Software Solution – Source Code Document

Ivy Tech Community College

Blue team – SDEV 265 – Fall 2025

October 15th 2025

# Main Launcher (tkinter GUI)

import tkinter as tk           # GUI library for building the interface

import subprocess              # Used to launch external Python scripts (games)

import threading               # Allows games to run in background threads

import os                      # For working with file paths

import sys                     # Gives access to the current Python interpreter

# Set base directory paths

BASE\_DIR = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), ".."))  # Root folder of the project

ASSETS\_DIR = os.path.join(BASE\_DIR, "gui\_assets")                          # Folder for button images

# Prevent images from being garbage collected (keeps them in memory)

images = []

# Create main window

root = tk.Tk()

root.title("BitBox Arcade")               # Window title

root.geometry("1000x700")                 # Window size

root.configure(bg="#1e1e1e")              # Background color (dark theme)

# Function to launch a game script

def launch\_game(path):

    abs\_path = os.path.abspath(path)      # Get absolute path to the game file

    print(f"Launching: {abs\_path}")       # Print path for debugging

    if not os.path.exists(abs\_path):      # Check if the file exists

        print(f"Game not found: {abs\_path}")

        return

    # Run the game in a separate thread so GUI doesn't freeze

    def run\_game():

        try:

            root.iconify()                # Minimize the launcher window

            proc = subprocess.Popen([sys.executable, abs\_path])  # Launch game using Python

            proc.wait()                   # Wait for game to finish

        except Exception as e:

            print(f"Error launching game: {e}")  # Print any errors

        finally:

            root.deiconify()              # Restore the launcher window

            root.geometry("1000x700")     # Reset window size

            root.lift()                   # Bring window to front

            root.focus\_force()            # Force focus

            root.attributes("-topmost", True)    # Temporarily keep window on top

            root.after(500, lambda: root.attributes("-topmost", False))  # Remove topmost after delay

    threading.Thread(target=run\_game, daemon=True).start()  # Start game thread

# Function to open the About window

def open\_about\_window():

    about = tk.Toplevel(root)            # Create a new popup window

    about.title("About BitBox Arcade")

    about.geometry("500x400")            # Size of the About window

    about.configure(bg="#2e2e2e")        # Background color

    # Message with team info

    message = (

        "This application was developed by the Blue Team\n"

        "Ivy Tech Community College – Fall 2025\n"

        "Course: SDEV 265\n\n"

        "Developers:\n"

        "• Makayla Harrison\n"

        "• Craig Andrew Hutson\n"

        "• Alex Michael Johnston\n"

        "• Brandon Kesner"

    )

    # Display message

    tk.Label(about, text=message, font=("Arial", 12), fg="white", bg="#2e2e2e", justify="center").pack(pady=40)

    # Close button

    tk.Button(about, text="Close", command=about.destroy, font=("Arial", 12),

              bg="#444", fg="white", activebackground="#666", activeforeground="white").pack(pady=20)

# Create side panel for About button

side\_panel = tk.Frame(root, bg="#1e1e1e")

side\_panel.pack(side="left", fill="y", padx=(20, 0), pady=20)

# Add About button to side panel

tk.Button(side\_panel, text="About Game", command=open\_about\_window,

          font=("Arial", 12), width=12, height=2,

          bg="#444", fg="white", activebackground="#666", activeforeground="white").pack(pady=10)

# Create main grid area for game buttons

grid = tk.Frame(root, bg="#1e1e1e")

grid.pack()

# Title label at the top

tk.Label(root, text="BitBox Arcade", font=("Arial", 32, "bold"), fg="white", bg="#1e1e1e").pack(pady=20)

# === Game Buttons ===

# Froggy Jump (Alex's game)

frog\_img = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "frog\_button.png"))  # Load button image

images.append(frog\_img)  # Prevent garbage collection

frog\_frame = tk.Frame(grid, bg="#1e1e1e")  # Create frame for button

frog\_frame.grid(row=0, column=0, padx=40, pady=20)  # Position in grid

tk.Button(frog\_frame, image=frog\_img, width=150, height=200,

          command=lambda: launch\_game(os.path.join(BASE\_DIR, "Alex", "froggy\_jump", "main.py")),

          borderwidth=0, bg="#1e1e1e").pack()  # Game launch button

tk.Label(frog\_frame, text="Froggy Jump", font=("Arial", 14), fg="white", bg="#1e1e1e").pack(pady=10)  # Game label

# Makayla's Game

mak\_img = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "placeholder.png"))

images.append(mak\_img)

mak\_frame = tk.Frame(grid, bg="#1e1e1e")

mak\_frame.grid(row=0, column=1, padx=40, pady=20)

tk.Button(mak\_frame, image=mak\_img, width=150, height=200,

          command=lambda: launch\_game(os.path.join(BASE\_DIR, "Makayla", "space\_blaster", "main.py")),

          borderwidth=0, bg="#1e1e1e").pack()

tk.Label(mak\_frame, text="Makayla's Game", font=("Arial", 14), fg="white", bg="#1e1e1e").pack(pady=10)

# Craig's Game

craig\_img = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "duck\_button.png"))

images.append(craig\_img)

craig\_frame = tk.Frame(grid, bg="#1e1e1e")

craig\_frame.grid(row=1, column=0, padx=40, pady=20)

tk.Button(craig\_frame, image=craig\_img, width=150, height=200,

          command=lambda: launch\_game(os.path.join(BASE\_DIR, "Craig", "duckhunt", "shoot.py")),

          borderwidth=0, bg="#1e1e1e").pack()

tk.Label(craig\_frame, text="Duck Hunt", font=("Arial", 14), fg="white", bg="#1e1e1e").pack(pady=10)

# Brandon's Game

brandon\_img = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "placeholder.png"))

images.append(brandon\_img)

brandon\_frame = tk.Frame(grid, bg="#1e1e1e")

brandon\_frame.grid(row=1, column=1, padx=40, pady=20)

tk.Button(brandon\_frame, image=brandon\_img, width=150, height=200,

          command=lambda: launch\_game(os.path.join(BASE\_DIR, "Brandon", "tower\_tactics", "main.py")),

          borderwidth=0, bg="#1e1e1e").pack()

tk.Label(brandon\_frame, text="Brandon's Game", font=("Arial", 14), fg="white", bg="#1e1e1e").pack(pady=10)

# Start the GUI event loop

root.mainloop()

### --------------------------------------------------------------------------------------------------

# Froggy Jump

import pygame, random, os  # Import game engine, randomness, and file path tools

# === Game Constants ===

WIDTH, HEIGHT = 400, 600                 # Window size

PLAYER\_W, PLAYER\_H = 40, 40              # Player size

PLATFORM\_W, PLATFORM\_H = 60, 10          # Platform size

GRAVITY = .6                             # Gravity strength

JUMP\_VEL = -20                           # Jump velocity

MOVE\_SPEED = 4                           # Horizontal movement speed

NUM\_PLATFORMS = 8                        # Number of platforms in play

# === Load Assets ===

ASSETS = os.path.join(os.path.dirname(\_\_file\_\_), "assets")  # Path to assets folder

# Background and platform images

bg\_img = pygame.image.load(os.path.join(ASSETS, "background.png"))

plat\_img = pygame.image.load(os.path.join(ASSETS, "platform.png"))

# Spider enemy images

spider\_img = pygame.image.load(os.path.join(ASSETS, "spider.png"))

spider\_flip = pygame.image.load(os.path.join(ASSETS, "spider\_flipped.png"))

# Player movement sprites

player\_left = pygame.image.load(os.path.join(ASSETS, "player.png"))

player\_right = pygame.image.load(os.path.join(ASSETS, "player\_flipped.png"))

player\_jump\_left = pygame.image.load(os.path.join(ASSETS, "player\_jump\_flipped.png"))

player\_jump\_right = pygame.image.load(os.path.join(ASSETS, "player\_jump.png"))

player\_fall\_left = pygame.image.load(os.path.join(ASSETS, "player\_fall\_flipped.png"))

player\_fall\_right = pygame.image.load(os.path.join(ASSETS, "player\_fall.png"))

# Start and death screens

start\_menu\_img = pygame.image.load(os.path.join(ASSETS, "start\_menu.png"))

death\_screens = [pygame.image.load(os.path.join(ASSETS, f"gameover{i}.png")) for i in range(1, 13)]

# === Sound Setup ===

pygame.init()

pygame.mixer.init()

screen = pygame.display.set\_mode((WIDTH, HEIGHT))  # Create game window

clock = pygame.time.Clock()                        # Control frame rate

font = pygame.font.SysFont("Arial", 16)            # Score font

# Background music

pygame.mixer.music.load(os.path.join(ASSETS, "background.mp3"))

pygame.mixer.music.play(-1)  # Loop forever

# Sound effects

jump\_sfx = pygame.mixer.Sound(os.path.join(ASSETS, "jump.wav"))

hurt\_sfx = pygame.mixer.Sound(os.path.join(ASSETS, "hurt.wav"))

die\_sfx = pygame.mixer.Sound(os.path.join(ASSETS, "die.wav"))

# === Game State Variables ===

score = 0

game\_active = False

game\_over = False

start\_menu\_shown = False

current\_death\_screen = None

# === Spider Enemy Class ===

class Spider:

    def \_\_init\_\_(self, x, y):

        self.rect = pygame.Rect(x, y, PLATFORM\_W, PLATFORM\_H)  # Position and size

        self.images = [spider\_img, spider\_flip]                # Animation frames

        self.current = 0                                       # Current frame index

        self.last\_switch = pygame.time.get\_ticks()             # Last animation switch

        self.interval = random.randint(3000, 5000)             # Time between switches

    def draw(self):

        screen.blit(self.images[self.current], self.rect.topleft)  # Draw spider

    def update(self):

        now = pygame.time.get\_ticks()

        if now - self.last\_switch > self.interval:  # Time to switch image

            self.current ^= 1                       # Flip between 0 and 1

            self.last\_switch = now

            self.interval = random.randint(3000, 5000)

    def move(self, dy):

        self.rect.y += dy  # Move spider vertically

    def reposition\_above(self, platform):

        self.rect.topleft = (platform.draw\_rect.x, platform.draw\_rect.y - PLATFORM\_H)

# === Platform Class ===

class Platform:

    def \_\_init\_\_(self, x, y, has\_spider=True):

        self.draw\_rect = pygame.Rect(x, y, PLATFORM\_W, PLATFORM\_H)  # Visual platform

        self.rect = pygame.Rect(x + 10, y + PLATFORM\_H // 2, PLATFORM\_W - 20, 4)  # Collision zone

        self.spider = Spider(x, y - PLATFORM\_H) if has\_spider else None  # Optional spider

    def draw(self):

        screen.blit(plat\_img, self.draw\_rect.topleft)

        if self.spider:

            self.spider.update()

            self.spider.draw()

    def move(self, dy):

        self.draw\_rect.y += dy

        self.rect.y += dy

        if self.spider:

            self.spider.move(dy)

    def recycle(self):

        # Move platform back to top with new random position

        new\_x = random.randint(0, WIDTH - PLATFORM\_W)

        new\_y = random.randint(-120, -40)

        self.draw\_rect.topleft = (new\_x, new\_y)

        self.rect.topleft = (new\_x + 10, new\_y + PLATFORM\_H // 4)

        # Random chance to spawn spider

        if random.random() < 0.08:

            if not self.spider:

                self.spider = Spider(new\_x, new\_y - PLATFORM\_H)

            else:

                self.spider.reposition\_above(self)

        else:

            self.spider = None

# === Player Class ===

class Player:

    def \_\_init\_\_(self):

        self.rect = pygame.Rect(WIDTH//2, HEIGHT-80, PLAYER\_W, PLAYER\_H)

        self.vx = 0

        self.vy = 0

        self.on\_ground = False

        self.disabled = False

        self.facing\_right = True

        self.image = player\_right

    def move(self):

        global score, game\_active, game\_over, current\_death\_screen

        self.vy += GRAVITY

        self.rect.x += self.vx

        self.rect.y += self.vy

        # Check for spider collision

        for plat in platforms:

            if plat.spider and self.rect.colliderect(plat.spider.rect):

                hurt\_sfx.play()

                self.vy = max(0, self.vy)

                self.disabled = True

        # Check for platform landing

        if not self.disabled:

            self.on\_ground = False

            for plat in platforms:

                if self.rect.colliderect(plat.rect) and self.vy >= 0:

                    self.rect.bottom = plat.rect.top

                    self.vy = 0

                    self.on\_ground = True

        # Scroll screen upward if player climbs

        if self.rect.top < HEIGHT//3:

            dy = HEIGHT//3 - self.rect.top

            self.rect.top += dy

            for plat in platforms:

                plat.move(dy)

                if plat.draw\_rect.top > HEIGHT:

                    plat.recycle()

            score += int(dy)

        # Check for falling off screen

        if self.rect.bottom > HEIGHT:

            die\_sfx.play()

            game\_active = False

            game\_over = True

            score = 0

            current\_death\_screen = random.choice(death\_screens)

        self.update\_image()

    def jump(self):

        if self.on\_ground:

            self.vy = JUMP\_VEL

            jump\_sfx.play()

    def set\_dir(self, d):

        self.vx = d \* MOVE\_SPEED

        if d != 0:

            self.facing\_right = d > 0

    def update\_image(self):

        # Choose sprite based on movement direction and velocity

        if self.vy < -1:

            self.image = player\_jump\_right if self.facing\_right else player\_jump\_left

        elif self.vy > 1:

            self.image = player\_fall\_right if self.facing\_right else player\_fall\_left

        else:

            self.image = player\_right if self.facing\_right else player\_left

# === Platform Generator ===

def generate():

    plats = []

    spacing = HEIGHT // NUM\_PLATFORMS

    for i in range(NUM\_PLATFORMS):

        x = random.randint(0, WIDTH - PLATFORM\_W)

        y = HEIGHT - (i + 1) \* spacing

        has\_spider = i != 0 and random.random() < 0.08

        plat = Platform(x, y, has\_spider)

        plats.append(plat)

        if i == 0:

            player.rect.midbottom = plat.rect.midtop  # Start player on bottom platform

    return plats

# === Game Setup ===

player = Player()

platforms = generate()

# === Main Game Loop ===

running = True

while running:

    # Show start menu first

    if not start\_menu\_shown:

        screen.blit(start\_menu\_img, (0, 0))

        pygame.display.flip()

        for e in pygame.event.get():

            if e.type == pygame.QUIT: running = False

            elif e.type == pygame.KEYDOWN and e.key == pygame.K\_s:

                game\_active = True

                start\_menu\_shown = True

                player = Player()

                platforms = generate()

        continue

    # Show death screen or restart menu

    if not game\_active:

        if game\_over and current\_death\_screen:

            screen.blit(current\_death\_screen, (0, 0))

        else:

            screen.blit(start\_menu\_img, (0, 0))  # Show start menu again if not game over

        pygame.display.flip()

        for e in pygame.event.get():

            if e.type == pygame.QUIT: running = False

            elif e.type == pygame.KEYDOWN and e.key == pygame.K\_s:

                # Restart game

                game\_active = True

                game\_over = False

                player = Player()

                platforms = generate()

        continue

    # === Active Gameplay ===

    screen.blit(bg\_img, (0, 0))  # Draw background

    # Handle input events

    for e in pygame.event.get():

        if e.type == pygame.QUIT:

            running = False

        elif e.type == pygame.KEYDOWN:

            if e.key == pygame.K\_w: player.jump()       # Jump

            elif e.key == pygame.K\_a: player.set\_dir(-1) # Move left

            elif e.key == pygame.K\_d: player.set\_dir(1)  # Move right

        elif e.type == pygame.KEYUP:

            if e.key in [pygame.K\_a, pygame.K\_d]: player.set\_dir(0)  # Stop moving

    # Update player and platforms

    player.move()

    for plat in platforms:

        plat.draw()

    # Draw player and score

    screen.blit(player.image, player.rect.topleft)

    screen.blit(font.render(f"Score: {score}", True, (0, 0, 0)), (10, 10))

    # Refresh display and control frame rate

    pygame.display.flip()

    clock.tick(50)

# === Exit Game ===

pygame.quit()

### --------------------------------------------------------------------------------------------------

# Hunt The Duck

import pygame

import os

from random import randint

# === Setup ===

os.environ['SDL\_VIDEO\_CENTERED'] = '1'  # Center the game window on screen

pygame.init()

pygame.font.init()

WIDTH, HEIGHT = 500, 500  # Window size

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("HUNT THE DUCK")  # Window title

clock = pygame.time.Clock()  # Controls frame rate

# === Game States ===

GAME\_STATE\_TITLE = 0

GAME\_STATE\_PLAYING = 1

GAME\_STATE\_GAME\_OVER = 2

GAME\_STATE\_WINNER = 3

current\_game\_state = GAME\_STATE\_TITLE  # Start at title screen

# === Asset Paths ===

SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))  # Current script location

IMAGES\_DIR = os.path.join(SCRIPT\_DIR, "images")          # Folder with image assets

# === Load Images Safely ===

def load\_image(name):

    path = os.path.join(IMAGES\_DIR, name)

    try:

        return pygame.image.load(path).convert\_alpha()

    except pygame.error as e:

        print(f"Failed to load image '{name}': {e}")

        raise SystemExit

# === Game Images ===

background = load\_image("field.png")

duck\_img = load\_image("duckfly3.png")

duck2\_img = load\_image("duckfly4.png")

dead\_duck\_img = load\_image("deadduck3.png")

sight\_img = load\_image("sight3.png")

# === Fonts ===

font\_big = pygame.font.SysFont("Arial", 60)

font\_med = pygame.font.SysFont("Arial", 30)

font\_huge = pygame.font.SysFont("Arial", 90)

# === Duck Class ===

class Duck:

    def \_\_init\_\_(self, image, x, y):

        self.image = image

        self.rect = self.image.get\_rect(center=(x, y))  # Position and size

        self.vy = 0

        self.dead = False

    def draw(self):

        screen.blit(self.image, self.rect)  # Draw duck

    def reset(self, x, y, image):

        self.image = image

        self.rect.center = (x, y)

        self.vy = 0

        self.dead = False

# === Game Objects ===

apple = Duck(duck\_img, randint(10, 200), randint(300, 400))  # Main duck

duck2 = Duck(duck2\_img, randint(480, 500), randint(300, 400))  # Second duck

duck2\_active = False  # Starts inactive

sight\_rect = sight\_img.get\_rect()  # Sight reticle

score = 0

game\_over = False

GRAVITY = 0.5  # Gravity for falling ducks

# === Reset Functions ===

def reset\_apple():

    apple.reset(randint(50, WIDTH - 50), randint(300, 400), duck\_img)

def reset\_duck2():

    duck2.reset(randint(480, 500), randint(300, 400), duck2\_img)

# === Text Drawing Helper ===

def draw\_text(text, font, color, center):

    surf = font.render(text, True, color)

    rect = surf.get\_rect(center=center)

    screen.blit(surf, rect)

# === Main Game Loop ===

running = True

while running:

    screen.fill((0, 0, 0))  # Clear screen

    mouse\_pos = pygame.mouse.get\_pos()

    sight\_rect.center = mouse\_pos  # Move sight to mouse

    # === Event Handling ===

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

        elif event.type == pygame.MOUSEBUTTONDOWN and current\_game\_state == GAME\_STATE\_PLAYING and not game\_over:

            # Check if duck was hit

            if apple.rect.collidepoint(mouse\_pos):

                score += 1

                apple.dead = True

                apple.vy = 0

            elif duck2\_active and duck2.rect.collidepoint(mouse\_pos):

                score += 1

                duck2.dead = True

            else:

                current\_game\_state = GAME\_STATE\_GAME\_OVER

                game\_over = True

        elif event.type == pygame.KEYDOWN:

            if current\_game\_state == GAME\_STATE\_TITLE and event.key == pygame.K\_SPACE:

                current\_game\_state = GAME\_STATE\_PLAYING

            elif current\_game\_state in [GAME\_STATE\_GAME\_OVER, GAME\_STATE\_WINNER] and event.key == pygame.K\_r:

                # Restart game

                current\_game\_state = GAME\_STATE\_PLAYING

                game\_over = False

                score = 0

                duck2\_active = False

                reset\_apple()

                reset\_duck2()

    # === Title Screen ===

    if current\_game\_state == GAME\_STATE\_TITLE:

        screen.fill((0, 100, 0))

        draw\_text("Hunt The Duck!", font\_huge, (255, 165, 0), (WIDTH // 2, HEIGHT // 3))

        draw\_text("Press SPACE to Start", font\_med, (255, 255, 255), (WIDTH // 2, HEIGHT // 1.5))

    # === Gameplay ===

    elif current\_game\_state == GAME\_STATE\_PLAYING:

        screen.blit(background, (0, 0))

        draw\_text(f"Score: {score}", font\_med, (255, 255, 255), (70, 20))

        if not game\_over:

            # Duck movement logic based on score

            if not apple.dead:

                if 5 <= score <= 10:

                    apple.rect.x += 3

                    apple.rect.y -= 2

                elif 10 < score <= 15:

                    apple.rect.x += 4

                    apple.rect.y -= 3

                    duck2.rect.x -= 3

                    duck2.rect.y -= 2

                elif 15 < score <= 20:

                    apple.rect.x += 5

                    apple.rect.y -= 4

                    duck2.rect.x -= 4

                    duck2.rect.y -= 3

                elif 20 < score <= 25:

                    apple.rect.x += 6

                    apple.rect.y -= 5

                    duck2.rect.x -= 5

                    duck2.rect.y -= 4

                elif 25 < score <= 30:

                    apple.rect.x += 8

                    apple.rect.y -= 8

                    duck2.rect.x -= 8

                    duck2.rect.y -= 8

                else:

                    apple.rect.x += 2

                    apple.rect.y -= 1

                # Wrap or reset duck if off screen

                if apple.rect.left > WIDTH:

                    apple.rect.right = 0

                if apple.rect.top < 0:

                    reset\_apple()

            else:

                # Dead duck falls

                apple.vy += GRAVITY

                apple.rect.y += apple.vy

                apple.image = dead\_duck\_img

            # Duck2 movement and falling

            if duck2\_active:

                if not duck2.dead:

                    if duck2.rect.right < 0:

                        duck2.rect.left = WIDTH

                    if duck2.rect.top < 0:

                        reset\_duck2()

                else:

                    duck2.vy += GRAVITY

                    duck2.rect.y += duck2.vy

                    duck2.image = dead\_duck\_img

            # Reset ducks if they fall off screen

            if apple.rect.bottom > HEIGHT:

                reset\_apple()

            if duck2.rect.bottom > HEIGHT:

                reset\_duck2()

            # Activate duck2 after score threshold

            if score >= 10 and not duck2\_active:

                duck2\_active = True

                reset\_duck2()

            # Win condition

            if score == 30:

                current\_game\_state = GAME\_STATE\_WINNER

                game\_over = True

            # Draw ducks

            apple.draw()

            if duck2\_active:

                duck2.draw()

    # === Game Over Screen ===

    elif current\_game\_state == GAME\_STATE\_GAME\_OVER:

        screen.fill((0, 0, 0))

        draw\_text("GAME OVER", font\_big, (255, 0, 0), (WIDTH // 2, HEIGHT // 2 - 40))

        draw\_text(f"Final Score: {score}", font\_med, (255, 255, 255), (WIDTH // 2, HEIGHT // 2 + 10))

        draw\_text("Hit 'R' to restart", font\_med, (255, 255, 255), (WIDTH // 2, HEIGHT // 2 + 50))

    # === Winner Screen ===

    elif current\_game\_state == GAME\_STATE\_WINNER:

        screen.fill((0, 100, 0))

        draw\_text("YOU WIN!!!", font\_big, (255, 165, 0), (WIDTH // 2, HEIGHT // 2 - 40))

        draw\_text(f"Final Score: {score}", font\_med, (255, 255, 255), (WIDTH // 2, HEIGHT // 2 + 10))

        draw\_text("Hit 'R' to restart", font\_med, (255, 255, 255), (WIDTH // 2, HEIGHT // 2 + 50))

    # === Draw Sight Reticle ===

    screen.blit(sight\_img, sight\_rect)

    # === Refresh Display ===

    pygame.display.flip()       # Update the screen with everything drawn

    clock.tick(60)              # Limit frame rate to 60 FPS

# === Exit Game ===

pygame.quit()                   # Cleanly close the game when loop ends

### --------------------------------------------------------------------------------------------------

# David Vs Goliath

import tkinter as tk

import random

from PIL import Image, ImageTk

import os

import pygame

#-------Get The Folder----------

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

ASSETS\_DIR = os.path.join(BASE\_DIR,"assets")

#-------Setup Main Game Window-------

window = tk.Tk()

window.title("David and Golaith")

WIDTH = 695

HEIGHT = 420

# Create canvas where game object will be drawn

canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT)

canvas.pack()

#Sound system

pygame.init() #Initialize the pygame sound system

pygame.mixer.init()

#load background music

background\_music = os.path.join(ASSETS\_DIR, "gameplay\_music.mp3")

pygame.mixer.music.load(background\_music)

pygame.mixer.music.play(-1) #loops forever

#Load sound effects

attack\_sound\_1 = pygame.mixer.Sound(os.path.join(ASSETS\_DIR, "attack1.wav")) #Slingshot release

attack\_sound\_2 = pygame.mixer.Sound(os.path.join(ASSETS\_DIR, "attack2.wav")) #hit sound

# Background Load resize Tile

bg\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "Groundtilebg.png"))

background = canvas.create\_image (4,4, image=bg\_image, anchor="nw")

window.bg\_image = bg\_image #Prevent garbage collection

# Menu Screen Assets

menu\_bg\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "menu\_background.png"))

play\_button\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "play\_button.png"))

#player positioned Right

david\_image\_right = tk.PhotoImage(file= os.path.join(ASSETS\_DIR,"david\_idle.png"))

david\_walk1\_right = tk.PhotoImage(file = os.path.join(ASSETS\_DIR, "david\_walk1.png"))

#Player Positoned Left

david\_idle\_left = tk.PhotoImage(file= os.path.join(ASSETS\_DIR, "david\_idle\_left.png"))

david\_walk1\_left = tk.PhotoImage(file = os.path.join(ASSETS\_DIR,"david\_walk1\_left.png"))

#David attack sprites

david\_attack\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "david\_attack.png"))

david\_attack\_left\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "david\_attack\_left.png"))

#david attack2 sprite 2

david\_attack\_image2 = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "david\_attack2.png"))

david\_attack\_left\_image2 = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "david\_attack2\_left.png"))

#Stone Icon and UI

stone\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "stone\_icon.png"))

stone\_icon = canvas.create\_image(620, 20, image=stone\_image) #adjust x/y to place top-right

stone\_text = canvas.create\_text (650, 20, text="10", font=("Arial", 17, "bold"), fill="black") #next to icon

#Win and lose Screens

win\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "win\_screen.png"))

lose\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "lose\_screen.png"))

#Stone Count

MAX\_STONES = 10

stone\_count = MAX\_STONES

#put in a list of animation

david\_facing\_right = True

david\_frames\_right = [david\_image\_right, david\_walk1\_right]

david\_frames\_left = [david\_idle\_left, david\_walk1\_left]

david\_frame\_index = 0

david\_animation\_counter = 0

#start with idle frame

david = canvas.create\_image(250, 365, image = david\_image\_right)

#Get David Image dimension for accurate collusion and bounds

david\_width = david\_image\_right.width()

david\_height = david\_image\_right.height()

#Enemy

goliath\_image = tk.PhotoImage(file= os.path.join(ASSETS\_DIR,"Goliath\_walk1\_left.png"))

goliath\_walk = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "Goliath\_walk2\_left.png"))

goliath\_walk1 = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "Goliath\_walk1.png"))

goliath\_walk2 = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "Goliath\_walk2.png"))

#HealthBar Images

goliath\_health\_image = [tk.PhotoImage(file=os.path.join(ASSETS\_DIR, f"health{i}.png")) for i in range (1, 7)]

#Put them into a list of animation

goliath\_frames = [ goliath\_walk1, goliath\_walk2]

goliath\_frame\_index = 0

goliath\_animation\_counter = 0

#Count the images you have (used for scaling health)

HEALTH\_IMAGE\_COUNT = len (goliath\_health\_image)

#start with idle

goliath = canvas.create\_image(250, 75, image = goliath\_image)

#Goliath HealthBar on Canvas

global goliath\_health\_bar

goliath\_health\_bar = canvas.create\_image( 100, 20, image=goliath\_health\_image[HEALTH\_IMAGE\_COUNT -1] ) # you can change (250, 40) to plave it higher or lower

canvas.create\_text(180, 20, text="HEALTH", font=("Arial", 17, "bold"), fill="black", anchor="w")

#Get Goliath Image dimension for accurate collusion and bounds

goliath\_width = goliath\_image.width()

goliath\_height = goliath\_image.height()

# Soldier sprites

soldier\_idle\_right = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "soldier\_idle\_right.png"))

soldier\_idle\_left = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "soldier\_idle\_left.png"))

soldier\_walk\_right = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "soldier\_walk\_right.png"))

soldier\_walk\_left = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "soldier\_walk\_left.png"))

soldier\_attack\_right = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "soldier\_attack\_right.png"))

soldier\_attack\_left = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "soldier\_attack\_left.png"))

#-------Game Variable-------

stones = [] #List to track all the fires stones

david\_speed = 0 #Left/right movement speed for David

goliath\_direction = 1 #Goliath moves hortizontally

#Health Sys

MAX\_GOLIATH\_HEALTH = 6 #You can change this anytime to make Goliath stronger or weaker

goliath\_health = MAX\_GOLIATH\_HEALTH #Goliath starts at full health

soldiers = []  # List to store active soldiers

#-------Functions-------

def restart\_game(): #Restart function for game

    global goliath\_health, stones, david, goliath, goliath\_health\_bar, stone\_count, stone\_icon, stone\_text

    pygame.mixer.music.play(-1)

    goliath\_health= MAX\_GOLIATH\_HEALTH #Restart Goliath health to full

    stone\_count= MAX\_STONES

    stones.clear() #clear the list properly

    canvas.delete("all")

    bg\_image = tk.PhotoImage(file=os.path.join(ASSETS\_DIR, "Groundtilebg.png"))

    canvas.create\_image(0,0, image=bg\_image, anchor="nw")

    window.bg\_image = bg\_image

    #redraw all players

    david = canvas.create\_image (250, 365, image=david\_image\_right)

    goliath = canvas.create\_image (250, 75, image = goliath\_image)

    #Redraw the goliath health bar

    goliath\_health\_bar = canvas.create\_image (100, 20, image=goliath\_health\_image[HEALTH\_IMAGE\_COUNT -1])

    canvas.create\_text(180, 20, text="HEALTH", font=("Arial", 17, "bold"), fill="black", anchor="w")

    #Redraw stone Ui

    global stone\_icon, stone\_text

    stone\_icon = canvas.create\_image(620, 20, image=stone\_image)

    stone\_text = canvas.create\_text(650, 20, text = str(stone\_count), font= ("Arial", 17, "bold"), fill="black")

    update\_game()

def move\_david():  #Move left and right based on the current speed

    global david\_speed, david\_frame\_index, david\_animation\_counter, david\_facing\_right

    canvas.move(david, david\_speed, 0)

    x,y = canvas.coords(david)

    #Prevent David from leaving screen

    if x - david\_width // 2 < 0:

        canvas.move(david, -(x - david\_width //2), 0)

    elif x + david\_width //2 > WIDTH:

        canvas.move(david, WIDTH -(x + david\_width // 2), 0)

#Handle facing direction

    if david\_speed > 0: #Moving right

        david\_facing\_right = True

    elif david\_speed < 0: #Moving Left

        david\_facing\_right = False

    #Animation Handling

    if david\_speed != 0:

        david\_animation\_counter +=1

        if david\_animation\_counter % 3 ==0: #Change every 3 ticks

            david\_frame\_index = (david\_frame\_index + 1) % 2 # toggle bewteen 0 and 1

    #Choosing the correct set

            if david\_facing\_right:

                canvas.itemconfig(david, image = david\_frames\_right[david\_frame\_index])

            else:

                canvas.itemconfig(david, image=david\_frames\_left[david\_frame\_index])

#-------Fire a stone upward from David's position-------

def shoot\_stone(event = None):

    global stone\_count

    if stone\_count <= 0:

        return #Dont shhot if no stones left

    #Play slingshot release sound

    attack\_sound\_1.play()

    x,y = canvas.coords(david)

    stone = canvas.create\_oval(x-5, y-20, x+5, y-10, fill = "grey")

    stones.append(stone)

    #Decrease stone count and update text

    stone\_count -= 1

    canvas.itemconfig(stone\_text, text=str(stone\_count))

    #STAGE 1: Play attack2 sprite

    if david\_facing\_right:

        canvas.itemconfig(david, image=david\_attack\_image2)

    else:

        canvas.itemconfig(david, image=david\_attack\_left\_image2)

    #STAGE 2: After 150ms play attack1

    def play\_attack1():

        if david\_facing\_right:

            canvas.itemconfig(david, image=david\_attack\_image)

        else:

            canvas.itemconfig(david, image=david\_attack\_left\_image)

    window.after(200, play\_attack1) #Delay before switching to attack1

    #Revert back to idle image after short delay

    def reset\_to\_idle():

        if david\_facing\_right:

            canvas.itemconfig(david, image=david\_frames\_right[0]) #idle right

        else:

            canvas.itemconfig(david, image=david\_frames\_left[0]) #idle left

    window.after(300, reset\_to\_idle) #show attack image for 200ms

#Move all stones upward, check for off-screen or collision with Goliath

def move\_stones():

    global goliath\_health

    for stone in stones [:]: # Copy the list to aviod modifying while iterating

        canvas.move(stone, 0, -10)

        x1,y1,x2,y2, = canvas.coords(stone)

#Remove stone if it goes off screen

        if y2 < 0:

            canvas.delete(stone)

            stones.remove(stone)

            continue

#Get Goliath center position and calculate bounding box

        gx, gy = canvas.coords(goliath)

        gx1 = gx - goliath\_width // 2

        gx2 = gx + goliath\_width // 2

        gy1 = gy - goliath\_height // 2

        gy2 = gy + goliath\_height // 2

        # Check for collision with soldiers

        hit\_soldier = False

        for soldier in soldiers:

            sx1, sy1, sx2, sy2 = soldier.get\_bbox()

            if sx1 < x1 < sx2 and sy1 < y1 < sy2:

                canvas.delete(stone)

                stones.remove(stone)

                hit\_soldier = True

                break  # Stop checking after one hit

        if hit\_soldier:

            continue  # Skip Goliath check if soldier blocked it

        #Check if stone hits Goliath

        if gx1 < x1 < gx2 and gy1 < y1 < gy2:

            attack\_sound\_2.play() #play sound when stone collides with Goliath

            goliath\_health -= 1 #Reduce Goliath Health by 1

            canvas.delete(stone) #Remove the stone

            stones.remove(stone)

            #Stop health so it wont go below zero

            goliath\_health = max (0, goliath\_health)

            #-------HEALTH BAR UPDATE

            #Convert current health to a value between 0 and (HEALTH\_IMAGE\_COUNT-1)

            #This lets any number of health points work with a fixed number of images

            ratio = goliath\_health / MAX\_GOLIATH\_HEALTH

            health\_index = int(ratio \* (HEALTH\_IMAGE\_COUNT -1))

            #Prevent health index from going out of range (0 to max index)

            health\_index = max (0, min(HEALTH\_IMAGE\_COUNT -1, health\_index))

            #update the health bar image

            canvas.itemconfig(goliath\_health\_bar, image=goliath\_health\_image[health\_index])

            #Print Goliath Health

            print(f"Goliath health: {goliath\_health} | health bar index: {health\_index}")

            #If Goliath has no health left, display win message

            if goliath\_health <= 0: #health bar

                canvas.create\_text(WIDTH // 2, HEIGHT // 2, text = "David! Wins!", font = ("Ariel", 24), fill = "black")

                #Create restart button

                restart\_btn = tk.Button(window, text = "Play Again", font= ("Arial", 14), command = restart\_game)

                canvas.create\_window (WIDTH // 2, HEIGHT //2 + 40, window = restart\_btn)

                return

#Move goliath back and forth on screen (((YOU ARE HERE)))

def move\_goliath():

    global goliath\_direction, goliath\_frame\_index, goliath\_animation\_counter, goliath\_frames

    #Move left and right

    move\_amount = goliath\_direction \* 6

    canvas.move(goliath, move\_amount, 0)

    gx, gy = canvas.coords(goliath)

    #Small movement range in middle of screen

    left\_limit = WIDTH // 2 - 100

    right\_limit = WIDTH // 2 + 100

   # Change direction and frames at limits

    if gx <= left\_limit:

        goliath\_direction = 1

        goliath\_frames = [goliath\_walk1, goliath\_walk2]  # Facing right

        goliath\_frame\_index = 0

    elif gx >= right\_limit:

        goliath\_direction = -1

        goliath\_frames = [goliath\_image, goliath\_walk]  # Facing left

        goliath\_frame\_index = 0

    #animation handling

    goliath\_animation\_counter += 1

    if goliath\_animation\_counter % 5 == 0: #change frame every 10 ticks

        goliath\_frame\_index = (goliath\_frame\_index + 1) % len(goliath\_frames)

        canvas.itemconfig(goliath, image=goliath\_frames[goliath\_frame\_index])

class Soldier:

    def \_\_init\_\_(self, x, y, direction):

        self.x = x

        self.y = y

        self.direction = direction  # 1 for right, -1 for left

        self.frame\_index = 0

        self.animation\_counter = 0

        if direction == 1:

            self.frames = [soldier\_idle\_right, soldier\_walk\_right]

        else:

            self.frames = [soldier\_idle\_left, soldier\_walk\_left]

        self.image = canvas.create\_image(x, y, image=self.frames[0])

        self.width = self.frames[0].width()

        self.height = self.frames[0].height()

    def move(self):

        self.x += self.direction \* 2

        canvas.move(self.image, self.direction \* 2, 0)

        # Animate

        self.animation\_counter += 1

        if self.animation\_counter % 8 == 0:

            self.frame\_index = (self.frame\_index + 1) % 2

            canvas.itemconfig(self.image, image=self.frames[self.frame\_index])

    def get\_bbox(self):

        return (

            self.x - self.width // 2,

            self.y - self.height // 2,

            self.x + self.width // 2,

            self.y + self.height // 2,

        )

    def destroy(self):

        canvas.delete(self.image)

def spawn\_soldier():

    if len(soldiers) < 3:  # Limit number of soldiers on screen

        x = random.randint(100, 600)

        direction = random.choice([-1, 1])

        y = 260  # Between David and Goliath

        soldier = Soldier(x, y, direction)

        soldiers.append(soldier)

def start\_game():

    canvas.delete("all")  # Clear menu

    pygame.mixer.music.play(-1)  # Restart background music

    restart\_game()  # Start the game loop

#Main game loop: Update all movements

def update\_game():

    move\_david()

    move\_goliath()

    move\_stones()

     # Move all soldiers

    for soldier in soldiers:

        soldier.move()

    # Occasionally spawn new soldier

    if random.randint(1, 50) == 1:  # Adjust spawn rate here

        spawn\_soldier()

     # Remove off-screen soldiers

    for soldier in soldiers[:]:

        if soldier.x < -50 or soldier.x > WIDTH + 50:

            soldier.destroy()

            soldiers.remove(soldier)

    if goliath\_health <= 0: # Keep running until goliath defeated

        show\_win\_screen()

        return

    elif stone\_count <=0 and goliath\_health > 0:

        show\_lose\_screen()

        return

    window.after (50, update\_game) #Repeat after 50 ms

#-----------Key Bindings------------

#Detect when keys are passed

def key\_press(event):

    global david\_speed

    if event.keysym == "Left":

        david\_speed = -10

    elif event.keysym == "Right":

        david\_speed = 10

# Stop movement when keys are released

def key\_release(event):

    global david\_speed

    if event.keysym in ("Left","Right"):

        david\_speed = 0

def show\_menu():

    canvas.delete("all")  # Clear everything

    # Show the menu background

    canvas.create\_image(WIDTH // 2, HEIGHT // 2, image=menu\_bg\_image)

    # Create the play button on top of the background

    play\_button = tk.Button(

        window,

        image=play\_button\_image,

        command=start\_game,

        borderwidth=0,

        highlightthickness=0,

        bg="orange",  # Match menu or make transparent

        activebackground="orange"  # Match on hover

    )

    # Place it centered below the title

    canvas.create\_window(WIDTH // 2, HEIGHT // 2 + 70, window=play\_button)

#Add win and lose screen functions

def show\_win\_screen():

    canvas.delete("all")

    canvas.create\_image(WIDTH // 2, HEIGHT // 2, image=win\_image)

    restart\_btn = tk.Button(window, text="Play Again", font=("Arial, 14"), command=restart\_game)

    canvas.create\_window(WIDTH //2, HEIGHT // 2 + 100, window=restart\_btn)

def show\_lose\_screen():

    canvas.delete("all")

    canvas.create\_image(WIDTH // 2, HEIGHT // 2, image=lose\_image)

    restart\_btn = tk.Button(window, text="Play Again", font=("Arial, 14"), command=restart\_game)

    canvas.create\_window(WIDTH //2, HEIGHT // 2 + 100, window=restart\_btn)

#Bind keys to movement and shooting

window.bind("<KeyPress>", key\_press)

window.bind("<KeyRelease>", key\_release)

window.bind("<space>",shoot\_stone)

# ------- Start the Game -------

pygame.mixer.music.load(background\_music)

pygame.mixer.music.play(-1)  # loops forever

# update\_game()

show\_menu()  # Show menu instead of jumping right into game

window.mainloop()

# -----------------------------------------------------------------------------------------------

# Tower Tactics:

Enemies.py :

import os

from PIL import Image, ImageTk

# Constants

TILE\_SIZE = 32

ENEMY\_SIZE = 96

def load\_sprite\_frames(path, frame\_width, frame\_height, num\_frames):

    """Split a sprite sheet into individual frames"""

    sprite\_sheet = Image.open(path)

    frames = []

    frames\_flipped = []

    for i in range(num\_frames):

        frame = sprite\_sheet.crop((i \* frame\_width, 0, (i + 1) \* frame\_width, frame\_height))

        frame = frame.resize((ENEMY\_SIZE, ENEMY\_SIZE), Image.Resampling.NEAREST)

        frames.append(ImageTk.PhotoImage(frame))

        frame\_flipped = frame.transpose(Image.FLIP\_LEFT\_RIGHT)

        frames\_flipped.append(ImageTk.PhotoImage(frame\_flipped))

    return frames, frames\_flipped

# Asset directory

ASSET\_DIR = os.path.join(os.path.dirname(\_\_file\_\_), "assets")

ENEMY\_DIR = os.path.join(ASSET\_DIR, "Tiny Swords (Enemy Pack)", "Enemy Pack")

# Enemy sprite storage (loaded after Tkinter window is created)

paddle\_fish\_idle = None

paddle\_fish\_idle\_flipped = None

paddle\_fish\_run = None

paddle\_fish\_run\_flipped = None

paddle\_fish\_attack = None

paddle\_fish\_attack\_flipped = None

troll\_idle = None

troll\_idle\_flipped = None

troll\_walk = None

troll\_walk\_flipped = None

troll\_attack = None

troll\_attack\_flipped = None

troll\_dead = None

troll\_dead\_flipped = None

def load\_enemy\_sprites():

    """Load all enemy sprites - must be called after Tkinter root is created"""

    global paddle\_fish\_idle, paddle\_fish\_idle\_flipped

    global paddle\_fish\_run, paddle\_fish\_run\_flipped

    global paddle\_fish\_attack, paddle\_fish\_attack\_flipped

    global troll\_idle, troll\_idle\_flipped

    global troll\_walk, troll\_walk\_flipped

    global troll\_attack, troll\_attack\_flipped

    global troll\_dead, troll\_dead\_flipped

    # Load Paddle Fish sprites - Idle: 8 frames (1536x192), Run: 6 frames (1152x192), Attack: 6 frames (1152x192)

    paddle\_fish\_idle, paddle\_fish\_idle\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Paddle Fish", "PaddleFish\_Idle.png"), 192, 192, 8

    )

    paddle\_fish\_run, paddle\_fish\_run\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Paddle Fish", "PaddleFish\_Run.png"), 192, 192, 6

    )

    paddle\_fish\_attack, paddle\_fish\_attack\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Paddle Fish", "PaddleFish\_Attack.png"), 192, 192, 6

    )

    # Load Troll sprites - Idle: 12 frames (4608x384), Walk: 10 frames (3840x384), Attack: 6 frames (2304x384), Dead: 10 frames (3840x384)

    troll\_idle, troll\_idle\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Troll", "Troll\_Idle.png"), 384, 384, 12

    )

    troll\_walk, troll\_walk\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Troll", "Troll\_Walk.png"), 384, 384, 10

    )

    troll\_attack, troll\_attack\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Troll", "Troll\_Attack.png"), 384, 384, 6

    )

    troll\_dead, troll\_dead\_flipped = load\_sprite\_frames(

        os.path.join(ENEMY\_DIR, "Troll", "Troll\_Dead.png"), 384, 384, 10

    )

class Enemy:

    def \_\_init\_\_(self, canvas, x, y, platform, enemy\_type='paddle\_fish', tile\_col=None, tile\_row=None):

        self.canvas = canvas

        self.x = x

        self.y = y

        self.platform = platform

        self.enemy\_type = enemy\_type

        self.state = 'idle'

        self.current\_frame = 0

        self.frame\_counter = 0

        self.facing\_right = False

        self.speed = 1  # Slower movement speed

        self.health = 3  # Enemy health

        self.is\_dead = False

        self.death\_animation\_complete = False

        self.hit\_flash = False

        self.hit\_flash\_timer = 0

        self.tile\_col = tile\_col

        self.tile\_row = tile\_row

        self.tile\_bounds = None

        # Get tile bounds if tile position specified

        if tile\_col is not None and tile\_row is not None:

            self.tile\_bounds = platform.get\_tile\_area(tile\_col, tile\_row)

        # Load appropriate sprites based on enemy type

        if enemy\_type == 'paddle\_fish':

            self.idle\_frames = paddle\_fish\_idle

            self.idle\_frames\_flipped = paddle\_fish\_idle\_flipped

            self.run\_frames = paddle\_fish\_run

            self.run\_frames\_flipped = paddle\_fish\_run\_flipped

            self.attack\_frames = paddle\_fish\_attack

            self.attack\_frames\_flipped = paddle\_fish\_attack\_flipped

            self.dead\_frames = None

            self.dead\_frames\_flipped = None

        elif enemy\_type == 'troll':

            self.idle\_frames = troll\_idle

            self.idle\_frames\_flipped = troll\_idle\_flipped

            self.run\_frames = troll\_walk

            self.run\_frames\_flipped = troll\_walk\_flipped

            self.attack\_frames = troll\_attack

            self.attack\_frames\_flipped = troll\_attack\_flipped

            self.dead\_frames = troll\_dead

            self.dead\_frames\_flipped = troll\_dead\_flipped

        self.sprite = canvas.create\_image(x, y, anchor="center", image=self.idle\_frames[0])

    def take\_damage(self, damage=1):

        """Enemy takes damage"""

        if not self.is\_dead:

            self.health -= damage

            self.hit\_flash = True

            self.hit\_flash\_timer = 10  # Flash for 10 frames

            if self.health <= 0:

                self.health = 0

                self.is\_dead = True

                self.state = 'dead'

                self.current\_frame = 0

                return True  # Enemy died

        return False

    def update(self):

        """Update enemy AI and animation"""

        # Handle hit flash effect

        if self.hit\_flash:

            self.hit\_flash\_timer -= 1

            # Toggle visibility every 2 frames for flashing effect

            if self.hit\_flash\_timer % 4 < 2:

                self.canvas.itemconfig(self.sprite, state='hidden')

            else:

                self.canvas.itemconfig(self.sprite, state='normal')

            if self.hit\_flash\_timer <= 0:

                self.hit\_flash = False

                self.canvas.itemconfig(self.sprite, state='normal')

        # If dead, only play death animation once

        if self.is\_dead:

            if not self.death\_animation\_complete and self.dead\_frames:

                self.frame\_counter += 1

                if self.frame\_counter >= 6:

                    self.frame\_counter = 0

                    if self.current\_frame < len(self.dead\_frames) - 1:

                        self.current\_frame += 1

                        frame = self.dead\_frames\_flipped[self.current\_frame] if not self.facing\_right else self.dead\_frames[self.current\_frame]

                        self.canvas.itemconfig(self.sprite, image=frame)

                    else:

                        self.death\_animation\_complete = True

            return  # Don't move or do anything else when dead

        # Simple patrol behavior - move back and forth on platform

        if self.state == 'idle':

            self.state = 'run'

        if self.state == 'run':

            # Move in current direction

            if self.facing\_right:

                new\_x = self.x + self.speed

            else:

                new\_x = self.x - self.speed

            # Use tile bounds if specified, otherwise use platform bounds

            if self.tile\_bounds:

                # Bounce at tile edges

                if new\_x < self.tile\_bounds['left']:

                    new\_x = self.tile\_bounds['left']

                    self.facing\_right = True

                elif new\_x > self.tile\_bounds['right']:

                    new\_x = self.tile\_bounds['right']

                    self.facing\_right = False

            else:

                # Check platform boundaries (same insets as player)

                inset\_left = TILE\_SIZE \* 0.75

                inset\_right = TILE\_SIZE \* 1.25

                platform\_left = self.platform.rect\_x + inset\_left

                platform\_right = self.platform.rect\_x + self.platform.rect\_width - inset\_right

                # Bounce at edges

                if new\_x < platform\_left:

                    new\_x = platform\_left

                    self.facing\_right = True

                elif new\_x > platform\_right:

                    new\_x = platform\_right

                    self.facing\_right = False

            self.x = new\_x

            self.canvas.coords(self.sprite, self.x, self.y)

        # Update animation

        self.frame\_counter += 1

        if self.frame\_counter >= 6:

            self.frame\_counter = 0

            if self.state == 'idle':

                self.current\_frame = (self.current\_frame + 1) % len(self.idle\_frames)

                frame = self.idle\_frames\_flipped[self.current\_frame] if not self.facing\_right else self.idle\_frames[self.current\_frame]

            elif self.state == 'run':

                self.current\_frame = (self.current\_frame + 1) % len(self.run\_frames)

                frame = self.run\_frames\_flipped[self.current\_frame] if not self.facing\_right else self.run\_frames[self.current\_frame]

            self.canvas.itemconfig(self.sprite, image=frame)

levels.py :

# Level configuration for Tower Tactics

LEVELS = [

    {

        'id': 1,

        'name': 'Training Grounds',

        'platform\_width': 400,

        'platform\_height': 128,

        'enemies': [

            {'type': 'troll', 'position': 'center'}

        ],

        'terrain\_color': 'color1',

        'terrain\_layout': None,  # Simple flat platform

        'player\_start': {'tile\_col': 6, 'tile\_row': 1}  # Center of platform

    },

    {

        'id': 1.5,

        'name': 'Mountain Steps',

        'platform\_width': 512,

        'platform\_height': 224,  # 7 rows \* 32px = 224px

        'enemies': [

            {'type': 'troll', 'tile\_col': 3, 'tile\_row': 1},  # Left raised platform

            {'type': 'troll', 'tile\_col': 11, 'tile\_row': 1}   # Right raised platform

        ],

        'player\_start': {'tile\_col': 1, 'tile\_row': 4},  # Base level near left edge

        'terrain\_color': 'color2',

        'terrain\_layout': [

            # Multi-level platform with lifted terrain and ramps

            # Each entry: (col, row, tile\_type)

            # Platform height is 224px = 7 rows of 32px tiles (rows 0-6)

            # LOWEST LEVEL - Complete base (rows 3-5, 3 rows tall)

            # Top row of base

            (0, 3, 'top\_corner\_left'),

            (1, 3, 'top\_middle'),

            (2, 3, 'top\_middle'),

            (3, 3, 'top\_middle'),

            (4, 3, 'top\_middle'),

            (5, 3, 'top\_middle'),

            (6, 3, 'top\_middle'),

            (7, 3, 'top\_middle'),

            (8, 3, 'top\_middle'),

            (9, 3, 'top\_middle'),

            (10, 3, 'top\_middle'),

            (11, 3, 'top\_middle'),

            (12, 3, 'top\_middle'),

            (13, 3, 'top\_middle'),

            (14, 3, 'top\_middle'),

            (15, 3, 'top\_corner\_right'),

            # Middle row

            (0, 4, 'side\_left'),

            (1, 4, 'middle'),

            (2, 4, 'middle'),

            (3, 4, 'middle'),

            (4, 4, 'middle'),

            (5, 4, 'middle'),

            (6, 4, 'middle'),

            (7, 4, 'middle'),

            (8, 4, 'middle'),

            (9, 4, 'middle'),

            (10, 4, 'middle'),

            (11, 4, 'middle'),

            (12, 4, 'middle'),

            (13, 4, 'middle'),

            (14, 4, 'middle'),

            (15, 4, 'side\_right'),

            # Middle row

            (0, 5, 'side\_left'),

            (1, 5, 'middle'),

            (2, 5, 'middle'),

            (3, 5, 'middle'),

            (4, 5, 'middle'),

            (5, 5, 'middle'),

            (6, 5, 'middle'),

            (7, 5, 'middle'),

            (8, 5, 'middle'),

            (9, 5, 'middle'),

            (10, 5, 'middle'),

            (11, 5, 'middle'),

            (12, 5, 'middle'),

            (13, 5, 'middle'),

            (14, 5, 'middle'),

            (15, 5, 'side\_right'),

            # Bottom row (front edge)

            (0, 6, 'bottom\_corner\_left'),

            (1, 6, 'bottom\_middle'),

            (2, 6, 'bottom\_middle'),

            (3, 6, 'bottom\_middle'),

            (4, 6, 'bottom\_middle'),

            (5, 6, 'bottom\_middle'),

            (6, 6, 'bottom\_middle'),

            (7, 6, 'bottom\_middle'),

            (8, 6, 'bottom\_middle'),

            (9, 6, 'bottom\_middle'),

            (10, 6, 'bottom\_middle'),

            (11, 6, 'bottom\_middle'),

            (12, 6, 'bottom\_middle'),

            (13, 6, 'bottom\_middle'),

            (14, 6, 'bottom\_middle'),

            (15, 6, 'bottom\_corner\_right'),

            # RAMP TO LEFT PLATFORM - column 1 (ramp up from base to raised platform)

            (1, 2, 'ramp\_left\_top'),      # Top part of ramp

            (1, 3, 'ramp\_left\_bottom'),   # Bottom part of ramp on base level

            # RAISED PLATFORM 1 - Left side platform (columns 2-5, rows 1-2)

            # Top surface

            (2, 1, 'top\_corner\_left'),

            (3, 1, 'top\_middle'),

            (4, 1, 'top\_middle'),

            (5, 1, 'top\_corner\_right'),

            # Bottom layer - use middle for ramp connections

            (2, 2, 'middle'),  # Connects to ramp at (1, 2)

            (3, 2, 'middle'),

            (4, 2, 'middle'),

            (5, 2, 'middle'),

            # Lifted edges showing on base level

            (2, 3, 'lifted\_left'),

            (3, 3, 'lifted\_middle'),

            (4, 3, 'lifted\_middle'),

            (5, 3, 'lifted\_right'),

            # RAMP DOWN FROM LEFT PLATFORM - column 6

            (6, 2, 'ramp\_right\_top'),     # Top part on raised platform

            (6, 3, 'ramp\_right\_bottom'),  # Bottom part extends down

            # CENTER WALKWAY - columns 7-8, row 3 (on base level)

            (7, 3, 'middle'),

            (8, 3, 'middle'),

            # RAMP UP TO RIGHT PLATFORM - column 9

            (9, 2, 'ramp\_left\_top'),      # Top part of ramp

            (9, 3, 'ramp\_left\_bottom'),   # Bottom part on base level

            # RAISED PLATFORM 2 - Right side platform (columns 10-13, rows 1-2)

            # Top surface

            (10, 1, 'top\_corner\_left'),

            (11, 1, 'top\_middle'),

            (12, 1, 'top\_middle'),

            (13, 1, 'top\_corner\_right'),

            # Bottom layer - use middle for ramp connections

            (10, 2, 'middle'),  # Connects to ramp at (9, 2)

            (11, 2, 'middle'),

            (12, 2, 'middle'),

            (13, 2, 'middle'),

            # Lifted edges showing on base level

            (10, 3, 'lifted\_left'),

            (11, 3, 'lifted\_middle'),

            (12, 3, 'lifted\_middle'),

            (13, 3, 'lifted\_right'),

            # RAMP DOWN FROM RIGHT PLATFORM - column 14

            (14, 2, 'ramp\_right\_top'),    # Top part on raised platform

            (14, 3, 'ramp\_right\_bottom'), # Bottom part extends down

        ]

    },

    {

        'id': 2,

        'name': 'Forest Edge',

        'platform\_width': 500,

        'platform\_height': 160,

        'enemies': [

            {'type': 'troll', 'position': 'left'},

            {'type': 'troll', 'position': 'right'}

        ],

        'terrain\_color': 'color2'

    },

    {

        'id': 3,

        'name': 'Mountain Path',

        'platform\_width': 450,

        'platform\_height': 128,

        'enemies': [

            {'type': 'troll', 'position': 'left'},

            {'type': 'troll', 'position': 'center'},

            {'type': 'troll', 'position': 'right'}

        ],

        'terrain\_color': 'color3'

    },

    {

        'id': 4,

        'name': 'Dark Woods',

        'platform\_width': 550,

        'platform\_height': 192,

        'enemies': [

            {'type': 'troll', 'position': 'left'},

            {'type': 'troll', 'position': 'center-left'},

            {'type': 'troll', 'position': 'center-right'},

            {'type': 'troll', 'position': 'right'}

        ],

        'terrain\_color': 'color1'

    },

    {

        'id': 5,

        'name': 'Troll Kingdom',

        'platform\_width': 600,

        'platform\_height': 192,

        'enemies': [

            {'type': 'troll', 'position': 'far-left'},

            {'type': 'troll', 'position': 'left'},

            {'type': 'troll', 'position': 'center'},

            {'type': 'troll', 'position': 'right'},

            {'type': 'troll', 'position': 'far-right'}

        ],

        'terrain\_color': 'color2'

    }

]

def get\_enemy\_x\_position(position, platform\_x, platform\_width):

    """Calculate enemy X position based on position name"""

    positions = {

        'far-left': platform\_x + 80,

        'left': platform\_x + platform\_width \* 0.25,

        'center-left': platform\_x + platform\_width \* 0.33,

        'center': platform\_x + platform\_width \* 0.5,

        'center-right': platform\_x + platform\_width \* 0.66,

        'right': platform\_x + platform\_width \* 0.75,

        'far-right': platform\_x + platform\_width - 80

    }

    return positions.get(position, platform\_x + platform\_width \* 0.5)

main.py:

import tkinter as tk

import random

import os

from PIL import Image, ImageTk

from enemies import Enemy, load\_enemy\_sprites

from levels import LEVELS, get\_enemy\_x\_position

# Constants

WIDTH, HEIGHT = 800, 600

PLAYER\_SIZE = 96  # Increased from 48 to 96 (200%)

TILE\_SIZE = 32

MOVE\_SPEED = 4

SCROLL\_THRESHOLD = HEIGHT // 3

# Asset directory

ASSET\_DIR = os.path.join(os.path.dirname(\_\_file\_\_), "assets")

SPRITE\_DIR = os.path.join(ASSET\_DIR, "Tiny Swords (Free Pack)")

# Setup

root = tk.Tk()

root.title("Tower Tactics")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="#7ec4a8")

canvas.pack()

# Load enemy sprites after Tkinter window is created

load\_enemy\_sprites()

# Load heart sprite for health display (5 frames: full, 3/4, half, 1/4, empty)

heart\_img = Image.open(os.path.join(ASSET\_DIR, "heart\_animated\_1.png"))

heart\_frames = []

for i in range(5):

    frame = heart\_img.crop((i \* 17, 0, (i + 1) \* 17, 17))

    frame = frame.resize((34, 34), Image.Resampling.NEAREST)  # 2x scale

    heart\_frames.append(ImageTk.PhotoImage(frame))

# Load and prepare images

def load\_sprite\_frames(path, frame\_width, frame\_height, num\_frames):

    """Split a sprite sheet into individual frames"""

    sprite\_sheet = Image.open(path)

    frames = []

    frames\_flipped = []

    for i in range(num\_frames):

        # Extract frame

        frame = sprite\_sheet.crop((i \* frame\_width, 0, (i + 1) \* frame\_width, frame\_height))

        # Resize to player size

        frame = frame.resize((PLAYER\_SIZE, PLAYER\_SIZE), Image.Resampling.NEAREST)

        frames.append(ImageTk.PhotoImage(frame))

        # Create flipped version

        frame\_flipped = frame.transpose(Image.FLIP\_LEFT\_RIGHT)

        frames\_flipped.append(ImageTk.PhotoImage(frame\_flipped))

    return frames, frames\_flipped

# Load player sprite frames (192x192 per frame)

player\_idle\_frames, player\_idle\_frames\_left = load\_sprite\_frames(

    os.path.join(SPRITE\_DIR, "Units/Blue Units/Warrior/Warrior\_Idle.png"), 192, 192, 6)

player\_run\_frames, player\_run\_frames\_left = load\_sprite\_frames(

    os.path.join(SPRITE\_DIR, "Units/Blue Units/Warrior/Warrior\_Run.png"), 192, 192, 6)

player\_attack1\_frames, player\_attack1\_frames\_left = load\_sprite\_frames(

    os.path.join(SPRITE\_DIR, "Units/Blue Units/Warrior/Warrior\_Attack1.png"), 192, 192, 4)

player\_attack2\_frames, player\_attack2\_frames\_left = load\_sprite\_frames(

    os.path.join(SPRITE\_DIR, "Units/Blue Units/Warrior/Warrior\_Attack2.png"), 192, 192, 4)

player\_guard\_frames, player\_guard\_frames\_left = load\_sprite\_frames(

    os.path.join(SPRITE\_DIR, "Units/Blue Units/Warrior/Warrior\_Guard.png"), 192, 192, 6)

# Load individual terrain tiles

TERRAIN\_DIR = os.path.join(ASSET\_DIR, "Terrain")

def load\_terrain\_tile(filename, scale\_size=TILE\_SIZE):

    """Load a single terrain tile and scale it"""

    img = Image.open(os.path.join(TERRAIN\_DIR, filename))

    img = img.resize((scale\_size, scale\_size), Image.Resampling.NEAREST)

    return ImageTk.PhotoImage(img)

# Organize terrain tiles by color/type

terrain\_tiles = {

    'color1': {  # Grass/green terrain

        'top\_solo': load\_terrain\_tile('Color1\_Top\_Solo.png'),

        'top\_middle': load\_terrain\_tile('Color1\_Top\_Middle.png'),

        'top\_corner\_left': load\_terrain\_tile('Color1\_Top\_Corner\_Left.png'),

        'top\_corner\_right': load\_terrain\_tile('Color1\_Top\_Corner\_Right.png'),

        'middle': load\_terrain\_tile('Color1\_Middle.png'),

        'middle\_horizontal\_solo': load\_terrain\_tile('Color1\_Middle\_Horizontal\_Solo.png'),

        'middle\_vertical\_solo': load\_terrain\_tile('Color1\_Middle\_Verticle\_Solo.png'),

        'bottom\_solo': load\_terrain\_tile('Color1\_Bottom\_Solo.png'),

        'bottom\_middle': load\_terrain\_tile('Color1\_Bottom\_Middle.png'),

        'bottom\_corner\_left': load\_terrain\_tile('Color1\_Bottom\_Corner\_Left.png'),

        'bottom\_corner\_right': load\_terrain\_tile('Color1\_Bottom\_Corner\_Right.png'),

        'left\_solo': load\_terrain\_tile('Color1\_Left\_Solo.png'),

        'right\_solo': load\_terrain\_tile('Color1\_Right\_Solo.png'),

        'side\_left': load\_terrain\_tile('Color1\_Side\_Left.png'),

        'side\_right': load\_terrain\_tile('Color1\_Side\_Right.png'),

        'only\_solo': load\_terrain\_tile('Color1\_Only\_Solo.png'),

        'ramp\_left\_top': load\_terrain\_tile('Color1\_Ramp\_Left\_Top.png'),

        'ramp\_left\_bottom': load\_terrain\_tile('Color1\_Ramp\_Left\_Bottom.png'),

        'ramp\_right\_top': load\_terrain\_tile('Color1\_Ramp\_Right\_Top.png'),

        'ramp\_right\_bottom': load\_terrain\_tile('Color1\_Ramp\_Right\_Bottom.png'),

        'lifted\_left': load\_terrain\_tile('Lifted\_Left.png'),

        'lifted\_middle': load\_terrain\_tile('Lifted\_Middle.png'),

        'lifted\_right': load\_terrain\_tile('Lifted\_Right.png'),

    },

    'color2': {  # Stone/gray terrain

        'top\_solo': load\_terrain\_tile('Color2\_Top\_Solo.png'),

        'top\_middle': load\_terrain\_tile('Color2\_Top\_Middle.png'),

        'top\_corner\_left': load\_terrain\_tile('Color2\_Top\_Corner\_Left.png'),

        'top\_corner\_right': load\_terrain\_tile('Color2\_Top\_Corner\_Right.png'),

        'middle': load\_terrain\_tile('Color2\_Middle.png'),

        'middle\_horizontal\_solo': load\_terrain\_tile('Color2\_Middle\_Horizontal\_Solo.png'),

        'middle\_vertical\_solo': load\_terrain\_tile('Color2\_Middle\_Verticle\_Solo.png'),

        'bottom\_solo': load\_terrain\_tile('Color2\_Bottom\_Solo.png'),

        'bottom\_middle': load\_terrain\_tile('Color2\_Bottom\_Middle.png'),

        'bottom\_corner\_left': load\_terrain\_tile('Color2\_Bottom\_Corner\_Left.png'),

        'bottom\_corner\_right': load\_terrain\_tile('Color2\_Bottom\_Corner\_Right.png'),

        'left\_solo': load\_terrain\_tile('Color2\_Left\_Solo.png'),

        'right\_solo': load\_terrain\_tile('Color2\_Right\_Solo.png'),

        'side\_left': load\_terrain\_tile('Color2\_Side\_Left.png'),

        'side\_right': load\_terrain\_tile('Color2\_Side\_Right.png'),

        'only\_solo': load\_terrain\_tile('Color2\_Only\_Solo.png'),

        'ramp\_left\_top': load\_terrain\_tile('Color2\_Ramp\_Left\_Top.png'),

        'ramp\_left\_bottom': load\_terrain\_tile('Color2\_Ramp\_Left\_Bottom.png'),

        'ramp\_right\_top': load\_terrain\_tile('Color2\_Ramp\_Right\_Top.png'),

        'ramp\_right\_bottom': load\_terrain\_tile('Color2\_Ramp\_Right\_Bottom.png'),

        'lifted\_left': load\_terrain\_tile('Lifted\_Left.png'),

        'lifted\_middle': load\_terrain\_tile('Lifted\_Middle.png'),

        'lifted\_right': load\_terrain\_tile('Lifted\_Right.png'),

    },

    'color3': {  # Dark/purple terrain

        'top\_solo': load\_terrain\_tile('Color3\_Top\_Solo.png'),

        'top\_middle': load\_terrain\_tile('Color3\_Top\_Middle.png'),

        'top\_corner\_left': load\_terrain\_tile('Color3\_Top\_Corner\_Left.png'),

        'top\_corner\_right': load\_terrain\_tile('Color3\_Top\_Corner\_Right.png'),

        'middle': load\_terrain\_tile('Color3\_Middle.png'),

        'middle\_horizontal\_solo': load\_terrain\_tile('Color3\_Middle\_Horizontal\_Solo.png'),

        'middle\_vertical\_solo': load\_terrain\_tile('Color3\_Middle\_Verticle\_Solo.png'),

        'bottom\_solo': load\_terrain\_tile('Color3\_Bottom\_Solo.png'),

        'bottom\_middle': load\_terrain\_tile('Color3\_Bottom\_Middle.png'),

        'bottom\_corner\_left': load\_terrain\_tile('Color3\_Bottom\_Corner\_Left.png'),

        'bottom\_corner\_right': load\_terrain\_tile('Color3\_Bottom\_Corner\_Right.png'),

        'left\_solo': load\_terrain\_tile('Color3\_Left\_Solo.png'),

        'right\_solo': load\_terrain\_tile('Color3\_Right\_Solo.png'),

        'side\_left': load\_terrain\_tile('Color3\_Side\_Left.png'),

        'side\_right': load\_terrain\_tile('Color3\_Side\_Right.png'),

        'only\_solo': load\_terrain\_tile('Color3\_Only\_Solo.png'),

        'ramp\_left\_top': load\_terrain\_tile('Color3\_Ramp\_Left\_Top.png'),

        'ramp\_left\_bottom': load\_terrain\_tile('Color3\_Ramp\_Left\_Bottom.png'),

        'ramp\_right\_top': load\_terrain\_tile('Color3\_Ramp\_Right\_Top.png'),

        'ramp\_right\_bottom': load\_terrain\_tile('Color3\_Ramp\_Right\_Bottom.png'),

        'lifted\_left': load\_terrain\_tile('Lifted\_Left.png'),

        'lifted\_middle': load\_terrain\_tile('Lifted\_Middle.png'),

        'lifted\_right': load\_terrain\_tile('Lifted\_Right.png'),

    },

    'lifted': {  # Floating platform tiles

        'left': load\_terrain\_tile('Lifted\_Left.png'),

        'middle': load\_terrain\_tile('Lifted\_Middle.png'),

        'right': load\_terrain\_tile('Lifted\_Right.png'),

        'solo': load\_terrain\_tile('Lifted\_Solo.png'),

    },

}

# Load water animation frames (all 12 frames for each type)

def load\_water\_animations():

    """Load all 12 frames for each water animation type"""

    water\_animations = {}

    for i in range(1, 4):  # Load water #1, #2, #3

        water\_img = Image.open(os.path.join(TERRAIN\_DIR, f"Water\_FlatGround\_#{i}\_(12frames).png"))

        frames = []

        # Extract all 12 frames (64x64 each)

        for frame\_num in range(12):

            frame = water\_img.crop((frame\_num \* 64, 0, (frame\_num + 1) \* 64, 64))

            # Scale to tile size

            frame = frame.resize((TILE\_SIZE, TILE\_SIZE), Image.Resampling.NEAREST)

            frames.append(ImageTk.PhotoImage(frame))

        water\_animations[i] = frames

    return water\_animations

water\_animations = load\_water\_animations()

# Load decorations

def load\_decoration(filename, scale=None, frame\_width=None, frame\_height=None):

    """Load decoration, extracting first frame if it's a sprite sheet"""

    img = Image.open(os.path.join(SPRITE\_DIR, "Decorations", filename))

    # If frame\_width is specified, extract first frame from sprite sheet

    if frame\_width and frame\_height:

        # Crop to get first frame and create a new image to avoid lazy evaluation

        cropped = img.crop((0, 0, frame\_width, frame\_height))

        # Create a new image with the same mode and copy the data

        img = Image.new(cropped.mode, cropped.size)

        img.paste(cropped, (0, 0))

    if scale:

        img = img.resize(scale, Image.Resampling.NEAREST)

    return ImageTk.PhotoImage(img)

# Load terrain trees (single frame images from Terrain folder)

TERRAIN\_DIR = os.path.join(SPRITE\_DIR, "Terrain")

def load\_terrain\_tree(filename, scale\_size):

    """Load tree from Terrain folder (not sprite sheets)"""

    img = Image.open(os.path.join(ASSET\_DIR, "Terrain", filename))

    img = img.resize(scale\_size, Image.Resampling.NEAREST)

    return ImageTk.PhotoImage(img)

# Load decoration assets

decorations = {

    'tree1': load\_terrain\_tree("Tree1.png", (64, 96)),  # 128x192 scaled to 64x96

    'tree2': load\_terrain\_tree("Tree2.png", (64, 122)),  # 128x243 scaled proportionally

    'tree3': load\_terrain\_tree("Tree3.png", (64, 84)),   # 128x168 scaled proportionally

    'tree4': load\_terrain\_tree("Tree4.png", (64, 64)),   # 128x128 scaled to 64x64

    'rock1': load\_decoration("Rocks/Rock1.png", (32, 32)),

    'rock2': load\_decoration("Rocks/Rock2.png", (32, 32)),

    'rock3': load\_decoration("Rocks/Rock3.png", (32, 32)),

    'rock4': load\_decoration("Rocks/Rock4.png", (32, 32)),

    'bush1': load\_decoration("Bushes/Bushe1.png", (32, 32), frame\_width=128, frame\_height=128),  # 8 frames, extract first

    'bush2': load\_decoration("Bushes/Bushe2.png", (32, 32), frame\_width=128, frame\_height=128),  # 8 frames, extract first

}

# Score setup

score = 0

score\_text = canvas.create\_text(10, 10, anchor="nw", font=("Arial", 16), fill="white", text=f"Height: {score}")

def restart\_game():

    canvas.delete("all")

    game.player = None  # Force player to be recreated

    game.load\_level(game.current\_level\_index)

class Player:

    def \_\_init\_\_(self, canvas, x, y):

        self.canvas = canvas

        self.x = x

        self.y = y

        self.vx = 0

        self.vy = 0

        self.facing\_right = True

        self.moving = False

        self.current\_frame = 0

        self.frame\_counter = 0

        self.animation\_speed = 8

        self.state = 'idle'

        self.animation\_lock = False

        self.current\_platform = None

        self.health = 3

        self.invulnerable = False

        self.invulnerable\_timer = 0

        self.id = canvas.create\_image(x, y, anchor="center", image=player\_idle\_frames[0])

    def take\_damage(self):

        if not self.invulnerable and self.health > 0:

            self.health -= 1

            self.invulnerable = True

            self.invulnerable\_timer = 60

            if self.health <= 0:

                self.handle\_player\_death()

                return True

        return False

    def handle\_player\_death(self):

        self.canvas.create\_text(WIDTH // 2, HEIGHT // 2 - 40, text="Game Over", font=("Arial", 32), fill="red")

        retry\_button = tk.Button(self.canvas.master, text="Retry", font=("Arial", 16), command=restart\_game)

        self.canvas.create\_window(WIDTH // 2, HEIGHT // 2 + 20, window=retry\_button)

    def update(self, platforms):

        if not self.animation\_lock:

            intended\_x = self.x + self.vx

            intended\_y = self.y + self.vy

            buffer = 4

            def is\_colliding\_with\_solid(x, y, solids):

                for obj in solids:

                    if abs(x - obj['x']) < obj['width'] // 2 and abs(y - obj['y']) < obj['height'] // 2:

                        return True

                return False

            if self.current\_platform and is\_colliding\_with\_solid(intended\_x, intended\_y, self.current\_platform.solid\_objects):

                self.vx = 0

                self.vy = 0

                intended\_x = self.x

                intended\_y = self.y

            if self.current\_platform and self.current\_platform.tile\_map:

                next\_tile = self.current\_platform.get\_tile\_at\_position(intended\_x, intended\_y)

                if next\_tile:

                    self.x = intended\_x

                    self.y = intended\_y

                else:

                    if abs(self.vx) < buffer and abs(self.vy) < buffer:

                        self.x = intended\_x

                        self.y = intended\_y

                    else:

                        self.vx = 0

                        self.vy = 0

            else:

                self.x = max(0, min(intended\_x, WIDTH))

                self.y = max(0, min(intended\_y, HEIGHT))

        if self.invulnerable:

            self.invulnerable\_timer -= 1

            if self.invulnerable\_timer <= 0:

                self.invulnerable = False

        self.update\_sprite()

        self.canvas.coords(self.id, self.x, self.y)

    def update\_sprite(self):

        is\_moving = self.vx != 0 or self.vy != 0

        if not self.animation\_lock:

            self.state = 'run' if is\_moving else 'idle'

        if self.state == 'idle':

            frames = player\_idle\_frames if self.facing\_right else player\_idle\_frames\_left

            max\_frames = 6

        elif self.state == 'run':

            frames = player\_run\_frames if self.facing\_right else player\_run\_frames\_left

            max\_frames = 6

        elif self.state == 'attack1':

            frames = player\_attack1\_frames if self.facing\_right else player\_attack1\_frames\_left

            max\_frames = 4

        elif self.state == 'attack2':

            frames = player\_attack2\_frames if self.facing\_right else player\_attack2\_frames\_left

            max\_frames = 4

        elif self.state == 'guard':

            frames = player\_guard\_frames if self.facing\_right else player\_guard\_frames\_left

            max\_frames = 6

        else:

            frames = player\_idle\_frames if self.facing\_right else player\_idle\_frames\_left

            max\_frames = 6

        self.frame\_counter += 1

        speed = 10 if self.state in ['attack1', 'attack2'] else self.animation\_speed

        if self.frame\_counter >= speed:

            self.frame\_counter = 0

            self.current\_frame = (self.current\_frame + 1) % max\_frames

            if self.animation\_lock and self.current\_frame == 0:

                self.animation\_lock = False

                self.state = 'idle'

        self.canvas.itemconfig(self.id, image=frames[self.current\_frame])

    def set\_velocity(self, vx, vy):

        self.vx = vx

        self.vy = vy

        if vx > 0:

            self.facing\_right = True

        elif vx < 0:

            self.facing\_right = False

    def set\_platform(self, platform):

        self.current\_platform = platform

    def attack1(self):

        if not self.animation\_lock:

            self.state = 'attack1'

            self.animation\_lock = True

            self.current\_frame = 0

            self.frame\_counter = 0

            return True

        return False

    def attack2(self):

        if not self.animation\_lock:

            self.state = 'attack2'

            self.animation\_lock = True

            self.current\_frame = 0

            self.frame\_counter = 0

    def guard(self):

        if not self.animation\_lock:

            self.state = 'guard'

            self.animation\_lock = True

            self.current\_frame = 0

            self.frame\_counter = 0

class Platform:

    def \_\_init\_\_(self, canvas, x, y, width, height, terrain\_color=None, terrain\_layout=None):

        """Create a rectangular contained platform"""

        self.canvas = canvas

        self.rect\_x = x

        self.rect\_y = y

        self.rect\_width = width

        self.rect\_height = height

        self.tiles = []

        self.solid\_objects = []  # Track solid decorations like trees

        self.decorations = []

        self.water\_tiles = []

        self.terrain\_layout = terrain\_layout

        self.tile\_map = {}  # Track walkable tiles by (col, row) -> bounds

        # Use specified terrain type or choose random

        self.terrain\_type = terrain\_color if terrain\_color else random.choice(['color1', 'color2', 'color3'])

        if terrain\_layout:

            self.create\_custom\_platform()

            self.add\_water\_below()  # Add water for custom platforms too

        else:

            self.create\_platform()

            self.add\_water\_below()

            self.add\_trees\_at\_edges()

    def create\_platform(self):

        """Create a rectangular platform filled with tiles"""

        tileset = terrain\_tiles[self.terrain\_type]

        tiles\_wide = self.rect\_width // TILE\_SIZE

        tiles\_tall = self.rect\_height // TILE\_SIZE

        for row in range(tiles\_tall):

            for col in range(tiles\_wide):

                x = self.rect\_x + col \* TILE\_SIZE + TILE\_SIZE // 2

                y = self.rect\_y + row \* TILE\_SIZE + TILE\_SIZE // 2

                # Determine which tile to use based on position

                is\_top = row == 0

                is\_bottom = row == tiles\_tall - 1

                is\_left = col == 0

                is\_right = col == tiles\_wide - 1

                # Select appropriate tile

                if is\_top and is\_left:

                    tile\_img = tileset['top\_corner\_left']

                    tile\_type = 'top\_corner\_left'

                elif is\_top and is\_right:

                    tile\_img = tileset['top\_corner\_right']

                    tile\_type = 'top\_corner\_right'

                elif is\_bottom and is\_left:

                    tile\_img = tileset['bottom\_corner\_left']

                    tile\_type = 'bottom\_corner\_left'

                elif is\_bottom and is\_right:

                    tile\_img = tileset['bottom\_corner\_right']

                    tile\_type = 'bottom\_corner\_right'

                elif is\_top:

                    tile\_img = tileset['top\_middle']

                    tile\_type = 'top\_middle'

                elif is\_bottom:

                    tile\_img = tileset['bottom\_middle']

                    tile\_type = 'bottom\_middle'

                elif is\_left:

                    tile\_img = tileset['side\_left']

                    tile\_type = 'side\_left'

                elif is\_right:

                    tile\_img = tileset['side\_right']

                    tile\_type = 'side\_right'

                else:

                    # Interior tile

                    tile\_img = tileset['middle']

                    tile\_type = 'middle'

                tile = self.canvas.create\_image(x, y, anchor="center", image=tile\_img)

                self.tiles.append(tile)

                # Track tile in tile\_map for consistent movement handling

                tile\_left = self.rect\_x + col \* TILE\_SIZE

                tile\_right = tile\_left + TILE\_SIZE

                tile\_top = self.rect\_y + row \* TILE\_SIZE

                tile\_bottom = tile\_top + TILE\_SIZE

                # Determine allowed movement directions

                allow\_up = True

                allow\_down = True

                allow\_left = True

                allow\_right = True

                if tile\_type == 'top\_corner\_left':

                    allow\_up = False

                    allow\_left = False

                elif tile\_type == 'top\_corner\_right':

                    allow\_up = False

                    allow\_right = False

                elif tile\_type == 'bottom\_corner\_left':

                    allow\_down = False

                    allow\_left = False

                elif tile\_type == 'bottom\_corner\_right':

                    allow\_down = False

                    allow\_right = False

                elif tile\_type == 'top\_middle':

                    allow\_up = False

                elif tile\_type == 'bottom\_middle':

                    allow\_down = False

                elif tile\_type == 'side\_left':

                    allow\_left = False

                elif tile\_type == 'side\_right':

                    allow\_right = False

                self.tile\_map[(col, row)] = {

                    'left': tile\_left,

                    'right': tile\_right,

                    'top': tile\_top,

                    'bottom': tile\_bottom,

                    'center\_x': x,

                    'center\_y': y,

                    'is\_ramp': False,

                    'ramp\_direction': None,

                    'tile\_type': tile\_type,

                    'allow\_up': allow\_up,

                    'allow\_down': allow\_down,

                    'allow\_left': allow\_left,

                    'allow\_right': allow\_right

                }

                # Add decorations on top row only

                if is\_top and random.random() > 0.7:

                    self.add\_decoration(x, y)

    def create\_custom\_platform(self):

        """Create a platform from a custom terrain layout"""

        tileset = terrain\_tiles[self.terrain\_type]

        # Track lifted tile positions for collision detection

        lifted\_coords = set()

        # Render each tile specified in the layout

        for col, row, tile\_type in self.terrain\_layout:

            x = self.rect\_x + col \* TILE\_SIZE + TILE\_SIZE // 2

            y = self.rect\_y + row \* TILE\_SIZE + TILE\_SIZE // 2

            tile\_img = tileset[tile\_type]

            tile = self.canvas.create\_image(x, y, anchor="center", image=tile\_img)

            self.tiles.append(tile)

            # Track lifted tiles (non-walkable cliff faces)

            if tile\_type in ['lifted\_left', 'lifted\_middle', 'lifted\_right']:

                lifted\_coords.add((col, row))

                continue  # Don't add to tile\_map

            # All other tiles are walkable

            is\_walkable = True

            if is\_walkable:

                tile\_left = self.rect\_x + col \* TILE\_SIZE

                tile\_right = tile\_left + TILE\_SIZE

                tile\_top = self.rect\_y + row \* TILE\_SIZE

                tile\_bottom = tile\_top + TILE\_SIZE

                # Check if this is a ramp tile

                is\_ramp = 'ramp' in tile\_type

                ramp\_direction = None

                if is\_ramp:

                    if 'left' in tile\_type:

                        ramp\_direction = 'left'  # Ramp going up to the left

                    elif 'right' in tile\_type:

                        ramp\_direction = 'right'  # Ramp going down to the right

                # Determine allowed movement directions based on tile type

                # By default, all directions allowed

                allow\_up = True

                allow\_down = True

                allow\_left = True

                allow\_right = True

                # Restrict based on specific tile types (only restrict platform edges)

                if tile\_type == 'top\_corner\_left':

                    allow\_up = False

                    allow\_left = False

                elif tile\_type == 'top\_corner\_right':

                    allow\_up = False

                    allow\_right = False

                elif tile\_type == 'bottom\_corner\_left':

                    allow\_down = False

                    allow\_left = False

                elif tile\_type == 'bottom\_corner\_right':

                    allow\_down = False

                    allow\_right = False

                elif tile\_type == 'top\_middle':

                    allow\_up = False

                elif tile\_type == 'bottom\_middle':

                    allow\_down = False

                elif tile\_type == 'side\_left':

                    allow\_left = False

                elif tile\_type == 'side\_right':

                    allow\_right = False

                # All other tiles (middle, ramps, etc.) have no restrictions - all directions allowed

                self.tile\_map[(col, row)] = {

                    'left': tile\_left,

                    'right': tile\_right,

                    'top': tile\_top,

                    'bottom': tile\_bottom,

                    'center\_x': x,

                    'center\_y': y,

                    'is\_ramp': is\_ramp,

                    'ramp\_direction': ramp\_direction,

                    'tile\_type': tile\_type,

                    'allow\_up': allow\_up,

                    'allow\_down': allow\_down,

                    'allow\_left': allow\_left,

                    'allow\_right': allow\_right

                }

        # Post-processing: Block movement into lifted edges

        self.\_apply\_lifted\_restrictions(lifted\_coords)

    def \_apply\_lifted\_restrictions(self, lifted\_coords):

        """

        Update adjacent walkable tiles to block movement into lifted edge tiles.

        Lifted tiles are non-walkable cliff faces that should act as collision walls.

        """

        for (col, row), tile\_data in self.tile\_map.items():

            # Block movement LEFT if a lifted tile is immediately to the left

            if (col - 1, row) in lifted\_coords:

                tile\_data['allow\_left'] = False

            # Block movement RIGHT if a lifted tile is immediately to the right

            if (col + 1, row) in lifted\_coords:

                tile\_data['allow\_right'] = False

            # Block movement DOWN if a lifted tile is immediately below

            if (col, row + 1) in lifted\_coords:

                tile\_data['allow\_down'] = False

            # Block movement UP if a lifted tile is immediately above

            # (Usually lifted tiles are below raised platforms, but handle all cases)

            if (col, row - 1) in lifted\_coords:

                tile\_data['allow\_up'] = False

    def get\_tile\_at\_position(self, x, y):

        """Get the tile at the given position"""

        if not self.tile\_map:

            return None

        # Convert position to tile coordinates

        col = int((x - self.rect\_x) / TILE\_SIZE)

        row = int((y - self.rect\_y) / TILE\_SIZE)

        if (col, row) in self.tile\_map:

            return (col, row, self.tile\_map[(col, row)])

        return None

    def get\_tile\_area(self, col, row):

        """Get walkable area for a specific tile, searching connected tiles in the same row"""

        if not self.tile\_map:

            # No tile map, use full platform bounds

            return None

        if (col, row) not in self.tile\_map:

            return None

        # Find all connected walkable tiles in the same row

        left\_col = col

        right\_col = col

        # Search left

        while (left\_col - 1, row) in self.tile\_map:

            left\_col -= 1

        # Search right

        while (right\_col + 1, row) in self.tile\_map:

            right\_col += 1

        # Return bounds of the connected tile area

        left\_bound = self.tile\_map[(left\_col, row)]['left'] + TILE\_SIZE \* 0.75

        right\_bound = self.tile\_map[(right\_col, row)]['right'] - TILE\_SIZE \* 0.75

        return {

            'left': left\_bound,

            'right': right\_bound,

            'center\_x': self.tile\_map[(col, row)]['center\_x'],

            'center\_y': self.tile\_map[(col, row)]['center\_y']

        }

    def add\_decoration(self, x, y, deco\_name=None):

        # If no decoration name provided, choose randomly (no trees in random selection)

        if deco\_name is None:

            deco\_choices = ['rock1', 'rock2', 'rock3', 'rock4', 'bush1', 'bush2']

            deco\_name = random.choice(deco\_choices)

        deco\_img = decorations[deco\_name]

        # Trees need to be raised above the ground

        y\_offset = -16 if 'tree' in deco\_name else 0

        # Place decoration centered on the tile

        deco = self.canvas.create\_image(x, y + y\_offset, anchor="center", image=deco\_img)

        self.decorations.append(deco)

    def add\_water\_below(self):

        """Add water tiles along the bottom edge of the platform"""

        tiles\_wide = self.rect\_width // TILE\_SIZE

        bottom\_y = self.rect\_y + self.rect\_height + TILE\_SIZE // 2

        for col in range(tiles\_wide):

            x = self.rect\_x + col \* TILE\_SIZE + TILE\_SIZE // 2

            # Determine which water animation to use based on position

            if col == 0:

                # Left corner - use water #1

                water\_type = 1

            elif col == tiles\_wide - 1:

                # Right corner - use water #3

                water\_type = 3

            else:

                # Middle - use water #2

                water\_type = 2

            # Create water tile with first frame

            water\_tile = self.canvas.create\_image(x, bottom\_y, anchor="center", image=water\_animations[water\_type][0])

            # Store tile ID and its animation type

            self.water\_tiles.append({'id': water\_tile, 'type': water\_type})

    def animate\_water(self, frame\_index):

        """Update water tiles to show the current frame"""

        for water\_info in self.water\_tiles:

            water\_type = water\_info['type']

            water\_id = water\_info['id']

            # Update to current frame

            self.canvas.itemconfig(water\_id, image=water\_animations[water\_type][frame\_index])

    def add\_trees\_at\_edges(self):

    # Get platform edges

        left\_x = self.rect\_x + TILE\_SIZE \* 2

        right\_x = self.rect\_x + self.rect\_width - TILE\_SIZE \* 2

        top\_y = self.rect\_y + TILE\_SIZE // 2

        # Top-left: tree1

        self.add\_decoration(left\_x, top\_y, 'tree1')

        self.solid\_objects.append({'x': left\_x, 'y': top\_y, 'width': 64, 'height': 96})

        # Top-right: tree2

        self.add\_decoration(right\_x, top\_y, 'tree2')

        self.solid\_objects.append({'x': right\_x, 'y': top\_y, 'width': 64, 'height': 122})

        # Mid-left: tree3

        mid\_left\_x = self.rect\_x + TILE\_SIZE \* 4

        self.add\_decoration(mid\_left\_x, top\_y, 'tree3')

        self.solid\_objects.append({'x': mid\_left\_x, 'y': top\_y, 'width': 64, 'height': 84})

        # Mid-right: tree4

        mid\_right\_x = self.rect\_x + self.rect\_width - TILE\_SIZE \* 4

        self.add\_decoration(mid\_right\_x, top\_y, 'tree4')

        self.solid\_objects.append({'x': mid\_right\_x, 'y': top\_y, 'width': 64, 'height': 64})

    def move(self, dy):

        """Move the platform vertically for scrolling"""

        self.rect\_y += dy

        for tile in self.tiles:

            self.canvas.move(tile, 0, dy)

        for deco in self.decorations:

            self.canvas.move(deco, 0, dy)

        for water\_info in self.water\_tiles:

            self.canvas.move(water\_info['id'], 0, dy)

    def is\_offscreen(self):

        """Check if platform has scrolled off screen"""

        return self.rect\_y > HEIGHT + self.rect\_height

class Game:

    def \_\_init\_\_(self):

        self.platforms = []

        self.camera\_y = 0

        self.score = 0

        self.water\_frame = 0

        self.water\_frame\_counter = 0

        self.water\_animation\_speed = 8

        self.current\_level\_index = 0

        self.level\_name\_text = None

        self.enemies = []

        self.player = None

        self.game\_over = False

        self.keys = {'w': False, 'a': False, 's': False, 'd': False}

        self.action\_keys = {'j': False, 'k': False, 'l': False}

        self.heart\_sprites = []

        for i in range(3):

            heart = canvas.create\_image(WIDTH - 40 - (i \* 40), 30, anchor="center", image=heart\_frames[0])

            self.heart\_sprites.append(heart)

        self.load\_level(0)

    def load\_level(self, level\_index):

        if level\_index >= len(LEVELS):

            self.show\_victory()

            return

        self.current\_level\_index = level\_index

        level = LEVELS[level\_index]

        self.game\_over = False

        # Clear canvas

        canvas.delete("all")

        # Reset hearts

        self.heart\_sprites = []

        for i in range(3):

            heart = canvas.create\_image(WIDTH - 40 - (i \* 40), 30, anchor="center", image=heart\_frames[0])

            self.heart\_sprites.append(heart)

        # Clear platforms and enemies

        self.platforms = []

        self.enemies = []

        platform\_x = (WIDTH - level['platform\_width']) // 2

        platform\_y = HEIGHT - 300

        start\_platform = Platform(

            canvas,

            platform\_x,

            platform\_y,

            level['platform\_width'],

            level['platform\_height'],

            terrain\_color=level['terrain\_color'],

            terrain\_layout=level.get('terrain\_layout')

        )

        self.platforms.append(start\_platform)

        # Player spawn

        if level.get('player\_start'):

            tile = level['player\_start']

            area = start\_platform.get\_tile\_area(tile['tile\_col'], tile['tile\_row'])

            if area:

                player\_start\_x = area['center\_x']

                player\_start\_y = area['center\_y']

            else:

                player\_start\_x = WIDTH // 2

                player\_start\_y = platform\_y + TILE\_SIZE // 2

        else:

            player\_start\_x = WIDTH // 2

            player\_start\_y = platform\_y + TILE\_SIZE // 2

        self.player = Player(canvas, player\_start\_x, player\_start\_y)

        self.player.set\_platform(start\_platform)

        canvas.tag\_raise(self.player.id)

        # Enemies

        for enemy\_data in level['enemies']:

            if 'tile\_col' in enemy\_data and 'tile\_row' in enemy\_data:

                area = start\_platform.get\_tile\_area(enemy\_data['tile\_col'], enemy\_data['tile\_row'])

                if area:

                    enemy\_x = area['center\_x']

                    enemy\_y = area['center\_y']

                    enemy = Enemy(canvas, enemy\_x, enemy\_y, start\_platform,

                                  enemy\_type=enemy\_data['type'],

                                  tile\_col=enemy\_data['tile\_col'],

                                  tile\_row=enemy\_data['tile\_row'])

                else:

                    continue

            else:

                enemy\_x = get\_enemy\_x\_position(enemy\_data['position'], platform\_x, level['platform\_width'])

                enemy\_y = player\_start\_y

                enemy = Enemy(canvas, enemy\_x, enemy\_y, start\_platform, enemy\_type=enemy\_data['type'])

            self.enemies.append(enemy)

            canvas.tag\_raise(enemy.sprite)

        # Level name

        self.level\_name\_text = canvas.create\_text(

            WIDTH // 2, 50,

            text=f"Level {level['id']}: {level['name']}",

            font=("Arial", 20, "bold"),

            fill="white",

            anchor="center"

        )

    def show\_game\_over(self):

        canvas.create\_text(WIDTH // 2, HEIGHT // 2 - 40, text="Game Over", font=("Arial", 32), fill="red")

        retry\_button = tk.Button(canvas.master, text="Retry", font=("Arial", 16), command=self.restart\_game)

        canvas.create\_window(WIDTH // 2, HEIGHT // 2 + 20, window=retry\_button)

    def restart\_game(self):

        self.player = None

        self.game\_over = False

        self.load\_level(0)

    def update(self):

        if self.game\_over:

            root.after(16, self.update)

            return

        vx = 0

        vy = 0

        if self.keys['a']: vx -= MOVE\_SPEED

        if self.keys['d']: vx += MOVE\_SPEED

        if self.keys['w']: vy -= MOVE\_SPEED

        if self.keys['s']: vy += MOVE\_SPEED

        self.player.set\_velocity(vx, vy)

        self.player.update(self.platforms)

        for enemy in self.enemies:

            enemy.update()

        for enemy in self.enemies:

            if not enemy.is\_dead:

                dx = self.player.x - enemy.x

                dy = self.player.y - enemy.y

                distance = (dx\*\*2 + dy\*\*2) \*\* 0.5

                if distance < 50:

                    if self.player.take\_damage():

                        self.game\_over = True

                        self.show\_game\_over()

                    else:

                        self.update\_hearts()

        if all(enemy.is\_dead for enemy in self.enemies) and self.enemies:

            if not hasattr(self, 'level\_complete\_timer'):

                self.level\_complete\_timer = 120

            else:

                self.level\_complete\_timer -= 1

                if self.level\_complete\_timer <= 0:

                    delattr(self, 'level\_complete\_timer')

                    self.load\_level(self.current\_level\_index + 1)

        self.water\_frame\_counter += 1

        if self.water\_frame\_counter >= self.water\_animation\_speed:

            self.water\_frame\_counter = 0

            self.water\_frame = (self.water\_frame + 1) % 12

            for platform in self.platforms:

                platform.animate\_water(self.water\_frame)

        root.after(16, self.update)

    def key\_press(self, event):

        key = event.keysym.lower()

        if key in self.keys:

            self.keys[key] = True

    def key\_release(self, event):

        key = event.keysym.lower()

        if key in self.keys:

            self.keys[key] = False

    def mouse\_click(self, event):

        if self.player.attack1():

            for enemy in self.enemies:

                if not enemy.is\_dead:

                    dx = self.player.x - enemy.x

                    dy = self.player.y - enemy.y

                    distance = (dx\*\*2 + dy\*\*2) \*\* 0.5

                    if distance < 70:

                        enemy.take\_damage(1)

    def mouse\_right\_click(self, event):

        self.player.guard()

    def update\_hearts(self):

        for i in range(3):

            if i < self.player.health:

                canvas.itemconfig(self.heart\_sprites[i], image=heart\_frames[0])

            else:

                canvas.itemconfig(self.heart\_sprites[i], image=heart\_frames[4])

# Initialize game

game = Game()

root.bind("<KeyPress>", game.key\_press)

root.bind("<KeyRelease>", game.key\_release)

root.bind("<Button-1>", game.mouse\_click)  # Left mouse click

root.bind("<Button-2>", game.mouse\_right\_click)  # Right mouse click (Button-2 on Mac, Button-3 on Windows/Linux)

root.bind("<Button-3>", game.mouse\_right\_click)  # Also bind Button-3 for cross-platform compatibility

game.update()

root.mainloop()

run.sh

#!/bin/bash

# Tower Tactics Game Launcher

# Get the directory where this script is located

SCRIPT\_DIR="$( cd "$( dirname "${BASH\_SOURCE[0]}" )" && pwd )"

# Change to the game directory

cd "$SCRIPT\_DIR"

# Run the game

python3 main.py

### --------------------------------------------------------------------------------------------------