আন্তর্জাতিক ইসলামী বিশ্ববিদ্যালয় চট্টগ্রাম
International Islamic University Chittagong
Department of Computer & Communication Engineering(CCE)

PROJECT REPORT

Project No: 01
Project Name: Simple 2D Platformer

Course Title: Computer Animation and Game Development Sessional
Course Code: CCE-3606

Submitted By

Student Name : Ahsanul Karim Tanim
Student Id : E221013
Semester : 6th
Section : A

Submitted To

Sukanta Paul,

Adjunct Faculty,
IIUC

Experiment Date: 19 / 11 / 2024

Submission Date: 26 / 11 / 2024

Remark

Project No: 1
Project Name: A simple 2D Platformer

Process:

1. Set up the Environment: Initialize Pygame and define the game window.
2. Design the Game World: Create a game character, platforms, and obstacles.
3. Implement Controls: Add functionality for movement and jumping.
4. Collision Detection: Ensure the character interacts correctly with platforms and obstacles.
5. Game Logic: Implement win/lose conditions.
6. Run and Test: Test the game for bugs and playability.

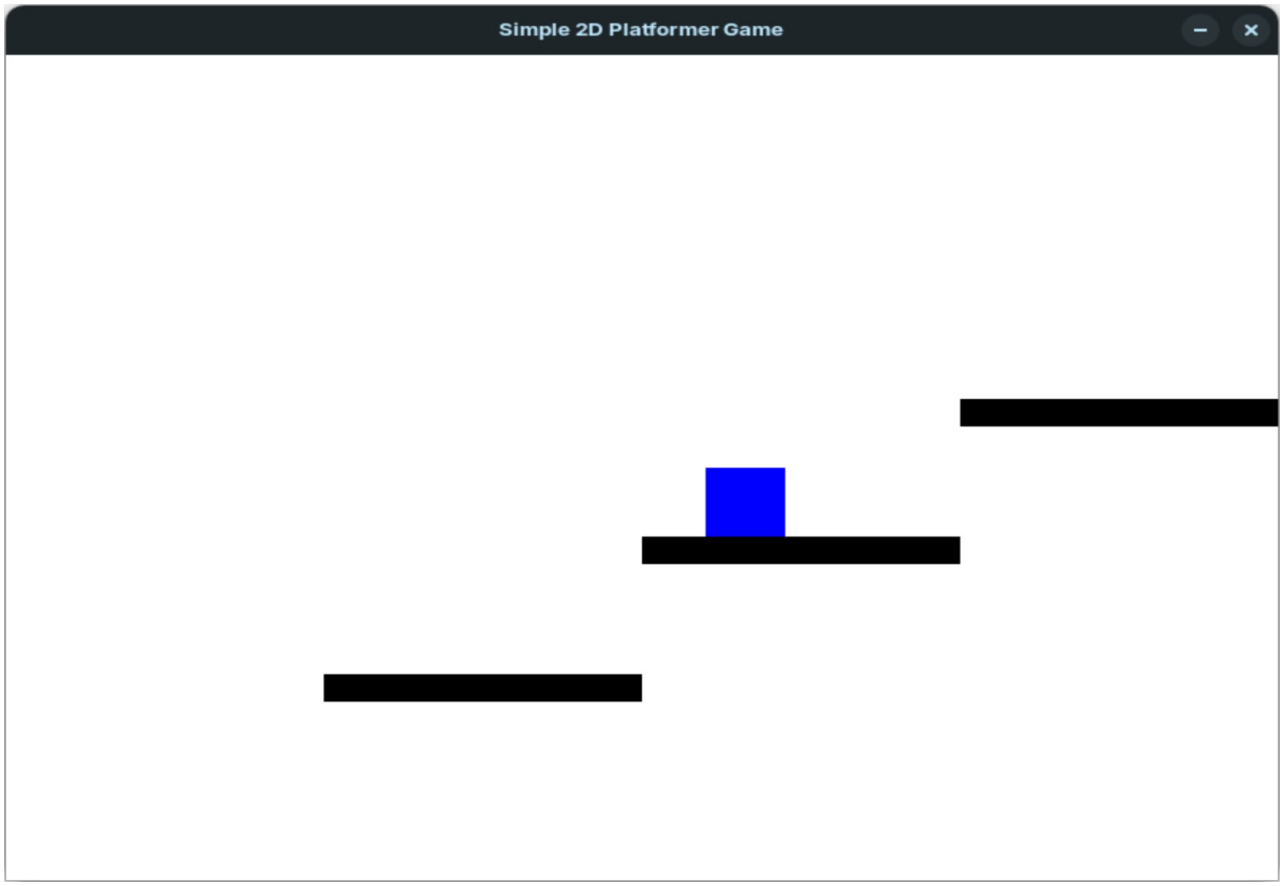
Code:

```
import pygame
import sys

# Step 1: Set up the Environment
pygame.init()

# Screen dimensions
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Simple 2D Platformer Game")

# Colors
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
BLACK = (0, 0, 0)

# Frame rate
clock = pygame.time.Clock()
FPS = 60

# Step 2: Design the Game World
# Player propertiesE221013
player_width, player_height = 50, 50
player_x, player_y = 250, 430 # Starting position
player_speed = 5
player_velocity_y = 0
gravity = 0.3
jump_power = -8
is_jumping = False

# Platforms
platforms = [
pygame.Rect(200, 450, 200, 20), # First platform
pygame.Rect(400, 350, 200, 20), # Middle platform
pygame.Rect(600, 250, 200, 20), # Highest platform
]
highest_platform = platforms[-1] # The highest platform is the last in the list

# Step 4: Collision DetectionTanim
```

```
def is_player_on_platform():
player_rect = pygame.Rect(player_x, player_y, player_width, player_height)
for platform in platforms:
if player_rect.colliderect(platform) and player_velocity_y >= 0:
return platform
return None

# End the gameE221013 with a message
def end_game(message):
print(message)
pygame.quit()
sys.exit()

# Main game loop
def main():
global player_x, player_y, player_velocity_y, is_jumping

run = True
while run:
# Step 6: Run and Test - Clear the screen
screen.fill(WHITE)

# Event handling
for event in pygame.event.get():
if event.type == pygame.QUIT:
run = False

# Step 3: Implement Controls
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
player_x -= player_speed
if keys[pygame.K_RIGHT]:
player_x += player_speed
if keys[pygame.K_SPACE] and not is_jumping:
is_jumping = True
player_velocity_y = jump_power

# Gravity
player_velocity_y += gravity
player_y += player_velocity_y

# Collision detection with platforms
platform = is_player_on_platform()
if platform:
player_y = platform.top - player_height
player_velocity_y = 0
is_jumping = False
# Step 5: Game Logic
if player_y > HEIGHT: # Lose condition: Player falls off
end_game("Game Over: You fell off the platforms!")
```

```
elif platform == highest_platform: # Win condition: Reach the highest platform
end_game("Congratulations: You reached the highest platform!")
# Keep the player within screen bounds
if player_x < 0:
player_x = 0
if player_x > WIDTH - player_width:
player_x = WIDTH - player_width
# Draw the player
pygame.draw.rect(screen, BLUE, (player_x, player_y, player_width, player_height))

# Draw platforms
for platform in platforms:
pygame.draw.rect(screen, BLACK, platform)

# Update the display
pygame.display.flip()

# Control the frame rate
clock.tick(FPS)

pygame.quit()
sys.exit()

if __name__ == "__main__":
main()
```

Output:



Limitations:

- **Basic Gameplay Mechanics:** The game lacks advanced mechanics like double jumps, wall sliding, or power-ups, which limit its depth and replayability.
- **Static Platforms:** Platforms are fixed in position, reducing the challenge and variety in gameplay. Moving or disappearing platforms could enhance engagement.
- **No Obstacles or Enemies:** The absence of dynamic elements like enemies or hazards makes the game less challenging.
- **Limited Visual Appeal:** The game has basic graphics with no animations, textures, or background music, which can affect user engagement.
- **Single-Level Design:** The game doesn't include multiple levels or progression, limiting its scope.
- **No Scoring System:** Players have no way to track their performance or achievements, reducing competitiveness.
- **Collision Edge Cases:** Platform edge collisions can sometimes behave unrealistically, affecting immersion.
- **Device Compatibility:** The game is designed for desktops and may not work well on other platforms like mobile devices.

Discussion: This project implemented key features like basic player movement, gravity simulation, collision detection, and win/lose conditions. Challenges included fine-tuning jump physics for smooth gameplay and handling edge cases, such as edge collisions. Future improvements could include sound effects, animations, enemies, obstacles, and scoring systems for a more engaging experience. Overall, the experiment effectively demonstrated fundamental concepts of 2D game development using Pygame, successfully implementing game logic and physics to create a functional platformer.