Reverse G

Τεκμηρίωση Υλοποίησης και Λειτουργικότητας

Ονόματα Φοιτητών:

- Νικόλας Νινιός – ΑΜ. 68
- Παναγιώτης Παναγιωτίδης – ΑΜ. 69

Μάθημα: Αλγοριθμική Σκέψη

Πίνακας Περιεχομένων

Εισαγωγή

Στο πλαίσιο της εργασίας για το μάθημα "Αλγοριθμική Σκέψη", αναπτύξαμε το παιχνίδι "ReverseG", ένα 2D platform game, χρησιμοποιώντας τη βιβλιοθήκη Pygame για την υλοποίησή του.

Ο παίκτης, πατώντας ένα κουμπί, μπορεί να αντιστρέψει τη βαρύτητα του χαρακτήρα του, ώστε να μετακινείται από εμπόδιο σε εμπόδιο. Εάν δεν αντιδράσει εγκαίρως, ο χαρακτήρας θα βγει εκτός οθόνης και θα χαθεί στο διάστημα.

Το παιχνίδι είναι τύπου endless runner, με επίπεδα δυσκολίας που επηρεάζουν την ταχύτητα κίνησης του παίκτη και των αντικειμένων που εμφανίζονται.

Στην παρούσα αναφορά, θα παρουσιάσουμε τις βασικές λειτουργίες του παιχνιδιού, καθώς και τις τεχνικές προσεγγίσεις που χρησιμοποιήθηκαν για την υλοποίησή του.

Τεχνική Υλοποίηση

Στην ενότητα αυτή, θα αναλύσουμε τις βασικές τεχνικές που χρησιμοποιήθηκαν για την υλοποίηση του παιχνιδιού "ReverseG**"**. Το παιχνίδι αναπτύχθηκε με τη βοήθεια της βιβλιοθήκης Pygame**.**

2.1 **Βιβλιοθήκες και Τεχνολογίες**

Για την υλοποίηση του παιχνιδιού, χρησιμοποιήθηκαν οι εξής βασικές βιβλιοθήκες και τεχνολογίες:

- **Pygame**: Η κύρια βιβλιοθήκη για την ανάπτυξη του παιχνιδιού. Παρέχει εργαλεία για τη διαχείριση της οθόνης, των ήχων, των γραφικών και της φυσικής κίνησης των αντικειμένων.
- **Python**: Η γλώσσα προγραμματισμού στην οποία γράφτηκε ο κώδικας. Προσφέρει ευχρηστία και ισχυρές δυνατότητες για την ανάπτυξη παιχνιδιών.
- **Random**: Η βιβλιοθήκη που χρησιμοποιείται για την τυχαία δημιουργία θέσεων των εμποδίων και των άλλων στοιχείων του παιχνιδιού. Χρησιμοποιείται για τη δημιουργία τυχαίων αριθμών, όπως για την τυχαία εμφάνιση των εμποδίων ή για την επιλογή τυχαίων στοιχείων στη διάρκεια του παιχνιδιού.
- **Math**: Η βιβλιοθήκη που χρησιμοποιείται για μαθηματικές λειτουργίες, όπως υπολογισμοί για την ταχύτητα, τη γωνία ή τη φυσική του παιχνιδιού.
- **Sys**: Η βιβλιοθήκη που χρησιμοποιείται για τη διαχείριση συστημικών παραμέτρων και την έξοδο του παιχνιδιού. Μέσω αυτής μπορούμε να ελέγξουμε την έξοδο του παιχνιδιού ή να τερματίσουμε το πρόγραμμα σε περίπτωση σφάλματος.

2.2 **Δομή Κώδικα**

Ο κώδικας του παιχνιδιού οργανώθηκε με τέτοιο τρόπο ώστε να είναι κατανοητός και επεκτάσιμος. Κάθε κύρια λειτουργία του παιχνιδιού (όπως δημιουργεία χαρακτήρα, η και η διαχείριση των εμποδίων) υλοποιείται σε ξεχωριστές κλάσεις. Η βασική δομή του παιχνιδιού περιλαμβάνει τα εξής μέρη:

- **Main Game Loop**:

Στη βασική λούπα του παιχνιδιού, γίνεται ο έλεγχος των σκηνών (scenes). Αυτές περιλαμβάνουν το κύριο μενού ,τη σκηνή του gameplay και τη σκηνή των αποτελεσμάτων. Στο βρόχο του παιχνιδιού ελέγχεται ποια σκηνή είναι ενεργή, και εκτελούνται οι αντίστοιχες ενέργειες.

- Ελέγχεται το παιχνίδι ανάλογα με την επιλεγμένη σκηνή (π.χ. αν είναι το main menu, αν είναι gameplay, ή αν είναι results).

- **Main Menu**:

Η πρώτη σκηνή που συναντάει ο παίκτης όταν ξεκινά το παιχνίδι. Είναι το κεντρικό μενού που επιτρέπει στον παίκτη να επιλέξει διάφορες ενέργειες. Από εδώ μπορεί να επιλέξει το επίπεδο δυσκολίας, να δει τα σκορ ή να ξεκινήσει το παιχνίδι.

- Παρουσιάζει επιλογές για την έναρξη του παιχνιδιού ή τη μετάβαση σε άλλες λειτουργίες, όπως την προβολή των scores.
- Ο παίκτης μπορεί να επιλέξει το επίπεδο δυσκολίας (π.χ. εύκολο, μεσαίο, δύσκολο).

- **Gameplay**:

Στη σκηνή του παιχνιδιού, ο παίκτης πηδάει από εμπόδιο σε εμπόδιο, αλλάζοντας τη βαρύτητα. Ο στόχος είναι να μην «χαθεί» στο κενό , καθώς ο χαρακτήρας κινείται στο διάστημα.

- Ο παίκτης αναστρέφει τη βαρύτητα για να μετακινηθεί από το έδαφος στην οροφή.
- Η οθόνη ανανεώνεται συνεχώς με τη νέα θέση του παίκτη και τα εμπόδια.
- Αν ο παίκτης χάσει το timing και πέσει στο κενό, χάνει το παιχνίδι.

- **Results**:

Η σκηνή στην οποία οι παίκτες μπορούν να δουν το σκορ τους μετά την ολοκλήρωση του παιχνιδιού. Εμφανίζεται το τελικό σκορ και άλλες σχετικές πληροφορίες.

- Παρουσιάζονται τα αποτελέσματα του παίκτη (π.χ. σκορ).
- Επιλογές για επιστροφή στο κύριο μενού ή για νέο παιχνίδι.

- **Player Class**:

Η κλάση που αντιπροσωπεύει τον παίκτη. Εδώ υλοποιούνται οι δυνατότητες του χαρακτήρα, όπως η κίνηση, η αντιστροφή βαρύτητας και η αλληλεπίδραση με τα εμπόδια.

- Η κλάση αυτή χειρίζεται τις κινήσεις του παίκτη και τη φυσική του κίνησης στην οθόνη.
- Διαχειρίζεται την αλλαγή της βαρύτητας όταν ο παίκτης πατά το κουμπί για αντιστροφή.

- **Obstacle Class**:

Η κλάση για τα εμπόδια που εμφανίζονται στο παιχνίδι και που πρέπει να αποφύγει ο παίκτης. Τα εμπόδια δημιουργούνται τυχαία και πρέπει να παρακολουθούν τη θέση τους στην οθόνη για να διασφαλιστεί ότι εμφανίζονται με σωστό τρόπο.

- Η κλάση αυτή ορίζει τα εμπόδια, τη θέση τους και τον τρόπο με τον οποίο τα αντικείμενα αυτά εμφανίζονται στο παιχνίδι.
- Ο παίκτης πρέπει να πατήσει πάνω τους

2.3 **Μηχανισμοί Παιχνιδιού**

Ο βασικός μηχανισμός του παιχνιδιού **"ReverseG"** επικεντρώνεται στην αλληλεπίδραση του παίκτη με τα εμπόδια, χρησιμοποιώντας την αντιστροφή της βαρύτητας για να επιβιώσει και να προχωρήσει στο παιχνίδι. Ο μηχανισμός αυτός δίνει στο παιχνίδι τη μοναδικότητά του και το καθιστά δυναμικό και απαιτητικό.

- **Αντιστροφή Βαρύτητας**: Ο παίκτης έχει τη δυνατότητα να αλλάζει τη βαρύτητα, επιτρέποντάς του να πηδάει από το έδαφος στην οροφή και αντίστροφα.
- **Κίνηση Παίκτη**: Ο παίκτης κινείται κάθετα, και πρέπει να χρησιμοποιεί την αντιστροφή της βαρύτητας για να προσγειωθεί με ασφάλεια από εμπόδιο σε εμπόδιο.
- **Αποφυγή Κενών**: Ο κύριος κίνδυνος για τον παίκτη είναι να «χαθεί» στο διάστημα. Αν ο παίκτης πέσει εκτός του παιχνιδιού, χάνει.
- **Εμπόδια**: Τα εμπόδια εμφανίζονται τυχαία στο πεδίο, αναγκάζοντας τον παίκτη να πηδάει από το ένα στο άλλο, προκειμένου να επιβιώσει και να συνεχίσει το παιχνίδι.

Χαρακτηριστικά και Gameplay

Το παιχνίδι "ReverseG" είναι ένα endless-run παιχνίδι όπου ο παίκτης ελέγχει έναν χαρακτήρα που πηδάει από εμπόδιο σε εμπόδιο, αλλάζοντας τη βαρύτητα για να παραμείνει στον χώρο. Ο στόχος είναι να αποφύγει το διάστημα και να συνεχίσει την πορεία του, έχοντας πάντα στο μυαλό την αλλαγή της βαρύτητας.

3.1. Ταμπλό και Σκηνή

Η σκηνή του παιχνιδιού είναι το περιβάλλον μέσα στο οποίο ο παίκτης αλληλεπιδρά με τον κόσμο του παιχνιδιού. Το ταμπλό ανανεώνεται συνεχώς, εμφανίζοντας τον χαρακτήρα του παίκτη, τα εμπόδια και την ένδειξη της τρέχουσας κατάστασης του παιχνιδιού (π.χ. σκορ).

- Το παιχνίδι έχει μία σταθερή σκηνή με δυναμική κίνηση, όπου η θέση του παίκτη και τα εμπόδια ανανεώνονται σε κάθε βήμα.
- Ο παίκτης κινείται κάθετα, με τα εμπόδια να εμφανίζονται με τυχαία συχνότητα και θέσεις.
- Η φυσική του παιχνιδιού είναι βασισμένη στην αντιστροφή της βαρύτητας.

3.2. Επίπεδα Δυσκολίας

Το παιχνίδι προσφέρει **τρία επίπεδα δυσκολίας** που επηρεάζουν την ταχύτητα του παίκτη και τη συχνότητα εμφάνισης των εμποδίων.

- **Εύκολο(MARS)**: Ο παίκτης κινείται πιο αργά και τα εμπόδια εμφανίζονται αραιότερα.
- **Μεσαίο(EARTH)**: Ο παίκτης κινείται με μέτρια ταχύτητα και τα εμπόδια έρχονται σε πιο γρήγορη ροή.
- **Δύσκολο(JUPITAR)**: Ο παίκτης κινείται γρήγορα και τα εμπόδια εμφανίζονται συχνά και γρήγορα, απαιτώντας υψηλή συγκέντρωση και ταχύτητα αντίδρασης.

3.3. Σύστημα Σκορ

Το παιχνίδι διαθέτει **σύστημα σκορ**, όπου ο παίκτης κερδίζει πόντους κάθε φορά που περνά από ένα εμπόδιο και παραμένει ζωντανός. Το σκορ αυξάνεται με την ολοκλήρωση κάθε κύκλου και την επιτυχία στην αποφυγή των εμποδίων.

- **Σκορ**: Ο παίκτης κερδίζει πόντους για κάθε επιτυχημένο άλμα από εμπόδιο σε εμπόδιο.
- Το σκορ εμφανίζεται σε πραγματικό χρόνο στην οθόνη, δίνοντας στον παίκτη άμεση ανατροφοδότηση.

3.4. Multiplayer Mode

Το παιχνίδι υποστηρίζει **multiplayer mode**, επιτρέποντας σε δύο παίκτες να παίξουν ταυτόχρονα.

- Οι παίκτες μπορούν να διαγωνιστούν για το καλύτερο σκορ.
- Το παιχνίδι διαχειρίζεται τον ανταγωνισμό μεταξύ των παικτών με κοινό ταμπλό και σκορ για κάθε παίκτη.

Γραφικά και Ήχος

Το παιχνίδι **"ReverseG"** χρησιμοποιεί **βασικά γραφικά 2D** για την απεικόνιση του χαρακτήρα, των εμποδίων και του περιβάλλοντος του παιχνιδιού. Η σκηνή είναι απλή με σαφή διάκριση των στοιχείων του παιχνιδιού για εύκολη αναγνώριση από τον παίκτη.

- **Χαρακτήρας**: Ο χαρακτήρας του παίκτη είναι εύκολα αναγνωρίσιμος και κινείται με φυσική κίνηση στο περιβάλλον.
- **Εμπόδια**: Τα εμπόδια είναι δυναμικά .
- **Σκηνικό**: Το σκηνικό είναι απλό και ανανεώνεται με βάση την πρόοδο του παιχνιδιού, ώστε να παρέχει μία συνεχιζόμενη εμπειρία χωρίς απόσπαση της προσοχής του παίκτη.

Η απλότητα των γραφικών επιτρέπει στο παιχνίδι να επικεντρωθεί στην αλληλεπίδραση του παίκτη με το περιβάλλον, χωρίς να υπερφορτώνει τον χρήστη με περίπλοκες εικόνες.

Ήχος

Το παιχνίδι **"ReverseG"** περιλαμβάνει μουσική Η χρήση ήχου προσφέρει την αίσθηση της έντασης και της πρόκλησης, ενώ παράλληλα ενισχύει την ατμόσφαιρα του παιχνιδιού.

Αποθήκευση και Δομές Δεδομένων

Η αποθήκευση των δεδομένων στο παιχνίδι γίνεται μέσω ενός αρχείου **JSON**, το οποίο χρησιμοποιείται για την αποθήκευση των σκορ των παικτών. Ο κύριος λόγος για τη χρήση του αρχείου JSON είναι η δυνατότητα αποθήκευσης δεδομένων με εύκολο και δομημένο τρόπο, επιτρέποντας την ανάγνωση και την εγγραφή σε αυτό κατά τη διάρκεια του παιχνιδιού.

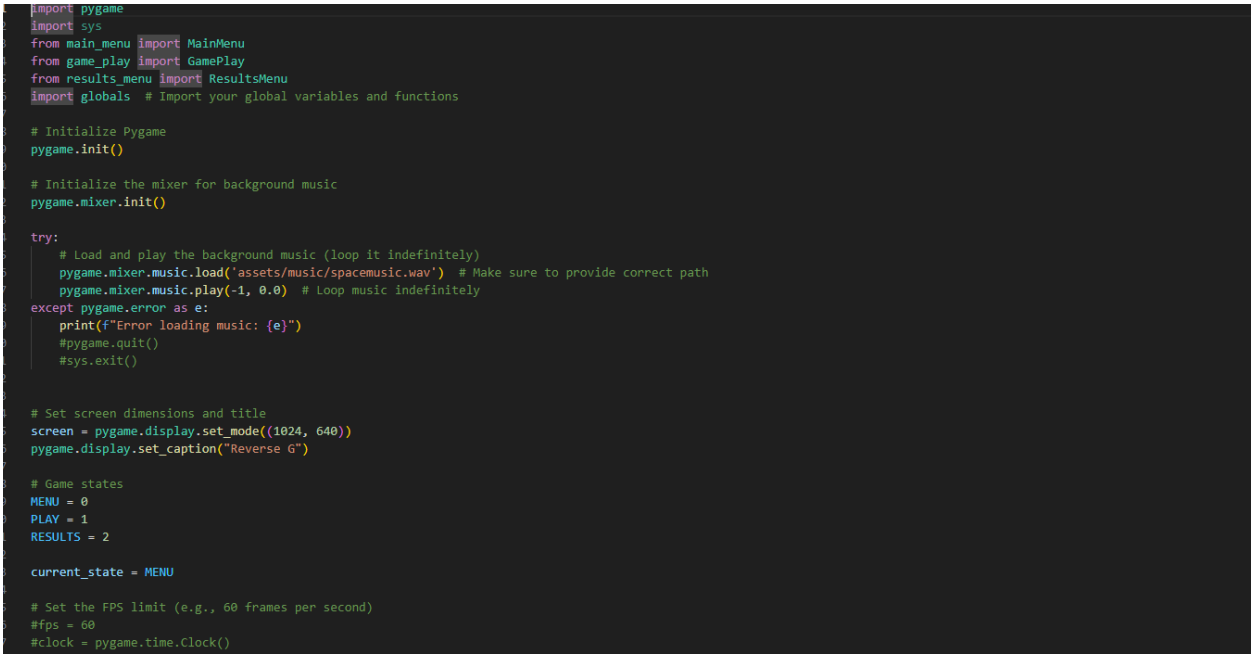
- **Αρχείο JSON**: Το αρχείο αποθηκεύει τα σκορ των παικτών και διασφαλίζει ότι τα δεδομένα παραμένουν διαθέσιμα ακόμα και αν ο χρήστης κλείσει και ξανανοίξει το παιχνίδι.
- **Αποθήκευση Σκορ**: Κάθε φορά που ο παίκτης ολοκληρώνει το παιχνίδι, το τρέχον σκορ αποθηκεύεται στο αρχείο JSON, έτσι ώστε να είναι προσβάσιμο και την επόμενη φορά που θα ξεκινήσει το παιχνίδι.
- **Διατήρηση Δεδομένων**: Με τη χρήση του αρχείου JSON, τα δεδομένα δεν χάνονται όταν ο χρήστης κλείνει το παιχνίδι και μπορούν να ανακτηθούν στην επόμενη εκκίνηση του παιχνιδιού.

Αυτός ο μηχανισμός αποθήκευσης επιτρέπει την εύκολη παρακολούθηση της προόδου των παικτών και τη διατήρηση των σκορ τους σε όλη τη διάρκεια του παιχνιδιού.

Ανάπτυξη και Δομή Κώδικα

Στην ενότητα αυτή, αναλύουμε τις βασικές λειτουργίες και κλάσεις του κώδικα που υλοποιεί το παιχνίδι **"ReverseG"**, καθώς και τη δομή του προγράμματος. Η δομή του κώδικα είναι οργανωμένη γύρω από σημαντικές λειτουργίες που επιτρέπουν την αλληλεπίδραση του παίκτη με το περιβάλλον του παιχνιδιού. Ακολουθεί η ανάλυση του κώδικα, χωρισμένη σε κλάσεις και συναρτήσεις.

main.py



```python
import pygame
import sys
from main_menu import MainMenu
from game_play import GamePlay
from results_menu import ResultsMenu
import globals  # Import your global variables and functions

# Initialize Pygame
pygame.init()

# Initialize the mixer for background music
pygame.mixer.init()

try:
    # Load and play the background music (loop it indefinitely)
    pygame.mixer.music.load('assets/music/spacemusic.wav')  # Make sure to provide correct path
    pygame.mixer.music.play(-1, 0.0)  # Loop music indefinitely
except pygame.error as e:
    print(f"Error loading music: {e}")
    #pygame.quit()
    #sys.exit()


# Set screen dimensions and title
screen = pygame.display.set_mode((1024, 640))
pygame.display.set_caption("Reverse G")

# Game states
MENU = 0
PLAY = 1
RESULTS = 2

current_state = MENU

# Set the FPS limit (e.g., 60 frames per second)
#fps = 60
#clock = pygame.time.Clock()
```

```python
# Game Loop
def game_loop():
    global current_state

    # Main menu
    if current_state == MENU:
        main_menu = MainMenu(screen)
        current_state = main_menu.run()

    # Gameplay
    elif current_state == PLAY:
        game_play = GamePlay(screen, globals.multiplayer)
        result_data = game_play.run()
        if result_data != 0:  # If gameplay returns result data, move to RESULTS
            current_state = RESULTS
            results_menu = ResultsMenu(screen, result_data)  # Pass scores to results menu
            current_state = results_menu.run()
        else:
            current_state=0

    # Results menu
    elif current_state == RESULTS:
        results_menu = ResultsMenu(screen)
        current_state = results_menu.run()


# Run the game loop
if __name__ == "__main__":
    while True:
        game_loop()
```

Το αρχείο **`main.py`** είναι υπεύθυνο για την εκκίνηση του παιχνιδιού και τη διαχείριση της κύριας λούπας του παιχνιδιού (main loop). Μέσω αυτής της λούπας, διαχειρίζεται όλες τις υπόλοιπες σκηνές του παιχνιδιού, όπως το **gameplay**, το **main menu**, και άλλες. Επιπλέον, ρυθμίζει το μέγεθος της σκηνής του παιχνιδιού σε **1024x640** και ενεργοποιεί τη μουσική του background.

main_menu.py

```python
import pygame
import sys
from scoreCls import scoreCls
import globals  # Import your global variables and functions

class MainMenu:
    def __init__(self, screen):
        self.screen = screen
        self.background = pygame.image.load('assets/images/menuBck.jpg')  # Load background image
        self.font = pygame.font.SysFont("Comic Sans MS", 40)  # Comic font for game title
        self.button_font = pygame.font.SysFont("Arial", 30)  # Regular font for buttons
        self.clean_font = pygame.font.SysFont("Arial", 15)
        self.text_color = (255, 255, 255)  # White text color
        self.running = True

        # Initialize JSONManager for handling scores
        self.scoreCls = scoreCls()  # Add this line

        # Colors
        self.title_color = (255, 0, 0)  # Red for "REVERSE G"
        self.default_color = (255, 255, 255)  # White
        self.selected_color = (0, 0, 255)  # Blue for selected difficulty
        self.box_color = (64, 224, 208)  # Turquoise for the box
        self.diff_box_color = (135, 206, 235)  # Light turquoise for difficulty box
```

Το αρχείο **`main_menu.py`** είναι η πρώτη σκηνή που εμφανίζεται στον χρήστη όταν ξεκινά το παιχνίδι. Στην **`init()`** του **main_menu.py**, ρυθμίζονται τα γραφικά του μενού, όπως το background, τα fonts και τα χρώματα των γραμμάτων. Επίσης, δημιουργείται το αντικείμενο **`scoreCls`**, το οποίο χρησιμοποιείται αργότερα για να εμφανιστούν τα σκορ των παικτών.

```python
def draw_background(self):
    # Blit the background image to the screen at the top-left corner (0, 0)
    self.screen.blit(self.background, (0, 0))

def draw_rounded_box(self, x, y, width, height, radius, color):
    """Draw a box with rounded corners."""
    pygame.draw.rect(self.screen, color, (x, y, width, height), border_radius=radius)
```

```python
def draw(self):
    # Draw the background first
    self.draw_background()

    # Draw the game title "REVERSE G"
    title_text = self.font.render("REVERSE G", True, self.title_color)  # Red font for title
    title_rect = title_text.get_rect(center=(self.screen.get_width() // 2, 100))  # Centered at the top

    # Draw the title in a turquoise box with curved corners
    title_box_width = title_rect.width + 40
    title_box_height = title_rect.height + 20
    self.draw_rounded_box(title_rect.x - 20, title_rect.y - 10, title_box_width, title_box_height, 20, self.box_color)

    # Draw title text on top of the box
    self.screen.blit(title_text, title_rect)

   # Box for menu options with fixed width (300) and height (180)
    menu_box_width = 310
    menu_box_height = 180

    # Calculate the x and y coordinates to center the box
    menu_box_x = self.screen.get_width() // 2 - menu_box_width // 2  # Center horizontally
    menu_box_y = 200  # Position it 200 pixels from the top
```

Οι τρεις συναρτήσεις **`draw()`** που ακολουθούν στο **`main_menu.py`** είναι υπεύθυνες για τη δημιουργία και την ανανέωση των γραφικών της σκηνής σε κάθε frame.Αυτές οι συναρτήσεις επιτρέπουν την ανανέωση και την αλληλεπίδραση του χρήστη με το περιβάλλον του παιχνιδιού σε πραγματικό χρόνο.

```python
def show_scores(self):
    """Display the top 20 scores."""
    self.screen.fill((0, 0, 0))  # Clear the screen with black background

    score_text = self.clean_font.render("Press 'R' to clean the data", True, (255, 255, 255))
    self.screen.blit(score_text, (10, 20))
    # Load top 20 scores from JSONManager
    top_scores = self.scoreCls.get_top_scores(top_n=20)

    # Display title
    title_text = self.font.render("Top 20 Scores", True, self.text_color)
    title_rect = title_text.get_rect(center=(self.screen.get_width() // 2, 50))
    self.screen.blit(title_text, title_rect)

    # Check if there are scores to display
    if not top_scores:
        no_scores_text = self.button_font.render("No scores available", True, (255, 255, 255))
        no_scores_rect = no_scores_text.get_rect(center=(self.screen.get_width() // 2, self.screen.get_height() // 2))
        self.screen.blit(no_scores_text, no_scores_rect)
    else:
        left_x = 50  # Starting x position for the left column
        right_x = self.screen.get_width() // 2 + 50  # Starting x position for the right column
        y_offset = 120  # Starting y position for both columns
        line_spacing = 40  # Space between each score

        # Iterate over the scores and split them into two columns
        for index, score_entry in enumerate(top_scores):
            score_text = self.button_font.render(
                f"{index + 1}. {score_entry['name']} - {score_entry['score']:.2f} pts - {score_entry['diff']}",
                True,
                (255, 255, 255),  # White color
            )

        # Determine if the score should go in the left or right column
            if index < 10:  # First 10 scores go to the left column
                self.screen.blit(score_text, (left_x, y_offset + index * line_spacing))
            else:  # Remaining scores go to the right column
                self.screen.blit(score_text, (right_x, y_offset + (index - 10) * line_spacing))
```

Η συνάρτηση **`show_scores()`** αναλαμβάνει την εμφάνιση των κορυφαίων σκορ στον χρήστη. Όταν καλείται, αλλάζει πλήρως το περιεχόμενο του μενού και εμφανίζει μια λίστα με τα σκορ, χρησιμοποιώντας το αντικείμενο **`scoreCls`**. Η **`get_top_scores()`** είναι μια εσωτερική συνάρτηση του **`scoreCls`**, η οποία παίρνει έναν αριθμό για το πόσα αποτελέσματα θέλουμε να εμφανίσουμε.

Η συνάρτηση **`show_scores()`** μπαίνει σε λούπα, εμφανίζοντας συνεχώς τα σκορ μέχρι ο χρήστης να επιλέξει να επιστρέψει στο κεντρικό μενού ή να πατήσει το κουμπί "R" για να επαναφέρει τα σκορ (reset). Αυτό επιτρέπει στον χρήστη να βλέπει τα σκορ χωρίς να χρειάζεται να ξαναμπεί στο παιχνίδι.

```python
def gameDiff(self,difficulty):
    """Update difficulty variable and highlight the selected difficulty."""
    globals.set_difficulty(difficulty)  # Update the global variable

def run(self):
    self.draw()
    pygame.display.update()

    # Event loop for menu navigation
    while self.running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_1:  # 1 Player selected
                    globals.playersmode(1)
                    return 1  # Go to gameplay
                elif event.key == pygame.K_2:  # 2 Players selected
                    globals.playersmode(2)
                    return 1  # Go to multiplayer gameplay (same as 1 Player for now)
                elif event.key == pygame.K_s:  # Scoreboard selected
                    self.show_scores()  # Show the scores
                    return 0
                elif event.key == pygame.K_a:
                    self.gameDiff("Mars(A)")
                    return 0
                elif event.key == pygame.K_e:
                    self.gameDiff("Earth(E)")
                    return 0
                elif event.key == pygame.K_j:
                    self.gameDiff("Jupiter(J)")
                    return 0
                elif event.key == pygame.K_m:
                    globals.toggle_music()
                    return 0
                if event.key == pygame.K_ESCAPE:
                    return 0  # Exit game or go back
```

Η συνάρτηση `run()` είναι η συνεχιζόμενη λούπα του **main menu**, η οποία ανανεώνει συνεχώς τα γραφικά ανάλογα με τις επιλογές του χρήστη. Σε αυτή τη λούπα, η συνάρτηση παρακολουθεί τις ενέργειες του χρήστη, όπως την πίεση κουμπιών, και αντιδρά αναλόγως. Όταν ο χρήστης πατήσει κάποιο κουμπί (π.χ. το κουμπί "1"), η συνάρτηση `run()` τον μεταφέρει σε μια άλλη λειτουργία.

Για παράδειγμα, αν ο χρήστης πατήσει το κουμπί "1", η `run()` θα καλέσει μια συνάρτηση που βρίσκεται στην `globals` class. Αυτή η συνάρτηση είναι υπεύθυνη για την ενημέρωση του multiplayer mode και την αλλαγή της κατάστασης του παιχνιδιού. Μετά, η `run()` μεταφέρει τον χρήστη στη σκηνή του παιχνιδιού (gameplay scene), όπου μπορεί να αρχίσει να παίζει.

Η **`run()`** λειτουργεί ως ο διαχειριστής της αλληλεπίδρασης με το μενού και το παιχνίδι, κάνοντας τις απαραίτητες αλλαγές στην κατάσταση του παιχνιδιού με βάση τις ενέργειες του χρήστη.

game_play.py

```python
class GamePlay:
    def __init__(self, screen,numofplayers):
        self.screen = screen
        self.font = pygame.font.SysFont("Arial", 20)  # Larger font for score, etc.
        self.fps_font = pygame.font.SysFont("Arial", 15)  # Smaller font for FPS display
        self.clock = pygame.time.Clock()
        self.start_flag = 1 # for the start only cause the player always start with 0 velocity in the first frame
        self.paused = False #for pause menu
        self.running = True # for loop to start
        # Create the background
        self.bg = Background(self.screen, 'assets/images/stageBck.png', speed=2)  # Adjust the speed to your liking

        #initialize 1 player
        self.player1 = playerCls(self.screen,1)

        #initialize 2 player
        if numofplayers == 2:
            self.SecondPlayer = True
            self.player2 = playerCls(self.screen,2)
        else:
            self.SecondPlayer = False


        # init first platform
        self.platforms = []  # List to hold platforms
        self.initplatform = platformCls(self.screen,self.platforms,globals.platform_width,True)
        self.platforms.append(self.initplatform.platform_rect)

    def draw(self):
        # Update and draw the background
        self.bg.update()  # Move the background
        self.bg.draw()  # Draw the background

        #self.screen.fill((0, 0, 0))
        # Your game elements go here (player, obstacles, etc.)
        score_text = self.font.render(f"Red Score: {self.player1.totalPoints:.2f}", True, (255, 255, 255))
        self.screen.blit(score_text, (10, 20))
```

```python
    def draw(self):
        # Update and draw the background
        self.bg.update()  # Move the background
        self.bg.draw()  # Draw the background

        #self.screen.fill((0, 0, 0))
        # Your game elements go here (player, obstacles, etc.)
        score_text = self.font.render(f"Red Score: {self.player1.totalPoints:.2f}", True, (255, 255, 255))
        self.screen.blit(score_text, (10, 20))

        if self.SecondPlayer:
            score_text = self.font.render(f"Blue Score: {self.player2.totalPoints:.2f}", True, (255, 255, 255))
            self.screen.blit(score_text, (200, 20))

        # Display FPS in the top-right corner with smaller font
        fps_text = self.fps_font.render(f"FPS: {self.clock.get_fps():.2f}", True, (255, 255, 255))
        fps_rect = fps_text.get_rect(topright=(self.screen.get_width() - 20, 20))  # Positioning to top-right
        self.screen.blit(fps_text, fps_rect)

        # Draw players
        self.player1.draw(self.screen,(255, 0, 0))

        if self.SecondPlayer:
            self.player2.draw(self.screen,(0, 0, 255))

        # Draw the platforms (as blue boxes)
        for platform in self.platforms:
            pygame.draw.rect(self.screen, (100, 100, 100), platform)

    def move_platforms(self,platformsList):
        # Move each platform left by 5 pixels per frame
        for platform in platformsList:
            platform.x -= globals.platformVelocity # Move platform to the left
```

```python
#Main game logic loop
def run(self):

    self.show_instructions()  # Show instructions before starting the game
    self.wait_for_keypress()  # Wait for key press to continue with the game


    while self.running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_p:  # Press 'P' to pause
                    #return 2  # Switch to Pause menu
                    self.paused = not self.paused
                    if self.paused:
                        self.mainmenu=0
                        self.pause_game()
                        if self.mainmenu == 1:
                            return 0
                elif event.key == pygame.K_ESCAPE:  # Press 'Escape' to go to main menu
                    return 0  # Exit game and go back to main menu
                elif event.key == pygame.K_r: #and not self.is_falling:  # Change gravity when on platform
                    if self.player1.player_on_object:
                       self.player1.update_gravity()
                       self.player1.player_on_object=False
                       self.player1.calculate_score()
                elif event.key == pygame.K_b and self.SecondPlayer:
                    if self.player2.player_on_object:
                       self.player2.update_gravity()
                       self.player2.player_on_object=False
                       self.player2.calculate_score()


        if not self.paused:

            if self.start_flag == 0:
            # Handle player collision with platforms (player stops at platform)
                if not self.player1.player_velocity == 0:
```

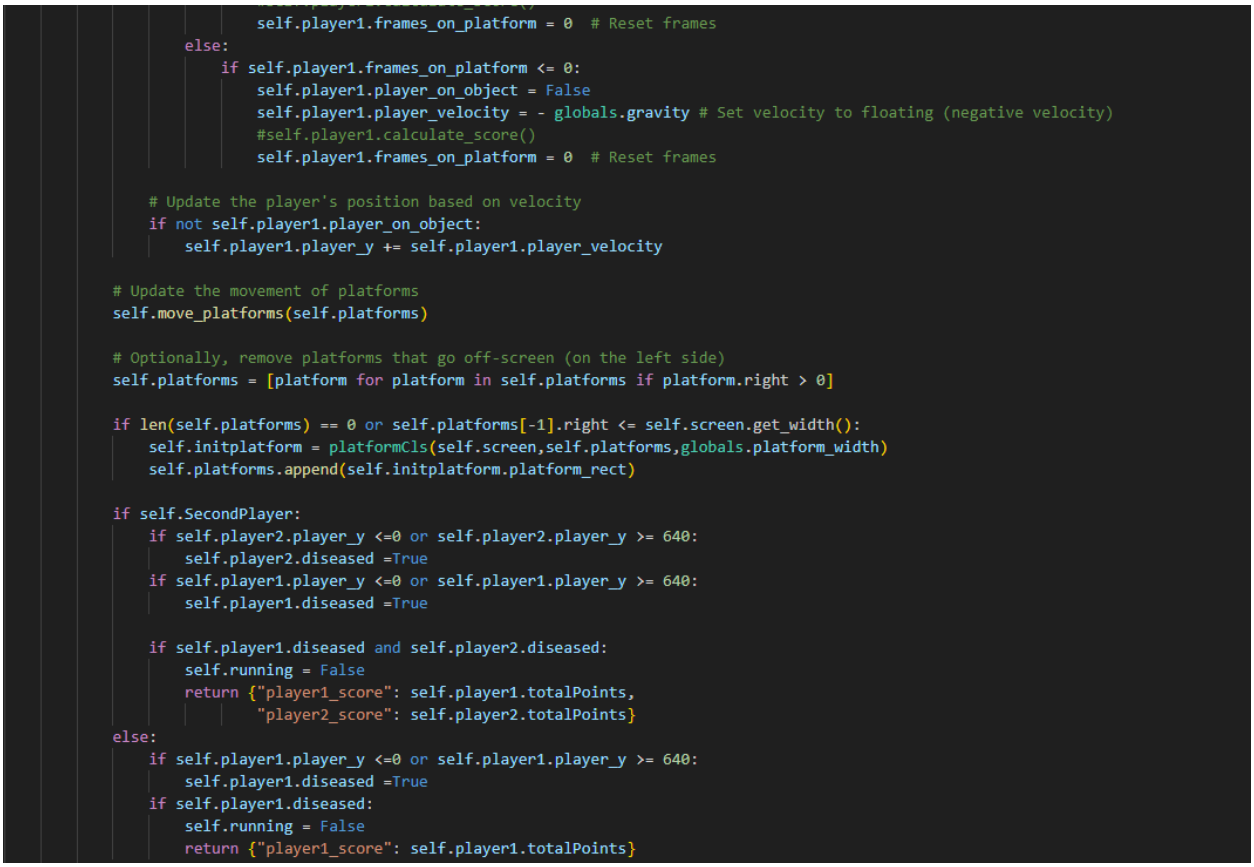
```python
    if not self.paused:

        if self.start_flag == 0:
        # Handle player collision with platforms (player stops at platform)
            if not self.player1.player_velocity == 0:
                self.player1.handle_collisions(self.platforms,globals.platformVelocity)
            if self.SecondPlayer:
                if not self.player2.player_velocity == 0:
                    self.player2.handle_collisions(self.platforms,globals.platformVelocity)
        else:
            self.start_flag = 0

        if self.SecondPlayer:
            if not  self.player2.player_on_object:
            # Apply gravity to make the player move up or down
                if self.player2.gravity_state == self.player2.gravity:  # Gravity pulling down
                   self.player2.player_velocity += self.player2.gravity  # Increase velocity by gravity value
                   if self.player2.player_velocity > globals.playerVelocity:  # Cap the velocity
                      self.player2.player_velocity = globals.playerVelocity - globals.gravity
                elif self.player2.gravity_state == self.player2.gravity_reversed:  # Gravity pulling up
                    self.player2.pla                                              ity  # Decrease velocity by gravity value
                    if self.player2.  (variable) player_velocity: int          Velocity:  # Cap the velocity (negative for reversed gravity)
                       self.player2.player_velocity = - globals.playerVelocity + globals.gravity
            else:
                self.player2.frames_on_platform -= 1

                if self.player2.is_falling:
                    if self.player2.frames_on_platform <= 0:
                        self.player2.player_on_object = False
                        self.player2.player_velocity = globals.gravity  # Set velocity to falling (positive velocity)
                        #self.player2.calculate_score()
                        self.player2.frames_on_platform = 0  # Reset frames
                else:
                    if self.player2.frames_on_platform <= 0:
                        self.player2.player_on_object = False
```

Η κεντρική loopa του `gameplay` διαχειρίζεται τη ροή του παιχνιδιού, ελέγχοντας τη θέση του παίκτη και ενημερώνοντας τις πλατφόρμες.

Κύριες λειτουργίες:

- Παρακολουθεί τη θέση του παίκτη και ελέγχει αν βρίσκεται σε επιτρεπτό σημείο ή αν έχει πέσει στο κενό.
- Ενημερώνει τις πλατφόρμες, δημιουργώντας νέες όταν χρειάζεται και διαγράφοντας όσες δεν είναι πλέον ορατές στη σκηνή, ώστε να εξοικονομούνται πόροι.
- Αλληλεπιδρά με τις υπόλοιπες κλάσεις, όπως `playerCls` και `platformCls`, για να εξασφαλίσει ομαλό gameplay.

Αυτή η loopa διασφαλίζει τη σωστή λειτουργία του παιχνιδιού, ανανεώνοντας συνεχώς το περιβάλλον και κρατώντας το παιχνίδι ενεργό και ισορροπημένο.



```
                    self.player1.frames_on_platform = 0  # Reset frames
            else:
                if self.player1.frames_on_platform <= 0:
                    self.player1.player_on_object = False
                    self.player1.player_velocity = - globals.gravity # Set velocity to floating (negative velocity)
                    #self.player1.calculate_score()
                    self.player1.frames_on_platform = 0  # Reset frames

        # Update the player's position based on velocity
        if not self.player1.player_on_object:
            self.player1.player_y += self.player1.player_velocity

    # Update the movement of platforms
    self.move_platforms(self.platforms)

    # Optionally, remove platforms that go off-screen (on the left side)
    self.platforms = [platform for platform in self.platforms if platform.right > 0]

    if len(self.platforms) == 0 or self.platforms[-1].right <= self.screen.get_width():
        self.initplatform = platformCls(self.screen,self.platforms,globals.platform_width)
        self.platforms.append(self.initplatform.platform_rect)

    if self.SecondPlayer:
        if self.player2.player_y <=0 or self.player2.player_y >= 640:
            self.player2.diseased =True
        if self.player1.player_y <=0 or self.player1.player_y >= 640:
            self.player1.diseased =True

        if self.player1.diseased and self.player2.diseased:
            self.running = False
            return {"player1_score": self.player1.totalPoints,
                    "player2_score": self.player2.totalPoints}
    else:
        if self.player1.player_y <=0 or self.player1.player_y >= 640:
            self.player1.diseased =True
        if self.player1.diseased:
            self.running = False
            return {"player1_score": self.player1.totalPoints}
```

```
                    self.running = False
                    return {"player1_score": self.player1.totalPoints}

            self.player1.update_movement()
            if self.SecondPlayer:
                self.player2.update_movement()

            # Draw the game elements
            self.draw()
            pygame.display.update()

            # Cap the frame rate
            self.clock.tick(60)  # 60 FPS limit



    #-----------------------------------------------------------------------------------------------------------
    #Game instructions Start ---------------
    def draw_text(self, text, y, font_size=32, color=(255, 255, 255)):
        font = pygame.font.SysFont('Arial', font_size)
        text_surface = font.render(text, True, color)
        text_rect = text_surface.get_rect()

        # Center the text horizontally and vertically
        text_rect.centerx = self.screen.get_width() // 2
        text_rect.y = y

        self.screen.blit(text_surface, text_rect)

    def show_instructions(self):
        # Display instructions to the players
        self.screen.fill((0, 0, 0))  # Fill the screen with black background
        self.draw_text('RED changes gravity with "R"', self.screen.get_height() // 2 - 50)
        if self.SecondPlayer:
            self.draw_text('BLUE changes gravity with "B"', self.screen.get_height() // 2)
            self.draw_text('"P" Pause The Game', self.screen.get_height() // 2+50)
            self.draw_text('Press any key to start', self.screen.get_height() // 2 + 100)
        else:
            self.draw_text('"P" Pause The Game', self.screen.get_height() // 2)
```

Η συνάρτηση `show_instructions` είναι υπεύθυνη για την εμφάνιση των οδηγιών του παιχνιδιού όταν αυτό ξεκινά. Δημιουργεί μια loopa που παρουσιάζει στον παίκτη τους χειρισμούς και τις βασικές λειτουργίες του παιχνιδιού. Ο παίκτης μπορεί να αποφασίσει είτε να συνεχίσει με το παιχνίδι, είτε να επιστρέψει στο κύριο μενού.

Κατά τη διάρκεια αυτής της loopa, εμφανίζονται οι οδηγίες, και ο χρήστης έχει τη δυνατότητα να πατήσει ένα κουμπί (π.χ. "r") για να προχωρήσει στο παιχνίδι ή να επιλέξει την επιλογή του να επιστρέψει στο κύριο μενού, εάν το επιθυμεί.

Αυτή η λειτουργία εξασφαλίζει ότι ο παίκτης γνωρίζει τους κανόνες πριν ξεκινήσει και έχει τη δυνατότητα να επιστρέψει στο μενού εάν χρειάζεται περισσότερες πληροφορίες ή να επαναφέρει το παιχνίδι από την αρχή.

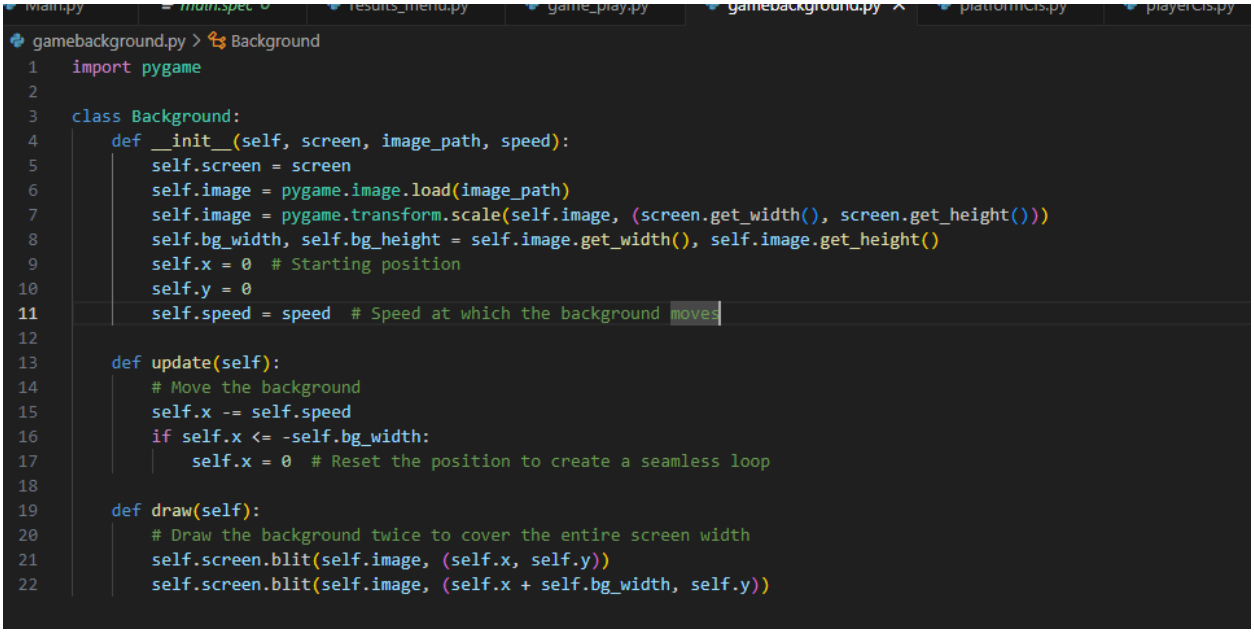
```python
# PAUSE MENU FUNCTIONS START------------
def draw_pause_screen(self):
    font = pygame.font.Font(None, 74)
    text = font.render("PAUSED", True, (255, 0, 0))
    self.screen.blit(text, (self.screen.get_width() // 2 - text.get_width() // 2, self.screen.get_height() // 3))

    resume_text = font.render("Press 'P' to Resume", True, (255, 255, 255))
    self.screen.blit(resume_text, (self.screen.get_width() // 2 - resume_text.get_width() // 2, self.screen.get_height() // 2))

    exit_text = font.render("Press 'ESC' to Exit", True, (255, 255, 255))
    self.screen.blit(exit_text, (self.screen.get_width() // 2 - exit_text.get_width() // 2, self.screen.get_height() // 1.5))

def draw_game_with_pause_overlay(self):
    # Draw the game as usual (background, player, platforms, etc.)
    self.draw()

    # Create a semi-transparent overlay for the frozen effect
    overlay_surface = pygame.Surface(self.screen.get_size())  # Full screen overlay
    overlay_surface.set_alpha(128)  # Set transparency (0-255 range)
    overlay_surface.fill((0, 0, 0))  # Black color with transparency
    self.screen.blit(overlay_surface, (0, 0))  # Draw the overlay on top of the game
    # Draw the pause screen (on top of the overlay)
    self.draw_pause_screen()

def pause_game(self):
    #Handles the pause state.
    while self.paused:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_p:  # Press 'P' again to resume
                    self.paused = False
                elif event.key == pygame.K_ESCAPE:  # Press 'Escape' to go to main menu
                    self.paused = False
                    self.mainmenu = 1
                    return 0  # Exit game
```

Η συνάρτηση `pause_game` επιτρέπει στον χρήστη να κάνει παύση στο παιχνίδι χωρίς να χρειάζεται να δημιουργήσει ξεχωριστό instance. Αυτή η συνάρτηση δημιουργεί μια συνεχιζόμενη loopa, επιτρέποντας στον χρήστη είτε να συνεχίσει το παιχνίδι από το σημείο που το άφησε είτε να επιστρέψει στο κύριο μενού.

Η συνάρτηση `draw_game_with_pause_overlay` εμφανίζει το περιβάλλον του παιχνιδιού με ένα overlay που δηλώνει ότι το παιχνίδι είναι σε παύση, ενώ η `draw_pause_screen` δημιουργεί την οθόνη παύσης που επιτρέπει στον χρήστη να επιλέξει αν θα συνεχίσει ή θα πάει στο κύριο μενού.

Αυτές οι λειτουργίες εξασφαλίζουν ότι ο χρήστης μπορεί να κάνει παύση χωρίς προβλήματα και να επιστρέψει εύκολα στο παιχνίδι ή στο μενού.

gamebackground.py



```python
import pygame

class Background:
    def __init__(self, screen, image_path, speed):
        self.screen = screen
        self.image = pygame.image.load(image_path)
        self.image = pygame.transform.scale(self.image, (screen.get_width(), screen.get_height()))
        self.bg_width, self.bg_height = self.image.get_width(), self.image.get_height()
        self.x = 0  # Starting position
        self.y = 0
        self.speed = speed  # Speed at which the background moves

    def update(self):
        # Move the background
        self.x -= self.speed
        if self.x <= -self.bg_width:
            self.x = 0  # Reset the position to create a seamless loop

    def draw(self):
        # Draw the background twice to cover the entire screen width
        self.screen.blit(self.image, (self.x, self.y))
        self.screen.blit(self.image, (self.x + self.bg_width, self.y))
```

Η **`gamebackground.py`** περιέχει μια κλάση που χρησιμοποιείται μόνο στη σκηνή του gameplay για να δημιουργήσει την αίσθηση κίνησης του φόντου. Αυτή η κλάση επιτρέπει στον χρήστη να νιώθει ότι κινείται στον κόσμο του παιχνιδιού, καθώς το φόντο κινείται συνεχώς.

Η κλάση περιέχει δύο βασικές συναρτήσεις:

1. **`update()`**: Αυτή η συνάρτηση μετακινεί το φόντο κατά ένα συγκεκριμένο αριθμό pixels σε κάθε frame. Η μετακίνηση αυτή δημιουργεί την ψευδαίσθηση ότι το σκηνικό αλλάζει, βοηθώντας στη συνεχιζόμενη κίνηση του παίκτη στον κόσμο του παιχνιδιού.
2. **`draw()`**: Η συνάρτηση αυτή ανανεώνει τη θέση του φόντου σε κάθε frame, έτσι ώστε το φόντο να παραμένει στη σωστή θέση στην οθόνη, ενώ κινείται συνεχώς με βάση τις ρυθμίσεις της σκηνής.

Με αυτές τις δύο συναρτήσεις, το παιχνίδι μπορεί να δημιουργήσει την αίσθηση της κίνησης του κόσμου, προσφέροντας μια πιο ρεαλιστική εμπειρία στον χρήστη.

globals.py

```python
# Difficulty settings (default to False)
difficulty = "Mars(A)"  # Can be "Easy", "Medium", or "Hard"
music_muted = False # Global variable to track whether the music is muted or not
multiplayer = 1 # Global variable to track multiplayer mode
gravity = 0.3 # Global gravity based on difficulty
playerVelocity = 8 #Global max speed based on difficulty
platformVelocity = 5 #Global max speed based on difficulty
platform_width = random.randint(150,300) # Global platofrom width based on diff
upordownobject = "down"
framesextraplat = 5
diffistats = "Mars"


def playersmode(multi):
    global multiplayer
    multiplayer = multi
    print(f"Difficulty set to {multiplayer}")

def upordown(obj):
    global upordownobject
    upordownobject = obj

def set_difficulty(level):
    #Set the difficulty level based on user input.
    #Mars(A) is Easy
    #Earth(E) is Medium
    #Jupiter(J) is Hard

    global difficulty
    global gravity
    global playerVelocity
    global platformVelocity
    global platform_width
    global framesextraplat
    global diffistats

    difficulty = level
```

```python
        playerVelocity = 8 * 1.2
        platformVelocity = 5 * 1.2
        platform_width = random.randint(150,300)
        framesextraplat = 5 * 1.2
        diffistats = "Earth"
    elif difficulty == "Jupiter(J)":
        gravity = 0.3 * 1.8
        playerVelocity = 8 * 1.8
        platformVelocity = 5  * 1.8
        platform_width = random.randint(150,250)
        framesextraplat = 5
        diffistats = "Jupiter"
    else:
        gravity = 0.3
        playerVelocity = 8
        platformVelocity = 5
        platform_width = random.randint(200,300)

    print(f"Difficulty set to {difficulty}")


def toggle_music():
    # Function to toggle music (mute/unmute)
    global music_muted
    if music_muted:
        pygame.mixer.music.unpause()  # Unpause the music
        music_muted = False
    else:
        pygame.mixer.music.pause()  # Pause the music
        music_muted = True


def draw_mute_icon(screen, is_muted):
    if music_muted:
        mute_icon = pygame.image.load('assets/images/mute.png')  # Muted icon
    else:
        mute_icon = pygame.image.load('assets/images/audio.png')  # Unmuted icon

    # Resize the icon to 20x20
    mute_icon = pygame.transform.scale(mute_icon, (20, 20))  # Resize the icon to 20x20 pixels
```

Το **`globals.py`** είναι ένα αρχείο που περιέχει διάφορες συναρτήσεις και μεταβλητές, οι οποίες χρησιμοποιούνται σε πολλές σκηνές του παιχνιδιού. Ο στόχος είναι να παρέχει κοινές λειτουργίες και δεδομένα που χρειάζονται σε διαφορετικά μέρη του παιχνιδιού, ώστε να μην επαναλαμβάνονται και να υπάρχει καλύτερη διαχείριση της ροής του παιχνιδιού.

platformCls.py

```python
class platformCls:
    def __init__(self,screen,platformsList,Gplatform_width,firstplatform=False):

        self.screen = screen
        platform_height = 20
        platform_width = Gplatform_width

        if firstplatform:

            platform_width = 1024  # first platform for players to start
            platform_x = self.screen.get_width()-1024  # Start at the right side of the screen
            platform_y = 620
            globals.upordown("up")
        else:
            last_platform = platformsList[-1]  # Get the most recently created platform
            last_platform_end_x = last_platform.x + last_platform.width
            last_platform_y = last_platform.y

            platform_x = last_platform_end_x + platform_width

            # Calculate vertical difference (Δy) to determine the fall distance
            if globals.upordownobject == "down":
                platform_y = random.randint(350, 600)  # Random vertical position for platforms
                globals.upordown("up")
            elif globals.upordownobject == "up" :
                platform_y = random.randint(40, 290)
                globals.upordown("down")
            else:
                platform_y = random.randint(20, 600)

            delta_y = last_platform_y - platform_y

            # Ensure there's enough distance vertically (for jump/fall mechanics)
            if abs(delta_y) < 100:
                platform_y = last_platform_y - 100 if delta_y > 0 else last_platform_y + 100
```

```python
            globals.upordown("up")
        elif globals.upordownobject == "up" :
            platform_y = random.randint(40, 290)
            globals.upordown("down")
        else:
            platform_y = random.randint(20, 600)

        delta_y = last_platform_y - platform_y

        # Ensure there's enough distance vertically (for jump/fall mechanics)
        if abs(delta_y) < 100:
            platform_y = last_platform_y - 100 if delta_y > 0 else last_platform_y + 100

        # Use gravity to calculate the fall time (t)
        gravity = globals.gravity  # Gravity per frame

        if delta_y > 0:  # Platform is above the current platform (jumping up)
        # Use player velocity and jump mechanics to calculate jump time
            jump_time = math.sqrt((2 * delta_y) / gravity)  # t = sqrt(2 * Δy / a)
            fall_time = 0  # No fall time needed
        elif delta_y < 0:  # Platform is below the current platform (falling)
        # Use gravity to calculate fall time
            fall_time = math.sqrt((-2 * delta_y) / gravity)  # t = sqrt(2 * Δy / a)
            jump_time = 0  # No jump time needed
        else:
            fall_time = 0  # No fall time if the platforms are at the same height
            jump_time = 0  # No jump time if no height difference


        # Calculate the maximum horizontal distance the player can cover in fall_time
        player_velocity = globals.playerVelocity  # Max player horizontal velocity
        platform_velocity = globals.platformVelocity  # Platforms move left speed
        relative_velocity = player_velocity - platform_velocity  # Net horizontal velocity

        # Calculate the max horizontal distance based on the jump or fall time
        max_horizontal_distance = relative_velocity * max(fall_time, jump_time)  # Pixels the player can cover horizontally

        # Calculate where the next platform should be placed
        platform_x = last_platform_end_x + max_horizontal_distance +40  # Add the max horizontal distance
```

```
        # Calculate the maximum horizontal distance the player can cover in fall_time
        player_velocity = globals.playerVelocity  # Max player horizontal velocity
        platform_velocity = globals.platformVelocity  # Platforms move left speed
        relative_velocity = player_velocity - platform_velocity  # Net horizontal velocity

        # Calculate the max horizontal distance based on the jump or fall time
        max_horizontal_distance = relative_velocity * max(fall_time, jump_time)  # Pixels the player can cover horizontally

        # Calculate where the next platform should be placed
        platform_x = last_platform_end_x + max_horizontal_distance +40  # Add the max horizontal distance

    # Create a pygame Rect object for the platform
    self.platform_rect = pygame.Rect(platform_x, platform_y, platform_width, platform_height)


def should_create_platform(self, platformsList,platform_width):
    #Determines whether a new platform should be created based on the last platform's position.
    last_platform = platformsList[-1]  # Get the most recently created platform
    last_platform_end_x = last_platform.x + last_platform.width

    platform_x = last_platform_end_x + platform_width
    # If platform_x exceeds the screen width (1024), return False
    if platform_x > 1024:
        return False
    else:
        return True
```

Η **`platformCls.py`** είναι υπεύθυνη για τη δημιουργία και τη διαχείριση των πλατφορμών του παιχνιδιού. Η κλάση αυτή διασφαλίζει ότι οι πλατφόρμες τοποθετούνται σωστά στον χώρο, με βάση τη ταχύτητα του παίκτη και τη δυσκολία του επιπέδου. Η κύρια λειτουργία της κλάσης είναι να δημιουργεί τις πλατφόρμες με τρόπο ώστε ο παίκτης να μπορεί πάντα να πηδάει από τη μία στην άλλη, χωρίς να υπάρχει κίνδυνος να "πέσει" στο κενό, εκτός και αν δεν προλάβει να πηδήξει.

Η **`should_create_platform()`**:

- Ελέγχει το **πλάτος** της προηγούμενης πλατφόρμας.
- Υπολογίζει το **πλάτος** της νέας πλατφόρμας και τον **χώρο** μεταξύ των πλατφορμών, για να βεβαιωθεί ότι ο παίκτης θα μπορεί να πηδήξει με ασφάλεια πάνω σε αυτήν.

Με αυτή τη μέθοδο, η κλάση εγγυάται ότι το παιχνίδι θα είναι **ευχάριστο και αποδοτικό**, χωρίς να προκαλείται δυσκολία στον παίκτη για να πηδήξει από πλατφόρμα σε πλατφόρμα.

playerCls.py

```
def __init__(self,screen,numofplayers):
    self.screen = screen
    self.player_size = 30

    if numofplayers == 2:
        self.player_x = (self.screen.get_width() * 2 // 6 ) -80 # Starting X position (2/5th of the width)
        self.player_y = self.screen.get_height() // 2  # Start in the middle of the screen vertically
        self.player_images = [
        pygame.image.load('assets/characters/blue/BB1.png'),
        pygame.image.load('assets/characters/blue/BB2.png'),
        pygame.image.load('assets/characters/blue/BB3.png'),
        pygame.image.load('assets/characters/blue/BB4.png'),
        pygame.image.load('assets/characters/blue/BB5.png'),
        pygame.image.load('assets/characters/blue/BB6.png'),
        pygame.image.load('assets/characters/blue/BB7.png'),
        ]
        self.player_images_rev = [
        pygame.image.load('assets/characters/blue/RevB1.png'),
        pygame.image.load('assets/characters/blue/RevB2.png'),
        pygame.image.load('assets/characters/blue/RevB3.png'),
        pygame.image.load('assets/characters/blue/RevB4.png'),
        pygame.image.load('assets/characters/blue/RevB5.png'),
        pygame.image.load('assets/characters/blue/RevB6.png'),
        pygame.image.load('assets/characters/blue/RevB7.png'),
    ]

    else:
        self.player_x = self.screen.get_width() * 2 // 6  # Starting X position (2/5th of the width)
        self.player_y = self.screen.get_height() // 2  # Start in the middle of the screen vertically
        self.player_images = [
        pygame.image.load('assets/characters/red/RR1.png'),
        pygame.image.load('assets/characters/red/RR2.png'),
        pygame.image.load('assets/characters/red/RR3.png'),
        pygame.image.load('assets/characters/red/RR4.png'),
        pygame.image.load('assets/characters/red/RR5.png'),
        pygame.image.load('assets/characters/red/RR6.png'),
        pygame.image.load('assets/characters/red/RR7.png'),
        ]
        self.player_images_rev = [
        pygame.image.load('assets/characters/red/RevR1.png'),
        pygame.image.load('assets/characters/red/RevR2.png'),
```

```
        self.player_velocity = 0  # Starting velocity (no initial speed)
        self.frame_counter = 0  # Frame counter to track frames passed
        self.current_frame = 0  # Start with the first frame
        self.gravity = globals.gravity # Gravity pulling downwards
        self.gravity_reversed = - globals.gravity  # Gravity pulling upwards
        self.gravity_state = self.gravity  # Default gravity is pulling downwards
        self.player_on_object= False # check if player is on object so he can jump
        self.is_falling = True  # Is the player falling?
        self.frames_on_platform = 0 #time left on platform so it can calculate the score
        self.total_frames = 0 # we need total and left on platform to calculate the score
        self.totalPoints = 0 #points gather till he finish the game
        self.diseased = False
        self.playerUporDown = 0 #1 is for up images 0 is down images

    def update_movement(self):
        #Update the player's animation frame."""
        self.frame_counter += 1  # Increment the frame counter

        if self.frame_counter >= 2:  # Every 2 frames
            self.current_frame = (self.current_frame + 1) % len(self.player_images)
            self.frame_counter = 0  # Reset the frame counter
```

```
def draw(self, screen, color):
    """Draw the player on the screen."""
    self.screen = screen
    player_image = self.player_images[self.current_frame]
    player_image_rev = self.player_images_rev[self.current_frame]

    if self.playerUporDown == 0:
        self.screen.blit(player_image, (self.player_x, self.player_y))
    else:
        self.screen.blit(player_image_rev, (self.player_x, self.player_y))
    #pygame.draw.rect(screen, color, (self.player_x, self.player_y, self.player_size, self.player_size))


def handle_collisions(self,platforms,platform_speed):
    player_rect = pygame.Rect(self.player_x, self.player_y, self.player_size, self.player_size)

    if self.player_velocity > 0:  # Falling
        for platform in platforms:
            platform_rect = platform
            if player_rect.bottom  <= platform_rect.top + 15:
                if player_rect.bottom + 15 >= platform_rect.top:        #300
                    if player_rect.right >= platform_rect.left  and player_rect.left <= platform_rect.right :
                        self.player_y = platform_rect.top - self.player_size  # Align player on top of the platform
                        self.player_velocity = 0  # Stop movement
                        self.player_on_object = True  # Mark as standing on a platform

                        # Calculate how many pixels left on the platform
                        player_land_position = player_rect.left  # Player's left position
                        platform_left = platform_rect.left  # Platform's left position
                        platform_right = platform_rect.right  # Platform's right position

                        # Calculate how many pixels player has left on the platform to stand on
                        remaining_space = platform_right - player_land_position  # e.g., 350 - 300 = 50 pixels left

                        # Track platform speed (let's assume the platform moves at a speed of 'platform_speed')
                        #platform_speed = 5  # Example: platform moves 5 pixels per frame to the left
```

```
                        frames_to_fall = remaining_space / platform_speed   # 50 pixels / 5 pixels per frame = 10 frames

                        # Store this value, so you can use it later to detect when to change velocity to falling
                        self.total_frames = frames_to_fall
                        self.frames_on_platform = frames_to_fall + globals.framesextraplat
                        self.is_falling=True
                        self.playerUporDown = 0
                        return  # Exit once collision is handled

    elif self.player_velocity < 0:  # Moving upwards
        for platform in platforms:
            platform_rect = platform
            if player_rect.top >= platform_rect.bottom - 15:
                if player_rect.top - 15 <= platform_rect.bottom:
                    if player_rect.right >= platform_rect.left  and player_rect.left <= platform_rect.right :
                        self.player_y = platform_rect.bottom  # Align player below the platform
                        self.player_velocity = 0  # Stop movement
                        self.player_on_object = True  # Mark as standing under a platform

                        # Calculate how many pixels left on the platform
                        player_land_position = player_rect.left  # Player's left position
                        platform_left = platform_rect.left  # Platform's left position
                        platform_right = platform_rect.right  # Platform's right position
```

```
def update_gravity(self):
#Handles gravity change only when jump is pressed.
    if self.gravity_state == self.gravity:
        self.gravity_state = self.gravity_reversed
    else:
        self.gravity_state = self.gravity

def calculate_score(self):
  #calculate the score he gather based on how long player stayed on platform
    if self.frames_on_platform < 10:
        self.totalPoints = self.totalPoints + self.total_frames
    else:
        self.totalPoints = self.totalPoints + self.total_frames - self.frames_on_platform
```

Η **`playerCls.py`** είναι υπεύθυνη για τη δημιουργία του παίκτη και τη διαχείριση όλων των χαρακτηριστικών του και της φυσικής του στο παιχνίδι. Αυτή η κλάση περιλαμβάνει διάφορες λειτουργίες για την κίνηση, τη βαρύτητα, τη σύγκρουση με τις πλατφόρμες και την παρακολούθηση της κατάστασης του παίκτη.

Κύρια Στοιχεία της Κλάσης **`playerCls.py`**:

1. **Μεταβλητές του Παίκτη**:
    - **`score`**: Αποθηκεύει το σκορ του παίκτη, το οποίο ενημερώνεται κατά τη διάρκεια του παιχνιδιού.
    - **`gravity`**: Υπεύθυνη για την εφαρμογή της βαρύτητας. Ελέγχει αν η βαρύτητα πρέπει να είναι ενεργή ή αν ο παίκτης βρίσκεται σε πλατφόρμα.
2. **Κύριες Λειτουργίες**:
    - **`update_movement()`**: Αυτή η συνάρτηση χρησιμοποιείται για την ενημέρωση της κίνησης του παίκτη σε κάθε frame του παιχνιδιού.
    - **`update_gravity()`**: Η συνάρτηση αυτή ελέγχει την κατάσταση της βαρύτητας για τον παίκτη.
    - **`handle_collision()`**: Αυτή η συνάρτηση είναι υπεύθυνη για τον έλεγχο της σύγκρουσης του παίκτη με τις πλατφόρμες. Αν ο παίκτης έρθει σε επαφή με μια πλατφόρμα, η κατάσταση του αλλάζει από **"falling"** ή **"jumping"** σε **"on platform"**. Αν ο παίκτης απομακρυνθεί από την πλατφόρμα, επανέρχεται στην κατάσταση πτώσης ή άλματος.
    - **`calculate_score()`**: Αυτή η συνάρτηση υπολογίζει το σκορ του παίκτη με βάση τη διάρκεια της παραμονής του σε πλατφόρμα και τις κινήσεις του στο παιχνίδι. Όταν ο παίκτης απομακρύνεται από μια πλατφόρμα ή κάνει άλμα, το σκορ ενημερώνεται.

Με αυτές τις βασικές λειτουργίες, η **`playerCls.py`** διαχειρίζεται την κίνηση, τη φυσική και το σκορ του παίκτη, προσφέροντας μία ολοκληρωμένη εμπειρία παιχνιδιού που ανταποκρίνεται στις αλληλεπιδράσεις του παίκτη με το περιβάλλον του παιχνιδιού.

results_menu.py

```python
class ResultsMenu:
    def __init__(self, screen, result_data):
        self.screen = screen
        self.result_data = result_data
        self.font = pygame.font.Font(None, 40)
        self.running = True
        self.updateresults = True
        self.scoreCls = scoreCls()

    def run(self):

        # Format the scores with two decimal places
        player1_score =  round(self.result_data['player1_score'],2)
        if globals.multiplayer == 2 :
            player2_score =  round(self.result_data['player2_score'],2)

        if self.updateresults:
            if globals.multiplayer == 2 :
               self.scoreCls.save_score("RED",player1_score,globals.diffistats)
               self.scoreCls.save_score("BLUE",player2_score,globals.diffistats)
               self.updateresults =False
            else:
               self.scoreCls.save_score("RED",self.result_data['player1_score'],globals.diffistats)

        while self.running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_r:  # Restart game
                        return 1  # Return to gameplay state
                    elif event.key == pygame.K_ESCAPE:  # Go back to main menu
                        return 0 # Return to main menu state

            # Draw results screen
            self.screen.fill((0, 0, 0))  # Clear screen

            if globals.multiplayer == 2 :
                # Display scores
```

Η **`results_menu.py`** είναι η σκηνή που εμφανίζεται μετά το τέλος του παιχνιδιού, παρέχοντας στους παίκτες την δυνατότητα να δουν τα αποτελέσματά τους και να επιλέξουν αν θέλουν να ξαναπαίξουν ή να επιστρέψουν στο κύριο μενού. Αυτή η σκηνή είναι υπεύθυνη για την ενημέρωση των σκορ στο αρχείο JSON και την εμφάνιση των αποτελεσμάτων στον χρήστη.

Βασικές λειτουργίες της `results_menu.py`:

1. **Ενημέρωση του Αρχείου JSON με τα Αποτελέσματα**:
   - Μόλις ο παίκτης ολοκληρώσει το παιχνίδι, η σκηνή **`results_menu.py`** θα ενημερώσει το αρχείο JSON με το νέο σκορ του παίκτη, χρησιμοποιώντας τις συναρτήσεις που παρέχονται από την κλάση **`scoreCls`** (π.χ., `save_score()`).
2. **Εμφάνιση των Αποτελεσμάτων στον Χρήστη**:
   - Η σκηνή εμφανίζει τα αποτελέσματα του παιχνιδιού (π.χ., το σκορ του παίκτη) στην οθόνη και δίνει τη δυνατότητα στον παίκτη να τα δει με έναν καθαρό και κατανοητό τρόπο.
3. **Περιμένοντας την Επιλογή του Χρήστη**:
   - Μετά την εμφάνιση των αποτελεσμάτων, η σκηνή μπαίνει σε έναν "loop" αναμονής για να δει αν ο χρήστης θα επιλέξει να ξαναπαίξει ή να επιστρέψει στο κύριο μενού.
     - Εάν ο παίκτης επιλέξει **"Να Ξαναπαίξει"**, το παιχνίδι ξεκινάει από την αρχή.
     - Εάν ο παίκτης επιλέξει **"Πίσω στο Κύριο Μενού"**, επιστρέφει στο αρχικό μενού για να επιλέξει άλλες ρυθμίσεις ή να παίξει ξανά.

scoreCls.py

```python
class scoreCls:
    def __init__(self, file_path="data/scores.json"):
        self.file_path = file_path
        self._ensure_file_exists()

    def _ensure_file_exists(self):
        """Ensure the JSON file and its directory exist."""
        if not os.path.exists(os.path.dirname(self.file_path)):
            os.makedirs(os.path.dirname(self.file_path))  # Create directory if it doesn't exist

        if not os.path.isfile(self.file_path):
            with open(self.file_path, "w") as file:
                json.dump([], file)  # Create an empty JSON array
            print(f"Created a new JSON file at {self.file_path}")

    def save_score(self, player_name, score, difficalty):
        """Save a player's score to the JSON file."""
        data = self.load_scores()  # Load existing data
        data.append({"name": player_name, "score": score, "diff": difficalty})

        # Save updated data back to the JSON file
        with open(self.file_path, "w") as file:
            json.dump(data, file, indent=4)
        print(f"Saved {player_name}'s score to {self.file_path}")

    def load_scores(self):
        """Load all scores from the JSON file."""
        if os.path.getsize(self.file_path) == 0:  # Check if the file is empty
            return []  # Return an empty list if the file is empty
        with open(self.file_path, "r") as file:
            return json.load(file)

    def get_top_scores(self, top_n=5):
        """Retrieve the top N scores."""
        data = self.load_scores()
        sorted_scores = sorted(data, key=lambda x: x['score'], reverse=True)  # Sort by score descending
        return sorted_scores[:top_n]

    def reset_scores(self):
        """Reset the JSON file to an empty state."""
        with open(self.file_path, "w") as file:
            json.dump([], file)
```

Η **`scoreCls`** διαχειρίζεται τα σκορ του παιχνιδιού και διασφαλίζει ότι αυτά αποθηκεύονται σωστά στο αρχείο **JSON** και ενημερώνονται κατάλληλα μετά από κάθε παιχνίδι.

Οι βασικές συναρτήσεις της **`scoreCls`** είναι:

1. **`ensure_file_exists()`**: Αυτή η συνάρτηση ελέγχει αν υπάρχει το αρχείο JSON που αποθηκεύει τα σκορ. Αν δεν υπάρχει, το δημιουργεί, εξασφαλίζοντας ότι το αρχείο είναι διαθέσιμο για αποθήκευση.
2. **`save_score()`**: Αυτή η συνάρτηση αποθηκεύει τα σκορ των παικτών στο αρχείο JSON, μετά από κάθε παιχνίδι. Ενημερώνει το αρχείο με τα τρέχοντα σκορ, προσθέτοντας τα στον υπάρχοντα κατάλογο των σκορ.
3. **`get_top_scores()`**: Χρησιμοποιείται στο **`main_menu.py`** και ανακτά τις καλύτερες επιδόσεις από το αρχείο JSON. Επιτρέπει στο παιχνίδι να εμφανίσει τις κορυφαίες βαθμολογίες στους χρήστες, παρέχοντας τους έτσι μια ανασκόπηση των καλύτερων παικτών.

4. **`reset_scores()`**: Αυτή η συνάρτηση καθαρίζει εντελώς το αρχείο JSON, αφαιρώντας όλα τα αποθηκευμένα σκορ, το οποίο είναι χρήσιμο όταν θέλουμε να "επαναφέρουμε" το παιχνίδι σε καθαρή κατάσταση ή να επανεκκινήσουμε τα σκορ.

Με αυτές τις συναρτήσεις, το **`scoreCls`** διασφαλίζει ότι τα σκορ αποθηκεύονται και ανακτώνται σωστά από το αρχείο **JSON**, ενώ παρέχει και τη δυνατότητα για επαναφορά των σκορ όταν είναι απαραίτητο.

Συμπεράσματα και Μελλοντικές Βελτιώσεις

Το "ReverseG" είναι ένα διασκεδαστικό παιχνίδι με μηχανισμό αντιστροφής της βαρύτητας, βασισμένο σε Pygame. Καλύπτει τις βασικές λειτουργίες όπως κίνηση, σκορ και multiplayer.

Συμπεράσματα:

- Το παιχνίδι προσφέρει πρόκληση μέσω των επιπέδων δυσκολίας.
- Η αποθήκευση σκορ μέσω JSON επιτρέπει τη συνεχιζόμενη παρακολούθηση των επιδόσεων.

Μελλοντικές Βελτιώσεις:

- Προσθήκη νέων επιπέδων και εμποδίων.
- Βελτίωση του multiplayer και AI.
- Αναβάθμιση γραφικών και ήχου.
- Περισσότερες επιλογές αποθήκευσης δεδομένων.

https://github.com/NikolasNinios/ReverseG_PyGame