

# Flappy VR: A Motion-Controlled Flying Experience

**Author:** Jordyn Rabinowitz

---

## Introduction

### Motivation

The goal of this project was to create an engaging and immersive virtual reality (VR) experience that leverages **motion-based controls**. Inspired by the simplicity and addictiveness of *Flappy Bird*, I wanted to reimagine the mechanics for VR, allowing players to control a bird by **flapping their arms**. The challenge was to design **intuitive motion-based mechanics** that feel natural while incorporating physics-based movement in Unity.

### What I Built

**Flappy VR** is a game where players embody a bird and navigate through a floating world. Instead of tapping a screen or pressing a button, players flap their arms while holding VR controllers to generate lift and propel forward.

The experience includes:

- **Realistic physics-based flight mechanics** using Unity's Rigidbody system.
  - **Flap detection** based on analyzing motion data from the left and right VR controllers.
  - **Dynamic obstacles and scoring system**, where players collect coins and land on platforms. Some platforms decrease the score, encouraging strategic movement.
  - **Randomized obstacle spawning** to ensure unique playthroughs.
  - **Sound effects that correspond to different collisions**, enhancing feedback and immersion.
- 

## Related Work

### Inspirations

- **Flappy Bird** – The core gameplay idea was inspired by *Flappy Bird*, where **timing and precision** are key.

- **VR Fitness Games** – Titles like *Beat Saber* and *Supernatural VR* inspired the motion-based control mechanism.
- **Birdly** – A VR flight simulator that allows users to control flight using full-body motion, reinforcing the feasibility of my approach.

## References

- **Unity Documentation**: Physics, Rigidbody, and XR Interaction Toolkit.
  - **Research on Motion-Based VR Interactions** – Papers on gesture-based input in VR guided my approach to flap detection.
- 

# Implementation

## Development Process

### 1. Setting Up Unity XR

- Created a Unity project with the **XR Interaction Toolkit**.
- Configured an **XR Origin** with a camera representing the player's head.
- Implemented **VR controller tracking** to detect motion.

### 2. Player Motion and Controls

- Attached a **Rigidbody** to the player to enable physics-based movement.
- Designed an **algorithm to detect arm flapping** and convert it into upward and forward motion.
- Implemented **force application** based on arm movement speed and frequency.

### 3. Physics & Movement

- Experimented with different **flap force values** for natural-feeling movement.
- Used Unity's built-in **gravity** instead of manually applying a downward force.
- Applied **constraints to prevent unintended rotation** and maintain stability.

### 4. Obstacle and Scoring System

- Coins and platforms have different **tags** to trigger specific score changes.
- Used **collision detection** to determine successful landings and score adjustments.
- Implemented **randomized spawning** of coins and platforms upon game start, ensuring replayability.

### 5. Audio Feedback System

- Different collision sounds were added for:
  - Collecting coins.

- Landing on score-increasing platforms.
- Hitting obstacles (negative feedback sound).
- Each sound is assigned based on the object's tag.

## 6. Debugging & Optimization

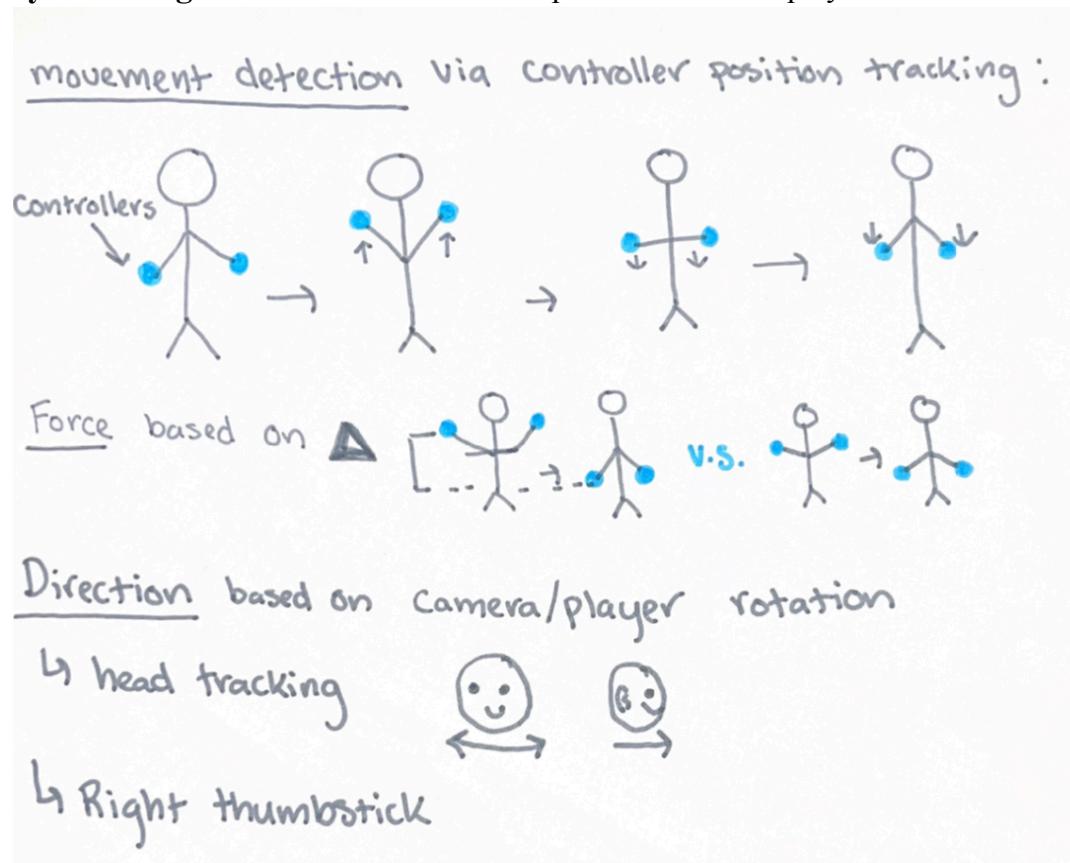
- Fixed issues related to **rigidbody physics** and unintended floating.
- Ensured **smooth flap detection** by refining motion thresholds.
- Improved **console logging** for better debugging within Unity.

## Technical Details

- **Unity XR Interaction Toolkit** for VR integration.
- **Rigidbody physics** for movement.
- **C# scripting** for motion detection and game mechanics.
- **TextMeshPro** for in-game UI elements.
- **Randomized spawning system** for obstacles.

## Figures & Diagrams

- **System Diagram:** Shows how motion input translates into player movement.



- **Code Snippets:** Key sections of the motion detection (flap) and physics handling logic.

```

• void FixedUpdate()
{
    if (!hasStarted) return; // Stop movement until game
starts

    // Detect controller movement
Vector3 leftPos, rightPos;

    if
(leftController.TryGetFeatureValue(CommonUsages.devicePosition,
out leftPos) &&

rightController.TryGetFeatureValue(CommonUsages.devicePosition,
out rightPos))
{
    Vector3 leftVelocity = leftPos - previousLeftPos;
    Vector3 rightVelocity = rightPos -
previousRightPos;

    previousLeftPos = leftPos;
    previousRightPos = rightPos;

    float flapSpeed = (leftVelocity.magnitude +
rightVelocity.magnitude) / 2;

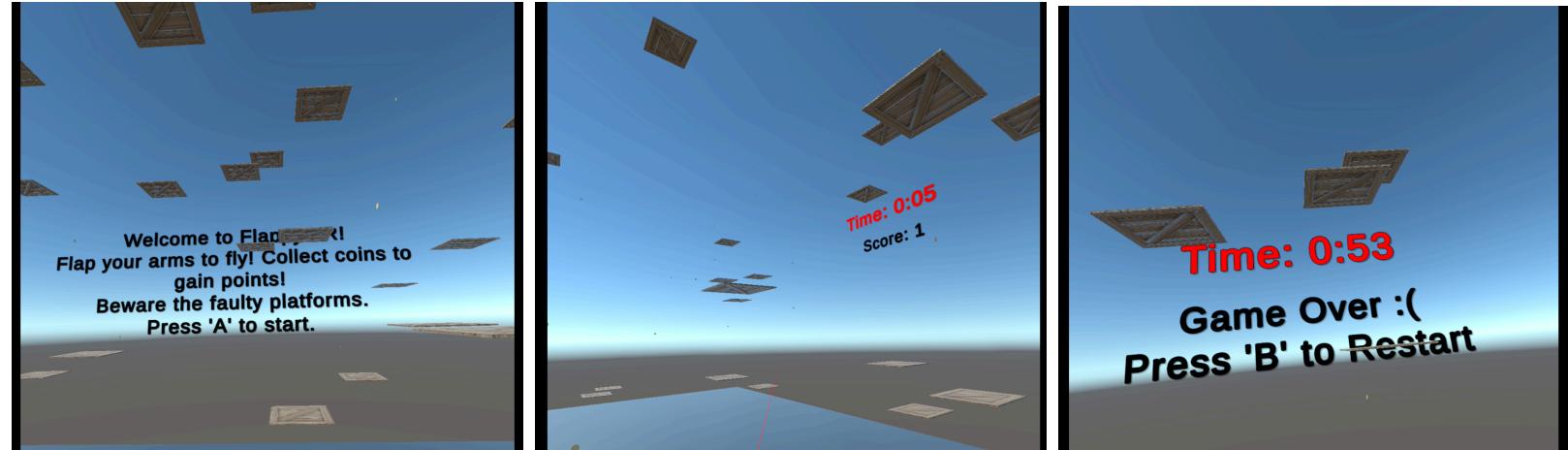
    Debug.Log("Flap Speed: " + flapSpeed);

    if (flapSpeed > flapThreshold)
    {
        Debug.Log("Flap detected! Applying force.");
        float flapMultiplier = Mathf.Clamp(flapSpeed *
2, 1, 3);
    }
}
}

```

```
        Vector3 force = (Vector3.up * flapStrength *  
flapMultiplier) +  
                           (headTransform.forward *  
forwardBoost * flapMultiplier);  
  
        rb.AddForce(force, ForceMode.Impulse);  
        // Play flap sound  
        if (flapSound != null)  
        {  
            flapSound.Play();  
        }  
    }  
}
```

- **Gameplay Screenshots:** Illustrates the player navigating between platforms.



## Future Work

## **Enhancements**

- **Better Flap Detection:** Improve the motion tracking algorithm to reduce false-positive flaps.

- **More Engaging Levels:** Introduce different platform types and varying distances to increase challenge.
- **Multiplayer Mode:** Implement a competitive mode where players can race against each other (stretch goal).

## Additional Features

- **Power-ups** (e.g., temporary speed boost, slow motion, extra points).
  - **Customizable bird avatars.**
  - **Leaderboard integration** for high scores (*coded this, but it currently needs debugging*).
- 

## Using AI for Debugging

### Challenges Faced

- **Unintended Floating on Game Start** – Incorrect Rigidbody settings caused uncontrolled movement.
- **Flap Detection Issues** – The controller motion wasn't being read correctly.
- **Console Logging Problems** – Debug messages weren't appearing, making issue tracking difficult.

### How AI Helped

- Identified **mistakes in my physics setup**, such as inconsistent force application.
- Debugged **motion tracking**, improving controller detection and movement calculations.
- Refined **force values** to ensure realistic flight mechanics.
- Optimized **logging techniques** for better debugging.

Through AI-assisted debugging, I was able to iterate faster, fix issues efficiently, and refine the game mechanics for a smoother experience.

---

## Conclusion

**Flappy VR** reimagines *Flappy Bird* in an **interactive, motion-controlled VR format**, making flight mechanics feel more intuitive and immersive. The game engages players through **real-world movement**, providing an exciting and physically active experience. Debugging with AI significantly accelerated development, helping to refine physics and motion detection while improving the overall playability.

**GitHub Link (code and builds):** <https://github.com/JordynRabinowitz/FlappyVR>