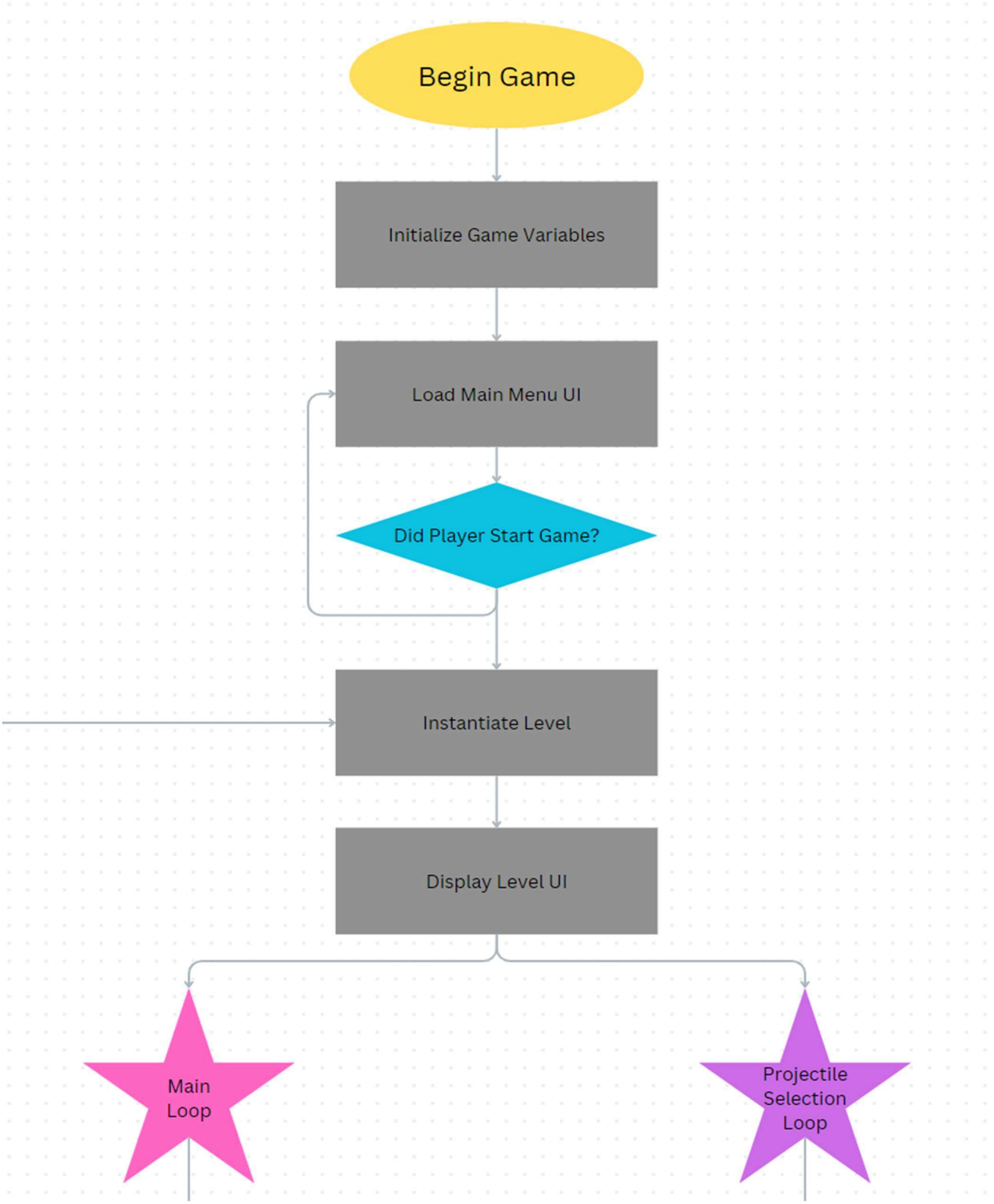
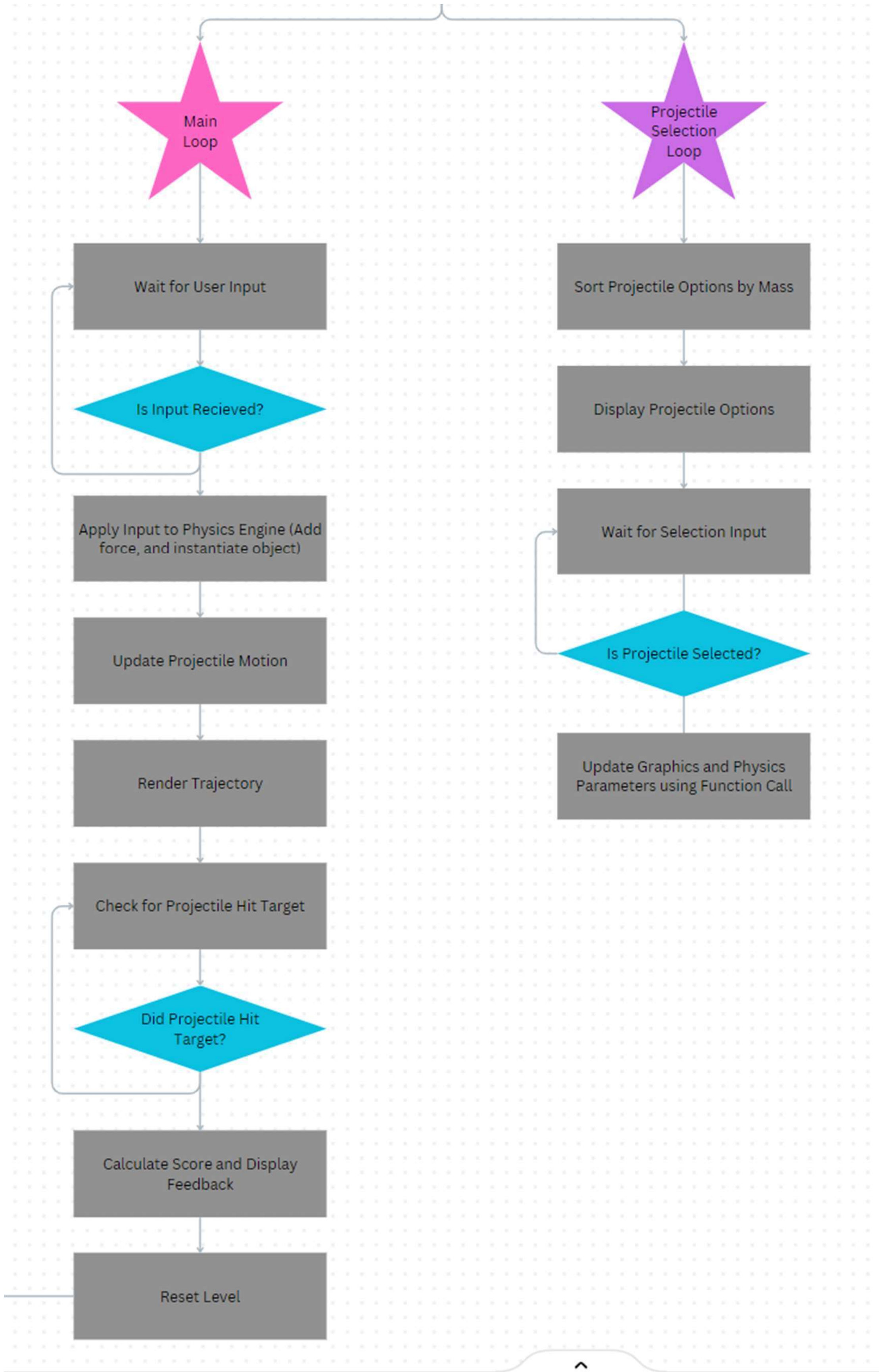
Test Plan

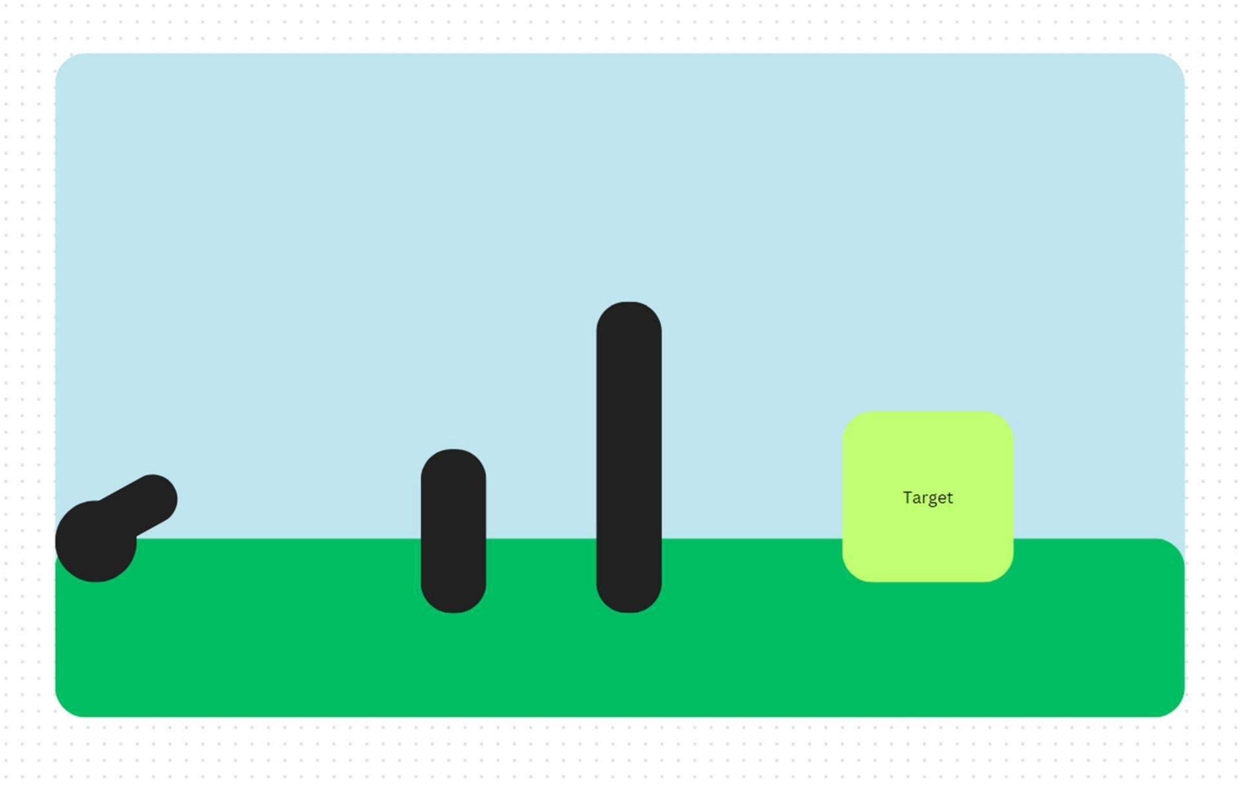
Test Type	Nature of Test	Example
Functionality Testing	Ensure the application correctly allows for projectile selection with different weights and sizes.	Select each projectile type and verify its properties (weight, size) are reflected in the gameplay.
Input Validation	Test the adjustability of launch parameters and verify the limits are enforced.	Attempt to set the launch angle and power beyond the allowed range and confirm the application clamps values to the specified limits.
Trajectory Visualization Test	Check if the trajectory of each launch is visualized accurately in real-time.	Launch a projectile and observe if the trajectory matches expected physics calculations.
Feedback System Test	Verify the application provides feedback on the success of hitting a target after each launch.	Hit and miss targets intentionally and check if the feedback (hit/miss, score) is accurate.
Comparison Feature Test	Test the side-by-side comparison feature for different launch parameters.	Launch projectiles with varying parameters and confirm if the application displays a comparative analysis accurately.
Content Integration Test	Confirm the educational content is present and corresponds correctly to the principles demonstrated by each launch.	Review the educational summaries post-launch to ensure they match the physics concepts involved in the projectile's motion.
Level Progression Test	Validate that the application includes levels of increasing difficulty with the introduction of new challenges.	Play through various levels to confirm the presence of obstacles and the incremental difficulty in line with educational goals.

Flowcharts

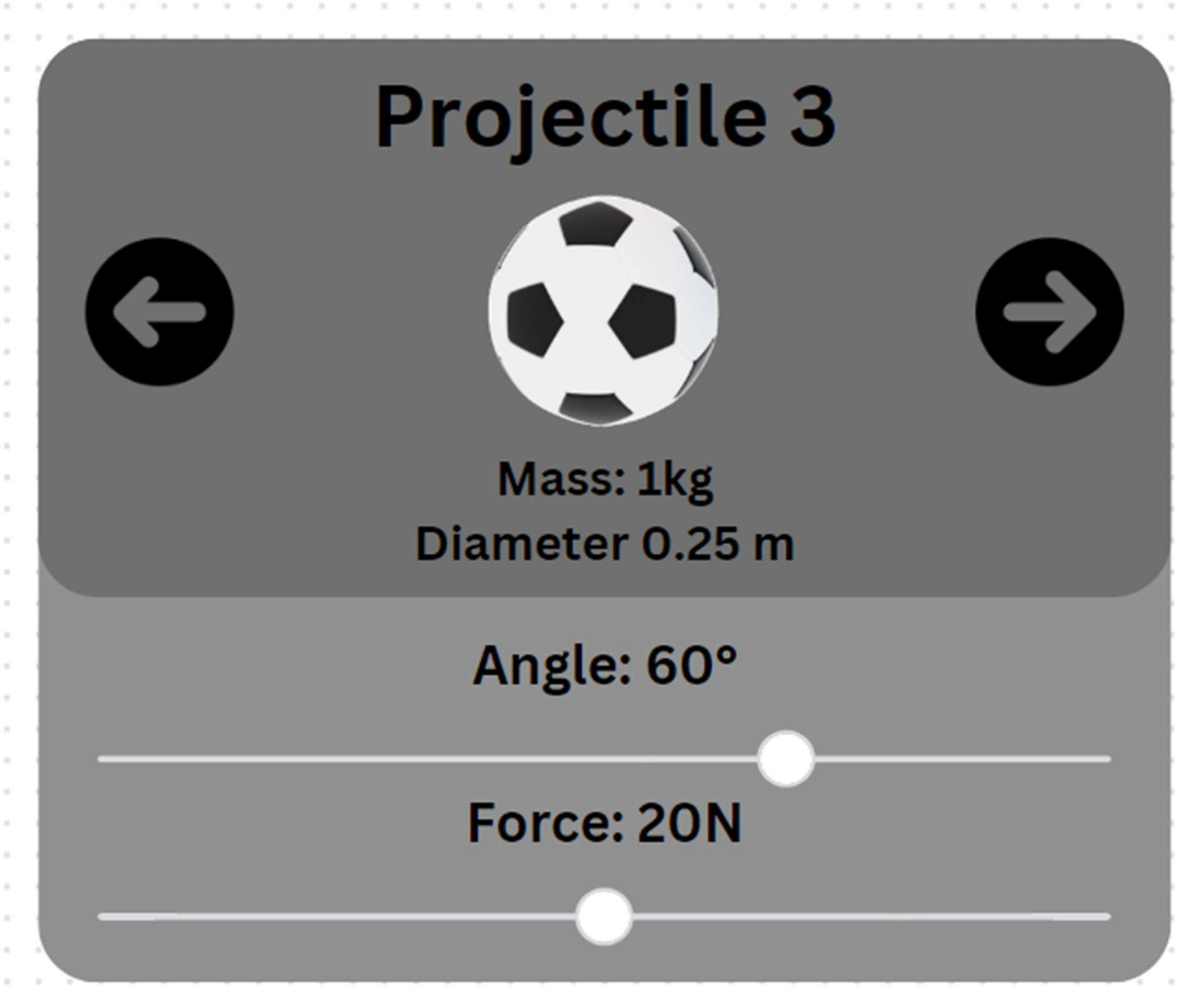

Begin Game
Initialize Game Variables
Load Main Menu UI
Did Player Start Game?
Instantiate Level
Display Level UI
Main Loop
Projectile Selection Loop


Main Loop
Wait for User Input
Is Input Recieved?
Apply Input to Physics Engine (Add force, and instantiate object)
Update Projectile Motion
Render Trajectory
Check for Projectile Hit Target
Did Projectile Hit Target?
Calculate Score and Display Feedback
Reset Level
Projectile Selection Loop
Sort Projectile Options by Mass
Display Projectile Options
Wait for Selection Input
Is Projectile Selected?
Update Graphics and Physics Parameters using Function Call

Graphics



This screen is a snapshot of the core mechanics of the game, incorporating elements of physics education through an engaging, goal-oriented task. It embodies several success criteria: the interactive visualization of projectile motion, the adjustability of launch parameters, and the inclusion of obstacles that necessitate the application of learned concepts.



This screen allows players to understand the differences between projectiles and make an informed selection based on their learning objectives or curiosity.

Explanation of Key Algorithms

// Pseudocode and explanation for trajectory calculations

```
// Constants

GRAVITY = 9.81 // Acceleration due to gravity (m/s^2)

AIR_RESISTANCE = False // Set to true if considering air resistance

DRAG_COEFFICIENT = 0.47 // Typical for a sphere, if needed

AIR_DENSITY = 1.225 // kg/m^3 at sea level, if needed

TIME_STEP = 0.01 // Time increment for the simulation loop, in seconds


// Inputs from user

angle = GetLaunchAngle() // In degrees

power = GetLaunchPower() // Arbitrary units based on user input


// Convert angle to radians for calculations

angle_in_radians = ConvertDegreesToRadians(angle)


// Initial calculations

initial_velocity_x = power * cos(angle_in_radians)

initial_velocity_y = power * sin(angle_in_radians)

position_x = 0

position_y = 0


// Main simulation loop

While position_y >= 0 // Continue until projectile lands

  // Update position with current velocities

  position_x = position_x + initial_velocity_x * TIME_STEP

  position_y = position_y + initial_velocity_y * TIME_STEP
```

```
  // Update vertical velocity with gravity
  initial_velocity_y = initial_velocity_y - GRAVITY * TIME_STEP

  // If air resistance is considered
  If AIR_RESISTANCE Then
    velocity_magnitude = sqrt(initial_velocity_x^2 + initial_velocity_y^2)
    drag_force = 0.5 * DRAG_COEFFICIENT * AIR_DENSITY * velocity_magnitude^2
    drag_acceleration_x = (drag_force / projectile_mass) * (initial_velocity_x / velocity_magnitude)
    drag_acceleration_y = (drag_force / projectile_mass) * (initial_velocity_y / velocity_magnitude)

    // Update velocities with drag
    initial_velocity_x = initial_velocity_x - drag_acceleration_x * TIME_STEP
    initial_velocity_y = initial_velocity_y - drag_acceleration_y * TIME_STEP
  End If

  // Check for collision with target or obstacles
  If CheckCollision(position_x, position_y) Then
    HandleCollision()
    Break // Exit the loop if a collision occurs
  End If
End While

// Functions used above:
Function ConvertDegreesToRadians(degrees)
  Return degrees * (PI / 180)
End Function

Function CheckCollision(x, y)
```

```
    // Implement collision detection logic here
    // Return true if collision detected, else false
End Function

Function HandleCollision()
    // Implement what happens on collision, e.g., scoring, feedback
End Function
```