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 Defense:

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