University of Southern Denmark
Mærsk Mc-Kinney Møller Institute
Faculty of Engineering

Bachelor's Project in Software Engineering

Developing Educational Games for Teaching Touch Typing

Klaes D. Sørensen

klsoe20@student.sdu.dk

182160602

Gustav B. Wanscher

guwan21@student.sdu.dk

4110804

Project Start	**Project End**	**Delivery Date**
01/09-2024	02/01-2025	29/12-2024

SDU Supervisor	**Vitec MV Supervisor**	**Keystrokes**
Henrik L. Bendixen	Esben Gaarsmand	116.548

I. Abstract

This Bachelors project, in collaboration with Vitec MV, aims to address the lack of touch typing education in Danish public schools by designing and developing an educational game, focusing on masking repetition (practice) behind interactive experiences. The project explores existing solutions and compares core features to identify areas of success and improvement while designing a unique solution.
As requested by Vitec MV, the product integrates into its existing “10-finger” platform, has internationalization support, a multiplayer minigame, and is appropriate for a target demographic of ages eight to 14 years old.
The project analyzed several technologies in regards to the expected end-user environment, eventually settling on Go, PostgreSQL, Gorilla WebSockets, SolidJS, and a custom schema-based, binary messaging format for multiplayer.
Testing the educational game solution, revealed that Vitec MV’s expectations were met and that the solution successfully engaged and challenged a majority of participants. A majority of participants even asked to try the solution again, however a significant number also had issues with the tutorial segment, due to a series of visual bugs and non-intuitive tasks.

II. Methods

The solution was developed with significant upfront planning to scope the project, and iterative and flexibel development to combat the unpredictable nature of software engineering, resulting in waterfall-agile hybrid development methodology. During development, members chose which project components and features they would invest time in, using a backlog through the service “Trello” (*Figure II*).

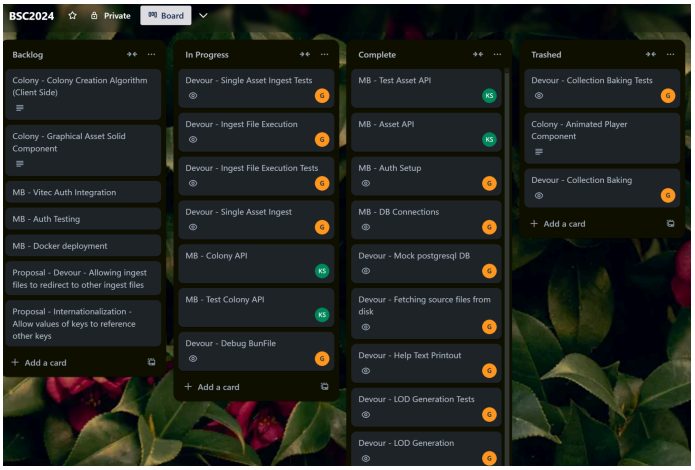


Figure II
Trello, backlog. An online development tool to communicate work packages, work in progress and completed work.

Additionally, general communication was maintained through the messaging service “Discord”.
Continued stakeholder communication and engagement [52] were maintained through meetings and email. Development Logs (Appendix E) were used at a later stage as progress reports.

Index

1 Introduction

In Denmark, based on examination of both “Bekendtgørelse af lov om folkeskolen” [48] and “Bekendtgørelse om formål, kompetencemål, færdigheds- og vidensområder og opmærksomhedspunkter for folkeskolens fag og emner” [49], there is emphasis on digital competencies in general terms, but none specifically indicating touch typing is part of these. Official teaching guidelines, extrapolated and developed based on these laws, makes no mention of touch typing being an official goal [50][51].
The skill of touch typing becomes important as soon as students begin to encounter computers during education. When it comes to kids, engagement in education can be challenging, but tools to combat this and engage students exist, one of which is educational games [07 - p. 9].
To address this problem, this project explores the avenue of gamifying the touch typing learning process, producing an educational touch typing game, available to students in the browser through Vitec MV’s platform. This project acts as an addition to the already existing ten-finger educational game solution on Vitec MV’s platform. It aims to reinvent core functionality and adapt it to support and later implement additional features, such as user interaction through multiplayer and user persistence.
This report details the analysis of the fields of touch typing and educational games, as well as technologies to implement fundamental components of an educational browser game with aforementioned features. It outlines design processes and development strategies used to produce the project deliverables. It documents user- and acceptance tests, discussing analysis, design, implementations and test results, as well as success, faults, improvements and future possibilities.

1.1 Who Is Vitec MV

Vitec MV, formerly known as: “MV-Nordic” [06], formerly known as: “Mikro Værkstedet A/S” [13 - §3], was acquired by Vitec Software Group (VMS), a global investment group, in 2017 [16]. VMS itself has a rather broad investment portfolio including: Commerce, education, energy, finance, healthcare, insurance, real estate industries, and even churches.

1.1.1 Products

Vitec MV specializes in education, aiming to provide students with supportive products. As of December 1st, 2024, Vitec MV’s main product is IntoWords, a piece of software that aids students with literacy, by providing text-to-speech word completion/suggestion, dictionaries, and more. IntoWords is a digital expansion and replacement of the predecessor CD-ORD, published in 1995 as the first reading and writing technology, which was distributed physically using CD-ROM.
Additionally, Vitec MV provides Subreader, a program for reading aloud subtitles in video, as well as 10-finger, to enhance keyboard writing efficiency [16]. The latter is the “parent” platform of this project.

1.1.2 Stakeholder Interests

Given the supplied project description (see Appendix A) and public information available on the 10-finger product, a surface level of this stakeholder's interests can be established. As emphasized in the project pitch (Appendix A), Vitec MV is interested in expanding upon its existing 10-finger solution and further exploring the concept of 10-finger gamification. They aim to make the 10-finger learning experience engaging and entertaining for children [18]. Presumably, the product must be implemented in the browser for accessibility purposes, but it could also be in the interest of later expansion.
As detailed in the project description, Vitec MV would like to explore multiplayer approaches, especially in a classroom setting, hoping to leverage a familiar (to some) system like Kahoot. Kahoot's multiplayer system works through a single, short numeric code, which can be shared for others to join the experience.
As an extension of this, they would like teacher participation to be optional since, at the end of the day, it would be the teachers who have to see value in the product, as some might prefer not to be involved. Additionally, Vitec MV would like to provide users with performance metrics to evaluate progress (Appendix B).

1.2 Project Origin

The original project, as presented by Vitec-MV representatives at "PP-day" ("Projects & Internships-day") 2023, describes adding simple games to their existing platform ("10-finger"), one at a time, with multiplayer being optional.
The intention was to mask the repetition and practice required to develop better typing skills behind games and other interactive activities to keep the attention and focus of the target demographic (ages eight to 14) for longer.
However, through further dialogue with Vitec MV CTO, Esben Gaarsmand, it became clear that there existed a wish for features outside of the scope presented in the initial handout. Namely, persistent user progress, visualizing the users' advances in other than statistics, support for multiple languages, and preferably a "space"-theme.

The full, original, pitch can be found in Appendix A**.**
A transcription of the meeting notes from the first project meeting can be found in Appendix B.

1.3 What is Touch Typing?

As defined by Ann Dobson, author of 'TOUCH TYPING IN TEN HOURS', touch typing is to: "Type each line until it is perfect and, most importantly, until it can be typed without looking at the keys or your fingers" [05].
Touch typing has also been called "blind typing", "ten-finger typing", "touch type" and "touch keyboarding". The goal of touch typing is to increase typing speed while maintaining accuracy, by removing the need for continued reorientation of the keyboard every new word or keystroke, and instead relying on muscle memory through practice.
An alternative, and vastly slower, writing method would be "hunt & peck", i.e. searching for each key individually, and pressing when found. This can usually be seen as people writing with two fingers and spending a significant amount of time looking at the keyboard.

1.4 Why is the lack of proficiency a problem?

The use of “devices” (computers, tablets, smartphones) in classrooms has increased dramatically over the years [11]. For this project, computer use is most relevant (tablets with attachable keyboards included), as touch typing relies on keyboards and grants any student a series of advantages if proficient, mainly in terms of time & distractions. A summary article by Harvard supports this notion, outlining that it is the distraction, not the device that is detrimental to learning [14].

Although not necessarily apparent, time usage does impact the classroom a lot. For instance when taking notes.

A proficient student will have a better chance of interpreting the subject at hand while writing, whereas a student using “hunt and peck” will be distracted as they have to divert more attention and time elsewhere. Another example would be tests and exams, specifically those with free-form/essay questions (i.e. not multiple-choice), where any proficient student would be able to express their answer faster, and potentially more fulfillingly.

1.5 What is an educational game?

As defined by Richard E. Mayor, “Computer games that are intended to promote learning” [07 - pp. 9], an educational game is one that means to teach the player in some regard. A major point raised by Mayer is that much of the data collected on educational games follow different approaches and have different preconditions.

Supporting this notion, in the meta-analytic review by Gui, Y., Cai, Z., Yang, Y. [08], there are several inconsistencies in the conclusions of many such studies. The meta-analysis took 86 studies into account and calculated an estimate of how much of an impact educational games had on science, technology, engineering, and math. The study found the effect of learning through a game in these fields, as opposed to traditional learning, was significant, described as a: “medium to large positive impact”.

The meta-analysis delved into a further 44 studies, to estimate the impact of improving game design elements, and found that incentive or interaction mechanisms further improved the learning impact by another ∼50%. The study noted that games that prioritized the subject over entertainment value, had a greater positive impact on the learning experience.

This suggests that educational games can significantly improve learning and that incentives and interactive elements play a vital role in this enhancement.

For touch typing specifically, a study by Weigelt Marom and Weintraub [09], concludes that, while short-term practice with touch typing programs decreases typing proficiency, long-term use of touch typing programs significantly improves typing speed while maintaining accuracy, and even more so for students with literacy impairment (e.g. dyslexia).

A study by Fodor, S., & Varga, M. [10] expands on Weigelt Marom and Weintraub’s observations. Their study had a group of participants playing the game ‘Dungeon Typer’, which gamified the touch typing learning process through positive reinforcement, progressions, multiplayer, and competition. They further focused on fostering a stress-free environment for the participants. Students reported both improvements in typing and enjoyment during the learning process, although those observations were self-reported and subject to bias.

While it is not entirely certain that gamification may inherently enhance typing proficiency more so than regular educational touch typing programs, gamification may increase motivation to participate, thereby indirectly improving learning.

2 Analysis & Planning

The solution needed to involve multiplayer, multiple languages, and integration with Vitec MV's existing platform. The target audience was students the ages eight to 14, in a school environment, on varying personal or school-provided devices, which required analysis into sufficiently efficient supporting technologies.
Based on further analysis and communication with Vitec MV, a set of conditions of satisfaction were made, a project overview statement was agreed upon, and specific requirements were derived, categorized, and prioritized.
Finally, a risk management process was used to identify risks, assess their characteristics, and produce mitigation plans where necessary. Additional emphasis was put on sensitive data regarding persistence and multiplayer.

2.1 Conditions of Satisfaction

Through dialogue with Vitec MV, the following conditions of satisfaction were derived.

ID	Description
Vitec MV	
COSV 01	The delivered product should be able to become part of Vitec-MV's existing platform.
COSV 02	The product should allow for the support of multiple languages
COSV 03	The product should allow for multiplayer within the same class of students, akin to Kahoot.
COSV 04	The product should offer persistent progress per user, i.e. a user may see things change or get improved as they progress through the game.
COSV 05	The teacher is a big part of the equation and should provide value in some form to them. Preferably by making teacher participation optional.
COSV 06	The product must contain, or be, at least 1 minigame, where a user may select from multiple difficulties.
SDU	
COSS 01	The project should demonstrate the student's ability to competently formulate, analyze, and address issues within a specific academic subject that reflects the main focus of the education.
COSS 02	The students must be able to explain, analyze and reflect upon theories, methods and practices within the education's curriculum.

Table 2.1.1
A list of conditions that are to be met for the Vitec MV to be satisfied.

2.2 Requirement Breakdown Structure

Each condition of satisfaction derived above (*Table 2.1.1*), was broken down into more concrete requirements in the following requirement breakdown structure (*Table 2.2.1*) [47].
Additionally, this was combined with more traditional MoSCoW and FURPS analysis for further prioritization and evaluation.

MoSCoW

A prioritization system for requirements, subdividing them into whether the inclusion in the final deliverable is a "must" / "should" / "could" or "want". Where "must" indicates something that cannot be left out, and "want" something that will. [46]

FURPS

A categorization tool for breaking any requirement into one of 5 categories: "Functional", "Usability", " Reliability", "Performance" & "Supportability". Each category extends to several adjacent fields, each noted in the table below. [45]

Legend:
REQ Requirement
FUN Function
SUB Sub-function

ID	Type	Description		MoSCoW
1	REQ	COSV01		
1.1	**FUN**	**Platform Compatibility**	**FURPS** S (Adaptability)	
1.1.1	SUB	Analyzing the existing Vitec MV platform		S
1.1.2	SUB	Identify integration points with Vitec-MV		S
1.1.3	SUB	Define necessary adaptations for compatibility		M
2	REQ	COSV02		
2.1	**FUN**	**Language Support**	**FURPS** U (Human Factors)	
2.1.1	SUB	Ability to store and retrieve text in multiple languages		S
2.1.2	SUB	Ensure support for language-specific characters across all text fields		S
2.1.3	SUB	API integration for seamless multi-language handling		M
2.1.4	SUB	Implementation of multi-language user interface elements		M
2.2	**FUN**	**Encoding Compliance**	**FURPS** U (Consistency)	
2.2.1	SUB	All stored text data must be encoded using a format supportive of special characters.		S

2.2.2	SUB	Ensure compatibility of chosen encoding	S
3	REQ	COSV03	
3.1	**FUN**	**Multiplayer Support**	**FURPS** F (Capability)
3.1.1	SUB	Enable multiplayer functionality within the same classroom	M
3.1.2	SUB	Support for seamless integration of multiplayer	S
3.1.3	SUB	Ensure scalability to handle multiple concurrent multiplayer sessions	S
3.1.4	SUB	Integration of automated deployment processes for multiplayer features	S
4	REQ	COSV04	
4.1	**FUN**	**Persistent user progress**	**FURPS** F (Capability)
4.1.1	SUB	Support for saving and retrieving user progress across sessions	S
4.1.2	SUB	Enable dynamic updates to the game based on user progress	S
4.1.3	SUB	Ensure data integrity and consistency in user progress tracking	M
4.1.4	SUB	Integration of user progress tracking with the main backend system	M
5	REQ	COSV05	
5.1	**FUN**	**Self-Teachable System**	**FURPS** U (Human Factors)
5.1.1	SUB	The system should include a self-teaching mechanism that allows students to progress without teacher intervention.	S
5.1.2	SUB	The product should provide an intuitive tutorial to guide students through the main gameplay mechanics.	S
5.1.3	SUB	The tutorial should be effective in conveying the necessary information to ensure student understanding and engagement.	S
6	REQ	COSV06	
6.1	**FUN**	**Minigames**	**FURPS** F (Capability)
6.1.1	SUB	The product should include a framework to support the development and integration of minigames.	S
6.1.2	SUB	The minigame(s) must allow users to select different difficulty levels	M
6.1.3	SUB	Ensure that all minigames are scalable and can handle various input formats and difficulty metrics.	S

6.1.4	SUB	Support for additional minigames, beyond the initial required game, should be possible within the framework.		S
7	REQ	n/a (derived)		
7.1	**FUN**	**Client Limitations**	**FURPS** P (Performance)	
7.1.1	SUB	The product should be able to run on lesser hardware		S
7.1.2	SUB	The product should be able to run on sub-par internet connections		S

Table 2.2.1
A breakdown of conditions of satisfaction, to identify and specify actionable requirements.

2.3 Project Overview Statement

Based on the agreed-upon Conditions of Satisfaction and the Requirements Breakdown Structure, a Project Overview Statement was developed (*Figure 2.3.1*). This document outlined the project's goals, objectives, success criteria, and associated risks. It served as a document of approval, seeking approval from Vitec MV, ensuring alignment among stakeholders, and establishing a shared understanding of the project's purpose, content, and deliverables.

Project Overview Statement	**Project Name**	**Project No.**	**Project Manager**
	Developing Educational Games for Teaching Touch Typing	PRJ-2024-01	Gustav B. Wanscher & Klaes D. Sørensen

Problem Opportunity
As computers have become the dominant writing tools, learning and understanding computer systems has become an intrinsic part of all education. Despite this, a severe lack of emphasis has been put on interacting with the computer itself - through keyboard and mouse (or trackpad). This leaves a vacuum in the tools and resources available, which is why Vitec MV has issued this project; to provide an educational, interactive product that teaches Touch Typing for children between the ages of eight and 14.

Goal
Help children aged eight to 14 learn how to write better on a computer keyboard through an educational, interactive web platform. Masking repetition and practice behind games.

Objectives
1. Teach Touch Typing
2. Engage the target audience
3. Allow for cooperation and/or competition through multiplayer.
4. Make the product relevant for more than just Danish children by supporting multiple languages.

Success Criteria
1. A majority (>50%) of the target audience should find the product engaging
2. A majority (>50%) of the target audience should be able to use and engage with the product without teacher engagement

3. When accessed on public school wifi, the majority of users must not experience delays greater than one second.

Assumptions, Risks & Obstacles

1. Teachers may find the product ineffective, illogical, taxing, or have other issues that lead to rejection of the product.
2. It is assumed that an educational game is a suitable medium for teaching the subject to the target audience.
3. The product is a balance between teaching the subject efficiently and being fun for the target audience, which may be a difficult balance to achieve.
4. To ensure Success Criteria #3, the product must be rather performant.
5. Inexperience with some implied technologies may cause delays.

Prepared By	Date	Approved By	Date
Klaes D. Sørensen & Gustav B. Wanscher	13082024	E. Gaarsmand	19082024

Figure 2.3.1

A document that outlines project objectives and goals, defines agreed-upon success criteria and potential risks and serves as a basis for securing final approval before initiating the project.

2.4 Existing Solutions

Vitec MV had their own browser-based touch typing product called '10-Finger' [19], which provided both a basic training program, a "history" course with stories integrated into the learning experience. It also had a game called 'Fishy Keys', with increasing difficulties to incentivize continual growth.

In total, the product offered 11 training courses with between 23-47 exercises per course. Uniquely, this was a Danish 10-finger service including integrations with other Vitec MV products and thereby offered literacy support by reading appearing words aloud.

The product attempted to visually guide students to correct hand placement with colorful and educational graphics hinting at the finger placements, as well as analytical feedback on user performance metrics [18].

With this in mind, it became clearer exactly what aspects and concepts of touch typing gamification Vitec MV had already explored, and painted a picture of what they desired to explore next.

2.4.1 Related Browser Solutions

At first glance when looking for similar games, a few appeared, such as 'Typing.com' [20] which like other online game platforms offered a set of different minigames.

In essence these games were very similar in terms of game mechanics, graphical themes, words and characters to be typed, with minor variations.

Mainly, either full words, random words, or singular characters were to be typed. While the games offered several options for difficulty and sentence/input structure, they lacked language support, multiplayer, and player persistence.

Despite this, they had a very simple and intuitive user interface and very satisfying and polished graphics that appeared to take inspiration from existing games such as Doodle Jump, Fruit Ninja, Plants vs Zombies, Space Invaders, and more.

A similar browser solution was ‘Nitrotype.com’ [21] which offered a set of typing courses and a single race car game that seemed to simulate a multiplayer experience.
Nitrotype.com claimed to offer teachers metrics on student performance, not only individually, but as a group. Similarly to Typing.com, this game lacked actual multiplayer and persistence but had simple and pleasing graphics.
‘Eclub.com’ [22] was another browser solution that offered an extensive course of up to 685 items, consisting of detailed video tutorials, engaging minigames, and plain typing exercises. Each had incredibly detailed visual feedback for each upcoming keystroke, visualizing hand placement, key location, and which exact finger was to be used.
Additionally, some exercises were specific to singular areas of the keyboard, such as the exercise ‘Travel L Pinky’ where users were to input a set of characters left of the keyboard using only their pinky finger. The site offered many more courses related to education and 10-finger education tailored to specific school grades, as well as multilingual support, although including none of the languages requested by Vitec MV. While the site did not appear to offer multiplayer, it did save user progress if an account was created.
Actively looking for a multiplayer game yielded ‘Type Racer’ [23], an online competitive racing game. The game itself was very simple and offered different versions of the game with numbers or repetitive words. The key difference with this game was the fact that it had multiplayer, though it lacked other aspects. The game was an example of 10-finger gamification, it did have user persistence in the form of user metrics but not in regards to in-game progress. Furthermore, it allowed users to create a game and join each other.

2.4.2 Related Gamification Solutions

‘Mario Teaches Typing’ was another gamification of the 10-finger learning process, and is one of, if not the first. It was of major production and did well on the market [24]. It was developed by Interplay Entertainment Corporation in 1997 [25] but was not listed on their official website [26].
The game was single-player and started slowly by using only singular keys and later on whole words. The game was a modification and simplification of ‘Super Mario World’ [27], where instead of using the traditional controls, players had to type prompted keys to interact with the world. While there was no obvious way to gain access to the game, emulators existed to run the game.

Mojang, the creators of ‘Minecraft’ [28] modified Minecraft to create ‘Minecraft: Education Edition’ [29], which was a version of Minecraft tailored to teach students a variety of subjects, such as literacy, math, and science [30]. While it was hard to say exactly how well it performed, using the intellectual property of Minecraft as an educational tool may have increased the reach of the product, much like Mario Teaches Typing.
While this project did not have access to any well-known existing intellectual properties, inspiration could be drawn from many of the aforementioned games. To aim at an audience within the specified frame of Vitec MV, it was deemed ideal to look at an existing educational game developed and shipped within Denmark, specifically ‘Matematik i Måneby’. Developed by ‘Bitfrost A/S’ in 2006 [31], it was an example of an educational game focused on math and rewarded users with visual changes to buildings. Minigames focusing on different topics were accessible at different buildings, and users could come back to a building and play on a greater difficulty level for greater rewards [32].

Side note: During research into Matematik i Måneby, it turns out that Vitec MV, while known as 'Mikro Værkstedet A/S' [13], was the distributor/publisher of Matematik i Måneby back in 2005. Persistent, visual, progress seemed useful for retaining user attention, as well as having several different minigames with different graphics, contents, and methods of 10-finger practice.
While many existing online and offline educational games existed, both of major and minor production and provided many features, none of them covered all of the functionalities proposed by Vitec MV (see Appendix B & Appendix A).

2.4.3 Product Feature Matrix

A compilation and comparison of existing solutions was made to visualize which components would differentiate this solution from existing solutions, if any (*Figure 2.4.3.1*).

Feature/ Solution	Vitec MV 10-finger	Typing .com	Nitro Type	Eclub	Type Racer	Mario Teaches Typing	Matematik i Måneby
Core Features							
10-finger Training	✔	✔	✔	✔	✔	✔	X
Structured Courses	✔	✔	X	✔	X	⚠	⚠
Performance Metrics	✔	✔	⚠	✔	✔	✔	X
Gamification							
Mini-games	⚠	✔	⚠	✔	X	✔	✔
Visual Feedback	✔	✔	✔	✔	✔	✔	✔
Progress Tracking	✔	✔	X	✔	⚠	✔	✔
Multiplayer Features							
Real-time Multiplayer	X	X	X	X	✔	X	X
User on User Interaction	X	X	X	X	✔	X	X
Technical Features							
Browser-based	✔	✔	✔	✔	✔	X	X
User Persistence	✔	✔	X	✔	✔	✔	✔
Language Support							
Danish Language	✔	X	X	X	X	X	✔

Multiple Languages	✔	✔	X	✔	✔	✔	X
Educational Features							
Teacher Analytics	X	✔	✔	✔	X	X	X
Literacy Support	✔	X	X	✔	X	X	X
Difficulty Progression	✔	✔	X	✔	⚠	✔	✔
✔ Feature exists			⚠ Feature partially exists			X Feature does not exist	

Figure 2.4.3.1
Product Feature Matrix. A comparison of features in existing solutions.

2.4.4 Uniqueness of Proposed Product

For this project, although the product would integrate with Vitec MV's existing 10-finger platform, it was considered an isolated offering, and so was not assumed to inherit any features or functionality.
Given the number of existing solutions and even some with rather extensive feature sets, it was difficult to provide a unique product from a naive, features-only perspective. But not impossible, rather an exercise in combining and expressing the current requirements in such a fashion, that the result would be unique.

According to Figure 2.4.3.1, multilingual- and multiplayer support alone would differentiate the product from all others but Type Racer. The final stretch was to ensure that the product was not a racing game or at least provided other, additional gameplay.

2.5 Work Breakdown Structure

Although phase-segregation of work packages might lend itself to a more discrete, waterfall-like approach, for this project, it was meant to be a rather more iterative process.
For instance, the services and features noted below were not implemented, tested, and then deployed as three separate steps as they might at first appear, but rather as one continuous loop.
With that said, the top-level phases (those marked as type "PHA") were meant as discrete.
Additionally, in the table below (Table 2.5.1), "activities" or "ACT" have been excluded for brevity, the full work breakdown structure can be found in Appendix G.

Legend
PHA Phase
SUB Sub-Phase
WPK Work Package
ACT Activity

ID	Type	Description
1	PHA	Research & Analysis
1.0.1	WPK	Architectural Constraints
1.0.2	WPK	Touch Typing Research
1.0.3	WPK	Learning Objectives
2	PHA	Design & Prototyping (Planning)
2.1	**SUB**	**Initial Designs**
2.1.1	WPK	Frontend
2.1.2	WPK	Main Backend
2.1.3	WPK	Databases
2.1.4	WPK	Multiplayer Backend
2.1.5	WPK	Minigames
2.1.6	WPK	Asset Mockups
2.2	**SUB**	**Prototyping**
2.2.1	WPK	Frontend
2.2.2	WPK	Databases
2.2.3	WPK	Multiplayer Backend
2.2.4	WPK	Integration Testing of Prototypes
2.3	**SUB**	**Reviewing Initial Designs**
2.3.1	WPK	Frontend Design
2.3.2	WPK	Review Main Backend API
2.3.3	WPK	Multiplayer Design
2.3.4	WPK	Colony & Player Database Structure
2.3.5	WPK	Language Database Structure
2.3.6	WPK	Revisit designs of indirectly affected entities
3	PHA	Implementation (Execution)
3.1	**SUB**	**Implementation**
3.1.1	WPK	Database Servers

3.1.2	WPK	Main Backend
3.1.3	WPK	Frontend
3.1.4	WPK	Multiplayer Backend
3.1.5	WPK	Implement First Game Across Services
3.1.6	WPK	DA Translation
3.2	**SUB**	**Unit Testing**
3.2.1	WPK	Unit Test Test-Suite Main Backend
3.2.2	WPK	Unit Test Test-Suite Frontend
3.2.3	WPK	Unit Test Test-Suite Multiplayer Backend
3.3	**SUB**	**Integration Testing**
3.3.1	WPK	Persistence
3.3.2	WPK	Platform Compatibility
3.3.1	WPK	Language
4	PHA	Deployment / Roll-out
4.1	**SUB**	**CI/CD Workflows**
4.1.1	WPK	Database Deployment
4.1.2	WPK	Frontend Deployment
4.1.3	WPK	Main Backend Deployment
4.1.4	WPK	Multiplayer Backend Deployment

Table 2.5.1 (condensed)
A hierarchical decomposition of requirements into detailed work packages.
See Appendix G for the full WBS.

2.5.1 Timeline

Prior to the official start of the project, a “Project Description” was made detailing the outline for the project, as well as an overall timeline as a GANTT chart. As of project start, it was not deemed necessary to reevaluate it, and so it became the official timeline, although not directly translating to the work packages as specified in Figure 2.5.1
Duly note that some tasks are in parallel rather than making a more fine grained chart.

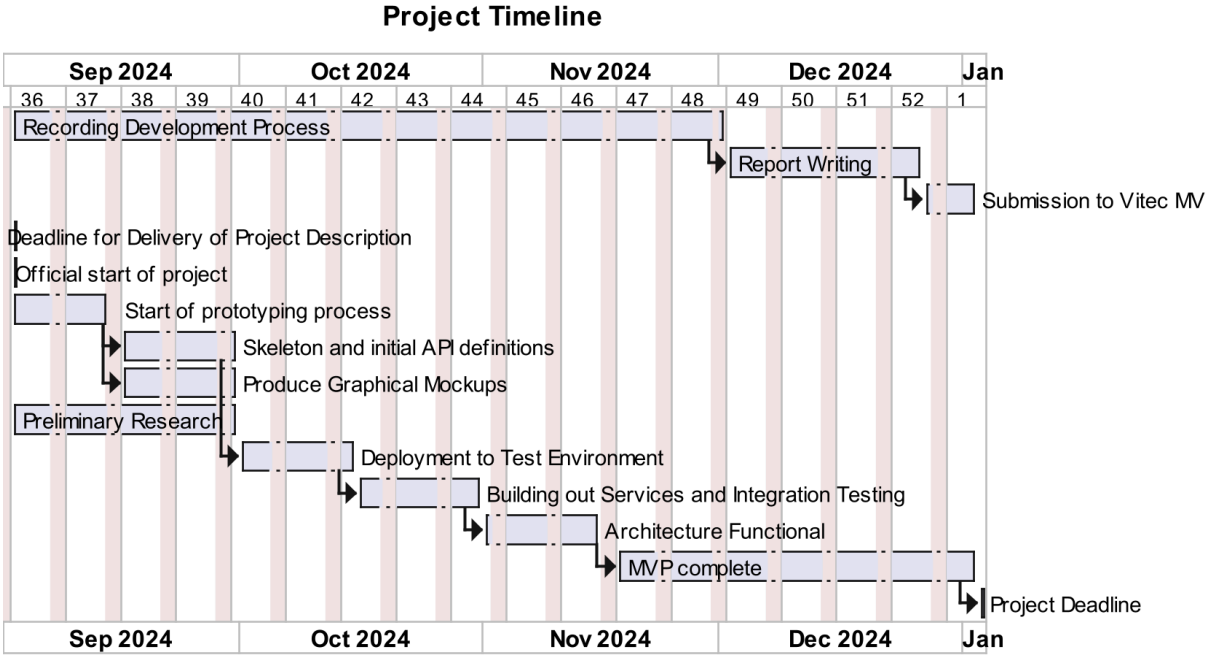


Figure 2.5.1.1
Subsection of Appendix U - Project Description

2.6 Stakeholders

According to the Project Management Institute, a stakeholder is: "An individual, group, or organization that may affect, be affected by, or perceive itself to be affected by a decision, activity, or outcome of a project, program, or portfolio." When developing a project, identifying key actors during the requirements definition and design phases is crucial to determine who is most affected or most influential, and thus whose feedback is most critical. Performing a stakeholder analysis is therefore imperative [52].

2.6.1 Stakeholder Matrix

A stakeholder matrix [01] is a tool used to classify stakeholders in regards to what communication strategy to use, and who to be aware of. Any stakeholder with the ability to influence the project should be accounted for and kept informed to a level of detail corresponding to their influence or preference.

That also meant that the end-user, the target demographic, with no direct influence did not require the same amount of information, and may have been content with a monthly status update, whereas Vitec MV, would probably prefer vastly more frequent updates.
As for the classification of the development team, for any other project, the current classification would be accurate, however, given the unique nature of this project, the influence of the development team was greater.
A notable exclusion from the matrix below (Table 2.6.1), is teachers and school principals, who would have a great deal of influence over the product's success on the market.

High Influence, High Interest *Key Players*	**High Influence, Low Interest** *Keep Informed*
Vitec MV	SDU
Low Influence, High Interest *Keep engaged*	**Low Influence, Low Interest** *Minimal Communication*
The Target Demographic The Developers	

Table 2.6.1
Displays stakeholder interest and influence.

2.7 Risk Management

Risk management is an analytical approach to identifying and mitigating unknowns that might impact a project. By identifying and evaluating potential issues before proceeding with the project process, valuable insights into the feasibility of solutions and a project as a whole become apparent. With this information at hand, a project can be steered clear of some obstacles, saving valuable time and resources.

2.7.1 Identification

Based on the work packages in the Work Breakdown Structure (condensed Table 2.5.1, full Appendix G), a series of risks was identified. A risk is not necessarily detrimental/negative, but any unknown variable [47]. Theoretically, this means that, for any project, there is an infinite amount of risks, e.g. getting a good grade at some nondescript exam, would be a valid risk.
However, in the interest of brevity, only the risks that may require further analysis during 2.7.2 Assessment and 2.7.3 Response have been listed.

ID	Description
R01	**Children** The targeted audience of the product is within a vulnerable minority, and as such is privy to a series of protections concerning Personally Identifiable Information beyond EU regulation 2016/679 (General Data Protection Regulation - or GDPR for short) [33]. The Danish Data Protection Agency (Datatilsynet) adheres to GDPR which was fully adopted into Danish law on the 8th of March 2024, statutory declaration 289 [34].
R02	**Limited Connectivity** Some schools do not have very sophisticated networking solutions and are subject to heavy peak loads (whenever some class is asked to visit some site). Thus any networked asset retrieval (of high-demand assets - such as graphical assets) could impact the user experience by causing hiccups or long loading screens. 1. To be stable and reliable on low-speed, low-broadband Wifi. 2. To be fast enough still, to not cause unnecessary disturbances or halts.

R03	**Limited Compute and Capacity** The target demographic cannot be expected to be a hardware enthusiast, so anything that runs on the client side of the system must be optimized to work in an environment with potentially very limited CPU resources and system RAM.
R04	**Project Scope** If the project scope creeps, some requirements that were “must”’s might be forcibly dropped due to time constraints. Which would leave the final outcome insufficient in the eyes of the stakeholders, but more significantly, the high-influence ones (Vitec MV & SDU).
R05	**Technological Limitations** Despite research, chosen technologies may prove incapable of performing certain functions to achieve desired goals. This would mean either a compromise on functionality or having to change technology during development.
R06	**Failure to Engage Target Audience** The product might, for any reason, fail to engage the target audience due to lacking game design, visual design, content, etc. which would be a fundamental failure of the product as a whole.
R07	**Malfunctions** Any disruptive events impacting the end-user experience. This may be bugs or oversights in the source code of the product and may, in the worst case, disengage the target audience (R06)
R08	**Platform Instability** The product may, for any reason, become unavailable to the end-user periodically or permanently.
R09	**Developer Team Unavailability** The development team, may for any reason (e.g. sickness), become incapacitated and therefore unable to further develop the product.
R10	**Breach of Contract/NDA** Breaching agreements, legally binding or not and intentionally or accidentally, may discontinue cooperation and invalidate the project.
R11	**Invalid Bachelors Project** The project may be insufficient for a bachelor's project according to SDU, thus failing its primary objective.
R12	**Internal Conflict** Developers may disagree on methodology, research, technology, or likewise, which may reduce effectiveness and introduce delays. This may range from inconsequential disputes to irreconcilable deadlocks causing major delays in development.
R13	**Conflict of Interests** The interests of SDU and Vitec MV might be, or become, too incompatible. Which would force the collaboration with Vitec MV to end.
R14	**Fraud & Identity** If the user authorization is insufficient, unlicensed users may access the product or

	commit identity fraud appearing as other users.
R15	**Database Breach/Leak** Malicious actors may try to infiltrate the system and gain access to valuable or sensitive data.
R16	**Integration** Integrating the product with Vitec MV's platform may prove too difficult. This is unacceptable as RBS 1.1 is a Must.
R17	**Multiplayer Leak** Multiplayer opens the door for a series of issues that may lead to leakage of other players' PII such as IP address and the like depending on how it is implemented. The severity of such a leak is exacerbated due to the target demographic.

Table 2.7.1.1
Displays identified risks.

2.7.2 Assessment

The magnitude/ impact of a risk unfolding can be modeled as a function of the severity of the outcome if it happens, and the likelihood that it does. Below a matrix is constituted from these parameters using an abstract scale for both likelihood and severity, where; "very low" likelihood is next to "never", and "very high" is assured.
As for severity; "very low" can be disregarded in most cases, whereas "very high" is disastrous.

Severity Likelihood	**Very Low**	**Low**	**Medium**	**High**	**Very High**
Very Low		R15		R08, R10	R01, R11
Low		R14	R09, R13	R06, R08	R16, R17
Medium		R12	R07	R05	
High		R02	R03		R04
Very High					

Table 2.7.2.1
Risk Severity Matrix. A matrix mapping the likelihood and severity of a risk.

For easier readability, the following (Table 2.7.2.2) is the same matrix, but in a list format including the name and with a computed score equal to the likelihood (%) multiplied by the severity (!).

ID	Name	!	%	Score
R01	Children	5	1	5
R02	Limited Connectivity	2	4	8
R03	Limited Compute and Capacity	3	4	12
R04	Project Scope	5	4	20
R05	Technological Limitations	4	3	12
R06	Failure to Engage Target Audience	4	2	8
R07	Malfunctions	3	3	9
R08	Platform Instability	4	2	8
R09	Developer Team Unavailability	3	2	6
R10	Breach of Contract/NDA	4	1	4
R11	Invalid Bachelors Project	5	1	5
R12	Internal Conflict	2	3	6
R13	Conflict of Interests	3	2	6
R14	Fraud & Identity	2	2	4
R15	Database Breach/Leak	2	1	2
R16	Integration	5	2	10
R17	Multiplayer Leak	5	2	10

Table 2.7.2.2
Risk Assessment Table. A table displaying ranked risks.

2.7.3 Response

Below, is a modified Risk Response Matrix (RRM) as known to [47], containing the pre-planned contingency plans for what to do, or how to prepare, for some risk. Traditionally, an RRM also contains who is accountable for enacting the contingency plan, but given the size of the development team, this was not deemed relevant.
Furthermore, explicit contingency plans have only been made and enacted for risks assessed as above a certain threshold in terms of severity and likelihood: Likelihood at high or above, OR, severity at high or above.

According to [47] each contingency plan also falls into one of 4 categories (slightly reworded):

- **Reduce** the severity of the risk.
- **Avoid** what would “trigger” the risk.
- **Displace** the risk onto another team, or to another point in time.
- **Accept** the risk for what it is.

ID	**Name** Strategy	Plan
R01	**Children** Avoid	Avoid storing any sensitive information in project systems at all, and rely on third-party systems to provide what is needed.
R02	**Limited Connectivity** Reduce	Use techniques like compression and LODs (storing multiple versions of the same asset in lower resolutions).
R03	**Limited Compute and Capacity** Reduce	Analyze available browser technologies (frontend frameworks & libraries) and choose based on performance metrics and benchmarks. Continuously profile the source code and implement further optimizations as required.
R04	**Project Scope** Avoid	Avoid scope creep by readily dropping new ideas or suggestions to stick to the plan and the timeline
R05	**Tech. Limitations** Avoid	Use prototyping to detect possible limits and find alternatives before the main design effort begins, as well as during development to identify problems early.
R06	**Failure to Engage Target Audience** Reduce	Enhance/modify graphics and/or rework game mechanics based on feedback. In the worst case, reevaluate the target audience, the product might have failed for the current one, but might still provide value to another.
R08	**Platform Instability** Avoid	In case of failure, make sure the system is easily deployable on other hosts or at least all services and databases can be rebooted without much effort.
R10	**Breach of Contract/NDA** Avoid	Carefully read, analyze, and understand contractual agreements. Highlight important clauses and sensitive information. Reread before sharing project information with outside parties, such as test subjects. Communicate with other parties if in doubt.
R11	**Invalid Bachelors Project** Reduce	To avoid the project not being accepted, a thorough analysis of the supplied bachelor's project requirements should be performed. A comprehensive list of necessary demonstrations of curriculum methodology and tools should be developed. In case of rejection, detailed notes should be made based on feedback. The list of demonstrations should be reevaluated and reapplied to the project to prepare for reapplication.
R16	**Integration** Reduce	Establish communication with Vitec MV employees related to the platform, to the extent Vitec MV wishes to cooperate on the matter.
R17	**Multiplayer Leak** Avoid	Completely avoid the issue by choosing a multiplayer implementation strategy not vulnerable to this type of leak.

Table 2.7.3.1

Risk Response Matrix. A table outlining responses to the most notable risks.

3 Design

The solution was designed to be a space-themed colony, where users would play minigames at some locations. These would be upgraded as a reward upon completion of a minigame difficulty. The game was designed to only take keyboard inputs to interact with the colony and minigames to increase repetition.
Several technologies were analyzed for the frontend to account for performance on lower-end computers, and several asset management strategies were planned to manage loading time and traffic on school networks. In the interest of iteration and deployment, an asset pipeline was developed to automate what could have been a lot of manual work.
The solution was designed to be capable of supporting any number of languages and the frontend was designed to be served through Vitec MV's existing Angular platform.
It was decided to handle users (in the interest of persistence) and authentication in optional collaboration with Vitec MV's existing authentication service. That way the integration could fail or be incomplete without consequence to the functioning of the solution or further testing.

3.1 Game Design

As a starter, to guide the design process, a plan had to be made for the intended gameplay.
During brainstorming sessions, several game design concepts were aired based on the requirements and discussions. Vitec MV expressed a desire for a "space"-theme; spaceships, space stations, outer-space colonies, etc., and the initial ideas involved this, with inspiration from existing educational games.

As for combining the different requirements, inspiration was drawn from existing multiplayer games - specifically, Helldivers 2, whose multiplayer solution, theme, and other aspects aligned pretty well (although the target audience most definitely does not). Specifically, the multiplayer solution (as of the 13th of December 2025) would see other players gather on one player's ship, from where they would venture out on different missions.

Along those lines, the first game design concept was drafted: An explorable spaceship with activities scattered around the inside, as well as missions down to planets or asteroids to gather resources.
This was however slated for thematic and complexity reasons: One could upgrade parts of a spaceship and have other players' ships join up for multiplayer, but it did not seem as intuitive as the second idea. Additionally, it would introduce significant complexity to a project that was already large in scope.

3.1.1 The Colony

The second idea was to let a user create space colonies, and on these, upgrade certain locations by playing minigames of varying difficulty. Locations like an Aquifer Plant, Fort, Spaceport, etc. whereof some would be special, and the rest just facilitating different challenges. The Spaceport location, for instance, was determined to be how a user might initialize multiplayer and let their friends join.

A story also guided the game design process: The player was sent to aid some colony rebuild after a devastating event. Meaning the aforementioned locations were not being upgraded but rather repaired. Lending to the gameplay primarily being cooperative as competitive aspects were deemed potentially problematic depending on the classroom.

3.1.2 General Concepts

Some purely technical, concepts remained from the initial game design concept:

Keyboard Only
It was deemed detrimental to the users' learning process, to ever have them lift their hands from the keyboard since a core aspect of touch typing is to get comfortable with the keyboard layout.
That also meant that ever requiring mouse use while playing the game was out of the question, regardless of the mental burden this would put on the user - who most likely otherwise used the mouse for navigation in the browser.

Action Input
With intentionally no mouse support, all current functionality otherwise taken for granted (like buttons) would have to be reinvented. TAB-based navigation is supported in the browser, however, this would do nothing towards furthering the learning process, and thus that would have to be disabled as well.
In a CLI-application fashion, the idea became to have an ever-present input field, in which a user may write keywords in order to trigger buttons that they would have clicked with the mouse before.
Additionally, to provide feedback about what the user currently was able to target, this 'Action Input' would display a series of different symbols indicating what would happen if they used it. For instance, whether they were moving their player character, pressing buttons, or if further inputs had temporarily been disabled (which might happen for any number of reasons).

Path Graph
For both the Spaceship and the Colony, the idea was always to attempt to allow players to walk around as freely as possible. WASD movement is common for games and would not require a mouse, however, this would do nothing to aid the learning process and was slated.
Building from the idea of the Action Input, the chosen solution became a graph-like structure of in-world locations whose names would act as "triggers" and would cause the player character to walk around when written.
Later, this graph structure became enforced, i.e. for path "A->B->C", any player had to go through B at all times to pass from A to C. The original would allow a player to pass from A directly to C if they remembered the name of location C.
This was also done to add a little more repetition, masked behind character movement.

Camera
Since any navigation would be valuable repetition, the idea of a camera following the player, and thus the colony (or spaceship at the time) would extend beyond the viewport, emerged. The intent was to encourage exploration.

3.1.3 Mockups & Storyboards

In the interest of a greater shared understanding between all parties (Vitec MV, SDU & the development team), visual mockups were made of all major elements of the proposed platform alongside key components. Vitec MV had also offered assistance in terms of graphics, but initially required the mockups as a basis.
Additionally, these mockups were arranged into storyboards wherever that made sense - i.e. following the intended user path through the presented content.

All mockups can be found in Appendix H.1-7. Of explicit storyboards are H.3 - Asteroids Minigame, and H.4 - Tutorial. Appendix H.1 - Main Menu & H.2 - Spaceport Location Card, also show intended navigation.

3.2 Client-Side Performance

Many complex operations were scheduled for the DOM: Displaying the colony, player character movement, and handling the gameplay itself. So the choice of supporting tools, libraries, and frameworks could have a big impact on implementational overhead, but also performance.
Duly note that no library or framework is faster than pure, raw, vanilla JS, however, some do come close according to the Krause Benchmarks [12].
Vanilla JS would be directly compatible with the target third-party platform (Vitec MV's Angular platform), however, it was not chosen for several reasons. The main one was the amount of manual work required to obtain:

- Reactivity, i.e. having the DOM reflect the game state without manually replacing DOM contents on any mutation.
- Repaint [43] & Reflow [44] optimizations.

The latter had the highest weight, especially with the plans for the colony layout algorithm (how to display a colony), the camera (displacing all contents continuously), and the path graph at the time, which would cause a lot of updates regardless of its implementation.
Therefore, SolidJS was chosen, as it was the only library to come close to vanilla JS' performance, with extensive support for complex state management, and where some aspects were somewhat familiar to the development team (component structure and JSX/TSX templating).

3.3 Multiplayer

The requirements for the multiplayer aspect of this project were not comprehensive, however, some design constraints did emerge due to the risks and circumstances.

3.3.1 Real-Time

For clarity, and the remainder of this report: "In real-time" or something "being real-time" will be used to refer to any "real-time" update being handled fast enough internally across server and client, that only the time from server to client (ping) may cause deviation that is noticeable by players. Given that real-time multiplayer and the only explicit constraint was from the Project Overview Statement (Figure 2.3.1), which gave an entire second in success criteria three, this definition was not given further thought nor tested and enforced.

3.3.2 Architecture

The simplest form of architecture that could support some kind of multiplayer is arguably peer-to-peer (P2P), where any amount of clients would directly, or in a structured fashion (structured P2P) message each other.
In the context of this project, that option was not available, due to the mitigation strategy to risk R17 (Multiplayer Leak). Relying on P2P would require each client to have an address for each other client, or determine one client (probably the Colony Owner) to have the addresses of all other clients and act as a server. That is unless all users were forced to employ VPNs or proxies, which is assumed to be outside the scope of the target demographic.
Not to mention whether or not enough browsers adequately support some web pages requesting to host a server.
Additionally, P2P is also affected by the characteristics of each client involved, so given the expected variance of hardware capabilities of users' devices, P2P could prove unreliable.

In short, any kind of architecture would have to route through a dedicated server that could keep the identity of all other clients anonymous. This introduces a single point of failure (if the centralized server goes down, so does all communication amongst clients), but also grants the possibility for authorization checks, message verification, and game simulation (running a clone of the game to verify user actions and to track upcoming events etc).

This centralized server would act as a broker between all clients, but also (in the case of game simulation) provide the single source of truth for any game state, as no client could be considered fully trustworthy. Although the requirements or other designs do not expect malicious actors or account for them, unforeseen events, bugs or other unexpected behavior might occur anyway.

As for the specific designs for this service, it would have to provide a series of endpoints:

1. Allowing a new multiplayer lobby to be created, alongside providing information about this lobby such as for what colony it has been made. This would also have to return some kind of ID for the lobby to be able to track it later and allow other clients to join.
2. Joining a lobby, somehow causes a protocol change over to WS or WSS (TCP/IP-based) for further near real-time communications. (See 3.3.4 State Management below for why a TCP-based protocol was chosen)
3. Obtaining the current state of a lobby, so that new clients joining may correctly display the positions of other connected clients and other information to the user.

The initial designs also called for the multiplayer backend's location (IP address) to be unknown to the frontend. Forcing lobby creation to be facilitated by the main backend, which would support any amount of multiplayer servers, by having the eventual address provided on request rather than hardcoded.
Scaling the multiplayer server horizontally was dropped during the design phase, but the idea of keeping the multiplayer server unknown unless a lobby was requested, was kept.
That meant that to open a colony, i.e. make a new multiplayer lobby and make it accessible to other players, the general series of events should follow this sequence diagram.

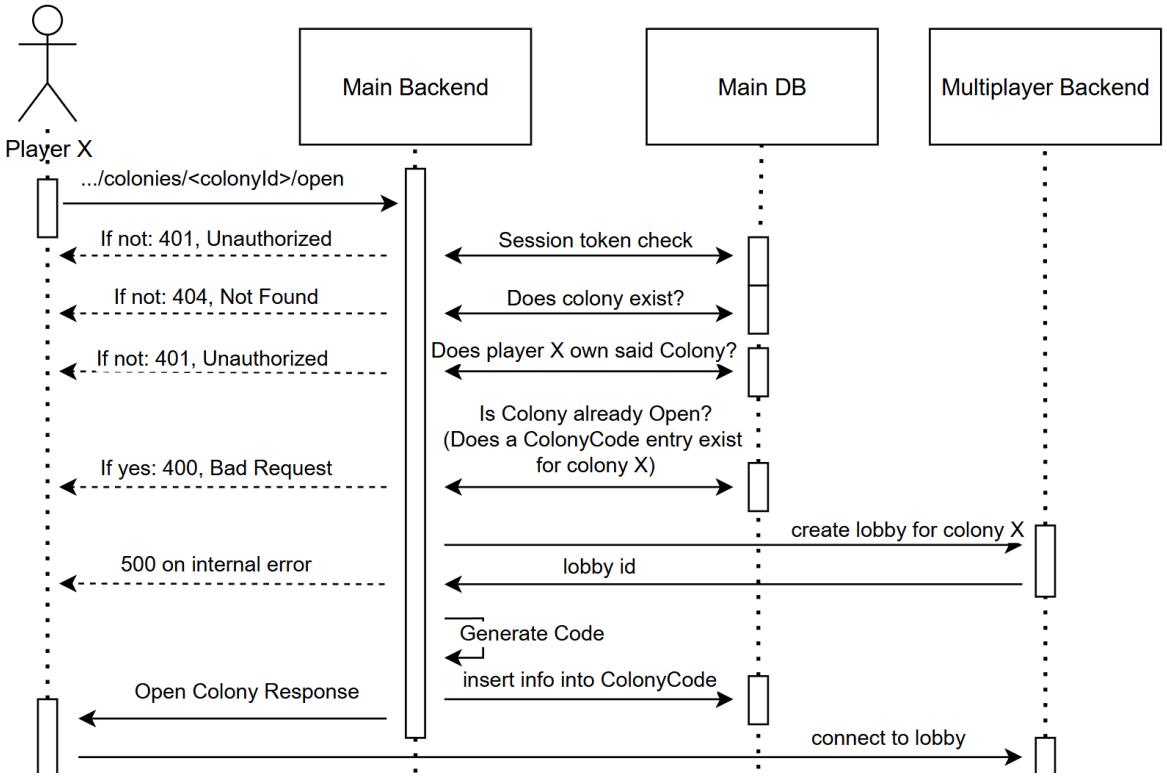


Figure 3.3.2.1

Open Colony Sequence Diagram. The sequence of events involved in opening a colony (Appendix K)

3.3.3 Agnostic Internal Frontend Architecture

In the pursuit of efficient use of system resources, as well as a more reliable end product, it would be detrimental to require the multiplayer backend to be involved for any player visiting any colony.

Therefore an abstraction layer had to be established between any incoming events originating from the multiplayer solution, and the internal consumers/handlers of those events in the client-side frontend. This would allow for an internal service to take over while in "single-player mode" (henceforth referred to as some colony being in a "closed" state) and to be switched off when "opening" the colony to multiplayer.

Additionally, having the frontend be internally unaware of whether the multiplayer solution is in play or not, would make the transition between the two, fundamentally different states, be seamless to the end user - i.e. indistinguishable.

The chosen approach was something between a broker design pattern [39] and a publish-subscribe pattern [40].

Traditionally, specifically a Pub/Sub pattern, has been seen as its own program, acting as a server handling message distribution amongst multiple clients (subscribers) and servers (publishers). One example of an existing solution would be Apache Kafka [41].

For this project, whether the eventual abstraction was closer to a Broker- or a Pub/Sub-pattern, is not that important.

However, in the interest of simplicity & reliability, any consuming component in the frontend should be forced to explicitly declare specifically what kind of / type of message they were consuming for maintenance and debugging reasons.

3.3.4 State Management

A series of issues arose when trying to reliably detect and control state across multiple servers.
The main issue facing this project was to detect when a colony should “close”. As in, closing the multiplayer session and lobby to prevent new clients from connecting or further modification/mutation by other connected clients.
The main problem was that the frontend was to be run in the browser, and any user may, at any time: Close the site, reload the site, leave the colony, access the currently interpreted source code, open the site in any other browser, etc.

Therefore it had been deemed impossible to reliably detect client-side.
However, one possible solution would be to handle it client-side whenever possible, or otherwise allow the multiplayer service to be the single source of truth [42].
One advantage the multiplayer service had was that the user could not directly interfere, so if a TCP-based connection was closed, the multiplayer service could be allowed to interpret that as when to close the lobby.
This did come with the caveat that the multiplayer service would then need awareness of the main backend service, to replicate this change to the database. Alternatively, it could have access to the database directly itself, which would require the multiplayer service to be on the same network and be given the required permissions. The latter was determined to be a less viable solution, due to the implementation overhead and flexibility issues

That decision also meant that there would then be a 2-way dependency between the services which would potentially complicate deployment and reliability:

1. The main backend would need access to the multiplayer backend to open lobbies.
2. The multiplayer backend would need access to the main backend to correctly close a colony.

This could be resolved by diverting colony management to a third microservice or likewise subtracting other areas of responsibility from the main backend. However, the current design was deemed excusable in the interest of time and simplicity.

3.3.5 Verification & Authorization of Messages

The centralized multiplayer backend architectural approach would also allow for trustworthy verification of message content & authorization of the sender (the origin of a message).
The owner of a colony was meant to be specially privileged, which could be enforced. As for verification of message content, given that the intent was for raw, binary messages, at least the message length could always be verified so as to not have malformed messages relayed to other clients.
Which would otherwise, potentially, cause one problematic message to get replicated to the other clients in the lobby, who, in the worst case, would crash. Or worse yet, crash the multiplayer server.

Admittedly, even with centralized verification of all messages to its furthest extent, the possibility of disaster is never zero.

3.3.6 Message Format

When deciding on how exactly the frontend and the multiplayer backend would communicate, 3 approaches were found:

1. Send encoded JSON strings.
2. Find an existing, schema-based, binary messaging solution.
3. Design a binary message format.

While JSON would most definitely be the easiest, the WebSocket technologies investigated at the time already had support for sending a bunch of text back and forth, the overhead was rather significant.

Imagine a message carrying a player ID and the ID of the message type. If the IDs mentioned were of type uint32, the message would be 8 bytes if written in binary. However, had it been JSON instead:

```
{"playerId": 1, "messageId": 1}
```

It would be 31 bytes. Give or take a few depending on the names of each key.

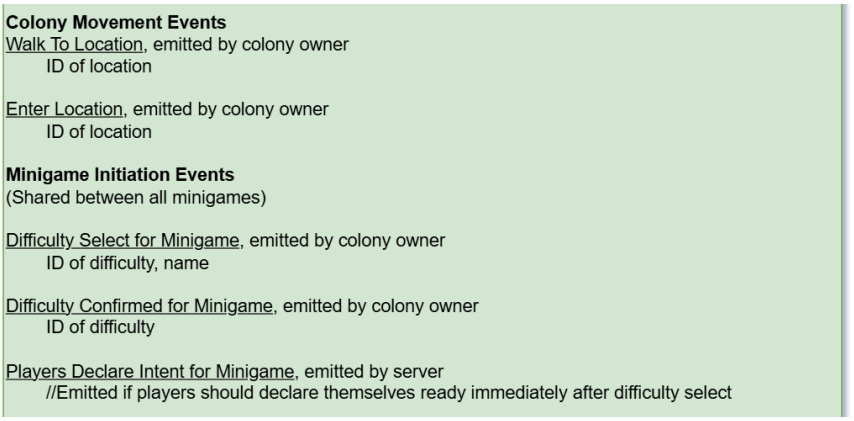
In other words, JSON would introduce significant overhead per message, which would be even worse depending on the amount of keys in each object. Given the latency-sensitive nature of the application intended for lesser or strained networks, that would not be acceptable.

This led to a process of analyzing existing solutions, which provided no results before it was too late (see Discussion section 6.5 Binary Messaging).

The conclusion of the endeavor remains that it was impossible to eliminate the overhead, unless the recipient already knew the complete structure of the message, the absolute byte offsets of each element, the types of each element, etc., beforehand.

So it was decided to define a generic schema structure that could describe any message and could be made compatible with the frontend. These schemas became known as “event specifications”.

Not much was designed as to how these event specifications would look but for events that had already been discussed. These were named and listed including some of the data they would have to include.



Colony Movement Events
Walk To Location, emitted by colony owner
ID of location

Enter Location, emitted by colony owner
ID of location

Minigame Initiation Events
(Shared between all minigames)

Difficulty Select for Minigame, emitted by colony owner
ID of difficulty, name

Difficulty Confirmed for Minigame, emitted by colony owner
ID of difficulty

Players Declare Intent for Minigame, emitted by server
//Emitted if players should declare themselves ready immediately after difficulty select

Figure 3.3.6.1
A subsection of Appendix T - Initial Event Specifications

Another design decision was to have an event specification describe the “body” of any message, whereas an ever-present header would include details required for all messages regardless. Furthermore, to prevent what could potentially be a lot of debugging, it was decided that big-endian would be used for everything, always.

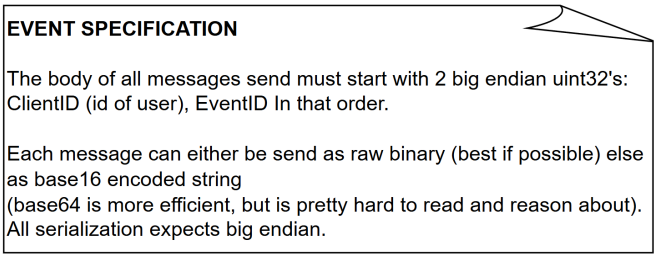


Figure 3.3.6.2
A subsection of Appendix Q - Multiplayer Backend API Specification

To ease debugging, the idea of supporting different encodings is mentioned in *Figure 3.3.6.2*, since most humans would probably be better at spotting an offset error or encoding error in base16 rather than raw binary.

3.4 Multi-Language Support

Many internationalization and localization approaches exist, commonly referred to as “i18n”. The difference between the two is the level of abstraction. “Localization” refers to the act of translating content, currencies, and characters according to user conditions, and internationalization refers to the abstractions that facilitate this translation according to W3C [54].
Some major frameworks already provide one and/or the other, however, the chosen technologies for this project do not (as of December 2024). Therefore a custom solution for both was needed.

3.4.1 Internationalization

To facilitate translations in the first place, the simplest approach was evaluated to be a table mapping some key and a language code to some translation (a text sentence).
The key itself was not important, however, it had to be unique, so for ease of implementation, it was kept abstract. That also meant that in any source code, what text was displayed would get obscure depending on the name given to the key used.
One competing solution would be to use a rosetta language instead (i.e. having the key be the English or Danish translation itself), which would leave the source code clearer and the functionality more convenient.
However, one problem was found in this approach: Depending on the use case and context, one sentence might not always have the same translation in other languages. To mitigate this, the context or use case would have to be part of the key as well, at which point the convenience is lost to some degree.

The translation table was mapped to an ER diagram with database usage in mind.

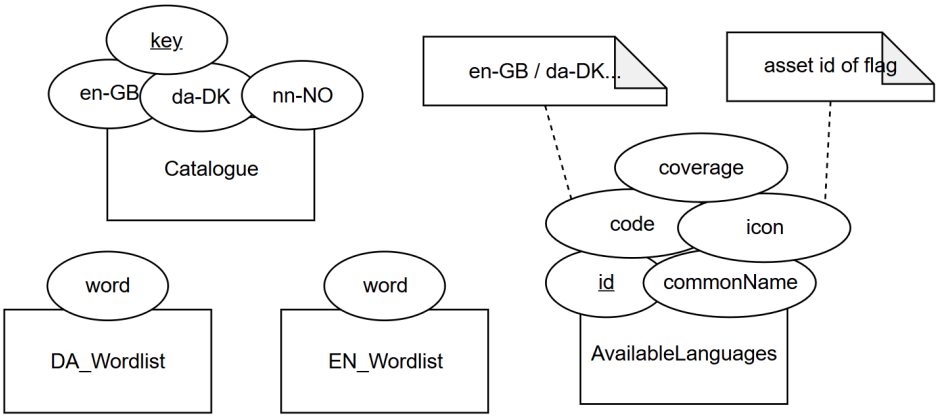


Figure 3.4.1.1
Language DB ER Diagram (Appendix L). A diagram displaying entities and relations in a database.

Placing this translation table (seen as the "Catalogue" entity) in a database would be the first step for Requirement 2.1.1, as well as extending endpoints from the main backend service, for retrieving different translations would fulfill Requirement 2.1.3.
Furthermore, any words used in the Catalogue could be gathered in word lists - which were key to some minigame concepts also considered.
This relationship could be expressed as a many-to-one relation between a word list and the Catalogue, however, the Catalogue was meant to contain complete sentences. They may happen to be singular words but are not necessarily, so any traditional relation between the entities might cause confusion as to the intent.
Additionally, it would be possible to detect exactly how complete the translation of the product was for each language and make all sufficiently covered languages available for user choice automatically.

3.4.2 Localization

The only thing missing for full support of any amount of languages would be the final components handling the adaptation to the user.
Traditionally, for a non-"Single Page Application" (SPA [55]) webpage, the user will select some language, and the webpage reloads and is then translated. For almost all websites facilitated through server-side templating [56], the backing template is simply re-rendered server-side with the new information available and then presented.
In this case, the served content will be served through a third-party platform and would have to be able to update itself "in place" SPA-style.
Designing such a dynamic service for the frontend was approachable since the keys were decided to be immutable (constant). Any retrieved catalog of translations could be loaded into a map of one observable per key. I.e. some managed value that may inform any amount of "observers" when the value is changed, given that the "observers" register themselves on the observable in some fashion. Something that SolidJS (the chosen technology for the frontend) has built-in support for.
This internal services' API would then only need to include some method to retrieve the observables by key and one to set the language. Setting the language should then cause a new catalog to be retrieved from the main backend service and replace all current values in the observables.

3.5 Asset Management

It became clear early in the design process that a lot of image material would be needed, and rather quickly too. Any delay would become very obvious to the end-user, not to mention requirements 7.1.1 and 7.1.2 concerning client and connectivity complications.
Vitec MV also offered to provide some graphical material at some point down the timeline. With that in mind, it was decided that however any asset was eventually referenced, the id or name would be constant, so that some asset could be replaced “in place” without further modifications to any source code.

3.5.1 Pipeline

To manage this image content and to decrease the amount of overhead of handling it manually during development, an asset pipeline was needed.
Admittedly, building an entire asset pipeline was quite the implementational overhead, however, the time it would take to spin the entire system up and down for integration testing without it, as well as deploying on other platforms, was deemed significantly greater.
One strategy for tackling image content the pipeline would automate, besides compression, and scalable vector graphics which was already intended, was using “Level of Detail”’s (LOD’s) - a practice seen in other game development.
The concept does not only extend to image data, but in this context, it boils down to storing multiple copies of the same image in exponentially lower resolutions. The exact equation may vary, but in this project, it followed this equation:

$$ResolutionLOD_{n} = k\frac{1}{2^{(n+1)}}$$

Where k is the resolution of the original, and n is the level of detail.
This correlates to downscaling the image by half in height and width of the previous LOD, so that LOD 0 is the full resolution, LOD 1 is a quarter resolution, LOD 2 is 1/8th resolution … etc.

Notable for this use case, would be the pre-calculation of strong ETags and the ability to target some arbitrary database (or other destination) to load all the image content, LODs, other metadata, etc. automatically. ETags are a requirement for more efficient browser caching [60], and strong ones can be expensive to calculate if not done in advance.

This specialized asset pipeline was named Devour and was built early in the process to be ready for iteration during development. Originally, it was intended for short CLI commands to load different assets individually, but later was expanded to accept a file of specific JSON definitions to load all assets, their positional information, LOD settings, database connection settings, etc. at once.

3.5.2 Colony Structure

In order to provide visual feedback in response to the user's progression the different locations should change their look. In other words, swap what graphical assets were being used to display it in the colony. Therefore it would be ideal to have an abstraction that allowed the definition of any amount of images, SVGs, or other content, to be what described the look of some Location. Another aspect missing was how to place the content to be shown in the colony.

As for having Locations on the colony in the first place, a distinction was made between a "ColonyLocation" entity and a "ColonyAsset" entity. A ColonyAsset was any graphical material with a "top-level" transform (another entity for positional and scale data), but not a Location, and ColonyLocation at a surface level was the same, but with a "level" property and associated minigame (not shown in Figure 3.5.2.1).

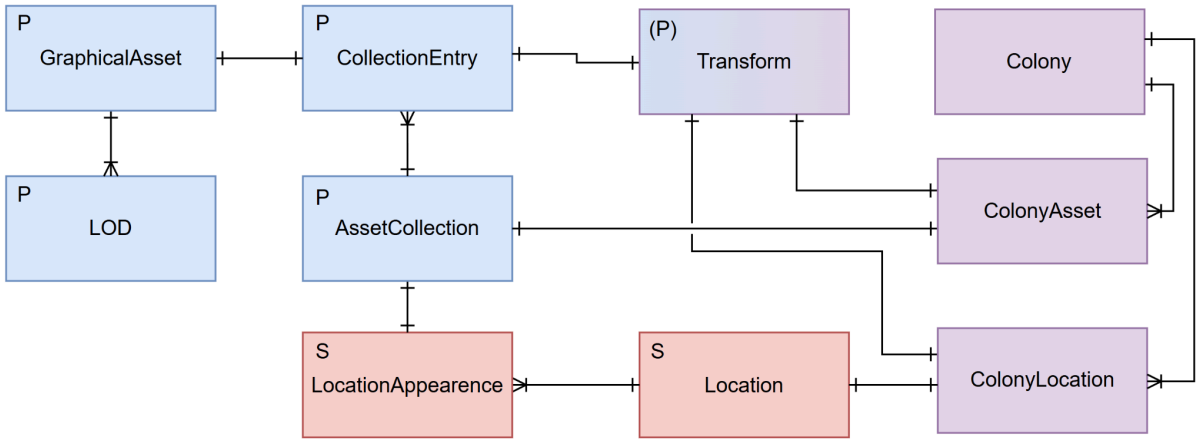


Figure 3.5.2.1
Heavily simplified cutout of Appendix O - Colony & Asset Database
"P" entities are to be handled by the Asset Pipeline
"S" entities are static

A distinction was also made between "static entities", and non-static ones, in an effort to make the eventual implementation and deployment simpler and more reliable. A "static entity" would never change during use, i.e. all instances were immutable (constant), none would be added, and none would be removed. Allegedly, similar concepts exist under the names "reference entity" or "lookup entity", but no concrete source was found.
Following this distinction, as many entities as possible were made static, even though it introduced additional entity relations. For instance: Location and LocationAppearance could technically be combined, however, this would make it impossible to eventually populate them individually, not to mention that all the descriptions and other data would be unnecessarily duplicated.
The Transform entity is a special case and colored slightly differently, as the rest of the purple (non-S and non-P) are marked on the documentation as "unique per colony" - i.e. will be dynamically populated per user-created colony. The difference was that the Transform entity also would be populated by the pipeline to describe the relative position and scale of some CollectionEntry in an AssetCollection.

3.6 Users

To approach top-level requirement 4.1 - “Persistent user progress”, how to handle users needed to be defined. Unlike other projects and products, in this context, any user would be coming from the Vitec MV platform, and already be defined with associated IDs, preferences, and the like.
Preferably, this project would only need a mechanism to tie some Vitec MV user to a user known to this project, so that this project would never have to involve sensitive or otherwise protected data.
Given that neither strict security nor user authentication were requirements for the project, it could theoretically be left out. However, to prevent accidental overwrites of other users’ data as well as be able to facilitate some level of persistence in the first place, some degree of both have to.

One major detail to this puzzle was that this project only had access to the source code for the 10-finger platform, whereas Vitec MV user sessions and authentication were handled by a separate service (“mv-signon” for further reference). To expect, and thus require, as little as possible, it was assumed that no changes or additions could be requested of mv-signon, and so any user authentication, login, etc., for this project could not solely rely on Vitec MV.
Additionally, given the design decisions about handling graphical assets, relying on mv-signon for authentication would cause an extreme increase in traffic per user compared to the current usage caused by say 10-finger: A static site server compared to the potential of the proposed product.
Due to this constraint, it became clear that this project had to facilitate user handling itself, although only what was required for the game to function.

3.6.1 Authentication Cross Check

To be able to require as little as possible, the idea was to provide game-specific sessions, in addition to the one already handled by mv-signon, but do a “cross-check” with mv-signon before granting a “game session”.
This cross-check would simply confirm with mv-signon whether or not the user was a Vitec MV user, based on the information the user provided. The implementation could be done rather flexibly in the main backend service, leaving the possibility open for turning the cross-check off in the case there was not enough time, mv-signon did not support this after further analysis, or both.
Turning the cross-check off would mean that any user, as long as the request to the endpoint was satisfactory, would be implicitly trusted as authorized, but at least all services should work.

The sequence diagram below (Figure 3.6.1.1) describes the expected order of events on any user visit to the product. One detail that it describes, but was later revisited and reevaluated, was hashing the user ID before sending the request to the main backend. Later during the design process, it became clear that mv-signon might need the original identifier, and so the connection to the main backend was planned to be encrypted instead - going from HTTP to HTTPS.

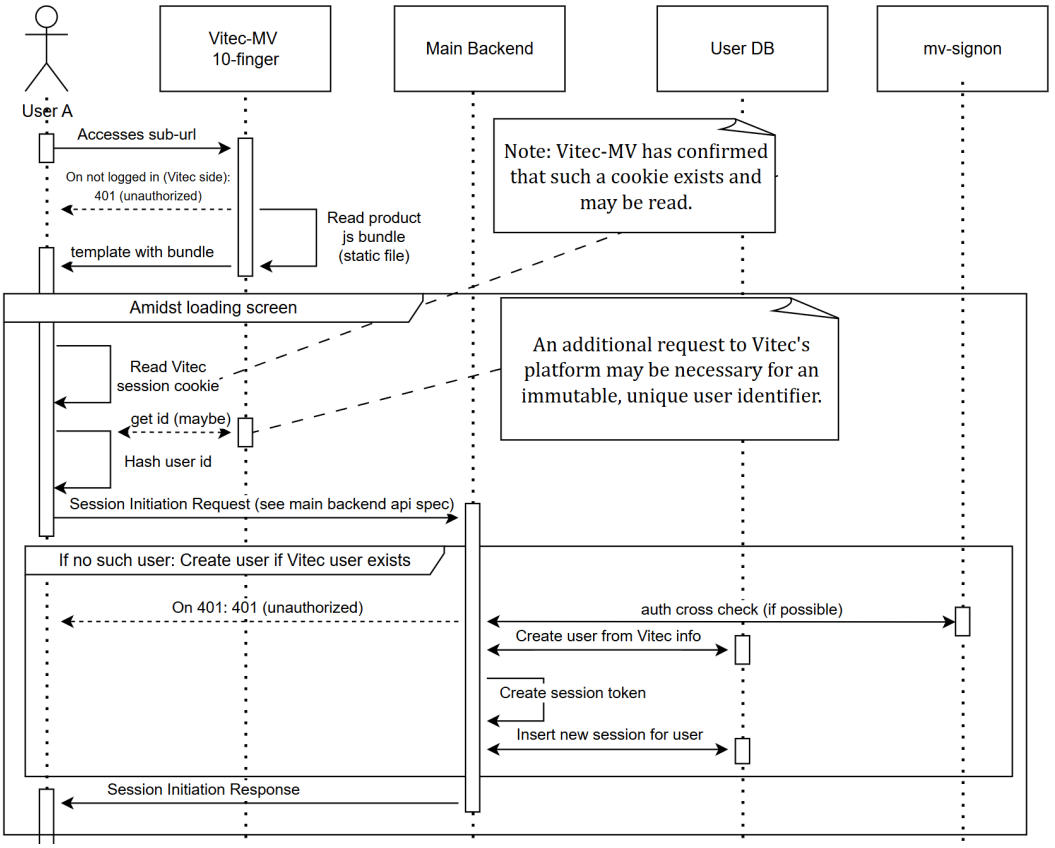


Figure 3.6.1.1
Upon User Visit Sequence Diagram (Appendix N)

3.6.2 Sessions

The aforementioned game-specific sessions only had one constraint: Do not disrupt the user while playing. As in, no session could automatically be invalidated after some period, as that could potentially disrupt the learning process.
With that in mind, the sessions were designed to extend themselves upon use. I.e. a session should only be invalidated after some period with no activity from the user. Where “activity” would be considered any request to any main backend endpoint with some session token.
This approach has some associated security issues, mainly that, if any malicious actor ever obtained a session (either their own or that of another authorized user), it could be kept alive indefinitely. This problem has solutions, e.g. manual invalidation, but this was not included.

3.6.4 Statistics

Judging whether or not a user has improved is difficult, however metrics like:

- **Input Reaction Time**
  How long it took a user to begin typing after they were supposed to type was displayed.
- **Words Per Minute**
  How many words, on average, the user can write per minute.
- **Time Between Inputs**
  How much time passed between two distinct keystrokes.

These metrics could attempt to give an objective score for a user's typing ability. Words per Minute would tell something about their general proficiency, whereas Time Between Inputs may tell something about how well they know their keyboard layout.
The reason these were relevant, was as feedback to the user, and thus a part of making their progress tangible in accordance with requirement 4.1.4 “Integration of user progress tracking…”.

3.7 Minigames

Initially, a set of minigames was discussed, but the ‘Asteroids’ minigame was decided upon since it was simple. The solution requested by Vitec MV was described as an MVP, and this minigame was determined to sufficiently display a minimum viable solution for the scope of the project.

3.7.1 The Asteroids Minigame

The Asteroids minigame was designed with simplicity in mind, not only in terms of implementation but also in terms of gameplay complexity, as it was the first minigame introduced to the solution. Players were to shoot down incoming Asteroids to maintain colony integrity until the danger had passed.
Failing to do so would crack the outer wall, lowering the colony’s stability, eventually faltering and falling apart, in which case the players lost (*Figure 3.7.1.1*). While simple, the game was designed to be visually appealing, with powerful and captivating laser weaponry, intimidating dangers in the form of massive approaching asteroids, and visual feedback on asteroid impact with cracks in the colony wall and asteroid hits with explosions.

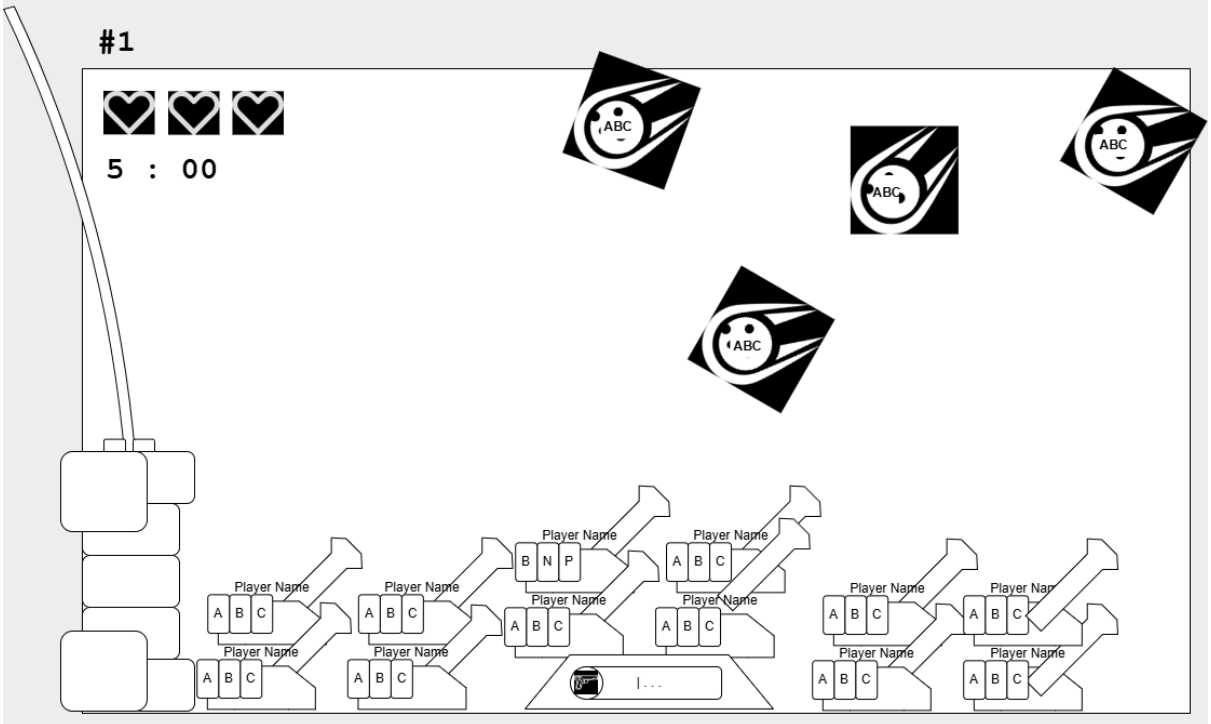


Figure 3.7.1.1
Asteroids Minigame Mock. Illustration of the Asteroids minigame design (appendix H - H.3).

3.7.2 Minigame Initiation Sequence

Given the abstract idea of supporting any amount of minigames within one colony, a generic way to start any one of these had to be defined. The basis became some kind of protocol, i.e. a series of messages, in order, that would be exchanged and synchronize data, as well as ensure that all players involved were informed and ready. The exact sequence of events was revisited many times, but eventually turned out as described in this sequence diagram (*Figure 3.7.2.1*):

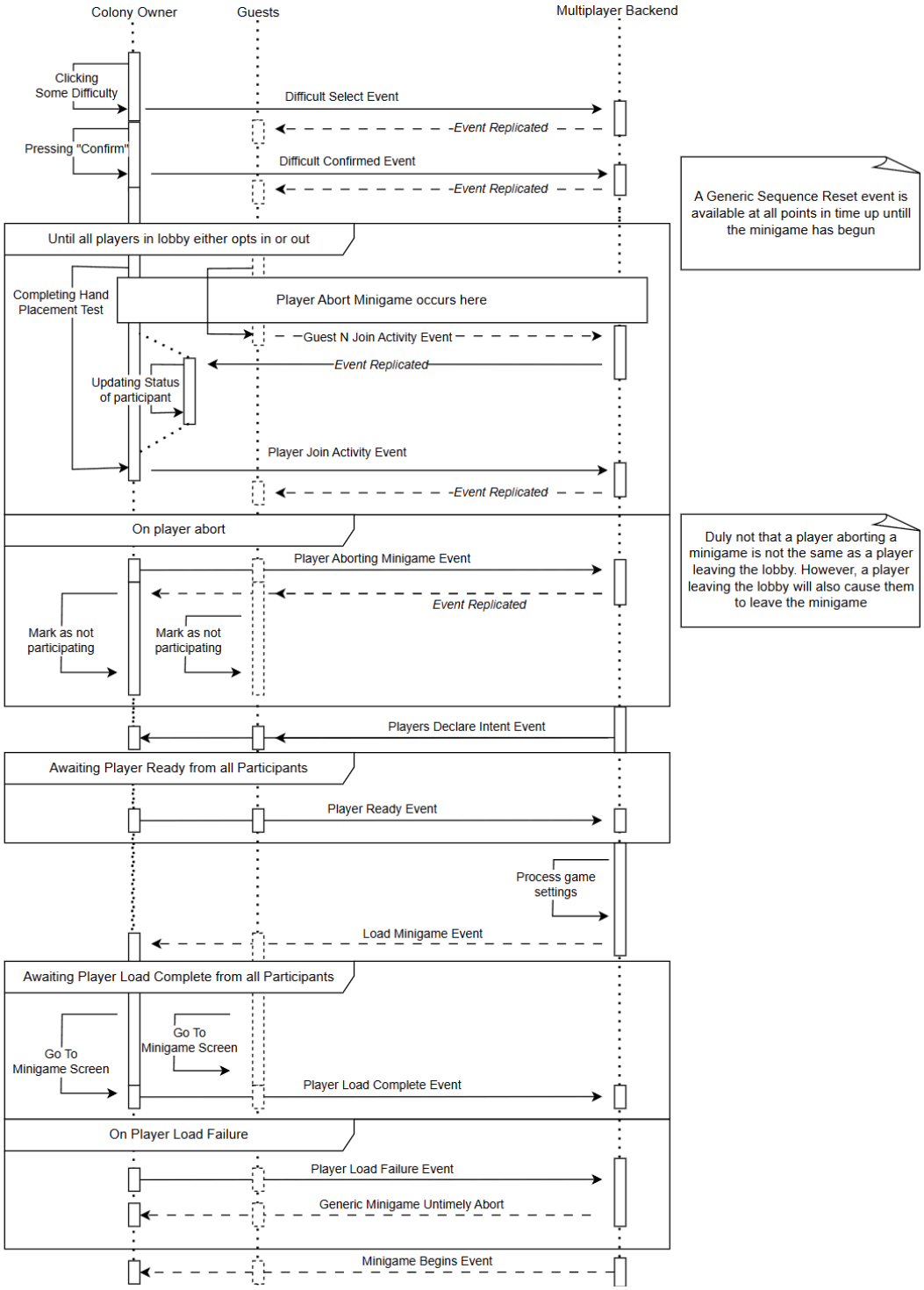


Figure 3.7.2.1
Minigame Initiation Sequence. The sequence of events involved in starting a minigame.

The diagram describes the highly concurrent nature of the process as best as possible, while also showing how events will get replicated to the other clients. From the end-users perspective, it was vastly simpler and was made as a storyboard (appendix H.8 - Minigame Initiation) alongside the other mock-ups made of different parts and sequences. Only when the colony owner had activated “confirm” (through keyboard input) after selecting the difficulty for some minigame, would a prompt be shown to all players:

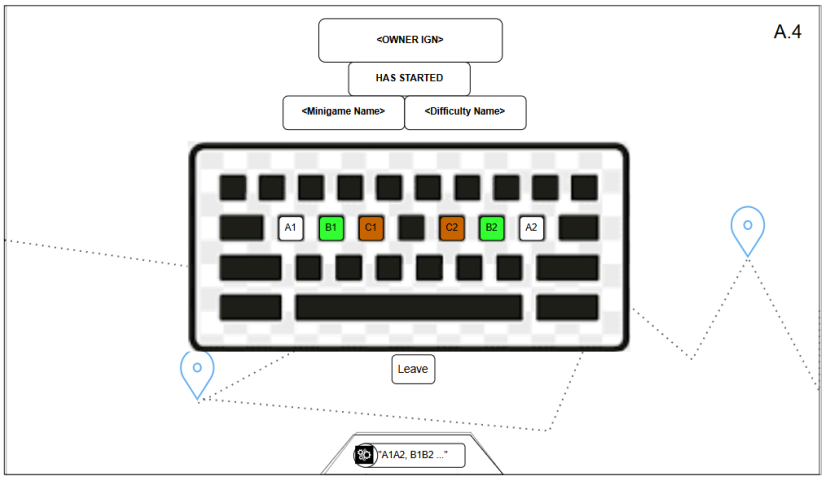


Figure 3.7.2.2

Mock of hand placement check.

Followed by a waiting screen showing the status of the other users currently on that colony.

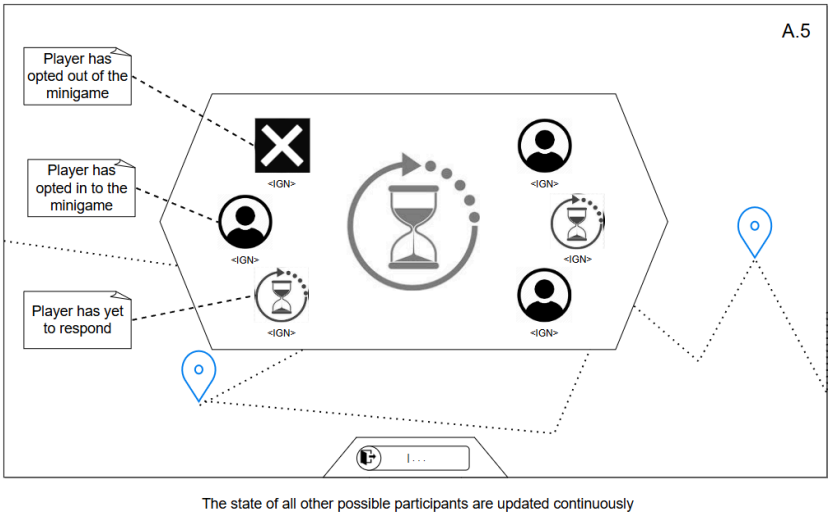


Figure 3.7.2.3

Mock of “waiting for players” screen. Appendix H.8

And as all users opted to either leave or participate, a final loading screen would be shown to ensure all assets were loaded and ready, before the game had begun.

In the single-player case, the same sequence of events would happen, although the waiting screen probably would not ever be visible to humans, as the stand-in server would immediately send the next event in the sequence, telling all clients to begin loading the minigame.

3.7.3 Difficulties

To facilitate requirement 6.1.2 “The minigame(s) must allow users to select different difficulty levels”, some way of generically supporting different parameters for a minigame had to be defined.
These different difficulties could be hardcoded into each game, but the number of distinct services involved across the frontend, its stand-in server for multiplayer, and the multiplayer server, would not make it very maintainable.
Rather, the intended solution was to separate some number of settings from each minigame in all places, extend an endpoint for retrieving these settings, and place them in the database. Then, as the game was about to begin, they could be retrieved wherever necessary (seen as “process game settings” in Figure 3.7.2.1 above). Their structure was intended to be a map-like structure, allowing for the values of keys to be overwritten, so that a minigame might define the default settings, and any changes could be defined and used to overwrite the default.

In data management terms, the structure was defined in this ER Diagram:

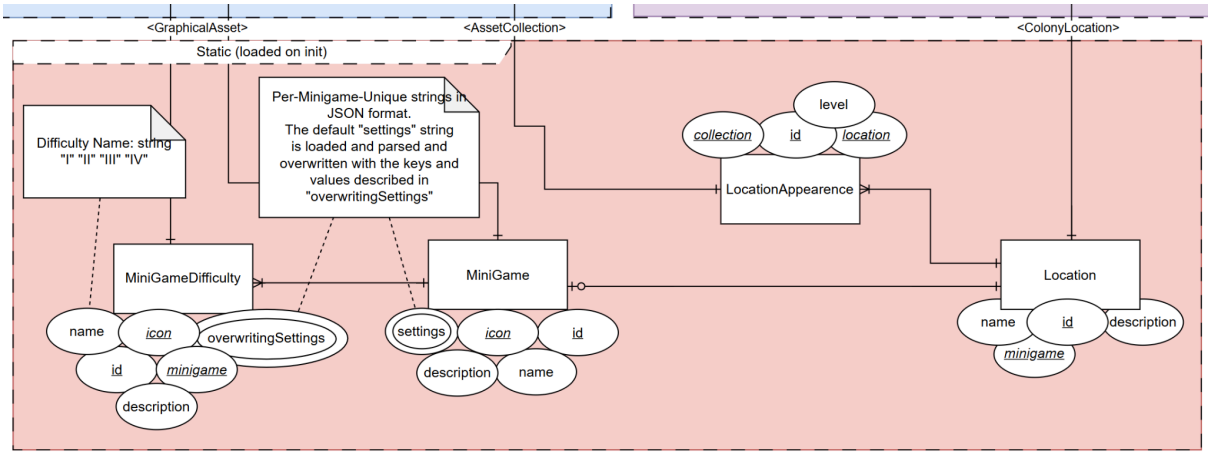


Figure 3.7.3.1
A subsection of Appendix O - Colony & Asset Database

As briefly mentioned in section 3.1.1 (‘The Colony’), some locations intentionally did not have minigames attached but rather served different functionality. Thus the Zero to One relation between the Location and Minigame entities.

3.8 API Design

When the overall, abstract designs for all services, their endpoints, and responsibilities, were beginning to solidify, their API were further defined for future reference and documentation purposes. Some general strategies like API versioning and a default debug header were also defined.

3.8.1 Specifications

All endpoints were defined for both the multiplayer backend and main backend with their URL extensions, path variables, request body Data Transfer Object (DTO [62]) definitions, response body DTOs, and possible error status codes.

The inspiration for this level of design documentation came from OpenAPI (aka Swagger), specifically OAS 3.1.1 [62].
Moreover, these also included examples of the expected SQL required to query the data, if stored as described by the ER Diagrams (Appendix L, Appendix M & Appendix O), although not intended to be the final queries.
The DTO's were defined using a custom TypeScript-based syntax [65], with more precise types like differentiating between signed and unsigned integers. This was done to include more information about the data in question than TypeScript otherwise would allow.

Colony Info Response [AI]

URL: <base-URL>/player/<playerId>/colony/<colonyId>
(playerId, colonyId) =>
Select * Colony WHERE owner is <...> and id is <...> =>

```
{
    "id": uint32,
    "accLevel": uint32,
    "name": string,
    "latestVisit": string,
    "assets": [
        {
            "assetCollectionID": uint32,
            "transform": TransformDTO
        }
    ],
    "locations": [
        {
            "id": uint32,
            "level": uint32,
            "locationID": uint32,
            "transform": TransformDTO
        }
    ]
}
```

Create Colony Resp. [AM]

URL (POST): <base-URL>/player/<playerId>/colony/create

```
{
    "name": string
} =>
{
    "id": uint32,
    "name": string,
    "accLevel": uint32,
    "latestVisit": string
}
```

Colony Overview Resp. [AN]

URL: <base-URL>/player/<playerId>/colonies
(playerId) =>
Select * Colony WHERE owner is <...> =>

```
{
    "colonies": ColonyInfoResponse[]
}
```

Figure 3.8.1.1
Subsection AI, AM & AN of Appendix P - Main Backend API Specifications

Some defaults were defined, like unless otherwise stated, the HTTP method was to be "GET", as well as the "`<base-URL>`" defined in additional documentation included with the models:

General Notes
Described as some function of some inputs, leading to the approximate DB query and response DTO (using expanded TS syntax).
Also includes expected URL extension
"https://host:port/api/vX/<extension>"
All routes, on error may include the header "OTTE-Debug-Header" on the response for further debugging

Auth Notes
Connection is TLS secured
Also all routes (but Initiate Session) expect the header: "URSA-Token" to be present with valid user session token as issued as result of the Initiate Session endpoint.

Figure 3.8.1.2
Subsection Notes of Appendix P - Main Backend API Specifications

This was intended to let the implementation of the backend services and the frontend run in parallel, as the specifications were the single source of truth.

Common DTOs that occurred frequently across different services (like the TransformDTO seen in specification "AI" above) or were defined before the rest, were defined separately and later referenced. E.g. the SessionInitiationDTO, a part of the main backend's API, which was defined before the rest of the main backend API, while analyzing how to handle users.

Transform [AB]

```
{
    "xOffset": float32,
    "yOffset": float32,
    "zIndex": uint32,
    "xScale": float32,
    "yScale": float32,  // duly note that the scale parameters take the
                           place of a width/height parameter if no such
                           exist
}
CLI shortHand: "xOff yOff zIndex, xScale yScale", example:
transform="0f 0f 0, 0f 0f"
```

SessionInitiation [AC]

```
{
    "userIdentifier": string, //vitec id
    "currentSessionToken": string, //Vitec session token
    "IGN": string
}
```

Figure 3.8.1.3
Subsection AB & AC of Appendix R - Known DTOs

The “CLI shorthand” description was made for the asset pipeline.

3.8.2 Versioning

A prefix was added before all endpoints for the main backend to allow changes to existing endpoints without necessarily breaking any integration against the former endpoint.
For example, instead of using: “`/player/<id>/preferences`” directly, the endpoint would have a version and become: “`/api/v1/player/<id>/preferences`”.
In the case that the handling of player preferences would later be changed, the old endpoint and its behavior could remain, and a new “`/api/v2/player/<id>/preferences`” could be added.

Notably, this was not used for the multiplayer backend as it was considered to be complicated enough as is.

3.8.3 Debugging

To make it easier to debug endpoints, a specific header was defined that could be present on all responses from all endpoints of both the main backend and the multiplayer backend.
This would make it easier to include additional information intended for the consumer of said endpoint. For instance, in the case of API versioning, a warning with the date of deprecation of some endpoint.

The header was defined as ‘OTTE-Debug-Header’, ‘OTTE’ was an early, temporary name for the project and was short for ‘Online Touch-Typing Education’.

3.9 Final Solution Design

While designing the different aspects and services of the MVP, an illustration was made to guide the process, which was updated continuously. Much like concept art, it shows the potential outcome as imagined in a production-like environment, what services accessed which other services, and where these could be located.

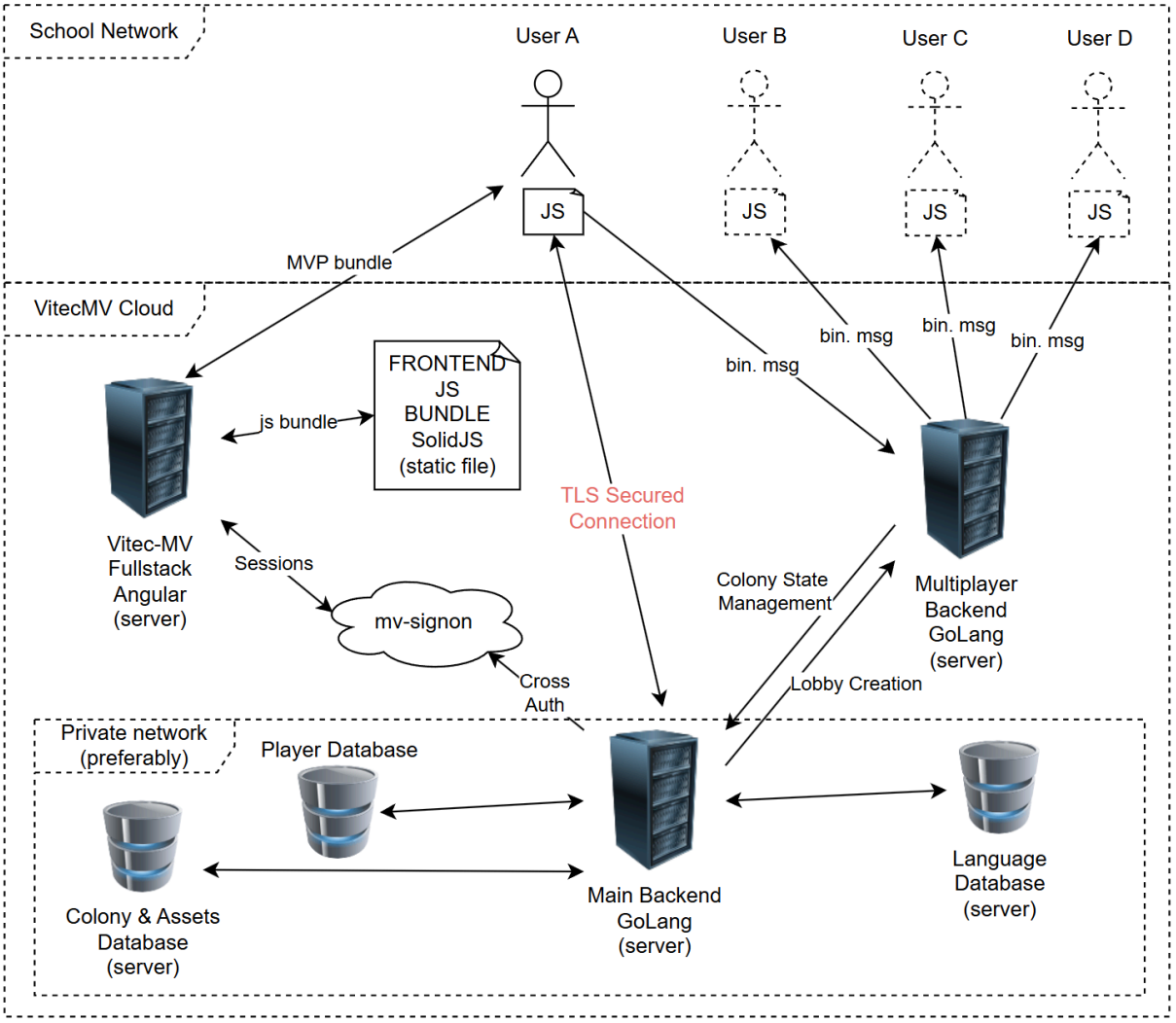


Figure 3.9.1
Appendix S - Simplified Service Overview

The databases were split out of convenience. Practically, nothing would prevent a single database from including everything required, but with the plans for deployment, it would be easier to keep the different areas of responsibility separated.

3.9.1 Deployment

Local deployment for testing purposes was planned to be facilitated through Docker, a containerization service that would, if all services were built with that in mind, allow the services to be spun up on any other machine as well.
Additionally, Docker Compose [66] would allow for automated deployment of all services as long as each service, which could not be found as an existing dedicated docker image, was defined in a Dockerfile. Docker Compose would also allow for centralized handling of environment variables as well as in what order each service would be deployed.
The frontend itself needed no further evaluation as to deployment since it was to be "deployed" through Vitec MV's platform.
Furthermore, some cloud providers also support Docker images as base images for virtual machines in their own way. For instance Google Cloud Platform's Artifact Registry [72].

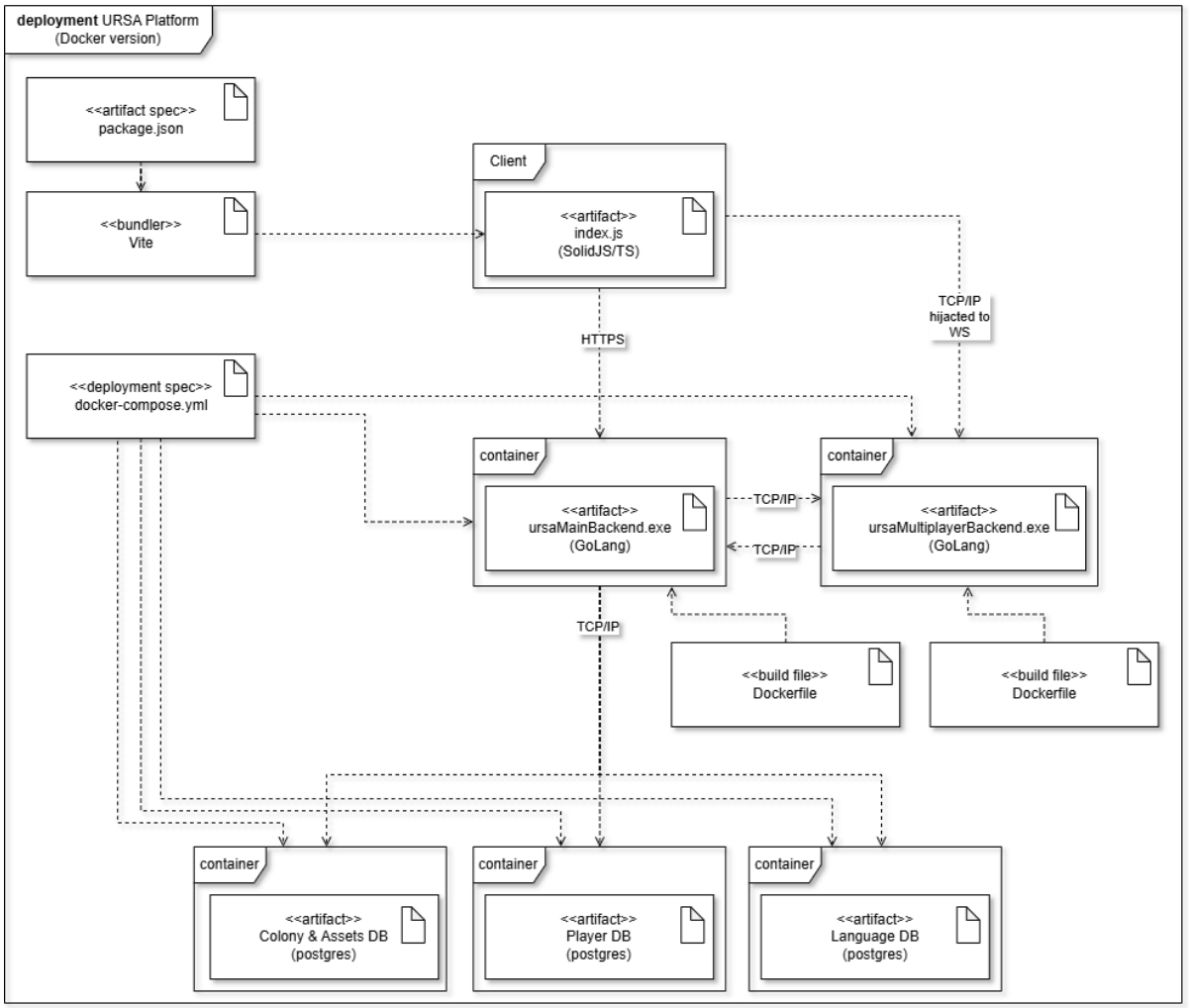


Figure 3.9.1.1
Appendix I - Deployment Diagram

3.10 Proof of Concepts

As the design process was about to settle, a series of proof of concepts were implemented to test certain aspects. In case any had failed, the corresponding designs would have been reevaluated.

Multiplayer Backend
Bun's default Pub/Sub implementation was used as a simple event broker. Adding message package content was used as well, but no further time was spent.

Frontend (Multiplayer)
An integration towards the multiplayer backend (the simplified Bun proof of concept) was built - as in a little display built in SolidJS where received events would be displayed. The two services were spun up locally in a development environment and the test was considered a success as events were successfully received and displayed.

Frontend (Angular Compatibility)
A separate, blank Angular project was established where a single template was defined. The template was empty but for a "div" with id "solidjs-inlay-root".
The aforementioned frontend proof of concept was configured to use some element with id "solidjs-inlay-root" to mount on (where to build from) and was configured to transpile to a single JS file. This required changing the styling approach (CSS) to using a library that would inline the CSS into the existing code, as CSS would otherwise be compiled into separate files (in the prototype, EmotionJS was used, but alternatives were present).
The Angular hook "afterNextRender", which runs any amount of JS after the DOM has been rendered once on the client, was then used to run all JS in the compiled file on the client, effectively using SolidJS through Angular.

Although mounting the actual frontend was slightly delayed with this method, the concept was considered proven.

Main Backend
Notably, the main backend was not prototyped since it was not evaluated to involve any inherent risks that prototyping might be able to address based on the initial API designs. Its main responsibilities were well-known, and the technology involved was mature and well-tested. Any changes, if any at all, would first occur in response to bugs and other unforeseen events during development.

4 Development

The frontend was split into three bundles (Menu, Tutorial, & Colony) to reduce bundle size and establish a separation of concern.
The colony was implemented to dynamically scale distances and dimensions to all screen sizes. SolidJS was utilized to achieve complex state management, as well as to enhance game performance in general. Unplanned roadblocks were encountered and workarounds became necessary to resolve these, such as integration with Vitec MV's platform.
The development environment consisted of Docker containers and deployment scripts. To ensure development aligned with VitechMV's expectations, regular communication was maintained throughout development, and later in development, development logs were shared regularly.

4.1 Strategy

The initial phase of the project focused heavily on research into reliable and suitable technologies to meet the desires of Vitec MV. Extensive planning and documentation were made to model desired components for the solution of the project, using API specifications, sequence diagrams for state management, user interactions, and other interactivity, ER diagrams for all databases, deployment diagrams, and mockups for the colony, colony locations, location cards, minigames, tutorial, etc.
A significant portion of this planning and these diagrams were made upfront, aligning with a non-iterative approach (waterfall) [52], as the project timeline spanned but three months, and in order to achieve all intended goals, most had to be parallelized.
Regardless of this planning, during development, an agile approach (or iterative approach) [53] was adopted, as unforeseen obstacles were expected and as new knowledge became available. Many of the diagrams and plans that were originally constructed were subject to minor modification and extensions, as unforeseen functionality was implemented to support other planned functionality, to resolve encountered roadblocks, and likewise. Essentially, the development strategy utilized was an agile and waterfall hybrid.

4.2 Bundles

During implementation, it became clear that significant amounts of code and components were not used in all areas of the frontend. Moreover, the larger a JS bundle, the slower the page load. Therefore, the frontend was split into three, by configuring the bundler used (Vite) to transpile from three different "entry points", and thus generate three distinct js bundles.

Support for fully modular projects (as known in other languages) has existed in ECMAScript since 2015 [58], but it was not used as the previously mentioned experimental configuration worked as intended.
The development team chose a bundle size target of 170kb, based on a LinkedIn post supposedly quoting a Google study stating that slowdowns usually occurred when a bundle surpassed that size [76].
This constraint was not enforced, but the separation had an impact, as the final bundle sizes came in vastly different (uncompressed): Menu 115kb, Colony 204kb, and Tutorial 156kb.

4.2.1 Naming

There existed no working title for the game itself during much of the process. However, during development, it became increasingly necessary for communication purposes (and for organizing the source code), so one was made: 'U.R.S.A'.
Officially, the name is from the small story behind the gameplay, and stands for 'The United Ressource And Space Administration'. The only reason this segment exists is because the name is consistently used throughout the source code.

4.3 Game Design

The game was implemented in SolidJS, separated into three bundles including the menu, a tutorial, the colony, and with the colony, the Asteroids minigame.

4.3.1 The Menu

The menu implementation consisted of four buttons (*Figure 4.3.1.1*). The "New" button led to a creation page which would redirect the user to the colony bundle when complete. The continue page would show a list of all user colonies, then redirect after selection. The join page forwarded players to a foreign colony using a 6-digit code - like Kahoot, and the tutorial button would redirect to the Tutorial bundle. Notably, the globe icon in the top-most right corner was a language select which expanded on hover.

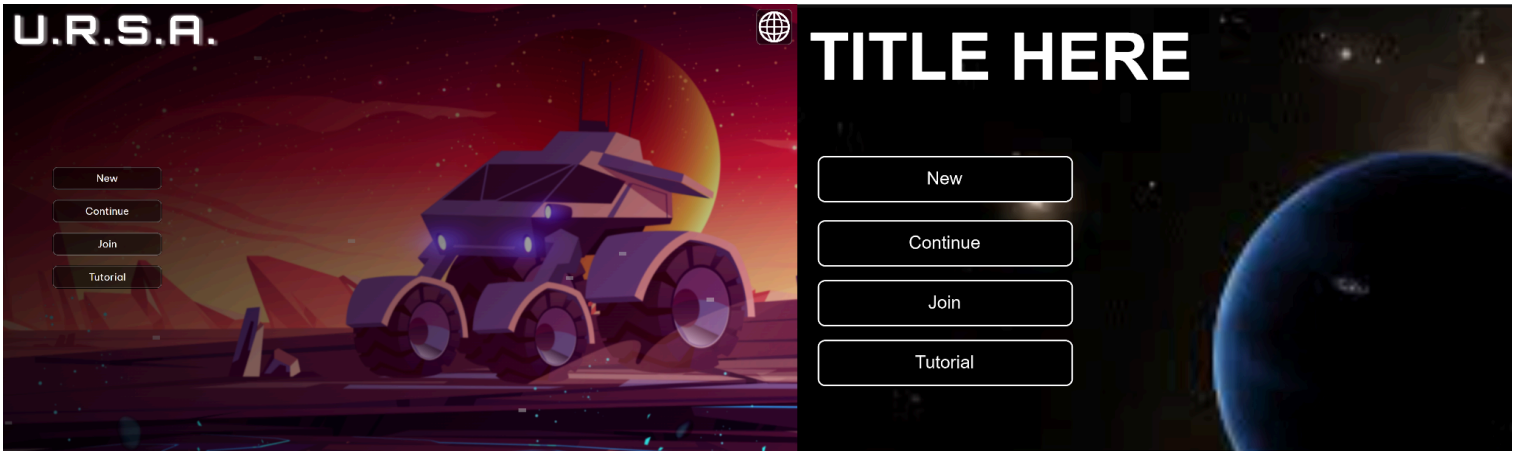


Figure 4.3.1.1
Main Menu. The first page users see (implementation vs mockup, Appendix H.1).

4.3.2 Tutorial

The tutorial aimed at isolating individual gameplay concepts, and presenting them as singular goals, illustrating how they are performed with demo animation, and finally a trial or test in which players had to demonstrate an understanding of the concept to progress (*Figure 4.3.2.1*).

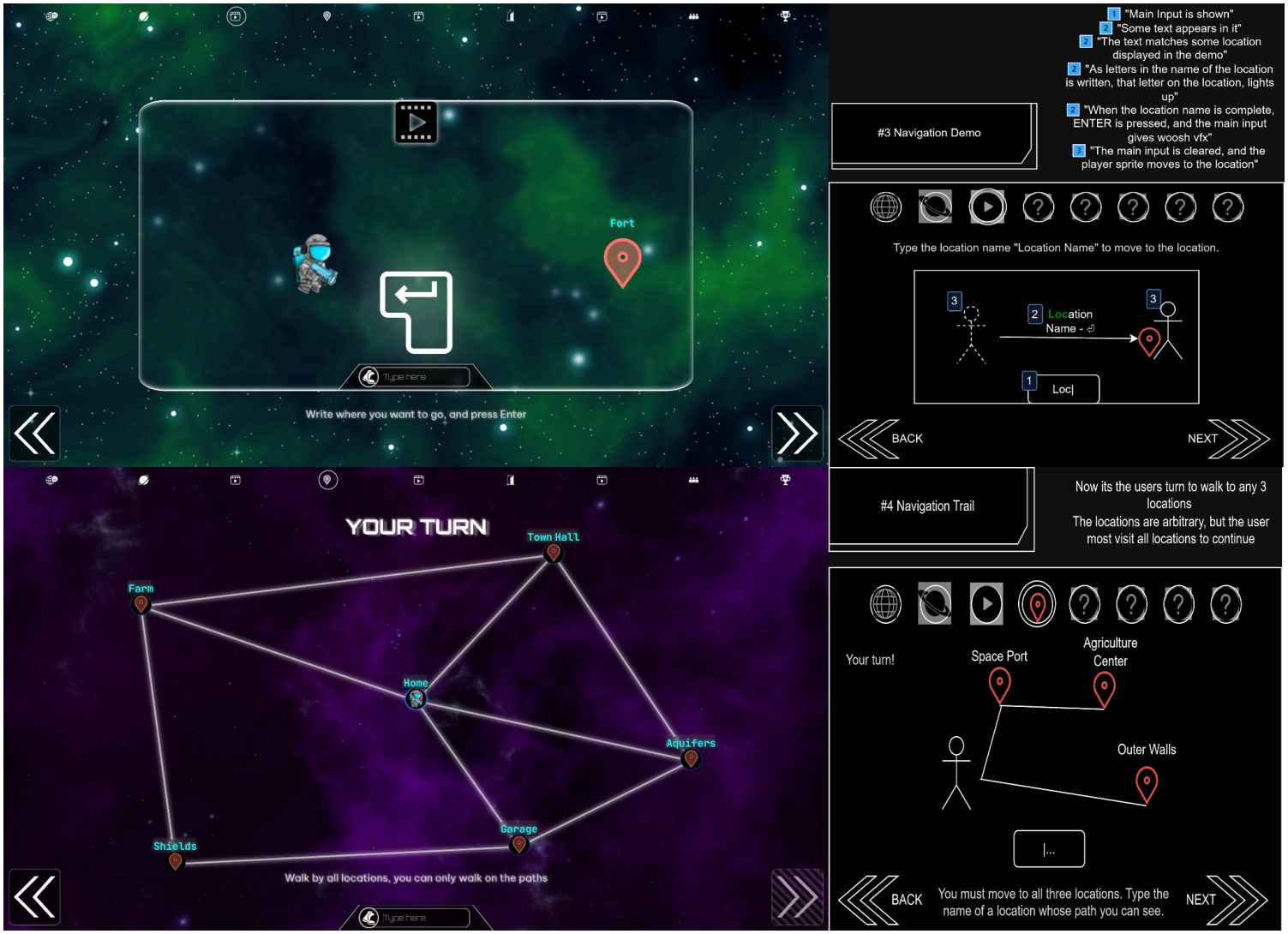


Figure 4.3.2.1
Tutorial. Navigation demo and navigation trial (implementation vs mockup, Appendix H.4).

To mimic the game environment as closely as possible, the tutorial utilized modified mock versions of the components that the colony was composed of, if not the very same.

4.3.3 The Colony

The colony contained a set of locations that could be traveled to and from. A series of paths between them defined how the colony could be traversed. This meant that multiple players could explore and see each other in real-time. This open-world layout algorithm used the transforms of each asset, path, and location. Additionally, some locations had decorations that were local to them, meaning they would have internal transforms. As of the final MVP, these decorations were used both decorationally and to show updated progression to buildings when a minigame had been won.

The layout of locations was hardcoded in the main backend, with a series of functions inserting transforms, ColonyAssets, and ColonyLocations to create a new colony. When fetched, the frontend used a set of scalars to scale image sizes and positions.
These scalars were based on screen size and would recalculate and reapply whenever the window was resized so that the colony layout appeared similar to each client.

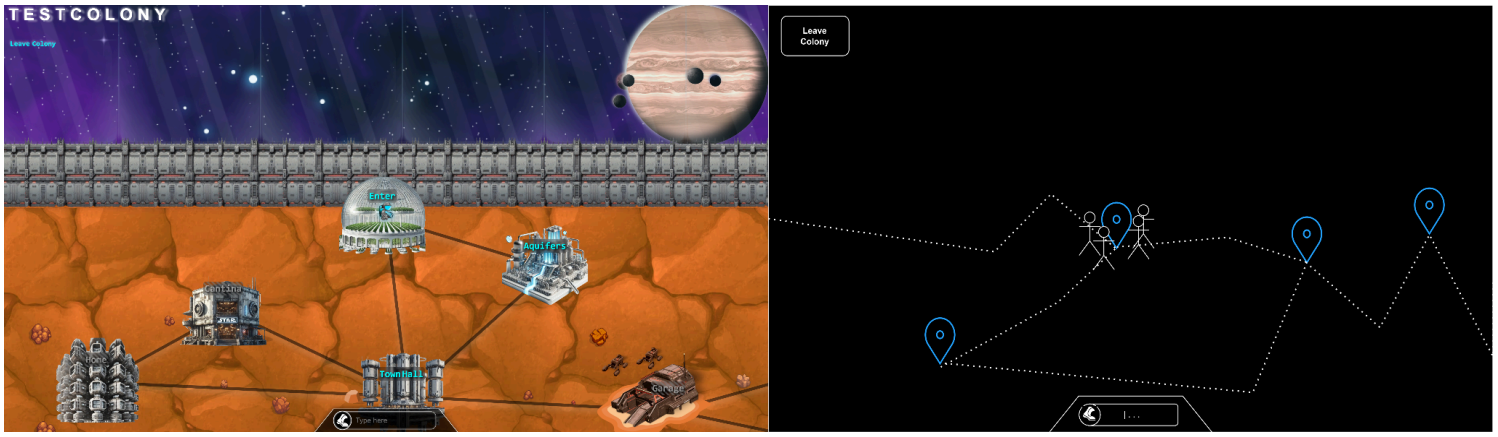


Figure 4.3.3.1
The Colony. The colony in which players navigate and explore to find locations and minigames (implementation vs mockup, Appendix H.8).

4.3.3.1 Frontend Layout Algorithm

To display the colony in the frontend, an algorithm was constructed to interpret the data from the database. Fetching metadata about locations, their asset IDs, positions, and names first, later fetching actual image content asynchronously and progressively, starting at the lowest resolution LOD available.
The main factors to this algorithm were the camera, vector scalars making the display content reactive to screen size, and the transforms and internal transforms of AssetCollections and Locations.

```
42      const colonyLocations = createArrayStore<ColonyLocationInfoWOriginalTransform>(
43          props.colony.locations.map((loc) => ({ ...loc, originalTransform: loc.transform })),
44      );
45      const colonyAssets = createArrayStore<ColonyAssetWOriginalTransform>(
46          props.colony.assets.map(colAss => ({
47              ...colAss,
48              originalTransform: colAss.transform,
49              wrappedTransform: createWrappedSignal(colAss.transform)
50          }))
51      );
```

Image 4.3.3.1.1
bsc-frontend/ursa_frontend/src/components/colony/PathGraph.tsx

In the image (*Image 4.3.3.1.1*) above are the segments of code that created and stored assets and locations reactively. Notably, these reactive signals contain "...OriginalTransform" types (*ursa_frontend/src/components/colony/PathGraphHelpers.ts*), that retain the original transform data fetched from the backend, in the format seen below (*Image 4.3.3.1.2*):

```
293  export type TransformDTO = {
294      xOffset: number;
295      yOffset: number;
296      zIndex: uint32;
297      xScale: number;
298      yScale: number;
299  };
```

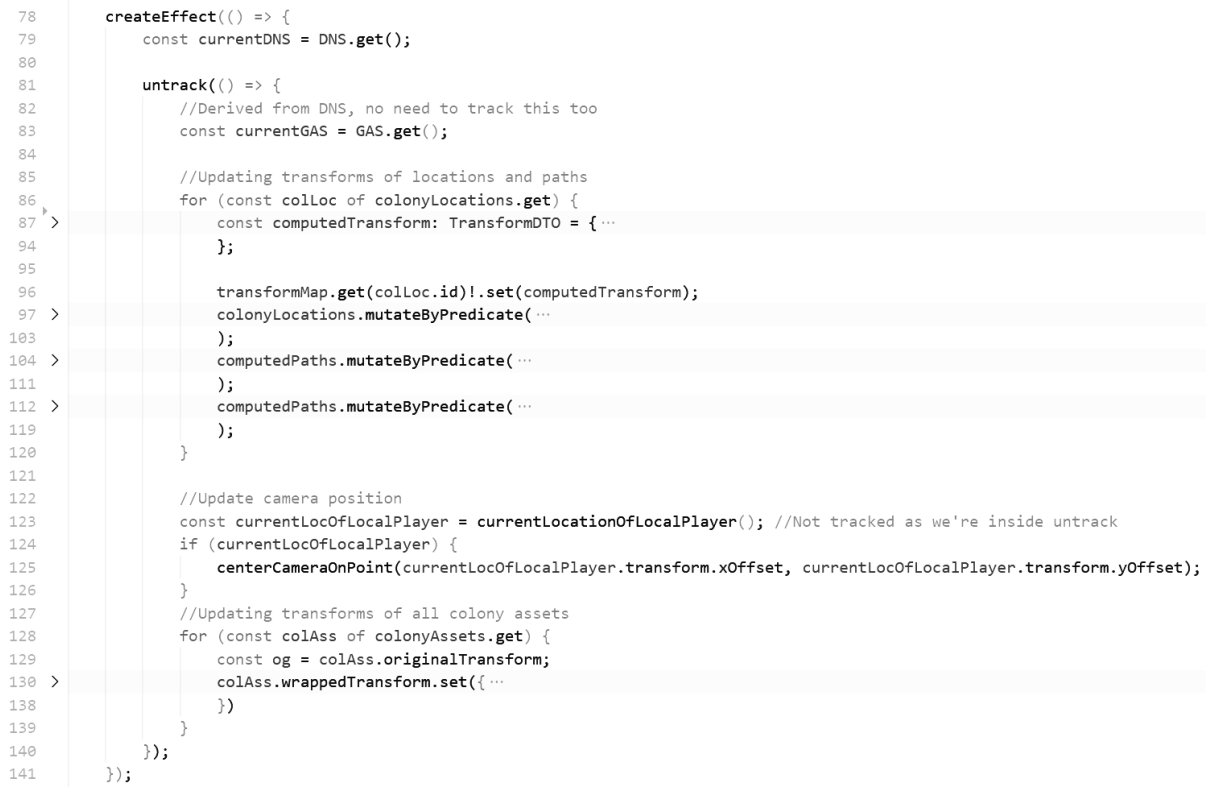
Image 4.3.3.1.2
bsc-frontend/ursa_frontend/src/integrations/main_backend/mainBackendDTOs.ts

This was necessary as recalculating scalings would need the original positions in mind rather than the latest, treating the original positions as a single source of truth.

```
176      const calculateScalars = () => {
177          const newWidth = window.innerWidth;
178          const newHeight = window.innerHeight;
179          setViewportDimensions({ width: newWidth, height: newHeight });
180          const dns = {
181              x: newWidth / EXPECTED_WIDTH,
182              y: newHeight / EXPECTED_HEIGHT,
183          };
184          GAS.set(Math.sqrt(Math.min(dns.x, dns.y)));
185          //DNS triggers re-positioning of all elements, which also uses GAS
186          //but doesn't depend on GAS as to not run everything twice.
187          //Thus, GAS is set first (not triggering re-position), then DNS
188          DNS.set(dns);
189      };
```

Image 4.3.3.1.3
bsc-frontend/ursa_frontend/src/components/colony/PathGraph.tsx

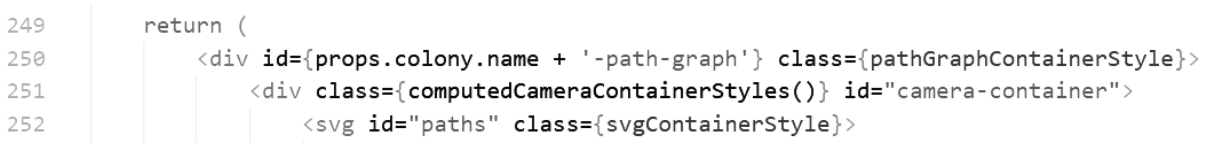
An event listener, listening for window resize events was established, calling the `calculateScalars()` (*Image 4.3.3.1.3*) method when triggered. This method recalculated the new scalars DNS (Vector2, not scalar) and GAS with which the colony world would reform, by updating the transforms of all elements with the new scalars. To prevent too many updates at once, only one update of these values were accepted every 500ms.



```
createEffect(() => {
    const currentDNS = DNS.get();

    untrack(() => {
        //Derived from DNS, no need to track this too
        const currentGAS = GAS.get();

        //Updating transforms of locations and paths
        for (const colLoc of colonyLocations.get) {
            const computedTransform: TransformDTO = {···
            };

            transformMap.get(colLoc.id)!.set(computedTransform);
            colonyLocations.mutateByPredicate(···
            );
            computedPaths.mutateByPredicate(···
            );
            computedPaths.mutateByPredicate(···
            );
        }

        //Update camera position
        const currentLocOfLocalPlayer = currentLocationOfLocalPlayer(); //Not tracked as we're inside untrack
        if (currentLocOfLocalPlayer) {
            centerCameraOnPoint(currentLocOfLocalPlayer.transform.xOffset, currentLocOfLocalPlayer.transform.yOffset);
        }
        //Updating transforms of all colony assets
        for (const colAss of colonyAssets.get) {
            const og = colAss.originalTransform;
            colAss.wrappedTransform.set({···
            })
        }
    });
});
```

Image 4.3.3.1.4
bsc-frontend/ursa_frontend/src/components/colony/PathGraph.tsx

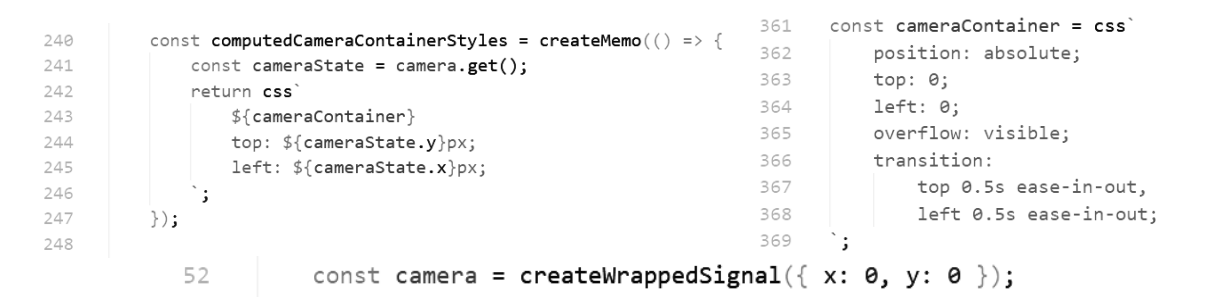
In the image above (*Image 4.3.3.1.4*), the value of DNS was accessed within an 'effect'. "`createEffect`" is a function that tracks all signal (observable-like) invocations using SolidJS' tracking scope [57]. This allows for any additional computations to occur when the value of a signal changes. In this case, the transforms of all assets, locations, and paths are recalculated.



```
return (
    <div id={props.colony.name + '-path-graph'} class={pathGraphContainerStyle}>
        <div class={computedCameraContainerStyles()} id="camera-container">
            <svg id="paths" class={svgContainerStyle}>
```

Image 4.3.3.1.5
bsc-frontend/ursa_frontend/src/components/colony/PathGraph.tsx

All assets, locations, and paths were rendered in the render function of `PathGraph.tsx` component. On line 251 (*Image 4.3.3.1.5*), a specific container wrapped all the aforementioned elements.



```
const computedCameraContainerStyles = createMemo(() => {
    const cameraState = camera.get();
    return css`
        ${cameraContainer}
        top: ${cameraState.y}px;
        left: ${cameraState.x}px;
    `;
});
```

```
const cameraContainer = css`
    position: absolute;
    top: 0;
    left: 0;
    overflow: visible;
    transition:
        top 0.5s ease-in-out,
        left 0.5s ease-in-out;
`;
```

```
const camera = createWrappedSignal({ x: 0, y: 0 });
```

Image 4.3.3.1.6
bsc-frontend/ursa_frontend/src/components/colony/PathGraph.tsx

This container and applied styles, `computedCameraContainerStyles` and `cameraContainer` (*Image 4.3.3.1.6*), essentially functioned as the camera, using the camera signal to displace the container to match the player's perceived position.

4.3.3.2 Colony Locations

The colony locations were meant to contain minigames, which as of the final MVP, only the Fort did. Within these locations was a location card, containing minigame information and initiation UI, if a minigame was present. However, in a few special cases, locations did not contain minigame information, but instead UI related to other functionality. One such location was the Spaceport, which contained UI elements for opening and closing the colony.

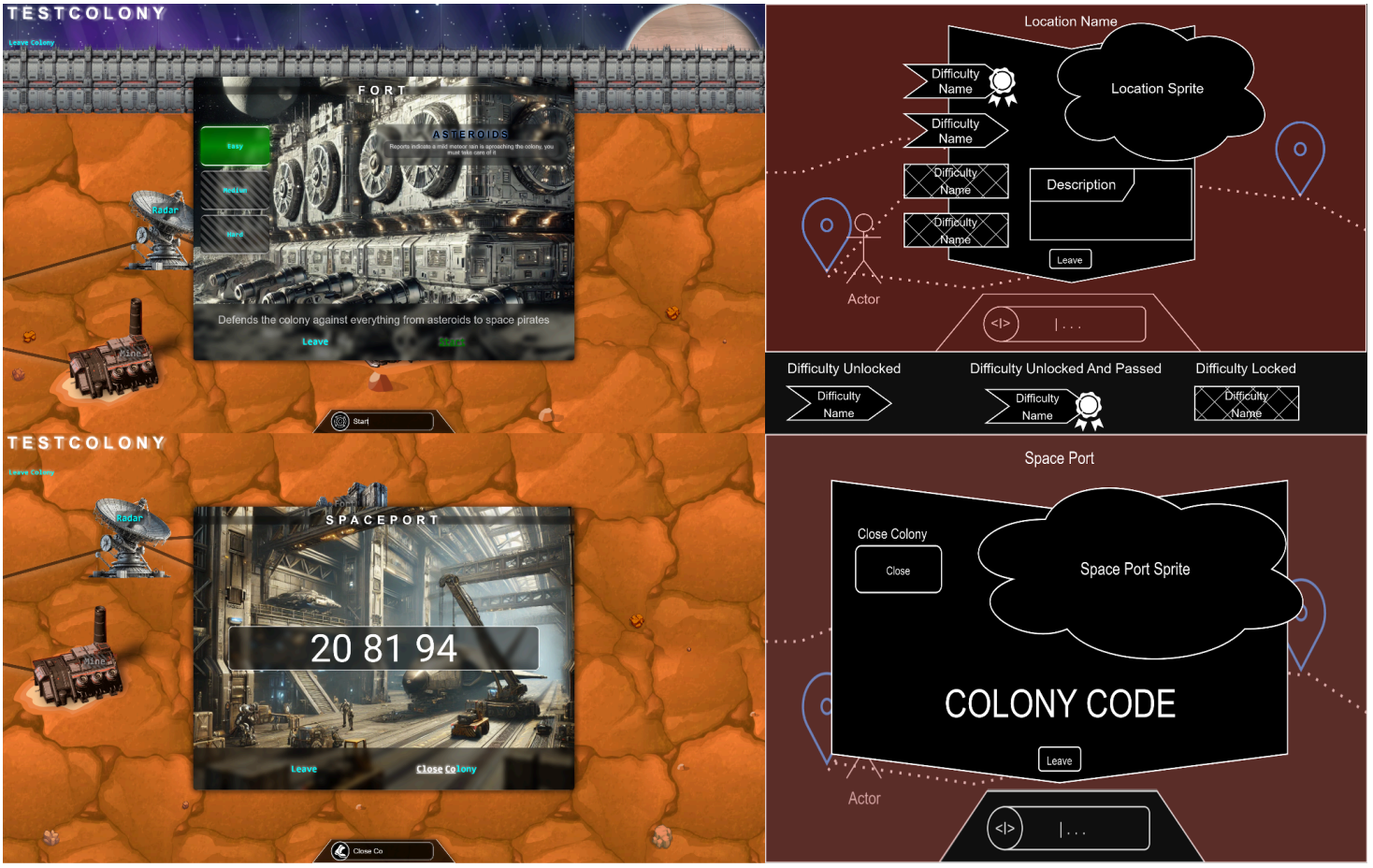


Figure 4.3.3.2.1
Locations cards of colony locations, visible when players enter a location. Appendix H.2 & H.5

4.3.4 The Asteroids Minigame

The Asteroids minigame was separated into three major files, `AsteroidsDisplay-Component.tsx`, `AsteroidsGameLoop.tsx` and `asteroids.go`. The display component was a frontend component that handled the display of the minigame itself, visualizing graphics and game events.
These files were located in:
“bsc-frontend\ursa_frontend\src\components\colony\mini_games\asteroids_mini_game\…” and “BSC-MULTIPLAYER-BACKEND - src\internal\asteroids.go” respectively.

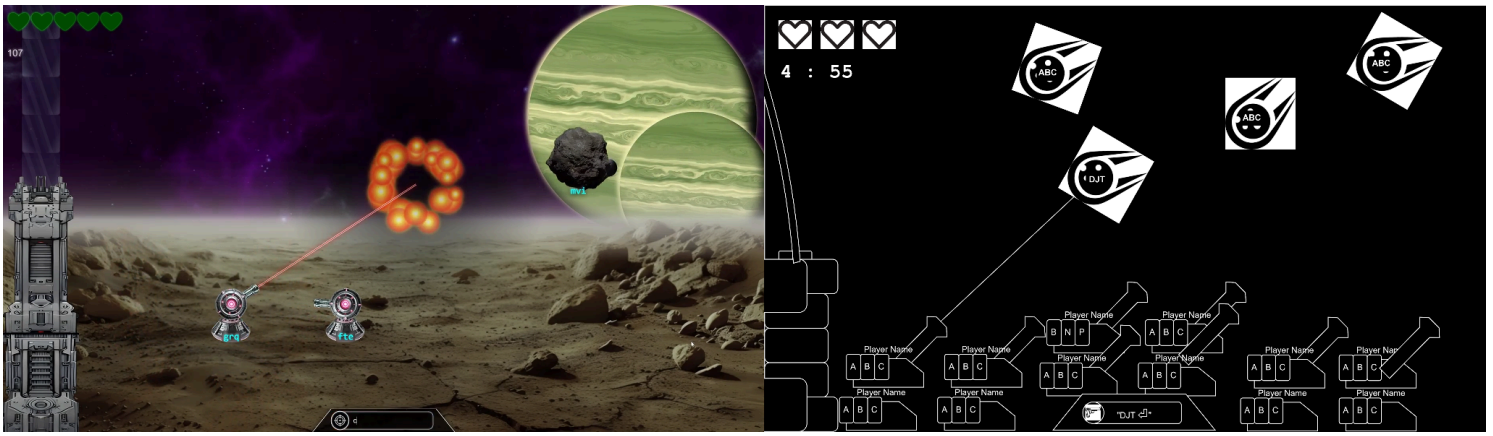


Figure 4.3.4.1
Asteroids Minigame. Players shoot asteroids to protect the integrity of the colony wall (implementation vs mockup, Appendix H.3)

4.4 Minigame Handling

The minigame initiation sequence had been generically implemented, with type-safety and separation of concerns. The implementation utilized predefined interface-like function aliases to allow for extensibility in regard to future minigame implementations (*figure 4.4.1*).

```
/**
 * Any function, that based on the provided ApplicationContext alone, can initialize all data needed
 * for establishing the minigame component. The resulting JSX element should be the root of the minigame,
 * and will be given the entire viewport.
 */
export type MinigameComponentInitFunc = (context: ApplicationContext, difficultyID: uint32)
    => Promise<ResErr<JSX.Element>>;
/**
 * Any function, that however it sees fit, starts the single player game loop.
 * Used by the mock server.
 */
export type GenericGameLoopStartFunction = () => void;
/**
 * Any function, that based on the provided ApplicationContext alone, can initialize all data needed
 * or return an error if something goes wrong during initialization.
 */
export type SingleplayerGameLoopInitFunc =
    (context: ApplicationContext, difficulty: DifficultyConfirmedForMinigameMessageDTO, colonyID: uint32)
    => Promise<ResErr<GenericGameLoopStartFunction>>;
```

Image 4.4.1
bsc-frontend/ursa_frontend/src/components/colony/mini_games/miniGame.tsx

By providing functions complying with these specifications, any minigame could be loaded and run in singleplayer or multiplayer (see *ursa_frontend\src\components\colony\mini_games\-asteroids_mini_game*). Using these generic constraints, the minigame game loop and UI component were loaded using the minigameID, from a list of known minigames defined as enums (*Image 4.4.2*), but could technically also be loaded from the database, reducing the size of the colony bundle.

```
41  export const loadMinigameSingleplayerLoop = (minigameID: number): ResErr<SingleplayerGameLoopInitFunc> => {
42      let gameLoopInitFunc: SingleplayerGameLoopInitFunc | null = null;
43
44      switch (minigameID) {
45          case KnownMinigames.ASTEROIDS: {
46              gameLoopInitFunc = createAsteroidsGameLoop;
47              break;
48          }
49          default: return { res: null, err: 'Could not load minigame gameloop, no loop found for minigame id: ' + minigameID };
50      }
51
52      return { res: gameLoopInitFunc, err: null };
53  };
54
55  export const getMinigameComponentInitFunction = (minigameID: number): ResErr<MinigameComponentInitFunc> => {
56      let initFunc: MinigameComponentInitFunc | null = null;
57
58      switch (minigameID) {
59          case KnownMinigames.ASTEROIDS: {
60              initFunc = initAsteroidsDisplayComponent;
61              break;
62          }
63          default:
64              return { res: null, err: 'Could not find minigame component init function for minigame id: ' + minigameID };
65      }
66
67      return { res: initFunc, err: null };
68  };
```

Image 4.4.2
bsc-frontend/ursa_frontend/src/components/colony/mini_games/miniGame.tsx

In singleplayer, the minigame loop init function, "`SingleplayerGameLoopInitFunc`", was passed predefined minigame-specific settings loaded from the backend by a mock server, using the difficulty parameters selected by the colony owner.
Once fully prepared, the "`MinigameComponentInitFunc`" and "`GenericGameLoopStart-Function`" were used by the mock server, which ran the game loop mimicking the multiplayer backend through the internal event broker implementation as described in section 3.3.3 Agnostic Internal Frontend Architecture.

```
116     private update = async () => {
117         if (this.messageQueue.length === 0) return;
118         let message: IMessage | undefined;
119         while ((message = this.messageQueue.pop()) !== undefined) {
120             switch (this.lobbyPhase) {
121 >               case LobbyPhase.RoamingColony: {…
129                 }
130 >               case LobbyPhase.AwaitingParticipants: {…
142                 }
143 >               case LobbyPhase.DeclareIntent: {…
165                 }
166                 case LobbyPhase.PlayersLoading: {
167                     if (message.eventID === PLAYER_LOAD_COMPLETE_EVENT.id) {
168                         if (this.minigameLoopInitFunc === null) {
169                             this.sequenceErrorGenericAbort(MOCK_SERVER_ID, 'Minigame loop init function is null');
170                             return;
171                         }
172                         this.lobbyPhase = LobbyPhase.InMinigame;
173                         this.log.info('Starting minigame game loop');
174                         const startFuncAttempt = await this.minigameLoopInitFunc(this.context, this.difficultyConfirmed!, this.colonyID);
175                         if (startFuncAttempt.err !== null) {
176                             this.sequenceErrorGenericAbort(MOCK_SERVER_ID, "Error starting game loop: " + startFuncAttempt.err);
177                             return;
178                         }
179                         startFuncAttempt.res();
180                         this.log.trace('Phase changed to InMinigame');
181                     }
182                     if (message.eventID === PLAYER_LOAD_FAILURE_EVENT.id) {
183                         const cast = message as PlayerLoadFailureMessageDTO;
184                         this.sequenceErrorGenericAbort(cast.senderID, cast.reason);
185                         this.reset();
186                     }
187                     break;
188                 }
189                 case LobbyPhase.InMinigame: {
190                 }
191             }
192         }
193     };
```

Image 4.4.3
bsc-frontend/ursa_frontend/src/components/colony/mini_games/miniGame.tsx

When in multiplayer, the mock server was turned off, and "`SingleplayerGameLoopInitFunc`" and "`GenericGameLoopStartFunction`" were not involved. For the multiplayer implementation, see 4.7 Multiplayer.

4.5 Integration With Third Party

Integrating with the Angular platform did have some difficulties which were not discovered during the proof of concept phase. Mainly due to the platform being out of date, whereas the proof of concept had used the latest version. However, workarounds were found without too much time spent.

One main change was using `ngAfterViewInit()` rather than `afterNextRender()` to execute the bundle, as well as having the bundle "place itself" on the window object as a callable function - allowing data to be passed between the two very distinct environments.

The integration within Angular, came in at about 90 lines of code, with some additional child-like components to hook into Angular's router:

```
64   {
65     path: 'games/ursa', //Dep: ursa_frontend/src/integrations/vitec/integrationConstants.ts
66     component: AppGamesURSAMenuComponent,
67     canActivate: [AuthGuardService],
68     title: 'URSA',
69   },
```

Image 4.5.1
10-finger-new/src/app/app-routing.module.ts

Declaring also the dependency on this project as “10-finger-new” is a clone of an older version of Vitec MV’s repository. Enabling routing this way through Angular, allowed for involving their internal authorization service. Each child provided the “parent” with the name of the bundle requested.

```
8   @Component({
9       selector: 'app-ursa-menu',
10      template: `
11        <app-ursa-inlay [bundleName]="'menu'"></app-ursa-inlay>
12      `
13  })
14  export class AppGamesURSAMenuComponent {}
```

Image 4.5.2
10-finger-new/src/app/modules/games/ursa/ursa.menu.component.ts

The “parent” was a custom HTML element “`<app-ursa-inlay>`” (*Image 4.5.2*) defined by this Angular component, facilitating the rest of the integration:

```
33  @Component({
34    selector: 'app-ursa-inlay',
35    template: `
36      <div #solidjsRoot id="solidjs-inlay-root" style="all: initial;"></div>
37    `,
38  })
39  export class AppGamesUrsaIntegrationComponent implements AfterViewInit, OnDestroy {
40    /**…
44    @Input() bundleName!: string;
45    private cleanup: (() => void) | null = null;
46    // Constructor is executed server-side
47    constructor(…
50    ) {}
51    // This hook runs in the browser
52    ngAfterViewInit() {
53      this.loadUserDataAndInitializeBundle();
54    }
55    private async loadUserDataAndInitializeBundle() {
56      try {
57        const userInfo = await firstValueFrom(this.userService.getUserInfo());
58        const vitecInfo: VitecIntegrationInformation = {…
66        };
67        // Inject the transpiled js output directly into this file, right here
68        await import(`../../../../../../bsc-frontend/dist/${this.bundleName}/index.js`);
69
70        // Find the function that the bundle just placed on the window
71        const initFunction: URSAInitializationFunction = (window as any)[URSA_INITIALIZATION_FUNCTION_NAME];
72        if (typeof initFunction === 'function') {
73          // Run that function with the gathered user info
74          this.cleanup = initFunction(vitecInfo);
75        } else {
76          console.error('URSA initialization function not found, expected to a function by the name of: '
77            + URSA_INITIALIZATION_FUNCTION_NAME + "', but got: " + JSON.stringify(initFunction));
78        }
79      } catch (error) {
80        console.error('Error initializing URSA bundle:', error);
81      }
82    }
```

Image 4.5.3
10-finger-new/src/app/modules/games/ursa/ursa.integration.component.ts

Note the “URSA_INITIALIZATION_FUNCTION_NAME” constant referenced on line 71, the HTML element id on line 36, and the relative path specified for the raw ESM import on line 68.
These were single points of failure, as in, if any of those ever changed and not correspondingly updated between projects, the entire integration would fail.
Therefore, for any of those declarations in either project, the single source of truth is noted:

```
8   /** Single source of truth: this */
9   export type VitecIntegrationInformation = {···
20  }
21  /** Single source of truth is bsc-frontend/ursa-frontend/src/integrations/vitec/integartionConstants.ts */
22  export type URSAInitializationFunction = (vitecInfo: VitecIntegrationInformation) => (() => void) | null;
23  /** Single source of truth is bsc-frontend/ursa-frontend/src/integrations/vitec/integartionConstants.ts */
24  export const URSA_INITIALIZATION_FUNCTION_NAME = 'initializeURSABundle';
```

Image 4.5.4
10-finger-new/src/app/modules/games/ursa/ursa.integration.component.ts

And correspondingly in the URSA code base:

```
3   /** Single source of truth: this */
4   export const SOLIDJS_MOUNT_ELEMENT_ID = 'solidjs-inlay-root';
5   export type URSAInitializationFunction = (vitecInfo: VitecIntegrationInformation) => (() => void) | null;
6   export const URSA_INITIALIZATION_FUNCTION_NAME = 'initializeURSABundle';
7
8   /** Single source of truth: The 10-finger angular project: ./src/app/app-routing.module.ts */
9   export enum SubURLs {···
13  }
14  export const VITEC_BASE_SUB_URL = '/games/ursa';
15
```

Image 4.5.5
bsc-frontend/ursa_frontend/src/integrations/vitec/integrationConstants.ts

On the frontend side of things, the entry point for each bundle (the contents of which would be run when the template reached the clients’ browser), would run an “initApp” function with a function reference to the top-level component for that bundle:

```
3   import ColonyApp from './ColonyApp';
4   import { initApp } from '@/setup';
5
6   initApp(ColonyApp);
```

Image 4.5.6
bsc-frontend/ursa_frontend/colony_root/colonyIndex.tsx

Which in turn would establish the aforementioned initialization function on the window object.

```ts
export const initApp = (app: BundleComponent<ApplicationProps>) => {
    // global function that Angular will call
    (window as any)[URSA_INITIALIZATION_FUNCTION_NAME] = (vitecInfo: VitecIntegrationInformation) => {
        const root = document.getElementById(SOLIDJS_MOUNT_ELEMENT_ID);
        if (!root) {
            console.error('[setup] Root element not found.');
            return;
        }

        if (app.bundle !== vitecInfo.bundleRequested) {
            console.error('[setup] Bundle mismatch. Expected: ' + vitecInfo.bundleRequested + ', but currently mounting: ' + app.bundle);
            return;
        }

        const dispose = render(() => GlobalContainer({ app, vitecInfo }), root);

        // Return a cleanup function
        return () => {
            dispose();
        };
    };
```

Image 4.5.7
bsc-frontend/ursa_frontend/src/setup.ts

The `GlobalContainer` was another component that would display a loading screen while the app initialized.

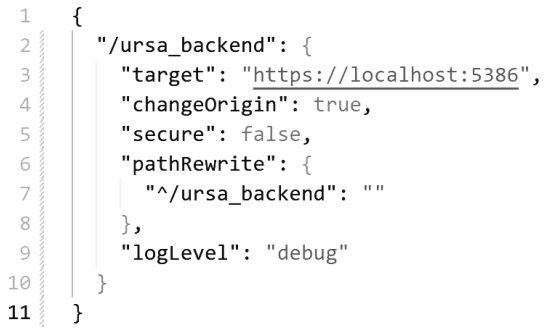
4.6 TLS & Browsers

Before any outbound communication can happen from any website client-side (i.e. requests made to external servers), the browser performs a Cross-Origin Resource Sharing (CORS) preflight request first [61]. This may fail if CORS has not been configured correctly on the server, causing the actual request not to be sent, and a CORS error to occur.

However, one other thing that might cause the preflight request to fail, is encryption - i.e. using HTTPS rather than HTTP. Given the sensitive nature of some information sent back and forth (specifically any user's Vitec session token and UUID assumed to be required for the cross-check), the connection had to be encrypted. Browsers, especially Chrome, do not accept self-signed TLS certificates however, and although workarounds do exist, none were deemed viable for any eventual user tests, where users might access the MVP on their devices [67].

The root of the problem is that a self-signed TLS certificate is not trusted by any Certificate Authority (CA [69]), and is as such rejected. It is possible to get a self-signed certificate authorized, however, the process requires proof of ownership of the IP address ([68] - note that TLS and SSL are sometimes used interchangeably, however, SSL is the predecessor), which was not possible.

To ensure that the MVP would be accessible regardless, a reverse proxy was used during development and testing to allow the connection to be over HTTP rather than HTTPS temporarily. The proxy was a built-in feature of Angular, and only needed to be configured.

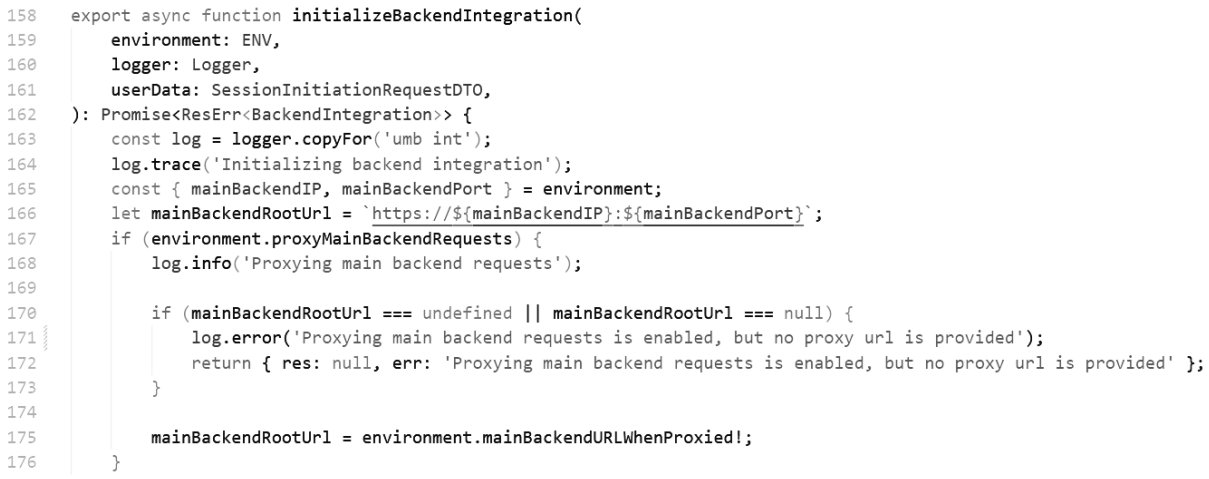


```
{
  "/ursa_backend": {
    "target": "https://localhost:5386",
    "changeOrigin": true,
    "secure": false,
    "pathRewrite": {
      "^/ursa_backend": ""
    },
    "logLevel": "debug"
  }
}
```

Image 4.6.1
10-finger-new/src/dev.proxy.conf.json

Above the configuration is set to reroute any request starting with "`/ursa_backend`" to the port on localhost. The target IP was changed when deployed on an AWS instance to match its IP address.

Likewise, the main backend integration in the frontend would replace the base URL if any substitute were present in the environment under a specific key during initialization:



```
export async function initializeBackendIntegration(
    environment: ENV,
    logger: Logger,
    userData: SessionInitiationRequestDTO,
): Promise<ResErr<BackendIntegration>> {
    const log = logger.copyFor('umb int');
    log.trace('Initializing backend integration');
    const { mainBackendIP, mainBackendPort } = environment;
    let mainBackendRootUrl = `https://${mainBackendIP}:${mainBackendPort}`;
    if (environment.proxyMainBackendRequests) {
        log.info('Proxying main backend requests');

        if (mainBackendRootUrl === undefined || mainBackendRootUrl === null) {
            log.error('Proxying main backend requests is enabled, but no proxy url is provided');
            return { res: null, err: 'Proxying main backend requests is enabled, but no proxy url is provided' };
        }

        mainBackendRootUrl = environment.mainBackendURLWhenProxied!;
    }
```

Image 4.6.2
bsc-frontend/ursa_frontend/src/integrations/main_backend/mainBackend.ts

Given the separation of bundles, some could be mounted outside of the 10-finger platform, and for those, the same kind of development proxy was used. This time facilitated by Vite instead.

4.7 Multiplayer

The multiplayer backend was implemented in GoLang using the standard library's "http" package [73] for some endpoints, and Gorilla WebSockets [74] for the WS "`/connect`" endpoint.
A message schema format was defined, as an integrated part of the source code, and program introspection was leveraged to convert the GoLang schemas into TypeScript. In turn, the frontend's multiplayer integration was written based on the generated code and could be kept synchronized by running a CLI command.

4.7.1 Event Specifications & Serialization

At first, a message schema was built as a list of objects containing an offset and a type, referencing some type from GoLang's reflect package: `reflect.Uint32, reflect.String`, … etc.
Later this design was made more compact, leveraging generic parameters and more introspection to generate these descriptions based on structs representing each message. The benefit of this alternative design was ease of use, and also, that each specification could carry a generic parameter with a reference to the struct type used to generate it.

The result of this implementation meant, that any event specification looked like this:

```
181  // 10-999: Lobby Management
182  var LOBBY_MANAGEMENT_EVENTS = NewSpecMap(PLAYER_JOINED_EVENT, PLAYER_LEFT_EVENT, LOBBY_CLOSING_EVENT)
183
184  var ENTER_LOCATION_EVENT = NewSpecification[EnterLocationMessageDTO](1001, "EnterLocation", "Send when the owner enters a location",
185      OWNER_ONLY, Handlers_NoCheckReplicate)
186
187  var PLAYER_MOVE_EVENT = NewSpecification[PlayerMoveMessageDTO](1002, "PlayerMove", "Sent when any player moves to some location",
188      OWNER_AND_GUESTS, Handlers_NoCheckReplicate)
```

Image 4.7.1.1
bsc-multiplayer-backend/src/internal/eventSpecs.go

Each call to "`NewSpecification`" included, in order: Generic reference type, ID of event, name of event, description of event purpose, who could send such event, and what to do when the server received an event with that ID.
The generated TypeScript schema became the following, using generics in the same fashion:

```
248  export interface EnterLocationMessageDTO extends IMessage {
249  >    /** Colony Location ID ···
253      id: number;
254  }
255  > /** EnterLocation Message Structure ···
259  export const ENTER_LOCATION_EVENT: EventSpecification<EnterLocationMessageDTO> = {
260      id: EventType.ENTER_LOCATION,
261      name: "EnterLocation",
262      permissions: {owner: true, server: false, guest: false},
263      expectedMinSize: 4,
264      structure: [
265          {
266              byteSize: 4,
267              offset: 8,
268              description: "Colony Location ID",
269              fieldName: "id",
270              type: GoType.UINT32
271          }
272      ]
273  }
```

Image 4.7.1.2
bsc-frontend/ursa_frontend/src/integrations/multiplayer_backend/EventSpecifications.ts

The use of generics meant that for any future reference, say deserializing some byte array according to a specification, the resulting type (if not error), would always be known at the call site.

```
13  // Deserialize any some data into a struct by the type given in the spec
14  //
15  // Based on remainderOnly, it will either expect the entire message (headers and all)
16  // or only the remainder (body) of the message.
17  func Deserialize[T any](spec *EventSpecification[T], data []byte, remainderOnly bool) (*T, error) {
18      // A specs offset is including the header although it does not itself describe it,
19      // so if remainderOnly == true, we need subtract the header size to get the correct offset
20      offsetAdjustment := util.Ternary(remainderOnly, MESSAGE_HEADER_SIZE, 0)
```

Image 4.7.1.4
bsc-multiplayer-backend/src/internal/deserialization.go

Additionally, this meant that no developer could provide the destination struct (i.e. some allocated memory to serialize the data into) in the first place, preventing any mismatch:

```
12  // Serializes the provided data according to the specification
13  // Includes the event id part of the header prefixed (i.e. only needs the sender id to be appended before sending)
14  func Serialize[T any](spec *EventSpecification[T], data T) ([]byte, error) {
15      // Calculate the exact size needed
16      messageSize, err := ComputeMessageSize(spec, data)
```

Image 4.7.1.3
bsc-multiplayer-backend/src/internal/serialization.go

Quick note: GoLang only has generic invariance (specifically lacking generic covariance as described by [75]). For instance, for type `B` extends `A`, `T[B]` is not assignable to type `T[A]`.
That led to some very unsafe pointer casting and intentional type erasure, to gather these specifications in a "`map[MessageID]*EventSpecification[any]`" structure for further use and lookups.
GoLang "`any`" is a type alias for "`interface`" which is somewhat comparable to Java "`Object`". GoLang does not have explicit classes, rather any struct may have a function whose signature complies with a function described by an interface - at which point the interface is considered "implemented", although at no point explicitly stated.

4.7.2 Concurrent Sessions

To allow for multiple concurrent multiplayer sessions a LobbyManager was defined to manage any amount of lobbies. Any instance of this singleton (practically, not implementation-wise), would also handle any connection attempt by some client (user):

```
14  // LobbyManager manages all the lobbies
15  type LobbyManager struct {
16      Lobbies           util.ConcurrentTypedMap[LobbyID, *Lobby]
17      nextLobbyID       atomic.Uint32
18      acceptsNewLobbies atomic.Bool
19      CloseQueue        chan *Lobby // Queue of lobbies that need to be closed
20      configuration     *meta.RuntimeConfiguration
21  }
```

Image 4.7.2.1
bsc-multiplayer-backend/src/internal/lobbyManager.go

This LobbyManager would also attempt to ensure a graceful shutdown, broadcasting the right messages (Lobby Closing and Server Shutdown) across all lobbies when the server closed - if any.

A Lobby was implemented as some struct, carrying the reference to each client web socket connection, and managing further handling of all messages sent by clients.

```
34  type Lobby struct {
35      ID       LobbyID
36      OwnerID  ClientID
37      ColonyID uint32
38      Clients  util.ConcurrentTypedMap[ClientID, *Client] // UserID to User mapping
39      Sync     sync.Mutex                                 // Protects access to the Users map
40      Closing  atomic.Bool                                // Indicates if the lobby is in the process of closing
41      //Prepends senderID
42      BroadcastMessage func(senderID ClientID, message []byte) []*Client
43      Encoding         meta.MessageEncoding
44      activityTracker  *ActivityTracker
45      currentActivity  *GenericMinigameControls
46      CloseQueue       chan<- *Lobby // Queue on which to register self for closing
47      // Queue of all messages to be further tracked
48      // All messages must have been through all pre-flight checks and handler before being added here
49      PostProcessQueue chan *MessageEntry
50      //Maybe introduce message channel for messages to be sent to the lobby
51  }
```

Image 4.7.2.2
bsc-multiplayer-backend/src/internal/lobby.go

4.7.3 Minigames

The '`ActivityTracker`' in *Image 4.7.2.2* handled the minigame initiation sequence, described in image *Figure 3.7.2.1* (3.7.2 Minigame Initiation Sequence), tracking what clients had opted in or out of participation. The '`GenericMinigameControls`' allowed the Lobby to handle what parts of the minigame were active and when.
Something that became clear during development was the need for "sub-phases" between when each player was ready (had sent a 'PlayerLoadComplete' message) and when the minigame began sending its specific messages, and right before it was considered over.
These sub-phases were named rising edge and falling edge, as a metaphor for the behavior of signals in a conductor, and would allow the Lobby to control when additional data was synchronized and when the minigame was assured to be over.

```
19  type GenericMinigameControls struct {
20      // Blocking. Executes pre-game start logic. If any
21      // Such as assigning players to teams, player data, etc, specific for the game loop
22      ExecRisingEdge MinigameRisingEdgeFunction
23      // Not blocking. Starts separate routine targeting game loop update function
24      StartLoop MinigameLoopStartFunction
25      // Blocking. Executes post-game end logic. If any
26      ExecFallingEdge MinigameFallingEdgeFunction
27      // Any error is returned as a debug event to the client
28      OnMessage func(msg *MessageEntry) error
29      State     *atomic.Uint32
30  }
```

Image 4.7.3.1
bsc-multiplayer-backend/src/internal/minigameSpec.go

As for the 'Asteroids' minigame specifically, on the rising edge, it would inform all players about the positions and codes of all other players, so that the images of their characters could be placed in the right spots. On the falling edge, however, it would check the "`State`" as mentioned in *Image 4.7.3.1*, and if the players successfully defended the colony, it would emit a '`LocationUpgradeEvent`'.
A '`MinigameWonEvent`' would be sent as part of the main game simulation loop to inform all clients that it was over. Likewise, a '`MinigameLostEvent`' also existed in the case of failure.
As for handling player inputs, any Lobby would call '`OnMessage`' (*Image 4.7.3.1*) on any client message not already handled after preflight checks and re-broadcast handling. Technically, these messages would first be put in the '`PostProcessQueue`' as seen in *Image 4.7.2.2*, and then taken out and handled by a dedicated thread. This order of operations also meant that the game simulation could be a little behind. However, due to the way a buffered channel in GoLang works ("ordered", blocking), this was not considered an issue.

4.7.4 Message Preflight Checks

In the interest of debugging, as well as using the fact that the centralized service was able to conduct some authorization and verification of each message, a series of preflight checks were introduced. I.e. checks happening right after some message was received from a client, and before any other handling, starting while extracting the message header:

```
42  // Extracts the client id and message id from a message, also verifies the length of the message
43  // Expects the msg to be raw binary data.
44  // # Returns client id, spec, rest of the message
45  func ExtractMessageHeader(msg []byte) (ClientID, *EventSpecification[any], []byte, error) {
46      if len(msg) < 8 {
47          return 0, nil, EMPTY_BYTE_ARR, fmt.Errorf("message size too small. Must at least include
48      }
49      // Extract userID and messageID (uint32)
50      userID := binary.BigEndian.Uint32(msg[:4])
51      messageID := binary.BigEndian.Uint32(msg[4:8])
52
53      var spec *EventSpecification[any]
54      var specExists bool
55      if spec, specExists = ALL_EVENTS[messageID]; !specExists {
56          return 0, nil, EMPTY_BYTE_ARR, fmt.Errorf("message ID %d not found", messageID)
57      } else if uint32(len(msg)) < spec.ExpectedMinSize+MESSAGE_HEADER_SIZE {
58          return 0, nil, EMPTY_BYTE_ARR, fmt.Errorf("message size too small. Expected at least %d b
59      }
60
61      return ClientID(userID), spec, msg[MESSAGE_HEADER_SIZE:], nil
62  }
```

Image 4.7.4.1
bsc-multiplayer-backend/src/internal/messaging.go

Further checks also included verifying that the sender (the client), was a part of said Lobby and that they had permission to send such a message. On any error, a dedicated debug message type was sent back to the client, and if that failed as well, the client was considered disconnected.
Other strategies do exist for unresponsive clients in an application like this, for instance allowing a timeout and storing all messages they are yet to receive in the meantime. This would be valuable for unreliable networks as a means of mitigating the impact of package loss but was cut for time.

4.8 Deployment

During development, the project was deployed in docker containers, building databases with deployment scripts, populating the Colony & Asset Database with the asset pipeline: Devour, and running the servers locally (*Figure 3.9.1.1* - 3.9.1 Deployment). This deployment approach meant the solution could be deployed on any machine that Docker could, as each service ran in its virtual environment, and communicated through exposed ports. This attempted to mimic an environment where each service has its own machine, but the main benefit of this development approach was that a single script could deploy all services, and a single execution of the asset pipeline would load all assets. This expedited and simplified the test setup for the user tests in AWS greatly.

5 Tests

To validate & verify the solution, gather intel on the performance of features, and confirm the implementation of required features, the solution was subject to unit testing, user testing, and acceptance tests.
User testing aimed to produce valuable data about user perception and engagement from which the success of individual feature sets could be gauged, both in a single-player and multiplayer setting. Acceptance testing determined whether or not Vitec MV was satisfied with the solution in regard to established conditions of satisfaction and mutual understanding of the objectives of the project.

5.1 Unit Testing

Unit testing was performed for parts of the project implementations, specifically core responsibilities and custom utility tools. These unit tests existed in the frontend, main backend, and multiplayer backend.
The asset pipeline was subject to the most extensive unit testing of all, but was not a component of the deliverable, despite greatly assisting development flow and speed.
No service had 100% unit test coverage, not even 90% since testing suites were only written for functionality found faulty as a part of a debugging process. Rather than prioritizing testing uniformly across the project, or even setting a coverage target in the first place, testing was focused on areas deemed especially complex or likely to fail while iterating.

5.1.1 Frontend

Vite, the bundler (build tool) used for the frontend includes a test runner: Vitest.
Vitest allows a `suite` to be declared with some description, followed by any number of `test` functions, with additional descriptions. Additionally, one may designate a timeout period and other options, but this was not used.

Specifically for the frontend, serialization, and deserialization of the binary messages used with the multiplayer backend were thoroughly tested.
This included tests of parsing and writing all data types used in the multiplayer backend:

```
145  suite('parseGoTypeAtOffsetInView', () => {
146      const encoder = new TextEncoder();
147
148  >   test('should parse UINT8 correctly', () => {···
153      });
154
155  >   test('should parse UINT16 correctly', () => {···
160      });
161
162      test('should parse UINT32 correctly', () => {
163          const buffer = new ArrayBuffer(4);
164          const view = new DataView(buffer);
165          view.setUint32(0, 4294967295, false);
166          expect(parseGoTypeAtOffsetInView<number>(view, 0, GoType.UINT32)).toBe(4294967295);
167      });
```

Image 5.1.1.1
bsc_frontend/src/integrations/multiplayer_backend/binUtil.test.ts

The internal message pub/sub service was also thoroughly tested:

```
describe('Multiplexer subscribe tests', () => {
    test('When making a subscription, a unique subscription ID must be returned', () => {···
    });
    test('When subscribed, the provided handler should be called on any emit of that type of event', async () => {
    });
    test('When initialized, it should invoke the given logger on any debug events', async () => {
        let logCount = 0;
        const logMessages: string[] = [];
        const spyLogger = {
            ...testLogger,
            info: (s: string) => {
                logCount++;
                logMessages.push(s);
            },
            copyFor: () => spyLogger,
        };
        const localPlayerID = 1;
        const plexer = initializeEventMultiplexer(spyLogger, localPlayerID);
```

Image 5.1.1.2
bsc-frontend/ursa_frontend/src/integrations/multiplayer_backend/multiplexer.test.ts

5.1.2 The Backends

Testing the backends also followed the same prioritization as the frontend, and was facilitated through GoLang's built-in test runner: '`go test`'.

For the main backend, some database queries used in the various endpoints were mocked using sqlmock:

```
func TestGetPlayerPreferences(t *testing.T) {
    mock, app, _, err := setupPlayerTest(t)
    if err != nil {
        t.Fatal("Setup failed:", err)
    }

    // Mocking the preferences row
    preferenceRows := sqlmock.NewRows([]string{"id", "preferenceKey", "chosenValue", "availableValues"}).
        AddRow(1, "Language", "EN", pq.Array([]string{"EN", "DK", "NO"}))

    mock.ExpectQuery(
            `SELECT "PlayerPreference".id, "PlayerPreference"."preferenceKey",
            "PlayerPreference"."chosenValue", "AvailablePreference"."availableValues"
            FROM "PlayerPreference" JOIN "AvailablePreference"`).
        WithArgs(1).
        WillReturnRows(preferenceRows)

    err = testPlayerRequest(t, app, "GET", "/api/v1/player/1/preferences", 200)
    if err != nil {
        t.Error("Failed to process the request:", err)
    }
```

Image 5.1.2.1
BSC-Main-Backend/src/api/playerApi_test.go

As for the multiplayer backend, deriving the event specifications from a generic parameter was tested:

```go
func TestDeriveReferenceStructure(t *testing.T) {
    ref, err := DeriveReferenceDescriptionFromT[testType1]()
    if err != nil {
        t.Error(err)
    }
    if len(ref) != 2 {
        t.Error("Expected 2 elements in reference structure")
    }
    if ref[0].FieldName != "field1" {
        t.Error("Expected field1 as fieldName in first element")
    }
    if ref[1].FieldName != "field2" {
        t.Error("Expected field2 as fieldName in second element")
    }
```

Image 5.1.2.2
bsc-multiplayer-backend/src/internal/eventSpecs_test.go

As well as tests of serialization and deserialization of most message types:

```go
func TestSerializePlayerJoined(t *testing.T) {
    data := PlayerJoinedMessageDTO{
        PlayerID: 1,
        IGN:      "test",
    }

    msg, err := Serialize(PLAYER_JOINED_EVENT, data)
    if err != nil {
        t.Fatalf("failed to serialize: %v", err)
    }
    if len(msg) != 12 {
        t.Fatalf("expected 12 bytes, got %d", len(msg))
    }
    eventIDBytes := msg[0:4]
    if binary.BigEndian.Uint32(eventIDBytes) != PLAYER_JOINED_EVENT.ID {
```

Image 5.1.2.3
bsc-multiplayer-backend/src/internal/serialization_test.go

These tests were also performed simulating a full “round trip” running through various message types.

```
341  func TestRoundTrip(t *testing.T) {
342      tests := []struct {
343          name string
344          spec interface{}
345          data interface{}
346  >       }{···
390      }
391
392      for _, tt := range tests {
393          t.Run(tt.name, func(t *testing.T) {
394              var genericSpec interface{} = tt.spec
395              var serialized []byte
396              var deserialized interface{}
397              var err error
398
399              // Serialize
400              switch s := genericSpec.(type) {
401              case *EventSpecification[BasicMessage]:
402                  serialized, err = Serialize(unsafeCastSpec(s), tt.data)
403                  if err != nil {
404                      t.Fatalf("failed to serialize: %v", err)
```

Image 5.1.2.4
bsc-multiplayer-backend/src/internal/serialization_test.go

And, following the philosophy of testing where it fails, various spurious tests were created. In the case below, a control flow error for dynamic size fields caused some message content to be at the wrong offsets:

```
436  func TestVerySpecificComputeSize(t *testing.T) {
437  >   data := AssignPlayerDataMessageDTO{···
443      }
444
445      _, err := Serialize(ASSIGN_PLAYER_DATA_EVENT, data)
446      if err != nil {
447          t.Fatalf("failed to serialize: %v", err)
448      }
449  }
```

Image 5.1.2.5
bsc-multiplayer-backend/src/internal/serialization_test.go

5.2 Integration Tests

Early testing was done through Insomnia, a third-party program like Postman, which can send many different types of requests, including creating web socket connections. The latter was used often to test the multiplayer backend.

No standardized integration test environment was set up in the interest of time. It is possible to automate tests of communication between services, however, it was deemed sufficient to use a lot of logging instead.

5.3 User Tests

User testing of individual students as well as group tests were conducted and compiled, to gain insight into the target audiences' perception of the product.

5.3.1 Constraints

User testing was limited in terms of time and resources. The timeline of the project was 3 months, thus testing objectives had to be prioritized.
With this in mind, testing was aimed at gathering valuable surface-level data on product feasibility and how engaging and intuitive it was to the end-user.

5.3.2 Individual Tests

In collaboration with a Danish public school located in Jutland, user tests were conducted with six students of the target demographic. The school has explicitly requested to remain anonymous, and will henceforth be referred to as 'John Doe Public School'.

5.3.2.1 Procedure

The user tests were scheduled to align as best as possible with the student's existing schedule. Over the school day, a short presentation was made at the start of each lecture for the three school classes involved, informing the students about the process in addition to what information would be gathered.
Since the students would always remain anonymous, this presentation was short to interfere as little as possible with their regular class schedule and deliberately kept concise so as not to reveal too much about the product, which could otherwise skew or bias their initial reactions during the following interviews.
The students were also informed that participation was voluntary, yet gathering a diverse sample across genders and reading capabilities (dyslexia) did not prove an issue.
The staff of the school were also very accommodating, allocating a classroom for the interviews which would remain undisturbed for the entire day.
Whenever a presentation was complete, the volunteer would be asked to gather their personal computer and then guided to the assigned room. The roles of interviewer (keeping the student engaged in conversation) and secretary (typing the interview summary), were swapped between interviews.
To guide each interview, each student was asked to perform a series of broad tasks like "complete the Tutorial" or "find the Fort", intentionally kept vague to observe how intuitive the user interface, controls, and gameplay were.

User 1 (Age 10)
User one expressed that they regularly play both Y8 and Roblox and are experienced with playing games. They enjoyed the menu theme and easily navigated to objectives. While initially struggling with concepts introduced in the tutorial, input scheme, and visual feedback, as well as text-heavy descriptions, the concepts were well understood when reaching colony navigation and minigame initiation. Despite the occurrence of a set of bugs, the user enjoyed and completed the minigame. As a closing remark, the user noted that they would like a clearer distinction between mouse and keyboard-based elements.

User 2 (Age 10)
Playing Fortnite, Minecraft, Roblox, and more, user two expressed that they are an avid enjoyer of computer games. The tutorial introduced difficulty as they needed more visual feedback and more concise explanations. They thoroughly enjoyed exploring the colony and visited numerous locations to enjoy the location graphics. The “easy” difficulty was too easy and bored the participant, but the concept of keyboard input was entertaining.

User 3 (Age 12)
User three was an avid computer games player and had played a large variety of games. The user was fluent in English and decided to play the game with the English translation. This user encountered little to no issues at any point and found all components of the game intuitive. After completing the minigame they immediately progressed to medium difficulty to play again.

User 4 (Age 11)
User four has played Roblox and several educational games such as online Jeopardy and a mathematics game. This user faced challenges during the tutorial mostly with upper/lower case letters, but also with labels being out of position in the navigation trial. While they enjoyed the minigame, they found the first difficulty to be too slow. They liked the focus on using keyboard input but would like clearer feedback on input success or failure.

User 5 (Age 12)
User five was immediately met with a unique set of challenges with a broken hand and severe dyslexia. They were however accustomed to using and playing on a keyboard, as they play Counter-Strike, Roblox, and Fortnite. They experienced difficulties with tutorial instructions and descriptions but understood concepts by trial and error. Despite decreased hand mobility they were able to play and enjoyed the minigame.

User 6 (Age 12)
User six plays FIFA, Valorant, and League Of Legends. This user struggled with severe dyslexia and had difficulty understanding tutorial objectives. By trial and error, they managed to complete the tutorial and experienced no further issues with performing similar tasks in the colony. They were very pleased with the minigame and overall graphics. As a last remark, they noted that they would have liked for the tutorial to be segmented into smaller objectives to better grasp and retain information.

Below is a list of all observations made throughout the interviews each categorized under some type. Some observations, with minor weight towards the purpose of these user tests, have been omitted for brevity. “Observation” and “type”, in this case, are used as:

- **Observation**
  Any comment made by any user about their experience with the product, or behavior exhibited by the user as observed by the interviewer.
- **#**
  Amount of users who exhibited this behavior.
- **Type**
  The type of cause of the expressed observation.
    - Clarity: Clarity in content purpose.

- Feedback: Feedback on user action.
- Bug: Unintended error in the implementation of a feature.
- Change Request: The user requested some changes to product content.
- User: The observations' cause seems to be rooted in the user, rather than the product.
- Visual: Visual Effects or visual design.
- Performance: The observation was caused by the users' device's hardware capabilities - or lack thereof.
  etc…

All observations have been slightly generalized and also categorized based on "where" in the product they were made. Importantly, lack of indication of observation does not mean that the users expressed differently/oppositely, but that the users made no indication towards that topic nor engaged in conversion about it. In other words, the users' opinion on the topic was inconclusive.

- M: Main Menu
- T: Tutorial
- C: Colony
- A: Asteroids Minigame
- G: General (across multiple areas)

ID	Observation	#	Type
Main Menu			
OM1	The user did not inherently understand the New Colony Page	1	Clarity
OM2	The user was not satisfied with the menu landing page visuals	1	Change Request
OM3	The user was satisfied with the menu navigation	1	n/a
OM4	The user was satisfied with the landing page visuals	5	Visual
OM5	The landing page did not reflect product contents	1	Clarity
Tutorial			
OT1	The user does not see that advancing through the tutorial was possible / or how to do so	4	Feed-back
OT2	The user did not gain any specific insights from the tutorial	2	Clarity
OT3	Text placements overlapping	3	Visual
OT4	The user was satisfied with the tutorial	4	n/a
OT5	The user did not understand the Navigation Trial at first	2	Clarity
OT6	Location nameplates during the Navigation Trial did not fit the	3	Bug

	viewport		
OT7	The user understood the path graph concept during the Navigation Trial	6	n/a
OT8	The user defaulted back to mouse use for buttons on pop-ups	1	Clarity
OT9	The user did not notice that Enter was necessary to execute Main Action Input content although shown in the demo.	1	Clarity
OT10	The user did understand the concept(s) shown in the Location Demo + Navigation Demo	3	n/a
OT11	The user did not understand the Multiplayer Demo	2	Clarity
OT12	The user found the Tutorial text-heavy	1	Change Request
OT13	The user likes the visuals in the Tutorial	1	Visual
OT14	The user would like something game-related in the Tutorial, as it currently gives a boring impression	1	Change Request
OT15	The user did not recognize the Main Action Input as a key component	0	Clarity
OT16	The user wants the tutorial broken up into smaller pieces	1	Change Request
General			
OG1	The user finds keyboard use challenging	1	User
OG2	User finds the different concepts presented unique	1	n/a
OG3	The user experienced a learning curve	4	User
OG4	Lack of reinforcement of what interactive elements are driven by the keyboard	3	Clarity
OG5	The user did not notice the capitalization of letters or spaces	5	User
OG6	The user has general reading problems	3	User
OG7	The user finds the keyboard-only concept fun	3	n/a
OG8	Ghosting Bug: The last symbol pressed in Main Action Input is ignored	1	Bug
OG9	Lack of indication when a button is disabled (Location Card)	1	Clarity
OG10	Dyslexic users are satisfied with font usage	1	Visual
Colony			
OC1	The user does not find navigating the colony difficult	5	n/a
OC2	The user was able to start a minigame without an issue	6	n/a

OC3	Hand Placement Check, the description was visually hard to read (lack of contrast)	1	Visual
OC4	User understands the concept of the Hand Placement Check easily	5	n/a
OC5	Hand Placement Check “accept” description is insufficient	2	Clarity
OC6	First-char-consumption bug during Hand Placement Check	2	Bug
OC7	The user found navigating the colony fun	2	n/a
OC8	The user enjoyed exploring the colony	2	n/a
OC9	The user liked the visuals in the colony	1	Visual
OC10	The user found the Hand Placement Check fun	2	n/a
OC11	The location nameplate is hidden behind the player's character	1	Bug
OC12	The user is confused as the default configuration of Hand Placement Check’s On-Screen Keyboard highlights under the mouse.	1	Bug
OC13	The user was technically challenged by the Hand Placement Check	1	User
Asteroids Minigame			
OA1	The user understands the game concept without further description	2	Clarity
OA2	The user is using hunt & peck technique	1	n/a
OA3	The user was satisfied with the minigame	5	n/a
OA4	The user would like to play the minigame again	6	n/a
OA5	Some undeterminable performance issues were observed during the minigame	1	Performance
OA6	User found difficulty as expected (“easy” was easy)	2	n/a
OA7	User finds the game fast-paced	1	n/a

Table 5.3.2.1.1

This table catalogs user feedback for further processing.

“#”: Amount of users exhibiting behavior / expressing certain statements.

For a more detailed breakdown of which user caused what observation, see Appendix F.

For the full, anonymized, annotated, transcripts (in Danish), see Appendix C.

5.3.3 Group Observations

In addition to the individual user tests, groups of students were observed as they went through the same objectives with minimal guidance, with the intent of testing the multiplayer aspects of the product. All students were new and had not before encountered the game, to not affect first impressions of other students.
The interview was more observational in the group test and the transcript was not made as interviews with each student but rather based on observation of the group. Students were given an individual user account and were not instructed about further objectives until all students had reached the same checkpoint, except for the very first to reach the colony, who was asked to open the colony and ask others to join. A set of noteworthy multiplayer-specific observations were made, noted in the table below.

Multiplayer		
OMP1	Users aided each other in solving and understanding objectives and roadblocks	2
OMP2	Users managed to expedite the completion of objectives as compared to individual users through communication.	2
OMP3	Users showed excitement related to multiplayer aspects	1

Table 5.3.3.1
This table catalogs multiplayer user feedback for further processing.

Multiplayer			
OMP1			n/a
OMP2			n/a
OMP3			n/a

Table 5.3.3.2
To view other group test observations not related to group aspects of the test, see Appendix D.

Notably, the impact of issues was significantly reduced and the objectives were met faster, as students were quick to aid others and point out solutions, minimizing roadblocks and frustrations. Additionally, several students displayed visible excitement when seeing other students in the colony and minigame, sparking immediate questioning as to who is who among each other. This seems to indicate that students would like to be able to identify each other and perhaps have in-game identities of their own other than simply having a custom name (usernames were that of the test accounts), such as custom appearances.

5.4 Acceptance Test

As a sign-off of the final deliverable (the MVP), the contact person from Vitec MV was asked to engage with the experience and its various facets, following a procedure much like the individual user tests.
Additionally, the results from the user tests were presented as seen in this report (Table 5.3.2.1.1, 5.3.3.1, 5.3.3.2 & Appendix F) alongside the anonymized transcripts (Appendix C), and discussed.
Finally, the contact person was asked to sign off on the Conditions of Satisfaction that were developed alongside them: COSV01 -> COSV06 and provide additional feedback if any.

ID	Description	Result
Vitec MV		
COSV 01	The delivered product should be able to become part of Vitec-MV's existing platform.	Fulfilled
COSV 02	The product should allow for the support of multiple languages	Fulfilled
COSV 03	The product should allow for multiplayer within the same class of students, akin to Kahoot.	Fulfilled
COSV 04	The product should offer persistent progress per user, i.e. a user may see things change or get improved as they progress through the game.	Fulfilled
COSV 05	The teacher is a big part of the equation and should provide value in some form to them. Preferably by making teacher participation optional.	Fulfilled
COSV 06	The product must contain, or be, at least 1 minigame, where a user may select from multiple difficulties.	Fulfilled

Table 5.4.1
This table displays established conditions of satisfaction with marks of satisfaction or dissatisfaction.

All conditions of satisfaction were accepted as fulfilled, and the Vitec MV contact expressed that they were impressed with the deliverable.

6 Discussion

While tests generally indicated that the solution was successful in most aspects, the user tests highlighted areas for improvement and areas of opportunity, as well as bugs that were not identified during development.
While the authentication was not fully integrated with Vitec MV's authentication service, the storage of sensitive data was avoided, but the solution may be subject to limited unauthorized access. The tutorial proved to be less intuitive than assumed, which is speculated to be due to developer bias through familiarity with the solution. Despite this, users generally found navigating the colony intuitive. The asteroid minigame functioned and users enjoyed the minigame, with a majority wanting to play again.

6.1 Authentication Integration with Third Party

The final deliverable was only accessible through the third-party platform, however, the planned authentication cross-check was not completed. Meaning, that any person with the right knowledge about the internals would be able to obtain a session token and thus interact with the different backends.
Although pretty significant, it was slated due to time, although possible.
The frontend bundles themselves were only accessible through Vitec MV's platform, however, while they would be able to generate colonies, modify these, access image assets, and so forth, they would never be able to actually have these displayed. Also, all handling of LODs, game mechanics, input handling, and so forth would be missing as well. Not to mention the deserialization schemas for the binary message format used by the multiplayer backend would also be missing.

6.2 Teacher Test

During the user tests, a teacher volunteered to try the game and comment on their experience. While not planned, this opportunity aligned well with the desires of the Vitec MV representative (see Appendix B) and defined Requirements (F04). Notably, the teacher experienced great difficulty with the 'Hand Placement Check' and had a hard time trying not to use the mouse to click on elements. A pattern their students picked up much quicker.
Regardless, they seemed to enjoy their time, especially with the available minigame, and asked for features ensuring even allocation of work amongst students in multiplayer. For instance, having a player's participation (how many asteroids they have shot down at some point) counts against them regarding when they would be allowed to participate again (i.e. firing their cannon again).

6.3 Game Design

While all planned features were implemented, some were more mature than others. Some design choices lead to unexpected outcomes and opportunities.

6.3.1 Menu

The menu itself was intuitive to most users and had very little feedback other than visual elements. While most feedback was positive and most navigation attempts were successful among testers, the main difficulty, while minor, was joining a colony. This aspect of the solution was only tested during multiplayer testing, of which there were only two. More information would have been useful, and could have been gathered had single-player participants also been asked to try to join another colony.

6.3.2 Tutorial

While it did not block users from progressing and completing the game, it did cause significant delays and difficulties during testing and may have skewed the first impression of testers. Some of the issues encountered in the tutorial were product-wide issues, but the tutorial faced significantly more than other components of the game, and many were unique to it.
In hindsight, the design process of the tutorial would likely benefit from a more objective and unbiased approach to design, as it appears the design process may have suffered from the very issues that were attempted to eliminate from the user tests, which is that it was not designed with a very first impression in mind. One such approach could be having more singular and frequent mocking and testing of the tutorial as it was iteratively designed, before finalizing the implementation.

6.3.3 The Colony

Visually, the colony needed work, but it very well displayed the vision, and testers were pleased with the looks and the interaction, especially when playing multiplayer. While further designing and planning would be necessary in identifying areas in need of change and refinement, a set of immediate work prospects is clear. Though some of the imagery is supplied by Vitec MV, a majority of the artwork was placeholder art, and will need replacement.
In addition to this, looking at user testing results, it appears there may be great value in expanding and refining exploration elements of the colony, as testers expressed excitement when traversing and exploring the colony and its many locations.

6.3.4 Asteroids Minigame

The Asteroids minigame demonstrates the gamification of the touch typing learning process. It is however very simple and repetitive. The learning experience could and should in future minigames be enhanced, as more developed touch typing methodology exists. As for the final MVP, the sequence of letters to press to shoot an asteroid was random, whereas it could have contained words or sentences.
In terms of graphics, testers were pleased, but the graphics were simple and could be enhanced. Enhancement could even improve the learning results, according to a meta-analytic review by Gui, Y., Cai, Z., Yang, Y. [08], in which adding more game design elements is claimed to improve learning outcomes.

6.4 Users & Progress Statistics

The original requirements required the user to be able to see their progress, and the user can. However, another aspect of progress, that was intended and is included in all designs, was statistics. Specifically statistics like “time between inputs”, “input reaction time”, and “words per minute”.
The intent was to gather these statistics while the user navigated the product, played minigames, etc, and then make them accessible to the user so that they could see if their “words per minute” or “time between inputs” had improved.
Snippets of the functionality exist in the product, however, these series of features were not complete in time.

6.4.1 Target Demographic

The target demographic, as desired by Vitec MV, was students of the ages eight to 14. User tests were performed with students of the ages of 10 to 12, due to the scope of the tests and time constraints. While not enough evidence exists to definitively claim that the younger participants were more challenged than the older, testing left the impression that the older students were generally tackling the test objectives and challenges more effectively.
This could simply be a coincidence, but could also suggest that less time in school may have had an impact on performance, perhaps through less developed literacy skills or mental development. Notably, those who played games more often than others also appeared to perform better and the sample size had a significant number of students with literacy impairment, so the exact cause remains uncertain.
It may however be of interest to expand testing to demographics outside the original range, as well as performing more tests in general on these ages to reliably identify areas of improvement. With this in mind, it is expected that first-time players experience a learning curve, however reducing it as much as possible may lower the barrier of entry.

Due to the limited time allocated for each user test, no evidence could be gathered as to the educational efficiency of the product. To obtain this evidence, a longer-running user test (over weeks, rather than hours) while collecting user statistics would be required.

6.5 Binary Messaging

Towards the end of the project, it was discovered that a high-performance, low package size overhead, schema-based solution did exist. Namely, Bebop [71], which already included TypeScript support and seemingly also GoLang support, although implemented by a third party [70]. And so unfortunately, the time used defining and implementing the message format, as well as the time spent writing the custom TypeScript output logic, could have been put to better use.

6.6 Testing

The reason the project did not have great test coverage, unit- and integration-testing, was primarily the size of the project - in terms of lines of code. There was simply not enough time to sufficiently cover the 29.804 lines of code it contained.
100% coverage is not necessary, and sometimes it is not even possible, but as of the end of the project, it might have approached 10-15% coverage - which is insufficient given that unexpected behavior was still encountered.
Testing is not the entire solution to bugs, some bugs will occur anyway, and the amount of logging did help a lot. Even though no automated integration tests were established, using the logs from the different services, most issues could be isolated quickly.
Whether or not the lack of testing was a detriment to the projects' success, is unclear given the positive results from Vitec MV and the users alike. But it most certainly could have helped.

7 Conclusion

This project addressed the challenge of touch typing proficiency in Danish schools through the development of an education game that aimed to make the learning process more engaging. The project scope and requirements were developed in collaboration with Vitec MV, who originated the project concept and sought a touch typing solution featuring multi-language support, multiplayer functionality, platform integration, and persistent user progress.

Comprehensive user testing was conducted with the solution deployed on Vitec MV's cloud infrastructure. Participants accessed the system through the 10-finger platform on their own devices, using test accounts, navigating through the tutorial, exploring the colony, and completing the minigame in both single-player and multiplayer.
The testing encompassed both English and Danish language options, with the infrastructure needed for the addition of any other languages, successfully delivering on the multi-language requirement.

The implementation successfully met most of the core requirements. The solution achieved seamless integration with Vitec MV's platform, including persistent storage for user progress, although the authentication cross-check functionality remained partially incomplete. The multiplayer component was successfully implemented and accessible in a Kahoot-style format, enhancing the social aspects of the learning experience. While one student experienced client-side performance issues during testing, asset handling performed as intended for all users.

The game design effectively masked the repetitive nature of touch typing practice, prioritizing engagement over explicit typing methodology. All testers completed the minigame component, indicating that the core gameplay mechanics were accessible and functional. The project culminated in acceptance testing with the Vitec MV stakeholder, who systematically evaluated each condition of satisfaction and expressed approval of the final deliverable, confirming the project's achievement of its primary objectives.

8 Future Work

The implementation of the MVP has established a solid foundation for an educational games platform. However, several areas have been identified for future development and improvement to enhance the platform's functionality, user experience, and technical robustness.

8.1 Bug Fixing

While user tests were performed with only minor bugs, some more breaking bugs were uncovered later at the most inopportune of moments: During the final acceptance test with the representative from Vitec MV.
While the cause was identified (a VPN) and a workaround was found, the bug rendered the existing minigame unplayable due to what seemed to be package loss (which should not have had the effect it did, given the connection was TCP-based). This major bug, while not entirely understood, is an edge case, while the minor bugs are common and not game-breaking.
Fixing these bugs is crucial to enhance the user experience and allow for uninterrupted play. More user tests will need to be performed to further identify unforeseen bugs and polish the game. Feedback from the user tests should be analyzed in depth and a plan to mitigate encountered issues should be established.

8.2 Minigames

As of the final MVP, the game contains a single minigame. This minigame contains a very simple input scheme, consisting of a random set of characters. While this is educational, it is very limited in regards to incentivizing touch typing at low difficulties.
The platform was built to support multiple minigames with their own unique gameplay and input schemes. By adhering to predefined interfaces, any minigame can be built and coupled to a location without having to modify the project outside of the minigame scope.
Future minigames should focus more on promoting touch typing through established touch typing methodology. While research into touch typing methodology has been performed, the scope of the project and established timeframe has not allowed for the integration of the aforementioned methodology into the MVP minigame [05].
A series of other minigames were drafted, but were too complex to implement within the timeframe, and were therefore put on ice, as the asteroid minigame was the most practical and viable solution given the project scope and time constraints.
'Transport Escort' was a minigame drafted for the vehicle storage location. It consisted of a large vehicle that was meant to reach the colony and was to be attacked by pirates. The goal was to shoot the pirates and avoid debris. Pirates were to have a random char sequence and cosmic debris would force players to avoid them by typing a full sentence to move their ship somewhere else.
The transporter was to have a large laser that would occasionally shoot at pirates, and players would need to move to avoid the laser.
A classic racing game was drafted to implement competitiveness among students. This racing game would consist of a set of lanes that players would race on side by side. Players would continually, but with diminishing effect, accelerate and would have to avoid obstacles that spawn on lanes at random by typing in words.

Finally “Battle Bots” was drafted, a modification of the classic ‘Battleships’ board game, that would be team-based and competitive. In the beginning, players would have a given timeframe to place their bots on their side of the tile-based board, where some ships would take one tile, some two, some three, and so on. The larger the bot, the more health it would have. At any time, players could move to avoid getting shot. This movement would be much like that of the classic snake game, a player would input the word in an adjacent tile and move to it, while the rest of the body would follow. If players moved onto the same tile, they would collide and the colliding segment of the bot would be damaged for both bots.

8.3 Graphics

While the game had passable graphics, most of the graphics were placeholders and would need to be replaced. A few graphical elements were supplied by Vitec MV, and inserted into the game to demonstrate a vision of the final look.

8.4 Architectural Considerations

There has been no effort in terms of server-level indirection layers, as in routing indirection, outside of reverse proxies for development purposes and thus any service in production is tightly coupled to any other service.
This is problematic, as no change can be made to any running service without direct impact on the platform's general availability and by extension the end user.
Calculating the magnitude of this impact on the end-user experience is not an exact science, however, strategies like:

- Blue/Green Deployments,
- A/B Testing, splitting traffic between multiple new versions to monitor some goal [15]
- Gradual rollouts, gradually transitioning traffic to one new version [15] or related:
- Canary Releases, for even more control.

Would allow one to reduce the risk in the first place.
Any level of abstraction is required before further deployment, two reverse proxies (e.g. nginx) in front of each of the backends would be a start.
Additionally, given the vastly different load profiles between the two, it might be ideal to look into load balancing & clustering the main backend due to the significant spikes caused by requests for image data.
Given that the main backend also facilitates all user sessions (auth), it mustn't fail.
Image data retrieval could be moved to a dedicated server which can be independently clustered. This could be a Content Delivery Network (CDN) solution, leaving the main backend handling authentication and dynamic content.
As for the multiplayer backend, it does not exhibit the same load spikes, however, latency matters a great deal more for its purposes. The solution does support multiple languages, so spreading multiple instances physically around could improve the experience if the user base becomes diverse.

Literature

[01] **Nadgaonkar 2024**
U. Nadgaonkar, "Stakeholder Matrix in Project Management," LinkedIn, Feb. 22, 2024. [Online].
Available:
https://www.linkedin.com/pulse/stakeholder-matrix-project-management-thought-individual--d5xxf. [Accessed: Dec. 2, 2024]

[02] **Pinet, Zielinski, Xavier & Longcamp 2022**
S. Pinet, C. Zielinski, A. F. Xavier, and M. Longcamp, "Typing expertise in a large student population," *National Library of Medicine* (PubMed Central), Aug. 5, 2022. DOI: 10.1186/s41235-022-00424-3. PMCID: PMC9356123, PMID: 35930064.

[03] **Weerdenburg, Tesselhof, & Meijden 2019**
M. van Weerdenburg, M. Tesselhof, and H. van der Meijden, "Touch-typing for better spelling and narrative-writing skills on the computer," *J. Comput. Assist. Learn.*, vol. 35, no. 1, pp. 143–152, Jan. 2019. DOI: 10.1111/jcal.12323.

[04] **Poole & Preciado 2016**
D. M. Poole and M. K. Preciado, "Touch typing instruction: Elementary teachers' beliefs and practices," *Comput. Educ.*, vol. 102, pp. 1–14, Sep. 2016. DOI: 10.1016/j.compedu.2016.06.008.

[05] **Dobson 2007**
A. Dobson, *Touch Typing in Ten Hours: Spend a Few Hours Now and Gain a Valuable Skill for Life*, 3rd ed. Oxford, United Kingdom: How To Books, 2007. ISBN: 978 1 84803 142 5.

[06] **Nielsen 2017**
B. Nielsen, "Fynsk virksomhed bliver svensk: Iværksætter fra Odense sælger IT-success," *Fyens.dk*, Aug. 17, 2017. [Online]. Available:
https://fyens.dk/erhverv/fynsk-virksomhed-bliver-svensk-ivaerksaetter-fra-odense-saelger-it-succes. [Accessed: Dec. 2, 2024].

[07] **Mayer, R. E. 2014.**
R. E. Mayer, *Computer Games for Learning: An Evidence-Based Approach*. Cambridge, MA: MIT Press, 2014. [Online]. Available:
https://archive.org/details/computergamesfor0000maye.

[08] **Gui, Y., Cai, Z., Yang, Y. et al 2023**
Y. Gui, Z. Cai, Y. Yang, et al., "Effectiveness of digital educational game and game design in STEM learning: A meta-analytic review," *IJ STEM Ed.*, vol. 10, p. 36, 2023. DOI: 10.1186/s40594-023-00424-9. [Online]. Available:
https://doi.org/10.1186/s40594-023-00424-9.

[09] **Weigelt, Marom, H., & Weintraub, N. 2015**
H. Weigelt Marom and N. Weintraub, "The effect of a touch-typing instructional program on keyboarding skills of higher education students with and without learning disabilities," *Res. Dev. Disabil.*, vol. 47, pp. 208–217, 2015. DOI: 10.1016/j.ridd.2015.09.014. [Online]. Available: https://doi.org/10.1016/j.ridd.2015.09.014.

[10] **Fodor, S., & Varga, M. 2020**

S. Fodor and M. Varga, "Using gamification to improve students' typing skills," in *Serious Games: Lecture Notes in Computer Science*, vol. 12572, K. Palovics, K. Holik, and T. Reti, Eds., Cham, Switzerland: Springer, 2020, DOI: 10.1007/978-3-030-63464-3_19. [Online]. Available: https://doi.org/10.1007/978-3-030-63464-3_19.

[11] **PISA 2022**
PISA 2022 report (analysis of the habits of children in schools across the world). Section 5.6 of the main report: "Brug af digitale apparater". Accessed the 28th of August at: "www.uvm.dk/internationalt-arbejde/internationale-undersoegelser/pisa/pisa-2022"

[12] **Krause Benchmarks**
Krause, Stefan, benchmarks:
https://krausest.github.io/js-framework-benchmark/index.html

[13] **Vitec - About Us**
Website link: "https://www.vitec-mv.com/om-vitec-mv/" latest visit the 2nd of December 2024

[14] **Derek Bok Center for Teaching and Learning**
Article: "Devices in the Classroom", published by the Derek Bok Center for Teaching and Learning. Author unknown. Release date unknown. Accessed through: "https://bokcenter.harvard.edu/technology-and-student-distraction", accessed the 5th of December 2024

[15] **Boger, Nir 2014**
Article: "A/B & Gradual Rollout - The Same but Different" published by Medium on the 3rd of February 2014. Accessed at: "https://nir-boger.medium.com/a-b-gradual-rollout-the-same-but-different-8b78b2035d25", latest visit the 9th of December 2025

[16] **Vitec Software Group**
Vitec Software Group, "Home," Vitec Software Group, [Online]. Available: https://www.vitecsoftware.com/en/. [Accessed: Dec. 10, 2024].

[17] **Vitec MV - Home**
Vitec MV, "Home," Vitec MV, [Online]. Available: https://www.vitec-mv.com/. [Accessed: Dec. 10, 2024].

[18] **Vitec MV - 10-finger vejledning**
Vitec MV, "10-finger vejledning," [Online]. Available: https://www.vitec-mv.com/media/hvjcxrwk/10-finger_vejledning.pdf. [Accessed: Dec. 10, 2024].

[19] **Vitec MV - 10-finger**
Vitec MV, "10-finger," Vitec MV, [Online]. Available: https://www.vitec-mv.com/programmer/10-finger/. [Accessed: Dec. 10, 2024].

[20] **Typing.com**
Typing.com, "Student Games," Typing.com, [Online]. Available: https://www.typing.com/student/games. [Accessed: Dec. 10, 2024].

[21] **Nitro Type**
Nitro Type, "Home," Nitro Type, [Online]. Available: https://www.nitrotype.com/. [Accessed: Dec. 10, 2024].

[22] **EdClub**

EdClub, "Sportal Program 3 Game," EdClub, [Online]. Available: https://www.edclub.com/sportal/program-3.game. [Accessed: Dec. 10, 2024].

[23] **TypeRacer**
TypeRacer, "Play," TypeRacer, [Online]. Available: https://play.typeracer.com/. [Accessed: Dec. 10, 2024].

[24] **The Learning Company**
The Learning Company, "Press release: The Learning Company announces groundbreaking educational software," The Learning Company, Apr. 6, 1998. [Online]. Available: https://web.archive.org/web/19981203045526/http://www.learningco.com:80/news/news98/980406a.htm. [Accessed: Dec. 10, 2024].

[25] **Newspapers.com**
Newspapers.com, "The Winnipeg Sun," Newspapers.com, [Online]. Available: https://www.newspapers.com/article/the-winnipeg-sun/112539580/. [Accessed: Dec. 10, 2024].

[26] **Interplay**
Interplay, "Home," Interplay, [Online]. Available: https://www.interplay.com/. [Accessed: Dec. 10, 2024].

[27] **Nintendo**
Nintendo, "Super Mario World - Overview," Nintendo, [Online]. Available: https://www.nintendo.com/en-za/Games/Super-Nintendo/Super-Mario-World-752133.html#Overview. [Accessed: Dec. 10, 2024].

[28] **Minecraft**
Minecraft, "Home," Minecraft, [Online]. Available: https://www.minecraft.net/en-us. [Accessed: Dec. 10, 2024].

[29] **Minecraft Education**
Minecraft Education, "Home," Minecraft Education, [Online]. Available: https://education.minecraft.net/en-us. [Accessed: Dec. 10, 2024].

[30] **Minecraft Education Whitepaper**
Minecraft Education, Minecraft Educational Benefits Whitepaper, [Online]. Available: https://education.minecraft.net/content/dam/education-edition/learning-experiences/research_folder/Minecraft%20Educational%20Benefits%20Whitepaper.pdf. [Accessed: Dec. 10, 2024].

[31] **Playright.dk**
Playright.dk, "Matematik i Måneby (PC)," Playright.dk, [Online]. Available: https://www.playright.dk/info/screens/matematik-i-maaneby/pc. [Accessed: Dec. 10, 2024].

[32] **Internet Archive**
Internet Archive, Matematik i Måneby Manual, [Online]. Available: https://ia601508.us.archive.org/view_archive.php?archive=/35/items/matematik_i_maaneby/MIM.zip&file=Matematik%20i%20M%C3%A5neby%2B%2FManual%2FManual.pdf. [Accessed: Dec. 10, 2024].

[33] **GDPR Info**
GDPR Info, "Home," GDPR Info, [Online]. Available: https://gdpr-info.eu/. [Accessed: Dec. 10, 2024].

[34] **Retsinformation**
Retsinformation, "Lovtidende A 2024 nr. 289," Retsinformation, [Online]. Available: https://www.retsinformation.dk/eli/lta/2024/289. [Accessed: Dec. 10, 2024].

[35] **ImageKit.io**
ImageKit.io, "Lazy Loading Images: The Complete Guide," ImageKit.io, [Online]. Available: https://imagekit.io/blog/lazy-loading-images-complete-guide/. [Accessed: Dec. 10, 2024].

[36] **Mozilla Developer Network**
Mozilla Developer Network, "Common MIME types," MDN Web Docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/MIME_types/Common_types. [Accessed: Dec. 10, 2024].

[37] **Isotropic**
Isotropic, "What is octet-stream?" Isotropic, [Online]. Available: https://isotropic.co/what-is-octet-stream/. [Accessed: Dec. 10, 2024].

[38] **ProgrammerAL**
ProgrammerAL, Serialization Benchmarks, GitHub repository, [Online]. Available: https://github.com/ProgrammerAL/SerializationBenchmarks/tree/main. [Accessed: Dec. 10, 2024].

[39] **Broker Pattern**
GeeksforGeeks, "Broker Pattern," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/broker-pattern/. [Accessed: Dec. 10, 2024].

[40] **Pub/Sub**
Ably, "What is Pub/Sub? The Publish/Subscribe model explained," Ably, [Online]. Available: https://ably.com/topic/pub-sub. [Accessed: Dec. 10, 2024].

[41] **Kafka**
GeeksforGeeks, "Apache Kafka Cluster Architecture," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/apache-kafka-cluster-architecture/. [Accessed: Dec. 10, 2024].

[42] **Datablast**
Datablast, "Building a Single Source of Truth (SSOT): 8 Best Practices for Data Integration," Datablast, [Online]. Available: https://www.datablast.io/post/building-a-ssot-8-best-practices-for-data-integration. [Accessed: Dec. 10, 2024].

[43] **MDN Repaint**
Definition, latest access the 10th of December 2024 at: https://developer.mozilla.org/en-US/docs/Glossary/Repaint

[44] **MDN Reflow**
Definition, latest access the 10th of December 2024 at: https://developer.mozilla.org/en-US/docs/Glossary/Reflow

[45] **Fischer & Jost 1989**
Fisher, William A. Jr., & Jost, James W., article: “Comparing structured and unstructured methodologies in firmware development” published in the Hewlett-Parckard Journal in April 198

[46] **D. Clegg and R. Barker 1994**
D. Clegg and R. Barker, Case Method Fast-Track: A RAD Approach. Reading, MA, USA: Addison-Wesley, 1994. [Online]. Available: https://archive.org/details/fasttrackradappr0000cleg

[47] **Larson, Erik W., & Gray, Clifford F. 2021**
Book: “Project Management: The Managerial Process”, 8. Edition, published by “McGraw-Hell Education”

[48] **Bekendtgørelse af lov om folkeskolen**
Bekendtgørelse af lov om folkeskolen (LBK nr 1396): "Bekendtgørelse af lov om folkeskolen (LBK nr 1396)," Retsinformation.dk, Dec. 7, 2020. [Online]. Available: https://www.retsinformation.dk/eli/lta/2020/1396. [Accessed: Dec. 13, 2024].

[49] Bekendtgørelse om formål, kompetencemål, færdigheds- og vidensområder og opmærksomhedspunkter for folkeskolens fag og emner
Bekendtgørelse om formål, kompetencemål, færdigheds- og vidensområder og opmærksomhedspunkter for folkeskolens fag og emner (BEK nr 185): "Bekendtgørelse om formål, kompetencemål, færdigheds- og vidensområder og opmærksomhedspunkter for folkeskolens fag og emner (BEK nr 185)," Retsinformation.dk, Feb. 25, 2019. [Online]. Available: https://www.retsinformation.dk/eli/lta/2019/185. [Accessed: Dec. 13, 2024].

[50] **IT og Teknologi (EMU)**
"IT og teknologi," EMU - Danmarks læringsportal. [Online]. Available: https://emu.dk/grundskole/it-og-teknologi. [Accessed: Dec. 13, 2024].

[51] **IT og Medier (UVM)**
"IT og medier," Børne- og Undervisningsministeriet. [Online]. Available: https://www.uvm.dk/folkeskolen/fag-timetal-og-overgange/fag-emner-og-tvaergaaende-temaer/it-og-medier. [Accessed: Dec. 13, 2024].

[52] **PMBOK 2021**
Project Management Institute, The Standard for Project Management and A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 7th ed. Newtown Square, PA: Project Management Institute, 2021. ISBN: 978-1-62825-664-2. Library of Congress Cataloging-in-Publication Data: LCCN 2021011107.

[53] **Agile Alliance 2001**
Agile Alliance, Manifesto for Agile Software Development, 2001. [Online]. Available: https://agilemanifesto.org/. [Accessed: Dec. 15, 2024].

[54] **W3, Ishida, R., & Miller, S. K.**
Article: “Localization vs Internationalization”, published 12th of May 2005 by W3.org, accessed 16th of December 2024, accessed through: “https://www.w3.org/International/questions/qa-i18n”

[55] **Flanagan, David**
Book: "JavaScript - The Definitive Guide", 5th ed., O'Reilly, Sebastopol, CA, 2006, p.497

[56] **Archer, Mike, 2021**
Article: “Templating Engines and Node.js” published by Medium on the 22nd of March 2021. Access through: “https://mlarcher531.medium.com/templating-engines-and-node-js-91b20b323f6” Latest access 16th of December 2024

[57] **SolidJS Team - createEffect**
SolidJS Team, "createEffect API," SolidJS Documentation, Dec. 2024. [Online]. Available: docs.solidjs.com/concepts/effects

[58] **ECMAScript 2015 Language Specification**
Sub-section 15.2: “Modules”, accessed through “https://262.ecma-international.org/6.0/#sec-modules” latest access the 16th of December 2024

[59] **SolidJS Team - DOM**
SolidJS Team, "Comparison: No Virtual DOM," SolidJS Documentation, Dec. 2024. [Online]. Available: https://www.solidjs.com/guides/comparison#no-virtual-dom

[60] **ETag - MDN**
MDN article: "ETag" accessed through "https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag", latest access: 16th of December 2024

[61] **CORS Preflight Request - MDN**
MDN article: "Preflight request" accessed through "https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request", latest access: 17th of December 2024

[62] **DTO - MS**
Microsoft article: "Data Transfer Object" published by Microsoft Learn the 17th of March 2014, accessed through: "https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff649585(v=pandp.10)?redirectedfrom=MSDN", latest access the 19th of December 2024

[64] **Open API Specification**
Open API Specification 3.1.1 accessed at: "https://swagger.io/specification/" latest access the 19th of December 2024

[65] **TypeScript Handbook**
The TypeScript Handbook: "https://www.typescriptlang.org/docs/handbook/"

[66] **Docker Compose**
https://docs.docker.com/compose/

[67] **Gu, David**
Gu, David, article: "Working With Self-Signed Certificates in Chrome (Walkthrough Edition)", published by Medium the 7th of September 2021, accessed through: "https://dgu2000.medium.com/working-with-self-signed-certificates-in-chrome-walkthrough-edition-a238486e6858" latest access 20th of December 2024

[68] **Gitlan, Dionisie**
Gitlan, Dionisie, article: "SSL Certificate Without a Domain Name. Is It Possible?" published by SSL Dragon the 19th of September 2024, accessed through: "https://www.ssldragon.com/blog/ssl-without-domain-name/" latest access the 20th of December 2024

[69] **Certificate Authority - MDN**
MDN definition: "Certificate authority" accessed through: "https://developer.mozilla.org/en-US/docs/Glossary/Certificate_authority" latest access 20th of December 2024

[70] **Bebop Golang**
GitHub repository: "https://github.com/200sc/bebop" latest access 20th of December 2024

[71] **Bebop Docs**
Documentation for the Bebop serialization & deserialization format: "https://docs.bebop.sh/" latest access 20th of December 2024

[72] **Artifact Registry - GCP**
Google Cloud Platform documentation on use of Artifact Registry in combination with Docker images: "https://cloud.google.com/artifact-registry/docs/docker" , latest access the 20th of December 2024.

[73] **GoLang HTTP**
GoLang standard library http package documentation: "https://pkg.go.dev/net/http" latest access 21st of December 2024.

[74] **Gorilla Websockets**
GitHub repository "https://github.com/gorilla/websocket" latest access 21st of December 2024.

[75] **Covariance & Contravariance - MS Learn**
Article: "Covariance & contravariance in generics" published by Microsoft Learn, accessed through:
"https://learn.microsoft.com/en-us/dotnet/standard/generics/covariance-and-contravariance" latest access the 21st of December 2024.

[76] **Bundle Size LinkedIn Post**
Linked in post supposedly quoting some Google study:
"https://www.linkedin.com/advice/0/what-best-tools-plugins-analyze-optimize-js-bundle" latest access 21st of December 2024

Appendix

Appendix A - Original Project Pitch

Transcribed from a handout.

Synopsis
The project involves adding more keyboard games to the 10-finger web application.

Goal
Learning typing is not that difficult, but mastering it requires repetition. Especially for children, it can be difficult to maintain the focus necessary to learn typing at a good level. Through games, their focus can be maintained for a longer period, and they thereby get more training and better typing skills.

The goal is to add at least 1 game that matches the room theme for 10-finger. There is an option to expand the project to support multiplayer gameplay.

Process
To make the process more manageable, we divide the project into the following 2 phases: start-up and development.

The start-up phase will focus on learning about 10-finger, programming languages and how the game should work. A graphic draft will also be made for the game.
In the development phase, the development itself will be started. The focus will be on "minimal viable product" and more features or games will be continuously built depending on project progress.

Study Relevance
In the project, you will encounter the programming languages: Angular, Typescript, HTML, CSS, & JavaScript. There will also be a need for planning, gamification, and process management.

The overall project is expected to have a scope of 3 months.

Appendix B - Meeting Notes A

Transcribed and translated.

VITEC MV Meeting 30052024

Present: Esben Gaarsmand (CTO), Klaes D. Sørensen, Gustav B. Wanscher

Q: Target demographic?
A: Public school, 8->14 år
Q: User testing & evaluation?
A: Not necessarily
Q: How do you determine user performance metrics?
A: Precision and tempo
Q: Application life-cycle user story
A: Accessed as separate segment of existing web platform.
Q: What language, and support of multiple language?
A: Multiple language support preferably (Danish, Swedish, Flemish)
Q: Persistent user-progress?
A: Yes - GDPR prone. (Use hashed user id's, separate db, … etc)
Q: Reference material and games?
A: "Matematik i Måneby". "Fizzykeys" *(existing game on own platform)*
Q: Reference material - graphic design
A: None, but a graphic designer is available given we have mock-ups.
Q: Existing infrastructure?
A: Has own cloud at Microsoft Azure.
Q: Integration within existing platform? *and if so, what is that platform made of*
A: Angular, TS - client side SPA setup. Auth is handled. A user can get redirected from multiple places.
Q: Available resources for project? *In terms of use of cloud services and or other hosts*
A: Carte blanc

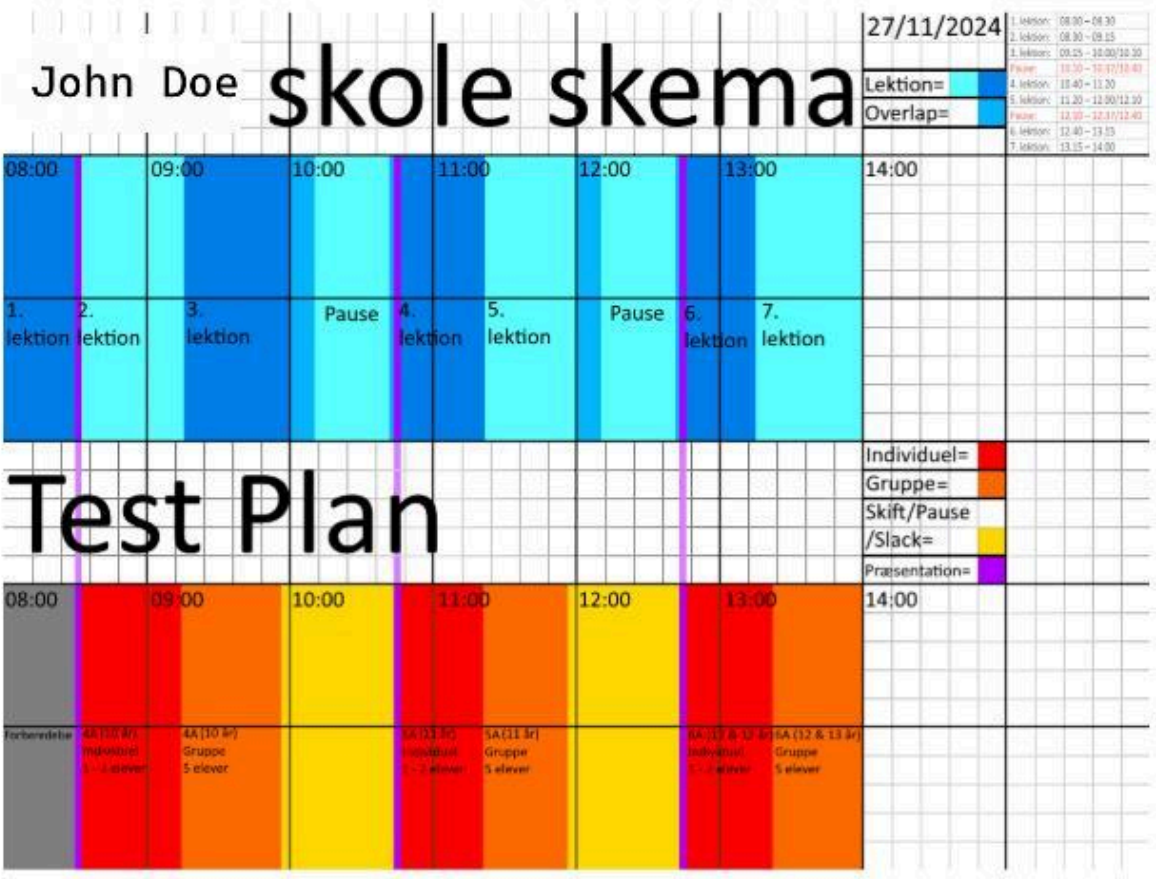
Further things mentioned:
User defined difficulty
Space theme

Proposal: Analysis -> Idea-phase -> Design -> …
Regarding multiplayer: Local, kahoot-style
The teacher is important in the equation.
No intentions for distribution on mobile devices / tablets.
Next up: NDA

Appendix C - Full, Anonymized, Annotated, Usertest

Brugertests

Dette dokument er et manuskript over en række spørgsmål og kontekst for brugertests af elever fra xxxx skole i aldersgruppen 10-13 år. Dokumentet bruges til at samle en række data i form af feedback af spiloplevelsen.



C.1 User 1

Preface Questions

Alder
10

Spiller du videospil? (Hvis så, hvilke)
Roblox (hver weekend)
Y8
Poki

Mål

- ☑ ~~Menu~~
- ☑ ~~Tutorial~~
- ☑ ~~Koloni~~
- ☑ ~~Minigame (asteroids)~~
  - ☑ ~~Sejr?~~

Menu

Hvordan var det at navigere menuen?	Fin **[OM3]**
Hvad synes du om menuen udseende?	"Jeg vil ik sige den flot, men jeg ville heller ikke sige at der er for mange farver" **[OM2]**
Har du andre meninger om menuen?	

"Skal jeg så bare skrive hvad den skal hedde?" - under Ny Koloni **[OM1]**

Tutorial

Hvordan var det at navigere i tutorial?	"Skal jeg trykke videre?" **[OT1]**
Forstod du hvad du skulle lære i denne tutorial?	Nej. **[OT2]**
Har du andre meninger om denne tutorial?	Den var god **[OT4]**

"Hvad skal man skrive?" - Navigation trial **[OT5]**
Kamera placering. Nameplates er ikke så godt synlige i Nagivation Trial **[OT6].**

Brugeren lader til at fange konceptet med stierne meget hurtigt. (Navigation Trial) **[OT7]**
“Hvad skal jeg trykke på” - Location Trial. Bruger kan ikke se hvornår den er fuldført **[OT1]**
“Hvad skal jeg trykke på?” - Multiplayer Trial. Bruger fanger ikke umiddelbart at alle knapper også er taste-baseret. **[OT8]**
suk - Bruger virker lidt træls over hvor meget, der skal skrives. **[OG1]**
Generel svær fordi den bryder vante koncepter. **[OG2]**

Koloni

Hvordan var det at navigere i kolonien?	“Nemt” **[OC1]**
Kunne du finde hvor du skulle hen?	Ja **[OC1]**
Har du andre meninger om denne kolonien?	Nej

Brugeren virker ikke til at have problemer med at navigere efter en lokation, de ikke kan se. **[OC1]**

Minigame

Hvordan var det at starte et minigame?	Brugeren lod ikke til at have problemer. **[OC2]**
Hvordan var Hand-Placement checked?	Problemer med at læse forklaringsteksten (visuel klarhed af teksten). **[OC3]** Brugeren fanger mønsteret ret hurtigt uden videre hjælp **[OC4]**. Dog kunne forklaringen godt være lidt bedre. **[OC5]** Brugeren oplevede state-bugget. **[OC6]** 4 forsøg
Hvordan var det at spille minigamet?	“Ej så jeg skal skrive det?!” - Brugeren så asteroiderne første gang Brugeren kan umiddelbart, intuitivt fange konceptet. **[OA1]** Brugeren lader til at default til Hunt & Peck efter lidt tid på Easy. **[OA2]**
Er det et minigame du kunne have lyst til at spille igen?	Ja **[OA3]** “Jeg syntes faktisk det et ret sjov spil” **[OA4]**
Har du andre meninger om minigamet?	

Minigamet hakkede en lille smule generelt **[OA5]**

Afsluttende

Hvad synes du om hele spillet?	“Det er jo lidt anderledes at man skal bruge tastaturet, men man lærer det at kende” **[OG2]** “I starten var det lidt svært” **[OG3]**
Hvordan er det at spille med et tastatur?	“Fint når man lærer det” **[OG3]**
Har du andre meninger om denne kolonien?	

Generelle problemer med at se hvad der er drevet af mus vs. tastatur **[OG4]**

C.2 User 2

Preface Questions

Alder
10 år
Spiller du videospil? (Hvis så, hvilke)
Firtnite, minecraft, roblox
Y8, crazy games
Lære at spille spillene.

Vertical scaleings problemer.

Mål

- ☑ ~~Menu~~
- ☑ ~~Tutorial~~
- ☑ ~~Koloni~~
- ☑ ~~Minigame (asteroids)~~
    - ☑ ~~Sejr?~~

Menu

Hvordan var det at navigere menuen?	
Hvad synes du om menuen udseende?	
Har du andre meninger om menuen?	- "Uhh, det er Mars!" **[OM4]**

Tutorial

Hvordan var det at navigere i tutorial?	- Store og små bogstaver **[OG5]** - Missede at man skulle trykke enter, fandt ud af det. **[OT9]** - Problemer med at navigere, i forhold til at kunne se navne, navigere frem og tilbage for udsyn. (navigation trial) **[OT6]**
Forstod du hvad du skulle lære i denne tutorial?	- Fangede godt hvordan man skulle navigere og gå ind i bygninger. **[OT10]** - Manglende feedback på succes. **[OT1]** - Havde problemer med at forstå hvad kolonikoden er og om den skal tastes ind. **[OT11]** - Forstod ikke at kolonikoden var til multiplayer brug. **[OT11]**
Har du andre meninger om	- Man skal læse meget **[OT12]**, men den er god **[OT4]**, hvis man ikke kan læse eller er ordblind er den nok svær, men den

denne tutorial?	var okay at forstå, medmindre man er ligeglad med at læse, så gætter man sig frem.

Koloni

Hvordan var det at navigere i kolonien?	- Meget sjovt **[OC7]** - Man skal holde øje med stierne. **[OT7]** - udforskning var sjov, gik rundt og åbenede andre lokationer **[OC8]**
Kunne du finde hvor du skulle hen?	- Man skal holde øje med hvor man går hen og hvor man ikke skal gå hen i forhold til stierne. **[OT7]**
Har du andre meninger om kolonien?	- Måske for meget grammatik? **[OG5]**

Bruger virker meget utålmodig / spændt på at komme igang med spillet

Minigame

Hvordan var det at starte et minigame?	- God feedback i forhold til slukkede knapper. **[OC2]** - Usikker på hvad vi vil have ham til.
Hvordan var Hand-Placement checked?	- Gik godt, 3 forsøg. **[OC4]**
Hvordan var det at spille minigamet?	- Let er for let, og en smule kedeligt. **[OA6]** - Uklare bogstaver, font (qp)? **[OG6]** - Skal orientere sig ofte på tastaturet, svært? (del af læringen) **[OG3]** - Godt kun med små bogstaver
Er det et minigame du kunne have lyst til at spille igen?	Ja **[OA4]**
Har du andre meninger om minigamet?	Nej, det går hurtigt **[OA7]**

Afsluttende

Hvad synes du om hele spillet?	

Hvordan er det at spille med et tastatur?	Ret sjov **[OG7]**
Har du andre meninger om denne kolonien?	

C.3 User 3

Preface Questions

Alder
12

Spiller du videospil? (Hvis så, hvilke)
Minecraft, Roblex, Wobblies Life
SpiderMan 2, Crazy Games

Mål

- ☑ ~~Menu~~
- ☑ ~~Tutorial~~
- ☑ ~~Koloni~~
- ☑ ~~Minigame (asteroids)~~
  - ☑ ~~Sejr?~~

Menu

Hvordan var det at navigere menuen?	
Hvad synes du om menuen udseende?	“Det et eller andet med Mars” “Jeg syntes det meget sejt” **[OM4]** “Det ser fedt ud” **[OM4]**
Har du andre meninger om menuen?	

Tutorial

Hvordan var det at navigere i tutorial?	
Forstod du hvad du skulle lære i denne tutorial?	Bruger lader til at fange konceptet med key-board styring. **[OT10]**

Har du andre meninger om denne tutorial?	Likes the visuals. **[OT13]** Would like to see something game related in the Tutorial because the current state gives the impression that there's no action and its boring. **[OT14]**

Bruger så ikke at videre-knappen "låste op" x2 **[OT1]**
Bruger syntes Welcome skærmen er meget fancy **[OT13]**
Bruger sagde de var flydende i engelsk - resten af interviewet blev gjort på engelsk. Spillet blev også spillet på engelsk
Tekst-placeringer er dårlige. **[OT6]**
Bruger fanger samtlige koncepter MEGET hurtigt **[OT7] [OT10]**. Bruger starter også med at læse beskrivelserne før de går igang med opgaverne.
"Pretty cool" x11 **[OT4]**

Koloni

Hvordan var det at navigere i kolonien?	Store og små bogstaver skal man lige vende sig til. Bruger så ikke nødvendigvis at der fx. kun var små bogstaver. Begge var fint. **[OG5]**
Kunne du finde hvor du skulle hen?	*user had no issues* **[OC1]**
Har du andre meninger om denne kolonien?	

after page load "Oh this is cool, I like this" **[OC9]**

Minigame

Hvordan var det at starte et minigame?	Let og overskueligt **[OC2]**
Hvordan var Hand-Placement checked?	Bruger fangede konceptet hurtigt. **[OC4]** 2 forsøg Bruger syntes det var sjovt. **[OC10]**
Hvordan var det at spille minigamet?	"I like this, this is fun" **[OA3]** "I love shooting myself" - sarkastisk. Godt humør **[OA3]** Bruger kan godt lide twistet med at man kan skyde sig selv
Er det et minigame du kunne have lyst til at spille igen?	Bruger begyndte spillet igen med det samme **[OA4]**

Har du andre meninger om minigamet?	“Maybe put like, uhm, minigames in different areas, so you can walk around and play them”

“Tell me when the game is available - cause I want to play this”. **[OA4]**
“Maybe we could have a minigame that is like pressing those sequences [Hand Placement Check]” **[OC10]**
user goes exploring because they want to explore the mine **[OC8]**

Afsluttende

Hvad synes du om hele spillet?	“I love it - its very fun”
Hvordan er det at spille med et tastatur?	“It is fun, you just kinda have to learn that not everything goes as fast . But it is definitely fun” **[OG7] [OG3]**
Har du andre meninger om denne kolonien?	“This place just looks like one where you place your cars [Points to Cantina]”

C.4 User 4

Preface Questions

Alder
11
Spiller du videospil? (Hvis så, hvilke)
Igen spil der bruger tastatur primært
Online spil, roblox
Læringsspil online jeperdy, matematik. Vælge regnestykke og svare rigtigt.

- ☑ ~~Menu~~
- ☑ ~~Tutorial~~
- ☑ ~~Koloni~~
- ☑ ~~Minigame (asteroids)~~
  - ☐ Sejr?

Menu

Hvordan var det at navigere menuen?	-
Hvad synes du om menuen udseende?	- Fint, men meget mørke farver. **[OM4]** - Fortrækker sort på hvid. (light theme) (knapper)
Har du andre meninger om menuen?	- Det ser ikke ud som om det er et tastaturspil, ligner mere bare et “normalt spil” - Ligner et bil spil **[OM5]**

Tutorial

Hvordan var det at navigere i tutorial?	- Fandt selv ud af at trykke viderer. -
Forstod du hvad du skulle lære i denne tutorial?	- Den prøver at lære at bruge tastaturret. - Det var forvirrende, når den ikke godkender det korrekte input. **[OG8]** -
Har du andre meninger om denne tutorial?	- Forstod ikke igennem videon hvad der skulle gøres **[OT2]**, men læste og fangede at tastatur skal bruges. - Problemer med store og små bogstaver - (Navigation Trial) **[OG5]** - Problemer med at se navne uden for skærmen, navigere til andre lokationer for at kunne se. **[OT6]** - Overlap af tekst **[OT3]** - Problemer med at stave uden at kunne se. **[OT3]** - Lagde ikke mærke til sti. **[OT5]** - Gik tilbage (demo video) for at forstå hvad der skulle gøre

	(location trial). - Bedre disabel feedback, (blå tekst - andre steder er den grå - location trial) **[OG9]** - Indtaster koloni kode i skrivefelt, forvirrer. **[OT11]**

Koloni

Hvordan var det at navigere i kolonien?	- Meget sjovt at man skal bruge tastur og ikke bare kan gå alle steder hen men skal følge en sti. **[OC1] [OC7]**
Kunne du finde hvor du skulle hen?	Bruger kunne intuitivt se at “Mine” var en blind gyde **[OT7]**
Har du andre meninger om denne kolonien?	

Bruger oplevede lang load tid for Colony

Dev Note: INGEN restriktioner på koloni navne

Minigame

Hvordan var det at starte et minigame?	- Ligetil at starte et minigame. **[OC2]**
Hvordan var Hand-Placement checked?	- Troede at capital skulle bruges.**[OG5]** - Bruger fangede umiddelbart konceptet. **[OC4]** - Ukendt bug med indtastning af bogstaver. **[OG8]**
Hvordan var det at spille minigamet?	- Bruger fangede intuitivt konceptet **[OA1]** - Problemer med at korrekt input ikke bliver godkendt. **[OG8]**
Er det et minigame du kunne have lyst til at spille igen?	Ja **[OA4]**
Har du andre meninger om minigamet?	- Det tager for lang tid og det går for langsomt (let) **[OA6]** - Meget sjovt. **[OA3]**

Afsluttende

Hvad synes du om hele spillet?	

Hvordan er det at spille med et tastatur?	
Har du andre meninger om denne kolonien?	

C.5 User 5

Preface Questions

Alder
12

Spiller du videospil? (Hvis så, hvilke)
Roblox, Counterstrike, Fortnite

Bruger har højre hånd i gibs
Bruger er ordblind (kraftigt) **[OG6]**

Menu

Hvordan var det at navigere menuen?	
Hvad synes du om menuen udseende?	“Det godt der flyver noget forbi [peger på Partikel Effekten]” Bruger syntes godt om den **[OM4]**
Har du andre meninger om menuen?	

Tutorial

Hvordan var det at navigere i tutorial?	Fint **[OT7]**
Forstod du hvad du skulle lære i denne tutorial?	- Lærte at gå rundt fra sted til sted og hvordan det fungerer **[OT4]**
Har du andre meninger om denne tutorial?	

Bruger ser ikke at “videre” knappen bliver oplåst x3 **[OT1]**

Bruger finder umiddelbart ud at bruge Main Action Input.
Bruger ser ikke det store start bogstav x1 **[OG5]**
Bruger har behov for screen reader support for beskrivelsen af Navigation Trial **[OG6]**
Bruger ser umiddelbart godt ud til at kunne læse stednavnene på diverse Locations.
“Den fungere helt fint” - Bruger om font brug i BufferBasedButton **[OG10]**
Tekstplaceringer er fortsat **redacted** i Tutorial **[OT3]**
Bruger ser ikke ud til at lægge mærke til at de skal skrive “Besøg” i anden omgang for at besøge en Location. (Location Trial) **[OT10]**
Bruger prøver umiddelbart at klikke på ting med musen (Multiplayer Trial) **[OG4]**

Dev Note: Bruger har en ekstremt lav opløsning skærm i kvadratisk format (<1000x1000) og altså fungerer mange placeringer af elementer ikke så godt.

Koloni

Hvordan var det at navigere i kolonien?	“Det var sjov. [...] Når man normalt skal skrive er det sådan, kedeligt.” **[OC1]**
Kunne du finde hvor du skulle hen?	
Har du andre meninger om denne kolonien?	

Bruger gik umiddelbart tilbage og prøvede at bruge musen **[OG4]**
“Besøg” / “Enter” er dækket af spiller karakteren. (Player zIndex --) **[OC11]**
Bruger kan umiddelbart godt se hvad sværhedsgrader er tilgængelige og ikke er.
At keyboarded under Hand Placement Check highlighter under musens cursor forvirrer brugeren. **[OC12]**
Input Bug er træls (spiser det første tegn under Hand Placement Check efter on-failure reset) **[OC6]**

Minigame

Hvordan var det at starte et minigame?	Let **[OC2]** Brugeren prøvede dog at bruge musen **[OG4]**
Hvordan var Hand-Placement checked?	Sjov. (Nærmest et minigame selv) **[OC4] [OC10]** Brugeren klarede den selv (selv med en brækket hånd)
Hvordan var det at spille minigamet?	Ser sjovt ud (nåede ikke så meget) **[OA3]**

Er det et minigame du kunne have lyst til at spille igen?	Ja **[OA4]**
Har du andre meninger om minigamet?	“Jeg syntes det vel-udført” **[OA3]**

Afsluttende

Hvad synes du om hele spillet?	Ingen yderligere kommentarer
Hvordan er det at spille med et tastatur?	
Har du andre meninger om denne kolonien?	

Pga ordblindhed har bruger generelt behov for at beskrivelser bliver læst op. (oplæsning på computer)

C.6 User 6

Preface Questions

Alder
12

Spiller du videospil? (Hvis så, hvilke)
Fifa, Fortnite, Valorant, LoL
“Ida & Emil” - læringsspil

Bruger er ordblind (rød, ud af “rød” | “gul” | “grøn”) **[OG6]**

Menu

Hvordan var det at navigere menuen?	-
Hvad synes du om menuen udseende?	- Det ser sjovt ud **[OM4]**
Har du andre meninger om	

menuen?	

Tutorial

Hvordan var det at navigere i tutorial?	- Spottede videre pil med det samme. - Forvirring med store og små bodstaver **[OG5]** - Overlap af tekst **[OT3]** - Ukendteovresættelser, såsom Vandværk (venndværk), Garage (Genrenge), alle “a” oversat med “en”. (det viser sig at være en default google translate oversættelse som brugeren har på sin pc og omskriver ting fundementalt på hjemmesiden.) [USER] - Prøve at vælge let - Bedre indikation af at knapper er til at taste, ikke med mus. **[OG4]**
Forstod du hvad du skulle lære i denne tutorial?	
Har du andre meninger om denne tutorial?	- Tutorial bør brydes op i mindre stykker. **[OT16]**

Brugers umiddelbare indtryk blev påvirket af deres google translate brug.

Koloni

Hvordan var det at navigere i kolonien?	- Stierne er ret intuitive. **[OT7]** - Det var sjovt. **[OC1]**
Kunne du finde hvor du skulle hen?	
Har du andre meninger om denne kolonien?	

Minigame

Hvordan var det at starte et minigame?	- Det var okay at vælge sværhedsgrad og starte spillet **[OC2]**
Hvordan var Hand-Placement checked?	- Skal have mere visuel forklaring. **[OC5]** - Svært at ramme tasterne i tide. **[OC13]** - Teknikken til handplacement checked var okay at forstå, men det er svært at gøre. **[OG3]**

Hvordan var det at spille minigamet?	- Minigame var meget sjovt. **[OA3]**
Er det et minigame du kunne have lyst til at spille igen?	Ja **[OA4]**
Har du andre meninger om minigamet?	Sjov **[OA3]**

Afsluttende

Hvad synes du om hele spillet?	Godt lavet **[OG7]**
Hvordan er det at spille med et tastatur?	- Det tog lidt tid at vænne sig til, men sjovt selvom det tog tid. **[OG3]**
Har du andre meninger om denne kolonien?	

DEVNOTE: Tilføj support for Enter når der laves ny koloni og brugeren står i navnefeltet. (Enter => Confirm)

Afsluttende

Hvad synes du om hele spillet?	
Hvordan er det at spille med et tastatur?	
Har du andre meninger om denne kolonien?	

C.7 Group 1

Mål

- ☑ ~~Menu~~
- ☑ ~~Tutorial~~
- ☑ ~~Koloni~~
- ☑ ~~Minigame (asteroids)~~
  - ☑ ~~Sejr?~~

Feedback

Preface Questions

Gnst. alder

10 (5 personer)

Menu

Hvordan var det at navigere menuen?	
Hvad synes du om menuen udseende?	
Har du andre meninger om menuen?	

Tutorial

Hvordan var det at navigere i tutorial?	- Problemer med at finde ud af at man skal gå viderer fra demo video **[OT1]** - Mere feedback på hvornår en lokation er besøgt i navigation trial **[OT5]** - Problemer med at se navne på lokationer, navigerer tilabe til hjem for at få udsyn **[OT6]** - Tileplacement er locationvard er ucorrect, tilepleacement. - Successfeedback (navigationtrial) **[OT1]** - Mere succefeeback **[OT1]** - Overlap af text, svært at læse **[OT3]** - Svært at vende sig til at bruge tastatur istedet for mus **[OG3]** - Bedre feedback på fejltast - Gør det tydligere at kolonien skal lukkes istedet for at forlade **[OT11]**
Forstod du hvad du skulle lære i denne tutorial?	
Har du andre meninger om denne tutorial?	

“Jeg trykker på dansk og der sker ik en skid” - Bruger X der ikke ser at de kan trykke videre **[OT1]**

Koloni

Hvordan var det at navigere i kolonien?	
Kunne du finde hvor du skulle hen?	
Kunne du finde ud af at åbne din koloni?	- Fjern eller gør placeholder text mere tydeligt. - Clear on enter - always. (Main Action Input default handling)
Har du andre meninger om denne kolonien?	

Minigame

Hvordan var det at starte et minigame?	
Hvordan var Hand-Placement checked?	- De fleste kunne finde ud af det **[OC4] [OMP1] [OMP2]**
Hvordan var det at spille minigamet?	
Hvordan var det at spille minigamet med andre?	
Er det et minigame du kunne have lyst til at spille igen?	
Har du andre meninger om minigamet?	

Afsluttende

Hvad synes du om hele spillet?	
Hvordan er det at spille med et tastatur?	

Har du andre meninger om denne kolonien?	

En bruger oplevede en gigantisk pop up hver gang de valgte Main Action Input. Virker som noget ordblindheds tech. Lidt uheldigt.

Multiplayer virker som en stor success umiddelbart: “Du kan ikke overhale mig XXXX!”

C.8 Group 2

Mål

- ☑ ~~Menu~~
- ☑ ~~Tutorial~~
- ☑ ~~Koloni~~
- ☑ ~~Minigame (asteroids)~~
    - ☑ ~~Sejr?~~

Feedback
Preface Questions
Alder
10-11 år

Spiller du videospil? (Hvis så, hvilke)

Menu

Hvordan var det at navigere menuen?	
Hvad synes du om menuen udseende?	“Rigtigt god - meget spændende” **[OM4]** “Space car”
Har du andre meninger om menuen?	

Tutorial

Hvordan var det at navigere i tutorial?	
Forstod du hvad du skulle lære i	

denne tutorial?	
Har du andre meninger om denne tutorial?	- Forvirring med store og små bogstaver **[OG5]** - Få problemer med at komme viderer, da der er stor kommunikation, mange problemer løses her ved at kommunikere. **[OMP1] [OMP2]** - Få problemer med success feedback. **[OT1]** - Få problemer med navne uden for skærmen. **[OT6]** - Succeffedback **[OT1]** - Brugerne går tilbage og genser video, meget effektivt, bør foreslåes i tutorial. - Forvirring med kolonikode **[OT11]** - Lidt store loadtider. - Man skal lige vende sig til at man skal skrive istedet for at bruge mus. **[OG1]**

Indikationen for hvornår Location Trial er done er ikke tilstrækkelig
Brugerne virker til at være istand til at hjælpe hinanden uden videre guidance.

Koloni

Hvordan var det at navigere i kolonien?	Igen problemer, folk havde styr på det. **[OC1]**
Kunne du finde hvor du skulle hen?	Ja **[OC1]**
Kunne du finde ud af at åbne din koloni?	- Ingen problemer - Folk var meget begestrede for ta kunne se hinanden gå rundt **[OMP3]**. - Det ser ud til at folk gerne vill vide hvem der er hvem (navne på spillerer) **[OMP3]**
Har du andre meninger om denne kolonien?	- Det var fovirrende at der står en kode iforvejen. **[OT11]**

Minigame

Hvordan var det at starte et minigame?	
Hvordan var Hand-Placement checked?	- Tekst placement af accept explainer er ret dårlig **[OC5]** - Ikke stor tekst **[OG5]** - Uklart at begge hænder skal være på tastatur **[OC13]** - Kommunikation gjorde processen meget hurtigere **[OPM1] [OPM2]**
Hvordan var det at	

spille minigamet?	
Hvordan var det at spille minigamet med andre?	
Er det et minigame du kunne have lyst til at spille igen?	
Har du andre meninger om minigamet?	- Input fektet manglede.

Afsluttende

Hvad synes du om hele spillet?	
Hvordan er det at spille med et tastatur?	
Har du andre meninger om denne kolonien?	

C.9 Teacher

Preface Questions

Alder

Spiller du videospil? (Hvis så, hvilke)

Menu

Hvordan var det at navigere menuen?	
Hvad synes du om menuen udseende?	
Har du andre meninger om menuen?	

Tutorial

Hvordan var det at navigere i tutorial?	
Forstod du hvad du skulle lære i denne tutorial?	
Har du andre meninger om denne tutorial?	- Sprog var godt - Navigation ser ud som om det er noget man kan trykke på - Introduktion af keyboard concepttet - Video demo skal være tydeligere, mere feedback på succes - Navigation trial, er ikke intiativ ved første blik - Store og små bogstaver forvirrer - Enter, gøre det endu mere klart - Navne ude for synsvide, navigation trial, navigerede tilbage til hjem. - Ovrelab af tekst - Mere feedback på besøgte navigationer i navigationtrial - Ulkart om man helt kan navigere til på stier - Jo længrer henne i toturial bliver det mere klart - Man har lyst til at trykke på ting - Mere tydeligt at noget er demo go noget er trial - For meget at læse - Merge demo og trial, bryd op stykvis med en mindre opgave af gangen, så åben i demo og så åben i trial osv. - Mere klare formuleringer. - Tydligere fonts, i forhold til store og små bugstaver - Kolonikode forvirrer nu hvor den ikke er forklaret.

Koloni

Hvordan var det at navigere i kolonien?	- Fint at navigere
Kunne du finde hvor du skulle hen?	
Har du andre meninger om denne kolonien?	- Nemt at lave ny koloni - Skal vende sig til at bruge tastatur

Minigame

Hvordan var det at starte et minigame?	- Ligetil, men bliver nogle gange i tvivl om mus eller tastatur. - Meget begestret overfor grafikken, specielt lazer.

Hvordan var Hand-Placement checked?	- Handplacement check er ikke intiutivt - Skal forlkares bedre med grafik
Hvordan var det at spille minigamet?	
Er det et minigame du kunne have lyst til at spille igen?	
Har du andre meninger om minigamet?	- Mangel på læring. - Ikke svært manage. - Den første læsning læringsspil, hop om bord. - Kun individuelle læringsspil. - Der skal være mere multiplayer feedback, hvem har gjort hvad. Altså skal man kunne se bedre hvem der gør hvad. - Ørge for at alle får bidraget og kan være med, mere balance blandt gode og dårlige spillerer.

Afsluttende

Hvad synes du om hele spillet?	
Hvordan er det at spille med et tastatur?	
Har du andre meninger om denne kolonien?	

Appendix D - Multiplayer Group Tests, Full Table of Observations

ID	Group 1	Group 2	Group 3 (n/a)
Main Menu			
OM1			
OM2			
OM3			
OM4			
OM5			
Tutorial			
OT1			
OT2			
OT3			
OT4			
OT5			
OT6			
OT7			
OT8			
OT9			
OT10			
OT11			
OT12			
OT13			
OT14			
OT15			
OT16			
General			

OG1			
OG2			
OG3			
OG4			
OG5			
OG6			
OG7			
OG8			
OG9			
OG10			
Colony			
OC1			
OC2			
OC3			
OC4			
OC5			
OC6			
OC7			
OC8			
OC9			
OC10			
OC11			
OC12			
OC13			
Asteroids Minigame			
OA1			
OA2			
OA3			

OA4			
OA5			
OA6			
OA7			
Mutiplayer			
OMP1			
OMP2			
OMP3			

Appendix E - Development logs

- URSA - DevLog - 01
  https://docs.google.com/presentation/d/1j0zoEk3cd7__cRRzdQxMxEwQp8fxK3EhDV9iUQdK-z4/edit?usp=sharing
- URSA - DevLog - 02
  https://docs.google.com/presentation/d/1yHTcq5bGMgWhoJXkiZAcKisQzK7NRcfiI0chZSU2flc/edit?usp=sharing
- URSA - DevLog - 03
  https://docs.google.com/presentation/d/159t4hGqHTm-LtODXHokytUgDSp9JEIWbR8w_GQbQMm0/edit?usp=sharing
- URSA - DevLog - 04
  https://docs.google.com/presentation/d/1eD-LQcYJw1_YXFr5GFlwTLY3V3V-Mmb9ixsYA-zoQ3U/edit?usp=sharing
- BSC URSA - DevLog 05 - Final MVP
  https://docs.google.com/presentation/d/16b6OMxfeC1k2FLQsf7nxapI6qBpJMDLQ8oedv0-BM6o/edit?usp=sharing

Appendix F - Detailed User Test Catalogue

ID	User 1	User 2	User 3	User 4	User 5	User 6
Main Menu						
OM1						
OM2						
OM3						
OM4						
OM5						
Tutorial						
OT1						
OT2						
OT3						
OT4						
OT5						
OT6						
OT7						
OT8						
OT9						
OT10						
OT11						
OT12						
OT13						
OT14						
OT15						
OT16						
General						
OG1						
OG2						

OG3						
OG4						
OG5						
OG6						
OG7						
OG8						
OG9						
OG10						
Colony						
OC1						
OC2						
OC3						
OC4						
OC5						
OC6						
OC7						
OC8						
OC9						
OC10						
OC11						
OC12						
OC13						
Asteroids Minigame						
OA1						
OA2						
OA3						
OA4						
OA5						

OA6						
OA7						

Appendix G - Full WBS

ID	Type	Description
1	PHA	Research & Analysis
1.0.1	WPK	Architectural Constraints
1.0.1.1	ACT	Research Vitec MV Platform
1.0.1.2	ACT	Research multiplayer architectures
1.0.1.3	ACT	Research storage and retrieval strategy for text translations
1.0.1.4	ACT	Research Database Options
1.0.2	WPK	Touch Typing Research
1.0.2.1	ACT	Touch typing definition and methodologies
1.0.2.2	ACT	Research metrics for recording user performance
1.0.3	WPK	Learning Objectives
1.0.3.1	ACT	Define Touch Typing learning objectives
1.0.3.2	ACT	Develop methods and exercises for addressing certain elements of Touch Typing
2	PHA	Design & Prototyping (Planning)
2.1	**SUB**	**Initial Designs**
2.1.1	WPK	Frontend
2.1.1.1	ACT	Design frontend to be compatible with Vitec MV platform
2.1.1.2	ACT	Design colony display algorithm
2.1.1.3	ACT	Design Auth solution between product and Vitec MV
2.1.2	WPK	Main Backend
2.1.2.1	ACT	Design Language Integration API
2.1.2.2	ACT	Design Player & Location API
2.1.2.3	ACT	Design Asset Retrieval API
2.1.3	WPK	Databases
2.1.3.1	ACT	Design Language Database
2.1.3.2	ACT	Design Player & Colony Database

2.1.3.3	ACT	Asset Management Pipeline (devtooling)
2.1.4	WPK	Multiplayer Backend
2.1.4.1	ACT	Design Multiplayer Backend API
2.1.4.2	ACT	Design Multiplayer Backend Architecture
2.1.5	WPK	Minigames
2.1.5.1	ACT	Define Locations
2.1.5.2	ACT	Design Minigame Framework
2.1.5.3	ACT	Design First Minigame Concept
2.1.5.4	ACT	Define Multiplayer Events for First Minigame
2.1.6	WPK	Asset Mockups
2.1.6.1	ACT	Define needed assets
2.1.6.2	ACT	Make mockups for test data and delivery
2.2	**SUB**	**Prototyping**
2.2.1	WPK	Frontend
2.2.1.1	ACT	Prototype Frontend Angular integration
2.2.1.2	ACT	Prototype Frontend Multiplayer part
2.2.2	WPK	Databases
2.2.2.1	ACT	Prototype Player & Colony Database
2.2.3	WPK	Multiplayer Backend
2.2.3.1	ACT	Prototype Multiplayer Backend
2.2.4	WPK	Integration Testing of Prototypes
2.2.4.1	ACT	Prototype Frontend Multiplayer part
2.3	**SUB**	**Reviewing Initial Designs**
2.3.1	WPK	Frontend Design
2.3.1.1	ACT	Review Initial Design based on findings from prototyping
2.3.2	WPK	Review Main Backend API
2.3.2.1	ACT	Review Initial Design based on findings from prototyping
2.3.3	WPK	Multiplayer Design

2.3.3.1	ACT	Review Initial Design based on findings from prototyping
2.3.4	WPK	Colony & Player Database Structure
2.3.4.1	ACT	Review Initial Design based on findings from prototyping
2.3.5	WPK	Language Database Structure
2.3.5.1	ACT	Review Initial Design based on findings from prototyping
2.3.6	WPK	Revisit designs of indirectly affected entities
2.3.6.1	ACT	Review Initial Design based on findings from prototyping
3	PHA	Implementation (Execution)
3.1	**SUB**	**Implementation**
3.1.1	WPK	Database Servers
3.1.1.1	ACT	establish deployment of Database servers in Dev Environment
3.1.1.2	ACT	Configure Language DB with UTF16 encoded strings
3.1.1.3	ACT	Database creation sql script for Player & Colony database
3.1.1.4	ACT	Database creation sql script for Language Database
3.1.2	WPK	Main Backend
3.1.2.1	ACT	Backend support for UTF16 encoded strings
3.1.2.2	ACT	Skeleton implementation of API
3.1.2.3	ACT	Setup OpenAPI/Swagger integration
3.1.2.4	ACT	Setup database integration(s) (establish connection(s))
3.1.2.5	ACT	Implement Asset Endpoints
3.1.2.6	ACT	Implement Colony Data Endpoints
3.1.2.7	ACT	Implement Player Data Endpoints
3.1.2.8	ACT	Implement language & translation Endpoints
3.1.2.9	ACT	Implement multiplayer session initiation flow (“Opening a Colony”)
3.1.3	WPK	Frontend
3.1.3.1	ACT	Frontend text-field support for UTF 16 encoded strings
3.1.3.2	ACT	Display text abstraction (language)
3.1.3.3	ACT	Asset retrieval abstraction

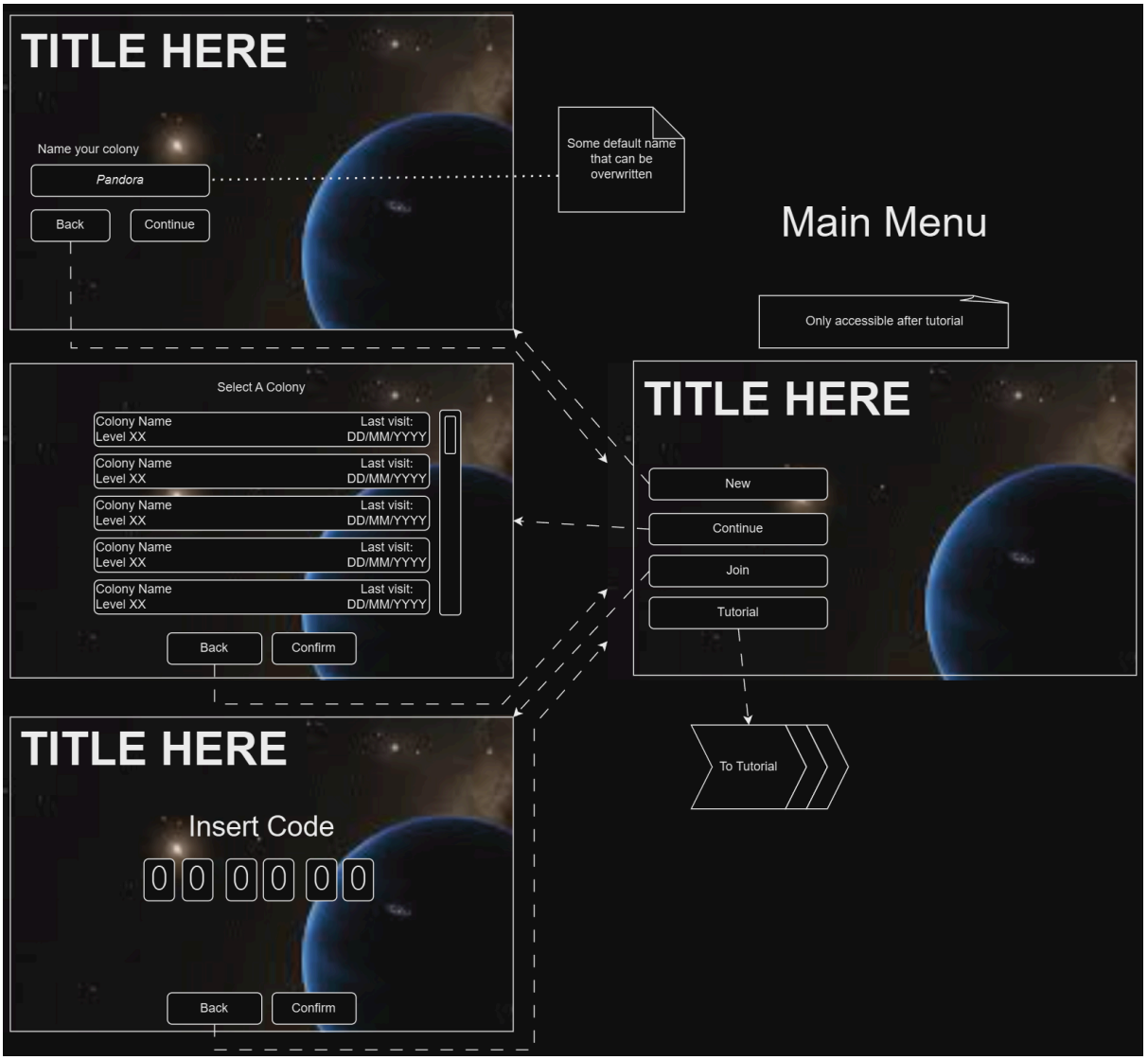
3.1.3.4	ACT	Colony layout generation algorithm (establish the colony, assets, and locations on first colony load per session)
3.1.3.5	ACT	Implementation of centralized event handling
3.1.3.6	ACT	Implementation of EventChannel
3.1.3.7	ACT	Handling of all “Colony Events”
3.1.3.8	ACT	Movement (as owner and as guest)
3.1.3.9	ACT	Functionality supporting first minigame
3.1.4	WPK	Multiplayer Backend
3.1.4.1	ACT	Setup server with pub/sub functionality
3.1.4.2	ACT	Implement “topics as lobbies”
3.1.4.3	ACT	Implement lobby creation endpoint
3.1.4.4	ACT	Implement event handling of “Colony Events”
3.1.4.5	ACT	Filter received events based on rules (based on who sent some event, should it be replicated to other ppl in the lobby)
3.1.4.6	ACT	Implement method for mounting/unmounting minigame game loops for Hybrid Peer-To-Peer digital twin approach
3.1.5	WPK	Implement First Game Across Services
3.1.5.1	ACT	Upload required assets to database
3.1.5.2	ACT	Implement frontend custom viewport for minigame
3.1.5.3	ACT	Implement frontend event handling of minigame events
3.1.5.4	ACT	Implement core game loop in multiplayer backend
3.1.6	WPK	DA Translation
3.1.6.1	ACT	Danish translation for all display text defined at this point
3.2	**SUB**	**Unit Testing**
3.2.1	WPK	Unit Test Test-Suite Main Backend
3.2.1.1	ACT	>95% code coverage on unit tests
3.2.1.2	ACT	Full coverage of endpoint behavior using Mocks
3.2.2	WPK	Unit Test Test-Suite Frontend
3.2.2.1	ACT	>95% code coverage on unit tests
3.2.2.2	ACT	>95% coverage on DOM events and DOM elements (using jest or other

		DOM testing tools)
3.2.3	WPK	Unit Test Test-Suite Multiplayer Backend
3.2.3.1	ACT	>95% code coverage on unit tests on game loops and other non-PUB-SUB functionality
3.2.3.2	ACT	Attempt to gain more coverage using mocks
3.3	**SUB**	**Integration Testing**
3.3.1	WPK	Persistence
3.3.1.1	ACT	The system can recognize and authorize the same user between sessions
3.3.1.2	ACT	Persistent User Data & Progress
3.3.2	WPK	Platform Compatibility
3.3.2.1	ACT	The frontend is correctly served through the Vitec MV platform
3.3.1	WPK	Language
3.3.3.1	ACT	The frontend requests internationalization catalogues.
4	PHA	Deployment / Roll-out
4.1	**SUB**	**CI/CD Workflows**
4.1.1	WPK	Database Deployment
4.1.1.1	ACT	Deployment to test environment in Vitec MV cloud
4.1.1.2	ACT	(Optional) Local deployment to containerization platform
4.1.1.3	ACT	Automatic pulls from GitHub into cloud dev environment (only if local unit tests pass)
4.1.2	WPK	Frontend Deployment
4.1.2.1	ACT	Deployment to test environment in Vitec MV cloud
4.1.2.2	ACT	(Optional) Local deployment to containerization platform
4.1.2.3	ACT	Automatic pulls from GitHub into cloud dev environment (only if local unit tests pass)
4.1.3	WPK	Main Backend Deployment
4.1.3.1	ACT	Deployment to test environment in Vitec MV cloud
4.1.3.2	ACT	(Optional) Local deployment to containerization platform
4.1.3.3	ACT	Automatic pulls from GitHub into cloud dev environment based on unit test results

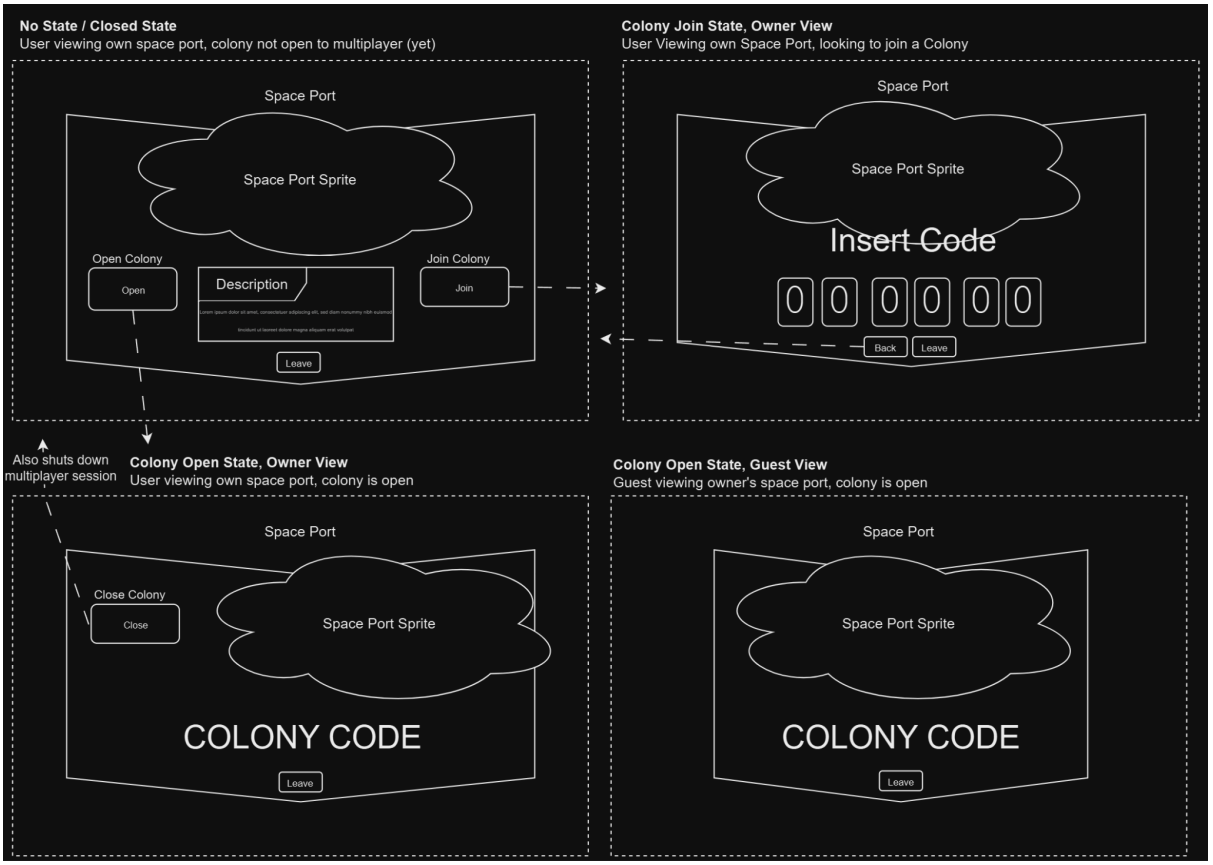
4.1.4	WPK	Multiplayer Backend Deployment
4.1.4.1	ACT	Deployment to test environment in Vitec MV cloud
4.1.4.2	ACT	(optional) Local deployment to containerization platform
4.1.4.3	ACT	Automatic pulls from GitHub into cloud dev environment based on unit tests
4.1.4.4	ACT	Estabilsh instance cluster (or other load balancing)

Appendix H - Graphical Mockups

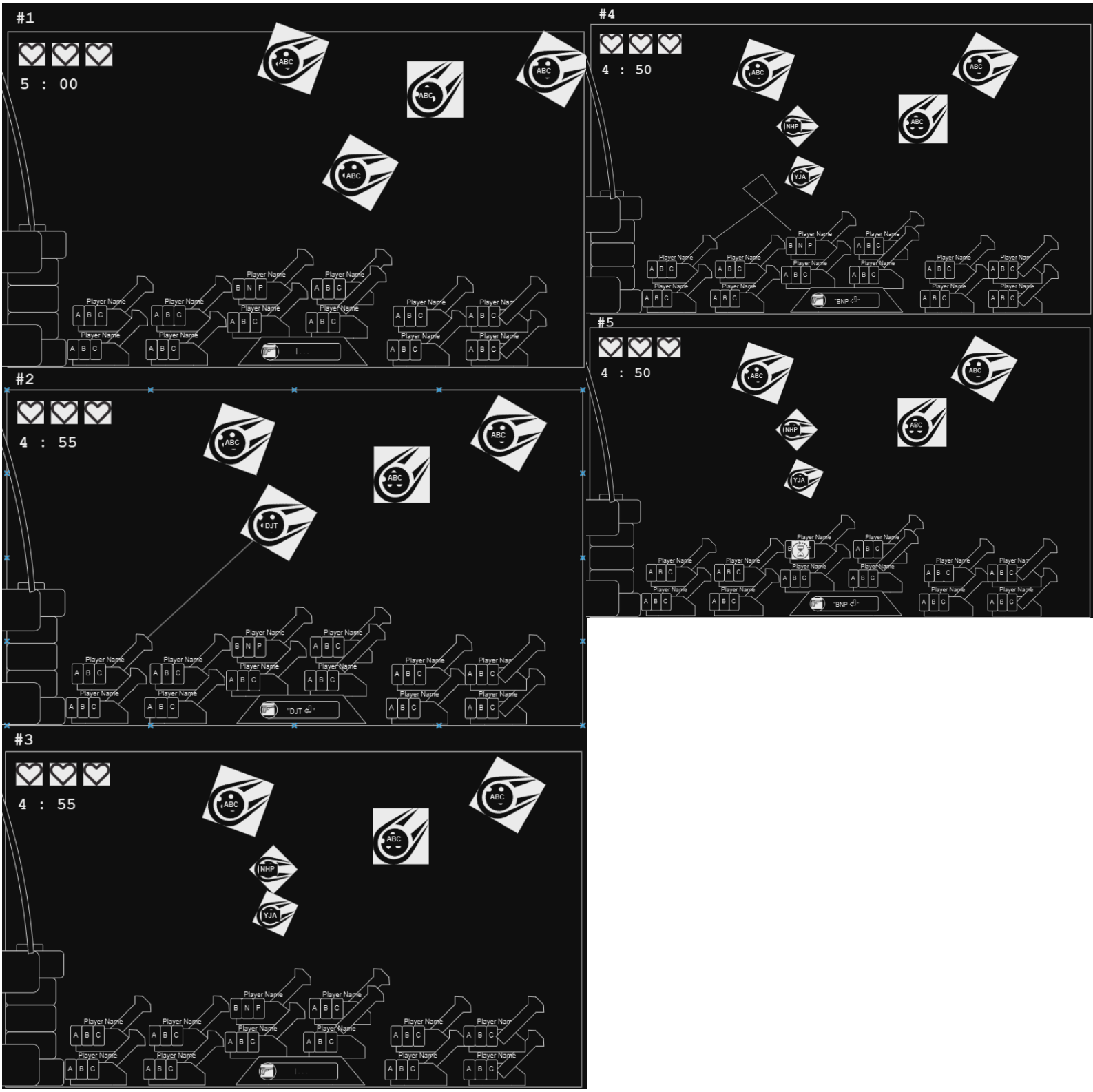
H.1 Main Menu



H.2 Spaceport Location Card



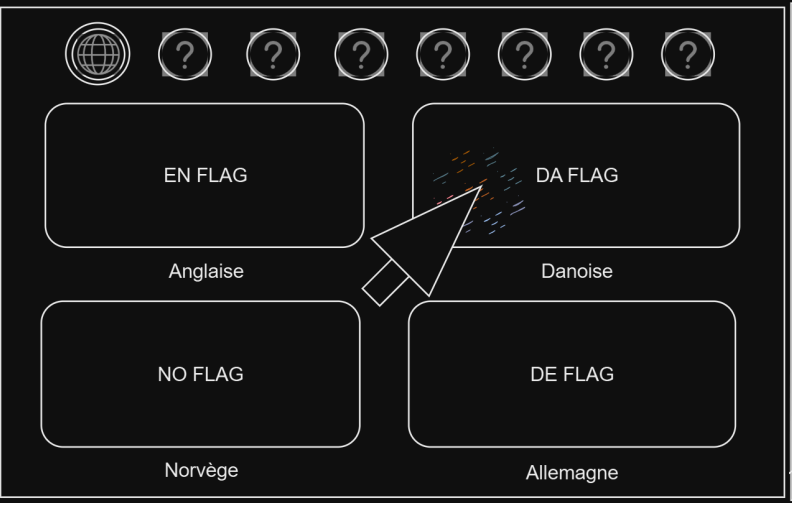
H.3 Asteroids Minigame



H.4 Tutorial

#1 Language Select

Name of country in own language
Flag of country
Mouse input

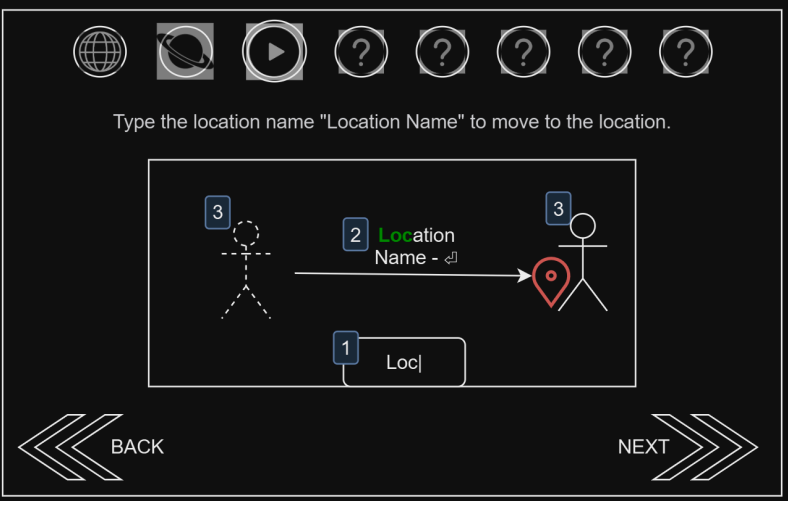


#2 Welcome

Obligatory welcome page

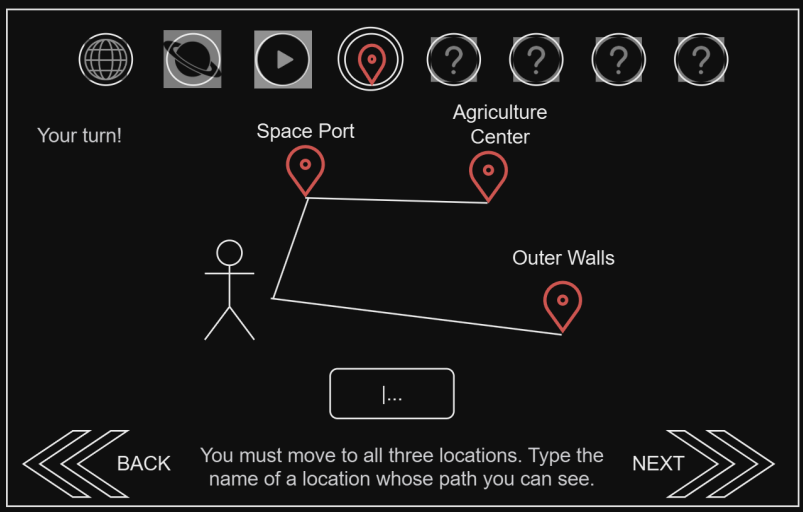
#3 Navigation Demo

1 "Main Input is shown"
2 "Some text appears in it"
2 "The text matches some location displayed in the demo"
2 "As letters in the name of the location is written, that letter on the location, lights up"
2 "When the location name is complete, ENTER is pressed, and the main input gives woosh vfx"
3 "The main input is cleared, and the player sprite moves to the location"



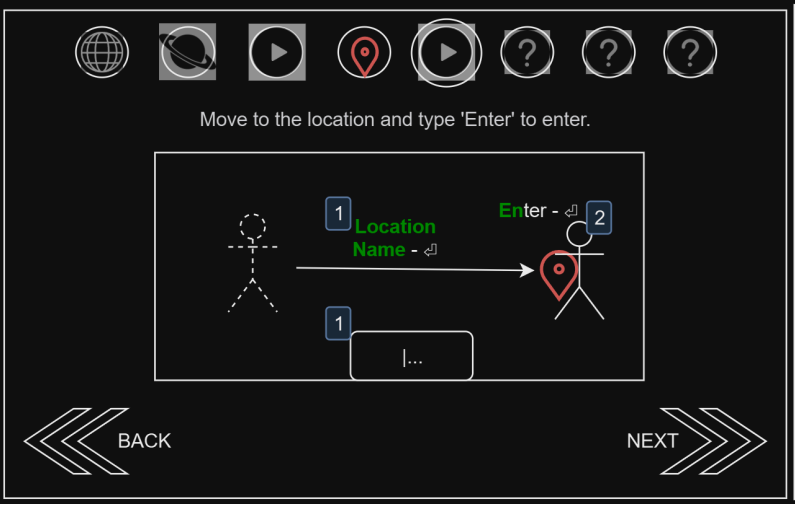
#4 Navigation Trail

Now its the users turn to walk to any 3 locations
The locations are arbitrary, but the user most visit all locations to continue



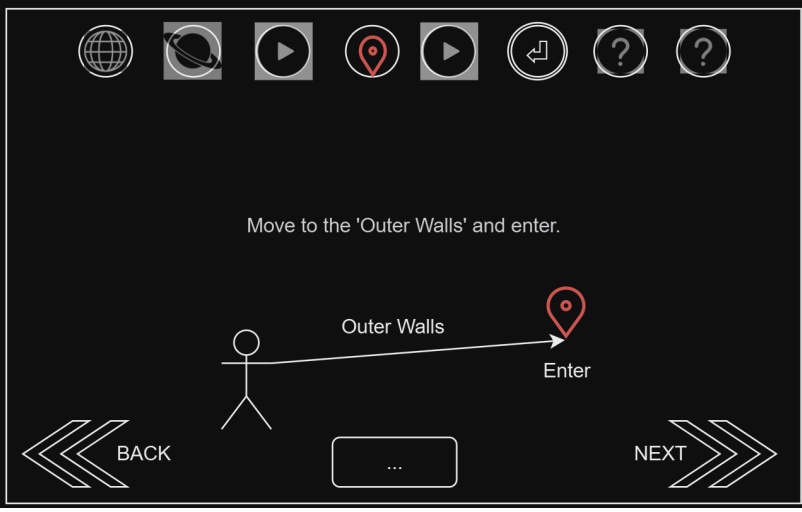
#5 Location Demo

1 "Move to a location by typing 'Location Name'"
2 "Once at location, type 'Enter' to enter the location"



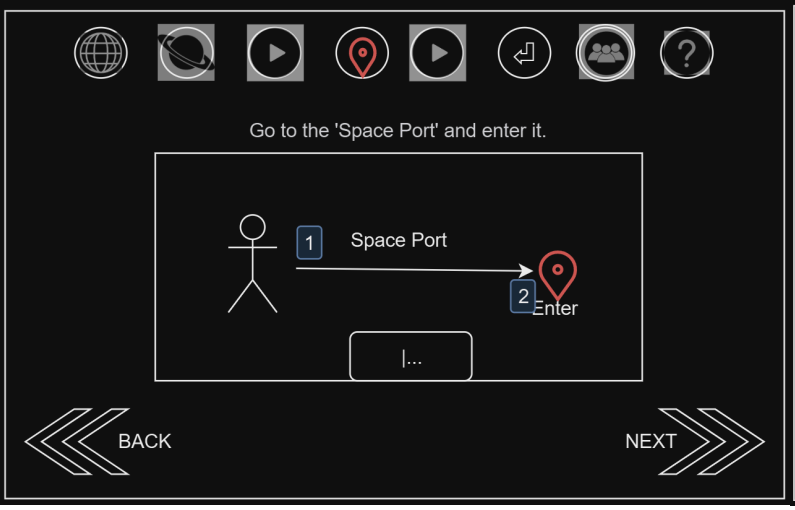
#6 Location Trial

Go to "Location_Name" and type "Enter" to Enter



1 "Go to 'Space Port'"
2 "Enter the 'Space Port'"
3 "Once inside the 'Space Port', type 'Open' to open your colony to guests"
4 "Type 'Close' to close the Colony and return to the Space Port menu"
5 "Type 'Join' to join another players Colony"
6 "Type a Colony Code of six numbers, such as '123456'"

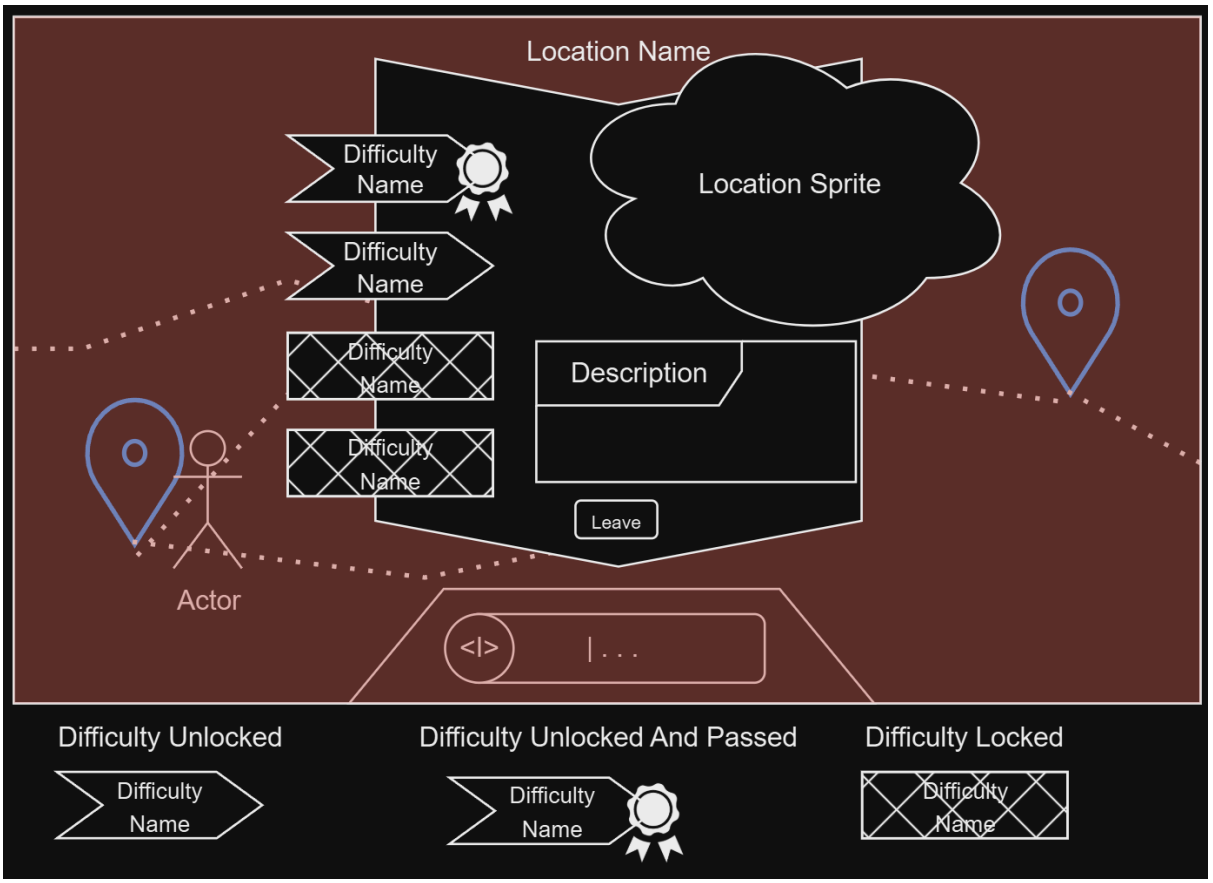
#7 Demo Multiplayer



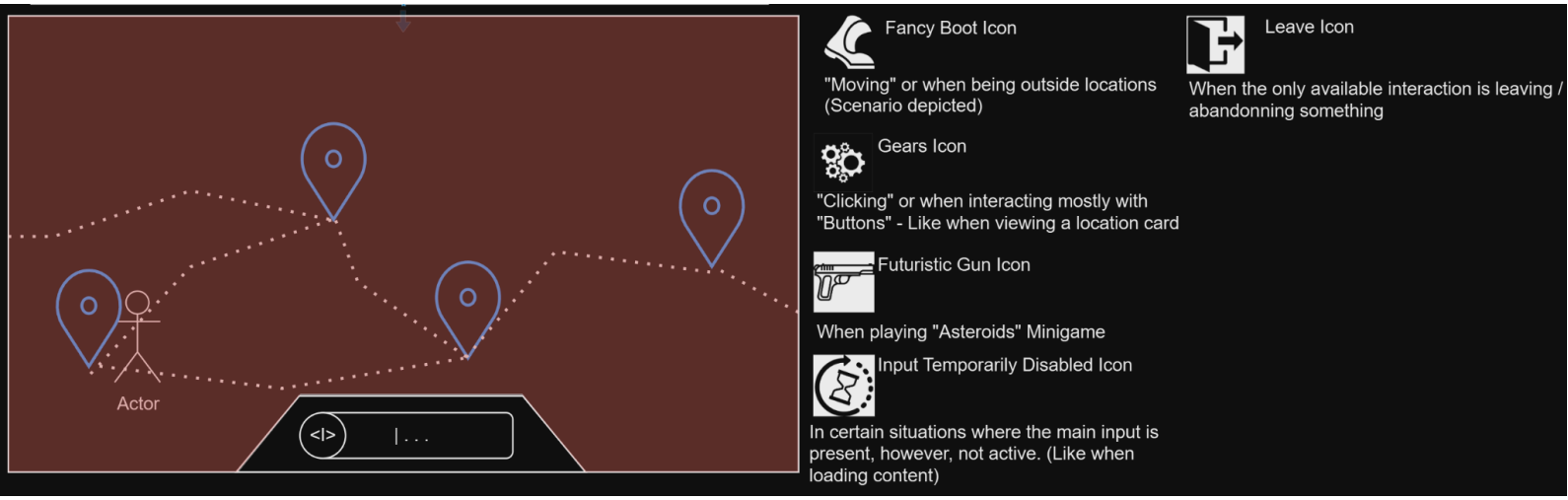
#8 Tutorial Complete

The player has completed the tutorial and recieve the "Toturial Complete" achievement.
(Something used by the rest of the system)

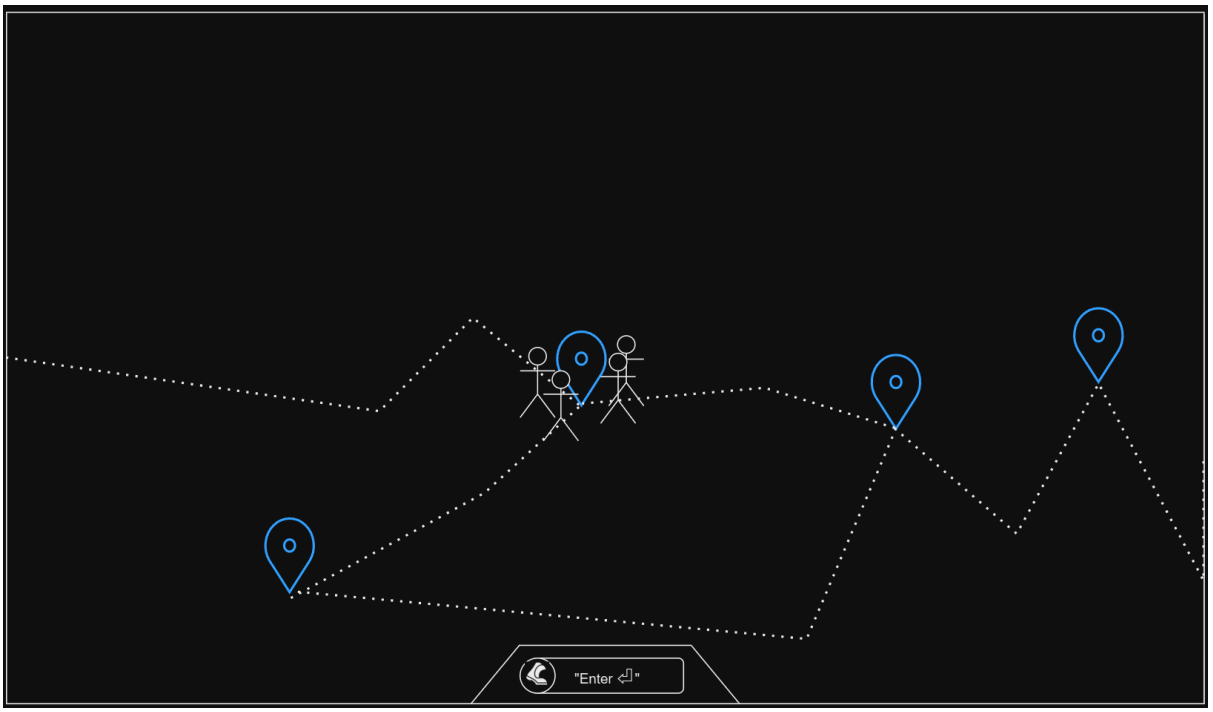
H.5 Generic Location Card



H.6 Main Action Input

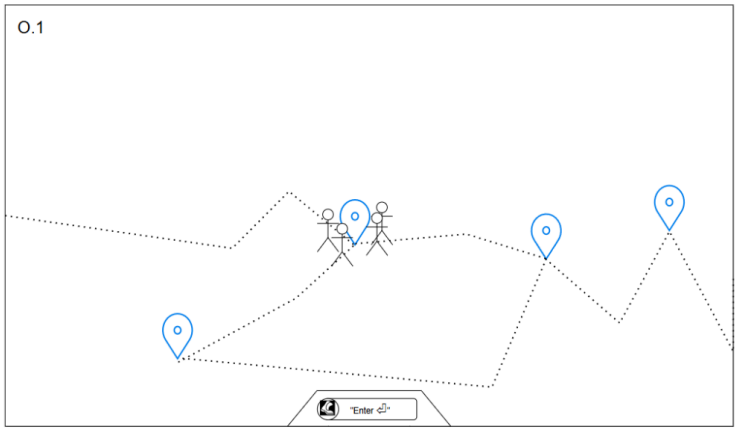


H.7 Path Graph Example



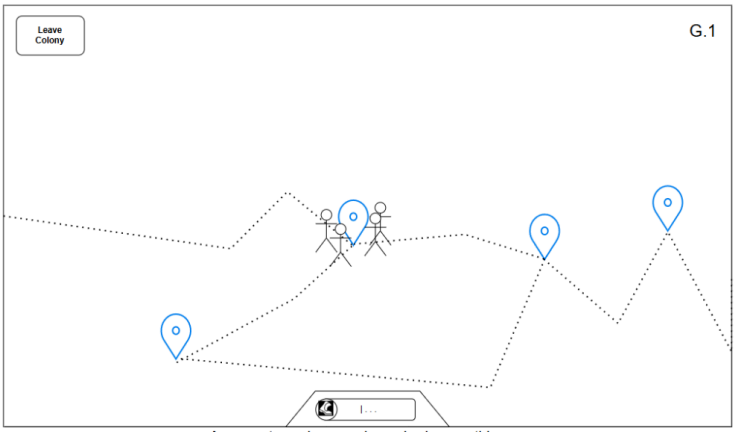
H.8 Minigame Initiation

Colony Owner Perspective

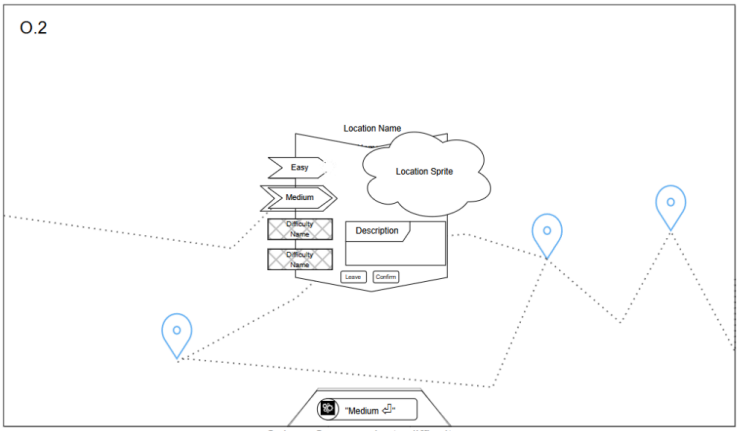


Colony Owner enters Location

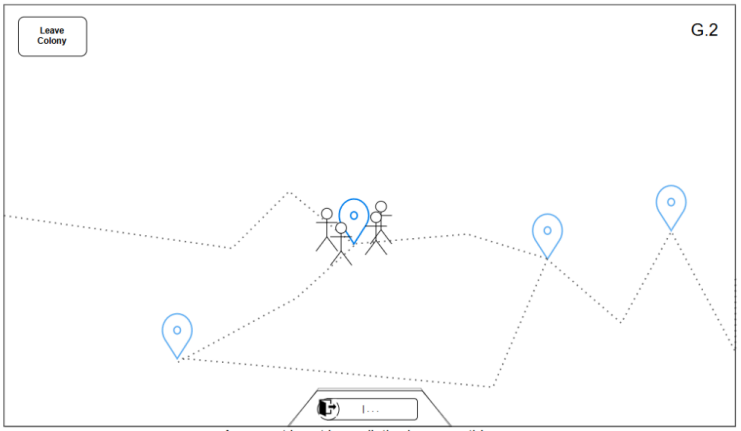
Guest Perspective



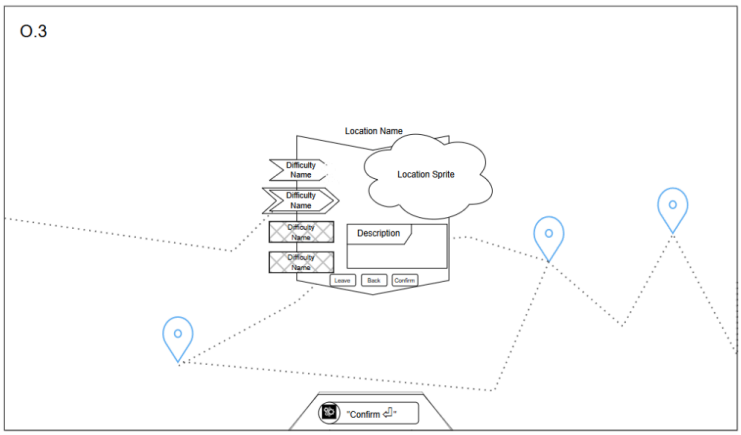
Any guest may be any where viewing any thing



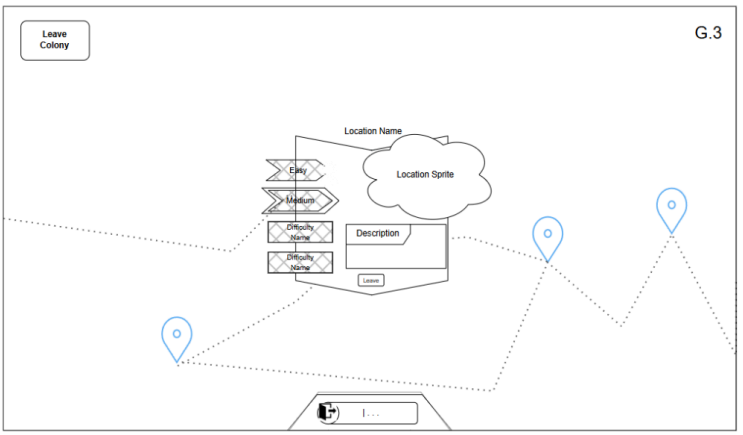
Colony Owner selects difficulty



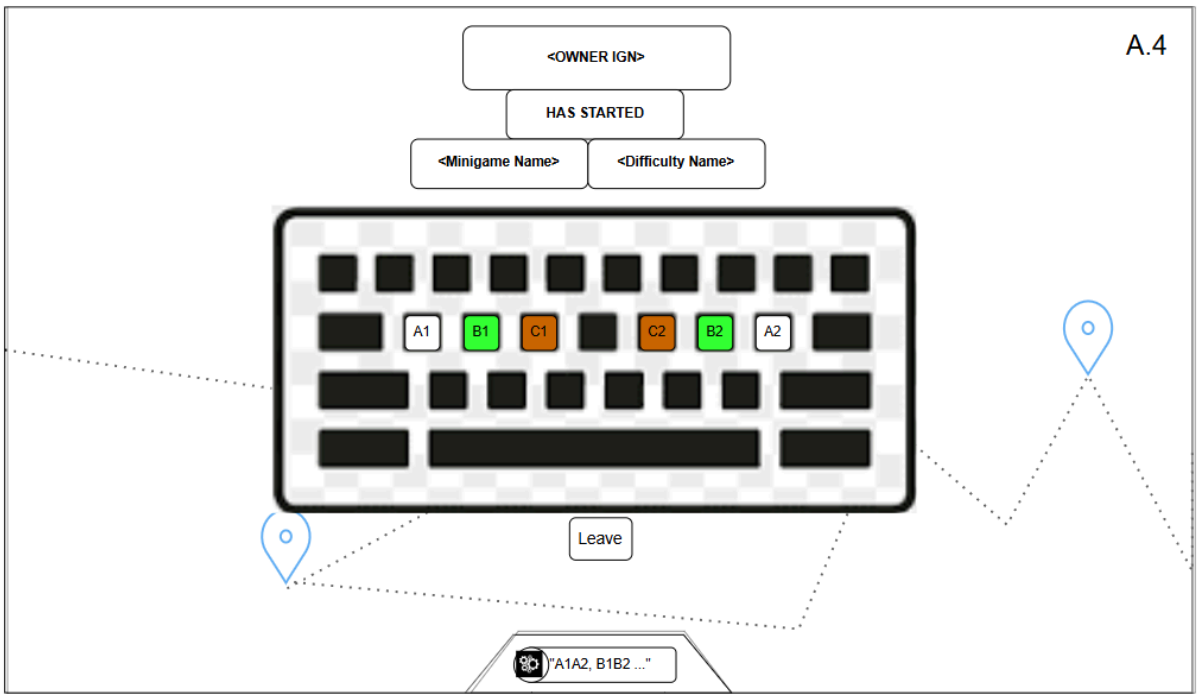
Any guest is not immediatly shown anything



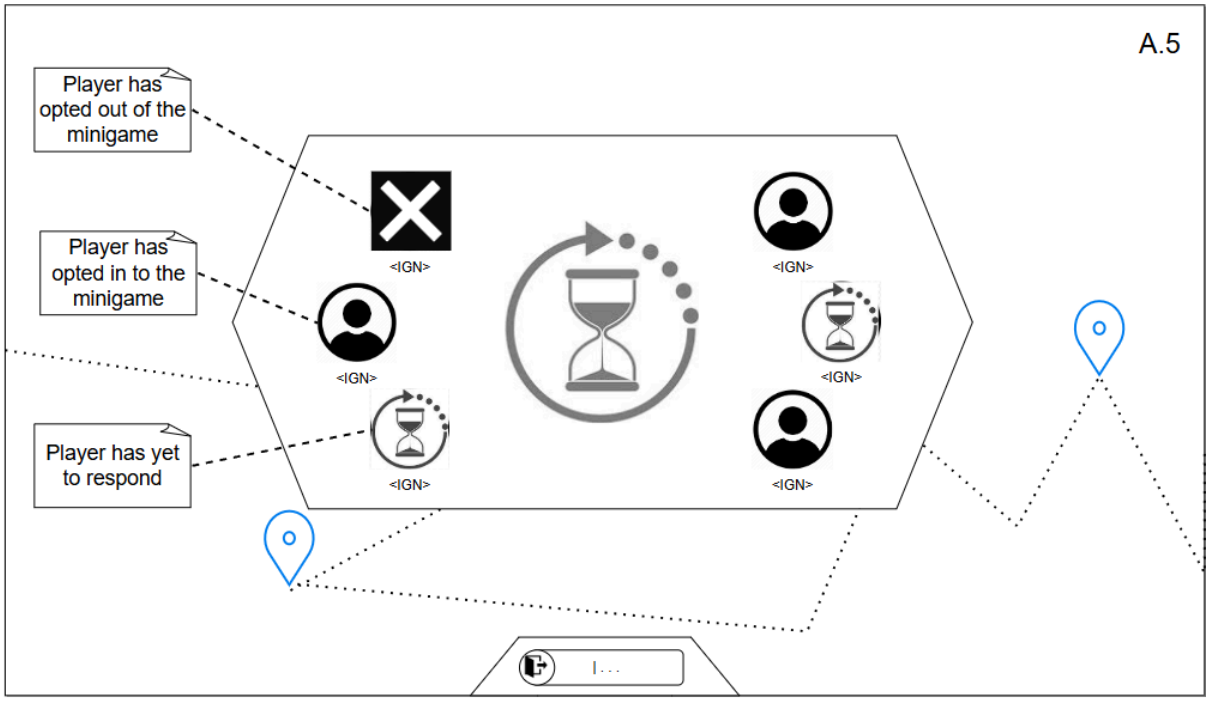
Colony owner confirms difficulty selected



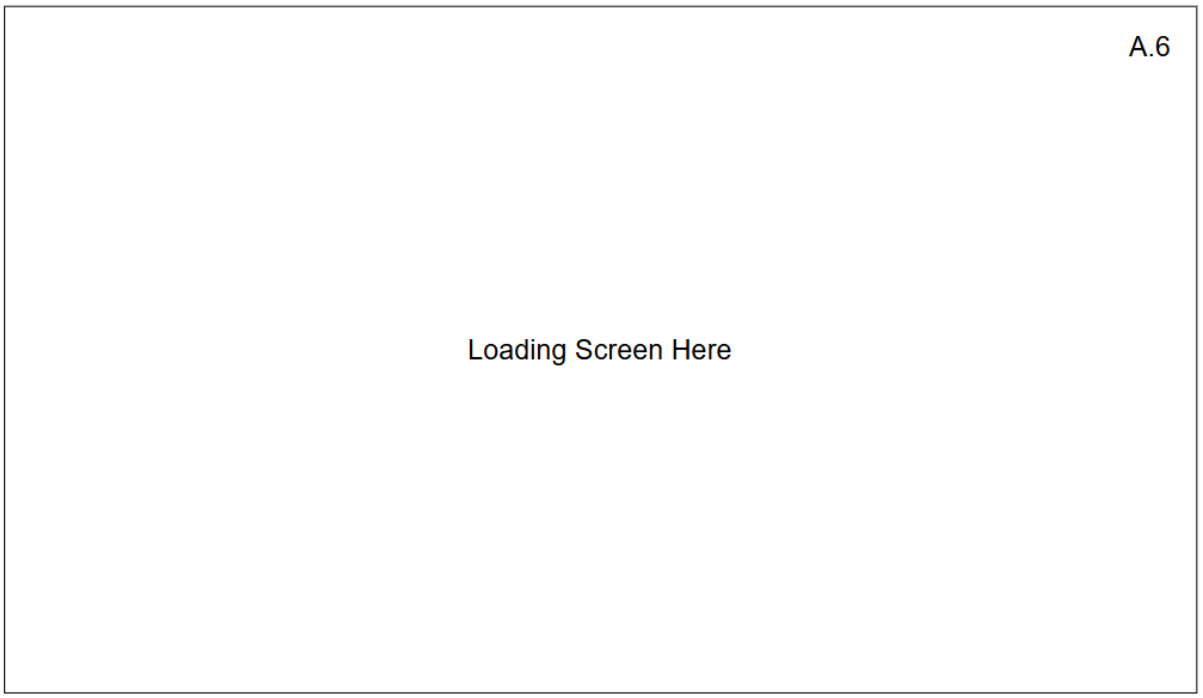
Any guest can do any thing



Player Join Activity is emitted on passing the check.
Player Abort emitted on Leave

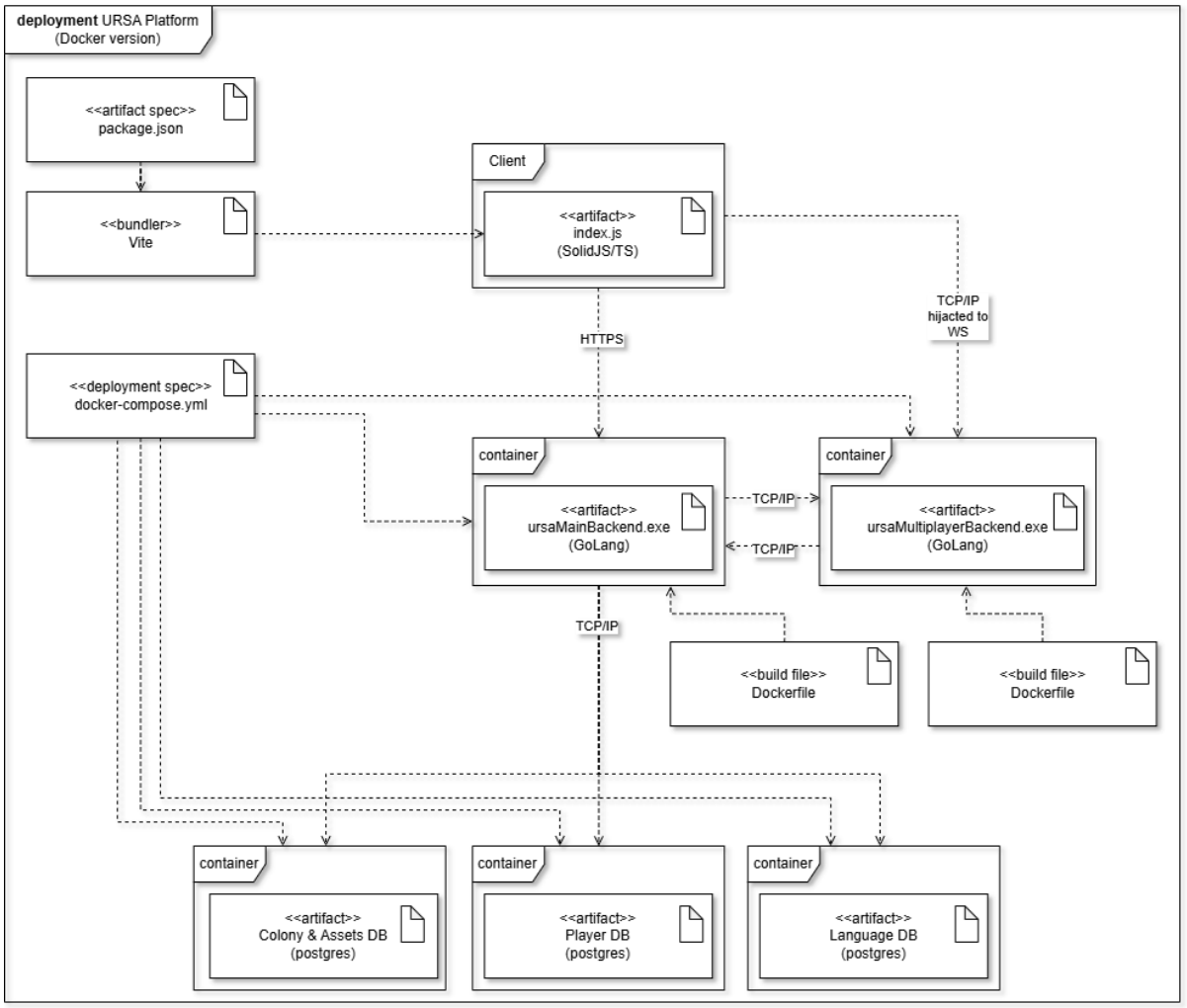


The state of all other possible participants are updated continuously

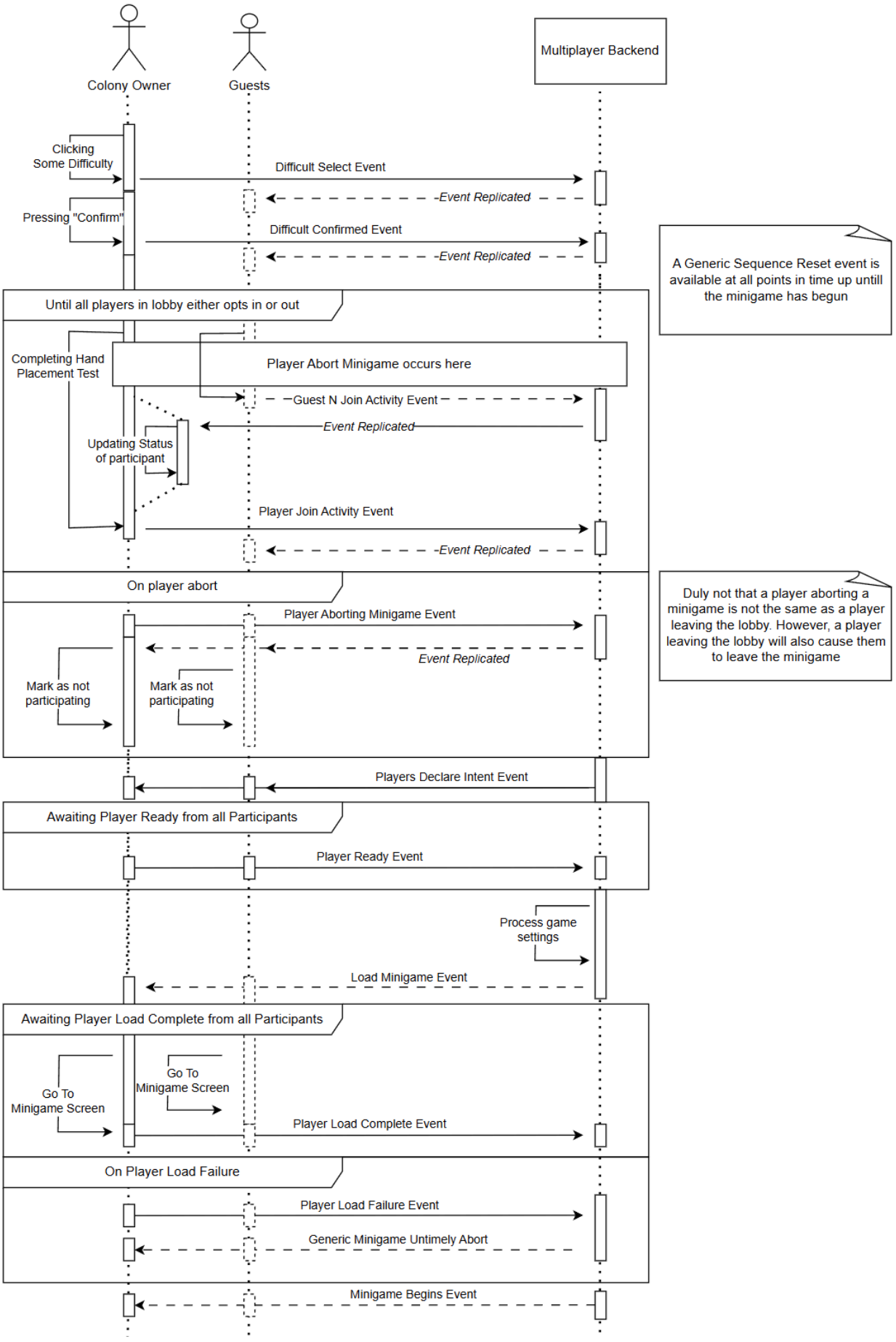


While awaiting Player Load Complete from all participants

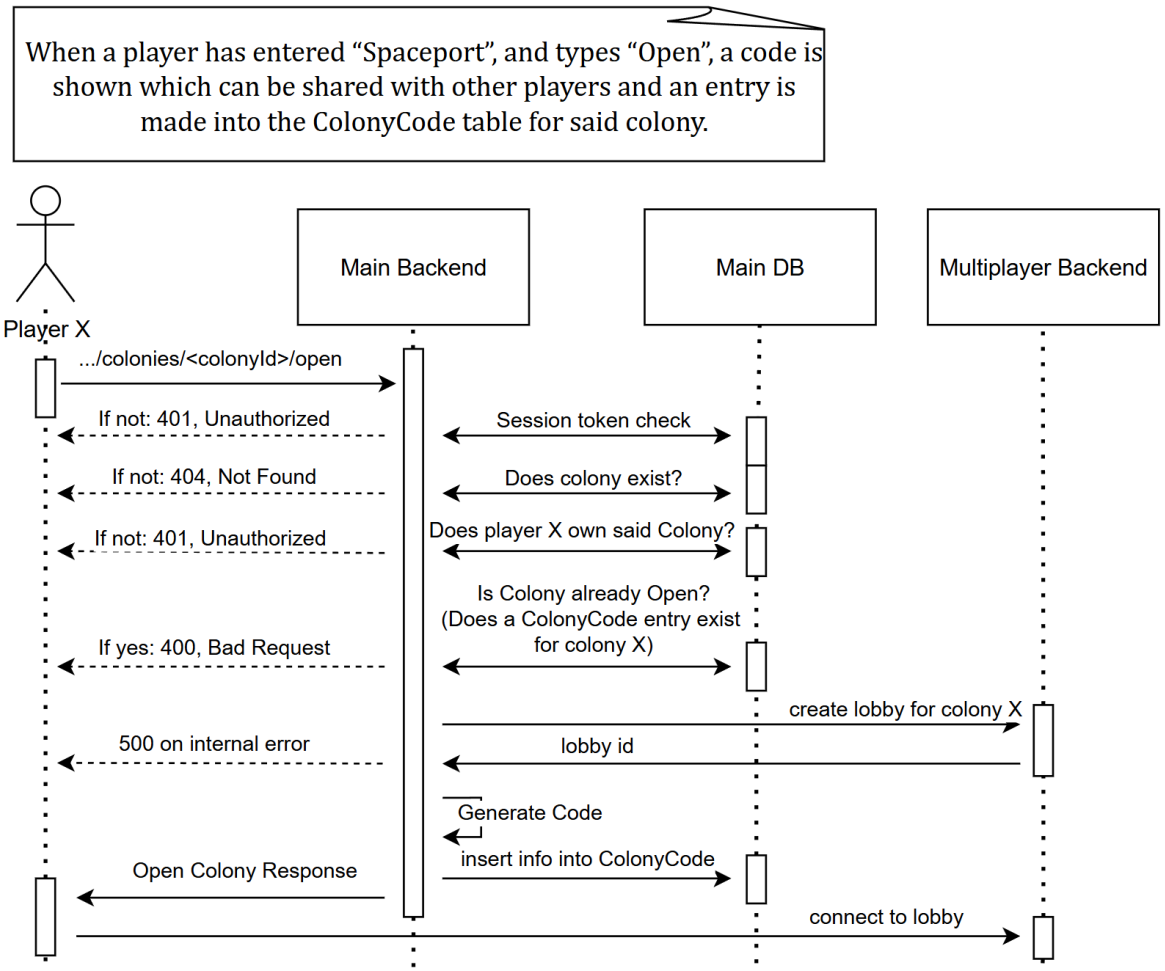
Appendix I - Deployment Diagram



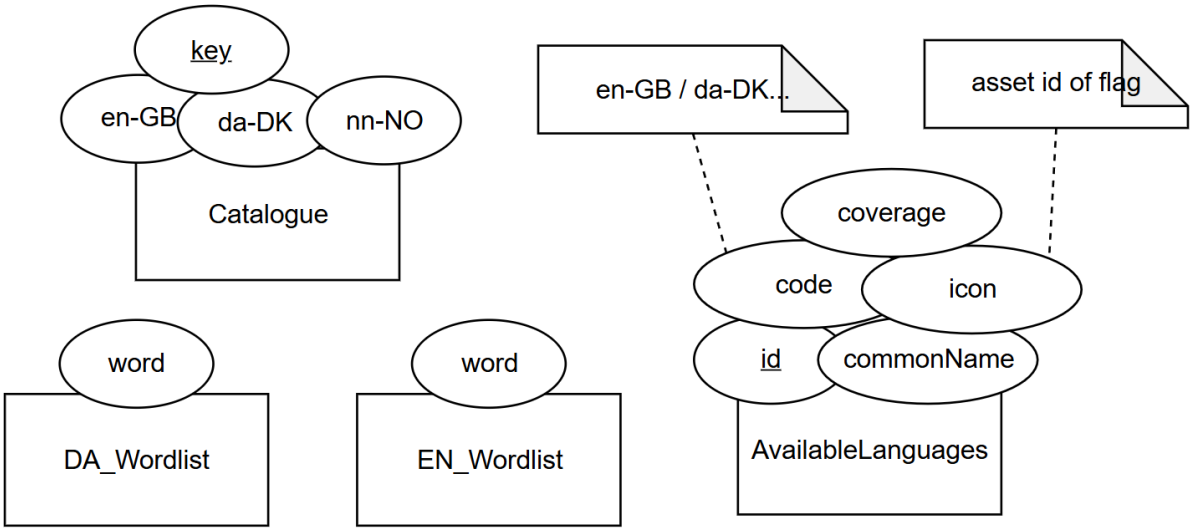
Appendix J - Minigame Initiation Sequence



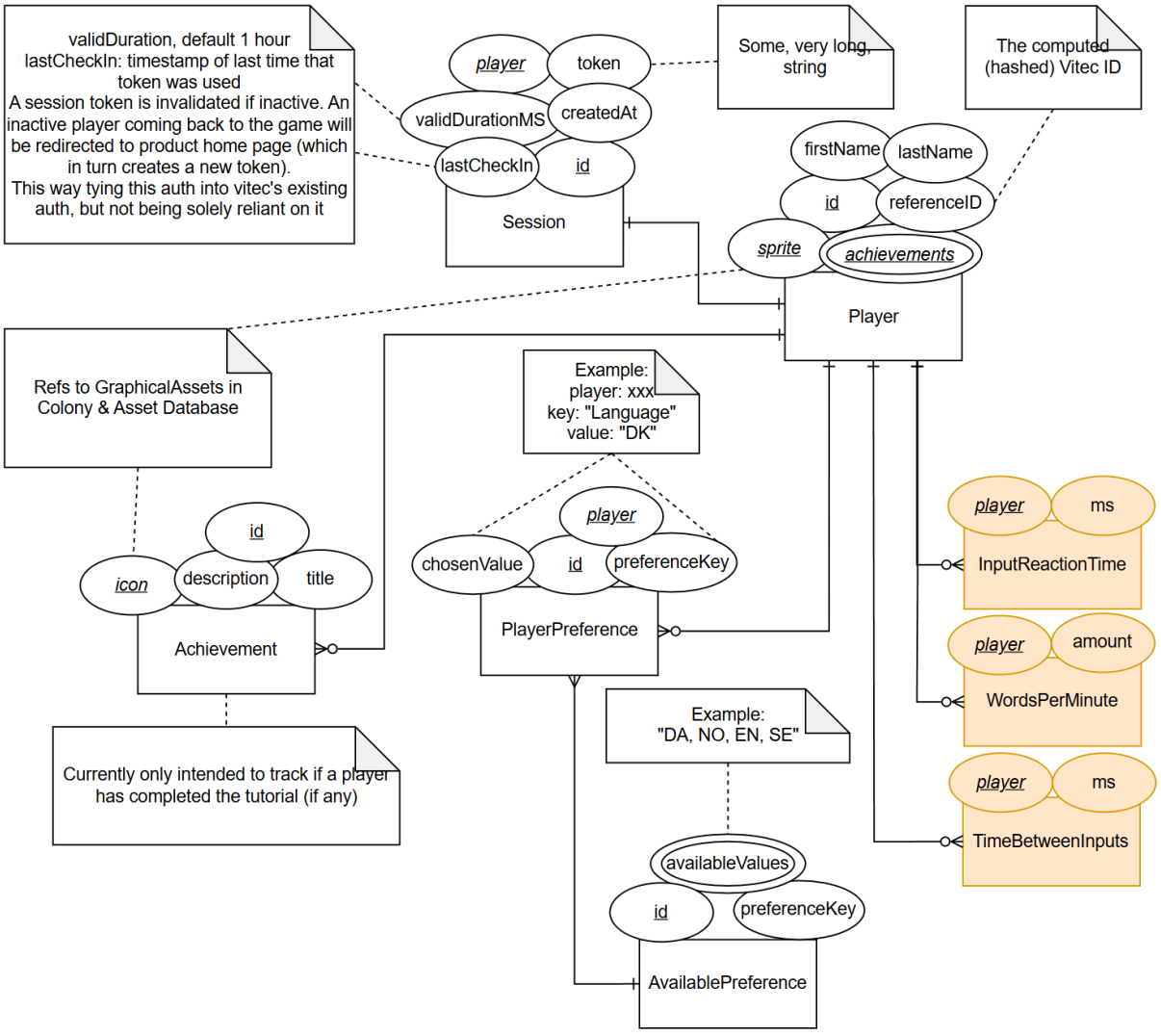
Appendix K - Open Colony Sequence Diagram



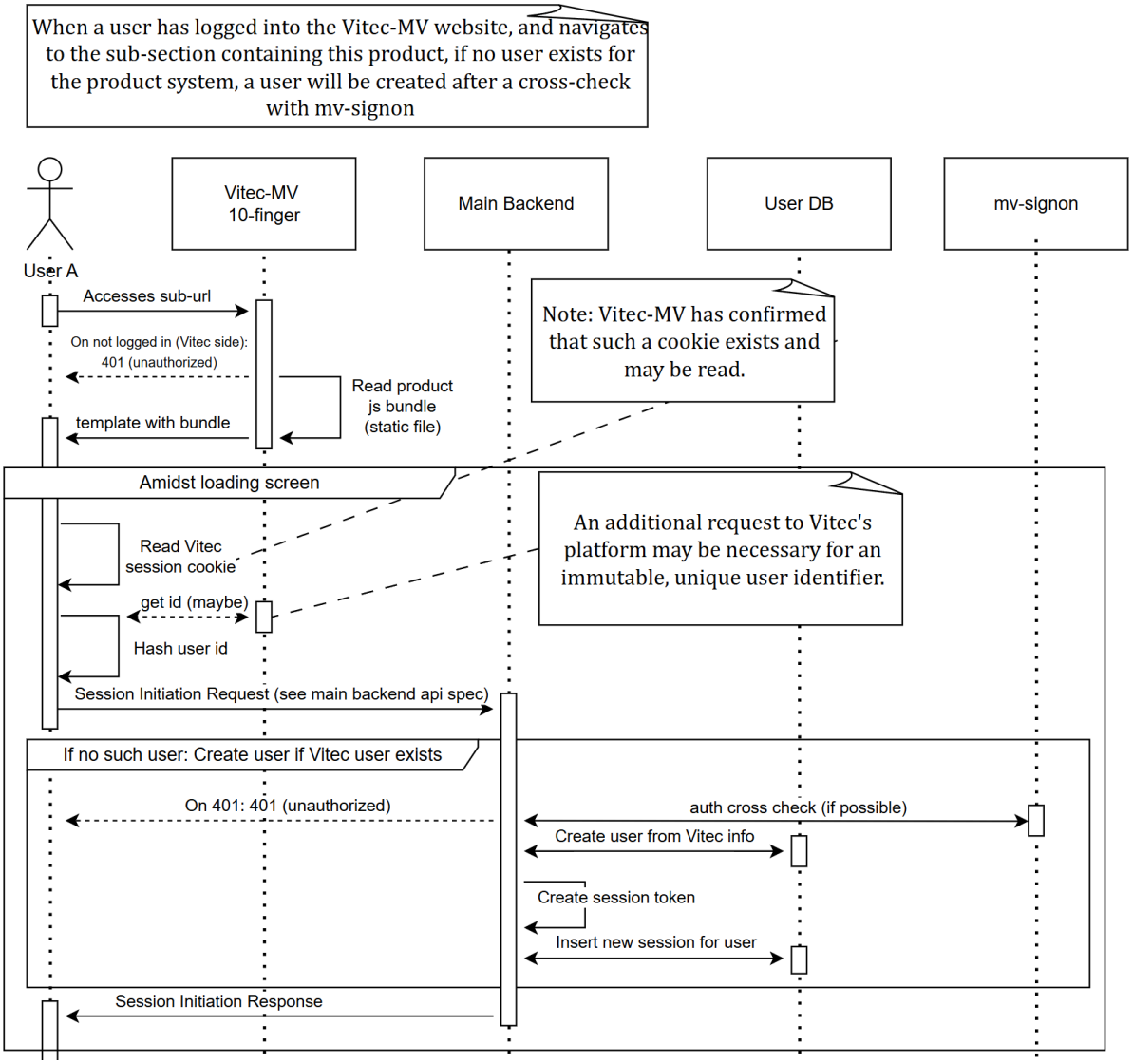
Appendix L - Language DB ER Diagram



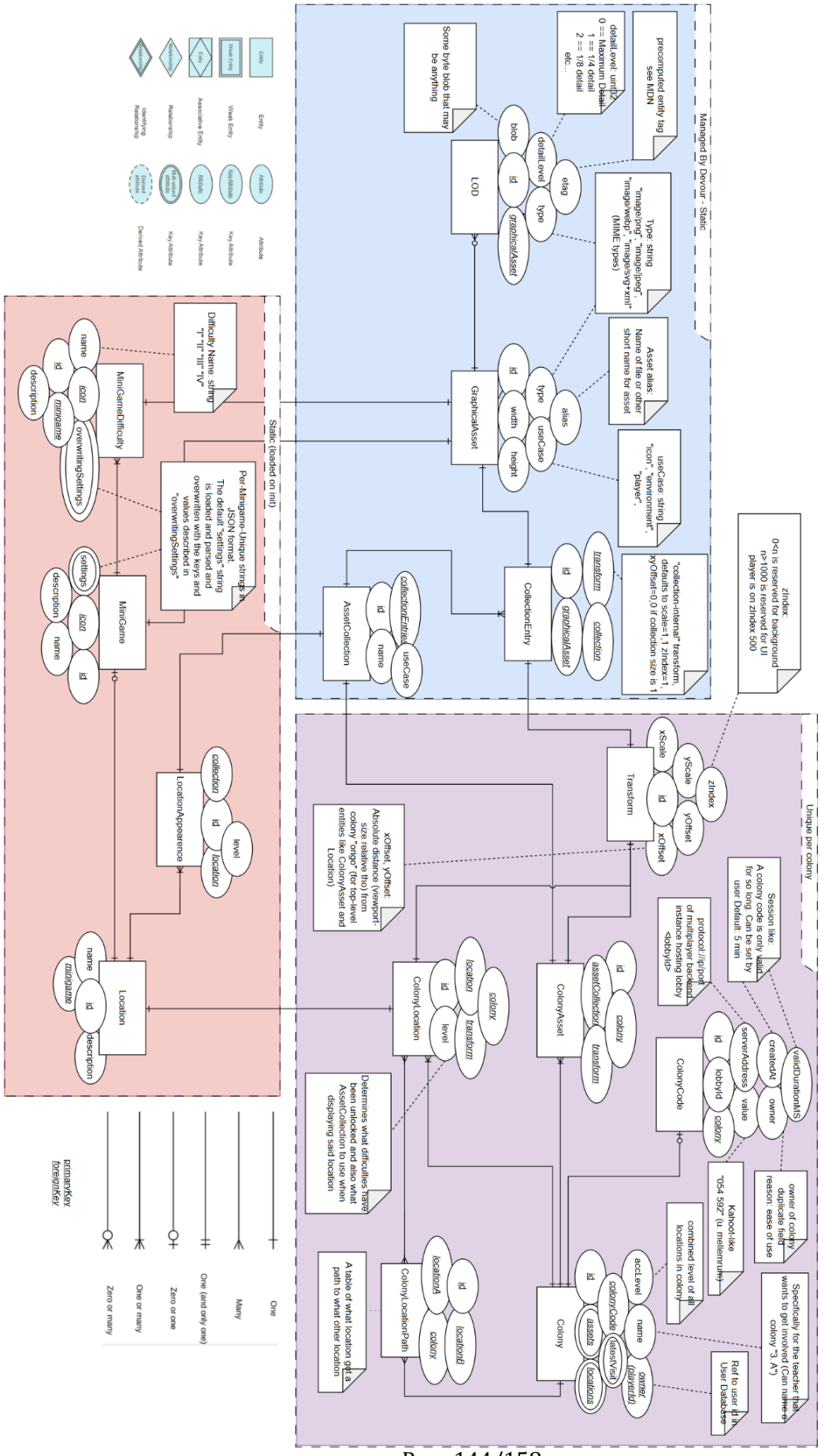
Appendix M - User DB ER Diagram



Appendix N - Upon User Visit Sequence Diagram



Appendix O - Colony & Asset ER Diagram



Appendix P - Main Backend API Specifications

General Notes
Described as some function of some inputs, leading to the approximate DB query and response dto (using expanded TS syntax).
Also includes expected URL extension
"https://host:port/api/vX/<extension>"
All routes, when faulty may include the header "OTTE-Debug-Header" on the response for further debugging

Auth Notes
Connection is TLS secured
Also all routes (but Initiate Session) expect the header: "URSA-Token" to be present with valid user session token as issued as result of the Initiate Session endpoint.

Grant Player Achievement [AA]

(POST) URL: <base-URL>/player/<playerId>/achievement/<achievementId>
(playerId, achievementId) =>
INSERT INTO Player (achievements) VALUES (...achievementID) WHERE id IS <playerId> =>
200 - all is well
400 - error in request
404 - no such player | no such achievement

Set Player Preference [AB]

(POST) URL: <base-URL>/player/<playerId>/preferences
body: { key: string; value: string } =>
=>
200 - all is well
400 - error in request
404 - no such player

Health Check [AC]

(GET) URL: <base-URL>/health =>

```
{
    "multiplayerStatus": {
        "status": bool,
        "lobbyCount": uint32
    },
    "colonyDBStatus": bool,
    "languageDBStatus": bool,
    "playerDBStatus": bool,
    "statusMessage": string,
    "timestamp": time.Time
}
```

Initiate Session Response [AD]

(POST) URL: <base-URL>/session
([body] SessionInitiationDTO) =>
Cross check with Vitec auth "do they be whom they say they are"
=>

```
{
    "internalID": uint32,
    "token": string
}
```

Player Info Response [AE]

URL: <base-URL>/player/<playerId>
(playerId) =>
Select * Player WHERE id is <playerId> =>

```
{
    "id": uint32,
    "firstName": string,
    "lastName": string,
    "sprite": uint32,
    "achievements": uint32[],
    "hasCompletedTutorial": uint1 // computed field
}
```

Player Preferences Response [AF]

URL: <base-URL>/player/<playerId>/preferences
(playerId) =>
Select pref from PlayerPreferences WHERE player is <playerId> JOIN (availableValues) from AvailablePreferences where key is pref.key =>

```
{
    "preferences": [
        {
            "id": uint32,
            "key": string,    //example: "Language"
            "chosenValue": string,    //example: "DK"
            "availableValues": string[]
        }
    ]
}
```

Intl. Catalog Response [AG]

URL: <base-URL>/catalog/<languageCode>?keys=string[]
//Keys parameter optional. If excluded, the whole catalogue is returned
(languageCode, keys?) =>
Select (key, <langAbbrevation>) from Catalogue
=>

```
{    // Using JS object as Dictionary
    [key: string]: value: <languageAbbrevation>-string
}
```

Available Languages Response [AH]

URL: <base-URL>/catalog/languages
() =>
Select * from AvailableLanguages
=>

```
{
    "languages": {
        "id": uint32, //not to be confused with code
        "coverage": float32, //percent, 0-1
        "commonName": string, //"Dansk" / "English" / "Svenska"
        "code": string, //"dk-DA", "nb-NO", "nn-NO"
        "icon": uint32, //Id of graphical asset in asset db
    }[]
}
```

Colony Info Response [AI]

URL: <base-URL>/player/<playerId>/colony/<colonyId>
(playerId, colonyId) =>
Select * Colony WHERE owner is <...> and id is <...> =>

```
{
    "id": uint32,
    "accLevel": uint32,
    "name": string,
    "latestVisit": string,
    "assets": [
        {
            "assetCollectionID": uint32,
            "transform": TransformDTO
        }
    ],
    "locations": [
        {
            "id": uint32,
            "level": uint32,
            "locationID": uint32,
            "transform": TransformDTO
        }
    ]
}
```

Colony Path Graph Resp. [AJ]

URL: <base-URL>/colony/<colonyId>/pathgraph
=> SELECT * FROM "ColonyLocationPath" WHERE colony = <colonyId> =>

```
{
    "paths": {
        "from": uint32, //Id of ColonyLocation
        "to": uint32 //Id of ColonyLocation
    }[]
}
```

Update Last Visit Resp. [AK]

(POST) URL: <base-URL>/colony/<colonyID>/update-last-visit
(colonyID, timestamp [body]) =>
UPDATE Colony (latestVisit) VALUES (<timestamp>) where id = <colonyID> <...> =>

```
{
    "timestamp": string
}
```

Upgrade Location Resp. [AL]

(POST) URL: <base-URL>/colony/<colonyID>/location/<colonyLocationID>/upgrade
(colonyID, colonyLocationID) =>
UPDATE ColonyLocation WHERE colony = <colonyID> && id = colonyLocationID SET "level" = level + 1;
=>

```
{
    "id": uint32, // ColonyLocationID
    "level": uint32,
}
```

Create Colony Resp. [AM]

URL (POST): <base-URL>/player/<playerId>/colony/create

```
{
    "name": string
} =>
{
    "id": uint32,
    "name": string,
    "accLevel": uint32,
    "latestVisit": string
}
```

Colony Overview Resp. [AN]

URL: <base-URL>/player/<playerId>/colonies
(playerId) =>
Select * Colony WHERE owner is <...> =>

```
{
    "colonies": ColonyInfoResponse[]
}
```

Open Colony Response [AO]

(POST) URL: <base-URL>/colony/<colonyId>/open
(playerId, colonyId, duration?) => //see Sequences -> Open Colony

```
{
    "code": uint32,
    "lobbyId": uint32,
    "multiplayerServerAddress": string, //"protocol://ip:port"
}
```

Join Colony Response [AP]

(POST) URL: <base-URL>/colony/join/<code>
(code) => //see Sequences -> Join Colony

```
{
    "owner": uint32,
    "lobbyId": uint32,
    "colonyID": uint32,
    "multiplayerServerAddress": string //"protocol://ip:port"
}
```

Close Colony Request [AQ]

(POST) URL: <base-URL>/colony/<colonyID>/close
(colonyID) => close Colony

Location Info Resp. [AR]

URL: <base-URL>/location/<locationID>
(locationID) =>
Select * Location WHERE id is <...> =>

```
{
    "id": uint32,
    "name": string,
    "description": string
    "appearances": {
        "level": uint32,
        "splashArt": uint32,
        "assetCollectionID": uint32
    }[],
    "minigameID": uint32
}
```

Location Info Full Resp. [AS]

URL: <base-URL>/location/<locationID>/full
(locationID) =>
//Very complicated query here combining data from Location, AssetCollection, CollectionEntry, GraphicalAsset and LOD. No image data yet tho. Just full overview =>

```
{
    "id": uint32,
    "name": string,
    "description": string
    "appearances": {
            "level": uint32,
            "splashArt": uint32,
            "assets": {
                "transform": TransformDTO,
                "asset": MinimizedAsset,
            }[]
        }[],
    "minigame": {
        "id": uint32
        "name": string,
        "description": string,
        "iconID": uint32,
        "difficulties": {
            "name": string,
            "description": string,
            "iconID": uint32
        }[]
    }
}
```

Asset Response [AT]

URL: <base-URL>/asset/<assetId>
(assetId) =>
Select * GraphicalAsset FULL JOIN (detailLevel, id) FROM LODs where graphicalAsset is <assetId> =>

```
{
    "id": uint32,
    "useCase": string
    "type": string
    "width": uint32, //res. x of LOD 0
    "height": uint32,  //res. y of LOD 0
    "alias": string,
    "LODs": [
        {id: uint32, detailLevel: uint32}
    ]?
}
```

Assets Response [AU]

URL: <base-URL>/assets?ids=x,y,z
(assetIds) =>
Select asset FROM GraphicalAsset where id is-any-of <ids> JOIN (detailLevel, id) FROM LODs where graphicalAsset is-any-of <assetIds> =>
AssetResponse[]

LOD Response [AV]

URL: <base-URL>/lod/<lodID>
(lodID) =>
Select * LOD where id is lodID => binary stream (byte[])

Headers:
"URSA-DETAIL-LEVEL": uint32
"URSA-ASSET-ID": uint32 //Id of graphicalAsset

LOD By Asset Resp. [AW]

URL: <base-URL>/asset/<assetID>/lod/<detailLevel>
(assetId, detailLevel) =>
Select * LOD where graphicalAsset is <assetID> AND detailLevel is <detailLevel> =>
LODResponse

Asset Collection Resp. [AX]

URL: <base-URL>/collection/<collectionId>
(collectionId) =>
Select * AssetCollection where id is <...> FULL JOIN (transform, graphicalAsset) FROM CollectionEntry where graphicalAsset is <assetId> =>

```
{
    "id": uint32,
    "name": string,
    "entries": [
        {
            "transform": TransformDTO,
            "asset": MinimizedAsset
        }
    ]
}
```

Minigame Info Resp. [AY]

URL: <base-URL>/minigame/<minigameId>
(minigameId) =>
Select * from MiniGame WHERE id is <minigameId>
FULL JOIN (name, icon, overwritingSettings) from
MiniGameDifficulty where minigame is <minigameId> =>

```
{
    "id": uint32,
    "name": string,
    "icon": uint32,
    "description": string,
    "settings": string, // JSON string
    "difficulties": [
        {
            "id": uint32,
            "name": string,
            "icon": uint32, //Graphical Asset ID
            "description": string,
            "overwritingSettings": any
        }[]

}
```

Minimized Minigame Info Resp. [AZ]

URL: <base-URL>/minigame/minimized?minigame=<minigameID>&difficulty=<difficultyID>
(minigameID, difficultyID) =>
Select (settings) from MiniGame WHERE id is <minigameID> JOIN
(overwritingSettings) from MiniGameDifficulty where id is <difficultyID> =>

```
{
    "settings": string, //JSON string
    "overwritingSettings: string, //JSON string
}
```

Proxy-like Api

Multiplayer Backend

Get Lobby State [BA]

(GET) URL: <base-URL>/proxy/v1/multiplayer/lobby/<id>
(id) => See Multiplayer Backend "Get Lobby State"

NB: Will respond with 502 (Bad Gateway) if the internal integration fails for any reason

Appendix Q - Multiplayer Backend API Specification

Connect

URL: <base-URL>/connect?clientID=<uint32>&lobbyID=<uint32>&IGN=<string>&colonyID=<uint32>&ownerID=<uint32>
(...args) =>

//WebSocket Protocol Upgrade

Create Lobby

(POST) URL: <base-URL>/create-lobby?ownerID=<uint32>&encoding=<base16|base64|binary>&colonyID=<colonyID> *//default: Binary*
(ownerID, colonyID, encoding) =>
{
    "id": uint32, //id of lobby
}

Get Lobby State

(GET) URL: <base-URL>/lobby/<lobbyID>
(lobbyID) =>
{
    "colonyID": uint32,
    "closing": bool,
    "phase": uint32,
    "encoding": string,
    "clients": {
        "id": uint32, //same as player id
        "IGN": string,
        "type": "owner | guest",
        "state": {
            // Colony Location id
            "lastKnownPosition": uint32,
            "msOfLastMessage": uint64 //since epoch
        }
    }[]
}

Health Check

URL: <base-URL>/health
() =>
{
    "status": boolean,
    "lobbyCount": uint32,
    "message": string
}

EVENT SPECIFICATION

The body of all messages send must start with 2 big endian uint32's: ClientID (id of user), EventID In that order.

Each message can either be send as raw binary (best if possible) else as base16 encoded string
(base64 is more efficient, but is pretty hard to read and reason about).
All serialization expects big endian.

Appendix R - Known DTOs

CollectionEntry [AA]

```
{
    "transform": Transform, // internal relative to collection origo
    "graphicalAssetId": uint32,
}
```

Transform [AB]

```
{
    "xOffset": float32,
    "yOffset": float32,
    "zIndex": uint32,
    "xScale": float32,
    "yScale": float32,  // duly note that the scale parameters take the
                           place of a width/height parameter if no such
                           exist
}
```

CLI shortHand: "xOff yOff zIndex, xScale yScale", example: transform="0f 0f 0, 0f 0f"

SessionInitiation [AC]

```
{
    "userIdentifier": string, //Hashed vitec id
    "currentSessionToken": string, //Vitec session token
    "IGN": string
}
```

DBDSN [AD]

```
{
    "host": string,
    "port": uint32,
    "dbName": string,
    "user": string,
    "password": string,
    "sslMode": string,
}
```

CLI input shorthand: "host port, user password, dbName, sslMode"

GraphicAsset [AE]

```
{
    "id": uint32,
    "useCase": "environment" | "icon" | "player",
    "type": string, //MIME type
    "width": uint32,
    "height": uint32,
    "alias": string,
}
```

MinimizedAsset [AF]

```
{
    "width": uint32,
    "height": uint32,
    "alias": string,
    "type": string,
    "LODs": [
        {
            "detailLevel": uint32,
            "id": uint32
        }
    ]
}
```

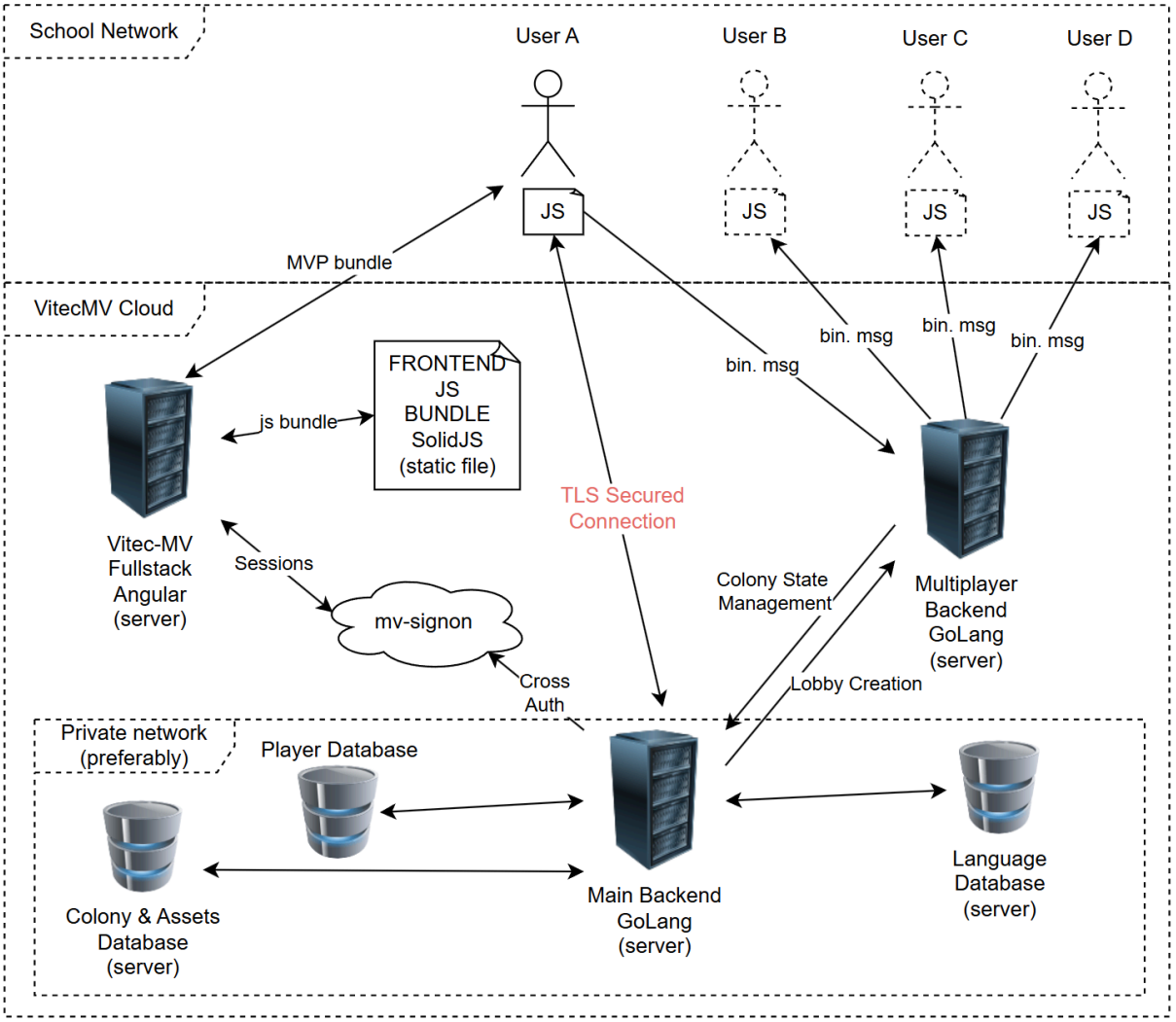
ColonyCreation [AG]

```
{
    "name": string
}
```

AsteroidMinigameSettings [AI]

```
{
    "minTimeTillImpactS": float32,
    "maxTimeTillImpactS": float32,
    "charCodeLength": uint32,
    "asteroidsPerSecondAtStart": float32,
    "asteroidsPersecondAt80Percent": float32,
    "colonyHealth": uint32,
    "asteroidMaxHealth": uint32,
    "stunDurationS": float32,
    "friendlyFirePenaltyS": float32,
    "friendlyFirePenaltyMultiplier": float32,
    "timeBetweenShotsS": float32,
    "survivalTimeS": float32,

    "spawnRateCoopModifier": float32
}
```

Appendix S - Simplified Service Overview



Appendix T - Initial Event Specifications

Colony Events

NB: These details have been superseeded by the contents in the bsc-multiplayer-backend repo /src/events.go

Multiplayer Management Events
Join Lobby, emitted by some guest
    utf 8 player IGN

Player Join Attempt, emitted by server
    utf 8 player IGN
    //Join process is halted until below is received

Player Join Accepted, emitted by colony owner
    Player Id
    //Guest connection attempt shall now succeed

Player Join Declined, emitted by colony owner
    Player Id
    //Guest connection attempt is declined

Player Leave, emitted by any player
    Player Id, IGN
    //Also causes "Player Aborting Minigame" Event if applicable

Multiplayer Session Closed, emitted by Multiplayer Backed
    //Fired if colony owner sends a "Player Leave" or the connection to the colony owner is dropped.

Colony Movement Events
Walk To Location, emitted by colony owner
    ID of location

Enter Location, emitted by colony owner
    ID of location

Minigame Initiation Events
(Shared between all minigames)

Difficulty Select for Minigame, emitted by colony owner
    ID of difficulty, name

Difficulty Confirmed for Minigame, emitted by colony owner
    ID of difficulty

Players Declare Intent for Minigame, emitted by server
    //Emitted if players should declare themselves ready immediately after difficulty select

Player Ready for Minigame, emitted by any player in lobby
    //Only valid after "On Difficulty Select"
    ID of player

Player Aborting Minigame, emitted by any player in lobby

Enter Minigame, emitted by server
    //Emitted when "Player Ready" has been recieved from all players in lobby
    ID of minigame, ID of difficulty

Appendix U - Project Description

Developing Educational Games for Teaching Touch Typing

Students
Gustav B. Wanscher
guwan21@student.sdu.dk
Exam nr: 4110804

Klaes D. Sørensen
klsoe20@student.sdu.dk
Exam nr: 182160602

SDU Supervisor
Henrik L. Bendixen, Soft. Arch. v. Novo Nordisk

Company Supervisor
Esben Gaarsmand, CTO VitecMV

Index

Problem

As computers have become the dominant writing tools, learning and understanding computer systems has become an intrinsic part of all education[1] [2]. This is why Vitec MV has issued this project; to provide an educational, interactive product that teaches Touch Typing for children between the ages of 8 and 14.

Context & Scope

This project will deliver a minimal viable product consisting of multiple services cooperating to provide the requested features and learning experience (see Appendix A for original project description, some requirements have also since been expressed during meetings with Vitec MV): The product will contain at least one multiplayer-capable minigame, persistent user progress and multilingual support.
Due to the involvement of children, the safe storage and transfer of their sensitive data present some unique constraints for the project. This, combined with the intent for multiplayer and multilingual support, will require extensive research into architecture and infrastructure.

Vitec MV has stated that they have an earlier bachelor project which has delved into further user-centric research on this very topic, which will become available to this project. Thus, this project will not delve further into that subject. They have also made clear that some existing services, a cloud environment, a platform, and other resources, will be available to this project, however, the extent of all of the above is unknown as of yet.

Finally, Vitec MV has made clear that this project should not involve mobile development.

[1] Pinet, Zielinski, Xavier & Longcamp 2022
[2] PISA 2022

Methods

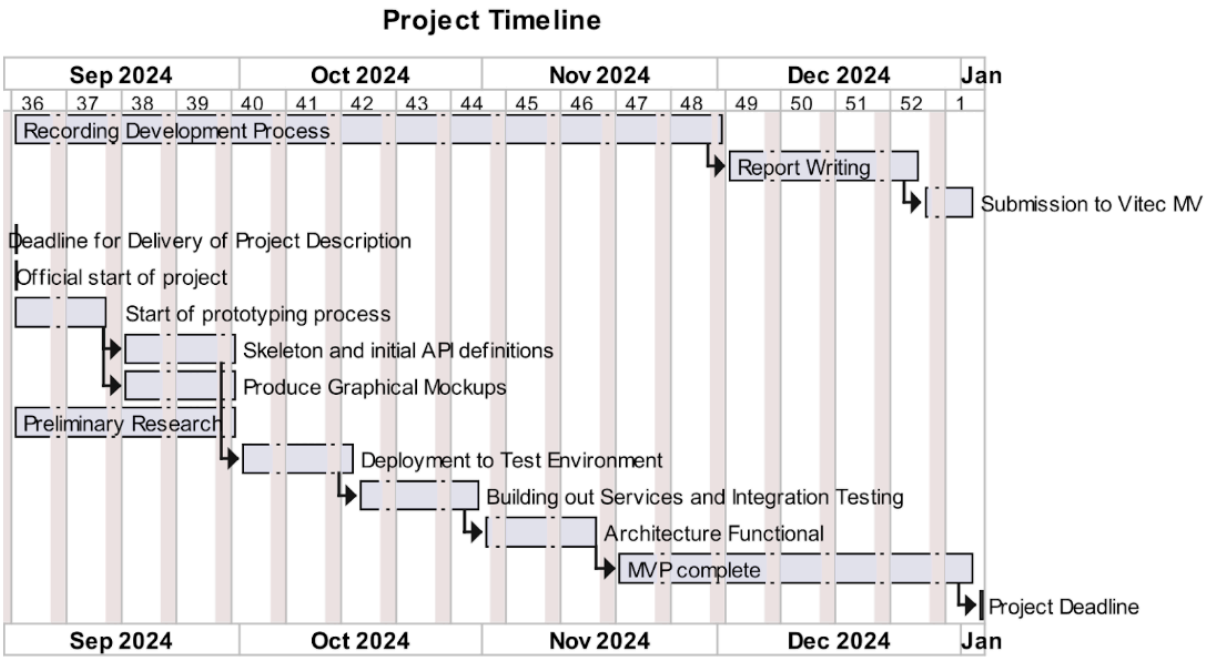
The project will be executed/implemented using an agile-traditional hybrid[3], leveraging a traditional approach for upfront general project management[4] research, designs, and prototyping, followed by a gradually more agile, incremental approach as the different cooperating services are integrated against each other and Vitec MV's existing infrastructure.
Project management tools and methods will be leveraged to manage project planning and execution. Namely tools for reaching common ground between the project team and the client (Vitec MV), as for the client's intent and desires.

With these constraints in mind, other engineering tools and methods will be used while designing the different services, to assure an unencumbered integration process and to reach project completion in time.
Various UML diagrams and modeling techniques will be used during the design process. For example, for integration between services, component and deployment diagrams will provide an integral view of system flow, functionality, and dependencies, indicative of implementation and structure. Additionally, abstract event sequences will prove useful for modeling the networking required for all client-on-client, or client-on-broker interactions necessary for all multiplayer-enabled minigames.

As for more implementation-specific methods & industry best practices: Design patterns[5], the S.O.L.I.D.-principles[6], & component-based software development through a microservice architecture[7], will be used to ensure a maintainable codebase and product.

To facilitate the storage of persistent user progress, dealing with potentially sensitive information, and managing system load; data management solutions must be analyzed, designed and established.

[3] Larson & Clifford 2021
[4] Larson & Clifford 2021
[5] Gamma, Helm, Jhonson & Vlissides 1994
[6] Martin 2014
[7] Lewis & Fowler 2014



Date	Activity / Milestone
31/08-2024	Deadline for delivery of project description
01/09-2024	Start of project
01/09-2024	Start of prototyping process
01/09-2024	Preliminary research
01/09-2024	Recording Development Process
16/09-2024	Skeleton and initial API definitions of all final deliverables
16/09-2024	Produce graphical mockups
01/10-2024	Deployment to Test Environment
15/10-2024	Building out services and Integration Testing
01/11-2024	Architecture Functional
15/11-2024	MVP complete
01/12-2024	Report writing
26/12-2024	Submission of report to VitecMV
02/01-2025	Project deadline

References

Neomí & Máximo 2014

- Neomí, Peña-Miguel, & Máximo, Sedano Hoyuelos 2014, paper: “Educational Games for Learning”, published by Horizon Research, DOI: “10.13189/ujer.2014.020305”

Larson & Clifford 2021

- Larson, Erik W., & Gray, Clifford F., 2021, book: “Project Management, The Managerial Process”, published by McGraw-Hill Education, eighth edition, ISBN: “978-1-260-23886-0”

Pinet, Zielinski, Xavier & Longcamp 2022

- Pinet, Svatlana, & Zielinski, Christelle, & Xavier, Alario F., & Longcamp, Marieke, academic study: “Typing Expertise in a Large Student Population”, published online the 5. of August on site: “National Library of Medicine” (PubMed Central), DOI: 10.1186/s41235-022-00424-3, PMCID: PMC9356123, PMID: 35930064

Poole & Preciado 2016

- Poole, Dawn M., & Preciado, Monique K., "Touch typing instruction: Elementary teachers’ beliefs and practices," published by: “Comput. Educ.”, vol. 102, pp. 1-14, Sep. 2016, doi: “10.1016/j.compedu.2016.06.008”, ISSN: “0360-1315”,

Dobson 2007

- A. Dobson, *Touch Typing in Ten Hours: Spend a Few Hours Now and Gain a Valuable Skill for Life*, 3rd ed. Oxford, United Kingdom: How To Books, 2007, ISBN: 978 1 84803 142 5.

IBM

- IBM, article: “What are Microservices?”, published by: IBM, date unknown, accessed the 26th of August 2024 at: “https://www.ibm.com/topics/microservices”

Gamma, Helm, Jhonson & Vlissides 1994

- Gamma, Erich, Helm, Richard, Johnson, Ralph, & Vlissides, John, book: “Design Patterns: Elements of Reusable Object-Oriented Software” published 31. October 1994 by: “Addison-Wesley Professional”, ISBN: “978-0201633610”

Lewis & Fowler 2014

- Lewis, James, & Fowler, Martin, article: “Microservices” published the 25th of March 2014 on own blog: “martinfowler.com”, accessed the 28th of August at: “martinfowler.com/articles/microservices.html”

Martin 2014

- Martin, Robert C., book from 2014, *”Agile Software Development, Principles, Patterns, and Practices”*, 1st ed. published by “Pearson Education Limited”, pp. 87-145. ISBN-10: “1-292-02594-8” ISBN-13: “ 978-1-292-02594-0”

PISA 2022

- PISA report from 2022 (Danish) (analysis of the habits of children in schools across the world). Section 5.6 of main report: “Brug af digitale apparater”. Accessed the 28th of August at: ”www.uvm.dk/internationalt-arbejde/internationale-undersoegelser/pisa/pisa-2022”

Appendix

A - Original Project Prompt

Transcribed from a handout.

Synopsis
The project involves adding more keyboard games to the 10-finger web application.

Goal
Learning typing is not that difficult, but mastering it requires repetition. Especially for children, it can be difficult to maintain the focus necessary to learn typing at a good level. Through games, their focus can be maintained for a longer period, and they thereby get more training and better typing skills.

The goal is to add at least 1 game that matches the room theme for 10-finger. There is an option to expand the project to support multiplayer gameplay.

Process
To make the process more manageable, we divide the project into the following 2 phases: start-up and development.

The start-up phase will focus on learning about 10-finger, programming languages and how the game should work. A graphic draft will also be made for the game.
In the development phase, the development itself will be started. The focus will be on “minimal viable product” and more features or games will be continuously built depending on project progress.

Study Relevance
In the project, you will encounter the programming languages: Angular, Typescript, HTML, CSS, & JavaScript. There will also be a need for planning, gamification, and process management.

The overall project is expected to have a scope of 3 months.