Module Interface Specification for Angry Birds Alike

Author: Hossain, Al Jubair

March 31, 2023

Revision History

Date	Version	Notes
March 5, 2023	1.0	Fixed hyperlinks.
March 13, 2023	1.0	Changed section 5.
March 31, 2023	1.5	Adding various changes in sections from 3, 4, 5, and 6, adding module information on Physics engine, UI, Game Logic, Environment variables , graphical examples of the game after implementation

Contents

1 Symbols, Abbreviations, and Acronyms

Symbol/Abbreviation/Acronym	Definition
MIS	Module Interface Specification
SRS	Software Requirements Specification
VnV	Verification and Validation

2 Introduction

This document provides the Module Interface Specifications (MIS) for the "Angry Birds Alike" project. It defines the interfaces for each module, detailing their inputs, outputs, and functionality. This MIS is intended to guide the implementation phase, ensuring that module interactions are clear and consistent. Complementary documents include the System Requirement Specifications (SRS), the Verification and Validation (VnV) plan, and the Module Guide (MG).

3 Notation

The notation used in this document follows a standardized format for specifying module interfaces: - **Data Types**: Descriptions of the data types used within the modules. - **Functions**: Specification of module functions, including inputs, outputs, and side effects. - **Constants**: Constants used within the modules. - **State Variables**: Variables that store the state of the module.
Data Type Notation: - *integer*: Represented as **int**, a whole number. - *real*: Represented as **float**, a floating-point number. - *string*: A sequence of characters.

4 Module Decomposition

The modules in "Angry Birds Alike" are decomposed according to the hierarchy established in the Module Guide. This decomposition supports the project's architectural design and clarifies the responsibilities of each module.

4.1 Overview of Modules

- **M1 Game Logic Module**: Handles the core gameplay mechanics. - **M2 Physics Engine Module**: Manages physics simulations. - **M3 User Interface Module**: Controls the game's user interface and input processing. - **M4 Graphics Rendering Module**: Responsible for rendering game graphics. - **M5 Sound Module**: Manages game audio.
Each module's interface specification is detailed in the following sections.

5 MIS of Game Logic Module

This section details the interface for the Game Logic Module, which is central to managing the game's rules, level progression, and scoring.

5.1 Module

Short Name: GameLogic

5.2 Uses

- Physics Engine Module (for physics calculations) - User Interface Module (for receiving player actions)

5.3 Syntax

5.3.1 Exported Constants

None

5.3.2 Exported Access Programs

Name	In	Out	Exceptions
initializeLevel	LevelID	–	LevelNotFound
getScore	–	integer	–
launchProjectile	Velocity, Angle	–	InvalidInput
resetLevel	–	–	–
endGame	–	–	–

5.4 Semantics

5.4.1 State Variables

- currentLevel: LevelID - score: integer

5.4.2 Environment Variables

None

5.4.3 Assumptions

- initializeLevel is called before any other access programs to set up the game environment. - The game's levels are pre-defined and correspond to valid LevelID inputs.

5.4.4 Access Routine Semantics

initializeLevel(LevelID): - transition: Loads the specified level, setting currentLevel to the given LevelID. - exception: LevelNotFound if the specified LevelID does not correspond to a defined level.
getScore(): - output: Returns the current score of the game.
launchProjectile(Velocity, Angle): - transition: Launches a projectile with the specified velocity and angle. Updates game state based on the physics simulation. - exception: InvalidInput if the given Velocity or Angle are out of acceptable ranges.
resetLevel(): - transition: Resets the current level to its initial state.
endGame(): - transition: Terminates the current game session.

5.4.5 Local Functions

None

6 MIS of Physics Engine Module

This section outlines the interface for the Physics Engine Module, responsible for calculating and simulating the physical interactions within the game.

6.1 Module

Short Name: PhysicsEngine

6.2 Uses

- Game Logic Module (to receive commands related to game actions)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
calculateTrajectory	Velocity, Angle	Sequence of Points	InvalidInput
detectCollision	Sequence of Points, Objects	Boolean, ImpactPoint	–
applyPhysicsEffects	ImpactPoint, Objects	–	–

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

- The physics engine is initialized and configured before any game levels are loaded or any projectiles are launched.

6.4.4 Access Routine Semantics

calculateTrajectory(Velocity, Angle): - output: Returns a sequence of points representing the trajectory of a projectile launched with the specified velocity and angle, considering gravitational forces. - exception: InvalidInput if the given Velocity or Angle are out of acceptable ranges.
detectCollision(Sequence of Points, Objects): - output: Returns a boolean indicating whether a collision occurs along the projectile's path, and if so, returns the point of impact (ImpactPoint). - transition: Checks the trajectory against the position of objects in the game to determine collisions.
applyPhysicsEffects(ImpactPoint, Objects): - transition: Applies physics effects (e.g., force, momentum transfer) to objects involved in a collision at the given ImpactPoint.

6.4.5 Local Functions

None

7 MIS of User Interface Module

This section describes the interface for the User Interface Module, which manages interactions between the player and the game, ensuring a seamless and intuitive gaming experience.

7.1 Module

Short Name: UserInterface

7.2 Uses

- Physics Engine Module (to initiate projectile motion based on player input) - Game Logic Module (to update the game state based on player actions)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
processInput	UserAction	–	InvalidAction
updateDisplay	GameState	–	–

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

keyboard: the device from which user input is received
display: the device on which the game state is visually rendered

7.4.3 Assumptions

- The user interface is refreshed at regular intervals to reflect the current game state accurately. - User input is captured continuously during gameplay.

7.4.4 Access Routine Semantics

processInput(UserAction): - transition: Interprets user actions (e.g., keyboard presses for moving the slingshot, launching the bird) and communicates these actions to the Game Logic Module for processing. - exception: InvalidAction if the user performs an action that is not recognized or not applicable in the current game state.
updateDisplay(GameState): - transition: Updates the visual display based on the current state of the game, including the position of objects, scores, and any other relevant gameplay information.

7.4.5 Local Functions

None

8 Simulation Interface

The Simulation Interface provides a visual representation of the projectile motion based on user inputs. The following figures show the simulation at different stages of a projectile's flight path.

8.1 Annotated Initial Launch

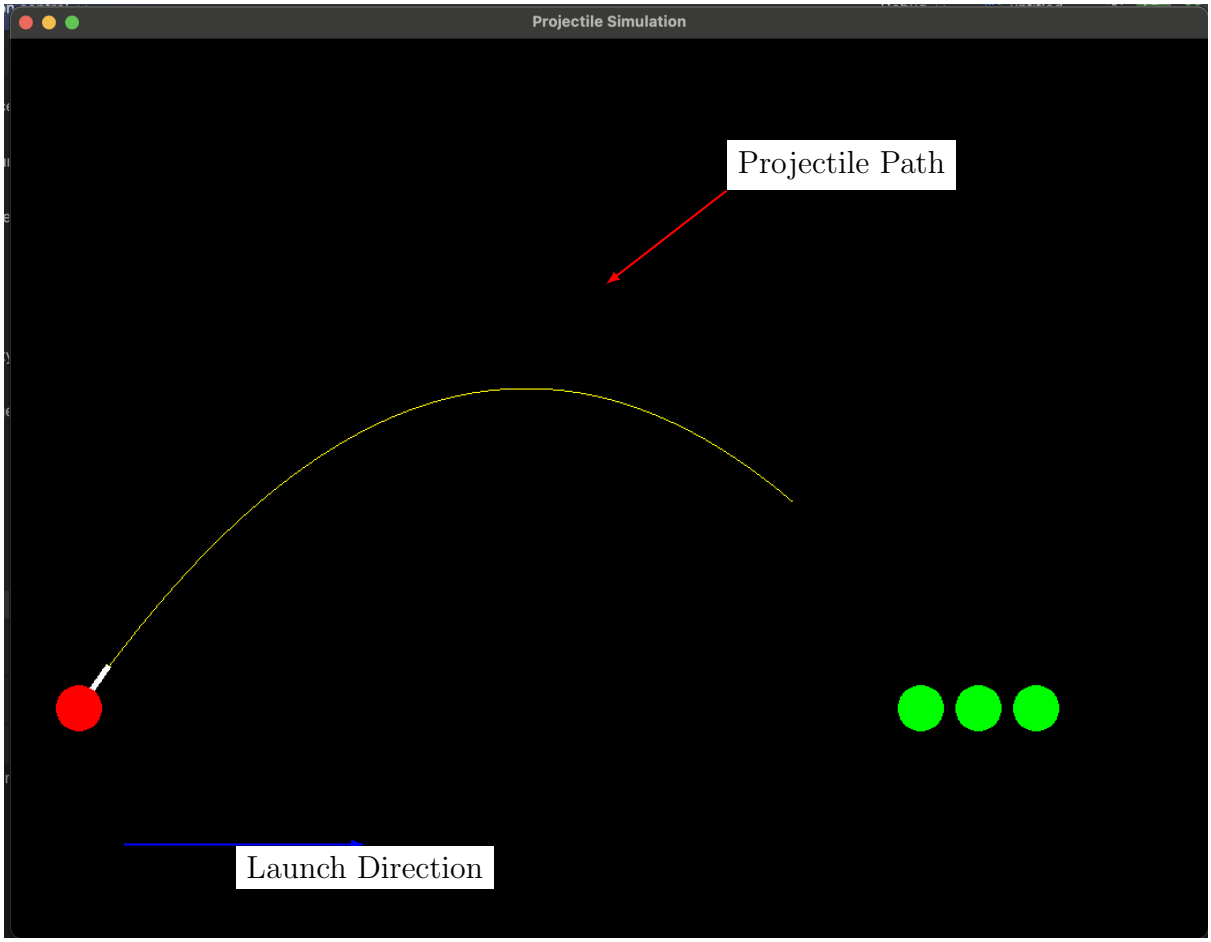

Figure 1: Annotated image of the initial stage of the projectile launch.

8.2 After Collision

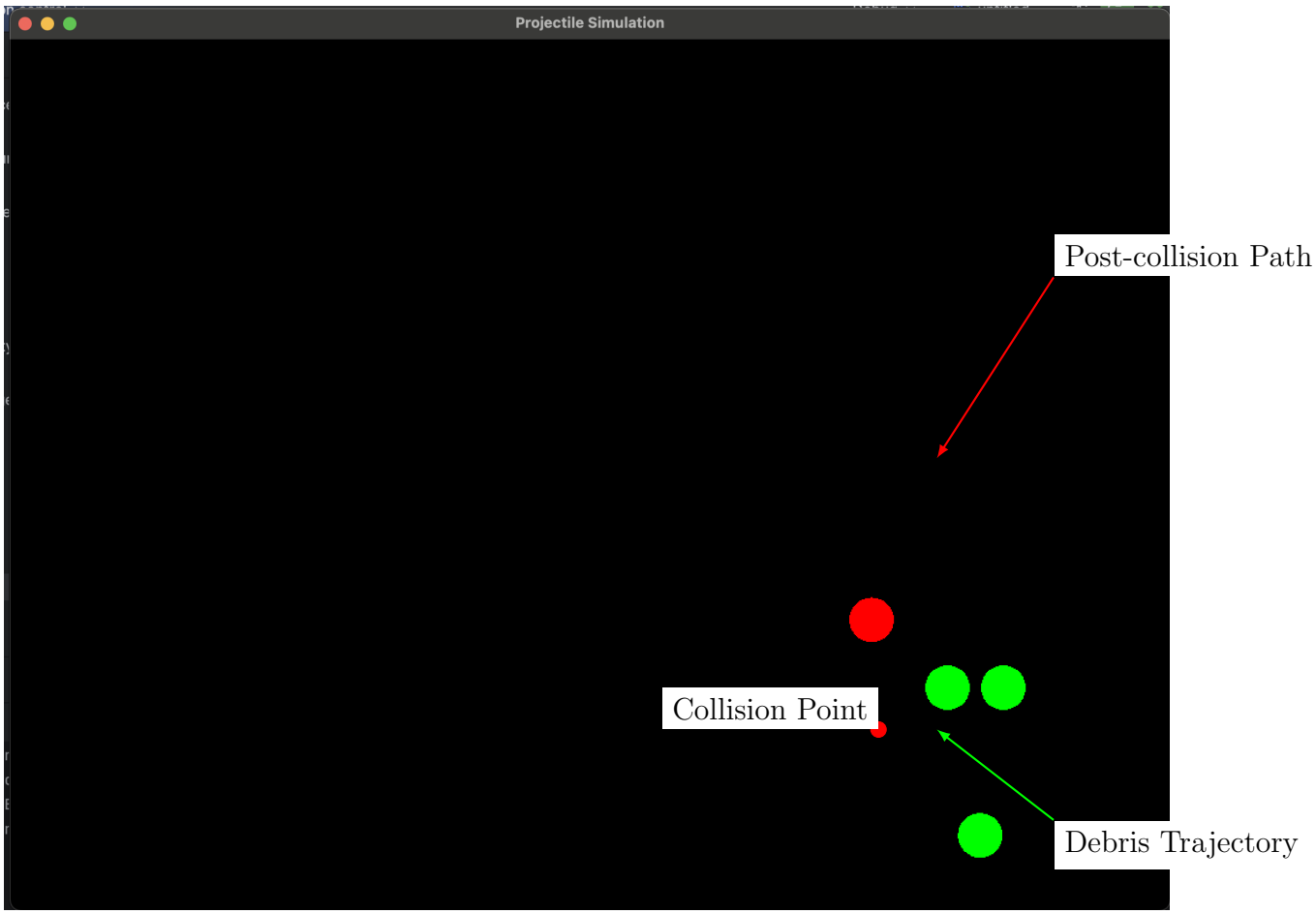

Figure 2: Projectile in mid-flight, after colliding with targets.

9 MIS of Game Logic Module

This section outlines the interface for the Game Logic Module. This module is essential for controlling the flow of the game, managing levels, player scores, and enforcing the game rules.

9.1 Module

Short Name: GameLogic

9.2 Uses

- Physics Engine Module (to simulate physical interactions) - User Interface Module (to receive player inputs)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
startLevel	LevelID	–	InvalidLevelException
endLevel	–	LevelOutcome	–
updateScore	ScoreUpdate	–	–
resetLevel	–	–	–

9.4 Semantics

9.4.1 State Variables

- currentLevel: LevelID - currentScore: integer

9.4.2 Environment Variables

None

9.4.3 Assumptions

- startLevel is called at the beginning of each level. - endLevel is called when a level is completed or the player fails.

9.4.4 Access Routine Semantics

startLevel(LevelID): - transition: Initializes the game state for the specified level. This includes setting up the initial positions of objects, resetting the score for the level, and preparing the physics engine for simulation. - exception: InvalidLevelException is thrown if the LevelID does not correspond to a valid level.
endLevel(): - output: Returns an outcome of the current level, indicating whether the player has succeeded or failed, and any rewards earned. The game state is updated accordingly. - exception: None
updateScore(ScoreUpdate): - transition: Adjusts the current score based on actions taken during the level, such as destroying targets or completing objectives. - exception: None
resetLevel(): - transition: Resets the current level to its initial state, including reinitializing the positions of objects and resetting the score. - exception: None

9.4.5 Local Functions

None

10 MIS of User Interface Module

This section provides details on the User Interface Module, which is responsible for handling all player interactions and updating the game display accordingly.

10.1 Module

Short Name: UserInterface

10.2 Uses

- Game Logic Module (to communicate player actions and receive game state updates)
- Physics Engine Module (for displaying physics-based animations)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayMainMenu	–	–	–
displayLevel	LevelID	–	InvalidLevelException
getPlayerInput	–	PlayerAction	–
updateDisplay	GameState	–	–

10.4 Semantics

10.4.1 State Variables

- currentScreen: ScreenType

10.4.2 Environment Variables

- screen: a display device

10.4.3 Assumptions

- The UI module is initialized before any player interaction. - The display device is capable of rendering the game's graphics.

10.4.4 Access Routine Semantics

displayMainMenu(): - transition: Changes the currentScreen to the main menu and updates the display device to show the main menu options. - exception: None
displayLevel(LevelID): - transition: Loads the specified level's visual elements and prepares the screen for gameplay. - exception: InvalidLevelException is thrown if the LevelID does not correspond to a valid level.
getPlayerInput(): - output: Captures and returns the player's input as a PlayerAction, which includes moving the launcher, firing a projectile, or navigating the UI. - exception: None
updateDisplay(GameState): - transition: Updates the game's visual representation on the display device based on the current game state, including the positions of objects, scores, and any UI elements. - exception: None

10.4.5 Local Functions

None

11 Appendix

11.1 Physics Equations and Concepts

This appendix provides a brief overview of the physics equations and concepts utilized within the Physics Engine Module to simulate realistic interactions within the game.

Projectile Motion

The trajectory of a projectile is calculated using the initial velocity (v_0), the angle of launch (θ), and the acceleration due to gravity (g). The horizontal (x) and vertical (y) positions at any time t can be calculated as follows:
- Horizontal position: $x(t) = v_{0x} \cdot t$ - Vertical position: $y(t) = v_{0y} \cdot t - \frac{1}{2}gt^2$
where $v_{0x} = v_0 \cdot \cos(\theta)$ and $v_{0y} = v_0 \cdot \sin(\theta)$.

Collision Detection and Response

The module employs basic principles of momentum and energy conservation to simulate collisions between objects. When two objects collide, the module calculates the new velocities based on the masses and initial velocities of the objects involved.

Gravity and Other Forces

Gravity is a constant force acting downward on all objects. The module also accounts for other forces, such as applied forces when a bird is launched, to update the velocities and positions of objects.

11.2 Glossary of Terms

- **Projectile**: An object thrown into space upon which the only force acting is gravity.
- **Trajectory**: The path followed by a projectile flying or an object moving under the action of given forces. - **Collision**: An event where two or more bodies exert forces on each other in a relatively short time. - **Gravity**: A force that attracts a body towards the center of the earth, or towards any other physical body having mass.

12 Reflection

The development of "Angry Birds Alike" has been a comprehensive exercise in applying theoretical knowledge to a practical, engaging project. This section reflects on various aspects of the project's development, highlighting challenges encountered, solutions devised, and learning outcomes.

12.1 Challenges and Solutions

One of the most significant challenges was accurately simulating the physics involved in the game. Implementing a realistic trajectory calculation for the birds required a deep understanding of projectile motion physics and the application of numerical methods for solving differential equations. To address this, extensive research into physics simulation techniques was undertaken, and the Runge-Kutta method (RK4) was implemented for its balance between accuracy and computational efficiency.
Another challenge I think, will be designing a user interface that is intuitive yet provides all necessary controls for the game. Achieving this required iterative design and testing, incorporating feedback from trial users to refine the UI/UX until it met the desired standards of accessibility and engagement.

12.2 Key Learning Outcomes

Problem-Solving Skills: Developing this game honed my ability to break down complex problems into manageable components, apply theoretical knowledge to practical situations, and devise effective solutions.
Software Engineering Principles: This project reinforced the importance of modular design, information hiding, and the use of design patterns in creating maintainable and scalable software.
Physics Simulation: I gained valuable experience in simulating real-world physics within a digital environment, an area that blends scientific knowledge with creative problem-solving.

12.3 Future Improvements

Given more time and resources, I would explore the implementation of more complex physics interactions, such as wind resistance, to add another layer of challenge and realism to the game. Additionally, integrating a more sophisticated AI for the game's opponents could enhance the gameplay, making it more dynamic and unpredictable.

12.4 Conclusion

"Angry Birds Alike" has been a challenging yet rewarding project that has significantly contributed to my growth as a software developer and problem-solver. It underscored the importance of a solid theoretical foundation, the value of user-centered design, and the necessity of perseverance through the challenges inherent in software development.