Unit Verification and Validation (VV) Plan

February 24, 2024

Revision History

Date	Version	Notes
February 3, 2024	1	Changed section 5
February 24, 2024	1.5	Changed section 9

Table of Contents

1. Symbols

- g: Acceleration due to gravity (m/s²)
- : Mathematical constant representing pi (approximately 3.14159265359)

1.1 Abbreviations

- SRS: Software Requirements Specification
- GUI: Graphical User Interface
- V&V: Verification and Validation
- FPS: Frames Per Second
- DT: Time step size for simulation
- 2D: Two-Dimensional
- API: Application Programming Interface
- OOP: Object-Oriented Programming
- HUD: Heads-Up Display

1.2 Acronyms

- API: Application Programming Interface
- GUI: Graphical User Interface
- IDE: Integrated Development Environment
- OOP: Object-Oriented Programming
- HUD: Heads-Up Display

2. General Information

2.1 Summary

The physics-based gaming application aims to provide an engaging gameplay experience centered around realistic collision dynamics and gravitational interactions. Users will manipulate objects, analyze trajectories, and apply physics principles in a challenging gaming environment. The game includes projectiles, targets/obstacles with different shapes and sizes, and incorporates gravitational forces along with the potential for additional challenges like wind. The project focuses on accurate physics simulations, user-friendly controls, and an immersive presentation of physics-based interactions.

2.2 Objectives

The primary objectives of the physics-based gaming application are as follows:

- Implement realistic collision interactions between rigid bodies (projectiles and targets/obstacles) based on initial conditions.
- Integrate gravitational forces that influence the trajectory and motion of rigid bodies.
- Consider additional forces, such as wind, to introduce complexity and challenge.
- Provide a physics engine that calculates the new configuration of the scene for rigid bodies.
- Define initial conditions for the scene, including locations, velocities, and forces acting upon rigid bodies.
- Allow specification of launch angle, force, and time step size for simulation accuracy.
- Output updated positions and velocities of rigid bodies after each simulation step.
- Enable user input controls for setting initial conditions and parameters.
- Implement controls for adjusting the time step size to control simulation accuracy.
- Utilize graphics to visually represent physics-based interactions, emphasizing rigid body collisions and movements.
- Implement sound effects corresponding to key physics events such as rigid body collisions and launches.

2.3 Relevant Documentation

The following documentation is relevant to the understanding and development of the physics-based gaming application:

- **Software Requirements Specification (SRS):** Detailed requirements for the application's functionality, performance, and constraints.
- **Code Documentation:** In-depth explanations of the code structure, modules, and functions.
- **Development Plan:** Outlines the planned phases, tasks, and timelines for application development.
- **Unit Verification and Validation (V&V) Plan:** Describes procedures for verifying and validating individual units or components.

3. Plan

3.1 Design Verification Plan

Objectives:

- Verify that the system design supports realistic physics simulation.
- Confirm that the design allows for scalability and integration of graphics and sound.

Activities:

1. **Physics Engine Design Inspection:**
    - Physics simulation experts inspect the design of the physics engine.
    - Ensure that it supports accurate collision dynamics, gravitational interactions, and additional forces.
2. **Graphics and Sound Integration Assessment:**
    - Evaluate the design for easy integration with graphics and sound libraries.
    - Confirm that visual and auditory feedback align with physics events.

Acceptance Criteria:

- Physics engine design meets accuracy requirements.
- Design supports seamless integration with graphics and sound.

3.2 Verification and Validation Plan Verification Plan

Objectives:

- Verify that the V&V plan covers all aspects crucial to the physics-based gaming application.

Activities:

1. **Plan Review:**
    - Verification team reviews the V&V plan.
    - Ensure that it includes specific tests for physics simulation, user interactions, and overall gaming experience.
2. **Simulation of Plan Execution:**
    - Simulate the execution of the V&V plan with hypothetical scenarios.
    - Identify any gaps or challenges in planned activities.

Acceptance Criteria:

- V&V plan covers all critical aspects of the physics-based gaming application.
- Simulated execution reveals no major gaps.

3.3 Implementation Verification Plan

Objectives:

- Verify that the implemented code accurately reflects the physics engine and integrates well with graphics and sound.

Activities:

1. **Physics Engine Code Review:**
    - Physics simulation experts conduct code reviews for the physics engine.
    - Ensure that the code accurately implements collision dynamics, gravitational interactions, and additional forces.
2. **Graphics and Sound Integration Testing:**
    - Test the integration of graphics and sound with the implemented code.
    - Confirm that visual and auditory feedback align with physics events.

Acceptance Criteria:

- Physics engine code reviews result in accurate implementations.
- Graphics and sound integration tests pass successfully.

4. SRS Verification Plan

Objectives:

- Verify the clarity and feasibility of physics-based requirements.
- Confirm that the SRS aligns with the intended gaming experience.

Activities:

1. **Physics Algorithm Review:**
    - Physics simulation experts review the algorithms proposed in the SRS.
    - Ensure the algorithms accurately model collision dynamics, gravitational interactions, and additional forces.
2. **User Experience Assessment:**
    - Game designers assess the SRS against the intended gaming experience.
    - Confirm that user interactions and visualizations align with the game's goals.

Acceptance Criteria:

- Physics experts confirm the accuracy of algorithms.
- Game designers approve the alignment of SRS with the gaming vision.

5. Verification Activities

5.1 Static Testing

Static testing is a verification activity that involves reviewing and analyzing code without executing it. This type of testing aims to identify issues in the early stages of development, ensuring that the code adheres to coding standards, is readable, and follows best practices.

5.1.1 Code Reviews

Purpose: The primary purpose of code reviews is to assess the correctness, maintainability, and adherence to coding standards of the implemented code. This activity involves a thorough examination of the code by team members.

Procedure:

1. **Selection of Reviewers:** Assign team members familiar with the specific module or unit to conduct the code review.
2. **Review Documentation:** Ensure that the documentation (Module Interface Specification, requirements) aligns with the implemented code.
3. **Check Coding Standards:** Confirm that the code follows the established coding standards, including naming conventions, indentation, and commenting.
4. **Functional Correctness:** Verify that the code implements the specified functionality and meets the requirements outlined in the MIS.
5. **Error Handling:** Assess the code for proper error handling and exception management.
6. **Interface Conformity:** Check if the interfaces are implemented as per the MIS, ensuring compatibility with other units.
7. **Review Comments:** Encourage reviewers to provide constructive comments and suggestions.
8. **Documentation Updates:** If necessary, update documentation based on the outcomes of the code review.

Output:

- Identified issues, if any, categorized by severity.
- Comments and suggestions for improvement.
- Documentation updates, if required.

5.1.2 Static Code Analysis

Purpose: Static code analysis involves using automated tools to analyze the source code without executing it. The aim is to identify potential issues related to coding standards, security vulnerabilities, and code complexity.

Procedure:

1. **Tool Selection:** Choose a static code analysis tool that is suitable for the programming language used (e.g., linters, code analyzers).
2. **Configuration:** Configure the tool based on the coding standards and rules applicable to the project.
3. **Run Static Analysis:** Execute the static code analysis tool on the codebase.
4. **Review Findings:** Examine the results of the analysis, paying attention to warnings, errors, and suggestions.
5. **Prioritize Issues:** Prioritize identified issues based on severity and impact.
6. **Address Issues:** Developers address identified issues by modifying the code as necessary.
7. **Re-run Analysis:** After addressing issues, re-run the static code analysis to ensure that problems have been resolved.

Output:

- Report from the static code analysis tool indicating issues found.
- Updated code reflecting changes made to address identified issues.

5.2 Dynamic Testing

Dynamic testing involves executing the code and evaluating its behavior during runtime. This type of testing aims to ensure that the units and integrated components of the software system perform as expected and meet the specified requirements.

5.2.1 Unit Testing

Purpose: Unit testing focuses on verifying the correctness of individual units or components within the physics-based gaming application. Each unit is tested in isolation to ensure that its functions and behaviors meet the specifications outlined in the Module Interface Specification (MIS).

Procedure:

1. **Test Case Design:** Develop test cases covering normal and boundary scenarios for each function within the unit.
2. **Isolation:** Ensure that each unit is isolated from the rest of the system to test its functionality independently.
3. **Test Execution:** Execute the designed test cases, including positive and negative scenarios.
4. **Assertion of Results:** Verify that the actual results match the expected results.

5. **Error Handling:** Test the unit's error-handling mechanisms by deliberately introducing invalid inputs.
6. **Coverage Analysis:** Assess code coverage to ensure that a significant portion of the unit's code is exercised by the tests.
7. **Iterative Refinement:** Refine test cases based on identified issues and re-run tests iteratively until all issues are resolved.

Output:

- Test results indicating pass/fail status for each test case.
- Log of executed tests and their outcomes.
- Identified defects and issues, if any.

5.2.2 Integration Testing

Purpose: Integration testing verifies the interactions between units or components within the physics-based gaming application. It ensures that integrated components collaborate correctly and share data as expected.

Procedure:

1. **Integration Test Plan:** Develop a plan outlining the sequence of integration and the units to be integrated.
2. **Interface Verification:** Verify that the interfaces between units function as intended, including proper data exchange.
3. **Incremental Integration:** Integrate units incrementally, starting with the most critical components.
4. **Test Execution:** Execute test cases designed for the integrated components.
5. **Scenario Testing:** Test realistic scenarios, including user interactions and physics simulations.
6. **Error Scenarios:** Assess how the system handles unexpected errors and edge cases.
7. **Performance Testing:** Evaluate the system's performance under expected load conditions.
8. **Regression Testing:** Ensure that previously working functionalities remain intact after new integrations.

Output:

- Results of integration tests, indicating pass/fail status.
- Log of executed tests and their outcomes.
- Identified defects and issues, if any.

6. Validation Activities

6.1 User Scenario Testing

User scenario testing involves validating the physics-based gaming application against realistic user interactions and scenarios. The goal is to ensure that the application meets user expectations and functions correctly in various usage contexts.

6.1.1 Simulation

Purpose: Simulation testing aims to replicate real-world scenarios within the gaming application to validate its behavior under different conditions. This includes testing user interactions, physics simulations, and responses to user inputs.

Procedure:

1. **Scenario Identification:** Define realistic user scenarios, considering different gameplay situations and interactions.
2. **Test Data Preparation:** Prepare test data, including initial conditions, launch parameters, and environmental settings.
3. **Simulation Execution:** Execute the defined scenarios within the application, ensuring that the physics simulations respond accurately.
4. **User Interaction Validation:** Validate the user interface's responsiveness and correctness during simulated scenarios.
5. **Error Scenarios:** Test the application's behavior in error scenarios, such as invalid user inputs or unexpected events.
6. **Documentation of Results:** Document the outcomes of each simulation, including any deviations from expected behavior.
7. **Iteration and Refinement:** If issues are identified, refine the scenarios and repeat the simulation testing iteratively.

Output:

- Results of simulation tests, indicating pass/fail status for each scenario.
- Log of executed scenarios and their outcomes.
- Identified defects and issues, if any.

6.2 Performance Testing

Performance testing assesses the responsiveness, stability, and scalability of the physics-based gaming application under different conditions.

6.2.1 Load Testing

Purpose: Load testing focuses on evaluating the application's performance under expected load conditions, ensuring that it can handle a specified number of users and interactions without degradation in performance.

Procedure:

1. **Load Scenario Definition:** Define realistic load scenarios, considering the expected number of concurrent users and user interactions.
2. **Test Environment Setup:** Set up a test environment that simulates the expected production environment.
3. **Load Execution:** Execute the load scenarios to simulate concurrent users interacting with the application.
4. **Performance Monitoring:** Monitor key performance metrics, including response times, resource utilization, and system stability.
5. **Stress Testing:** Gradually increase the load to assess the application's behavior under stress conditions.
6. **Documentation of Results:** Document the performance metrics and any issues encountered during load testing.
7. **Analysis and Optimization:** Analyze the results, identify performance bottlenecks, and optimize the application as needed.
8. **Reporting:** Provide a comprehensive report detailing the performance characteristics and any recommendations for improvement.

Output:

- Performance metrics, including response times, throughput, and resource utilization.
- Log of executed load scenarios and their outcomes.
- Identified performance bottlenecks and issues, if any.

7. Schedule

- **Static Testing:**
  - Code Reviews: Concurrently during the coding phase.
  - Static Code Analysis: Concurrently during the coding phase.
- **Dynamic Testing:**
  - Unit Testing: Ongoing as each unit is developed.
  - Integration Testing: As units are integrated, following completion of unit testing.
- **Validation Activities:**

  - User Scenario Testing: Conducted after successful completion of dynamic testing.
  - Performance Testing: Conducted after user scenario testing.
- **Exit Criteria Review:**
  - Conducted after completion of all testing and validation activities.
- **Documentation Updates:**
  - Ongoing throughout the verification and validation process.

8 System Test Description

8.1 Tests for Functional Requirements

8.1.1 Area of Testing

Test Case 1: Projectile Motion

Objective: To verify that the game accurately implements projectile motion for the launched objects.

Steps:

1. Launch a projectile with specific initial conditions.
2. Observe the trajectory of the projectile.
3. Verify that the trajectory follows the expected projectile motion equations.

Test Case 2: Collision Detection

Objective: To ensure that collision detection functions correctly between game elements.

Steps:

1. Launch multiple projectiles towards targets.
2. Verify that collisions are detected accurately based on bounding box comparison.
3. Confirm that collision events trigger appropriate responses in the game.

8.1.2 Area of Testing

Test Case 3: User Controls

Objective: To validate that user controls enable players to interact with the game effectively.

Steps:

1. Test keyboard inputs for launching projectiles.
2. Verify that users can set initial conditions and adjust parameters easily.

3. Confirm that the specified controls are intuitive for players.

Test Case 4: Graphics and Sound

Objective: To assess the implementation of graphics and sound effects in representing physics-based interactions.

Steps:

1. Evaluate the visual representation of rigid body collisions and movements.
2. Verify that sound effects correspond to key physics events (collisions, launches).
3. Confirm that the overall sensory experience enhances gameplay.

8.2 Tests for Nonfunctional Requirements

8.2.1 Area of Testing

Test Case 5: Performance

Objective: To ensure the game meets performance requirements.

Steps:

1. Monitor and measure frames per second (FPS) during gameplay.
2. Assess the efficiency of collision detection and response algorithms.
3. Confirm that the game maintains smooth performance under different scenarios.

8.2.2 Area of Testing

Test Case 6: Usability

Objective: To evaluate the usability of the game.

Steps:

1. Conduct user testing to assess the intuitiveness of controls.
2. Gather feedback on the overall user interface and experience.
3. Evaluate the ease of understanding and using the game features.

8.3 Traceability Between Test Cases and Requirements

Traceability Matrix:

Tast case	**Requirement IDs**
Test case 1	FR-1, FR-2
Test case 2	FR-3, FR-4
Test case 3	FR-5, FR-
Test case 4	FR-7, FR-8

Tast case	Requirement IDs
Test case 5	FR-1, FR-2
Test case 6	FR-3, FR-4

9 Unit Test Description

9.1 Unit Testing Scope

Unit testing focuses on testing individual modules or components of the system to ensure their correctness and functionality. The scope of unit testing includes validating both functional and nonfunctional aspects of each module.

9.2 Tests for Functional Requirements

9.2.1 Module: Physics Engine

Unit Test 1: Test Projectile Motion Calculation

Objective: To verify that the physics engine correctly calculates the trajectory of a projectile based on initial conditions.

Steps:

1. Provide a set of initial conditions for a projectile.
2. Execute the projectile motion calculation.
3. Verify that the calculated trajectory matches the expected results.

Unit Test 2: Test Collision Dynamics

Objective: To ensure that the physics engine accurately detects collisions between rigid bodies.

Steps:

1. Simulate the collision between two objects with known initial conditions.
2. Check if the collision detection algorithm identifies the collision.
3. Confirm that the collision response preserves momentum and kinetic energy.

9.2.2 Module: User Input Controls

Unit Test 3: Test Launch Control

Objective: To validate that the module responsible for user input controls effectively triggers the launch action.

Steps:

1. Simulate user input for launching an object.
2. Check if the launch action is initiated.

3. Verify that the specified initial conditions are applied to the launched object.

Unit Test 4: Test Parameter Adjustment

Objective: To confirm that users can adjust launch parameters effectively.

Steps:

1. Simulate user input for adjusting launch parameters.
2. Check if the specified parameters (launch angle, force) are updated.
3. Verify that the changes influence the subsequent simulation.

9.3 Tests for Nonfunctional Requirements

9.3.1 Module: Performance Optimization

Unit Test 5: Test Computational Efficiency

Objective: To assess the computational efficiency of algorithms in the performance optimization module.

Steps:

1. Measure the execution time of critical algorithms.
2. Compare the results against performance requirements.
3. Optimize algorithms if necessary to meet performance criteria.

9.3.2 Module: Usability Enhancement

Unit Test 6: Test User Interface Clarity

Objective: To ensure that the module for usability enhancement improves the clarity of the user interface.

Steps:

1. Evaluate changes made to the user interface.
2. Gather feedback on the clarity and intuitiveness of controls.
3. Confirm that users find the interface enhancements beneficial.

9.4 Traceability Between Test Cases and Modules

Traceability Matrix:

Unit Test	Module
Unit Test 1	Physics engine
Unit Test 2	Physics engine
Unit Test 3	User input Controls
Unit Test 4	User input Controls

Unit Test	Module
Unit Test 5	Performance Optimization
Unit Test 6	Usability Enhancement

10. Exit Criteria

Exit criteria define the conditions that must be met before concluding the verification and validation activities for the physics-based gaming application. The exit criteria include:

- Successful completion of all static and dynamic testing activities.
- All identified defects and issues addressed and resolved.
- Code coverage meets the defined threshold.
- User scenario testing and performance testing meet predefined acceptance criteria.
- Documentation is updated to reflect any changes made during testing.
- Approval from stakeholders or the quality assurance team.

11. Responsibilities

Responsibilities for verification and validation activities are distributed as follows:

- **Developers:**
  - Responsible for coding and addressing issues identified during code reviews.
  - Conduct unit testing for individual units.
- **Testers:**
  - Conduct integration testing and user scenario testing.
  - Perform performance testing.
- **Project Managers:**
  - Oversee the entire verification and validation process.
  - Ensure that activities are conducted according to the schedule.
  - Approve exit criteria for each testing phase.

12. Tools and Resources

Tools and resources used during the verification and validation activities include:

- **Code Review Tools:** Integrated into the development environment for collaborative code reviews.

- **Static Code Analysis Tools:** Automated tools for analyzing source code.
- **Testing Frameworks:** Used for unit testing and integration testing.
- **Performance Testing Tools:** Tools to simulate load conditions and monitor performance metrics.
- **Documentation Tools:** Tools for updating and maintaining project documentation.

13. Risks and Mitigation Strategies

Risks:

1. **Resource Constraints:** Insufficient resources for testing activities.
    - **Mitigation:** Prioritize critical testing activities and allocate resources accordingly.
2. **Changing Requirements:** Unstable or changing requirements impacting testing efforts.
    - **Mitigation:** Establish clear communication channels for requirement changes and conduct thorough impact analysis.
3. **Tool Limitations:** Issues with the performance or compatibility of testing tools.
    - **Mitigation:** Regularly update and maintain testing tools and have contingency plans in case of tool failures.
4. **Integration Challenges:** Difficulties in integrating individual units into a cohesive system.
    - **Mitigation:** Conduct incremental integration testing, addressing integration issues as they arise.
5. **Performance Bottlenecks:** Identification of performance bottlenecks during load testing.
    - **Mitigation:** Optimize code, databases, and infrastructure based on performance testing results.

Contingency Plans:

- Establish a rollback plan in case issues arise during testing.
- Regularly update risk assessments and mitigation strategies throughout the verification and validation process.
- Have contingency testing plans for critical scenarios to ensure the application's stability.

14. Appendix

14.1 Symbolic Parameters

Symbolic parameters are variables or placeholders used in mathematical equations or models. In the context of the physics-based gaming application, certain parameters are represented symbolically for flexibility and ease of adjustment. These parameters contribute to the dynamic nature of the simulation and can be tuned for different gameplay experiences.

14.1.1 Symbolic Parameters List

1. v0 : Initial velocity vector.
2. r0 : Initial position vector.
3. Fnet : Net force vector.
4. t : Time step size.
5. a : Acceleration vector.

These parameters play a crucial role in defining the behavior of game elements and their interactions within the physics simulation.

14.2 Usability Survey Questions

A **usability survey** helps gather feedback from users to evaluate the effectiveness and user-friendliness of the application. The following set of questions can be included in a usability survey for the physics-based gaming application:

14.2.1 Usability Survey Questions

1. **On a scale of 1 to 5, how would you rate the overall user interface of the game in terms of clarity and ease of use?**
    - 1: Very Poor
    - 2: Poor
    - 3: Neutral
    - 4: Good
    - 5: Excellent
2. **Did you find the controls intuitive for manipulating the initial conditions and parameters of the simulation?**
    - Yes
    - No
3. **How satisfied are you with the visual representation of physics-based interactions in the game?**
    - Very Dissatisfied

- Dissatisfied
- Neutral
- Satisfied
- Very Satisfied

4. **Were the sound effects corresponding to key physics events (collisions, launches) effective and engaging?**
    - Yes
    - No
5. **Did you encounter any difficulties or confusion while adjusting launch angles, forces, or other physics parameters? If yes, please specify.**
6. **How would you rate the responsiveness and smoothness of the gameplay in terms of graphics rendering and collision detection?**
    - 1: Very Poor
    - 2: Poor
    - 3: Neutral
    - 4: Good
    - 5: Excellent
7. **Would you like to see any additional features or improvements in future updates of the game? If yes, please provide suggestions.**
8. **On a scale of 1 to 5, how likely are you to recommend this game to others?**
    - 1: Not Likely
    - 2: Somewhat Likely
    - 3: Neutral
    - 4: Likely
    - 5: Very Likely
9. **Please share any other comments, feedback, or concerns you have about the physics-based gaming application.**

Physics-Based Game Working Principle with Formulas

1. Objective:

- The primary objective of the game is to launch projectiles (birds) to hit and eliminate target objects (pigs) by utilizing realistic physics principles.

2. **Projectile Launch:**
   - Players drag a projectile (bird) on the slingshot, adjusting the launch angle () and force (F).
   - The launch angle and force determine the initial velocity (v0) of the projectile.

     v0 = F/m

3. **Distance Measurement:**
   - Distance is measured from the slingshot to the target using a realistic physics-based algorithm.
   - The horizontal distance (d) is calculated using the projectile motion formula:

     D = v0ˆ2 sin(2)/g

4. **Trajectory Calculation:**
   - Trajectory calculation involves using kinematic equations to predict the bird's path based on its initial conditions (launch angle, force).
   - The vertical position (y) of the projectile over time (t) is given by:

     Y = v0sin()t 1/2gtˆ2

5. **Collision Detection:**
   - Collision detection is implemented to identify when the projectile collides with game objects (structures, pigs).
   - Bounding box or pixel-perfect collision algorithms are utilized for accurate detection.

6. **Hitting Targets (Pigs):**
   - The objective is to hit and eliminate pigs strategically placed within the game environment.
   - Scoring is based on the number of pigs hit and the destruction caused to the surrounding structures.
   - Pigs may have different properties affecting the gameplay, such as size and durability.

7. **Environmental Interaction:**
   - The game environment includes structures, obstacles, and other elements influenced by physics.
   - Objects can be affected by the projectile's impact, leading to realistic destruction and movement.
   - Dynamic interactions contribute to the challenge and creativity of gameplay.

8. **Realistic Physics Behaviors:**
   - The game adheres to real-world physics principles, including gravity, projectile motion, and collision dynamics.
   - Developers use accurate physics formulas to ensure a realistic and engaging gaming experience.
9. **Scoring System:**
   - Scoring is based on the efficiency of hitting pigs and causing destruction.
   - Players are awarded points for accuracy, creativity in strategy, and the efficiency of resource usage (number of birds launched).
10. **Iterative Gameplay:**
    - The game features multiple levels with increasing complexity, introducing new challenges and variations in environmental elements.
    - Players progress through levels by successfully completing objectives, requiring strategic thinking and skill.

Setting Distance, Projectile Launch, and Usability:

1. **Setting the Distance:**
   - The game should allow players to set the launch distance by adjusting the slingshot.
   - **Physics Consideration:**
     - The horizontal distance is determined by the launch angle and initial velocity of the projectile.
     - Adjusting the slingshot directly impacts the launch angle and, consequently, the distance.
2. **Launching Projectiles:**
   - Players should be able to launch projectiles with varying initial conditions.
   - **Physics Consideration:**
     - The launch angle () and force (F) directly affect the initial velocity (v0) of the projectile.
     - Players can experiment with different launch parameters to optimize their strategy.
3. **Usability:**
   - The game interface should be intuitive, allowing players to easily interact with the slingshot and adjust launch parameters.
   - **Physics Consideration:**

- Ensure that the controls for adjusting launch angle and force are user-friendly.
- Provide visual indicators of the launch trajectory to assist players in making accurate shots.

4. Test Cases:

a. Projectile Landing Within Boundaries:

- Objective: Verify that the projectile lands within the game boundaries.
- Steps:
  - Launch a projectile with varying parameters.
  - Check if the projectile's trajectory and landing position are within the defined play area.
- Expected Result: Projectile lands within the game boundaries.

b. Projectile Landing Outside Boundaries:

- Objective: Test the response when a projectile lands outside the game boundaries.
- Steps:
  - Intentionally launch a projectile to land outside the play area.
  - Observe the game's reaction and handling of out-of-bounds scenarios.
- Expected Result: The game responds appropriately, considering the projectile's position outside the boundaries.

c. Distance Measurement Accuracy:

- Objective: Validate the accuracy of the distance measurement.
- Steps:
  - Launch projectiles at various distances and measure the horizontal displacement.
  - Compare the measured distances with the expected values based on physics formulas.
- Expected Result: The measured distances align closely with the physics-calculated values.

d. Launch Parameter Adjustment:

- Objective: Confirm the effectiveness of adjusting launch parameters.
- Steps:
  - Vary launch angle and force settings and launch projectiles.

- Observe how changes in launch parameters impact the trajectory and distance.

- Expected Result: Adjusting launch parameters influences the projectile's behavior as per physics principles.

e. Collision Detection:

- Objective: Ensure accurate collision detection with game objects (structures, pigs).
- Steps:
  - Launch projectiles towards structures and pigs.
  - Verify that collisions are detected accurately based on the physics of projectile-object interactions.
- Expected Result: Collisions are detected precisely, reflecting real-world physics.

f. Trajectory Prediction:

- Objective: Test the accuracy of the trajectory prediction.
- Steps:
  - Launch projectiles with known initial conditions.
  - Compare the predicted trajectory with the actual trajectory during flight.
- Expected Result: The predicted trajectory closely matches the actual trajectory based on physics calculations.

Flowchart for Physics-Based Bird Launch

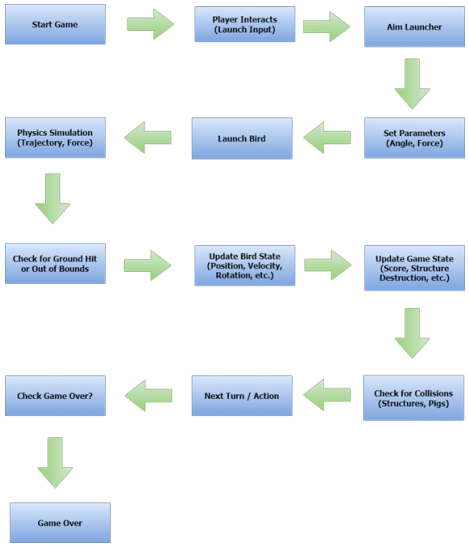

Figure 1: flow chart