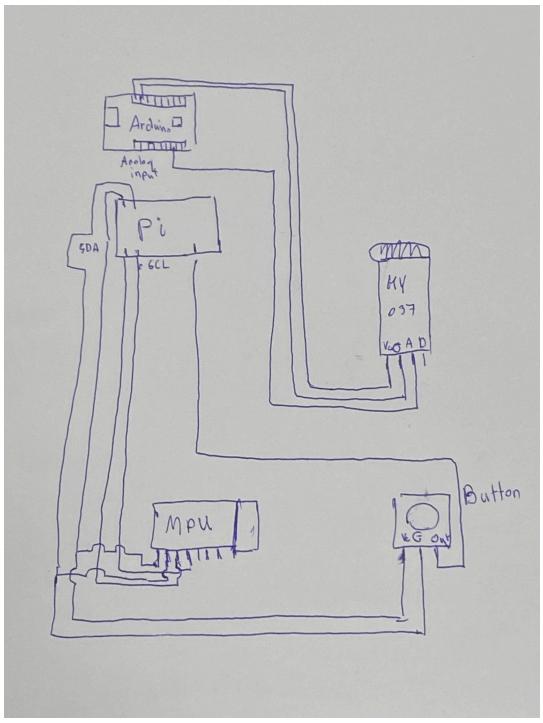
DinoScream

The game that you might played in the past during the internet cutoff. This game is similar to the original one but I implemented some simple logic to integrate with hardware like sound module and gyro. The main move is using your voice to control the jump of the dinosaur that is why using the name DinoScream I created the mechanic that can move forward and backward during the game. The difficulty of this game will increase as the score increases over time, so please enjoy my simple game project that tries to integrate with the hardware in Raspberry Pi board.

Requirement Hardware

- Raspberry Pi (for playing game)
- Arduino (for receiving the analog signal from the sound sensor module)
- KY-037 (for sound sensor)
- MPU6050 (for gyro tilting move and crouching mechanic)
- Switch (to select the menu)
- Breadboard (for wire and module connection)

Circuit Diagram.



Let's get through some interesting section code.

```
class DinoScream:
    def __init__(self):
        self.__setting = Setting()
        self.__screen = pygame.display.set_mode(*self.__setting.getVideoMode())
        self.__sceneManager = SceneManager("main_menu")
```

```
    def run(self):
        clock = pygame.time.Clock()
        start_serial_reader()
        while True:
            delta_time = clock.tick(FPS)

            events = pygame.event.get()
            # handle quiting the game event.
            for event in events:
                if event.type == pygame.QUIT:
                    pygame.quit()
                    quit()

            self.__sceneManager.run(delta_time, self.__screen, events)
            pygame.display.update()
```

This is the main game loop that will run when you type the command python main.py it will call the function run from the main.py file. This method will pass the parameter delta time that indicate the elapsed time per frame, screen that use to display, and the pygame event that occur during the game loop to the screen manager. The task of the screen manager is the class that will change the screen as desired.

```
from src.gamescene.Gameplay import Gameplay
from src.gamescene.MainMenu import MainMenu


class SceneManager:
    def __init__(self, initial_scene):
        self.__scene = {"gameplay": Gameplay(self), "main_menu": MainMenu(self)}
        self.current_scene = self.__scene[initial_scene]

    def change_scene(self, new_scene):
        self.current_scene = self.__scene[new_scene]
        return self.current_scene

    def get_current_scene(self):
        return self.current_scene

    def run(self, delta_time, screen, events):
        if self.current_scene:
            self.current_scene.run(delta_time, screen, events)
```

Here is the class scene manager as you see it will hold each scene as python dictionary. In each scene will be there own class that i implement from the base class call GameState.

```
"""Abstract class for game states."""


class GameState:
    def __init__(self):
```

```
        pass

    def update(self):
        """Update the game state."""
        pass

    def render(self):
        """Render the current game state"""
        pass

    def handle_event(self):
        """Handle the events of the current game state."""
        pass

    def run(self):
        """Run the current game state."""
        pass
```

The method run will call function update, render, and handle_event of the current state or current scene. In the DinoScream you will see there function called name start_serial_reader() this function will initialize the sound module to let it detect the sound spike and check the intensity let see the code.

```
def detected_module():
    try:
        ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
        ser.close()
        return True
    except serial.serialutil.SerialException:
        return False


def read_serial():
    with serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1) as ser:
        time.sleep(1)  # Wait for the serial connection to establish
        print("Connected to the serial port")
        last_action_time = 0
        while True:
            line = ser.readline().decode("utf-8").strip()  # Read and decode the line
            if line:
                try:
                    # Parse the line as a float if it's a spike intensity
                    spike_intensity = float(line)
                    print("Intesity: ", spike_intensity)
                    current_time = time.time()
                    if current_time - last_action_time >= COOLDOWN:
                        if spike_intensity > 6:
                            # Trigger the jump event
                            pygame.event.post(pygame.event.Event(JUMP_EVENT))
```

```
                            last_action_time = current_time
                except ValueError:
                    # If parsing fails, ignore the line
                    pass
            time.sleep(0.1)


# Reading the serial must be done in a separate thread to prevent blocking
def start_serial_reader():
    # handle to use mock or read real serial
    if detected_module():
        serial_thread = threading.Thread(target=read_serial, daemon=True)
        serial_thread.start()
    else:
        print("Module not detected, using mock data.")
        pass
```

Here is inside the KY-037 module file. The function detected_module for handling the if the module is connect or not the next function that call from another file is start_serial_reader it will check if the module is detect or not then i use another thread to do function read_serial parallelism. These function will waitingfor receive the serial from the arduino that will send the voice intensity from the module and check if it reach the condition it will post event to let the dinosaur jump and the cooldown between the spike is 0.7 sec.

Let move onto the gameplay state or scene. After the game change to this scene it will initialize the dinosaur, obstacles. Function update will be the place where we use to loop the game logic like random the obstacles between the crow and cactus and check whether the dino rectangle or collision box is collide with the obstacles collision box. the logic where increasing the score and check score to change the difficulty of the game (make the obstacle spawn faster) and if the score reach the maximum it will end the game.

```
        # Randomly spawn obstacles
        self.__obstacles_last_spawn += delta_time
        if self.__obstacles_last_spawn >= self.__obstacles_spawn_speed:
            self.random_obstacle()
            self.__obstacles_last_spawn = 0

        """
        Dino update function
        param: screen_width: width of the screen for range of movement
        param: tilt_angle: angle of the gyro make sure the direction of tilting
        """
        self.__dino.update(
            screen.get_width(),
            tilt_angle=get_tilt_angle(),
            elapsed_time=delta_time,
        )

        # Update the obstacles objects
        for obj in self.__obstacles:
```

```
            obj.update()
            if obj.check_collision(self.__dino.rect):
                self.__game_over = True

        # Score increment mechanism
        if self.__scorePS >= 90:
            self.__score += 1
            self.__scorePS = 0
        self.__scorePS += delta_time

        # Score mechanism increase game difficulty
        match self.__score:
            case 100:
                self.__obstacles_spawn_speed = 1800
            case 300:
                self.__obstacles_spawn_speed = 1600
            case 500:
                self.__obstacles_spawn_speed = 1200
            case 700:
                self.__obstacles_spawn_speed = 1000
            case 1000:
                self.__obstacles_spawn_speed = 800
            case 1500:
                self.__obstacles_spawn_speed = 600

        # Check if the score is reach 99,999 will end the game
        if self.__score >= 99999:
            self.__game_over = True
```

The get_tilt_angle function will return the value of the current y-axis of the gyro and handle the dino will move forward or backward.

```
# Global the sensor variable
sensor = None


# initialize the sensor by platform type
def init_sensor():
    global sensor
    try:
        from mpu6050 import mpu6050

        sensor = mpu6050(0x68)
    except ImportError:

        # Mock the sensor for Windows Operating System
        class MockMPU6050:
            def get_accel_data(self):
                return {"x": 0.0, "y": 0.0, "z": 0.0}

        sensor = MockMPU6050()
```

```
# This function will return the tilt angle of the MPU6050 module
def get_tilt_angle():
    global sensor
    if sensor is None:
        init_sensor()

    # handle the case where the sensor is not connected
    try:
        accel_data = sensor.get_accel_data()
        y = accel_data["y"]  # use only y-axis data for moving the dino object
        return y
    except Exception as e:
        print("Error reading from sensor: ", e)
        return None


def get_tiltX_angle():
    global sensor
    if sensor is None:
        init_sensor()

    # handle the case where the sensor is not connected
    try:
        accel_data = sensor.get_accel_data()
        x = accel_data["x"]  # use only x-axis data for moving the dino object
        return x
    except Exception as e:
        print("Error reading from sensor: ", e)
        return None
```

Function get_tilt_angle for handle moving forward and backward and function tiltX is for handle the crouch mechanic. I need to initialize the sensor which handle whether you connect to the module or not.

The next function is render it will make the dinosaur look moveable and having animation. The animation of each sprite to these class and get the each frames of the move.

```
import pygame

SPRITE_SHEET = pygame.image.load("resources/game_sprites.png")


# Animation class to store the animation of the sprite


class Animation:
    def __init__(self, start_point=(0, 0), width=50, height=50, frames: int = 1):
        self.start_point = start_point
```

```
        self.width = width
        self.height = height
        self.frames = frames

    def getAnimationFrames(self) -> list:
        frames = []
        for i in range(self.frames):
            frames.append(
                SPRITE_SHEET.subsurface(
                    (
                        self.start_point[0] + self.width * i,
                        self.start_point[1],
                        self.width,
                        self.height,
                    )
                )
            )
        return frames
```

It will loop through the list then render the current frame it will look like the sprite is moving. The next one that will render also is tile of the game. I make it as there own system because make render consecutively no disjointed line.

```
import pygame

# Load the image that contains the game sprites
IMAGE = pygame.image.load("resources/game_sprites.png")

# Constants position of the tile in the image
START_POS = (0, 53)
TILE_WIDTH = 1203
TILE_HEIGHT = 14

GAP_DINO = 10
SCROLL_SPEED = 5


class Tile:
    def __init__(self, dino_rect):
        self.under_dino = dino_rect.bottom
        self.tile_width = TILE_WIDTH  # Width of each tile
        self.tile_rect = pygame.Rect(
            START_POS[0], START_POS[1], TILE_WIDTH, TILE_HEIGHT
        )

        self.offset = 0  # Scrolling offset

    def draw(self, screen, speed=SCROLL_SPEED):
        """Draws the tiles on the screen with a looping effect."""
        y_pos = self.under_dino - GAP_DINO
        screen.blit(IMAGE, (self.offset, y_pos), self.tile_rect)
```

```
        screen.blit(IMAGE, (self.offset + self.tile_width, y_pos), self.tile_rect)

        self.offset -= speed
        if self.offset <= -self.tile_width:
            self.offset = 0

    def stop(self):
        self.offset = 0
```

The last function is handle_event i make it can play whether u have module or not but only jumpin and crouching only like the original game.

```
            if event.type == pygame.QUIT:
                self._running = False

            # Handle jump events separate platform controls
            elif event.type == JUMP_EVENT or (
                detected_module() is False
                and event.type == pygame.KEYDOWN
                and event.key == pygame.K_UP
            ):
                self.__dino.jump()
            # Handle crouch event if it doesn't connect to the module
            elif (
                get_tiltX_angle() is None
                and event.type == pygame.KEYDOWN
                and event.key == pygame.K_DOWN
            ):
                self.__dino.crouch()
            # Handle the stand up after crouching
            elif event.type == pygame.KEYUP:
                if event.key == pygame.K_DOWN:
                    self.__dino.stand_up()
            # Handle if there is no modules switch
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    if self.__game_over:
                        if self.__default_option == 0:
                            self.reset_game()
                        elif self.__default_option == 1:
                            self.reset_game()
                            self._screenManager.change_scene("main_menu")
```

In these 3 function will handle if the game is over by collide with obstacles or reach the maximum score in function render will update and render dinosaur die sprite abd freeze title and stop the update function and in handle function will waiting for receive quit or restart.
Example of handle the event.

```
if self.__game_over:
            current_time = pygame.time.get_ticks()

            # Button selection logic based on press timing
```

```
            if check_button_press():
                if current_time - self.button_last_pressed <= 500:
                    # Double press to select the option
                    if self.__default_option == 0:
                        self.reset_game()
                    elif self.__default_option == 1:
                        self.reset_game()
                        self._screenManager.change_scene("main_menu")
                    self.button_last_pressed = 0
                else:
                    # Toggle option with single press
                    self.__default_option = (
                        1 - self.__default_option
                    )  # toggles between 0 and 1
                    self.select_option(self.__default_option)

                    # Update last pressed time
                    self.button_last_pressed = current_time
```

My full code is in github you can findout:
https://github.com/plscallMeAlex/DinoScream.git

Project By:
(66011525)Audthanee Supeeramongkolkul
(66011072)Naphat Umpa