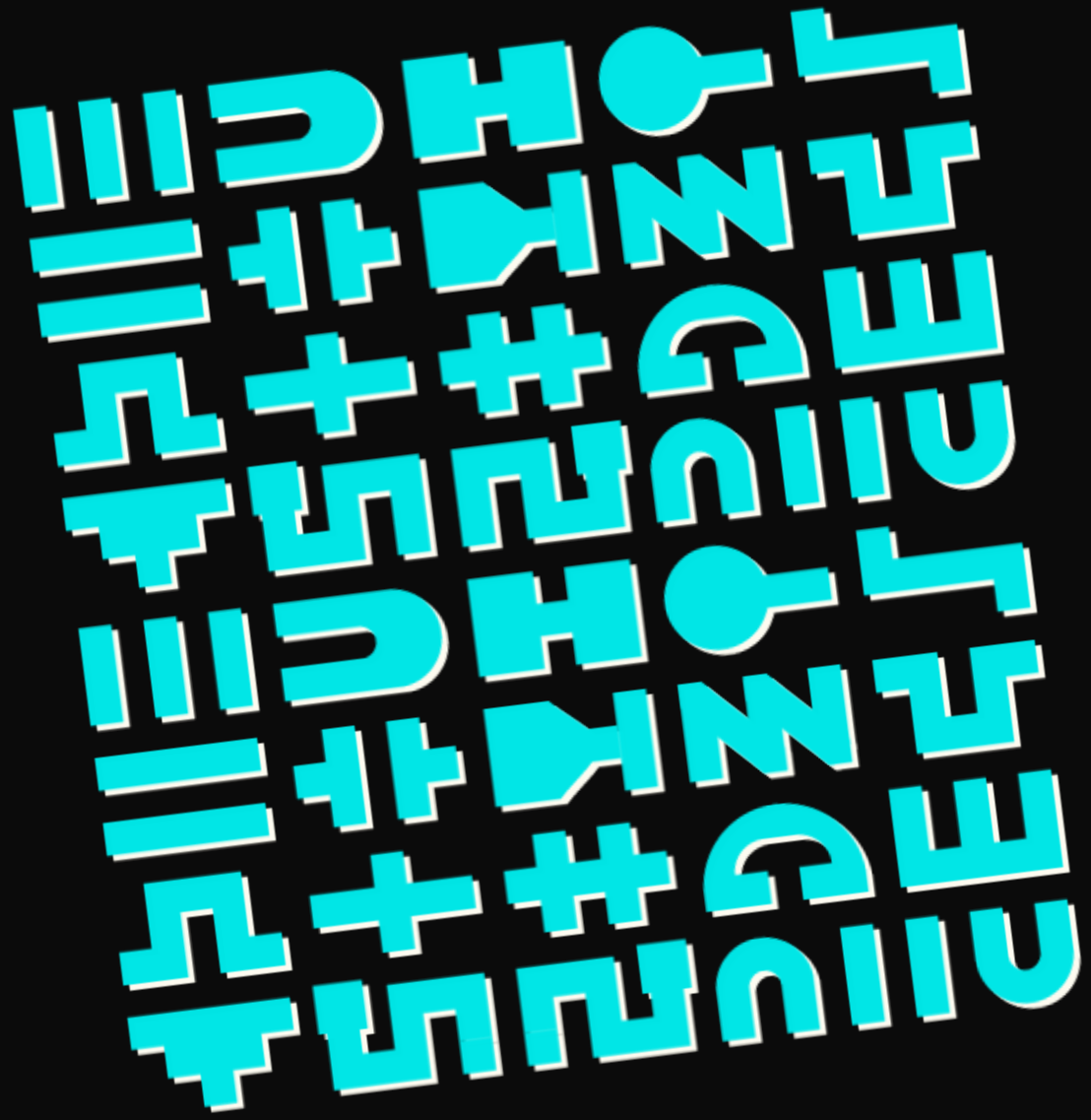
Software Design and Implementation of a Gyroscope-Based Game Interface

By Morteda Hokman Saya

Outlines

- Introduction
- System Components & Tools
- Software Process Models
- Requirements Engineering
- System Modeling
- Project Management and Planning
- System Design
- Software Testing
- Software Evolution
- Challenges & Solutions
- Conclusion
- References



Introduction

- traditional game controllers rely on buttons and joysticks, they often lack the intuitive feel of natural motion. This project explores the potential of using accessible technology to create more engaging interactions. We'll delve into a software engineering endeavor focused on harnessing the power of an Arduino microcontroller and a gyroscope sensor to control mini-games, such as maze navigation challenge

- purely through physical tilt. This presentation will cover the software design decisions, the crucial steps involved in processing real-time sensor data, the implementation of the game logic, and the challenges encountered in translating physical movement into a responsive digital experience."

System Components & Tools

- Software

C# Language

Arduino IDE

Unity Game Engin

System Components & Tools

- Hardware

Arduino nano
6000 IQD

MPU 6050 Gyroscope
4,500 IQD

Total Cost
10,500

Requirements Engineering

User Requirements (UR)

UR1: Game Control via Tilt: The user shall be able to control the main actions of the mini-game(s) (e.g., movement, aiming, balancing) by physically tilting the Arduino-based controller device.
UR2: Play Mini-Game(s): The user shall be able to play at least one functional mini-game implemented on the system.
UR3: Game Initiation: The user shall be able to start or restart a game session (e.g., via a reset button or specific initial condition).

System Requirements (SR)

functional

SR-F1: Sensor Interfacing: The system shall successfully interface with the specified gyroscope sensor (e.g., MPU-6050) using the appropriate communication protocol (e.g., I2C).
SR-F2: Data Acquisition: The system shall periodically read raw angular velocity and/or acceleration data from the gyroscope sensor.
SR-F3: Data Processing & Filtering: The system shall process the raw sensor data to calculate usable tilt angle(s) or orientation data. This must include methods to mitigate sensor noise and potentially drift (e.g., using averaging, complementary filter, or basic thresholding).
SR-F4: Input Mapping: The system shall map the processed tilt angle(s) to specific game control commands based on predefined sensitivity thresholds and ranges. (e.g., Tilt > X degrees on Y-axis = move player up).
SR-F5: Game Logic Implementation: The system shall implement the rules, state transitions (e.g., tracking score, lives, position), and win/loss conditions for the defined mini-game(s).

Requirements Engineering

System Requirements (SR)

non-functional

SR-NF1: Performance/Responsiveness: The system shall read sensor data, process it, and update the game state frequently enough to provide a perception of real-time control with minimal noticeable lag (e.g., target a complete control loop cycle time under 100 milliseconds).

SR-NF2: Usability: The mapping from physical tilt to game control should feel intuitive and predictable to the average user after a brief familiarization period. Calibration (if implemented) should be straightforward to perform.

SR-NF3: Reliability: The system software shall operate stably during gameplay without unexplained freezes or crashes under normal operating conditions.

SR-NF4: Resource Constraints: The system software (sketch) must run within the memory (Flash, SRAM) and processing limitations of the target Arduino board (e.g., Arduino Uno R3).

SR-NF5: Hardware Constraints: The system shall utilize the specified Arduino model and gyroscope sensor model.

Software Process Models

- The incremental model allows the developers to quickly release a version of the software with limited functionality, and then at each development iteration add additional, incremental functionality. The development in each iteration occurs in a linear method, as with the waterfall model. Ideally, the most important functions are implemented first and successive stages add new functionality in order of priority.

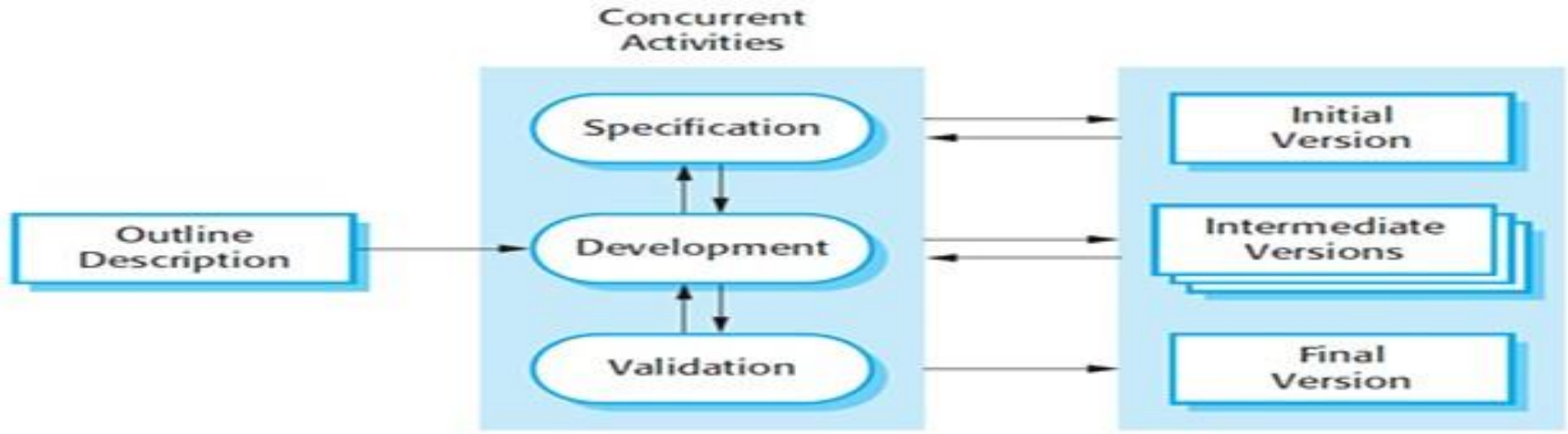
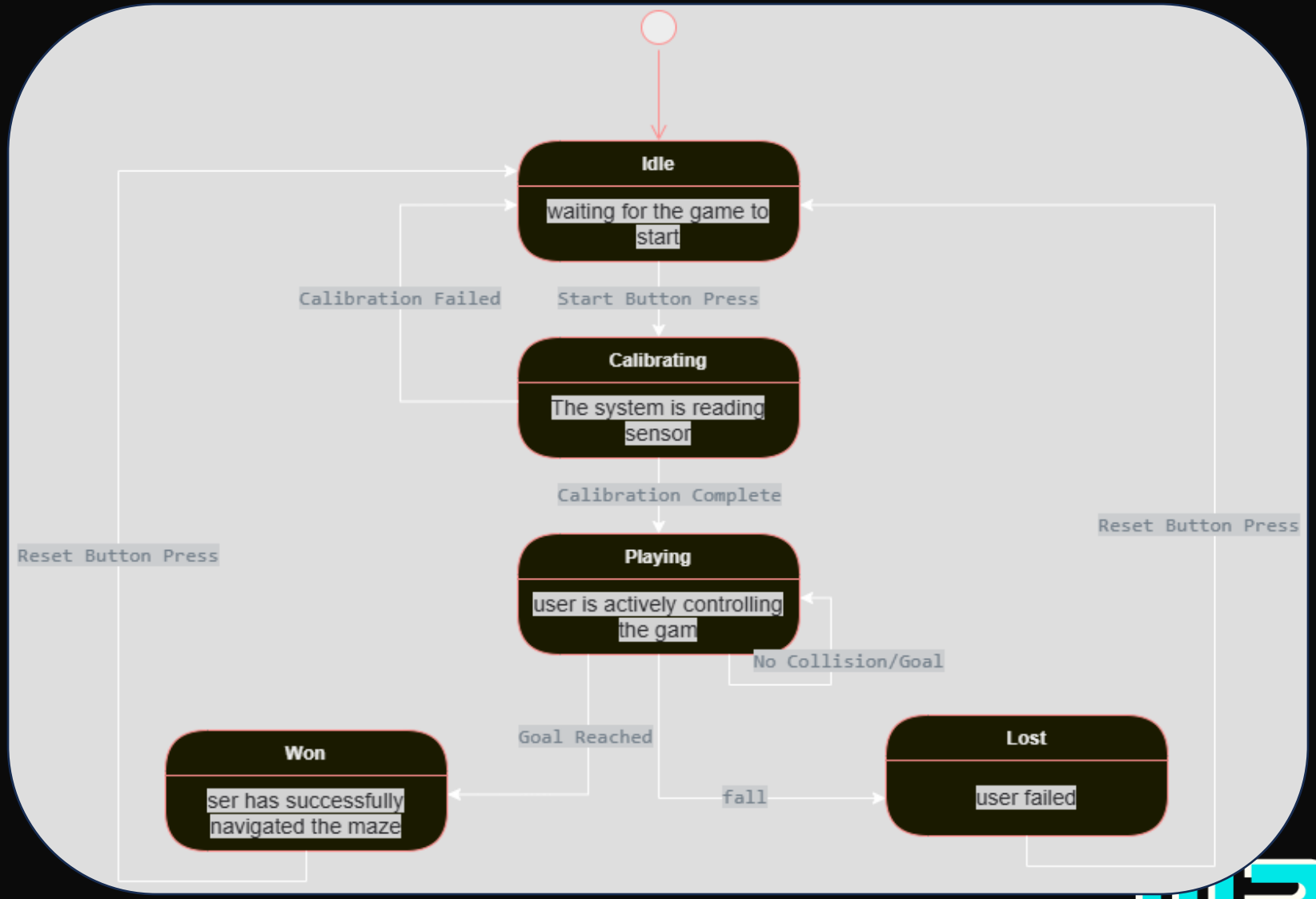


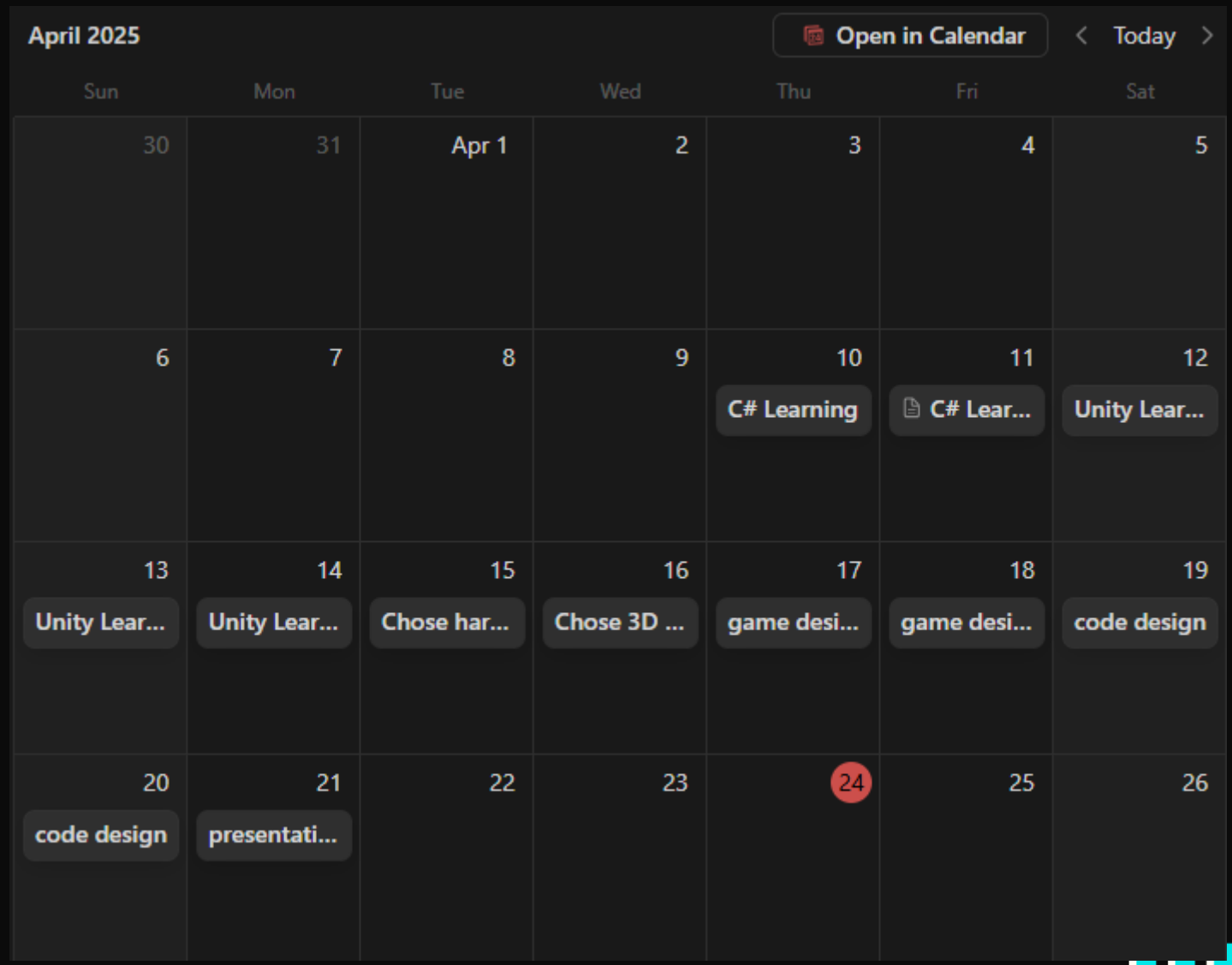
Figure 2.3 The Incremental l model

System Modeling

Maze game Modeling



Project Management and Planning



System Modeling

- Object-Oriented Design (OOD): This strategy is highly relevant to Unity.
- Unity uses C#, which is an object-oriented language.
- Game development in Unity revolves around GameObjects (the entities/objects).
- Scripts (like MonoBehaviours) attached to GameObjects are Classes.
- These scripts contain Attributes (variables like health, speed) and Methods (functions like Move(), Attack(), Update()).
- Unity heavily utilizes Encapsulation (using public, private etc.), Inheritance (scripts inherit from MonoBehaviour or other custom classes), and Polymorphism (e.g., different scripts implementing the same interface method).

System Testing

- Testing was conducted using an informal, exploratory approach. Initial testing rounds were performed by myself (developer testing) during and after implementation phases to ensure basic functionality and catch obvious errors.
- Subsequently, the system was tested by a small group of peers (2 friends) to gather external perspectives and basic usability feedback.
- Testing focused on playing the game through its complete cycle: Initialization/Calibration -> Gameplay -> Win/Loss Condition -> Reset.
- Testers were encouraged to try different tilt speeds and angles to assess the control responsiveness.

System Evolution

Evolution During Project Development:

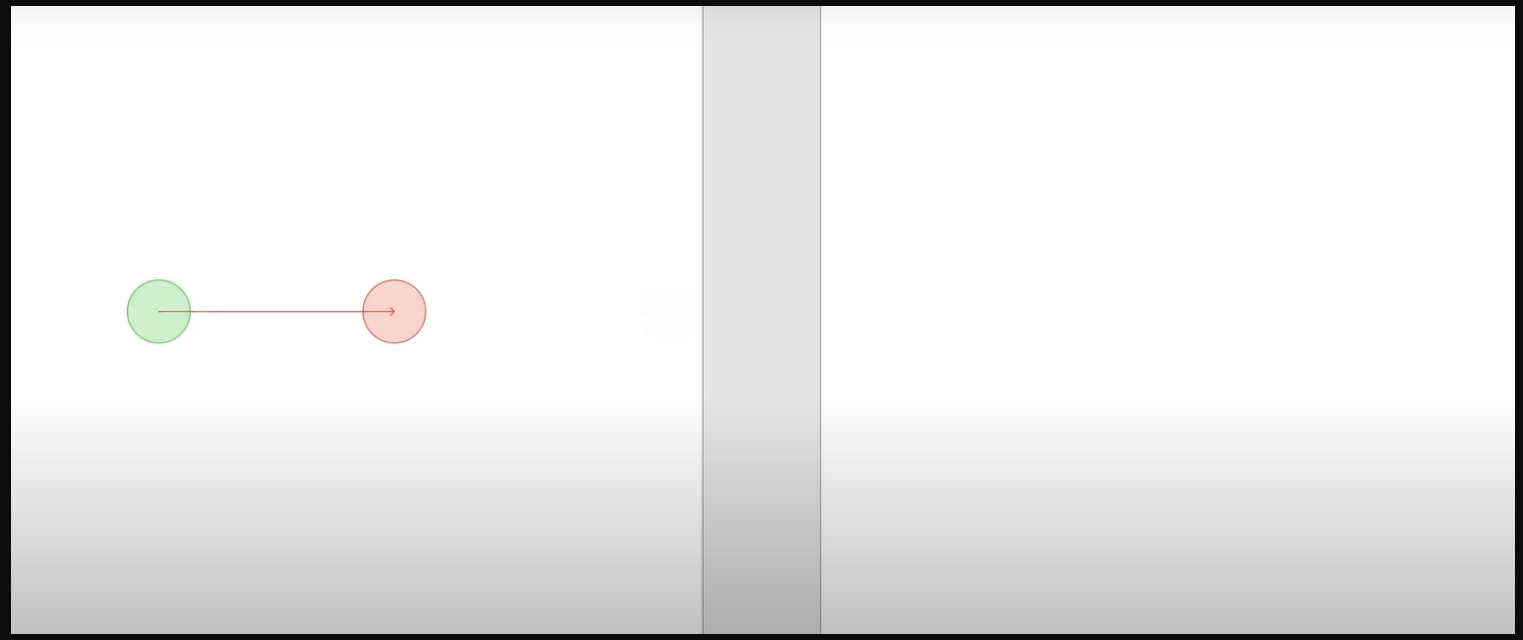
- Initial control mapping felt too jerky : Evolved the filtering algorithm or adjusted sensitivity thresholds.
- Collision detection was unreliable : Refined the game logic for checking boundaries/obstacles.
- Found sensor drift : Added a simple calibration step at the start.

Potential Future Evolution Paths:

- Improve the software to make the game smoother.
- Add additional controls
- Making a complete console for interactive games

Challenges & Solutions

- This was my first time entering the world of game development, so I encountered many problems, the most important of which was the tunnel problem.
- This problem made me change the entire course of the project.



Conclusion

- Ultimately, we successfully integrated the Arduino gyroscope controller with the Unity Engine, bridging physical motion with the digital game world. This project served as a compelling proof-of-concept, effectively demonstrating the potential of motion-based controls to create highly interactive and engaging gameplay experiences that offer a distinct and enjoyable alternative to traditional input methods

References

- Software Engineering Lectures by: Assist. Prof. Rana Riad K.
- GitHub - wengaoy/MPU6050-Arduino-and-unity-3d
- What is tunneling?
- GitHub - zigurous/unity-brick-breaker-tutorial: 🧱🏓 Learn to make Brick Breaker / Breakout in Unity.

Thanks