Software Engineering Group Project

COMP2002 / G52GRP: Final Report

Project information:

Project title: UoN Quantum Maze Game

Academic supervisor: Venanzio Capretta

Company name: The University of Nottingham

Team information:

Team number: 12

George Stanway 20541013 psygs6

Xian Diao 20513832 scyxd6

Tabitha Blindu 20549316 psytb12

Hei Kwok 20620504 psyhk6

Matteo Romano 20566511 psymr8

Saad Saqib 20568424 psyss24

Jing Xu 20514985 smyjx3

Mohammed Miah 20551822 psymm14

Contents

Part 1: Final Report

1. Project Background & Understanding

1.1 Introduction

What if learning quantum mechanics could feel like solving a captivating game? Our project turns this idea into reality by offering a dynamic two-dimensional maze game that introduces GCSE-level players to several fundamental principles of quantum mechanics: superposition, entanglement, interference, measurements, and randomness. Each level is created to educate the player by combining interactive gameplay with scientific knowledge to entertain and challenge them throughout. While the game is primarily designed as an educational tool for GCSE and A-level classrooms, it is also accessible to broader audiences of different ages, aiming to balance authentic scientific theory with playful and intuitive interactions.

Beyond learning physical concepts, players are encouraged to reflect on the philosophical dimensions of quantum mechanics—such as observation, uncertainty, and the nature of reality—gaining knowledge, enjoyment, and a deeper appreciation for the quantum world. The project is developed under the guidance of physics professor and draws from a multidisciplinary team combining expertise in computer science, game design, art, and philosophy—resulting in a well-rounded, accessible, and thought-provoking educational experience.

1.2 Depth and context of problem presented

Many existing games have inspired the design of our quantum maze game, each offering unique strengths that we analysed and adapted. Portal, for example, influenced our approach to progressive complexity, cause-and-effect-based puzzle logic, and a clean, sci-fi aesthetic. Fireboy and Watergirl demonstrated how maze-based layouts and interactive triggers can enhance player engagement and cooperative navigation. Meanwhile, Quantum Odyssey showed the potential of combining engaging gameplay with structured educational content in the field of quantum physics.

Building on these inspirations, our game is not just about entertainment or concept delivery—it also encourages philosophical reflection, using interactive levels to guide players through concepts such as uncertainty, entanglement, and observation. Our entire visual and audio design is original, crafted by our team to support immersion and coherence.

We conducted a deep dive into the theoretical foundations of quantum physics, reviewing a wide range of literature and experiments. By transforming complex scientific ideas and rarely seen experimental setups into intuitive, game-based visualisations, we aim to let players see and feel the wonder of quantum phenomena that are otherwise invisible in daily life.

Prior reviews of STEM educational games have primarily focused on knowledge acquisition (Kayan-Fadlelmula et al., 2022; Klopfer & Thompson, 2020), while the effects of such games on cognitive skill development and deeper thinking remain underexplored. Our project seeks to contribute to this growing field with a practical example that fuses education, gameplay, and reflection.

Additionally, while some studies argue that realistic environments support stronger learning outcomes (Fiorella et al., 2018), others suggest that less realistic, more symbolic environments better serve abstract science learning (Riopel et al., 2019). Taking this into account, we intentionally designed a non-photorealistic yet cognitively accessible environment to ensure that players focus on core quantum ideas rather than superficial detail.

To summarise, our game draws inspiration from existing titles while making distinct contributions: philosophical depth, visual originality, academic grounding, and accessibility for a wide range of users—positioning it as both a teaching tool and a research-informed educational innovation.

1.3 Understanding of Quantum Concepts and Game Mechanisms

This game transforms core quantum physics ideas—such as wave-particle duality, superposition, and entanglement—into intuitive gameplay. Through literature review and real-world experiments, we designed mechanics that reflect these principles and invite players to explore concepts like observation, uncertainty, and reality.

1.3.1 Wave-Particle Duality

Concept: Quantum objects act like waves or particles depending on observation.
Game Mechanic: Players switch between Particle Mode (physical interaction) and Wave Mode (passing through obstacles), simulating how observation affects behaviour.

1.3.2 Superposition and Collapse

Concept: A quantum system exists in multiple states until it is observed.
Game Mechanic: In Wave Mode, the player exists in multiple positions, collapsing into one when observed. Some objects only "resolve" upon interaction, reflecting measurement collapse.

1.3.3 Entanglement

Concept: Entangled particles are instantly connected, regardless of distance.
Game Mechanic: Linked elements like doors and switches change state together. Solving puzzles requires understanding these non-local connections.

1.3.4 Interference

Concept: Interference patterns emerge from overlapping wave functions and disappear when observed.
Game Mechanic: Certain puzzles change depending on whether the player is being observed, introducing uncertainty and strategy.

1.4 Our scope

Category	Description
Target Audience	The game is designed primarily for GCSE-level students with an interest in quantum mechanics, or those aiming to strengthen their understanding of the subject.
Accessibility	No prior physics knowledge is required. This aligns with our core objective of making quantum mechanics more approachable.
Conceptual Depth	Quantum principles such as superposition, entanglement, interference, and measurement are integrated directly into gameplay, offering hands-on interaction for reinforcing theoretical understanding.

Application Context	The game is suitable for both classroom and individual use, making it a flexible teaching and learning tool for formal education or independent exploration.
Pedagogical Approach	The gameplay mirrors the core behaviours in quantum systems, enabling a visual and kinaesthetic learning experience. This supports a wide range of learners, particularly visual learners.
Scaffolded Learning Design	Each level increases in complexity, gradually introducing quantum principles. This supports self-driven learning at the player's own pace, illustrating the diversity in learning styles.
Multisensory Engagement	The game features original graphics and a custom soundtrack to create an immersive educational experience.

1.5 Summary of Project Background & Understanding

This project aims to create an educational 2D maze game that introduces key quantum mechanics concepts through interactive and accessible gameplay. Motivated by the difficulty of teaching abstract quantum principles, the game translates phenomena such as superposition, entanglement, and interference into visual and playable mechanics. The project is grounded in quantum theory, game-based learning research, and philosophical inquiry, and is designed for use in GCSE/A-level classrooms as well as by broader audiences. With a low entry barrier, original design, and a strong interdisciplinary foundation, the game serves both as a learning tool and a thought-provoking experience.

2. Change Section

This section highlights all key additions, removals, and updates made since the Interim Report.

2.1 Structural Changes

Sections Removed

- "Skills Transfer"
- "Forward Planning"
- "Initial Understanding of Brief"

Sections Added

- "Scope" & "Scope Limitations"
- Individual Level subsections (Levels 1–4), each with:
  - Design & puzzle planning
  - Implementation & programming
  - Testing & iteration
  - Art & sound integration
  - Supervisor feedback
- Comprehensive "Reflection" (6.1–6.6)
- "Risks & Mitigations" under Project Management
- LSEPI Reflection (Legal, Societal, Ethical, Professional Issues)
- Expanded Future Work & Roadmap

2.2 Content Updates

- Revision of the "General Requirements" section to reflect evolved project requirements, changes in scope, and revised priorities based on supervisor input and team discussions.
- Project Management & Progress updated with real meeting cadences, supervisor-gap adaptations, and risk log.
- Team Roles updated to reflect Hana's departure and Mohammed's arrival, plus evolving cross-functional responsibilities.
- "Level Design" sections rewritten to include final puzzles, diagrams, and playtest refinements.
- "Lessons Learned" and "Future Improvements" with next steps, version roadmap, and out-of-time features.

2.3 Workflow and Methodology Updates

The team adopted a more structured and efficient workflow in the second term. We established a rhythm of two weekly internal meetings (on Wednesdays and Fridays) for progress tracking and task reallocation, alongside a weekly supervisor meeting for theoretical guidance and project review.

We migrated all project management activities—including documentation, resource/link sharing, scheduling, and milestone tracking—into a centralised Notion workspace, custom-built for our team. This platform significantly improved coordination, version control, and responsiveness to changes.

2.4 Team Adjustments and Role Evolution

Several team changes occurred during this phase. Hana left the team due to an exchange program in New Zealand, while Jing assumed the role of team admin, and a new member, Mohammed, joined the team.

Initial roles were clearly defined:

- Coding: George and Selena
- Testing: Saad and Xian
- Reporting: Tabitha
- Level Design: Jing and Matteo
- Art: Jing

As the project progressed, responsibilities became more flexible. Jing contributed across coding, visual design, narrative development, and documentation. Tabitha, Saad, Xian, and Mohammed became involved in level design and refinement. George continued leading code integration and later compiled the User Manual.

2.5 Progress and Challenges

The project has achieved a high level of completeness, with all core levels implemented and polished. While certain early ideas proved technically infeasible or more time-consuming than expected, the team-maintained momentum through effective task reassignment and open communication. Thanks to our adaptive planning, collaborative spirit, and shared commitment, we addressed challenges in a timely and efficient manner.

3. Requirements & Critical Analysis

3.1 Initial Requirements

The initial requirements of the project were established through collaborative discussions, supervisor feedback, and an extensive review of quantum physics literature and educational game design research. From the outset, the primary objective was to develop a game that serves both as an interactive learning tool and an engaging gameplay experience, aiming to introduce complex quantum mechanics concepts in an intuitive and accessible way.

Core gameplay features were derived from key quantum phenomena—such as superposition, entanglement, and interference—and were translated into mechanics designed to be both pedagogically meaningful and playable. At the same time, user experience considerations such as responsiveness, visual clarity, and progression pacing were included to ensure usability across a wide audience, including students with no prior exposure to physics.

To support decision-making and development focus, we applied a simplified MoSCoW prioritisation model, categorising each requirement as *Must have*, *Should have*, or *Could have* based on its relevance to the learning objectives, gameplay coherence, and technical feasibility.

A key narrative framework that shaped the game's structure was The Story of the Particle—a philosophical journey across four levels. Each level introduces different quantum principle while exploring a philosophical question (see Appendix).

This narrative-driven structure not only supports gameplay progression but also allows players to reflect on the deeper implications of the quantum world, bridging scientific understanding with personal discovery.

A detailed breakdown of these requirements, including their descriptions, implementation targets, and priority levels, is presented in the table which is putted at Appendix. (For more specific requirements refer to Obsidian.)

3.2 Evolution of requirements

The game's evolving features, gameplay, and narrative elements have been developed to ensure that they meet technical feasibility and the realistic random nature of quantum processes.

Initially, the game featured basic electrical inputs and outputs, but these have been upgraded to include features such as probability detectors and multiple tile types. These are developed to provide a dual gameplay mechanic, where players experience the world as both a particle and a wave, adding a unique dimension to the game.

In addition, new interactable entities, such as collectables, have been introduced, and advanced mechanics have been developed to enhance the maze. These collectables introduce power-ups, allowing the player to gain control over features such as time and energy, highlighting how quantum processes are similarly influenced by external factors.

As players advance through the maze, they encounter checkpoints that mark their current position, fuelling the exploration of new quantum events in the next level. These checkpoints also encourage personal discovery, raising important philosophical questions about identity.

A new game feature introduced is the ghost mechanic, where the player encounters a mysterious, seemingly omniscient figure. This ghost is initially perceived as simply a guide but is later revealed to be the player's future self, a quantum-entangled counterpart from a parallel world. The ghost guides the player through introductory puzzles and environmental changes, exhibiting subtle clues and behaviours. The modular design allowed the development of the ghost to mirror the player, utilising the same movement, logic, and interaction with puzzles.

The player and the particle both evolve throughout the game, as just as the player uncovers the mechanics of the game world, the particle navigates its own path. This joint journey blurs the line between learning and gameplay, where the player is learning at the same time as the particle, emphasising the game's theme of self-discovery and exploration.

As the game progresses, the connection to the ghost strengthens, allowing the two worlds to collide and interact.

To enhance the learning experience and immersion, we implemented original visual assets and animations using Procreate and Adobe After Effects, added effects, sound design, and refined UI elements. Dialogue-based tutorials and progressive player guidance were also developed to improve accessibility for first-time players.

3.3 Technical Tools and Design Strategy

3.3.1 Game Engine and Development Tools

We used several of Godot's systems in our project, the main ones being the 2D physics, navigation, and tile systems as preparation for the prototype level. The Godot Git plugin, alongside Visual Studio Code, allowed for efficient version control and conflict resolution.

Midway through development, we upgraded from Godot 4.4 to 4.4.1 to take advantage of improved signal handling and bug fixes that it offered, resulting in a smoother and more stable gameplay experience.

3.3.2 Programming Workflow and Code Quality

Two primary programmers led development using paired programming, in which one developer focused on real-time coding, whilst the other reviewed and improved the existing program. This approach led to improved code quality and significantly reduced major bugs, ensuring that sprint deadlines were met.

By collaborating on interdependent systems—such as player control and puzzle logic—development was more cohesive and iterative, allowing for a more stable and integrated feature implementation.

3.3.3 Level Design and System Architecture

Prior to implementation, level layouts were prototyped using tools like Dungeon Scrawl. This iterative process allowed the design team to experiment with structure and puzzle composition before committing to finalised tilemaps in Godot.

These designs were then integrated into a modular system architecture, where individual components—such as entities, triggers, and tiles—could be reused across levels. This ensured both scalability and consistency, and improved post-development testing, as components could be tested in isolation before being introduced into the full game system.

3.4 Feedback and User Insights

Due to time constraints and the complexity of our development process, we were unable to conduct formal user studies or structured surveys. Unlike simplified or template-based games that reuse existing mechanics with minor visual modifications, our project was built from the ground up with original gameplay systems, hand-crafted art, and educational content grounded in quantum theory.

Ensuring the game's theoretical accuracy, playability, and aesthetic coherence required constant iteration, substantial testing, and extensive creative effort.

Throughout the development cycle, we invested significant time in playtesting, debugging, and fine-tuning, which led to a slower development pace. As a result, the final playable version of the game was completed relatively close to the project deadline, leaving insufficient time for conducting structured external evaluations.

Nevertheless, we actively collected informal yet highly valuable feedback throughout development. We conducted gameplay testing with team members, computer science peers, and friends or family members with varying levels of gaming experience and familiarity with quantum physics. These sessions helped us observe points of confusion or difficulty—such as unexpected puzzle bottlenecks, unclear mechanics, or insufficient visual cues.

Based on these observations, we made iterative adjustments to:

- The difficulty progression and pacing of new mechanics
- The visual clarity of interactable elements
- The tutorial structure and guidance flow

Although informal, this feedback significantly shaped the game's overall user experience and educational impact. It helped us ensure the game remained **intuitive** and **accessible**, while still delivering on its conceptual depth.

4. Project Management & Progress

4.1 Project Management Approach

Our team adopted a structured and collaborative project management strategy, grounded in Agile principles and tailored to suit the iterative demands of educational game development. While not following a strict textbook SCRUM process, we maintained a lightweight Agile methodology focused on continuous feedback, adaptive planning, and modular delivery.

The central hub of our management process was a custom-designed Notion workspace, developed specifically for our team. This workspace served as an integrated platform for: Sprint planning and Kanban-style task management. Meeting records, including agendas, summaries, and discussion outcomes. Resource organisation, including links to Git repositories, milestone plans, and key documents (e.g., midterm report, project overview). A visualised timeline and calendar, connecting milestones with daily, weekly, and long-term tasks. Quick-access templates for creating tasks, meetings, and new milestones. A linked architecture, in which individual tasks were nested under broader milestones, which were themselves connected to major project modules

This level of organisation allowed us to track progress in real-time, reassign responsibilities dynamically, and ensure team-wide transparency throughout development.

4.2 Weekly Workflow and Meetings

To maintain steady momentum and responsiveness, our team followed a structured weekly cycle that combined remote coordination, on-site collaboration, and expert supervision.

Every Wednesday, we held an online team sync via Microsoft Teams to report progress, adjust tasks, and align on short-term goals. Each session was guided by a prepared slide deck outlining task status, blockers, and priorities, ensuring meetings were focused and actionable. On Fridays, we gathered for in-person working sessions on campus to consolidate deliverables such as code integration, visual asset assembly, and level testing.

These sessions enabled real-time collaboration across programming, art, and design roles. Immediately afterward, we met with our academic supervisor to validate the scientific accuracy of our quantum mechanics implementations, review level design proposals, and resolve conceptual or technical challenges. Feedback from these meetings often led to critical course corrections, particularly in areas involving complex quantum principles like entanglement and uncertainty.

4.3 Planning and Tracking

Our project followed a detailed multi-phase development plan, spanning from early February to mid-May. The plan included clearly defined deliverables for code, design, testing, documentation, and presentation work. Overall, we were able to adhere closely to the original timeline, with only minor shifts in scope and scheduling.

During implementation, we adjusted the time allocated for art production, level design, and code integration. In particular, the design and implementation of Levels 1 and 2 took longer than expected due to iterative redesign and trial-and-error playtesting. However, this foundational work accelerated the development of Levels 3 and 4, which were completed more efficiently. The art task also required more time than initially anticipated, especially for animation, visual effects, and UI polish. These changes required continuous coordination with the development team, and therefore, code development remained active until the final stages of the project.

Due to the Easter break, the team report was completed slightly later than planned. However, thanks to strong team communication and a proactive restart process after the holiday, we were able to quickly resume tasks and stay aligned with remaining deadlines.

Please refer to the table in the Appendix for planned phases and corresponding tasks.

4.4 Team Roles and Collaboration

At the outset of the project, team responsibilities were divided based on members' previous experience, strengths, and familiarity with tools such as Godot. Roles were initially assigned as follows: George and Selena focused on coding, Saad and Xian were responsible for testing, Tabitha led reporting and documentation, Jing and Matteo handled level design, while Jing also undertook the visual art and UI design.

As development progressed, the team demonstrated a high degree of flexibility and cross-functional collaboration. Once core functionality and initial testing were completed, Saad and Xian shifted from testing to level design, applying their strong sense of gameplay mechanics to create engaging puzzles. Tabitha, after establishing the documentation framework, also contributed to level logic and puzzle design.

A new member, Mohammed, joined in the second semester. As he had not previously participated in the Godot learning phase, his contributions were focused on level design, where he helped implement ideas and refine layout details.

George and Selena continued leading software development, delivering the core functionality and integrating gameplay systems. As the project neared completion, George also contributed to the User Manual and group report. Meanwhile, Jing maintained an ongoing administrative role—organising meetings, managing the Notion workspace, and assigning tasks. In addition, Jing played a major role in defining the game's final visual style through extensive experimentation with 2.5D/3D layouts and colour palettes. After finalising the artwork, she expanded her involvement to coding and puzzle design and later joined in writing the final report and software manual.

Throughout development, Matteo remained a central contributor in level design, efficiently integrating levels into Godot. His initiative helped streamline level development and provided structural consistency across the game.

This adaptive collaboration model, informed by our shared experience from the previous term, led to more efficient communication, faster integration cycles, and a stronger alignment between gameplay, visuals, and educational goals.

5.Game Design and Implementation

This section summarises key aspects of the implementation of the project. Detailed architecture, class diagrams, and modular documentation can be found in the accompanying Software Manual. The focus here is to highlight the core systems, game logic, and implementation strategies that underpin the interactive quantum mechanics experience.

5.1 Core Systems Overview

The game consists of four progressively complex levels, each introducing new quantum-inspired mechanics and gameplay elements. From foundational tutorials on wave-particle switching to advanced puzzles involving entanglement and energy management, each level is designed to scaffold player understanding while maintaining engaging, fast-paced gameplay.
For detailed explanation, please refer to the Software Manual.

5.2 Level Design

Level 1
Level 1 introduces both the maze structure and the fundamentals of quantum physics, guiding players through basic electrical components and the core Particle-Wave mechanic. Initial designs were sketched on paper, numbered by puzzle introduction points, then mapped in Dungeon Scrawl and refined in Godot through supervisor and playtesting feedback.

The electrical system—comprising buttons, switches, levers, and doors—teaches cause-and-effect in a safe environment. Players switch between Particle and Wave Modes, with Wave Mode lasting briefly and indicated by an energy bar and timer. This mechanic adds urgency, requiring players to navigate obstacles quickly before collapsing back into stable Particle Mode. Quantum tunnelling, a rare move allowing passage through walls, adds strategic variety. A patrolling enemy introduces timing and dodging challenges, while countdown-based doors reinforce pressure. Falling into bottomless pits in Particle Mode results in game over, reloading from the last checkpoint.

Level 2
Building on the basics, Level 2 deepens the wave-particle duality concept with hazards, energy management, and checkpoint-based progression. Complex ideas like entanglement are postponed keeping the pace manageable. Crystals scattered across the maze replenish State Energy, encouraging risk-reward decision-making during dashing or other high-cost actions. Probability detectors activate in Wave Mode and collapse states to open doors, introducing measurement-driven mechanics.

The dash ability consumes energy and is key to momentum-based puzzles—e.g., activating heavy buttons or breaking through marked walls using particle accelerators. These systems collectively create a level design that rewards efficiency, enabling speed-running and encouraging replays.

Level 3

Originally designed as a large, complex maze, Level 3 was streamlined based on feedback to improve engagement. The map was simplified for more linear progression, while barren areas were repurposed with puzzles and collectables to increase element density. Wiring bugs were fixed to ensure reliable functionality of switches and detectors.

To maintain pacing, the level introduces new hazards such as spikes and black holes alongside more particle accelerators and enemies. New mechanics like entanglement are explained briefly, then reinforced through action, maintaining flow without overwhelming the player. The revised design balances traversal, learning, and gameplay diversity.

Level 4

Level 4 removes most on-screen prompts, encouraging players to apply their accumulated knowledge in more seamless, integrated scenarios. Familiar elements—spikes, black holes, electrical systems—are combined into layered puzzles. For example, the opening challenge merges spotlights, Wave Mode, and electrical interactions to test understanding in a compact space.

A major innovation is the introduction of enemy entanglements, enhancing the quantum theme while adding unpredictability to encounters. The level continues the trend toward linearity, with tighter pacing achieved through mini puzzles and reduced traversal between gameplay segments. This results in a smoother, more focused gameplay experience.

5.3 Testing and Debugging Practices

As the project progressed into its later stages, our approach to testing became more formalised. Recognising the growing complexity of our systems and the limitations of manual playtesting, we introduced structured unit testing using GUT (Godot Unit Testing). We created dedicated test scripts for core components—such as the player, enemies, buttons, and prisms—allowing us to validate functionality in isolation and ensure consistent behaviour. Initially, the focus was on mocking dependencies and isolating units under test. This helped us test specific behaviours (like the player's health system or button activation) without relying on full scene instantiation. We also used GUT methods like before_each() to ensure a clean state before every test, preventing interference between cases.

As our experience with GUT grew, we began testing more dynamic interactions. For physics-based components, we manually advanced physics frames in our tests to simulate runtime behaviour, allowing us to verify collision-based logic more accurately. For example, to test a button being pressed by the player, we explicitly processed multiple frames in the script using physics_process() and then checked both visual changes and whether connected components received the correct signal. These tests became particularly valuable helping us to catch subtle bugs that wouldn't have been obvious through manual testing alone.

While we still relied on our prototype level for integration testing, unit tests improved our workflow by making it easier to validate individual scripts, debug issues quickly, and feel confident when making changes. Although we didn't reach full coverage, especially for peripheral items (like the solar panel), the testing we implemented played an important role in stabilising our project in the final stages.

6. Contributions & Reflections

6.1 Project Outcome Summary

The project successfully delivered a complete playable game featuring:

- Original visual art, animations, effects, and UI
- Background music and in-game sound effects
- Four fully developed levels, each comprising a set of puzzles and gameplay sequences
- A complete set of deliverables, including the code, team report, Software Manual, User Manual, and a showcase poster

However, a few planned elements were not fully realised. These include:

- A more detailed in-game explanation of the narrative and worldbuilding
- Expanded story guidance throughout the levels
- Multiple sets of alternative art style explorations beyond the final chosen visual theme

6.2 Key Challenges and Solutions

Several major challenges were encountered throughout development:

Challenge	**Impact**	**Mitigation/Solution**
Rapidly evolving mechanics requiring continuous retesting	Frequent changes broke existing scripts, and testing. This delayed milestones.	Adopted iterative prototyping, using paper sketches for proof-of-concept scenes before a full implementation.
Balancing playability with scientific accuracy in level design	Complex puzzles risked being non-immersive and too simplified to reflect quantum concepts.	Used informal playtests with non-expert peers to validate clarity and provide guidance for puzzle designing. Regular supervisor feedback was implemented for each level.
Art was handled by one team member	This required significant trial and error, which was further constrained by limitations in early-stage code architecture that restricted visual implementation.	Reallocated team responsibilities for team member to focus solely on the art.
Increasingly complex code integration and debugging workload	Merging multiple feature branches slowed down the final phase.	Major code changes and updates were pushed with all members of development team present to avoid any possible conflicting merges.

In summary, our team prioritised flexible role adjustments, reallocated responsibilities as needed and made decisive scope trade-offs to maintain project focus. We were also conscious in avoiding sunk-cost fallacies during decision-making, which helped in preserving time and effort for high-impact features.

6.3 Tool and Methodology Rationale

At the start of the project, we selected Godot as our development engine after a structured decision-making process. In this phase, we upgraded to Godot 4, leveraging its enhanced signal handling and feature stability.

To streamline project coordination, we consolidated multiple planning and file management tools (including Jira Sprint Planning, Microsoft Teams file sharing, and Google Docs) into a customised Notion workspace. This consolidation enabled better accessibility, reduced friction in task tracking, and avoided lost documentation or duplicated work. The integrated system also supported agile iteration and more consistent sprint planning.

6.4 Team Operations Reflection

While the team showed strong initiative, one area of improvement was in early-stage planning. Some design and scheduling decisions lacked foresight regarding their technical complexity or the team's available bandwidth. This led to frequent adjustments to timelines and priorities, which, although effectively handled, resulted in some avoidable overhead and inefficiencies.

Compared to the previous term, team communication and coordination improved significantly. Adjustments to meeting formats and schedules led to more focused and productive discussions. With a clearer understanding of each member's strengths and limitations, the team collaborated more smoothly and responsively.

In high-pressure stages, the team demonstrated strong mutual support, flexibility, and a willingness to adapt. Active listening, respectful debate, and shared decision-making played a vital role in maintaining cohesion and motivation during critical milestones.

Midway through the second semester, out academic supervisor, Dr Capretta, left the University of Nottingham, so we quickly shifted to a student-led model. We independently scheduled one last meeting where he reviewed all the work completed over the Easter break- finalised levels, polished art and music, and near-final tutorial sequences. His focused feedback on the level design and pacing was integrated into Level 4, greatly improving the player experience.

6.5 Lessons Learned

As our project evolved, our team discovered key lessons that were applied to future development:

- **Realistic scoping** is required during early design phases, especially regarding implementation difficulty and the availability of team capacity. By considering technical limitations, tool constraints, and members' time more carefully, we could have reduced unnecessary rework and enhanced delivery efficiency.
- **Iterative prototyping** is the key to game design. Specifically, utilising a range of low and high-fidelity prototyping such as Dungeon Scrawl is key to combining multiple electrical components. This loop saved time and ensured the requirements were achieved.
- **Playtesting** is necessary. Whether this is within the team, or by peers, these tests highlighted UI ambiguities and pacing issues that would otherwise go unnoticed. These formed the basis of future levels.
- **Early expert validation** is required to avoid costly rewrites. Consulting our supervisor before coding major physics mechanics would have significantly reduced rework for all teams. This is especially important when dealing with a more challenging concept i.e., entanglement.

6.6 Future Improvements

Category	Planned Improvement	Description
Version Updates	V1.0: Level Upgrades V1.1: Touch Controls V1.2: Circuit Update	Improve levels, add mobile gestures, and refactor the electrical system for stability and accessibility.
Tutorial Enhancement	Expanded Ghost Dialogue	Add branching dialogue and contextual hints to improve player onboarding.
Visual Design	Alternative Art Stylised 3D Visuals	Enable multiple art themes (e.g., hand-painted, retro-futuristic); transition from 2D to stylised 3D assets for richer immersion.
Platform Compatibility	Mobile-Friendly UIVR Demonstrations	Adapt UI/controls for mobile devices; build VR classroom/exhibit versions with spatial audio and interaction.
Audio	Upgraded Sound Effects	Add dynamic sound effects and ambient audio to enhance immersion and feedback.
Gameplay Mechanics	Item System Overhaul Multiplayer Mode Additional Levels	Redesign item tracking; introduce cooperative puzzles; expand with maps based on advanced quantum experiments.
Narrative & Style	Revised Story & Art Direction	Rework the game's narrative and give each level a distinct art style to enhance story immersion.
Quantum Integration	Physics-Based Extensions	Incorporate emerging quantum theories and phenomena into new puzzle mechanics.
Community Features	Mod Support	Allow user-generated levels and mechanics to encourage community creativity and content expansion.

7. LSEPI Reflection (Legal, Societal, Ethical, and Professional Issues)

The project posed minimal legal, ethical, or data-related risks, but several relevant considerations were addressed during development.

Intellectual Property

All core assets—including game code, visual art, UI, animation, level designs, and logic systems—were developed by the project team and are original in nature. The only exception is the **background music**, which was generated using **Suno**, an AI-based music generation platform. The output is non-commercial and royalty-free, aligning with our project's educational purpose.

Unless otherwise noted, code and assets will be released under the **MIT License**, allowing open reuse for educational or non-commercial development.

Research Ethics and Data Protection

The project did not involve any formal human research or data collection. No personal or identifiable user data was gathered during development or playtesting, and thus compliance with the **GDPR** or similar frameworks was not applicable.

Accessibility and Legal Compliance

Although the game was not explicitly designed to meet full accessibility standards (e.g., WCAG), basic principles such as **clear visual feedback, text-based prompts**, and **keyboard operability** were followed. No security-sensitive data is used or stored, so no risk under the **Computer Misuse Act** arises.

Broader Societal Impact

The game aims to make **quantum mechanics more approachable to learners**, especially those in GCSE or A-level education. By reducing cognitive and technical entry barriers, the project has the potential to **support STEM outreach and inspire interest in physics**.

7. References

Fiorella, L., Kuhlmann, S., & Vogel-Walcutt, J. J. (2018). Effects of playing an educational math game that incorporates learning by teaching. *Journal of Educational Computing Research, 57*(6), 1495–1512. https://doi.org/10.1177/0735633118797133

Kayan-Fadlelmula, F., Sellami, A., Abdelkader, N., & Umer, S. (2022). A systematic review of STEM education research in the GCC countries: Trends, gaps and barriers. *International Journal of STEM Education, 9*, Article 2. https://doi.org/10.1186/s40594-021-00319-7

Klopfer, E., & Thompson, M. (2020). Game-based learning in science, technology, engineering, and mathematics. In J. L. Plass, R. E. Mayer, & B. D. Homer (Eds.), *Handbook of game-based learning* (p. 387). MIT Press.

Riopel, M., Nenciovici, L., Potvin, P., Chastenay, P., Charland, P., Sarrasin, J. B., & Masson, S. (2019). Impact of serious games on science learning achievement compared with more conventional instruction: An overview and a meta-analysis. *Studies in Science Education, 55*(2), 169–214. https://doi.org/10.1080/03057267.2019.1722420

8. Appendices

- **Code repository** (including obsidian volt with further documentation)
  https://projects.cs.nott.ac.uk/comp2002/2024-2025/team12_project
- **Notion workspace**
  - **Meeting Notes:**
    https://www.notion.so/18b0109b44de807fa672f82d36433840?v=18b0109b44de80d5813b000cc71cf9e9&pvs=4
  - **Collaborative Planning:**
    https://www.notion.so/18b0109b44de819ca4b6ef3b8fc5cf2b?v=18b0109b44de811d916c000ca9847b42&pvs=4
  - **Project Management:**
    https://www.notion.so/18b0109b44de8134b13beafeaf510987?v=18b0109b44de81659784000cb4983017&pvs=4
- **Microsoft planner**
  https://planner.cloud.microsoft/webui/plan/6u3UlmuTGEyAKLD-6-l5bZYAEwQ1?tid=67bda7ee-fd80-41ef-ac91-358418290a1e
- **Godot introduction tutorial**
  https://youtu.be/nAh_Kx5Zh5Q?si=GrtjZHCkBC_t1fLx

Requirements table:

ID	Feature	Description	Must Do	Priority
1.1	Particle Mode	Player can play as an atom with energy management as the core mechanic. Rest Mass Energy acts as base health, while Quantum State Energy is used for actions like attacks and movement. Particle should also have abilities such as dash, shooting photons and shrinking/expanding.	The game must provide Particle Mode as the primary state with energy as the central mechanic.	High
1.2	Rest Mass Energy	Represents the atom's intrinsic energy; acts as base health and is unaffected by most external factors.	Rest Mass Energy must act as the player's base health and remain stable under normal gameplay.	High
1.3	Quantum State Energy	Represents the energy associated with the atom's electrons. Increases by absorbing energy sources like photons or environmental energy. Used for abilities and acceleration.	The game must include a dynamic Quantum State Energy system for abilities and energy absorption.	High

2.1	Energy Absorption	Player can absorb energy via photons or environmental sources to increase Quantum State Energy.	Players must gain energy through photons and environmental sources to power actions.	High
2.1.1	Photon Absorption	Different coloured photons provide varying energy levels. Red (low energy), Yellow (medium energy), Blue (high energy). Absorbing photons increases Quantum State Energy.	The game must support photon absorption with distinct energy levels tied to colours.	High
2.1.2	Environmental Energy Fields	Heat sources, electromagnetic fields, and other areas provide ambient energy to recover Quantum State Energy during unstable states.	Players must have access to recover Quantum State Energy through environmental energy fields.	Medium
2.1.3	Energy Collection Devices	Solar Panels convert player-emitted photons to energy. Energy Prisms disperse photons by colour for selective absorption based on the player's needs.	The game must include energy collection devices for puzzle-solving and energy management mechanics.	Medium
2.2	Energy Emission Abilities	Quantum State Energy powers various abilities.	The player must use Quantum State Energy for specific abilities like photon shooting and acceleration.	High
2.2.1	Photon Shooting	Consumes Quantum State Energy to emit gamma photons, damaging enemies and powering electrical inputs.	Players must shoot photons as an attack method and a way to interact with certain devices.	High
2.2.2	Accelerated Movement	Consumes energy to simulate high-speed electron motion. Excessive energy usage may cause instability, represented by heat generation and quantum fluctuations.	Players must accelerate movement using Quantum State Energy with consequences for overuse.	Medium

2.2.3	Energy Interaction with Devices	Player can activate devices (e.g., buttons, levers, particle accelerators, breakable walls) or manipulate neutral particles by transferring energy, inducing transitions, or altering particle states.	Players must interact with devices and neutral particles using energy mechanics for puzzle-solving.	High
3.1	Wave Mode Setup	Activated via key press. Simulates quantum wave behaviour with a time-limited state. Amplitude decreases over time, simulating dispersion. External disturbances (e.g., attacks or measurements) accelerate wave collapse.	The game must allow players to switch to Wave Mode for specific tasks, with a decay and collapse timer.	High
3.1.1	Wave State Fluctuations	Quantum State Energy in Wave Mode exhibits random small fluctuations, visually represented by jittering or dimming effects.	Wave Mode must visually represent fluctuations through effects like jittering and dimming.	Medium
3.2	Wave Mode Gameplay	Core mechanics leveraging quantum physics principles like duality, entanglement, and interference.	The game must implement unique Wave Mode interactions based on quantum physics concepts.	High
3.2.1	Double-Slit Experiment	Player can pass through slits in Wave Mode. Observing interference patterns demonstrates wave-particle duality. Observation by enemies or devices forces wave collapse into Particle Mode.	Players must pass through double slits in Wave Mode and observe interference patterns.	High
3.2.2	Entanglement Operations	Player can link together the switches in sequence to entangle and observe the state of the door.	The game must allow entanglement between buttons and doors to allow puzzle progression.	Medium
3.2.3	Wave Interference Effects	Interaction with environmental waves: Blue Waves amplify and extend Wave Mode duration (constructive interference); Red Waves accelerate wave decay (destructive interference).	Players must interact with environmental waves to affect Wave Mode duration.	Low

4.1	Electrical Components	Various map elements interactable by the player to impact outputs, such as doors or mechanisms.	The game must include interactive electrical components for puzzles and progression.	High
4.1.1	Electrical Inputs	Includes buttons, switches, solar panels, possibility detectors, and prism docks. These components are manipulated using Particle or Wave mechanics for puzzle-solving or progression.	Players must interact with different types of electrical inputs to solve puzzles.	High
5.1	Tutorial Ghost	Player is guided through the introductory process by its future self (the 'ghost'). This includes both solving simple puzzles and reacting to the system.	Ghost must demonstrate completion of introduction puzzles, and interaction with electrical inputs.	High
5.1.1	Gradual Player-System Collaboration	Player explores the system while advancing through levels, discovering a much more vibrant and interactive environment,	The game's graphics and animations must become more complex and interactive through progression.	High
5.1.2	Connection with Another Self	As the player advances, they begin to suspect they are connected to another entity in the Mirror Quantum World.	The game must introduce a system that allows the player to start interacting with the entangled entity in subtle ways (e.g., mirrored actions, shared experiences).	Medium
5.1.3	Multiple Pathways and Probabilistic Reality	Based on the Heisenberg Uncertainty Principle, the system is introduced to provide multiple outcomes from the same action, representing different possibilities collapsing into reality.	The game must create environments where the same actions (e.g., pulling a lever), leads to multiple outcomes, where only one becomes 'real' once the player chooses.	High

5.1.4	Philosophical exploration through guidance	The game presents multiple versions of the world, shaped by the player’s decisions, with the player’s future self serving as a guide on their path of self-discovery.	The game has dialogue and visual cues that guide and inform the player through their exploration.	High
5.1.5	Mirrored worlds	The player discovers that their actions in one world directly affects the other. E.g., a door opening in both worlds.	The game must have mechanics that allows the player to interact with entities in one world, causing events to unfold in the other.	High
6.1.1	Gameplay Collectables	Players can collect various items that provide power-ups such as extra time and energy.	The game must have collectable items that can be picked up throughout the game.	High
6.1.2	Regular Checkpoints	Players collect flags as checkpoints to save progress while navigating the maze.	The game must allow the player to collect flags to mark their position in the maze.	High
6.1.3	Main Menu with Save Progress	Main menu that allows players to navigate to previously saved game states.	The game must have a main menu where players can load previous game states to continue their progress.	High
6.1.4	Recording of Best Time	Record and display the best time for each level completed to encourage competition and replay.	The game must track and display the player’s best time for each level.	Medium

Planning table:

Phase	**Key Tasks**
Feb 6 – Feb 25 Basic Features & Art Design	- Develop core mechanics (quantum entanglement, UI, save/load) - Design characters and environments - Define level structure and goals - Unit testing - Draft music direction
Feb 26 – Mar 10 Finalise Basic Features & Core Art	- Complete all core code features and UI - Finalise character, UI, and environment art - Create animations and effects - Functional testing
Mar 11 – Mar 24 Game Integration & Optimisation	- Integrate features into full game loop - Optimise mechanics and performance - Finalise level design documents - Debug and balance gameplay
Mar 25 – Apr 7 Level Development & Alpha Testing	- Implement logic for Levels 1–4 - Apply feedback from early playtests - Draft User & Software Manuals
Apr 8 – Apr 22 Beta Testing & Final Report	- Conduct beta testing and optimisation - Finalise reports and manuals - Record gameplay demo - Rehearse showcase
Apr 23 – Apr 25 Review & Final Adjustments	- Proofread and finalise documentation - Optimise game performance - Final polish and Q&A prep

Apr 26 – May 2 Submission & Showcase Prep	- Submit all deliverables (code, documents, demo) - Train for live presentation and demo

Story table:

Level	Title	Core Philosophy	Key Mechanics
1	The Knowability of the World- Observer and the Observed	Quantum Observer Effect	Buttons, doors, antiparticles, state-switching mechanisms
2	The Uncertainty of Existence- Is What I Experience the Only Reality?	Heisenberg Uncertainty Principle, Superposition	Unstable paths, probability-based outcomes, collapsing options
3	The Entangled Self- Who Am I?	Quantum Entanglement, Relational Philosophy of the Self	Synchronised switches, mirrored rooms, indirect effects
4	Consciousness and Reality- Does the World Exist Because I Exist?	Quantum Consciousness Theory, Participatory Universe	Memory reversal, past-future causality, black hole domain, ghost (future self)

Part 2: Software Manual

1. Introduction

This project presents an educational puzzle game designed to introduce players to the core ideas of quantum physics through hands-on, interactive gameplay.

Built around foundational concepts like wave-particle duality, interference, superposition, and entanglement, the game weaves these ideas into both its mechanics and its narrative. Rather than simply teaching theory, it encourages players to engage with quantum logic and reflect on how observation might shape identity and reality.

Players assume the role of a quantum particle navigating a layered, maze-like world. Through environmental interaction and the use of quantum abilities, they solve puzzles that gradually reveal a deeper connection to an entangled particle in a mirrored dimension.

The game is developed using Godot 4, with GDScript as the core scripting language. Visuals are hand-drawn in Procreate, animations crafted in After Effects, and selected 3D assets modelled in Blender.

2. Glossary

2.1 Quantum Game Concepts

Term	**Definition**
Superposition	Certain in-game objects (e.g., beams, doors) exist in multiple possible states until observed by the player, causing a state collapse.
Entanglement	A core mechanic where entities (such as antiparticles and doors) are linked such that changing the state of one affects the others.
Observer Effect	The act of observing a player or an object causes a change in its state or alters available paths.
Quantum Interference	The phenomenon where quantum waves overlap and influence each other, impacting puzzle outcomes.
Entangled Particle / Shadow Particle	Representation of the player’s entangled counterpart in another dimension, used to advance levels and embed philosophical themes.
Particle Mode	A player’s discrete physical state, interacting with the environment as a solid object.
Wave Mode	A player's expanded wavefunction state, allowing non-standard interactions like wall-passing.

Entanglement Machine	A device within the game that establishes entanglement between objects.
Reset Machine	A device that resets existing entanglement states.
Certainty Machine	A device that influences quantum states toward deterministic outcomes.
Superposed Door	A door whose open/closed state depends on quantum superposition and observation events.

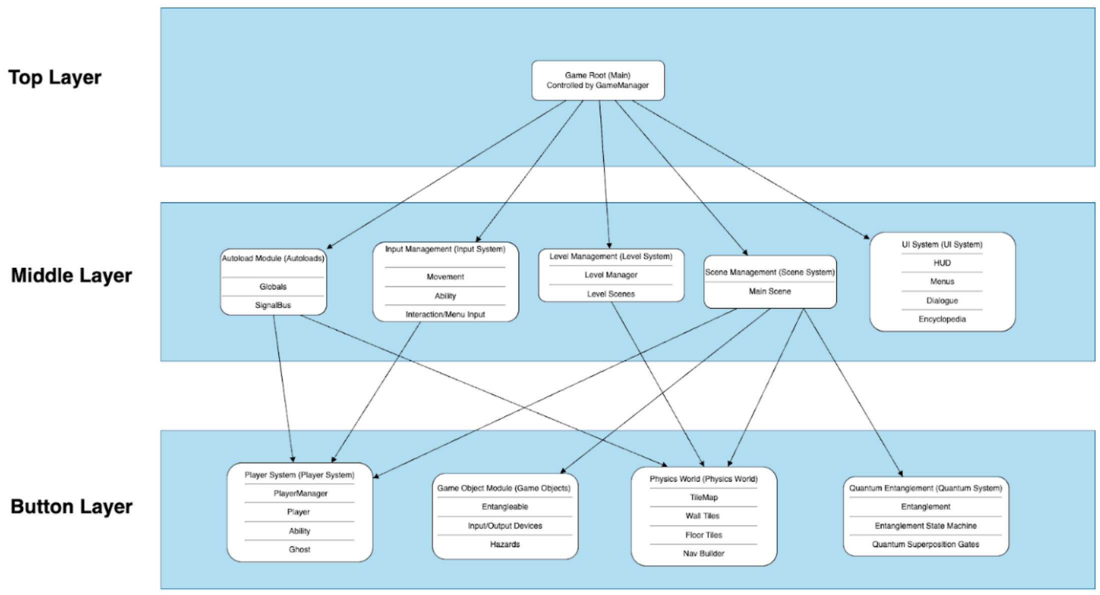
2.2 Godot Engine Concepts

Term	**Definition**
Node	The fundamental building block in Godot; every game object extends from Node.
Scene	A tree of nodes that can be instantiated and reused independently.
SceneTree	The hierarchy of active scenes during gameplay, serving as the main runtime structure.
Autoload	Global singleton scripts used for persistent state and communication across scenes (e.g., Globals, Signals).
Node2D	Base node for 2D scenes, providing position, rotation, and scale attributes.
Control	Base node for UI elements.
CanvasLayer	A rendering layer independent of the main scene, often used for HUDs and menus.
Sprite2D	Node for displaying a single 2D texture.
AnimatedSprite2D	Node for displaying frame-by-frame animations.
TextureRect	UI control node for displaying textures.
VideoStreamPlayer	Node for playing video streams.
CharacterBody2D	Physics node for character control, supporting movement and collisions.
CollisionShape2D	Node that defines the collision boundaries for physics objects.
Area2D	Node for detecting overlaps without providing physics reactions.

TileMap	Node for creating a 2D grid-based map using tilesets.
TileMapLayer	Organises different types of tiles within a TileMap.
Button	Clickable UI button node.
Label	UI control for displaying text.
Rect	Node that draws a coloured rectangle.
HUD	Customized in-game display for user information.
AnimationPlayer	Node for controlling animations on other nodes.
AnimationLibrary	Resource for managing multiple animations.
Tween	Node for smooth interpolation of properties over time.
GameManager	Script managing global game state, level transitions, and save/load systems.
LevelManager	Script handling level-specific states and logic.
PlayerManager	Script managing player instance and related data.
SignalBus	Global signal system for decoupled communication between components.
Globals	Singleton storing global variables and references.
Dash Ability	Player’s ability for quick directional movement.
Entanglement Ability	Player’s ability to create quantum entanglement.
Parry Ability	Player’s defensive counteraction ability.
Shoot Ability	Player’s ability for ranged attacks.
Collision Layer	Defines collision interactions between different types of objects.
Navigation Layer	Used for AI pathfinding layers.
Physics Layer	Defines physics-based interaction rules.
JSON Data	Format used for storing game configuration and player progress.
Save System	System for saving and loading game states.
Recording System	System for recording and replaying player actions (ghost replays).
GUT (Godot Unit Testing)	Unit testing plugin for Godot projects.

OptimiseFloorTiles	Plugin for optimising floor tiles.
QuickEntanglement	Plugin for rapidly creating entangled objects.
GitPlugin	Plugin for integrating Git version control within Godot.

3. High-Level Overview


Top Layer
Middle Layer
Button Layer
Game Root (Main)
Controlled by GameManager
Autoload Module (Autoloads)
Globals
SignalBus
Input Management (Input System)
Movement
Ability
Interaction/Menu Input
Level Management (Level System)
Level Manager
Level Scenes
Scene Management (Scene System)
Main Scene
UI System (UI System)
HUD
Menus
Dialogue
Encyclopedia
Player System (Player System)
PlayerManager
Player
Ability
Ghost
Game Object Module (Game Objects)
Entangleable
Input/Output Devices
Hazards
Physics World (Physics World)
TileMap
Wall Tiles
Floor Tiles
Nav Builder
Quantum Entanglement (Quantum System)
Entanglement
Entanglement State Machine
Quantum Superposition Gates

This project is developed based on the Godot engine, utilizing a node-scene architecture and a modular design approach. The system is divided into several key modules, with communication between modules achieved via signals and global scripts, ensuring a clear and maintainable architecture. The main components of the system include:

Game Control Module

Managed by the 'GameManager' script, this module handles global game state, level transitions, menu operations, and data storage. It communicates with other modules through the Autoloaded 'Globals' and 'Signals' nodes, ensuring consistent state management and event broadcasting.

User Interface Management Module

Built with multiple CanvasLayer nodes, this module displays HUD elements, menus, and pop-up prompts. The HUD system provides real-time feedback on player status, while the menu system includes the main menu, pause screen, and options menu. The interface logic is decoupled from the game logic via signals.

Player System Module

Managed through the 'PlayerManager', this module handles player entities and interactions, supporting particle/wave mode switching, a modular ability system (dash, entanglement, parry, shoot), and a ghost replay feature. The player system directly interacts with the physics engine and the input system to ensure responsive control.

Level Management Module

Controlled by the 'LevelManager', this module manages level initialisation, state tracking, and completion conditions. Each level contains a tile-based map, interactive objects, and entanglement mechanisms, organised as independent scenes.

Quantum Entanglement System Module

Implements the core gameplay mechanics through the 'Entanglement' class and related devices such as entanglement machines, reset machines, certainty machines, and superposed doors. Using a composition design pattern, all entangleable objects are grouped under the 'Entangleable' category for unified management.

Physics and Collision System

Built upon Godot's 2D physics engine, organising interaction layers such as player, walls, floors, enemies, items, and trigger zones. Collision layers and masks define the physical interactions between different types of objects.

Input Management System

Configured through the 'project.godot' file, mapping keyboard and mouse inputs to in-game actions. Inputs include character movement, mode switching, ability activation, interaction, and menu navigation, tightly integrated with the player and UI systems.

4. Architecture & Design

4.1 Architectural Style and Philosophy

To support efficient team collaboration and future scalability, our project adopts a hybrid structure that combines layered architecture with modular design. This setup makes the system easier to maintain and adapt, especially as different subsystems are developed concurrently.

The architecture is divided into three main layers: the UI layer, the game logic layer, and the physics/input layer. Each layer handles a focused set of responsibilities with minimal overlap. At the core is the GameManager, which coordinates subsystems via Godot's signal system and global variables (Globals). Modules like the Player System, Level System, and Entanglement System are designed to function independently, avoiding tight coupling.

Our design is guided by three key principles:

- **Functional Decoupling**: Modules are focused and cohesive, each handling one responsibility to keep the system clean and predictable.
- **Maintainability**: Clear separation ensures that updates or fixes in one module don't ripple into others.
- **Reusability**: Core mechanics such as the Ability System and Interaction System are designed for reuse across levels and scenes, reducing redundant code.

This architectural approach gives us both structure and flexibility—ideal for iterative development, feature expansion, and smooth teamwork.

4.2 Logical and Physical Architecture

4.2.1 Logical Architecture

The logical architecture has been detailed in the High-Level Overview section.

4.2.2 Physical Architecture

The project directory is organised by function, as follows:

```
/godot-project
├── scenes/           # Game scenes
│   ├── main.tscn       # Main entry scene
│   ├── gui/           # UI scenes
│   ├── player/         # Player scenes
│   ├── levels/         # Level scenes
│   ├── objects/        # Interactable objects
│   ├── entangleable/    # Entangleable entities
│   └── inputs/outputs/  # Input/Output devices
├── scripts/          # Scripts corresponding to scenes
│   ├── autoload/       # Autoload global scripts
│   ├── player/         # Player logic scripts
│   ├── logic/          # Core game logic scripts
│   ├── gui/           # UI control scripts
│   └── objects/        # Object scripts
├── assets/           # Game assets (images/audio/videos)
├── data/             # Configuration and save files
└── addons/           # Plugins and third-party extensions
```

4.3 Design Patterns & Conventions

During the development of Quantum Maze, we applied multiple design patterns to address specific problems and improve code quality. The main patterns used are as follows:

4.3.1. Singleton / Autoload Pattern

Through Godot's Autoload mechanism, we achieved global state management and cross-scene access. Main singletons include:

- **Globals**: Managing global game states, references, and configurations such as player energy and camera control.
- **Signals**: Implementing a global event bus for cross-module communication.

4.3.2. Observer Pattern

Implemented using Godot's signal system to decouple UI from game logic:

- **SignalBus**: Centralised management of custom signals like entanglement creation and navigation updates.

- **UI response**: HUD updates its display by listening to player status signals without direct dependency on the player object.

4.3.3. Component-Based Ability System

We implemented player abilities through composition rather than inheritance:
• Basic ability classes: Each ability (Dash, Entanglement, Parry, Shoot) is an independent component.
• Player composition: The Player class combines these ability components rather than inheriting them.
• Flexibility: Abilities can be dynamically enabled/disabled, supporting level design and gameplay progression.

4.3.4. Factory / Scene Instancing

Using Godot's scene system and PackedScene as factories for object creation:

- Ghost creation: Instantiate ghost replays dynamically.
- Level loading: Dynamically load levels via the GameManager.
- Dynamic objects: Objects like photons and entanglement connections are created via scene instancing.

4.3.5. State Pattern

Used for managing different states of players and entangled objects:

- Mode management: Switch between particle and wave modes using enumerations.
- State execution: Different behaviours and physical properties based on the current state.

4.3.6. Command Pattern

Applied in the replay system:

- Action recording: Serialize actions into commands.
- Action replay: Replay behaviours by parsing recorded commands.

By applying these design patterns, we achieved high cohesion and low coupling within the system, improving maintainability and scalability, while maintaining clear communication and collaboration between system components.

4.4 Typical Runtime Flow

The typical runtime flow of the game is as follows:

1. Initialisation Phase: Load Autoload singletons → Load the main scene →load the story introduction →GameManager initialises game states and UI.

2. Menu Phase: Display the main menu → Player selects a level → Load the selected level scene.

3. Level Loading: Load the map and objects → Initialise the player and HUD → Display level guides.

4. Gameplay Phase: Capture player inputs → Player system processes abilities → Physics and interactions are updated → UI updates in real time.

5. State Events: Events like level completion, death, or pause trigger responses via signals.

6. Level Completion/Switching: Save progress → Unload the current scene → Load a new level or return to the menu.

4.5 Class Structure Overview

The codebase is organised into multiple subsystems, each encapsulating a specific functional domain within the game. These subsystems are designed with modularity and clarity in mind, allowing for maintainable growth and cross-functional development.

To provide a clearer view of how key systems are structured internally, we include representative class diagrams for selected subsystems where complexity and modular design are especially evident. These diagrams serve as reference points for understanding inheritance relationships, composition structures, and the organisation of critical gameplay logic.

4.5.1 Core Management System

This system handles game-wide state management, level transitions, and global signal dispatching.

- GameManager (Node): Orchestrates overall game flow, including transitions and pause states.
- LevelManager (Node): Coordinates per-level lifecycle and success conditions.
- PlayerManager (Node2D): Creates and manages player instances, modes, and respawns.
- Globals, SignalBus (Node): Handle global constants and decoupled signal-based communication.

4.5.2 Player System

This subsystem defines the core mechanics of the controllable character, including dual-mode behaviour and modular quantum abilities.

- Player (CharacterBody2D): The main playable entity, managing state transitions and interactions.
- Ghost (extends Player): Replays player actions for narrative and gameplay purposes.
- Ability Components (DashAbility, ShootAbility, etc.): Modular abilities attached to the player via composition.

The following diagram illustrates the internal structure of the **Player System**, emphasising component composition and state management logic.

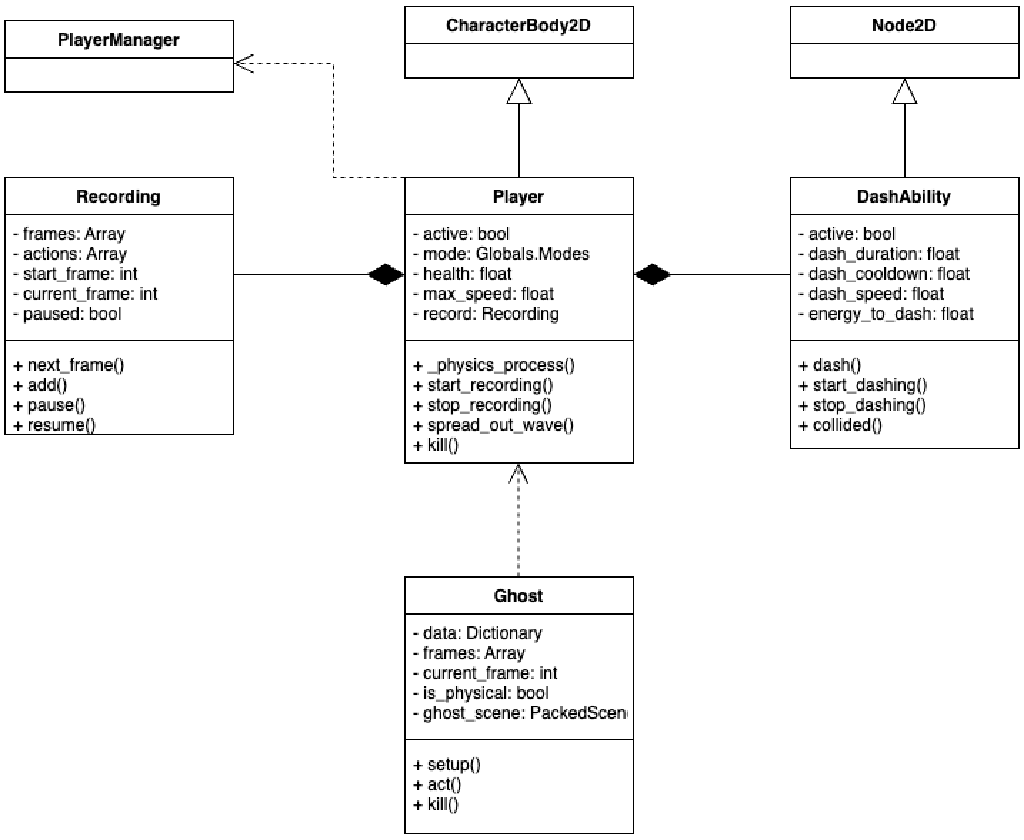


Figure 4.4.1 – UML Class Diagram: Player System

4.5.3 GUI Subsystem

The GUI subsystem contains user interface components for displaying information, handling menu navigation, and supporting interaction with in-game data.

- HUD, LevelCard, Encyclopaedia, Minimap: Each implements a specific UI function, built on top of Godot's Control node hierarchy.

4.5.4 Game Object System

This system manages interactive objects, collectible items, and environmental hazards that form the core of the game's level design.

- **Interactables**: Prism, EnergyLamp
- **Collectables**: Collectable, EnergyItem, AbilityItem
- **Hazards**: Enemy, Spotlight, Photon

The following diagram presents the object class hierarchy and functional segmentation within the Game Object System.

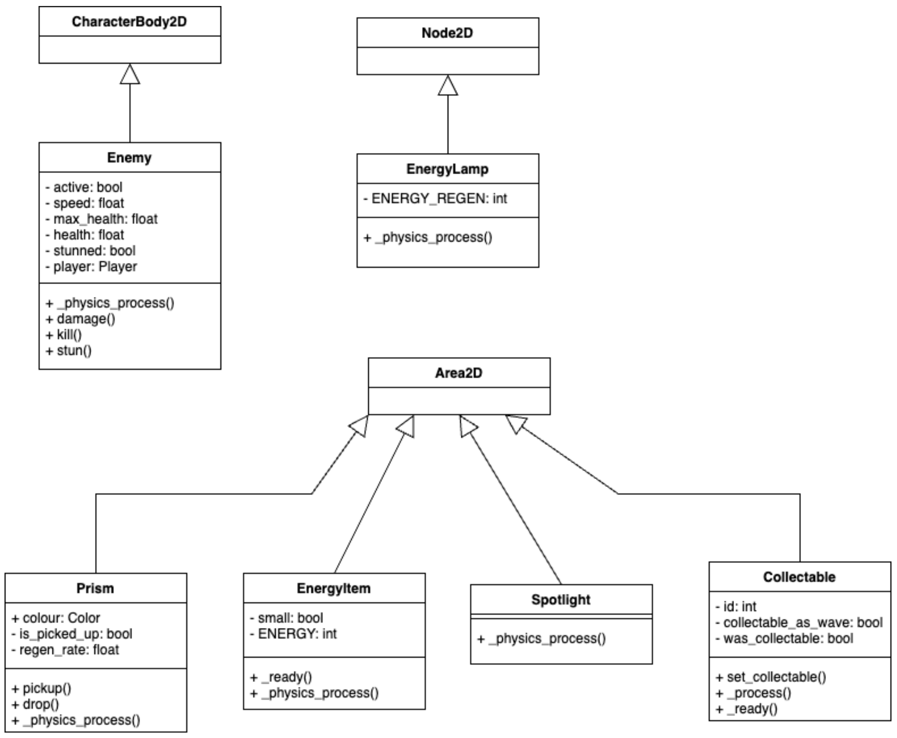


Figure 4.4.2 – UML Class Diagram: Game Object System

4.5.5 Electrical Logic System

This subsystem simulates logical connections between inputs and outputs, allowing players to solve puzzles through interactive components.

- Inputs: Button, Switch, SolarPanel
- Outputs: Door

Each component reacts to environmental triggers or signals and modifies the game state accordingly.

4.5.6 Quantum Entanglement Subsystem

Inspired by real-world quantum theory, this system introduces mechanics such as entanglement, superposition, and state collapse.

- EntanglementMachine, SuperposedDoor, CertaintyMachine, ResetMachine

These classes allow the player to affect distant objects, solve puzzles indirectly, or create deterministic outcomes from probabilistic systems.

5. Core Quantum-Based Mechanisms

The game implements several core gameplay elements inspired by quantum mechanics. Each mechanic is handled by dedicated components and follows clean separation of concerns. Below, we outline the major quantum concepts integrated into the gameplay, along with high-level pseudocode illustrating key logic.

5.1 Wave-Particle Duality

In gameplay, the player can switch between a particle mode and a wave mode. This is managed by the PlayerManager and individual Player instances, affecting collision behaviour, abilities, and interactions.

Pseudocode:

```
IF current mode is PARTICLE THEN
  Switch to WAVE mode
  Disable collision
  Start wave expansion
ELSE
  Switch to PARTICLE mode
  Enable collision
  Collapse wave function
END IF
```

5.2 Quantum Superposition & Collapse

When in wave mode, the player can create multiple possibilities. PlayerManager tracks these possibilities and handles wavefunction collapse triggered by observation or interaction.

Pseudocode:

```
WHEN creating a possibility:
  IF enough energy available THEN
    Create a new wave instance at a given position
    Deduct energy cost
  END IF
WHEN observation occurs:
  Select one instance randomly or based on conditions
  Destroy all other instances
  Collapse chosen instance into particle mode
```

5.3 Observer Effects

Observation mechanisms such as spotlights and probability detectors trigger state changes in wave instances.

Pseudocode:

```
FOR EACH entity within observation area:
  IF entity is in WAVE mode THEN
    Trigger collapse
    Convert to PARTICLE mode
  END IF
END FOR
```

Each mechanism abstracts quantum behaviours into playable systems. State management, physics interactions, and quantum events are separated cleanly into dedicated components. The PlayerManager coordinates these systems to maintain consistency and support extendibility across gameplay features.

6. Data Storage & Resources

6.1 Data Storage Formats and Management

In Quantum Maze, we employ multiple data storage formats to ensure efficient management, maintainability, and readability of game data.

6.1.1 JSON Data Storage

JSON files are primarily used to store game configuration and player save data:

- Game Configuration Data (game_data.json): Stores basic information for all levels, including level names, unlock statuses, ability configurations, collectable counts, and progression sequences.
- Player Save Data (player_save.json): Records player progress, completed levels, best times, and collectable statuses, supporting persistent storage.
- Ghost Replay Data (temp_ghost_data.txt): Stores serialized dictionaries encoded in Base64, recording player action sequences for ghost playback, refreshed every game session.

6.1.2 Resource File Management

In addition to JSON data, the following resource types are managed:

- TileSet Resources (.tres): Wall and floor tile sets, including collision shapes, navigation properties, and custom metadata.
- Scene Files (.tscn): Prefab objects stored as independent scenes for instancing during gameplay, such as buttons, doors, and accelerators.
- Script Files (.gd): GDScript files organised by functional modules, supporting code reuse through inheritance and composition.

6.2 External Resource Management

Quantum Maze incorporates a variety of external resources to enhance visual and audio experiences, all carefully selected and optimised.

6.2.1 Visual Resources

- Particle Effects: Custom textures for effects such as dash trails, photon shots, and wave mode visualisations, powered by GPUParticles2D.
- UI Elements: Custom buttons, icons, HUD components, game logos, and special effect assets.
- Fonts: Pixel-style fonts for the main interface and system fonts for information displays, with varied styles across different scenes.

6.2.2 Audio Resources

- Sound Effects: Interaction sounds (buttons, doors, collisions), ability activation sounds (dash, shoot, entanglement), and environmental sounds (machines, electricity).
- Background Music: Dedicated tracks for the main menu, individual levels, and special events such as death and victory.

6.2.3 Video Resources

- Cutscene Animations: Opening logo, menu background videos, level transition scenes, and ending animations, all encoded in .ogv format.

6.3 Third-Party Plugins

Several essential Godot plugins were integrated to improve development efficiency and expand game functionality:

- GUT (Godot Unit Test): Unit testing framework supporting test-driven development and ensuring logic stability.
- OptimiseFloorTiles: Custom plugin designed to optimise TileMap rendering performance by reducing draw calls.
- QuickEntanglement: Custom editor tool that simplifies the creation and editing of quantum entanglement links.
- GitPlugin: Enables direct version control operations within the Godot editor, facilitating smoother team collaboration.

6.4 Resource Import and Optimisation Settings

To balance performance and visual quality, we applied specific optimisation settings within Godot's import system:

Texture Settings

- Pixel Art Optimisation: Disabled texture filtering (set to 'nearest neighbour') to preserve pixel clarity without blurring.
- UI Texture Optimisation: Applied VRAM compression to speed up UI loading and tailored compression formats for different platforms.

TileSet Generation

- Physical Property Settings: Walls and floors are configured with specific collision layers and navigation attributes.
- Custom Metadata: Additional interaction properties such as 'breakable', 'untunnelable', and 'bottomless' are embedded in tile data.

Audio Settings

- Sound Effect Optimisation: Used Ogg Vorbis format to balance quality and file size; important sounds are preloaded.
- Music Optimisation: Implemented streaming to reduce memory footprint and configured seamless looping for background tracks.

Video Settings

- Cutscene Optimisation: All videos encoded with Theora (.ogv) for balance between visual quality and loading performance, with resolutions optimised for various screen sizes.
- Resource Management: Implemented a custom VideoResourceManager singleton that:
    - Dynamically loads video resources only when needed rather than preloading all videos at startup
    - Explicitly releases large video resources (especially the 234MB death animation) after use
    - Intelligently caches frequently used small videos while aggressively cleaning up larger ones
    - Provides centralised control over video playback, reducing memory leaks and CPU overhead
    - Separates resource management from scene logic, improving maintainability and performance
- Cross-Platform Considerations: Added platform-specific optimisations like deferred loading on mobile/lower-end devices while maintaining immediate loading on desktop platforms.

7. Coding Conventions & Maintainability

7.1 Naming Conventions (GDScript)

In this project, we followed a consistent set of naming conventions to improve code readability and maintainability.

- Class names use PascalCase (e.g., Player, LevelManager), while file names and function names follow snake_case (e.g., player.gd, create_instance()). We ensured that each class name matches its corresponding file name. Constants are written in ALL_CAPS with underscores (e.g., MAX_SPEED), and enum values use PascalCase (e.g., Modes.PARTICLE).
- For variables, member variables use snake_case, and internal-use variables have an underscore prefix (e.g., _registry). We also made full use of GDScript's type annotations to enhance type safety and clarity. For example:

```
@export var max_speed: float = 500
func is_valid(start: Vector2, end: Vector2) -> bool:
```

7.2 Commenting Style and Script Granularity

Throughout development, we maintain consistent commenting practices to help other team members understand the logic and ensure long-term maintainability. We include:

- Class comments: brief summaries at the top of each class
- Variable documentation: using Godot-style `##` comments to describe key variables
- Function documentation: describing purpose, parameters, and return values for complex functions
- Inline comments: short explanations for complex logic or edge cases
- TODO notes: clearly marked sections for future improvements or deferred logic

We also followed the single-responsibility principle, where each script focuses on one specific function or system. For example, player.gd manages basic player behaviour, while dash_ability.gd is dedicated to dash mechanics. Most scripts are kept under 200 lines for clarity, even in larger files.

7.3 Script Management Practices

To avoid common scripting chaos in Godot projects, we followed several key practices:

- Dependency Management: Use Autoload singletons like Globals and Signals to handle global state and events. Use @onready and % for node access to avoid hardcoded paths.
- Logical Organisation: Separate lifecycle methods (e.g., _ready, _process) from custom logic. Group functions by feature and isolate logic by state.
- Signal-Based Decoupling: Use signals for node-to-node communication. UI updates are triggered by state-change signals rather than direct control.
- Component Design: Each player ability is an independent component. We favour composition over deep inheritance to build complex behaviours.
- Consistent Practices: Follow naming rules, replace magic numbers with meaningful constants, and apply type annotations where possible.
- Structured Error Handling: Use defensive programming practices, log important warnings/errors with push_warning/push_error, and provide fallback behaviours when assets fail to load.

8. Testing Approaches

8.1 Testing Framework and Tools

Our project adopts the GUT (Godot Unit Test) framework as the core testing system. GUT is specifically designed for the Godot engine, providing a rich set of assertion methods, test collection and organisation tools, and support for mocking, signal watching, and verification.

8.2 Test Types and Coverage

Our testing strategy covers multiple layers to ensure both component reliability and overall gameplay quality.

8.2.1 Unit Testing

Unit testing is the core of our testing setup. All tests are organised under the tests/unit directory, grouped by functional domain:

- tests/unit/player/: Tests for player abilities and behaviour
- tests/unit/objects/: Tests for interactive scene objects
- tests/unit/hazards/: Tests for hazardous element interactions

These tests verify player abilities like dash and parry, object interactions (prisms, buttons, particle accelerators), physics behaviours, and the reliability of core game mechanics.

8.2.2 Integration Testing

While integration tests are not separated into a distinct directory, many unit tests include cross-component logic such as player interactions with objects, signal propagation between connected elements, and state synchronisation across systems.

8.2.3 Alpha Testing

We conducted multiple rounds of internal alpha testing during development. Key focuses included:

- Gameplay flow and overall player experience
- UI functionality and intuitive control mapping
- Logical progression of level difficulty
- Performance and stability across platforms

8.3 Running Tests

We provided three different ways for running tests:

8.3.1 Using the GUT Panel in the Godot Editor

1. Enable the GUT plugin in Project Settings.
2. Open the GUT panel at the bottom of the editor.
3. Select the directories or specific tests to run.
4. Click 'Run' to execute tests.

8.3.2 Running the GutRunner Scene

1. Open addons/gut/gui/GutRunner.tscn.
2. Configure the test directory paths if needed.
3. Run the scene to execute all tests.

8.3.3 Command Line Testing

Tests can also be run through the Godot command line interface, useful for continuous integration setups. For example:

```
godot --script=res://addons/gut/gut_cmdln.gd -gdir=res://tests
```

9. Installation/Build Manual

Download the appropriate file for your operating system from the **releases** directory in our Git repository, which also includes the full Godot project.

- **Windows**:
  Download and run the .exe file (bundled with all game code and assets).
  Requires 64-bit (x86_64) architecture.
- **Linux**:
  Download and run the .x86_64 executable.
  Requires 64-bit (x86_64) architecture.

- **macOS**:
  Download and run the .app bundle.
  Requires a 64-bit machine.
  - Minimum version (x86_64): macOS 10.12
  - Minimum version (arm64): macOS 11.00

10. Additional Documentation & References

Git link: https://projects.cs.nott.ac.uk/comp2002/2024-2025/team12_project

.

Part 3: User Manual

Welcome!

Welcome to the immense and dangerous Quantum Maze, where nothing never makes sense!

You are a single atom exploring an apparently infinite maze, searching for… something. Maybe treasure at the centre, maybe an escape, maybe something more. There's only one way to find out, however, to understand the quantum mechanisms in this world to solve the puzzles and defeat the enemies blocking your way.

Good luck Quantum Explorer!

Tutorial and Help

Stand on **tutorial triggers** to learn how to interact with the world. These are the main way to understand the game and should teach you everything you'll need to know, from controls to mechanics.

(A mysterious yet oddly familiar **entity** may guide you as well sometimes…)

Also, the **Encyclopaedia** can be accessed at any time by pressing the button below the mini map, giving detail on all gameplay elements.

Controls

Basic Controls

- **Movement**: W (up) A (left) S (down) D (right) or Arrow Keys
- **Toggle mode**: Space
- **Cycle between possibilities**: Mouse Wheel up and down
- **Create a new possibility**: E (in Wave mode)
- **Pick up/Drop Energy Prisms**: E (in Particle mode)
- **Pause/Unpause**: P or Escape
- **Navigate Dialogue**: Q

Abilities

- **Shoot photons**: Left Mouse Button (in Particle mode)
- **Energy shield**: Right Mouse Button (in Particle mode)
- **Dash**: Left or Right Shift (in Particle mode)
- **Create entanglement**: Left Mouse Button (in Wave mode)

The HUD

Energy Bar

The large bar at the top left of the screen shows the player's remaining energy.

Energy is consumed when attacking or duplicating and can be replenished by standing in light.

You won't die if you run out of energy, but you'll be slow and unprotected, so be careful!

Mini map

This helpful display lets you see further than normal, vital as you explore the endless maze ahead.

Encyclopedia Button

Press the button just below the mini map to open the **Encyclopaedia**. This contains information on everything in the game, explaining how it works if you get stuck or want to learn about the real-life physics behind the quantum world!

Wave Mode Timer

When in wave mode you'll notice the ominous timer counting down. This shows how spread out in space you are, that is, how likely you are to be found anywhere when you collapse back into a particle. So don't let the timer reach zero, or you won't ever be found and its game over!

Saving

The game **autosaves** to a single save file (a .json file) every time a level is completed.

This file keeps track of which levels have been unlocked, which collectables have been found, and the best times levels have been completed in.

No personal data is stored, and no account/internet connection is required.

Requirements and Installation

Minimum System Requirements

- 0.5 GB Storage space (for game file(s) and player save data)
- 4 GB RAM
- 4 GB GPU

Required Hardware

Our game requires a keyboard and mouse. Trackpads and controllers are not supported.

Installing and Playing the game

Download the correct file for your operating system from the **releases** directory in our git repository, which also contains the entire Godot project.

Windows

Download and run the single .exe file containing all the game's code and assets.

Requires x86_64 architecture (64-bit machine)

Linux

Download and run the single .86_64 file containing all the game's code and assets.

Requires x86_64 architecture (64-bit machine)

MacOS

Download and run the .app bundle.

Requires a 64-bit machine

Minimum macOS Version x86_64: 10.12

Minimum macOS Version arm64: 11.00