

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:

```
1 vertical_tile_number = 11
2 tile_size = 64
3
4 screen_height = vertical_tile_number *
   tile_size
5 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:

```
1 from csv import reader
2 from settings import tile_size
3 from os import walk
4 import pygame
5
6 def import_folder(path):
7     surface_list = []
8
9     for __, __, image_files in walk(path):
10         for image in image_files:
11             full_path = path + '/' + image
12             image_surf = pygame.image.load(full_path)
13             image_surf.convert_alpha()
14             surface_list.append(image_surf)
15
16     return surface_list
17
18 def import_csv_layout(path):
19     terrain_map = []
20     with open(path) as map:
21         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

```
1 level_0 = {
2     'terrain': '../gamefin/levels/0/
   level_0_terrain.csv',
3     'coins': '../gamefin/levels/0/level_0_coins
   .csv',
4     'fg palms': '../gamefin/levels/0/
   level_0_fg_palms.csv',
5     'bg palms': '../gamefin/levels/0/
   level_0_bg_palms.csv',
6     'crates': '../gamefin/levels/0/
   level_0_crates.csv',
7     'enemies': '../gamefin/levels/0/
   level_0_enemies.csv',
8     'constraints': '../gamefin/levels/0/
   level_0_constraints.csv',
9     'player': '../gamefin/levels/0/
   level_0_player.csv',
10    'grass': '../gamefin/levels/0/
   level_0_grass.csv',
11    'node_pos': (110,400),
12    'unlock': 1,
13    'node_graphics': '../gamefin/graphics/
   overworld/0'}
```

```

14 level_1 = {
15     'terrain': '../gamefin/levels/1/
16         level_1_terrain.csv',
17     'coins': '../gamefin/levels/1/level_1_coins
18         .csv',
19     'fg palms': '../gamefin/levels/1/
20         level_1_fg_palms.csv',
21     'bg palms': '../gamefin/levels/1/
22         level_1_bg_palms.csv',
23     'crates': '../gamefin/levels/1/
24         level_1_crates.csv',
25     'enemies': '../gamefin/levels/1/
26         level_1_enemies.csv',
27     'constraints': '../gamefin/levels/1/
28         level_1_constraints.csv',
29     'player': '../gamefin/levels/1/
30         level_1_player.csv',
31     'grass': '../gamefin/levels/1/
32         level_1_grass.csv',
33     'node_pos': (300,220),
34     'node_graphics': '../gamefin/graphics/
35         overworld/1',
36     'unlock': 2}
37 level_2 = {
38     'terrain': '../gamefin/levels/2/
39         level_2_terrain.csv',
40     'coins': '../gamefin/levels/2/level_2_coins
41         .csv',
42     'fg palms': '../gamefin/levels/2/
43         level_2_fg_palms.csv',
44     'bg palms': '../gamefin/levels/2/
45         level_2_bg_palms.csv',
46     'crates': '../gamefin/levels/2/
47         level_2_crates.csv',
48     'enemies': '../gamefin/levels/2/
49         level_2_enemies.csv',
50     'constraints': '../gamefin/levels/2/
51         level_2_constraints.csv',
52     'player': '../gamefin/levels/2/
53         level_2_player.csv',
54     'grass': '../gamefin/levels/2/
55         level_2_grass.csv',
56     'node_pos': (480,610),
57     'node_graphics': '../gamefin/graphics/
58         overworld/2',
59     'unlock': 3}
60 level_3 = {
61     'terrain': '../gamefin/levels/2/
62         level_2_terrain.csv',
63     'coins': '../gamefin/levels/2/level_2_coins
64         .csv',
65     'fg palms': '../gamefin/levels/2/
66         level_2_fg_palms.csv',
67     'bg palms': '../gamefin/levels/2/
68         level_2_bg_palms.csv',
69     'crates': '../gamefin/levels/2/
70         level_2_crates.csv',
71     'enemies': '../gamefin/levels/2/
72         level_2_enemies.csv',
73     'constraints': '../gamefin/levels/2/
74         level_2_constraints.csv',
75     'player': '../gamefin/levels/2/
76         level_2_player.csv',
77     'grass': '../gamefin/levels/2/
78         level_2_grass.csv',
79     'node_pos': (610,350),
80     'node_graphics': '../gamefin/graphics/
81         overworld/3',
82     'unlock': 4}
83 level_4 = {
84     'terrain': '../gamefin/levels/2/
85         level_2_terrain.csv',
86     'coins': '../gamefin/levels/2/level_2_coins
87         .csv',
88     'fg palms': '../gamefin/levels/2/
89         level_2_fg_palms.csv',
90     'bg palms': '../gamefin/levels/2/
91         level_2_bg_palms.csv',
92     'crates': '../gamefin/levels/2/
93         level_2_crates.csv',
94     'enemies': '../gamefin/levels/2/
95         level_2_enemies.csv',
96     'constraints': '../gamefin/levels/2/
97         level_2_constraints.csv',
98     'player': '../gamefin/levels/2/
99         level_2_player.csv',
100    'grass': '../gamefin/levels/2/
101        level_2_grass.csv',
102    'node_pos': (880,210),
103    'node_graphics': '../gamefin/graphics/
104        overworld/4',
105    'unlock': 5}
106 level_5 = {
107     'terrain': '../gamefin/levels/2/
108         level_2_terrain.csv',
109     'coins': '../gamefin/levels/2/level_2_coins
110         .csv',
111     'fg palms': '../gamefin/levels/2/
112         level_2_fg_palms.csv',
113     'bg palms': '../gamefin/levels/2/
114         level_2_bg_palms.csv',
115     'crates': '../gamefin/levels/2/
116         level_2_crates.csv',
117     'enemies': '../gamefin/levels/2/
118         level_2_enemies.csv',
119     'constraints': '../gamefin/levels/2/
120         level_2_constraints.csv',
121     'player': '../gamefin/levels/2/
122         level_2_player.csv',
123     'grass': '../gamefin/levels/2/
124         level_2_grass.csv',
125     'node_pos': (1050,400),
126     'node_graphics': '../gamefin/graphics/
127         overworld/5',
128     'unlock': 5}
129 levels = {
130     0: level_0,
131     1: level_1,
132     2: level_2,
133     3: level_3,
134     4: level_4,
135     5: level_5}

```

Código 3. game_data.py

II-D. tiles.py

El siguiente código muestra los niveles y los archivos csv que tendrá cada nivel, siendo importados directamente desde el directorio para ser leídos posteriormente

```

1 import pygame
2 from support import import_folder

```

```

3 class Tile(pygame.sprite.Sprite):
4     def __init__(self, size, x, y):
5         super().__init__()
6         self.image = pygame.Surface((size, size))
7         self.rect = self.image.get_rect(topleft =
8             (x, y))
9
10    def update(self, shift):
11        self.rect.x += shift
12
13    class StaticTile(Tile):
14        def __init__(self, size, x, y, surface):
15            super().__init__(size, x, y)
16            self.image = surface
17
18    class Crate(StaticTile):
19        def __init__(self, size, x, y):
20            super().__init__(size, x, y, pygame.image.
21                load('../gamefin/graphics/terrain/
22                crate.png').convert_alpha())
23            offset_y = y + size
24            self.rect = self.image.get_rect(bottomleft
25                = (x, offset_y))
26
27    class AnimatedTile(Tile):
28        def __init__(self, size, x, y, path):
29            super().__init__(size, x, y)
30            self.frames = import_folder(path)
31            self.frame_index = 0
32            self.image = self.frames[self.frame_index]
33
34        def animate(self):
35            self.frame_index += 0.15
36            if self.frame_index >= len(self.frames):
37                self.frame_index = 0
38            self.image = self.frames[int(self.
39                frame_index)]
40
41        def update(self, shift):
42            self.animate()
43            self.rect.x += shift
44
45    class Coin(AnimatedTile):
46        def __init__(self, size, x, y, path, value):
47            super().__init__(size, x, y, path)
48            center_x = x + int(size / 2)
49            center_y = y + int(size / 2)
50            self.rect = self.image.get_rect(center = (
51                center_x, center_y))
52            self.value = value
53
54    class Palm(AnimatedTile):
55        def __init__(self, size, x, y, path, offset):
56            super().__init__(size, x, y, path)
57            offset_y = y - offset
58            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

```

1 from tiles import AnimatedTile
2 from random import randint
3
4 class Enemy(AnimatedTile):
5     def __init__(self, size, x, y):
6         super().__init__(size, x, y, '../gamefin/
7             graphics/enemy/run')
8         self.rect.y += size - self.image.get_size
9             ()[1]
10        self.speed = randint(3, 5)
11
12    def move(self):
13        self.rect.x += self.speed
14
15    def reverse_image(self):
16        if self.speed > 0:
17            self.image = pygame.transform.flip(self.
18                image, True, False)
19
20    def reverse(self):
21        self.speed *= -1
22
23    def update(self, shift):
24        self.rect.x += shift
25        self.animate()
26        self.move()
27        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

```

1 import pygame
2 from support import import_folder
3 from math import sin
4
5 class Player(pygame.sprite.Sprite):
6     def __init__(self, pos, surface,
7         create_jump_particles, change_health):
8         super().__init__()
9         self.import_character_assets()
10        self.frame_index = 0
11        self.animation_speed = 0.15
12        self.image = self.animations['idle'][self.
13            frame_index]
14        self.rect = self.image.get_rect(topleft =
15            pos)
16
17        # dust particles
18        self.import_dust_run_particles()
19        self.dust_frame_index = 0
20        self.dust_animation_speed = 0.15
21        self.display_surface = surface
22        self.create_jump_particles =
23            create_jump_particles
24
25        # player movement
26        self.direction = pygame.math.Vector2(0, 0)
27        self.speed = 8
28        self.gravity = 0.8
29        self.jump_speed = -16
30        self.collison_rect = pygame.Rect(self.
31            rect.topleft, (50, self.rect.height))

```

```

27 # player status
28 self.status = 'idle'
29 self.facing_right = True
30 self.on_ground = False
31 self.on_ceiling = False
32 self.on_left = False
33 self.on_right = False
34
35 # health management
36 self.change_health = change_health
37 self.invincible = False
38 self.invincibility_duration = 500
39 self.hurt_time = 0
40
41 # audio
42 self.jump_sound = pygame.mixer.Sound('../
43     gamefin/audio/effects/jump.wav')
44 self.jump_sound.set_volume(0.5)
45 self.hit_sound = pygame.mixer.Sound('../
46     gamefin/audio/effects/hit.wav')
47
48 def import_character_assets(self):
49     character_path = '../gamefin/graphics/
50         character/'
51     self.animations = {'idle':[], 'run':[], '
52         jump':[], 'fall':[]}
53
54     for animation in self.animations.keys():
55         full_path = character_path + animation
56         self.animations[animation] =
57             import_folder(full_path)
58
59 def import_dust_run_particles(self):
60     self.dust_run_particles = import_folder('
61         ../gamefin/graphics/character/
62         dust_particles/run')
63
64 def animate(self):
65     animation = self.animations[self.status]
66
67     # loop over frame index
68     self.frame_index += self.animation_speed
69     if self.frame_index >= len(animation):
70         self.frame_index = 0
71
72     image = animation[int(self.frame_index)]
73     if self.facing_right:
74         self.image = image
75         self.rect.bottomleft = self.
76             collision_rect.bottomleft
77     else:
78         flipped_image = pygame.transform.flip(
79             image, True, False)
80         self.image = flipped_image
81         self.rect.bottomright = self.
82             collision_rect.bottomright
83
84     if self.invincible:
85         alpha = self.wave_value()
86         self.image.set_alpha(alpha)
87     else:
88         self.image.set_alpha(255)
89
90     self.rect = self.image.get_rect(midbottom
91         = self.rect.midbottom)
92
93 def run_dust_animation(self):
94     if self.status == 'run' and self.on_ground
95         :
96         self.dust_frame_index += self.
97             dust_animation_speed
98         if self.dust_frame_index >= len(self.
99             dust_run_particles):
100             self.dust_frame_index = 0
101
102     dust_particle = self.dust_run_particles[
103         int(self.dust_frame_index)]
104
105     if self.facing_right:
106         pos = self.rect.bottomleft - pygame.
107             math.Vector2(6,10)
108         self.display_surface.blit(
109             dust_particle, pos)
110     else:
111         pos = self.rect.bottomright - pygame.
112             math.Vector2(6,10)
113         flipped_dust_particle = pygame.
114             transform.flip(dust_particle, True,
115                 False)
116         self.display_surface.blit(
117             flipped_dust_particle, pos)
118
119 def get_input(self):
120     keys = pygame.key.get_pressed()
121
122     if keys[pygame.K_RIGHT] or keys[pygame.K_d
123         ]:
124         self.direction.x = 1
125         self.facing_right = True
126     elif keys[pygame.K_LEFT] or keys[pygame.
127         K_a]:
128         self.direction.x = -1
129         self.facing_right = False
130     else:
131         self.direction.x = 0
132
133     if keys[pygame.K_SPACE] and self.on_ground
134         or keys[pygame.K_UP] and self.
135             on_ground:
136         self.jump()
137         self.create_jump_particles(self.rect.
138             midbottom)
139     elif keys[pygame.K_DOWN]:
140         exit(0)
141
142 def get_status(self):
143     if self.direction.y < 0:
144         self.status = 'jump'
145     elif self.direction.y > 1:
146         self.status = 'fall'
147     else:
148         if self.direction.x != 0:
149             self.status = 'run'
150         else:
151             self.status = 'idle'
152
153 def apply_gravity(self):
154     self.direction.y += self.gravity
155     self.collision_rect.y += self.direction.y
156
157 def jump(self):
158     self.direction.y = self.jump_speed
159     self.jump_sound.play()

```

```

135
136 def get_damage(self):
137     if not self.invincible:
138         self.hit_sound.play()
139         self.change_health(-10)
140         self.invincible = True
141         self.hurt_time = pygame.time.get_ticks()
142
143 def invincibility_timer(self):
144     if self.invincible:
145         current_time = pygame.time.get_ticks()
146         if current_time - self.hurt_time >= self
147             .invincibility_duration:
148             self.invincible = False
149
150 def wave_value(self):
151     value = sin(pygame.time.get_ticks())
152     if value >= 0: return 255
153     else: return 0
154
155 def update(self):
156     self.get_input()
157     self.get_status()
158     self.animate()
159     self.run_dust_animation()
160     self.invincibility_timer()
161     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

```

1 import pygame
2 from support import import_folder
3
4 class ParticleEffect(pygame.sprite.Sprite):
5     def __init__(self,pos,type):
6         super().__init__()
7         self.frame_index = 0
8         self.animation_speed = 0.5
9         if type == 'jump':
10             self.frames = import_folder('../gamefin/
11                 graphics/character/dust_particles/
12                 jump')
13             if type == 'land':
14                 self.frames = import_folder('../gamefin/
15                     graphics/character/dust_particles/
16                     land')
17             if type == 'explosion':
18                 self.frames = import_folder('../gamefin/
19                     graphics/enemy/explosion')
20             self.image = self.frames[self.frame_index]
21             self.rect = self.image.get_rect(center =
22                 pos)
23
24 def animate(self):
25     self.frame_index += self.animation_speed
26     if self.frame_index >= len(self.frames):
27         self.kill()
28     else:
29         self.image = self.frames[int(self.
30             frame_index)]
31
32
33
34
35
36
37

```

```

25 def update(self,x_shift):
26     self.animate()
27     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 nubes es aleatorio

```

1 from settings import vertical_tile_number,
2     tile_size, screen_width
3 import pygame
4 from tiles import AnimatedTile, StaticTile
5 from support import import_folder
6 from random import choice, randint
7
8 class Sky:
9     def __init__(self,horizon,style = 'level'):
10         self.top = pygame.image.load('../gamefin/
11             graphics/decoration/sky/sky_top.png').
12             convert()
13         self.bottom = pygame.image.load('../
14             gamefin/graphics/decoration/sky/
15             sky_bottom.png').convert()
16         self.middle = pygame.image.load('../
17             gamefin/graphics/decoration/sky/
18             sky_middle.png').convert()
19         self.horizon = horizon
20
21 # stretch
22 self.top = pygame.transform.scale(self.top
23     ,(screen_width,tile_size))
24 self.bottom = pygame.transform.scale(self.
25     bottom,(screen_width,tile_size))
26 self.middle = pygame.transform.scale(self.
27     middle,(screen_width,tile_size))
28
29 self.style = style
30 if self.style == 'overworld':
31     palm_surfaces = import_folder('../
32         gamefin/graphics/overworld/palms')
33     self.palms = []
34
35 for surface in [choice(palm_surfaces)
36     for image in range(10)]:
37     x = randint(0,screen_width)
38     y = (self.horizon * tile_size) +
39         randint(50,100)
40     rect = surface.get_rect(midbottom = (x
41         ,y))
42     self.palms.append((surface,rect))
43
44 cloud_surfaces = import_folder('../
45     gamefin/graphics/overworld/clouds')
46 self.clouds = []
47
48 for surface in [choice(cloud_surfaces)
49     for image in range(10)]:
50     x = randint(0,screen_width)
51     y = randint(0,(self.horizon *
52         tile_size) - 100)
53     rect = surface.get_rect(midbottom = (x
54         ,y))
55     self.clouds.append((surface,rect))

```

```

38 def draw(self, surface):
39     for row in range(vertical_tile_number):
40         y = row * tile_size
41         if row < self.horizon:
42             surface.blit(self.top, (0, y))
43         elif row == self.horizon:
44             surface.blit(self.middle, (0, y))
45         else:
46             surface.blit(self.bottom, (0, y))
47
48     if self.style == 'overworld':
49         for palm in self.palms:
50             surface.blit(palm[0], palm[1])
51         for cloud in self.clouds:
52             surface.blit(cloud[0], cloud[1])
53
54 class Water:
55     def __init__(self, top, level_width):
56         water_start = -screen_width
57         water_tile_width = 192
58         tile_x_amount = int((level_width +
59                             screen_width * 2) / water_tile_width)
60         self.water_sprites = pygame.sprite.Group()
61
62         for tile in range(tile_x_amount):
63             x = tile * water_tile_width +
64                 water_start
65             y = top
66             sprite = AnimatedTile(192, x, y, '../
67                                     gamefin/graphics/decoration/water')
68             self.water_sprites.add(sprite)
69
70     def draw(self, surface, shift):
71         self.water_sprites.update(shift)
72         self.water_sprites.draw(surface)
73
74 class Clouds:
75     def __init__(self, horizon, level_width,
76                 cloud_number):
77         cloud_surf_list = import_folder('../
78                                     gamefin/graphics/decoration/clouds')
79         min_x = -screen_width
80         max_x = level_width + screen_width
81         min_y = 0
82         max_y = horizon
83         self.cloud_sprites = pygame.sprite.Group()
84
85         for cloud in range(cloud_number):
86             cloud = choice(cloud_surf_list)
87             x = randint(min_x, max_x)
88             y = randint(min_y, max_y)
89             sprite = StaticTile(0, x, y, cloud)
90             self.cloud_sprites.add(sprite)
91
92     def draw(self, surface, shift):
93         self.cloud_sprites.update(shift)
94         self.cloud_sprites.draw(surface)
95
96 import pygame
97 from game_data import levels
98 from support import import_folder
99 from decoration import Sky
100
101 class Node(pygame.sprite.Sprite):
102     def __init__(self, pos, status, icon_speed, path):
103         super().__init__()
104         self.frames = import_folder(path)
105         self.frame_index = 0
106         self.image = self.frames[self.frame_index]
107         if status == 'available':
108             self.status = 'available'
109         else:
110             self.status = 'locked'
111         self.rect = self.image.get_rect(center =
112                                         pos)
113
114         self.detection_zone = pygame.Rect(self.
115                                         rect.centerx- (icon_speed/2), self.rect.
116                                         centery- (icon_speed/2), icon_speed,
117                                         icon_speed)
118
119     def animate(self):
120         self.frame_index += 0.15
121         if self.frame_index >= len(self.frames):
122             self.frame_index = 0
123         self.image = self.frames[int(self.
124                                     frame_index)]
125
126     def update(self):
127         if self.status == 'available':
128             self.animate()
129         else:
130             tint_surf = self.image.copy()
131             tint_surf.fill('black', None, pygame.
132                             BLEND_RGBA_MULT)
133             self.image.blit(tint_surf, (0, 0))
134
135 class Icon(pygame.sprite.Sprite):
136     def __init__(self, pos):
137         super().__init__()
138         self.pos = pos
139         self.image = pygame.image.load('../gamefin
140                                     /graphics/overworld/hat.png').
141             convert_alpha()
142         self.rect = self.image.get_rect(center =
143                                         pos)
144
145     def update(self):
146         self.rect.center = self.pos
147
148 class Overworld:
149     def __init__(self, start_level, max_level,
150                 surface, create_level):
151
152         # setup
153         self.display_surface = surface
154         self.max_level = max_level
155         self.current_level = start_level
156         self.create_level = create_level
157
158         # movement logic
159         self.moving = False
160         self.move_direction = pygame.math.Vector2

```

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


```

(0,0)
56 self.speed = 8
57
58 # sprites
59 self.setup_nodes()
60 self.setup_icon()
61 self.sky = Sky(8, 'overworld')
62
63 # time
64 self.start_time = pygame.time.get_ticks()
65 self.allow_input = False
66 self.timer_length = 300
67
68 def setup_nodes(self):
69     self.nodes = pygame.sprite.Group()
70
71     for index, node_data in enumerate(levels.
72         values()):
73         if index <= self.max_level:
74             node_sprite = Node(node_data['node_pos
75                 '], 'available', self.speed,
76                 node_data['node_graphics'])
77         else:
78             node_sprite = Node(node_data['node_pos
79                 '], 'locked', self.speed, node_data['
80                 node_graphics'])
81         self.nodes.add(node_sprite)
82
83 def draw_paths(self):
84     if self.max_level > 0:
85         points = [node['node_pos'] for index,
86             node in enumerate(levels.values())
87             if index <= self.max_level]
88         pygame.draw.lines(self.display_surface, '#a04f45', False, points, 6)
89
90 def setup_icon(self):
91     self.icon = pygame.sprite.GroupSingle()
92     icon_sprite = Icon(self.nodes.sprites()[
93         self.current_level].rect.center)
94     self.icon.add(icon_sprite)
95
96 def input(self):
97     keys = pygame.key.get_pressed()
98
99     if not self.moving and self.allow_input:
100         if (keys[pygame.K_RIGHT] or keys[pygame.
101             K_d]) and self.current_level < self.
102             max_level:
103             self.move_direction = self.
104                 get_movement_data('next')
105             self.current_level += 1
106             self.moving = True
107         elif (keys[pygame.K_LEFT] or keys[pygame.
108             K_a]) and self.current_level > 0:
109             self.move_direction = self.
110                 get_movement_data('previous')
111             self.current_level -= 1
112             self.moving = True
113         elif keys[pygame.K_SPACE] or keys[pygame.
114             K_UP]:
115             self.create_level(self.current_level)
116         elif keys[pygame.K_DOWN]:
117             exit(0)
118
119 def get_movement_data(self, target):
120     start = pygame.math.Vector2(self.nodes.
121         sprites()[self.current_level].rect.
122         center)
123
124     if target == 'next':
125         end = pygame.math.Vector2(self.nodes.
126             sprites()[self.current_level + 1].
127             rect.center)
128     else:
129         end = pygame.math.Vector2(self.nodes.
130             sprites()[self.current_level - 1].
131             rect.center)
132
133     return (end - start).normalize()
134
135 def update_icon_pos(self):
136     if self.moving and self.move_direction:
137         self.icon.sprite.pos += self.
138             move_direction * self.speed
139         target_node = self.nodes.sprites()[self.
140             current_level]
141         if target_node.detection_zone.
142             collidepoint(self.icon.sprite.pos):
143             self.moving = False
144             self.move_direction = pygame.math.
145                 Vector2(0,0)
146
147 def input_timer(self):
148     if not self.allow_input:
149         current_time = pygame.time.get_ticks()
150         if current_time - self.start_time >=
151             self.timer_length:
152             self.allow_input = True
153
154 def run(self):
155     self.input_timer()
156     self.input()
157     self.update_icon_pos()
158     self.icon.update()
159     self.nodes.update()
160
161     self.sky.draw(self.display_surface)
162     self.draw_paths()
163     self.nodes.draw(self.display_surface)
164     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

```

```

14 # coins
15 self.coin = pygame.image.load('../gamefin/
16 graphics/ui/coin.png').convert_alpha()
17 self.coin_rect = self.coin.get_rect(
18 topleft = (50, 61))
19 self.font = pygame.font.Font('../gamefin/
20 graphics/ui/ARCADEPTI.ttf', 30)
21
22 def show_health(self, current, full):
23     self.display_surface.blit(self.health_bar
24     , (20, 10))
25     current_health_ratio = current / full
26     current_bar_width = self.bar_max_width *
27     current_health_ratio
28     health_bar_rect = pygame.Rect(self.
29     health_bar_topleft, (current_bar_width,
30     self.bar_height))
31     pygame.draw.rect(self.display_surface, '#
32     dc4949', health_bar_rect)
33
34 def show_coins(self, amount):
35     self.display_surface.blit(self.coin, self.
36     coin_rect)
37     coin_amount_surf = self.font.render(str(
38     amount), False, '#33323d')
39     coin_amount_rect = coin_amount_surf.
40     get_rect(midleft = (self.coin_rect.
41     right + 4, self.coin_rect.centery))
42     self.display_surface.blit(coin_amount_surf
43     , coin_amount_rect)
44
45

```

Código 10. ui.py

II-K. level.py

Para ir finalizando con todo, se hara el nivel, se utilizaran la mayoría 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

```

1 import pygame
2 from support import import_csv_layout,
3 import_cut_graphics
4 from settings import tile_size, screen_height,
5 screen_width
6 from tiles import Tile, StaticTile, Crate,
7 Coin, Palm
8 from enemy import Enemy
9 from decoration import Sky, Water, Clouds
10 from player import Player
11 from particles import ParticleEffect
12 from game_data import levels
13
14 class Level:
15     def __init__(self, current_level, surface,
16     create_overworld, change_coins,
17     change_health):
18         # general setup
19         self.display_surface = surface
20         self.world_shift = 0
21         self.current_x = None
22
23         # audio
24         self.coin_sound = pygame.mixer.Sound('../
25         gamefin/audio/effects/coin.wav')
26
27

```

```

28 self.stomp_sound = pygame.mixer.Sound('../
29 gamefin/audio/effects/stomp.wav')
30
31 # overworld connection
32 self.create_overworld = create_overworld
33 self.current_level = current_level
34 level_data = levels[self.current_level]
35 self.new_max_level = level_data['unlock']
36
37 # player
38 player_layout = import_csv_layout(
39     level_data['player'])
40 self.player = pygame.sprite.GroupSingle()
41 self.goal = pygame.sprite.GroupSingle()
42 self.player_setup(player_layout,
43     change_health)
44
45 # user interface
46 self.change_coins = change_coins
47
48 # dust
49 self.dust_sprite = pygame.sprite.
50     GroupSingle()
51 self.player_on_ground = False
52
53 # explosion particles
54 self.explosion_sprites = pygame.sprite.
55     Group()
56
57 # terrain setup
58 terrain_layout = import_csv_layout(
59     level_data['terrain'])
60 self.terrain_sprites = self.
61     create_tile_group(terrain_layout, '
62     terrain')
63
64 # grass setup
65 grass_layout = import_csv_layout(
66     level_data['grass'])
67 self.grass_sprites = self.
68     create_tile_group(grass_layout, 'grass'
69     )
70
71 # crates
72 crate_layout = import_csv_layout(
73     level_data['crates'])
74 self.crate_sprites = self.
75     create_tile_group(crate_layout, 'crates'
76     )
77
78 # coins
79 coin_layout = import_csv_layout(level_data
80     ['coins'])
81 self.coin_sprites = self.create_tile_group
82     (coin_layout, 'coins')
83
84 # foreground palms
85 fg_palm_layout = import_csv_layout(
86     level_data['fg palms'])
87 self.fg_palm_sprites = self.
88     create_tile_group(fg_palm_layout, 'fg
89     palms')
90
91 # background palms
92 bg_palm_layout = import_csv_layout(
93     level_data['bg palms'])
94 self.bg_palm_sprites = self.
95

```



```

        create_tile_group(bg_palm_layout, 'bg
        palms')
67
68 # enemy
69 enemy_layout = import_csv_layout(
70     level_data['enemies'])
71 self.enemy_sprites = self.
72     create_tile_group(enemy_layout, '
73     enemies')
74
75 # constraint
76 constraint_layout = import_csv_layout(
77     level_data['constraints'])
78 self.constraint_sprites = self.
79     create_tile_group(constraint_layout, '
80     constraint')
81
82 # decoration
83 self.sky = Sky(8)
84 level_width = len(terrain_layout[0]) *
85     tile_size
86 self.water = Water(screen_height - 20,
87     level_width)
88 self.clouds = Clouds(400, level_width, 30)
89
90 def create_tile_group(self, layout, type):
91     sprite_group = pygame.sprite.Group()
92
93     for row_index, row in enumerate(layout):
94         for col_index, val in enumerate(row):
95             if val != '-1':
96                 x = col_index * tile_size
97                 y = row_index * tile_size
98
99                 if type == 'terrain':
100                     terrain_tile_list =
101                         import_cut_graphics('../
102                         gamefin/graphics/terrain/
103                         terrain_tiles.png')
104                     tile_surface = terrain_tile_list[
105                         int(val)]
106                     sprite = StaticTile(tile_size, x, y,
107                         tile_surface)
108
109                 if type == 'grass':
110                     grass_tile_list =
111                         import_cut_graphics('../
112                         gamefin/graphics/decoration/
113                         grass/grass.png')
114                     tile_surface = grass_tile_list[int(
115                         val)]
116                     sprite = StaticTile(tile_size, x, y,
117                         tile_surface)
118
119                 if type == 'crates':
120                     sprite = Crate(tile_size, x, y)
121
122                 if type == 'coins':
123                     if val == '0': sprite = Coin(
124                         tile_size, x, y, '../gamefin/
125                         graphics/coins/gold', 5)
126                     if val == '1': sprite = Coin(
127                         tile_size, x, y, '../gamefin/
128                         graphics/coins/silver', 1)
129
130                 if type == 'fg palms':
131                     if val == '0': sprite = Palm(
132                         tile_size, x, y, '../gamefin/
133                         graphics/terrain/palm_small'
134                         , 38)
135                     if val == '1': sprite = Palm(
136                         tile_size, x, y, '../gamefin/
137                         graphics/terrain/palm_large'
138                         , 64)
139
140                     if type == 'bg palms':
141                         sprite = Palm(tile_size, x, y, '../
142                         gamefin/graphics/terrain/
143                         palm_bg', 64)
144
145                     if type == 'enemies':
146                         sprite = Enemy(tile_size, x, y)
147
148                     if type == 'constraint':
149                         sprite = Tile(tile_size, x, y)
150
151                     sprite_group.add(sprite)
152
153     return sprite_group
154
155 def player_setup(self, layout, change_health):
156     for row_index, row in enumerate(layout):
157         for col_index, val in enumerate(row):
158             x = col_index * tile_size
159             y = row_index * tile_size
160             if val == '0':
161                 sprite = Player((x, y), self.
162                     display_surface, self.
163                     create_jump_particles,
164                     change_health)
165                 self.player.add(sprite)
166             if val == '1':
167                 hat_surface = pygame.image.load('../
168                 gamefin/graphics/character/hat.
169                 png').convert_alpha()
170                 sprite = StaticTile(tile_size, x, y,
171                     hat_surface)
172                 self.goal.add(sprite)
173
174 def enemy_collision_reverse(self):
175     for enemy in self.enemy_sprites.sprites():
176         if pygame.sprite.spritecollide(enemy,
177             self.constraint_sprites, False):
178             enemy.reverse()
179
180 def create_jump_particles(self, pos):
181     if self.player.sprite.facing_right:
182         pos -= pygame.math.Vector2(10, 5)
183     else:
184         pos += pygame.math.Vector2(10, -5)
185     jump_particle_sprite = ParticleEffect(pos,
186         'jump')
187     self.dust_sprite.add(jump_particle_sprite)
188
189 def horizontal_movement_collision(self):
190     player = self.player.sprite
191     player.collision_rect.x += player.
192         direction.x * player.speed
193     collidable_sprites = self.terrain_sprites.
194         sprites() + self.crate_sprites.sprites
195         () + self.fg_palm_sprites.sprites()
196     for sprite in collidable_sprites:
197         if sprite.rect.colliderect(player.
198             collision_rect):

```

```

157         if player.direction.x < 0:
158             player.collision_rect.left = sprite.
159                 rect.right
160             player.on_left = True
161             self.current_x = player.rect.left
162         elif player.direction.x > 0:
163             player.collision_rect.right = sprite
164                 .rect.left
165             player.on_right = True
166             self.current_x = player.rect.right
167
168     def vertical_movement_collision(self):
169         player = self.player.sprite
170         player.apply_gravity()
171         collidable_sprites = self.terrain_sprites.
172             sprites() + self.crate_sprites.sprites
173             () + self.fg_palm_sprites.sprites()
174
175         for sprite in collidable_sprites:
176             if sprite.rect.colliderect(player.
177                 collision_rect):
178                 if player.direction.y > 0:
179                     player.collision_rect.bottom =
180                         sprite.rect.top
181                     player.direction.y = 0
182                     player.on_ground = True
183                 elif player.direction.y < 0:
184                     player.collision_rect.top = sprite.
185                         rect.bottom
186                     player.direction.y = 0
187                     player.on_ceiling = True
188
189             if player.on_ground and player.direction.y
190                 < 0 or player.direction.y > 1:
191                 player.on_ground = False
192
193     def scroll_x(self):
194         player = self.player.sprite
195         player_x = player.rect.centerx
196         direction_x = player.direction.x
197
198         if player_x < screen_width / 4 and
199             direction_x < 0:
200             self.world_shift = 8
201             player.speed = 0
202         elif player_x > screen_width - (
203             screen_width / 4) and direction_x > 0:
204             self.world_shift = -8
205             player.speed = 0
206         else:
207             self.world_shift = 0
208             player.speed = 8
209
210     def get_player_on_ground(self):
211         if self.player.sprite.on_ground:
212             self.player_on_ground = True
213         else:
214             self.player_on_ground = False
215
216     def create_landing_dust(self):
217         if not self.player_on_ground and self.
218             player.sprite.on_ground and not self.
219             dust_sprite.sprites():
220             if self.player.sprite.facing_right:
221                 offset = pygame.math.Vector2(10,15)
222             else:
223                 offset = pygame.math.Vector2(-10,15)
224
225         fall_dust_particle = ParticleEffect(self
226             .player.sprite.rect.midbottom -
227             offset, 'land')
228         self.dust_sprite.add(fall_dust_particle)
229
230     def check_death(self):
231         if self.player.sprite.rect.top >
232             screen_height:
233             self.create_overworld(self.current_level
234                 , 0)
235
236     def check_win(self):
237         if pygame.sprite.spritecollide(self.player
238             .sprite, self.goal, False):
239             self.create_overworld(self.current_level
240                 , self.new_max_level)
241
242     def check_coin_collisions(self):
243         collided_coins = pygame.sprite.
244             spritecollide(self.player.sprite, self.
245                 coin_sprites, True)
246         if collided_coins:
247             self.coin_sound.play()
248             for coin in collided_coins:
249                 self.change_coins(coin.value)
250
251     def check_enemy_collisions(self):
252         enemy_collisions = pygame.sprite.
253             spritecollide(self.player.sprite, self.
254                 enemy_sprites, False)
255
256         if enemy_collisions:
257             for enemy in enemy_collisions:
258                 enemy_center = enemy.rect.centery
259                 enemy_top = enemy.rect.top
260                 player_bottom = self.player.sprite.
261                     rect.bottom
262                 if enemy_top < player_bottom <
263                     enemy_center and self.player.
264                     sprite.direction.y >= 0:
265                     self.stomp_sound.play()
266                     self.player.sprite.direction.y = -15
267                     explosion_sprite = ParticleEffect(
268                         enemy.rect.center, 'explosion')
269                     self.explosion_sprites.add(
270                         explosion_sprite)
271                     enemy.kill()
272                 else:
273                     self.player.sprite.get_damage()
274
275     def run(self):
276         # run the entire game / level
277
278         # sky
279         self.sky.draw(self.display_surface)
280         self.clouds.draw(self.display_surface, self
281             .world_shift)
282
283         # background palms
284         self.bg_palm_sprites.update(self.
285             world_shift)
286         self.bg_palm_sprites.draw(self.
287             display_surface)
288
289         # dust particles
290         self.dust_sprite.update(self.world_shift)
291         self.dust_sprite.draw(self.display_surface

```

```

    )
261
262 # terrain
263 self.terrain_sprites.update(self.
    world_shift)
264 self.terrain_sprites.draw(self.
    display_surface)
265
266 # enemy
267 self.enemy_sprites.update(self.world_shift
    )
268 self.constraint_sprites.update(self.
    world_shift)
269 self.enemy_collision_reverse()
270 self.enemy_sprites.draw(self.
    display_surface)
271 self.explosion_sprites.update(self.
    world_shift)
272 self.explosion_sprites.draw(self.
    display_surface)
273
274 # crate
275 self.crate_sprites.update(self.world_shift
    )
276 self.crate_sprites.draw(self.
    display_surface)
277
278 # grass
279 self.grass_sprites.update(self.world_shift
    )
280 self.grass_sprites.draw(self.
    display_surface)
281
282 # coins
283 self.coin_sprites.update(self.world_shift)
284 self.coin_sprites.draw(self.
    display_surface)
285
286 # foreground palms
287 self.fg_palm_sprites.update(self.
    world_shift)
288 self.fg_palm_sprites.draw(self.
    display_surface)
289
290 # player sprites
291 self.player.update()
292 self.horizontal_movement_collision()
293
294 self.get_player_on_ground()
295 self.vertical_movement_collision()
296 self.create_landing_dust()
297
298 self.scroll_x()
299 self.player.draw(self.display_surface)
300 self.goal.update(self.world_shift)
301 self.goal.draw(self.display_surface)
302
303 self.check_death()
304 self.check_win()
305
306 self.check_coin_collisions()
307 self.check_enemy_collisions()
308
309 # water
310 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

```

1 import pygame, sys
2 from settings import *
3 from level import Level
4 from overworld import Overworld
5 from ui import UI
6
7 class Game:
8     def __init__(self):
9
10         # game attributes
11         self.max_level = 0
12         self.max_health = 100
13         self.cur_health = 100
14         self.coins = 0
15
16         # audio
17         self.level_bg_music = pygame.mixer.Sound('
18             ../gamefin/audio/level_music.wav')
19         self.overworld_bg_music = pygame.mixer.
20             Sound('../gamefin/audio/
21                 overworld_music.wav')
22
23         # overworld creation
24         self.overworld = Overworld(0, self.
25             max_level, screen, self.create_level)
26         self.status = 'overworld'
27         self.overworld_bg_music.play(loops = -1)
28
29         # user interface
30         self.ui = UI(screen)
31
32     def create_level(self, current_level):
33         self.level = Level(current_level, screen,
34             self.create_overworld, self.
35             change_coins, self.change_health)
36         self.status = 'level'
37         self.overworld_bg_music.stop()
38         self.level_bg_music.play(loops = -1)
39
40     def create_overworld(self, current_level,
41         new_max_level):
42         if new_max_level > self.max_level:
43             self.max_level = new_max_level
44         self.overworld = Overworld(current_level,
45             self.max_level, screen, self.
46             create_level)
47         self.status = 'overworld'
48         self.overworld_bg_music.play(loops = -1)
49         self.level_bg_music.stop()
50
51     def change_coins(self, amount):
52         self.coins += amount
53
54     def change_health(self, amount):
55         self.cur_health += amount
56
57     def check_game_over(self):
58         if self.cur_health <= 0:
59             self.cur_health = 100

```

```

51     self.coins = 0
52     self.max_level = 0
53     self.overworld = Overworld(0, self.
54         max_level, screen, self.create_level)
55     self.status = 'overworld'
56     self.level_bg_music.stop()
57     self.overworld_bg_music.play(loops = -1)
58
59     def run(self):
60         if self.status == 'overworld':
61             self.overworld.run()
62         else:
63             self.level.run()
64             self.ui.show_health(self.cur_health, self
65                 .max_health)
66             self.ui.show_coins(self.coins)
67             self.check_game_over()
68
69 # Pygame setup
70 pygame.init()
71 screen = pygame.display.set_mode((screen_width
72     , screen_height))
73 clock = pygame.time.Clock()
74 game = Game()
75
76 while True:
77     for event in pygame.event.get():
78         if event.type == pygame.QUIT:
79             pygame.quit()
80             sys.exit()
81
82     screen.fill('grey')
83     game.run()
84
85     pygame.display.update()
86     clock.tick(60)

```

Código 12. main.py

II-M. arduino.py

Aquí viene el código para poder ejecutar el archivo de arduino y poder utilizar el joystick shield en el juego

```

1  import time
2  import serial
3  import pyautogui
4  import os
5
6  # create a translation table for how keys are
7  # interpreted
8  KEY_MAPPING = {
9      'A' : '',
10     'B' : 'up',
11     'C' : '',
12     'D' : '',
13     'E' : 'e',
14     'F' : 'down',
15     'JOY_ENTER' : '',
16     'JOY_LEFT' : 'left',
17     'JOY_RIGHT' : 'right',
18     'JOY_UP' : '',
19     'JOY_DOWN' : ''
20 }
21
22 def handle_line(line):
23     "Handles a single line of EVENT sent over
24     serial"

```

```

23     key, arg = [x.decode("utf-8") for x in
24         line.split()]
25
26     if key not in KEY_MAPPING:
27         raise NotImplementedError("key %s
28             unrecognized, no mapping" % key)
29
30     if arg == 'UP':
31         pyautogui.keyUp(KEY_MAPPING[key])
32     elif arg == 'DOWN':
33         pyautogui.keyDown(KEY_MAPPING[key])
34     else:
35         raise NotImplementedError("Unknown
36             event %s" % arg)
37
38     # Verificar si la tecla es 'F' para salir
39     # del bucle
40     if key == 'F':
41         print("Tecla F detectada. Saliendo del
42             programa.")
43         os._exit(0)
44
45     def handle_com(path):
46         "Opens the serial socket and reads all
47         events interpreting them"
48         with serial.Serial(path, 9600) as arduino:
49             while True:
50                 line = arduino.readline()
51                 handle_line(line)
52
53     if __name__ == '__main__':
54         while True:
55             try:
56                 print("Connecting to device")
57                 handle_com('COM3')
58             except serial.Serialutil.
59                 SerialException as se:
60                 print("Error with connection, will
61                     attempt to connect again in
62                     10 secs...")
63                 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

```

1  import subprocess
2
3  def ejecutar_archivo(archivo):
4      subprocess.run(["python", archivo])
5
6  if __name__ == "__main__":
7      archivo1 = "../gamefin/code/arduino.py"
8      archivo2 = "../gamefin/code/main.py"
9
10     proceso1 = subprocess.Popen(["python",
11         archivo1])
12     proceso2 = subprocess.Popen(["python",
13         archivo2])
14
15     proceso1.wait()
16     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

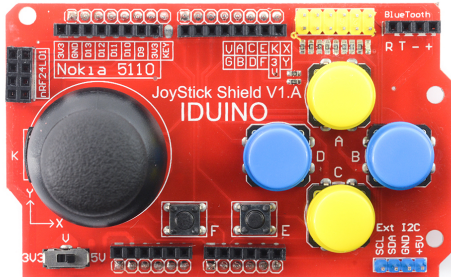


Figura 1. JoyStick Shield V1.a.

El juego se ve de la siguiente manera, empezando por el overworld, como se muestra en la figura 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 recorrerlo, coleccionar monedas o matar enemigos

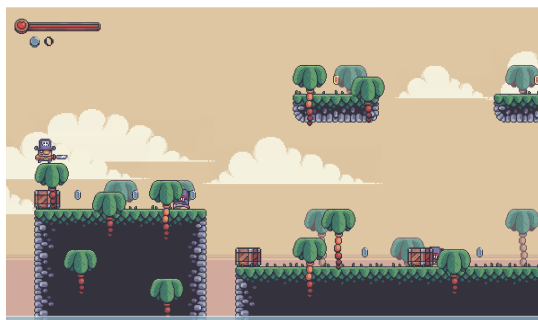


Figura 3. nivel 1.

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

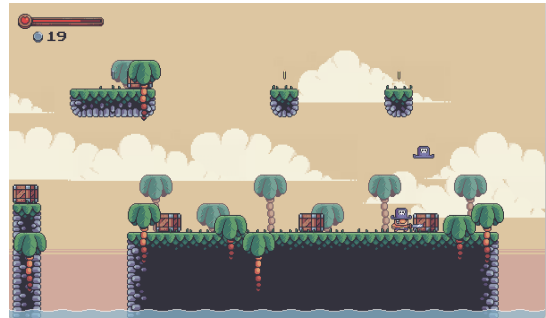


Figura 4. nivel 1.

si conseguimos el sombrero 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 código que se complicaron y otras que estuvieron rápido, pero aun así es muy interesante desarrollar en python. Volver a desarrollar apps en python después de dejarlo un tiempo fue interesante

REFERENCIAS

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