Name : Azzouz Nazim/Mahroum Amayess

Rapport de projet Breaker IT

1. *Introduction :*

Game Unleashed, une entreprise technologique spécialisée dans le développement de jeux pour PC, MacOS et Linux, qui existe depuis 1978 ! Selon une analyse du marché, l'univers des jeux rétro est en plein essor et devrait se poursuivre jusqu'en 2023 ! Dans le cadre de leur stratégie, ils souhaitent ressusciter certains "jeux rétro", également connus sous le nom de "jeux auxquels je jouais quand j'étais enfant !" En 1979, l'entreprise a sorti son tout premier jeu vidéo : '' Break/IT '', Donc nous avons choisi de réaliser ce jeux pour notre projet.

2. *Les fonctionnalités de jeux :*

- ✓ *Fonctionnalités de base :*
  - Gestion des collisions entre balle, barre et briques
  - Gestion du rebond sur la barre : changement de direction selon l'endroit où elle rebondit.
- ✓ Fonctionnalités supplémentaires :
  - Un système de score qui correspond au temps mis par l'utilisateur pour faire tous les niveaux
  - Une prise en compte de divers effets avec des briques qui permettent d'obtenir des actions spéciales une fois cassées
  - Bonus :
    - Multiplication de la balle par 3 : Ce bonus permet au joueur d'avoir trois balles en jeu simultanément, ce qui augmente les chances de détruire davantage de briques en un seul coup.
    - Colle pour la raquette : Ce bonus permet de coller la balle à la raquette, la maintenant en place jusqu'à ce que le joueur décide de la lancer à nouveau. Cela donne au joueur plus de contrôle sur la trajectoire de la balle et permet de planifier des tirs plus précis.
    - Balle de feu : Ce bonus rend la balle incontrôlable et lui permet de détruire les briques sans rebondir. La balle traverse directement les briques, ce qui facilite la destruction rapide des obstacles.
    - Ralentissement des balles : Ce malus réduit la vitesse des balles de 33%. Cela rend le jeu un peu plus facile, car les balles se déplacent plus lentement, ce qui donne au joueur plus de temps pour réagir et ajuster la position de la raquette.
    - Augmentation de la vitesse des balles : Ce malus augmente la vitesse des balles de 50%. Cela rend le jeu plus difficile, car les balles se déplacent plus rapidement, nécessitant des réflexes. Rapides et une précision accrue pour les intercepter avec la raquette.

3. Rapports techniques :

Notre jeu casse brique est réalisé en utilisant le langage de programmation python, grâce aux bibliothèques :

- Bibliothèque os : C'est une bibliothèque standard qui fournit des fonctionnalités permettant d'interagir avec le système d'exploitation. Elle offre un large éventail de fonctions pour la gestion des fichiers, des répertoires, des processus, des variables d'environnement et d'autres opérations système.
- Bibliothèque json : est une bibliothèque standard qui permet de travailler avec des données au format JSON (JavaScript Object Notation). JSON est un format de données largement utilisé pour échanger des informations structurées entre différentes applications. La bibliothèque "json" en Python offre des fonctions pour sérialiser (encoder) des objets Python en JSON et désérialiser (décoder) des données JSON en objets Python
- Bibliothèque pygame : C'est une bibliothèque open-source populaire utilisée pour le développement de jeux en 2D avec le langage de programmation Python. Elle fournit un ensemble complet d'outils et de fonctionnalités pour la création de jeux, y compris la gestion des graphiques, des événements, du son et de l'interaction utilisateur.
- Bibliothèque math : La bibliothèque "math" est une bibliothèque standard de Python qui fournit des fonctions mathématiques avancées pour effectuer des calculs numériques.
- Bibliothèque random : est une bibliothèque standard de Python qui permet de générer des valeurs aléatoires.

Notre jeu est composé de plusieurs programme :

- Programme main.py c'est le programme principal, il s'agit de point de départ pour lancer le jeu "Break_IT. Ce code contient les fonctionnalités suivantes :
  - Création de la fenêtre de jeu : Une instance de la classe "Window" est créée avec une résolution de 1080x720 pixels.
  - Chargement du jeu : La fonction "load_complete()"

```python
from layoutlib.window import Window
from GameRunner import Core

def main():
    # Create an instance of the game Window
    game = Window(1080, 720, True, 'Break_IT')

    # Load the game GameRunner into the game Window
    game.load_complete(Core(), 'data', 'Resources.dat', 'Levels.dat')

    # Start the main game loop
    game.gInstance.main_loop()

    # Destroy the game Window and clean up resources
    game.destroy()

if __name__ == "__main__":
    # Entry point of the program
    main()
```

- Programme window.py : contient le code de la classe window qui représente la fenêtre du jeu et contient les fonctionnalités nécessaires pour l'initialiser, le charger et le détruire.
  - Initialisation de la fenêtre avec les paramètres (screenwidth pour définir la largeur de la fenêtre en pixels, screenheight pour définir la hauteur de la fenêtre en pixel , fullscreen s'agit d'un Boolean indiquant si la fenêtre doit être en mode plein écran ,

windowtitle pour définir le titre de la fenêtre du jeu) => l'initiation et la création de la fenêtre se fais grâce au bibliothèque pygame.

- Chargement de la fenêtre avec les paramètres (instance qui représente le jeu en lui-même ,fichier contenant les données de jeu ,fichier contenant des infirmations sur le jeu , fichier contenant les niveaux de jeu)
- Détruire le fenêtre et quitter le jeu en libérant les ressources spécifiques au jeu.

```python
import pygame as pg
import sys

class Window(object):
    """ Window : Contain the whole game,window, resources...etc """
    def __init__(self,screenWidth,screenHeight,fullscreen,windowTitle):
        # full screen & optimization
        flag = 1
        # init pygame
        pg.init()
        if not pg.display.get_init():
            print('unable to init display pygame')
            self.destroy()
        else:
            pg.display.set_caption(windowTitle)
            if fullscreen:
                flag |= pg.FULLSCREEN
            flag |= pg.HWSURFACE
            flag |= pg.DOUBLEBUF
            pg.display.set_mode((screenWidth, screenHeight), flag)
            # temporarily set which modifier keys are pressed to 0.
            pg.key.set_mods(0)
            pg.key.set_repeat(10, 10)

    def load_complete(self,instance,dataFolder,persistenceLayer,fileLevels):
        # display infos
        print('Display driver: ' + pg.display.get_driver())
        print(pg.display.Info())
        # get instance of the game:
        self.gInstance = instance
        # load game resources & get infos from persistence layer.
        self.gInstance.load_game(dataFolder, persistenceLayer)
        self.gInstance.load_levels(dataFolder, fileLevels)

    def destroy(self):
        if self.gInstance:
            self.gInstance.destroy_game()
        pg.quit()
        sys.exit()
```

- Programme utlity.py contient plusieurs fonctions utilitaires utilisées dans le jeu :
  - La fonction dist calcule la distance entre deux points en utilisant la formule de la distance euclidienne (ou norme euclidienne)
  - Les fonctions rad2deg et deg2rad sont utilisées pour convertir les angles entre radians et degrés
  - La fonction rotate effectue une rotation d'un point (x, y) autour de l'origine (0, 0).

- La fonction rotate2p effectue une rotation d'un point (x1, y1) autour d'un point donné (x0, y0) .
- La fonction getTime retourne le temps écoulé en millisecondes ou en secondes depuis le début du programme.
- La fonction rotateDeg effectue une rotation d'une image par l'angle spécifié tout en maintenant son centre et sa taille d'origine.
- Les fonctions xIntersection et yIntersection vérifient si deux objets (corps) s'intersectent le long de l'axe x et de l'axe y respectivement. Ces fonctions comparent les coordonnées des objets pour déterminer s'ils se chevauchent ou non.

```python
import math
import pygame as pg

#calculate the distance between two points using the formula
def dist(x1, y1, x2, y2):
    return math.hypot(x2 - x1, y2 - y1)


# give the result of conversion from radians to degrees
def rad2deg(rad):
    return math.degrees(rad)

# give the result of conversion from degress to radians
def deg2rad(deg):
    return math.radians(deg)

# Rotate a point (x, y) around the origin (0, 0) by the specified angle
def rotate(x, y, angle):
    cos_angle = math.cos(angle)
    sin_angle = math.sin(angle)
    new_x = x * cos_angle - y * sin_angle
    new_y = x * sin_angle + y * cos_angle
    return new_x, new_y
# Rotate a point (x1, y1) around a given point (x0, y0) by the specified angle
def rotate2p(x1, y1, x0, y0, angle):
    dx = x1 - x0
    dy = y1 - y0
    cos_angle = math.cos(angle)
    sin_angle = math.sin(angle)
    new_dx = dx * cos_angle - dy * sin_angle
    new_dy = dx * sin_angle + dy * cos_angle
    new_x = new_dx + x0
    new_y = new_dy + y0
    return new_x, new_y

# Get the current time in milliseconds or seconds
def getTime(type='ms'):
    ticks = pg.time.get_ticks()
    if type == 's':
        return ticks / 1000.0
    else:
        return ticks
```

```python
# Rotate an image by the specified angle while keeping its center and size
def rotateDeg(image, angle):
    angle %= 360.0
    orig_rect = image.get_rect()
    rot_image = pg.transform.rotate(image, angle)
    orig_rect.center = rot_image.get_rect().center
    rot_image = rot_image.subsurface(orig_rect).copy()
    return rot_image

# Check if two bodies intersect along the x-axis
def xIntersection(body1, body2):
    return (body1.x + body1.w) < (body2.x + body2.w) * 0.5 or \
           (body1.x + body1.w) >= (body2.x + body2.w) * 0.5
# Check if two bodies intersect along the y-axis
def yIntersection(body1, body2):
    return (body1.y + body1.h) < (body2.y + body2.h) or \
           (body1.y + body1.h) >= (body2.y + body2.h)
```

- Programme ressources.py définit la classe ressources qui est utilisée pour charger et gérer diverses ressources telles que des sons, des images, des polices de caractères et des fichiers de données.
  - Initialisation : initialise l'objet Resources avec le chemin principal (mainPath) spécifié. Elle initialise également des compteurs de référence pour les différents types de ressources
  - Load_sound => Chargement du sound avec son volume
  - Load-snd-list => charge une liste de fichiers son en appelant la méthode load_sound
  - Load-image => charger une image
  - Get-music-list => liste de chemins de fichiers musicaux
  - Load -img-list => charger liste de fichier images en appelant la methode load-image
  - Load-font => charge un fichier de police
  - Load-font-list => charge une liste de fichiers de police en appelant la méthode load_font
  - Print-font => affiche du texte à l'écran en utilisant une police spécifiée
  - Les méthodes load_global et destroy_global sont utilisées pour charger et détruire les ressources globalement

```python
import os
import json
import pygame as pg


SOUND_TITLE = 0
SOUND_VOLUME = 1
FONT_NAME = 0
FONT_SIZE = 1


class Resources(object):

    def __init__(self, mainPath=''):
        # Initialize the Resources object with the mainPath
        self.mainPath = mainPath
        self.sound_ref = 0
        self.music_ref = 0
        self.font_ref = 0
        self.image_ref = 0
        self.persiLayer = ''
        self.screen = pg.display.get_surface()
```

```python
def load_sound(self, name, volume=1.0):
    # Load a sound file and set its volume
    name = os.path.join(self.mainPath, name)
    sound = pg.mixer.Sound(name)
    sound.set_volume(volume)
    assert sound
    self.sound_ref += 1
    return sound

def load_snd_list(self, soundList):
    # Load a list of sound files
    sounds = []
    for snd in soundList:
        sounds.append(self.load_sound(snd[SOUND_TITLE], snd[SOUND_VOLUME]))
    return sounds

def load_image(self, name, flag=None):
    # Load an image file
    if flag == 'alpha':
        image = pg.image.load(os.path.join(self.mainPath, name)).convert_alpha()
    else:
        image = pg.image.load(os.path.join(self.mainPath, name)).convert()
    assert image
    self.image_ref += 1
    return image
```

```python
def get_music_list(self, musicList):
    # Return a list of music file paths
    music_path = []
    for title in musicList:
        music_path.append(os.path.join(self.mainPath, title))
        self.music_ref += 1
    return music_path

def load_img_list(self, imgList, flag=None):
    # Load a list of image files
    image = []
    for name in imgList:
        image.append(self.load_image(name, flag))
    return image

def load_font(self, name, size):
    # Load a font file
    name = os.path.join(self.mainPath, name)
    font = pg.font.Font(name, size)
    assert font
    self.font_ref += 1
    return font
```

```python
    def load_fnt_list(self, fntList):
        # Load a list of font files
        fonts = []
        for fnt in fntList:
            fonts.append(self.load_font(fnt[FONT_NAME], fnt[FONT_SIZE]))
        return fonts

    def print_font(self, font, message, vect, color=(255, 255, 255)):
        # Print text on the screen using a font
        self.screen.blit(font.render(message, True, color), vect)

    def load_global(self, imgList, sndList, fntList, mscList):
        # Load resources globally
        img = self.load_img_list(imgList)
        snd = self.load_snd_list(sndList)
        fnt = self.load_fnt_list(fntList)
        msc = self.get_music_list(mscList)
        return img, snd, fnt, msc

    def destroy_global(self):
        # Destroy global resources
        pass
```

- Programme config.py pour définir différentes constantes utilisées dans le jeu

```python
# Colors
DARK = (0, 0, 0)                # Dark color
WHITE = (255, 255, 255)         # White color
MENU_COL = (230, 230, 230)      # Menu color
BALL_COL = (220, 20, 60)         # Ball color
BACK_COL = (255, 208, 160)      # Background color

# Circle properties
CIRCLE_RADIUS = 7               # Circle radius
CIRCLE_VEL = 72.0               # Circle velocity
BALL_SIZE = 1                   # Ball size

# Angle limits
MIN_ANGLE = 0.885               # Minimum angle
MAX_ANGLE = 1.333               # Maximum angle

# Paddle properties
PADDLE_VEL = 19.0               # Paddle velocity
PAD_W = 95                      # Paddle width
PAD_H = 15                      # Paddle height
PAD_SIZE = 1                    # Paddle size
```

```
# Brick properties
BRICK_W = 45                    # Brick width
BRICK_H = 25                    # Brick height
BRICK_ROW = 5                   # Number of brick rows
BRICK_COL = 20                  # Number of brick columns

# Brick types and colors
BONUS_BRICK = (174, 129, 213)   # Bonus brick color (1 hit to break)
SOLID_BRICK = (47, 79, 79)    # Solid brick color (2 hits to break)
ROCK_BRICK = (178, 34, 34)       # Rock brick color (3 hits to break)
NORMAL_BRICK = (211, 211, 211)     # Normal brick color (1 hit to break)

# Object types
TYPE_CIRCLE = 0                 # Circle type
TYPE_PADDLE = 1                 # Paddle type
TYPE_BRICK = 2                  # Brick type

# Default frames per second
DEFAULT_FPS = 350
```

- Programme Elementarbody.py pour définir la classe Body qui représente un objet du jeu, tel qu'un cercle, une palette ou une brique. en définissant leurs propriétés et en fournissant des méthodes pour les dessiner et les mettre à jour en fonction du temps et des collisions.

```python
from math import cos, sin
from random import uniform
import pygame as pg
from config import *

class Body:
    def __init__(self, type, x, y, color=WHITE):
        self.type = type
        self.Xpos, self.Ypos = x, y
        self.color = color
        self.thickness = 0
        self.screen = pg.display.get_surface()
        self.boundary = self.screen.get_rect()

        if self.type == TYPE_CIRCLE:
            self.init_circle()
        elif self.type == TYPE_PADDLE:
            self.init_paddle()
        elif self.type == TYPE_BRICK:
            self.init_brick()

    def init_circle(self):
        # Initialize properties for a circle body
        self.alive = False
        self.radius = CIRCLE_RADIUS
        self.Shape = pg.Rect(self.Xpos, self.Ypos, CIRCLE_RADIUS, CIRCLE_RADIUS)
        self.radius = CIRCLE_RADIUS
        self.Shape = pg.Rect(self.Xpos, self.Ypos, CIRCLE_RADIUS, CIRCLE_RADIUS)
        self.angle = MAX_ANGLE
        self.xvel = self.yvel = -CIRCLE_VEL

    def init_paddle(self):
        # Initialize properties for a paddle body
        self.Shape = pg.Rect(self.Xpos, self.Ypos, PAD_W, PAD_H)
        self.xvel = PADDLE_VEL
        self.side = 0

    def init_brick(self):
        # Initialize properties for a brick body
        self.Shape = pg.Rect(self.Xpos, self.Ypos, BRICK_W, BRICK_H)
        self.alive = True

        if self.color == NORMAL_BRICK:
            self.duration = 1
        elif self.color == SOLID_BRICK:
            self.duration = 2
        elif self.color == ROCK_BRICK:
            self.duration = 3
        elif self.color == BONUS_BRICK:
            self.duration = 1
```

```python
def draw(self):
    # Draw the body object on the screen
    if self.type == TYPE_CIRCLE:
        pg.draw.circle(self.screen, self.color, (self.Shape.x, self.Shape.y), self.radius, self.thickness)
    else:
        pg.draw.rect(self.screen, self.color, self.Shape, self.thickness)

def update(self, dt):
    # Update the position of the body object
    if self.type == TYPE_CIRCLE:
        self.update_circle(dt)
    elif self.type == TYPE_PADDLE:
        self.update_paddle(dt)

def update_circle(self, dt):
    # Update the position of a circle body
    self.Xpos += self.xvel * cos(self.angle) * dt
    self.Ypos += self.yvel * sin(self.angle) * dt

    if self.collide_x():
        self.xvel *= -1
    if self.collide_y() == 1:
        self.angle = uniform(MIN_ANGLE, MAX_ANGLE + 0.02)
        self.yvel *= -1
    self.Shape.x = int(self.Xpos)
    self.Shape.y = int(self.Ypos)
```

```python
def update_paddle(self, dt):
    # Update the position of a paddle body
    self.Xpos += self.side * self.xvel * dt
    self.collide_x()
    self.Shape.x = int(self.Xpos)

def collide_x(self):
    # Check for collisions with the horizontal boundaries
    if self.Xpos <= 0:
        self.Xpos = 1.0
        return True
    elif self.Xpos + self.Shape.w >= self.boundary.w:
        self.Xpos = self.boundary.w - (self.Shape.w + 1)
        return True

def collide_y(self):
    # Check for collisions with the vertical boundaries
    if self.Ypos <= 0:
        self.Ypos = 1.0
        return 1
    elif self.Ypos + self.Shape.h >= self.boundary.h:
        return 2
```

- Programme GameRunner.py definit la classe Core qui est responsable de la gestion principale du jeu.
  - Initialisation : Le programme initialise divers attributs et charge les ressources du jeu, telles que les images, les sons et les polices.
  - Construction du niveau : Le jeu comporte plusieurs niveaux. La méthode build_level est responsable de la création des éléments du niveau, tels que la raquette, la balle et les briques, en utilisant les informations spécifiques au niveau actuel.
  - Gestion des événements : Le programme écoute les événements tels que les pressions de touches du clavier et les fermetures de fenêtre. Les méthodes key_up et key_down sont utilisées pour gérer les événements de pression et de relâchement des touches.

- Boucle principale : Le jeu est exécuté dans une boucle principale. À chaque itération de la boucle, les événements sont écoutés, les objets du jeu sont mis à jour en fonction du temps écoulé, et l'écran est rafraîchi pour afficher les éléments du jeu.
- Gestion des collisions : Le programme détecte les collisions entre la balle et la raquette, ainsi qu'entre la balle et les briques. Il gère les conséquences de ces collisions, telles que le rebond de la balle, la destruction des briques et le score du joueur.
- Passage au niveau suivant : Lorsque toutes les briques du niveau actuel ont été détruites, le jeu passe automatiquement au niveau suivant. Une musique de fond aléatoire est jouée.
- Affichage à l'écran : Le programme affiche les différents éléments du jeu, tels que la raquette, la balle, les briques, le niveau, le score et le nombre de vies. En cas de fin de partie, un écran de fin s'affiche avec la possibilité de recommencer en appuyant sur la touche "Enter".
- Nettoyage et destruction : À la fin du jeu, le programme libère les ressources utilisées pour éviter les fuites de mémoire. En résumé, ce code met en œuvre les fonctionnalités principales d'un jeu de casse-briques en utilisant la bibliothèque Pygame. Il gère la logique du jeu, la détection des collisions, l'affichage à l'écran et l'interaction avec le joueur.

```python
import random
import pygame as pg
from layoutlib.resources import Resources
from layoutlib.utility import xIntersection
from ElementaryBody import Body
from config import *


class Core(object):
    def __init__(self):
        self.left = self.right = False
        self.screen = pg.display.get_surface()
        self.boundary = self.screen.get_rect()
        self.gameDone = False
        self.restart = False
        self.clock = pg.time.Clock()
        self.dt = 0.05
        self.fps = 0.0
        self.res = None
        self.snd = None

    def build_level(self):

        # There are 7 levels, reset to the first level if level exceeds 7
        self.level %= 7
```

```
    # Get configuration of current level
    level_config = self.levels[self.level]['cnf']
    this_level_map = self.levels[self.level]['map']
    start_x, start_y = level_config[0:2]  # Start positions of the first brick
    this_thickness = level_config[2]  # Thickness of drawable stuff
    this_pad_w = level_config[3]  # Paddle width
    this_pad_vel = level_config[4]  # Paddle velocity
    this_ball_vel = level_config[5]  # Ball velocity
    self.level_title = level_config[6]  # Level title

    # Create the pad
    for i in range(PAD_SIZE):
        x_pos = self.boundary.w / 2 - (PAD_W / 2)
        y_pos = self.boundary.h - (PAD_H + 5)
        self.pad.append(Body(TYPE_PADDLE, x_pos, y_pos, BALL_COL))
        self.pad[i].Shape.w = this_pad_w
        self.pad[i].thickness = this_thickness
        self.pad[i].xvel = this_pad_vel

    # Create the ball and start it from the center of the pad
    for i in range(BALL_SIZE):
        x_pos = self.pad[0].Shape.x + (PAD_W / 2)
        y_pos = self.pad[0].Shape.y - (CIRCLE_RADIUS + 1)
        self.ball.append(Body(TYPE_CIRCLE, x_pos, y_pos, BALL_COL))
        self.ball[i].xvel = self.ball[i].yvel = -this_ball_vel
        self.ball[i].thickness = this_thickness
```

```
    # Build the level by creating bricks
    x_pos = start_x
    y_pos = start_y
    i = 0
    for row in this_level_map:
        for brick in row:
            if brick == 'n':  # Normal brick
                self.color = NORMAL_BRICK
            elif brick == 's':  # Solid brick
                self.color = SOLID_BRICK
            elif brick == 'r':  # Rock brick
                self.color = ROCK_BRICK
            elif brick == 'b':  # Bonus brick
                self.color = BONUS_BRICK
            elif brick == ' ':  # Empty space
                x_pos += BRICK_W + 3
                continue
            self.bricks.append(Body(TYPE_BRICK, x_pos, y_pos, self.color))
            self.bricks[i].thickness = this_thickness
            i += 1
            x_pos += BRICK_W + 3
        y_pos += BRICK_H + 1
        x_pos = start_x
```

```
    def key_up(self, key):

        if key == pg.K_LEFT:
            self.left = False
        elif key == pg.K_RIGHT:
            self.right = False
        elif key == pg.K_SPACE:
            self.restart = False
        elif key == pg.K_ESCAPE:
            self.gameDone = True

    def key_down(self, key):

        if key == pg.K_LEFT:
            self.left = True
        elif key == pg.K_RIGHT:
            self.right = True
        elif key == pg.K_SPACE:
            if not self.gameOver:
                self.restart = True
        elif key == pg.K_RETURN:
            if self.gameOver:
                self.new_game()
```

```
    def event_listener(self):

        ev = pg.event.poll()
        if ev.type == pg.QUIT:
            self.gameDone = True
        elif ev.type == pg.KEYDOWN:
            self.key_down(ev.key)
        elif ev.type == pg.KEYUP:
            self.key_up(ev.key)

    def new_game(self):

        self.lives = 3
        self.score = 0
        self.gameOver = False
        self.level = 0
        self.gameDone = False
        self.restart = False
        self.next_level()  # Go to level 1

    def next_level(self):

        self.pad = []
        self.ball = []
        self.bricks = []

        # Create bricks for the next level
        self.build_level()

        # Prepare next level
        self.level += 1

        # Play random background music
        this_music = random.choice(self.music_titles)
        pg.mixer.music.load(this_music)
        pg.mixer.music.play()

    def draw(self):

        if not self.gameOver:
            self.screen.fill(BACK_COL)

            # Draw the pad list
            for pd in self.pad:
                pd.draw()

            # Draw the ball list
            for bl in self.ball:
                bl.draw()
```

```
        # Draw the bricks list
        for br in self.bricks:
            br.draw()
    else:  # Game over
        self.screen.fill(WHITE)
        self.res.print_font(self.fnt[2], 'Game is over now want a second try ? ', (260, 250), DARK)
        self.res.print_font(self.fnt[1], 'press Enter to play', (440, 500), DARK)

    # Draw all text elements
    self.res.print_font(self.fnt[0], str("{0:.2f} FPS".format(self.fps)), (1005, 2), DARK)
    self.res.print_font(self.fnt[0], self.level_title, (5, 2), DARK)
    self.res.print_font(self.fnt[0], str("{} X Points".format(self.score)), (560, 2), DARK)
    self.res.print_font(self.fnt[0], str("{} X Balls".format(self.lives)), (465, 2), DARK)

    pg.display.update()

def update(self, dt):

    if not self.gameOver:
        # Update the pad
        for pd in self.pad:
            pd.side = int(self.right) - int(self.left)
            pd.update(dt)

        # Update ball/bricks
```

```
        # Update ball/bricks
        for bl in self.ball:
            if bl.alive:
                if bl.Ypos > self.boundary.h:
                    self.death(bl)
                    break

                for pd in self.pad:
                    # Check for ball/pad collision
                    if pd.Shape.colliderect(bl.Shape):
                        bl.yvel *= -1
                        bl.Ypos = pd.Shape.y - (bl.Shape.h + 1)
                        self.snd[0].play()

                for br in self.bricks:
                    # Check for ball/bricks collision and update state
                    if br.Shape.colliderect(bl.Shape):
                        bl.yvel *= -1
                        if xIntersection(bl.Shape, br.Shape):
                            bl.xvel *= -1
                        bl.angle = random.uniform(MIN_ANGLE, MAX_ANGLE)
                        self.snd[random.randint(1, 6)].play()
                        # Check brick type
                        self.kill_bricks_type(br)
                        self.score += 1
```

```
            else:
                # Ball has fallen down, reset it if game is not over
                bl.Xpos = pd.Xpos + pd.Shape.w / 2
                bl.Ypos = pd.Ypos - (bl.Shape.w + 1)
                bl.alive = self.restart

            bl.update(dt)

        # Check for win and proceed to next level
        if not self.bricks:
            self.snd[8].play()
            self.next_level()

    def kill_bricks_type(self, brick):

        if brick.duration > 0:
            if brick.color is NORMAL_BRICK:
                brick.alive = False
            elif brick.color is SOLID_BRICK:
                brick.color = NORMAL_BRICK
            elif brick.color is ROCK_BRICK:
                brick.color = SOLID_BRICK
            elif brick.color is BONUS_BRICK:
                brick.alive = False
                self.snd[7].play()
                self.lives += 1
            brick.duration -= 1

        self.bricks = [br for br in self.bricks if br.alive]

    def death(self, body):

        if not self.lives:
            self.gameOver = True
        else:
            self.lives -= 1
            body.alive = False

    def load_game(self, dataFolder, persistenceLayer):

        self.res = Resources(dataFolder)
        data = self.res.get_res_info(persistenceLayer)
        self.img = self.res.load_img_list(data['imgList'])
        self.snd = self.res.load_snd_list(data['sndList'])
        self.fnt = self.res.load_fnt_list(data['fntList'])
        self.music_titles = self.res.get_music_list(data['mscList'])
        print(self.res.__dict__)

    def load_levels(self, dataFolder, fileLevels):

        res = Resources(dataFolder)
        self.levels = res.get_res_info(fileLevels)
```

```
def load_levels(self, dataFolder, fileLevels):

    res = Resources(dataFolder)
    self.levels = res.get_res_info(fileLevels)
    del res

def destroy_game(self):

    if self.img:
        del self.img
    if self.snd:
        del self.snd
    if self.fnt:
        del self.fnt
    if self.res:
        del self.res

def main_loop(self):

    self.new_game()

    while not self.gameDone:
        self.event_listener()
        self.update(self.dt)
        self.draw()
```

- Programme database.py définit la classe connect_to_database qui établit une connexion à la base de données MySQL en utilisant les informations d'identification spécifiées. Elle renvoie un objet de connexion qui peut être utilisé pour interagir avec la base de données.

-
- Fonction send_score_to_website(pseudo, score) Cette fonction enregistre un score associé à un pseudo dans la base de données. Elle utilise la fonction connect_to_database() pour établir une connexion, puis exécute une requête INSERT pour insérer les données dans la table "result". Ensuite, la fonction effectue une validation de la transaction en appelant connection.commit(), et ferme le curseur et la connexion à la base de données.
- Fonction get_top_scores(limit=10) Cette fonction récupère les meilleurs scores enregistrés dans la base de données. Elle utilise la fonction connect_to_database() pour établir une connexion, puis exécute une requête SELECT pour obtenir les pseudo, scores et dates de création des résultats. Les résultats sont triés par score décroissant et limités au nombre spécifié par le paramètre limit. Ensuite, la fonction récupère les données renvoyées par la requête et les renvoie sous forme de liste de tuples. Enfin, elle ferme le curseur et la connexion à la base de données.

```python
import mysql.connector
from datetime import datetime


def connect_to_database():
    connection = mysql.connector.connect(
        host='127.0.0.1',
        user='root',
        password='',
        database='breaker'
    )
    return connection

def send_score_to_website(pseudo, score):

    connection = connect_to_database()
    cursor = connection.cursor()
    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    query = "INSERT INTO result (pseudo, score, created) VALUES (%s, %s, %s)"
    data = (pseudo, score, current_time)
    cursor.execute(query, data)
    connection.commit()
    cursor.close()
    connection.close()
```

```python
def get_top_scores(limit=10):

    connection = connect_to_database()
    cursor = connection.cursor()
    query = "SELECT pseudo, score, created FROM result ORDER BY score DESC LIMIT %s"
    data = (limit,)
    cursor.execute(query, data)
    results = cursor.fetchall()
    cursor.close()
    connection.close()
    return results
```

- *Programme player_data qui définit la Fonction save_player_name(name) Cette fonction enregistre le nom du joueur dans un fichier texte. Elle crée un fichier player_name.txt et y écrit le nom fourni en argument. Si le fichier existe déjà, son contenu sera écrasé.*
    - *Fonction load_player_name() :Cette fonction charge le nom du joueur à partir d'un fichier texte. Elle vérifie d'abord si le fichier player_name.txt existe. Si oui, elle ouvre le fichier en mode lecture et renvoie le contenu du fichier après avoir supprimé les espaces vides en début et en fin. Si le fichier n'existe pas, la fonction renvoie None.*

```python
import os

def save_player_name(name):
    filename = 'player_name.txt'
    with open(filename, 'w') as file:
        file.write(name)

def load_player_name():
    filename = 'player_name.txt'
    if os.path.exists(filename):
        with open(filename, 'r') as file:
            return file.read().strip()
    else:
        return None
```