December 2025

Semester Thesis

Educational Multiplayer Game for the Planetarium

Exploring the World of the Ants in an Interactive Game

Leo Oetterli
leo.oetterli@ost.ch

Simon Ott
simon.ott@ost.ch

Advisor
Prof. Dr. Andreas Peter
andreas.peter@ost.ch

Computer Science

Eastern Switzerland University of Applied Sciences

1. Abstract

Planetariums are traditionally used for the presentation of non-interactive fulldome films, positioning visitors as passive spectators. This project investigates how a planetarium dome can be used as a shared interactive medium by implementing a real-time multiplayer game tailored for large audiences.

The project was developed in collaboration with the Swiss Museum of Transport (Verkehrshaus Luzern).

The work presents a distributed system that enables visitors to participate using their personal smartphones as web-based controllers. The game is implemented in Unreal Engine and rendered onto a 360° dome projection. A custom backend coordinates real-time communication between the game, multiple concurrent controller clients, and an administrative interface used by the show host, which allows game control and monitoring during live events.

To demonstrate the approach, an educational multiplayer game centered around ant colony behavior was created. Gamification principles are applied to convey knowledge implicitly through game mechanics, making the experience accessible to a broad audience. Although designed for a planetarium context, the underlying interaction model and system architecture are intended to be transferable to other large-audience venues such as cinemas or event spaces.

The result of this work is a functional prototype consisting of the multiplayer game, a backend infrastructure, and platform-independent web-based controller and administration interfaces. The prototype demonstrates the technical feasibility of large-audience interaction in a fulldome environment and provides a solid foundation for a subsequent bachelor thesis, in which the framework will be generalized and the educational gameplay further expanded.

2. Management Summary

In the project "Educational Multiplayer Game for the Planetarium", an interactive group experience for a 360° fulldome environment such as the Swiss Museum of Transport planetarium in Lucerne was developed. While planetariums traditionally focus on film-based content, this project explores how visitors can actively participate in a game using their personal smartphones as controllers. The goal was to design and implement a scalable, low-latency multiplayer game that works reliably with around 20 simultaneous users and uses the fulldome projection of the planetarium.

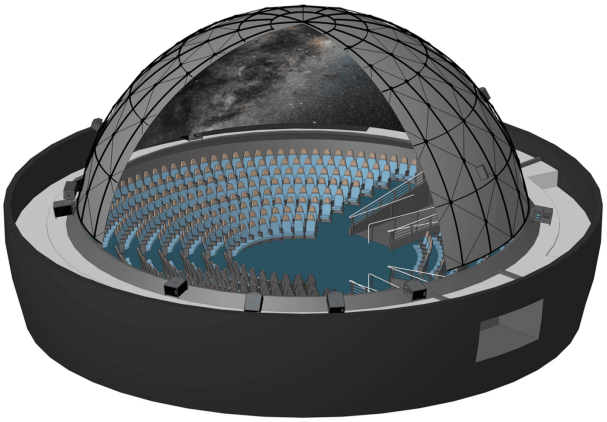
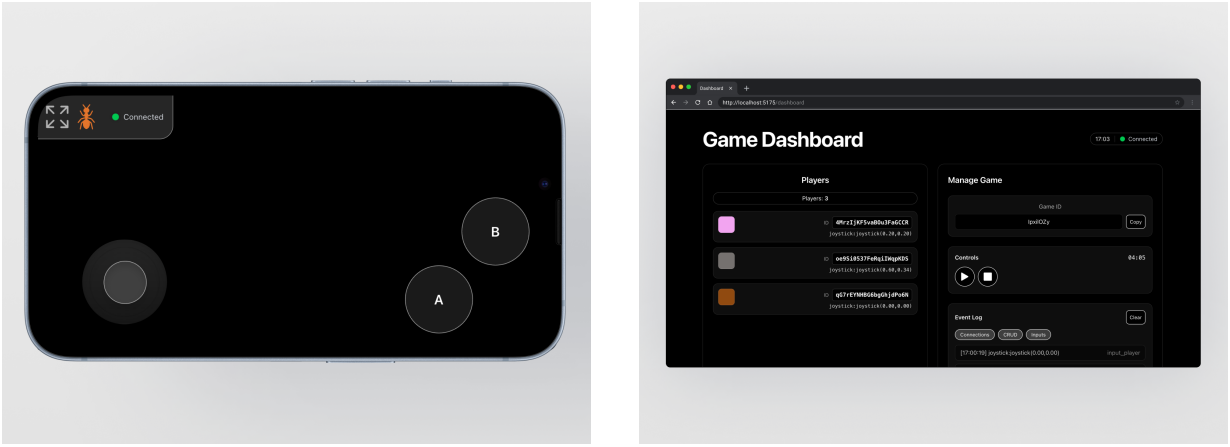


Figure 1.: Visualization of a 360° Fulldome Projection (source: [1]).

A modular system architecture was developed consisting of four components:

- A custom backend, implemented with NestJS and Socket.IO, responsible for session management, routing of player input, and admin monitoring.
- A web controller for phones, built with React, Redux, and Tailwind CSS, which allows players to join instantly by scanning a QR code.
- A web-based dashboard for presenters, built with the same technology stack as the controller. It provides real-time monitoring of all connected players and allows administrators to start the gameplay and oversee the session.
- A 3D multiplayer game built with Unreal Engine and tailored to run on the planetarium's dome mapping.

The game prototype explores the world of ants and their characteristic behaviors. From a didactic standpoint, the project leverages gamification principles to convey scientific concepts in an engaging and accessible manner. Key technical challenges included designing a communication protocol, using Unreal with C++, integrating the various system modules and ensuring stable reconnection handling the setup for a distributed dome-projection environment. Two user tests with groups were conducted to validate the system's scalability, robustness, and overall usability.



(a) Web Controller (b) Dashboard

Figure 2.: Gameplay interfaces

This work establishes the foundation for the upcoming bachelor thesis, which aims to develop more advanced game mechanics embedded in an educational narrative. Additionally, modular system components will be generalized to support reuse in other multiplayer experiences within the planetarium context or similar large-audience applications.

Figure 3.: Gameplay on Fulldome

3. Acknowledgment

We would like to express our sincere gratitude to everyone who supported us throughout the development of this semester thesis.

Our biggest thanks go to Prof. Dr. Andreas Peter, whose guidance, encouragement, and openness to new ideas enabled this project to take shape. His expertise and constructive feedback helped us refine our concepts and turn them into a functioning prototype.

We would also like to warmly thank Marc Horat from the Swiss Museum of Transport. He conceived the initial idea of creating a multiplayer game for the dome environment and supported us with feedback and on-site testing opportunities. We are grateful for the chance to develop an interactive experience for such a unique venue.

A special thank you goes to all participants who took part in our user tests. Their time, observations and feedback were essential in validating and improving the system.

Contents

4. Introduction

4.1. Project Background

The initial motivation of the team was to develop a project that combines programming and three-dimensional computer graphics within a scientific visualization context. In February 2024, the project team contacted Marc Horat, the leader of the planetarium of the Swiss Museum of Transport, to ask if he sees any opportunities for collaborating on a semester thesis.

During the initial meeting, Marc Horat proposed several possible project directions. These included data-driven visualizations for the planetarium dome, generative visuals for live concert performances, real-time audio translation during shows, and the development of an interactive multiplayer experience with mobile phones as controllers. The collaboration was intentionally left open-ended, giving the team freedom to choose a preferred concept that would suit their technical interests.

From a technical perspective, it was restricted to two approaches to render content in the dome. An option was a visualization tool developed by the planetarium, and the alternative was to use Unreal Engine, supported through a dedicated plugin.

4.2. Project Scope

From the range of proposed project directions, we chose to develop an interactive multiplayer game for a planetarium dome, defining the following core objectives:

- Investigate how an interactive multiplayer experience for a planetarium dome can be conceptually designed using the participants' personal mobile devices as input controllers.
- Develop a game concept based on a gamification approach aimed at conveying scientific content in an accessible, age-independent, and engaging manner, and implement a proof-of-concept prototype based on this concept.

4.3. Methodology and Roadmap

The project followed an iterative, prototype-driven development approach. This methodology was chosen due to the use of unfamiliar technologies such as Unreal Engine and dome projection.

Instead of defining a fixed solution upfront, requirements and design decisions were refined incrementally based on technical feasibility, testing results, and user feedback. The work alternated between analysis, design, implementation, and evaluation, allowing early validation of assumptions and timely adjustments.

4.3.1. Project Roadmap

The project was structured into three main phases:

- **Planning and Concept Development:** Definition of the project scope, analysis of existing solutions, selection of technologies, and initial system and game concept design.
- **Implementation and Prototyping:** Incremental development of the backend, web-based controller, administration interface, and Unreal Engine game with user testing at the end of the prototype cycle.
- **Stabilization and Documentation:** Bug fixing, evaluation of results, and documentation of the project.

4.4. Purpose of Documentation

The purpose of this report is to document the methodological approach and the technical decision-making process applied throughout the project. Its goal is to ensure the development process is transparent by showing what requirements were identified, how design decisions were made, and how the resulting artifacts were assessed against the defined objectives.

This document is intended for technically experienced readers who were not directly involved in the project. It is written to enable independent understanding and critical review of the work, without requiring prior knowledge of the project's execution or context.

4.5. Thesis Composition

Product Evaluation
The initial chapters establish the problem context by conducting a market analysis of what other solutions exist in this space, followed by a description of the game concept that was developed during this thesis.

Product Description
The evaluation is followed by the description of the product where requirements and domain context are established.

Product Realization
This is followed by sections detailing the system design, implementation, and applied quality measures.

Conclusion
Conclusion evaluates the results, reflects on limitations, and outlines potential directions for future work.

Appendix
Supplementary materials like personal report, project plan, time tracking report and risk assessment are provided in the appendix.

Part I.

Product Evaluation

1. Market Analysis

This chapter examines existing interactive systems and products used in planetarium and fulldome environments to contextualize the presented project. The analysis serves to identify comparable concepts, assess the degree of innovation, and potential of the project relative to current solutions.

1.1. Existing Products

Kinetarium (Multiplayer Planetarium)

Kinetarium [2] focuses on interactive multiplayer experiences in a planetarium dome. Smartphones are used as controllers, and the system is designed for large audiences. It provides dome-based games or shows and emphasizes a low barrier of entry for visitors.

The concept is closely related to the core interaction idea of this project, combining large audiences, dome projection, and smartphone-based web controllers. Gamification is also employed to deliver information to a wide range of users.

This work mainly differs from Kinetarium in terms of the game itself and the technologies employed. In both visual design and gameplay mechanics, their games differ significantly.

Cosmic Orbiters (Multiplayer Fulldome Game)

Cosmic Orbiters [3] is a multiplayer fulldome game in which each participant uses a smartphone as an input device. The system is designed for planetarium environments and reports support for up to 24 simultaneous players.

Although it follows a comparable concept, it does not appear to scale particularly well and is focused on a space theme. As with the Kinetarium, both the technology and the game itself differ from this product.

Planetarium platform vendors with interactive features

Several established planetarium system vendors include interactive capabilities as part of their show-control platforms:

- Sky-Skan DigitalSky [4] provides planetariums, with an app to explore the space interactively.
- The Swiss Museum of Transport has its own developed applications to present the space, which can be interactively controlled by the show presenter.

These systems represent the dominant professional planetarium platforms, where interactivity is typically implemented as a feature of a show-control ecosystem rather than as a multiplayer game framework.

1.2. Summary

With Kinetarium and Cosmic Orbiters, two existing products were identified that are conceptually very close to the presented project, as both focus on large-scale multiplayer interaction in planetarium dome environments using smartphones as input devices. Kinetarium was known from the early stages of the project and was actively reviewed for inspiration as well as to avoid developing an overly similar experience. Even with the existence of similar solutions, the market remains extremely limited because planetariums are relatively uncommon. However, this specialization also underscores the potential to expand into other public spaces—such as movie theaters, exhibitions, or large live events—where comparable interactive, audience-driven experiences could be implemented.

2. Game Concept

This chapter describes the conceptual game design developed during the project. Due to the scope of the project, the plot was not implemented in the prototype, but its development was an integral part of the conceptual work and serves as a foundation for future iterations.

2.1. Concept Development

The concept of the game was developed through an iterative collaborative ideation process. Several in-person sessions were held in which both team members generated ideas using Post-it notes. During the initial phase of each iteration, ideas were created without constraints on feasibility, scope, or technical limitations to explore a wide design space. The ideas collected were then thematically grouped and evaluated using simple voting. This method was applied for all three ideation rounds and proved to be very helpful.

Initial inspiration was influenced by suggestions from the project partner, Marc Horat, including ideas such as data visualization for dome projection, generative visuals for concerts, live audio translation, and an interactive multiplayer game using smartphones as controllers. In the first iteration, the decision was made to focus on the concept of multiplayer games, as it best used the planetarium dome as a shared interactive medium.

The next iterations focused on defining the theme of the game and the core mechanics in parallel, as both aspects are closely interrelated. Several criteria were developed during the process and guided this process. As a multiplayer experience, the game was designed around cooperative rather than competitive mechanics. Furthermore, the theme needed to naturally support group-based interaction, leading to a focus on collective systems such as animals living in groups. Space-related themes were deliberately avoided, as they are already prevalent in planetarium contexts.

Based on these considerations, the ants were selected as the central theme of the game. Although not an obvious choice, the theme quickly proved convincing, as ant colonies inherently combine cooperation and communication while still incorporating the objective of traffic, but in a more abstract way.

2.2. Educational Approach

The educational concept of the game is based on a gamification approach, where scientific information and interesting facts are embedded in the plot and gameplay mechanics. Players learn through interaction: ant behaviors such as pheromone trails, cooperation, or collective defense are experienced directly by playing, rather than explained in an abstract manner.

Short interludes are planned between the levels. In these segments, information and real film footage about ant behavior are shown, which also appears in the game as mechanics that the player can experience.

2.3. Plot

As part of the long-term planning and further development, a preliminary narrative structure was developed to connect the individual game mechanics into a coherent experience.

The story is structured into four levels:

Level 1: Pheromone Trails The game begins in a dark environment resembling a “fog of war”. Players explore the world by moving through it, gradually illuminating the areas they visit. This mechanic introduces the concept of pheromone trails used by ants for orientation and communication. From a gameplay perspective, it also serves as an introduction that helps players become familiar with the controller.

Level 2: Food for the Colony In the second level, the ant colony faces an increased demand for food. Players must locate and transport food and larger objects require multiple players to cooperate and carry them together, emphasizing teamwork.

Level 3: Colony Under Attack The third level introduces a threat in the form of attacking beetles. Individual ants are too weak to defend themselves alone, making cooperation essential.

Level 4: Recovery and Stability After the conflict, the colony returns to a state of balance. The final level focuses on a calmer atmosphere, accompanied by visually pleasing elements such as a sunset.

2.4. Sound

In the scope of this study project, only atmospheric forest sounds were implemented [5].

2.5. Visual Research

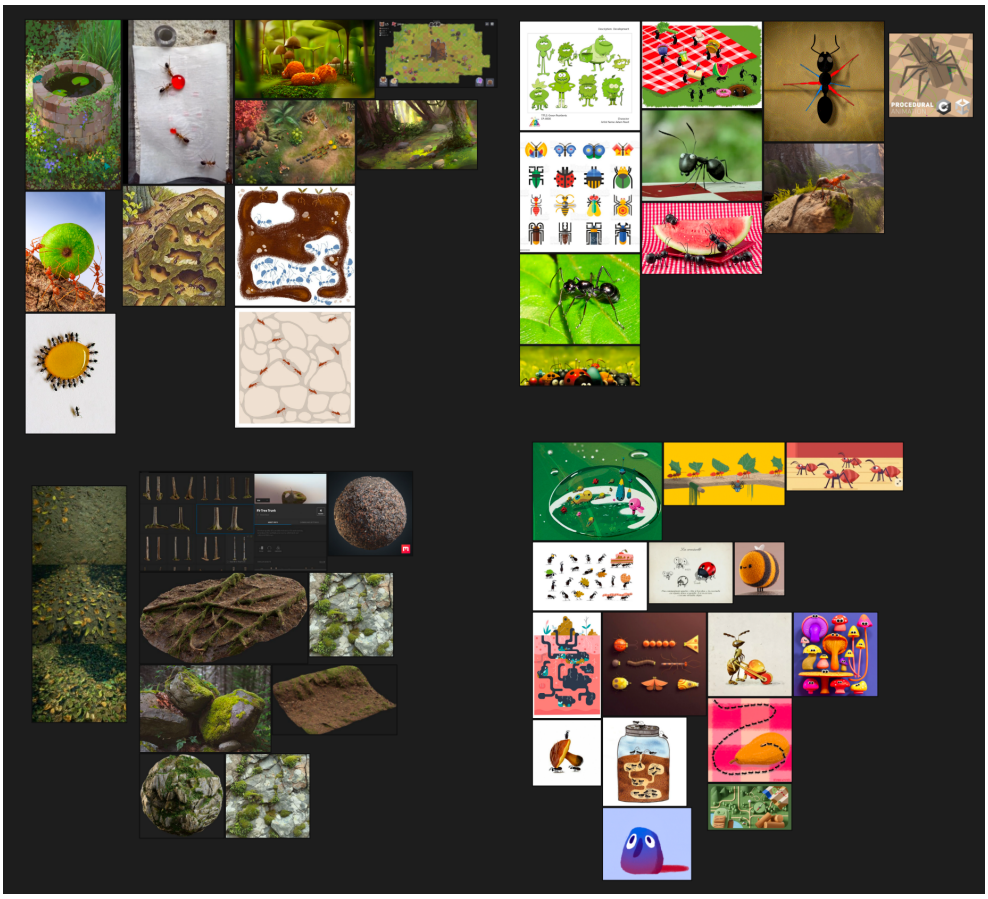


Figure 4.: Visual reference material exploring different representations of ants, environments, and stylistic approaches used during early concept development (various sources).

The images from Figure 4 originate from publicly available online sources and are not the author's own work. They are included exclusively as visual inspiration to communicate the conceptual design space explored during the project.

As part of the early concept development, visual research was conducted to explore visual styles for environments and characters. The images collected served as inspirational reference material to support ideation and discussion rather than as objects of direct analysis.

Although initially a more stylized visual approach was planned, some research led to the conclusion that a realistic style would be significantly more time-efficient. The choice was mainly driven by the use of Quixel Megascans, an asset library that offers high-quality models that are seamlessly integrated into Unreal Engine and are visually consistent. Since this is a computer science thesis, this decision was ultimately pragmatic. Still, the team is very satisfied with the realistic look, and the first feedback was very positive.

However, in a realistic environment, the ant is more difficult to spot. To ensure that players can still easily identify their ant, each ant is assigned a distinct color, making it simpler for the player to track their ant.

Part II.

Product Description

1. Requirements

In this chapter, the requirements for the product are laid out. The requirements are split into functional and non-functional requirements.

1.1. Functional Requirements

This chapter describes the functional requirements of the system. The requirements are split into four parts:

1. Actors
2. Use Cases
3. System Requirements
4. Game Mechanics

1.1.1. Actors

This chapter describes the actors of the system and what their goals are.

Player

Goal The goal of the player is to join a game session and control a character in real time in the game.

Admin

Goal The goal of the admin is to monitor and manipulate a game session.

1.1.2. Use Cases

This chapter describes the use cases of the involved actors and how they are related to one another. A hybrid of brief and casual use case formats is used [6], following the structure outlined below.

1. Title
2. Primary Actor
3. Goal
4. Precondition
5. Postcondition

Use Case Diagram

The following use case diagram shows the interaction between the actors and the system.

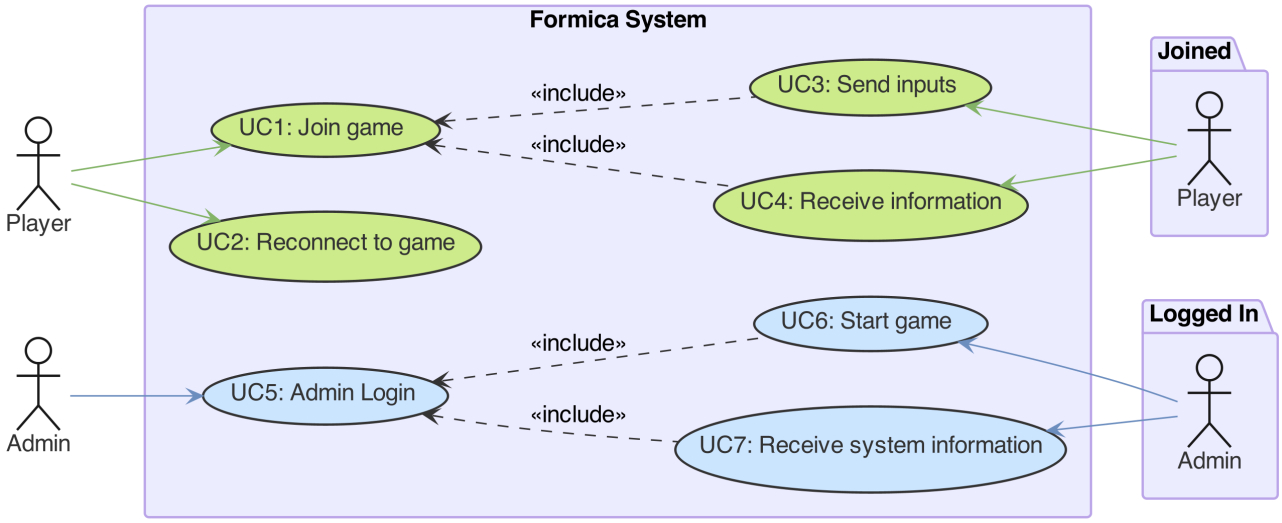


Figure 5.: Use Case Diagram

UC 1

Name: Join game
Primary Actor: Player
Goal: A player can join a game session by sending a game ID.
Preconditions: A game session is open.
Postconditions: The player is redirected to the controller view.

UC 2

Name: Reconnect to game
Primary Actor: Player
Goal: A player can reconnect to a running game session when the connection drops, by sending a game ID and their player ID.
Preconditions: The player has already connected once and has a player ID.
Postconditions: The player is redirected to the controller view.

UC 3

Name: Send inputs
Primary Actor: Player
Goal: A player can send inputs to their in-game character.
Preconditions: The player joined the game: UC-1
Postconditions: None

UC 4

Name: Receive information
Primary Actor: Player
Goal: A player receives information about his in-game character.
Preconditions: The user joined the game: UC-1
Postconditions: None

UC 5

Name: Admin login
Primary Actor: Admin
Goal: An admin can log in to the admin page using his credentials.
Preconditions: None
Postconditions: Admin is redirected to the admin dashboard.

UC 6

Name: Start game
Primary Actor: Admin
Goal: An admin can start the gameplay when all players have entered
Preconditions: The admin is logged in: UC-5
Postconditions: None

UC 7

Name: Receive system information
Primary Actor: Admin
Goal: An admin can view the monitoring page containing game and player information.
Preconditions: The admin is logged in: UC-5
Postconditions: None

Mockups

A mockup of the application was developed using Figma, based on the specified use cases. The GUI designs are intentionally simplistic to ensure user-friendly navigation and intuitive understanding. Additionally, due to the intended usage of the frontend in dimly lit environments, the design adopts a darker theme to prevent visual discomfort.

Landing Page

Figure 6.: Mockup landing page

As an alternative to the QR code, there is a landing page where the user can enter a game ID to join an existing game session.

Controller View

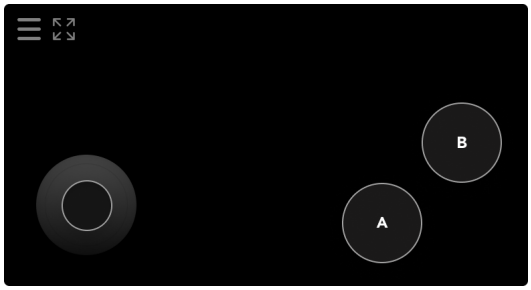


Figure 7.: Mockup controller

During the game, the player is presented with a control interface that represents the layout of a physical game controller. It includes a joystick, A and B buttons, along with a button to toggle fullscreen mode and a hamburger icon for accessing the menu.

Administrator Dashboard

Because the dashboard was created after the web controller and is intended solely for the show master, the team chose to implement it directly without creating prior mockups. Its design was adapted to closely resemble the already existing GUI elements.

1.1.3. System Requirements

The system requirements describe the expectations that arise from the use cases and specify aspects not covered by them. Each requirement is described using a brief text that states its objective.

SR 1

Name: Create game session
Goal: The system has to create a game session where players can join. There can only be one game session at a time.

SR 2

Name: Manage game session
Goal: The system has to manage the game session by keeping an up-to-date record of the game state, as well as all player states and connection information.

SR 3

Name: Create player
Goal: When a player successfully connects to the system for the first time (see SR 4 for prerequisites). The system must create a new player.

SR 4

Name: Validate player
Goal: When connecting the player has to send an auth payload containing a game ID and possibly a player ID. The system has to validate this payload. The connection is successful if the game ID is valid, else the connection has to be rejected. When a player ID is part of the payload, the system has to check if this player already exists (reconnect) or if a new player has to be created.

SR 5

Name: Validate admin
Goal: When connecting the admin has to send an auth payload containing their credentials. The system has to validate this payload. The connection is successful if the given credentials match the stored ones, else the connection has to be rejected.

SR 6

Name: Route messages
Goal: The system has to route the real time messages coming from outside actors and inside services.

SR 7

Name: Terminate connections
Goal: The system has to be able to terminate connections that are not or no longer valid.

1.1.4. Game Mechanics

This chapter defines the game loop and the mechanics implemented in the prototype. These elements are derived from the developed game concept.

Game Loop Diagram

The following diagram shows the simple game loop implemented in this project.

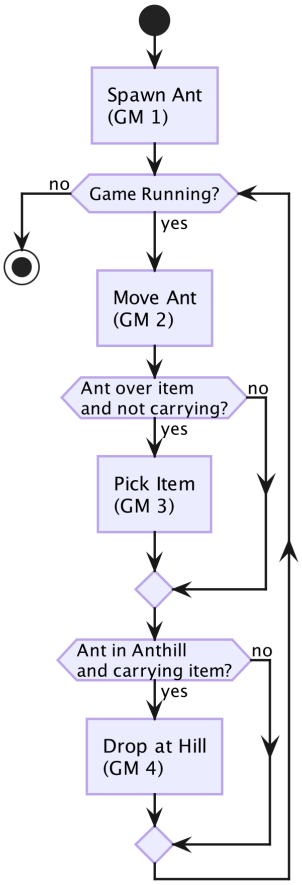


Figure 8.: Game Loop Diagram

GM 1

Name: Spawn ant
Goal: The game creates a new ant character and spawns it at the anthill.

GM 2

Name: Move ant
Goal: The ant character moves according to inputs received from the player.

GM 3

Name: Pick item
Goal: When an ant walks over an item and is not already carrying an item, it will pick it up

GM 4

Name: Drop at hill
Goal: When the ant walks into the anthill, it will drop the item. This will increase the score.

1.2. Non-Functional Requirements

This section specifies the non-functional requirements according to the FURPS classification [7] (Usability, Reliability, Performance, Supportability/Maintainability, and Security where applicable). Since performance is strongly influenced by the available infrastructure and the limited testing conditions in the planetarium during this project, no specific requirements were defined for the game's frame rate or the input latency.

NFR 1

Subject: Scalability (concurrent players) **Priority:** High
Category: Performance
Description: System must support at least 20 concurrent connected controllers within one session.
Justification: Shows can involve large audiences.

NFR 2

Subject: Join/leave robustness **Priority:** High
Category: Reliability
Description: Player disconnects, reconnects, and late joins shall not interrupt the running session.
Justification: The game must continue uninterrupted.

NFR 3

Subject: Cross-platform controller access **Priority:** High
Category: Usability
Description: Controllers must run without installation on Chrome, Safari and Firefox via QR-code on boarding.
Justification: Zero-install lowers friction and accessibility

NFR 4

Subject: Content portability (venues) **Priority:** Low
Category: Portability
Description: Maintain configurable camera/projector mapping to deploy on both hemisphere domes and flat cinema screens.
Justification: Enables reuse in different venues.

2. Domain Analysis

2.1. Domain Model

The following model illustrates the domain of the system. It describes the relations between the core entities.

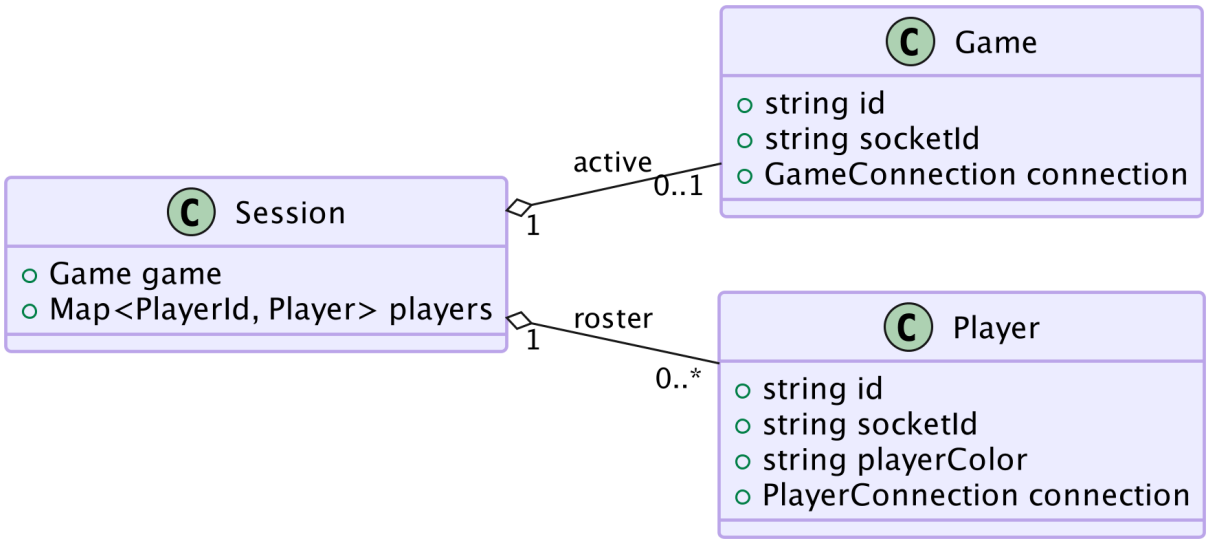


Figure 9.: Domain Model of Project

Part III.

Product Realization

1. Architecture

This chapter describes the overall software architecture of the system.

1.1. C4 Architecture

The software architecture is described with the C4 standard following the three main layers context, app and component. The C4 model was used because of past experience in the team with it in other projects and is a developer-friendly approach to software architecture diagramming [8].

Architect role: Leo will act as the architect.

1.1.1. Context

The first layer of the C4-Model describes the context in which the system operates.
Besides our own system and the actors derived from the requirement analysis, we had the prerequisite to use the CXRC plugin, provided by COSM a partner of the Swiss Museum of Transportation planetarium, to be able to project onto the dome. Read more about CXRC in chapter game engine.

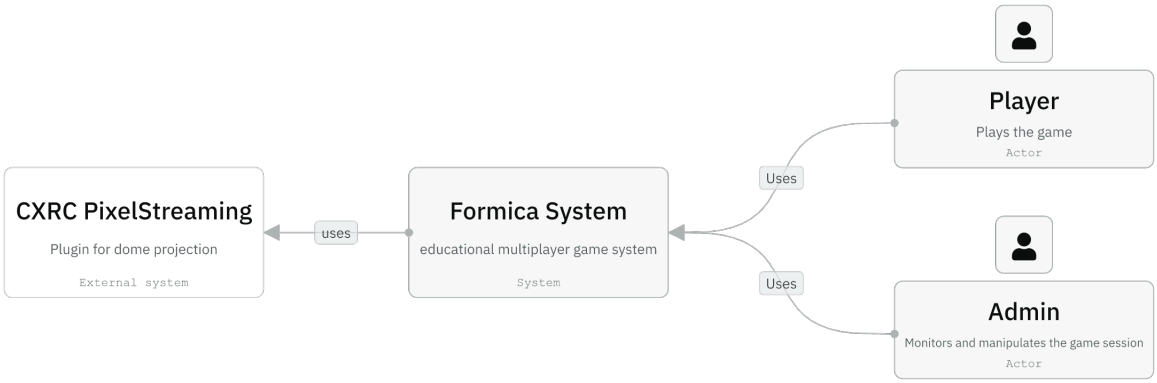


Figure 10.: Context Diagram

1.1.2. App

Looking one layer deeper, the system is divided into four applications. Data is exchanged between them using the WebSocket protocol. The reasoning behind choosing this protocol is discussed in chapter Frameworks and Libraries.

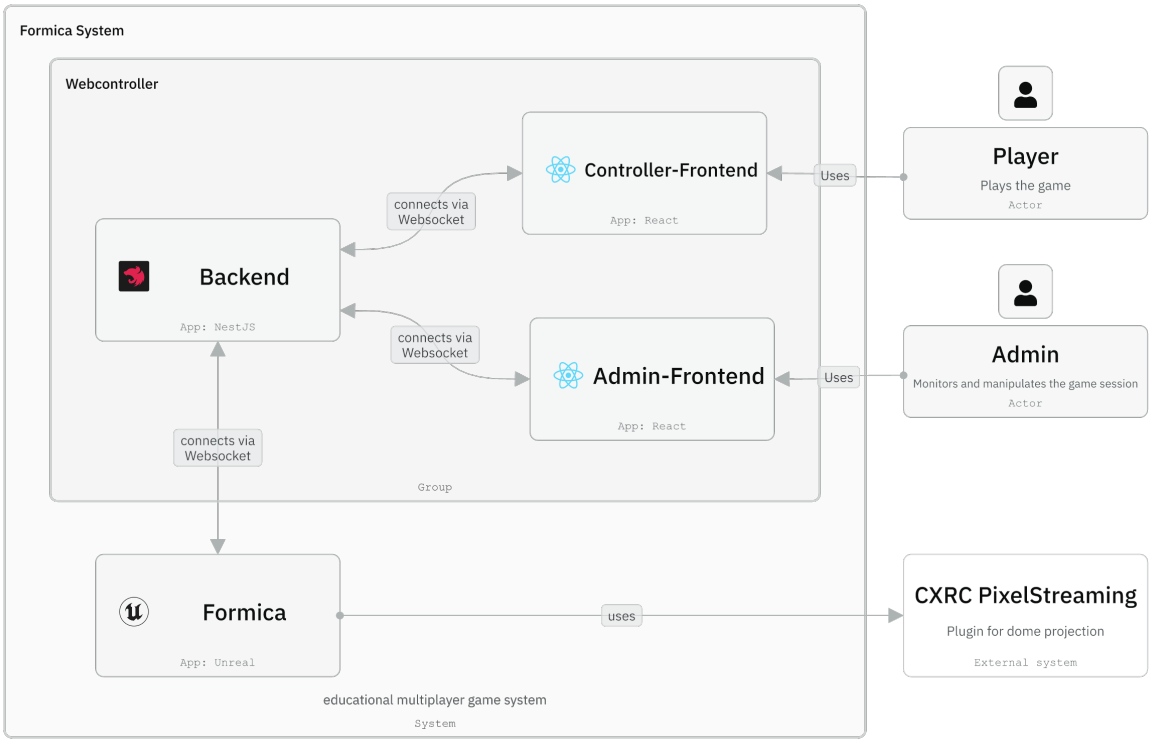


Figure 11.: App Diagram

The applications are split into two repositories:

- web-controller: holding the web applications (frontend / backend)
- formica: holding the game.

This gives a clear separation between the game and the controls, but keeps the frontends and backend close for simpler maintenance. The decision for this approach is based on our learning from the course Distributed Systems [9].

1.1.3. Components

At the deepest layer, the component diagrams illustrate the structure and internal connections of each individual application.

Frontend Components

The two frontend applications look structurally the same. The diagram shows their source code structure. They differ in visual appearance and functionality. Learn more in chapter GUI and Functionality.

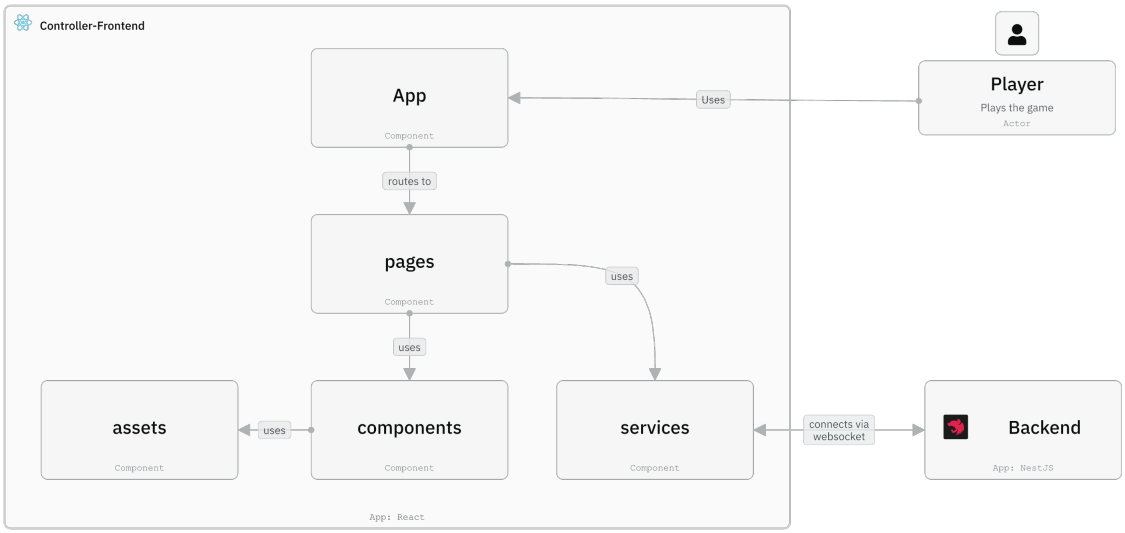


Figure 12.: Controller Frontend Component Diagram

The frontends are built with React and handle the WebSocket communication with Socket.IO. The reasoning behind this can be read in chapter Frameworks and Libraries. To enable fast and easy deployment, both applications are dockerized.

Backend Components

Like the frontends, the backend source structure aligns with the architecture depicted in the diagram below. It is the backbone of the internal messaging system between the frontends and the game. Learn more about this in chapter Internal Messaging System.

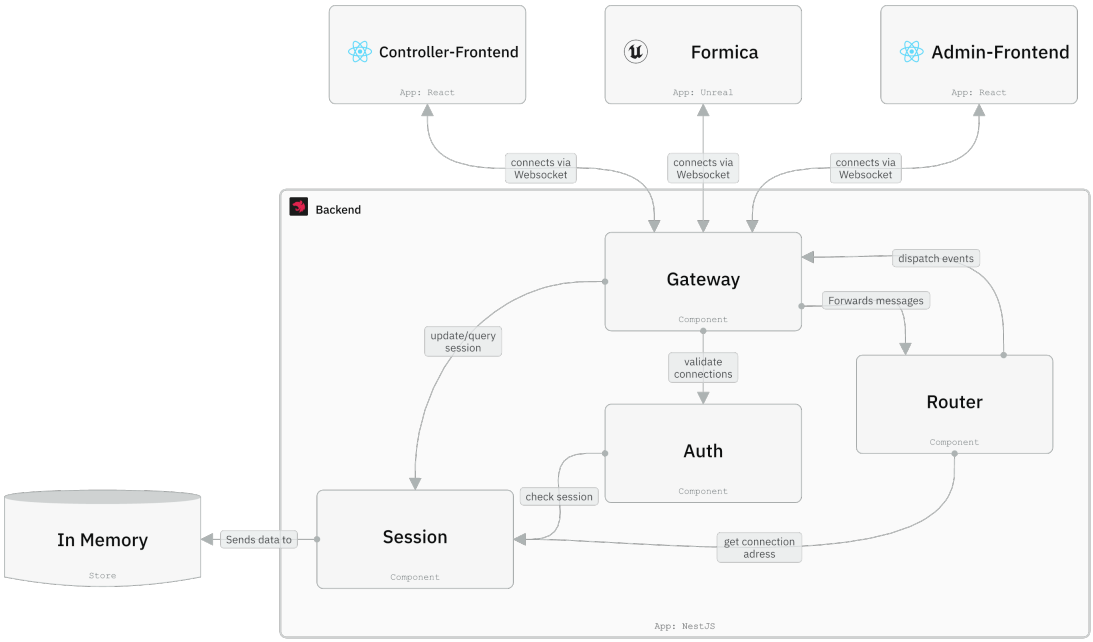


Figure 13.: Backend Component Diagram

The backend is built with Node.js using the NestJS framework and the Socket.IO library. The game session store is an in-memory store. Learn more about the reasoning behind this in chapter Frameworks and Libraries. To enable fast and easy deployment, the application is dockerized.

Formica Components

The game "Formica" is implemented in Unreal Engine. The Component diagram tries to follow the structural setup of the directory "Content/Formica/" in the git-repository. The Maps are found under "Content/CXRC" following the template project received from our external partners.

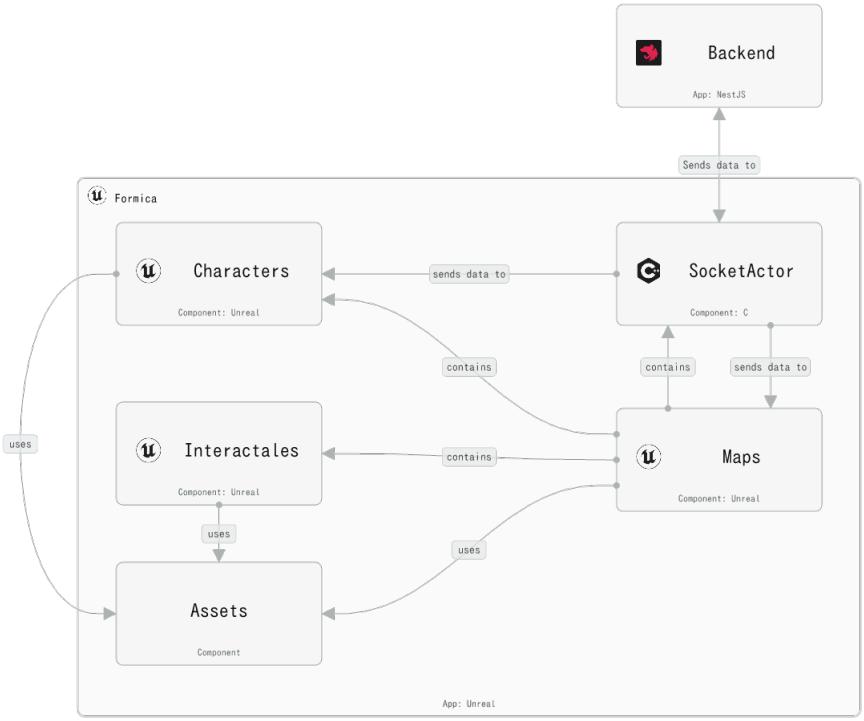


Figure 14.: Formica Component Diagram

The SocketActor is implemented in C++ and is located under "Source/Formica/" which contains all C++ files. Learn more about this in chapter Game Engine.

2. Evaluation of Frameworks and Libraries

2.1. Frontend

The following section evaluates the frontend frameworks and libraries considered for the project.

2.1.1. React

To ensure a clean architecture for the frontend, a component-based approach was chosen, to enable reusable, maintainable, and modular code [10]. The decision was made between the most well-known frameworks / libraries.

React	Vue	Angular
UI library	Progressive framework	Full-fledged framework
Lightweight	Lightweight	Complex applications
Low boilerplate	Low boilerplate	High boilerplate
Most knowledge	Some knowledge	Minimal knowledge
High performance	High performance	Medium performance
Rare breaking changes	Rare breaking changes	More frequent breaking changes

Table 1.: Comparison of Frontend Frameworks

Since Angular would be too bloated for this small frontend and the knowledge of the group was limited, the decision focused on React and Vue [11, 12]. Although Vue offers an intuitive structure and built-in reactivity, React was selected due to stronger team expertise and excellent community support [10].

2.1.2. Redux

Redux provides a centralized state management system that simplifies debugging of data flow [13]. Even though it introduces additional complexity, it streamlines CRUD operations and consolidates state debugging in a single location [14].

For communication with the server via WebSocket, an optimistic update strategy was chosen. The state is updated locally first and only corrected if the server returns a differing state. With this approach, the reactive user experience is guaranteed. The state of the buttons and joystick of the controller are not stored in the redux store, because of their volatile nature [13].

2.1.3. Tailwind CSS

Tailwind CSS was chosen for its simplicity, flexibility, and positive experience in the SE-Project. Its seamless integration with React supports consistent design practices and improves the maintainability of the user interface [15].

2.2. Backend

2.2.1. Language

The language selection was based on a paper which researched the performance of WebSockets in various programming languages [16]. It suggests Node.js as a solid option for its simplicity and performance. TypeScript was chosen based on the fact that typed languages improve code quality, reduce runtime errors and were preferred by the team [17].

2.2.2. NestJS

The backend is implemented using NestJS, a Node.js framework designed for building scalable and maintainable server-side applications [18]. NestJS was chosen mainly due to its excellent documentation and its built-in support for Socket.IO, which is essential for real-time communication in a multiplayer environment [19].

The project relies on low-latency, bidirectional communication between the game server and many web-based controller clients. NestJS provides a clean abstraction for WebSocket gateways, making it straightforward to implement event-based communication, connection handling, and session management [19].

Additionally, NestJS enforces a modular architecture with dependency injection, which helps to clearly separate responsibilities such as player management, game sessions, and admin controls.

2.2.3. Store

The backend is responsible for managing the active game session. The session is short-lived and exists only for the duration of a single gameplay. Persistence across backend restarts was therefore considered out of scope for this project, as unexpected reboots are highly unlikely and the goal of the project is a prototype. Given these constraints, fast access to session data was prioritized, especially due to the high frequency of real-time requests communicated via WebSockets. To meet these requirements, the backend employs a simple in-memory store, providing immediate data access without additional latency. Other solutions like Redis [20] were considered but discarded due to integration- and operational overhead.

2.3. Protocol

2.3.1. WebSocket

To communicate between the game, the server, and the frontend, WebSocket was the natural choice because of its capability for real-time, bidirectional communication [21]. This protocol allows the server to push updates immediately, without polling, which helps ensure responsive, low-latency interactions [21]. Additionally, Unreal Engine integrates well with WebSocket through plugins [22].

Compared to traditional REST-based communication over HTTP, WebSocket offers persistent connections and is beneficial for real time interaction [21]. Also, alternatives like HTTP polling or Server-Sent Events (SSE) were considered but discarded because they introduce latency and do not offer a bidirectional control flow [21].

There are multiple libraries to support the usage of WebSocket.

	WS	**Socket.IO**	**uWebSockets.js**
Fallback transports	No	Yes (HTTP long-polling, WebTransport)	No
Auto-reconnect	No	Built-in	No
Rooms	No	Built-in	No
Performance	High	Medium	Very high

Table 2.: Comparison of WS, Socket.IO, and uWebSockets.js capabilities.

Despite Socket.IO's lower performance compared to other options, its convenience features, such as built-in auto-reconnect, were considered more important. Since the goal is to develop a functional prototype within a limited time frame, these practical advantages outweighed the potential drawbacks of reduced performance [23, 24, 25]. Additionally, Socket.IO provides an Unreal Engine plugin and an API [23].

3. Game Engine

3.1. Unreal Engine

The game is developed using Unreal Engine [22], which was chosen primarily due to its compatibility with the projection infrastructure at the Swiss Museum of Transport planetarium in Lucerne. The planetarium uses a multi-projector setup with six beamers driven by multiple graphics cards. This setup is supported through a dedicated Unreal Engine plugin provided by the venue, enabling the game to be rendered and distributed correctly across all projectors.

In addition, Unreal Engine is a powerful and mature game engine that makes it straightforward to achieve high-quality, realistic real-time visuals. Its advanced rendering capabilities, combined with a large ecosystem and strong tooling, make it well suited to create immersive experiences in a dome environment [22].

3.2. Blueprints and C++

Unreal Engine supports both visual scripting via Blueprints and native development in C++. In this project, a hybrid approach was applied. The real-time communication interface based on Socket.IO was implemented in C++, as well as QR Code generation and some other helper functions. Most other gameplay logic and interaction mechanics were implemented using Blueprints.

The project team did not have prior experience with Unreal Engine, Blueprints, or C++. However, since Unreal Engine was a fixed requirement due to the projection infrastructure of the planetarium and the provided tooling, this technology stack was an implicit choice.

3.3. CX Realtime Creator Plugin

The Swiss Museum of Transport provided both the custom plugin (CX Realtime Creator) and an Unreal Engine project template, defining the required camera setup, projection mapping, and rendering configuration [26]. This ensured that the game integrates seamlessly into the existing planetarium system and behaves consistently during dome projection.

During development, the same projection mapping can be visually emulated on standard development computers. This allows testing and iteration without requiring access to the planetarium hardware, significantly simplifying the development workflow.

3.4. SocketIOClient-Unreal Plugin

The SocketIOClient-Unreal plugin [27] made it possible to use Socket.IO for backend communication directly within Unreal, and it was one of the reasons why Socket.IO was chosen as the WebSocket protocol in the first place.

3.5. Quixel Megascans Assets

For environment assembly the forest environment, Quixel Megascans assets were used [28]. Megascans provides a large library of high-quality, photorealistic scanned assets such as vegetation, rocks, and ground surfaces, enabling the creation of realistic environments with minimal manual modeling effort.

The assets integrate seamlessly into Unreal Engine via Quixel Bridge [29] and are free to use with Unreal Engine, including for educational and student projects [30]. This made Megascans a suitable choice for efficiently achieving a visually convincing forest environment within the scope of the project.

4. GUI and Functionality

4.1. Web Controller

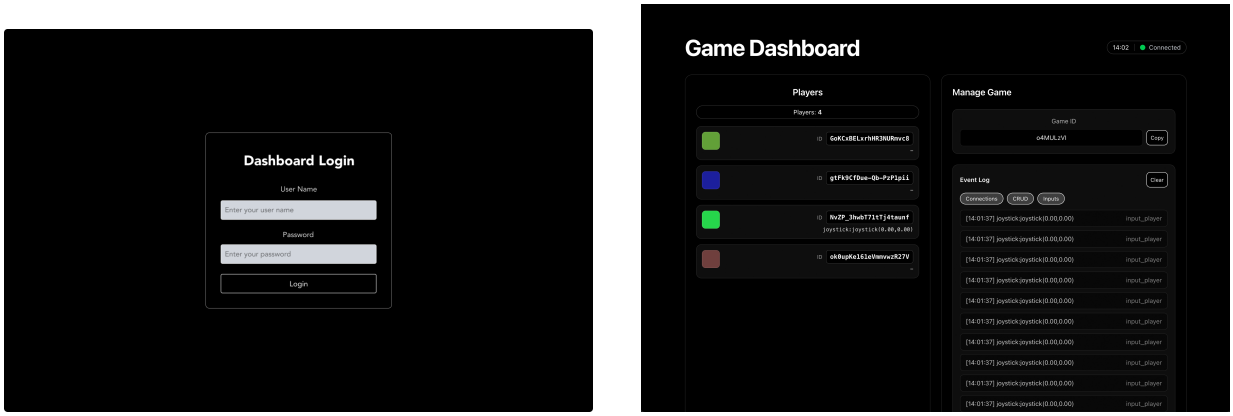
Figure 15.: GUI Web Controller

The GUI of the web controller closely matches the initial mockups. A color indicator in the shape of an ant was added, while the hamburger menu was removed, as additional menu options were considered unnecessary and would have added avoidable complexity to the user experience.

Although the proof-of-concept game technically integrates all controller components, only the joystick is currently used in Unreal Engine to control the ant's movement. The two action buttons are not utilized yet, but will be used for game mechanics in the future.

4.2. Administrator Dashboard

The administrator dashboard is used to monitor and control an active game session in real time. Access is protected by a simple login to prevent unauthorized use during live events. Once authenticated, the dashboard provides an overview of all connected players, including their connection status and assigned colors, as well as the currently active game session.
The administrator can start the game and observe player inputs and system events through a live event log. The dashboard is intended for use by a show host during a planetarium presentation.



(a) Screenshot Login Dashboard (b) Screenshot Administrator Dashboard

Figure 16.: Screenshots Dashboard

4.3. Unreal

Figure 17.: Scene composition for dome projection

The scene composition in Unreal Engine was straightforward but required careful consideration due to the dome projection and the underlying game mechanics. In a planetarium setting, the primary visual focus should be located at the top center of the projection, as this area is visible to all audience members. In the flat image this is the middle position. Consequently, the anthill was placed close to this position, ensuring that the main point of interest is centrally located. The peripheral areas of the projection are not intended for gameplay and serve a purely visual purpose.

5. Internal Messaging System

This chapter presents the messages defined for inter-application communication, detailing intended usage and their structure.

5.1. Message Flow

The following sequence diagrams describe the most relevant interaction patterns between the game, backend, player controllers, and the admin dashboard.

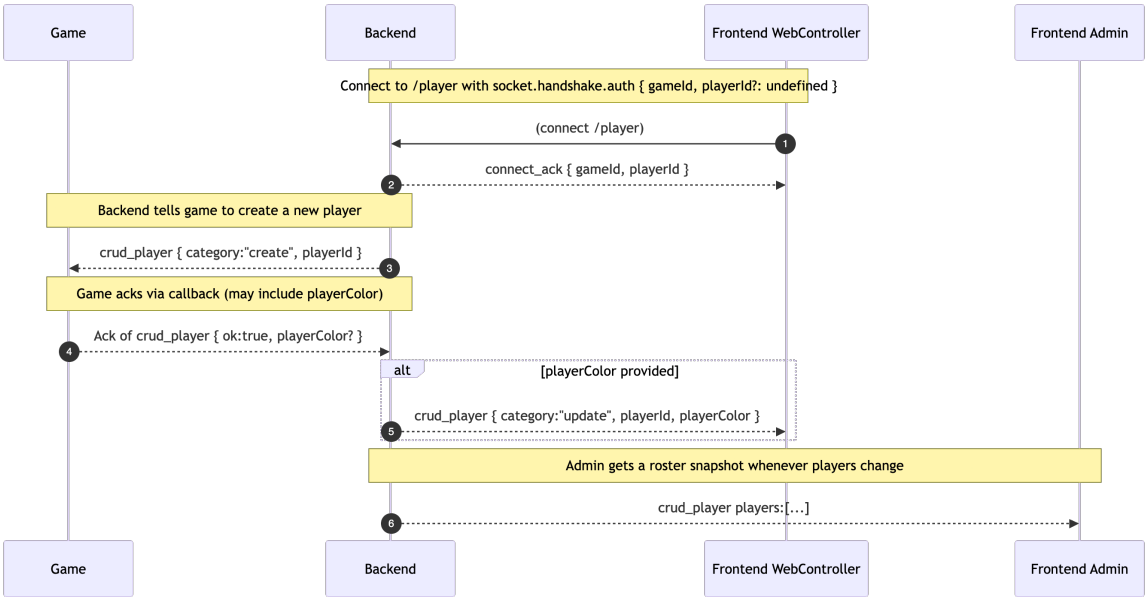


Figure 18.: Message Flow: User Creation

When a player joins an active session, the controller establishes a WebSocket connection to the backend and provides authentication data via the handshake. The backend validates the request, creates a new player entity if necessary, and notifies the game instance about the new participant. Once the game confirms the creation, optional player attributes (such as color) are synchronized across all connected clients. The admin dashboard is continuously updated with the current player status.

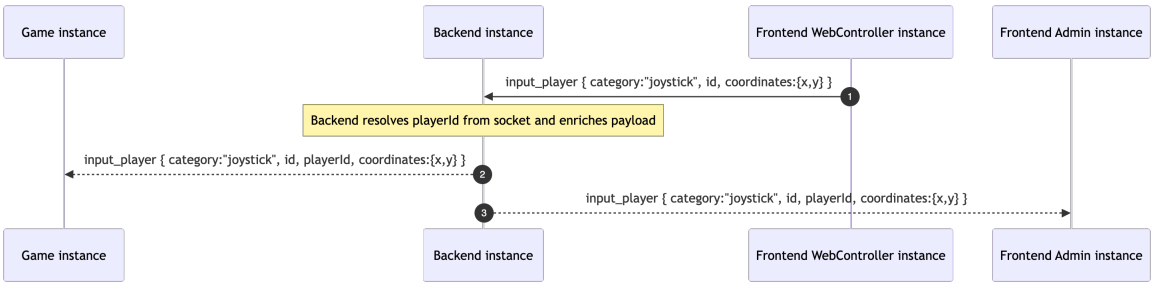


Figure 19.: Message Flow: Joystick Input

Player input is transmitted in real time from the controller to the backend. The backend resolves the player identity based on the socket connection, enriches the message with session-related metadata, and forwards the input to the game instance. In parallel, the same input events are mirrored to the admin dashboard for monitoring purposes.

5.2. Message definitions

The following tables describe the categorization and structure of the messages.

5.2.1. Game Namespace

Game Client ↔ Backend

Sender ↔ Receiver	**Channel**	**Payload Schema**	**Notes**
Game → Backend	`connect (handshake.auth)`	`{ gameId?: string }`	Omitting `gameId` creates a new game session
Backend → Game	`connect_ack`	`{ gameId: string }`	Confirms active game ID
Backend → Game	`crud_player`	`{ category: create|delete, playerId: string`	Player updates from backend
Game → Backend	`crud_player_ack`	`{ category, ok: bool, reason?: string, playerId, playerColor?: #RRGGBB }`	Acknowledgement callback, with optional player color
Backend → Game	`input_player`	`{ category, id, playerId, action?, coordinates? }`	Passing player control inputs
Backend → Game	`game_io`	`{ category: start }`	Passing start event
Backend → Game	`error`	JavaScript `Error` object	Validation or routing error

5.2.2. Player Namespace

Player Controller Frontend ↔ Backend

Sender ↔ Receiver	**Channel**	**Payload Schema**	**Notes**
Controller → Backend	`connect (handshake.auth)`	`{ gameId: string, playerId?: string }`	Initial connect or reconnect
Backend → Controller	`connect_ack`	`{ gameId: string, playerId: string }`	Client persists both identifiers
Controller → Backend	`input_player`	`{ category, id, action?, coordinates? }`	Passing player inputs
Backend → Controller	`crud_player`	`{ category: update|delete, playerColor?: #RRGGBB }`	create, read, update or delete player
Backend → Controller	`error`	JavaScript `Error` object	Emitted on validation failure

5.2.3. Admin Namespace

Admin Dashboard Frontend ↔ Backend

Sender ↔ Receiver	**Channel**	**Payload Schema**	**Notes**
Dashboard → Backend	`connect (handshake.auth)`	`{ password: string }`	Password-based authentication
Backend → Dashboard	`connect_ack`	`{ gameId, players: [ id, socketId, playerColor, connection ] }`	Acknowledgement, with serialized player map and game Id
Backend → Dashboard	`crud_player`	`players: [ id, socketId, playerColor, connection ] }`	Serialized player map
Backend → Dashboard	`input_player`	`{ playerId, category, id, action?, coordinates? }`	Passing player controller inputs
Dashboard → Backend	`game_io`	`{ category: start }`	Sending game start event
Backend → Dashboard	`error`	JavaScript `Error` object	Admin validation or session error

6. Production Setup

This chapter describes how the final system is packaged and delivered to the customer, including the provided artifacts and their deployment structure. The deliverables were packaged into a ZIP archive and delivered electronically via email.

6.1. Overview

The system is distributed as a Windows executable with the corresponding folder structure for the game, alongside a Docker-based setup for the web modules. It is designed to run within a local network, with both the backend services and the Unreal application running on the same machine. Client devices connect to the system over the WLAN. However, the setup can be easily adapted for deployment over the internet.

6.2. Unreal

The Unreal Engine project is manually built for Windows. The build process produces a standalone .exe file along with the required directory structure.

6.3. Frontend and Backend

The web controller, dashboard frontend, and backend services are containerized and deployed via a GitHub Actions CI pipeline. During the build process, a Docker image is created and pushed to the GitHub Container Registry.

The React-based frontends (controller and dashboard) are compiled into static HTML, JavaScript, and CSS assets. These assets are served by an `nginx` web server running inside the container. The backend service is included in the same Docker image, ensuring a unified and consistent deployment across all environments.

7. Quality Measures

7.1. Working Environment

7.1.1. Git organization

The project code is managed across two GitHub repositories. One for the web part (including both its web controller and dashboard frontend with the backend) and another for the Unreal-based game. This setup helps decouple the controller, making it easier to maintain and reuse in other interactive experiences.

- Backend, Web Controller Frontend, Admin Frontend:
  https://github.com/pheonix8/web-controller
- Unreal game:
  https://github.com/pheonix8/Formica

Both of these git repositories are private. Get in touch with the authors if you would like to have access.

7.1.2. Code Guidelines

To ensure good quality of our code, we will adhere to the guidelines explained in the following subsections.

Branching: The branching has the following structure: Main > Feature. The main branch is protected and will not allow direct commits and pushes.

Merge request: New code will be integrated via merge requests. Merge requests follow the 4-eyes principle and must be reviewed by the other team member.

Merge conflicts: Merge conflicts will be resolved with rebases if possible and will be resolved by the developer who created the merge request.

Naming conventions: We use naming conventions that are common for the languages and context.

Inline comments: Inline comments are to be kept to a minimum. They should only be used when the code is not self-explanatory.

7.1.3. Definition of Done

To ensure the completeness of our features, each YouTrack user story is checked against a checklist that serves as the Definition of Done. We use the following template:

- Code reviewed under the four-eyes principle
- System tests passed
- NFRs met
- User tests passed

7.1.4. Documentation

The project documentation is written in LaTeX and edited in Overleaf.

7.1.5. Project Planning

Project planning and management are handled in YouTrack. For detailed information, please refer to the chapter *Project Planning Method*.

7.1.6. Tools for Building and Deployment

The web repository's code is constructed using npm, with ESLint ensuring quality.

All web parts undergo testing and deployment by the CI system, which runs with every modification on the main branch.

7.2. Testing

7.2.1. Component Tests

For integration tests in the frontend, we use Cypress. To test the frontend parts and the backend separately, there are multiple mock servers. These mock servers were also a big help during production, so during changes, we could change the mock server to implement our ideas and then apply the changes to the real services.

7.2.2. System Tests

System tests were done manually to reduce complexity while learning Unreal Engine. For future development proper system tests are planned.

7.2.3. Smoke Tests

Both developers perform smoke tests while developing. This is done to ensure that the system works as expected and that the changes made do not accidentally break anything.

7.2.4. Manual Tests

The game was tested only manually because it was developed in Unreal Engine, where automated testing would have required significant additional effort and custom tooling beyond the project scope. Given the strong visual and interactive focus of the game, manual testing allowed developers to more effectively evaluate gameplay behavior, user interaction, and overall experience.

7.2.5. On-site Tests

Within the planetarium infrastructure, Prototype 1 was tested by the team of the Swiss Museum of Transport. The test confirmed that the game is displayed correctly on the dome. However, the Docker-based services including the web controller could not be started during the session, and the available testing time was very limited. The reported error messages indicate an issue with the Docker daemon, which suggests that Docker was not running.

7.2.6. User Tests

During development, we ran two user test sessions with groups, to understand how the system behaves with multiple instances and how users responded to the project. The tests were done after the milestones "Prototype 1" and "Prototype 2".

7.2.7. User test - Prototype 1

The first user test was conducted on 30 October 2025 with a group of 18 participants. The main goal of this test was to evaluate basic system functionality, performance, and user interaction under real usage conditions.

Before the test, significant performance issues were observed on the laptop running the game. In particular, the rendered 3D graphics caused severe slowdowns, leading to noticeable delays between user input and in-game response. To ensure that the test could continue, most of the 3D visual elements had to be removed, after which the system became usable and functioned reliably overall.

From a usability perspective, the core interaction concept worked, but several issues became apparent. The controls were perceived as unintuitive by multiple users. Additionally, users accessing the game via iPhones were unable to use fullscreen mode.

Furthermore, a bug in the spawning logic caused an excessive number of ants to be generated during gameplay.

Despite these issues, an important positive outcome of the test was that users were generally able to locate and recognize their own ant on the shared screen, even without additional color

indicators displayed on their mobile devices. Also, some of them were hooked and still played, even though we announced the test was over.

Overall, the first prototype demonstrated that the core concept was functional, but highlighted several performance, usability, and stability issues that needed to be addressed in subsequent iterations.

7.2.8. User test - Prototype 2

The second user test was conducted on 17 December 2025 with a group of five participants. Entering the game worked reliably for all participants. However, a user was initially unable to join because he was not connected to the WLAN, but after resolving the network issue, the connection was successful. The core gameplay functionality operated as intended, and the general feedback was positive. Collecting items was perceived as intuitive and easy to understand.

Nevertheless, several issues were identified. Some collision shapes in the environment were still inaccurate, which led to ants intersecting with environmental geometry. In addition, controlling the direction of the movement remained challenging. In the planetarium setting, it is not clear in which direction players are facing the screen, as seating is arranged in a circular layout. The current approach is not sufficiently intuitive for use by a wide audience. This indicates that an alternative steering concept is required. Finally, a bug was discovered. Joining the game via QR code did not consistently assign the correct player color and, in some cases, resulted in multiple ants being spawned for a single participant.

Part IV.

Conclusion

1. Conclusion

1.1. Evaluation

The objective of this study project was to assess the feasibility of using a planetarium dome as a shared, interactive medium for large audiences through a real-time multiplayer game controlled via personal smartphones. This objective was achieved. The implemented prototype demonstrates that simultaneous interaction of many participants within a fulldome environment is technically feasible using standard consumer devices.

The system architecture proved to be suitable for the intended use case. The backend, web-based controller and administration interfaces, and the Unreal Engine game interacted reliably during testing. Core challenges such as session management, real-time input routing, and reconnect handling were successfully addressed. User tests with up to 18 concurrent participants showed stable system behavior and acceptable responsiveness on development hardware, indicating sufficient scalability for the scope of this project.

The smartphone-based interaction concept was validated in practice. Participants were able to join sessions quickly and interact with the system with minimal instruction. Platform-specific constraints, such as limited fullscreen support on iOS devices, were identified as technical limitations and were subsequently handled in the following iterations.

Overall, the project confirms the feasibility of interactive, large-audience experiences in a fulldome environment and provides a solid foundation for further development and more comprehensive evaluation in a subsequent bachelor thesis.

1.1.1. Limitations

Prototype scope:
The scope of the project was initially very ambiguous, and already during the planning phase it became evident that the implementation of the game would result in a proof of concept. Consequently, several planned game features were deliberately omitted to keep the project feasible within the given time constraints. In addition, Unreal Engine and C++ were entirely new technologies for the project team. In hindsight, allocating dedicated time exclusively for learning and familiarization with these technologies would have been beneficial.

Limited user testing diversity
User tests were conducted with a small and homogeneous number of participants. Also, no user tests were done in the planetarium context. As a result, the findings may not fully represent behavior, usability, or performance in large public planetarium audiences.

Controller orientation ambiguity
Due to the circular seating arrangement in planetariums, the current control scheme does not reliably account for differing viewer orientations, which affects intuitive navigation.

Manual testing
The Unreal Engine game component, backend and system tests were executed manually. Automated testing will be crucial for scaling and generalizing the system.

Educational impact not formally evaluated
Although the game follows the principles of gamification, no structured pedagogical studies were conducted.

C++ Code
Since we used Vibe Coding for some C++ code, we ran into bigger issues to maintain it, and the code quality is still not at the targeted level.

1.1.2. Fulfillment of Functional and Non-Functional Requirements

All functional requirements have been met. Also, all non-functional requirements were fulfilled, with two requiring additional clarification:

- **NFR-1 (Scalability – concurrent players):** Although we were unable to test this directly within the planetarium's infrastructure, we performed the tests on a laptop with lower specifications. In this environment, the system handled 20 concurrent players without issues.
- **NFR-3 (Cross-platform controller access):** The controller application runs without installation in the tested mobile browsers. However, due to Apple's technical restrictions on iPhones, fullscreen mode is only available when the application is used as a PWA (Progressive Web Application), which requires installation.

1.2. Future Work

The planned bachelor thesis will be built directly on this study project. On a technical level, the web structure will be generalized and further developed to build a reusable framework for future multiplayer experiences in planetariums or other large-audience venues. On the gameplay and content side, the current prototype will be extended with game mechanics and the educational narrative. Particular emphasis will be placed on user experience design and the integration of automated agents to ensure a meaningful experience even with smaller groups. Furthermore, the didactic concept will be deepened, making the experience both entertaining and pedagogically.

The bachelor thesis will include evaluation of usability, performance and user tests in the planetarium context. By doing so, the project aims to transition from a prototype to an interactive application, which is evolved enough, so it can be used by the Swiss Museum of Transport and other similar venues.

Part V.

Appendix

1. Personal Reports

The collaboration within the team was consistently very positive and motivating. From the beginning, there was a high level of trust regarding task allocation and responsibilities. When unexpected obstacles arose, we supported each other effectively and worked together to find solutions. Tasks were completed as agreed, communication was open and constructive, and the overall working atmosphere remained positive throughout the project.

A significant portion of the work was carried out together in the same room. This proved to be very beneficial for communication. Previous experience in the SE project (SEP) was particularly helpful, as it allowed us to make informed decisions and reuse proven approaches.

It was extremely motivating to know that the result of our work is intended for real-world use in the planetarium. Interactive experiences tailored for large audiences are still rare, revealing a niche. Knowing that our work addresses this gap and has realistic prospects for deployment significantly increased our engagement.

From a planning perspective, we were largely able to follow our long-term plan and implement all of the features, except the pheromone trail mechanic. Time-wise, however, we tended to fall slightly behind schedule. The main reasons were unplanned refactoring phases and underestimated learning time, particularly for Unreal Engine and C++. In hindsight, allocating more explicit time for tutorials and structured learning would have been beneficial.

Another area identified for improvement is documentation. Although documentation existed, it would have been beneficial to document more key aspects such as the system architecture and message protocols at an earlier stage in detail. This would have provided a stronger reference during implementation.

With the technology choices we are very happy and everything went relatively smoothly. The only exception was a late downgrade of the Unreal Engine version, which could not have been foreseen, as the dome-mapping plugin was provided only at the end of November. Once the incompatibility became apparent, we reacted quickly, but lost more than a day for the downgrade.

The combination of the team members' skill sets worked especially well and the communication with both the academic mentor and the Swiss Museum of Transport was smooth and supportive throughout the project. Direct access to the external partner allowed for fast feedback and practical insights, and their openness to new ideas was highly appreciated.

In general, the project was not only a valuable learning experience, but also a solid foundation for a fun product. We are highly motivated to further develop the prototype into a complete and polished product as part of the bachelor thesis.

2. Project Plan

2.1. Processes, Meetings and Roles

2.1.1. People

The project team consisted of Leo Oetterli and Simon Ott. The project was supervised by Prof. Dr. Andreas Peter, who supported the team throughout the semester. For the Swiss Museum of Transport, Marc Horat (Head of the Planetarium) served as the primary contact person and provided the initial idea for the multiplayer game. Additional feedback and creative input were provided by Foteini Salveridou (Head of the Creative Lab).

2.1.2. Meetings

The team met with the academic advisor approximately every two weeks to discuss the project progress, clarify open questions, and receive feedback.

Communication with the external project partner, Marc Horat from the Swiss Museum of Transport, took place primarily via e-mail. In addition, one in-person meeting and one video meeting were held, during which the project concept was discussed.

Within the team, meetings were held in person at least once a week. These internal meetings were used for coordination, task planning, development and discussion of implementation decisions.

Formal meeting minutes were not maintained. Instead, the most important discussion points were documented in Notion.

2.1.3. Project roles

As a two-person team, no formal roles were assigned. Instead, all tasks were distributed dynamically via the ticket system in YouTrack.

Both team members collaborated on the Unreal Engine implementation. The backend and overall system architecture were primarily developed by Leo, while Simon focused on the frontend implementation and some design tasks.

2.1.4. Time

The project began at the start of the autumn semester on 15 September 2025 and concluded on 17 December 2025. In total, the team invested 470 hours into the development, documentation, coordination, and testing of the project.

A detailed breakdown of the planned and actual effort can be found in the chapters Project Plan and Time Tracking.

2.1.5. Cost

Aside from personnel costs, which are covered by the Eastern Switzerland University of Applied Sciences, and the internal staff costs at the planetarium (handled by the Swiss Museum of Transport itself), the project did not incur additional expenses.

Unreal Engine and all other software tools used were available free of charge for educational purposes, and all development work was performed on the personal devices of the team members.

2.2. Project Plan

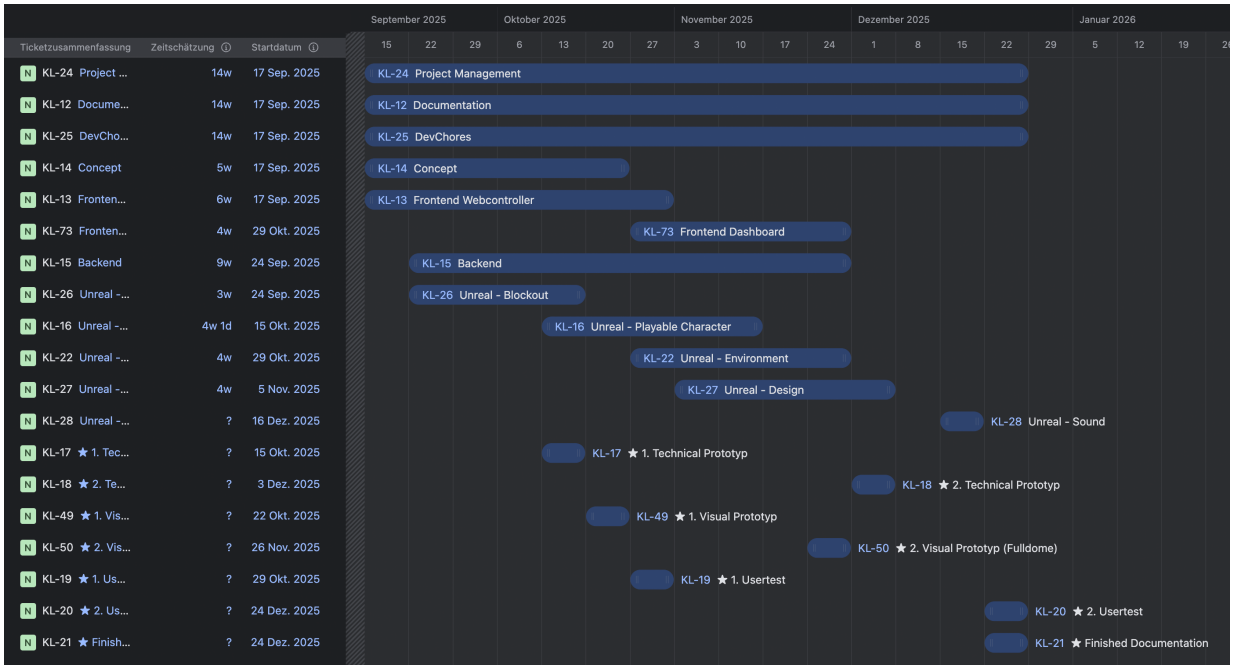


Figure 20.: Project Plan

The project was structured into three main phases: planning, implementation and finalization. During the planning phase, the game objective was developed, requirements were clarified and first architectural decisions were documented. The implementation phase focused on developing the core system components and creating an initial playable prototype. In the final phase, the prototype was stabilized.

Work was planned in two-week sprints. Although the sprint duration was two weeks, the team held weekly meetings to synchronize progress and sometimes adapt priorities.

Key milestones and epics are visualized in the screenshot of the long-time planning in YouTrack.

2.2.1. Short-term Planning and Change Management

Short-term planning was based on the active sprint. Tasks were selected from the backlog, assigned to team members, and adjusted as needed. Due to the small team size, planning remained lightweight and communication-driven.

Changes in scope or priorities were handled pragmatically. When new insights or technical obstacles emerged, the tasks were adjusted accordingly.

3. Time Tracking Report

3.1. Time Tracking Procedure

While project planning and task management was done using YouTrack, which served as the tool for defining epics, user stories, and tickets, we used a shared Excel sheet for time tracking. Even though YouTrack does offer some dedicated functions, for only two persons it seemed as a more practical approach and avoided over-engineering the time-tracking process.

The shared table allowed both team members to continuously monitor progress and directly check burndown charts. Time was tracked per epic, aligned with the corresponding YouTrack tickets. This level of abstraction was sufficient to understand how effort was distributed and to assess whether the overall workload stayed within the expected guidelines.

3.2. Overview

In total, Simon Ott invested 242.5 hours and Leo Oetterli invested 243.5 hours, resulting in 486 hours of total project effort. We worked every week one day together and distributed a second day individually through the week, which resulted in a fairly linear burn-down chart, with only a bit of fluctuation through the weeks and a balanced contribution from both team members.

Person	Category	KW38	KW39	KW40	KW41	KW42	KW43	KW44	KW45	KW46	KW47	KW48	KW49	KW50	KW51	Total per Category
Leo Oetterli	Project Management	4.5	2	0.5	3	1	0.5	1	1.5	0.5	3	1	1.5	0.5	1	21.5
Leo Oetterli	Documentation	7	5	2	1	5			2.5	4	3.5	4	8	2	19	63
Leo Oetterli	DevChores															0
Leo Oetterli	Concept	8	8.5	3	5											24.5
Leo Oetterli	Frontend Webcontroller						2.5	4								6.5
Leo Oetterli	Frontend Dashboard															0
Leo Oetterli	Backend		2	10	9	12	11.5	4	11	4.5	4					68
Leo Oetterli	Unreal - Setup							4		4.5	5					13.5
Leo Oetterli	Unreal - Character Logic							3.5		2			8	12	15	40.5
Leo Oetterli	Unreal - Environment												1			1
Leo Oetterli	Unreal - Design												2			2
Leo Oetterli	Unreal - Sound														3	3
Total per KW		19.5	17.5	15.5	18	18	14.5	16.5	15	15.5	15.5	16	21.5	20.5	20	
														Summe	243.5	

Person	Category	KW38	KW39	KW40	KW41	KW42	KW43	KW44	KW45	KW46	KW47	KW48	KW49	KW50	KW51	Total per Category
Simon Ott	Project Management	4.5	5	0.5	3	1	0.5	1	1.5	1.5	3	1	1.5	0.5	1	25.5
Simon Ott	Documentation	6.5	1				1			2		3	8.5	19	18	59
Simon Ott	DevChores															0
Simon Ott	Concept		7				3									10
Simon Ott	Frontend Webcontroller	8	4	6.5	12	1.5		4	9				1		1	47
Simon Ott	Frontend Dashboard											12	5	4		21
Simon Ott	Backend							4	1.5	4			2	3		14.5
Simon Ott	Unreal - Setup			3			7	4	4	12						30
Simon Ott	Unreal - Character Logic						5	12	3.5						2	22.5
Simon Ott	Unreal - Environment						4						2			6
Simon Ott	Unreal - Design			3	4											7
Simon Ott	Unreal - Sound															0
Total per KW		19	17	13	19	18.5	16.5	16.5	16	19.5	15	14	17	22.5	19	
														Summe	242.5	

Figure 21.: Time Tracking

At the beginning of the project, a significant amount of time was dedicated to project management as well as to documenting the architecture and conceptual decisions. After the planning phase, Leo started working on the backend, while Simon focused on the web controller and conducted initial tests in Unreal to establish a proof of concept. By the time of the first user test on October 30, an initial prototype was available, which subsequently required refactoring.

The next steps included developing the dashboard and implementing the necessary features in Unreal. Due to our lack of prior experience with Unreal, this phase proved to be challenging and time-consuming. Towards the end of the project, more time was allocated to documentation, and final code polishing was carried out.

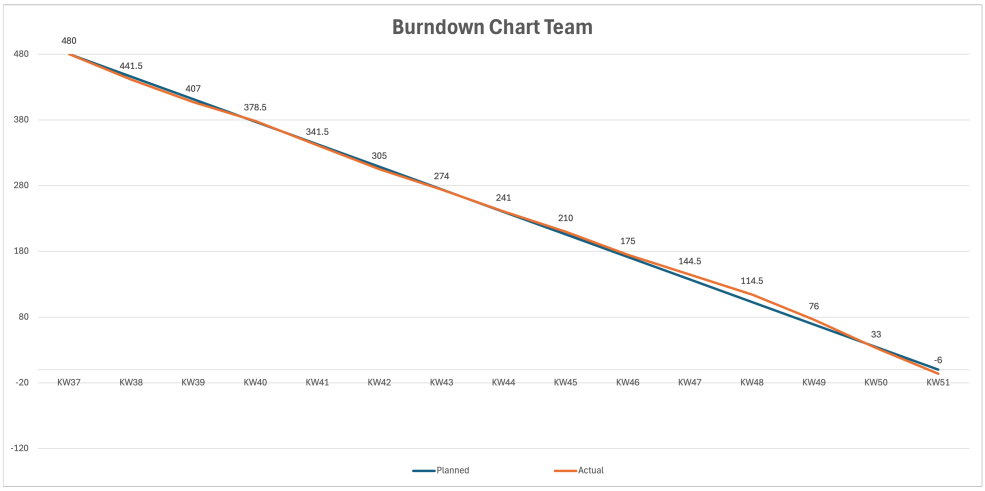


Figure 22.: Burn-Down Chart Team

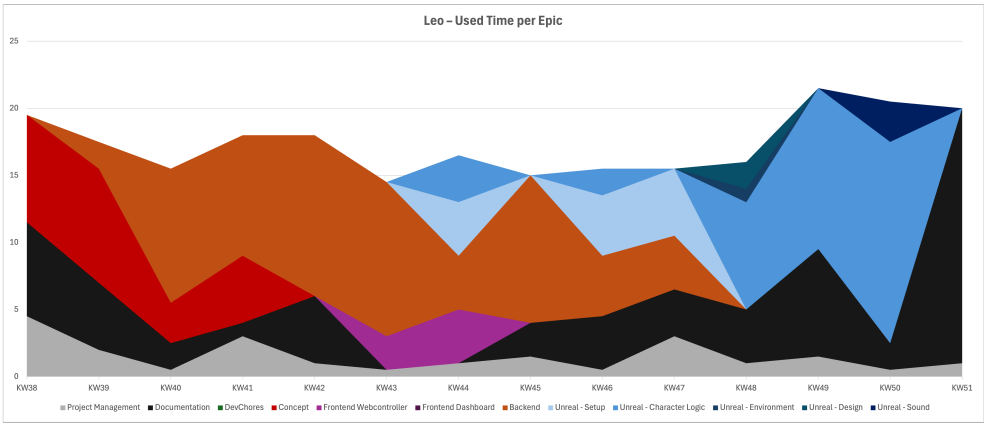


Figure 23.: Leo – Used Time per Epic

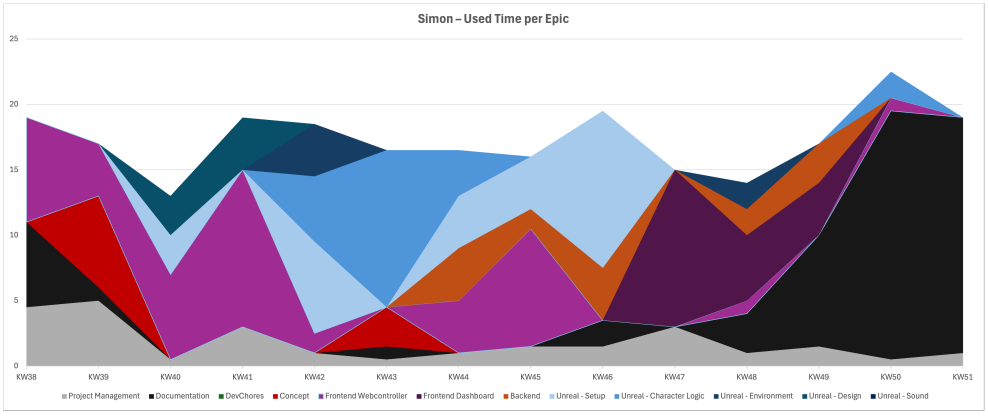


Figure 24.: Simon – Used Time per Epic

4. Risk Assessment

Figure 25.: Risk Matrix

R1 Limited testing opportunities in the planetarium

R2 Platform-specific constraints of mobile devices and browsers

R3 Dependency on plugin for dome projection

R4 Performance degradation under concurrent load

R5 Illness or unavailability of team members

R6 Steep learning curve of Unreal Engine and C++

R7 Development progress slower than planned

R8 Overengineering beyond prototype requirements

R1

Name: Limited testing opportunities in the planetarium
Probability: Likely
Impact: Insignificant

Risk: Testing in the planetarium is limited due to restricted access.

Mitigation: The project focuses on building the infrastructure and the experience itself is only a proof of concept.

R2

Name: Platform-specific constraints of mobile devices and browsers
Probability: Almost Certain
Impact: Significant

Risk: Platform differences may restrict functionality or affect usability.

Mitigation: A browser-based controller is used and limitations for specific tested devices are documented and mitigated if possible.

R3

Name: Dependency on Plugin for Dome Projection
Probability: Moderate
Impact: Major

Risk: The fulldome projection on the Planetarium relies on an Unreal Engine plugin.

Mitigation: Test the plugin as early as possible. If there is no technical solution, adapt the project objective to standard screens instead, such as those used in cinemas.

R4

Name: Performance degradation under concurrent load
Probability: Unlikely
Impact: Significant

Risk: System performance may degrade with many simultaneous users.

Mitigation: A more lightweight architecture is applied or the game concept has to be changed.

R5

Name: Illness or unavailability of team members
Probability: Moderate
Impact: Significant

Risk: Reduced team availability may slow down development progress.

Mitigation: Core features are prioritized and non-essential functionality is postponed if necessary.

R6

Name: Steep learning curve of Unreal Engine and C++
Probability: Likely
Impact: Significant

Risk: Limited prior experience may reduce development efficiency.

Mitigation: Core features are prioritized and logic is implemented more in Blueprints.

R7

Name: Development progress slower than planned
Probability: Likely
Impact: Significant

Risk: Learning curves and unforeseen technical issues may delay implementation.

Mitigation: Project scope gets adjusted.

R8

Name: Overengineering beyond prototype requirements
Probability: Unlikely
Impact: Minor

Risk: Unnecessary complexity may reduce development efficiency.

Mitigation: Implementation decisions are continuously aligned with the proof-of-concept scope.

4.1. Risk Update

Several of the identified risks partially materialized during the project. Limited access to the planetarium (R1) required that most testing be performed using local dome emulation. The Platform-specific limitations of mobile browsers (R2), particularly on iOS, were observed and documented during user tests. The steep learning curve of Unreal Engine and C++ (R6) slowed early development and was mitigated by prioritizing core features and relying more heavily on Blueprints. Performance problems under concurrent load (R4) occurred during initial tests, but were resolved through scene simplification and optimization. As a result, the scope and project plan of the project were adjusted (R7) and some planned game mechanics, such as the level of the pheromone trail, food drop and collaborative food carrying, were not implemented.

Additionally, the dome projection plugin (R3) was provided late in the project and introduced compatibility issues, which required a downgrade of the Unreal Engine version from version 5.6 to 5.5. This resulted in around two additional days of rework due to the rebuilding of the project. Since the documentation for the plugin is not publicly available, more detailed communication with the client could have helped identify this issue earlier and potentially avoid the delay. The project plan was adjusted accordingly, and despite this setback, the system operated correctly during the first test on the planetarium infrastructure.

5. Software and Tools Used

In the process of writing this thesis, a variety of tools were used to support research, design, implementation, collaboration, and deployment.

Task Area	**Tools**
Literature Research and Management	Google, Google Scholar, Swisscovery, ChatGPT
Data Visualization	Excel, Mermaid, Figma, Procreate, Photoshop
Ideation and Concept Development	Figma, Pinterest, ChatGPT
Translation	ChatGPT, DeepL
Mockup	Figma
Coding	Visual Studio Code, Visual Studio, WebStorm, Unreal Engine (Blueprints), GitHub Copilot, Create React App, React, Redux, Tailwind CSS, ESLint, NestJS, Socket.IO, TypeScript, C++
Text Creation, Editing, Spell Checking and Grammar	Overleaf, ChatGPT, DeepL
Collaboration and Project Management	Microsoft Teams, GitHub, YouTrack, Notion
DevOps	Docker, GitHub Actions, GitHub Container Registry, Nginx, Cypress
Game Development	Unreal Engine, CX Realtime Creator (plug-in for COSM dome mapping)
Assets and Content Creation	Blender, SideFX Houdini, Quixel Megascans

Table 6.: Overview of Tools and Resources

List of Figures

Bibliography

[1] ZEISS. *Fulldome Systems Featuring Standard Video Projectors*. Image used: "Fulldome System Example". URL: https://www.zeiss.com/planetariums/en/products-and-solutions/planetarium-technique/fulldome-systems/standard-projectors.html (visited on 12/18/2025).

[2] Kinetarium. *Kinetarium – Interactive Multiplayer Experiences for Planetariums*. Product website. Accessed: 2025-12-14. 2025. URL: https://www.kinetarium.com.

[3] Cosmic Orbiters. *Cosmic Orbiters – Multiplayer Fulldome Game*. Online project description. Accessed: 2025-12-14. 2025. URL: https://cosmic-orbiters.com.

[4] Sky-Skan. *DigitalSky Planetarium Systems*. Accessed: 2025-12-14. Sky-Skan, Inc. 2025. URL: https://www.skyskan.com.

[5] Villehardt. *Forest Atmosphere Sound*. Accessed: 2025-12-13. 2021. URL: https://freesound.org/people/Villehardt/sounds/613995/.

[6] Wikipedia. *Use case*. URL: https://en.wikipedia.org/wiki/Use_case.

[7] Robert B. Grady and Deborah L. Caswell. *Software Metrics: Establishing a Company-Wide Program*. Introduces the FURPS model; URPS denotes the non-functional requirement categories. Prentice-Hall, 1987.

[8] Simon Brown. *Official website for the C4 Model*. Accessed: 2025-12-17. 2025. URL: https://c4model.com.

[9] OST - Eastern Switzerland University of Applied Science. *Distributed Systems, Syllabus*. Accessed: 2025-12-17. OST - Eastern Switzerland University of Applied Science. 2025. URL: https://studien.ost.ch/allModules/39054_M_DSy.html.

[10] Meta Platforms, Inc. *React Documentation: Learn*. Accessed: 2025-12-13. React. 2025. URL: https://react.dev/learn.

[11] Google. *Angular Documentation: What is Angular*. Accessed: 2025-12-13. Angular. 2025. URL: https://angular.io/guide/what-is-angular.

[12] Vue.js. *Vue.js Documentation: Introduction*. Accessed: 2025-12-13. Vue.js. 2025. URL: https://vuejs.org/guide/introduction.html.

[13] Redux Team. *Redux Documentation: Core Concepts*. Accessed: 2025-12-13. Redux. 2025. URL: https://redux.js.org/introduction/core-concepts.

[14] Redux Team. *Redux Documentation: Why Redux*. Accessed: 2025-12-13. Redux. 2025. URL: https://redux.js.org/introduction/why-redux.

[15] Tailwind Labs. *Tailwind CSS Documentation: Utility-First Fundamentals*. Accessed: 2025-12-13. Tailwind CSS. 2025. URL: https://tailwindcss.com/docs/utility-first.

[16] Matt Tomasetti. “An Analysis of the Performance of Websockets in Various Programming Languages and Libraries”. Accessed: 2025-12-18. PhD thesis. Ramapo College of New Jersey, 2021. DOI: 10.13140/RG.2.2.18552.57607. URL: https://www.researchgate.net/publication/348993267_An_Analysis_of_the_Performance_of_Websockets_in_Various_Programming_Languages_and_Libraries.

[17] Laxmi Narayana Pattanayak. *Reasons to Choose TypeScript Over JavaScript*. URL: https://dev.to/laxminarayana31/reasons-to-choose-typescript-over-javascript-16nd (visited on 12/18/2025).

[18] NestJS. *NestJS Documentation*. Accessed: 2025-12-13. NestJS. 2025. URL: https://docs.nestjs.com.

[19] NestJS. *NestJS Documentation: WebSockets (Gateways)*. Accessed: 2025-12-13. NestJS. 2025. URL: https://docs.nestjs.com/websockets/gateways.

[20] Redis. *Redis - The Real-time Data Platform*. URL: https://redis.io/ (visited on 12/18/2025).

[21] Ian Fette and Alexey Melnikov. *The WebSocket Protocol*. Tech. rep. RFC 6455. IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455.

[22] Epic Games. *Unreal Engine Documentation*. Accessed: 2025-12-13. Epic Games. 2025. URL: https://docs.unrealengine.com.

[23] Socket.IO. *Socket.IO Documentation*. Accessed: 2025-12-13. Socket.IO. 2025. URL: https://socket.io/docs/v4.

[24] websockets. *ws: A Node.js WebSocket library*. GitHub repository. Accessed: 2025-12-13. 2025. URL: https://github.com/websockets/ws.

[25] uNetworking. *uWebSockets.js: High-performance WebSockets*. GitHub repository. Accessed: 2025-12-13. 2025. URL: https://github.com/uNetworking/uWebSockets.js.

[26] COSM. *CX Realtime Creator for Unreal Engine 5.0+*. Internal documentation. Provided by Swiss Museum of Transport; accessed 2025-12-13. 2023.

[27] getnamo. *SocketIOClient-Unreal*. Accessed: 2025-12-13. GitHub. 2025. URL: https://github.com/getnamo/SocketIOClient-Unreal.

[28] Epic Games. *Quixel Megascans*. Accessed: 2025-12-13. Epic Games. 2025. URL: https://quixel.com/megascans.

[29] Epic Games. *Quixel Bridge Documentation (Unreal Engine)*. Accessed: 2025-12-13. Epic Games. 2025. URL: https://docs.unrealengine.com/QuixelBridge.

[30] Epic Games. *Students and Schools*. URL: https://www.unrealengine.com/en-US/students-and-schools (visited on 12/18/2025).

Attached Documents

- Agreement of own contribution
- Copyright agreement
- Publication agreement
- SA Abstract
- Plain-text abstract
- ZIP archive