Αλγοριθμική Σκέψη

Κωνσταντίνος Γούλας

A.M. 00054

1. Γενική περιγραφή

Στο πλαίσιο του μαθήματος "Αλγοριθμική Σκέψη" υλοποίησα ένα δισδιάστατο (2D) παιχνίδι χρησιμοποιώντας τη γλώσσα προγραμματισμού Python και τη βιβλιοθήκη Pygame.

Το παιχνίδι είναι ένα δισδιάστατο **Space Shooter** με δυνατότητα επιλογής μεταξύ Single Player και Multiplayer. Στο πλαίσιο της υλοποίησης υπάρχει μια κεντρική σελίδα με τρία κουμπιά: ένα για Single Player, ένα για Multiplayer ως Client και ένα για Multiplayer ως Server.

Στη λειτουργία **Single Player**, το παιχνίδι περιλαμβάνει δύο επίπεδα (levels).

Το πρώτο επίπεδο περιλαμβάνει τέσσερις αντιπάλους και τον παίκτη, ο οποίος ξεκινά με 8 ζωές. Υπάρχουν δύο τρόποι να χάσει ο παίκτης ζωή: είτε αν χτυπηθεί από κάποιον αντίπαλο, είτε αν οι σφαίρες του χτυπήσουν κατά λάθος τον πίνακα που εμφανίζει τις ζωές του. Οι αντίπαλοι κινούνται περιοδικά (δεξιά-αριστερά και πάνω-κάτω) και, ανάλογα με τη κίνηση του παίκτη, τους έχει δοθεί μια πιθανότητα να αποφεύγουν τα χτυπήματα του με κίνηση προς την αντίθετη φορά. Όταν ένας αντίπαλος δεχτεί τέσσερα χτυπήματα και εξοντωθεί, πέφτει προς τον παίκτη σε μια προσπάθεια να τον χτυπήσει. Αν το καταφέρει, ο παίκτης χάνει 2 ζωές.

Το δεύτερο επίπεδο έχει παρόμοια δομή, αλλά με διαφορετικά assets. Σε αυτό το επίπεδο, οι αντίπαλοι κινούνται πιο γρήγορα, αυξάνοντας τη δυσκολία για τον παίκτη να τους πετύχει με τις σφαίρες του.

Το παιχνίδι ολοκληρώνεται όταν ο παίκτης εξοντώσει όλους τους αντιπάλους. Στόχος του παιχνιδιού είναι να επιτευχθεί το υψηλότερο δυνατό σκορ. Το σκορ υπολογίζεται με βάση τρεις παράγοντες: την ποσότητα ζωών που διατηρεί ο παίκτης στο τέλος, τον αριθμό σφαιρών που χρησιμοποίησε για να εξοντώσει τους αντιπάλους και τον χρόνο που χρειάστηκε για να τερματίσει το παιχνίδι.

Η εξίσωση υπολογισμού του σκορ στο παιχνίδι είναι η εξής:
(100 * ζωές) - (σφαίρες * 0.2) - (χρόνος * 0.2).

Μόλις ολοκληρωθεί το παιχνίδι, εμφανίζεται μια τελική οθόνη που ενημερώνει τον παίκτη ότι το παιχνίδι έφτασε στο τέλος (Game Over). Σε αυτήν την οθόνη παρουσιάζονται, επίσης, το σκορ που κατάφερε να πετύχει ο παίκτης στο συγκεκριμένο παιχνίδι αλλά και το υψηλότερο σκορ που έχει επιτευχθεί μέχρι εκείνη τη στιγμή, συμπεριλαμβανομένων και προηγούμενων παιχνιδιών.

Στην ίδια σελίδα υπάρχουν δύο κουμπιά: ένα για τερματισμό του παιχνιδιού και ένα για επανεκκίνηση (Replay).

Στο αρχικό παράθυρο του παιχνιδιού υπάρχει και η επιλογή για πολλαπλούς παίκτες **Multiplayer**. Αν πατηθεί το κουμπί για Multiplayer, ένας παίκτης πρέπει πρώτα να επιλέξει τη λειτουργία **Server** και ο δεύτερος τη λειτουργία **Client**. Στη συνέχεια, και οι δύο παίκτες μεταφέρονται σε ένα νέο παράθυρο παιχνιδιού.

Στο Multiplayer, ο πρώτος παίκτης (κόκκινος) ελέγχει τον αριστερό χαρακτήρα, ενώ ο δεύτερος παίκτης (μπλε) ελέγχει τον δεξιό χαρακτήρα. Το παιχνίδι σε αυτή τη λειτουργία είναι πιο απλό: οι παίκτες προσπαθούν να εξουδετερώσουν ο ένας τον άλλον πυροβολώντας (με το πλήκτρο Space Bar). Κάθε παίκτης έχει 4 ζωές, και ο πρώτος που θα εξαντλήσει τις ζωές του αντιπάλου κερδίζει.

Μετά τη λήξη του παιχνιδιού, εμφανίζεται μια οθόνη που αναγράφει το χρώμα του νικητή.

Τέλος, η κίνηση των χαρακτήρων γίνεται με τα πλήκτρα:

- **Βέλη πάνω και κάτω** για το Multiplayer.
- **Βέλη δεξιά και αριστερά** για το Single Player.
- Το πλήκτρο **Space Bar** χρησιμοποιείται για πυροβολισμούς.

2. Ανάλυση του παιχνιδιού σε επίπεδο κώδικα

2.1 Οι βιβλιοθήκες που χρησιμοποιήθηκαν

Για την κατασκευή του παιχνιδιού χρησιμοποιήθηκαν οι παρακάτω βιβλιοθήκες:

- **Pygame**: Η βασική βιβλιοθήκη πάνω στην οποία χτίστηκε όλο το παιχνίδι.
- **Math**: Η μαθηματική βιβλιοθήκη της Python που χρησιμοποιήθηκε για υπολογισμούς τιμών, όπως το cos(x) και το sin(x).
- **Numpy**: Βασική βιβλιοθήκη της Python για μαθηματικούς υπολογισμούς και για δομές δεδομένων.
- **Random**: Βιβλιοθήκη που χρησιμοποιήθηκε για την παραγωγή τυχαίων αριθμών.
- **Sqlite3**: Για την χρήση βάσεων δεδομένων SQL.
- **Time**: Για τη διαχείριση και μέτρηση του χρόνου στο παιχνίδι.
- **Subprocess**: Για την εκτέλεση πολλαπλών Python scripts.

- **Platform**: Για την αναγνώριση του λειτουργικού συστήματος που χρησιμοποιείται.
- **Socket**: Για τη δημιουργία δικτύου μεταξύ server και clients, υποστηρίζοντας τη λειτουργία Multiplayer.

2.2 Οι βασικές κλάσεις του παιχνιδιού:

Στην υλοποίηση του παιχνιδιού, αρχικά, έχω δύο κλάσεις:

Κλάση: ***ControlState***

Αυτή η κλάση διαχειρίζεται την κατάσταση των ελέγχων του παιχνιδιού, δηλαδή τα πλήκτρα και τις ενέργειες του παίκτη. Περιέχει τις παρακάτω μεταβλητές:

- **Right**: Έλεγχος για το πλήκτρο δεξιά
- **Left**: Έλεγχος για το πλήκτρο αριστερά
- **Up**: Έλεγχος για το πλήκτρο πάνω
- **Down**: Έλεγχος για το πλήκτρο κάτω
- **Space**: Έλεγχος για το πλήκτρο Space (πυροβολισμός)
- **MouseDown**: Κατάσταση του ποντικιού (όταν είναι πατημένο)
- **MouseUp**: Κατάσταση του ποντικιού (όταν αφήνεται)

Στην κλάση περιλαμβάνονται και οι αντίστοιχες μέθοδοι για τη διαχείριση αυτών των μεταβλητών.

Κλάση: ***GameState***

Αυτή η κλάση διαχειρίζεται τα διαφορετικά “στάδια” του παιχνιδιού. Περιέχει τις εξής καταστάσεις:

- **Start_screen**: Σελίδα για την αρχή του παιχνιδιού
- **Multiplayer_screen**: Σελίδα για τη λειτουργία Multiplayer
- **Level1**: Πρώτο επίπεδο του παιχνιδιού
- **Level2**: Δεύτερο επίπεδο του παιχνιδιού
- **End_game**: Οθόνη τερματισμού του παιχνιδιού
- **Exit**: Κατάσταση εξόδου από το παιχνίδι

Και σε αυτή την κλάση υπάρχουν οι αντίστοιχες μέθοδοι που διαχειρίζονται τη μετάβαση και τη λειτουργία αυτών των σταδίων.

Singleton αντικείμενα

Στην αρχή του παιχνιδιού, δημιουργούνται δύο αντικείμενα, ένα για την κλάση ControlState και ένα για την κλάση GameState. Αυτά τα αντικείμενα χρησιμοποιούνται ως singleton αντικείμενα σε όλη τη διάρκεια του παιχνιδιού, διασφαλίζοντας ότι υπάρχει μόνο μία μοναδική κατάσταση ελέγχων και παιχνιδιού.

Κλάση: ***GameWindow***

Αυτή η κλάση αναλαμβάνει τη βασική διαχείριση του παραθύρου και της κύριας λούπας του παιχνιδιού.

- **pygame.init()**: Καλεί τη βιβλιοθήκη Pygame
- **screen = pygame.display.set_mode(dim)**: Ορίζει το παράθυρο του παιχνιδιού με διαστάσεις dim
- **clock = pygame.time.Clock()**: Δημιουργεί έναν χρονομετρητή για τον έλεγχο του ρυθμού ανανέωσης (FPS)
- **end = False**: Ορίζει αν η λούπα του παιχνιδιού έχει ολοκληρωθεί

Μέθοδοι της κλάσης:

1. **OnEvent()**: "Virtual" μέθοδος για τη διαχείριση γεγονότων (events).
2. **OnControl()**: "Virtual" μέθοδος για τον έλεγχο των εισόδων του χρήστη.
3. **OnDraw()**: "Virtual" μέθοδος για την απόδοση γραφικών στην οθόνη.
4. **Run()**: Εκτελεί τη βασική λούπα του παιχνιδιού και καλεί, με τη σειρά, τις OnEvent(), OnControl() και OnDraw().

Κλάση:***Game (GameWindow)***

Η κλάση Game κληρονομεί τις λειτουργίες της GameWindow και προσθέτει επιπλέον λειτουργικότητες.

Στοιχεία της κλάσης:

- **GameManager**: Αρχικοποιεί ένα αντικείμενο της κλάσης GameManager, το οποίο διαχειρίζεται τα στάδια του παιχνιδιού.

Μέθοδοι της κλάσης:

- **OnEvent()**: Επανεγγράφεται για να διαχειρίζεται τα γεγονότα του Pygame μέσω της pygame.key.get_pressed(). Ανάλογα με το αποτέλεσμα, ενημερώνει το αντικείμενο ControlState με τις τρέχουσες καταστάσεις εισόδου του παίκτη.

Κλάση: ***GameManager***

Αποτελεί τη βασικότερη κλάση του παιχνιδιού, καθώς ελέγχει το στάδιο του παιχνιδιού στο οποίο βρισκόμαστε και εκτελεί τις αντίστοιχες onDraw, onEvent και onControl.

Διαχειρίζεται την εναλλαγή μεταξύ των σταδίων, βασιζόμενη στο ControlState και το GameState. Για κάθε “στάδιο” έχώ υλοποιήσει και μια ξεχωριστή κλάση με τις δικιές του onEvent, onDraw, onControl. Έτσι, είναι εύκολο, ελέγχοντας το control_state (game_state αντικείμενο) να εκτελείται το κατάλληλο κομμάτι κώδικα.

Όπως προανέφερα για καθε “στάδιο” του παιχνιδιού έχω και μια κλάση . Έτσι, υπάρχουν οι κλάσεις:

- **StartGame**
- **EndGame**
- **Level1**
- **Level2**
- **Multiplayer**

Όλες οι παραπάνω κλάσεις ακολουθούν παρόμοια δομή, οπότε θα αναλυθεί η πιο «φορτωμένη» από αυτές:

Αρχικά έχουμε μια κλάση ***GameAssets*** η οποία αναλαμβάνει την διαχείριση των διαφόρων assets του “σταδίου” του πρώτου επιπέδου.

Επιπλέον, έχουμε μια γενική κλάση:

Κλάση:***GameObject***

- Asset
- Θέση στον άξονα x
- Θέση στον άξονα y

Μέθοδοι της κλάσης:

- **Position()**: Επιστρέφει τις συντεταγμένες (x, y)
- **Collision(obj)**: Ελέγχει αν το αντικείμενο βρίσκεται σε επαφή με κάποιο άλλο αντικείμενο
- **Update(dx, dy)**: Ενημερώνει τη θέση του αντικειμένου στον χώρο

Πάνω σε αυτή την κλάση δημιουργώ οποιαδήποτε άλλη κλάση χρειάζεται το επίπεδο μου και τους προσθέτω έξτρα χαρακτηριστικά, όπως η κλάση ***Bullet*****,** η κλάση ***Player,*** η κλάση ***Enemy*** και η κλάση ***HealthBar***. Σε κάθε μια από αυτές τις κλάσεις προσθέτω και την δικιά τους onDraw συνάρτηση (μια τέτοια κλάση μπορεί να έχει παραπάνω από ένα asset).

Και έτσι καταλήγουμε στην:

Κλάση: ***Level1***

- Αρχικοποιεί όλα τα παραπάνω (δημιουργεί λίστα με τους αντιπάλους και λίστα με τις σφαίρες)
- Υλοποιεί την λογική του επιπέδου.
- Ελέγχει το ControlState και κάνει τα ανάλογα updates ελέγχοντας κάθε φορά για collisions. Διαγράφει σφαίρες και αντίπαλους όταν πρέπει .
- Ενημερώνει το GameState όταν ολοκληρωθεί το επίπεδο.

2.3 Βάση Δεδομένων

Για τη διαχείριση των δεδομένων του παιχνιδιού, δημιουργήθηκε η κλάση GameDataBase.

Κλάση: ***GameDataBase***

Η κλάση αυτή αναλαμβάνει τη διαχείριση της σύνδεσης και των λειτουργιών της βάσης δεδομένων. Χρησιμοποιεί τη βιβλιοθήκη sqlite3 για την επικοινωνία με τη βάση δεδομένων.

Μέθοδοι:

- **__create_connection(name)** *(private)*:
  Δημιουργεί σύνδεση με τη βάση δεδομένων που έχει όνομα name.
- **__execute_query(query)** *(private)*:
  Εκτελεί ένα ερώτημα που τροποποιεί δεδομένα (π.χ., INSERT, UPDATE, DELETE).
- **__execute_read_query(query)** *(private)*:
  Εκτελεί ένα ερώτημα που ανακτά δεδομένα από τη βάση (π.χ., SELECT).
- **Insert(name, score)**:
  Εισάγει ένα νέο σκορ στη βάση δεδομένων με τα πεδία name (όνομα παίκτη) και score (σκορ).
- **ReadTable()**:
  Επιστρέφει όλα τα δεδομένα από έναν πίνακα της βάσης δεδομένων.
- **ReadFirst()**:
  Επιστρέφει την πρώτη εγγραφή από τον πίνακα σκορ (συνήθως το υψηλότερο σκορ).

- **Read(id)**:
  Ανακτά δεδομένα ενός συγκεκριμένου παίκτη, βάσει του id.

- **Update(id, name, score)**:
  Ενημερώνει τα δεδομένα ενός παίκτη (όνομα και σκορ) βάσει του id.

Κλάση: ***Score***

Η κλάση Score είναι σχεδιασμένη για να ενσωματώνεται απευθείας στη λογική του παιχνιδιού, διασυνδέοντας τη βάση δεδομένων με τη λειτουργία του σκορ στην οθόνη.

Έχει ως μεταβλητή ένα αντικείμενο της κλάσης *GameDataBase* και μεθόδους:

- **UpdateScore(score)**:
  Ενημερώνει το σκορ του παίκτη κατά τη διάρκεια του παιχνιδιού.
- **UpdateHighScore()**:
  Ελέγχει και ενημερώνει το υψηλότερο σκορ, αν ο παίκτης καταφέρει να το ξεπεράσει.
- **InsertToDataBase()**:
  Εισάγει το σκορ του παίκτη στη βάση δεδομένων στο τέλος του παιχνιδιού.
- **UpdateDataBase(id, name, score)**:
  Ενημερώνει τα δεδομένα ενός συγκεκριμένου παίκτη στη βάση δεδομένων.
- **ReadFromDataBase()**:
  Διαβάζει όλα τα δεδομένα από τη βάση, συνήθως για να εμφανίσει τα σκορ ή να ελέγξει τα στατιστικά.
- **ReadFirst()**:
  Ανακτά το υψηλότερο σκορ από τη βάση δεδομένων.
- **OnDraw(position)**:
  Ζωγραφίζει το σκορ του παίκτη στην οθόνη στη συγκεκριμένη θέση position. Αυτή η μέθοδος συνδέει τη λογική του παιχνιδιού με την εμφάνιση των δεδομένων στην οθόνη.

2.4 Δίκτυο

Για τη δημιουργία και εκτέλεση του παιχνιδιού σε λειτουργία πολλαπλών παικτών, υλοποιήθηκε ένα αρχείο **network.py**, το οποίο περιλαμβάνει τις βασικές συναρτήσεις για την εκκίνηση server, τη σύνδεση client, και τη διαχείριση της επικοινωνίας μεταξύ των παικτών.

Βασικές Συναρτήσεις του αρχείου network.py:

1. **Start_server(port)**:
   Εκκινεί τον server στη θύρα port και περιμένει συνδέσεις από clients.

2. **Start_client(ip, port)**:
   Επιτρέπει τη σύνδεση ενός client στον server μέσω της διεύθυνσης IP ip και της θύρας port.
3. **Send_message()**:
   Στέλνει μηνύματα από τον server προς τον client ή το αντίστροφο.
4. **Receive_message()**:
   Λαμβάνει μηνύματα από τον client ή τον server αντίστοιχα.
5. **Send_data(conn, data)**:
   Αποστέλλει δεδομένα μέσω μιας σύνδεσης conn σε δυαδική μορφή για μεγαλύτερη ασφάλεια και ταχύτητα.
6. **Receive_data(data)**:
   Λαμβάνει δεδομένα από τον server ή τον client και τα επεξεργάζεται.
7. **Start_server_multy(port, timeout)**:
   Εκκινεί τον server για πολλαπλές συνδέσεις με ορισμένο timeout για την αποφυγή καθυστερήσεων και διαχείριση πολλαπλών παικτών.

Όλες οι παραπάνω συναρτήσεις περιλαμβάνουν τους απαραίτητους ελέγχους για την αποφυγή σφαλμάτων, όπως αποτυχίες σύνδεσης ή λάθη κατά τη μετάδοση δεδομένων.

Επιπλέον δημιουργήθηκε και μια κλάση.

Κλάση: ***MultiplayerNetwork***

Η κλάση MultiplayerNetwork χρησιμοποιεί τις παραπάνω συναρτήσεις για να υλοποιήσει μια ολοκληρωμένη λύση επικοινωνίας μεταξύ server και client.

Μέθοδοι:

1. **ConnectServer()**:
   Εκκινεί τον server και τον προετοιμάζει για να δεχτεί συνδέσεις.
2. **ConnectClient()**:
   Επιτρέπει στον client να συνδεθεί σε έναν server.

Οι μέθοδοι αυτές καλούν τις αντίστοιχες συναρτήσεις του αρχείου network.py και είναι σχεδιασμένες ώστε να είναι εύκολα επεκτάσιμες.

Στην κλάση που αντιστοιχεί στο στάδιο **Multiplayer**, υπάρχει μια μεταβλητή που καθορίζει αν το αντικείμενο λειτουργεί ως **client** ή **server**. Ανάλογα με την τιμή αυτής της μεταβλητής, εκτελούνται τα αντίστοιχα κομμάτια κώδικα για τη διαχείριση της επικοινωνίας.

Η επικοινωνία μεταξύ server και client είναι **αμφίδρομη** και γίνεται μέσω των μεθόδων που περιγράφηκαν παραπάνω. Κατά τη διάρκεια κάθε frame του παιχνιδιού, τα δεδομένα που αποστέλλονται περιλαμβάνουν:

- Θέσεις των παικτών.
- Πληροφορίες για τις ενέργειες (π.χ., πυροβολισμοί).
- Ενημερώσεις για το status των παικτών (π.χ., ζωές, σκορ).

Η επικοινωνία είναι συγχρονισμένη ώστε να διασφαλίζεται η ομαλή εμπειρία παιχνιδιού, χωρίς καθυστερήσεις ή σφάλματα.

3. Game Assets

Για την κατασκευή του παιχνιδιού χρησιμοποίησα δωρεάν **assets** από το διαδίκτυο, τα οποία κάλυψαν τις ανάγκες του παιχνιδιού σε γραφικά και ήχο. Παρακάτω παρατίθεται μια αναλυτική περιγραφή των **sprites** και των **ήχων** που χρησιμοποιήθηκαν για κάθε στάδιο του παιχνιδιού.

3.1 Sprites

Η δομή των σταδίων του παιχνιδιού είναι παρόμοια, αλλά κάθε στάδιο έχει τα δικά του **sprites**. Για τη διαχείριση αυτών, δημιουργήθηκε μια ξεχωριστή κλάση για κάθε στάδιο, η οποία καλεί την **pygame.image.load()** για να φορτώσει τα απαραίτητα αρχεία εικόνας από το φάκελο assets. Επίσης, χρησιμοποιείται η **pygame.transform.scale()** για να προσαρμόσω τις διαστάσεις του κάθε sprite στα δεδομένα του επιπέδου.

Τα sprites που χρησιμοποιούνται στο παιχνίδι είναι τα εξής:

- **Δύο εικόνες διαστήματος** για το background.
- **Μια εικόνα από πύραυλο** για τον παίκτη (player).
- **Μια λίστα από εικονίδια** για τους αντιπάλους.
- **Ένα εικονίδιο για το περίγραμμα της ζωής του παίκτη**.
- **Ένα εικονίδιο μπάρας ζωής του παίκτη** (πράσινο).
- **Ένα εικονίδιο μπάρας ζωής του αντίπαλου** (κόκκινο).
- **Δύο εικονίδια περιγράμματος κουμπιών** για την αρχή του παιχνιδιού, ανάλογα με την κατάσταση του κέρσορα (πάνω από το κουμπί ή μακριά).
- **Ένα background για το τελικό στάδιο απεικόνισης του σκορ**.

Η χρήση αυτών των **sprites** επιτρέπει στο παιχνίδι να είναι οπτικά πιο ενδιαφέρον αλλά και λειτουργικό σε κάθε στάδιο του.

3.2 Ήχος

Για τους ήχους του παιχνιδιού χρησιμοποιήθηκαν έτοιμοι ήχοι από το διαδίκτυο. Συγκεκριμένα, χρησιμοποιήθηκαν τέσσερις βασικοί ήχοι:

- Μια **μουσική υπόκρουση** που παίζει σε λούπα (background music).
- Ένας **ήχος για τον πυροβολισμό**.
- Ένας **ήχος για το κλικ του ποντικιού**.
- Ένας **ήχος για το χτύπημα ενός αντίπαλου από σφαίρα ή του παίκτη από αντίπαλο**.

Για την αναπαραγωγή αυτών των ήχων δημιουργήθηκε μια κλάση ***GameSound*** που καλεί την **pygame.mixer.Sound()** για να αρχικοποιήσει και να αναπαράγει τους ήχους.

Η κλάση αυτή περιέχει τις εξής μεθόδους:

- **Clicked()** – Εκτελεί τον ήχο του κλικ του ποντικιού.
- **Shooted()** – Εκτελεί τον ήχο του πυροβολισμού.
- **Hitted()** – Εκτελεί τον ήχο της σύγκρουσης (χτύπημα).

Για τη διαχείριση των ήχων μέσα στο παιχνίδι, χρησιμοποιήθηκε το **Control Pattern**. Δημιουργήθηκε μια **Control class** και μια ξεχωριστή κλάση για κάθε ήχο, π.χ.:

```
class ClickControl(Control):
    def execute(self, receiver):
        receiver.clicked()
```

Και ετσι αρχικοποιώντας ένα τέτοιο αντικείμενο μέσα στην κλάση του κάθε επιπέδου, π.χ.:

```
self.controlClick = ClickControl()
self.sounds  = GameSounds()
```

μπορώ να την καλέσω

```
self.controlClick = ClickControl()
```

4. Επιπλέον Ρουτίνες - Αλγόριθμοι

Για την υλοποίηση του παιχνιδιού, δημιούργησα κάποιες πρόσθετες συναρτήσεις και αλγορίθμους που βοηθούν στην εύρυθμη λειτουργία του παιχνιδιού και στην υλοποίηση κρίσιμων μηχανισμών του, όπως η σύγκρουση, η κίνηση και η συμπεριφορά των αντιπάλων. Αυτές οι συναρτήσεις καταχωρήθηκαν σε ένα αρχείο που ονόμασα **algo.py**. Ακολουθούν οι βασικές ρουτίνες και αλγόριθμοι που χρησιμοποίησα:

4.1 Σύγκρουση (Collision)

Για την υλοποίηση του συστήματος σύγκρουσης, δεδομένου ότι το παιχνίδι είναι δισδιάστατο, η λογική που ακολούθησα βασίστηκε στον έλεγχο **box collision**. Αυτό σημαίνει ότι ελέγχω αν δύο ορθογώνια κουτιά (που αντιπροσωπεύουν τα αντικείμενα του παιχνιδιού) τέμνονται μεταξύ τους.

Για την υλοποίηση του ελέγχου αυτού δημιούργησα μια συνάρτηση που ελέγχει αν ένα σημείο βρίσκετε εντός ενός κουτιού.

Is_point_inside_box(point, box_point, box_width, box_height)

Έτσι, αν έστω και μια από τις κορυφές ενός κουτιού είναι εντός ενός άλλου κουτιού τότε έχουμε επαφή. Ο παραπάνω έλεγχος είναι αρκετά εύκολος, καθώς αρκεί να ελέγξουμε αν το x του σημείου είναι μετάξυ των x των σημείων του κουτιού και αντίστοιχα το y του σημείου μετάξι των y των σημείων του κουτιού. Η απλή αυτή έκδοση της συνάρτησης είναι αποτελεσματική καθώς έχουμε πάντοτε ορθοκανονικά κουτιά.

4.2 Υπολογισμός Διανύσματος (Vector Calculation)

Επιπλέον, δημιούργησα μια ρουτίνα για τον υπολογισμό διανύσματος

```
def dv( p1, p2 ):
return ( p2[0] - p1[0], p2[1] - p1[1]  )
```

Καθώς και μια για την κανονικοποίηση διανύσματος (normalize())

```
def normalize( p ):
    len = math.sqrt(p[0]*p[0] + p[1]*p[1])
    return ( p[0]/len, p[1]/len )
```

Και οι δυο παραπάνω συναρτήσεις χρησιμοποιήθηκαν κυρίως για την συμπεριφορά - κίνηση των αντιπάλων . Όπως ανέφερα και στην πρώτη ενότητα ένας αντίπαλος μόλις εξουδετερωθεί πέφτει και προσπαθεί να χτυπήσει τον παίχτη. Αυτή η “προσπάθεια” στην ουσία είναι ο υπολογισμός του διανύσματος που δείχνει προς τα

που βρίσκετε ο παίχτης σε σχέση με τον αντίπαλο . Έτσι, στην δύναμη (πτώσης) προσθέτω και ένα τέτοιο διάνυσμα ώστε να προσπαθήσει να τον φτάσει.

Αυτές οι εξτρά ρουτίνες και αλγόριθμοι ενσωματώνονται απρόσκοπτα στο παιχνίδι και βοηθούν στην υλοποίηση δυναμικών και ρεαλιστικών κινήσεων και αλληλεπιδράσεων, δημιουργώντας μια πιο ενδιαφέρουσα και αμφίδρομη εμπειρία για τον παίκτη.

5. Επίλογος

Η υλοποίηση του παιχνιδιού αποτέλεσε μια ενδιαφέρουσα και απαιτητική πρόκληση, με την εφαρμογή διαφορετικών τεχνικών και βιβλιοθηκών. Η χρήση της γλώσσας προγραμματισμού **Python** σε συνδυασμό με τη βιβλιοθήκη **Pygame** για την ανάπτυξη του παιχνιδιού μας, επιτρέπει τη δημιουργία ενός **Spaceshooter** παιχνιδιού, το οποίο μπορεί να παιχτεί είτε με 2, είτε με 3 παίκτες.

Το παιχνίδι περιλαμβάνει τα εξής χαρακτηριστικά:

- ✓ **Πολλαπλά επίπεδα δυσκολίας**: Το παιχνίδι διαθέτει δύο επίπεδα δυσκολίας, με κάθε επίπεδο να έχει διαφορετική δυσκολία και αντιπάλους, δίνοντας στον παίκτη μια αυξανόμενη πρόκληση.
- ✓ **Αποθήκευση σκορ**: Το σκορ του παίκτη υπολογίζεται και αποθηκεύεται σε μια βάση δεδομένων **SQLite3**. Αυτό επιτρέπει στους παίκτες να παρακολουθούν την πρόοδό τους και να συγκρίνουν τα αποτελέσματά τους με άλλα υψηλά σκορ.
- ✓ **Ήχοι και Γραφικά**: Το παιχνίδι αξιοποιεί διάφορους ήχους και γραφικά για να ενισχύσει την εμπειρία του χρήστη, περιλαμβάνοντας ηχητικά εφέ για τα πυροβολήματα, τα χτυπήματα και τη μουσική υπόκρουση.
- ✓ **Multiplayer**: Το παιχνίδι υποστηρίζει **multiplayer σε πραγματικό χρόνο**, όπου οι παίκτες μπορούν να αγωνιστούν μεταξύ τους μέσω δικτύου. Ο κάθε παίκτης συνδέεται είτε ως **client**, είτε ως **server**, με πλήρη αμφίδρομη επικοινωνία.
- ✓ **Έλεγχοι Σφάλματος**: Όλο το παιχνίδι έχει δομηθεί με τέτοιο τρόπο ώστε να περιλαμβάνει **ελέγχους σφαλμάτων** σε κρίσιμα σημεία του κώδικα, εξασφαλίζοντας την ομαλή λειτουργία του παιχνιδιού χωρίς απροσδόκητα προβλήματα ή σφάλματα κατά τη διάρκεια του παιχνιδιού.

Τέλος, με την υπάρχουσα αρχιτεκτονική του κώδικα, είναι εύκολη η προσθήκη νέων επιπέδων, η τροποποίηση του βαθμού δυσκολίας και η αλλαγή των γραφικών και ήχων του παιχνιδιού. Αυτή η ευελιξία επιτρέπει στο παιχνίδι να επεκταθεί μελλοντικά, καλύπτοντας πιθανές ανάγκες για προσθήκη νέων χαρακτηριστικών ή ακόμα και τη μετατροπή του σε ένα πιο σύνθετο παιχνίδι με περισσότερους παίκτες ή πιο απαιτητικά επίπεδα.