

**Πανεπιστήμιο  
Δυτικής Μακεδονίας**



**Τμήμα Επικοινωνίας και Ψηφιακών Μέσων**

**RECYCLE RANGER**

**«Τεκμηρίωση της Υλοποίησης και της  
Λειτουργικότητας του Παιχνιδιού»**

**Βέργος Νικόλαος Α.Μ 20**

**Μελισσουργάκης Ιωάννης Α.Μ 32**

Καστοριά 2024

GitHub Repository

[https://github.com/sv2nru/Recycle\\_Ranger](https://github.com/sv2nru/Recycle_Ranger)

## Πίνακας περιεχομένων

Περιγραφή παιχνιδιού .....	5
Δομή φακέλων και αρχείων .....	6
Σχεδιασμός επιπέδων .....	6
Περιγραφή κλάσεων .....	7
Κλάση Player.....	7
Κλάση World.....	8
Κλάση Enemies.....	9
Κλάση Button .....	9
Κλάση Garbage.....	10
Κλάση Recyclebin .....	10
Κλάση Cans.....	10
Κλάση Platform .....	10
Περιγραφή κυρίως προγράμματος .....	12
Εισαγωγή βιβλιοθηκών και αρχικοποίηση τους.....	13
Μεταβλητές.....	14
Δημιουργία παραθύρου.....	16
Εικόνες.....	17
Ήχοι.....	17
Αρχικοποίηση αντικείμενων .....	17
Sprite Groups.....	18
Κυρίως πρόγραμμα .....	18
Κυρίως βρόγχος του παιχνιδιού.....	19
Καθορισμός του ρυθμού ανανέωσης .....	19
Εμφάνιση αρχικού μενού.....	19
Εμφάνιση επιπέδου και εκτέλεση παιχνιδιού .....	20
gameState == "Playing" .....	20
gameState == "GameOver" .....	21
gameState == "NextLevel" .....	21
Ανανέωση την οθόνης και εμφάνιση γραφικών.....	23
Έξοδος από το παιχνίδι .....	23
Συναρτήσεις .....	23
Η συνάρτηση resetLevel() .....	23
Η συνάρτηση drawText().....	24

Η συνάρτηση inputName() .....	24
Η συνάρτηση scoreBoard() .....	24
Κίνηση του player .....	25
Animation .....	26
Προσομοίωση της βαρύτητας .....	27
Συγκρούσεις – Collisions .....	27
Συγκρούσεις με τα πλακίδια – Tiles collisions .....	27
Συγκρούσεις με τα Sprite Group .....	28
Συγκρούσεις με τις κινούμενες πλατφόρμες .....	29
Κίνηση των Enemies .....	29
Σχεδιασμός των επιπέδων .....	30
Βάση δεδομένων .....	32
Συναρτήσεις .....	32
Η συνάρτηση createConnection() .....	32
Η συνάρτηση executeQuery() .....	32
Η συνάρτηση executeReadQuery() .....	33
Ερωτήματα SQL .....	33
CreateUserTable .....	33
addPlayerScore .....	33
selectHighScores .....	33
Εικόνες – Γραφικά παιχνιδιού .....	34
Βιβλιογραφία .....	35

## Περιγραφή παιχνιδιού

Το παιχνίδι ονομάζεται Recycle Ranger. Πρόκειται για ένα εκπαιδευτικό παιχνίδι που στόχο έχει να εισάγει τα μικρά παιδιά στην ιδέα της ανακύκλωσης. Αποτελεί ένα 2D platform game κατά το οποίο ο παίκτης χειρίζεται ένα χαρακτήρα ο οποίος προσπαθεί να συλλέξει όλα τα χρησιμοποιημένα τενεκεδάκια και να φτάσει στο καλάθι ανακύκλωσης όπου είναι και η έξοδος κάθε επιπέδου. Υπάρχουν συνολικά 5 επίπεδα κλιμακούμενης δυσκολίας τα οποία περιλαμβάνουν εμπόδια και εχθρούς. Ο σχεδιασμός του κώδικα επιτρέπει την εύκολη προσθήκη επιπλέον επιπέδων και εμποδίων διευκολύνοντας μια μελλοντική περαιτέρω εξέλιξή του. Το παιχνίδι έχει αναπτυχθεί χρησιμοποιώντας την βιβλιοθήκη Pygame, μια ευρέως διαδεδομένη βιβλιοθήκη ελεύθερου λογισμικού (Pygame, n.d).

Ο χειρισμός του παίκτη πραγματοποιείται με τα πλήκτρα κατεύθυνσης (δεξί, αριστερό βέλος) και το κενό (space) για το άλμα. Το παιχνίδι εκτελείται σε ένα παράθυρο με ανάλυση 800 x 800 pixel με 40 frames per second (FPS) ρυθμό ανανέωσης.

## Δομή φακέλων και αρχείων




Η δομή αρχείων και φακέλων του παιχνιδιού αποτελείται από 2 αρχεία .py και έναν φάκελο assets όπου περιέχει τις εικόνες, τους ήχους και τις γραμματοσειρές που χρησιμοποιούνται στο παιχνίδι.

- recycleRanger\_game.py : αποτελεί το αρχείο στο οποίο περιλαμβάνεται ο εκτελέσιμος κώδικας του παιχνιδιού.
- levels.py : στο αρχείο αυτό περιέχονται τα δεδομένα που χρησιμοποιούνται για τη δημιουργία των επιπέδων του παιχνιδιού. Αξίζει να σημειωθεί πως η δομή του levels.py επιτρέπει την εύκολη επεξεργασία και την προσθήκη επιπλέον επιπέδων και εμποδίων διευκολύνοντας μια μελλοντική περαιτέρω εξέλιξη του παιχνιδιού.
- scored.sqlite: Αν το αρχείο δεν υπάρχει κατά την εκκίνηση του παιχνιδιού δημιουργείται. Είναι το αρχείο της βάσης δεδομένων SQLite3 και χρησιμοποιείται για την αποθήκευση των scores του παιχνιδιού.

## Σχεδιασμός επιπέδων

Όλα τα επίπεδα αποθηκεύονται σε μια δομή λίστας (levelList) όπου κάθε στοιχείο της αποτελεί ένα δισδιάστατο πίνακα 20 x 20 ακεραίων (level\_1, level\_2 κλπ). Η τιμή του ακεραίου μέσα στο πίνακα προσδιορίζει και το αντικείμενο το οποίο θα τοποθετηθεί σε εκείνη τη θέση σύμφωνα με τον Πίνακα 1. Η λίστα levelList ορίζεται στο αρχείο levels.py και χρησιμοποιείται από την κλάση World για τον σχεδιασμό του κόσμου του παιχνιδιού.

Τιμή	Επεξήγηση	Εικόνα
0	Κενός χώρος	
1	Τοίχος	
2	Έδαφος	
3	Εχθρός	
4	Εμπόδιο	
5	Κάδος Ανακύκλωσης (Έξοδος από το επίπεδο)	

6	Τενεκεδάκια	
7	Πλατφόρμα – Οριζόντια κίνηση	
8	Πλατφόρμα – Κάθετη κίνηση	

Πίνακας 1 - Επεξήγηση των τιμών σχεδίασης του επιπέδου

## Περιγραφή κλάσεων

Ο κώδικας περιλαμβάνει 8 κλάσεις οι οποίες προσδιορίζουν όλα τα αντικείμενα του παιχνιδιού και αναλύονται παρακάτω.

### Κλάση Player

Αποτελεί τη σημαντικότερη κλάση του παιχνιδιού και είναι υπεύθυνη για τον χειρισμό του χαρακτήρα του παίκτη. Περιέχει τις παρακάτω ιδιότητες:

- `playerImagesR`: λίστα με εικόνες (frames) του παίκτη για την κίνηση του προς τα δεξιά
- `playerImagesL`: λίστα με εικόνες (frames) του παίκτη για την κίνηση του προς τα αριστερά
- `rect`: είναι ένα αντικείμενο που αντιπροσωπεύει την περιοχή της οθόνης που καταλαμβάνει η εικόνα
- `imgIndex`: δείκτης για την τρέχουσα εικόνα των λιστών `playerImageR` και `playerImageL`
- `imgCounter`: μετρητής που χρησιμοποιείται στο animation του χαρακτήρα
- `jump`: μετρητής που χρησιμοποιείται για τη προσομοίωση βαρύτητας κατά τη διαδικασία του άλματος
- `isJump`: λογική μεταβλητή που καταχωρεί αν πραγματοποιείται άλμα
- `direction`: καταχωρεί την κατεύθυνση που κινείται ο παίκτης. Λαμβάνει τις τιμές "R" ή "L" για την δεξιά ή αριστερή κατεύθυνση αντίστοιχα
- `deadImage`: Αντικείμενο της εικόνας του παίκτη όταν χάσει
- `width` και `height`: οι διαστάσεις της εικόνας του παίκτη

Η κλάση `player` διαθέτει τις παρακάτω μεθόδους:

- `__init__(x, y)`: Αποτελεί τον constructor της κλάσης και χρησιμοποιείται για να αρχικοποιήσει τις ιδιότητες της κλάσης και τη θέση του παίκτη στις συντεταγμένες (x, y). Μέσω βρόχου επανάληψης, οι λίστες `playerImagesR` και `playerImagesL` αρχικοποιούνται με τις εικόνες από “ranger1.png” έως και “ranger5.png”. Ειδικότερα για τη λίστα `playerImagesL` καλείται η μέθοδος `pygame.transform.flip()` ώστε να εκτελέσει οριζόντια περιστροφή των εικόνων της λίστας `playerImagesR`, εξοικονομώντας με αυτόν τον τρόπο πόρους αποθήκευσης.
- `update(gameState)`: Η κύρια μέθοδος της κλάσης η οποία όταν καλείται ενημερώνει την θέση του παίκτη. Ανάλογα με το αν πατήθηκε το δεξί ή το αριστερό πλήκτρο δημιουργεί animation μεταβάλλοντας την εικόνα του παίκτη με βάση τον δείκτη `imgIndex` στις λίστες `playerImagesR` και `playerImagesL`. Η μεταβλητή `walkAnimation` χρησιμοποιείται για τον έλεγχο της ταχύτητας εναλλαγής των frame του χαρακτήρα ώστε να μπορέσουμε να έχουμε ρεαλιστική κίνηση του χαρακτήρα σε συνάρτηση με την ταχύτητα που κινείται. Αντίστοιχα, σε περίπτωση άλματος η μεταβλητή `jump` αυξάνει μέχρι το 10 μεταβάλλοντας μέσω της `dif_y` την κάθετη θέση του παίκτη προσομοιώνοντας έτσι τη δύναμη της βαρύτητας. Στη μέθοδο εκτελείται επίσης ο έλεγχος σύγκρουσης με τα αντικείμενα του κόσμου όπως το έδαφος, τον τοίχο, τα εμπόδια, παγίδες και εχθρούς. Σε περίπτωση σύγκρουσης με εχθρό ή παγίδα η μεταβλητή `gameState` παίρνει την τιμή “GameOver”, το παιχνίδι τερματίζει και η εικόνα του παίκτη αλλάζει σε `deadImage`. Σε περίπτωση σύγκρουσης με το αντικείμενο της εξόδου (κάδο ανακύκλωσης), η μεταβλητή `gameState` παίρνει την τιμή “NextLevel” ώστε να ο παίκτης να προχωρήσει στο επόμενο επίπεδο. Η τιμή της μεταβλητής `gameState` επιστρέφεται από τη μέθοδο `update()`.

## Κλάση World

Η κλάση είναι υπεύθυνη για τη δημιουργία και τη σχεδίαση του κόσμου του παιχνιδιού. Διαθέτει την ιδιότητα `tileList` η οποία αποτελεί λίστα στην οποία αποθηκεύονται αντικείμενα τύπου πλειάδας (tuple) και δύο μεθόδους:

- `__init__(levelData)`: Η μέθοδος αποτελεί τον constructor της κλάσης και αρχικοποιεί την κλάση δίνοντας τιμές στη λίστα `tileList`. Δέχεται ως όρισμα το αντικείμενο `levelData` το οποίο είναι τύπου λίστας. Χρησιμοποιώντας έναν εμφωλευμένο βρόχο επανάληψης, η μέθοδος διατρέχει τη λίστα `levelData` και προσθέτει αντικείμενα τύπου `image` στη λίστα `tileList` σύμφωνα με τον Πίνακα 1.



- `drawWorld(screen)`: Η μέθοδος σχεδιάζει τον κόσμο του παιχνιδιού στην οθόνη, διατρέχοντας τη λίστα `tileList` και καλώντας τη μέθοδο `screen.blit()` με παραμέτρους τις `tile[0]` και `tile[1]` όπου δηλώνουν την εικόνα και τη θέση της αντίστοιχα.

## Κλάση Enemies

Η κλάση είναι υπεύθυνη για τη δημιουργία και χειρισμό των εχθρών του παιχνιδιού. Κληρονομεί τη βασική της λειτουργικότητα από την κλάση `pygame.sprite.Sprite`. Διαθέτει δύο ιδιότητες (`direction`, `counter`) που χρησιμοποιούνται για την κίνηση των εχθρών και δύο μεθόδους:

- `__init__(x, y)`: Αποτελεί τον constructor και αρχικοποιεί τη θέση του αντικειμένου (`x`, `y`) καθώς επίσης φορτώνει την προκαθορισμένη εικόνα (`roop.png`). Επίσης αρχικοποιούνται και οι τιμές των ιδιοτήτων (`direction = 1`, `counter = 0`)
- `update()`: Ενημέρωση της νέας θέσης του αντικειμένου. Η ιδιότητα `direction` δηλώνει την κατεύθυνση της κίνησης του εχθρού (δεξιά = 1, αριστερά = -1). Σε κάθε κλήση της ο μετρητής `counter` αυξάνει την τιμή του κατά 1. Όταν ξεπεράσει το όριο 40, η τιμή του `direction` πολλαπλασιάζεται με το -1 με αποτέλεσμα να αντιστραφεί η κατεύθυνση της κίνησης.

## Κλάση Button

Η κλάση είναι υπεύθυνη για τη δημιουργία και χειρισμό των κουμπιών που εμφανίζονται στο παιχνίδι και είναι κοινή για όλα τα κουμπιά. Διαθέτει δύο μεθόδους:

- `__init__(x, y, image)`: Αρχικοποίηση του κουμπιού μέσω των ιδιοτήτων της εικόνας και θέσης (`x`, `y`). Το μέγεθός του προσδιορίζεται ανάλογα με τις διαστάσεις της εικόνας του.
- `draw()`: Σχεδίαση του κουμπιού στην οθόνη. Επιστρέφεται λογική μεταβλητή ανάλογα με το αν το πλήκτρο έχει πατηθεί ή όχι για τον έλεγχο των χειρισμών του παίκτη.

## Κλάση Garbage

Η κλάση είναι υπεύθυνη για τη δημιουργία και χειρισμό των εμποδίων στο παιχνίδι. Η βασική της λειτουργικότητα κληρονομείται από τη κλάση `pygame.sprite.Sprite` η οποία περιέχει μεθόδους που επιτρέπουν τη σχεδίαση των αντικειμένων στην οθόνη καθώς και την ανίχνευση σύγκρουσης με άλλο αντικείμενο. Τα αντικείμενα των εμποδίων είναι στατικά, για αυτό και η κλάση δεν υλοποιεί κώδικα κίνησης. Περιλαμβάνει μόνο τη μέθοδο αρχικοποίησης:

- `__init__(x, y)`: Φόρτωση της προκαθορισμένης εικόνας (`garbage.png`) στις συντεταγμένες `x, y`

## Κλάση Recyclebin

Η κλάση είναι υπεύθυνη για τον χειρισμό του αντικειμένου στόχου ολοκλήρωσης του επιπέδου. Σχεδιάζει το σημείο στο οποίο πρέπει να οδηγηθεί ο παίκτης για να ολοκληρώσει το επίπεδο. Η βασική της λειτουργικότητα κληρονομείται από τη κλάση `pygame.sprite.Sprite`. Περιλαμβάνει μόνο τη μέθοδο αρχικοποίησης:

- `__init__(x, y)`: Φόρτωση της προκαθορισμένης εικόνας (`recycle_bin.png`) στις συντεταγμένες `x, y`.

## Κλάση Cans

Η κλάση είναι υπεύθυνη για την εμφάνιση του αντικειμένων που συλλέγει ο παίκτης (τενεκεδάκια). Περιλαμβάνει τη μέθοδο αρχικοποίησης:

- `__init__(x, y)`: Για τα τενεκεδάκια χρησιμοποιούμε 3 διαφορετικές εικόνες (χρώματα τενεκεδάκια) τα οποία επιλέγονται τυχαία μέσω της συνάρτησης `random.randint(1, 3)`. Στη συνέχεια αναλόγως της τιμής που μας έχει δώσει η `random` φορτώνουμε μία από τις 3 εικόνες.

## Κλάση Platform

Η κλάση είναι υπεύθυνη για τη δημιουργία και χειρισμό των κινούμενων πλατφόρμων του παιχνιδιού. Κληρονομεί τη βασική της λειτουργικότητα από την κλάση `pygame.sprite.Sprite`. Διαθέτει τέσσερις ιδιότητες (`direction`, `counter`, `moveX`, `moveY`) που χρησιμοποιούνται για την κίνηση των πλατφόρμων και δύο μεθόδους:

- `__init__(x, y, moveX, moveY)`: Αποτελεί τον constructor και αρχικοποιεί τη θέση του αντικειμένου (x, y) καθώς επίσης φορτώνει την προκαθορισμένη εικόνα (platform.png). Επίσης αρχικοποιούνται και οι τιμές των ιδιοτήτων (direction = 1, counter = 0). Τέλος οι ιδιότητες moveX και moveY χρησιμοποιούνται για να δηλώσουμε αν η πλατφόρμα θα κινείται στον οριζόντιο ή κάθετο άξονα.
- `update()`: Κίνηση της πλατφόρμας. Οι ιδιότητες moveX, moveY δηλώνουν την κατεύθυνση της κίνησης της πλατφόρμας, moveX = 1 οριζόντια κίνηση, moveY = 1 κάθετη κίνηση. Σε κάθε κλήση της ο μετρητής counter αυξάνει την τιμή του κατά 1. Όταν ξεπεράσει το όριο 40, η τιμή του direction πολλαπλασιάζεται με το -1 με αποτέλεσμα να αντιστραφεί η κατεύθυνση της κίνησης.

## Περιγραφή κυρίως προγράμματος

Αυτή η ενότητα εξετάζει τη λειτουργία του κυρίως προγράμματος που βρίσκεται στο αρχείο recycleRanger\_game.py. Η Εικόνα 1 παρουσιάζει το αρχικό μενού ενώ η Εικόνα 2 μία σκηνή από το πρώτο επίπεδο του παιχνιδιού.



Εικόνα 1 - Το αρχικό μενού του παιχνιδιού



Εικόνα 2 - Το πρώτο επίπεδο του παιχνιδιού

## Εισαγωγή βιβλιοθηκών και αρχικοποίηση τους

Οι βιβλιοθήκες που χρησιμοποιούνται και τις οποίες εισάγουμε στο πρόγραμμά μας είναι οι pygame, sys, random και sqlite3.

Επίσης από την pygame εισάγουμε και την mixer η οποία μας είναι χρήσιμη για την αναπαραγωγή των ήχων που χρησιμοποιούμε στο παιχνίδι. Τέλος εισάγουμε το αρχείο levels το οποίο περιέχει τα δεδομένα για τη σχεδίαση των επιπέδων του παιχνιδιού.

```
import pygame
import sys
import random
import sqlite3
from pygame.locals import *
from pygame import mixer
from levels import *
from sqlite3 import Error
```

Στη συνέχεια αρχικοποιούμε την pygame και την mixer ενεργοποιώντας έτσι όλα τα modules που είναι απαραίτητα για το πρόγραμμά μας.

```
# Initialize pygame and mixer
pygame.init()
pygame.mixer.pre_init(44100, -16, 2, 512)
mixer.init()
```

## Μεταβλητές

Ορισμός της ανάλυσης της οθόνης του παιχνιδιού. Αξίζει να σημειωθεί πως σε όλο το παιχνίδι προσπαθήσαμε να μη χρησιμοποιούμε όσο ήταν δυνατό hardcoded τιμές στις μεταβλητές έτσι ώστε να είναι εύκολος ο έλεγχος και η αλλαγή αυτών.

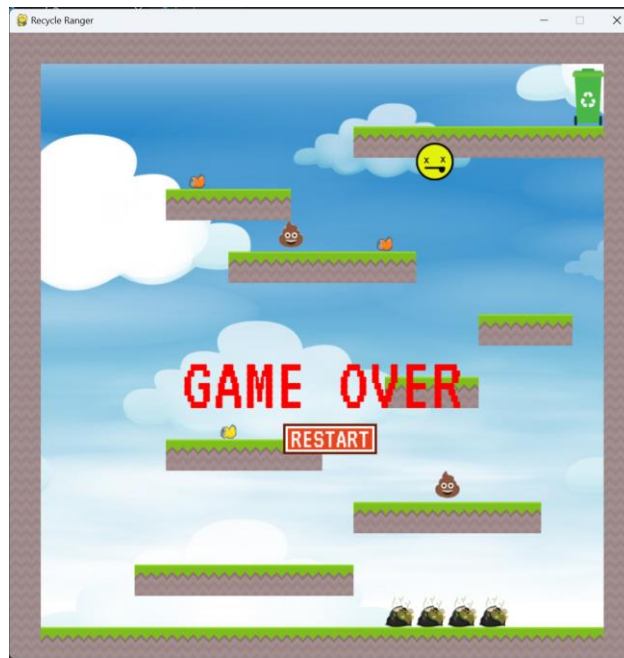
```
# Set game window size
screenWidth = 800
screenHeight = 800
SCREENSIZE = (screenWidth, screenHeight)
```

Ορισμός του clock και του FPS. Η χρήση αυτών των δύο μεταβλητών μας επιτρέπει των έλεγχο ανανέωσης των frame των παιχνιδιών ώστε αυτό να τρέχει με τον ίδιο ρυθμό ανανέωσης άρα και ταχύτητας σε όλους του υπολογιστές.

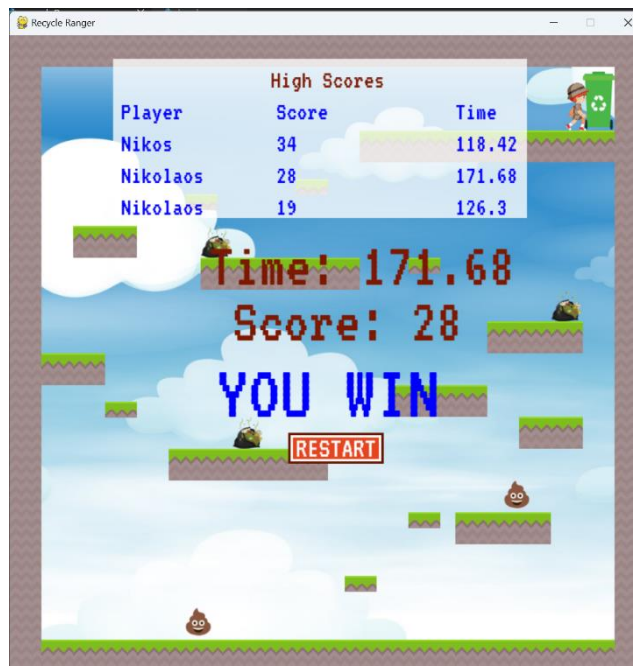
```
# Clock and FPS
clock = pygame.time.Clock()
FPS = 40
```

Ορισμός των γραμματοσειρών που χρησιμοποιούνται στο παιχνίδι καθώς και των διαστάσεων αυτών. Χρησιμοποιούμε μια μεταβλητή την scoreFont για την εμφάνιση του score του παιχνιδιού και την msgFont για την εμφάνιση των μηνυμάτων όπως τα GAME OVER και YOU WIN εικόνες 3 και 4.

```
# Fonts
scoreFont = pygame.font.Font('assets/VT323.ttf', 32)
msgFont = pygame.font.Font('assets/VT323.ttf', 100)
scoreMsgFont = pygame.font.Font('assets/VT323.ttf', 80)
nameFont = pygame.font.Font('assets/VT323.ttf', 40)
```



Εικόνα 2 - Game Over



Εικόνα 3 - You Win

Ορισμός των μεταβλητών που χρησιμοποιούνται στον κώδικα του παιχνιδιού.

tileSize – είναι οι διαστάσεις των tile (πλακιδίων) που δημιουργούν τα επίπεδα του παιχνιδιού. Είναι τύπου integer.

gameState – χρησιμοποιείται για τον έλεγχο της κατάστασης του παιχνιδιού. Στο παιχνίδι λαμβάνει τις τιμές "Playing", "GameOver", "NextLevel". Είναι τύπου string.

mainMenu – είναι τύπου Boolean και χρησιμοποιείται για να ελέγχουμε αν θα εμφανίζουμε στην οθόνη το αρχικό μενού του παιχνιδιού ή όχι.

levelIndex – χρησιμοποιείται ως δείκτης στην λίστα levelList. Είναι τύπου integer.

maxLevel – είναι τύπου integer και χρησιμοποιείται για το ορίσουμε το πλήθος των επιπέδων που περιλαμβάνονται στο παιχνίδι μας. Η levelList ορίζεται στο αρχείο levels.py.

cansScore – χρησιμοποιείται για την καταγραφή του score και είναι τύπου integer.

playerName – αποθηκεύεται το όνομα του παίκτη το οποίο πληκτρολογεί ο παίκτης στην αρχή του παιχνιδιού. Αν δεν πληκτρολογήσει κάποιο όνομα τότε λαμβάνει την τιμή "No Name"

elapsedTime - χρησιμοποιείται για την καταγραφή του χρόνου που προσπελάστηκε καθώς παίζουμε το παιχνίδι.

dbWriteState – Είναι ένα flag το οποίο το χρησιμοποιούμε ώστε να γίνεται μόνο μία φορά η εγγραφή των δεδομένων στην βάση δεδομένων.

```
# Game variables
tileSize = 40
gameState = "Playing"
mainMenu = True
levelIndex = 0
maxLevel = len(levelList) - 1
cansScore = 0
playerName = ""
elapsedTime = 0
dbWriteState = True
```

Οι μεταβλητές WHITE, BLUE, BROWN, RED είναι τύπου tuple και σε αυτές αποθηκεύουμε τις τιμές RGB που θέλουμε για το κάθε χρώμα.

```
# Colors
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
BROWN = (125, 25, 3)
RED = (255, 0, 0)
```

## Δημιουργία παραθύρου

Στη μεταβλητή screen ορίζουμε την τιμή της συνάρτησης pygame.display.set\_mode() η οποία δημιουργεί και επιστρέφει την οθόνη σύμφωνα με τις διαστάσεις του ορίσματος. Στη συνέχεια με την set\_caption() ορίζουμε τον τίτλο που εμφανίζεται στο παράθυρο της εφαρμογής μας.

```
# Create game window
screen = pygame.display.set_mode(SCREENSIZE)
```



```
pygame.display.set_caption('Recycle Ranger')
```

## Εικόνες

Στη συνέχεια φορτώνουμε τις εικόνες που χρησιμοποιούνται στον κώδικά μας όπως είναι το φόντο του κεντρικού μενού και του παιχνιδιού. Οι υπόλοιπες εικόνες φορτώνονται μέσα στις κλάσεις των αντικειμένων.

```
# Load background images
bgImage = pygame.image.load('assets/main_menu.jpg')
bgImage2 = pygame.image.load('assets/sky.jpg')

# Load buttons images
restartImage = pygame.image.load('assets/restart_btn.png')
startImage = pygame.image.load('assets/start_btn.png')
exitImage = pygame.image.load('assets/exit_btn.png')
```

## Ήχοι

Στον κώδικα που παρουσιάζουμε παρακάτω φορτώνουμε τους ήχους που χρησιμοποιούμε στο παιχνίδι μας. Αυτοί είναι το τραγούδι του παρασκηνίου (game\_music\_loop.mp3), ο ήχος όταν ο χαρακτήρας κάνει άλμα (jump.mp3) καθώς και ήχος όταν συλλέγει ένα τενεκεδάκι (pickup.mp3). Το τραγούδι του παρασκηνίου δε χρειάζεται να το ορίσουμε σε κάποια μεταβλητή και να το καλέσουμε. Ξεκινάει να παίζει όταν εκτελείται ο κώδικας μας. Με τις `set_volume()` ορίζουμε την ένταση που θα έχουν οι ήχοι μας ενώ με την `music.play()` με το όρισμα `-1` δηλώνουμε ότι θέλουμε το τραγούδι να παίζει σε loop και με το `2000` να κάνει fade in `2000ms`.

```
# Load sounds
pygame.mixer.music.load('assets/game_music_loop.mp3')
pygame.mixer.music.play(-1, 0.0, 2000)
pygame.mixer.music.set_volume(0.2)
jumpSound = pygame.mixer.Sound('assets/jump.mp3')
jumpSound.set_volume(0.6)
pickupSound = pygame.mixer.Sound('assets/pickup.mp3')
pickupSound.set_volume(0.1)
```

## Αρχικοποίηση αντικείμενων

Αφού ορίσουμε τις κλάσεις και τις συναρτήσεις προχωράμε στο κυρίως πρόγραμμα όπου δημιουργούμε τα διάφορα αντικείμενα των κλάσεων.

Δημιουργούμε ένα αντικείμενο τύπου `Player` το οποίο παίρνει ως ορίσματα τις θέσεις (x, y) που θα εμφανιστεί στην οθόνη.

```
# Create Player instance
player = Player(tileSize, screenHeight-100)
```

Στη συνέχεια δημιουργούμε τα αντικείμενα τύπου `Button` τα οποία παίρνουν σαν ορίσματα τη θέση στην οποία θα εμφανιστούν καθώς και την εικόνα που θα εμφανίσουν

```
# Create Buttons instances
restartButton = Button(screenWidth // 2 - 50, screenHeight // 2 + 100,
restartImage)
startButton = Button(screenWidth // 2 - 340, screenHeight // 2,
startImage)
exitButton = Button(screenWidth // 2 + 70, screenHeight // 2,
exitImage)
```

## Sprite Groups

Όπως αναφέραμε οι κλάσεις `Enemy`, `Garbage`, `Cans Platform` και `RecycleBin` κληρονομούν χαρακτηριστικά και ιδιότητες της `pygame.sprite.Sprite`. Μία από αυτές είναι και τα `sprite.Group()` η οποία μοιάζει με τις λίστες και μας επιτρέπει να αποθηκεύουμε αντικείμενα της κλάσης σε ομάδες. Στη συνέχεια τα χρησιμοποιούμε για να εμφανίσουμε τα αντικείμενα στην οθόνη μας.

```
# Create Sprite Group
enemyGroup = pygame.sprite.Group()
garbageGroup = pygame.sprite.Group()
cansGroup = pygame.sprite.Group()
recycleBinGroup = pygame.sprite.Group()
platformGroup = pygame.sprite.Group()
```

## Κυρίως πρόγραμμα

Ξεκινώντας το κυρίως πρόγραμμα κάνουμε έναν έλεγχο αν το `levelIndex` είναι μικρότερο του μήκους της λίστας `levelList`. Αυτό το κάνουμε για να ελέγξουμε ότι η `levelIndex` και η `levelList` έχουν όντως οριστεί και αρχικοποιηθεί σωστά ώστε να αποφύγουμε κάποιο `error`. Στη συνέχεια αν η συνθήκη είναι αληθής τότε δημιουργούμε ένα αντικείμενο τύπου `World` με όρισμα το πρώτο στοιχείο της λίστας `levelList` όπου στην περίπτωση μας είναι και το πρώτο επίπεδο του παιχνιδιού. Η `runGame` είναι μια μεταβλητή τύπου `Boolean` και χρησιμοποιείται στη συνθήκη ελέγχου του βρόγχου συνεχής εκτέλεσης του παιχνιδιού.

```
# Check if levelIndex is inside the levellist and start the game
if levelIndex < len(levellist):
    world = World(levellist[levelIndex])
    runGame = True
else:
    print("Level index is outside the levellist")
    runGame = False
```

## Κυρίως βρόγχος του παιχνιδιού

Ότι κώδικας βρίσκεται μέσα στον κύριο βρόγχο του παιχνιδιού εκτελείται συνέχεια μέχρι η συνθήκη ελέγχου γίνει ψευδής. Έτσι επιτυγχάνουμε τον συνεχή έλεγχο των εντολών του παίκτη καθώς και την εμφάνιση των αντίστοιχων γραφικών στην οθόνη.

```
# Game Loop
while runGame:
```

## Καθορισμός του ρυθμού ανανέωσης

Η εντολή `clock.tick(FPS)` στην Pygame περιορίζει τον αριθμό των frames που εμφανίζονται στην οθόνη κάθε δευτερόλεπτο, με "FPS" να αντιπροσωπεύει τα "Frames Per Second". Η `clock.tick(FPS)` διασφαλίζει ότι η οθόνη σας δεν ενημερώνεται πιο γρήγορα από το καθορισμένο FPS. Αυτό βοηθά στην ομαλή εκτέλεση του παιχνιδιού εξασφαλίζοντας σταθερό gameplay, ανεξάρτητα από την ταχύτητα του υπολογιστή στον οποίο εκτελείται το παιχνίδι.

## Εμφάνιση αρχικού μενού

Στη συνέχεια εκτέλεσης του προγράμματος εμφανίζουμε την εικόνα του background του αρχικού μενού με την εντολή `screen.blit()` η οποία λαμβάνει ως ορίσματα την εικόνα και τη θέση στην οποία θα εμφανιστεί. Επιπλέον με τη συνάρτηση `drawText()` εμφανίζουμε το κείμενο Recycle Ranger πάνω από την εικόνα του background Εικόνα 1. Για να εμφανιστούν οι αλλαγές στην οθόνη θα πρέπει να εκτελεστεί η `pygame.display.update()`.

```
screen.blit(bgImage, (0, 0))
drawText('Recycle Ranger', msgFont, BROWN, 130, 150)
```

Στη συνέχεια ελέγχουμε αν θέλουμε να εμφανιστεί το αρχικό μενού, αυτό επιτυγχάνεται με τον έλεγχο της μεταβλητής `mainMenu` και ελέγχουμε την τιμή που μας επιστρέφει η `startButton.draw()`. Αν ο παίκτης πατήσει το κουμπί `exit` τότε η εφαρμογή τερματίζει διαφορετικά αν πατήσει το κουμπί `start` τότε συνεχίζουμε στην εμφάνιση της πρώτης πίστας του παιχνιδιού.

```

if mainMenu == True:
    playerName = inputName(playerName)
    if startButton.draw():
        mainMenu = False
        if playerName == "":
            playerName = "No Name"
        startTime = pygame.time.get_ticks() # Start counter time
    if exitButton.draw():
        runGame = False
else:
    world.drawWorld(screen)

```

## Εμφάνιση επιπέδου και εκτέλεση παιχνιδιού

Με την `world.drawWorld(screen)` εμφανίζουμε την πίστα του παιχνιδιού. Στη συνέχεια μέσω διαδοχικών συνθηκών ελέγχων `if` ελέγχουμε την τιμή της `gameState` και αναλόγως την τιμή της εκτελούμε τον αντίστοιχο κώδικα.

`gameState == "Playing"`

Είναι η κατάσταση στην οποία το παιχνίδι εκτελείται και ο παίκτης κινείται μέσα στο παιχνίδι χωρίς να έχει συγκρουστεί (collision) με κάποιον εχθρό ή εμπόδιο.

```

if gameState == "Playing":
    enemyGroup.update()
    if pygame.sprite.spritecollide(player, cansGroup, True):
        pickupSound.play()
        score += 1
        drawText('Score: ' + str(score), scoreFont, WHITE, 40, 10)

    enemyGroup.draw(screen)
    garbageGroup.draw(screen)
    cansGroup.draw(screen)
    recycleBinGroup.draw(screen)
    gameState = player.update(gameState)

```

Η `update()` είναι μέθοδος της κλάσης `Sprite` την οποία κληρονομεί η `Enemies` και την οποία την έχουμε κάνει override προκειμένου να δώσουμε κίνηση στα αντικείμενα τύπου `Enemies`.

```

# Override update method to make enemy moves
def update(self):
    self.rect.x += self.direction
    self.counter += 1
    if abs(self.counter) > 40:

```

```
self.direction *= -1
self.counter *=-1
```

Στη συνέχεια ελέγχουμε αν έχουμε κάποια σύγκρουση (collision) του player με κάποιο αντικείμενο που ανήκει στην cansGroup. Σε περίπτωση που υπάρχει collision μεταξύ των δύο αντικειμένων το αντικείμενο που ανήκει στην cansGroup αφαιρείται από την ομάδα (doKill = True), εκτελείται ο ήχος pickup.mp3 με την pickupSound.play() και η μεταβλητή score αυξάνεται κατά 1. Στη συνέχεια με την συνάρτηση drawText() ενημερώνουμε το κείμενο του score που εμφανίζεται στην οθόνη.

```
gameState == "GameOver"
```

Είναι η κατάσταση στην οποία ο παίκτης έχει συγκρουστεί (collision) με κάποιο εμπόδιο ή εχθρό.

```
if gameState == "GameOver":
    drawText('GAME OVER', msgFont, RED, screenWidth // 2 -180,
screenHeight // 2)
    if restartButton.draw():
        world = resetLevel(levelIndex)
        player = Player(tileSize, screenHeight-100)
        gameState = "Playing"
        score = 0
```

Με τη συνάρτηση drawText() εμφανίζουμε το αντίστοιχο μήνυμα στην οθόνη και στη συνέχεια στον έλεγχο καλούμε την μέθοδο restartButton.draw() η οποία εμφανίζει το κουμπί restart στην οθόνη, ελέγχει την είσοδο που κάνει ο χρήστης με την χρήση του ποντικιού και επιστρέφει True/False αναλόγως με το αν ο χρήστης κλικάρει το κουμπί ή όχι.

Αν η συνθήκη ελέγχου γίνει True τότε κάνουμε reset το επίπεδο στο οποίο βρισκόμαστε, αρκικοποιούμε το αντικείμενο τύπου Player, μηδενίζουμε το score και αλλάζουμε την τιμή της gameState σε "Playing".

```
gameState == "NextLevel"
```

Είναι η κατάσταση στην οποία ο παίκτης έχει συγκρουστεί (collision) με αντικείμενο τύπου recycleBin οπότε και έχει ολοκληρώσει το επίπεδο.

```
if gameState == "NextLevel":
    levelIndex += 1
    if levelIndex <= maxLevel:
        world = resetLevel(levelIndex)
        player = Player(tileSize, screenHeight-100)
        gameState = "Playing"
```

Αυξάνουμε το levelIndex κατά 1 ώστε να καλέσουμε το επόμενο στοιχείο της λίστας levelList άρα και να προχωρήσουμε στο επόμενο επίπεδο. Στη συνέχεια ελέγχουμε αν υπάρχει άλλο επίπεδο να καλέσουμε στο παιχνίδι μας ή έχουμε ολοκληρώσει όλα τα επίπεδα. Αυτό επιτυγχάνεται με την συνθήκη ελέγχου if levelIndex <= maxLevel

Στην περίπτωση που είναι αληθής καλούμε τη συνάρτηση resetLevel() η οποία πραγματοποιεί αρχικοποίηση του επιπέδου και των αντικείμενων που υπάρχουν στο επίπεδο, αρχικοποιούμε το αντικείμενο τύπου Player και το εμφανίζουμε στην αντίστοιχη θέση στην οθόνη και ορίζουμε την τιμή της μεταβλητής gameState σε "Playing" ώστε να συνεχίσει η εκτέλεση του παιχνιδιού στο νέο επίπεδο.

Στην περίπτωση που δεν υπάρχει άλλο επίπεδο για να συνεχίσουμε, δηλαδή levelIndex > maxLevel, τότε εμφανίζουμε τα αντίστοιχα μηνύματα στην οθόνη YOU WIN, το τελικό Score και χρόνο του παίκτη καθώς και το High Score Board με την συνάρτηση scoreboard(). Με την μέθοδο restartButton.draw() εμφανίζουμε το κουμπί restart και ελέγχουμε την είσοδο που κάνει ο χρήστης με την χρήση του ποντικιού. Στην περίπτωση που ο χρήστης κλικάρει το κουμπί τότε μηδενίζουμε το cansScore, elapsedTime και το levelIndex ώστε να ξεκινήσουμε το παιχνίδι από το πρώτο επίπεδο. Αρχικοποιούμε τα world και player και ορίζουμε την gameState σε "Playing".

```
else:
    drawText('YOU WIN', msgFont, BLUE, screenWidth // 2 -
140, screenHeight // 2)
    drawText('Score: ' + str(cansScore), scoreMsgFont,
BROWN, screenWidth // 2 - 120, screenHeight // 2 - 80)
    drawText(f"Time: {elapsedTime:.2f}", scoreMsgFont,
BROWN, screenWidth // 2 - 150, screenHeight // 2 - 150)
    addPlayerScore = f"INSERT INTO highscores(playername,
score, time) VALUES ('{playerName}', '{cansScore}',
'{elapsedTime:.2f}')"
    if dbWriteState:
        executeQuery(connection, addPlayerScore)
        dbexport = executeReadQuery(connection,
selectHighScores) # Return the 3 best scores
        print(dbexport)
        dbWriteState = False # Flag to write only once to
the database

    scoreBoard() # Show high scores board
    if restartButton.draw():
        levelIndex = 0
        world = resetLevel(levelIndex)
        player = Player(tileSize, screenHeight-100)
        cansScore = 0
        elapsedTime = 0 # Reset Timer
        startTime = pygame.time.get_ticks() # Reset timer
        gameState = "Playing"
        dbWriteState = True
```

## Ανανέωση την οθόνης και εμφάνιση γραφικών

Η `pygame.display.update()` ενημερώνει την οθόνη με τις αλλαγές που έχουν γίνει από την τελευταία φορά που κλήθηκε και ενημερώνονται τα γραφικά στην οθόνη.

```
pygame.display.update()
```

## Έξοδος από το παιχνίδι

Με την `pygame.event.get()` ελέγχουμε την είσοδο του παίκτη και σε περίπτωση που ο χρήστης επιλέξει να κλείσει το παράθυρο δηλαδή το `event.type == pygame.QUIT` τότε η μεταβλητή `runGame` λαμβάνει την τιμή `False`, βγαίνουμε από τον βρόγχο και καλούμε την `pygame.quit()`

```
for event in pygame.event.get():  
    if event.type == pygame.QUIT:  
        runGame = False
```

```
pygame.quit()
```

## Συναρτήσεις

Στο κώδικα του παιχνιδιού χρησιμοποιούνται δύο συναρτήσεις.

### Η συνάρτηση `resetLevel()`

Η συνάρτηση `resetLevel(levelIndex)` παίρνει σαν όρισμα το `levelIndex` το οποίο είναι ο δείκτης για το επίπεδο που θέλουμε να φορτώσουμε στο παιχνίδι. Αυτό επιτυγχάνεται δημιουργώντας ένα αντικείμενο τύπου `World`. Στη συνέχεια αδειάζουμε τα `enemyGroup`, `garbageGroup`, `recycleBinGroup` και `cansGroup` ώστε στην συνέχεια τα `Group` να “γεμίσουν” με τα αντικείμενα και τις αντίστοιχες θέσεις τους σύμφωνα με τον σχεδιασμό του κάθε επιπέδου.

```
def resetLevel(levelIndex):  
    enemyGroup.empty()  
    garbageGroup.empty()  
    recycleBinGroup.empty()  
    cansGroup.empty()  
    world = World(levelList[levelIndex])  
    return world
```

Τέλος η συνάρτηση επιστρέφει ένα αντικείμενο τύπου World.

### Η συνάρτηση drawText()

Η συνάρτηση drawText(text, font, color, x, y) χρησιμοποιείται για την εμφάνιση των κειμένων στην οθόνη. Σας ορίσματα δέχεται το κείμενο που θέλουμε να εμφανίσουμε, την γραμματοσειρά, το χρώμα που θα έχει το κείμενο, καθώς και τις θέσεις x y στις οποίες θα εμφανιστεί το κείμενο.

```
def drawText(text, font, color, x, y):
    textImage = font.render(text, True, color)
    screen.blit(textImage, (x, y))
```

### Η συνάρτηση inputName()

Η συνάρτηση inputName(playerName) χρησιμοποιείται για να πάρουμε είσοδο από τον χρήστη. Συγκεκριμένα του εμφανίζει το σχετικό κείμενο και ένα λευκό πλαίσιο όπου μέσα σε αυτό πληκτρολογεί το όνομα του. Με τους κατάλληλους ελέγχους έχουμε την δυνατότητα να σβήσουμε χαρακτήρες καθώς και να περιορίσουμε τον όνομα στους 12 χαρακτήρες.

```
def inputName(playerName):
    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_BACKSPACE:
                playerName = playerName[:-1]
            else:
                if len(playerName) < 12: # maximum 12 characters
                    playerName += event.unicode
    drawText('Enter your Name: ', nameFont, BROWN, 130, 250)
    pygame.draw.rect(screen, WHITE, uiRect)
    inputSurface = nameFont.render(playerName, True, BROWN)
    screen.blit(inputSurface, (uiRect.x+5, uiRect.y))
    uiRect.w = max(150, inputSurface.get_width()+10)
    return playerName
```

### Η συνάρτηση scoreBoard()

Η συνάρτηση scoreBoard() χρησιμοποιείται για να εμφανίσουμε στην οθόνη έναν πίνακα ο οποίος περιέχει τον τίτλο High Score, τρεις επικεφαλίδες (Player, Score, Time) και στην συνέχεια από κάτω με μια δομή επανάληψης εμφανίζουμε τα αποτελέσματα που βρίσκονται στην μεταβλητή dbexport.



```

# Show high scores board
def scoreBoard():
    rect = pygame.Surface((520,200), pygame.SRCALPHA, 32)
    rect.fill((255, 255, 255, 200))
    screen.blit(rect, (screenWidth / 2 - 270, 30))

    header = scoreFont.render("High Scores", True, BROWN)
    screen.blit(header, (screenWidth / 2 - header.get_width() / 2, 40))
    header1 = scoreFont.render("Player", True, BLUE)
    screen.blit(header1, (screenWidth / 2 - 260, 80))
    header2 = scoreFont.render("Score", True, BLUE)
    screen.blit(header2, (screenWidth / 2 - header2.get_width(), 80))
    header3 = scoreFont.render("Time", True, BLUE)
    screen.blit(header3, (screenWidth / 2 + 160, 80))

    y_pos = 120
    for playername, score, time in dbexport:
        playerText = scoreFont.render(playername, True, BLUE)
        scoreText = scoreFont.render(str(score), True, BLUE)
        timeText = scoreFont.render(str(time), True, BLUE)
        screen.blit(playerText, (screenWidth / 2 - 260, y_pos))
        screen.blit(scoreText, (screenWidth / 2 - header2.get_width(),
y_pos))
        screen.blit(timeText, (screenWidth / 2 + 160, y_pos))
        y_pos += 40

```

## Κίνηση του player

Με τη μέθοδο `update()` της κλάσης `Player` επιτυγχάνουμε την κίνηση του player. Ο χρήστης έχει να τη δυνατότητα να κινήσει τον player με τα βελάκια αριστερά και δεξιά ή να κάνει άλμα με το πλήκτρο `space`. Στην κίνηση του παίκτη χρησιμοποιούμε δύο μεταβλητές τις `dif_x` για τον έλεγχο της κίνησης στο άξονα `x` (οριζόντια κίνηση) και την `dif_y` για τον έλεγχο της κίνησης στο άξονα `y` (κάθετη κίνηση, άλμα). Έτσι αν το παιχνίδι βρίσκεται σε κατάσταση εκτέλεσης, `gameState == "Playing"` τότε πατώντας τα αντίστοιχα πλήκτρα κινούμε τον χαρακτήρα στην οθόνη. Οι `imageCounter` και `direction` χρησιμοποιούνται στον έλεγχο της κατεύθυνσης στην οποία κινείται ο player και εμφάνιση του αντίστοιχου animation.

```

def update(self, gameState):
    dif_x = 0
    dif_y = 0
    walkAnimation = 5

    if gameState == "Playing":
        # Player movement
        key = pygame.key.get_pressed()
        if key[pygame.K_LEFT]:
            dif_x -= 5

```

```

        self.imgCounter += 1
        self.direction = "L"
    if key[pygame.K_RIGHT]:
        dif_x += 5
        self.imgCounter += 1
        self.direction = "R"
    if key[pygame.K_SPACE] and self.isJump == False:
        jumpSound.play()
        self.jump = -15
        self.isJump = True

```

Στην περίπτωση που ο χρήστης δεν πατάει κάποιο από τα πλήκτρα RIGHT και LEFT τότε αναλόγως προς ποια κατεύθυνση κινούνταν ο παίκτης εμφανίζουμε την αντίστοιχη εικόνα.

```

if key[pygame.K_RIGHT] == False and key[pygame.K_LEFT] == False:
    self.imgCounter = 0
    self.imgIndex = 0
    if self.direction == "R":
        self.image = self.playerImagesR[self.imgIndex]
    if self.direction == "L":
        self.image = self.playerImagesL[self.imgIndex]

```

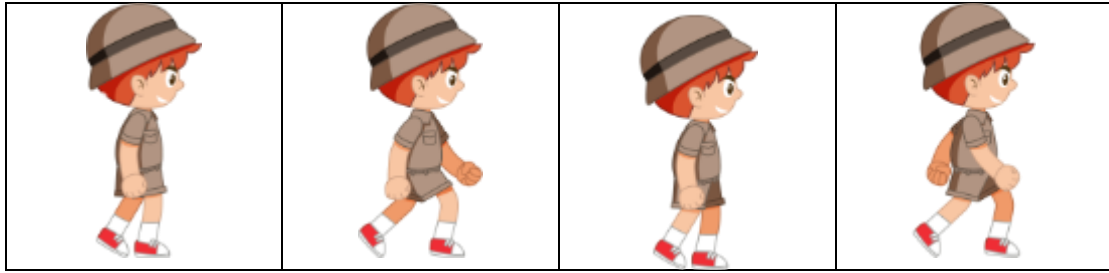
## Animation

Η κίνηση του χαρακτήρα (animation) επιτυγχάνεται με την εναλλαγή εικόνων που δείχνουν τον χαρακτήρα σε διαφορετικά στάδια καθώς περπατάει Πίνακας 2. Η μεταβλητή walkAnimation χρησιμοποιείται για να ελέγξουμε την ταχύτητα με την οποία γίνεται η εναλλαγή των εικόνων. Αναλόγως αν ο παίκτης κινείται δεξιά ή αριστερά εμφανίζουμε διαδοχικά τις εικόνες που είναι αποθηκευμένες στη λίστα playerImagesR ή playerImagesL αντίστοιχα.

```

# Player image animation
if self.imgCounter > walkAnimation:
    self.imgIndex += 1
    if self.imgIndex >= len(self.playerImagesR):
        self.imgIndex = 0
    if self.direction == "R":
        self.image = self.playerImagesR[self.imgIndex]
    if self.direction == "L":
        self.image = self.playerImagesL[self.imgIndex]
    self.imgCounter = 0

```



Πίνακας 2 - Εικόνες καρτέ του χαρακτήρα Player

## Προσομοίωση της βαρύτητας

Η προσομοίωση της βαρύτητας επιτυγχάνεται μέσω της μεταβλητής `jump`. Όταν ο χρήστης πατήσει το πλήκτρο `space` και ο παίκτης είναι σε θέση να κάνει άλμα, δηλαδή βρίσκεται στο έδαφος, τότε η `jump` παίρνει τιμή `-15`. Σε κάθε ανανέωση η τιμή `dif_y` λαμβάνει την τιμή `dif_y = dif_y + jump`. Ο παίκτης φτάνει στην ανώτερη τιμή `-15` και σιγά σιγά μειώνεται μέχρι να φτάσει στην τιμή `10` όπου και την κρατάμε σταθερή. Με αυτόν τον τρόπο ο παίκτης μας έχει ομαλή πτώση πετυχαίνοντας έτσι ωραιότερο οπτικό αποτέλεσμα στην κάθετη κίνηση του παίκτη.

```
# Add some gravity
self.jump += 1
if self.jump > 10:
    self.jump = 10
dif_y += self.jump
```

## Συγκρούσεις – Collisions

Μέσα στη μέθοδο `update()` της κλάσης `Player` γίνεται και ο έλεγχος των συγκρούσεων (`collisions`) με τα υπόλοιπα αντικείμενα του παιχνιδιού.

Με τη χρήση της `dif_x` και `dif_y` αποφεύγουμε επίσης προβλήματα όπως ο παίκτης να εμφανίζεται μέσα στα αντικείμενα με τα οποία συγκρούεται καθώς ο έλεγχος των συγκρούσεων γίνεται πριν εμφανίσουμε τη νέα θέση του παίκτη στην οθόνη. Έτσι αναλόγως με τον αν έχουμε κάποια σύγκρουση ή όχι, η νέα θέση του παίκτη ενημερώνεται ανάλογα.

## Συγκρούσεις με τα πλακίδια – Tiles collisions

Με μία δομή επανάληψης ελέγχουμε για κάθε πλακίδιο (`tile`) που βρίσκεται στη λίστα `tileList` αν ο παίκτης κάνει σύγκρουση με κάποιο από αυτά. Για να το πετύχουμε αυτό χρησιμοποιούμε τη συνάρτηση `colliderect()`. Σαν ορίσματα λαμβάνει τη θέση που θα έχει ο παίκτης καθώς και τις διαστάσεις του `rect` του `player`. Στον κώδικα χρησιμοποιούμε το `tile[1]`

γιατί εκεί βρίσκεται αποθηκευμένη η πληροφορία για το rect των πλακιδίων. Στο tile[0] είναι αποθηκευμένη η εικόνα.

```
# Check for collision with tiles
for tile in world.tileList:
    # Check for collision in x axis
    if tile[1].colliderect(self.rect.x + dif_x,
self.rect.y, self.width, self.height):
        dif_x = 0
    # Check for collision in y axis
    if tile[1].colliderect(self.rect.x, self.rect.y +
dif_y, self.width, self.height):
        # Disable double jumps
        if key[pygame.K_SPACE] == False:
            self.isJump = False
        # Check if below the ground - jumping
        if self.jump < 0:
            dif_y = tile[1].bottom - self.rect.top
            self.jump = 0
        # Check if above the ground - falling
        elif self.jump >= 0:
            dif_y = tile[1].top - self.rect.bottom
            self.jump = 0
```

### Συγκρούσεις με τα Sprite Group

Για τον έλεγχο των συγκρούσεων του παίκτη με τα αντικείμενα των κλάσεων Enemies, Garbage, Cans και Recyclebin χρησιμοποιούμε τη μέθοδο που κληρονομούν οι παραπάνω κλάσεις από την Sprite την `spritecollide()` η οποία λαμβάνει σαν ορίσματα το αντικείμενο το οποίο θέλουμε να ελέγξουμε αν συγκρούεται, το γκρουπ των sprites που θέλουμε να ελέγξουμε για συγκρούσεις, και τη μεταβλητή `doKill` η οποία είναι τύπου Boolean και με την οποία δηλώνουμε αν θέλουμε να αφαιρεθεί το αντικείμενο από το γκρουπ.

```
# Check for collision with enemies
if pygame.sprite.spritecollide(self, enemyGroup, False):
    gameState = "GameOver"

# Check for collision with garbage
if pygame.sprite.spritecollide(self, garbageGroup, False):
    gameState = "GameOver"

# Check for collision with recycle bin
if pygame.sprite.spritecollide(self, recycleBinGroup,
False):
    gameState = "NextLevel"
```

Στη συνέχεια αναλόγως το αποτέλεσμα της σύγκρουσης ορίζουμε την τιμή στη μεταβλητή gameState την οποία και επιστρέφουμε στο τέλος της update().

### Συγκρούσεις με τις κινούμενες πλατφόρμες

Με μία δομή επανάληψης ελέγχουμε για κάθε αντικείμενο τύπου platform που βρίσκεται μέσα στο Group platformGroup αν ο παίκτης κάνει σύγκρουση με κάποιο από αυτά. Για να το πετύχουμε αυτό χρησιμοποιούμε τη συνάρτηση colliderect(). Σαν ορίσματα λαμβάνει τη θέση που θα έχει ο παίκτης καθώς και τις διαστάσεις του rect του player.

```
# Check for collision with moving platforms
for platform in platformGroup:
    # Check for collision in x axis
    if platform.rect.colliderect(self.rect.x + dif_x,
self.rect.y, self.width, self.height):
        dif_x = 0
    # Check for collision in y axis
    if platform.rect.colliderect(self.rect.x, self.rect.y +
dif_y, self.width, self.height):
        # Check if player is below the platform
        if abs((self.rect.top + dif_y) -
platform.rect.bottom) <= collisionThreshold:
            self.jump = 0
            dif_y = platform.rect.bottom - self.rect.top #
Stop because player hit platform
        # Check if player is above platform
        elif abs((self.rect.bottom + dif_y) -
platform.rect.top) <= collisionThreshold:
            self.rect.bottom = platform.rect.top - 1 # Stop
because player standing on the platform
            self.jump = 0
            if key[pygame.K_SPACE] == False:
                self.isJump = False # Disable double jump
```

Στην κάθετη μετατόπιση ο player κινείται μαζί με την πλατφόρμα καθώς αυτό επιτυγχάνεται με το collision. Για την οριζόντια μετατόπιση ελέγχουμε αν ο παίκτης βρίσκεται πάνω σε πλατφόρμα που κινείται οριζόντια (το moveX δεν είναι 0) και ανανεώνουμε την θέση x του player ώστε να ακολουθεί την κίνηση της πλατφόρμας.

```
# Move player together with platform
if platform.moveX != 0:
    self.rect.x += platform.direction
```

### Κίνηση των Enemies

Η κλάση Enemies είναι υπεύθυνη για τη δημιουργία και την κίνηση των εχθρών του παιχνιδιού. Κληρονομεί τη βασική της λειτουργικότητα από την κλάση pygame.sprite.Sprite. Σε αντίθεση με τις άλλες κλάσεις που κληρονομούν τη βασική τους λειτουργικότητα από την κλάση pygame.sprite.Sprite επειδή θέλουμε να έχουμε κίνηση στα αντικείμενα που δημιουργούνται, κάνουμε override τη μέθοδο update().

Με τη χρήση των μεταβλητών direction και counter μετακινούμε το αντικείμενο κατά 40 μονάδες προς τα δεξιά. Στην συνέχεια το direction και το counter λαμβάνουν αρνητικές τιμές και κινούμε το αντικείμενο κατά 80 μονάδες αριστερά. Αυτό επιτυγχάνεται με την της abs() η οποία μας επιστρέφει θετικές τιμές. Έτσι δημιουργείται μια συνεχόμενη κίνηση στα αντικείμενα τύπου Enemies.

```
# Override update method to make enemy moves
def update(self):
    self.rect.x += self.direction
    self.counter += 1
    if abs(self.counter) > 40:
        self.direction *= -1
        self.counter *=-1
```

## Σχεδιασμός των επιπέδων

Ο σχεδιασμός των επιπέδων πραγματοποιείται με την κλάση World. Πρώτα φορτώνουμε τις εικόνες wall.png και ground.png στις μεταβλητές wallImage και groundImage αντίστοιχα. Στη συνέχεια με τη χρήση δύο δομών επαναλήψεων διατρέχουμε όλα τα στοιχεία της λίστας που ορίσαμε στην levelData.

Ελέγχουμε ένα-ένα όλα τα στοιχεία της λίστας καθώς τα διατρέχουμε και αναλόγως την τιμή που έχουν (βλέπε Πίνακα1) προσθέτουμε την αντίστοιχη εικόνα και το rect στην λίστα tileList ή προσθέτουμε το αντικείμενο στο ανάλογο sprite group.

```
# Load tile images
wallImage = pygame.image.load('assets/wall.png')
groundImage = pygame.image.load('assets/ground.png')

rowCount = 0
for row in levelData:
    columnCount = 0
    for tile in row:
        if tile == 1:
            image = pygame.transform.scale(wallImage,
(tileSize, tileSize))
            rect = image.get_rect()
            rect.x = columnCount * tileSize
            rect.y = rowCount * tileSize
            tile = (image, rect)
            self.tileList.append(tile)
```

```

        if tile == 2:
            image = pygame.transform.scale(groundImage,
(tileSize, tileSize))
            rect = image.get_rect()
            rect.x = columnCount * tileSize
            rect.y = rowCount * tileSize
            tile = (image, rect)
            self.tileList.append(tile)
        if tile == 3:
            poopEnemy = Enemies(columnCount * tileSize,
rowCount * tileSize)
            enemyGroup.add(poopEnemy)
        if tile == 4:
            garbage = Garbage(columnCount * tileSize, rowCount
* tileSize)
            garbageGroup.add(garbage)
        if tile == 5:
            recycleBin = Recyclebin(columnCount * tileSize,
rowCount * tileSize)
            recycleBinGroup.add(recycleBin)
        if tile == 6:
            can = Cans(columnCount * tileSize, rowCount *
tileSize + 30)
            cansGroup.add(can)
        if tile == 7:
            platform = Platform(columnCount * tileSize,
rowCount * tileSize, 1, 0) # Horizontal moving platforms
            platformGroup.add(platform)
        if tile == 8:
            platform = Platform(columnCount * tileSize,
rowCount * tileSize, 0, 1) # Vertical moving platforms
            platformGroup.add(platform)
        columnCount += 1
        rowCount +=1

```

Τα tile τα οποία κάνουμε append() στη λίστα tileList έχουν τη μορφή πλειάδας (tuple) tile = (image, rect), όπου στην πρώτη θέση αποθηκεύεται η εικόνα και στη δεύτερη θέση το rect.

Τέλος, με την drawWorld() η οποία λαμβάνει ως όρισμα την οθόνη στην οποία θα εμφανιστούν τα γραφικά, εμφανίζουμε την εικόνα του background. Διατρέχοντας όλα τα στοιχεία της tileList στην συνέχεια, εμφανίζουμε την εικόνα του tile και την θέση που είναι αποθηκευμένα σε κάθε θέση της λίστας.

```

# Draw background image and the tiles of the level
def drawWorld(self, screen):
    screen.blit(bgImage2, (0, 0))
    for tile in self.tileList:
        screen.blit(tile[0], tile[1])

```

## Βάση δεδομένων

Για την αποθήκευση των δεδομένων του παιχνιδιού χρησιμοποιήθηκε η βάση δεδομένων SQLite3.

Η βάση δεδομένων αποτελείται από έναν πίνακα (higscores) ο οποίος περιέχει τα:

id: Το μοναδικό κλειδί του πίνακα

playername: για την αποθήκευση του ονόματος του παίκτη – τύπου text

score: για την αποθήκευση του σκορ του παίκτη – τύπου integer

time: για την αποθήκευση του συνολικού χρόνου που έκανε ο παίκτης μέχρι να ολοκληρώσει όλα τα επίπεδα του παιχνιδιού – τύπου float

## Συναρτήσεις

Για την διαχείριση και την εκτέλεση ερωτημάτων στη βάση δεδομένων χρησιμοποιούμε τρεις συναρτήσεις.

Η συνάρτηση `createConnection()`

Χρησιμοποιείτε για την δημιουργία της σύνδεσης του προγράμματος μας με τη βάση δεδομένων.

```
# DB Create connection
def createConnection(path):
    connection = None
    try:
        connection = sqlite3.connect(path)
        print("Connection to SQLite DB successful")
    except Error as e:
        print(f"The error '{e}' occurred")
    return connection
```

Η συνάρτηση `executeQuery()`

Χρησιμοποιείτε για την εκτέλεση ερωτημάτων προς τη βάση δεδομένων. Στην περίπτωση μας όταν θέλουμε να κάνουμε εγγραφή στη βάση δεδομένων.

```
# DB Execute write SQL query
def executeQuery(connection, query):
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        connection.commit()
        print("Query (write) executed successfully")
    except Error as e:
        print(f"The error '{e}' occurred")
```



Η συνάρτηση `executeReadQuery()`

Χρησιμοποιείτε για την εκτέλεση ερωτημάτων προς τη βάση δεδομένων προκειμένου να διαβάσουμε δεδομένα και να επιστρέψουμε κάποιο αποτέλεσμα.

```
# DB Execute read SQL query
def executeReadQuery(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
        print("Query (read) executed successfully")
    except Error as e:
        print(f"The error '{e}' occurred")
```

## Ερωτήματα SQL

### CreateUserTable

Αν δεν υπάρχει ο πίνακας `highscores` στη βάση δεδομένων τότε τον δημιουργεί.

```
createUserTable = """
CREATE TABLE IF NOT EXISTS highscores (
id INTEGER PRIMARY KEY AUTOINCREMENT,
playername TEXT NOT NULL,
score INTEGER,
time float
);
"""
```

### addPlayerScore

Ενημερώνει τον πίνακα `highscores` με τις τιμές `playerName`, `cansScore` και `elapsedTime`

```
addPlayerScore = f"INSERT INTO highscores(playername, score, time)
VALUES ('{playerName}', '{cansScore}', '{elapsedTime:.2f}')
```

### selectHighScores

Διαβάζει και επιστρέφει τις 3 πρώτες εγγραφές από τον πίνακα `highscores` αφού πρώτα τις ταξινομήσει κατά `score` (από το μεγαλύτερο στο μικρότερο) και στην συνέχεια κατά `time` (από το μικρότερο προς το μεγαλύτερο).

```
selectHighScores = "SELECT playername, score, time FROM highscores
ORDER BY score DESC, time ASC LIMIT 3"
```

## Εικόνες – Γραφικά παιχνιδιού

Για τη δημιουργία των γραφικών του παιχνιδιού χρησιμοποιήθηκαν εικόνες και γραφικά από την online τράπεζα εικόνων και γραφικών freepik. <https://www.freepik.com/>

Στη συνέχεια έγινε επεξεργασία αυτών για τις ανάγκες του παιχνιδιού με την εφαρμογή Affinity Photo 2.

## Βιβλιογραφία

1. Pygame (n.d.). Pygame documentation. Ανακτήθηκε από <https://www.pygame.org/docs/>
2. Python (n.d.) Python documentation. Ανακτήθηκε από <https://docs.python.org/>