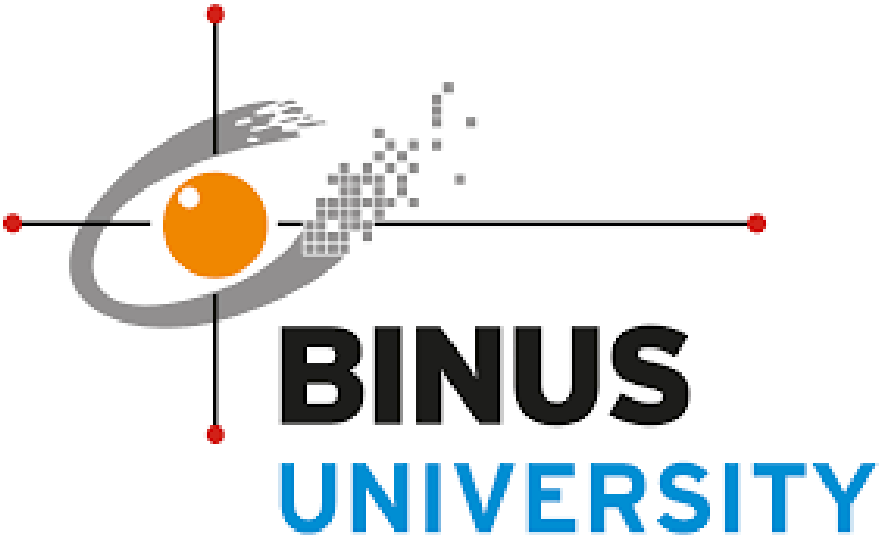
2D Game Development

Dino Level-Up

2D Game Development Final Project Report



Lecturer:

Jude Joseph Lamug Martinez, MCS

Student Name:

Aliansky Winston Viando

Student ID:

2902708224

Class L1AC

Computer Science - International Class

Semester I - Algorithm & Programming

COMP6047001

A. Background

The involvement of game industries has reached to its point whereas realistic graphic technology dominate the game market. However, the endless runner game genre is not being played anymore by the young generations because of its simple mechanics, accessibility, and highly addictive nature. One of the biggest examples is the Chrome Dino Game, which serves as a basic survival offline game.

While the classic game concept is still effective, but it often lacks in terms of variety and specialty. This project called “Dino Level-Up” was developed because of a purpose to evolve the traditional way by adding strategic elements and modern programming techniques. The game was designed to improve the user experience through several key of innovations:

- Dynamic Power Up System: Not like the original version of dino game, “Dino Level-Up” involves items such as game Shield, Jump Booster, and Double Points, forcing players to make decisions on the game.
- Scaling Difficulty: The game uses an acceleration to its game speed through the code algorithm where the speed increases, and obstacle spawn speed is calculated based on its live game speed.
- Modular Architecture: From a technical coding perspective, the game was built using Object Oriented Programming or usually called as OOP principles in Python programming language. By separating the game logic into the configuration, sprite, and the main logic of the game. Leaving the project code a professional approach to software scalability and clean to read code.

In conclusion, this 2D game project acts as a practical application of the Pygame external libraries that focus on real time event management, sprite collision detection, and persistent data management (Game Score Tracking). The goal of this project is to transform a simple 2D Dino Game into a fresher and rich features 2D Dino game.

B. Full Code Explanation

The game code is seperated into 3 different files: main.py, config.py, and, sprites.py

- **Main.py (Game logic and loop)**

```python
import pygame
import sys
import random
import config
from sprites import Cloud, Dino, Cactus, Ptero, Powerup

# groups (bundle draw, update, and collision for multiple sprites)
cloud_group = pygame.sprite.Group()
obstacle_group = pygame.sprite.Group()
dino_group = pygame.sprite.GroupSingle()
powerup_group = pygame.sprite.Group()
```

The first 5 lines of this code imports pygame for its main library to make the game including the window, event, sprite, collision, etc. The "import sys" used for system level operations like when exiting the game, and "random" use to ensure that obstacle spawning is unpredictable and randomized. The "import config" imports a local file called config.py, for global settings and classes like (Cloud, Dino, etc.) from sprites.py file. This is one of examples of a Modular Software Architecture, keeping the code organized and easy to read. The code lines between 7-11 groups multiple objects at once. Instead of updating the sprites one by one, the objects inside this game are bundled into several groups like cloud, obstacle, dino, and powerup group. It allows the program to just call .update() and .draw() on the group within just a single line of code, this increase the code efficiency rather than using standard Python lists.

```python
# objects
dinosaur = Dino(50, 360) #create dinosaur object at position (50, 360)
dino_group.add(dinosaur) #add dinosaur to dino group

config.all_sfx.play(loops=-1) #play background music in loop
```

The above lines of code create the dino sprite object position in the window at x = 50 and y = 360. The pygame x and y (0, 0) coordinates start at the top left of the window. When the Y value becomes larger than its position on the

window become lower and when the X value becomes larger the position will move to the right. The .add() function will add dinosaur to the dino group which we have make earlier. Line 17th will play background music in a loop so it will always play the background music until the game was over.

```
19  # functions
20  def end_game():
21      game_over_text = config.game_font.render("Game Over!",
        True, "black") #render game over text
22      game_over_rect = game_over_text.get_rect(center=(640, 300))
        #center rect for game over text
23      score_text = config.game_font.render(f"Score: {int(config.
        player_score)}", True, "black") #render score text
24      score_rect = score_text.get_rect(center=(640, 340)) #center
        rect for score text
25      config.screen.blit(game_over_text, game_over_rect) #draw
        game over text on screen
26      config.screen.blit(score_text, score_rect) #draw score text
        on screen
27      high_text = config.game_font.render(f"High Score: {config.
        high_score}", True, "black") #render high score text
28      high_rect = high_text.get_rect(center=(640, 380)) #center
        rect for high score text
29      config.screen.blit(high_text, high_rect) #draw high score
        text on screen
```

The code above defines a function called “end_game()” that was primarily purpose to render the game over text, game score text, high score text and its own position. Also draw all texts into the game window.

```
32  def reset_powerups():
33      config.shield_active = False
34      config.shield_end_time = 0
35      config.jump_boost_active = False
36      config.jump_boost_end_time = 0
37      config.double_points_active = False
38      config.double_points_end_time = 0
39
40  def draw_menu():
41      config.screen.blit(config.background_img, (0, 0)) #draw
        background image
42
43      # title
44      title_surf = config.title_font.render("DINO LEVEL-UP",
        True, (255, 255, 255)) #render title text
45      title_rect = title_surf.get_rect(center=(640, 250))
46      config.screen.blit(title_surf, title_rect)
```

The reset powerups function is acts as a state manager that reset all active powerups when a game session is restart or start. By setting values into False and 0. This code make sure that players do not carry over things from previous game. The draw menu function address the visual rendering of the game interface like background image and the title using the Pygame library. By using get_rect(center=(640, 250)), the code compute and centers the title at specific

coordinate. This function essentially prepares the window frame, background image and font.

```
48      # start button
49      pygame.draw.rect(config.screen, (200, 0, 0), config.
        start_btn_rect, border_radius=10)
50      start_text = config.start_font.render("START", True, (255,
        255, 255))
51      start_rect = start_text.get_rect(center=config.
        start_btn_rect.center)
52      config.screen.blit(start_text, start_rect)
```

The code snippet above renders the functional start button by layering a text label over a shape by using the Pygame library. It initiates by drawing a dark red rectangle with rounded corners to the screen, that acts as a visual background for the button. The code generates a white colour start text. Then, use the get_rect method to align to the center. Finally, the blit function draws the centered text on the top of the start button.

```
55  def draw_shield_aura(sprite):
56      # draw semi transparent circle around the dino to show
        shield exist
57      aura_surf = pygame.Surface((sprite.rect.width*2, sprite.
        rect.height*2), pygame.SRCALPHA) #create transparent surface
58      pygame.draw.circle(aura_surf, (0, 160, 255, 100),
        (aura_surf.get_width()//2, aura_surf.get_height()//2), max
        (sprite.rect.width, sprite.rect.height)) #draw
        semi-transparent blue circle
59      config.screen.blit(aura_surf, (sprite.rect.centerx -
        aura_surf.get_width()//2, sprite.rect.centery - aura_surf.
        get_height()//2)) #blit aura centered on sprite blit is
        like overlay
```

The code above generates a function to draw shield powerup visual elements by creating a transparent surface first then create a semi transparent blue circle on top of the dino sprite and using blit for centered on the dino sprite.

```
62  # main Loop
63  while True:
64      keys = pygame.key.get_pressed() #get current all key presses
65      if keys[pygame.K_DOWN]: #if down key pressed
66          dinosaur.duck() #call duck method
67      else:
68          if dinosaur.ducking: #if currently ducking
69              dinosaur.unduck() #call unduck method
70
71      for event in pygame.event.get():
72          if event.type == pygame.MOUSEBUTTONDOWN and config.
            game_state == "menu": #mouse click event in menu state
73              if config.start_btn_rect.collidepoint(event.pos):
                #if click is within start button rect
74                  config.game_state = "playing" #change game
                    state to playing
75
76          if event.type == pygame.QUIT: #quit event
77              pygame.quit() #quit pygame
78              sys.exit() #exit program
```

The code snippet above make the reaction to the game when an actions is being made. It starts by looking at the keyboard and check if the down arrow is being pressed then the dino sprite will duck and when it is released the dino sprite will unduck and back to normal. Next, it check for mouse clicks if the game is on the menu screen and user click start button then the game state will switch to playing state. Finally it checks if the player clicks the X mark to close the game window it will safely shut down the game and exit.

```
80          if event.type == config.CLOUD_EVENT: #cloud spawn event
81              current_cloud_y = random.randint(50, 300) #random y
                position for cloud
82              current_cloud = Cloud(config.cloud_img, 1380,
                current_cloud_y) #create cloud object Cloud(image,
                x_pos, y_pos)
83              cloud_group.add(current_cloud) #add cloud to cloud
                group
84
85          if event.type == pygame.KEYDOWN: #key down event
86              # ESC quit
87              if event.key == pygame.K_ESCAPE: #if escape key
                pressed
88                  pygame.quit()
89                  sys.exit()
```

The first block lines 80-83 adding visual variety to the game sky. When the cloud event occurs the code will picks a random height and chooses y axis randomly between 50 until 300. It also generates a new cloud object at 1380 position. Adds this new cloud into the cloud group so the game draws and move it on the screen. The second block line 85-89 provides a fast way for user to exit the game, it receive all the keydown event and immediately shut down Pygame and close application when escape key is pressed.

```
91          # Jump + Restart logic
92          if event.key == pygame.K_SPACE or event.key ==
            pygame.K_UP: #jump keys can with space or up arrow
93              if config.game_state == "playing": # Only allow
                jump when playing
94                  dinosaur.jump()
95
96                  # Restart game when pressing jump after
                    game over
97                  if config.game_over:
98                      config.game_over = False #from game
                        over to playing
99                      config.game_speed = 3 #reset game speed
100                     config.player_score = 0 #reset player
                        score
101                     obstacle_group.empty() #clear obstacles
102                     cloud_group.empty() #clear clouds
103                     powerup_group.empty() #clear powerups
104                     reset_powerups() #reset powerup states
105                     config.all_sfx.play(loops=-1)  #
                        restart backsound
```

The code check for spacebar or up arrow keys when being pressed, during the gameplay pressing those keys will make the dino jump. But after the game is over these keys acts a restart button, by resets the score to 0 and returns the game speed to normal, clear all obstacles and clouds, and restart the background music.

```
107 if config.game_state == "menu": #in menu state
108     draw_menu() #draw menu screen
109
110 elif config.game_state == "playing": #in playing state
111     config.screen.blit(config.background_img, (0, 0)) #draw
        background image
112
113     # collision with obstacles (consider shield)
114     hit = pygame.sprite.spritecollide(dino_group.sprite,
        obstacle_group, False) #check collision between dino
        and obstacles (cactus/ptero) and obstacles not removed
        on hit
115     if hit: #check if there was a collision
116         if not config.game_over: #only process if it is not
            in game over state
117             if config.shield_active: #check if shield
                powerup is active
118                 # if shield active, destroy the obstacle
                    but do not end game
119                 for o in hit: #for each collided obstacle
                    (o means the shield)
```

The next block of code first checks the game state, if the user in on the menu, it calls the draw menu function to draw the menu screen. If the user is playing it draws background image onto the screen. While playing, the code always checks for the collision or a hit between the dino and obstacles. If a hit is detected, the code will checks if the player has a shield power up active or not, if it is on then the shield will destroy the obstacles and keep the game running.

```
121                 else:
122                     config.game_over = True #set game over
                        state to true
123                     config.death_sfx.play() #play death sound
                        effect
124                     config.all_sfx.stop() #stop background music
125                     # HIGH SCORE UPDATE
126                     if config.player_score > config.high_score:
                        #if current score is greater than high score
127                         config.high_score = int(config.
                            player_score) #convert current player
                            score to int and set as new high score
128                         with open("highscore.txt", "w") as f:
                            #open high score file in write mode
129                             f.write(str(config.high_score))
                                #write new high score to file as
                                string
```

The code above manage what reactions will occur when the user hits an obstacle without shield. It changes the game over state into true and plays a death sound effect and stop the background music. The code also check and compares the current score with the existing high score, if the score is higher, it updates the high score and save the new record in file called highscore.txt.

```
131     # collision with other powerups except shield
132     collected = pygame.sprite.spritecollide(dino_group.
        sprite, powerup_group, True) #check collision between
        dino and powerups, remove powerup on collection
133
134     for pu in collected: # for each unknown collected
        powerup
135         # activate powerup
136         now = pygame.time.get_ticks() #current time in
            milliseconds
137         # all powerups last for 3 seconds (3000ms)
138         POWERUP_DURATION = 3000
139         if pu.kind == "shield":
140             config.shield_active = True
141             config.shield_end_time = now + POWERUP_DURATION
142         elif pu.kind == "jump":
143             config.jump_boost_active = True
144             config.jump_boost_end_time = now +
                POWERUP_DURATION
145         elif pu.kind == "double":
146             config.double_points_active = True
```

The code manage when the dino collides with powerups items in game. First it checks for collisions between the dino and any sprites in the powerup group and removes power up that collected. Each collected powerup then process in a loop and has a fixed duration of 3 seconds or 3000 milliseconds for all power ups. Depending of the power up type, the power up feature is activate by set flag in the config.py object. Finally if powerups is being pickup the pickup sound effects will start and play sound.

```
151         # update powerup timers
152         t = pygame.time.get_ticks() #current time in
            milliseconds
153         if config.shield_active and t >= config.
            shield_end_time: #if shield active and current time
            exceeds end time
154             config.shield_active = False #deactivate shield
155         if config.jump_boost_active and t >= config.
            jump_boost_end_time:
156             config.jump_boost_active = False
157         if config.double_points_active and t >= config.
            double_points_end_time:
158             config.double_points_active = False
```

The code starts by grabbing the current game time in milliseconds. For each active powerup it will compare the current time to the end time that was set when the item was pickup. If the powerups is active and the timer has pass the limit then the powerup will set to False results turned off.

```
160         # game over screen
161         if config.game_over:
162             end_game() #call end game function
163         else:
164             # game running logic
165             config.game_speed += 0.001 # slowly accelerate
                (base speed)
166
167             # points sound when hit 100s
168             if round(config.player_score, 1) % 100 == 0 and int
                (config.player_score) > 0: #check if score is
                multiple of 100
169                 config.points_sfx.play() #play points sound
                    effect
```

The code checks if the user lost it will call the end game function. But if the game is still running the code makes it more challenging by slowly increase the game speed by adding 0.001 every frame. The code keeps track the player and play sound effect every time the score hits multiple of 100.

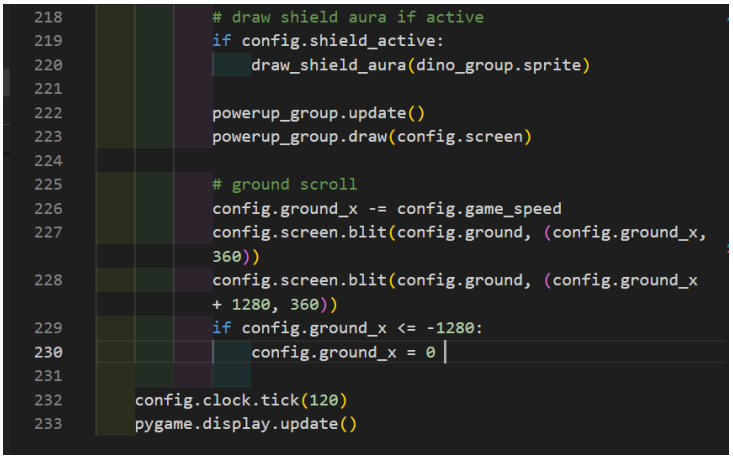
```
171         # --- spawn cooldown logic ---
172         BASE_COOLDOWN = 2000 #base (initial spawn time)
            cooldown in milliseconds
173         MIN_COOLDOWN = 800 #minimum cooldown in milliseconds
174         speed_factor = max(1, config.game_speed / 3)
175         dynamic_cooldown = max(MIN_COOLDOWN,
            BASE_COOLDOWN / speed_factor)
176         spawn_delay = dynamic_cooldown + random.randint(0,
            200)
177
178         # obstacles spawn
179         if pygame.time.get_ticks() - config.obstacle_timer
            >= spawn_delay:
180             obstacle_random = random.randint(1, 70)
181             if obstacle_random in range(1, 7): #1-6 spawn
                cactus
182                 new_obstacle = Cactus(1280, 340)
183                 obstacle_group.add(new_obstacle)
184                 config.obstacle_timer = pygame.time.
                    get_ticks()
185             elif obstacle_random in range(7, 10): #7-9
```

The code controls the dynamic spawning of obstacles and powerups based on the game speed. It starts by defining a base spawn cooldown and a minimum cooldown. The speed factor increase as the game speed increase which reduces rhe cooldown so obstacle will spawn more frequently as the game gets faster. The dynamic cooldown ensure the spawn delay never goes below the minimum limit that is 800 milliseconds. Then the code checks whether enough time has passed since the last spawn using get ticks and obstacle timer. A random number determines what appears, ach spawned object is added to its respective sprite group, and the obstacle timer is reset to track the next spawn window.

```
196         # update score
197         if config.double_points_active:
198             config.player_score += 0.2
199         else:
200             config.player_score += 0.1
201
202         score_surface = config.game_font.render(str(int
            (config.player_score)), True, "black")
203         config.screen.blit(score_surface, (1150, 10))
204
205         high_surface = config.game_font.render(f"HI {config.
            high_score}", True, "black")
206         config.screen.blit(high_surface, (950, 10))
207
208         # Update groups
209         cloud_group.update()
210         cloud_group.draw(config.screen)
211
212         obstacle_group.update()
213         obstacle_group.draw(config.screen)
```

The code updates and display the player score, then refresh and draw all objects each frame. The score increases over the time, but if the double points power up is active, the score will increase 2x faster. The current score is rendered as text using the game font and drawn at the top-right of the screen, while the high

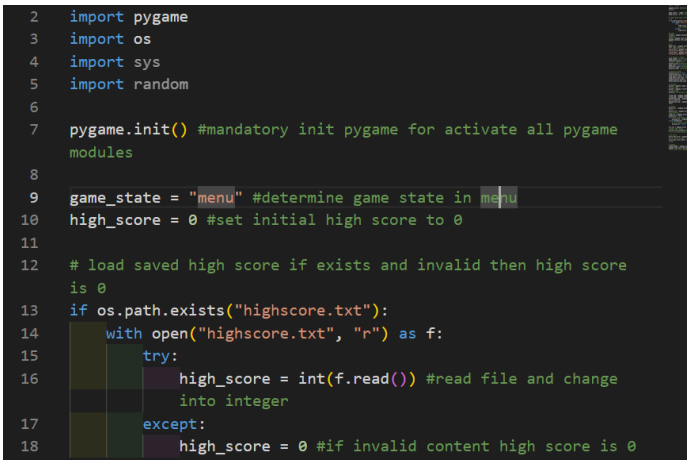
score is rendered separately and displayed next to it. The code then updates and draws each sprite group.



```
        # draw shield aura if active
        if config.shield_active:
            draw_shield_aura(dino_group.sprite)

        powerup_group.update()
        powerup_group.draw(config.screen)

        # ground scroll
        config.ground_x -= config.game_speed
        config.screen.blit(config.ground, (config.ground_x,
        360))
        config.screen.blit(config.ground, (config.ground_x
        + 1280, 360))
        if config.ground_x <= -1280:
            config.ground_x = 0

    config.clock.tick(120)
    pygame.display.update()
```

The code draw shield when it is active, and updates the positions of any powerup items on the screen and draws them for the user to see. The horizontal position of the ground moves to the left based on the game speed. To make ground endless, the code draw 2 ground image side by side and reset it making the ground like one continuous track. The code sets a tick limit of 120 which maintain the game to run smoothly and tells the computer to update the display to show the movements and all game drawing that exist.

- **config.py (Manage all initialization, global settings and assets loading)**



```
import pygame
import os
import sys
import random

pygame.init() #mandatory init pygame for activate all pygame
modules

game_state = "menu" #determine game state in menu
high_score = 0 #set initial high score to 0

# load saved high score if exists and invalid then high score
is 0
if os.path.exists("highscore.txt"):
    with open("highscore.txt", "r") as f:
        try:
            high_score = int(f.read()) #read file and change
            into integer
        except:
            high_score = 0 #if invalid content high score is 0
```

The first 4 lines of code will import pygame library, operating system, system and random. It sets the initial game state to menu so the player sees the

start screen first. Sets the initial high score into 0. It also read if highscore.txt file is exist but if it is invalid and cannot be found then high score will set into 0.

```
20  #screen
21  screen = pygame.display.set_mode((1280, 720)) #window size
    1280x720
22  clock = pygame.time.Clock() #FPS control
23  pygame.display.set_caption("Dino LevelUp - Powerups") #window
    title
24
25  #font
26  game_font = pygame.font.Font("Assets/Other/PressStart2P-Regular.
    ttf", 24) #game font
27  start_font = pygame.font.Font("Assets/Other/
    PressStart2P-Regular.ttf", 40)
28  title_font = pygame.font.Font("Assets/Other/
    PressStart2P-Regular.ttf", 60)
29
30  # variables & state
31  game_speed = 3 #initial game speed
32  speed_multiplier = 1 # used for slow motion powerup (1 =
    constant speed at 3)
33  player_score = 0 #initial player score
34  game_over = False #initial game over state
```

Snippet code above will sets up the game window, fonts, and core game state variables. First it create a pygame display with resolution of 1280x720 pixels, sets clock object to control the game frame rate and assign title to the game window. It loads the same pixel font at different size for general game text. It also initialize important game variables and its values.

```
38  # powerup states
39  shield_active = False
40  shield_end_time = 0 #initial shield end time
41  jump_boost_active = False
42  jump_boost_end_time = 0
43  double_points_active = False
44  double_points_end_time = 0
45
46  # surfaces
47  ground = pygame.image.load("Assets/Other/Track.png") #load
    ground image
48  ground = pygame.transform.scale(ground, (1280, 20)) #scale
    ground image
49  ground_x = 0 #initial ground x position at left
50
51  cloud_img = pygame.image.load("Assets/Other/Cloud.png")
52  cloud_img = pygame.transform.scale(cloud_img, (200, 80))
53  background_img = pygame.image.load("Assets/Other/christmas_bg.
    png").convert()
54  background_img = pygame.transform.scale(background_img, (1280,
    720))
```

The first block of the code will initialize the powerups variables and its values. Then it will load the ground track image and resize it and set the initial horizontal x position into 0. It also load the cloud objects and scale it, lastly it will load the christmas background image and resize it.

```
# sounds
death_sfx = pygame.mixer.Sound("sfx/lose.mp3") #death sound effect
points_sfx = pygame.mixer.Sound("sfx/100points.mp3") #points sound effect
jump_sfx = pygame.mixer.Sound("sfx/jump.mp3") #jump sound effect
pickup_sfx = None
try: #try load pickup sound effect
    pickup_sfx = pygame.mixer.Sound("sfx/pickup.mp3")
except Exception:
    # if no sound, ignore and no resulting error
    pickup_sfx = None

all_sfx = pygame.mixer.Sound("sfx/AllSound.mp3") #background jinggle bell music
all_sfx.set_volume(0.2) #set volume lower for background music
# all_sfx.play(loops=-1) #this will be called in main.py
```

That block of code manage the game audio experience by loading various sound effects files and setting the volume. It will play specific sound effects for actions like dying, jumping and reaching score within the multiplies of 100, and powerup pickup (it uses a try except block to attempt to load the file, if the file is missing it will automatically set the sound to none and prevent game from crashing. In the end it will load a background jingle bell sound and set 20% of volume and play it in a loop.

```
# UI Rects
start_btn_rect = pygame.Rect(0, 0, 300, 80) #width 300 height 80 for start button
start_btn_rect.center = (640, 500) #center start button on screen

# events
CLOUD_EVENT = pygame.USEREVENT #custom event for cloud spawn
pygame.time.set_timer(CLOUD_EVENT, 2000) #set timer to trigger cloud event every 2000ms (2 seconds)
```

It creates a rect for the start button within 300 pixels of width and 80 pixels of height and centers the start button on the screen. It also initializes cloud spawn events for the game and sets a timer to trigger the clouds every 2000 milliseconds.

- **sprites.py (Manage the object or sprites blueprints library)**

```
1   import pygame
2   import random
3   import config
4
5   class Cloud(pygame.sprite.Sprite): #cloud class, cloud is
    sprite that move left
6       def __init__(self, image, x_pos, y_pos):
7           super().__init__() #inherit  from pygame sprite
8           self.image = image #cloud image
9           self.rect = self.image.get_rect(center=(x_pos, y_pos))
            #cloud position centered at x_pos and y_pos, rect is
            rectangle around the image or collision box
```

The block of code above define the cloud class for a game by using pygame library sprite system to manage object that appear on the screen. By inheriting from pygame.sprite in line 5, enable the class to get access to built in methods for rendering and detect collision. The init method initialize the cloud instance with 3 parameters (the image and their x and y coordinates). Then it creates a rect object that provide a rectangular boundary around the image and centered. This rect is essential and acts as a collision box.

```
14  class Dino(pygame.sprite.Sprite): #dinosaur class the main
    character
15      def __init__(self, x_pos, y_pos):
16          super().__init__() #inherit from pygame sprite
17          self.running_sprites = [] #list contain running sprites
18          self.ducking_sprites = [] #list contain ducking sprites
19
20          self.running_sprites.append(pygame.transform.scale(
21              pygame.image.load("Assets/Dino/DinoRun1revv.png"),
                (80, 100))) #load and scale running sprite 1
22          self.running_sprites.append(pygame.transform.scale(
23              pygame.image.load("Assets/Dino/DinoRun2revv.png"),
                (80, 100)))
24
25          self.ducking_sprites.append(pygame.transform.scale(
26              pygame.image.load("Assets/Dino/DinoDuck1revv.png"),
                (110, 60)))
27          self.ducking_sprites.append(pygame.transform.scale(
28              pygame.image.load("Assets/Dino/DinoDuck2revv.png"),
                (110, 60)))
```

The dino class defines the main user character by inheriting from pygame.sprite, in the init method the parent sprite is caled to properly set the sprite behaviour, then two list is created to store the animation of the sprite and the size was scaled so it will present properly on the screen. These list allow the game to switch frames smoothly.

```
30          self.x_pos = x_pos #dinosaur x position
31          self.y_pos = y_pos #dinosaur y position
32          self.current_image = 0 #current image index is 0
            (start) for animation
33          self.image = self.running_sprites[self.current_image]
34          self.rect = self.image.get_rect(center=(self.x_pos,
            self.y_pos))
35          self.velocity = 0 #initial vertical velocity
36          self.ducking = False #initial ducking state is false
```

This part of code set the dinosaur position, animation state, and the movement attributes. The x pos and y pos values are stored to define the dino start position on the screen. The current image is set to 0 meaning the animation will start from the first frame. The image attribute is then assign using the first running sprite with a rect on it. Finally, the velocity is set to 0 to represent no vertical movement when start, and ducking also set into false.

```
38      def jump(self):
39          # jump allowed only if on ground (prevent double jump
            click)
40          if self.rect.centery >= 360:
41              config.jump_sfx.play() #play jump sound effect
42              # Jump height depends on jump boost
43              if config.jump_boost_active:
44                  self.velocity = -15 # higher jump with boost
45              else:
46                  self.velocity = -12 # normal jump
47
48      def duck(self):
49          self.ducking = True #set ducking state to true
50          self.rect.centery = 380 #adjust y position for ducking
51
52      def unduck(self):
53          self.ducking = False #set ducking state to false
54          self.rect.centery = 360 #adjust y position back to
            normal
```

The jump method allow the dino to jump only when it is on the ground, which is checked by make sure its vertical position is at or below a fixed ground level and it also can prevent double jumps. When a jump is triggered, a sound is played, and its vertical velocity is set to negative value so it can move upward, within a higher jump when the jump boost is active. The duck method put the dino into a crouching mode by setting the ducking value into true. The unduck method resets the dinosaur to its normal by setting the value to false.

```
56      def apply_gravity(self):
57          self.rect.centery += self.velocity #apply vertical
            velocity to y position
58          self.velocity += 0.25 # gravity add to velocity
59
60          if self.rect.centery >= 360: #if on ground
61              self.rect.centery = 360  #reset y position to
                ground level
62              self.velocity = 0 #reset velocity when on ground
63
64      def update(self): #update method called every frame
65          self.animate() #call animate method
66          self.apply_gravity() #apply gravity effect
67
68      def animate(self):
69          self.current_image += 0.05 #increment current image for
            dino animation speed
70          if self.current_image >= 2: #loop back to first image >
            index 0
71              self.current_image = 0 #reset to first image
```

The apply gravity method simulates physical weight by adding a constant 0f 0.25 to the dino character vertical velocity and updating its y coordinate, while setting floor check position at 360 to stop the character from falling indefinitely. The animate method creates a smooth running effect by add 0.05 each frame and looping back again to the first frame. These actions are coordinated by update method.

```
78  class Cactus(pygame.sprite.Sprite): #cactus obstacle class
79      def __init__(self, x_pos, y_pos):
80          super().__init__()
81          self.x_pos = x_pos
82          self.y_pos = y_pos
83          self.sprites = []
84          for i in range(1, 7):
85              current_sprite = pygame.transform.scale(
86                  pygame.image.load(f"Assets/Cactus/cactus{i}revv.
                    png"), (100, 150))
87              self.sprites.append(current_sprite)
88          self.image = random.choice(self.sprites) #randomly
            choose one cactus sprite out of 7
89          self.rect = self.image.get_rect(center=(self.x_pos,
            self.y_pos))
```

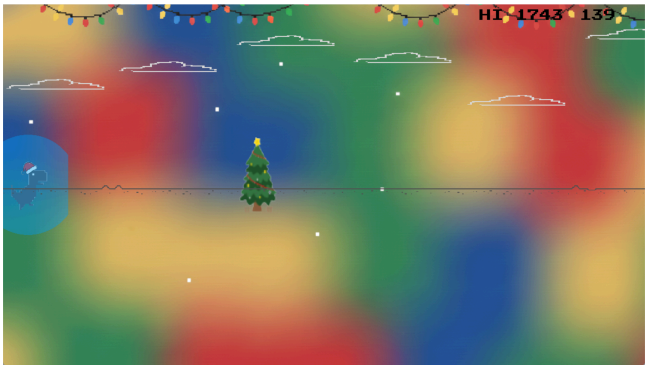
The cactus clss manage the obstacle system for the game by inheriting from pygame.sprite. The class uses a for loop to iterate from 1 to 6, loading cactus image assets from the project folder and scale them. These processed image are stored in self.sprites list. And random.choice use for select random variation of the cactus sprites 1 out of 7. Finally, the code establishes the object's physical presence by creating a rect.

```
91      def update(self):
92          # use speed multiplier for slow motion
93          self.x_pos -= config.game_speed * config.
            speed_multiplier #move left by game speed and increase
            with speed multiplier
94          self.rect = self.image.get_rect(center=(self.x_pos,
            self.y_pos))
95
96          if self.rect.right < -50: #if cactus is offscreen to
            the left
97              self.kill() #remove cactus from all groups and ram
                memory
```

The update method for class manages the movement and lifecycle of the obstacle. After updating the position the code repositions the object collision rectangle to ensure precise collision detection. Memory management checks if the cactus right edge has passed the coordinate -50, if the obstacle is safely off screen to the left kill method is triggered to remove all active groups. The rest of the codes apply and do the same thing for all objects including clouds, dino, pterodactyl (bird), and also the powerups.

C. Project Documentation

- **Onedrive Video Link (Maximize volume to hear the sound effects in the gameplay):** https://binusianorg-my.sharepoint.com/personal/aliansky_viando_binus_ac_id/_layouts/15/guestaccess.aspx?share=IQCtNenWU6DnQIYPrDiR7WfZAZaY58XsMNSCIYDJDeu_e8w&e=nHbzak
- **Screenshots of the project:**



D. Dino Level-Up Project Poster

E. Statement on The Use of Artificial Intelligence

1. AI Tools and Versions

- Tools: Gemini 2.0 and ChatGPT 4.0
- Version: Web-based

2. Specific Purpose and Project Phases

AI was used during the Development and Debugging phase for my Dino Level-Up Project:

- **Mathematical Logic:** Assisting with the dynamic calculation of spawn rates (cooldown logic) so that obstacle density fits properly with its game speed.
- **Error Handling:** Implementing try and except blocks for external asset loading (sounds and images) to ensure the game keeps working even if a specific file is missing.
- **Ideation:** Used to brainstorm power ups types that would balance gameplay difficulty.

3. Examples of Prompt

- How do I make and calculate a dynamic cooldown in pygame so that as game speed increases the obstacle spawn time decreases but never goes below 800ms?
- Give me five power-up ideas for a dino runner chrome game that are easy to implement in Pygame but change the gameplay experience so it will be more fun
- Show me how to use pygame.time.get_ticks to make a 3 second timer for a power up that automatically turn off

4. Reflection

The AI that i use as a logic consultant rather than a primary code maker. While the AI suggested mathematical formulas for example like the speed factor scaling, I was responsible for setting numbers and values like the 0.25 gravity constant and the -12 vs -15 jump velocities to ensure the sense of game play was correct. All visual assets and the overall project structure including the use of specific class inheritances were my original design.

References

Pygame Library: Shin, Y. (2020). *Beginning game development with Python and Pygame: From novice to professional* (2nd ed.). Apress.

Object-Oriented Programming (OOP) in Games: Phillips, D. (2021). *Python 3 object-oriented programming: Build robust and maintainable software with object-oriented design patterns in Python* (4th ed.). Packt Publishing.

Physics and Gravity in 2D Games: Millett, S. (2011). *Beginning game level design*. Cengage Learning.

Pygame and Sprite Framework: Shin, Y. (2020). *Beginning game development with Python and Pygame: From novice to professional* (2nd ed.). Apress.

Google Gemini. (2026). *Christmas-themed game background and asset concepts* [Artificial intelligence]. https://gemini.google.com