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

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

# Tower Defense:

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

# David Vs Goliath

--------------------------------------------------------------------------------------------------