System Requirements Specification (SRS) for Physics-Based Game

February 29, 2024

Revision History

Date	Version	Notes
February 3, 2024	1	Changed section (6, 7, 11)
February 26, 2024	2	Changed section (1.4, 4.2.1, 4.4.2, 4.4.8)

Table of Contents

1. Reference Material

1.1 Table of Units

Unit	Description
m	Meter
sec	Second
kg	Kilogram
N	Newton

1.2 Table of Symbols

Symbol	Description
R	Position Vector
V	Velocity Vector
F	Force Vector
M	Mass
G	Gravitational acceleration vector
	Launch angle
Fwind	Wind force

1.3 Abbreviations and Acronyms

Abbreviation	Description
SRS	Software requirement Specification
UI	User interface
API	Application Programming interface
FPS	Frames per second

1.4 Mathematical Notation

- v0: Initial velocity vector
- r0: Initial position vector
- Fnet : Net force vector
- t : Time step size
- a : Acceleration vector

2. Introduction

2.1 Purpose of the Document

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive and detailed outline of the requirements for the development of a physics-based gaming application. It serves as a guide for the development team, stakeholders, and any parties involved in the project, ensuring a clear understanding of the features, functionalities, and constraints of the proposed gaming application.

2.2 Scope of Requirements

The scope of this document encompasses the specifications and requirements for the physics-based gaming application, emphasizing realistic collision dynamics and gravitational interactions. It includes the essential features such as projectile motion, collision handling, gravitational forces, additional forces (e.g., wind), user controls, graphics, and sound effects. The document also outlines the external interfaces, non-functional requirements, and other aspects relevant to the successful development of the gaming application.

2.3 Characteristics of Intended Reader

- **Development Team:** Software engineers, programmers, and designers involved in the implementation of the physics-based gaming application.
- **Project Stakeholders:** Individuals or groups with a vested interest in the success of the project, such as project managers, quality assurance teams, and sponsors.
- **Testing Team:** Professionals responsible for testing the application against specified requirements.
- **End Users:** Gamers and individuals interested in understanding the technical aspects and functionalities of the gaming application.

3. General System Description:

3.1 System Context

The physics-based gaming application, as represented in the code, operates within a console-based environment, interacting with the user and simulating a simplified gaming experience.

- **External Systems:**
  - **Console Environment:** The application relies on console-based input/output for user interaction and rendering game elements.
  - **Graphics and Sound Libraries:** While the current implementation lacks explicit integration with external graphics and sound libraries, future development might benefit from incorporating these elements to enhance the gaming experience.
- **Users:**
  - **Console Users (Gamers):**

    - Interact with the game through keyboard inputs in the console.
    - Experience a simplified representation of game elements through ASCII art.
    - May be familiar with console-based games and basic keyboard controls.
  - **Developers:**
    - Utilize C++ as the programming language for development.
    - Require a console environment for testing and debugging.
    - May enhance the application by integrating graphics and sound libraries in future iterations.

3.2 User Characteristics

- **Console Users (Gamers):**
  - Comfortable with simple keyboard controls ('w', 'a', 'd', 'r', 'q') for gameplay.
  - Familiar with console-based interactions.
  - Seeking a basic, retro-style gaming experience.
- **Developers:**
  - Proficient in C++ programming.
  - Familiar with console-based development and debugging.
  - May have the capability and interest in extending the application with more advanced features.

3.3 System Constraints

- **Hardware Constraints:**
  - The application's performance is constrained by the capabilities of the console environment, limiting advanced graphical and audio features.
- **Physics Simulation Constraints:**
  - The physics simulation is simplistic, suitable for a console-based environment but may lack the complexity of a full-fledged physics engine.
- **User Interface Constraints:**
  - The user interface is confined to console-based interactions, restricting the visual representation and overall user experience.

4. Specific System Description

4.1 Problem Description

The physics-based gaming application aims to create an engaging and challenging gaming experience where players can manipulate objects, analyze trajectories, and apply physics principles within a dynamic and interactive virtual environment. The primary focus is on realistic collision dynamics, gravitational interactions, and the integration of additional forces, such as wind.

4.1.1 Terminology and Definitions

- **Projectile:** A game element, represented as an object (e.g., bird, pig), possessing an initial position, velocity, and mass.
- **Targets/Obstacles:** Various objects within the gaming environment, each having different shapes and sizes, serving as elements for interaction and challenge.
- **Physics Simulation:** The process of modeling and simulating realistic collision interactions and gravitational forces between rigid bodies within the gaming environment.
- **Gravitational Forces:** The influence of gravity on the trajectory and motion of rigid bodies, affecting their movement within the virtual space.
- **Additional Forces (e.g., Wind):** Extra forces applied to rigid bodies to introduce complexity and challenge, enhancing the overall gameplay experience.
- **Physics Engine:** The computational framework responsible for accurately modeling collision dynamics, gravitational interactions, and other physical phenomena within the gaming environment.

4.1.2 Physical System Description

The physical system of the gaming application comprises virtual entities that adhere to the principles of physics. These entities include:

- **Projectile:** An object with defined mass, initial position, and velocity, subject to the laws of motion and gravitational forces.
- **Targets/Obstacles:** Diverse objects within the gaming environment, each with specific geometric characteristics, interacting with projectiles through collisions.
- **Gravitational Field:** A virtual representation of gravity, influencing the trajectories of projectiles and contributing to the overall physics simulation.
- **Additional Forces (e.g., Wind):** External forces that impact the motion of projectiles, introducing variability and complexity to the gameplay.

1. **Scalability:** Design the system to be extensible for potential future expansions, allowing for the addition of new features, levels, and challenges.

4.2 Solution Characteristics Specification

The Solution Characteristics Specification outlines the key features and types that define the characteristics of the physics-based gaming application.

4.2.1 Types

4.2.1.1 Game Elements

1. **Projectile Type:**
   - *Example Instance:*
     - Initial Position: (0, 0)
     - Initial Velocity: (5, 5)
     - Mass: 1.0
2. **Targets/Obstacles Type:**
   - *Example Instance:*
     - Position: (10, 8)
     - Shape: Rectangular
     - Size: 3 units

4.2.1.2 Physics Simulation

3. **Collision Dynamics Type:**
   - *Example Instance:*
     - Collision Detection Algorithm: Pixel-perfect collision
     - Collision Response Algorithm: Destructive collision (objects break apart)
4. **Gravitational Forces Type:**
   - *Example Instance:*
     - Gravitational Acceleration: 0 (ignoring gravity for simplicity)
5. **Additional Forces Type (e.g., Wind):**
   - *Example Instance:*
     - Type of Force: Wind
     - Magnitude and Direction: (2, 0) units/s^2 (horizontal)
6. **Physics Engine Type:**
   - *Example Instance:*
     - Simulation Accuracy: Medium

- Computational Efficiency: Balanced

4.2.1.3 User Interaction

7. **Physics Inputs Type:**
   - *Example Instance:*
     - Initial Positions and Velocities: Player sets the initial launch conditions
     - External Forces: None initially
8. **Physics Outputs Type:**
   - *Example Instance:*
     - Updated Positions and Velocities: Calculated after each simulation step

4.2.1.4 User Controls

9. **User Input Controls Type:**
   - *Example Instance:*
     - Launch Angle: Adjustable via touch or drag
     - Force: Adjustable via touch or drag
     - Time Step Size: Fixed at 0.1 seconds
10. **Time Step Size Control Type:**
    - *Example Instance:*
      - Time Step Size: Fixed at 0.1 seconds

4.2.1.5 Graphics and Sound

11. **Graphics Type:**
    - *Example Instance:*
      - Visual Effects: Cartoonish animations on collision
      - Graphics Rendering Engine: Custom 2D engine
12. **Sound Effects Type:**
    - *Example Instance:*
      - Collision Sounds: Comical crashing sounds
      - Launch Sounds: Slingshot stretching and bird squawking

4.3 Scope Decisions

The scope decisions outline the specific boundaries and inclusions for the development of the physics-based gaming application. These decisions help define the

project's limits and guide the development team throughout the implementation process.

4.3.1 Inclusions

1. **Realistic Physics Simulation:**
    - Inclusion: The application will include a physics engine that accurately models collision dynamics, gravitational interactions, and additional forces (e.g., wind) to create a realistic gaming experience.
2. **Engaging Gameplay:**
    - Inclusion: The gameplay will focus on challenging players to apply physics principles strategically, encouraging experimentation and skill development.
3. **User-Friendly Interface:**
    - Inclusion: The application will provide a user-friendly interface allowing players to manipulate initial conditions, adjust parameters, and actively participate in the simulation.
4. **Visual and Auditory Feedback:**
    - Inclusion: Graphics will be used to visually represent physics-based interactions, emphasizing rigid body collisions and movements. Sound effects corresponding to key physics events will be implemented to enhance the overall sensory experience.
5. **Dynamic Challenges:**
    - Inclusion: Additional forces, such as wind, will be introduced to create dynamic challenges, requiring adaptability and skill from the players.
6. **Configurability:**
    - Inclusion: Users will be able to specify initial conditions, launch angles, forces, and time step sizes, providing a customizable and personalized gaming experience.
7. **Scalability:**
    - Inclusion: The system will be designed to be extensible for potential future expansions, allowing for the addition of new features, levels, and challenges.

4.4 Modelling Decisions

In the modelling decisions, various assumptions, theoretical models, and data definitions are established to guide the development of the physics-based gaming application.

4.4.1 Assumptions

4.4.1.1 Environment Assumptions

1. **Dimensional Constraints:**
    - Assumption: The gaming environment is strictly two-dimensional, simplifying the physics simulation and rendering processes. There is no need to account for the complexities of a three-dimensional space.

4.4.1.2 Gravity Assumptions

2. **Uniform Gravitational Field:**
    - Assumption: Gravity acts uniformly on all objects, neglecting variations due to altitude, geographic location, or other factors. This simplification ensures a consistent and predictable gravitational force throughout the game.

4.4.1.3 Collision Dynamics Assumptions

3. **Elastic Collisions:**
    - Assumption: Collisions between objects are assumed to be perfectly elastic, meaning kinetic energy is conserved during interactions. This simplification allows for straightforward and predictable object responses in the game.

4.4.1.4 Wind Force Assumptions

4. **Constant Wind Forces:**
    - Assumption: Wind forces acting on objects are considered constant within a simulation step, simplifying their application. Dynamic or varying wind conditions are not taken into account for simplicity in the physics calculations.

4.4.1.5 Time Step Assumptions

5. **Fixed Time Step:**
    - Assumption: The physics simulation operates on a fixed time step, simplifying calculations and ensuring a consistent update rate. This assumption may introduce minor inaccuracies but contributes to a stable and predictable simulation.

4.4.1.6 Object Properties Assumptions

6. **Homogeneous Object Properties:**
    - Assumption: Objects within the game environment are considered homogeneous in terms of physical properties, such as mass and size. This simplification streamlines the physics calculations for various object interactions.

4.4.1.7 Friction and Air Resistance Assumptions

7. **Simplified Friction Model:**
    - Assumption: The friction and air resistance models applied to objects are simplified, neglecting certain complexities such as surface irregularities. This simplification is made for computational efficiency and ease of implementation.

4.4.2 Theoretical Models

1. **Projectile Motion Model:**
    - Model: Utilizes classical projectile motion equations to determine the trajectory of the projectile based on initial conditions and external forces.

Equations:

- X (t) = x0 + v0x t
- y (t) = y0 + v0y t – 1/2 gtˆ2

2. **Collision Dynamics Model:**
    - Model: Employs simplified collision detection and response algorithms to handle interactions between rigid bodies.

Equations:

- Collision Detection: Determine if two objects A and B collide.
- Collision Response: Adjust the velocities of colliding objects based on the collision type.

3. **Gravitational Model:**
    - Model: Applies Newton's law of universal gravitation to compute the gravitational forces acting on each object.

Equation:

- F = Gm1 m2/rˆ2

4. **Wind Force Model:**
    - Model: Introduces an additional force (wind) with constant magnitude and direction to create dynamic challenges.

Equation:

FWind = Magnitude × Direction

4.4.3 General Definitions

1. **Position:**
    - Definition: The location of an object in the two-dimensional gaming environment, represented by coordinates (x, y).

2. **Velocity:**
    - Definition: The rate of change of an object's position with respect to time, represented by components (vx, vy).

4.4.4 Data Definitions

1. **Projectile Data:**
    - Definition: Data structure representing a projectile with attributes for position, velocity, and mass.
2. **Target/Obstacle Data:**
    - Definition: Data structure representing a target/obstacle with attributes for position, shape, and size.

4.4.5 Data Types

1. **Position Data Type:**
    - Type: Double precision floating-point numbers (x, y).
2. **Velocity Data Type:**
    - Type: Double precision floating-point numbers (vx, vy).
3. **Mass Data Type:**
    - Type: Double precision floating-point number representing the mass of an object.

4.4.6 Instance Models

1. **Projectile Instance Model:**
    - Model: An instance of the projectile data structure with specific values for position, velocity, and mass.
2. **Target/Obstacle Instance Model:**
    - Model: An instance of the target/obstacle data structure with specific values for position, shape, and size.

4.4.7 Input Data Constraints

1. **Launch Angle Constraint:**
    - Constraint: Launch angles are limited to a predefined range (e.g., 0 to 90 degrees) for realistic projectile motion.
2. **Force Constraint:**
    - Constraint: Forces applied to projectiles are within a specified range to maintain a balanced gaming experience.

4.4.8 Properties of a Correct Solution

1. **Conservation of Energy:**
    - Property: The simulation maintains the conservation of energy, ensuring that the total energy of the system remains constant.

The conservation of energy is expressed by the equation:

Etotal = Ekinetic + EPotential

Where:

- Etotal is the total energy of the system.
- Ekinetic is the kinetic energy associated with the motion of objects.
- Epotential the potential energy associated with the position of objects.

2. **Accuracy of Collision Response:**
    - Property: The collision response accurately reflects the conservation of momentum and kinetic energy.
3. **Realistic Projectile Motion:**
    - Property: The projectile motion adheres to realistic physics principles, considering launch angles, initial velocities, and gravitational forces.
4. **Dynamic Gameplay:**
    - Property: The introduction of wind forces creates dynamic and challenging gameplay scenarios.

5. Requirements:

5.1 Functional Requirements

5.1.1 Game Physics

1. The game shall implement gravity using the formula: **y = y_initial + v_initial * t - 0.5 * g * t^2**.
2. Horizontal movement shall be implemented using the formula: **x = x_initial + v_horizontal * t**.
3. Collision detection between objects shall be based on bounding box comparison.
4. Elastic collision response shall be implemented for basic object interaction.

5.1.2 Game Elements

1. The game shall include objects with properties such as position, velocity, and dimensions.
2. Objects shall have sprite images representing their visual appearance.

5.1.3 User Interaction

1. Players shall interact with the game through controls to trigger actions such as launching objects.

5.1.4 Use Cases

Use Case 1: Launching an Object

Primary Actor: Player

Description: The player interacts with the game to launch an object.

Preconditions:

- The game is in an active state.

Steps:

1. The player selects the object to launch.
2. The player sets the launch parameters.
3. The player triggers the launch action.

Postconditions:

- The launched object follows the specified trajectory.

Use Case 2: Collision Detection

Primary Actor: Game Engine

Description: The game engine detects collisions between game objects.

Preconditions:

- Game objects are in motion.

Steps:

1. Continuously monitor the positions of game objects.
2. Detect collisions based on bounding box comparison.

Postconditions:

- Collision events trigger appropriate responses in the game.

5.2 Non-functional Requirements

5.2.1 Performance

1. **Minimum Frame Rate:**
    - The game shall maintain a minimum frame rate of 30 frames per second (FPS) to ensure smooth and enjoyable gameplay.
2. Efficient Collision Detection and Response:

- Collision detection algorithms and response mechanisms shall be optimized to ensure efficient processing and minimal impact on gameplay performance.

5.2.2 Usability

1. **Intuitive User Interface:**
   - The game shall feature an intuitive and visually appealing user interface, ensuring that players can easily navigate through menus and interact with game elements.
2. **User-friendly Controls:**
   - Controls for launching objects and other interactions shall be designed to be easy to understand, responsive, and comfortable for players to use.

5.2.3 Reliability

1. **Stability in Physics Calculations:**
   - The game's physics calculations, including gravity and object interactions, shall be implemented in a stable and reliable manner to minimize bugs and unexpected behaviors.
2. **Robust Collision Handling:**
   - The collision handling system shall be robust, capable of accurately detecting collisions and responding appropriately to ensure the integrity of the game's physics simulation.

Non-functional Requirements Physics Simulation

1. **Realism and Accuracy:**
   - The physics simulation shall aim for a balance between realism and gameplay, ensuring that the behavior of game objects aligns with real-world physics principles.
   - Deviations from real-world physics shall be intentional and communicated clearly to players.
2. **Stability in Complex Scenes:**
   - The physics engine shall be capable of maintaining stability and accuracy even in scenes with a high number of interacting objects or complex geometric shapes.
3. **Dynamic Time Step:**
   - The physics simulation shall incorporate a dynamic time step mechanism to adapt to varying processing speeds, maintaining stability and accuracy across different hardware configurations.

5.2.5 Object Interactions

1. **Elasticity Consistency:**
    - Objects in the game shall exhibit consistent elasticity in collisions, providing predictable reactions based on their physical properties.
2. **Friction and Drag:**
    - The game shall simulate realistic friction and air resistance, affecting the motion of objects in a manner that aligns with real-world physics.
3. **Object Material Properties:**
    - Game objects shall have customizable material properties (e.g., density, elasticity, friction) to allow developers to fine-tune the physics simulation for specific gameplay scenarios.

5.2.6 Debugging and Analysis Tools

1. **Physics Debugging Tools:**
    - The game shall include built-in tools for debugging and visualizing physics interactions, helping developers identify and address issues related to object movements, collisions, and forces.
2. **Performance Profiling:**
    - Tools for performance profiling of the physics engine shall be available, allowing developers to identify bottlenecks and optimize the simulation for better performance.

5.2.7 Adaptability and Extensibility

1. **Modular Physics Components:**
    - The physics system shall be designed with modularity in mind, allowing for easy integration of additional physics features or third-party physics libraries.
2. **Parameter Tuning:**
    - The game shall provide parameters that can be tuned to adjust the overall physics behavior, catering to different game genres or design preferences.

6. Likely Changes:

6.1 Physics Model Refinement:

Recognize the possibility of refining the physics model to enhance realism based on player feedback or advancements in simulation techniques.

6.2 Additional Forces:

Anticipate the introduction of new forces or environmental factors for added complexity and variety in gameplay.

6.3 User Interface Enhancements:

Acknowledge potential adjustments to the user interface to improve user experience and accommodate additional controls.

7. Unlikely Changes:

7.1 Core Physics Algorithms:

Specify that fundamental physics algorithms governing collisions, gravitational interactions, and rigid body dynamics are unlikely to change unless there are groundbreaking advancements in simulation techniques.

7.2 Rigid Body Assumption:

Clarify that the assumption of rigid bodies is a foundational element and is unlikely to change, ensuring stability in the physics simulation.

7.3 Simulation Framework:

Highlight that the overall simulation framework, once established, is unlikely to undergo significant changes, providing stability for integration with other game elements.

8. Traceability Matrix

A traceability matrix and graphs are typically tools used in project management and software development to ensure that requirements, features, and decisions can be traced back to their origins and that there is a clear mapping between various elements of the project. In our case, let's focus on creating a simplified traceability matrix.

Traceability Matrix :

Requirement ID	**4.2.1 Types 1**	**4.2.2 Solution Characteristics Specification**	**4.4 Modelling Decisions**
1	Projectile Type	Game Elements	Assumptions
2	Targets/Obstacles type	Game Elements	Assumptions
3	Collision Dynamics type	Physics Simulation	Assumptions, Theoretical Models
4	Gravitational Forces type	Physics Simulation	Assumptions, Theoretical Models

Requirement ID	4.2.1 Types 1	4.2.2 Solution Characteristics Specification	4.4 Modelling Decisions
5	Additional Forces Type (e.g., Wind)	Physics Simulation	Assumptions, Theoretical Models
6	Physics Engine Type	Physics Simulation	Assumptions, Theoretical Models
7	Physics Inputs Type	Physics Inputs	Assumptions, Theoretical Models
8	Physics outputs Type	Physics 0utputs	Assumptions, Theoretical Models
9	User Input Controls Type	Controls	Assumptions, Theoretical Models
10	Time Step Size Control type	Controls	Assumptions, Theoretical Models
11	Graphics Type	Graphics and Sound	Assumptions, Theoretical Models
12	Sound Effects Type	Graphics and Sound	Assumptions, Theoretical Models

Models to Code

Model	Code Functions/Structures
Projectile Type	Bird struct
Targets/Obstacles Type	Pig struct
Collision Dynamics Type	checkCollision() function
Gravitational Forces Type	updateBird Position() function
Additional Forces Type (e.g., Wind)	updateBird Position() function
Physics Engine Type	updateBird Position() function
Physics Inputs Type	handleInput() function
Physics outputs Type	updateBird Position() function, renderGame() function
User Input Controls type	handleInput() function
Time Step Size Control Type	Main loop with time step
Graphics Type	renderGame() function

Model	Code Functions/Structures
Sound Effects Type	renderGame() function

This directed acyclic graph illustrates the dependencies between different components of the project. For example, the "Physics Simulation" node depends on "Game Elements," "Controls," and "Graphics and Sound." This graph helps visualize the flow of dependencies.

Requirement Traceability Graph

This graph shows the relationships between different project elements, starting from the requirements. It demonstrates how requirements are connected to different types, models, and eventually, the code components.

9. Development Plan

A development plan outlines the steps and activities required to achieve the goals of a project. Here's a simplified development plan for the physics-based gaming application:

1. **Project Kickoff (Week 1-2)**
   - **Objectives:**
     - Clarify project scope, goals, and requirements.
     - Assemble development team.
     - Setup project repository and version control.
   - **Tasks:**
     - Hold a kickoff meeting.
     - Define roles and responsibilities.
     - Create and share project documentation.
     - Establish version control using Git.
2. **Requirement Analysis (Week 3-4)**
   - **Objectives:**
     - Refine and clarify project requirements.
     - Prioritize features and functionalities.
     - Finalize the system requirements specification (SRS).
   - **Tasks:**
     - Conduct meetings with stakeholders.
     - Revise and update SRS document.

- Prioritize features based on importance and complexity.

3. **Design and Architecture (Week 5-8)**
   - **Objectives:**
     - Define system architecture.
     - Create class diagrams and data flow diagrams.
     - Select appropriate frameworks and libraries.
   - **Tasks:**
     - Develop a high-level architecture plan.
     - Create detailed class diagrams for key components.
     - Choose graphics and sound libraries/frameworks.
     - Decide on the overall system structure.
4. **Implementation (Week 9-12)**
   - **Objectives:**
     - Build the core functionalities of the game.
     - Develop physics engine, collision detection, and rendering.
     - Implement user controls and input handling.
   - **Tasks:**
     - Begin coding the physics engine.
     - Implement basic game rendering.
     - Develop user input controls.
     - Integrate collision detection algorithms.
5. **Testing and Debugging (Week 13-16)**
   - **Objectives:**
     - Identify and fix software defects.
     - Conduct unit testing and integration testing.
     - Ensure physics simulations are accurate.
   - **Tasks:**
     - Perform unit tests on individual components.
     - Conduct integration tests for system interactions.
     - Debug and resolve any issues identified during testing.
6. **Refinement and Optimization (Week 17-18)**

- **Objectives:**
  - Optimize code for performance.
  - Refine user interface and graphics.
  - Enhance overall user experience.
- **Tasks:**
  - Optimize algorithms for computational efficiency.
  - Improve graphics rendering.
  - Gather user feedback for UI/UX improvements.

7. **Documentation (Week 19-20)**
   - **Objectives:**
     - Document code, design decisions, and usage instructions.
     - Finalize system documentation.
   - **Tasks:**
     - Write code comments for clarity.
     - Update design documentation.
     - Create a user guide for the application.
8. **Deployment (Week 21-22)**
   - **Objectives:**
     - Prepare the application for release.
     - Set up distribution channels (if applicable).
   - **Tasks:**
     - Package the application for deployment.
     - Create installers or deployment scripts.
     - Publish the application on relevant platforms.
9. **Post-Launch Support and Updates (Ongoing)**
   - **Objectives:**
     - Address user feedback and bug reports.
     - Implement updates and improvements.
   - **Tasks:**
     - Monitor user feedback channels.
     - Prioritize and address reported issues.

– Release updates as needed.

10. Physical Constants

Physical constants are fundamental values used to model real-world physics within the game. They contribute to the accuracy and authenticity of the in-game physics simulation.

Gravity Constant (g)

- Standard Earth's gravity = 9.81m/sˆ2.

Air Density ()

- Average air density at sea level: 1.225kg/mˆ3.

Friction Coefficient ()

- Coefficient of friction for surfaces: Typically, between 0 and 1, representing different friction levels.

Game Parameters

Game parameters are constants that define specific aspects of the gaming experience, such as scoring, difficulty, and user interactions.

1. **Score Multiplier (SCORE_MULTIPLIERSCORE)**
    - A factor that multiplies the base score for achieving certain in-game objectives.
2. **Difficulty Threshold (DIFFICULTY_THRESHOLD)**
    - A value that determines the difficulty level, affecting factors like enemy behavior, speed, or obstacle complexity.
3. **Time Step (t)**
    - The time step used in the physics simulation, influencing the accuracy of the simulation. Smaller values result in more accurate but computationally intensive simulations.

Environmental Constants

Environmental constants define features related to the virtual environment in which the game takes place. These constants contribute to the atmosphere and ambiance of the gaming world.

1. **Wind Speed (WIND_SPEED)**
    - The speed of the virtual wind, affecting the trajectory of projectiles or objects in motion.
2. **Ambient Light Intensity (AMBIENT_LIGHT)**
    - The intensity of the ambient light in the virtual environment, influencing the overall visual atmosphere of the game.

3. **Background Music Volume (MUSIC_VOLUME)**
    - The volume level of the background music, contributing to the auditory experience of the game.

11. Project Goal

The primary ambition of this project is to achieve a breakthrough in physics-based computation within the gaming realm. By focusing on the intricacies of realistic collision dynamics, gravitational interactions, and pioneering the implementation of dynamic momentum disruptions, the project aims to elevate the gaming experience to a level where players actively engage with and manipulate fundamental physics principles. When a bird collides, the goal is to authentically break the gravity momentum, simulating a quantum leap in computational realism. This project aspires to contribute to the advancement of physics understanding in the gaming community, pushing the boundaries of what is achievable in virtual worlds through cutting-edge computational physics.

Realistic Physics Simulation:

Implement a physics engine that accurately models collision dynamics, gravitational interactions, and additional forces to create a realistic and immersive gaming experience.

Engaging Gameplay:

Design a gaming environment that challenges players to apply physics principles strategically, encouraging experimentation and skill development.

User-Friendly Interface:

Provide a user-friendly interface allowing players to manipulate initial conditions, adjust parameters, and actively participate in the simulation.

Visual and Auditory Feedback:

Utilize graphics to visually represent physics-based interactions, emphasizing rigid body collisions and movements. Implement sound effects corresponding to key physics events, enhancing the overall sensory experience.

Dynamic Challenges:

Introduce variability through additional forces like wind, creating dynamic challenges that require adaptability and skill from the players.

Configurability:

Allow users to specify initial conditions, launch angles, forces, and time step sizes, providing a customizable and personalized gaming experience.

Scalability:

Design the system to be extensible for potential future expansions, allowing for the addition of new features, levels, and challenges.

Achieve Physics/Computing Breakthrough:

Develop a physics-based gaming application that explores breakthroughs in physics computation, emphasizing the accurate representation of momentum changes during collisions. Strive for a level of computational fidelity that contributes to advancements in real-time physics simulations within gaming environments.

12. Goal in Physics

Achieving in Quantum Precision Physics-Based Gaming: Unleashing Realistic Momentum Dynamics

The paramount goal of this project is to attain a pinnacle of precision in physics-based gaming. By intricately modeling realistic collision dynamics, gravitational interactions, and introducing innovative momentum dynamics, the aim is to achieve a gaming experience that mirrors the precision of quantum physics principles. When a bird collides, pushing the boundaries of what is achievable in virtual physics simulations. This project is dedicated to advancing the understanding of physics principles in gaming, offering players an immersive experience that reflects the intricacies and precision of real-world physics.