Juego Plataformero 2d estilo Mario Bros en Python

Luis de Jesus Alferez Hernandez
M16130783

DR. Emmanuel Gomez Ramirez

***Resumen*—Se busca crear un juego usando python como lenguaje principal y arduino para poder controlarlo**

I. Introducción

Se busca crear un videojuego en python usando todo lo aprendido en clase mas extras, a la vez, se estara ejecutando en paralelo con un programa para poder controlarlo externamente con arduino y un joystick shield v1.a

II. Python en Acción

A continuacion se muestra el codigo de cada archivo .py que se realizo

II-A. settings.py

Primero, tenemos que definir las configuraciones que llevara nuestro programa, como el tamaño de la pantalla, como se muestra en el siguiente codigo:

```
vertical_tile_number = 11
tile_size = 64

screen_height = vertical_tile_number *
    tile_size
screen_width = 1200
```

Código 1. settings.py

II-B. support.py

En esta parte del codigo implementaremos los soportes del juego, que seran de como tiene que buscar y poner cada tile en el programa:

```
from csv import reader
from settings import tile_size
from os import walk
import pygame

def import_folder(path):
  surface_list = []

  for _,__,image_files in walk(path):
    for image in image_files:
      full_path = path + '/' + image
      image_surf = pygame.image.load(full_path
          ).convert_alpha()
      surface_list.append(image_surf)

  return surface_list

def import_csv_layout(path):
  terrain_map = []
  with open(path) as map:
    level = reader(map,delimiter = ',')
    for row in level:
      terrain_map.append(list(row))
    return terrain_map

def import_cut_graphics(path):
  surface = pygame.image.load(path).
      convert_alpha()
  tile_num_x = int(surface.get_size()[0] /
      tile_size)
  tile_num_y = int(surface.get_size()[1] /
      tile_size)

  cut_tiles = []
  for row in range(tile_num_y):
    for col in range(tile_num_x):
      x = col * tile_size
      y = row * tile_size
      new_surf = pygame.Surface((tile_size,
          tile_size),flags = pygame.SRCALPHA)
      new_surf.blit(surface,(0,0),pygame.Rect(
          x,y,tile_size,tile_size))
      cut_tiles.append(new_surf)

  return cut_tiles
```

Código 2. settings.py

II-C. game_data.py

El siguiente codigo muestra los niveles y y los archivos csv que tendra cada nivel, siendo importados directamente desde el directorio para ser leidos posteriormente

```
level_0 = {
    'terrain': '../gamefin/levels/0/
        level_0_terrain.csv',
    'coins':'../gamefin/levels/0/level_0_coins
        .csv',
    'fg palms':'../gamefin/levels/0/
        level_0_fg_palms.csv',
    'bg palms':'../gamefin/levels/0/
        level_0_bg_palms.csv',
    'crates': '../gamefin/levels/0/
        level_0_crates.csv',
    'enemies':'../gamefin/levels/0/
        level_0_enemies.csv',
    'constraints':'../gamefin/levels/0/
        level_0_constraints.csv',
    'player': '../gamefin/levels/0/
        level_0_player.csv',
    'grass': '../gamefin/levels/0/
        level_0_grass.csv',
    'node_pos': (110,400),
    'unlock': 1,
    'node_graphics': '../gamefin/graphics/
        overworld/0'}
```

```
level_1 = {
    'terrain': '../gamefin/levels/1/level_1_terrain.csv',
    'coins':'../gamefin/levels/1/level_1_coins.csv',
    'fg palms':'../gamefin/levels/1/level_1_fg_palms.csv',
    'bg palms':'../gamefin/levels/1/level_1_bg_palms.csv',
    'crates': '../gamefin/levels/1/level_1_crates.csv',
    'enemies':'../gamefin/levels/1/level_1_enemies.csv',
    'constraints':'../gamefin/levels/1/level_1_constraints.csv',
    'player': '../gamefin/levels/1/level_1_player.csv',
    'grass': '../gamefin/levels/1/level_1_grass.csv',
    'node_pos': (300,220),
    'node_graphics': '../gamefin/graphics/overworld/1',
    'unlock': 2}
level_2 = {
    'terrain': '../gamefin/levels/2/level_2_terrain.csv',
    'coins':'../gamefin/levels/2/level_2_coins.csv',
    'fg palms':'../gamefin/levels/2/level_2_fg_palms.csv',
    'bg palms':'../gamefin/levels/2/level_2_bg_palms.csv',
    'crates': '../gamefin/levels/2/level_2_crates.csv',
    'enemies':'../gamefin/levels/2/level_2_enemies.csv',
    'constraints':'../gamefin/levels/2/level_2_constraints.csv',
    'player': '../gamefin/levels/2/level_2_player.csv',
    'grass': '../gamefin/levels/2/level_2_grass.csv',
    'node_pos': (480,610),
    'node_graphics': '../gamefin/graphics/overworld/2',
    'unlock': 3}
level_3 = {
    'terrain': '../gamefin/levels/2/level_2_terrain.csv',
    'coins':'../gamefin/levels/2/level_2_coins.csv',
    'fg palms':'../gamefin/levels/2/level_2_fg_palms.csv',
    'bg palms':'../gamefin/levels/2/level_2_bg_palms.csv',
    'crates': '../gamefin/levels/2/level_2_crates.csv',
    'enemies':'../gamefin/levels/2/level_2_enemies.csv',
    'constraints':'../gamefin/levels/2/level_2_constraints.csv',
    'player': '../gamefin/levels/2/level_2_player.csv',
    'grass': '../gamefin/levels/2/level_2_grass.csv',
    'node_pos': (610,350),
    'node_graphics': '../gamefin/graphics/overworld/3',
    'unlock': 4}
level_4 = {
    'terrain': '../gamefin/levels/2/level_2_terrain.csv',
    'coins':'../gamefin/levels/2/level_2_coins.csv',
    'fg palms':'../gamefin/levels/2/level_2_fg_palms.csv',
    'bg palms':'../gamefin/levels/2/level_2_bg_palms.csv',
    'crates': '../gamefin/levels/2/level_2_crates.csv',
    'enemies':'../gamefin/levels/2/level_2_enemies.csv',
    'constraints':'../gamefin/levels/2/level_2_constraints.csv',
    'player': '../gamefin/levels/2/level_2_player.csv',
    'grass': '../gamefin/levels/2/level_2_grass.csv',
    'node_pos': (880,210),
    'node_graphics': '../gamefin/graphics/overworld/4',
    'unlock': 5}
level_5 = {
    'terrain': '../gamefin/levels/2/level_2_terrain.csv',
    'coins':'../gamefin/levels/2/level_2_coins.csv',
    'fg palms':'../gamefin/levels/2/level_2_fg_palms.csv',
    'bg palms':'../gamefin/levels/2/level_2_bg_palms.csv',
    'crates': '../gamefin/levels/2/level_2_crates.csv',
    'enemies':'../gamefin/levels/2/level_2_enemies.csv',
    'constraints':'../gamefin/levels/2/level_2_constraints.csv',
    'player': '../gamefin/levels/2/level_2_player.csv',
    'grass': '../gamefin/levels/2/level_2_grass.csv',
    'node_pos': (1050,400),
    'node_graphics':'../gamefin/graphics/overworld/5',
    'unlock': 5}

levels = {
  0: level_0,
  1: level_1,
  2: level_2,
  3: level_3,
  4: level_4,
  5: level_5}
```

Código 3. game_data.py

II-D. tiles.py

El siguiente codigo muestra los niveles y y los archivos csv que tendra cada nivel, siendo importados directamente desde el directorio para ser leidos posteriormente

```
import pygame
from support import import_folder
```

```

class Tile(pygame.sprite.Sprite):
  def __init__(self,size,x,y):
    super().__init__()
    self.image = pygame.Surface((size,size))
    self.rect = self.image.get_rect(topleft =
        (x,y))

  def update(self,shift):
    self.rect.x += shift

class StaticTile(Tile):
  def __init__(self,size,x,y,surface):
    super().__init__(size,x,y)
    self.image = surface

class Crate(StaticTile):
  def __init__(self,size,x,y):
    super().__init__(size,x,y,pygame.image.
        load('../gamefin/graphics/terrain/
        crate.png').convert_alpha())
    offset_y = y + size
    self.rect = self.image.get_rect(bottomleft
         = (x,offset_y))

class AnimatedTile(Tile):
  def __init__(self,size,x,y,path):
    super().__init__(size,x,y)
    self.frames = import_folder(path)
    self.frame_index = 0
    self.image = self.frames[self.frame_index]

  def animate(self):
    self.frame_index += 0.15
    if self.frame_index >= len(self.frames):
      self.frame_index = 0
    self.image = self.frames[int(self.
        frame_index)]

  def update(self,shift):
    self.animate()
    self.rect.x += shift

class Coin(AnimatedTile):
  def __init__(self,size,x,y,path,value):
    super().__init__(size,x,y,path)
    center_x = x + int(size / 2)
    center_y = y + int(size / 2)
    self.rect = self.image.get_rect(center = (
        center_x,center_y))
    self.value = value

class Palm(AnimatedTile):
  def __init__(self,size,x,y,path,offset):
    super().__init__(size,x,y,path)
    offset_y = y - offset
    self.rect.topleft = (x,offset_y)
```

Código 4. tiles.py

II-E. enemy.py

Con enemy.py usaremos los sprites animados de tiles, y las configuraciones para poder crear a los enemigos que tendran que caminar hasta un trigger establecido en game_data

```

import pygame
from tiles import AnimatedTile
from random import randint

class Enemy(AnimatedTile):
  def __init__(self,size,x,y):
    super().__init__(size,x,y,'../gamefin/
        graphics/enemy/run')
    self.rect.y += size - self.image.get_size
        ()[1]
    self.speed = randint(3,5)

  def move(self):
    self.rect.x += self.speed

  def reverse_image(self):
    if self.speed > 0:
      self.image = pygame.transform.flip(self.
          image,True,False)

  def reverse(self):
    self.speed *= -1

  def update(self,shift):
    self.rect.x += shift
    self.animate()
    self.move()
    self.reverse_image()
```

Código 5. enemy.py

II-F. player.py

Para poder crear al jugador/player se hara un codigo un poco mas largo, pero es debido a que se haran todas las mecanicas del juego junto al jugador, incluyendo sus texturas y colisiones

```
import pygame
from support import import_folder
from math import sin

class Player(pygame.sprite.Sprite):
  def __init__(self,pos,surface,
      create_jump_particles,change_health):
    super().__init__()
    self.import_character_assets()
    self.frame_index = 0
    self.animation_speed = 0.15
    self.image = self.animations['idle'][self.
        frame_index]
    self.rect = self.image.get_rect(topleft =
        pos)

    # dust particles
    self.import_dust_run_particles()
    self.dust_frame_index = 0
    self.dust_animation_speed = 0.15
    self.display_surface = surface
    self.create_jump_particles =
        create_jump_particles

    # player movement
    self.direction = pygame.math.Vector2(0,0)
    self.speed = 8
    self.gravity = 0.8
    self.jump_speed = -16
    self.collision_rect = pygame.Rect(self.
        rect.topleft,(50,self.rect.height))
```

```
    # player status
    self.status = 'idle'
    self.facing_right = True
    self.on_ground = False
    self.on_ceiling = False
    self.on_left = False
    self.on_right = False

    # health management
    self.change_health = change_health
    self.invincible = False
    self.invincibility_duration = 500
    self.hurt_time = 0

    # audio
    self.jump_sound = pygame.mixer.Sound('../gamefin/audio/effects/jump.wav')
    self.jump_sound.set_volume(0.5)
    self.hit_sound = pygame.mixer.Sound('../gamefin/audio/effects/hit.wav')

  def import_character_assets(self):
    character_path = '../gamefin/graphics/character/'
    self.animations = {'idle':[],'run':[],'jump':[],'fall':[]}

    for animation in self.animations.keys():
      full_path = character_path + animation
      self.animations[animation] = import_folder(full_path)

  def import_dust_run_particles(self):
    self.dust_run_particles = import_folder('../gamefin/graphics/character/dust_particles/run')

  def animate(self):
    animation = self.animations[self.status]

    # loop over frame index
    self.frame_index += self.animation_speed
    if self.frame_index >= len(animation):
      self.frame_index = 0

    image = animation[int(self.frame_index)]
    if self.facing_right:
      self.image = image
      self.rect.bottomleft = self.collision_rect.bottomleft
    else:
      flipped_image = pygame.transform.flip(image,True,False)
      self.image = flipped_image
      self.rect.bottomright = self.collision_rect.bottomright

    if self.invincible:
      alpha = self.wave_value()
      self.image.set_alpha(alpha)
    else:
      self.image.set_alpha(255)

    self.rect = self.image.get_rect(midbottom = self.rect.midbottom)

  def run_dust_animation(self):
    if self.status == 'run' and self.on_ground:
      self.dust_frame_index += self.dust_animation_speed
      if self.dust_frame_index >= len(self.dust_run_particles):
        self.dust_frame_index = 0

      dust_particle = self.dust_run_particles[int(self.dust_frame_index)]

      if self.facing_right:
        pos = self.rect.bottomleft - pygame.math.Vector2(6,10)
        self.display_surface.blit(dust_particle,pos)
      else:
        pos = self.rect.bottomright - pygame.math.Vector2(6,10)
        flipped_dust_particle = pygame.transform.flip(dust_particle,True,False)
        self.display_surface.blit(flipped_dust_particle,pos)

  def get_input(self):
    keys = pygame.key.get_pressed()

    if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
      self.direction.x = 1
      self.facing_right = True
    elif keys[pygame.K_LEFT] or keys[pygame.K_a]:
      self.direction.x = -1
      self.facing_right = False
    else:
      self.direction.x = 0

    if keys[pygame.K_SPACE] and self.on_ground or keys[pygame.K_UP] and self.on_ground:
      self.jump()
      self.create_jump_particles(self.rect.midbottom)
    elif keys[pygame.K_DOWN]:
      exit(0)

  def get_status(self):
    if self.direction.y < 0:
      self.status = 'jump'
    elif self.direction.y > 1:
      self.status = 'fall'
    else:
      if self.direction.x != 0:
        self.status = 'run'
      else:
        self.status = 'idle'

  def apply_gravity(self):
    self.direction.y += self.gravity
    self.collision_rect.y += self.direction.y

  def jump(self):
    self.direction.y = self.jump_speed
    self.jump_sound.play()
```

```
  def get_damage(self):
    if not self.invincible:
      self.hit_sound.play()
      self.change_health(-10)
      self.invincible = True
      self.hurt_time = pygame.time.get_ticks()

  def invincibility_timer(self):
    if self.invincible:
      current_time = pygame.time.get_ticks()
      if current_time - self.hurt_time >= self
          .invincibility_duration:
        self.invincible = False

  def wave_value(self):
    value = sin(pygame.time.get_ticks())
    if value >= 0: return 255
    else: return 0

  def update(self):
    self.get_input()
    self.get_status()
    self.animate()
    self.run_dust_animation()
    self.invincibility_timer()
    self.wave_value()
```

Código 6. player.py

II-G. *particles.py*

Para seguir e intentar mostrar algo mas de vida en el player, se utilizan particulas, cuando esta cayendo, o acaba de pisar a un enemigo tendra particulas especiales

```
import pygame
from support import import_folder

class ParticleEffect(pygame.sprite.Sprite):
  def __init__(self,pos,type):
    super().__init__()
    self.frame_index = 0
    self.animation_speed = 0.5
    if type == 'jump':
      self.frames = import_folder('../gamefin/
          graphics/character/dust_particles/
          jump')
    if type == 'land':
      self.frames = import_folder('../gamefin/
          graphics/character/dust_particles/
          land')
    if type == 'explosion':
      self.frames = import_folder('../gamefin/
          graphics/enemy/explosion')
    self.image = self.frames[self.frame_index]
    self.rect = self.image.get_rect(center =
        pos)

  def animate(self):
    self.frame_index += self.animation_speed
    if self.frame_index >= len(self.frames):
      self.kill()
    else:
      self.image = self.frames[int(self.
          frame_index)]

  def update(self,x_shift):
    self.animate()
    self.rect.x += x_shift
```

Código 7. particles.py

II-H. *decoration.py*

La decoracion del nivel, tenga 3 fondos, 1 tile animado de agua y nubes para que el nivel sea mas vivo, el ordenamiento de las nuves es aleatorio

```
from settings import vertical_tile_number,
    tile_size, screen_width
import pygame
from tiles import AnimatedTile, StaticTile
from support import import_folder
from random import choice, randint

class Sky:
  def __init__(self,horizon,style = 'level'):
    self.top = pygame.image.load('../gamefin/
        graphics/decoration/sky/sky_top.png').
        convert()
    self.bottom = pygame.image.load('../
        gamefin/graphics/decoration/sky/
        sky_bottom.png').convert()
    self.middle = pygame.image.load('../
        gamefin/graphics/decoration/sky/
        sky_middle.png').convert()
    self.horizon = horizon

    # stretch
    self.top = pygame.transform.scale(self.top
        ,(screen_width,tile_size))
    self.bottom = pygame.transform.scale(self.
        bottom,(screen_width,tile_size))
    self.middle = pygame.transform.scale(self.
        middle,(screen_width,tile_size))

    self.style = style
    if self.style == 'overworld':
      palm_surfaces = import_folder('../
          gamefin/graphics/overworld/palms')
      self.palms = []

      for surface in [choice(palm_surfaces)
          for image in range(10)]:
        x = randint(0,screen_width)
        y = (self.horizon * tile_size) +
            randint(50,100)
        rect = surface.get_rect(midbottom = (x
            ,y))
        self.palms.append((surface,rect))

      cloud_surfaces = import_folder('../
          gamefin/graphics/overworld/clouds')
      self.clouds = []

      for surface in [choice(cloud_surfaces)
          for image in range(10)]:
        x = randint(0,screen_width)
        y = randint(0,(self.horizon *
            tile_size) - 100)
        rect = surface.get_rect(midbottom = (x
            ,y))
        self.clouds.append((surface,rect))
```

```
  def draw(self,surface):
    for row in range(vertical_tile_number):
      y = row * tile_size
      if row < self.horizon:
        surface.blit(self.top,(0,y))
      elif row == self.horizon:
        surface.blit(self.middle,(0,y))
      else:
        surface.blit(self.bottom,(0,y))

    if self.style == 'overworld':
      for palm in self.palms:
        surface.blit(palm[0],palm[1])
      for cloud in self.clouds:
        surface.blit(cloud[0],cloud[1])

class Water:
  def __init__(self,top,level_width):
    water_start = -screen_width
    water_tile_width = 192
    tile_x_amount = int((level_width +
        screen_width * 2) / water_tile_width)
    self.water_sprites = pygame.sprite.Group()

    for tile in range(tile_x_amount):
      x = tile * water_tile_width +
          water_start
      y = top
      sprite = AnimatedTile(192,x,y,'../
          gamefin/graphics/decoration/water')
      self.water_sprites.add(sprite)

  def draw(self,surface,shift):
    self.water_sprites.update(shift)
    self.water_sprites.draw(surface)

class Clouds:
  def __init__(self,horizon,level_width,
      cloud_number):
    cloud_surf_list = import_folder('../
        gamefin/graphics/decoration/clouds')
    min_x = -screen_width
    max_x = level_width + screen_width
    min_y = 0
    max_y = horizon
    self.cloud_sprites = pygame.sprite.Group()

    for cloud in range(cloud_number):
      cloud = choice(cloud_surf_list)
      x = randint(min_x,max_x)
      y = randint(min_y,max_y)
      sprite = StaticTile(0,x,y,cloud)
      self.cloud_sprites.add(sprite)

  def draw(self,surface,shift):
    self.cloud_sprites.update(shift)
    self.cloud_sprites.draw(surface)
```

Código 8. decoration.py

II-I. overworld.py

Para la seleccion de mapas o overworld se usaran nodos para activar los niveles bloqueados y poder acceder a ellos, el icono del jugador para escoger el mapa

```
import pygame
from game_data import levels
from support import import_folder
from decoration import Sky

class Node(pygame.sprite.Sprite):
  def __init__(self,pos,status,icon_speed,path
      ):
    super().__init__()
    self.frames = import_folder(path)
    self.frame_index = 0
    self.image = self.frames[self.frame_index]
    if status == 'available':
      self.status = 'available'
    else:
      self.status = 'locked'
    self.rect = self.image.get_rect(center =
        pos)

    self.detection_zone = pygame.Rect(self.
        rect.centerx-(icon_speed/2),self.rect.
        centery-(icon_speed/2),icon_speed,
        icon_speed)

  def animate(self):
    self.frame_index += 0.15
    if self.frame_index >= len(self.frames):
      self.frame_index = 0
    self.image = self.frames[int(self.
        frame_index)]

  def update(self):
    if self.status == 'available':
      self.animate()
    else:
      tint_surf = self.image.copy()
      tint_surf.fill('black',None,pygame.
          BLEND_RGBA_MULT)
      self.image.blit(tint_surf,(0,0))

class Icon(pygame.sprite.Sprite):
  def __init__(self,pos):
    super().__init__()
    self.pos = pos
    self.image = pygame.image.load('../gamefin
        /graphics/overworld/hat.png').
        convert_alpha()
    self.rect = self.image.get_rect(center =
        pos)

  def update(self):
    self.rect.center = self.pos

class Overworld:
  def __init__(self,start_level,max_level,
      surface,create_level):

    # setup
    self.display_surface = surface
    self.max_level = max_level
    self.current_level = start_level
    self.create_level = create_level

    # movement logic
    self.moving = False
    self.move_direction = pygame.math.Vector2
```

```
        (0,0)
    self.speed = 8

    # sprites
    self.setup_nodes()
    self.setup_icon()
    self.sky = Sky(8,'overworld')

    # time
    self.start_time = pygame.time.get_ticks()
    self.allow_input = False
    self.timer_length = 300

  def setup_nodes(self):
    self.nodes = pygame.sprite.Group()

    for index, node_data in enumerate(levels.
        values()):
      if index <= self.max_level:
        node_sprite = Node(node_data['node_pos
            '],'available',self.speed,
            node_data['node_graphics'])
      else:
        node_sprite = Node(node_data['node_pos
            '],'locked',self.speed,node_data['
            node_graphics'])
      self.nodes.add(node_sprite)

  def draw_paths(self):
    if self.max_level > 0:
      points = [node['node_pos'] for index,
          node in enumerate(levels.values())
          if index <= self.max_level]
      pygame.draw.lines(self.display_surface,'
          #a04f45',False,points,6)

  def setup_icon(self):
    self.icon = pygame.sprite.GroupSingle()
    icon_sprite = Icon(self.nodes.sprites()[
        self.current_level].rect.center)
    self.icon.add(icon_sprite)

  def input(self):
    keys = pygame.key.get_pressed()

    if not self.moving and self.allow_input:
      if (keys[pygame.K_RIGHT] or keys[pygame.
          K_d]) and self.current_level < self.
          max_level:
        self.move_direction = self.
            get_movement_data('next')
        self.current_level += 1
        self.moving = True
      elif (keys[pygame.K_LEFT] or keys[pygame
          .K_a]) and self.current_level > 0:
        self.move_direction = self.
            get_movement_data('previous')
        self.current_level -= 1
        self.moving = True
      elif keys[pygame.K_SPACE] or keys[pygame
          .K_UP]:
        self.create_level(self.current_level)
      elif keys[pygame.K_DOWN]:
        exit(0)

  def get_movement_data(self,target):
    start = pygame.math.Vector2(self.nodes.
        sprites()[self.current_level].rect.
        center)

    if target == 'next':
      end = pygame.math.Vector2(self.nodes.
          sprites()[self.current_level + 1].
          rect.center)
    else:
      end = pygame.math.Vector2(self.nodes.
          sprites()[self.current_level - 1].
          rect.center)

    return (end - start).normalize()

  def update_icon_pos(self):
    if self.moving and self.move_direction:
      self.icon.sprite.pos += self.
          move_direction * self.speed
      target_node = self.nodes.sprites()[self.
          current_level]
      if target_node.detection_zone.
          collidepoint(self.icon.sprite.pos):
        self.moving = False
        self.move_direction = pygame.math.
            Vector2(0,0)

  def input_timer(self):
    if not self.allow_input:
      current_time = pygame.time.get_ticks()
      if current_time - self.start_time >=
          self.timer_length:
        self.allow_input = True

  def run(self):
    self.input_timer()
    self.input()
    self.update_icon_pos()
    self.icon.update()
    self.nodes.update()

    self.sky.draw(self.display_surface)
    self.draw_paths()
    self.nodes.draw(self.display_surface)
    self.icon.draw(self.display_surface)
```

Código 9. overworld.py

II-J. ui.py

Para la interfaz de usuario que sea amigable con el usuario y poder mostrar la vida y monedas se usara el siguiente codigo:

```
import pygame

class UI:
  def __init__(self,surface):

    # setup
    self.display_surface = surface

    # health
    self.health_bar = pygame.image.load('../
        gamefin/graphics/ui/health_bar.png').
        convert_alpha()
    self.health_bar_topleft = (54,39)
    self.bar_max_width = 152
    self.bar_height = 4
```

```
    # coins
    self.coin = pygame.image.load('../gamefin/graphics/ui/coin.png').convert_alpha()
    self.coin_rect = self.coin.get_rect(topleft = (50,61))
    self.font = pygame.font.Font('../gamefin/graphics/ui/ARCADEPI.ttf',30)

  def show_health(self,current,full):
    self.display_surface.blit(self.health_bar,(20,10))
    current_health_ratio = current / full
    current_bar_width = self.bar_max_width * current_health_ratio
    health_bar_rect = pygame.Rect(self.health_bar_topleft,(current_bar_width,self.bar_height))
    pygame.draw.rect(self.display_surface,'#dc4949',health_bar_rect)

  def show_coins(self,amount):
    self.display_surface.blit(self.coin,self.coin_rect)
    coin_amount_surf = self.font.render(str(amount),False,'#33323d')
    coin_amount_rect = coin_amount_surf.get_rect(midleft = (self.coin_rect.right + 4,self.coin_rect.centery))
    self.display_surface.blit(coin_amount_surf,coin_amount_rect)
```

Código 10. ui.py

II-K. level.py

Para ir finalizando con todo, se hara el nivel, se utilizaran la mayoria de archivos e importados para poder usar sus componentes, a la vez, de aqui se exportaran en main.py y hacer la ejecucion del nivel

```
import pygame
from support import import_csv_layout, import_cut_graphics
from settings import tile_size, screen_height, screen_width
from tiles import Tile, StaticTile, Crate, Coin, Palm
from enemy import Enemy
from decoration import Sky, Water, Clouds
from player import Player
from particles import ParticleEffect
from game_data import levels

class Level:
  def __init__(self,current_level,surface, create_overworld,change_coins, change_health):
    # general setup
    self.display_surface = surface
    self.world_shift = 0
    self.current_x = None

    # audio
    self.coin_sound = pygame.mixer.Sound('../gamefin/audio/effects/coin.wav')
    self.stomp_sound = pygame.mixer.Sound('../gamefin/audio/effects/stomp.wav')

    # overworld connection
    self.create_overworld = create_overworld
    self.current_level = current_level
    level_data = levels[self.current_level]
    self.new_max_level = level_data['unlock']

    # player
    player_layout = import_csv_layout(level_data['player'])
    self.player = pygame.sprite.GroupSingle()
    self.goal = pygame.sprite.GroupSingle()
    self.player_setup(player_layout, change_health)

    # user interface
    self.change_coins = change_coins

    # dust
    self.dust_sprite = pygame.sprite.GroupSingle()
    self.player_on_ground = False

    # explosion particles
    self.explosion_sprites = pygame.sprite.Group()

    # terrain setup
    terrain_layout = import_csv_layout(level_data['terrain'])
    self.terrain_sprites = self.create_tile_group(terrain_layout,'terrain')

    # grass setup
    grass_layout = import_csv_layout(level_data['grass'])
    self.grass_sprites = self.create_tile_group(grass_layout,'grass')

    # crates
    crate_layout = import_csv_layout(level_data['crates'])
    self.crate_sprites = self.create_tile_group(crate_layout,'crates')

    # coins
    coin_layout = import_csv_layout(level_data['coins'])
    self.coin_sprites = self.create_tile_group(coin_layout,'coins')

    # foreground palms
    fg_palm_layout = import_csv_layout(level_data['fg palms'])
    self.fg_palm_sprites = self.create_tile_group(fg_palm_layout,'fg palms')

    # background palms
    bg_palm_layout = import_csv_layout(level_data['bg palms'])
    self.bg_palm_sprites = self.
```

```
        create_tile_group(bg_palm_layout,'bg palms')

    # enemy
    enemy_layout = import_csv_layout(level_data['enemies'])
    self.enemy_sprites = self.create_tile_group(enemy_layout,'enemies')

    # constraint
    constraint_layout = import_csv_layout(level_data['constraints'])
    self.constraint_sprites = self.create_tile_group(constraint_layout,'constraint')

    # decoration
    self.sky = Sky(8)
    level_width = len(terrain_layout[0]) * tile_size
    self.water = Water(screen_height - 20,level_width)
    self.clouds = Clouds(400,level_width,30)

  def create_tile_group(self,layout,type):
    sprite_group = pygame.sprite.Group()

    for row_index, row in enumerate(layout):
      for col_index,val in enumerate(row):
        if val != '-1':
          x = col_index * tile_size
          y = row_index * tile_size

          if type == 'terrain':
            terrain_tile_list = import_cut_graphics('../gamefin/graphics/terrain/terrain_tiles.png')
            tile_surface = terrain_tile_list[int(val)]
            sprite = StaticTile(tile_size,x,y,tile_surface)

          if type == 'grass':
            grass_tile_list = import_cut_graphics('../gamefin/graphics/decoration/grass/grass.png')
            tile_surface = grass_tile_list[int(val)]
            sprite = StaticTile(tile_size,x,y,tile_surface)

          if type == 'crates':
            sprite = Crate(tile_size,x,y)

          if type == 'coins':
            if val == '0': sprite = Coin(tile_size,x,y,'../gamefin/graphics/coins/gold',5)
            if val == '1': sprite = Coin(tile_size,x,y,'../gamefin/graphics/coins/silver',1)

          if type == 'fg palms':
            if val == '0': sprite = Palm(tile_size,x,y,'../gamefin/graphics/terrain/palm_small',38)
            if val == '1': sprite = Palm(tile_size,x,y,'../gamefin/graphics/terrain/palm_large',64)

          if type == 'bg palms':
            sprite = Palm(tile_size,x,y,'../gamefin/graphics/terrain/palm_bg',64)

          if type == 'enemies':
            sprite = Enemy(tile_size,x,y)

          if type == 'constraint':
            sprite = Tile(tile_size,x,y)

          sprite_group.add(sprite)

    return sprite_group

  def player_setup(self,layout,change_health):
    for row_index, row in enumerate(layout):
      for col_index,val in enumerate(row):
        x = col_index * tile_size
        y = row_index * tile_size
        if val == '0':
          sprite = Player((x,y),self.display_surface,self.create_jump_particles,change_health)
          self.player.add(sprite)
        if val == '1':
          hat_surface = pygame.image.load('../gamefin/graphics/character/hat.png').convert_alpha()
          sprite = StaticTile(tile_size,x,y,hat_surface)
          self.goal.add(sprite)

  def enemy_collision_reverse(self):
    for enemy in self.enemy_sprites.sprites():
      if pygame.sprite.spritecollide(enemy,self.constraint_sprites,False):
        enemy.reverse()

  def create_jump_particles(self,pos):
    if self.player.sprite.facing_right:
      pos -= pygame.math.Vector2(10,5)
    else:
      pos += pygame.math.Vector2(10,-5)
    jump_particle_sprite = ParticleEffect(pos,'jump')
    self.dust_sprite.add(jump_particle_sprite)

  def horizontal_movement_collision(self):
    player = self.player.sprite
    player.collision_rect.x += player.direction.x * player.speed
    collidable_sprites = self.terrain_sprites.sprites() + self.crate_sprites.sprites() + self.fg_palm_sprites.sprites()
    for sprite in collidable_sprites:
      if sprite.rect.colliderect(player.collision_rect):
```

```
        if player.direction.x < 0:
          player.collision_rect.left = sprite.rect.right
          player.on_left = True
          self.current_x = player.rect.left
        elif player.direction.x > 0:
          player.collision_rect.right = sprite.rect.left
          player.on_right = True
          self.current_x = player.rect.right

    def vertical_movement_collision(self):
      player = self.player.sprite
      player.apply_gravity()
      collidable_sprites = self.terrain_sprites.sprites() + self.crate_sprites.sprites() + self.fg_palm_sprites.sprites()

      for sprite in collidable_sprites:
        if sprite.rect.colliderect(player.collision_rect):
          if player.direction.y > 0:
            player.collision_rect.bottom = sprite.rect.top
            player.direction.y = 0
            player.on_ground = True
          elif player.direction.y < 0:
            player.collision_rect.top = sprite.rect.bottom
            player.direction.y = 0
            player.on_ceiling = True

      if player.on_ground and player.direction.y < 0 or player.direction.y > 1:
        player.on_ground = False

    def scroll_x(self):
      player = self.player.sprite
      player_x = player.rect.centerx
      direction_x = player.direction.x

      if player_x < screen_width / 4 and direction_x < 0:
        self.world_shift = 8
        player.speed = 0
      elif player_x > screen_width - (screen_width / 4) and direction_x > 0:
        self.world_shift = -8
        player.speed = 0
      else:
        self.world_shift = 0
        player.speed = 8

    def get_player_on_ground(self):
      if self.player.sprite.on_ground:
        self.player_on_ground = True
      else:
        self.player_on_ground = False

    def create_landing_dust(self):
      if not self.player_on_ground and self.player.sprite.on_ground and not self.dust_sprite.sprites():
        if self.player.sprite.facing_right:
          offset = pygame.math.Vector2(10,15)
        else:
          offset = pygame.math.Vector2(-10,15)
        fall_dust_particle = ParticleEffect(self.player.sprite.rect.midbottom - offset,'land')
        self.dust_sprite.add(fall_dust_particle)

    def check_death(self):
      if self.player.sprite.rect.top > screen_height:
        self.create_overworld(self.current_level,0)

    def check_win(self):
      if pygame.sprite.spritecollide(self.player.sprite,self.goal,False):
        self.create_overworld(self.current_level,self.new_max_level)

    def check_coin_collisions(self):
      collided_coins = pygame.sprite.spritecollide(self.player.sprite,self.coin_sprites,True)
      if collided_coins:
        self.coin_sound.play()
        for coin in collided_coins:
          self.change_coins(coin.value)

    def check_enemy_collisions(self):
      enemy_collisions = pygame.sprite.spritecollide(self.player.sprite,self.enemy_sprites,False)

      if enemy_collisions:
        for enemy in enemy_collisions:
          enemy_center = enemy.rect.centery
          enemy_top = enemy.rect.top
          player_bottom = self.player.sprite.rect.bottom
          if enemy_top < player_bottom < enemy_center and self.player.sprite.direction.y >= 0:
            self.stomp_sound.play()
            self.player.sprite.direction.y = -15
            explosion_sprite = ParticleEffect(enemy.rect.center,'explosion')
            self.explosion_sprites.add(explosion_sprite)
            enemy.kill()
          else:
            self.player.sprite.get_damage()

    def run(self):
      # run the entire game / level

      # sky
      self.sky.draw(self.display_surface)
      self.clouds.draw(self.display_surface,self.world_shift)

      # background palms
      self.bg_palm_sprites.update(self.world_shift)
      self.bg_palm_sprites.draw(self.display_surface)

      # dust particles
      self.dust_sprite.update(self.world_shift)
      self.dust_sprite.draw(self.display_surface
```

```
        )

    # terrain
    self.terrain_sprites.update(self.
        world_shift)
    self.terrain_sprites.draw(self.
        display_surface)

    # enemy
    self.enemy_sprites.update(self.world_shift
        )
    self.constraint_sprites.update(self.
        world_shift)
    self.enemy_collision_reverse()
    self.enemy_sprites.draw(self.
        display_surface)
    self.explosion_sprites.update(self.
        world_shift)
    self.explosion_sprites.draw(self.
        display_surface)

    # crate
    self.crate_sprites.update(self.world_shift
        )
    self.crate_sprites.draw(self.
        display_surface)

    # grass
    self.grass_sprites.update(self.world_shift
        )
    self.grass_sprites.draw(self.
        display_surface)

    # coins
    self.coin_sprites.update(self.world_shift)
    self.coin_sprites.draw(self.
        display_surface)

    # foreground palms
    self.fg_palm_sprites.update(self.
        world_shift)
    self.fg_palm_sprites.draw(self.
        display_surface)

    # player sprites
    self.player.update()
    self.horizontal_movement_collision()

    self.get_player_on_ground()
    self.vertical_movement_collision()
    self.create_landing_dust()

    self.scroll_x()
    self.player.draw(self.display_surface)
    self.goal.update(self.world_shift)
    self.goal.draw(self.display_surface)

    self.check_death()
    self.check_win()

    self.check_coin_collisions()
    self.check_enemy_collisions()

    # water
    self.water.draw(self.display_surface,self.
        world_shift)
```

Código 11. level.py

II-L. main.py

Para poder ejecutar el juego se hace desde main.py y mostrar la pantalla del juego

```
import pygame, sys
from settings import *
from level import Level
from overworld import Overworld
from ui import UI

class Game:
  def __init__(self):

    # game attributes
    self.max_level = 0
    self.max_health = 100
    self.cur_health = 100
    self.coins = 0

    # audio
    self.level_bg_music = pygame.mixer.Sound('
        ../gamefin/audio/level_music.wav')
    self.overworld_bg_music = pygame.mixer.
        Sound('../gamefin/audio/
        overworld_music.wav')

    # overworld creation
    self.overworld = Overworld(0,self.
        max_level,screen,self.create_level)
    self.status = 'overworld'
    self.overworld_bg_music.play(loops = -1)

    # user interface
    self.ui = UI(screen)

  def create_level(self,current_level):
    self.level = Level(current_level,screen,
        self.create_overworld,self.
        change_coins,self.change_health)
    self.status = 'level'
    self.overworld_bg_music.stop()
    self.level_bg_music.play(loops = -1)

  def create_overworld(self,current_level,
      new_max_level):
    if new_max_level > self.max_level:
      self.max_level = new_max_level
    self.overworld = Overworld(current_level,
        self.max_level,screen,self.
        create_level)
    self.status = 'overworld'
    self.overworld_bg_music.play(loops = -1)
    self.level_bg_music.stop()

  def change_coins(self,amount):
    self.coins += amount

  def change_health(self,amount):
    self.cur_health += amount

  def check_game_over(self):
    if self.cur_health <= 0:
      self.cur_health = 100
```

```
      self.coins = 0
      self.max_level = 0
      self.overworld = Overworld(0,self.
          max_level,screen,self.create_level)
      self.status = 'overworld'
      self.level_bg_music.stop()
      self.overworld_bg_music.play(loops = -1)

  def run(self):
    if self.status == 'overworld':
      self.overworld.run()
    else:
      self.level.run()
      self.ui.show_health(self.cur_health,self
          .max_health)
      self.ui.show_coins(self.coins)
      self.check_game_over()

# Pygame setup
pygame.init()
screen = pygame.display.set_mode((screen_width
    ,screen_height))
clock = pygame.time.Clock()
game = Game()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit()

  screen.fill('grey')
  game.run()

  pygame.display.update()
  clock.tick(60)
```

Código 12. main.py

II-M. arduino.py

Aqui viene el codigo para poder ejecutar el archivo de arduino y poder utilizar el joystick shield en el juego

```
import time
import serial
import pyautogui
import os

# create a translation table for how keys are
    interpreted
KEY_MAPPING = {
    'A'          : '',
    'B'          : 'up',
    'C'          : '',
    'D'          : '',
    'E'          : 'e',
    'F'          : 'down',
    'JOY_ENTER'  : '',
    'JOY_LEFT'   : 'left',
    'JOY_RIGHT'  : 'right',
    'JOY_UP'     : '',
    'JOY_DOWN'   : ''
}

def handle_line(line):
    "Handles a single line of EVENT sent over
        serial"
    key, arg = [x.decode("utf-8") for x in
        line.split()]

    if key not in KEY_MAPPING:
        raise NotImplementedError("key %s
            unrecognised, no mapping" % key)

    if arg == 'UP':
        pyautogui.keyUp(KEY_MAPPING[key])
    elif arg == 'DOWN':
        pyautogui.keyDown(KEY_MAPPING[key])
    else:
        raise NotImplementedError('Unknown
            event %s' % arg)

    # Verificar si la tecla es 'F' para salir
        del bucle
    if key == 'F':
        print("Tecla F detectada. Saliendo del
            programa.")
        os._exit(0)

def handle_com(path):
    "Opens the serial socket and reads all
        events interpreting them"
    with serial.Serial(path, 9600) as arduino:
        while True:
            line = arduino.readline()
            handle_line(line)

if __name__ == '__main__':
    while True:
        try:
            print("Connecting to device")
            handle_com('COM3')
        except serial.serialutil.
            SerialException as se:
            print("Error with connection, will
                attempt to connect again in
                10 secs...")
            time.sleep(10)
```

Código 13. arduino.py

II-N. twos.py

Ahora, para poder ejecutar ambos sin errores de tiempo (clock y time) se corren en paralelo

```
import subprocess

def ejecutar_archivo(archivo):
    subprocess.run(["python", archivo])

if __name__ == "__main__":
    archivo1 = "../gamefin/code/arduino.py"
    archivo2 = "../gamefin/code/main.py"

    proceso1 = subprocess.Popen(["python",
        archivo1])
    proceso2 = subprocess.Popen(["python",
        archivo2])

    proceso1.wait()
    proceso2.wait()
```

Código 14. twos.py

III. Resultados

Para ejecutar el videojuego de manera paralela y poder usar el joystick, ejecutamos el archivo twos.py

Los botones que utilizaremos seran el [A][B][C][D] y el joystick en el eje X, con el boton [F] cerraremos el programa y el juego

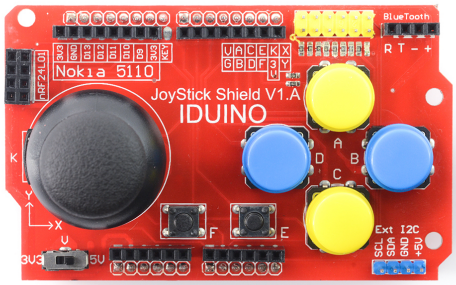

Figura 1. Joystick Shield V1.a.

El juego se ve de la siguiente manera, empezando por el overworld, como se muestra en la siguira 1

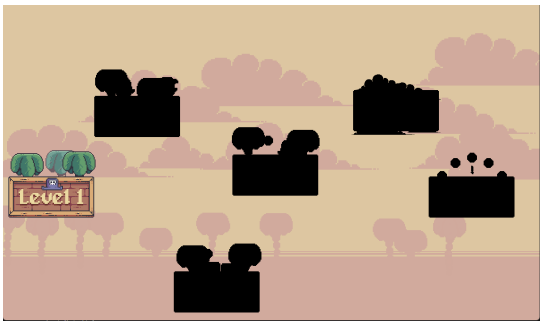

Figura 2. Seleccion de niveles (nivel 1).

una vez presionamos [boton de saltar] entraremos en el nivel, donde nos podremos mover con [A][D]/[←][→]/[Jostick] y podemos saltar con los botones [espacio]/[↑]/[boton de saltar], como se muestra en la figura 2, donde ya estamos dentro del nivel y podemos recorlerlo, coleccionar monedas o matar enemigos

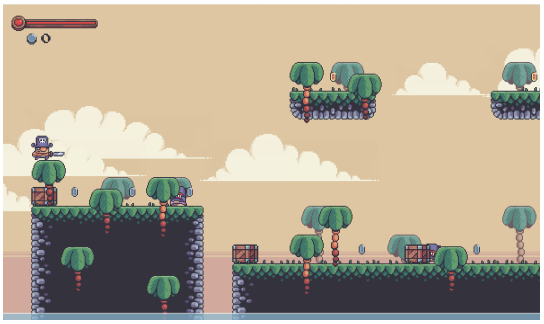

Figura 3. nivel 1.

Una vez lleguemos al final del nivel se encuentra un sombrero pirata, que sera una prueba de que se paso el nivel, y se desbloquea el nivel 2, asi sera en cada uno, hasta el nivel 6

Figura 4. nivel 1.

si conseguimos el sombre al final del nivel, nos saca del nivel, pero abremos desbloqueado el nivel 2, eso se hace con cada nivel, hasta el nivel 6 que es el limite, como se muestran en las figuras 4 y 5

Figura 5. Seleccion de niveles (nivel 1 y 2

.

El juego contando con 6 niveles se puede pasar rapido y sin problemas, puedes volver a jugar niveles anteriores o los que uno quiera

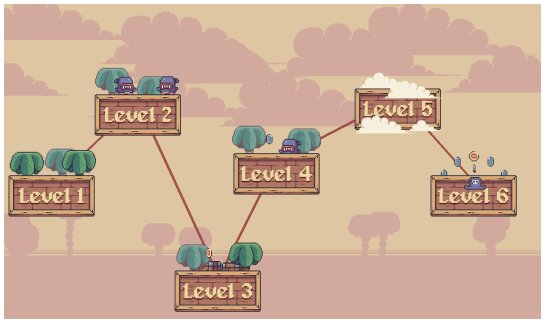

Figura 6. Seleccion de niveles (niveles completos

.

IV. Conclusiones

Hacer un videojuego en Python con PyGame y Arduino fue muy interesante, sobre todo porque no es una interfaz de desarrollo de videojuegos y tiene sus problemas, pero una vez arreglados puede terminar un juego interesante con resultados llamativos, hubo partes del codigo que se complicaron y otras que estuvieron rapido, pero aun asi es muy interesante desarrollar en python. Volver a desarrollar apps en python despues de dejarlo un tiempo fue interesante

Referencias

- WorkOfArtiz (2018). *funduino_controller.ino*. Enlace al código en GitHub