****

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ**

**Σχολή Κοινωνικών και Ανθρωπιστικών Επιστημών**

**Τμήμα Επικοινωνίας και Ψηφιακών Μέσων**

**Μεταπτυχιακός Κύκλος Σπουδών**

**Ανάπτυξη Ψηφιακών Παιχνιδιών και**

**Πολυμεσικών Εφαρμογών**

**ΤΙΤΛΟΣ ΜΑΘΗΜΑΤΟΣ:**

**Αλγοριθμική Σκέψη**

**Διδάσκοντες: Νίκος Πλόσκας, Μηνάς Δασυγένης**

**«Υλοποίηση παιχνιδιού με Python»**

**ΚΑΡΑΠΟΣΤΟΛΗΣ ΚΥΡΙΑΖΗΣ**

**Α.Μ.: mpw00029**

**ΜΠΟΥΤΣΗ ΑΓΝΗ**

**Α.Μ.: mpw00035**

**Εργασία Εξαμήνου**

ΚΑΣΤΟΡΙΑ 2024

Περιεχόμενα

[**Πίνακας Εικόνων** 1](#_Toc158854542)

[**Σενάριο παιχνιδιού: Galaxy Shooter** 3](#_Toc158854543)

[**Η ιδέα πίσω από το Galaxy Shooter** 4](#_Toc158854544)

[**Σκοπός της παρούσας αναφοράς** 5](#_Toc158854545)

[**Γενική επισκόπηση του κώδικα και των λειτουργιών του Galaxy Shooter** 6](#_Toc158854546)

[**Σχεδιασμός του Galaxy Shooter – Game Mechanics** 7](#_Toc158854547)

[**Σχεδιασμός των Επιπέδων του Galaxy Shooter** 13](#_Toc158854548)

[**Σχεδιασμός των χαρακτήρων και των εχθρών του Galaxy Shooter** 18](#_Toc158854549)

[**Υλοποίηση του Galaxy Shooter** 21](#_Toc158854550)

[**Αλγόριθμοι και δομή του Κώδικα** 23](#_Toc158854551)

[**Λειτουργικότητα του παιχνιδιού** 28](#_Toc158854552)

[**Διεπαφή χρήστη και εμπειρία χρήστη** 34](#_Toc158854553)

[**Παράρτημα** 36](#_Toc158854554)

**Πίνακας Εικόνων**

[Εικόνα 1 Μηχανισμός πυροβολισμού (κώδικας Python) 8](#_Toc158854555)

[Εικόνα 2 Γραφική απεικόνιση απλής βολής (Α) και βολής μετά το power-up (B) 9](#_Toc158854556)

[Εικόνα 3 Μηχανισμός κίνησης των παικτών και αποφυγής (κώδικας Python) 10](#_Toc158854557)

[Εικόνα 4 Μηχανισμός PowerUp – Δημιουργία της κλάσης (κώδικας Python) 11](#_Toc158854558)

[Εικόνα 5 Μηχανισμός PowerUp (κώδικας Python) 11](#_Toc158854559)

[Εικόνα 6 Στιγμιότυπο του gameplay με γραφική απεικόνιση του PowerUp 12](#_Toc158854560)

[Εικόνα 7 Δημιουργία των instances των παικτών(κώδικας Python) 13](#_Toc158854561)

[Εικόνα 8 Διαχείριση και κλήση του πυροβολισμού από τον παίκτη (κώδικας Python) 14](#_Toc158854562)

[Εικόνα 9 Δημιουργία των instances των εχθρών (κώδικας Python) 14](#_Toc158854563)

[Εικόνα 10 Διαχείριση και κλήση των εχθρών (κώδικας Python) 15](#_Toc158854564)

[Εικόνα 11 Καθορισμός του επιπέδου δυσκολίας (κώδικας Python) 15](#_Toc158854565)

[Εικόνα 12 Κώδικας για την ενεργοποίηση του power-up 16](#_Toc158854566)

[Εικόνα 13 Συγκρούσεις με τους εχθρούς και Αύξηση του σκορ (κώδικας Python) 17](#_Toc158854567)

[Εικόνα 14 Οπτική αναπαράσταση παίκτη 18](#_Toc158854568)

[Εικόνα 15 Μηχανική του gameplay 18](#_Toc158854569)

[Εικόνα 16 Απόκτηση του Power-up μέσω σύγκρουσης με αυτό 18](#_Toc158854570)

[Εικόνα 17 Τύποι εχθρών 19](#_Toc158854571)

[Εικόνα 18 Κλάση και συμπεριφορά βασικού εχθρού 19](#_Toc158854572)

[Εικόνα 19 Κλάση και συμπεριφορά δεύτερου εχθρού 20](#_Toc158854573)

[Εικόνα 20 Σκορ από την εξουδετέρωση του βασικού εχθρού 20](#_Toc158854574)

[Εικόνα 21 Σκορ από την εξουδετέρωση του δεύτερου εχθρού 21](#_Toc158854575)

[Εικόνα 22 Συγκρούσεις εχθρών και σφαιρών 23](#_Toc158854576)

[Εικόνα 23 Συγκρούσεις παικτών και βασικών εχθρών 24](#_Toc158854577)

[Εικόνα 24 Συγκρούσεις παικτών και δεύτερων εχθρών 24](#_Toc158854578)

[Εικόνα 25 Κίνηση των παικτών (διαστημοπλοίων) 25](#_Toc158854579)

[Εικόνα 26 Κίνηση του βασικού εχθρού 26](#_Toc158854580)

[Εικόνα 27 Κίνηση του δεύτερου εχθρού 27](#_Toc158854581)

[Εικόνα 28 Συνάρτηση εμφάνισης του σκορ 27](#_Toc158854582)

[Εικόνα 29 Αύξηση του σκορ κατά 10 βαθμούς για την εξουδετέρωση του βασικού εχθρού και κατά 20 βαθμούς για την εξουδετέρωση του δεύτερου εχθρού 28](#_Toc158854583)

[Εικόνα 30 Αρχική οθόνη του παιχνιδιού με τα κουμπιά επιλογών 29](#_Toc158854584)

[Εικόνα 31 Κώδικας για τη δημιουργία των αντικειμένων των παικτών 30](#_Toc158854585)

[Εικόνα 32 Επιλογή επιπέδου δυσκολίας 31](#_Toc158854586)

[Εικόνα 33 Ο κώδικας της επιλογής του επιπέδο δυσκολίας 32](#_Toc158854587)

[Εικόνα 34 Εφαρμογή του επιπέδου δυσκολίας 32](#_Toc158854588)

[Εικόνα 35 Μεταβλητή enemy\_speed για τον βασικό εχθρό 33](#_Toc158854589)

[Εικόνα 36 Μεταβλητή enemy\_speed για τον δεύτερο εχθρό 33](#_Toc158854590)

[Εικόνα 37 Gameplay interface σε single player mode (A) και σε multiplayer (B) 35](#_Toc158854591)

**Σενάριο παιχνιδιού: Galaxy Shooter**

Στις αχανείς, αχαρτογράφητες εκτάσεις του σύμπαντος, όπου η υφή του χώρου και του χρόνου εκτείνεται στο άγνωστο, υπάρχει ένα γαλαξιακό πεδίο μάχης γεμάτο κινδύνους και περιπέτειες. Το “Galaxy Shooter” προσκαλεί τους παίκτες σε ένα σύμπαν γεμάτο κινδύνους, όπου το θάρρος συναντά το χάος και όπου ένας μοναχικός ήρωας ξεκινά μια τολμηρή αναζήτηση για να σώσει τον γαλαξία από μια επικείμενη καταστροφή!

**Η αρχή**

Η ηρεμία του γαλαξία έχει διαταραχθεί. Μια τρομερή συμμαχία εξωγήινων φυλών, γνωστή ως Συνομοσπονδία του Σκοτεινού Νεφελώματος, έχει εξαπολύσει ανελέητη επίθεση στους ειρηνικούς πλανήτες. Αυτές οι κακόβουλες δυνάμεις, καθοδηγούμενες από την επιθυμία για παγκόσμια κυριαρχία, αναπτύσσουν στόλους εχθρικών πλοίων και τερατώδεις οντότητες, αφήνοντας στο πέρασμά τους ένα ίχνος καταστροφής. Το οικοδόμημα της γαλαξιακής κοινωνίας βρίσκεται στα πρόθυρα της κατάρρευσης και κάθε ελπίδα μοιάζει χαμένη.

**Ο ήρωας**

Μέσα από την απελπισία αναδύεται ένας φάρος ελπίδας. Ο παίκτης αναλαμβάνει το ρόλο ενός ειδικευμένου πιλότου της Γαλαξιακής Αντίστασης, ενός συνασπισμού φυλών αποφασισμένων να αποκρούσουν την εισβολή. Αυτός ο ήρωας, που πιλοτάρει ένα διαστημόπλοιο εφοδιασμένο με αρχαία τεχνολογία και ισχυρά όπλα, αντιπροσωπεύει την τελευταία αντίσταση ενάντια στο σκοτάδι που έρχεται. Με την ανθρωπότητα και τις συμμαχικές εξωγήινες φυλές στα πρόθυρα της εξαφάνισης, το διακύβευμα είναι τεράστιο.

**Η αποστολή**

Το ταξίδι ξεκινά στις απώτερες περιοχές του γαλαξία, όπου ο παίκτης πρέπει να πλοηγηθεί μέσα σε πεδία αστεροειδών, να εμπλακεί σε αερομαχίες με εχθρικά αναγνωριστικά και να διαλύσει τα φυλάκια της Συνομοσπονδίας. Καθώς η περιπέτεια εξελίσσεται, η ένταση των μαχών κλιμακώνεται, οδηγώντας σε αντιπαραθέσεις με κολοσσιαίες εχθρικές ναυαρχίδες και συναντήσεις με τους σκοτεινούς στρατηγούς της Συνομοσπονδίας.

Κατά τη διάρκεια της αναζήτησής του, ο παίκτης ανακαλύπτει αρχαία τεχνουργήματα και power-ups, απομεινάρια ενός χαμένου πολιτισμού με προηγμένη τεχνολογία, που του χαρίζουν όπλα ικανά να αποδεκατίζουν εχθρικούς στόλους. Ωστόσο, με τη μεγάλη δύναμη έρχονται και μεγάλες προκλήσεις: η Συνομοσπονδία, στέλνει τις πιο τρομερές μονάδες της, έτοιμη να καταστείλει την επανάσταση.

**Η μάχη**

Ο ήρωας μάχεται ασταμάτητα με τους εχθρικού στόλους, στην καρδιά του Σκοτεινού Νεφελώματος, ανάμεσα στις στροβιλιζόμενες δίνες σκοτεινής ενέργειας, σε έναν αέναο αγώνα που δοκιμάζει όλες τις δεξιότητες και την ανθεκτικότητά του. Η νίκη είναι απαραίτητη. Όχι μόνο για την ελευθερία του γαλαξία, αλλά και για την αποκάλυψη των μυστηρίων των αρχαίων πολιτισμών, που, με τις μαγικές τεχνολογίες τους, άλλαξαν τη ροή του χρόνου. Το “Galaxy Shooter” είναι κάτι περισσότερο από ένα απλό παιχνίδι, είναι ένα έπος στο οποίο το φως μάχεται με το σκοτάδι, η ανδρεία είναι αντιμέτωπη με το χάος. Όλα εξαρτώνται από την ανθεκτικότητα του παίκτη, του ήρωα που με το θάρρος του πυροδοτεί τις φλόγες της εξέγερσης σε ολόκληρο τον γαλαξία. Οι παίκτες δεν συμμετέχουν απλώς σε μάχες, αλλά γράφουν την ιστορία ενός σύμπαντος όπου η γενναιότητα και η αποφασιστικότητα μπορούν να αλλάξουν τη μοίρα πολλών. Το ταξίδι περιμένει - είστε έτοιμοι να συμμετάσχετε στη μάχη και να εξασφαλίσετε τη θέση σας ανάμεσα στα αστέρια;

**Η ιδέα πίσω από το Galaxy Shooter**

Τις προηγούμενες δεκαετίες ο κόσμος φαίνεται ότι φωτιζόταν από τις οθόνες των ηλεκτρονικών παιχνιδιών και τη λάμψη των επών επιστημονικής φαντασίας. Ο «Πόλεμος των Άστρων» και το “Star Trek” δεν ήταν απλώς περιπέτειες που εκτυλίσσονταν σε μακρινούς γαλαξίες - ήταν πύλες σε κόσμους όπου το αδύνατο ήταν εφικτό. Οι ήρωες και οι «κακοί», η συντροφικότητα μεταξύ διαφορετικών όντων και ο χορός του πεπρωμένου και των επιλογών είναι, ακόμη και σήμερα, χαρακτηριστικά του Sci-Fi και του gaming κόσμου.

Υπάρχει κάτι στην πάλη μεταξύ φωτός και σκότους, στο πλούσιο μωσαϊκό των χαρακτήρων και στις φιλοσοφικές αναζητήσεις στον «Πόλεμο των Άστρων» και στο “Star Trek” που πάντα μου έμενε. Δεν ήταν απλώς ιστορίες- ήταν μαθήματα γενναιότητας, αξίας της διαφορετικότητας και δύναμης της πίστης. Αναπτύσσοντας αυτό το παιχνίδι - το “Galaxy Shooter” – θελήσαμε να συλλάβουμε αυτήν την μαγεία, δημιουργώντας ένα απλό παιχνίδι που θα έμοιαζε με φόρο τιμής προς αυτά τα σύμπαντα, ενσωματώνοντας, παράλληλα στοιχεία από τα ιστορικά πλέον Arcade παιχνίδια, όπως το “Space Invaders”.

Το “Galaxy Shooter” είναι ένα απλό παιχνίδι, στο οποίο, οι παίκτες κατευθύνουν ένα διαστημόπλοιο μέσα από έναν καταιγισμό εχθρών και εμποδίων, αναλαμβάνοντας το ρόλο ενός μοναχικού πολεμιστή απέναντι σε μια κοσμική επίθεση, με μια ιστορία αντοχής, στρατηγικής και επιβίωσης, με φόντο ένα όμορφα σχεδιασμένο σύμπαν γεμάτο κινδύνους και θαύματα.

**Σκοπός της παρούσας αναφοράς**

Η παρούσα αναφορά αποτελεί μια περιγραφή της διαδικασίας ανάπτυξης, των αποφάσεων σχεδιασμού και των λειτουργιών που χαρακτηρίζουν το “Galaxy Shooter”. Ο σκοπός της είναι να περιγράψει τις διαδικασίες για τη δημιουργία του παιχνιδιού και να καταγράψει τις τεχνικές και δημιουργικές γνώσεις που αποκτήθηκαν κατά την ανάπτυξή του, μέσα από την καταγραφή των προσεγγίσεων κατά τη διάρκεια της υλοποίησης του “Galacy Shooter”.

Στις επόμενες ενότητες, θα εξετάσουμε τις σχεδιαστικές εκτιμήσεις που επηρέασαν τους μηχανισμούς, την αισθητική και την αφήγηση του παιχνιδιού. Στη συνέχεια, θα εξετάσουμε την τεχνική υλοποίησης, επισημαίνοντας τις πρακτικές κωδικοποίησης, τους αλγόριθμους και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του παιχνιδιού. Επίσης, ασχολούμαστε με τη λειτουργικότητα του παιχνιδιού, συμπεριλαμβανομένων των χαρακτηριστικών του παιχνιδιού, του σχεδιασμού της διεπαφής χρήστη και της ενσωμάτωσης μηχανισμών ανάδρασης.

**Γενική επισκόπηση του κώδικα και των λειτουργιών του Galaxy Shooter**

Πρόκειται για μια πλήρη υλοποίηση ενός arcade παιχνιδιού σκοποβολής με το όνομα “Galaxy Shooter”, που έγινε με τη χρήση της βιβλιοθήκης Pygame για την Python. Στη συνέχεια δίνεται μια γενική επισκόπηση της λογικής και των λειτουργιών του παιχνιδιού, ώστε να γίνει κατανοητός ο τρόπος ανάπτυξης και λειτουργίας του, ενώ, στα επόμενα κεφάλαια εξετάζονται χωριστά τα διαφορετικά τμήματα του κώδικα που αναπτύχθηκε:

**Αρχικοποίηση και Ρυθμίσεις**

Αρχικά, ο κώδικας ξεκινάει με την αρχικοποίηση της βιβλιοθήκης Pygame. Παράλληλα, ορίζονται βασικές σταθερές, όπως οι διαστάσεις του παραθύρου και οι ταχύτητες των αντικειμένων.

Στη συνέχεια, ορίζεται η φόρτωση εικόνων για το background, τους εχθρούς, τις σφαίρες και τα power-ups, καθώς και ηχητικών εφέ και μουσικής φόντου.

**Ορισμός Κλάσεων**

Δημιουργούνται κλάσεις για τον παίκτη (Player), τους εχθρούς (Enemy, SecondEnemy), τις σφαίρες (Bullet, TripleLaserBullet) και τα power-ups (PowerUp).

Κάθε κλάση έχει μεθόδους για την αρχικοποίηση, ενημέρωση της κατάστασης και σχεδίαση των αντικειμένων στο παιχνίδι.

**Λειτουργία παιχνιδιού**

Στη συνέχεια ορίζονται οι λειτουργίες του παιχνιδιού (functions) με τη χρήση της def (Κύριο μενού, εμφάνιση σκορ, οθόνη επιλογών, οθόνη HighScore).

**Λογική Παιχνιδιού**

Η βασική λογική του παιχνιδιού εκτελείται μέσα σε μια επαναληπτική διαδικασία (game loop) όπου ελέγχονται συμβάντα (όπως πατήματα πλήκτρων), ενημερώνονται οι θέσεις των αντικειμένων και σχεδιάζεται η νέα κατάσταση του παιχνιδιού στην οθόνη.

Υποστηρίζει single player mode και multiplayer, με δυνατότητα επιλογής δυσκολίας.

**Διαχείριση Συγκρούσεων και Σκορ**

Κατά τη διάρκεια του παιχνιδιού γίνεται έλεγχος για συγκρούσεις μεταξύ των σφαιρών και των εχθρών (οπότε και αυξάνεται το σκορ), των παικτών και των εχθρών (οπότε και ο παίκτης χάνει και τελειώνει το παιχνίδι) και των παικτών με τα power-ups (οπότε και αυτά ενεργοποιούνται).

Επίσης, το σκορ του παίκτη καταγράφεται και εμφανίζεται το τελικό σκορ όταν τελειώσει το παιχνίδι.

**Επιλογές (Options) και High Score**

Περιλαμβάνεται ένα μενού επιλογών για την αλλαγή της δυσκολίας και ένα μενού για την προβολή των υψηλών σκορ.

Το παιχνίδι χρησιμοποιεί μια εξωτερική βάση δεδομένων (SQLite) για την αποθήκευση και ανάκτηση των HighScores.

**Σχεδιασμός του Galaxy Shooter – Game Mechanics**

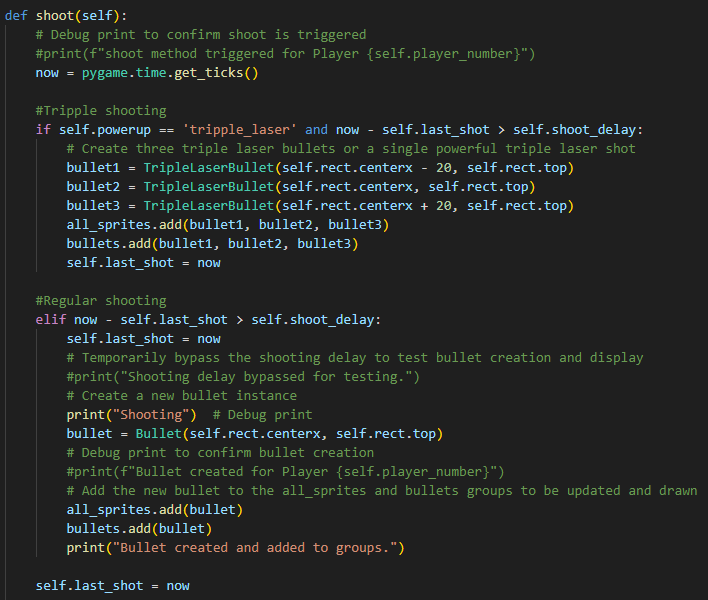
Στο “Galaxy Shooter”, οι παίκτες οδηγούν ένα διαστημόπλοιο εν μέσω εχθρών και εμποδίων. Οι βασικοί μηχανισμοί του παιχνιδιού περιλαμβάνουν γύρω από τον πυροβολισμό, την αποφυγή των εχθρών μέσω της κίνησης του παίκτη στον άξονα χ΄χ και τη χρήση power-up για την ενδυνάμωση του όπλου (laser).

**Μηχανισμός σκοποβολής:**

Η κύρια αλληλεπίδραση στο “Galaxy Shooter” είναι ο μηχανισμός πυροβολισμών. Οι παίκτες ελέγχουν ένα διαστημόπλοιο εξοπλισμένο με ένα κανόνι λέιζερ, ικανό να εκτοξεύει σφαίρες για να καταστρέφει τους εισερχόμενους εχθρούς. Ο μηχανισμός πυροβολισμών ενεργοποιείται με το πάτημα συγκεκριμένων πλήκτρων (spacebar για τον παίκτη 1, αριστερό Ctrl για τον παίκτη 2), επιτρέποντας ένα ευέλικτο gameplay. Το παιχνίδι διαθέτει καθυστέρηση βολής για την εξισορρόπηση του ρυθμού βολής, με σκοπό να διασφαλιστεί ότι το παιχνίδι παραμένει «δίκαιο», με την έννοια ότι δεν είναι δυνατή η ρίψη συνεχών βολών που θα είχε ως συνέπεια τον άμεσο καθαρισμό της οθόνης από τους εχθρούς, ενώ, έτσι, τίθεται και βαθμός δυσκολίας για τον παίκτη.

Χρησιμοποιείται το power-up «Triple Laser Bullet», το οποίο αναβαθμίζει τον μηχανισμό πυροβολισμού. Μόλις ενεργοποιηθεί, το διαστημόπλοιο του παίκτη αναβαθμίζεται προσωρινά ώστε να εκτοξεύει τρεις σφαίρες ταυτόχρονα, αυξάνοντας σημαντικά τη δύναμη πυρός του παίκτη. Αυτό το power-up, εφόσον το «κερδίσει» ο παίκτης με την συλλογή «καρδιών», αναπαρίσταται οπτικά με μια ξεχωριστή εικόνα σφαίρας (Triple\_Laser.png), διαφοροποιώντας το από τις τυπικές σφαίρες λέιζερ και προσθέτοντας μια νέα δυναμική στις μάχες.

Στην Εικόνα 1 φαίνεται το σχετικό στιγμιότυπο του κώδικα για τον ορισμό του μηχανισμού πυροβολισμού, όπου ορίζεται η συμπεριφορά για τον πυροβολισμό χωρίς και με το power-up (στην απλή – κανονική βολή αποδεσμεύεται μία σφαίρα η οποία έχει μια απλή μορφή laser beam, ενώ στην βολή μετά το power-up αποδεσμεύονται ταυτόχρονα τρεις σφαίρες, οι οποίες μπορούν να «σκοτώσουν» τρεις εχθρούς, με τη χρήση και διαφορετικού γραφικού).



Εικόνα 1 Μηχανισμός πυροβολισμού (κώδικας Python)

Αυτό το στιγμιότυπο από τη μέθοδο shoot της κλάσης Player ελέγχει αν έχει περάσει αρκετός χρόνος από την τελευταία βολή (για την υλοποίηση μιας καθυστέρησης βολής). Στη συνέχεια, δημιουργεί μια νέα περίπτωση Bullet στη θέση του παίκτη και την προσθέτει στις σχετικές ομάδες sprite για απόδοση και ανίχνευση σύγκρουσης. Το ίδιο συμβαίνει και στην περίπτωση της βολή μετά τη λήψη του powerup με τη διαφοροποίηση ότι τώρα ενεργοποιούνται τρεις περιπτώσεις Bullet στη θέση του παίκτη.

Στην Εικόνα 2 φαίνονται στιγμιότυπα του gameplay για τις δύο περιπτώσεις.



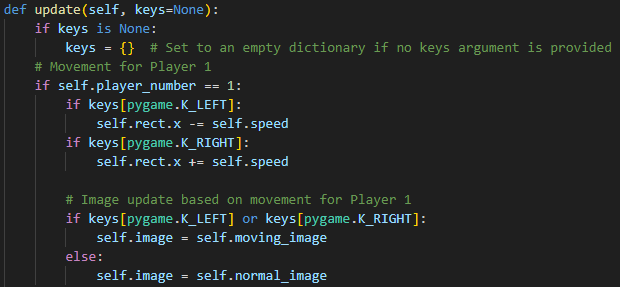
B

A

Εικόνα 2 Γραφική απεικόνιση απλής βολής (Α) και βολής μετά το power-up (B)

**Μηχανισμός κίνησης και αποφυγής:**

Η αποφυγή των εχθρών είναι εξίσου σημαντική για το gameplay. Οι παίκτες πρέπει να κατευθύνουν το διαστημόπλοιό τους μέσα από έναν λαβύρινθο εχθρών, απαιτώντας γρήγορα αντανακλαστικά και στρατηγική κίνηση. Το σύστημα ελέγχου του παιχνιδιού είναι σχεδιασμένο για ευελιξία, με τους παίκτες να κινούν το διαστημόπλοιο τους αριστερά ή δεξιά στον άξονα χ΄χ για να αποφεύγουν την απειλή, καθώς η σύγκρουση με εχθρό συνεπάγεται ότι ο παίκτης θα χάσει. Για την κίνηση των διαστημοπλοίων χρησιμοποιούνται τα πλήκτρα K\_LEFT και K-RIGHT (αριστερό και δεξί βελάκι) στην περίπτωση του player 1 (Single Player και Παίκτης 1 στο Multiplayer) και τα πλήκτρα K\_a και K\_d στην περίπτωση του player 2 (Multiplayer). Στιγμιότυπο του κώδικα φαίνεται στην Εικόνα 3.

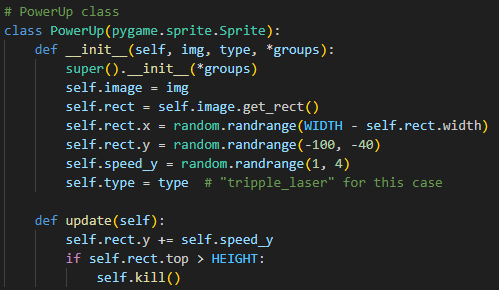


Εικόνα 3 Μηχανισμός κίνησης των παικτών και αποφυγής (κώδικας Python)

Αυτό το στιγμιότυπο από τη μέθοδο ενημέρωσης της κλάσης Player, αυτός ο κώδικας προσαρμόζει τη θέση του παίκτη με βάση τα πλήκτρα αριστερού και δεξιού βέλους. Η ταχύτητα του παίκτη καθορίζει την απόσταση κίνησης ανά καρέ, διευκολύνοντας την ομαλή αποφυγή.

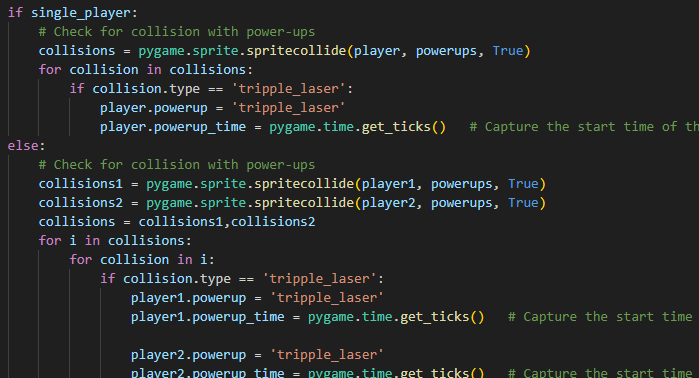
**Power-ups:**

Τα power-ups, προσφέρουν στους παίκτες προσωρινά πλεονεκτήματα, καθώς είναι δυνατή η εξουδετέρωση τριών εχθρών με μία βολή (εφόσον οι εχθροί αυτοί έρθουν σε επαφή με μία από τις τρεις bullets). Το power-up που χρησιμοποιείται είναι το “Triple Laser”, το οποίο μετατρέπει την κανονική σφαίρα του παίκτη σε μια πιο ισχυρή τριπλή βολή (ταυτόχρονη ρίψη τριών bullet instances) για δέκα δευτερόλεπτα. Τα power-ups εμφανίζονται τυχαία, με την πρώτη εμφάνιση να συνδέεται με την επίτευξη σκορ 250. Η συλλογή ενός power-up είναι απλή, καθώς η σύγκρουση του διαστημόπλοιου του παίκτη με αυτό, ενεργοποιεί άμεσα τον σχετικό μηχανισμό. Στιγμιότυπο του σχετικού κώδικα για τη δημιουργία της κλάσης του power up φαίνεται στην Εικόνα 4, ενώ για την ενεργοποίηση του powerup με τη σύγκρουση με τον παίκτη, φαίνεται στην Εικόνα 5 και στιγμιότυπο κατά το gameplay με το power-up εμφανές στην οθόνη φαίνεται στην Εικόνα 6.



Εικόνα 4 Μηχανισμός PowerUp – Δημιουργία της κλάσης (κώδικας Python)

Η κλάση PowerUp σε αυτό το στιγμιότυπο (Εικόνα 4) έχει σχεδιαστεί για να δημιουργεί και να διαχειρίζεται αντικείμενα power-up μέσα στο παιχνίδι. Είναι υποκλάση της pygame.sprite.Sprite. Ενσωματώνει τη δημιουργία, τη συμπεριφορά και τον αυτόματο καθαρισμό των αντικειμένων power-up. Όταν ενεργοποιείται, εμφανίζεται ένα power-up σε μια τυχαία θέση στην οθόνη και κινείται προς τα κάτω μέχρι είτε να συγκρουστεί με τον παίκτη (ενεργοποιώντας το επιδιωκόμενο αποτέλεσμα) είτε να περάσει την κάτω άκρη της οθόνης, οπότε και δεν ενεργοποιείται από το παιχνίδι. Επίσης, υποστηρίζει πολλαπλούς τύπους power-up μέσω του χαρακτηριστικού type.



Εικόνα 5 Μηχανισμός PowerUp (κώδικας Python)

Στο στιγμιότυπο αυτό (Εικόνα 5) φαίνεται ο μηχανισμός διαχείρισης της σύγκρουσης μεταξύ του παίκτη ή των παικτών και των power-ups στο παιχνίδι. Η λογική στον ξεχωριστό έλεγχο για κάθε παίκτη (player1, player2) της πιθανής σύγκρουσης με power-ups. Για κάθε παίκτη, αν συγκρουστεί με power-up τύπου “tripple\_laser”, ενεργοποιεί την εν λόγω βελτίωση γι’ αυτόν, καταγράφοντας το χρόνο ενεργοποίησης, καθώς η διάρκεια των power-ups εφόσον συγκρουστούν είναι 10 δευτερόλεπτα. Αν και οι δύο παίκτες συγκρουστούν με power-up, και οι δύο λαμβάνουν την ενίσχυση και οι μεταβλητές player1.powerup και player2.powerup ενημερώνονται αντίστοιχα.



Εικόνα 6 Στιγμιότυπο του gameplay με γραφική απεικόνιση του PowerUp

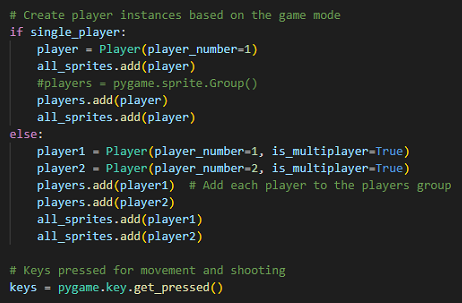
Επιπλέον, το παιχνίδι ενσωματώνει ηχητικά εφέ για να ενισχύσει την εμπειρία του παιχνιδιού. Οι ξεχωριστοί ήχοι για τα χτυπήματα του εχθρού (enemy\_hit.wav) και τα χτυπήματα του παίκτη (player\_hit.wav) παρέχουν ακουστική ανατροφοδότηση στον παίκτη, ενώ συμβάλλουν και στην ατμόσφαιρα του παιχνιδιού.

**Σχεδιασμός των Επιπέδων του Galaxy Shooter**

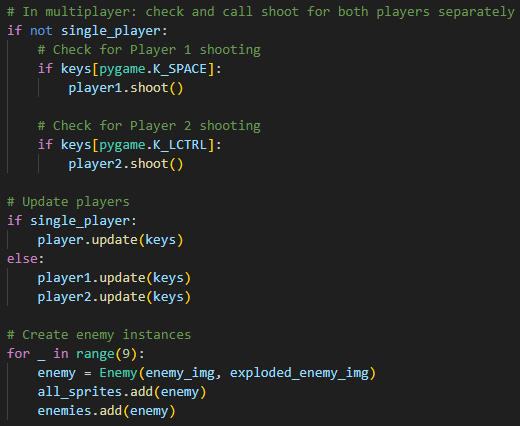
Όπως προαναφέρεται, το “Galaxy Shooter”, είναι ένα παιχνίδι στο οποίο οι παίκτες οδηγούν ένα διαστημόπλοιο εν μέσω εχθρών και εμποδίων, τα οποία πρέπει να πυροβολήσουν για να τα εξοντώσουν και, έτσι, να αυξήσουν το σκορ τους και να μη χάσουν λόγω της σύγκρουσής τους με αυτούς. Ο σχεδιασμός των επιπέδων στο “Galaxy Shooter” βασίζεται στη σταδιακή αύξηση της δυσκολίας, με την προσθήκη διαφορετικών τύπων εχθρών, αλλαγές στην ταχύτητα και στην παροχή power-ups.

**Διαχείριση Παικτών**

Καταρχάς στο gameplay αρχικοποιούνται οι παίκτες (ένας παίκτης για το single player mode και δύο παίκτες στο multiplayer) και προστίθενται στις αντίστοιχες ομάδες (sprites), ενώ, ταυτόχρονα γίνεται διαχείριση και της κλήσης του πυροβολισμού από τους παίκτες, όπως φαίνεται στα στιγμιότυπα του κώδικα των Εικόνων 7 και 8.



Εικόνα 7 Δημιουργία των instances των παικτών(κώδικας Python)



Εικόνα 8 Διαχείριση και κλήση του πυροβολισμού από τον παίκτη (κώδικας Python)

**Διαχείριση Εχθρών**

Ο κώδικας για τη δημιουργία των εχθρών κατά το gameplay αρχικοποιεί τους εχθρούς και τους προσθέτει στις αντίστοιχες ομάδες (sprites). Η ποσότητα και ο τύπος των εχθρών μπορεί να προσαρμοστεί για να δημιουργήσει διάφορα επίπεδα δυσκολίας. Στο παιχνίδι περιλαμβάνονται δύο τύποι εχθρών. Ο πρώτος εχθρός κινείται με μικρότερη ταχύτητα, έχει τη μορφή UFO, εμφανίζεται από την αρχή και η εξουδετέρωσή του συνεπάγεται αύξηση του σκορ κατά 10 βαθμούς, ενώ ο δεύτερος κινείται με μεγαλύτερη ταχύτητα, έχει τη μορφή μετεωρίτη και εμφανίζεται όταν ο παίκτης φτάσει σε σκορ 200, στην οθόνη μπορούν να συνυπάρχουν μόνο τρεις εχθροί του δεύτερου τύπου και η εξουδετέρωσή του συνεπάγεται αύξηση του σκορ κατά 20 βαθμούς (Εικόνες 9 και 10).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 9 Δημιουργία των instances των εχθρών (κώδικας Python)

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 10 Διαχείριση και κλήση των εχθρών (κώδικας Python)

**Αύξηση Δυσκολίας**

Το τμήμα του κώδικα που φαίνεται στην Εικόνα 11, προσαρμόζει την ταχύτητα των εχθρών ανάλογα με την επιλεγμένη δυσκολία (από την αντίστοιχη οθόνη επιλογής δυσκολίας – options screen), προσθέτοντας ενδιαφέρουσες εναλλαγές.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λογισμικό πολυμέσων

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 11 Καθορισμός του επιπέδου δυσκολίας (κώδικας Python)

**Power-Ups**

Τα Power-ups εμφανίζονται τυχαία (βαθμός τυχαιότητας <0,005), εφόσον το σκορ φτάσει στους 250 βαθμούς, προσφέροντας στον παίκτη προσωρινές βελτιώσεις με τη μορφή ρίψης σφαιρών (Εικόνα 12).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 12 Κώδικας για την ενεργοποίηση του power-up

**Συγκρούσεις με τους εχθρούς και Αύξηση του σκορ**

Κάθε φορά που ο παίκτης καταρρίπτει έναν εχθρό, κερδίζει πόντους και ενεργοποιείται η εμφάνιση νέων εχθρών, αυξάνοντας τη δυσκολία και την πρόκληση. Το στιγμιότυπο του κώδικα που φαίνεται στην Εικόνα 13 χειρίζεται την ανίχνευση σύγκρουσης και την επακόλουθη λογική για το πότε οι σφαίρες συγκρούονται με τους εχθρούς στο παιχνίδι. Χρησιμοποιεί τη μέθοδο ανίχνευσης σύγκρουσης ομάδων sprite του Pygame για τον εντοπισμό και την επεξεργασία αλληλεπιδράσεων μεταξύ αντικειμένων σφαίρας και εχθρών. Η pygame.sprite.groupcollide(enemies, bullets, True, True) ελέγχει για συγκρούσεις μεταξύ δύο ομάδων: εχθρών και σφαιρών. Εάν εντοπιστεί σύγκρουση, τόσο ο εχθρός όσο και η σφαίρα που εμπλέκονται στη σύγκρουση αφαιρούνται από τις αντίστοιχες ομάδες τους. Αυτό υποδεικνύεται από τις παραμέτρους True, True, οι οποίες υπαγορεύουν ότι τα sprites θα πρέπει να σκοτώνονται (να αφαιρούνται από όλες τις ομάδες sprites) κατά τη σύγκρουση. Για κάθε σύγκρουση που ανιχνεύεται (each hit in hits), πραγματοποιούνται διάφορες ενέργειες:

* Ένα ηχητικό εφέ χτυπήματος του εχθρού αναπαράγεται χρησιμοποιώντας την enemy\_hit\_sound.play().
* Το σκορ του παίκτη αυξάνεται κατά 10 πόντους για να ανταμείψει τον παίκτη για το χτύπημα ενός εχθρού.
* Η μέθοδος hit\_by\_bullet καλείται στο εχθρικό αντικείμενο που χτυπήθηκε, προκαλώντας την εξουδετέρωσή του.
* Ένας νέος εχθρός εμφανίζεται και προστίθεται στις ομάδες all\_sprites και enemies, αντικαθιστώντας ουσιαστικά τον εχθρό που μόλις εξουδετερώθηκε ώστε να εξασφαλίζεται μια συνεχής ροή εχθρών.

Μια παρόμοια διαδικασία ακολουθείται για τις συγκρούσεις μεταξύ των σφαιρών και του δεύτερου τύπου εχθρών (second\_enemies). Ωστόσο, υπάρχουν μερικές βασικές διαφορές:

* Η προσαύξηση της βαθμολογίας είναι υψηλότερη (20 πόντοι) για κάθε χτύπημα δεύτερου εχθρού, οπότε αυτοί οι εχθροί θεωρούνται πιο δύσκολοι και σημαντικοί στόχοι.
* Μετά από ένα χτύπημα, δημιουργείται ένας νέος δεύτερος εχθρός με ενδεχομένως διαφορετικές ιδιότητες (speed\_factor=1.3) και προστίθεται στο παιχνίδι.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

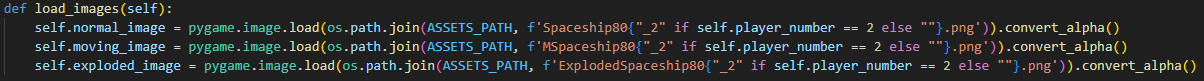
Εικόνα 13 Συγκρούσεις με τους εχθρούς και Αύξηση του σκορ (κώδικας Python)

**Σχεδιασμός των χαρακτήρων και των εχθρών του Galaxy Shooter**

Ο σχεδιασμός των χαρακτήρων του παίκτη και των εχθρών στο “Galaxy Shooter” αξιοποιεί διάφορες οπτικές αναπαραστάσεις και συμπεριφορές για να δημιουργήσει μια πλούσια και συναρπαστική εμπειρία παιχνιδιού. Μέσω της χρήσης sprites, κινούμενων σχεδίων και power-ups, το παιχνίδι προσφέρει δυναμικό περιβάλλον.

**Σχεδιασμός χαρακτήρα παίκτη**

Το διαστημόπλοιο του παίκτη αναπαρίσταται οπτικά με διαφορετικές εικόνες για την ακινησία, την κίνηση και την ανατίναξή του. Αυτό επιτυγχάνεται με τη φόρτωση συγκεκριμένων εικόνων για κάθε κατάσταση (Εικόνα 14). Όσον αφορά τη μηχανική του gameplay, το διαστημόπλοιο του παίκτη έχει ιδιότητες για την ταχύτητα, την καθυστέρηση βολής και τις ζωές, κάνοντας την εμπειρία του παιχνιδιού δυναμική (Εικόνα 15). Ο παίκτης μπορεί να αποκτήσει ένα power-up, που ενισχύει προσωρινά την ικανότητα πυροβολισμού (Εικόνα 16).



Εικόνα 14 Οπτική αναπαράσταση παίκτη

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 15 Μηχανική του gameplay

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 16 Απόκτηση του Power-up μέσω σύγκρουσης με αυτό

**Σχεδιασμός εχθρού**

Το παιχνίδι διαθέτει δύο κύριους τύπους εχθρών, ο καθένας με ξεχωριστή εικόνα και συμπεριφορά. Ο βασικός εχθρός κινείται με τυπική ταχύτητα. Ο δεύτερος τύπος, που απεικονίζεται ως εχθρός meteo, κινείται ταχύτερα και έχει διαφορετικό εφέ έκρηξης (Εικόνα 17). Οι εχθροί κινούνται κατακόρυφα προς τα κάτω στην οθόνη με ταχύτητα που καθορίζεται από τον τύπο τους. Όταν χτυπηθούν από σφαίρα, μεταβαίνουν σε κατάσταση έκρηξης πριν καταστραφούν. Αυτή η συμπεριφορά ελέγχεται μέσω μιας μεθόδου ενημέρωσης μέσα σε κάθε κλάση εχθρού (Εικόνες 18 και 19).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 17 Τύποι εχθρών

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 18 Κλάση και συμπεριφορά βασικού εχθρού

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 19 Κλάση και συμπεριφορά δεύτερου εχθρού

**Σκορ**

Οι παίκτες κερδίζουν πόντους για την καταστροφή των εχθρών, με διαφορετικούς πόντους ανάλογα με τον τύπο του εχθρού. Αυτό ενθαρρύνει τους παίκτες να στοχεύουν όλους τους εχθρούς, δίνοντας προτεραιότητα σε αυτούς που προσφέρουν υψηλότερους πόντους (Εικόνες 20 και 21).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 20 Σκορ από την εξουδετέρωση του βασικού εχθρού

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 21 Σκορ από την εξουδετέρωση του δεύτερου εχθρού

**Υλοποίηση του Galaxy Shooter**

**Τεχνικό πλαίσιο**

Το τεχνικό πλαίσιο του παιχνιδιού βασίζεται στην αξιοποίηση της Python και του Pygame για τη δημιουργία ενός παιχνιδιού τύπου arcade, μέσω της χρήσης βασικών βιβλιοθηκών για γραφικά, ήχο και χειρισμό συμβάντων, σε συνδυασμό με τις λοιπές γραμμές του προγράμματος, το παιχνίδι προσφέρει μια απρόσκοπτη και ευχάριστη εμπειρία χρήσης. Αυτό το τεχνικό θεμέλιο όχι μόνο υποστηρίζει τη λειτουργικότητα του τρέχοντος παιχνιδιού, αλλά παρέχει επίσης μια επεκτάσιμη δομή για μελλοντικές βελτιώσεις και εξελίξεις.

**Περιβάλλον ανάπτυξης**

Γλώσσα προγραμματισμού: Python

Κύρια βιβλιοθήκη: Pygame

Το Pygame, ένα σύνολο από modules της Python που έχουν σχεδιαστεί για τη συγγραφή βιντεοπαιχνιδιών, προσφέρει λειτουργικότητα για τη δημιουργία παιχνιδιών και εφαρμογών πολυμέσων. Παρέχει εύκολες στη χρήση εντολές για γραφικά, ήχους και χειρισμό εισόδου, καθιστώντας το μια δημοφιλή επιλογή για τους προγραμματιστές. Χρησιμοποιείται για τη δημιουργία του παραθύρου του παιχνιδιού, το χειρισμό συμβάντων, τη σχεδίαση γραφικών, την αναπαραγωγή ήχων και τη διαχείριση των καταστάσεων του παιχνιδιού. Αποτελεί τη βάση ανάπτυξης του παιχνιδιού, διευκολύνοντας την απόδοση των γραφικών, την ανίχνευση της εισόδου και την αναπαραγωγή του ήχου (import pygame, pygame.init()).

Βασικές βιβλιοθήκες και modules:

* Random: χρησιμοποιείται για τη δημιουργία τυχαίων αριθμών, οι οποίοι είναι απαραίτητοι για την εμφάνιση εχθρών σε τυχαίες θέσεις και ταχύτητες, δημιουργώντας απρόβλεπτη κίνηση των εχθρών και κάνοντας το παιχνίδι πιο ενδιαφέρον και δυναμικό (import random).
* OS: χρησιμοποιείται για την αλληλεπίδραση με το λειτουργικό σύστημα για τη διαχείριση των διαδρομών αρχείων, ώστε να επιτευχθεί η φόρτωση των στοιχείων του παιχνιδιού (εικόνες και ήχοι) με τρόπο ανεξάρτητο από την πλατφόρμα (import os).
* Datetime: χρησιμοποιείται για την καταγραφή της τρέχουσας ημερομηνίας και ώρας κάθε φορά που επιτυγχάνεται νέο υψηλό σκορ, με τις πληροφορίες αυτές να αποθηκεύονται σε βάση δεδομένων (import datetime).

**Περιγραφή υλοποίησης**

Βρόχος παιχνιδιού (Game Loop):

Όπου ενημερώνεται η κατάσταση του παιχνιδιού, επεξεργάζεται η είσοδος και απεικονίζονται τα γραφικά. Αυτός ο βρόχος εκτελείται συνεχώς μέχρι την έξοδο από το παιχνίδι, εξασφαλίζοντας ομαλό παιχνίδι και απόκριση σε πραγματικό χρόνο στις ενέργειες του χρήστη.

Διαχείριση Sprite:

Η κλάση Sprite του Pygame χρησιμοποιείται εκτενώς για την αναπαράσταση οντοτήτων του παιχνιδιού, όπως το διαστημόπλοιο του παίκτη, οι εχθροί, οι σφαίρες και τα power-ups, απλοποιώντας την ανίχνευση συγκρούσεων, την απόδοση και τη διαχείριση καταστάσεων.

Ανίχνευση σύγκρουσης:

Το παιχνίδι χρησιμοποιεί τη λειτουργικότητα ανίχνευσης σύγκρουσης του Pygame για τη διαχείριση των αλληλεπιδράσεων μεταξύ του παίκτη, των εχθρών, των σφαιρών και των power-ups. Αυτό είναι απαραίτητο για την υλοποίηση μηχανισμών παιχνιδιού όπως η πυροβολισμός, η καταστροφή εχθρών και η συλλογή power-up.

Ηχητικά εφέ και μουσική:

Το module mixer του Pygame χρησιμοποιείται για την προσθήκη μουσικής υπόκρουσης και ηχητικών εφέ, ενισχύοντας την καθηλωτική εμπειρία του παιχνιδιού. Οι ήχοι ενεργοποιούνται από συγκεκριμένα γεγονότα, όπως πυροβολισμοί, εκρήξεις εχθρών και εξουδετέρωση του παίκτη.

**Αλγόριθμοι και δομή του Κώδικα**

**Ανίχνευση σύγκρουσης**

Ένα από τα κεντρικά χαρακτηριστικά του παιχνιδιού είναι ο μηχανισμός ανίχνευσης σύγκρουσης, ο οποίος καθορίζει τις αλληλεπιδράσεις μεταξύ των διαφόρων οντοτήτων του παιχνιδιού, όπως το σκάφος του παίκτη, οι εχθροί, οι σφαίρες και τα power-ups. Το Pygame παρέχει αποτελεσματικές μεθόδους για τον χειρισμό των συγκρούσεων, κυρίως μέσω των συναρτήσεων pygame.sprite.groupcollide και pygame.sprite.spritecollide.

Συγκρούσεις εχθρών και σφαιρών:

Το παιχνίδι ελέγχει για συγκρούσεις μεταξύ σφαιρών και εχθρών χρησιμοποιώντας τη συνάρτηση pygame.sprite.groupcollide. Αυτή η μέθοδος εντοπίζει αποτελεσματικά τις επικαλύψεις sprites μεταξύ δύο ομάδων - σε αυτή την περίπτωση, σφαίρες και εχθρούς. Όταν μια σφαίρα χτυπήσει έναν εχθρό, τόσο η σφαίρα όσο και ο εχθρός αφαιρούνται από την οθόνη και ενημερώνεται το σκορ. Αυτός ο μηχανισμός χρησιμοποιείται επίσης για την ενεργοποίηση των κινήσεων έκρηξης του εχθρού και των ηχητικών εφέ (Εικόνα 22).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 22 Συγκρούσεις εχθρών και σφαιρών

Συγκρούσεις παικτών και εχθρών:

Η συνάρτηση pygame.sprite.spritecollide ανιχνεύει τις συγκρούσεις μεταξύ του παίκτη και των εχθρών. Ένα τέτοιο συμβάν συνήθως έχει ως αποτέλεσμα ο παίκτης να χάνει και είναι καθοριστικό στην πρόκληση του παιχνιδιού, καθώς αναγκάζει τον παίκτη να αποφεύγει τους εισερχόμενους εχθρούς (Εικόνες 23 και 24).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 23 Συγκρούσεις παικτών και βασικών εχθρών

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 24 Συγκρούσεις παικτών και δεύτερων εχθρών

**Μηχανική κίνησης**

Η κίνηση του σκάφους του παίκτη και των εχθρών είναι βασικά χαρακτηριστικά του παιχνιδιού. Το σκάφος του παίκτη ελέγχεται με τη χρήση εισόδων από το πληκτρολόγιο, τις οποίες διαχειρίζεται το σύστημα συμβάντων του Pygame. Οι εχθροί κινούνται αυτόματα με προκαθορισμένο τρόπο, αυξάνοντας την πρόκληση του παιχνιδιού.

Κίνηση του παίκτη:

Τα συμβάντα του πληκτρολογίου καταγράφονται για να μετακινηθεί το σκάφος του παίκτη αριστερά ή δεξιά μέσα στο παράθυρο του παιχνιδιού. Η ταχύτητα της κίνησης ελέγχεται από μια προκαθορισμένη μεταβλητή ταχύτητας, επιτρέποντας τον ομαλό και ευέλικτο έλεγχο (Εικόνα 25).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 25 Κίνηση των παικτών (διαστημοπλοίων)

Κίνηση των εχθρών:

Οι εχθροί κινούνται στην οθόνη με διαφορετικές ταχύτητες. Το παιχνίδι δημιουργεί τυχαία αυτές τις ταχύτητες και τις θέσεις εκκίνησης για να διασφαλίσει τη μεταβλητότητα του παιχνιδιού. Μόλις ένας εχθρός απομακρυνθεί από την οθόνη ή καταστραφεί, επανέρχεται στην κορυφή με νέα τυχαία ταχύτητα και θέση (Εικόνα 26 για τον βασικό εχθρό και Εικόνα 27 για τον δεύτερο εχθρό).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 26 Κίνηση του βασικού εχθρού

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λειτουργικό σύστημα

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 27 Κίνηση του δεύτερου εχθρού

Σύστημα Σκορ

Το σύστημα μέτρησης σκορ στο παιχνίδι ανταμείβει τον παίκτη για την καταστροφή των εχθρών. Κάθε εχθρός που καταστρέφεται αυξάνει το σκορ του παίκτη, το οποίο εμφανίζεται στην οθόνη. Το σύστημα σκορ είναι ένας απλός αλλά αποτελεσματικός τρόπος για να παρέχει ανατροφοδότηση στον παίκτη, ενθαρρύνοντάς τον να συνεχίσει να παίζει και να βελτιώνει τις ικανότητές του. Το παιχνίδι εμφανίζει το τρέχον σκορ χρησιμοποιώντας το σύστημα γραμματοσειρών του Pygame, ενημερώνοντας την οθόνη κάθε φορά που αλλάζει το σκορ. Αυτή η ανατροφοδότηση σε πραγματικό χρόνο συμβάλει στη διατήρηση της συμμετοχής και του κινήτρου του παίκτη (Εικόνες 28 και 29).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 28 Συνάρτηση εμφάνισης του σκορ

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 29 Αύξηση του σκορ κατά 10 βαθμούς για την εξουδετέρωση του βασικού εχθρού και κατά 20 βαθμούς για την εξουδετέρωση του δεύτερου εχθρού

**Λειτουργικότητα του παιχνιδιού**

**Τρόποι παιχνιδιού**

Το παιχνίδι παρέχει δύο κύριες λειτουργίες παιχνιδιού: Single Player και Multiplayer, οι οποίες επιλέγονται από την αρχική οθόνη του παιχνιδιού (Main Menu). Το κύριο μενού του παιχνιδιού χρησιμεύει ως ο κεντρικός άξονας για την επιλογή τρόπου λειτουργίας. Παρουσιάζει στον παίκτη διάφορες επιλογές, (Single Player, Multiplayer, High Score, Options και Exit). Η επιλογή γίνεται χρησιμοποιώντας το ποντίκι για να γίνει κλικ στην επιθυμητή λειτουργία. Η οπτική ανατροφοδότηση παρέχεται μέσω της αλλαγής χρώματος στο κουμπί όταν περνάει το ποντίκι από πάνω του, ενισχύοντας την αλληλεπίδραση του χρήστη. (Εικόνα 30).

Όταν ο παίκτης επιλέξει μια λειτουργία, καλείται η συνάρτηση start\_game με μια παράμετρο που υποδεικνύει αν το παιχνίδι πρέπει να εκτελεστεί σε λειτουργία Single Player ή Multiplayer. Αυτή η παράμετρος (single\_player) είναι μια boolean τιμή που ελέγχει διάφορες πτυχές της συμπεριφοράς του παιχνιδιού, συμπεριλαμβανομένης της εμφάνισης των αντικειμένων παίκτη, των αλληλεπιδράσεων με τους εχθρούς και του σκορ.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 30 Αρχική οθόνη του παιχνιδιού με τα κουμπιά επιλογών

Λειτουργία Single Player:

Σε αυτή τη λειτουργία, ένας παίκτης ελέγχει ένα διαστημόπλοιο και πλοηγείται σε επίπεδα γεμάτα εχθρούς και εμπόδια. Ο στόχος είναι να επιβιώσει αποφεύγοντας τα εχθρικά πυρά και καταστρέφοντας τους εχθρούς. Δημιουργείται μόνο ένα αντικείμενο παίκτη και το παιχνίδι επικεντρώνεται στην αλληλεπίδραση αυτού του παίκτη με το περιβάλλον, συμπεριλαμβανομένων των πυροβολισμών σε εχθρούς, της συλλογής power-ups και της αποφυγής εισερχόμενων πυρών (Εικόνα 31).

Λειτουργία πολλαπλών παικτών (Multiplayer):

Αυτή η λειτουργία επιτρέπει σε δύο παίκτες να παίζουν ταυτόχρονα, μοιραζόμενοι την ίδια οθόνη. Κάθε παίκτης ελέγχει το δικό του διαστημόπλοιο, συνεργαζόμενος με τον άλλον προσθέτοντας ένα κοινωνικό στοιχείο στο παιχνίδι, ενθαρρύνοντας τη συνεργασία μεταξύ των παικτών. Δημιουργούνται δύο αντικείμενα παικτών, το καθένα από τα οποία αντιστοιχεί σε έναν παίκτη στην οθόνη. Το παιχνίδι προσαρμόζεται για να εξυπηρετήσει και τους δύο παίκτες, από τη δημιουργία εχθρών μέχρι τον υπολογισμό του σκορ (Εικόνα 31).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 31 Κώδικας για τη δημιουργία των αντικειμένων των παικτών

Power-ups

Τα Power-ups είναι ειδικά αντικείμενα που παρέχουν στο διαστημόπλοιο του παίκτη προσωρινά πλεονεκτήματα. Έχουν σχεδιαστεί για να προσθέτουν ένα επιπλέον επίπεδο στρατηγικής και ενθουσιασμού στο παιχνίδι. Τα power-ups στο παιχνίδι οδηγούν στην εμφάνιση και χρήση από τον παίκτη του τριπλού λέιζερ. Αυτό το power-up επιτρέπει στο σκάφος του παίκτη να εκτοξεύει τρία λέιζερ ταυτόχρονα, αυξάνοντας τη δύναμη πυρός για περιορισμένο χρονικό διάστημα. Βελτιώνει σημαντικά την ικανότητα του παίκτη να εξουδετερώνει τους εχθρούς γρηγορότερα. Τα power-ups εμφανίζονται τυχαία και μπορούν να συλλεχθούν με τη σύγκρουση του γραφικού του παίκτη (ή των παικτών) με αυτά. Η υλοποίηση αυτών των power-ups περιλαμβάνει την ανίχνευση των συγκρούσεων με το σκάφος του παίκτη και την εφαρμογή του συγκεκριμένου power-up εφέ για συγκεκριμένη διάρκεια (Εικόνες 5 και 6) .

**Επίπεδα δυσκολίας**

Το παιχνίδι διαθέτει ρυθμιζόμενα επίπεδα δυσκολίας (δυνατότητα επιλογής από την οθόνη “options” όπως φαίνεται στην Εικόνα 32), τα οποία επηρεάζουν την πρόκληση του παιχνιδιού. Η δυσκολία μπορεί να επηρεάσει την ταχύτητα των κινήσεων των εχθρών και τον αριθμό των εχθρών που αναπαράγονται. Η προσαρμογή των επιπέδων δυσκολίας επιτρέπει στο παιχνίδι να είναι προσιτό σε ένα ευρύτερο κοινό, από αρχάριους έως έμπειρους παίκτες, και παρέχει πρόσθετες προκλήσεις για όσους θέλουν να δοκιμάσουν τις ικανότητές τους .

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, αριθμός

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 32 Επιλογή επιπέδου δυσκολίας

Το επίπεδο δυσκολίας ορίζεται μέσα από τη συνάρτηση show\_options\_screen, όπου οι παίκτες μπορούν να επιλέξουν τη δυσκολία που προτιμούν από τις διαθέσιμες επιλογές. Η επιλογή αυτή γίνεται μέσω μιας απλής διεπαφής χρήστη στο μενού επιλογών του παιχνιδιού. (Εικόνα 33). Το επιλεγμένο επίπεδο δυσκολίας επηρεάζει την ταχύτητα των εχθρών, με την προσαρμογή των σχετικών παραμέτρων του παιχνιδιού ανάλογα με την επιλεγμένη δυσκολία (Εικόνα 34). Η ταχύτητα των εχθρών είναι μια άμεση εκδήλωση του επιπέδου δυσκολίας του παιχνιδιού. Τροποποιώντας τη μεταβλητή enemy\_speed, το παιχνίδι μπορεί να ελέγξει πόσο γρήγορα κινούνται οι εχθροί, κάνοντάς τους πιο δύσκολο να εξουδετερωθούν όσο αυξάνεται το επίπεδο δυσκολίας (Εικόνες 35 και 36).

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 33 Ο κώδικας της επιλογής του επιπέδο δυσκολίας

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 34 Εφαρμογή του επιπέδου δυσκολίας

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 35 Μεταβλητή enemy\_speed για τον βασικό εχθρό

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 36 Μεταβλητή enemy\_speed για τον δεύτερο εχθρό

**Διεπαφή χρήστη και εμπειρία χρήστη**

Η διεπαφή χρήστη (UI) και η εμπειρία χρήστη (UX) είναι σημαντικά χαρακτηριστικά κάθε παιχνιδιού, καθώς επηρεάζουν άμεσα τον τρόπο με τον οποίο οι παίκτες αλληλεπιδρούν με το παιχνίδι και αντιλαμβάνονται την εμπειρία τους.

**Κύριο μενού**

Το κύριο μενού είναι το πρώτο σημείο αλληλεπίδρασης μεταξύ του παίκτη και του παιχνιδιού. Χρησιμεύει ως πύλη εισόδου σε όλα τα χαρακτηριστικά και τους τρόπους λειτουργίας του παιχνιδιού. Το μενού υλοποιείται με τη χρήση κουμπιών με εφέ αιώρησης, που αλλάζουν χρώματα για να παρέχουν οπτική ανατροφοδότηση όταν το ποντίκι βρίσκεται πάνω από ένα διαδραστικό στοιχείο. Αυτή η άμεση ανατροφοδότηση αποτελεί σημαντική πτυχή μιας θετικής εμπειρίας χρήστη, βοηθώντας στην καθοδήγηση των αλληλεπιδράσεων του παίκτη (Εικόνα 30). Σε αυτό το παιχνίδι, το κύριο μενού περιλαμβάνει:

* Λειτουργία μεμονωμένου παίκτη: Μια τυπική λειτουργία παιχνιδιού όπου ένας παίκτης αντιμετωπίζει τις προκλήσεις του παιχνιδιού.
* Λειτουργία πολλαπλών παικτών: Μια επιλογή για πολλούς παίκτες να παίξουν συνεργατικά.
* Οθόνη highscore: Μια οθόνη που εμφανίζει τα υψηλότερα σκορ που έχουν επιτευχθεί στο παιχνίδι, παρέχοντας μια αίσθηση ανταγωνισμού και επίτευξης.
* Μενού επιλογών: Επιτρέπει στους παίκτες να προσαρμόσουν τη δυσκολία του για να προσαρμόσουν την εμπειρία στις προτιμήσεις τους.
* Επιλογή εξόδου: Ένας απλός τρόπος εξόδου από το παιχνίδι.

**Διεπαφή χρήστη (UI) εντός του παιχνιδιού**

Κατά τη διάρκεια του παιχνιδιού, το UI παρέχει στον παίκτη ένδειξη για το σκορ βρίσκεται στο κάτω μέρος της οθόνης και δείχνει το τρέχον σκορ, το οποίο αποτελεί έναν άμεσο μηχανισμό ανατροφοδότησης που δείχνει την απόδοση του παίκτη. Επίσης, στο παιχνίδι χρησιμοποιούνται μηχανισμοί ανατροφοδότησης για την επικοινωνία με τον παίκτη, μέσω των ηχητικών εφέ και τη γραφική αναπαράσταση του διαστημοπλοίου που διαφοροποιείται όταν αυτό είναι στάσιμο ή όταν κινείται, ενώ στη λειτουργία multiplayer τα διαστημόπλοια έχουν και έντονη χρωματική αντίθεση μεταξύ τους ώστε να μην μπερδεύονται οι χρήστες. Οι δύο τύποι εχθρών είναι τελείως διαφορετικά σχεδιασμένοι και έχουν διαφορετικά χρώματα για να γίνονται άμεσα αντιληπτοί. Άμεσα αντιληπτό είναι και το γραφικό που χρησιμοποιείται για το Powerup (Εικόνα .

Εικόνα που περιέχει στιγμιότυπο οθόνης, χώρος, Σύμπαν, διάστημα

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει χώρος, διάστημα, στιγμιότυπο οθόνης, Σύμπαν

Περιγραφή που δημιουργήθηκε αυτόματα

Β

Α

Εικόνα 37 Gameplay interface σε single player mode (A) και σε multiplayer (B)

**Προσβασιμότητα και ευκολία χρήσης**

Το παιχνίδι χρησιμοποιεί διαισθητικά και απλά χειριστήρια πληκτρολογίου για την κίνηση και τη σκοποβολή, τα οποία είναι εύκολο να μάθουν και να χρησιμοποιήσουν οι παίκτες. Επίσης, η διαθεσιμότητα διαφορετικών επιπέδων δυσκολίας επιτρέπει στο παιχνίδι να είναι προσβάσιμο σε ένα ευρύτερο φάσμα παικτών, από αρχάριους έως έμπειρους. Καθώς το σκορ του παίκτη αυξάνεται, το παιχνίδι μπορεί να εισάγει περισσότερους εχθρούς ή ταχύτερους εχθρούς για να διατηρήσει το επίπεδο της πρόκλησης σύμφωνο με τις βελτιούμενες δεξιότητες του παίκτη.

Ο οπτικός σχεδιασμός έγινε με προσοχή, ώστε το οπτικό θέμα να συνάδει με το είδος των διαστημικών shooter arcade games, με σκούρα φόντα που αντιπροσωπεύουν το διάστημα και έντονα χρωματιστά sprites για τα πλοία, τα λέιζερ και τις εκρήξεις. Αυτή η αντίθεση δεν είναι μόνο αισθητικά ευχάριστη, αλλά βοηθά επίσης στη διάκριση μεταξύ διαφορετικών στοιχείων κατά τη διάρκεια του παιχνιδιού.

**Παράρτημα**

import datetime

import pygame

import random

import os

from Sqlite3 import \*

# Initialize Pygame

pygame.init()

# Constants

WIDTH, HEIGHT = 800, 600

FPS = 60

# Standard RGB colors

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

CYAN = (0, 100, 100)

YELLOW = (255, 255, 0)

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

BRIGHT\_RED = (255,51,51)

BRIGHT\_GREEN = (51,204,51)

# Paths

ASSETS\_PATH = os.path.join(os.path.dirname(\_\_file\_\_), 'Assets')

# Create the game window

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Galaxy Shooter")

# Load main menu background image

main\_menu\_background\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'main.png'))

main\_menu\_background\_img = pygame.transform.scale(main\_menu\_background\_img, (WIDTH, HEIGHT))

# Load background image

background\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'back.png'))

background\_img = pygame.transform.scale(background\_img, (WIDTH, HEIGHT))

# Load enemy images

enemy\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'Enemy60.png'))

exploded\_enemy\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'ExplodedEnemy30.png'))

# Load second enemy images

meteo\_enemy\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'MeteoEnemy60.png'))

exploded\_meteo\_enemy\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'ExplodedMeteoEnemy60.png'))

# Load bullet image

bullet\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'Laser.png'))

# Load power-up image

powerup\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'Powerup\_30.png'))

# Load tripple\_laser image

tripple\_laser = pygame.image.load(os.path.join(ASSETS\_PATH, 'Tripple\_Laser.png'))

# Load background music

pygame.mixer.music.load(os.path.join(ASSETS\_PATH, 'soundtrack.mp3'))

# Play the music, loop indefinitely

pygame.mixer.music.play(-1)

# Set the volume to half

pygame.mixer.music.set\_volume(0.5)

# Load sound effects for hits

enemy\_hit\_sound = pygame.mixer.Sound('Assets/enemy\_hit.wav')

player\_hit\_sound = pygame.mixer.Sound('Assets/player\_hit.wav')

# Enemy modes

ENEMY\_NORMAL = 'normal'

ENEMY\_EXPLODING = 'exploding'

ENEMY\_DESTROYED = 'destroyed'

# Initial Enemy mode

Enemy\_mode = ENEMY\_NORMAL

# Second Enemy modes

SECOND\_ENEMY\_NORMAL = 'normal'

SECOND\_ENEMY\_EXPLODING = 'exploding'

SECOND\_ENEMY\_DESTROYED = 'destroyed'

# Initial Second Enemy mode

SecondEnemy\_mode = SECOND\_ENEMY\_NORMAL

# Initialize Game Score

score = 0

# Initialize the Game Difficulty

game\_difficulty = "Medium"

#  Enemy Speed

enemy\_speed = 1.0

# Flag for single or Multi Player

# True for SinglePlayer and false for multiplayer

game\_Flag = True

# Get current time

current\_Time = datetime.datetime.now()

# Create clock for controlling the frame rate

clock = pygame.time.Clock()

# Create sprite groups for enemies, bullets, and second enemies

all\_sprites = pygame.sprite.Group()

enemies = pygame.sprite.Group()

bullets = pygame.sprite.Group()

second\_enemies = pygame.sprite.Group()

powerups = pygame.sprite.Group()

players = pygame.sprite.Group()

# Global Variable Pause

pause = False

# Define a Player Class

class Player(pygame.sprite.Sprite):

    def \_\_init\_\_(self, player\_number=1, is\_multiplayer=False):

        super().\_\_init\_\_()

        #self.all\_sprites = pygame.sprite.Group()

        #self.bullets = pygame.sprite.Group()

        self.player\_number = player\_number

        self.is\_multiplayer = is\_multiplayer

        # Load images

        self.load\_images()

        self.image = self.normal\_image

        self.rect = self.image.get\_rect()

        # Positioning

        self.set\_initial\_position()

        # Speed and shooting delay

        self.speed = 5

        self.shoot\_delay = 250

        self.last\_shot = pygame.time.get\_ticks()

        # Players lives

        self.lives = 3

        # Initialize the power-up related attributes

        self.powerup = None  # No power-up initially

        self.powerup\_time = 0  # Time when the power-up was activated

    def load\_images(self):

        self.normal\_image = pygame.image.load(os.path.join(ASSETS\_PATH, f'Spaceship80{"\_2" if self.player\_number == 2 else ""}.png')).convert\_alpha()

        self.moving\_image = pygame.image.load(os.path.join(ASSETS\_PATH, f'MSpaceship80{"\_2" if self.player\_number == 2 else ""}.png')).convert\_alpha()

        self.exploded\_image = pygame.image.load(os.path.join(ASSETS\_PATH, f'ExplodedSpaceship80{"\_2" if self.player\_number == 2 else ""}.png')).convert\_alpha()

    def set\_initial\_position(self):

        if self.is\_multiplayer:

            position = WIDTH // 4 if self.player\_number == 1 else WIDTH \* 3 // 4

        else:

            position = WIDTH // 2

        self.rect.midbottom = (position, HEIGHT - 10)

    def update(self, keys=None):

        if keys is None:

            keys = {}  # Set to an empty dictionary if no keys argument is provided

        # Movement for Player 1

        if self.player\_number == 1:

            if keys[pygame.K\_LEFT]:

                self.rect.x -= self.speed

            if keys[pygame.K\_RIGHT]:

                self.rect.x += self.speed

            # Image update based on movement for Player 1

            if keys[pygame.K\_LEFT] or keys[pygame.K\_RIGHT]:

                self.image = self.moving\_image

            else:

                self.image = self.normal\_image

        # Movement for Player 2

        elif self.player\_number == 2:

            if keys[pygame.K\_a]:

                self.rect.x -= self.speed

            if keys[pygame.K\_d]:

                self.rect.x += self.speed

            # Image update based on movement for Player 2

            if keys[pygame.K\_a] or keys[pygame.K\_d]:

                self.image = self.moving\_image

            else:

                self.image = self.normal\_image

        # Ensure the player stays within the screen bounds

        self.rect.x = max(0, min(WIDTH - self.rect.width, self.rect.x))

        # Call try\_shoot if the shoot key is pressed

        self.try\_shoot(keys)

    def try\_shoot(self, keys):

        # Check if it's time to shoot based on the player number and key pressed

        can\_shoot = (self.player\_number == 1 and keys[pygame.K\_SPACE]) or \

                    (self.player\_number == 2 and keys[pygame.K\_LCTRL])

        if can\_shoot:

            self.shoot()

    def shoot(self):

        # Debug print to confirm shoot is triggered

        #print(f"shoot method triggered for Player {self.player\_number}")

        now = pygame.time.get\_ticks()

        #Tripple shooting

        if self.powerup == 'tripple\_laser' and now - self.last\_shot > self.shoot\_delay:

            # Create three triple laser bullets or a single powerful triple laser shot

            bullet1 = TripleLaserBullet(self.rect.centerx - 20, self.rect.top)

            bullet2 = TripleLaserBullet(self.rect.centerx, self.rect.top)

            bullet3 = TripleLaserBullet(self.rect.centerx + 20, self.rect.top)

            all\_sprites.add(bullet1, bullet2, bullet3)

            bullets.add(bullet1, bullet2, bullet3)

            self.last\_shot = now

        #Regular shooting

        elif now - self.last\_shot > self.shoot\_delay:

            self.last\_shot = now

            # Temporarily bypass the shooting delay to test bullet creation and display

            #print("Shooting delay bypassed for testing.")

            # Create a new bullet instance

            print("Shooting")  # Debug print

            bullet = Bullet(self.rect.centerx, self.rect.top)

            # Debug print to confirm bullet creation

            #print(f"Bullet created for Player {self.player\_number}")

            # Add the new bullet to the all\_sprites and bullets groups to be updated and drawn

            all\_sprites.add(bullet)

            bullets.add(bullet)

            print("Bullet created and added to groups.")

        self.last\_shot = now

       # Create player objects

       # player = Player(player\_number=1, is\_multiplayer=False)

       # player1 = Player(player\_number=1, is\_multiplayer=True)

       # player2 = Player(player\_number=2, is\_multiplayer=True)

# Enemy class

class Enemy(pygame.sprite.Sprite):

    global enemy\_speed

    def \_\_init\_\_(self, image, exploded\_image, speed\_factor=enemy\_speed, explosion\_duration=500):

        super().\_\_init\_\_()

        self.image = image

        self.exploded\_image = exploded\_image

        self.rect = self.image.get\_rect()

        self.rect.x = random.randrange(WIDTH - self.rect.width)

        self.rect.y = random.randrange(-100, -40)

        self.speed\_y = random.randrange(1, 2) \* speed\_factor

        self.mode = ENEMY\_NORMAL

        self.explosion\_timer = 0

        self.explosion\_duration = explosion\_duration

    def update(self, \*args, \*\*kwargs):

        if self.mode == ENEMY\_NORMAL:

            self.rect.y += self.speed\_y

            if self.rect.top > HEIGHT + 10:

                self.rect.topleft = (random.randrange(WIDTH - self.rect.width), random.randrange(-100, -40))

                self.speed\_y = random.randrange(1, 2) \* self.speed\_y

        elif self.mode == ENEMY\_EXPLODING:

            # Handle exploding animation logic

            self.explosion\_timer += 1

            if self.explosion\_timer >= self.explosion\_duration:

                self.mode = ENEMY\_DESTROYED

                self.explosion\_timer = 0

                # Reset the image to the non-exploded state

                self.image = self.exploded\_image

        elif self.mode == ENEMY\_DESTROYED:

            # Handle logic for destroyed state

            pass

    def hit\_by\_bullet(self):

        # Perform actions when the enemy is hit by a bullet

        self.mode = ENEMY\_EXPLODING

        # Change the image of the sprite directly

        self.image = self.exploded\_image

# Second Enemy class

class SecondEnemy(pygame.sprite.Sprite):

    def \_\_init\_\_(self, image, exploded\_image, speed\_factor=enemy\_speed, explosion\_duration=60):

        super().\_\_init\_\_()

        self.image = image

        self.exploded\_image = exploded\_image

        self.rect = self.image.get\_rect()

        self.rect.x = random.randrange(WIDTH - self.rect.width)

        self.rect.y = random.randrange(-100, -40)

        self.speed\_y = random.uniform(1.3, 2.5) \* speed\_factor

        self.mode = SECOND\_ENEMY\_NORMAL

        self.explosion\_timer = 0

        self.explosion\_duration = explosion\_duration

    def update(self, \*args, \*\*kwargs):

        if self.mode == SECOND\_ENEMY\_NORMAL:

            self.rect.y += self.speed\_y

            # Ensure the speed does not exceed a maximum value

            max\_speed\_y = 10  #  maximum speed value

            self.speed\_y = min(self.speed\_y, max\_speed\_y)

            if self.rect.top > HEIGHT + 10:

                self.rect.topleft = (random.randrange(WIDTH - self.rect.width), random.randrange(-100, -40))

                # Reset speed to a base value instead of increasing it

                self.speed\_y = random.uniform(1.3, 2.5)  #  base speed range

        elif self.mode == SECOND\_ENEMY\_EXPLODING:

            # Handle exploding animation logic

            self.explosion\_timer += 1

            if self.explosion\_timer >= self.explosion\_duration:

                self.mode = SECOND\_ENEMY\_DESTROYED

                self.explosion\_timer = 0

        elif self.mode == SECOND\_ENEMY\_DESTROYED:

            # Handle logic for destroyed state

            pass

    def hit\_by\_bullet(self):

        # Perform actions when the second enemy is hit by a bullet

        self.mode = SECOND\_ENEMY\_EXPLODING

        self.image = self.exploded\_image

# Bullet class

class Bullet(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.image = bullet\_img

        self.rect = self.image.get\_rect()

        self.rect.centerx = x

        self.rect.bottom = y

        self.speed\_y = -10

#        self.rect.y -= self.speed\_y  # Move the bullet up

    def update(self, \*args, \*\*kwargs):

        self.rect.y += self.speed\_y

        # Remove the bullet if it goes off the screen

        if self.rect.bottom < 0:

            self.kill()

# PowerUp class

class PowerUp(pygame.sprite.Sprite):

    def \_\_init\_\_(self, img, type, \*groups):

        super().\_\_init\_\_(\*groups)

        self.image = img

        self.rect = self.image.get\_rect()

        self.rect.x = random.randrange(WIDTH - self.rect.width)

        self.rect.y = random.randrange(-100, -40)

        self.speed\_y = random.randrange(1, 4)

        self.type = type  # "tripple\_laser" for this case

    def update(self):

        self.rect.y += self.speed\_y

        if self.rect.top > HEIGHT:

            self.kill()

# TripleLaserBullet class

class TripleLaserBullet(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.image = pygame.image.load(os.path.join(ASSETS\_PATH, 'Tripple\_Laser.png'))

        self.rect = self.image.get\_rect()

        self.rect.centerx = x

        self.rect.bottom = y

        self.speed\_y = -10

    def update(self):

        self.rect.y += self.speed\_y

        if self.rect.bottom < 0:

            self.kill()

# Function to display score

def display\_score(score):

    font = pygame.font.Font(None, 36)

    score\_text = font.render("Score: {}".format(score), True, (255, 255, 0))  # Yellow color

    screen.blit(score\_text, (WIDTH - 150, HEIGHT - 50))

#Options screen function

def show\_options\_screen():

    global game\_difficulty

    options\_running = True

    difficulty\_levels = ["Easy", "Medium", "Hard"]

    selected\_difficulty = "Medium"  # Default selection

    # Font settings

    font = pygame.font.Font(None, 48)

    title\_font = pygame.font.Font(None, 74)

    # Calculate button positions and sizes

    button\_width = 200

    button\_height = 50

    button\_margin = 20  # Space between buttons

    start\_y = 150  # Starting Y position for the first button

    # Generate a list of Rects for each difficulty level

    difficulty\_buttons = [pygame.Rect(WIDTH // 2 - button\_width // 2, start\_y + (button\_height + button\_margin) \* i, button\_width, button\_height) for i in range(len(difficulty\_levels))]

    # Return button setup

    return\_rect = pygame.Rect(WIDTH - 220, HEIGHT - 70, 200, 50)

    while options\_running:

        screen.fill(CYAN)  # Background color

        mouse\_pos = pygame.mouse.get\_pos()

        # Draw the "Options" title

        options\_title = title\_font.render("Options", True, WHITE)

        screen.blit(options\_title, (WIDTH // 2 - options\_title.get\_rect().width // 2, 50))

        # Draw and interact with the difficulty buttons

        for i, rect in enumerate(difficulty\_buttons):

            level = difficulty\_levels[i]

            pygame.draw.rect(screen, YELLOW if level == selected\_difficulty else WHITE, rect)  # Draw button

            text = font.render(level, True, BLACK)  # Black text on the button

            text\_rect = text.get\_rect(center=rect.center)

            screen.blit(text, text\_rect)

            if rect.collidepoint(mouse\_pos) and pygame.mouse.get\_pressed()[0]:

                selected\_difficulty = level  # Update the selected difficulty

                game\_difficulty = selected\_difficulty

        # Draw the "Return to Main Menu" button

        pygame.draw.rect(screen, YELLOW if return\_rect.collidepoint(mouse\_pos) else WHITE, return\_rect)  # Button shape

        return\_text = font.render("Return", True, BLACK)  # Black text on the button

        return\_text\_rect = return\_text.get\_rect(center=return\_rect.center)

        screen.blit(return\_text, return\_text\_rect)

        # Event handling

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                options\_running = False

                pygame.quit()

                quit()

            elif event.type == pygame.MOUSEBUTTONDOWN and return\_rect.collidepoint(mouse\_pos):

                options\_running = False  # Exit options screen

        pygame.display.flip()

        clock.tick(FPS)

def show\_highScore\_screen():

    global game\_difficulty

    options\_running = True

    # Font settings

    font = pygame.font.Font(None, 48)

    title\_font = pygame.font.Font(None, 74)

    # Return button setup

    return\_rect = pygame.Rect(WIDTH - 220, HEIGHT - 70, 200, 50)

    while options\_running:

        screen.fill(CYAN)  # Background color

        mouse\_pos = pygame.mouse.get\_pos()

        # Draw the "Options" title

        options\_title = title\_font.render("Options", True, WHITE)

        screen.blit(options\_title, (WIDTH // 2 - options\_title.get\_rect().width // 2, 50))

        # Display the top 3 scores dynamically using get\_High\_Score function

        high\_scores = get\_High\_Score()

        text\_positions = [(50, 250), (50, 350), (50, 450)]

        for i, score\_data in enumerate(high\_scores):

            text = f"{score\_data[1]} - Score: {score\_data[2]}, Time: {score\_data[3]}"

            text\_surface = font.render(text, True, BLACK)

            text\_rect = text\_surface.get\_rect(topleft=text\_positions[i])

            screen.blit(text\_surface, text\_rect)

        # Draw the "Return to Main Menu" button

        pygame.draw.rect(screen, YELLOW if return\_rect.collidepoint(mouse\_pos) else WHITE, return\_rect)  # Button shape

        return\_text = font.render("Return", True, BLACK)  # Black text on the button

        return\_text\_rect = return\_text.get\_rect(center=return\_rect.center)

        screen.blit(return\_text, return\_text\_rect)

        # Event handling

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                options\_running = False

                pygame.quit()

                quit()

            elif event.type == pygame.MOUSEBUTTONDOWN and return\_rect.collidepoint(mouse\_pos):

                options\_running = False  # Exit options screen

        pygame.display.flip()

        clock.tick(FPS)

# Main menu function

def main\_menu():

    global clock  # Use the global clock variable

    global game\_Flag

    font = pygame.font.Font(None, 42)

    title\_text = font.render("Game Play", True, (255, 255, 0))  # Yellow color

    # Define colors for button states

    button\_color = YELLOW

    button\_hover\_color = BLUE

    # Define button dimensions and positions

    button\_width = 200

    button\_height = 50

    gap = 20  # Vertical gap between buttons

    start\_y = HEIGHT // 3  # Starting Y position of the first button

    # Define button rectangles

    single\_player\_rect = pygame.Rect(WIDTH // 2 - button\_width // 2, start\_y, button\_width, button\_height)

    multiplayer\_rect = pygame.Rect(WIDTH // 2 - button\_width // 2, start\_y + button\_height + gap, button\_width, button\_height)

    highScore\_rect =  pygame.Rect(WIDTH // 2 - button\_width // 2, start\_y + 2 \* (button\_height + gap), button\_width, button\_height)

    options\_rect = pygame.Rect(WIDTH // 2 - button\_width // 2, start\_y + 3 \* (button\_height + gap), button\_width, button\_height)

    exit\_rect = pygame.Rect(WIDTH // 2 - button\_width // 2, start\_y + 4 \* (button\_height + gap), button\_width, button\_height)

    menu\_running = True

    while menu\_running:

        screen.blit(main\_menu\_background\_img, (0, 0))

        screen.blit(title\_text, (WIDTH // 2 - title\_text.get\_width() // 2, 20))

        mouse\_pos = pygame.mouse.get\_pos()

        # Draw buttons and check for mouse hover to change color

        for rect in [single\_player\_rect, multiplayer\_rect,  highScore\_rect,options\_rect, exit\_rect]:

            pygame.draw.rect(screen, button\_hover\_color if rect.collidepoint(mouse\_pos) else button\_color, rect)

        # Draw button text

        screen.blit(font.render("Single Player", True, BLACK), (single\_player\_rect.x + 5, single\_player\_rect.y + 5))

        screen.blit(font.render("Multiplayer", True, BLACK), (multiplayer\_rect.x + 5, multiplayer\_rect.y + 5))

        screen.blit(font.render("High Score", True, BLACK), (highScore\_rect.x + 5, highScore\_rect.y + 5))

        screen.blit(font.render("Options", True, BLACK), (options\_rect.x + 5, options\_rect.y + 5))

        screen.blit(font.render("Exit", True, BLACK), (exit\_rect.x + 5, exit\_rect.y + 5))

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                menu\_running = False

                pygame.quit()

                quit()

            elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:  # Left click

                if single\_player\_rect.collidepoint(mouse\_pos):

                    game\_Flag = True

                    start\_game(single\_player=True)

                    menu\_running = False

                elif multiplayer\_rect.collidepoint(mouse\_pos):

                    game\_Flag = False

                    start\_game(single\_player=False)

                    menu\_running = False

                elif highScore\_rect.collidepoint(mouse\_pos):

                    show\_highScore\_screen()

                elif options\_rect.collidepoint(mouse\_pos):

                    show\_options\_screen()  # Make sure this function returns to main\_menu or handles closing properly

                elif exit\_rect.collidepoint(mouse\_pos):

                    pygame.quit()

                    quit()

        pygame.display.flip()

        clock.tick(FPS)

# Game over screen function

def game\_over\_screen():

    global clock  # Use the global clock variable

    global game\_Flag

    # Load the main menu background image

    game\_over\_background\_img = pygame.image.load(os.path.join(ASSETS\_PATH, 'main.png'))

    game\_over\_background\_img = pygame.transform.scale(game\_over\_background\_img, (WIDTH, HEIGHT))

    font = pygame.font.Font(None, 42)

    # Define button dimensions and positions

    button\_width = 200

    button\_height = 50

    button\_gap = 30  # Gap between buttons

    # Define colors

    button\_color = YELLOW

    button\_hover\_color = BLUE

    # Create rectangles for buttons

    restart\_button\_rect = pygame.Rect(WIDTH // 2 - button\_width // 2, HEIGHT // 2 - button\_height // 2 - button\_gap, button\_width, button\_height)

    exit\_button\_rect = pygame.Rect(WIDTH // 2 - button\_width // 2, HEIGHT // 2 - button\_height // 2 + button\_gap, button\_width, button\_height)

    game\_over\_running = True

    while game\_over\_running:

        screen.blit(game\_over\_background\_img, (0, 0))

        mouse\_pos = pygame.mouse.get\_pos()

        # Draw "You Lost!" text

        game\_over\_text = font.render("You Lost!", True, (255, 0, 0))

        game\_over\_text\_rect = game\_over\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 3 \* button\_gap))

        screen.blit(game\_over\_text, game\_over\_text\_rect)

        # Draw buttons

        pygame.draw.rect(screen, button\_hover\_color if restart\_button\_rect.collidepoint(mouse\_pos) else button\_color, restart\_button\_rect)

        pygame.draw.rect(screen, button\_hover\_color if exit\_button\_rect.collidepoint(mouse\_pos) else button\_color, exit\_button\_rect)

        # Draw button texts

        restart\_text = font.render("Restart", True, BLACK)

        exit\_text = font.render("Exit", True, BLACK)

        screen.blit(restart\_text, (restart\_button\_rect.x + 20, restart\_button\_rect.y + 5))

        screen.blit(exit\_text, (exit\_button\_rect.x + 50, exit\_button\_rect.y + 5))

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

            elif event.type == pygame.MOUSEBUTTONDOWN:

                if restart\_button\_rect.collidepoint(mouse\_pos):

                    start\_game(single\_player=game\_Flag)

                    game\_over\_running = False

                elif exit\_button\_rect.collidepoint(mouse\_pos):

                    main\_menu()

                    game\_over\_running = False

        pygame.display.flip()

        clock.tick(FPS)

# Game loop

def start\_game(single\_player=True, difficulty='medium'):

    global score, all\_sprites, enemies, bullets, second\_enemies, players,game\_difficulty,enemy\_speed

    print("Game difficulty is = " ,game\_difficulty)

    if(game\_difficulty == "Medium"):

        enemy\_speed = 1.0

    elif(game\_difficulty == "Easy"):

        enemy\_speed = 0.7

    else:

        enemy\_speed = 1.5

    print("Enemy speed is = " ,enemy\_speed)

    # Reset the score

    score = 0

    # Reset the score

    score = 0

    second\_enemy\_counter = 0

    # Set the threshold for second enemy appearance

    second\_enemy\_threshold = 200

    # Initialize sprite groups

    all\_sprites = pygame.sprite.Group()

    enemies = pygame.sprite.Group()

    bullets = pygame.sprite.Group()

    second\_enemies = pygame.sprite.Group()

    # Add the power-up related variables

    power\_up\_active = False

    power\_up\_start\_time = 0

    # Clear all existing sprites from previous games

    all\_sprites.empty()

    enemies.empty()

    bullets.empty()

    second\_enemies.empty()

    players.empty()

    # Create player instances based on the game mode

    if single\_player:

        player = Player(player\_number=1)

        all\_sprites.add(player)

        #players = pygame.sprite.Group()

        players.add(player)

        all\_sprites.add(player)

    else:

        player1 = Player(player\_number=1, is\_multiplayer=True)

        player2 = Player(player\_number=2, is\_multiplayer=True)

        players.add(player1)  # Add each player to the players group

        players.add(player2)

        all\_sprites.add(player1)

        all\_sprites.add(player2)

    # Keys pressed for movement and shooting

    keys = pygame.key.get\_pressed()

    # In multiplayer: check and call shoot for both players separately

    if not single\_player:

        # Check for Player 1 shooting

        if keys[pygame.K\_SPACE]:

            player1.shoot()

        # Check for Player 2 shooting

        if keys[pygame.K\_LCTRL]:

            player2.shoot()

    # Update players

    if single\_player:

        player.update(keys)

    else:

        player1.update(keys)

        player2.update(keys)

    # Create enemy instances

    for \_ in range(9):

        enemy = Enemy(enemy\_img, exploded\_enemy\_img)

        all\_sprites.add(enemy)

        enemies.add(enemy)

    # Maximum number of active second enemies

    max\_active\_second\_enemies = 3

    # Counter for second enemy appearance

    second\_enemy\_counter = 0

    # Clock for controlling the frame rate

    clock = pygame.time.Clock()

    # Game loop

    running = True

    while running:

        # Process events

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

    # Update the current state of the keyboard

        keys = pygame.key.get\_pressed()

        # Update player sprites with keys

        for player in players:

            player.update(keys) # Update player position based on keys

#            player.try\_shoot(keys) # Check if the player is trying to shoot

        # Update the sprites

        for sprite in all\_sprites:

            if sprite not in players:

                sprite.update()

        # Draw screen

        screen.blit(background\_img, (0, 0))

        # Draw the sprites

        all\_sprites.draw(screen)

        # Increment the second enemy counter when the first enemy is shot

        second\_enemy\_counter += 1

        # Check if the score exceeds the threshold for second enemy appearance

        if score >= second\_enemy\_threshold:

            # Check if it's time to spawn the second type of enemy

            if second\_enemy\_counter >= 20:

            # Count the currently active second enemies

                active\_second\_enemies = len(second\_enemies.sprites())

            # Spawn a new second enemy only if the limit is not reached

            if active\_second\_enemies < max\_active\_second\_enemies:

                second\_enemy = SecondEnemy(meteo\_enemy\_img, exploded\_meteo\_enemy\_img, speed\_factor=1.3)

                all\_sprites.add(second\_enemy)

                second\_enemies.add(second\_enemy)

            # Spawning the power-up based on score and randomness

            if score >= 250 and random.random() < 0.005:  # Adjust 0.005 to control frequency

                powerup = PowerUp(powerup\_img, 'tripple\_laser')

                all\_sprites.add(powerup)

                powerups.add(powerup)

        # Check for collisions between bullets and enemies

        hits = pygame.sprite.groupcollide(enemies, bullets, True, True)

        for hit in hits:

            # Play sound effect for enemy hit

            enemy\_hit\_sound.play()

            # Add scoring logic for the first enemy

            score += 10

            # Call the hit\_by\_bullet method for each hit enemy

            hit.hit\_by\_bullet()

            # Create a new enemy when one is shot

            enemy = Enemy(enemy\_img, exploded\_enemy\_img)

            all\_sprites.add(enemy)

            enemies.add(enemy)

        # Check for collisions between bullets and second enemies

        second\_enemy\_hits = pygame.sprite.groupcollide(second\_enemies, bullets, True, True)

        for hit in second\_enemy\_hits:

            # Play sound effect for enemy hit

            enemy\_hit\_sound.play()

            # Add scoring logic for the second enemy

            score += 20

            # Call the hit\_by\_bullet method for each hit second enemy

            hit.hit\_by\_bullet()

            # Create a new second enemy when one is shot

            second\_enemy = SecondEnemy(meteo\_enemy\_img, exploded\_meteo\_enemy\_img, speed\_factor=1.3)

            all\_sprites.add(second\_enemy)

            second\_enemies.add(second\_enemy)

        if single\_player:

            # Check for collision with power-ups

            collisions = pygame.sprite.spritecollide(player, powerups, True)

            for collision in collisions:

                if collision.type == 'tripple\_laser':

                    player.powerup = 'tripple\_laser'

                    player.powerup\_time = pygame.time.get\_ticks()   # Capture the start time of the power-up effect

        else:

            # Check for collision with power-ups

            collisions1 = pygame.sprite.spritecollide(player1, powerups, True)

            collisions2 = pygame.sprite.spritecollide(player2, powerups, True)

            collisions = collisions1,collisions2

            for i in collisions:

                for collision in i:

                    if collision.type == 'tripple\_laser':

                        player1.powerup = 'tripple\_laser'

                        player1.powerup\_time = pygame.time.get\_ticks()   # Capture the start time of the power-up effect

                        player2.powerup = 'tripple\_laser'

                        player2.powerup\_time = pygame.time.get\_ticks()   # Capture the start time of the power-up effect

        # Implement the Power-up Timer Logic

        if player.powerup == 'tripple\_laser':

            if pygame.time.get\_ticks() - player.powerup\_time > 10000:  # 10 seconds duration

                player.powerup = None  # Revert to normal bullets

        # Check for collisions between the player and the first type of enemies

        if single\_player:

            # Single-player collision detection

            player\_hits = pygame.sprite.spritecollide(player, enemies, True)

            if player\_hits:

                insert\_Values("Player",score,str(current\_Time.strftime("%Y-%m-%d %H:%M:%S")))

                # Play sound effect for player hit

                player\_hit\_sound.play()

                game\_over\_screen()

                running = False

        else:

            # Multiplayer collision detection

            player1\_hits = pygame.sprite.spritecollide(player1, enemies, True)

            player2\_hits = pygame.sprite.spritecollide(player2, enemies, True)

            if player1\_hits or player2\_hits:

                # Enters the

                insert\_Values("Player",score,str(current\_Time.strftime("%Y-%m-%d %H:%M:%S")))

                # Play sound effect for player hit

                player\_hit\_sound.play()

                game\_over\_screen()

                running = False

        # Check for collisions between the player and the second type of enemies

        if single\_player:

            # Single-player collision detection

            player\_hits = pygame.sprite.spritecollide(player, second\_enemies, True)

            if player\_hits:

                insert\_Values("Player",score,str(current\_Time.strftime("%Y-%m-%d %H:%M:%S")))

                # Play sound effect for player hit

                player\_hit\_sound.play()

                game\_over\_screen()

                running = False

        else:

            # Multiplayer collision detection

            player1\_hits = pygame.sprite.spritecollide(player1, second\_enemies, True)

            player2\_hits = pygame.sprite.spritecollide(player2, second\_enemies, True)

            if player1\_hits or player2\_hits:

                insert\_Values("Player",score,str(current\_Time.strftime("%Y-%m-%d %H:%M:%S")))

                # Play sound effect for player hit

                player\_hit\_sound.play()

                game\_over\_screen()

                running = False

        # Display score

        display\_score(score)

        # Refresh the display

        pygame.display.flip()

        # Cap the frame rate

        clock.tick(FPS)

    # Quit Pygame when the game loop exits

    pygame.quit()

# Placeholder function for showing options

def show\_options():

    print("Options function placeholder")

# Run the main menu

main\_menu()