Ryth-Slice

1. Project Overview

"Ryth-Slice" is a rhythm game where players slice falling slime creatures to the beat of a song. Inspired by Fruit Ninja, this game adds unique mechanics That turns a slice "n" dice game into a rhythm game

2. Project Review

I studied Fruit Ninja and enhanced its gameplay by integrating rhythm-based mechanics. By syncing slime spawns to the music, players must slice with both timing and precision, creating a dynamic and engaging experience.

3. Programming Development

3.1 Game Concept

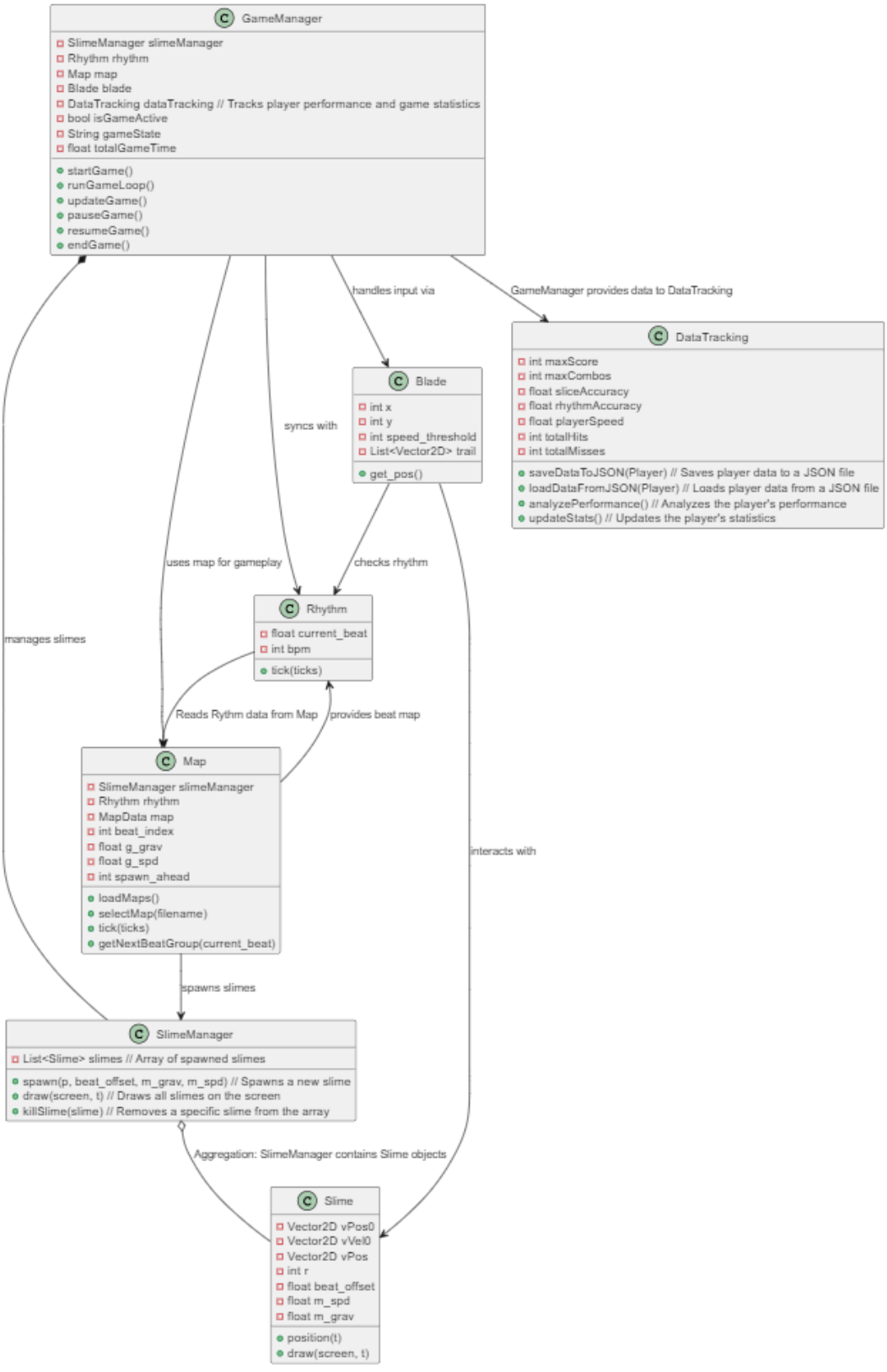
Players swipe to slice slimes while avoiding obstacles to the beat of the song. Each song features a "Map" that dictates the spawn patterns of the slimes. Using an inverse kinematics-based physics system to predict the slime's position at the point of contact "hit". Slimes rise from the bottom of the screen toward the hitpoint, and players earn scores based on their timing: ***Perfect (300), Good (100), OK (50), or Miss (0).*** *Hitting obstacles like barrels and crates results in a* ***-50 point penalty.***

3.2 Object-Oriented Programming Implementation

- ***GameManager*** *(handles overall gameplay, timing and scoring)*
- ***DataTracking*** *(handles data tracking)*
- ***Slime*** *(defines slime properties and behaviors)*
- ***Blade*** *(tracks player swipes and collisions)*
- ***Map*** *(loads map data)*
- ***Rhythm (****Manages sounds and Rhythm of the game)*

- ***SlimeManager*** *(controls the spawn behavior and patterns)*
- ***Player*** *(current pr instance*

UML DIAGRAM:



3.3 Algorithms Involved

The algorithms used in the game

- **Inverse kinematics physics** (to perfectly control the slime's position at the point of contact "hit" on the song's "beat")
- **Rhythm-Based Spawning** (slimes spawn in sync with the song's BPM and beat patterns)
- **Hit Position Prediction-Based Spawning** (slimes spawn below the screen on paths that meet the player's expected swipe arc near the top of the curve)
- **Collision Detection** (detects blade swipes intersecting with slimes in real time)
- **Hit Accuracy** (scores slices based on timing accuracy relative to the beat)
- Combo System (builds score multipliers for consecutive on-beat slices)
- **Obstacle & Power-Up Timing (optional)** (spawns these elements at rhythm-friendly moments for balance)
- **Adaptive Difficulty (optional)** (adjusts spawn density and rhythm complexity based on player performance)

4. Statistical Data (Prop Stats)

4.1 Data Features

Tracked Datas

	Why is it good to have this data? What can it be used for	***How will you obtain 50 values of this feature data?***	***Which variable (and which class will you collect this from?)***	***How will you display this feature data (via summarization statistics or via graph)?***
Score	It tracks the player's performance and progress. Can be used for leaderboard systems and game balancing.	Collected in the game loop from DataTracking.trackMaxScore().	maxScore from DataTracking.	Display as Leaderboard ranking highest to lowest Bar graph. Line graph to show score progression over time per song.
Combos	Tracks player's skill and timing. Used to determine difficulty and progression.	Recorded each time a combo is achieved in the game, using DataTracking.trackMaxCombos().	maxCombos from DataTracking.	Display as Leaderboard bar graph ranking highest combo amount (hits without misses)

Slice Accuracy over Time	Measures how accurately the player slices in sync with the rhythm, important for feedback.	Logged each time Slice.applySliceEffect() is called, measuring how close to the beat the slice was.	sliceAccuracy from DataTracking.	Display as a line graph tracking accuracy over time or a scatter plot showing accuracy per slice event.
Rhythm Accuracy over SongTime	Tracks how well the player is synchronizing their actions with the rhythm of the song.	Tracked every beat in Rhythm.getNextBeat() compared with player slice timing.	rhythmAccuracy from DataTracking.	Display as a scatter plot showing timing deviations or a line graph to show changes over the song's progression.
Song	Indicates which song is being played. Useful for tracking performance by song and for generating stats.	Pulled from Map.beatMapFile each time a new game is started or loaded.	beatMapFile from Map.	Display the current song name in the UI.
Player	Keeps track of player-specific data, including progress and scores.	Obtained from GameManager.updatePlayerInfo() every time the player makes progress in the game.	playerName, playerID from Player.	Display player info in a summary box or as part of the user interface during gameplay.
Hits & Misses	Tracks player performance in terms of hit and miss ratio, useful for difficulty tuning and feedback.	Tracked in CollisionDetection.checkCollision() every time a hit or miss occurs.	totalHits, totalMisses from DataTracking.	Display as a ratio or percentage. Could also be shown in a bar graph or pie chart comparing hits vs misses.

4.2 Data Recording Method

Data will be saved in a JSON object database and analyzed with charts to track player performance and game balance.

4.3 Data Analysis Report

- ***Accuracy Over Time (Line Graph)****: Shows how accuracy changes over time or with game progression.*
- ***Player Performance by Difficulty (Bar Graph)****: Compares player accuracy or score across different difficulty levels.*
- ***Time Played vs. Accuracy/Score (Scatter Plot)****: Displays the relationship between time played and performance.*

- ***Player Comparison (Table)****: Compares player data like score, accuracy, and playtime.*

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (2 April) (25%)	Develop Spawning mechanics Songs Maps
4 (16 March) (50%)	Implement rhythm Sound Mechanics
5 (23 April) (75)	Implement slicing Graphics data tracking
6 (11 May)	Submission week (Draft) refine gameplay test performance

6. Document version

Version: *3.0*
Date: *18 March 2025*