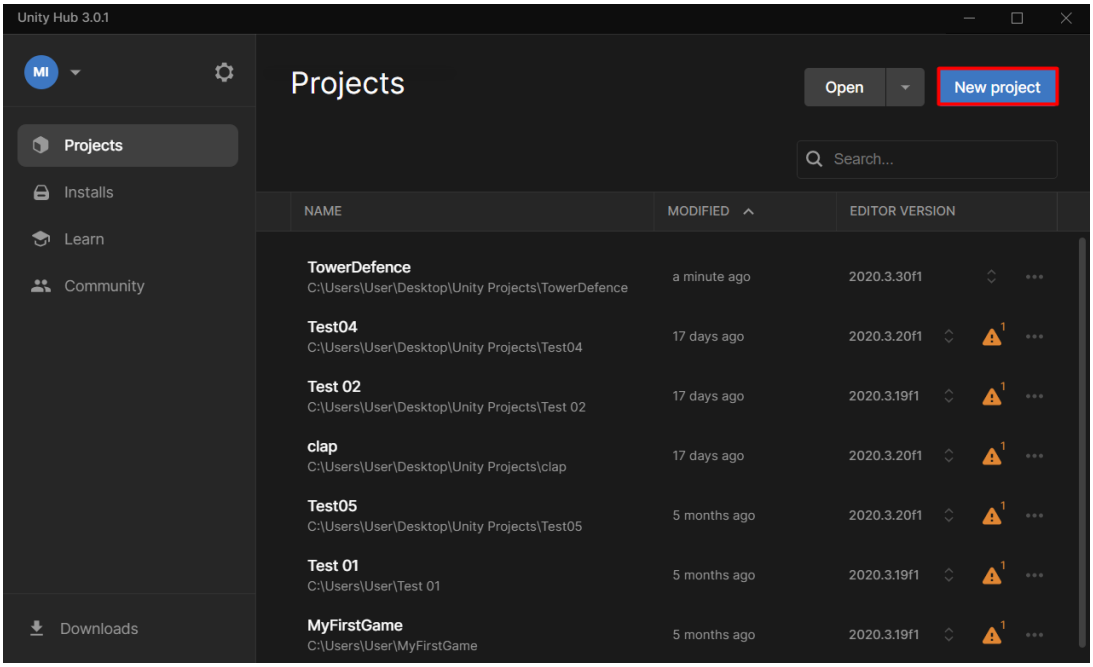


Μπορούμε να πατήσουμε το γρανάζι και να προσθέσουμε περισσότερες λειτουργίες και ρυθμίσεις για επιπλέον λειτουργικά και γλώσσες.

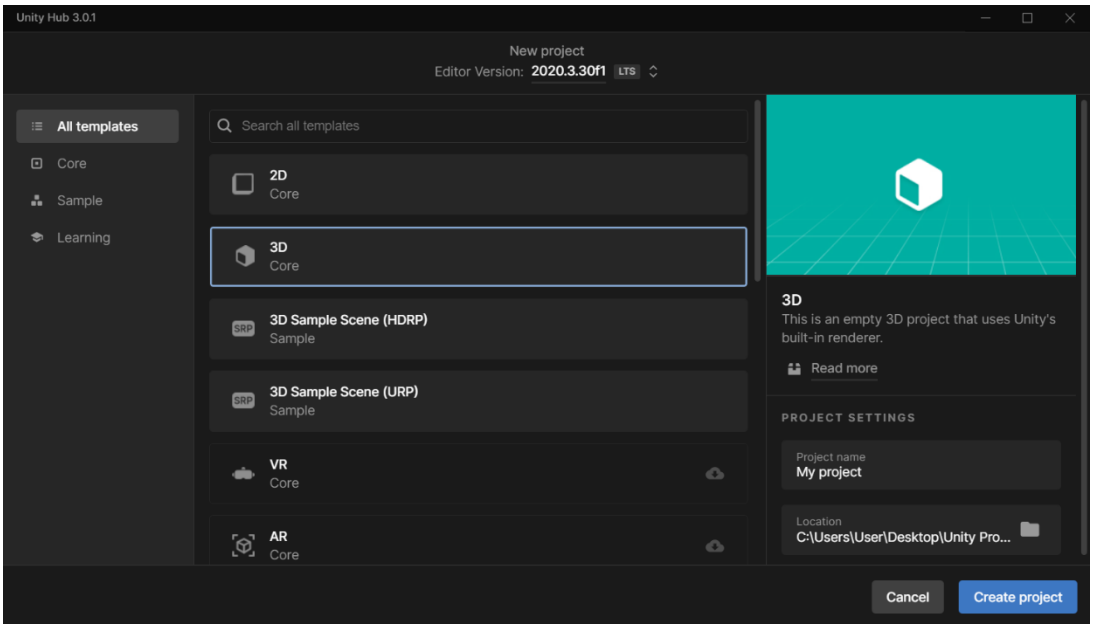




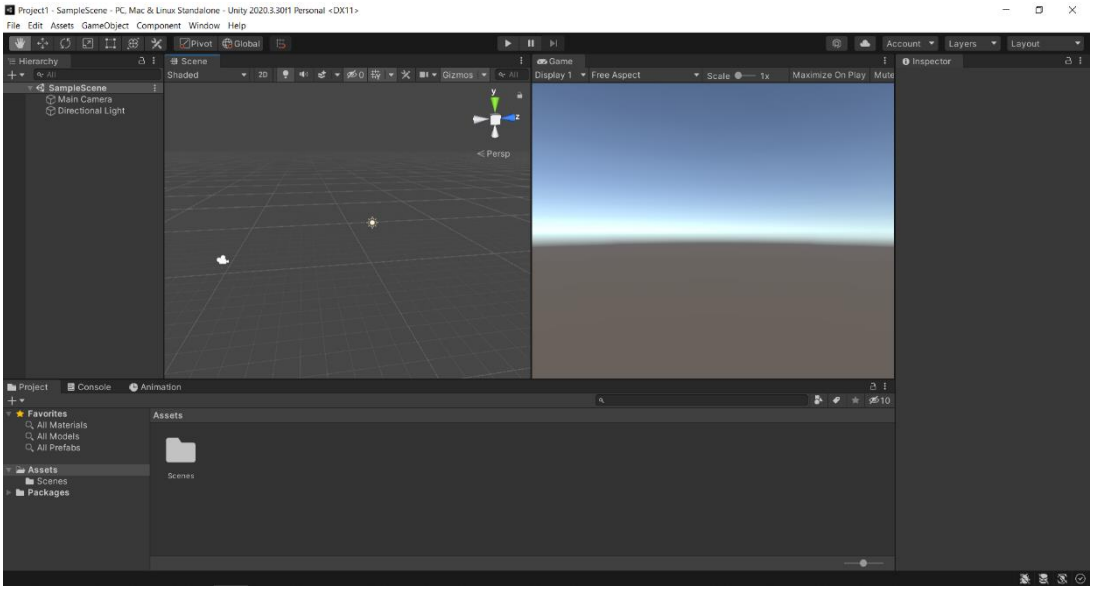
Θα δούμε τώρα πως ξεκινάμε νέο project,



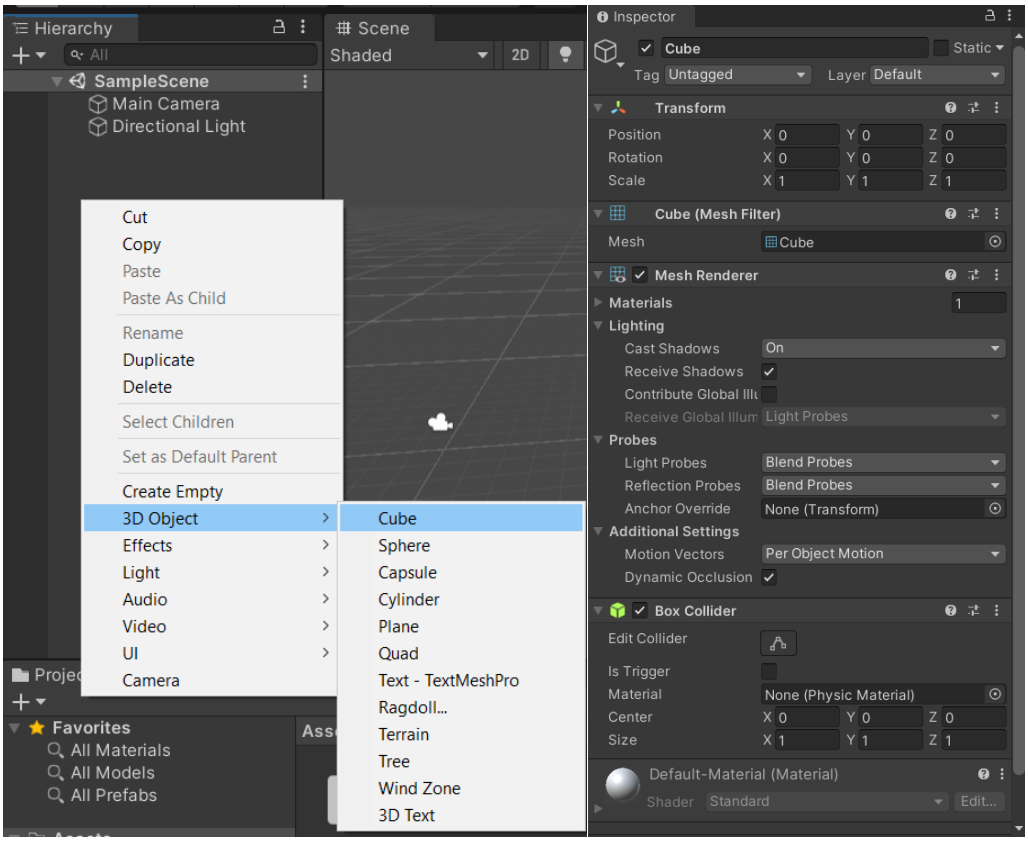
Στην ενότητα Projects επιλέγουμε το *"New project"*, και περνάμε στην επόμενη σελίδα που διαλέγουμε ποιο template θέλουμε για το παιχνίδι μας,



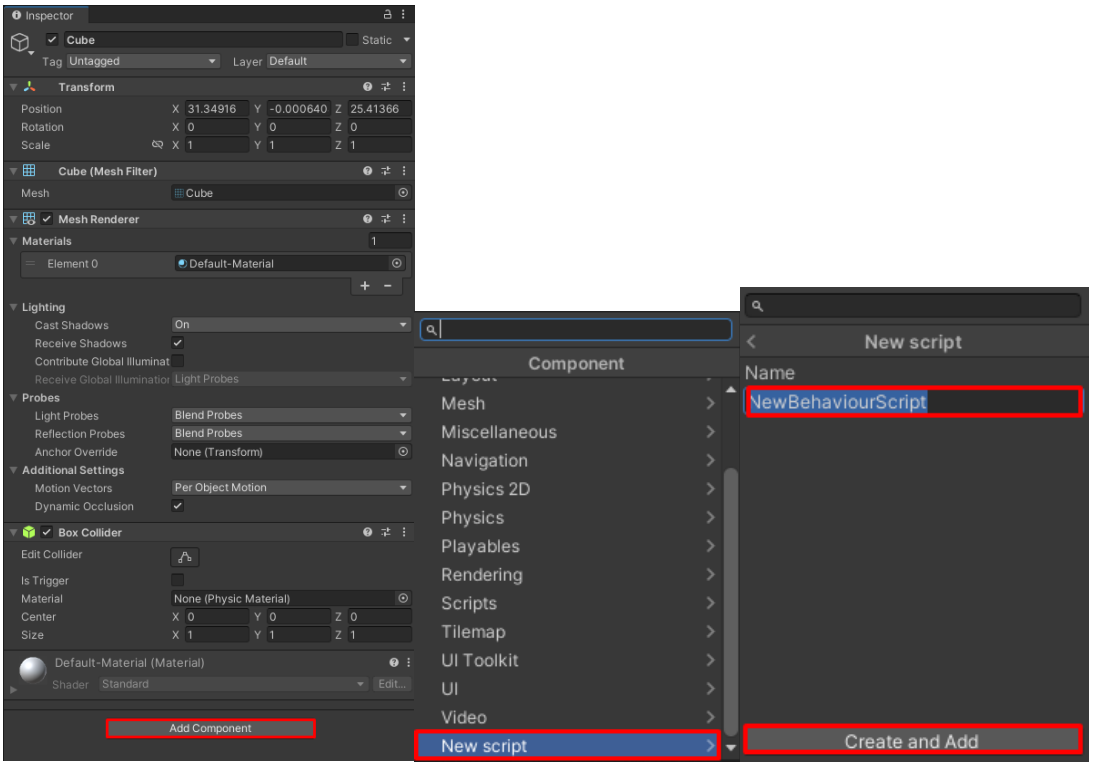
Διαθέσιμα templates: 2D (δύο διαστάσεων), 3D (τρισδιάστατο), 3D Sample Scene (HDRP και URP) ρεαλιστικά γραφικά με σκιές και επιπλέον λειτουργίες από το απλό 3D. Εμείς επιλέγουμε το απλό 3D, αλλάζουμε το όνομα του project και την τοποθεσία αποθήκευσης και πατάμε "*Create Project*",



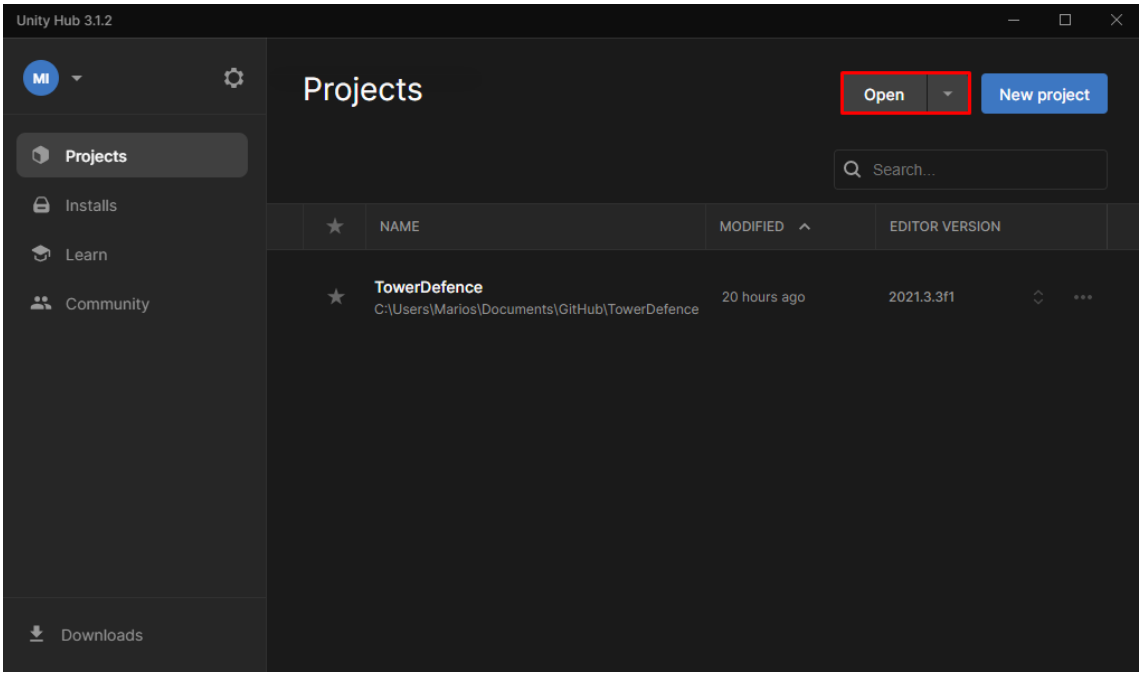
Αυτό είναι το περιβάλλον του Unity. Αριστερά είναι το Hierarchy όπου βλέπει όλα τα αντικείμενα που είναι στη συγκεκριμένη σκηνή, ακριβώς από δίπλα είναι η σκηνή την οποία επεξεργαζόμαστε, αμέσως δεξιά είναι το παιχνίδι και στο δεξιά παράθυρο είναι το Inspector στο οποίο βλέπουμε τις παραμέτρους των αντικειμένων.



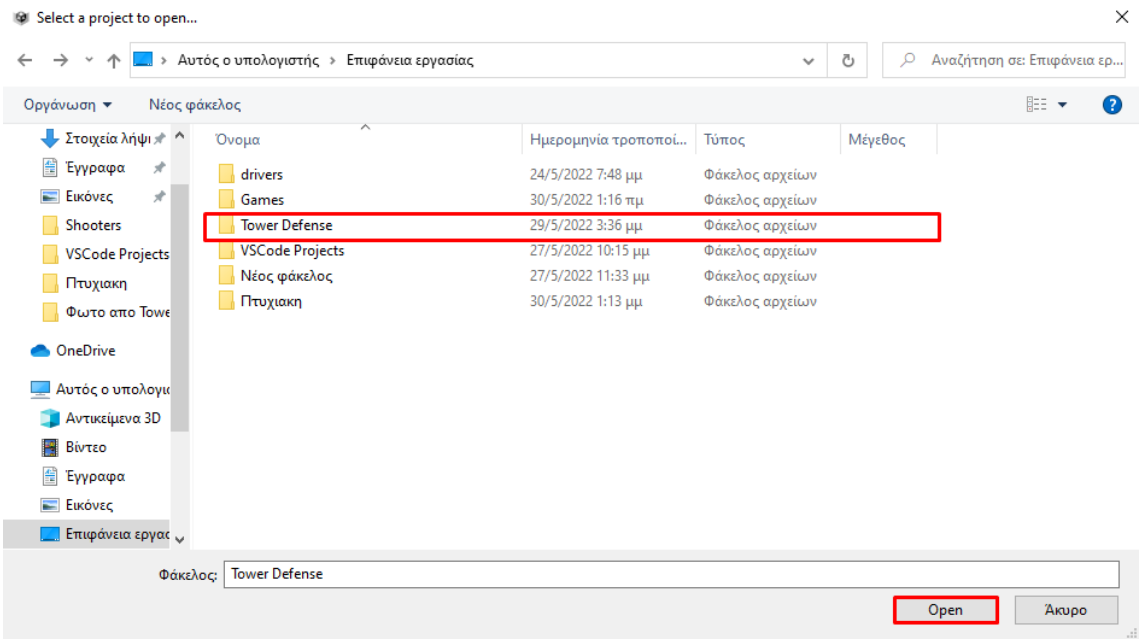
Τώρα πάμε να δημιουργήσουμε το πρώτο και βασικό αντικείμενο, ένα κύβο. Πατάμε δεξί κλικ στο Hierarchy, 3D Object και διαλέγουμε το Cube. Στο Inspector βλέπουμε τις ρυθμίσεις που έχει ο κύβος, δηλαδή σε ποιες συντεταγμένες είναι στη σκηνή όπου αποτελείται από τους άξονες X Y Z και τι μέγεθος έχει. Για να προσθέσουμε κάποιο script στο αντικείμενο ακολουθούμε τα εξής βήματα:



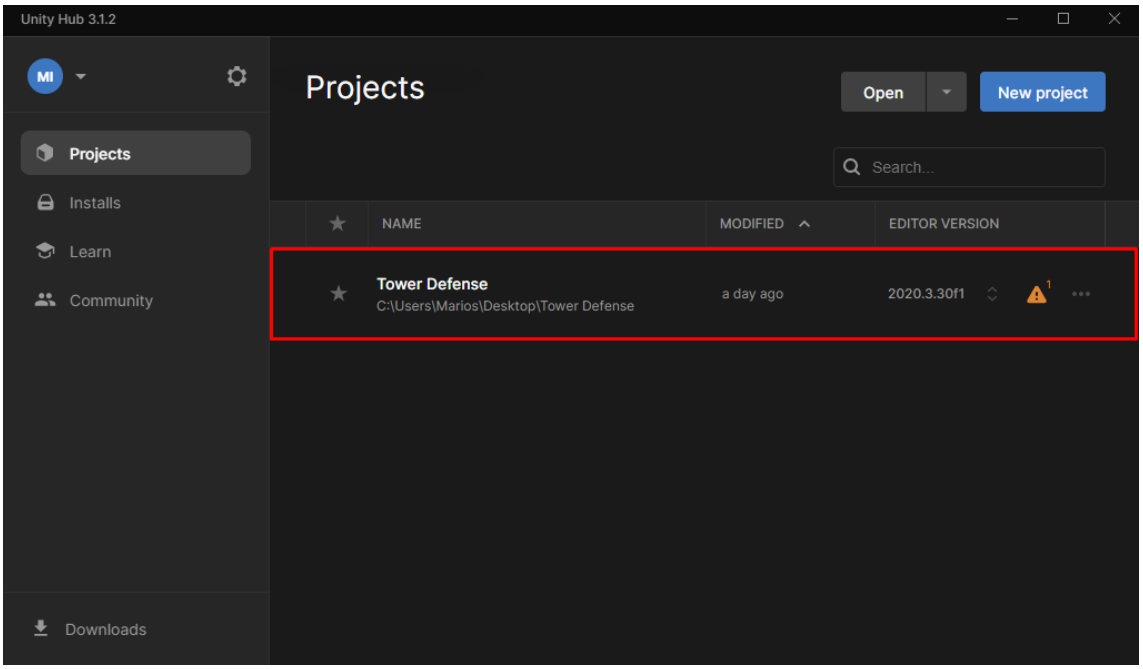
Αν θέλουμε τώρα να ανοίξουμε κάποιο ήδη υπάρχον project, στο αρχικό μενού του Unity Hub, πατάμε “Open”



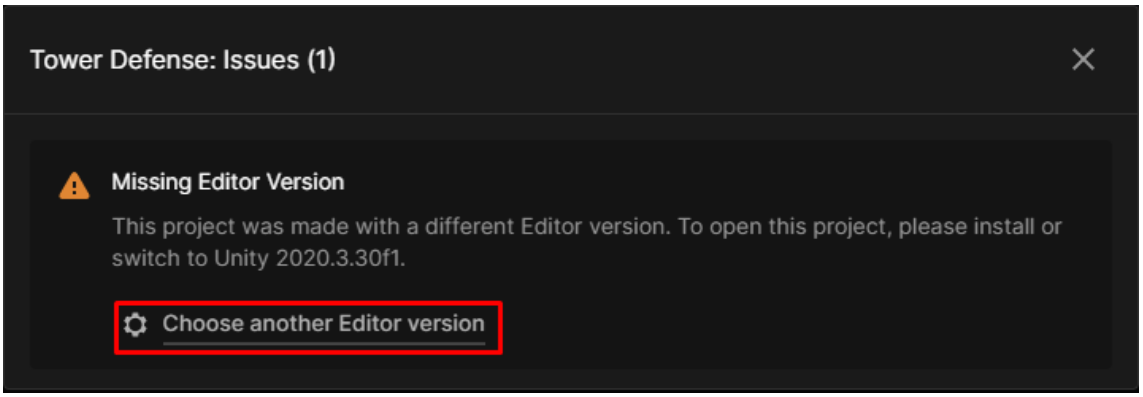
Διαλέγουμε ολόκληρο το φάκελο του project και πατάμε “Open”

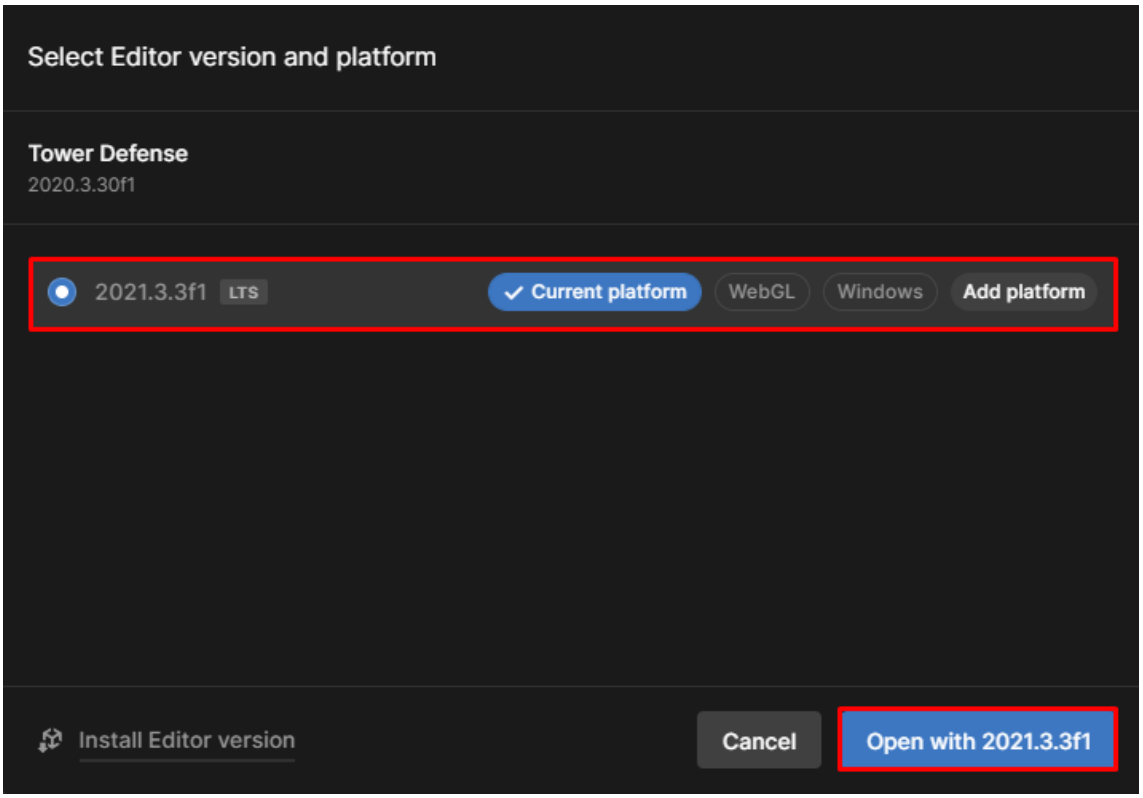


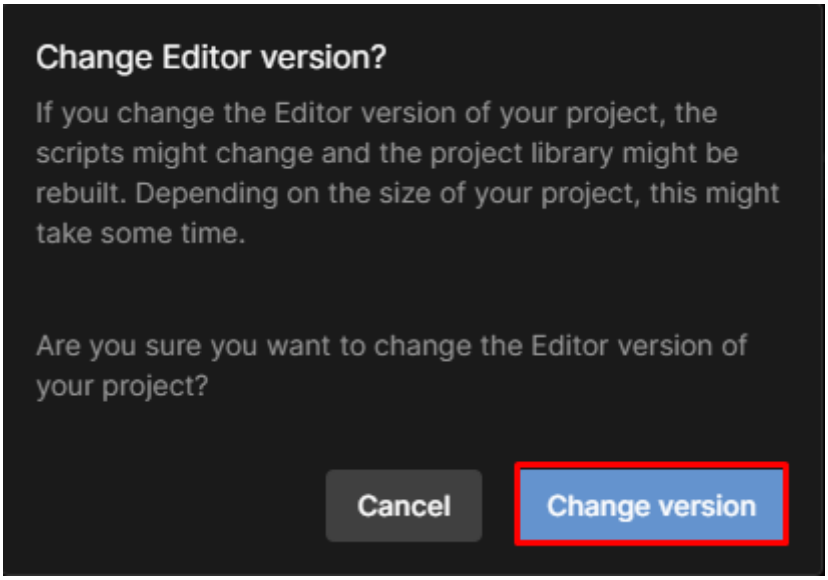
Αφού φορτώσουμε το project στο Unity Hub, υπάρχει περίπτωση να είναι άλλη έκδοση από αυτή του Unity.



Πατάμε πάνω στο θαυμαστικό και διαλέγουμε έκδοση







Αλλάζουμε έκδοση και είναι έτοιμο.

Αυτά ήταν τα βήματα για την εγκατάσταση του Unity και τη δημιουργία ενός απλού αντικειμένου.