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

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.

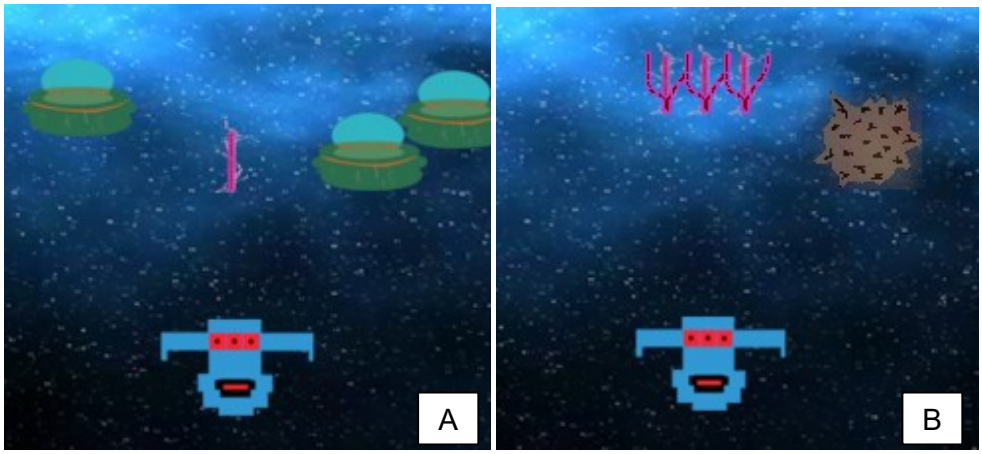
Στην Εικόνα 1 φαίνεται το σχετικό στιγμιότυπο του κώδικα για τον ορισμό του μηχανισμού πυροβολισμού, όπου ορίζεται η συμπεριφορά για τον πυροβολισμό χωρίς και με το power-up (στην απλή – κανονική βολή αποδεσμεύεται μία σφαίρα η οποία έχει μια απλή μορφή laser beam, ενώ στην βολή μετά το power-up αποδεσμεύονται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

```python
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
```

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.



Εικόνα 2 Γραφική απεικόνιση απλής βολής (A) 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.

```python
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
```

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.

```python
# 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()
```

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.

```python
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 th
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

                player2.powerup = 'tripple_laser'
                player2.powerup_time = pygame.time.get_ticks()   # Capture the start time
```

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.

```
# 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()
```

Εικόνα 7 Δημιουργία των instances των παικτών(κώδικας Python)

```python
# 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)
```

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 και αυξάνετε ο αριθμός των εχθρών κάθε ένα λεπτό, 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).

```
enemy_spawn_counter = 9
# Get the current time
current_time = pygame.time.get_ticks()
# Create enemy instances
for _ in range(enemy_spawn_counter):
    enemy = Enemy(enemy_img, exploded_enemy_img)
    all_sprites.add(enemy)
    enemies.add(enemy)
    if current_time - start_time >= delay_time:
        enemy_spawn_counter += 1
        print("Enemy frequency increased")
```

Εικόνα 9 Δημιουργία των instances των εχθρών (κώδικας Python)

```
# 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=
        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)
```

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.

```python
# 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
```

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).

```python
# 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)
```

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.

```python
# 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)
```

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).

```
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()
```

Εικόνα 14 Οπτική αναπαράσταση παίκτη

```
# Speed and shooting delay
self.speed = 5
self.shoot_delay = 250
self.last_shot = pygame.time.get_ticks()
```

Εικόνα 15 Μηχανική του gameplay

```
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
```

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).

```python
# 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'))
```

Εικόνα 17 Τύποι εχθρών

```python
# 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
```

Εικόνα 18 Κλάση και συμπεριφορά βασικού εχθρού

```python
# 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
```

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).

```python
# 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
```

Εικόνα 20 Σκορ από την εξουδετέρωση του βασικού εχθρού

```
# 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
```

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).

```python
# 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)
```

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).

```
# 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
```

Εικόνα 23 Συγκρούσεις παικτών και βασικών εχθρών

```
# 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
```

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).

```
def update(self, keys=None):
    if keys is None:
        keys = {}  # Set to an empty dictionary if no keys argument
    # 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))
```

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).

```python
# 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
```

Εικόνα 26 Κίνηση του βασικού εχθρού

```python
# 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
```

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).

```python
# 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))
```

Εικόνα 28 Συνάρτηση εμφάνισης του σκορ

```python
# 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)
```

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 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).

```
# 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()
```

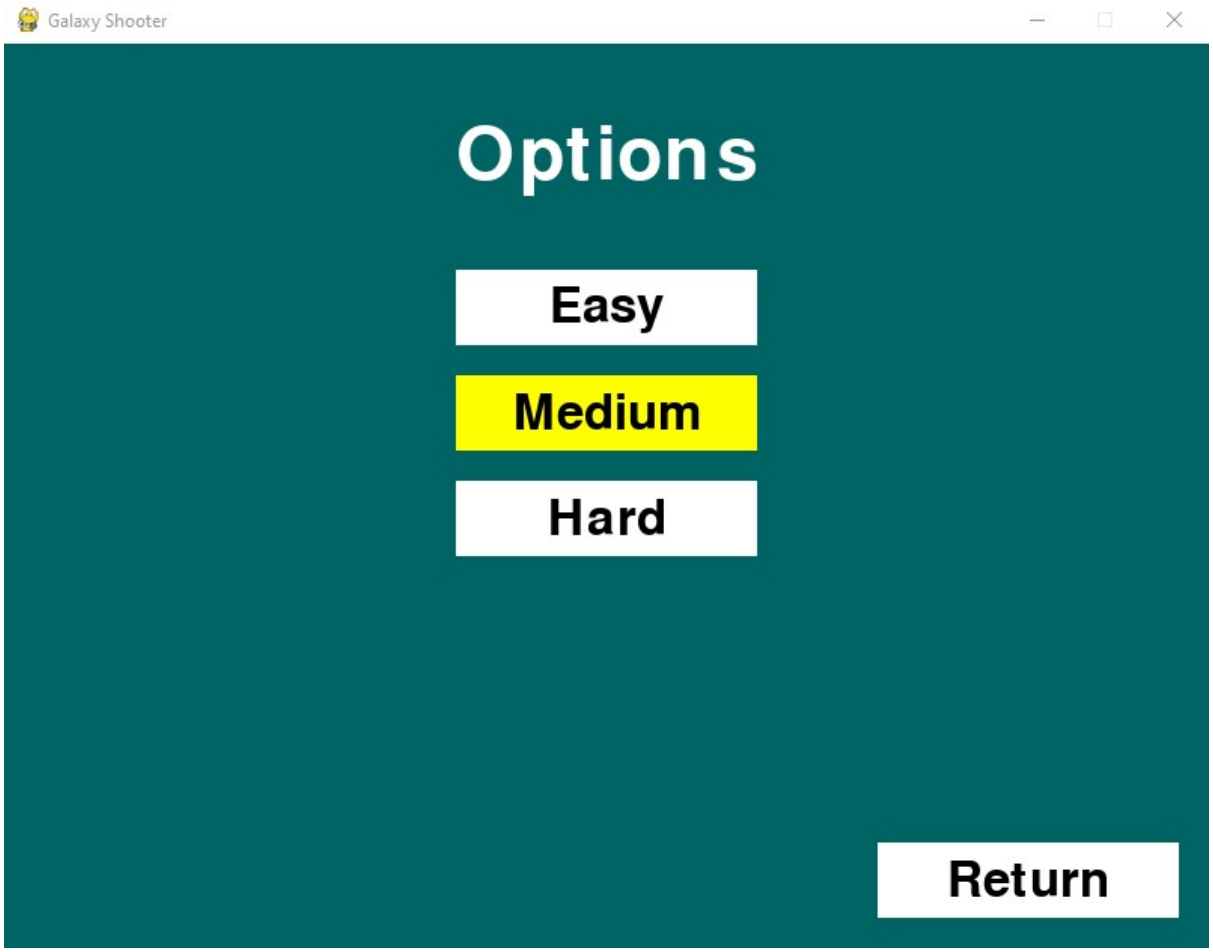
Εικόνα 31 Κώδικας για τη δημιουργία των αντικειμένων των παικτών

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 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).

```python
#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)
```

Εικόνα 33 Ο κώδικας της επιλογής του επιπέδο δυσκολίας

```python
# 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)
```

Εικόνα 34 Εφαρμογή του επιπέδου δυσκολίας

```python
# 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
```

Εικόνα 35 Μεταβλητή enemy_speed για τον βασικό εχθρό

```python
# 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
```

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

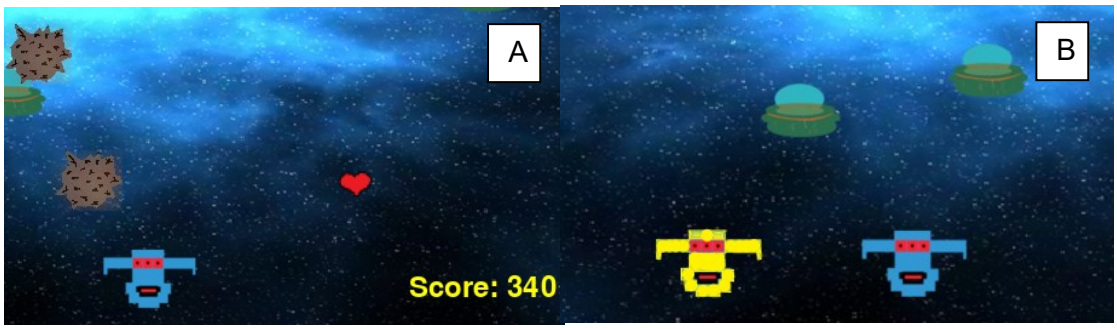
Το κύριο μενού είναι το πρώτο σημείο αλληλεπίδρασης μεταξύ του παίκτη και του παιχνιδιού. Χρησιμεύει ως πύλη εισόδου σε όλα τα χαρακτηριστικά και τους τρόπους λειτουργίας του παιχνιδιού. Το μενού υλοποιείται με τη χρήση κουμπιών με εφέ αιώρησης, που αλλάζουν χρώματα για να παρέχουν οπτική ανατροφοδότηση 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 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()
```