ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και ανάπτυξη βιντεοπαιχνιδιού σε πλατφόρμα Unity με χρήση τεχνητής νοημοσύνης

Διπλωματική Εργασία

Μπακατσής Ορέστης

Επιβλέπουσα: Τσομπανοπούλου Παναγιώτα

Φεβρουάριος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και ανάπτυξη βιντεοπαιχνιδιού σε πλατφόρμα Unity με χρήση τεχνητής νοημοσύνης

Διπλωματική Εργασία

Μπακατσής Ορέστης

Επιβλέπουσα: Τσομπανοπούλου Παναγιώτα

Φεβρουάριος 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Design and development of a video game on the Unity platform with the use of artificial intelligence

Diploma Thesis

Bakatsis Orestis

Supervisor: Tsompanopoulou Panagiota

February 2022

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπουσα	**Τσομπανοπούλου Παναγιώτα** Αναπληρώτρια Καθηγήτρια, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Βασιλακόπουλος Μιχαήλ** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Τσουκαλάς Ελευθέριος** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

Μπακατσής Ορέστης

Διπλωματική Εργασία

Σχεδίαση και ανάπτυξη βιντεοπαιχνιδιού σε πλατφόρμα Unity με χρήση τεχνητής νοημοσύνης

Μπακατσής Ορέστης

Περίληψη

Όταν το 1958 δημιουργείται το πρώτο βιντεοπαιχνίδι, κανείς δεν μπορούσε να προβλέψει την εξέλιξη μίας νέας κυρίαρχης δύναμης στην βιομηχανία της ψυχαγωγίας. Από τις τεράστιες, χοντροκομμένες, ασπρόμαυρες οθόνες μετά στα μεγάλα μηχανήματα σε μαγαζιά ηλεκτρονικών παιχνιδιών και πλέον στην παλάμη του χεριού μέσω κινητών τηλεφώνων, τα βιντεοπαιχνίδια έχουν κάνει μεγάλο ταξίδι στο οποίο δεν διαφαίνεται ο τερματισμός δημιουργώντας παράλληλα νέα επαγγέλματα όπως προγραμματιστές, σχεδιαστές γραφικών κλπ. Μέσα σε όλες τις blockbuster νέες κυκλοφορίες των κολοσσών της βιομηχανίας υπάρχουν και τα remaster, γνωστά και ως αναμορφώσεις, παλιών βιντεοπαιχνιδιών χρησιμοποιώντας τη νέα τεχνολογία με σκοπό την επανακυκλοφόρηση παλαιότερων τίτλων στα νέα standards της εποχής (μηχανές δημιουργίας μεγαλύτερων δυνατοτήτων, εξελιγμένη τεχνητή νοημοσύνη, καλύτερα γραφικά κλπ.). Στόχος αυτής της εργασίας είναι ο σχεδιασμός και η ανάπτυξη ενός remaster βιντεοπαιχνιδιού, συγκεκριμένα του κλασικού παιχνιδιού τύπου “shoot ‘em up – πυροβόλησέ τους” του 1978, Space Invaders, χρησιμοποιώντας την πλατφόρμα Unity και ενσωματώνοντας τεχνητή νοημοσύνη. Θα γίνει μια σύντομη ιστορική αναδρομή, ανάλυση της πλατφόρμας καθώς και των βιβλιοθηκών που χρησιμοποιήθηκαν, εμπόδια και δυσκολίες που αντιμετωπίστηκαν και τέλος συζήτηση των αποτελεσμάτων της εργασίας και προτάσεις για βελτιώσεις.

Λέξεις-κλειδιά:

Βιντεοπαιχνίδι, παιχνίδι, Unity, τεχνητή νοημοσύνη

Diploma Thesis

Design and development of a video game on the Unity platform with the use of artificial intelligence

Bakatsis Orestis

Abstract

When the first video game was created in 1958, nobody could foresee the rise of a new dominant force in the entertainment industry. From the huge, chunky, black and white screens to the big machines at the arcades and currently in the palm of our hand through smartphones, the journey of video games has been a big one and appears to be no end in sight creating new professions along the way like developers, graphic designers, etc. Among all the blockbuster new releases from the industry's bests there are also remasters of old video games using new technologies with the purpose of re-releasing older titles with new-era standards (engines with more possibilities, advanced artificial intelligence, better graphics, etc.). The aim of this project is the design and development of a remaster video game, specifically of the classic "shoot 'em up" game of 1978, Space Invaders, using the Unity platform and intergrading artificial intelligence. There will be a short chronology, analysis of the platform as well as the libraries that were used, obstacles and difficulties that were faced and finally a discussion about the results of this project and suggestions for possible improvements.

Keywords:

Videogame, game, Unity, artificial intelligence

Περιεχόμενα

Λίστα Εικόνων

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ

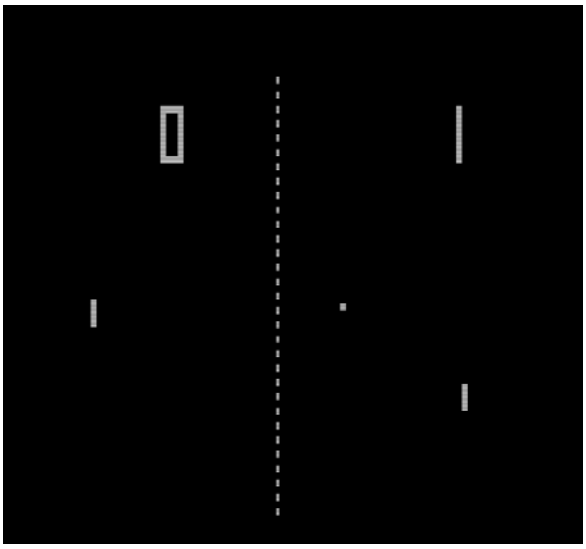
1.α Ιστορική Αναδρομή Βιντεοπαιχνιδιών

Το 1958, ο φυσικός William Higinbotham προσπαθώντας να κάνει πιο ενδιαφέρουσα την επίσκεψη του κοινού στο Εθνικό Εργαστήριο του Brookhaven, στο οποίο ήταν και μέλος, αποφάσισε να δημιουργήσει για αυτούς μία διαδραστική εμπειρία. Του γεννήθηκε η ιδέα ενός παιχνιδιού τέννις και έτσι με την βοήθεια του τεχνικού Robert Dvorak μετά από δύο περίπου εβδομάδες κατασκευής της συσκευής και εντοπισμού σφαλμάτων, το πρώτο βιντεοπαιχνίδι ήταν γεγονός. Το ονόμασαν Tennis for Two (Εικόνα 1).

Εικόνα 1: Το πρώτο βιντεοπαιχνίδι με την συσκευή που δημιουργήθηκε, Tennis for Two [1]

Τα γραφικά του παιχνιδιού δεν είχαν καμία σχέση με τα σημερινά. Το γήπεδο του τέννις απεικονιζόταν με δύο γραμμές, μία για το έδαφος και μία για το φιλέ, και μία κουκίδα η οποία έπαιζε το ρόλο της μπάλας. Το βιντεοπαιχνίδι λατρεύτηκε από τους επισκέπτες που σχημάτιζαν ουρές για να έχουν την ευκαιρία να το παίξουν. Ο ίδιος δήλωσε ότι δεν πίστευε ότι αυτό που δημιούργησε ήταν συναρπαστικό και νόμιζε πως ο κόσμος περίμενε εκεί επειδή τα υπόλοιπα εκθέματα ήταν βαρετά. Η αρχή όμως είχε γίνει.[2]

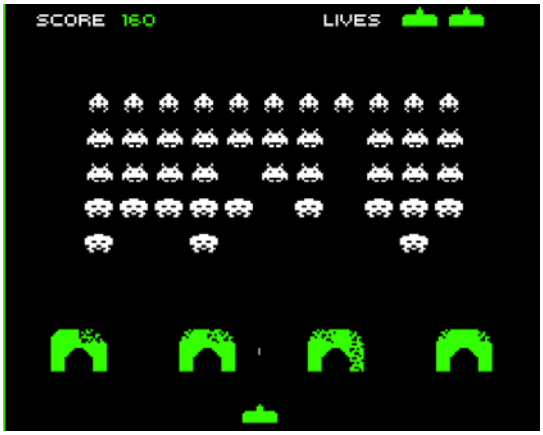
Το 1972, η εταιρεία Atari κυκλοφορεί το πρώτο εμπορικά επιτυχημένο βιντεοπαιχίδι, Pong (Εικόνα 2i), η πρωτότυπη ηλεκτρονική έκδοση του οποίου πούλησε πάνω από 19000 μηχανήματα arcade (Εικόνα 2ii) και έμοιαζε αρκετά με μία εξέλιξη του Tennis for Two.



(i) (ii)

Εικόνα 2: i) Το βιντεοπαιχνίδι Pong, ii) Πρώιμη μορφή μηχανήματος arcade [3]

Φτάνοντας έτσι στο 1978 και στην κυκλοφορία της μεγάλης επιτυχίας, Space Invaders (Εικόνα 3), η οποία και σήμανε την αναγέννηση της βιομηχανίας και οδήγησε στη χρυσή εποχή των arcade βιντεοπαιχνιδιών.



Εικόνα 3: Το βιντεοπαιχνίδι Space Invaders [4]

Η συγκεκριμένη επιτυχία ήταν ο λόγος που τα μηχανήματα arcade έγιναν διαδεδομένα και σε πιο mainstream τοποθεσίες, όπως εμπορικά κέντρα, εστιατόρια κα., πουλώντας συνολικά πάνω από 360000 μηχανήματα παγκοσμίως και μέχρι το 1982 δημιούργησε έσοδα δύο δισεκατομμυρίων δολαρίων.[5]

Το πιο επιτυχημένο βιντεοπαιχνίδι της δεκαετίας του 80 ήταν το Pac-Man, που κυκλοφόρησε το 1980 και μέσα σε ένα χρόνο δημιούργησε έσοδα ενός δισεκατομμυρίου δολαρίων. Το 1982, η βιομηχανία των βιντεοπαιχνιδιών θα ξεπεράσει τα ετήσια έσοδα της μουσικής pop καθώς και των ταινιών του Hollywood μαζί.[6]

Στην αρχή αυτής της δεκαετίας έχουμε και την άνοδο των 8-bit οικιακών υπολογιστών και των βιντεοπαιχνιδιών φτιαγμένων στο σπίτι. Το 1985, έχουμε την κυκλοφορία του Nintendo Entertainment System (Εικόνα 4) που οδήγησε στην κυριαρχία της αγοράς των οικιακών κονσόλων απο ιαπωνικές εταιρείες όπως η Nintendo.

Εικόνα 4: Η κονσόλα Nintendo Entertainment System [7]

Μέσα στην δεκαετία του 90 υπήρξαν μεγάλες τεχνολογικές εξελίξεις στον τομέα των βιντεοπαιχνιδιών όπως η διάδοση των CD μέσων αποθήκευσης και διανομής λογισμικού, η υιοθέτηση GUI λειτουργικών συστημάτων (Microsoft Windows), γραφικά 3D τεχνολογίας, σμίκρυνση του hardware και κινητά τηλέφωνα που επέτρεψαν τα βιντεοπαιχνίδια για κινητά και τέλος η εμφάνιση του ίντερνετ το οποίο επέτρεψε το διαδικτυακό συνεργατικό αλλά και ανταγωνιστικό παιχνίδι.

Την επόμενη δεκαετία (2000), η βιομηχανία των βιντεοπαιχνιδιών είναι μεγαθήριο στις εξελίξεις καθώς τα κέρδη οδηγούν σε τεχνολογική αναβάθμιση με τεχνολογίες όπως έξυπνα τηλέφωνα (smartphones), εικονική πραγματικότητα (virtual reality) και επαυξημένη πραγματικότητα (augmented reality). Έχουμε και την εμφάνιση των ανεξάρτητων (indie) βιντεοπαιχνιδιών.

Φτάνοντας στο 2010 τα indie βιντεοπαιχνίδια έχουν σημαντικό αντίκτυπο στη βιομηχανία. Επίσης, υπάρχει μία τάση προς τα παιχνίδια για κινητά, η αγορά των οποίων το 2016 σημειώνει 38 δισεκατομμύρια δολάρια κέρδη.[8]

Αυτή τη δεκαετία έχουμε επίσης και την άνοδο του ηλεκτρονικού αθλητισμού (eSports), με επαγγελματίες παίκτες σε οργανωμένα πρωταθλήματα, με έπαθλα πολλών εκατομμυρίων, δημιουργώντας έναν νέο τρόπο ψυχαγωγίας και συγκεντρώνοντας εκατομμύρια θεατών όπως μπορούμε να δούμε στην Εικόνα 5.

Εικόνα 5: Τελικός του παγκόσμιου πρωταθλήματος League of Legends στο Staples Center [9]

Με τέτοιο παρελθόν και με την τεχνολογία να κάνει συνεχώς άλματα προόδου, το μέλλον των βιντεοπαιχνιδιών φαίνεται φωτεινό.

1.β Οι Λόγοι Που Παίζουμε Βιντεοπαιχνίδια

Η έκρηξη της βιομηχανίας σε σημείο που κάποια βιντεοπαιχνίδια να μοιάζουν με ταινίες του Hollywood, δεν θα έρχοταν ποτέ αν πέρα από τις τεχνολογικές εξελίξεις δεν υπήρχε το αμείωτο ενδιαφέρον και η ενασχόληση των καταναλωτών προς κάθε μορφής παιχνίδι για μία ποικιλία από λόγους. Από ψυχαγωγικούς και ψυχολογικούς μέχρι και λόγους υγείας τα βιντεοπαιχνίδια έχουν κάτι για όλους:

-**Ποικιλία επιλογών.** Με εκατομμύρια διαφορετικά βιντεοπαιχνίδια σε κυκλοφορία υπάρχουν αμέτρητες εμπειρίες παιχνιδιών που δίνουν την ευκαιρία να δοκιμάσεις πάντα κάτι νέο.

-**Επίδειξη ικανοτήτων.** Πολλά βιντεοπαιχνίδια απαιτούν επιδεξιότητες από τους παίχτες και οι περισσότεροι ξοδεύουν χιλιάδες ώρες για να γίνουν καλύτεροι και να βελτιώσουν τις επιδόσεις τους. Τα βιντεοπαιχνίδια παρέχουν ένα ασφαλές περιβάλλον για αυτό.

-**Εξερεύνηση.** Υπάρχει μία πληθώρα από ψηφιακούς κόσμους να εξερευνήσεις και να ανακαλύψεις χωρίς κανόνες και όρια.

-Διαφορετικές ζωές. Τη μία στιγμή μπορείς να είσαι βίκινγκ, την άλλη στρατιώτης του Β’ Παγκοσμίου Πολέμου και την άλλη κυνηγός στα δάση του Αμαζονίου. Τα βιντεοπαιχνίδια δίνουν την δυνατότητα να πάρεις όποια ταυτότητα θέλεις και να κάνεις πράγματα που δεν θα έκανες ποτέ στην πραγματική σου ζωή.

-Κοινωνικές σχέσεις. Υπάρχουν αρκετά βιντεοπαιχνίδια τα οποία παίζονται με άλλους παίκτες (online) με τους οποίους αλληλεπιδράς σε πραγματικό χρόνο και έτσι μπορούν να σχηματιστούν νέες φιλίες.

-Ανακούφιση από το στρες. Τα βιντεοπαιχνίδια παρέχουν το κατάλληλο περιβάλλον για την καταπολέμηση του άγχους καθώς και για την αποφυγή δυσάρεστων καταστάσεων ή/και ανθρώπων.

-Επαγγελματική αποκατάσταση. Όλα τα ανταγωνιστικά βιντεοπαιχνίδια χρειάζονται γρήγορα αντανακλαστικά, συντονισμό χεριού-ματιού, στρατηγική κλπ. έτσι μπορούν να παιχτούν επαγγελματικά καθώς τραβάνε την προσοχή. Για αυτό υπάρχει αρκετός κόσμος που κάνει καριέρα στον ηλεκτρονικό αθλητισμό και στη ζωντανή μετάδοση παιχνιδιών (streaming).

-Ελευθερία αποφάσεων. Για αρκετούς ανθρώπους που δεν έχουν ανεξαρτησία στον πραγματικό κόσμο, τα βιντεοπαιχνίδια παρέχουν μία διέξοδο στην οποία είναι ελεύθεροι να κάνουν τις δικές τους επιλογές.

-Αποτυχία με ασφάλεια. Στον κόσμο των βιντεοπαιχνιδιών μπορείς με ασφάλεια να πάρεις όσα ρίσκα θέλεις και να αποτύχεις χωρίς σημαντικές επιπτώσεις. Η επιμονή στην προσπάθεια μετά από κάποια αποτυχία είναι μία καλή δεξιότητα για τον πραγματικό κόσμο.

-Μάθηση. Αρκετά βιντεοπαιχνίδια μπορούν να χρησιμοποιηθούν σαν εργαλεία εκπαίδευσης είτε επειδή για αυτό δημιουργήθηκαν είτε άθελά τους (ιστορικά βιντεοπαιχνίδια).

-Πρόοδος. Στον άνθρωπο αρέσει να δουλεύει σκληρά για κάτι και να βλέπει θετικά αποτελέσματα, κάτι που τα βιντεοπαιχνίδια εκτελούν σε μεγάλο βαθμό καθώς παρακολουθούν την πρόοδο εργασιών και παρέχουν θετική ενίσχυση όταν κάποια ολοκληρώνεται.[10]

Στο πιο βασικό τους επίπεδο η εργασία και το παιχνίδι έχουν πολλά κοινά. Η διαφορά των δύο είναι ότι τα παιχνίδια μεταφέρουν την όποια εργασία που συμβαίνει στα πλαίσια τους σε ένα φανταστικό περιβάλλον που την κάνει ευχάριστη. Η πλοκή ενός παιχνιδιού κάνει τις επιλογές μας να μοιάζουν σημαντικές και έτσι συνδεόμαστε συναισθηματικά με αυτό. Κάποιοι κοινωνιολόγοι πιστεύουν ότι τα βιντεοπαιχνίδια είναι μία εξιδανικευμένη μορφή εργασίας. Όπως λέει ο Andrew Przybylski, Ph.D. και λέκτορας στο πανεπιστήμιο του Essex: «Οι περισσότεροι άνθρωποι βρίσκουν την εργασία ικανοποιητική· έχουμε ενσωματωμένα συναισθηματικά κέντρα ανταμοιβών που μας παρακινούν να ολοκληρώνουμε εργασίες. Ένα από τα πράγματα που είναι πολύ σημαντικό για τα βιντεοπαιχνίδια είναι πόσο πολύ συνδέεται η σκληρή δουλειά με την ανατροφοδότηση που λαμβάνεις». Τα παιχνίδια είναι πιο συνεπή με τις ανταμοιβές δίνοντας ταυτόχρονα περισσότερες επιλογές από την πραγματική ζωή. Παρέχουν άμεση ανατροφοδότηση όταν κάνουμε κάτι σωστά καθώς και πόσο σωστά το κάνουμε. Αυτό το σύστημα ανατροφοδότησης μπορεί να μετατρέψει τα βιντεοπαιχνίδια σε ένα σημαντικό εργαλείο για την βελτίωση του μέλλοντός μας.[11]

1.γ Τι Είναι Τεχνητή Νοημοσύνη

Τεχνητή νοημοσύνη (Τ.Ν.) ή artificial intelligence (A.I.) στα αγγλικά είναι η νοημοσύνη που επιδεικνύεται από μηχανές, σε αντίθεση με την φυσική νοημοσύνη που επιδεικνύεται από ανθρώπους και ζώα και περιλαμβάνει συναίσθηση και συναισθηματικότητα.

Ο κύριος περιορισμός όμως αυτού του ορισμού για την Τ.Ν. είναι ότι δεν εξηγεί ακριβώς τι είναι Τ.Ν. και τι κάνει μία μηχανή ευφυή. Πολλά εγχειρίδια πάνω στην Τ.Ν. ορίζουν το πεδίο της ως τη μελέτη ευφυών πρακτόρων, συσκευών δηλαδή που αντιλαμβάνονται το περιβάλλον στο οποίο βρίσκονται και εκτελούν πράξεις που μεγιστοποιούν την πιθανότητα να εκπληρώσουν επιτυχώς τους στόχους τους. Επίσης, ο όρος «τεχνητή νοημοσύνη» χρησιμοποιείται για να περιγράψουμε μηχανές που μιμούνται «γνωσιακές» λειτουργίες που συνδέουμε με τον ανθρώπινο εγκέφαλο όπως η μάθηση και η επίλυση προβλημάτων.[12]

Οι Stuart Russell και Peter Norvig στο πρωτοποριακό βιβλίο τους: «Τεχνητή Νοημοσύνη: Μια σύγχρονη προσέγγιση» προσπαθούν να δώσουν μία εξήγηση συγκεντρώνοντας την δουλειά τους γύρω από το θέμα των ευφυών πρακτόρων στις μηχανές και εξετάζουν τέσσερις διαφορετικές προσεγγίσεις του ορισμού που έχουν αποδοθεί ιστορικά στον τομέα της Τ.Ν.

-Ανθρώπινη Σκέψη

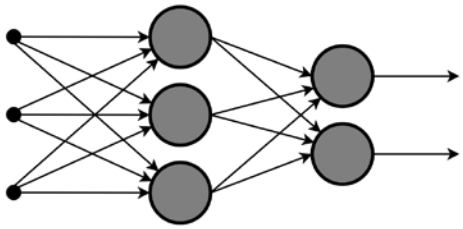
-Λογική Σκέψη

-Ανθρώπινη Ενέργεια

-Λογική Ενέργεια

Η Τ.Ν. χωρίζεται σε δύο κατηγορίες την ασθενή και την ισχυρή. Η ασθενής βρίσκεται παντού γύρω μας και μέχρι σήμερα είναι η πιο επιτυχημένη εφαρμογή τεχνητής νοημοσύνης και έχει στόχο την εκτέλεση συγκεκριμένων έργων. Υποκατηγορία της ασθενούς είναι η Μηχανική Μάθηση (Machine Learning) και υποκατηγορία αυτής η Βαθιά Μάθηση (Deep Learning). Με απλά λόγια η Μηχανική Μάθηση τροφοδοτεί έναν υπολογιστή δεδομένα και χρησιμοποιεί τεχνικές στατιστικής που τον βοηθάνε να

«μάθει» πως να γίνει προοδευτικά καλύτερος στο έργο που έχει να φέρει εις πέρας χωρίς να ξοδευτούν αμέτρητες γραμμές κώδικα για να προγραμματιστεί. Στη Βαθιά Μάθηση τα δεδομένα περνάνε από ένα δίκτυο «νευρώνων» (neural network) εμπνευσμένο από τον ανθρώπινο εγκέφαλο όπως φαίνεται στο διάγραμμα της Εικόνας 6, το οποίο περιέχει κρυφά επίπεδα στα οποία επεξεργάζονται τα δεδομένα για καλύτερα αποτελέσματα.



Εικόνα 6: Διάγραμμα ενός απλού neural network [13]

Μερικά παραδείγματα ασθενούς Τ.Ν. είναι:

-«Έξυπνοι» βοηθοί (Siri, Alexa)

-Φίλτρα spam στα email

-Εργαλεία χαρτογράφησης και πρόγνωσης ασθενειών

-Εργαλεία παρακολούθησης social media για επικίνδυνο περιεχόμενο ή ψευδείς ειδήσεις

-Προτάσεις τραγουδιών, τηλεοπτικών εκπομπών ή βίντεο (Spotify, Netflix, YouTube)[14]

Η ισχυρή Τ.Ν. είναι ο μακρυπρόθεσμος στόχος και το «άπιαστο όνειρο» αυτού του κλάδου και αφορά την δημιουργία μιας μηχανής που θα κατέχει ανθρώπινη ευφυΐα και θα μπορεί να εκτελέσει οποιοδήποτε έργο σε οποιοδήποτε περιβάλλον. Η συγκεκριμένη εκτός από στόχο αποτελεί και έμπνευση για δυστοπική επιστημονική φαντασία όπου πανέξυπνες μηχανές κυριεύουν τον άνθρωπο, κάτι που οι ειδικοί απορρίπτουν.[15]

Ορόσημα στην ιστορία την τεχνητής νοημοσύνης:

1943

Οι Warren McCullough και Walter Pitts εκδίδουν το "A Logical Calculus of Ideas

Immanent in Nervous Activity" το οποίο πρότεινε το πρώτο μαθηματικό μοντέλο για την κατασκευή neural network.

1950

Ο Alan Turing εκδίδει το "Computing Machinery and Intelligence" προτείνοντας μία μέθοδο που καθορίζει αν μία μηχανή είναι ευφυής ή όχι, γνωστή πλέον ως Turing Test.

1956

Επινοείται ο όρος «τεχνητή νοημοσύνη» στο συνέδριο "Dartmouth Summer Research Project on Artificial Intelligence". Με επικεφαλή τον John McCarthy, καθορίστηκαν το πεδίο δράσης και οι στόχοι της τεχνητής νοημοσύνης και θεωρείται η γενέτειρα της Τ.Ν. όπως την ξέρουμε σήμερα.

1958

Ο John McCarthy αναπτύσσει την πρώτη προγραμματιστική γλώσσα για Τ.Ν. με το όνομα Lisp.

1972

Δημιουργείται η γλώσσα λογικού προγραμματισμού PROLOG.

1997

Το Deep Blue της εταιρείας IBM κερδίζει στο σκάκι τον παγκόσμιο πρωταθλητή Gary Kasparov (Εικόνα 7).

Εικόνα 7: Deep Blue εναντίον Gary Kasparov [16]

2008

Η Google πραγματοποιεί σημαντικές ανακαλύψεις στον τομέα της φωνητικής αναγνώρισης και εισάγει την δυνατότητα στην εφαρμογή της στα iPhone.

2012

Ο Andrew Ng, ιδρυτής του Google Brain Deep Learning project, τροφοδοτεί ένα neural network 10 εκατομμύρια βίντεο YouTube χρησιμοποιώντας αλγορίθμους Βαθιάς Μάθησης. Το neural network μαθαίνει να αναγνωρίζει μία γάτα χωρίς να του έχουν πει τι είναι η γάτα.

2014

Η Google κατασκευάζει το πρώτο αυτο-οδηγούμενο αυτοκίνητο που περνάει την εξέταση οδήγησης.

2016

Το AlphaGo του Google DeepMind κερδίζει στον αρχαίο κινέζικο παιχνίδι Go τον παγκόσμιο πρωταθλητή Lee Sedol.[14]

1.δ Στόχος Εργασίας Και Διάρθρωση Ενοτήτων

Στην εργασία αυτή σκοπεύουμε να σχεδιάσουμε και να αναπτύξουμε ένα remaster του βιντεοπαιχνιδιού Space Invaders της κατηγορίας “shoot ‘em up” σύμφωνα με τα πρότυπα των σύγχρονων παιχνιδιών όπως καλύτερα γραφικά, περισσότερες επιλογές και δυνατότητες για τον χρήστη και να ενσωματώσουμε πιο «δυνατή» τεχνητή νοημοσύνη απο τις δυνατότητες που παρείχε η τεχνολογία της εποχής όταν κυκλοφόρησε για πρώτη φορά.

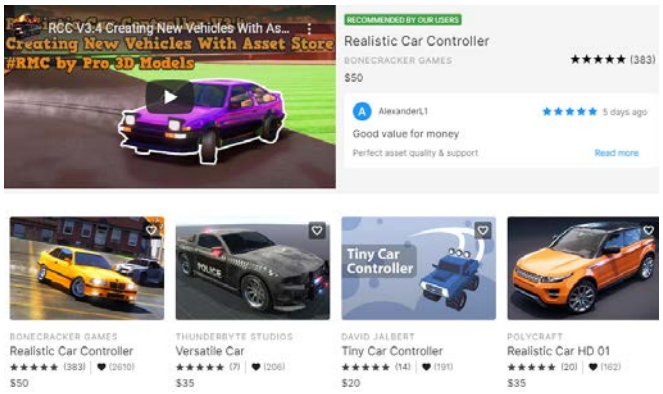
Στόχος μας είναι η ανάλυση των εργαλειών και των προγραμματιστικών τεχνικών που χρησιμοποιήθηκαν και είναι απαραίτητα για την εκτέλεση των βασικών μηχανισμών ενός παιχνιδιού καθώς και των όποιων προβλημάτων και εμποδίων συναντήσαμε και χρειάστηκε να ξεπεράσουμε. Η ανάπτυξη του παιχνιδιού γίνεται στην πλατφόρμα Unity 2020.3.4f1 που διατίθεται δωρεάν για προσωπική χρήση και ο προγραμματισμός γίνεται στη γλώσσα γενικού σκοπού C# που συνήθως είναι η γλώσσα που χρησιμοποιείται στη συγκεκριμένη πλατφόρμα.

Στις επόμενες ενότητες ακολουθεί αναλυτική παρουσίαση των δομικών στοιχείων και των αρχών λειτουργίας της πλατφόρμας Unity καθώς και της γλώσσας προγραμματισμού C# και κάποιων βιβλιοθηκών της. Η διερεύνηση αυτού του υπόβαθρου γίνεται με σκοπό την βαθύτερη κατανόηση της διαδικασίας της ανάπτυξης και σχεδίασης ενός βιντεοπαιχνιδιού από την αρχή και απευθύνεται τόσο σε αναγνώστες χωρίς εμπειρία όσο και σε αυτούς που έχουν κάποια επαφή με τον προγραμματισμό και θέλουν να ασχοληθούν με την δημιουργία βιντεοπαιχνιδιών. Στη συνέχεια, παρουσιάζονται η διαδικασία σχεδίασης και ανάπτυξης του βιντεοπαιχνιδιού μας που έχουμε ονομάσει “Invaders of the Galaxy” και συνοδεύεται από screenshots και επεξήγηση του κώδικα. Στις τελευταίες ενότητες γίνεται συζήτηση και ανάλυση των αποτελεσμάτων καθώς και προτάσεις για μελλοντική εξέλιξη της εργασίας και βελτιώσεις. Τέλος, παρουσιάζονται τα συμπεράσματα που προκύπτουν από μία τέτοια διαδικασία.

ΚΕΦΑΛΑΙΟ 2 Η ΠΛΑΤΦΟΡΜΑ UNITY

2.α Τι είναι Unity

Η Unity είναι μία μηχανή 3D/2D παιχνιδιών καθώς και ένα ισχυρό Ολοκληρωμένο Περιβάλλον Ανάπτυξης (IDE) για πολλαπλές πλατφόρμες. Πιο συγκεκριμένα, ως μηχανή παιχνιδιών παρέχει στον προγραμματιστή πολλές σημαντικές δυνατότητες που κάνουν ένα παιχνίδι να λειτουργεί όπως φυσική, 3D απόδοση, εντοπισμό κρούσεων κλπ. Ουσιαστικά αυτό σημαίνει ότι δεν χρειάζεται να σπαταληθεί χρόνος στη δημιουργία ενός συστήματος φυσικής από την αρχή (ιδιότητες υλικών, αντανάκλαση φωτός κλπ.) και μπορεί να ξεκινήσει άμεσα κάποιο νέο project. Αυτό όμως που κάνει τη Unity να ξεχωρίζει είναι το Κατάστημα Στοιχείων (Asset Store) που φαίνεται στην Εικόνα 8. Στο κατάστημα αυτό μπορεί κάποιος δημιουργός να μεταφορτώσει αντικείμενα που έχει φτιάξει (π.χ. μία σειρά από πολυτελή αυτοκίνητα) ή προγραμματιστικά εργαλεία τα οποία μπορεί κάποιος άλλος δημιουργός να χρησιμοποιήσει στο δικό του project, ώστε να το βελτιώσει χωρίς να αφιερώσει επιπλέον χρόνο, είτε δωρεάν είτε επί πληρωμή καθώς και να τα αξιολογήσει. Η μεγάλη κοινότητα της Unity βοηθάει σε αυτό καθώς υπάρχουν αμέτρητες επιλογές για τον νέο ή και τον έμπειρο προγραμματιστή ώστε να συγκεντρωθεί σε αυτά που θα κάνουν το παιχνίδι του να ξεχωρίσει.



Εικόνα 8: Έτοιμα αυτοκίνητα στο Asset Store [17]

Ως IDE η πλατφόρμα Unity έχει έναν οπτικό επεξεργαστή σκηνής ο οποίος υποστηρίζει το drag and drop (σύρε και άφησε) στοιχείων και την επεξεργασία των ιδιοτήτων τους. Επίσης υποστηρίζει δυνατότητες όπως πλοήγηση στους φακέλους του project κλπ. Για την συγγραφή κώδικα η Unity χρησιμοποιεί από προεπιλογή τον επεξεργαστή κώδικα της Microsoft, Visual Studio. Η γλώσσα που χρησιμοποιείται πιο συχνά είναι η C# και θα την αναλύσουμε στο Κεφάλαιο 3. Με την Unity δεν είναι απαραίτητο να γραφούν πολλές γραμμές κώδικα για να δημιουργηθεί κάποια σκηνή, παρόλα αυτά γνωρίζοντας από προγραμματισμό υπάρχουν περισσότερες επιλογές και δυνατότητες καθώς η Unity σου επιτρέπει να αλλάξεις μία μεγάλη λίστα από ιδιότητες.

Εξίσου σημαντικό χαρακτηριστικό της συγκεκριμένης πλατφόρμας είναι ότι υποστηρίζει πολλαπλές πλατφόρμες τις οποίες βλέπουμε στην Εικόνα 9. Πρακτικά αυτό σημαίνει ότι ένας προγραμματιστής μπορεί να δημιουργήσει ένα παιχνίδι για κινητά τηλέφωνα και αν το θελήσει, η Unity μπορεί να το μετατρέψει σε ελάχιστο χρόνο ώστε να υποστηρίζεται και από ηλεκτρονικούς υπολογιστές κα. Επίσης, με τον τρόπο που διαχειρίζεται τα γραφικά κάνει πιο εύκολη την προσαρμογή των παιχνιδιών σε παλαιότερο hardware και αυτός είναι ένας από τους λόγους που τα περισσότερα παιχνίδια στο Google Play Store είναι φτιαγμένα με Unity. Τέλος, παρέχει και υποστήριξη για παιχνίδια εικονικής πραγματικότητας (VR).

Εικόνα 9: Πλατφόρμες που υποστηρίζει η Unity [18]

Εκεί που υστερεί σε σχέση με τους ανταγωνιστές της (Unreal, Cryengine) είναι στις δυνατότητες των γραφικών τις οποίες υποστηρίζει. Ωστόσο με τις συνεχείς ενημερώσεις που γίνονται έχει μικρύνει η ψαλίδα. Παραδείγματα παιχνιδιών που έχουν γίνει στη Unity μπορούμε να δούμε στην Εικόνα 10.

(i) (ii) (iii)

Εικόνα 10: Παιχνίδια που έχουν δημιουργηθεί με Unity: i) Hearthstone [19], ii) Rust [20], iii) Super Mario Run [21]

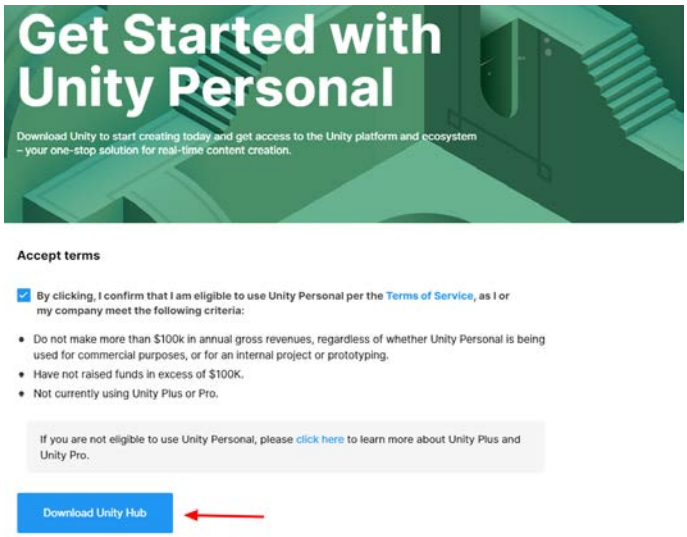
2.β Λήψη Και Εγκατάσταση Unity

Η διαδικασία λήψης και εγκατάστασης της Unity είναι πολύ απλή.

Πηγαίνουμε στη σελίδα:

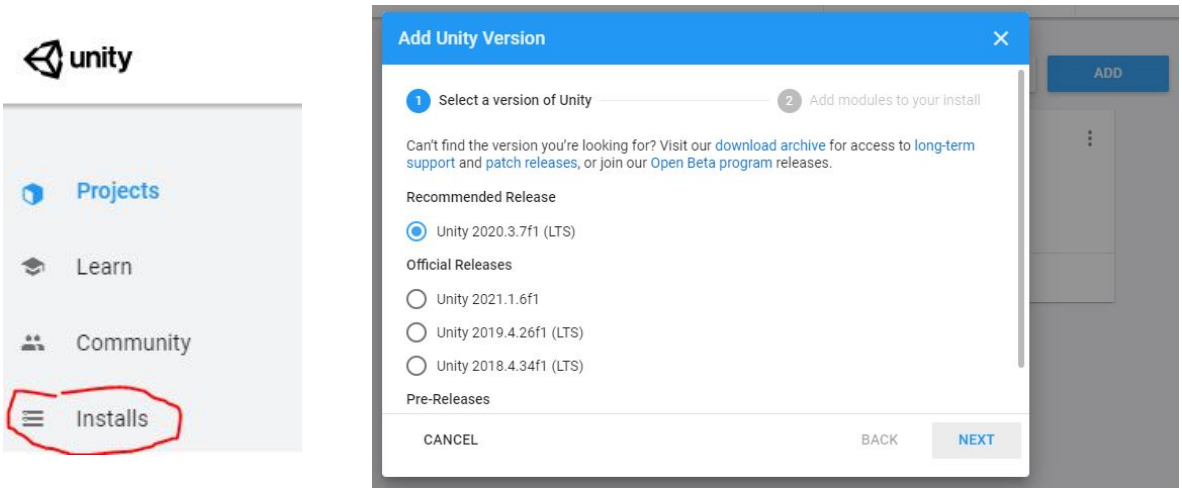
https://store.unity.com/download?ref=personal

Πατάμε το κουμπί “Download Unity Hub” και περιμένουμε να γίνει η λήψη και να ολοκληρωθεί η εγκατάσταση (Εικόνα 11).



Εικόνα 11: Οδηγίες εγκατάστασης Unity 1 [22]

Ανοίγουμε το Unity Hub πατάμε “Installs” και μετά “Add” και επιλέγουμε το Recommended Release (Εικόνα 12). Στη συνέχεια επιλέγουμε τι πλατφόρμες θέλουμε να υποστηρίξουμε στο συγκεκριμένο project (μπορεί να αλλάξει και στη συνέχεια) και πατάμε “Done”. Όταν ολοκληρωθεί η λήψη και η εγκατάσταση είμαστε έτοιμοι.



Εικόνα 12: Οδηγίες εγκατάστασης Unity 2 (πηγή δημιουργός)

Η χρήση της πλατφόρμας Unity εφόσον είναι για προσωπική χρήση είναι δωρεάν. Τέλος, το Unity Hub είναι μία εφαρμογή που διευκολύνει την αναζήτηση, λήψη και διαχείριση των project και των εγκαταστάσεων Unity.

2.γ Το Interface Της Unity

Αφού επιλέξουμε την προσθήκη ενός νέου project (NEW), διαλέγουμε 2D ή 3D ανάλογα με τον τύπο παιχνιδιού που θέλουμε να δημιουργήσουμε και το όνομα και εμφανίζεται μπροστά μας η διεπαφή χρήστη (interface) της πλατφόρμας, ένα παράδειγμα της οποίας φαίνεται στην Εικόνα 13.

Εικόνα 13: Παράδειγμα διεπαφής Unity [23]

Α) Μπάρα Εργαλειών (Toolbar). Παρέχει πρόσβαση στις πιο απαραίτητες λειτουργίες εργασίας. Στα αριστερά υπάρχουν τα βασικά εργαλεία διαχείρισης των Σκηνών (Scenes) και των Αντικειμένων (GameObjects) που υπάρχουν σε αυτές. Στο κέντρο υπάρχουν τα κουμπιά play, pause και step τα οποία ελέγχουν την ροή του χρόνου κατά το τρέξιμο ενός παιχνιδιού. Τα κουμπιά στα δεξιά δίνουν πρόσβαση σε λειτουργίες συνεργασίας και Cloud της Unity όπως και τον λογαριασμό χρήστη. Στη συνέχεια υπάρχει το μενού ορατότητας επιπέδων (layer) και τέλος το μενού με επιλογές για αλλαγή της διάταξης των στοιχείων της διεπαφής.

Β) Παράθυρο Ιεραρχίας (Hierarchy Window). Η απεικόνιση κάθε Αντικειμένου που υπάρχει στη Σκηνή με χρήση κειμένου. Η δομή που έχει φανερώνει και την σχέση που έχουν μεταξύ τους τα GameObjects. Εδώ υπάρχει η δυνατότητα δημιουργίας κάποιου GameObject αλλά και η ενεργοποίηση/απενεργοποίηση ενός που ήδη υπάρχει. Η Unity χρησιμοποιεί την έννοια της κληρονομικότητας των αντικειμένων και με αυτόν τον τρόπο αλλαγές σε γονικά GameObject γίνονται και σε όλα τα παιδιά τους ενώ μεμονωμένες αλλαγές σε παιδιά δεν επηρεάζουν άλλα αντικείμενα. Τέλος, στο συγκεκριμένο παράθυρο πατώντας τα κουμπιά Ctrl+D έχοντας επιλεγμένο ένα Αντικείμενο, δημιουργείται ένα ακριβές αντίγραφό του.

C) Όψη Παιχνιδιού (Game View). Προσομοιώνει την τωρινή μορφή του παιχνιδιού ή κάποια μεμονωμένη Σκηνή μέσα από τις Κάμερες της Σκηνής. Εδώ γίνεται ο έλεγχος και το testing του παιχνιδιού. Πατώντας το κουμπί play ξεκινάει η προσομοίωση.

Οποιεσδήποτε αλλαγές κατά τη διάρκεια της προσομοίωσης είναι προσωρινές και επαναφέρονται όταν αυτή σταματήσει. Στην προεπιλεγμένη διάταξη αυτή η όψη είναι “κρυμμένη” πίσω από την Όψη Σκηνής και μπορεί να προβληθεί πατώντας την καρτέλα στην κορυφή. Τέλος, υπάρχουν ρυθμίσεις για την ανάλυση του παραθύρου καθώς και για την επιλογή της οθόνης στην οποία προβάλλεται κλπ.

D) Όψη Σκηνής (Scene View). Το πιο διαδραστικό σημείο της διεπαφής Unity και το σημείο δημιουργίας, επεξεργασίας και πλοήγησης του περιβάλλοντος του παιχνιδιού. Απεικονίζει την Σκηνή στην οποία βρισκόμαστε και επιτρέπει την τροποποίηση του σημείου,των διαστάσεων κλπ. των Αντικειμένων. Υπάρχει η επιλογή για απεικόνιση 3D ή 2D ανάλογα το project. Εκτενέστερη ανάλυση γίνεται στην ενότητα 2.δ.

E) Παράθυρο Παρατήρησης (Inspector Window). Προβάλλει τα Συστατικά του επιλεγμένου Αντικειμένου και επιτρέπει την επεξεργασία τους. Εδώ επίσης μπορεί να γίνει προσθήκη/αφαίρεση Συστατικών του Αντικειμένου όπως script κώδικα (C#, Javascript), φυσική κα. Επιπροσθέτως, υπάρχει η επιλογή για ομαδοποίηση Αντικειμένων με χρήση ετικετών (tags) για ευκολότερη πρόσβαση σε αυτά μέσα από τον κώδικα, η επιλογή layer στο οποίο ανήκει αλλά και η επιλογή ενεργοποίησης/απενεργοποίησης του Αντικειμένου. Τέλος, με την εναλλαγή του Inspector σε Debug Mode μπορούμε να δούμε περισσοτερες πληροφορίες όπως τις private μεταβλητές των ανατεθειμένων script.

F) Παράθυρο Εργασίας (Project Window). Σε αυτό το παράθυρο υπάρχουν όλα τα αρχεία που συμμετέχουν στη δημιουργία του παιχνιδιού ιεραρχημένα με βάση ένα σύστημα φακέλων. Τα αρχεία περιλαμβάνουν εικόνες, ήχους, αρχεία υφών κα. και υπάρχει η δυνατότητα εισαγωγής με drag-and-drop. Στη συνέχεια μπορούν είτε να αποτελέσουν συστατικό της Σκηνής είτε να επεξεργαστούν κάνοντας διπλό κλικ πάνω τους που οδηγεί στα προεπιλεγμένα προγράμματα επεξεργασίας ανάλογα με τον τύπο αρχείου. Εδώ επίσης μπορούν να δημιουργηθούν scripts, αρχεία υλικών (materials), animations κλπ. Επιπροσθέτως, παρέχεται δυνατότητα αναζήτησης σε αυτά τα αρχεία.

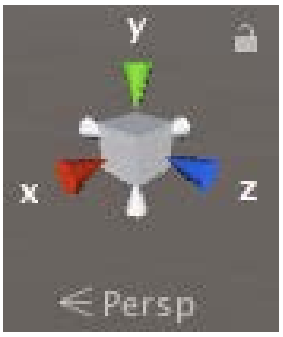
G) Μπάρα Κατάστασης (Status Bar). Παρέχει ειδοποιήσεις για διάφορες διεργασίες της Unity όπως τελευταία μήνυμα της κονσόλας, μπάρα προόδου για εργασίες που συμβαίνουν ασύγχρονα όπως εφαρμογή φωτισμού στις Σκηνές και δείκτης αναμονής για script που γίνονται compile.

2.δ Ανάλυση Όψης Σκηνής

Καθώς η Όψη Σκηνής είναι πολύ σημαντική, ίσως το σημαντικότερο σημείο της διεπαφής, για την ανάπτυξη ενός παιχνιδιού με την πλατφόρμα Unity και παρέχει μια πληθώρα από λειτουργίες που μπορεί να εκτελέσει θα κάνουμε μια εκτενέστερη ανάλυση της σε αυτήν την ενότητα. Αποτελεί έναν από τους δύο βασικούς πυλώνες στη δημιουργία του παιχνιδιού (ο άλλος είναι η γραφή κώδικα) και σε αυτή ο προγραμματιστής θα αφιερώσει πολύ χρόνο κατά το στάδιο της ανάπτυξης για αυτό και η κατανόηση κάποιων δυνατοτήτων της είναι απαραίτητη.

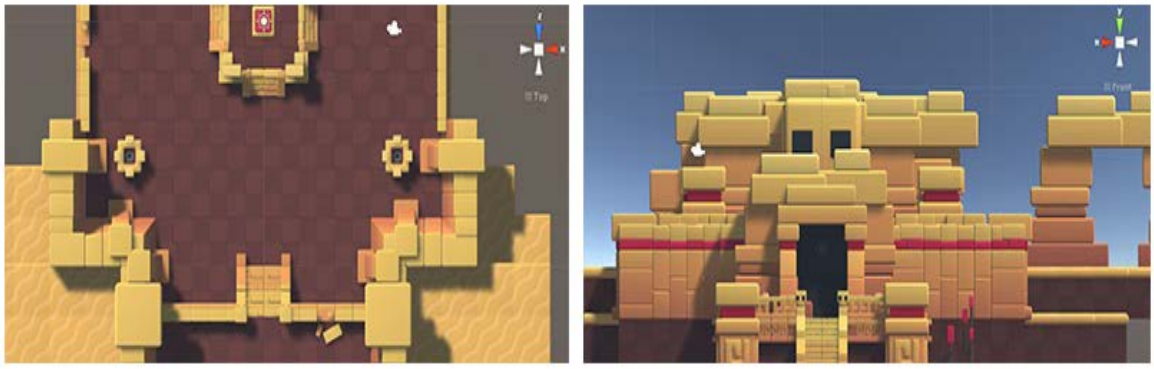
2.δ.i Πλοήγηση

Υπάρχουν πολλοί τρόποι για την πλοήγηση στη Σκηνή ανάλογα τον λόγο που θέλουμε (επισκόπηση ολόκληρης Σκηνής, εργασία πάνω σε λεπτομέρειες κα.) καθώς και πλήκτρα συντομεύσεων τα οποία βοηθούν στην εναλλαγή τους. Ένα βασικό στοιχείο είναι το Εργαλείο Σκηνής (Scene Gizmo) το οποίο μας δείχνει τον προσανατολισμό με τον οποίο βλέπουμε τη Σκηνή (Εικόνα 14).



Εικόνα 14: Το Εργαλείο Σκηνής (πηγή δημιουργός)

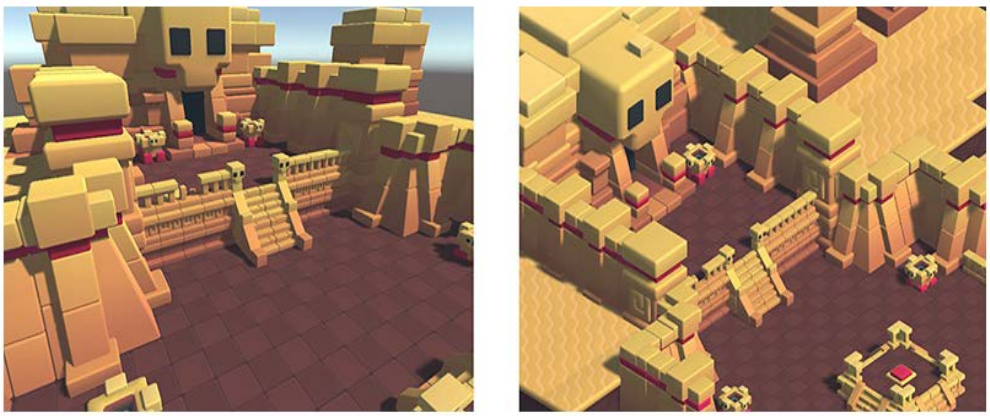
Το εργαλείο αυτό εμφανίζεται στην επάνω δεξιά γωνία της Όψης Σκηνής και δείχνει τον προσανατολισμό που βλέπουμε την Σκηνή. Κάνωντας αριστερό κλικ στους κόνους (x, y, z) που απεικονίζονται, αλλάζουμε άμεσα τον προσανατολισμό και βλέπουμε τη Σκηνή από τον συγκεκριμένο άξονα κάτι που μπορούμε να δούμε στην Εικόνα 15.



(i) (ii)

Εικόνα 15: Η ίδια Σκηνή από (i) τον άξονα y (ii) τον άξονα z [24]

Με δεξί κλικ στο εργαλείο εμφανίζεται μια λίστα από διαθέσιμες γωνίες θέασης. Κάνοντας κλικ είτε στον κύβο είτε στο κείμενο κάτω από αυτό ενεργοποιεί (Perspective) ή απενεργοποιεί (Orthographic/Isometric) την προοπτική από τον τρόπο που βλέπουμε την Σκηνή το οποίο είναι ιδιαίτερα χρήσιμο όταν έχουμε διαφορές στο ύψος των Αντικειμένων και θέλουμε να κάνουμε επισκόπηση μεγαλύτερου μέρος της Σκηνής χωρίς να παρεμβάλλονται εμπόδια (Εικόνα 16).



(i) (ii)

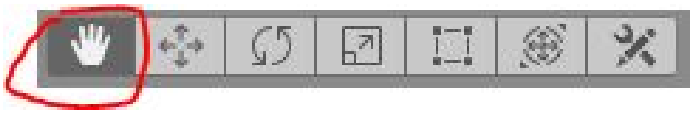
Εικόνα 16: Το ίδιο σημείο της Σκηνής (i) Perspective (ii) Orthographic [24]

Πατώντας shift και κάνοντας κλικ στο Εργαλείο το επαναφέρει στον προεπιλεγμένο προσανατολισμό. Το λουκέτο πάνω δεξία κλειδώνει/ξεκλειδώνει την αλλαγή προσανατολισμού. Στην περίπτωση που το παιχνίδι μας είναι σε 2D τότε δεν εμφανίζεται το συγκεκριμένο Εργαλείο.

Για την πλοήγηση στο περιβάλλον της Σκηνής η Unity παρέχει πολλές επιλογές:

1) Πλοήγηση με τα βελάκια. Πατώντας τα βελάκια μετακινούμε την κάμερα θέασης μέσα, έξω, αριστερά και δεξιά.

2) Πλοήγηση με "Το Χέρι". Όταν το συγκριμένο Εργαλείο (Εικόνα 17) είναι επιλεγμένο (είτε με κλικ στο εικονίδιο του στην Μπάρα Εργαλειών είτε με το πλήκτρο Q), κάνοντας κλικ σε ένα σημείο της Σκηνής και σέρνοντας τον κέρσορα μπορούμε να μετακινήσουμε την κάμερα θέασης.



Εικόνα 17: Το Εργαλείο "Χέρι" (πηγή δημιουργός)

2i) Με αυτό το Εργαλείο επιλεγμένο και πατώντας ταυτόχρονα το πληκτρο Alt η λειτουργία του αλλάζει και πλέον το αριστερό κλικ σε συνδυασμό με την μετακίνηση του κέρσορα περιστρέφει την κάμερα θέασης γύρω από το σημείο που έγινε το κλικ. Αυτή η επιλογή δεν είναι διαθέσιμη σε 2D παιχνίδια.

2ii) Αν αντί για αριστερό κλικ, κάνουμε δεξί κλικ η λειτουργία αλλάζει και πλέον με την μετακίνηση του κέρσορα κάνουμε zoom in/zoom out.

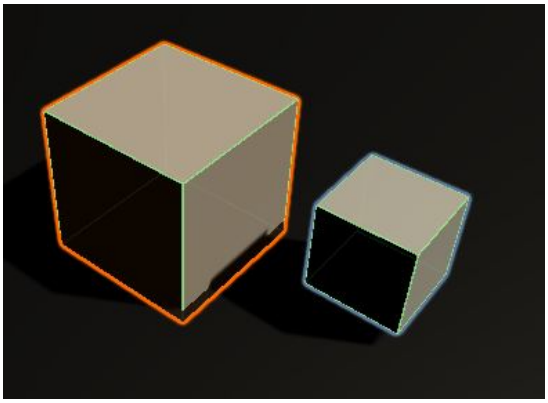
3) Λειτουργία Flythrough (Πλοήγηση σε Πρώτο Πρόσωπο). Έχοντας πατημένο το δεξί κλικ και χρησιμοποιώντας τα πλήκτρα WASD (αριστερά/δεξιά/μπροστά/πίσω) και QE (πάνω/κάτω) μπορούμε να μετακινήθουμε μέσα στη Σκηνή σε πρώτο πρόσωπο. Δεν παρέχεται σε λειτουργία Orthographic ή 2D.

Σε όλους τους παραπάνω τρόπους αν έχουμε πατημένο το πλήκτρο shift επιταχύνουμε την πλοήγηση.

Τέλος, μπορούμε να κεντράρουμε την κάμερα θέασης σε ένα Αντικείμενο επιλέγοντας το είτε στο Παράθυρο Ιεραρχίας είτε στην Όψη Σκηνής και με τον κέρσορα πάνω από την Όψη Σκηνής πατήσουμε το πλήκτρο F.

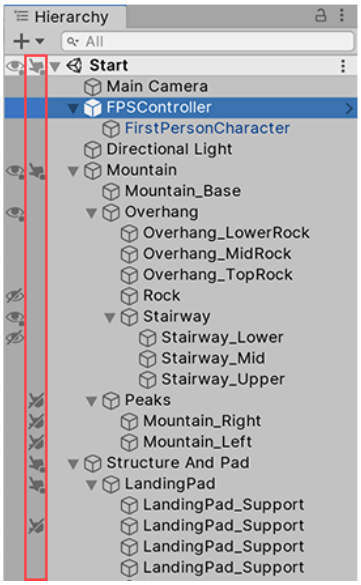
2.δ.ii Επιλογή Αντικειμένων

Η επιλογή ενός Αντικειμένου μπορεί να γίνει κάνοντας κλικ σε αυτό είτε στην Όψη Σκηνής είτε στο Παράθυρο Ιεραρχίας. Πολλαπλή επιλογή γίνεται έχοντας πατημένο το πλήκτρο shift καθώς κάνουμε κλικ στα Αντικείμενα. Αυτά που έχουμε επιλέξει ξεχωρίζουν από το προεπιλεγμένο περίγραμμα που εφαρμόζει σε αυτά η Unity, πορτοκαλί για κύρια αντικείμενα και μπλε για τα παιδιά τους (Εικόνα 18).



Εικόνα 18: Επιλεγμένο Αντικείμενο και το παιδί του (μικρός κύβος)
(πηγή δημιουργός)

Σε μεγάλα project με πολλά Αντικείμενα μπορεί να προκύψει μπέρδεμα και δυσκολία στην επιλογή αυτών που θέλουμε. Η Unity παρέχει την δυνατότητα να κάνουμε κάποια Αντικείμενα μη-επιλέξιμα κάτι που μπορούμε να ενεργοποιήσουμε στο Παράθυρο Ιεραρχίας (Εικόνα 19).

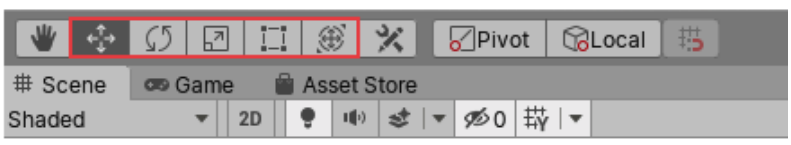


Εικόνα 19: Παράδειγμα δυνατότητας μη-επιλογής Αντικειμένων [25]

Με παρόμοιο τρόπο μπορούμε να ορίσουμε Αντικείμενα ως μη-εμφανίσιμα στην Όψη Σκηνής κάνοντας κλικ στη στήλη στα αριστερά της στήλης που φαίνεται στην παραπάνω εικόνα. Αυτή η πρακτική είναι πιο ασφαλής από το να απενεργοποιήσουμε το Αντικείμενο καθώς αποφεύγουμε καταλάθος αφαιρέσεις από την Σκηνή κατά τη διάρκεια της προσομοίωσης ή/και του τελικού project.

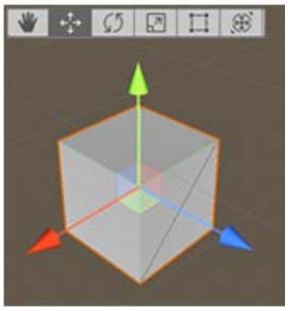
2.δ.iii Τοποθέτηση Αντικειμένων

Το μέγεθος, οι συντεταγμένες και ο προσανατολισμός ενός Αντικειμένου περιέχονται στο Συστατικό τους, Transform. Αυτό μπορεί να τροποποιηθεί είτε με απευθείας ανάθεση τιμών στο Παράθυρο Παρατήρησης είτε με τα εργαλεία που βρίσκονται στην Μπάρα Εργαλειών (Εικόνα 20) και χρήση του κέρσορα για άμεσες αλλαγές των Αντικειμένων στην Όψη Σκηνής.



Εικόνα 20: Εργαλεία τροποποίησης Αντικειμένου [26]

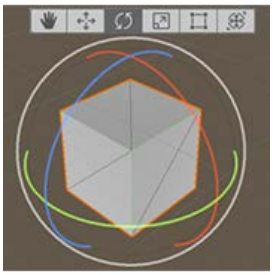
-Μετακίνηση (Move). Πλήκτρο συντόμευσης: W (Εικόνα 21).



Εικόνα 21: Μετακίνηση [26]

Υποστηρίζει ελεύθερη μετακίνηση του Αντικειμένου στη Σκηνή, μετακίνηση μόνο σε έναν άξονα (κλικ σε κάποιο βέλος) και μετακίνηση σε επίπεδο κρατώντας έναν άξονα σταθερό (κλικ σε κάποιο απο τα μικρά τετράγωνα στο κέντρο).

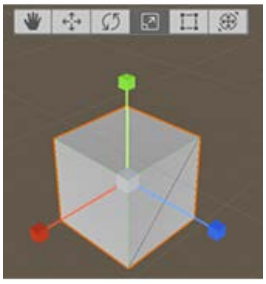
-Περιστροφή (Rotate). Πλήκτρο συντόμευσης: E (Εικόνα 22).



Εικόνα 22: Περιστροφή [26]

Περιστρέφει το Αντικείμενο γύρω από τον άξονα των x, y, z του (x-κόκκινο, y-πράσινο, z-μπλε). Ο εξωτερικός άσπρος κύκλος περιστρέφει το Αντικείμενο γύρω από τον z άξονα της Σκηνής.

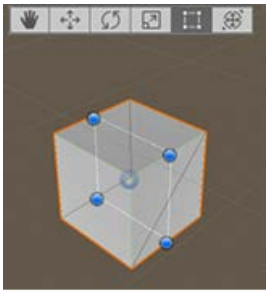
-Κλίμακα (Scale). Πλήκτρο συντόμευσης: R (Εικόνα 23).



Εικόνα 23: Κλίμακα [26]

Επιτρέπει την αλλαγή μεγέθους του αντικειμένου είτε σε συγκεκριμένο άξονα με κλικ στους χρωματισμένους κύβους είτε σε όλους τους άξονες ταυτόχρονα με κλικ στον άσπρο κύβο στο κέντρο.

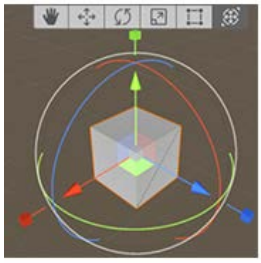
-RectTransform. Πλήκτρο συντόμευσης: R (Εικόνα 24).



Εικόνα 24: RectTransform [26]

Αυτή η λειτουργία συνήθως χρησιμοποίειται όταν δουλεύουμε σε κάποιο 2D project και βοηθάει στην τροποποίηση εικόνων και στοιχείων διεπαφής χρήστη. Συνδυάζει τις λειτουργίες της μετακίνησης, της περιστροφής και της κλίμακας. Με κλικ στο παραλληλόγραμμο μπορούμε να μετακινήσουμε το Αντικείμενο. Με κλικ σε κάποια ακμή ή γωνία μπορούμε να αλλάξουμε το μέγεθος ανάλογα. Για να πραγματοποιήσουμε περιστροφή πρέπει να βάλουμε τον κέρσορα λίγο πάνω από κάποια από τις γωνίες του παραλληλογράμμου μέχρι να εμφανιστεί ο ανάλογος κέρσορας.

-Transform. Πλήκτρο συντόμευσης: T (Εικόνα 25).



Εικόνα 25: Transform [26]

Περιλαμβάνει τα εργαλεία Μετακίνηση, Περιστροφή και Κλίμακα σε ένα.

Για πιο ακριβή μεταβολή του Transform ενός Αντικειμένου η Unity παρέχει λειτουργία προσκόλλησης (snaping) τριών τρόπων:

-Καθολικού Πλέγματος (World Grid). Μεταβολή με προσκόλληση στους άξονες x,y,z του πλέγματος της Όψης Σκηνής.

-Επιφάνειας (Surface). Χρησιμοποιώντας το εργαλείο Μετακίνησης και πατώντας shift και ctrl γίνεται προσκόλληση του Αντικειμένου στο σημείο τομής με οποιονδήποτε Συγκρουστήρα (Collider).

-Κορυφής (Vertex). Με ενεργό το εργαλείο Μετακίνησης και πατώντας το πλήκτρο V μπορούμε να διαλέξουμε μία κορυφή από το πλέγμα ενός Αντικειμένου και να την προσκολλήσουμε σε κάποια άλλη. Μπορεί να συνδυαστεί με την προσκόλληση Επιφάνειας αν έχουμε πατημένα τα πλήκτρα shift και ctrl.

2.δ.iv Μπάρα Ελέγχου Όψης Σκηνής

Στην κορυφή αυτής της Όψης υπάρχει μία μπάρα που περιέχει διάφορες επιλογές για τον τρόπο θέασης της Σκηνής καθώς και για την ενεργοποίηση/απενεργοποίηση φωτισμού και ήχου (Εικόνα 26). Αφορά αποκλειστικά την διαδικασία της ανάπτυξης και δεν επηρεάζει το τελικό αποτέλεσμα.

Εικόνα 26: Μπάρα Ελέγχου Όψης Σκηνής [27]

Ακολουθεί περιγραφή μερικών σημαντικών λειτουργιών που παρέχει.

1) Μενού τρόπου σχεδίασης: Περιέχει επιλογές για τον τρόπο σχεδίασης της Σκηνής όπως απεικόνιση Αντικειμένων με πλέγματα, απεικόνιση με σιλουέτες, εμφάνιση ή όχι σκιών κα.

2) Διακόπτες διαστάσεων, φωτισμού και ήχου: Από τα αριστερά προς τα δεξία ο πρώτος διακόπτης εναλλάσσει μεταξύ διαστάσεις 3D και 2D, ο δεύτερος ενεργοποιεί/απενεργοποιεί τα στοιχεία φωτισμού και ο τρίτος τα στοιχεία ήχου.

3) Μενού εφέ, διακόπτης ορατότητας και μενού πλέγματος: Το μενού εφέ παρέχει δυνατότητες ενεργοποίησης/απενεργοποίησης ορισμένων εφέ όπως εμφάνιση υφής ουρανού (Skybox), φαινόμενο ομίχλης κα. Λειτουργεί και ως διακόπτης ενεργοποίησης/απενεργοποίησης όλων τον εφέ. Ο διακόπτης ορατότητας δείχνει πόσα Αντικείμενα έχουν οριστεί ως μη-εμφανίσιμα και σε περίπτωση που είναι απενεργοποιημένος, η Unity αγνοεί τις ρυθμίσεις ορατότητας. Το μενού πλέγματος ενεργοποιεί/απενεργοποιεί την εμφάνιση του πλέγματος απο την Όψη Σκηνής. Επίσης παρέχει την επιλογή για τον άξονα που εμφανίζεται το πλέγμα.

4) Ρυθμίσεις Κάμερας, εμφάνιση Εργαλειών, πλάισιο αναζήτησης: Στις ρυθμίσεις Κάμερας υπάρχουν επιλογές για την διαμόρφωση της Κάμερας Όψης Σκηνής όπως οπτικό πεδίο, τρόπος και απόσταση αποκοπής, ταχύτητα μετακίνησης Κάμερας κα. Η εμφάνιση Εργαλειών περιλαμβάνει μία πληθώρα επιλογών που αφορούν τον τρόπο εμφάνισης εικονιδίων, αντικειμένων και εργαλείων αλλά και την απενεργοποίηση εμφάνισης αυτών. Στο πλαίσιο αναζήτησης μπορεί να γίνει φιλτράρισμα των στοιχείων που υπάρχουν στην Όψη Σκηνής με το όνομά τους ή/και με τον τύπο τους. Τα στοιχεία που αντιστοιχούν εμφανίζονται επίσης στο Παράθυρο Ιεραρχίας.

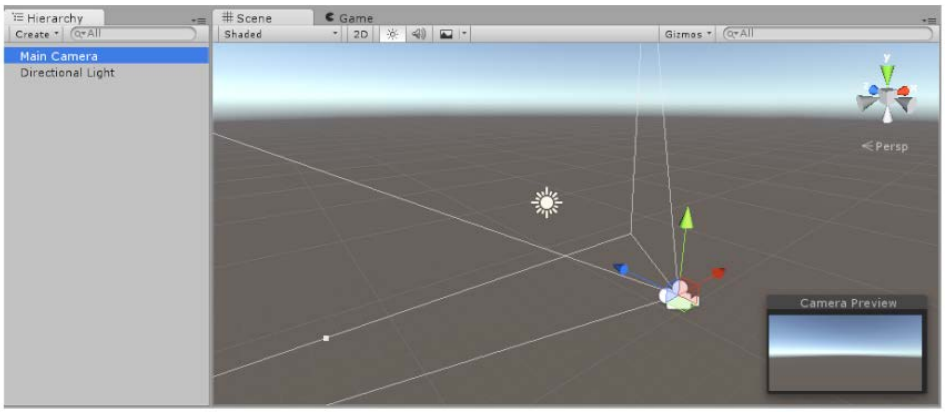
2.ε Στοιχεία Και Αρχές Λειτουργίας Της Unity

Στις προηγούμενες ενότητες κάναμε ανάλυση της διεπαφής Unity έτσι ώστε να κατανοήσει ο χρήστης πως μπορεί να έχει πρόσβαση σε κάποιες από τις βασικότερες λειτουργίες που του παρέχει η πλατφόρμα και ποια η χρησιμότητα τους. Πέρα όμως από το περιβάλλον σχεδίασης, ένα εξίσου βασικό κομμάτι είναι τα στοιχεία που συμβάλλουν

στη δημιουργία ενός παιχνιδιού, η διαμόρφωσή τους, ο ρόλος τους καθώς και η αλληλεπίδρασή τους. Η εξοικίωση με τα στοιχεία αυτά σε συνδυασμό με τη γνώση της διεπαφής θα βοηθήσουν στην κατανόηση της διαδικασίας κατασκεύης ενός παιχνιδιού.

2.ε.i Σκηνές (Scenes)

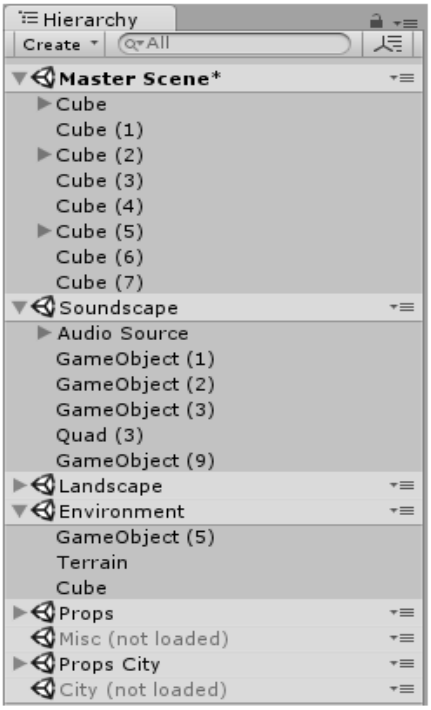
Οι Σκηνές είναι το μέρος όπου γίνεται η επεξεργασία περιεχομένου και περιλαμβάνουν είτε ολόκληρο είτε κάποιο μέρος του παιχνιδιού. Μία καλή αναλογία είναι αν σκεφτούμε ότι κάθε Σκηνή είναι και ένα διαφορετικό επίπεδο στο παιχνίδι, η κάθε μία με το δικό της περιβάλλον, χαρακτήρες, εμπόδια κλπ. Ένας εύκολος τρόπος να δημιουργήσουμε μία Σκηνή είναι είτε κάνοντας δεξί κλικ στο Παράθυρο Εργασίας και στη συνέχεια Create > Scene είτε κάνοντας κλικ πάνω αριστερά στο παράθυρο της Unity, File > New Scene (ctrl+N). Με διπλό κλικ σε ένα αρχείο Σκηνής το ανοίγουμε στην Όψη Σκηνής για επεξεργασία (Εικόνα 27). Μπορούμε να δούμε τα αρχεία Σκηνής στο Παράθυρο Εργασίας (Εικόνα 28).



Εικόνα 27: Προεπιλεγμένο δείγμα Σκηνής Unity (πηγή δημιουργός)

Εικόνα 28: Αρχεία Σκηνής στο Παράθυρο Εργασίας
(πηγή δημιουργός)

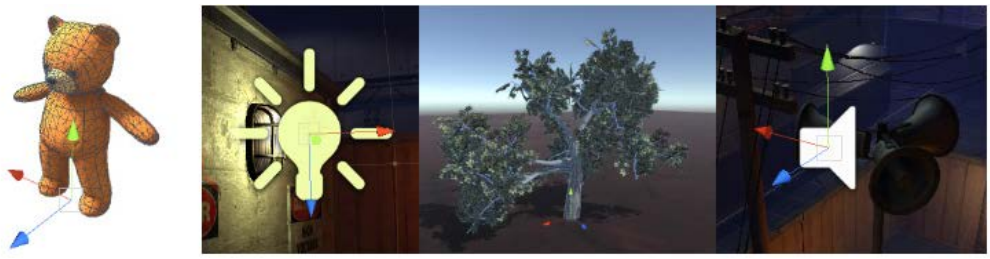
Μπορούμε να επεξεργαστούμε ταυτόχρονα παραπάνω από μία Σκηνές σέρνοντας με τον κέρσορα ένα αρχείο Σκηνής από το Παράθυρο Εργασίας στο Παράθυρο Ιεραρχίας (Εικόνα 29) και με δεξί κλικ > Set Active Scene να επιλέξουμε σε ποια Σκηνή θέλουμε να δουλέψουμε.



Εικόνα 29: Παράδειγμα πολλαπλών Σκηνών στο Παράθυρο Ιεραρχίας [28]

2.ε.ii Αντικείμενα (GameObjects)

Τα Αντικείμενα είναι το σημαντικότερο στοιχείο στην πλατφόρμα Unity. Οτιδήποτε υπάρχει στο παιχνίδι είναι ένα Αντικείμενο, από χαρακτήρες και συλλεκτικά αντικείμενα έως κάμερες και φωτισμός (Εικόνα 30). Τα Αντικείμενα όμως δεν μπορούν να κάνουν τίποτα από μόνα τους αλλά χρειάζονται ιδιότητες τις οποίες ονομάζουμε Συστατικά (Components). Μπορούμε να φανταστούμε τα Αντικείμενα σαν μια μαριονέτα και κάθε Συστατικό τους σαν μια χορδή που ελέγχει διαφορετικό μέρος του σώματος της. Η πιο απλή μορφή Αντικειμένου περιέχει μόνο το Συστατικό Transform και για να το δημιουργήσουμε κάνουμε δεξί κλικ στο Παράθυρο Ιεραρχίας και Create Empty. Θα εμβαθύνουμε στα Αντικείμενα στην επόμενη ενότητα.



Εικόνα 30: Τέσσερα διαφορετικά είδη Αντικειμένων [29]

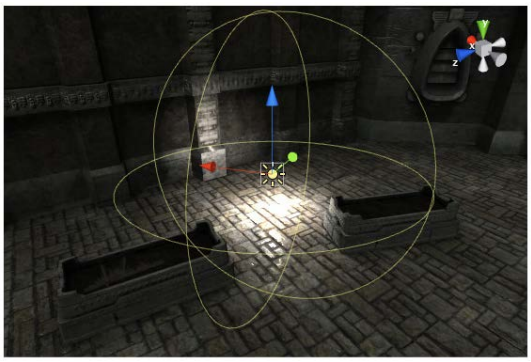
2.ε.iii Φώτα (Lights)

Τα Φώτα είναι ένας ιδιάιτερος τύπος Αντικειμένου και είναι απαραίτητο κομμάτι κάθε Σκηνής καθώς προσδιορίζουν τον τόνο του 3D περιβάλλοντος. Δημιουργούμε ένα με δεξί κλικ στο Παράθυρο Ιεραρχίας > Light και επιλέγουμε τον τύπο Φωτός που θέλουμε. Στην Εικόνα 31 βλέπουμε ένα Συστατικό Φωτός.

Εικόνα 31: Παράδειγμα Συστατικού Φωτός (πηγή δημιουργός)

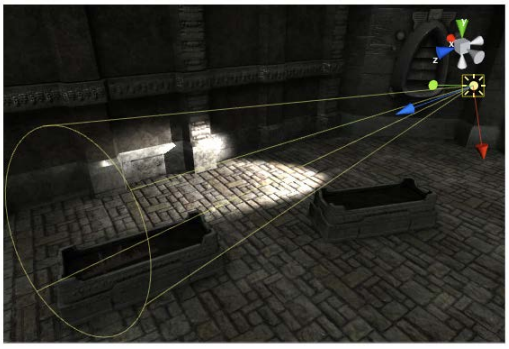
Στο Συστατικό Φωτός μπορούμε να επιλέξουμε τον τύπο του και ανάλογα με αυτόν να μεταβάλλουμε παραμέτρους όπως την απόσταση μεταφοράς, την γωνία βάσης του Φωτός, το χρώμα, την ένταση, τον τύπο σκιάς κα. Υπάρχουν τέσσερις τύποι Φωτός:

-Σημείου (Point). Τοποθετείται σε ένα σημείο στη Σκηνή και εκπέμπει φως προς όλες τις κατευθύνσεις με ίση ένταση. Χρησιμοποιείται συνήθως για να προσομοιώσει λάμπες (Εικόνα 32).



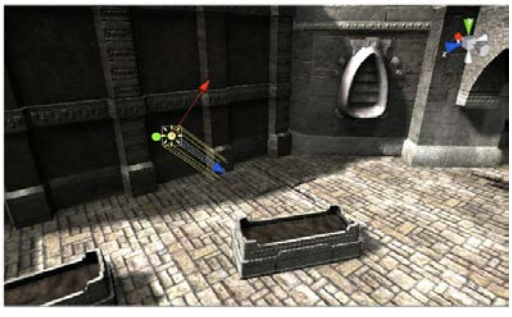
Εικόνα 32: Φως Σημείου [30]

-Προβολέα (Spot). Τοποθετείται σε ένα σημείο και εκπέμπει φως σε κωνικό σχήμα. Χρησιμοποιείται για προσομοίωση φακών, φωτών αυτοκινήτων κλπ (Εικόνα 33).



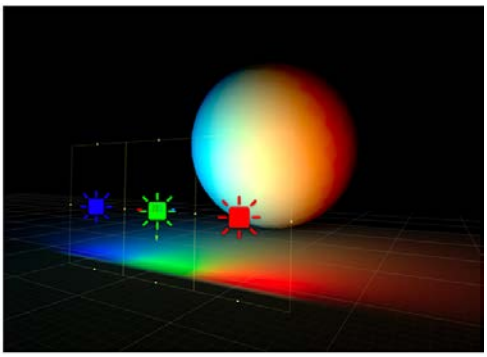
Εικόνα 33: Φως Προβολέα [30]

-Κατεύθυνσης (Directional). Βρίσκεται σε πολύ μεγάλη απόσταση και εκπέμπει φως προς μία συγκεκριμένη κατεύθυνση φωτίζοντας όλα τα αντικείμενα στη Σκηνή χωρίς μείωση έντασης. Συνήθως προσομοιώνει τον ήλιο (Εικόνα 34). Κάθε νέα Σκηνή περιέχει αυτόν τον τύπο Φωτός.



Εικόνα 34: Φως Κατεύθυνσης [30]

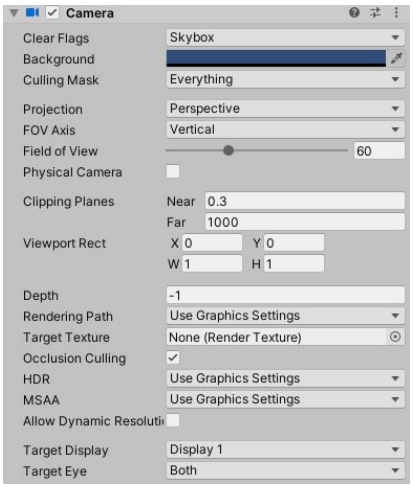
-Περιοχής (Area). Προσδιορίζεται από ένα ορθογώνιο παραλληλόγραμμο στη Σκηνή και εκπέμπει φως ομοιόμορφα από την επιφάνειά του προς μία του πλευρά. Χρησιμοποιείται για πιο ρεαλιστική προσομοίωση φωτών δρόμου, σπιτιού κλπ (Εικόνα 35).



Εικόνα 35: Φως Περιοχής [30]

2.ε.iv Κάμερες

Ακόμα ένας ξεχωριστός τύπος Αντικειμένου είναι οι Κάμερες. Όπως οι κάμερες στις ταινίες χρησιμοποιούνται για την απεικόνιση της ιστορίας στον θεατή, έτσι και οι Κάμερες στη Unity απεικονίζουν τον κόσμο του παιχνιδιού στον παίχτη. Μπορούμε σε κάθε Σκηνή να έχουμε παραπάνω από μία Κάμερες και να δείχνουν είτε ολόκληρη τη Σκηνή είτε μέρος αυτής αλλά πάντα πρέπει να υπάρχει τουλάχιστον μία. Έχουμε τη δυνατότητα να τους δώσουμε κίνηση αλλά και να τις ελέγξουμε με δυνάμεις φυσικής ανάλογα με το στυλ παιχνιδιού που δημιουργούμε. Κάνοντας κλικ σε ένα Αντικείμενο Κάμερας, εμφανίζεται στην Όψη Σκηνής η εικόνα που δείχνει στο παράθυρο Camera Preview. Στην Εικόνα 36 βλέπουμε ένα Συστατικό Κάμερας.



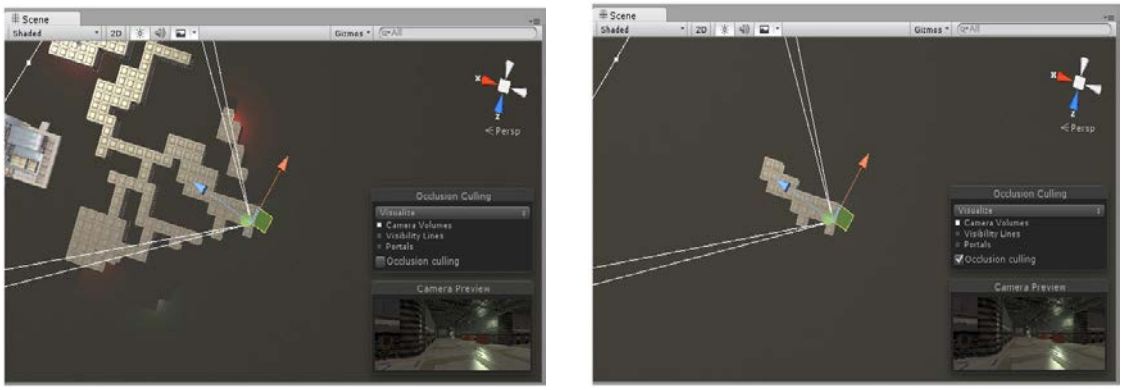
Εικόνα 36: Παράδειγμα Συστατικού Κάμερας
(πηγή δημιουργός)

Το παραπάνω παράδειγμα Συστατικού Κάμερας κάνει κάποιο Αντικείμενο να λειτουργεί ως κάμερα με τον ενσωματωμένο στη Unity βασικό τρόπο απόδοσης γραφικών (Built-in Render Pipeline). Υπάρχουν άλλοι δύο τρόποι απόδοσης γραφικών (Universal Render Pipeline και High Definition Render Pipeline) τα οποία παρέχουν το καθένα περισσότερες δυνατότητες και χρησιμοποιούνται από πιο εξοικειωμένους δημιουργούς για την ανάπτυξη προηγμένων γραφικών.

Κάποιες δυνατότητες που παρέχει το βασικό Συστατικό Κάμερας περιλαμβάνουν την επιλογή ύπαρξης Skybox ή σταθερού χρώματος και την επιλογή αυτού, την μάσκα επιλογής προβολής στοιχείων, ενεργοποίηση/απενεργοποίηση προοπτικής, την προσομοίωση πραγματικής κάμερας (Physical Camera), την απόσταση αποκοπής, την σειρά σχεδιασμού (Depth, μεγαλύτερες τιμές σχεδιάζονται μετά από μικρότερες, πρακτική χρήσιμη για προβολή διεπαφής χρήστη) κα.

Μία ακόμα πολύ δυνατή λειτουργία που παρέχουν οι Κάμερες στη Unity είναι το Occlusion Culling. Από προεπιλογή η Unity δεν κάνει render Αντικείμενα και περιβάλλον που βρίσκονται εκτός της όψης της Κάμερας όπως μπορούμε να δούμε στην Εικόνα 37. Στην περίπτωση που το Occlusion Culling είναι ενεργοποιημένο αποτρέπει την περαιτέρω εκτέλεση υπολογισμών που αφορούν το rendering Αντικειμένων που επικαλύπτονται εξ

ολοκλήρου κατά την όψη από άλλα. Η συγκεκριμένη πρακτική χρησιμοποιείται συνήθως στην ανάπτυξη μεγαλύτερων σε μέγεθος παιχνιδιών έτσι ώστε να γίνει οικονομία πόρων που χρειάζονται από τον επεξεργαστή και την κάρτας γραφικών.

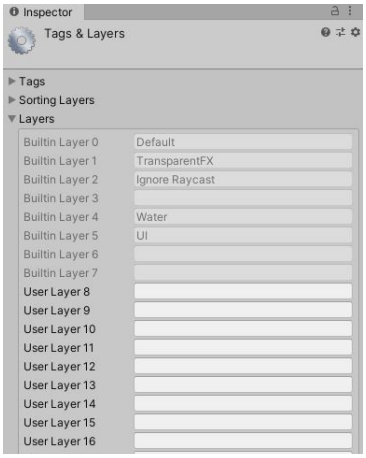


(i) (ii)

Εικόνα 37: Render Σκηνής στη Unity (i) χωρίς Occlusion Culling (ii) με Occlusion Culling [31]

2.ε.ν Επίπεδα Σχεδίασης (Layers)

Τα Layers στη Unity καθορίζουν ποια Αντικείμενα μπορούν να αλληλεπιδράσουν με ποια χαρακτηριστικά αλλά και μεταξύ τους. Χρησιμοποιούνται συχνά από Κάμερες για render ενός μέρους της Σκηνής καθώς και για την δημιουργία ακριβέστερων Συγκρούσεων. Για πρόσβαση στο παράθυρο Ετικέτες και Επίπεδα Σχεδίασης (Tags and Layers) που μας επιτρέπει την προσθαφαίρεση και την επεξεργασία των ονομάτων Επιπέδων, κάνουμε κλικ στο μενού Layers που βρίσκεται πάνω δεξία στο παράθυρο της Unity και στη συνέχεια Edit Layers (Εικόνα 38). Μπορούμε να ορίσουμε το Επίπεδο κάποιου Αντικειμένου από το μενού που βρίσκεται στην κορυφή του Παράθυρου Παρατήρησης.



Εικόνα 38: Το μενού Ετικέτες και Επίπεδα Σχεδίασης
(πηγή δημιουργός)

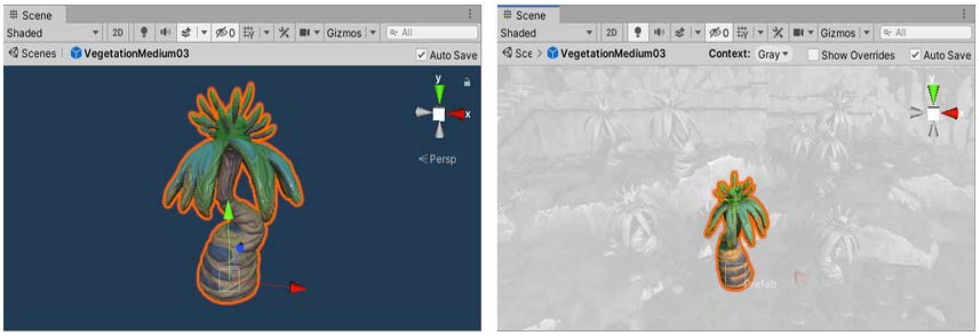
2.ε.vi Προκατασκευές (Prefabs)

Το σύστημα Προκατασκευών της Unity δίνει τη δυνατότητα της δημιουργίας, διαμόρφωσης και αποθήκευσης ενός Αντικειμένου συμπεριλαμβάνοντας τα Συστατικά του και τις τιμές αυτών ως επαναχρησιμοποιούμενο αρχείο. Ουσιαστικά λειτουργεί σαν ένα καλούπι από το οποίο μπορούν να δημιουργηθούν άμεσα στιγμιότυπα Προκατασκευασμένων Αντικειμένων στη Σκηνή. Είναι καλύτερο από το να κάνουμε απλά αντιγραφή και επικόλληση Αντικείμενα καθώς οποιαδήποτε αλλαγή στο αρχείο Προκατασκευής μεταφέρεται στα Προκατασκευασμένα Αντικείμενα που έχουμε δημιουργήσει και έτσι παραμένουν ενημερωμένα. Μία καλή πρακτική στην ανάπτυξη παιχνιδιών με Unity είναι να δημιουργούμε Προκατασκευασμένα αρχεία Αντικειμένων που έχουμε σκοπό να χρησιμοποιήσουμε περισσότερες από μία φορές. Μία ακόμη περίπτωση που γίνεται χρήση των Prefab είναι όταν θέλουμε να δημιουργούμε Αντικείμενα κατά την εκτέλεση του παιχνιδιού τα οποία δεν υπήρχαν από την αρχή στη Σκηνή όπως αναβαθμίσεις (powerups), βλήματα, χαρακτήρες σε συγκεκριμένες στιγμές κατα τη διάρκεια της ιστορίας κα.

Για να δημιουργήσουμε ένα Prefab το μόνο που πρέπει να κάνουμε είναι σύρουμε το Αντικείμενο που επιθυμούμε από το Παράθυρο Ιεραρχίας στο Παράθυρο Εργασίας και το Αντικείμενο με τα Συστατικά του και τα Αντικείμενα-παιδιά του γίνεται ένα αρχείο

Προκατασκευής και ξεχωρίζουν στον κατάλογο αρχείων από τον χαρακτηρισικό μπλε κύβο δίπλα από το όνομά τους. Πλέον το Αντικείμενο το οποίο κάναμε Prefab είναι “στιγμιότυπο” (instance) αυτού και ξεχωρίζουν στο Παράθυρο Ιεραρχίας από το μπλε χρώμα στο όνομά τους και τον μπλε κύβο. Μπορούμε τώρα, κάνοντας την ανάποδη διαδικασία να δημιουργήσουμε περισσότερα στιγμιότυπα του Prefab στη Σκηνή σέρνοντας το αρχείο Προκατασκευής είτε στο Παράθυρο Ιεραρχίας είτε στην Όψη Σκηνής.

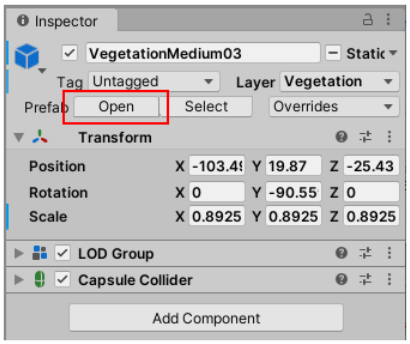
Υπάρχει η δυνατότητα μετατροπής ενός Prefab είτε σε απομόνωση (isolation) κατά την οποία αποκρύπτεται η υπόλοιπη Σκηνή και εμφανίζεται μόνο το Προκατασκευασμένο Αντικείμενο (Εικόνα 39i), είτε στο πλαίσιο (context) στο οποίο παραμένει ορατή η υπόλοιπη Σκηνή αλλά είναι κλειδωμένη η επεξεργασία της και η οποία έχει περιορισμένες δυνατότητες σε σχέση με την επεξεργασία σε απομόνωση (Εικόνα 39ii).



(i) (ii)

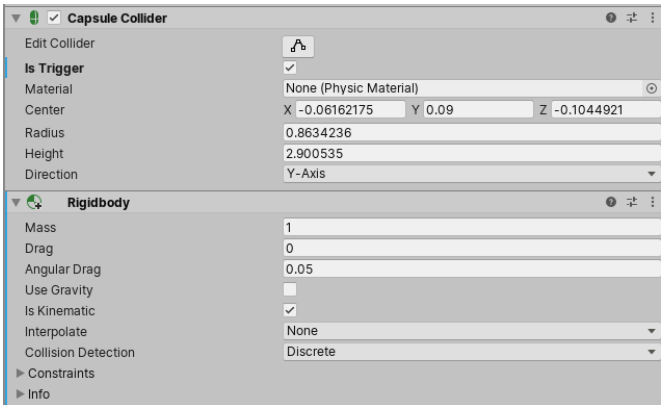
Εικόνα 39: Επεξεργασία Prefab σε (i) απομόνωση (ii) πλαίσιο [32]

Για να ανοίξουμε την επεξεργασία σε απομόνωση κάνουμε διπλό κλικ στο αρχείο Προκατασκευής ή το επιλέγουμε και κάνουμε κλικ στο Open Prefab που εμφανίζεται στο Παράθυρο Παρατήρησης. Για την επεξεργασία σε πλαίσιο, επιλέγουμε το στιγμιότυπο του Prefab που θέλουμε στο Παράθυρο Ιεραρχίας και κάνουμε κλικ στο Open που εμφανίζεται στο Παράθυρο Παρατήρησης (Εικόνα 40) ή χρησιμοποιούμε το βελάκι που υπάρχει στα δεξιά του ονόματός του. Πρέπει να αναφερθεί ότι οι αλλαγές που συμβαίνουν στους δύο αυτούς τρόπους επεξεργασίας μπορούν να αναιρεθούν μόνο όσο ο χρήστης βρίσκεται σε αυτούς και σε περίπτωση εξόδου από την επεξεργασία αφαιρούνται από το ιστορικό αναίρεσης.



Εικόνα 40: Άνοιγμα επεξεργασίας σε πλάισιο [32]

Όταν μετατρέπουμε ένα Prefab όλες οι αλλαγές αντικατοπτρίζονται σε κάθε ένα από τα στιγμιότυπά του. Παρόλα αυτά, υπάρχει και η επιλογή για απευθείας μετατροπή ενός μεμονομένου στιγμιοτύπου το οποίο δημιουργεί την λεγόμενη "επικάλυψη στιγμιοτύπου" (instance override), όπως π.χ. στην περίπτωση που έχουμε ένα Prefab για διαστημόπλοια και θέλουμε να αναθέσουμε διαφορετική τιμή ταχύτητας σε έναν συγκεκριμένο. Επικάλυψη στιγμιοτύπου μπορούμε να έχουμε με αλλαγή σε κάποια τιμή πεδίου, προσθήκη/αφαίρεση Συστατικων Αντικειμένων και προσθήκη παιδιών στα Αντικείμενα. Δεν μπορούμε όμως να αλλάξουμε γονέα στο Αντικείμενο καθώς και να αφαιρέσουμε Αντικείμενο που είναι μέλος του συγκεκριμένου Prefab (μπορούμε να απενεργοποιήσουμε το οποίο λειτουργεί σαν υποκατάστατο). Κοιτώντας το Παράθυρο Παρατήρησης διακρίνουμε εύκολα τα πεδία στα οποία υπάρχει επικάλυψη στιγμιοτύπου από την έντονη γραφή και από τα σύμβολα + και – που δηλώνουν προσθήκη και αφαίρεση στοιχείου αντίστοιχα (Εικόνα 41).



Εικόνα 41: Παράδειγμα instance override. Έχει γίνει αλλαγή σε πεδία του Capsule Collider και έχει προστεθεί το Συστατικό RigidBody [33]

Η επικάλυψη στιγμιοτύπου υπερέχει των όποιων αλλαγών γίνονται στη βάση του Prefab για αυτό καλό είναι να μην χρησιμοποιείται σε πολλά καθώς σε μεγαλύτερα project είναι δύσκολο να υπάρχει έλεγχος όλων των διαφορών. Τέλος, η Unity δεν θεωρεί επικάλυψη τις διαφορετικές τιμές στα πεδία του Transform κάθε στιγμιοτύπου καθώς είναι εξαιρετικά σπάνια η περίπτωση να χρειαζόμαστε στο ίδιο σημείο και με τον ίδιο προσανατολισμό παραπάνω από ένα στιγμιότυπα.

Στην Εικόνα 40 παρατηρούμε ότι στα δεξιά του Open υπάρχει το μενού Overrides το οποίο εμφανίζει όλες τις επικαλύψεις που υπάρχουν στο συγκεκριμένο στιγμιότυπο. Από αυτό το μενού μπορούν να εφαρμοστούν οι επικαλύψεις αυτές στη βάση του Prefab αλλά και την απαλοιφή των επικαλύψεων αυτών ώστε να επιστρέψει το στιγμιότυπο στη αρχική μορφή του Prefab.

Μία ακόμη δυνατότητα που παρέχει η Unity είναι τα εμφωλευμένα Prefabs (Nested Prefabs) τα οποία αποτελούν μέρος Προκατασκευασμένων Αντικειμένων ώντας και τα ίδια Προκατασκευασμένα Αντικείμενα. Ένα εμφωλευμένο Prefab γίνεται με παρόμοιο τρόπο με την δημιουργία παιδιών Αντικειμένων. Εφόσον έχουμε το Prefab που θέλουμε να κάνουμε nest μπορούμε να το σύρουμε στο Prefab που θέλουμε να κάνουμε γονέα στο Παράθυρο Ιεραρχίας.

Τέλος, μία ακόμη μορφή Prefab είναι οι Παραλλαγές Prefab (Prefab Variants). Χρησιμοποιούνται όταν θέλουμε να έχουμε διαφορετικές εκδοχές της βάσης του Prefab αλλά δεν θέλουμε να αλλάξουμε την βάση και δεν θέλουμε να κάνουμε μεμονωμένες επικαλύψεις στιγμιοτύπων καθώς είναι για δημιουργία πολλαπλών Αντικειμένων. Στο παράδειγμα με το διαστημόπλοιο, το συγκεκριμένο είδος Prefab χρησιμοποιείται γιατί αντί για ένα μόνο με διαφορετική ταχύτητα θέλουμε να δημιουργήσουμε περισσότερα και δεν χρειάζεται να ξοδέψουμε χρόνο στο να φτιάξουμε από την αρχή ένα καινούριο Prefab που η μόνη διαφορά του θα είναι στην ταχύτητα. Επίσης μπορούμε να δημιουργήσουμε ενα Prefab Variant για διαστημόπλοια που εκτοξεύουν πυραύλους κλπ. Αυτή η έννοια είναι παρόμοια με την κληρονομικότητα των αντικειμένων στη Java. Ένας εύκολος τρόπος να δημιουργήσουμε μία παραλλαγή ενός Prefab είναι κάνοντας δεξί κλικ

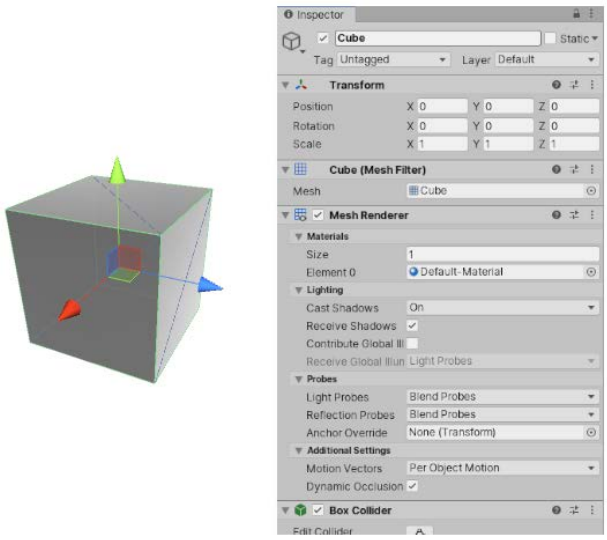
στο Prefab που θέλουμε σαν βάση και στη συνέχεια Create > Prefab Variant το οποίο δημιουργεί ένα νέο Prefab που αρχικά δεν έχει διαφορές σε σχέση με την βάση.

2.ζ Ανάλυση Αντικειμένου

Όπως αναφέραμε παραπάνω το Αντικείμενο (GameObject) είναι το πιο σημαντικό στοιχείο στην Unity καθώς είναι το θεμέλιο πάνω στο οποίο χτίζονται οι Σκηνές και τα παιχνίδια ολόκληρα. Οτιδήποτε υπάρχει μέσα σε μία Σκηνή στην πλατφόρμα πρέπει να είναι “περιτυλιγμένο” από ένα από αυτά τα Αντικείμενα (Εικόνα 42) που στην ουσία τους είναι απλά και βαρετά “δοχεία” τα οποία παρόλα αυτά έχουν μεγάλες δυνατότητες επέκτασης με σκοπό να δημιουργηθούν σύνθετες λειτουργίες ή γραφικά.

Εικόνα 42: Αντικείμενα σε μία Σκηνή
(πηγή δημιουργός)

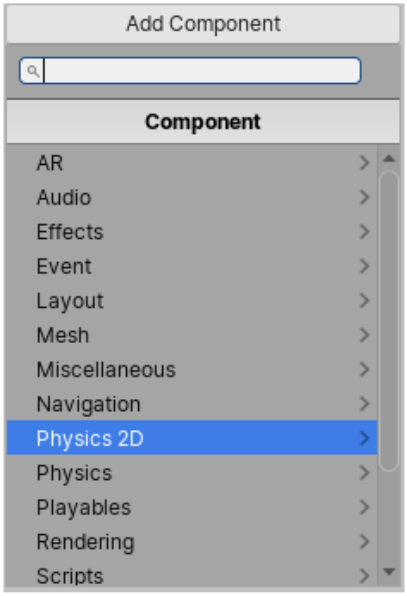
Από μόνα τους δεν προσφέρουν κάτι για αυτό χρειάζονται τα Συστατικά (Components) που θα τους προσδώσουν λειτουργικότητα και συνήθως είναι scripts γραμμένα σε γλώσσα C# ή Javascript. Η επικρατέστερη γλώσσα προγραμματισμού στη Unity είναι η C# καθώς παρέχει περισσότερες λειτουργίες με πιο εύκολο τρόπο για την πλατφόρμα και θα αναλυθεί στην επόμενη ενότητα. Ανάλογα με το τι θέλουμε να παριστάνει ένα Αντικείμενο θα πρέπει να δώσουμε τον κατάλληλο συνδυασμό Συστατικών όπως μπορούμε να δούμε στην Εικόνα 43. Αρκετά από αυτά παρέχει έτοιμα η Unity.



Εικόνα 43: Παράδειγμα Αντικειμένου Κύβος με Συστατικά για συντεταγμένες, σχήμα, κρούση κλπ. [29]

2.ζ.i Συστατικά (Components)

Τα Συστατικά είναι η “ραχοκοκαλιά” των Αντικειμένων και των συμπεριφορών αυτών στο παιχνίδι καθώς είναι τα λειτουργικά τους κομμάτια. Κάθε Αντικείμενο μπορεί να είναι δοχείο για πολλά διαφορετικά Συστατικά. Εξ ορισμού όλα τα Αντικείμενα περιέχουν το Συστατικό Transform που καθορίζει τη θέση, το μέγεθος και τον προσανατολισμό του Αντικειμένου και χωρίς αυτό δεν θα υπήρχε στον χώρο της Σκηνής. Στο Παράθυρο Παρατήρησης μπορούμε να δούμε όλα τα Συστατικά που υπάρχουν στο επιλεγμένο Αντικείμενο και να προσθέσουμε νέα ή να τα αφαιρέσουμε. Στο συγκεκριμένο Παράθυρο μπορούμε επίσης να επεξεργαστούμε τα πεδία και τις ιδιότητες των Συστατικών. Για να προσθέσουμε κάποιο Συστατικό πατάμε το κουμπί Add Component που βρίσκεται στο κάτω μέρος του Παραθύρου Παρατήρησης και από τον περιηγητή Συστατικών που εμφανίζεται (Εικόνα 44) επιλέγουμε αυτό που θέλουμε είτε επιλέγοντας κατηγορία είτε χρησιμοποιώντας την αναζήτηση.



Εικόνα 44: Περιηγητής Συστατικών
(πηγή δημιουργός)

Παραδείγματα χρήσιμων έτοιμων Συστατικών:

-Rigidbody: Προσομοιώνει ρεαλιστικές δυνάμεις φυσικής που ενεργούν πάνω στο Αντικείμενο και εντοπίζει κρούσεις με box colliders

-Box Collider: Ενεργοποιεί την δυνατότητα εντοπισμού κρούσεων με το Αντικείμενο

-Mesh Filter: Δημιουργεί πλέγμα ακμών για το Αντικείμενο το οποίο βοηθάει στον καλύτερη απεικόνιση του σχήματός του

-Mesh Renderer: Περιέχει πληροφορίες και ιδιότητες που αφορούν το πλέγμα ακμών του Αντικειμένου

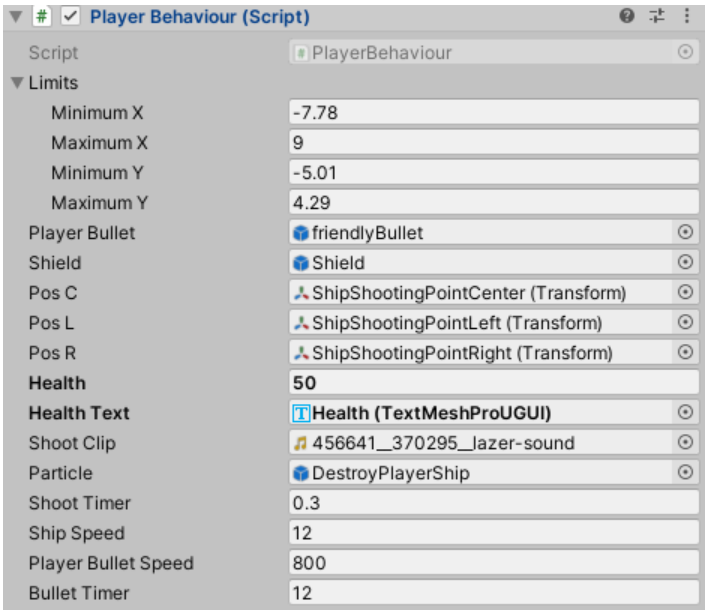
-Light: Φωτίζει περιοχές της Σκηνής

-Camera: Ορίζει το οπτικό πεδίο του παίχτη

-UI Canvas: Ομάδα Συστατικών που βοηθάει στην προβολή GUI

Όπως μπορούμε να καταλάβουμε κάποια Συστατικά λειτουργούν καλύτερα όταν συνδυάζονται όπως, για παράδειγμα, το Rigidbody μαζί με ένα Collider επιτρέπουν σε ένα Αντικείμενο να υπακούει στις δυνάμεις της φυσικής και να συγκρούεται και να αλληλεπιδρά με άλλα Colliders.

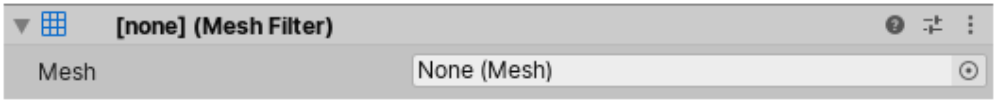
Όταν προσθέσουμε ένα Συστατικό σε κάποιο Αντικείμενο, υπάρχουν διάφορες ιδιότητες που μπορούμε να επεξεργαστούμε είτε κατά την διάρκεια της δημιουργίας του παιχνιδιού είτε σε πραγματικό χρόνο ενώ τρέχει μέσω των script (Εικόνα 45). Υπάρχουν δύο ειδών ιδιότητες: Τιμές (Values) και Αναφορές (References).



Εικόνα 45: Ιδιότητες Συστατικού (πηγή δημιουργός)

Στην παραπάνω εικόνα μπορούμε να δούμε ένα παράδειγμα ιδιοτήτων Συστατικού. Μπορούμε να ξεχωρίσουμε τις Αναφορές από το ότι στα πεδία τους δεν υπάρχει απλά ένας αριθμός και στα δεξία του όνοματός τους υπάρχει ένα κυκλάκι με μία τελεία. Τα Συστατικά περιλαμβάνουν Αναφορές σε κάποιο άλλο Συστατικό, Αντικείμενο ή Asset. Για να αναθέσουμε μία τέτοια Αναφορά απλά “σέρνουμε” έναν συμβατό με το πεδίο στοιχείο από το ανάλογο παράθυρο στο πεδίο της Αναφοράς.

Για παράδειγμα σε ένα πεδίο Αναφοράς που περιμένει Mesh εμείς δεν μπορούμε να δώσουμε Collider (Εικόνα 46) κοκ.



Εικόνα 46: Αναφορά Mesh [34]

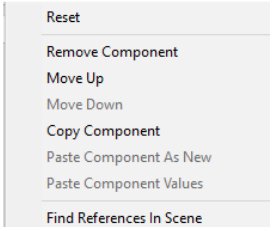
Σε περίπτωση που σύρουμε ολόκληρο Αντικείμενο στο συγκεκριμένο πεδίο η Unity εντοπίζει το πρώτο Συστατικό του απαιτούμενου τύπου στο Αντικείμενο και το αναθέτει. Ένα ακόμη τρόπος ανάθεσης Αναφορών είναι μέσα από το Παράθυρο Επιλογής Στοιχείων που ανοίγει όταν πατήσουμε τον κύκλο με την τελεία στα δεξιά του πεδίου. Από εκεί βρίσκουμε το στοιχείο που θέλουμε και με διπλό κλικ το αναθέτουμε.

Όσον αφορά τις Τιμές στο παράδειγμα της Εικόνας 45 είναι απλά αριθμοί αλλά μπορούν να είναι και λίστες drop-down, διακόπτες, χρώματα κλπ. Οι αλλαγές μπορούν να γίνουν απευθείας στα πεδία του Συστατικού.

Έχουμε αναφέρει προηγουμένως ότι οποιεσδήποτε αλλαγές κατά την διάρκεια της προσομοίωσης παιχνιδιού δεν αποθηκεύονται όταν αυτή σταματήσει. Μία καλή πρακτική είναι οι αλλαγές ιδιοτήτων Συστατικών κατά την διάρκεια αυτής ώστε να δούμε άμεσα τι πηγαίνει καλύτερα με αυτό που θέλουμε να κάνουμε καθώς τα αποτελέσματα φαίνονται σε πραγματικό χρόνο. Αυτή η διαδικασία είναι ιδιαίτερα χρήσιμη γιατί μπορούμε να πειραματιστούμε, να προσαρμόσουμε και να τελειοποιήσουμε το παιχνίδι μας χωρίς να ξοδέψουμε ιδιαίτερα χρόνο.

Παρόλο που η Unity παρέχει πληθώρα έτοιμων Συστατικών, για την δημιουργία ενός παιχνιδιού χρειάζονται όπως θα δούμε στην επόμενη ενότητα και Συστατικά κατασκευασμένα από εμάς.

Τέλος, στο Μενού Context Συστατικών (Εικόνα 47) μπορούμε να βρούμε χρήσιμες εντολές για αυτά. Το εμφανίζουμε είτε πατώντας τις τρεις κάθετες τελείες στο πάνω δεξία μέρος του Συστατικού είτε κάνοντας δεξί κλικ στον τίτλο του Συστατικού.

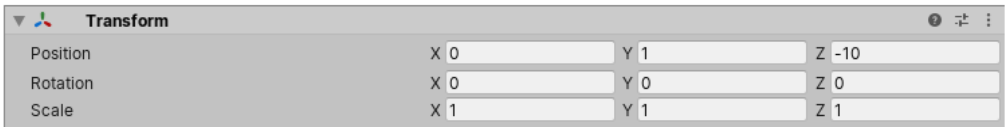


Εικόνα 47: Μενού Context Συστατικού
(πηγή δημιουργός)

Εδώ βρίσκουμε εντολές για επαναφορά πεδίων (Reset), αφαίρεση/αντιγραφή Συστατικού (Remove/Copy Component), Αναφορές στο συγκεκριμένο Συστατικό στη Σκηνή αν υπάρχουν (Find References In Scene) κα.

2.ζ.ii Transforms

Το Transform είναι ένα Συστατικό που υπάρχει σε κάθε Αντικείμενο και περιέχει την θέση, τον προσανατολισμό και το μέγεθός του (Εικόνα 48).



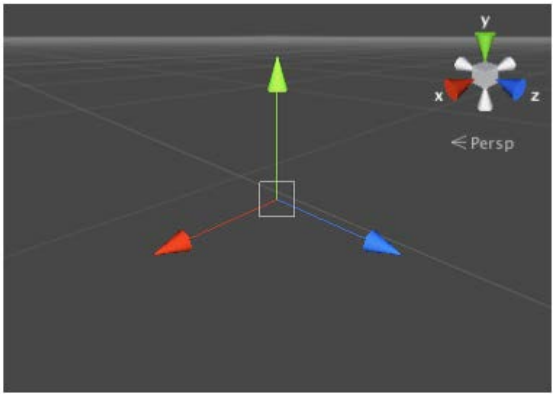
Εικόνα 48: Το Συστατικό Transform [35]

-Position: Η θέση του Αντικειμένου στους τρεις άξονες

-Rotation: Ο προσανατολισμός του Αντικειμένου στους τρεις άξονες σε μοίρες

-Scale: Το μέγεθος του Αντικειμένου στους τρεις άξονες όπου 1 το μέγεθος που δημιουργήθηκε

Τα παραπάνω πεδία υπολογίζονται σε σχέση με τον γονέα του εκάστοτε Αντικειμένου. Σε περίπτωση που δεν υπάρχει γονέας υπολογίζονται στο χώρο της Σκηνής. Ο άξονας Z δεν υπάρχει σε 2D παιχνίδια (οι τιμές ειναι 0). Στην Εικόνα 49 μπορούμε να δούμε πως φαίνονται οι άξονες στην Όψη Σκηνής.



Εικόνα 49: Οι άξονες και τα χρώματά τους [35]

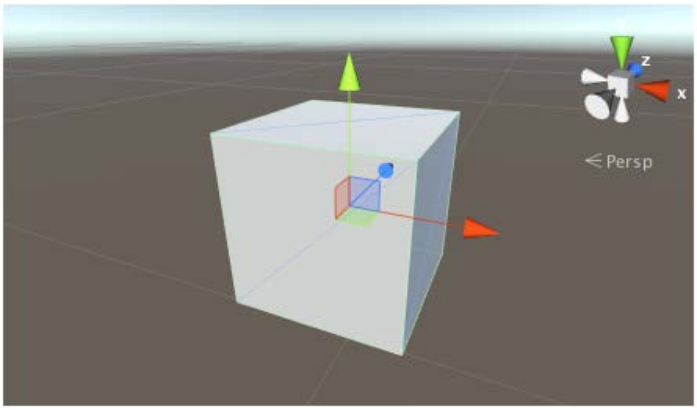
Μπορούμε να επεξεργαστούμε ένα Transform είτε από την Όψη Σκηνής όπως έχουμε δει στην παράγραφο 2.δ.iii είτε από το Παράθυρο Παρατήρησης. Όταν επεξεργαζόμαστε το πεδίο Transform ενός γονέα όλα τα Αντικείμενα παιδιά του αλλάζουν όπως ακριβώς αλλάζει το γονικό Αντικείμενο.

Μία καλή πρακτική όσον αφορά το Transform είναι ότι κατά τη δημιουργία κάποιου γονικού Αντικειμένου να θέτουμε την τοποθεσία του σε <0,0,0> πριν βάλουμε κάποιο παιδί για να αποφύγουμε λάθη κατή την τοποθέτηση του παιδιού στον χώρο της Σκηνής.

2.ζ.iii Πρότυπα Αντικείμενα

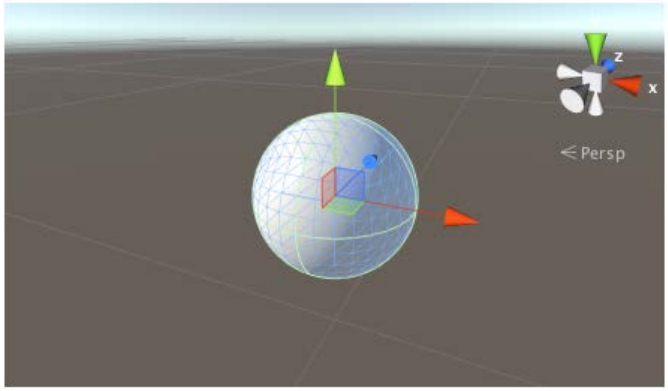
Στην πλατφόρμα Unity μπορούμε να δουλέψουμε με 3D σχήματα Αντικειμένων που έχουμε δημιουργήσει εμείς. Παρόλα αυτά, μας παρέχει με κάποια Αντικείμενα Πρότυπα (Προκατασκευασμένα) τα οποία μπορούν να καλύψουν γρήγορα αρκετές ανάγκες κατά την υλοποίηση του παιχνιδιού και βοηθούν στο testing του. Αυτά τα Αντικείμενα βρίσκονται στο μενού GameObject > 3D Object και από εκεί μπορούμε να τα προσθέσουμε στη Σκηνή μας.

-Cube (Κύβος): (Εικόνα 50) Χαρακτηριστικό παράδειγμα Πρότυπου Αντικειμένου που συνήθως χρησιμοποιείται κατά την δοκιμή του παιχνιδιού όταν δεν είναι έτοιμα πιο λεπτομερή μοντέλα (πχ Αντικείμενο αυτοκίνητο).



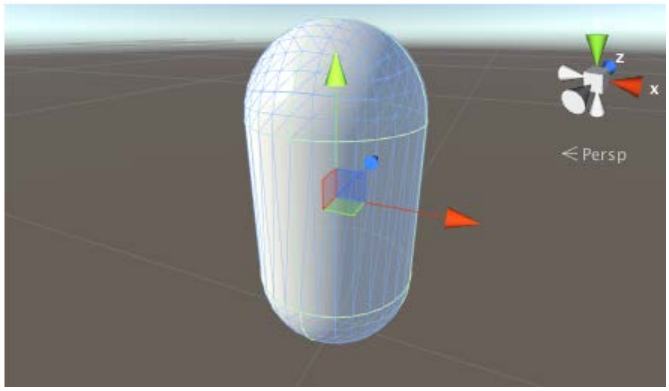
Εικόνα 50: Αντικείμενο Κύβος [36]

-Sphere (Σφαίρα): (Εικόνα 51) Χρήσιμο για την απεικόνιση μπάλων, πλανητών κα.



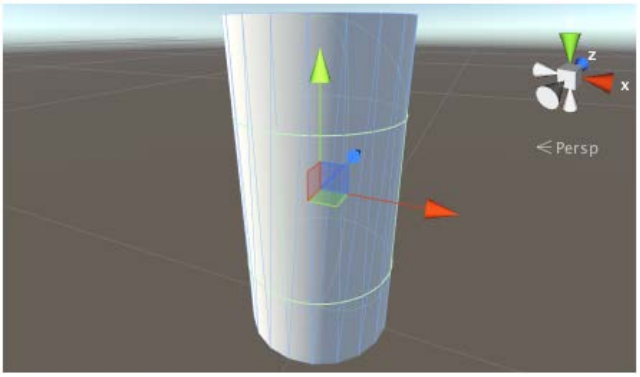
Εικόνα 51: Αντικείμενο Σφαίρα [36]

-Capsule (Κάψουλα): (Εικόνα 52) Συνήθως χρησιμοποιείται για να προσομοιώσει και να δώσει την αίσθηση του προσώπου που ελέγχει ο παίκτης σε παιχνίδια Πρώτου Προσώπου.



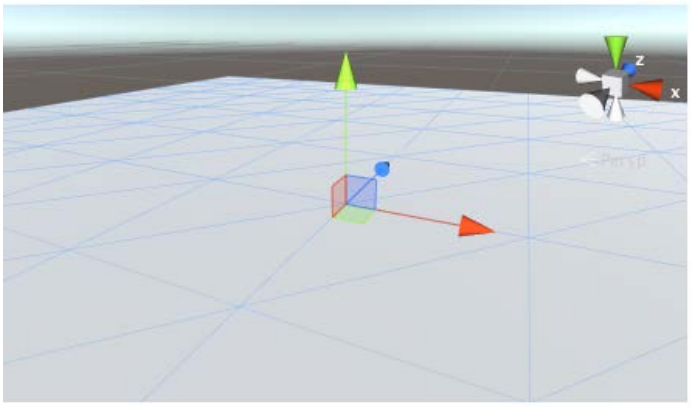
Εικόνα 52: Αντικέιμενο Κάψουλα [36]

-Cylinder (Κύλινδρος): (Εικόνα 53) Χρησιμοποιείται κυρίως για την κατασκευή κολόνων, καλαμιών, ρόδες αυτοκινήτων κλπ.



Εικόνα 53: Αντικείμενο Κύλινδρος [36]

-Plane (Επίπεδο): (Εικόνα 54) Είναι ένα επίπεδο τετράγωνο δύο αξόνων και χρησιμεύει στην κατασκευή πολλών ειδών λείων επιφανειών όπως τείχη και πατώματα. Μπορεί επίσης να χρησιμοποιηθεί για εμφάνιση εικόνων, ταινιών κλπ. σε GUI αλλά για μία τέτοια δουλειά είναι πιο εύχρηστο το μικρότερο **Quad** το οποίο βοηθάει και στην απόδοση φιγούρων.



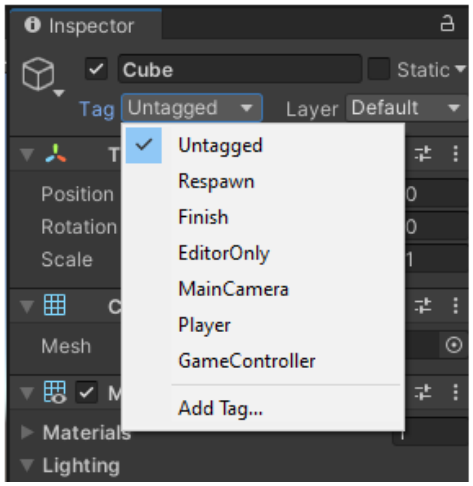
Εικόνα 54: Αντικείμενο Επίπεδο [36]

2.ζ.iv Ετικέτες (Tags)

Οι Ετικέτες είναι λέξεις αναφοράς για να κατηγοριοποιήσουμε ένα ή παραπάνω Αντικείμενα σε ομάδες όπως για παράδειγμα την Ετικέτα “Player” για Αντικείμενα που ελέγχει ο παίχτης και την Ετικέτα “Enemy” για Αντικείμενα που πρέπει να αποφύγει ο παίχτης.

Ο κύριος σκοπός της Ετικέτας είναι να διευκολύνει την γραφή κώδικα καθώς μπορούμε πιο εύκολα να ελέγξουμε μία Κρούση που έγινε μεταξύ “Player” και “Enemy” από το να ελέγχουμε ένα ένα τα Αντικείμενα.

Για να δημιουργήσουμε μία νέα Ετικέτα, βρίσκουμε το μενού Ετικετών που υπάρχει σε κάθε Αντικείμενο στο Παράθυρο Παρατήρησης κάτω από το όνομά του (Εικόνα 55) και πατάμε Add Tag… μία διαδικασία που μοιάζει γνώριμη καθώς είναι παρόμοια με την δημιουργία νέου Layer. Στην περίπτωση που δημιουργήσουμε μία Ετικέτα δεν μπορούμε να αλλάξουμε το όνομά της στη συνέχεια. Η Unity παρέχει κάποιες Ετικέτες ως προεπιλογή. Κάθε Αντικείμενο μπορεί να έχει μόνο μία Ετικέτα.

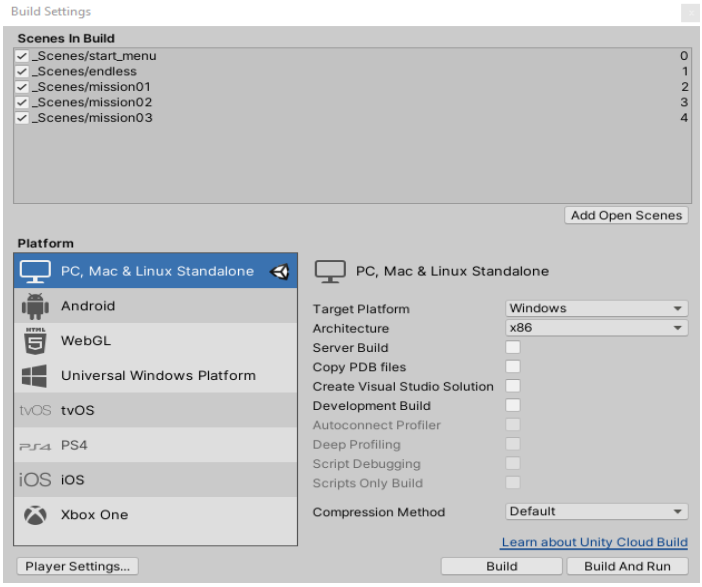


Εικόνα 55: Μενού Ετικετών [37]

2.η Κατασκευή & Εξαγωγή Παιχνιδιού

Όταν το παιχνίδι μας βρίσκεται σε ένα ικανοποιητικό επίπεδο από άποψη υλοποίησης και δοκιμών φτάνουμε στην καλύτερη στιγμή στη ροή της δημιουργίας ενός βιντεοπαιχνιδιού καθώς είναι ώρα να το διαμοιράσουμε ώστε να δουν και άλλοι αυτό που δημιουργήσαμε όπως και να δοκιμαστεί εκτός της πλατφόρμας Unity. Ευτυχώς η Unity κάνει την συγκεκριμένη διαδικασία αρκετά εύκολη παρέχοντας μας πολλές επιλογές όσον αφορά την έκδοση του παιχνιδιού σε διαφορετικά συστήματα και για διάφορες ποιότητες γραφικών. Αυτές τις επιλογές μπορούμε να τις βρούμε στο

παράθυρο με τις ρυθμίσεις Εξαγωγής το οποίο το ανοίγουμε πατώντας File > Build Settings και φαίνεται στην Εικόνα 56.



Εικόνα 56: Παράθυρο ρυθμίσεων Εξαγωγής
(πηγή δημιουργός)

-Scenes in Build: Στο συγκεκριμένο σημείο μπορούμε να διαχειριστούμε τις Σκηνές που θα συμπεριλάβει η Unity σε αυτήν την εξαγωγή του παιχνιδιού. Έχουμε την δυνατότητα για προσθήκη, αφαίρεση Σκηνών καθώς και την σειρά δημιουργίας τους. Προσθέτουμε μία νέα Σκηνή σέρνοντας το αρχείο της και αφήνοντάς το σε αυτό το πλάισιο. Η Σκηνή με τον αριθμό 0 είναι και η πρώτη Σκηνή με την οποία ξεκινάει το παιχνίδι όταν τρέχει.
-Platform: Εδώ μπορούμε να επιλέξουμε το σύστημα για το οποίο θέλουμε να εξαχθεί το παιχνίδι. Εφόσον διαλέξουμε το επιθυμητό στα δεξιά του μπορούμε να δούμε περισσότερες ρυθμίσεις που αφορούν το σύστημα. Σε αυτό το σημείο βλέπουμε τις δυνατότητες της πλατφόρμας Unity καθώς μέχρι τώρα για την υλοποίηση σε διαφορετικά συστήματα δεν έχει χρειαστεί να κάνουμε τίποτα άλλο πάρα μόνο να διαλέξουμε με ένα κλικ αυτό που θέλουμε και η Unity αναλαμβάνει τα υπόλοιπα.

Για να ολοκληρώσουμε τη διαδικασία της Εξαγωγής πατάμε είτε Build το οποίο δημιουργεί τα κατάλληλα αρχεία και ένα εκτελέσιμο με το όνομα που έχουμε επιλέξει είτε Build And Run που έχει σαν πρόσθετη λειτουργία ότι τρέχει το εκτελέσιμο μόλις όλα ολοκληρωθούν. Είμαστε έτοιμοι να παίξουμε κάτι που δημιουργήσαμε εμείς!

ΚΕΦΑΛΑΙΟ 3 UNITY & Η ΓΛΩΣΣΑ C#

3.α Σκοπός Ενότητας

Όπως έχουμε αναφέρει ο προγραμματισμός είναι αναπόσπαστο κομμάτι της διαδικασίας της ανάπτυξης και σχεδίασης βιντεοπαιχνιδιού με την Unity ειδικά αν θέλουμε το παιχνίδι που θα δημιουργήσουμε να ξεχωρίζει. Για αυτό χρησιμοποιούμε scripts που ενεργούν ανάλογα τα δεδομένα που δίνει ο παίχτης και φροντίζουν τα γεγονότα που πρέπει να συμβούν κατά τη διάρκεια του παιχνιδιού να γίνονται τη σωστή στιγμή. Επιπροσθέτως μπορούμε να δημιουργήσουμε ειδικά γραφικά εφέ αλλά και να ενσωματώσουμε τεχνητή νοημοσύνη σε χαρακτήρες του παιχνιδιού.

Παρόλα αυτά, το εγχείρημα να μάθουμε γραφή κώδικα και προγραμματισμού δεν είναι εφικτό εδώ και ούτε εντός των πλαισίων της συγκεκριμένης διπλωματικής. Ενδείκνυται ο ενδιαφερόμενος αναγνώστης να ανατρέξει είτε στο επίσημο site για εκμάθηση γραφής κώδικα της Unity:
(https://learn.unity.com/search?k=%5B"q%3AScripting"%5D)
καθώς παρέχει μεγάλη ποικιλία μαθημάτων για όλα τα επίπεδα εμπειρίας του προγραμματιστή, είτε με μία αναζήτηση του θέματος στο διαδίκτυο καθώς υπάρχει μεγάλη συλλογή πληροφοριών.

Επομένως, ο σκοπός αυτής την ενότητας είναι να δείξει στοιχεία της C# που την κάνουν να λειτουργεί ιδανικά στα πλαίσια της πλατφόρμας, σημαντικά σημεία στα οποία υπάρχει συνεργασία μεταξύ Unity και C# αλλά και επίδειξη ιδιαίτερων δυνατοτήτων που καθιστούν αυτόν τον συνδυασμό εξαιρετικά ισχυρό στον τομέα της ανάπτυξης βιντεοπαιχνιδιών.

3.β Περιβάλλον Προγραμματισμού

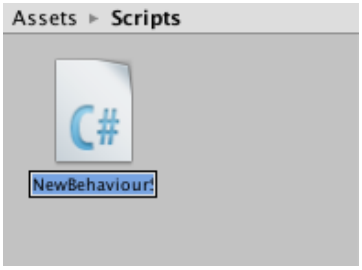
Ένα Ενσωματωμένο Περιβάλλον Ανάπτυξης (IDE) είναι ουσιαστικά ένα εργαλείο λογισμικού που βοηθάει στην πιο εύκολη γραφή και ανάπυξη κώδικα. Η Unity υποστηρίζει τα Visual Studio, Visual Studio Code και JetBrains Rider.

Το Visual Studio είναι το προεπιλεγμένο IDE που χρησιμοποιεί η Unity καθώς κατά την εγκατάσταση της πλατφόρμας γίνεται και εγκατάσταση αυτού. Σε περίπτωση που έχουμε κάποιο άλλο IDE και δεν θέλουμε να χρησιμοποιήσουμε το Visual Studio μπορούμε να αλλάξουμε πηγαίνοντας από το μενού Edit > Preferences > External Tools > External Script Editor και να διαλέξουμε αυτό που θέλουμε. Όποιο πρόγραμμα υπάρχει εδώ θα είναι και το πρόγραμμα με το οποίο ανοίγουν τα scripts της πλατφόρμας.

3.γ Βασικά Scripting Στη Unity

Έχουμε δει ότι οι συμπεριφορές των Αντικειμένων καθορίζονται από τα Συστατικά που διαθέτουν και αρκετά από αυτά τα παρέχει η Unity. Για την δημιουργία όμως ξεχωριστών εμπειριών παιχνιδιού χρειάζεται να υλοποιήσουμε τα δικά μας Συστατικά χρησιμοποιώντας scripts. Στη συγκεκριμένη διπλωματική θα αναφερθούμε στη γλώσσα C# καθώς σε αυτήν είναι υλοποιημένο το παιχνίδι μας και είναι και η γλώσσα που συνιστά η Unity καθώς έχει την μεγαλύτερη συμβατότητα με την πλατφόρμα.

Σε αντίθεση με άλλα Στοιχεία που εισάγουμε στο παιχνίδι μας, τα scripts δημιουργούνται απευθείας μέσα στη Unity. Μπορούμε να δημιουργήσουμε ένα script με αρκετούς τρόπους. Είτε με δεξί κλικ στο Παράθυρο Εργασίας και μετά Create > C# Script είτε με απευθείας επίσυναψη σε ένα Αντικείμενο πατώντας Add Component στο Παράθυρο Παρατήρησης και New script (Εικόνα 57). Συνιστάται η ονομασία του αρχείου τώρα για να αποφευχθούν λάθη αργότερα.



Εικόνα 57: Νέο script [38]

Ανοίγοντας το νέο αυτό αρχείο μπορούμε να παρατηρήσουμε τον σκελετό και την δομή που έχει κάθε καινούριο script. Ένα παράδειγμα νέου script βλέπουμε στην Εικόνα 58.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

Εικόνα 58: Περιεχόμενα νέου script [38]

Ένα script συνδέεται με τον τρόπο λειτουργίας της Unity μέσω μιας ενσωματωμένης κλάσης που ονομάζεται MonoBehavior. Μπορούμε να σκεφτούμε μία κλάση ως ένα είδος σχεδίου για την δημιουργία νέων Συστατικών. Το όνομα της κλάσης που δημιουργούμε είναι το όνομα του αρχείου script (στην παραπάνω εικόνα “MainPlayer”) και πρέπει να παραμείνουν ίδια για να λειτουργήσει. Σημαντικό επίσης είναι για να λειτουργεί ένα script να έχει γίνει η επισύναψή του σαν Συστατικό σε κάποιο Αντικείμενο στη Σκηνή και θα ξεκινήσουν οι ενέργειές του όταν αρχίσει το παιχνίδι.

Παρατηρούμε ότι υπάρχουν δύο συναρτήσεις στο νέο script. Η συνάρτηση **Update** εκτελείται με κάθε ανανέωση frame και συνήθως περιλαμβάνει κώδικα για ενέργειες όπως κινήσεις, δράσεις και ανταπόκριση σε δεδομένα από τον χρήστη (Εικόνα 59).

```
// Methods
gameObject.transform.Translate(new Vector3(1, 0, 0));
gameObject.transform.Rotate(new Vector3(180, 0, 0));
gameObject.transform.localScale = 2 * gameObject.transform.localScale;
```

Εικόνα 59: Παράδειγμα συνάρτησης Update
(πηγή δημιουργός)

Στην Εικόνα 59 βλέπουμε ένα τέτοιο παράδειγμα κώδικα. Η πρώτη γραμμή μετακινεί το Αντικείμενο κατά ένα μέτρο στον χ άξονα, η δεύτερη το περιστρέφει κατά 180 μοίρες στον χ άξονα και η τρίτη διπλασσιάζει το μέγεθος. Αυτές οι τρεις ενέργειες συμβαίνουν κάθε φορά που ανανεώνονται τα frames.

Η συνάρτηση **Start** καλείται στην αρχή πριν ξεκινήσει να τρέχει το παιχνίδι και είναι το κατάλληλο μέρος για να γίνουν οποιεσδήποτε αρχικοποιήσεις (Εικόνα 60).

```
// Use this for initialization
void Start ()
{
    Debug.Log("I am alive!");
}
```

Εικόνα 60: Παράδειγμα συνάρτησης Start [38]

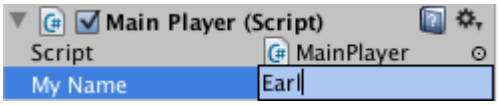
Η συνάρτηση Start της Εικόνας 60 θα εμφανίσει στο παράθυρο της κονσόλας της Unity το μήνυμα “I am alive!” όταν πατήσουμε το κουμπί Play.

Όταν δημιουργούμε ένα script μπορούμε να εμφανίσουμε στο Παράθυρο Παρατήρησης κάποια πεδία ως ιδιότητες του Συστατικού για επεξεργασία (Εικόνα 61). Αυτό είναι ιδιαίτερα χρήσιμο όταν χρειαζόμαστε Αναφορές σε άλλα Αντικείμενα (για παράδειγμα ένα script που διαχειρίζεται τις κινήσεις ενός διαστημόπλοιου και χρειάζεται μία

αναφορά σε ένα Αντικείμενο ρουκέτας για να το δημιουργεί όταν ο παίκτης πατάει κάποιο κουμπί).

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour
{
    public string myName;
```



(i) (ii)

Εικόνα 61: i) Δημιουργία πεδίου myName ii) Το πεδίο στο Παράθυρο Παρατήρησης [39]

Αναφέραμε στην προηγούμενη ενότητα πως τα Συστατικά Collider και RigidBody δουλεύουν μαζί για τον εντοπισμό κρούσεων. Μία συνάρτηση που μας δίνει πληροφορίες πάνω σε αυτήν την κρούση είναι η συνάρτηση της Εικόνας 62 που μας δίνει το Αντικείμενο με το οποίο έγινε η κρούση τη στιγμή που συμβαίνει αυτή.

```
private void OnCollisionEnter(Collision collision)
{
    GameObject collidedWith = collision.gameObject;
}
```

Εικόνα 62: Παράδειγμα συνάρτησης OnCollisionEnter
(πηγή δημιουργός)

3.δ Αρχικοποίηση Prefab Σε Πραγματικό Χρόνο

Όπως έχουμε δει τα Prefabs είναι ένα πολύ δυνατό εργαλείο για την δημιουργία πολύπλοκων Αντικειμένων κατά την ανάπτυξη βιντεοπαιχνιδιών σε Unity. Μία από τις πιο ισχυρές δυνατότητες που μας παρέχει ο συνδυασμός της πλατφόρμας με την C# είναι ότι μπορούμε με μία γραμμή κώδικα να αρχικοποιήσουμε ένα Αντικείμενο μέσω Prefab στον χώρο της Σκηνής!

Για να αρχικοποιήσουμε με αυτόν τον τρόπο ένα Prefab χρειάζεται για αρχή ο κώδικάς μας να δημιουργεί μία Αναφορά στο Συστατικό του script όπως είδαμε στην πρηγούμενη παράγραφο για αυτό πρέπει να έχουμε μία public μεταβλητή που θα περιέχει αυτήν την Αναφορά (Εικόνα 63).

```
// Reference to the Prefab. Drag a Prefab into this field in the Inspector.
public GameObject myPrefab;
```

Εικόνα 63: Δημιουργία μεταβλητής που περιέχει την Αναφορα στο Prefab [40]

Στη παραπάνω εικόνα βλέπουμε τον κώδικα που πρέπει να έχουμε στην κλάση μας ώστε να υπάρχει η Αναφορά στο Prefab. Το συγκεκριμένο κομμάτι κώδικα θα δημιουργήσει ένα πεδίο Αναφοράς στο Συστατικό του script με το όνομα “My Prefab” το οποίο θα περιμένει να του ανατεθεί ένα Αντικείμενο (η λέξη GameObject στον κώδικά μας).

Για να γίνει η αρχικοποίηση κατά το τρέξιμο του παιχνιδιού (συγκεκριμένα στην Εικόνα 64 θα γίνει μία φορά καθώς το κάλεσμα της μεθόδου βρίσκεται στη συνάρτηση Start) πρέπει να καλέσουμε την μέθοδο **Instantiate** που είναι και η μέθοδος που αρχικοποιεί το Prefab με παραμέτρους: (Prefab_προς_αρχικοποίηση, σημείο_στη_Σκηνή, προσανατολισμός).

```
// This script will simply instantiate the Prefab when the game starts.
void Start()
{
    // Instantiate at position (0, 0, 0) and zero rotation.
    Instantiate(myPrefab, new Vector3(0, 0, 0), Quaternion.identity);
}
```

Εικόνα 64: Αρχικοποίηση Prefab [40]

Όταν πατήσουμε το κουμπί Play θα δούμε στο σημείο (0,0,0) της Σκηνής το Prefab που έχουμε δώσει σαν Αναφορά. Έχουμε και την δυνατότητα να αλλάξουμε Prefab βάζοντας ένα καινούριο στο πεδίο που όπως έχουμε αναφέρει είναι μία καλή πρακτική για δοκιμές.

Αν το προηγούμενο παράδειγμα φαίνεται απλό και δεν δείχνει όλη τη δύναμη αυτής της μεθόδου, παρακάτω θα δούμε κάποια σενάρια στα οποία η αρχικοποίηση Prefab σε πραγματικό χρόνο κάνει την δουλειά μας πολύ πιο εύκολη.

-Σενάριο 1: Δημιουργία κατασκευής

Μπορούμε να δημιουργήσουμε μία κατασκευή αρχικοποιώντας πολλές φορές ένα Prefab σε συγκεκριμένη διάταξη. Ο τρόπος αυτός δημιουργίας κατασκευών ονομάζεται διαδικαστική δημιουργία (procedural generation) και είναι ιδιαίτερα χρήσιμη κατά την ανάπτυξη βιντεοπαιχνιδιών καθώς χάρη σε αυτήν μπορούμε να κατασκευάσουμε περιβάλλοντα, χαρακτήρες, animations κλπ. αλγοριθμικά μέσω του υπολογιστή κατά το τρέξιμο του παιχνιδιού.[41] Στο συγκεκριμένο σενάριο θα δημιουργήσουμε ένα τείχο από Αντικείμενα Κύβους με τον κώδικα της Εικόνας 65.

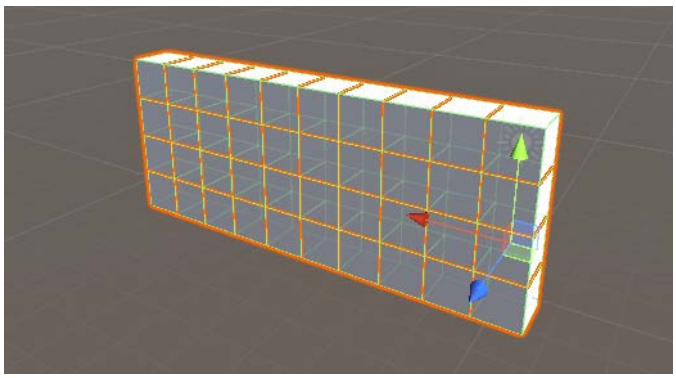
```
using UnityEngine;
public class Wall : MonoBehaviour
{
   public GameObject block;
   public int width = 10;
   public int height = 4;

   void Start()
   {
       for (int y=0; y<height; ++y)
       {
           for (int x=0; x<width; ++x)
           {
               Instantiate(block, new Vector3(x,y,0), Quaternion.identity);
           }
       }
   }
}
```

Εικόνα 65: Script δημιουργίας τείχου [40]

Έχουμε ένα script με το όνομα Wall το οποίο φανερώνει τρία πεδία στο Συστατικό, τις δύο μεταβλητές width και height και μία Αναφορά σε Αντικείμενο block. Στη συνέχεια σε μία επανάληψη δύο βρόγχων που φτάνουν μέχρι τον αριθμό που έχουμε δώσει σαν height και width, αρχικοποιούμε ένα block με συντεταγμένες ανάλογα την φάση που βρίσκεται η επανάληψη. Όταν τελειώσει αυτή η διαδικασία θα έχουμε έναν τείχο με

ύψος 4 block και πλάτος 10 block. Με τον τρόπο που έχουμε κατασκευάσει το script μπορούμε πολύ εύκολα να αλλάξουμε τα παραπάνω μεγέθη ακόμα και το είδος του Αντικειμένου από το οποίο φτιάχνεται ο τείχος στο Παράθυρο Παρατήρησης. Στην Εικόνα 66 μπορούμε να δούμε το αποτέλεσμα στην περίπτωση που πατήσουμε Play.



Εικόνα 66: Τείχος από Prefab [40]

-Σενάριο 2: Αρχικοποίηση βλημάτων και εκρήξεων

Έστω ότι έχουμε ένα τύπο κανονιού και θέλουμε να βάλλει κάποιο είδος βλήματος όταν ο παίχτης δώσει την ανάλογη εντολή. Το βλήμα είναι ένα Prefab το οποίο περιέχει Συστατικά Mesh, RigidBody και Collider για να μπορεί να εκτοξευτεί στον αέρα και να εντοπίζει κρούσεις με άλλα Αντικείμενα. Όταν αυτό συγκρουστεί με κάποιο άλλο Collider τότε θέλουμε να αρχικοποιήσουμε το Prefab της έκρηξης το οποίο περιέχει εφέ σωματιδίων (particle effect). Η αρχικοποίηση του βλήματος συμβαίνει μέσα στην συνάρτηση Update καθώς θέλουμε να ελέγχουμε κάθε φορά που ανανεώνονται τα frames αν ο παίχτης έχει πατήσει το κουμπί (Εικόνα 67).

```
using UnityEngine;
public class FireProjectile : MonoBehaviour
{
    public Rigidbody projectile;
    public float speed = 4;
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Rigidbody p = Instantiate(projectile, transform.position, transform.rotation);
            p.velocity = transform.forward * speed;
        }
    }
}
```

Εικόνα 67: Script αρχικοποίησης βλημάτων [40]

Το script μας έχει την ονομασία FireProjectile, το επισυνάπτουμε στο Αντικείμενο του κανονιού μας και φανερώνει δύο πεδία στο Συστατικό, την μεταβλητή speed και την Αναφορά σε ένα RigidBody projectile. Στην συνάρτηση Update ελέγχουμε αν ο χρήστης έχει πατήσει το κουμπί που έχουμε ορίσει στο παιχνίδι μας ως το κουμπί του πυροβολισμού με το Input.GetButtonDown(“Fire1”) και εφόσον το πάτησε αρχικοποιούμε το βλήμα με την θέση και τον προσανατολισμό του κανονιού μας δίνοντας του και επιτάχυνση ανάλογη με την τιμή που έχουμε δώσει στην μεταβλητή speed. Παρατηρούμε ότι η Αναφορά μας περιμένει τύπο RigidBody. Αυτό συμβαίνει για να αποφύγουμε να βάλουμε λάθος Αντικείμενο που δεν περιέχει RigidBody αλλά και μας διευκολύνει στο να έχουμε πιο άμεση πρόσβαση στο πεδίο velocity του RigidBody που αφορά την επιτάχυνση.

Στην Εικόνα 68 βλέπουμε το script με το όνομα Projectile που επισυνάπτεται στο Prefab του βλήματος. Με μία Αναφορά στο Prefab της έκρηξης, εντοπίζει κρούσεις πάνω του και σε περίπτωση που συμβεί κάποια αρχικοποιεί την έκρηξη και καταστρέφει το Αντικείμενο του βλήματος.

```
using UnityEngine;
public class Projectile : MonoBehaviour
{
   public GameObject explosion;
   void OnCollisionEnter()
   {
       Instantiate(explosion,transform.position,transform.rotation);
       Destroy(gameObject);
   }
}
```

Εικόνα 68: Script αρχικοποίησης εκρήξεων [40]

-Σενάριο 3: Αντικτάσταση Αντικειμένου με την κατεστραμμένη του μορφή

Σε μία παραλλαγή του προηγούμενου σεναρίου μπορούμε να σκεφτούμε ότι σε περίπτωση που το βλήμα μας χτυπήσει ένα άλλο Αντικείμενο εκείνο καταστρέφεται και μένει στην θέση του μία κατεστραμμένη μορφή του. Με παρόμοιο τρόπο, την στιγμή που εντοπίζεται η κρούση, καταστρέφουμε το Αντικείμενο που δείχνει την αρχική μη κατεστραμμένη μορφή και αρχικοποιούμε το Prefab που περιέχει την κατεστραμμένη.

Έτσι μία τέτοια διαδικασία γίνεται απλή καθώς δεν χρειάζεται να μετατρέψουμε την αρχική μορφή του Αντικειμένου μας.

Τα σενάρια αυτής της παραγράφου καθώς και πολλά ακόμα μας δίνουν μια μικρή ιδέα για τις δυνατότητες που παρέχει σε έναν προγραμματιστή η πλατφόρμα Unity σε συνδυασμό με την C# για την όσο το δυνατόν γρηγορότερη και καλύτερη υλοποίηση ενός βιντεοπαιχνιδιού.

3.ε Συναρτήσεις Γεγονότων

Ένα script στη Unity δεν ακολουθεί την ιδέα ενός προγράμματος που ο κώδικας τρέχει συνεχώς σε επανάληψη μέχρι να ολοκληρώσει κάποια εργασία. Αντί για αυτό, η πλατφόρμα δίνει τον έλεγχο κατά διαστήματα στο script καλώντας συγκεκριμένες συναρτήσεις που έχουν δηλωθεί σε αυτό. Όταν αυτή η συνάρτηση τελειώσει τότε ο έλεγχος επιστρέφει στη Unity. Αυτές οι συναρτήσεις είναι γνωστές ως **συναρτήσεις γεγονότων** και έχουμε ήδη δει κάποιες όπως Start και Update. Ονομάζονται συναρτήσεις γεγονότων επειδή ενεργοποιούνται απο την Unity ως ανταπόκριση σε γεγονότα που συμβαίνουν κατά τη ροή του παιχνιδιού. Παρακάτω θα δούμε μερικές συχνές και σημαντικές συναρτήσεις.

-Συναρτήσεις Γεγονότων Ανανέωσης

Μία βασική έννοια στην ανάπτυξη βιντεοπαιχνιδιών είναι ότι οι αλλαγές σε θέση, κατάσταση και συμπεριφορά στοιχείων στο παιχνίδι πρέπει να γίνονται πριν ακριβώς την στιγμή της απεικόνισης νέου frame. Για αυτό έχουμε δει τη συνάρτηση **Update** που καλείται πριν γίνει απεικόνιση νέου frame και πριν υπολογιστούν τα animations. Η μηχανή φυσικής ανανεώνεται με παρόμοιο τρόπο όμως οι αλλαγές δε συμβαίνουν με την ίδια συχνότητα με τα frames. Για πιο ακριβή αποτελέσματα κατά την εφαρμογή φυσικής χρησιμοποιούμε τη συνάρτηση **FixedUpdate** που καλείται πριν από ανανεώσεις φυσικής. Τέλος, κάποιες φορές θέλουμε να πραγματοποιήσουμε επιπρόσθετες αλλαγές μετά από τις προηγούμενες δύο συναρτήσεις (πχ το που δείχνει η κάμερα μας). Σε τέτοιες περιπτώσεις χρησιμοποιούμε τη συνάρτηση **LateUpdate.**

-Συναρτήσεις Γεγονότων Αρχικοποίησης

Είναι σύνηθες να θέλουμε να αρχικοποιήσουμε κάποιο κομμάτι κώδικα πριν συμβούν ανανεώσεις κατά τη διάρκεια του παιχνιδιού. Όπως είδαμε με τη συνάρτηση **Start** που συμβαίνει πριν γίνει η πρώτη ανανέωση frame ή φυσικής σε κάποιο στοιχείο. Η συνάρτηση **Awake** καλείται για κάθε στοιχείο τη στιγμή που θα φορτωθεί η Σκηνή. Όλα τα καλέσματα σε συναρτήσεις Awake θα έχουν τελειώσει όταν κληθούν οι συναρτήσεις Start άρα η Start μπορεί να χρησιμοποιήσει δεδομένα που έχουν αρχικοποιηθεί στην Awake.

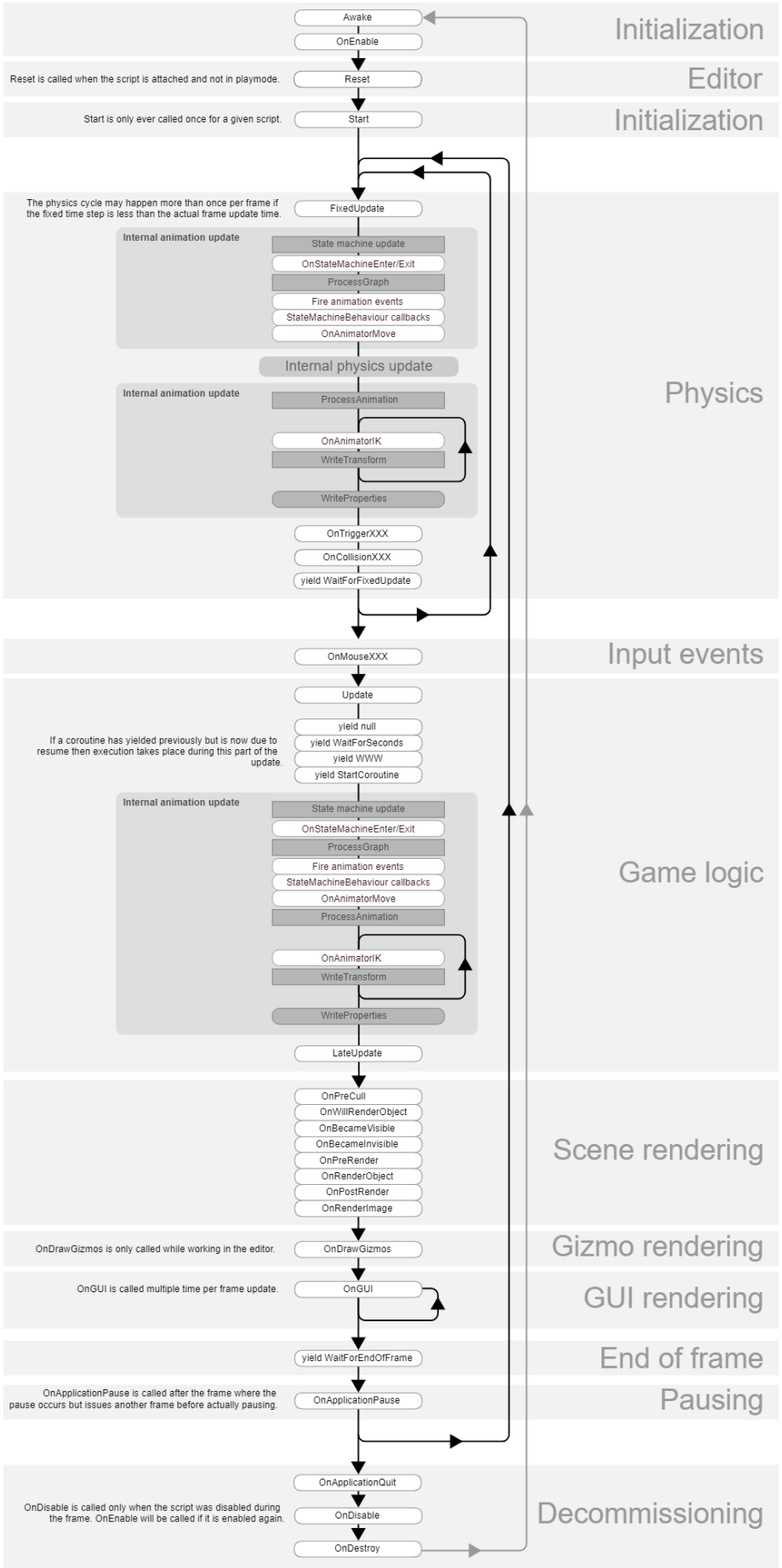
-Συναρτήσεις Γεγονότων GUI

Η Unity διαθέτει σύστημα για απεικόνιση GUI πάνω από την κύρια δράση που συμβαίνει σε μία Σκηνή και αντίδραση σε όποια κλικ συμβούν σε αυτό. Για τέτοιες ενέργειες χρησιμοποιούμε τη συνάρτηση **OnGUI** που καλείται περιοδικά. Μπορούμε επίσης να εντοπίσουμε γεγονότα που αφορούν τις ενέργειες που κάνει ο χρήστης με το ποντίκι. Για παράδειγμα η συνάρτηση **OnMouseDown** ελέγχει αν ο χρήστης έχει πατήσει κάποιο κουμπί του ποντικιού πάνω από ένα στοιχείο που περιέχει τη συγκεκριμένη συνάρτηση.

-Συναρτήσεις Γεγονότων Φυσικής

Η μηχανή φυσικής θα αναφέρει οποιεσδήποτε κρούσεις συμβούν έχοντας τις ακόλουθες συναρτήσεις. Η **OnCollisionEnter** καλείται τη στιγμή της επαφής που προκαλεί η κρούση, η **OnCollisionStay** καλείται όσο συμβαίνει η κρούση και η **OnCollisionExit** τη στιγμή που σταματάει η κρούση.

Στην Εικόνα 69 μπορούμε να δούμε όλες τις συναρτήσεις γεγονότων, τη σειρά που καλούνται αλλά και το γεγονός που έχει ως αποτέλεσμα να καλεστούν.



Εικόνα 69: Pipeline συναρτήσεων γεγονότων [42]

ΚΕΦΑΛΑΙΟ 4 ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ “INVADERS OF THE GALAXY”

4.α Επιλογές Ανάπτυξης Παιχνιδιού

Στη συγκεκριμένη παράγραφο θα δούμε κάποιες επιλογές που χρειάστηκε να κάνουμε κατά την ανάπτυξη του βιντεοπαιχνιδιού μας καθώς και τους λόγους που μας οδήγησαν σε αυτές τις επιλογές.

4.α.i Επιλογή Παιχνιδιού & Επιρροές

Με δεδομένο ότι έχουμε κάποια εμπειρία στον προγραμματισμό και στη λογική του και θέλοντας να δημιουργήσουμε ένα ολοκληρωμένο βιντεοπαιχνίδι αποφασίσαμε να κάνουμε κάτι που θα βοηθήσει στην εκμάθηση της πλατφόρμας Unity σε σχεδόν όλες τις βασικές έννοιες που χρειάζεται κάποιος φιλόδοξος δημιουργός βιντεοπαιχνιδιών. Αποφύγαμε τις δύσκολες στην κατανόηση έννοιες και τις περιττές λειτουργίες ακολουθώντας την αρχιτεκτονική μέθοδο “κρεμμυδιού” (onion architecture) που έχει σαν βάση την ανάπτυξη των απολύτως απαραίτητων λειτουργιών που χρειάζεται ένα πρόγραμμα για να παρέχει μία υπηρεσία (πυρήνας κρεμμυδιού) και κάθε έξτρα λειτουργία χτίζεται πάνω σε αυτήν (στρώσεις κρεμμυδιού) και βοηθάει στο να αποφευχθεί η παράδοση μη ολοκληρωμένων εργασιών και μπερδεμένων προγραμμάτων.[43] Παρόλα αυτά έχουμε βάλει και κάποια στοιχεία πιο πολύπλοκα για να κάνουμε τη διαφορά στο παιχνίδι μας αλλά ο σκοπός μας παραμένει να ολοκληρώσουμε ένα βιντεοπαιχνίδι το οποίο θα είναι διασκεδαστικό, το σημαντικότερο στοιχείο για ένα βιντεοπαιχνίδι, θα έχει εύκολο στην κατανόηση αντικείμενο χωρίς πολλές οδηγίες χρήσης, θα μας μάθει να χρησιμοποιούμε τη Unity και θα είναι το πρώτο μας σκαλοπάτι στην ανάπτυξη βιντεοπαιχνιδιών.

Όσον αφορά τις επιρροές που είχαμε για το παιχνίδι έχουμε αναφέρει και στην εισαγωγή ότι έχουμε σκοπό να υλοποιήσουμε ένα remaster του βιντεοπαιχνιδιού Space Invaders που είναι της κατηγορίας “shoot ‘em up” καθώς πιστεύουμε ότι καλύπτει όλους

τους στόχους που θέσαμε προηγουμένως με έξτρα πλεονέκτημα ότι είναι ένα κλασικό βιντεοπαιχνίδι arcade και ένας από τους βασικούς παράγοντες στην εκτίναξη των μετοχών της βιομηχανίας και θα είναι άμεσα αναγνωρίσιμο στους λάτρεις του είδους.[44]

Σκοπός του χρήστη σε ένα shoot ‘em up βιντεοπαιχνίδι είναι να ελέγξει τον κύριο χαρακτήρα (συνήθως κάποιο είδους αεροσκάφος) και να αντιμετωπίσει μεγάλο πλήθος εχθρών πυροβολώντας τα ενώ ταυτόχρονα αποφεύγει τα δικά τους πυρά. Η επιτυχία βασίζεται κυρίως στα αντανακλαστικά του παίχτη και αρκετές φορές ο στόχος είναι να καταστρέψει όσους πιο πολλούς εχθρούς μπορεί μέχρι να καταστραφεί ο ίδιος σημειώνοντας μεγαλύτερο σκορ από άλλους παίχτες κάτι που περιέχει και το στοιχείο του ανταγωνισμού.[45] Ένα παράδειγμα shoot ‘em up βιντεοπαιχνιδιού φαίνεται στην Εικόνα 70.

Εικόνα 70: Το shoot 'em up βιντεοπαιχνίδι Nuclear Throne [46]

4.α.ii Επιλογή Συστήματος Ανάπτυξης

Για την ανάπτυξη του βιντεοπαιχνιδιού μας αποφασίσαμε να γίνει στο λειτουργικό Windows με κύριο κριτήριο τα συστήματα που έχουμε στην κατοχή μας ώστε να είναι πιο εύκολη η δοκιμή του. Επιπροσθέτως υπολογίζεται ότι το 2022 θα υπάρχουν περίπου 1,8 δισεκατομμύρια υπολογιστές παγκοσμίως άρα με την επιλογή των Windows ξεκινάμε

από μία μεγάλη αγορά με πολλούς πελάτες.[47] Μία ακόμη επιλογή που μπορούμε να πάρουμε υπόψιν είναι να κάνουμε την ανάπτυξη του βιντεοπαιχνιδιού για συστήματα Android κάτι που έχουμε δει ότι είναι αρκετά γρήγορο με τις δυνατότητες της πλατφόρμας και το συγκεκριμένο είδος παιχνιδιού που αναπτύσσουμε μεταφράζεται εύκολα σε οθόνες κινητών και τάμπλετ. Όπως και με την αγορά των υπολογιστών έτσι και η αγορά των κινητών είναι τεράστια με αρκετούς διαθέσιμους πελάτες.[48]

4.β Σκοπός Παιχνιδιού & Οδηγίες Χρήσης

Όπως αναφέραμε στην προηγούμενη παράγραφο, έχοντας δεδομένο ότι το παιχνίδι μας θα είναι remaster ενός shoot ‘em up βιντεοπαιχνιδιού και συγκεκριμένα του Space Invaders, δεν θα παρεκκλίνουμε από τα βασικά στοιχεία του σκοπού του παιχνιδιού απλά θα κάνουμε την κίνηση των αεροσκαφών (παίκτη και εχθρικά) πιο δυναμική και πιο κοντά στα σύγχρονα δεδομένα. Θα κρατήσουμε τη βασική αρχή του να κρατάει το σκορ του παίκτη αλλά θα προσθέσουμε έξτρα στοιχεία αποστολών για έναν εναλλακτικό τρόπο παιχνιδιού ώστε να μην είναι μονοδιάστατο. Άρα μπορούμε να συνοψίσουμε τον σκοπό του παιχνιδιού μας ως ότι είμαστε πιλότοι που χειριζόμαστε ένα αεροσκάφος πετώντας σε γαλαξίες με σκοπό να φέρουμε εις πέρας τις αποστολές που μας έχουν ανατεθεί, καταστρέφοντας αντίπαλα αεροσκάφη και προσπαθώντας να μην χτυπηθούμε από αυτά και καταστραφεί το δικό μας.

4.β.i Τρόποι Παιχνιδιού

Το παιχνίδι μας διαθέτει δύο διαφορετικούς τρόπους-επιλογές για να παίξει ο χρήστης:

-Missions (Αποστολές): Στο συγκεκριμένο τρόπο παιχνιδιού έχουμε μία σειρά απο επίπεδα με αρχικά διαθέσιμο μόνο ένα. Σκοπός του παίκτη είναι να ολοκληρώσει την συγκεκριμένη αποστολή που του ανατίθεται στο επίπεδο (πχ καταστροφή συγκεκριμένο αριθμό αντίπαλων αεροσκαφών, επιβίωση για κάποιο χρονικο διάστημα κλπ.). Αφού την

ολοκληρώσει τότε τελειώνει το επίπεδο με επιτυχία και ξεκλειδώνει το επόμενο στη σειρά.

-Endless (Ατελείωτο): Το επίπεδο αυτό δεν τελειώνει ποτέ. Σκοπός του παίκτη είναι να επιβιώσει όσο το δυνατόν περισσότερο και να μαζέψει όσο περισσότερους πόντους μπορεί καταστρέφοντας αντίπαλα αεροσκάφη προσπαθώντας να κάνει highscore. Όσο περισσότερο επιβιώνει τόσο πιο πολύ αυξάνεται η δυσκολία του επιπέδου. Ο συγκεκριμένος τρόπος παιχνιδιού είναι και αυτός που θυμίζει περισσότερο το αρχικό Space Invaders.

4.β.ii Οδηγίες Χρήσης (Controls)

Η πλοήγηση στα μενού του παιχνιδιού γίνεται με τη χρήση του ποντικιού (mouse). Μέσα στα επίπεδα ελέγχουμε την κίνηση του αεροσκάφους μας είτε με τα βελάκια είτε με τα πλήκτρα “W” “A” “S” “D” που προσφέρουν τις ίδιες λειτουργίες με τα βελάκια πάνω, αριστερά, κάτω και δεξία αντίστοιχα. Πυροβολούμε πατώντας το πλήκτρο space. Πατώντας το πλήκτρο esc ενεργοποιούμε το μενού παύσης από το οποίο μπορούμε να βγούμε είτε χρησιμοποιώντας το ποντίκι είτε πατώντας ξανα το πλήκτρο esc. Παρατηρούμε ότι πετυχαίνουμε το στόχο το παιχνίδι μας να παραμείνει απλό και χωρίς πολλές οδηγίες χρήσης.

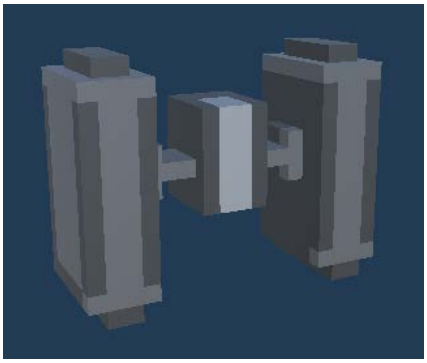
4.γ Μοντέλα & Στοιχεία

Σε αυτήν την παράγραφο θα κάνουμε μία αναφορά στα μοντέλα που δημιουργήσαμε για το παιχνίδι μας καθώς και σε όποια άλλα στοιχεία υπάρχουν σε αυτό δίνοντας μία συνοπτική περιγραφή για το κάθε ένα. Το μεγαλύτερο μέρος των μοντέλων έγινε με την χρήση του προγράμματος Blender και στη συνέχεια εισαγωγή τους στη Unity, ακόμη ένα μεγάλο πλεονέκτημα της πλατφόρμας καθώς είναι μία πολύ απλή διαδικασία.

4.γ.i Αεροσκάφη

Στο παιχνίδι μας υπάρχουν πέντε διαφορετικά ήδη αεροσκαφών:

-Αεροσκάφος παίκτη (Εικόνα 71)



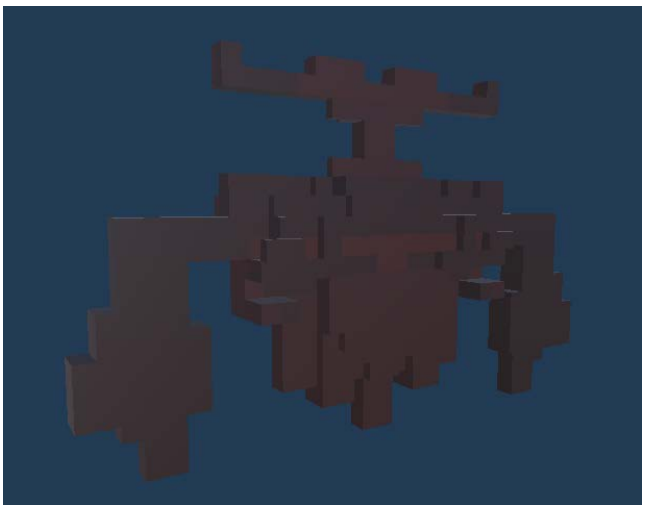
Εικόνα 71: Μοντέλο αεροσκάφους παίκτη
(πηγή δημιουργός)

Είναι το αεροσκάφος που χειρίζεται ο παίκτης με τον τρόπο που αναφέραμε στην προηγούμενη παράγραφο.

-Εχθρικά αεροσκάφη (Εικόνες 72, 73)

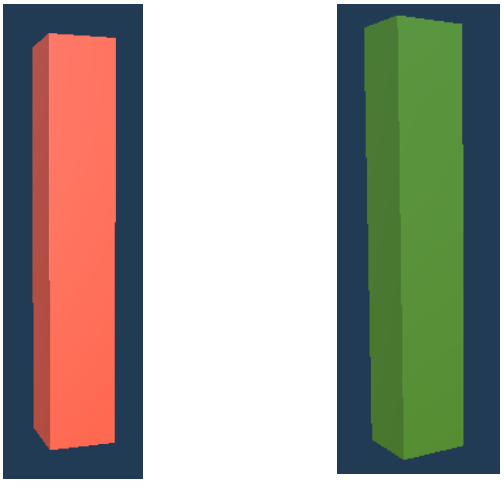
(α) (β) (γ)

Εικόνα 72: Μοντέλα εχθρικών αεροσκαφών
(πηγή δημιουργός)



Εικόνα 73: Μοντέλο αεροσκάφους τύπου boss
(πηγή δημιουργός)

Στην Εικόνα 72 βλέπουμε τους τρεις διαφορετικούς τύπους αεροσκαφών που έχει να αντιμετωπίσει ο παίκτης μας. Ο τύπος (α) κινείται γρήγορα αλλά καταστρέφεται εύκολα και δεν ρίχνει πυρά, ο (β) κινείται πιο αργά από τον (α) αλλά έχει μεγαλύτερη αντοχή και ρίχνει πυρά και ο τύπος (γ) που κινείται αρκετά αργά αλλά έχει τη μεγαλύτερη αντοχή και ρίχνει πυρά. Επιπροσθέτως στο τελευταίο επίπεδο αποστολών που έχουμε υλοποιήσει υπάρχει το αεροσκάφος τύπου boss που βλέπουμε στην Εικόνα 73. Το συγκεκριμένο δεν κινείται, καταλαμβάνει το πάνω μισό της οθόνης και αντί για πυρά δημιουργεί άλλα εχθρικά αεροσκάφη έχοντας επίσης πολύ μεγάλη αντοχή σε πυρά. Μπορούμε να δούμε τα μοντέλα από τα πυρά στην Εικόνα 74.



(i) (ii)

Εικόνα 74: Μοντέλα πυρών (i) πυρά παίκτη (ii) εχθρικά πυρά
(πηγή δημιουργός)

4.γ.ii Αναβαθμίσεις (Power Ups)

Κάθε φορά που καταστρέφουμε ένα εχθρικό αεροσκάφος υπάρχει πιθανότητα να αφήσει πίσω του μία αναβάθμιση για το δικό μας αεροσκάφος ώστε να μας βοηθήσει στην αποστολή μας. Οι αναβαθμίσεις είναι τρεις:

-Αναβάθμιση Ασπίδα (Εικόνα 75)

Εικόνα 75: Μοντέλο αναβάθμισης Ασπίδα
(πηγή δημιουργός)

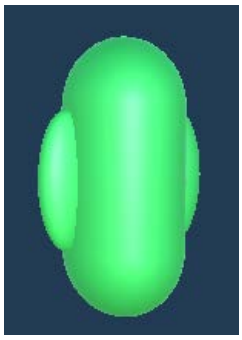
Αυτή η αναβάθμιση δημιουργεί μία μπλε σφαίρα γύρω από το αεροσκάφος του παίκτη και τον προστατεύει απο μία σύγκρουση που θα προκαλούσε μείωση ζωής.

-Αναβάθμιση Ζωή (Εικόνα 76)

Εικόνα 76: Μοντέλο αναβάθμισης Ζωή
(πηγή δημιουργός)

Μόλις ο παίκτης μαζέψει τη συγκεκριμένη αναβάθμιση οι τωρινοί πόντοι ζωής του αυξάνονται κατά έναν.

-Αναβάθμιση Πυρά (Εικόνα 77)



Εικόνα 77: Μοντέλο αναβάθμισης Πυρά
(πηγή δημιουργός)

Η πιο σπάνια και επιδραστική αναβάθμιση καθώς επιτρέπει στο αεροσκάφος του παίκτη να ρίχνει πυρά από περισσότερα του ενός σημεία μέχρι να φτάσει στα τρία. Διαρκεί συγκεκριμένο χρονικό διάστημα για λόγους ισορροπίας παιχνιδιού.

4.γ.iii Σωματίδια (Particles)

Στο παιχνίδι μας κάθε φορά που καταστρέφεται είτε κάποιο αεροσκάφος είτε κάποια σφαίρα, αφήνει πίσω του για μερικά δευτερόλεπτα ένα είδος Particle ώστε να δώσουμε παραπάνω κίνηση, χρώμα και δράση στη Σκηνή μας δημιουργώντας ένα ελεγχόμενο "χάος" στην οθόνη, ένα χαρακτηριστικό των σύγχρονων shoot 'em up.

Έχουμε υλοποιήσει εννιά διαφορετικά Particle: ένα για κάθε είδος σφαίρας, ένα για κάθε είδος απλού εχθρικού αεροσκάφους, ένα για το αεροσκάφος του παίκτη και τρία για το αεροσκάφος τύπου boss (Εικόνα 78).

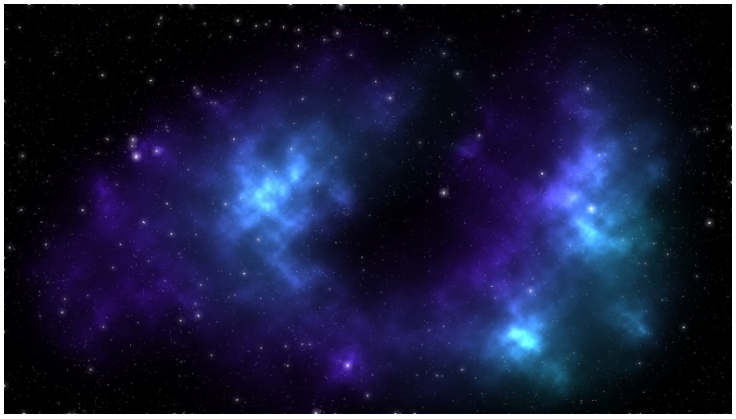
Εικόνα 78: Τα στοιχεία Particle στο παιχνίδι μας
(πηγή δημιουργός)

4.γ.iv Υλικά (Materials)

Στην κατηγορία των Υλικών ανήκουν τα χρώματα-υφές που έχουμε ενσωματώσει σαν Συστατικά στα Αντικείμενα μας για να είναι ευδιάκριτα και να διαφέρουν το ένα από το άλλο. Το παιχνίδι μας έχει δεκατρία διαφορετικά στοιχεία Υλικών: ένα για κάθε είδος σφαίρας, ένα για κάθε αεροσκάφος που υπάρχει, ένα για κάθε αναβάθμιση με την αναβάθμιση τύπου Ασπίδα να έχει ένα ακόμα καθώς θέλουμε διαφορετικό χρώμα όταν προβάλλουμε την ασπίδα γύρω από το αεροσκάφος του παίκτη και τέλος δύο για τα χρώματα των γραμμάτων στο μενού μας.

4.γ.v Φόντο (Backgrounds)

Για την βελτίωση της εμφάνισης από το κλασικό μαύρο φόντο του original παιχνιδιού και για να υπάρχει διαφορά μεταξύ των επιπέδων μας έχουμε χρησιμοποιήσει κάποιες εικόνες ως background τέτοιες ώστε να δίνεται η αίσθηση του διαστήματος. Κάθε background διαφέρει αρκετά από τα άλλα έτσι ώστε με μία πρώτη ματιά ο χρήστης να μπορεί να αναγνωρίσει το επίπεδο στο οποίο βρίσκεται (Εικόνες 79, 80, 81, 82). Έχουμε επίσης χρησιμοποιήσει ένα background για το κεντρικό μενού που αποτελεί μνεία στο Space Invaders (Εικόνα 83).



Εικόνα 79: Background επιπέδου Endless [49]

Εικόνα 80: Background επιπέδου Missions 1 [50]

Εικόνα 81: Background επιπέδου Missions 2 [51]

Εικόνα 82: Background επιπέδου Missions 3 [52]

Εικόνα 83: Background κεντρικού μενού [53]

4.γ.vi Γραμματοσειρές

Για να δώσουμε ένα χαρακτήρα και να πλησιάσουμε πιο κοντά στη θεματολογία του παιχνιδιού έχουμε φροντίσει να εισάγουμε κάποιες γραμματοσειρές για χρήση στα μενού μας και στο HUD καθώς η προεπιλεγμένη της Unity δεν μας καλύπτει σε αυτήν την περίπτωση. Οι εισαγωγές έγιναν με την χρήση της επέκτασης TextMesh Pro που ενσωματώνεται στην πλατφόρμα και μας δίνει μεγάλη ποικιλία επιλογών για την επεξεργασία της εμφάνισης κειμένου. Τα μενού χρησιμοποιούν την γραμματοσειρά SpaceAge, και το HUD την Fox Cavalier.

4.γ.vii Ήχοι

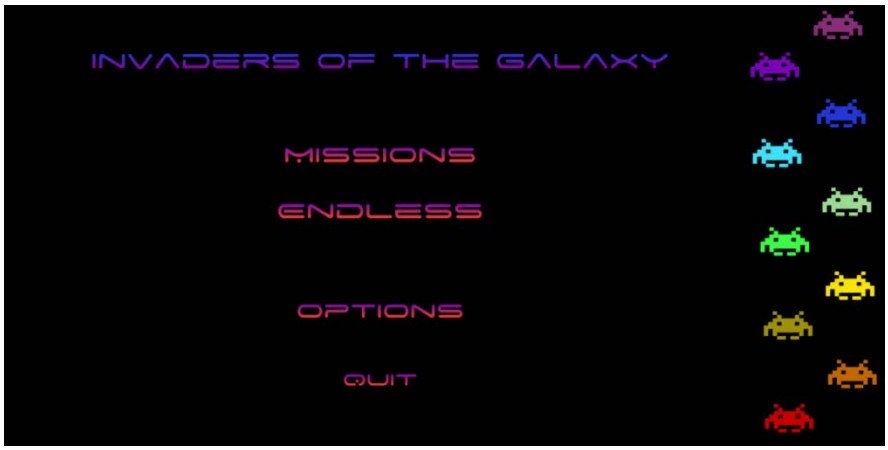
Όταν ο χρήστης πατάει το κουμπί για εκτόξευση πυρών ακούγεται ένας χαρακτηριστικός ήχος που βοηθάει στο να αισθανθεί ότι βρίσκεται μέσα στο παιχνίδι. Επίσης έτσι αποφεύγουμε να είναι βουβό το βιντεοπαιχνίδι μας και βοηθάμε στο να γίνει πιο διασκεδαστικό. Τον ήχο τον εισάγαμε από το site δωρεάν ηχητικών κλιπ freesound.org.

4.δ Μενού Παιχνιδιού

Σε αυτήν την παράγραφο θα αναλύσουμε τα δύο κυριότερα μενού του παιχνιδιού και τις λειτουργίες τους.

4.δ.i Κύριο Μενού (Main Menu)

Το κύριο μενού είναι το κεντρικό σημείο από το οποίο έχουμε πρόσβαση σε όλες τις λειτουργίες του παιχνιδιού και είναι το πρώτο πράγμα που βλέπει ο χρήστης όταν το ανοίγει το βιντεοπαιχνίδι μας (Εικόνα 84).



Εικόνα 84: Κύριο Μενού
(πηγή δημιουργός)

Αποτελείται από έναν διάφανο καμβά που καλύπτει όλη την έκταση της οθόνης και έχει στο πάνω μέρος το τίτλο του βιντεοπαιχνιδιού (Invaders of the Galaxy), από κάτω τους δύο διαφορετικούς τρόπους παιχνιδιού (Missions και Endless), κάτω από αυτά το κουμπί των επιλογών (Options) και τέλος το κουμπί εξόδου από το παιχνίδι (Quit).

Πατώντας το κουμπί Missions οδηγούμαστε σε μία νέα σελίδα του μενού που δείχνει τις διαθέσιμες αποστολές (Εικόνα 85).

Εικόνα 85: Μενού αποστολών
(πηγή δημιουργός)

Εδώ μπορούμε να δούμε τους τίτλους των αποστολών που υπάρχουν στο παιχνίδι. Μπορούμε να επιλέξουμε μία αποστολή μόνο εφόσον έχουμε νικήσει την προηγούμενή της και την έχουμε ξεκλειδώσει όπως έχουμε αναφέρει σε προηγούμενη παράγραφο με την πρώτη αποστολή (Hunt) να είναι πάντα ξεκλειδωμένη. Εφόσον επιλέξουμε μία αποστολή μεταφερόμαστε στο επίπεδό της για να ξεκινήσουμε. Πατώντας το κουμπί Back επιστρέφουμε στην προηγούμενη σελίδα του μενού.

Αν πατήσουμε το κουμπί Endless στο κύριο μενού ξεκινάμε να παίζουμε το συγκεκριμένο επίπεδο. Με το κουμπί Options οδηγούμαστε στις επιλογές του χρήστη όσον αφορά τις ρυθμίσεις του παιχνιδιού (Εικόνα 86). Σε αυτήν την έκδοση υπάρχει ένας slider για έλεγχο της έντασης του ήχου. Με το κουμπί Back επιστρέφουμε στην προηγούμενη σελίδα.

Εικόνα 86: Μενού ρυθμίσεων
(πηγή δημιουργός)

Τέλος, με το κουμπί Quit κλείνουμε το παράθυρο της εφαρμογής του παιχνιδιού.

4.δ.ii Μενού Παύσης (Pause Menu)

Εξίσου σημαντικό και με παρόμοιες λειτουργίες είναι το μενού παύσης (Εικόνα 87). Το ενεργοποιούμε πατώντας το πλήκτρο Esc ενώ βρισκόμαστε σε κάποιο επίπεδο.

Εικόνα 87: Μενού παύσης
(πηγή δημιουργός)

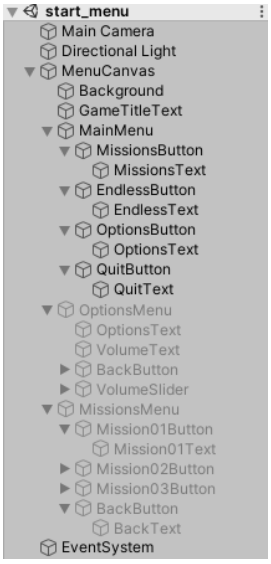
Είναι δομημένο με τον ίδιο τρόπο με το κύριο μενού δηλαδή ένας διάφανος καμβάς που καλύπτει όλη την οθόνη και όταν ενεργοποιηθεί παγώνει τη δράση που συμβαίνει στο επίπεδο. Στο πάνω μέρος αναφέρεται η κατάσταση του παιχνιδιού (Game Paused), αμέσως από κάτω υπάρχει το κουμπί επιστροφής στο επίπεδο (Resume), το κουμπί επιστροφής στο κύριο μενού (Main Menu), το κουμπί επιλογών (Options) που έχει την ίδια λειτουργία με το αντίστοιχο κουμπί στο κύριο μενού και το κουμπί εξόδου από το παιχνίδι (Quit). Μπορούμε επίσης να ξαναπατήσουμε το κουμπί Esc όταν είμαστε στη συγκεκριμένη σελίδα του μενού για να επιστρέψουμε στο επίπεδο (σαν να έχουμε πατήσει το Resume).

4.ε Σκηνές

Στη συγκεκριμένη παράγραφο θα δούμε τις Σκηνές που έχουμε υλοποιήσει για το βιντεοπαιχνίδι μας και θα δώσουμε μία περιγραφή της λειτουργίας τους.

4.ε.i Σκηνή start_menu

Είναι η Σκηνή που έχουμε αναλύσει στην παράγραφο 4.δ.i. Στην Εικόνα 88 βλέπουμε την ιεραρχεία των Αντικειμένων της Σκηνής.

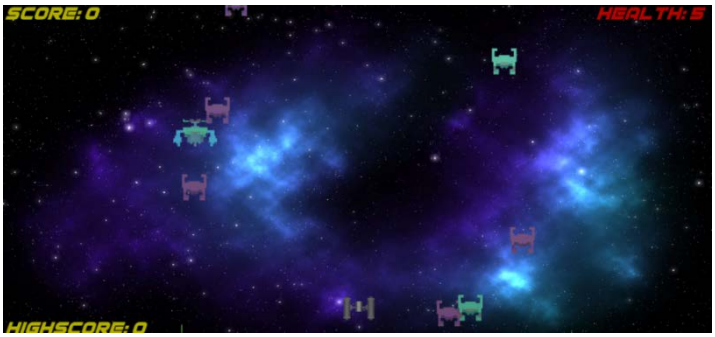


Εικόνα 88: Αντικείμενα start_menu
(πηγή δημιουργός)

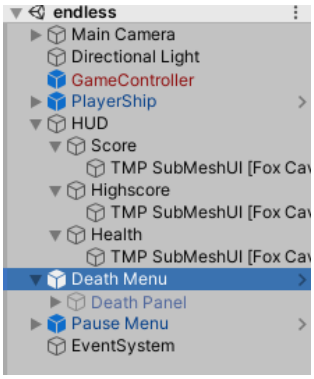
Όλα τα στοιχεία που βλέπουμε στην οθόνη είναι παιδιά του καμβά μενού. Ανάλογα σε ποια σελίδα του μενού βρισκόμαστε είναι απενεργοποιημένα και συγκεκριμένα Αντικείμενα όπως για παράδειγμα όταν ξεκινάει το παιχνίδι και είμαστε στην κεντρική σελίδα του, οι σελίδες με το μενού αποστολών και με τις επιλογές ρυθμίσεων δεν είναι ενεργές (MissionsMenu και OptionsMenu αντίστοιχα) κοκ. Επιπροσθέτως ο καμβάς περιέχει και script που βασικός του ρόλος είναι να εκτελεί τις λειτουργίες των κουμπιών και να ελέγχει διαβάζοντας το αρχείο PlayerPrefs (αρχείο που αποθηκεύει η Unity δεδομένα παιχνιδιού για τον συγκεκριμένο χρήστη) αν και ποια επίπεδα αποστολών έχει ξεκλειδώσει ώστε να του επιτρέψει να τα πατήσει.

4.ε.ii Σκηνή endless

Αυτή η Σκηνή είναι η υλοποίηση του επιπέδου Endless και στην Εικόνα 89 βλέπουμε ένα στιγμιότυπό της. Στην Εικόνα 90 βλέπουμε τα Αντικείμενα που την απαρτίζουν.



Εικόνα 89: Στιγμιότυπο επιπέδου Endless
(πηγή δημιουργός)



Εικόνα 90: Αντικείμενα endless
(πηγή δημιουργός)

Το Αντικείμενο GameController είναι υπεύθυνο για αρκετές παραμέτρους στα επίπεδα του παιχνιδιού όπως ο βαθμός δυσκολίας και η ολοκλήρωση της αποστολής και θα αναλυθεί σε επόμενη παράγραφο. Το Αντικείμενο PlayerShip είναι το αεροσκάφος που ελέγχει ο παίκτης το οποίο επίσης θα αναλυθεί σε επόμενη παράγραφο. Το Αντικείμενο HUD είναι ένας καμβάς που περιέχει πληροφορίες για τον χρήστη για την συγκεκριμένη συνεδρία του παιχνιδιού όπως το τωρινό σκορ του (Score), του τωρινούς πόντους ζωής του (Health) καθώς και το μεγαλύτερο σκορ που έχει γίνει στο συγκεκριμένο μηχάνημα (Highscore) το οποίο υπάρχει στο αρχείο PlayerPrefs και ανανεώνεται αυτόματα όταν το Score ξεπεράσει το Highscore. Το Death Menu είναι ένας καμβάς που βλέπει ο χρήστης όταν καταστραφεί το αεροσκάφος του (Εικόνα 91).

Εικόνα 91: Death Menu στο επίπεδο Endless
(πηγή δημιουργός)

Μας ανακοινώνει το τέλος της συνεδρίας παιχνιδιού καθώς καταστράφηκε το αεροσκάφος μας, το σκορ που πετύχαμε και μας δίνει τις επιλογές είτε να επαναλάβουμε το επίπεδο (Restart) είτε να επιστρέψουμε στο κεντρικό μενού (Main Menu) είτε να κλείσουμε το παράθυρο του παιχνιδιού (Quit).

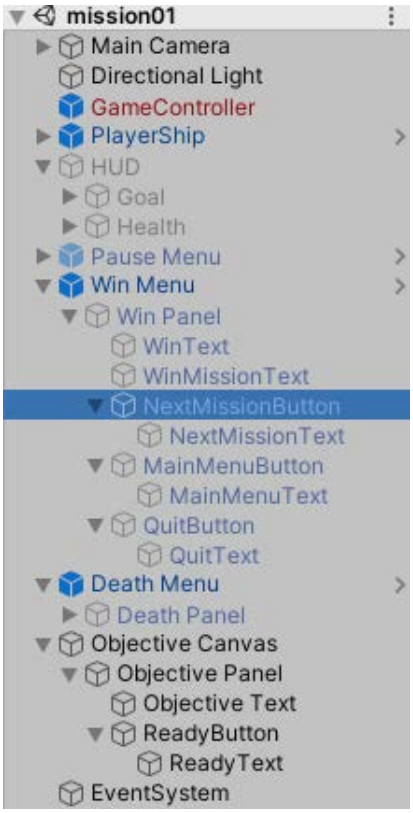
Τέλος το Pause Menu το έχουμε αναλύσει στην προηγούμενη παράγραφο.

4.ε.iii Σκηνές Αποστολών

Καθώς οι Σκηνές αποστολών είναι παρόμοιες σε λειτουργίες και Αντικείμενα θα αναλυθούν όλες μαζί.

-Mission01 “Hunt”

Τα Αντικείμενα της Σκηνής φαίνονται στην Εικόνα 92.



Εικόνα 92 Αντικείμενα mission01
(πηγή δημιουργός)

Σε σχέση με το επίπεδο Endless παρατηρούμε ότι υπάρχει το Αντικείμενο Objective Canvas που είναι ένας καμβάς που βλέπει ο χρήστης όταν ξεκινήσει το επίπεδο και λειτουργεί σαν ενημέρωση ως προς την αποστολή που έχει να ολοκληρώσει (Εικόνα 93). Στην πρώτη αποστολή έχει να καταστρέψει έναν συγκεκριμένο αριθμό ενός συγκεκριμένου τύπου εχθρικών αεροσκαφών. Το επίπεδο δεν ξεκινάει αν ο παίκτης δεν πατήσει το κουμπί Ready.

Εικόνα 93: Objective canvas mission01
(πηγή δημιουργός)

Τα Αντικείμενα HUD και Pause Menu είναι επίσης απενεργοποιημένα όσο υπάρχει αυτός ο καμβάς στην οθόνη. Επίσης εδώ το HUD δείχνει μόνο τους τωρινούς πόντους ζωής και την πρόοδο της αποστολής καθώς δεν υπάρχει κάποιο σκορ όπως δεν φαίνεται και το σκορ στο Death Menu στα επίπεδα αποστολών.

Τέλος, ένα σημαντικό νέο στοιχείο είναι το Αντικείμενο Win Menu το οποίο εκτελεί παρόμοια λειτουργία με το Death Menu αλλά εμφανίζεται σε περίπτωση επιτυχημένης ολοκλήρωσης της αποστολής (Εικόνα 94).

Εικόνα 94: Win Menu στο επίπεδο Hunt
(πηγή δημιουργός)

Μας εμφανίζει μήνυμα επιτυχίας και αντί για την επιλογή Restart του Death Menu μας δίνει την επιλογή να πάμε αμέσως στο επόμενο επίπεδο αποστολών καθώς πλεόν το έχουμε ξεκλειδώσει.

-Mission02 “Backup”

Παρόμοια υλοποίηση έχει και το δεύτερο επίπεδο αποστολών με τη διαφορά στον τύπο της αποστολής καθώς εδώ ο χρήστης πρέπει να επιβιώσει για συγκεκριμένο χρονικό διάστημα (Εικόνα 95).

Εικόνα 95: Objective canvas mission02
(πηγή δημιουργός)

-Mission03 “Mother”

Εδώ τα πράγματα δυσκολεύουν για τον παίκτη καθώς έχει να αντιμετωπίσει το αεροσκάφος τύπου boss και η αποστολή του δεν είναι άλλη από το να το καταστρέψει (Εικόνα 96).

Εικόνα 96: Objective canvas mission03
(πηγή δημιουργός)

Η μόνη διαφορά στα Αντικείμενα σε αυτήν τη Σκηνή είναι ότι καθώς αυτό είναι το τελευταίο επίπεδο στην συγκεκριμένη έκδοση του παιχνιδιού το Win Menu αντί για την

δυνατότητα να πάμε το επόμενο επίπεδο μας έχει την επιλογή να επαναλάβουμε το συγκεκριμένο (Restart).

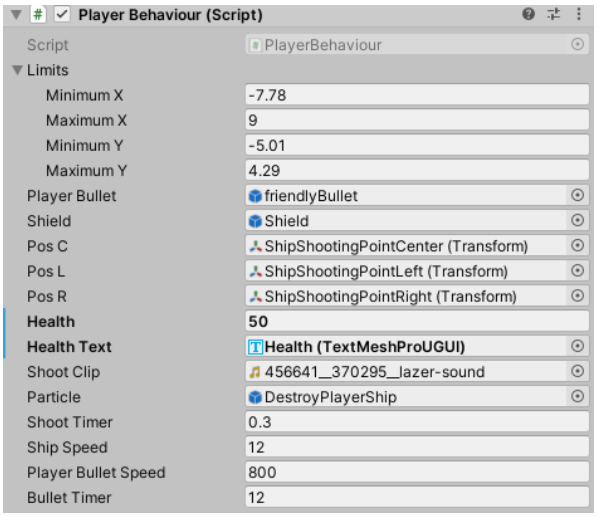
4.ζ Κίνηση Παίκτη

Το Αντικείμενο PlayerShip είναι ο χαρακτήρας που ελέγχει ο παίκτης. Είναι ένα Prefab και αποτελείται απο τα Αντικείμενα της Εικόνας 97.

Εικόνα 97: Prefab PlayerShip
(πηγή δημιουργός)

Ο γονέας PlayerShip περιέχει σαν Συστατικά τον Collider, την πηγή ήχου (Audio Source) και το script διαχείρισης του αεροσκάφους. Το Αντικείμενο default περιέχει το μοντέλο του αεροσκάφους, το Material καθώς και το Mesh. Τα επόμενα τρία Αντικείμενα είναι απλά σημεία πάνω στο αεροσκάφος από τα οποία δημιουργούμε τα πυρά ανάλογα και με τις αναβαθμίσεις που έχουμε. Τέλος το Αντικείμενο Shield που ξεκινάει απενεργοποιημένο, γίνεται ενεργό μόνο σε περίπτωση που πάρουμε αναβάθμιση Ασπίδα και εμφανίζει μία ασπίδα γύρω από το αεροσκάφος μας.

Στην Εικόνα 98 μπορούμε να δούμε όλα τα πεδία που εμφανίζει σαν ιδιότητες το script της συμπεριφοράς του αεροσκάφους παίκτη.



Εικόνα 98: Το Συστατικό Player Behaviour Script
(πηγή δημιουργός)

-Limits: Τα όρια της οθόνης ώστε να μην μπορεί ο παίκτης να βγει εκτός απο αυτήν

-Player Bullet: Αναφορά σε Αντικείμενο για τα πυρά του παίκτη

-Shield: Αναφορά σε Αντικείμενο για την εμφάνιση της Ασπίδας γύρω από το αεροσκάφος

-Pos C/L/R: Αναφορές σε Transforms για τα σημεία στο αεροσκάφος που δημιουργούμε τα πυρά

-Health: Οι πόντοι ζωής του παίκτη

-Health Text: Αναφορά σε TextMeshProUGUI για το κείμενο Health στο HUD

-Shoot Clip: Αναφορά σε AudioClip για τον ήχο που ακούγεται όταν ο χρήστης βάλλει πυρά

-Particle: Αναφορά σε Αντικείμενο για το Particle που θα εμφανιστεί όταν καταστραφεί το αεροσκάφος παίκτη

-Shoot Timer: Χρόνος καθυστέρησης μεταξύ πυρών, για λόγους ισορροπίας παιχνιδιού

-Ship Speed: Η ταχύτητα με την οποία κινείται το αεροσκάφος παίκτη

-Player Bullet Speed: Η ταχύτητα με την οποία κινούνται τα πυρά του παίκτη

-Bullet Timer: Ο χρόνος μέχρι να εξασθενήσει η αναβάθμιση Πυρά

Όσον αφορά το περιεχόμενο του script πέρα από τις συναρτήσεις γεγονότων έχουμε υλοποιήσει και δικές μας. Μπορούμε να δούμε ολόκληρο το script στα Παραρτήματα με το όνομα PlayerBehaviour.

-Start: Θέτει την μεταβλητή-flag hasLost που ελέγχει αν έχει καταστραφεί το αεροσκάφος μας σε false καθώς μόλις ξεκινήσαμε τη συνεδρία του παιχνιδιού. Επίσης παίρνουμε την αναφορά για την πηγή ήχου, ρυθμίζουμε τους χρόνους για αντίστροφη μέτρηση των Shoot Timer και Bullet Timer και αρχικοποιούμε την αναβάθμιση πυρά στο 0.

-Update: Εδώ καλούμε συναρτήσεις που θέλουμε να γίνονται σε κάθε ανανέωση frame. Αυτές είναι οι DisplayHealth, PlayerInput, Bounds, Death και FadeBullet.

-DisplayHealth: Είναι υπεύθυνη για το τι δείχνει το κείμενο του Health στο HUD.

-PlayerInput: Συνάρτηση υπεύθυνη για τα δεδομένα που δίνει ο παίκτης μέσα από τα κουμπιά που πατάει όσον αφορά την κίνηση και τα πυρά. Επίσης αναλαμβάνει τον υπολογισμό για το αν έχει περάσει ο χρόνος καθυστέρησης μεταξύ πυρών και εφόσον έχει περάσει και ο χρήστης πατάει το κουμπί για ρίψη πυρών τότε μόνο του το επιτρέπει και ανανεώνει τον χρόνο.

-Bounds: Κρατάει τον παίκτη εντός ορίων οθόνης.

-Death: Ελέγχει αν οι πόντοι ζωής είναι από 0 και κάτω και στην περίπτωση που είναι θέτει το hasLost σε true, καταστρέφει το αεροσκάφος και εμφανίζει το Particle.

-FadeBullet: Στην περίπτωση που η αναβάθμιση πυρά είναι μεγαλύτερη του 0 σημαίνει ότι έχουμε κάποια ενεργή και πρέπει να αρχίσει η αντίστροφη μέτρηση για εξασθένηση της αναβάθμισης. Στην περίπτωση που ο χρόνος αυτός φτάσει στο 0 τότε μειώνουμε τον αριθμό της αναβάθμισης κατά 1 και ανανεώνουμε τον χρόνο.

-FixedUpdate: Εδώ καλούμε τις συναρτήσεις που σχετίζονται με το σύστημα φυσικής. Είναι οι Movement και Shoot.

-Movement: Εφαρμόζει κίνηση ανάλογα το ποια κουμπιά έχουμε δει ότι έχει πατήσει ο χρήστης στην συνάρτηση PlayerInput και την ταχύτητα που έχουμε δώσει στο πεδίο Ship Speed.

-Shoot: Αν ο χρήστης έχει πατήσει το κουμπί για ρήψη πυρών, παίζουμε μία φορά το κλιπ ήχου και ανάλογα με την τιμή της αναβάθμισης πυρών δημιουργούμε τα Αντικείμενα πυρών παίκτη από τα αντίστοιχα σημεία και τους δίνουμε ταχύτητα ανάλογη της τιμής στο πεδίο Player Bullet Speed.

-OnTriggerEnter: Αυτή η συνάρτηση είναι υπεύθυνη για να αναγνωρίζει τις κρούσεις του αεροσκάφους με τα υπόλοιπα Αντικείμενα και να εκτελεί τις ανάλογες ενέργειες. Αν έχουμε κρούση με Αντικείμενο αναβάθμισης πυρών (tag BulletPowerUp) τότε το

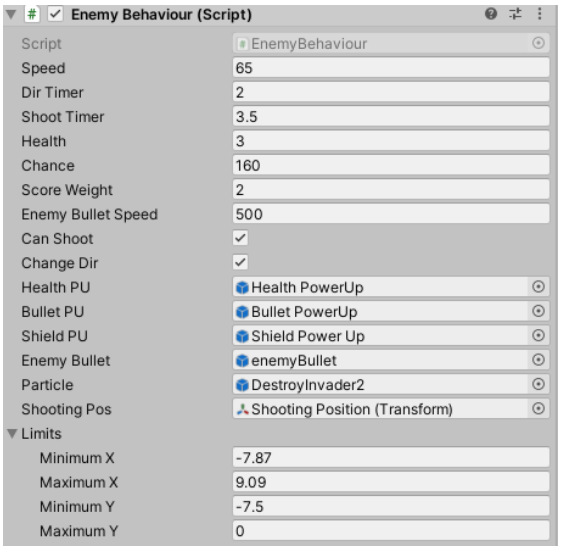
καταστρέφουμε, ανανεώνουμε τον χρόνο εξασθένησης και αν δεν έχουμε ήδη φτάσει στο μέγιστο αριθμό που μπορεί να έχει αυτη η αναβάθμιση (2) τότε αυξάνουμε τη μεταβλητή κατά 1. Σε κρούση με Αντικείμενο αναβάθμισης ασπίδα (tag ShieldPowerUp) το καταστρέφουμε και αν δεν είναι ενεργοποιημένο το Αντικείμενο shield το ενεργοποιούμε. Στην περίπτωση που το Αντικείμενο είναι αναβάθμιση ζωής (tag HealthPowerUp) το καταστρέφουμε και αυξάνουμε τους τωρινούς πόντους ζωής του παίκτη κατά 1. Αν υπάρχει κρούση με εχθρικό αεροσκάφος (tag Enemy) μειώνουμε τους πόντους ζωής του παίκτη κατά 2, με εχθρικά πυρά (tag Enemy Bullet) τους μειώνουμε κατά 1 και τέλος αν υπάρχει κρούση με το εχθρικό σκάφος τύπου boss (tag Mothership) θέτουμε τους τωρινούς πόντους ζωής στο 0.

4.η Συμπεριφορές & Τεχνητή Νοημοσύνη Εχθρών

Όπως στην προηγούμενη παράγραφο είδαμε το script για το αεροσκάφος που ελέγχει ο παίκτης, σε αυτήν θα δούμε τα scripts συμπεριφοράς των εχθρών και την τεχνητή νοημοσύνη που χρησιμοποιούν. Επιπροσθέτως θα δούμε και το script ελέγχου παιχνιδιού (GameController). Στα Παραρτήματα μπορούμε να τα βρούμε με το όνομα EnemyBehaviour, MothershipBehaviour και GameController.

4.η.i EnemyBehaviour Script

Αυτό το script επισυνάπτεται σε όλα τα εχθρικά αεροσκάφη πλην του αεροσκάφους τύπου boss και στην Εικόνα 99 μπορούμε να δούμε τα πεδία που εμφανίζει σαν ιδιότητες. Οι τιμές και οι αναφορές στα πεδία διαφέρουν ανάλογα με τον τύπο αεροσκάφους.



Εικόνα 99: Το Συστατικό Enemy Behaviour Script
(πηγή δημιουργός)

-Speed: Η ταχύτητα κίνησης του εχθρικού αεροσκάφους

-Dir Timer: Χρόνος μεταξύ αλλαγής κατεύθυνσης κατά την κίνηση

-Shoot Timer: Χρόνος καθυστέρησης μεταξύ πυρών

-Health: Πόντοι ζωής εχθρικού αεροσκάφους

-Chance: Πιθανότητα να αφήσει πίσω του αναβάθμιση όταν καταστραφεί το συγκεκριμένο αεροσκάφος, όσο μεγαλύτερο το νούμερο τόσο πιο σπάνια συμβαίνει

-Score Weight: Πολλαπλασιαστής σκορ ανάλογα με το τύπο αεροσκάφους με τιμή ανάλογη της δυσκολίας να καταστραφούν

-Enemy Bullet Speed: Ταχύτητα εχθρικών πυρών

-Can Shoot: Τσεκαρισμένο σε περίπτωση που το συγκεκριμένο αεροσκάφος βάλλει πυρά

-Change Dir: Διακόπτης αλλαγής κατεύθυνσης (αριστερά/δεξιά)

-Health/Bullet/Shield PU: Αναφορές στα Αντικείμενα αναβαθμίσεων

-Enemy Bullet: Αναφορά στο Αντικείμενο για τα εχθρικά πυρά

-Particle: Αναφορά σε Αντικείμενο για το Particle που θα εμφανιστεί όταν καταστραφεί το αεροσκάφος

-Shooting Pos: Αναφορά σε Transform για το σημείο στο αεροσκάφος που δημιουργούμε τα πυρά

-Limits: Τα όρια της οθόνης ώστε να μην μπορεί να βγει το εχθρικό αεροσκάφος εκτός

Ακολουθούν οι συναρτήσεις του script:

-Start: Εδώ αρχικοποιούμε μία μεταβλητή (curScene) με το όνομα της Σκηνής στην οποία βρισκόμαστε καθώς μας χρειάζεται η συγκεκριμένη πληροφορία. Επίσης δημιουργούμε μία αναφορά για το Rigidbody του αεροσκάφους και ρυθμίζουμε τους χρόνους για αντίστροφη μέτρηση των Shoot Timer και Dir Timer.

-Update: Εδώ καλούμε συναρτήσεις που θέλουμε να γίνονται σε κάθε ανανέωση frame. Αυτές είναι οι Change, Death και Shooting.

-Change: Συνάρτηση υπεύθυνη για να αλλάζει τον διακόπτη changeDir που δίνει την κατεύθυνση κίνησης. Μετρά αντίστροφα και όταν ο Dir Timer φτάσει στο 0 αλλάζει την τιμή και ανανεώνει τον χρόνο.

-Death: Αν το αεροσκάφος βγει εκτός της οθόνης από την κάτω μεριά της τότε καταστρέφει το αεροσκάφος. Επίσης αν οι πόντοι ζωής του πέσουν στους 0 φροντίζει να καλέσει τη συνάρτηση DropPowerUp και στη συνέχει ανάλογα τη Σκηνή εκτελεί συγκεκριμένες ενέργειες. Αν η Σκηνή είναι η endless τότε φροντίζει να προσθέσει το σκορ επί τον πολλαπλασιαστή του στο συνολικό σκορ του επιπέδου. Αν η Σκηνή είναι η mission01 (επίπεδο που έχουμε να καταστρέψουμε συγκεκριμένο τύπο αεροσκάφους) και το αεροσκάφος είναι ο τύπος που θέλουμε τότε αυξάνει την πρόοδο της αποστολής κατά 1. Τέλος φροντίζει να καταστρέψει το Αντικείμενο και να εμφανίσει το Particle.

-DropPowerUp: Δημιουργεί έναν τυχαίο αριθμό από το 1 μέχρι την τιμή του πεδίου Chance και βγάζει την ανάλογη αναβάθμιση. Περισσότερες πιθανότητες έχει να μην βγάλει καμία αναβάθμιση και λιγότερες πιθανότητες έχει να βγάλει αναβάθμιση πυρών.

-Shooting: Ελέγχει αν το συγκεκριμένο αεροσκάφος βάλλει πυρά και σε αυτήν την περίπτωση μετράει αντίστροφα από τον χρόνο καθυστέρησης πυρών. Όταν φτάσει στο 0, το αεροσκάφος βάλλει πυρά από το σημείο ρίψης με την ταχύτητα του αντίστοιχου πεδίου και ανανεώνεται ο χρόνος καθυστέρησης.

-FixedUpdate: Εδώ καλούμε την συνάρτηση που σχετίζεται με το σύστημα φυσικής και είναι η Movement.

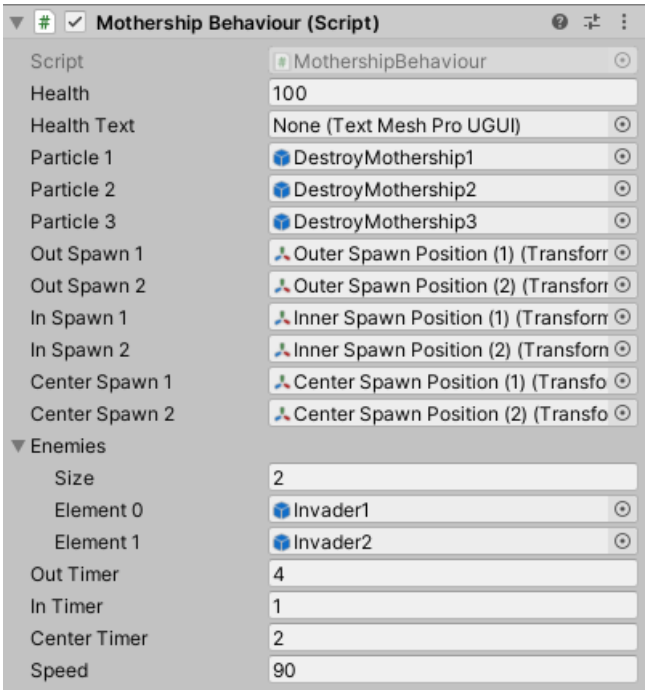
-Movement: Η συνάρτηση αυτή είναι υπεύθυνη για την τήρηση των ορίων της οθόνης στον χ άξονα από το αεροσκάφος και σε περίπτωση που πάει να βγει αλλάζει την κατεύθυνσή του. Επίσης φροντίζει να δώσει επιτάχυνση στο αεροσκάφος ανάλογη της τιμής του πεδίου Speed και με κατεύθυνση ανάλογα την τιμή του Change Dir.

-OnTriggerEnter: Αυτή η συνάρτηση είναι υπεύθυνη για να αναγνωρίζει τις κρούσεις του αεροσκάφους με τα υπόλοιπα Αντικείμενα και να εκτελεί τις ανάλογες ενέργειες. Αν έχουμε κρούση με Αντικείμενο πυρών παίκτη (tag Friendly Bullet) μειώνουμε τους πόντους ζωής του κατά 1 και αν η Σκηνή μας είναι η endless τότε προσθέτουμε πόντους στο συνολικό σκορ. Σε κρούση με το Αντικείμενο παίκτη (tag Player) θέτουμε τους πόντους ζωής στο 0. Το ίδιο κάνουμε και στην περίπτωση που το Αντικείμενο είναι η ασπίδα του παίκτη (tag Shield) αλλά σε αυτήν την περίπτωση θέτουμε επιπλέον την ασπίδα ανενεργή.
-SetHealth: Συνάρτηση που θέτει τους πόντους ζωής του αεροσκάφους στο 0 γα χρήση από το MothershipBehaviour Script.

Επομένως η συμπεριφορά των εχθρικών αεροσκαφών είναι από την στιγμή που δημιουργούνται να κατεβαίνουν προς το κάτω μέρος της οθόνης με μία ταχύτητα και έναν ρυθμό αλλαγής κατεύθυνσης και να βάλλουν πυρά ανάλογα με τον τύπο τους. Το διάστημα ζωής τους είναι μέχρι οι πόντοι ζωής τους να μηδενιστούν ή να βγούν εκτός οθόνης από το κάτω μέρος της.

4.η.ii MothershipBehaviour Script

Το συγκεκριμένο script (Εικόνα 100) επισυνάπτεται μόνο στο αεροσκάφος τύπου boss.



Εικόνα 100: Το Συστατικό Mothership Behaviour Script
(πηγή δημιουργός)

-Health: Οι πόντοι ζωής του αεροσκάφους

-Health Text: Αναφορά σε TextMeshProUGUI για το κείμενο του Mothership Health στο HUD

-Particle 1/2/3: Αναφορές σε Αντικείμενα για τα Particle που θα εμφανιστούν όταν καταστραφεί το αεροσκάφος

-Out/In/Center Spawn 1/2: Αναφορές σε έξι διαφορετικά σημεία πάνω στο αεροσκάφος από τα οποία δημιουργούνται άλλα αεροσκάφη, δύο εξωτερικά, δύο εσωτερικά και δύο στο κέντρο

-Enemies: Πίνακας με αναφορές σε Αντικείμενα εχθρικών αεροσκαφών που θα δημιουργηθούν, στην συγκεκριμένη έκδοση αεροσκάφη τύπου α και β

-Out/In/Center Timer: Χρόνοι μεταξύ δημιουργίας νέου εχθρικού αεροσκάφους από τα αντίστοιχα σημεία

-Speed: Η ταχύτητα με την οποία εμφανίζεται το αεροσκάφος

Ακολουθούν οι συναρτήσεις του script:

-Start: Εδώ γίνονται οι ακόλουθες αρχικοποιήσεις. Θέτουμε την μεταβλήτη-flag του GameController, hasWon σε false για να μην μας εμφανίζει το μενού νίκης εφόσον δεν χρησιμοποιείται το GameController σε αυτή τη Σκηνή. Θέτουμε τις μεταβλητές για αυξανόμενα στάδια δυσκολίας του επιπέδου (firstStage και secondStage) επίσης false όπως το ίδιο και τη μεταβλητή για το αν έχει νικηθεί το αεροσκάφος, wonBoss. Θέτουμε τη μεταβλητή lvlStart σε true για να δηλώσουμε ότι ξεκίνησε το επίπεδο και να εμφανίσουμε τον καμβά αποστολής. Επιπροσθέτως αρχικοποιούμε τους χρόνους για αντίστροφη μέτρηση των Spawn, δημιουργουμε μία αναφορά για το Rigidbody του αεροσκάφους και δίνουμε την επιτάχυνση ανάλογη του πεδίου Speed ώστε να εμφανιστεί στην οθόνη μας το αεροσκάφος.

-Update: Ελέγχει αν το αεροσκάφος έχει φτάσει στη σωστή θέση στην οθόνη. Αν είναι η πρώτη φορά που φτάνει θέτει το flag lvlStart σε false και μηδενίζει την επιτάχυνσή του. Επίσης καλεί σε κάθε ανανέωση frame τις συναρτήσεις DisplayHealth, Death και Spawner.

-DisplayHealth: Είναι υπεύθυνη για το τι δείχνει το κείμενο του Mothership Health στο HUD.

-Death: Ελέγχει αν οι πόντοι ζωής του αεροσκάφους πέσουν στο 0. Σε αυτήν την περίπτωση ελέγχει τη Σκηνή για Αντικείμενα με tag Enemy και τα καταστρέφει μέσω της συνάρτησης SetHealth που αναφέραμε καθώς θέλουμε όταν καταστραφεί το boss να καταστραφεί και όποιο άλλο εχθρικό αεροσκάφος υπάρχει στην οθόνη δημιουργώντας μία ωραία στιγμή. Στη συνέχεια εμφανίζει τα Particle καταστρέφει το αεροσκάφος και θέτει το flag wonBoss σε true.

-Spawner: Συνάρτηση υπεύθυνη για τη δημιουργία εχθρικών αεροσκαφών από τα αντίστοιχα σημεία. Μετράει αντίστροφα όλους τους χρόνους για τα Spawns και στη συνέχεια ελέγχει τους πόντους ζωής του αεροσκάφους boss. Όταν ξεπεράσουν ένα όριο κάνει με τη σειρά τα flags firstStage και secondStage true και μειώνει τους χρόνους μεταξύ κάθε νέας δημιουργίας εχθρικού αεροσκάφους ανάλογα με αυτά τα flags. Για παράδειγμα όταν το secondStage είναι true οι χρόνοι έχουν μειωθεί στο μισό και έχουμε αρκετά μεγαλύτερο βαθμό δυσκολίας. Όταν οποιοσδήποτε χρόνος φτάσει στο 0 καλείται

και η ανάλογη συνάρτηση δημιουργίας (SpawnCenterEnemy, SpawnInEnemy, SpawnOutEnemy) και ανανεώνεται ο συγκεκριμένος χρόνος.

-SpawnCenterEnemy: Συνάρτηση δημιουργίας εχθρικών αεροσκαφών στο κέντρο του αεροσκάφους boss. Δημιουργεί έναν τυχαίο αριθμό για να επιλέξει ποιον από τους 2 τύπους θα δημιουργήσει. Όταν αποφασίσει για τον τύπο τότε από τα αντίστοιχα σημεία (με μία μικρή απόκλιση για λόγους ποικιλίας) δημιουργεί τα αεροσκάφη.

-SpawnInEnemy: Παρόμοια με την προηγούμενη συνάρτηση αλλά για τα σημεία δημιουργίας εσωτερικά.

-SpawnOutEnemy: Παρόμοια με την προηγούμενη συνάρτηση αλλά για τα σημεία δημιουργίας εξωτερικά.

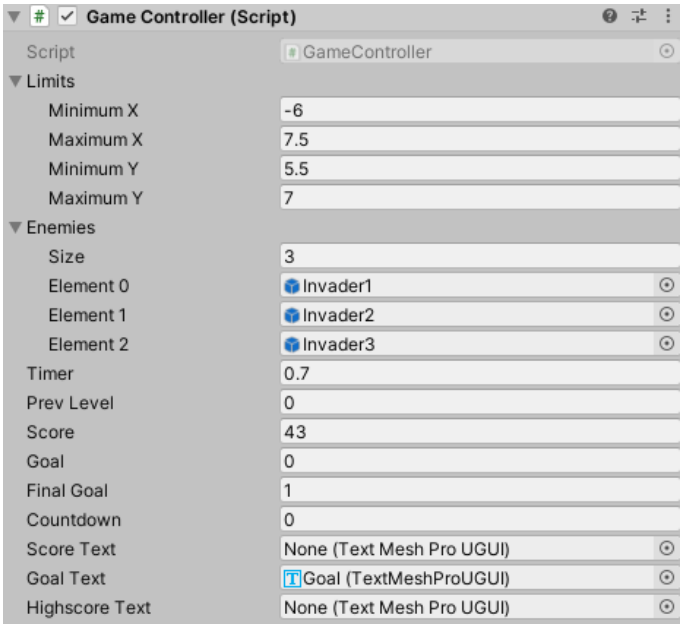
-OnTriggerEnter: Αυτή η συνάρτηση είναι υπεύθυνη για να αναγνωρίζει τις κρούσεις του αεροσκάφους με τα υπόλοιπα Αντικείμενα και να εκτελεί τις ανάλογες ενέργειες. Σε κρούση με το Αντικείμενο ασπίδα (tag Shield) μειώνουμε τους πόντους ζωής κατά 1 και θέτουμε την ασπίδα ανενεργή.

-CollidedWithBullet: Συνάρτηση που μειώνει τους πόντους ζωής κατά 1 και καλείται απο το script τον πυρών του παίκτη.

Επομένως η συμπεριφορά του συγκεκριμένου αεροσκάφους είναι να εμφανίζεται επιβλητικά και να παραμένει στο ίδιο σημείο δημιουργώντας συνεχώς εχθρικά αεροσκάφη όλο και πιο γρήγορα ανάλογα τους πόντους ζωής που έχει μέχρι αυτοί να πέσουν στο 0 και να ολοκληρωθεί επιτυχώς η αποστολή.

4.η.iii GameController Script

Το συγκεκριμένο script (Εικόνα 101) είναι πολύ σημαντικό καθώς είναι αυτό που ελέγχει τη ροή παιχνιδιού, ελέγχει το σκορ και την πρόοδο των αποστολών και δημιουργεί τα εχθρικά αεροσκάφη. Υπάρχει σε όλα τα επίπεδα σε αυτήν την έκδοση του παιχνιδιού εκτός από το επίπεδο mission03 καθώς οι βασικές λειτουργίες του έχουν ενσωματωθεί στο MothershipBehaviour Script. Στα Παραρτήματα το βρίσκουμε με το όνομα GameController.



Εικόνα 101: Το Συστατικό Game Controller Script
(πηγή δημιουργός)

-Limits: Πίνακας τιμών με τα όρια της δημιουργίας

-Enemies: Πίνακας με αναφορές σε Αντικείμενα εχθρικών αεροσκαφών που θα δημιουργηθούν

-Timer: Χρόνος μεταξύ δημιουργίας νέων αεροσκαφών

-Prev Level: Κρατάει το προηγούμενο επίπεδο δυσκολίας (μόνο στο επίπεδο endless)

-Score: Κρατάει το σκορ του παιχνιδιού (μόνο στο επίπεδο endless)

-Goal: Κρατάει την πρόοδο που έχει γίνει στην αποστολή του παιχνιδιού (μόνο στο επίπεδο mission01)

-Final Goal: Περιέχει τον τελικό στόχο του επιπέδου (μόνο στο επίπεδο mission01)

-Countdown: Περιέχει τον χρόνο της αντίστροφης μέτρησης του επιπέδου (μόνο στο επίπεδο mission02)

-Score Text: Αναφορά σε TextMeshProUGUI για το κείμενο Score στο HUD (μόνο στο επίπεδο endless)

-Goal Text: Αναφορά σε TextMeshProUGUI για το κείμενο Goal στο HUD (μόνο στα επίπεδα mission01 και mission02)

-Highscore Text: Αναφορά σε TextMeshProUGUI για το κείμενο Highscore στο HUD (μόνο στο επίπεδο endless)

Ακολουθούν οι συναρτήσεις του script:

-Start: Κάνει αρχικοποιήσεις ανάλογα το επίπεδο για αυτό και ξεκινάει δημιουργώντας μία μεταβλητή που περιέχει το όνομα της Σκηνής (curScene). Στην περίπτωση του επιπέδου endless ελέγχει αν υπάρχει Highscore στο μηχάνημα μέσω του αρχείου PlayerPrefs και στην περίπτωση που δεν υπάρχει το δημιουργεί και το θέτει 0. Αν υπάρχει παίρνει την τιμή του. Επίσης θέτει το σκορ της συνεδρίας στο 0 και το προηγούμενο επίπεδο δυσκολίας στο 0. Αν βρισκόμαστε στο επίπεδο mission01 τότε θέτει το flag hasWon σε false και την πρόοδο της αποστολής στο 0. Αν βρισκόμαστε στο επίπεδο mission02 πάλι θέτει το hasWon false και αρχικοποιεί το χρόνο της αντίστροφης μέτρησης. Σε κάθε επίπεδο αρχικοποιεί το χρόνο μεταξύ κάθε δημιουργίας εχθρικού αεροσκάφους και καλεί τη συνάρτηση SpawnEnemy.

-SpawnEnemy: Λειτουργεί με παρόμοιο τρόπο με τις συναρτήσεις SpawnEnemy του Mothership Behaviour Script με τη διαφορά ότι επιλέγει μεταξύ τριών αεροσκαφών και τα δημιουργεί σε όλο το εύρος των ορίων δημιουργίας.

-Update: Εδώ ανάλογα τη Σκηνή εκτελούμε κάποιες ενέργειες με κάθε ανανέωση frame. Στην περίπτωση της Σκηνής endless καλούμε τις συναρτήσεις DisplayScores και Difficultize. Για τη Σκηνή mission01 καλούμε την DisplayGoal και ελέγχουμε αν η πρόοδος της αποστολής μας έχει φτάσει τον στόχο και στη συγκεκριμένη περίπτωση θέτουμε το hasWon σε true. Για τη Σκηνή mission02 καλούμε τη συνάρτηση UpdateTimer και ελέγχουμε αν η αντίστροφη μέτρηση έχει φτάσει στο 0 και στη συγκεκριμένη περίπτωση θέτουμε το hasWon σε true. Τέλος, αυτή η συνάρτηση είναι επίσης υπεύθυνη, όταν ο χρόνος δημιουργίας φτάσει στο 0, να καλέσει τη συνάρτηση SpawnEnemy και να ανανεώσει τον χρόνο, με πρόσθετη λειτουργία στη Σκηνή mission02 να δημιουργεί περισσότερους εχθρούς καλώντας παραπάνω φορές τη συνάρτηση όταν φτάνουμε σε συγκεκριμένα σημεία της αντίστροφης μέτρησης.

-DisplayScores: Υπεύθυνη για το τι δείχνει το κείμενο του Score και Highscore στο HUD. Επιπροσθέτως σε περίπτωση που το Score περάσει το Highscore αλλάζει την τιμή του Highscore και ενημερώνει το αρχείο PlayerPrefs.

-Difficultize: Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για την αύξηση της δυσκολίας στο επίπεδο endless. Ανά συγκεκριμένες τιμές στο σκορ της συνεδρίας μας η δυσκολία

αυξάνεται και οι δημιουργίες εχθρικών αεροσκαφών γίνονται συχνότερες καθώς μειώνεται ο χρόνος μεταξύ κάθε νέας δημιουργίας.

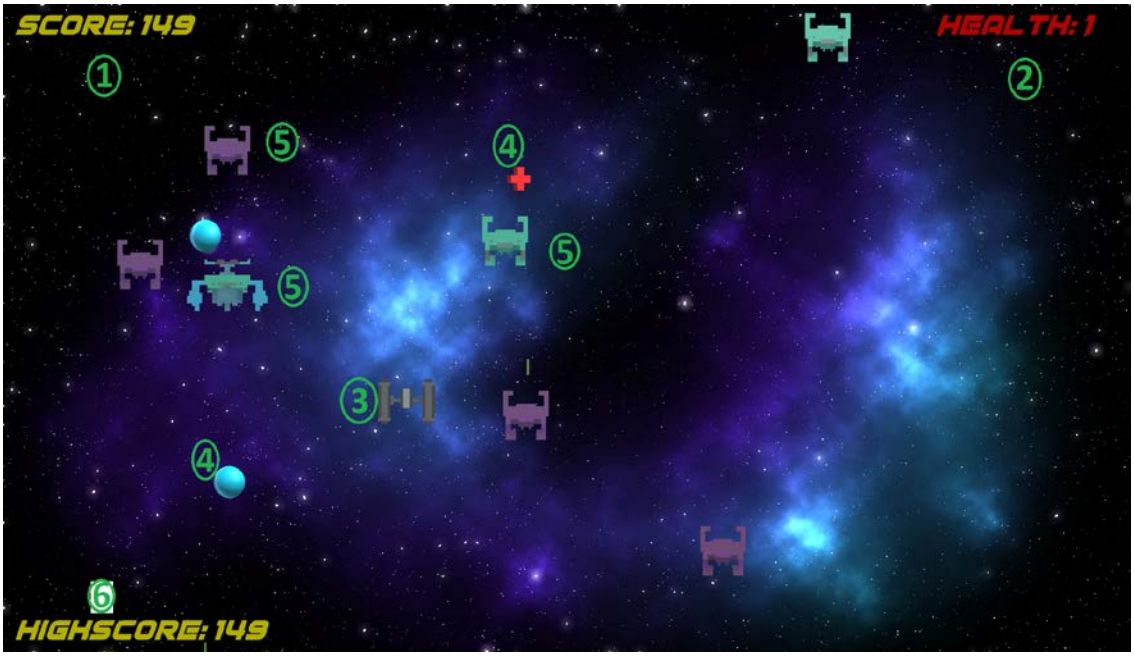
-DisplayGoal: Υπεύθυνη για το τι δείχνει το κείμενο του Goal στο HUD στο επίπεδο mission01.

-UpdateTimer: Υπεύθυνη για το τι δείχνει το κείμενο του Goal στο HUD στο επίπεδο mission02 καθώς και για την αντίστροφη μέτρηση.

Όπως μπορούμε να δούμε το GameController είναι ένα πολύ χρήσιμο και ευέλικτο script που μας βοηθάει να έχουμε καλύτερο έλεγχο των επιπέδων μας.

4.θ Παράδειγμα Συνεδρίας Παιχνιδιού

Στην Εικόνα 102 βλέπουμε ένα στιγμιότυπο του παιχνιδιού από μία συνεδρία του επιπέδου Endless.



Εικόνα 102: Στιγμιότυπο παιχνιδιού
(πηγή δημιουργός)

1) Οι πόντοι που έχει μαζέψει ο παίκτης σε αυτή τη συνεδρία

2) Οι πόντοι ζωής που απομένουν στον παίκτη

3) Το αεροσκάφος που ελέγχει ο παίκτης

4) Αναβαθμίσεις που μπορεί να συλλέξει ο παίκτης

5) Εχθρικά αεροσκάφη

6) Ο υψηλότερος αριθμός πόντων που έχει μαζέψει ο παίκτης στον συγκεκριμένο τρόπο παιχνιδιού (highscore)

ΚΕΦΑΛΑΙΟ 5 ΑΠΟΤΙΜΗΣΗ ΕΡΓΑΣΙΑΣ

5.α Συνοπτικά

Φτάνοντας στην τελευταία ενότητα αυτής της διπλωματικής ας κάνουμε μία σύνοψη της και μία αναδρομή σε αυτά που είδαμε και μάθαμε.

Στην **Ενότητα 1** κάναμε μία εισαγωγή στο αντικείμενο που θα βλέπαμε σε αυτήν την εργασία ξεκινώντας με την ιστορική αναδρομή του κόσμου των βιντεοπαιχνιδιών και αναλύοντας τους παράγοντες που μας ωθούν στο να τα παίξουμε κλείνοντας με κάποιες πληροφορίες που αφορούσαν την τεχνητή νοημοσύνη.

Στην **Ενότητα 2** είδαμε τις βασικότερες και πιο συχνές έννοιες που θα συναντήσει κάποιος στο ταξίδι της ανάπτυξης και δημιουργίας βιντεοπαιχνιδιών με την πλατφόρμα Unity. Μάθαμε τι πόσο δυνατό εργαλείο είναι αυτή η πλατφόρμα, εξοικειωθήκαμε με το interface της είδαμε τα στοιχεία και τον τρόπο που λειτουργεί και κάναμε εκτενής ανάλυση στο βασικότερο από αυτά, το GameObject και τέλος είδαμε πως μπορούμε να εξάγουμε το παιχνίδι που δημιουργήσαμε.

Στην **Ενότητα 3** είδαμε τον τρόπο που συνεργάζεται η γλώσσα C# με την Unity, παίρνωντας παράλληλα μία γεύση από τα βασικά στοιχεία του scripting που βοηθούν στην υλοποίηση του βιντεοπαιχνιδιού. Πέρα από αυτά είδαμε και κάποιες βασικές συναρτήσεις γεγονότων της Unity.

Για να φτάσουμε στην **Ενότητα 4** όπου εφαρμόζοντας στην πράξη όλο αυτό το ύλικο που ψάξαμε και μάθαμε, καταφέραμε να αναπτύξουμε το δικό μας ολοκληρωμένο και διασκεδαστικό βιντεοπαιχνίδι, ένας στόχος που είχε τεθεί εξαρχής αλλά στο ξεκίνημα της διπλωματικής έμοιαζε μακρινός. Αναφέραμε τους λόγους που μας οδήγησαν στις επιλογές μας και δείξαμε τους διαθέσιμους τρόπους παιχνιδιού, τα μοντέλα και τα στοιχεία που χρησιμοποιήσαμε, τις Σκηνές καθώς και από προγραμματιστικής πλευράς το πως λειτουργούν τα βασικότερα στοιχεία του.

5.β Μελλοντικά Σχέδια

Καθώς ο σκοπός αυτής της εργασίας ήταν να μάθουμε τα βασικά στοιχεία της πλατφόρμας Unity μέσα από την ανάπτυξη και δημιουργία ενός απλού και διασκεδαστικού παιχνιδιού, σίγουρα υπάρχει χώρος για βελτίωση και υλοποίηση περισσότερων λειτουργιών. Για αυτό και χρησιμοποιήσαμε το Onion Architecture έτσι ώστε έχοντας τις απαραίτητες λειτουργίες που κάνουν το Invaders of the Galaxy αυτό που είναι, να μπορέσουμε οποιαδήποτε άλλη έμπνευση έχουμε στο μυαλό μας να την ενσωματώσουμε στον πυρήνα του παιχνιδιού. Η συγκεκριμένη διαδικασία είναι πολύ συνηθισμένη κατά την ανάπτυξη βιντεοπαιχνιδιών καθώς ο δημιουργός μπορεί να έχει αμέτρητες ιδέες αλλά πρέπει να επιλέξει κάποιες από αυτές για να έχει μια πρώτη λειτουργική έκδοση σε λογικό χρονικό διάστημα και σε συνδυασμό με την ανατροφοδότηση από τους χρήστες να δει τι έχει προτεραιότητα να υλοποιηθεί.

Με βάση όλα αυτά, το κυριότερο σημείο που θα μπορούσαμε να βελτιώσουμε στο παιχνίδι μας είναι τα μοντέλα και τα γραφικά καθώς η ενασχόληση με αυτά δεν έγινε σε βάθος εφόσον δεν ήταν στο αντικείμενο της διπλωματικής. Είτε μέσω του Asset Store είτε δημιουργώντας τα δικά μας από την αρχή, αυτός είναι ένας τομέας που μπορεί να γίνει μεγάλη αναβάθμιση και πολύ σημαντικός καθώς τα γραφικά ενός παιχνιδιού είναι η πρώτη εικόνα που έχει ο υποψήφιος χρήστης για αυτό και βοηθούν στο να τον προσελκύσουν, και είναι και ένας από τους λόγους που οι εταιρείες της βιομηχανίας δίνουν μεγάλη βαρύτητα στα γραφικά.[54]

Θα μπορούσαμε επίσης να προσθέσουμε νέους τρόπους παιχνιδιού, όπως για παράδειγμα ένα επίπεδο στο οποίο το αεροσκάφος του παίκτη βάλλει πυρά ασταμάτητα και θα είχε σκοπό απλά τη διασκέδαση αφαιρώντας το ανταγωνιστικό κομμάτι των πόντων ή της εκπλήρωσης αποστολών. Επίσης, μπορούμε να προσθέσουμε περισσότερα είδη αναβαθμίσεων και περισσότερα είδη εχθρικών αεροσκαφών για μεγαλύτερη ποικιλία και διαφορετικές στιγμές για το χρήστη στη ροή παιχνιδιού.

Μία προσθήκη μεγαλύτερης κλίμακας θα είναι να συλλέγει ο παίκτης κάποιο είδος συναλλάγματος με κάθε εχθρικό αεροσκάφος που καταστρέφει και κάθε επίπεδο που

ολοκληρώνει το οποίο θα μπορεί να χρησιμοποιήσει για να βελτιώσει το δικό του αεροσκάφος όπως πιο δυνατά πυρά, περισσότεροι πόντοι ζωής κα. Βελτιώσεις οι οποίες θα διατηρούνται σε κάθε μηχάνημα μέσω του PlayerPrefs. Η συγκεκριμένη προσθήκη θα βοηθήσει στη διατήρηση της βάσης χρηστών καθώς θα θέλουν να συνεχίσουν να παίζουν ώστε να δημιουργήσουν το καλύτερο αεροσκάφος.

Τέλος, μία γρήγορη και εύκολη προσθήκη είναι η αύξηση των επιπέδων αποστολών αλλά και η υλοποίηση έκδοσης για Android και iOS πλατφόρμες που, όπως έχουμε δει, μας παρέχει τη δυνατότητα να γίνει αυτό χωρίς πολλές αλλαγές η Unity.

5.γ Συμπεράσματα

Μπορούμε να συμπεράνουμε ότι με την ολοκλήρωση αυτής της διπλωματικής εργασίας έχουμε αποκτήσει βασικές γνώσεις πάνω στον τομέα της δημιουργίας και ανάπτυξης βιντεοπαιχνιδιών καθώς και μία πρώτη επαφή με τη συγκεκριμένη βιομηχανία. Μέσα από την εκμάθηση των λειτουργιών της πλατφόρμας Unity και με μία μικρή εμπειρία στον προγραμματισμό καταφέραμε από το μηδέν να δημιουργήσουμε το δικό μας απλό και διασκεδαστικό βιντεοπαιχνίδι και να το διαμοιράσουμε προς χρήση. Ακολουθώντας την διαδικασία που περιγράψαμε στη διπλωματική και χρησιμοποιώντας την φαντασία μας για έμπνευση κάναμε το πρώτο αυτό βήμα προς την ενασχόληση με τον κλάδο. Είδαμε ότι πέρα από όρεξη για αυτό που κάνουμε δεν χρειάζονται ιδιαίτερες γνώσεις καθώς υπάρχει μεγάλη ποικιλία διδακτικού υλικού και έρευνας στο διαδίκτυο και συγκεκριμένα στην κοινότητα της Unity που παρείχε σημαντική βοήθεια σε όλη τη διαδικασία της υλοποίησης.[55] Η εκπόνηση αυτής της εργασίας έγινε μεθοδικά και με πολύ έρευνα όσον αφορά τον τρόπο και την διαδικασία σύλληψης της ιδέας και ανάπτυξης του βιντεοπαιχνιδιού, δείχνοντας παράλληλα την εφαρμογή στην πλατφόρμα Unity ενός συνδυασμού γνώσεων από γραφικά υπολογιστών, γλώσσα C# και τεχνητή νοημοσύνη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] “Before, Now and Then: Gaming Technology”, www.medium.com, 2022. [Online]. Available: https://medium.com/goldenrecord/before-now-and-then-gaming-technology-ab336980e6ef. [Accessed: 07- Feb- 2022].

[2] “October 1958: Physicist Invents First Video Game”, www.aps.org, 2021. [Online]. Available: https://www.aps.org/publications/apsnews/200810/physicshistory.cfm#:~:text=in%20Physics%20History-,October%201958%3A%20Physicist%20Invents%20First%20Video%20Game,Brookhaven%20National%20Laboratory%20open%20house. [Accessed: 03- Apr- 2021].

[3] “The history of Pong, the arcade classic that changed the game”, www.theverge.com, 2022. [Online]. Available: https://www.theverge.com/2012/11/30/3709758/history-of-pong. [Accessed: 07- Feb- 2022].

[4] “The Original ‘Space Invaders’ Is a Meditation on 1970s America’s Deepest Fears”, www.smithsonianmag.com, 2022. [Online]. Available: https://www.smithsonianmag.com/science-nature/original-space-invaders-icon-1970s-America-180969393. [Accessed: 07- Feb- 2022].

[5] “The 30 Defining Moments In Gaming”, www.next-gen.biz, 2021. [Online]. Available: https://web.archive.org/web/20111029232528/http://www.next-gen.biz/features/30-defining-moments-gaming.
[Accessed: 03- Apr- 2021].

[6] Everett M. R. & Judith K. L., “Silicon Valley fever: growth of high-technology culture”, New York, Basic Books, 1984, p. 263.

[7] “Nintendo Entertainment System”, en.wikipedia.org, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Nintendo_Entertainment_System. [Accessed: 07- Feb- 2022].

[8] “Welcome to the New Era: Games as Media”, www.gamesindustry.biz, 2021. [Online]. Available: https://www.gamesindustry.biz/articles/2016-10-24-welcome-to-the-new-era-games-as-media. [Accessed: 03- Apr- 2021].

[9] “Diving Deeper into eSports: Why Do People Play Video Games?”, www.admerasia.com, 2022. [Online]. Available: https://www.admerasia.com/diving-deeper-into-esports-why-do-people-play-video-games. [Accessed: 07- Feb- 2022].

[10] “15 Reasons People Play Video Games”, www.gamequitters.com, 2021. [Online]. Available: https://gamequitters.com/15-reasons-people-play-video-games/#:~:text=Everyone%20wants%20connection.,enhance%20it%20through%20online%20gaming. [Accessed: 10- Apr- 2021].

[11] “Why People Play Video Games”, www.teachthought.com, 2021. [Online]. Available: https://www.teachthought.com/learning/why-people-play-video-games. [Accessed: 10- Apr- 2021].

[12] Russell S. J. & Norvig P., “Artificial Intelligence: A Modern Approach”, New Jersey, Prentice Hall, 2009, pp. 2, 55.

[13] “Artificial neural network”, en.wikipedia.org, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed: 07- Feb- 2022].

[14] “Artificial Intelligence”, www.builtin.com, 2021. [Online]. Available: https://builtin.com/artificial-intelligence. [Accessed: 17- Apr- 2021].

[15] Kurzweil R., “The Singularity is Near”, Penguin Books, 2005.

[16] “20 Years after Deep Blue: How AI Has Advanced Since Conquering Chess”, www.scientificamerican.com, 2022. [Online]. Available: https://www.scientificamerican.com/article/20-years-after-deep-blue-how-ai-has-advanced-since-conquering-chess. [Accessed: 07- Feb- 2022].

[17] “Unity Asset Store”, assetstore.unity.com, 2022. [Online]. Available: https://assetstore.unity.com. [Accessed: 07- Feb- 2022].

[18] “Build once, reach billions”, unity.com, 2022. [Online]. Available: https://unity.com/solutions/multiplatform. [Accessed: 07- Feb- 2022].

[19] “Hearthstone”, playhearthstone.com, 2022. [Online]. Available: https://playhearthstone.com/en-us. [Accessed: 07- Feb- 2022].

[20] “Rust”, rust.facepunch.com, 2022. [Online]. Available: https://rust.facepunch.com. [Accessed: 07- Feb- 2022].

[21] “Nintendo's Super Mario Run Revenue Dashes Past $60 Million Worldwide”, sensortower.com, 2022. [Online]. Available: https://sensortower.com/blog/super-mario-run-60-million-revenue. [Accessed: 07- Feb- 2022].

[22] “Get Started with Unity Personal”, store.unity.com, 2022. [Online]. Available: https://store.unity.com/download?ref=personal. [Accessed: 07- Feb- 2022].

[23] “Unity’s interface”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/UsingTheEditor.html. [Accessed: 07- Feb- 2022].

[24] “Scene view navigation”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/SceneViewNavigation.html. [Accessed: 07- Feb- 2022].

[25] “Picking and selecting GameObjects”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/ScenePicking.html. [Accessed: 07- Feb- 2022].

[26] “Positioning GameObjects”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/PositioningGameObjects.html. [Accessed: 07- Feb- 2022].

[27] “Scene view control bar”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/ViewModes.html. [Accessed: 07- Feb- 2022].

[28] “Multi-Scene editing”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/MultiSceneEditing.html. [Accessed: 07- Feb- 2022].

[29] “GameObjects”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/GameObjects.html. [Accessed: 07- Feb- 2022].

[30] “Types of light”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/Lighting.html. [Accessed: 07- Feb- 2022].

[31] “Occlusion culling”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/OcclusionCulling.html [Accessed: 07- Feb- 2022].

[32] “Editing a Prefab in Prefab Mode”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/EditingInPrefabMode.html. [Accessed: 07- Feb- 2022].

[33] “Instance overrides”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/PrefabInstanceOverrides.html. [Accessed: 07- Feb- 2022].

[34] “Editing properties”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/EditingValueProperties.html. [Accessed: 07- Feb- 2022].

[35] “Transforms”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/class-Transform.html. [Accessed: 07- Feb- 2022].

[36] “Primitive and placeholder objects”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/PrimitiveObjects.html. [Accessed: 07- Feb- 2022].

[37] “Tags”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/Tags.html. [Accessed: 07- Feb- 2022].

[38] “Creating and Using Scripts”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. [Accessed: 07- Feb- 2022].

[39] “Variables and the Inspector”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/VariablesAndTheInspector.html. [Accessed: 07- Feb- 2022].

[40] “Instantiating Prefabs at run time”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/InstantiatingPrefabs.html. [Accessed: 07- Feb- 2022].

[41] “Procedural Generation”, www.mit.edu, 2022. [Online]. Available: http://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html. [Accessed: 12- Jan- 2022].

[42] “Order of execution for event functions”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/ExecutionOrder.html. [Accessed: 07- Feb- 2022].

[43] “Understanding Onion Architecture”, www.codeguru.com, 2022. [Online]. Available: https://www.codeguru.com/csharp/understanding-onion-architecture. [Accessed: 14- Jan- 2022].

[44] “World Video Game Hall of Fame Class of 2016 announced: Legend of Zelda, GTA3, Sonic, Space Invaders, Oregon Trail, The Sims”, www.warpzoned.com, 2022. [Online]. Available: http://www.warpzoned.com/2016/05/world-video-game-hall-of-fame-class-of-2016-announced-legend-of-zelda-gta3-sonic-space-invaders-oregon-trail-the-sims. [Accessed: 14- Jan- 2022].

[45] Bielby M., "The Complete YS Guide to Shoot 'Em Ups", Your Sinclair, July 1990 (issue 55), p. 33.

[46] “Nuclear Throne”, store.steampowered.com, 2022. [Online]. Available: https://store.steampowered.com/app/242680/Nuclear_Throne. [Accessed: 09- Feb- 2022].

[47] “Number of PC gaming users worldwide from 2008 to 2025”, www.statista.com, 2022. [Online]. Available: https://www.statista.com/statistics/420621/number-of-pc-gamers. [Accessed: 14- Jan- 2022].

[48] “Number of mobile gaming users worldwide in 2021”, www.statista.com, 2022. [Online]. Available: https://www.statista.com/statistics/512112/number-mobile-gamers-world-by-region. [Accessed: 14- Jan- 2022].

[49] “Ludum Dare 38 Game Backgrounds by Robyn Domanico”, prouser.me, 2022. [Online]. Available: https://prouser.me/ludum-dare-38-game-backgrounds-by-robyn-domanico, Robyn Domanico. [Accessed: 09- Feb- 2022].

[50] “Nebula royalty-free stock footage”, www.shutterstock.com, 2022. [Online]. Available: https://www.shutterstock.com/video/search/nebula. [Accessed: 09- Feb- 2022].

[51] “Galaxy BG 2, beauty, HD wallpaper”, www.peakpx.com, 2022. [Online]. Available: https://www.peakpx.com/en/hd-wallpaper-desktop-vviql. [Accessed: 09- Feb- 2022].

[52] “Imágenes en Movimiento del Espacio - Fondo de Pantalla de Astronomía”, www.wallpapertip.com, 2022. [Online]. Available: https://www.wallpapertip.com/es/oJmbii, Samantha Hoopes. [Accessed: 09- Feb- 2022].

[53] “Space Invaders Wallpapers”, wallpapercave.com, 2022. [Online]. Available: https://wallpapercave.com/space-invaders-wallpaper. [Accessed: 09- Feb- 2022].

[54] “How graphic design is used in video games”, www.graphicsmob.com, 2022. [Online]. Available: https://graphicsmob.com/how-graphic-design-is-used-in-video-games. [Accessed: 26- Jan- 2022].

[55] “Unity User Manual”, docs.unity3d.com, 2022. [Online]. Available: https://docs.unity3d.com/Manual/index.html. [Accessed: 27- Jan- 2022].

ΠΑΡΑΡΤΗΜΑΤΑ

A. GameController Script

```
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
    [SerializeField] MapLimits limits;
    [SerializeField] GameObject[] enemies;           //an array that has our enemies

    [SerializeField] float timer;                     //countdown for the next spawn
    float holdTimer;               //holds the starting time for the spawn countdown

    [SerializeField] int prevLevel;         //holds the previous level of difficulty
    public int score;                                        //the score of the game
    int highscore;                                      //the highscore of the game

    public int goal;                                //the level of the goal completion
    [SerializeField] int finalGoal;                //the goal that is to be achieved

    float holdCountdown;
//holds the starting countdown value so we can use it to make the level more
difficult
    [SerializeField] float countdown;
//the countdown timer for mission02, must be in seconds

    string curScene;
    public static bool hasWon;

    [SerializeField] TextMeshProUGUI scoreText;
    [SerializeField] TextMeshProUGUI goalText;
    [SerializeField] TextMeshProUGUI highscoreText;

    void Start()
    {
        curScene = SceneManager.GetActiveScene().name;
//get the name of the current scene

        if (curScene.Equals("endless"))
        {
            if (!PlayerPrefs.HasKey("highscore"))
//if there is no key named highscore in the playerprefs make it and set it to 0
            {
                highscore = 0;
                PlayerPrefs.SetInt("highscore", highscore);
            }
            highscore = PlayerPrefs.GetInt("highscore"); //get the current highscore

            score = 0;
            prevLevel = 0;
        }

        if (curScene.Equals("mission01"))
        {
```

```
            hasWon = false;
            goal = 0;
        }

        if (curScene.Equals("mission02"))
        {
            holdCountdown = countdown;
            hasWon = false;
        }

        holdTimer = timer;
        SpawnEnemy();                                   //spawn the first enemy
    }

    void Update()
    {
        if (curScene.Equals("endless"))
        {
            DisplayScores();                                //display the scores
            Difficultize();                     //make the game progressively harder
        }

        if (curScene.Equals("mission01"))
        {
            DisplayGoal();                                      //display the goal
            if (goal == finalGoal)                  //if we have completed the goal
            {
                hasWon = true;
            }
        }

        if (curScene.Equals("mission02"))
        {
            UpdateTimer();              //update countdown timer and display the time
            if (countdown <= 0)                         //if the countdown reached 0
            {
                hasWon = true;
            }
        }

        timer -= Time.deltaTime;                                            //countdown
        if (timer <= 0)
        {
            if (curScene.Equals("mission02"))
//when we are on mission02 and we reach certain checkpoints in the countdown we
start spawning more enemies
            {
                if (countdown <= holdCountdown / 3.0f)
                {
                    SpawnEnemy();
                }
                if (countdown <= holdCountdown / 8.0f)
                {
                    SpawnEnemy();
                }
            }
            SpawnEnemy();                                                   //spawn enemy
            timer = holdTimer;                                              //reset timer
        }
    }

    void DisplayScores()
```

```
    {
        scoreText.text = "Score: " + score.ToString();
//display the current score
        highscoreText.text = "Highscore: " + highscore.ToString();
//display the current highscore

        if (score > highscore)
//if the current score is higher than the current highscore
        {
            highscore = score;                                  //change the highscore
            PlayerPrefs.SetInt("highscore", highscore);
        }
    }

    void Difficultize()
    {
        int level = score / 250;
//upgrade difficulty level every 250 score (subject to change)
        if (level > prevLevel)
//sort of a flag, whenever the level changes (depending on the score) we make the
timer between spawns smaller
        {
            prevLevel = level;
            holdTimer /= (prevLevel * 0.6f + 1);
        }
    }

    void DisplayGoal()
    {
        goalText.text = "Goal: " + goal.ToString() + "/" + finalGoal.ToString();
//display the goal
    }

    void UpdateTimer()
    {
        countdown -= Time.deltaTime;
        if (countdown <= 0)
        {
            countdown = 0;
        }
        string minutes = Mathf.Floor(countdown / 60).ToString("00");
        string seconds = Mathf.Floor(countdown % 60).ToString("00");

        goalText.text = "Backup arrival: " + minutes + "." + seconds; //display time
    }

    void SpawnEnemy()
    {
        int picker = Random.Range(1, 100);
//randomize the spawn of the enemies (weighted)

        if (picker <= 50)
        {
            Instantiate(enemies[0], new Vector3(Random.Range(limits.minimumX,
limits.maximumX), Random.Range(limits.minimumY, limits.maximumY), (float) -0.3),
enemies[0].transform.rotation);
//spawn enemy1 in the space we have set in the maplimits of the gamecontroller (-0.3
to allign with the player)
        }
        else if (picker > 50 && picker <= 80)
        {
```

```
            Instantiate(enemies[1], new Vector3(Random.Range(limits.minimumX,
limits.maximumX), Random.Range(limits.minimumY, limits.maximumY), (float) -0.3),
enemies[1].transform.rotation);
//spawn enemy2 in the space we have set in the maplimits of the gamecontroller (-0.3
to allign with the player)
        }
        else
        {
            Instantiate(enemies[2], new Vector3(Random.Range(limits.minimumX,
limits.maximumX), Random.Range(limits.minimumY, limits.maximumY), (float)-0.3),
enemies[2].transform.rotation);
//spawn enemy3 in the space we have set in the maplimits of the gamecontroller (-0.3
to allign with the player)
        }
    }
}
```

Β. MapLimits Script

```
[System.Serializable]               //this is for the class to show in the inspector
public class MapLimits              //the map boundaries
{
    public float minimumX;
    public float maximumX;
    public float minimumY;
    public float maximumY;
}
```

Γ. Rotator Script

```
using UnityEngine;

public class Rotator : MonoBehaviour
{
    void Update()
    {
        transform.Rotate(new Vector3(0, 80, 0) * Time.deltaTime);
//rotates gameobjects that it is attached on the y axis
    }
}
```

Δ. VolumeEditor Script

```
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;

public class VolumeEditor : MonoBehaviour
{
    [SerializeField] AudioMixer mixer;
```

```
    [SerializeField] Slider slider;

    float sliderValue;

    void Start()
    {
        if (!PlayerPrefs.HasKey("MusicVolume"))
//set the default volume level the first time the game is played
        {
            slider.value = 0.6f;
            PlayerPrefs.SetFloat("MusicVolume", slider.value);
        }
        sliderValue = PlayerPrefs.GetFloat("MusicVolume");
        slider.value = sliderValue;
        mixer.SetFloat("MusicVol", Mathf.Log10(sliderValue) * 20);
//set the volume level
    }

    public void EditVolume()
    {
        sliderValue = slider.value;

        mixer.SetFloat("MusicVol", Mathf.Log10(sliderValue) * 20);
/*we make changes to the audio mixer depending on the changes to the slider, we pass
the slider value logarithmically and not linearly in order to have a smooth working
volume slider*/

        PlayerPrefs.SetFloat("MusicVolume", sliderValue);
//save the volume the player sets
    }
}
```

E. DeathMenu Script

```
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class DeathMenu : MonoBehaviour
{
    [SerializeField] GameObject deathMenuUI;
    [SerializeField] GameObject HUD;
    [SerializeField] TextMeshProUGUI scoreText;

    int finalScore;              //for the cool counter effect on the score display

    string curScene;

    void Start()
    {
        curScene = SceneManager.GetActiveScene().name;
        finalScore = 0;
    }

    void Update()
    {
        if (PlayerBehaviour.hasLost)
        {
```

```
            Lost();
        }
    }

    void Lost()
    {
        deathMenuUI.SetActive(true);
        HUD.SetActive(false);

        if (curScene.Equals("endless"))
        {
            if(finalScore < GameObject.FindGameObjectWithTag("Game
Controller").GetComponent<GameController>().score)  //the cool counter effect
            {
                finalScore++;
                scoreText.text = "Your Score: " + finalScore.ToString();
            }
        }
        if (curScene.Equals("predator"))
        {
            scoreText.text = "Wave Reached: ";
        }
    }

    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    public void MainMenu()
    {
        SceneManager.LoadScene("start_menu");
    }

    public void Exit()
    {
        #if (UNITY_EDITOR)
            UnityEditor.EditorApplication.isPlaying = false;
        #elif (UNITY_STANDALONE)
            Application.Quit();
        #elif (UNITY_WEBGL)
            Application.OpenURL("about:blank");
        #endif
    }
}
```

Z. MainMenu Script

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenu : MonoBehaviour
{
    [SerializeField] Button b2;
    [SerializeField] Button b3;

    int unlock;  //the mission unlocked: 1->mission02 unlocked 2->mission03 unlocked
```

```
    void Start()
    {
        if (PlayerPrefs.HasKey("unlock"))
//check the unlock level and turn on the corresponding buttons
        {
            unlock = PlayerPrefs.GetInt("unlock");
            switch (unlock)
            {
                case 1:
                    b2.interactable = true;
                    break;
                case 2:
                    b2.interactable = true;
                    b3.interactable = true;
                    break;
                default:
                    break;
            }
        }
    }

    public void PlayEndless()
    {
        SceneManager.LoadScene("endless");
    }

    public void PlayMission01()
    {
        SceneManager.LoadScene("mission01");
    }

    public void PlayMission02()
    {
        SceneManager.LoadScene("mission02");
    }

    public void PlayMission03()
    {
        SceneManager.LoadScene("mission03");
    }

    public void Exit()
    {
        #if (UNITY_EDITOR)
            UnityEditor.EditorApplication.isPlaying = false;
        #elif (UNITY_STANDALONE)
            Application.Quit();
        #elif (UNITY_WEBGL)
            Application.OpenURL("about:blank");
        #endif
    }
}
```

H. Objective Script

```
using UnityEngine;
```

```
public class Objective : MonoBehaviour
{
    void Start()
    {
        Time.timeScale = 0f;
//start each mission with objective screen and frozen time
    }

    public void Ready()
    {
        Time.timeScale = 1f;        //when we press ready start the mission
    }
}
```

Θ. PauseMenu Script

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public static bool gameIsPaused = false;
    [SerializeField] GameObject pauseMenuUI;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape) && !PlayerBehaviour.hasLost &&
!GameController.hasWon)
//the esc button is the pause button and we are not on the death screen or the win
screen
        {
            if (gameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        Time.timeScale = 1f;                        //resume time
        gameIsPaused = false;
        pauseMenuUI.SetActive(false);
    }

    void Pause()
    {
        Time.timeScale = 0f;                        //freeze time
        gameIsPaused = true;
        pauseMenuUI.SetActive(true);
    }

    public void MainMenu()
    {
```

```
        Time.timeScale = 1f;
        gameIsPaused = false;
        SceneManager.LoadScene("start_menu");
    }

    public void Exit()
    {
        #if (UNITY_EDITOR)
            UnityEditor.EditorApplication.isPlaying = false;
        #elif (UNITY_STANDALONE)
            Application.Quit();
        #elif (UNITY_WEBGL)
            Application.OpenURL("about:blank");
        #endif
    }
}
```

I. WinMenu Script

```
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class WinMenu : MonoBehaviour
{
    [SerializeField] GameObject winMenuUI;
    [SerializeField] GameObject HUD;
    [SerializeField] TextMeshProUGUI winText;

    int sayingPicker;
//pick randomly between 3 congratulatory sayings to display to the player

    string curScene;

    void Start()
    {
        curScene = SceneManager.GetActiveScene().name;
        sayingPicker = Random.Range(0, 3);
    }

    void Update()
    {
        if ((GameController.hasWon || MothershipBehaviour.wonBoss) &&
!PlayerBehaviour.hasLost)                    //we don't accidentally win when we die
        {
            if (MothershipBehaviour.wonBoss)             //if we win the boss level
            {
                Invoke("Won", 4.8f);
//call Won function with delay so we see the cool particles
            }
            else
            {
                Won();
            }
        }
    }

    void Won()
```

```
    {
        Time.timeScale = 0f;                                    //freeze time

        if (!PlayerPrefs.HasKey("unlock"))
//when we have no unlocked missions (except mission01 which is always unlocked) and
we win unlock level is 1
        {
            PlayerPrefs.SetInt("unlock", 1);
        }
        else if (curScene.Equals("mission02"))
//when we win mission02 unlock level is 2
        {
            PlayerPrefs.SetInt("unlock", 2);
        }

        winMenuUI.SetActive(true);
        HUD.SetActive(false);

        if (curScene.Equals("mission03"))
//if we win the boss level show unique message
        {
            winText.text = "You have done it";
        }
        else
        {
            switch (sayingPicker)
            {
                case 0:
                    winText.text = "That will show them";
                    break;
                case 1:
                    winText.text = "Congratulations";
                    break;
                case 2:
                    winText.text = "Flawless work";
                    break;
                default:
                    break;
            }
        }
    }

    public void NextMission()
    {
        Time.timeScale = 1f;                                            //resume time
        GameController.hasWon = false;
//we MUST set this here or else the game is stuck on 0f timescale
        MothershipBehaviour.wonBoss = false;
//we MUST set this here or else the game is stuck on 0f timescale
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
//get the next mission in the buildIndex queue
    }

    public void Restart()
    {
        Time.timeScale = 1f;                    //resume time
        GameController.hasWon = false;
//we MUST set this here or else the game is stuck on 0f timescale
        MothershipBehaviour.wonBoss = false;
//we MUST set this here or else the game is stuck on 0f timescale
        SceneManager.LoadScene(curScene);
    }
```

```
    public void MainMenu()
    {
        Time.timeScale = 1f;                    //resume time
        GameController.hasWon = false;
//we MUST set this here or else the game is stuck on 0f timescale
        MothershipBehaviour.wonBoss = false;
//we MUST set this here or else the game is stuck on 0f timescale
        SceneManager.LoadScene("start_menu");
    }

    public void Exit()
    {
        #if (UNITY_EDITOR)
            UnityEditor.EditorApplication.isPlaying = false;
        #elif (UNITY_STANDALONE)
            Application.Quit();
        #elif (UNITY_WEBGL)
            Application.OpenURL("about:blank");
        #endif
    }
}
```

K. PlayerBehaviour Script

```
using TMPro;
using UnityEngine;

public class PlayerBehaviour : MonoBehaviour
{
    [SerializeField] MapLimits limits; //the boundaries in which the player can move

    [SerializeField] GameObject playerBullet;                   //the bullet we fire
    [SerializeField] GameObject shield;
//the shield that is activated with the shield power-up

    [SerializeField] Transform posC;                          //center shooting point
    [SerializeField] Transform posL;                            //left shooting point
    [SerializeField] Transform posR;                           //right shooting point

    [SerializeField] int health;                             //the health of the player
    [SerializeField] TextMeshProUGUI healthText;
//the text that displays the health on the screen

    AudioSource audio;
    [SerializeField] AudioClip shootClip;

    [SerializeField] GameObject particle;  //the particle when the ship is destroyed

    [SerializeField] float shootTimer;                  //the time between each shot
    float holdShootTimer;                          //holds the time between each shot

    [SerializeField] float shipSpeed;
    [SerializeField] float playerBulletSpeed;

    [SerializeField] float bulletTimer;    //the time before a bullet power-up fades
    float holdBulletTimer;           //holds the time before a bullet power-up fades
    int bulletPowerUp;            //how many power-ups we have picked for the bullet
```

```
    float upDownPressed;                                    //movement on the Y-axis
    float leftRightPressed;                                 //movement on the X-axis
    bool shootingPressed;                                           //space pressed

    public static bool hasLost;

    void Start()
    {
        hasLost = false;
        audio = GetComponent<AudioSource>();
        holdShootTimer = shootTimer;
        holdBulletTimer = bulletTimer;
        bulletPowerUp = 0;
//no power-ups when we start(maybe transfer them to the next level?)
    }

    void Update()
    {
        DisplayHealth();                        //display player health on the screen
        PlayerInput();                                  //checking input in the update
        Bounds();                                           //check bounds every frame
        Death();                                  //check if we have lost every frame
        FadeBullet();       //check if we have to lose a bullet power-up every frame
    }

    private void FixedUpdate()
    {
        Movement();                                     //apply physics in FixedUpdate
        Shoot();
    }

    void Movement()
    {
        transform.Translate(new Vector3(leftRightPressed * shipSpeed *
Time.deltaTime, upDownPressed * shipSpeed * Time.deltaTime));
//apply the speed in the direction we have pressed
    }

    void Shoot()
    {
        if (shootingPressed)                            //if we pressed space to shoot
        {
            audio.PlayOneShot(shootClip);                   //play the shooting sound
            switch (bulletPowerUp)         //check how many bullet power-ups we have
            {
                case 0:
                    GameObject centerBullet = Instantiate(playerBullet,
posC.position, transform.rotation);                 //instantiate the bullet(center)
                    centerBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;                                //freeze rotation
                    centerBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;                                   //apply speed
                    break;
                case 1:
                    GameObject leftBullet = Instantiate(playerBullet, posL.position,
transform.rotation);                            //instantiate the bullets(left-right)
                    leftBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;
                    leftBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;
```

```
                    GameObject rightBullet = Instantiate(playerBullet,
posR.position, transform.rotation);
                    rightBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;
                    rightBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;
                    break;
                case 2:
                    GameObject cBullet = Instantiate(playerBullet, posC.position,
transform.rotation);                    //instantiate the bullets(center-left-right)
                    cBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;
                    cBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;
                    GameObject lBullet = Instantiate(playerBullet, posL.position,
transform.rotation);
                    lBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;
                    lBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;
                    GameObject rBullet = Instantiate(playerBullet, posR.position,
transform.rotation);
                    rBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;
                    rBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;
                    break;
                default:
                    GameObject cdBullet = Instantiate(playerBullet, posC.position,
transform.rotation);
                    cdBullet.GetComponent<Rigidbody>().velocity = Vector3.up *
playerBulletSpeed * Time.deltaTime;
                    cdBullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;
                    break;
            }
        }
    }

    void DisplayHealth()
    {
        if (health < 0) //display our health (if the health falls under 0 display 0)
        {
            healthText.text = "Health: " + 0;
        }
        else
        {
            healthText.text = "Health: " + health.ToString();
        }
    }

    void PlayerInput()
    {
        upDownPressed = Input.GetAxis("Vertical");
//give the direction on the vertical axis (0 if "w", "up", "s", or "down" are not
pressed)
        leftRightPressed = Input.GetAxis("Horizontal");
//give the direction on the horizontal axis (0 if "a", "left", "d" or "right" are
not pressed)

        shootTimer -= Time.deltaTime;
```

```
        if (Input.GetKey(KeyCode.Space) && shootTimer <=0)
//if we press space we shoot
        {
            shootingPressed = true;
            shootTimer = holdShootTimer;
        }
        else
        {
            shootingPressed = false;
        }
    }

    void Bounds()
//gameobject must stay inside the boundaries of the camera, Math.Clamp helps with
the limits
    {
        transform.position = new Vector3(Mathf.Clamp(transform.position.x,
limits.minimumX, limits.maximumX),
                                         Mathf.Clamp(transform.position.y,
limits.minimumY, limits.maximumY));
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("BulletPowerUp"))
//if we collide with a gameobject tagged as bulletpowerup
        {
            Destroy(other.gameObject);                                  //destroy it
            bulletTimer = holdBulletTimer;                  //reset the bullet timer
            if (bulletPowerUp < 2)
//if we have not maxed the bullet power-up upgrade it
            {
                bulletPowerUp++;
            }
        }
        if (other.gameObject.CompareTag("ShieldPowerUp"))
//if we collide with a gameobject tagged as shieldpowerup
        {
            Destroy(other.gameObject);                                  //destroy it
            if (!shield.active)             //if shield is deactivated, activate it
            {
                shield.SetActive(true);
            }
        }
        if (other.gameObject.CompareTag("HealthPowerUp"))
//if we collide with a gameobject tagged as healthpowerup
        {
            Destroy(other.gameObject);                                  //destroy it
            health++;                                           //increase our health
        }
        if (other.gameObject.CompareTag("Enemy"))      //if we collide with an enemy
        {
            health -= 2;                                    //we lose 2 health points
        }
        if (other.gameObject.CompareTag("Enemy Bullet"))
//if the enemy bullet hits us
        {
            health--;                                       //we lose 1 health point
        }
        if (other.gameObject.CompareTag("Mothership"))
//if we touch the mothership we lose
        {
```

```
            health = 0;
        }
    }

    void FadeBullet()
//this method is here because the bullet power-up is overpowered we needed to nerf
it
    {
        if (bulletPowerUp > 0)                //if we have an active bullet power-up
        {
            bulletTimer -= Time.deltaTime;                                  //countdown
            if (bulletTimer <= 0)                          //when countdown reaches 0
            {
                bulletPowerUp--;                             //lose a bullet power-up
                bulletTimer = holdBulletTimer;
            }
        }
    }

    void Death()
    {
        if (health<=0)      //when our health reaches 0 we destroy the ship and lose
        {
            hasLost = true;
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);
            Destroy(gameObject);
            Destroy(tempParticle, 2f);
        }
    }
}
```

Λ. MotherShipBehaviour Script

```
using TMPro;
using UnityEngine;

public class MothershipBehaviour : MonoBehaviour
{
    [SerializeField] int health;                        //the health of the mothership
    [SerializeField] TextMeshProUGUI healthText;
//the text that displays the health on the screen

    [SerializeField] GameObject particle1;                      //the death particles
    [SerializeField] GameObject particle2;
    [SerializeField] GameObject particle3;

    [SerializeField] Transform outSpawn1;                         //outer spawn point
    [SerializeField] Transform outSpawn2;
    [SerializeField] Transform inSpawn1;                          //inner spawn point
    [SerializeField] Transform inSpawn2;
    [SerializeField] Transform centerSpawn1;                     //center spawn point
    [SerializeField] Transform centerSpawn2;

    [SerializeField] GameObject[] enemies;           //an array that has our enemies

    [SerializeField] float outTimer;            //countdown for the next outer spawn
    float holdOutTimer;      //holds the starting time for the outer spawn countdown
```

```
    [SerializeField] float inTimer;             //countdown for the next inner spawn
    float holdInTimer;       //holds the starting time for the inner spawn countdown
    [SerializeField] float centerTimer;        //countdown for the next center spawn
    float holdCenterTimer;  //holds the starting time for the center spawn countdown

    public static bool wonBoss;

    bool firstStage;              //whether we are in the first stage of difficulty
    bool secondStage;            //whether we are in the second stage of difficulty

    [SerializeField] GameObject objectiveCanvas;
//the objective canvas we make active when the mothership appears
    bool lvlStart;   //flag to make the cool animation in the beggining of the level

    [SerializeField] float speed;
    Rigidbody rb;

    void Start()
    {
        GameController.hasWon = false;
//MUST BE HERE IN MOTHERSHIP LEVELS (and levels we do not have GameController in
general to avoid always-winscreen bug)

        lvlStart = true;
        wonBoss = false;
        firstStage = false;
        secondStage = false;
        holdOutTimer = outTimer;
        holdInTimer = inTimer;
        holdCenterTimer = centerTimer;

        rb = GetComponent<Rigidbody>();
        rb.velocity = new Vector3(0, -speed * 0.015f, 0);              //give a
starting speed to the mothership
    }

    void Update()
    {
        if (transform.position.y <= 6.15 && lvlStart)
//check if we reached the y-coordinates we want the first time
        {
            lvlStart = false;                                  //do not do this again
            rb.velocity = Vector3.zero;                         //stop the mothership
        }

        DisplayHealth();
        Death();
        Spawner();
    }

    private void Spawner()
//does the spawn countdowns and depending on the mothership's health reduces the
time between each spawn
    {
        centerTimer -= Time.deltaTime;
        inTimer -= Time.deltaTime;
        outTimer -= Time.deltaTime;

        if (health <= 65 && !firstStage)
        {
            firstStage = true;
            holdCenterTimer /= 1.5f;
```

```
            holdInTimer /= 1.5f;
            holdOutTimer /= 1.5f;
        }
        if (health <= 20 && !secondStage)
        {
            secondStage = true;
            holdCenterTimer /= 2;
            holdInTimer /= 2;
            holdOutTimer /= 2;
        }

        if (centerTimer <= 0)
        {
            SpawnCenterEnemy();
            centerTimer = holdCenterTimer;
        }
        if (inTimer <= 0)
        {
            SpawnInEnemy();
            inTimer = holdInTimer;
        }
        if (outTimer <= 0)
        {
            SpawnOutEnemy();
            outTimer = holdOutTimer;
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Shield"))
//if the mothership hits the shield it loses 1 health point and the shield is
deactivated
        {
            health--;
            other.gameObject.SetActive(false);
        }
    }

    public void CollidedWithBullet()
    {
        health--;
    }

    void DisplayHealth()
    {
        if (health < 0)
//display mothership's health (if the health falls under 0 display 0)
        {
            healthText.text = "Mothership Health: " + 0;
        }
        else
        {
            healthText.text = "Mothership Health: " + health.ToString();
        }
    }


    void Death()
    {
        if (health <= 0)                    //when the mothership's health reaches 0
        {
```

```
            GameObject[] stillAlive = GameObject.FindGameObjectsWithTag("Enemy");
//find all the enemies that are still alive and set their health to 0
            foreach (GameObject go in stillAlive)
            {
                go.GetComponent<EnemyBehaviour>().SetHealth();
            }

            GameObject outerParticle1 = Instantiate(particle3, transform.position +
new Vector3(-5.5f, -3, 0), transform.rotation);               //spawn cool particles
            GameObject outerParticle2 = Instantiate(particle3, transform.position +
new Vector3(5.5f, -3, 0), transform.rotation);
            GameObject innerParticle1 = Instantiate(particle2, transform.position +
new Vector3(-2.5f, -3, 0), transform.rotation);
            GameObject innerParticle2 = Instantiate(particle2, transform.position +
new Vector3(2.5f, -3, 0), transform.rotation);
            GameObject centerParticle = Instantiate(particle1, transform.position +
new Vector3(0, -3, 0), transform.rotation);

            Destroy(gameObject);

            Destroy(outerParticle1, 2.0f);
            Destroy(outerParticle2, 2.0f);
            Destroy(innerParticle1, 4.5f);
            Destroy(innerParticle2, 4.5f);
            Destroy(centerParticle, 7.0f);

            wonBoss = true;                                          //we win the level
        }
    }

    void SpawnOutEnemy()   //outer spawn method: 75% spawns enemy0 25% spawns enemy1
    {
        int picker = Random.Range(1, 100);

        if (picker <= 75)
        {
            Instantiate(enemies[0], new Vector3(outSpawn1.position.x +
Random.Range(-1.0f, 1.0f), outSpawn1.position.y,  outSpawn1.position.z),
enemies[0].transform.rotation);
            Instantiate(enemies[0], new Vector3(outSpawn2.position.x +
Random.Range(-1.0f, 1.0f), outSpawn2.position.y, outSpawn2.position.z),
enemies[0].transform.rotation);
        }
        else if (picker > 75)
        {
            Instantiate(enemies[1], new Vector3(outSpawn1.position.x +
Random.Range(-1.0f, 1.0f), outSpawn1.position.y, outSpawn1.position.z),
enemies[1].transform.rotation);
            Instantiate(enemies[1], new Vector3(outSpawn2.position.x +
Random.Range(-1.0f, 1.0f), outSpawn2.position.y, outSpawn2.position.z),
enemies[1].transform.rotation);
        }
    }

    void SpawnInEnemy()    //inner spawn method: 10% spawns enemy0 90% spawns enemy1
    {
        int picker = Random.Range(1, 100);

        if (picker <= 10)
        {
```

```
            Instantiate(enemies[0], new Vector3(inSpawn1.position.x + Random.Range(-
3.5f, 3.5f), inSpawn1.position.y, inSpawn1.position.z),
enemies[0].transform.rotation);
            Instantiate(enemies[0], new Vector3(inSpawn2.position.x + Random.Range(-
3.5f, 3.5f), inSpawn2.position.y, inSpawn2.position.z),
enemies[0].transform.rotation);
        }
        else if (picker > 90)
        {
            Instantiate(enemies[1], new Vector3(inSpawn1.position.x + Random.Range(-
3.5f, 3.5f), inSpawn1.position.y, inSpawn1.position.z),
enemies[1].transform.rotation);
            Instantiate(enemies[1], new Vector3(inSpawn2.position.x + Random.Range(-
3.5f, 3.5f), inSpawn2.position.y, inSpawn2.position.z),
enemies[1].transform.rotation);
        }
    }

    void SpawnCenterEnemy()
//center spawn method: spawns both, 50% which side spawns what enemy
    {
        int picker = Random.Range(1, 100);

        if (picker <= 50)
        {
            Instantiate(enemies[0], new Vector3(centerSpawn1.position.x +
Random.Range(-3.5f, 3.5f), centerSpawn1.position.y, centerSpawn1.position.z),
enemies[0].transform.rotation);
            Instantiate(enemies[1], new Vector3(centerSpawn2.position.x +
Random.Range(-3.5f, 3.5f), centerSpawn2.position.y, centerSpawn2.position.z),
enemies[1].transform.rotation);
        }
        else if (picker > 50)
        {
            Instantiate(enemies[1], new Vector3(centerSpawn1.position.x +
Random.Range(-3.5f, 3.5f), centerSpawn1.position.y, centerSpawn1.position.z),
enemies[1].transform.rotation);
            Instantiate(enemies[0], new Vector3(centerSpawn2.position.x +
Random.Range(-3.5f, 3.5f), centerSpawn2.position.y, centerSpawn2.position.z),
enemies[0].transform.rotation);
        }
    }
}
```

M. EnemyBullet Script

```
using UnityEngine;

public class EnemyBullet : MonoBehaviour
{
    [SerializeField] float time;                    //how long the bullet stays alive
    [SerializeField] float particleTime;
    [SerializeField] GameObject particle;

    private void Update()
    {
        time -= Time.deltaTime;
        if (time <= 0)                                      //when the time gets to 0
```

```
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                        //spawn particle effect
            Destroy(gameObject);                                   //destroy the bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
    }

    private void OnTriggerEnter(Collider collision)
    {
        if (collision.gameObject.CompareTag("Player"))   //when this hits the player
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                        //spawn particle effect
            Destroy(gameObject);                                          //destroy bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
        if (collision.gameObject.CompareTag("Friendly Bullet"))
//when this hits a friendly bullet
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                        //spawn particle effect
            Destroy(gameObject);                                          //destroy bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
        if (collision.gameObject.CompareTag("Shield"))
//when this hits the player shield
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                        //spawn particle effect
            Destroy(gameObject);                                          //destroy bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
            collision.gameObject.SetActive(false);        //deactivate player shield
        }
    }
}
```

N. EnemyBehaviour Script

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class EnemyBehaviour : MonoBehaviour
{
    [SerializeField] float speed;
    [SerializeField] float dirTimer;       //the countdown timer to change direction
    [SerializeField] float shootTimer;       //the countdown timer to shoot a bullet
    [SerializeField] int health;                              //the health of the enemy
    [SerializeField] int chance;
//the power-up dropping chance depending on the enemy
    [SerializeField] int scoreWeight;
//the score each enemy gives depending on its type
    [SerializeField] float enemyBulletSpeed;
```

```
    [SerializeField] bool canShoot;
//whether the enemy this is attached to can shoot
    [SerializeField] bool changeDir;     //the switch for the direction (right/left)

    [SerializeField] GameObject healthPU;
    [SerializeField] GameObject bulletPU;
    [SerializeField] GameObject shieldPU;
    [SerializeField] GameObject enemyBullet;            //the bullet the enemy fires
    [SerializeField] GameObject particle;  //the particle when an enemy is destroyed

    [SerializeField] Transform shootingPos;  //the position the bullet is fired from
    [SerializeField] MapLimits limits;

    float holdShootTimer;      //holds the seconds that need to pass before shooting
    float holdDirTimer;
//holds the seconds that need to pass before changing direction

    string curScene;

    Rigidbody rb;

    void Start()
    {
        curScene = SceneManager.GetActiveScene().name;
//get the name of the current scene
        rb = GetComponent<Rigidbody>();        //get the rigidbody of the gameobject
        holdDirTimer = dirTimer;
//store the starting seconds before the direction change
        holdShootTimer = shootTimer;
//store the starting seconds before the enemy starts shooting
    }

    void Update()
    {
        Change();
        Death();
        Shooting();
    }

    private void FixedUpdate()
    {
        Movement();
    }

    void Movement()
    {
        if (transform.position.x >= limits.maximumX || transform.position.x <=
limits.minimumX)
//if the enemies have reached the boundaries on the x-axis change their direction
        {
            changeDir = !changeDir;
//this happens in the fixed update to avoid the always changing direction bug on the
update
        }

        if (changeDir)
//depending on the switch we apply velocity right or left while going down
        {
            rb.velocity = new Vector3(speed * Time.deltaTime, -speed *
Time.deltaTime);
        }
        else
```

```
        {
            rb.velocity = new Vector3(-speed * Time.deltaTime, -speed *
Time.deltaTime);
        }
    }

    void Change()
    {
        dirTimer -= Time.deltaTime;                                     //countdown
        if(dirTimer<=0)                                     //when countdown reaches 0
        {
            changeDir = !changeDir;                                 //change direction
            dirTimer = holdDirTimer;                                    //reset timer
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Friendly Bullet"))
//if a bullet hits the enemy they lose 1 health point
        {
            if (curScene.Equals("endless"))
            {
                GameObject.FindGameObjectWithTag("Game
Controller").GetComponent<GameController>().score += scoreWeight;
//whenever our bullet hits an enemy we get score points
            }
            health--;
        }
        if (other.gameObject.CompareTag("Player"))
//if the enemy hits the player the enemy is destroyed
        {
            health = 0;
        }
        if (other.gameObject.CompareTag("Shield"))
//if the enemy hits the shield the enemy is destroyed and the shield is deactivated
        {
            health = 0;
            other.gameObject.SetActive(false);
        }
    }

    void Shooting()
    {
        if (canShoot)                                       //if the enemy can shoot
        {
            shootTimer -= Time.deltaTime;                                   //countdown
            if (shootTimer <= 0)                            //when countdown reaches 0
            {
                GameObject bullet = Instantiate(enemyBullet, shootingPos.position,
shootingPos.rotation);                                  //instantiate the enemy bullet
                bullet.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeRotation;                                //freeze rotation
                bullet.GetComponent<Rigidbody>().velocity = Vector3.down *
enemyBulletSpeed * Time.deltaTime;                                      //apply speed
                shootTimer = holdShootTimer;                                //reset timer
            }
        }
    }

    void Death()
    {
```

```
        if (transform.position.y <= limits.minimumY)
//if the enemy leaves the playing area (only downwards allowed) destroy it
        {
            Destroy(gameObject);
        }
        if (health<=0)
//if health has reached 0 destroy the enemy and spawn particle effect
        {
            DropPowerUp();                //check if this enemy will drop a power-up

            if (curScene.Equals("endless"))                    //on the endless level
            {
                GameObject.FindGameObjectWithTag("Game
Controller").GetComponent<GameController>().score += scoreWeight * 3;
//whenever we destroy an enemy we get score points
            }
            if (curScene.Equals("mission01"))                  //on the mission level
            {
                if (scoreWeight == 3)
//whenever we kill the biggest enemy (this is the goal on mission01)
                {
                    GameObject.FindGameObjectWithTag("Game
Controller").GetComponent<GameController>().goal++;             //increment goal by 1
                }
            }

            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);
            Destroy(gameObject);
            Destroy(tempParticle, 2f);
        }
    }

    void DropPowerUp()
    {
        int picker = Random.Range(1, chance);
//random chance to spawn a power-up upon death, chance goes shield>health>bullet
        if (picker <= 15)
        {
            Instantiate(shieldPU, transform.position + new Vector3(0, 0,
(float)0.3), shieldPU.transform.rotation);
        }
        else if (picker > 15 && picker <= 25)
        {
            Instantiate(healthPU, transform.position + new Vector3(0, 0,
(float)0.3), healthPU.transform.rotation);
        }
        else if (picker > 25 && picker <= 30)
        {
            Instantiate(bulletPU, transform.position + new Vector3(0, 0,
(float)0.3), bulletPU.transform.rotation);
        }
    }

    public void SetHealth()
//used when we destroy the mothership to destroy all the enemies
    {
        health = 0;
    }
}
```

Ξ. BulletBehaviour Script

```
using UnityEngine;

public class BulletBehaviour : MonoBehaviour
{
    [SerializeField] float time;
    [SerializeField] float particleTime;
    [SerializeField] GameObject particle;

    private void Update()
    {
        time -= Time.deltaTime;
        if (time <= 0)                                          //when the time gets to 0
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                            //spawn particle effect
            Destroy(gameObject);                                   //destroy the bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
    }

    private void OnTriggerEnter(Collider collision)
    {
        if (collision.gameObject.CompareTag("Enemy"))      //when this hits an enemy
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                            //spawn particle effect
            Destroy(gameObject);                                          //destroy bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
        if (collision.gameObject.CompareTag("Mothership"))
//when this hits the mothership
        {

collision.gameObject.GetComponent<MothershipBehaviour>().CollidedWithBullet();
//call the method to lower the health of the mothership (this is here because we
have multiple colliders on mothership)
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                            //spawn particle effect
            Destroy(gameObject);                                          //destroy bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
        if (collision.gameObject.CompareTag("Enemy Bullet"))
//when this hits an enemy bullet
        {
            GameObject tempParticle = Instantiate(particle, transform.position,
transform.rotation);                                            //spawn particle effect
            Destroy(gameObject);                                          //destroy bullet
            Destroy(tempParticle, particleTime);
//destroy particle after particleTime
        }
    }
}
```