COSC3072 | COSC3073

# Game Studio 1 - Assignment 3:

## Implementing a 3D Clone of Marble Madness

Design Document

**Le Viet Bao - S3979654**

**Nguyen Huu Quoc Huy - S3986423**

**Nguyen Ngo Hoang Nam - S3980297**

## The game design and mechanics

The game is designed as a 3D platformer with a focus on precision jumping, obstacle navigation, and progressive difficulty. The core objective is simple: the player must traverse a series of stages filled with dynamic hazards, reach checkpoints, and ultimately complete the level by arriving at the goal area.

### Player Controls

- **Movement:** The player navigates the world using the WASD or Arrow keys.
- **Jumping:** The Spacebar triggers jumps, allowing the player to overcome gaps and obstacles.
- **Camera Control:** Mouse input allows the player to look around freely and adjust their perspective, enhancing both spatial awareness and immersion.

### Tillable Worlds

Our tileable world is constructed using **Godot's GridMap system**, which enables efficient, modular level design. By importing 3D meshes from online sources, automatically generating collision meshes, and bundling them into a custom resource pack, we establish a reusable toolkit for rapid prototyping. This streamlined process allows layouts to be assembled, tested, and refined with ease—turning level creation into a flexible, almost LEGO-like experience that encourages both creativity and experimentation. However, this methodology is not without limitations:

- The **resource pack is fragile to modification**. When updating or reimporting assets, the order of components must remain identical; otherwise, elements may be mismatched, resulting in incorrect tiles.
- **Editing flexibility is restricted.** There is no native support for grouping or cutting multiple components at once. Misplaced structures require to be cleared and rebuilt entirely.

Once a functional environment is established, we enrich the play experience by populating it with interactive hazards and features. These elements transform static landscapes into lively arenas, pushing players to adapt and refine their strategies.

The game comprises **three progressively challenging levels**:

- **Level One – The Introduction:**

Serving as a gentle onboarding stage, this level minimizes obstacles while presenting zigzagging paths, and platforms that extend both vertically and horizontally. Its purpose is not to punish, but to familiarize players with the full range of movement options available.

- **Level Two – The Challenge Emerges:**

Here, the environment grows more demanding. Subtle nuances in design encourage **trial and error** problem-solving. Yet, the level also rewards ingenuity: players can discover alternate "skips" that allow them to bypass challenges and improve completion time. It's a playground for both patience and daring.

- **Level Three – The Final Gauntlet:**

A dramatic downhill slope forms the backbone of this climax. Obstacles—spikes, swinging hammers, yawning holes—conspire to block the player's descent. Progress can be achieved safely with slow, careful navigation. However, to secure the coveted three-star rating, players must master precise maneuvers to reach the finish line with style and speed.

The design philosophy balances **challenge and fairness**: every obstacle has a predictable pattern that can be mastered through repeated attempts. The game rewards skill, timing, and adaptability.

## How the physics system was implemented

The project runs on **Godot 4** with the **Jolt physics engine**, chosen for its stability, collision handling, and performance over GodotPhysics. While Jolt manages collision detection and contact resolution, core gameplay physics—such as movement, gravity, and obstacle behavior—are handled through custom code to ensure tight and predictable controls.

### Scripted Character Movement

- The player character uses a player script extends `CharacterBody3D`. Instead of relying on raw rigidbody, the script maintains a `velocity` vector and updates it each physics frame.
- **Gravity & jumping** are simulated in code via functions such as `apply_gravity()` and `handle_jump()`. Ground checks use the engine's contact queries (e.g. `is_on_floor()`) to determine when a jump is permitted.

- Movement is applied using Godot's character movement helper (`move_and_slide()`), which combines the scripted velocity with the engine's collision responses to produce stable motion that still feels responsive to player input.

### Interaction with Moving Platforms & Obstacles

- **Moving platforms** are handled in script: when the player stands on a platform the platform's movement offset is applied to the player (rather than relying on constraints). This prevents jitter and keeps platform rides smooth and predictable.
- **Slope handling and surface response** are tuned in code (helper functions like slope-handling routines adjust velocity and prevent sliding), giving precise control over how the player traverses angled surfaces.
- **Hazards and death** are implemented as scripted triggers: collisions with spike rollers and death planes call the game's respawn/reset flow.

### Why this hybrid approach?

- Relying exclusively on engine-driven rigidbody physics produced unpredictable results in early prototypes (sliding, jitter on moving platforms and slopes, inconsistent jump feel). By scripting core gameplay physics (gravity, jump gating, platform offsets, velocity clamping) on top of Jolt's robust collision detection, we achieved both **predictability** and **stability**: collisions remain accurate thanks to Jolt, while player movement remains tight and game-feel is consistent because it's under direct code control.
- Practical example from the codebase: the player script updates `velocity` (horizontal input → acceleration, vertical component → gravity/jump), uses `is_on_floor()` to gate `handle_jump()`, then calls `move_and_slide()`; moving platforms contribute their offset to the player's position each frame so the player moves with the platform without physics jitter.

## Any challenges faced and how they were overcome

During our game design course using Godot, we encountered several significant challenges that required dedicated problem-solving and learning. The implementation of 3D physics proved to be one of the most time-consuming aspects, as achieving realistic ball interactions with various objects demanded extensive trial and error experimentation until the physics felt natural and responsive.

Additionally, applying appropriate textures to the player ball presented its own set of obstacles, particularly since we lacked familiarity with sourcing suitable assets and understanding which texture formats work best within Godot's pipeline. This led to numerous attempts with incompatible or low-quality textures before we discovered the importance of consulting tutorials and the official Godot documentation to understand proper texture implementation techniques.

Furthermore, collaboration within our team became increasingly difficult due to competing priorities from our Capstone Project and insufficient communication between team members, which sometimes resulted in duplicated efforts or conflicting design decisions that had to be resolved through additional meetings and coordination.

Despite all that, we have managed to put together, through massive efforts, a functional and comprehensive game to meet all the requirements given in this assignment.

## Appendix

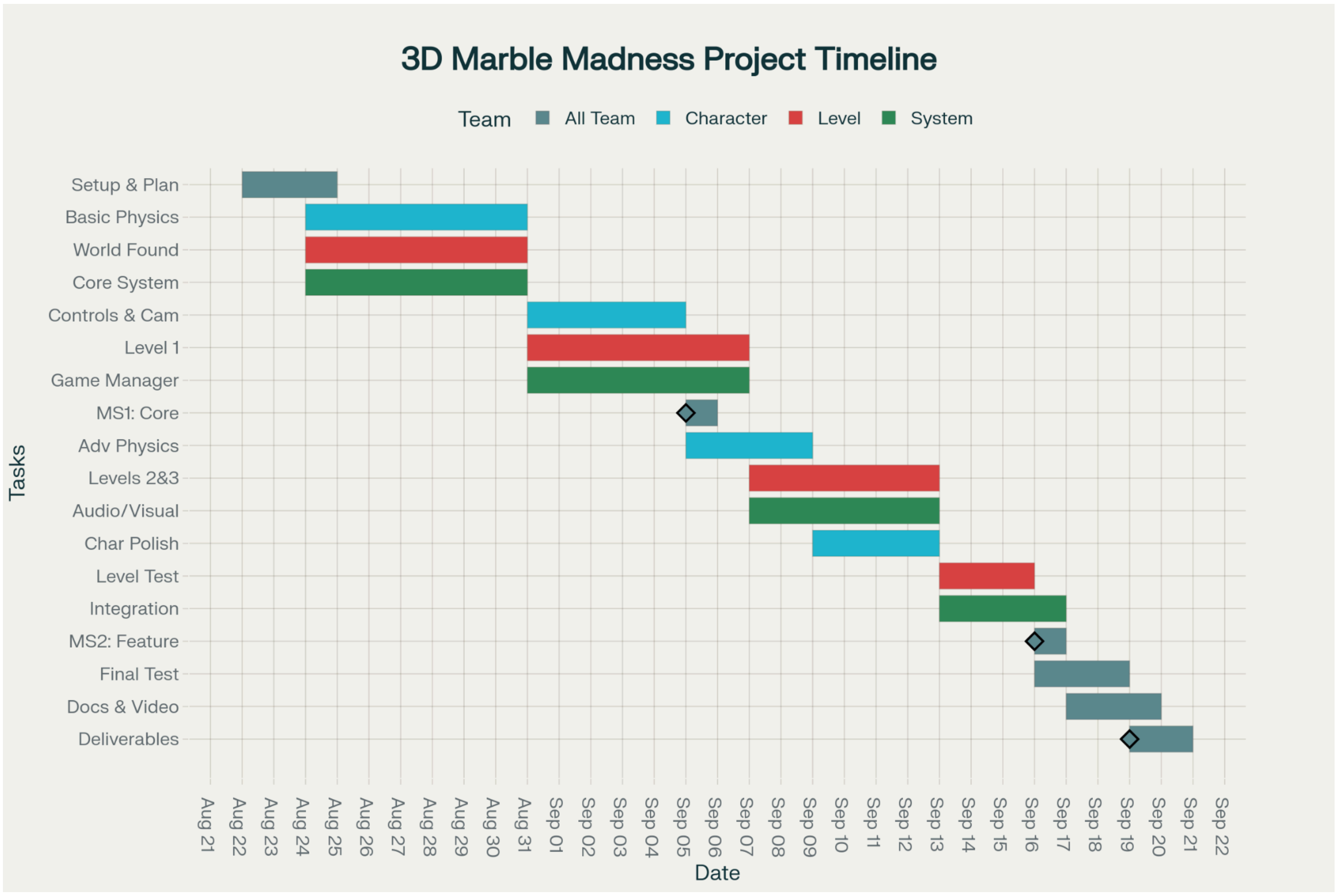

Figure 1: Project Plan