S5 REPORT

I. INTRODUCTION

Motivation / Purpose

Agile development and principles are incredibly important to software engineering. Agile development aids to "*ensure that development teams complete projects on time and within budget*". Moreover they "*reduce the risks associated with complex projects*" [1]. Kanban boards, a software development methodology commonly used by agile teams are highly beneficial. Reported benefits include "improved lead time to deliver software, improved quality of software, improved communication and coordination, increased consistency of delivery, and decreased customer reported defects" [2].

However, despite the importance of Agile principles, many first-year university students face challenges in grasping these concepts. A problem our team identified is that Agile development was not adequately taught during our first year of study. This problem aligns with recent research by Kropp et al.. They argue that "*graduates and undergraduates of computer science often lack the collaborative and communicative skills necessary for agile methods and, thus, are not yet well enough educated for agile development approaches. Therefore, new approaches or more adequate educational methods for teaching the necessary communication and collaboration skills need to be developed*" [3]. Kropp et al. further state that there needs to be an "approach on how to integrate these concepts into university courses, that focuses on active learning of agile collaboration"[3].

Our group resonated with this notion of 'active learning of agile collaboration'. Hence our team has developed a multiplayer, *Among Us* inspired game to teach first-year students how to use Kanban boards effectively in a software engineering context. The game incorporates simple tasks themed around common software engineering activities. Players work collaboratively in an ECS lab-themed map, and the Kanban board serves as the central tool for organising and managing tasks.

Project and Sustainability Goals

The main goal of this project is to improve first-year students' understanding of Agile principles, specifically Kanban boards, through an interactive and engaging multiplayer game. Our main objectives were to foster collaboration, increase understanding of agile and create an engaging experience for players.

Our project goals can be limited down to two factors; 1. create an online multiplayer game, 2. teach Agile through the use of the Kanban board.

In an even wider world context, our project also aligns with the sustainability goals outlined by the UN regarding People, Prosperity and Planet [4]. In particular our game aims to reiterate Goal 9 of the UN's sustainable development goals which is to "*Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation*". It also aims to emphasise goal 4, which is to "*Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all*". Ensuring that our project goals and sustainability goals are aligned will promote our games objective to provide valuable learning experiences to those wanting to further improve their Agile understanding.

II. RELATED WORK

Literature Review

We explored existing solutions to the problem identified. The intention was to take inspiration from the strengths of these existing solutions into our design while eliminating the weaknesses of existing solutions from our own design.

A major influence was the game *Among Us* [5]. It is known for its emphasis on communication, which aligns with our goal of fostering collaborative teamwork (a skill crucial to agile practices). A downside is that *Among Us* is very competitive and elimination-focused, this aspect we will be removing from our own design, especially as elimination and competition are values directly in contrast to the educational aim of our design.

The *Kanban Pizza* [6] game is designed to teach agile principles via an in-person, hands-on approach. It has a focus on the theory of constraints [7] and frequent reviews. The physical format and narrow target demographic of the *Kanban Pizza* game was considered a limitation. We aimed to improve upon this limitation in our own design, by shifting to a web-based environment in order to allow a much broader audience to access our game, aligning with one of our sustainability goals which is to 'ensure inclusive and equitable quality education'. Features in this existing solution such as reviewing and continuous development had to be abandoned in our product due to time constraints. However, they would have been ideal to have within our product.

The *Software Development Kanban Game* [8] demonstrates practical use of a Kanban board to simulate the software development lifecycle. The way it visualises task flow mirrors real world processes. The visualisation of the task flow is a key

component in our product. This ensures that our game mimics the real world/industry. The more our game mimics the real world/industry the more we address the key issue of "graduates and undergraduates not being well enough educated for agile development approaches" as discussed by Kropp et al [3]. The rigid structure of the *Software Development Kanban Game* is something we aim to eliminate from our design by making the Kanban task management more fluid, this aligns with the game being beginner-friendly.

The game *Retropoly* [9] has reflective intervals to assess performance. This mechanism was not adapted to our design due to time constraints but is a valuable feature that could be added in further progression. An issue with *Retropoly* was its focus on Scrum rather than Kanban. Our team found the gameplay of *Retropoly* quite complex. This led us to conclude that Kanban would be a better introductory agile principle to teach than Scrum. Hence this reiterated our choice to teach Kanban as opposed to a different framework.

The *getKanBan Board Game*[10] is a game that had a very lengthy gameplay, taking at least two and a half hours to play in its entirety. The game was good in terms of its team-focused nature. However, the length of it was a major downside. It made our team realise how essential to our design it was to have shorter tasks. Having shorter tasks means having a shorter game overall. Keeping the game fast-paced and engaging especially for students in a classroom/tutorial context is a key focus of our game. If players lose interest, the game fails its inherent purpose to educate.

In contrast to these games, there are tools like Trello [11] and Jira [12] which are widely used in task management. While they provide comprehensive Kanban features, the key limitation is that their complexity could overwhelm first-year students unfamiliar with agile principles. Hence this drove the need for our product to specifically be game-focused, as opposed to designing a tool. We did not want this limitation in our product. We ensured our product to be specifically gameplay and learning orientated, and not a design tool. Creating an easy-to-play game as opposed to a complex design tool also makes our design more equitable aligning with our sustainability goal.

Tools and Methodology

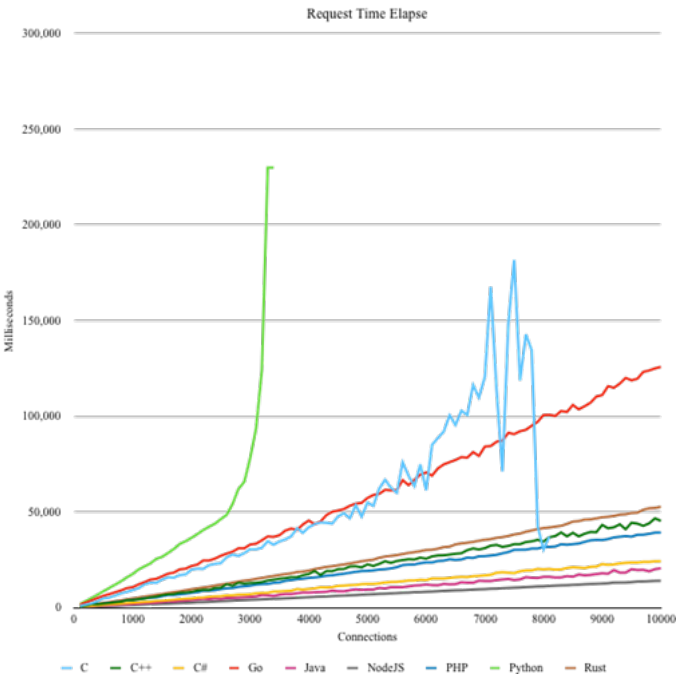
To implement our game we evaluated the options Unity, Godot, Unreal and JavaScript with Node.js and Phaser. Our focuses for evaluating these options were scalability, usability and familiarity.

In regards to *Unity* [13], it offered robust cross-platform support and a large asset store. However its WebGL builds had the potential to be slow and large, thus it was less suited for lightweight web games such as our design. *Godot* [14] was open source and lightweight, making it ideal for small web-based games, hence giving it an edge over Unity [15] in that regard. However, Godot lacked the community available from Unity or *Unreal*. Unreal was excellent for large-scale multiplayer environments and high-end graphics, however, its complexity and large build sizes would have been a hindrance to performance in web-based games.

These above options were hence considered but were discarded in favour of JavaScript with Phaser and Node.js. This was due to its easy integration with web browsers, real-time communication support using Socket.io and lightweight performance [16]. Given our constraints such as funds and time, this was the most suitable option for a fast-loading 2D game.

Unlike Unity or Unreal, JavaScript doesn't require conversion to WebGL as JavaScript is the native language of web browsers. This means that JavaScript code can run immediately in the browser environment without the need for additional frameworks or translation layers. This convenience was a key decision factor. Phaser can be easily integrated with Node.js and Socket.io for real-time multiplayer functionality.

When researching WebSocket Performance, our team found that NodeJS had the fastest connection time in comparison to all other languages and libraries evaluated (such as C, C++, C#, Go, Java, PHP, Python, and Rust). [17]. The figure below also from [17] illustrates this.



In regards to our planning and development process, the best way for our team to organise requirements was using the MoSCoW prioritisation framework. It '*provides a clear and easy-to-understand framework for prioritising project requirements'* [18]. We intended to use the priority levels to define the order in which we would implement functional and

non-functional components. The intention was all requirements specified as MUST HAVE should be completed before SHOULD HAVE. All SHOULD HAVE should be done before COULD HAVE.

III. DESIGN

Initial Design (CRC Cards)

At the midway stage of the project (approx. 16/08/2024) we had an initial design. We used CRC (Class-Responsibility-Collaborator) cards as *'They provide a clear and concise way to represent the design of a system, and facilitate communication and understanding about the design'* [19].

Class	Responsibility	Collaborator
server	- Handle Time - Store And Handles KanBan Board - Send Task Status To Clients - Handles Game Rooms - Handles Update Function	- KanBan Board
kanbanBoard	- Stores To Do Tasks - Stores Doing Tasks - Stores QA Tasks - Stores Done Tasks - Handle Changes of Task status	-
character	- Stores current Phaser Scene - Stores Asset Key - Stores Player Position - Stores Asset Frame - Handles is Moving - Stores and Handle changes of current direction	- Config - Direction
player <extends character>	- Handles player movement and idle animations	- Character asset keys
font-utils	- Handles loading font from file	
data-utils	- Handles getting the animation config from json file	
audio-utils	- Handles sound effects - Handles background music	
controls	- Handles player input (i.e. movement) - Handles locking player movement	- Direction
direction	- Handles player directions enum (i.e. UP, DOWN, etc.)	
direction	- Handles player directions enum (i.e. UP, DOWN, etc.)	
options	- Stores users settings	
asset-keys	- Acts as a getter for keys of loaded assets	
font-keys	- Acts as a getter for keys of loaded fonts	
scene-keys	- Acts as a getter for keys of loaded scenes	
world-scene.js	- Handles presenting the main game - Handles sending updates to the server - Handles player disconnecting from room - Handle task positioning on the map - Knows the current room of the client	- Kanban Board - Scene keys - Asset keys - Player - Controls - Directions - config
kanbanboard.js	- Handles request and receiving of the data from the server - Handles update of the KanBan board - Handles recursing current KanBan board status for client rooms	- Task handler
task-handler.js	- Generic class to handle tasks - Calls specific class that handles task - Hold current state of tasks	

When making our CRC cards it became evident that the two components that were of critical importance at this initial stage were the **server** and the **kanban board** (the first two cards show the Server class and kanbanBoard class). Our team found that these two components were the absolute bare minimum needed to make the game meet the project goals of having an online, multiplayer game (server) that teaches agile in some way (kanban board).

We had 8 **MUST HAVE functional requirements.** Having these 8 functional requirements covered was the priority.

Having a **Server** component was driven by the need to meet the requirements:

- The game must be designed to run on a local connection (localhost). Local network users can play the game.
- The game must be multiplayer (2- 6 players).

Having a **Kanban board** component was driven by the need to meet the requirements:

- The game must have a Kanban board that visually represents the workflow/tasks to be carried out.
- The Kanban board is shared between all members, and updated in real time.
- The board has four columns representing different workflow stages: TODO (tasks to start), ONGOING (tasks in progress), COMPLETED (finished tasks), and REVIEW (tasks to review).
- The kanban board must be accessible by each player.

Initial Design (Stage One Goals, what Requirements they are focused on)

We made an initial prototype design. This prototype design aimed to achieve these stage one goals:

Stage One Goals	
	1. Create game that can be seen by other users
	2. Join same room
	3. Sent object to server corresponding game room (GR)
	4. Server acts accordingly to received object
	5. Host multiple game room on localhost
	6. Send simultaneous inputs to multiple GR's
	7. Check that GR's act
	8. Check that GR's act accordingly without error

The Stage One Goals were derived from the MUST HAVE functional requirements related to the server component.

Evolution of the Design (constraints and design adjustments)

By the midway stage of the project (approx. 16/08/2024) we had successfully achieved the stage one goals. The server component design was quite thorough. The most critical next step was to focus on expanding/fleshing out the kanban board component that had been outlined in the CRC cards. The Kanban board was critical to achieving the majority of the rest

of the functional MUST HAVE requirements as well as essential to achieving the overall project goal "to improve first-year students' understanding of Agile principles, specifically Kanban boards, through an interactive and engaging multiplayer game".

At this stage, we had to focus on the kanban component. At the same time, we had to sacrifice certain theoretical components of our design. We learnt first-hand about the engineering principle of the theory of constraints. We needed to balance time, functionality and scope. Our initial idea for the game in our planning stage was too far out of scope. It did not seem possible that many of our COULD HAVE requirements could be achieved. We were limited in time with only a few weeks to go. We realised we had to focus on the key functionalities outlined in the requirements that would get the game to achieve the project goal. We had to downscale/condense our design components into what has been outlined below due to these constraints.

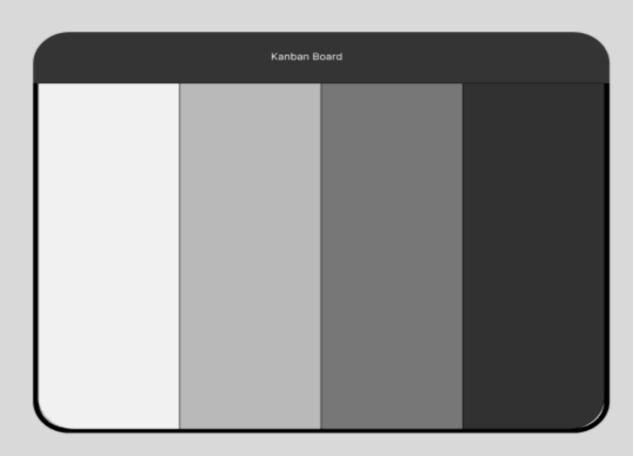
Final Design (Requirements it is driven by)

Server

The server is built to handle communications using WebSocket protocols. This allows for a real-time exchange of data between clients and the server. This architecture allows multiple players to connect simultaneously without a reduction in performance. Each game room is designed as a self-contained unit which players can join. The server handles the creation and management of these rooms. The server is designed to ensure that any changes made by one player are reflected almost instantaneously to all other players in the same room. These choices were driven by the need to address the **MUST HAVE functional requirement** *'The game must be multiplayer (2-6 players)'*, the **SHOULD HAVE functional requirement** *'The server should handle multiple game rooms efficiently'* as well as partially address the **MUST HAVE functional requirement** *'The kanban board is shared between all members, updating in real time'.* The game runs on a local network, allowing users on the same local network to play together. This was done to align with the **MUST HAVE functional requirement** that *'the game must be designed to run on a local connection (localhost)'.*

Kanban Board

The User Interface (UI) incorporates a kanban board. A player would be in the game (so they would see the lab world scene), they can then press the spacebar which makes a kanban board popup on top of the world scene. The kanban board will showcase various tasks, as well as columns indicating different states in the task flow. The user can interact with the board, dragging and dropping the various task buttons between the states (represented as columns). The ability to view and interact with the kanban board directly addresses the **MUST HAVE functional requirements** *'The game must have a kanban board that visually represents the workflow/tasks to be carried out'* and *'the board has four columns representing different workflow stages: TODO (tasks to start), ONGOING (tasks in progress), COMPLETED (finished tasks), and REVIEW (tasks to review)* as well as the **MUST HAVE non-functional requirements** *'The game must teach the user about agile software engineering, specifically kanban boards.'*



Game Map

The game map is designed to represent the ECS labs, which is where the player navigates and interacts with various tasks. The map contains walls and tables that restrict player movement, creating an environment with obstacles and defined spaces. Tasks are laid out at specific points on the map, and when a player reaches one of these locations, they can engage with and complete the corresponding task. These design choices were driven by the **COULD HAVE functional requirement** 'The game could have a proximity detection system for identifying tasks', the **MUST HAVE functional requirement** 'The game displays a map' as well as the **SHOULD HAVE functional requirement** 'Tasks are positioned at points on the map'.

Tasks

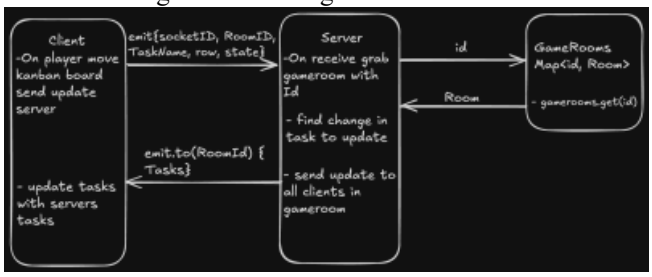
The various tasks in the game follow a consistent design pattern to provide a uniform experience for players. Each task presents a contextual image related to the task's theme, which primarily serves an aesthetic purpose. The core gameplay mechanics are simple: the player presses a button to gradually fill a progress bar located at the bottom of the screen. Once the progress bar is filled, the task is considered complete. Due to the constraints mentioned above, the original idea for the task component of the game had to be scaled down. Initially, the idea was that each task would be similar in the regard that it is a pop-up. However, the actual content of the task would be distinctly different. In reality, we had to downscale so that each task followed a generic format, with slight variations (this is accurate at the time of submitting this report, we may improve upon this).

IV. IMPLEMENTATION

Server

The game server was implemented using JavaScript, Node.js (version 22.8.0), and the Phaser 3 framework. Node.js was chosen for its compatibility with Phaser [16] and its event-driven, non-blocking architecture, which efficiently manages real-time multiplayer connections. The server runs concurrently with the client, using the 'concurrently' package in the 'dev' script to streamline the process. The main server is run off one file (server.js). It uses Express which builds the HTML. Express [20] is defined as an app. To be able to send resources (such as HTML, JavaScript, and PNG files) to a user's web browser cross-origin resource sharing is used. If we didn't use cross-origin resource sharing (CORS), the browser would consider downloading data from an unknown source as unsafe [21]. The server is created using the line in code http.createServer(app). The server has listeners and handlers to listen for a given port, on connection a function handles the connection.

Below is a diagram illustrating server interactions:

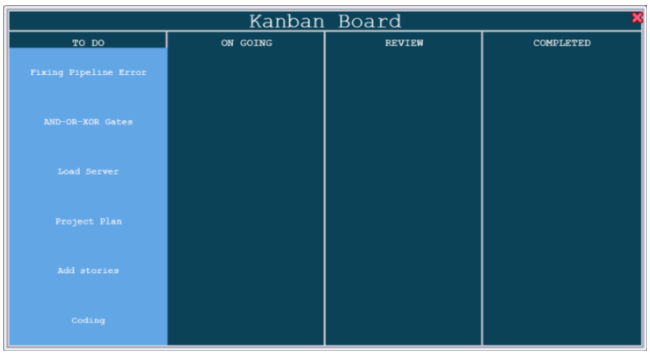


Kanban Board

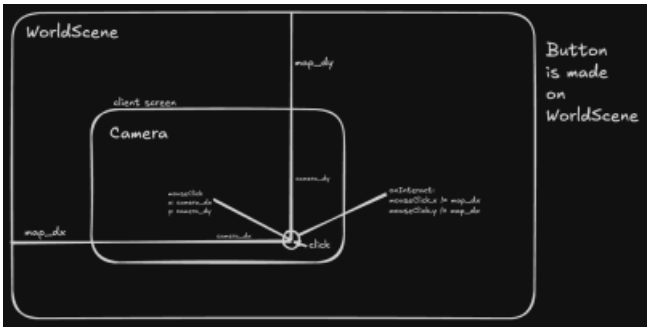
The crucial implementation of the UI was the Kanban board. The Kanban board was implemented by extending a WorldContainer, a class in Phaser that provides the ability to create pop-ups within the game. Extending WorldContainer allowed us to integrate the Kanban board smoothly with the game's overall design, rather than creating disjointed or random elements. The KanbanBoard class (which extends the WorldContainer) initialises empty task boxes arranged in a grid, allowing users to interact with draggable task buttons. These buttons can be moved across the board, with their positions updated dynamically based on available empty boxes.

Problems with implementing the Kanban board that were overcome

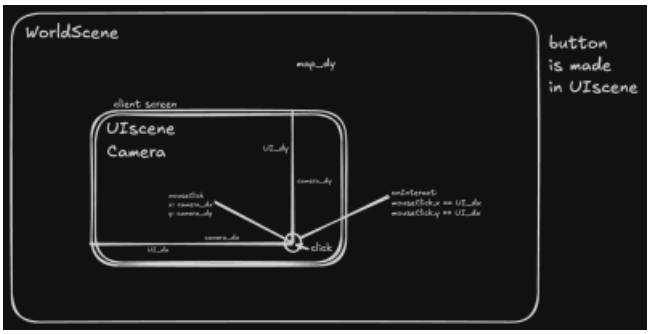
Initially, there was a problem where the draggable task buttons, implemented using Phaser, could not move as expected due to an offset in the 'camera' that tracks the player. This offset caused the button positions to appear misaligned with the visual grid. The solution involved accounting for the camera's position when calculating the button's coordinates, ensuring the buttons correctly moved to the player's view (as illustrated in the problem and solution diagrams).



Problem:



Solution:



Another issue arose when a task was moved from one column (e.g., "Review") to an invalid position (outside the board). The task should have "snapped back" (been returned) to the column it was in just previous to the invalid move (e.g., "Review"), but instead, it always reverted to the starting column ("To Do") on other players' screens, while it appeared correctly on the screen of the player who moved it. This implementation problem was resolved by having the server continuously observe changes to the Kanban board. With this synchronisation mechanism in place, updates to the board are consistently reflected across all players, ensuring tasks stay in the correct column for everyone.

Solving these issues regarding the implementation of the Kanban board was crucial. If the Kanban board was not draggable/interactable (due to the camera offset issue) or not synced across all players, the whole concept of agile and specifically kanban board (which our overarching project goal is reliant on) was not being presented effectively.

Game Map

For the visual elements of the map and characters, sprite packs were obtained from itch.io in PNG format. To manage and design the map, we used Tiled, a tile-based map editor. Tiled allowed us to generate the map using sprite images without the need to manually code the layering mechanics (e.g., ensuring that characters appear in front of walls or objects). Using Tiled, we were able to produce a CSV file that mapped out the game's grid layout, where each tile on the map is represented by a unique number. This CSV file was particularly useful for two key aspects of the game. Firstly Collision Detection: By associating specific numbers in the CSV file with obstacles like walls and tables, we were able to implement collision detection easily. Secondly Task Proximity Detection: The CSV file also made it easier to implement the proximity detection system for tasks. When the player's position matches the location of a task (as defined in the CSV file), the game recognizes this, enabling the player to complete the task.

Tasks

The task system was implemented using a common structure, with slight variations in the background imagery to match the task context. The tasks follow a basic framework where they extend a TaskSkeleton. Each task displays a background image loaded from sprite assets. The player interacts with a button, and as the player interacts with the button a progressValue iterates upwards. Once the progress reaches a certain value, the task boolean isComplete switches to true. A "TASK COMPLETED" button appears in the centre of the screen, and the task can then be closed, returning the player to the main game map.

V. Evaluation

Requirement alignment

We categorized our functional requirements in the order we should ideally implement. As mentioned in related work, we utilised the MoSCoW prioritisation framework. This method *'helps project teams and stakeholders determine what is essential for the project's success and what can be deferred or excluded'* [22]. Seeing how well we were able to align with the various level requirements is one way to evaluate our project's success. Below is a breakdown of the progress made in each relevant category:

MUST HAVE: We identified 8 essential functional requirements that were critical for the project's success. All 8 of these requirements (100%) have been successfully achieved.

SHOULD HAVE: In this category, we outlined 10 important functional requirements that would greatly enhance the project. Out of these, 6 have been completed, representing 60% of the SHOULD HAVE requirements.

COULD HAVE: 8 additional functional requirements were considered desirable but not essential. Of these, 2 have been fully implemented, accounting for 25% of the COULD HAVE requirements.

The 8 **MUST HAVE** functional requirements were:

- The game must be designed to run on a local connection (localhost). Local network users can play the game.
- The game must be multiplayer (2- 6 players)
- The game must have a kanban board that visually represents the workflow/tasks to be carried out.
- The Kanban board is shared between all members, updating in real time.
- The board has four columns representing different workflow stages: TODO (tasks to start), ONGOING (tasks in progress), COMPLETED (finished tasks), and REVIEW (tasks to review).
- Players must be able to navigate the game with movement keys.
- The kanban board must be accessible by each player.
- The game displays a map.

These requirements are all linked to the key components (Server, Kanban Board, Map and Tasks) addressed in the Design and Implementation section. Achieving this led us to have a web-based, multiplayer game that has an aspect of an agile methodology. So based on this metric the bare minimum, 'high level' project specification has been met.

MoSCoW prioritisation aids in identifying the Minimum Viable Product. This is *'the set of features that deliver the most value with the least effort. This helps in delivering a usable product'*[22]. From the analysis above we were able to deliver at least the Minimum Viable Product.

In our current implementation, we have implemented some COULD HAVE functional requirements (20%) before the completion of all SHOULD HAVE functional requirements (only 60% of the SHOULD HAVE functional requirements are complete). This contradicted our 'plan' of using MoSCow to identify the project implementation lifecycle. This has led to some functional components not being addressed that are critical to the game's scope.

User testing

With a more 'holistic' evaluation of the implementation, we conducted user testing and feedback following ethical guidelines. We are trying to evaluate certain holistic questions regarding our product such as; trying to understand how our implementation could be improved, is the current UI intuitive, did the user enjoy their experience, what they disliked about it, evaluation of the kanban board itself (which is directly involved

with addressing the problem statement) and feedback on what can be improved.

We were granted ethics approval to carry out user testing for the project. This meant we could collect user data following guidelines and regulations provided by the university. Based on the ethics approval notice that was provided to us [23], we structured a script which highlighted various components of peer testing regarding our game. This script highlighted; what our game was about, what the purpose of the game was, what user participation involved, and what gameplay consisted of. The script also contained the necessary required information for ethical participation. This included; acknowledgement that participation was voluntary, mention of confidentiality and privacy laws 'for the information the user provided', and reiterated that the information was owned by the user and would be safely stored. This ethics script would be signed/approved by the user so they could participate in the peer test. Below is a summary of key findings from the user feedback:

1. Aesthetics

The game was praised for its visual appeal, indicating success in design.

2. Usability

Players found task interactions, especially with the Kanban board, unintuitive, leading to confusion and reducing playability.

3. Overall Experience

On average participants rated their experience, 5 out of 10, reflecting a need for significant improvements in gameplay.

4. Gameplay Difficulty

Feedback was split. Some found the game too easy while others found it too difficult. No one rated the difficulty as "Just Right". This may reflect the fact that our actual game is very simplistic, regarding task completion (just pressing a button) hence making it considered easy. Whereas understanding how to interact with different elements such as the kanban board was not intuitive making the concept of the game difficult to understand.

5. User Interface & Controls

The controls, particularly for movement and task interaction, were rated poorly, with an average score of 4.66, signalling a need for more intuitive design.

6. Conclusion

While visually appealing, the game lacks balance in difficulty and intuitive interactions, limiting its effectiveness as an educational tool. Further refinement is needed.

User Satisfaction

To evaluate user satisfaction we established these key takeaways from users written feedback (this feedback was obtained part way through implementation):

- Confusion around the overall point/objective of the game as well as how to play it effectively.
- UI was unintuitive as UI components at that stage in the codebase were key-related.
- Kanban board seemed unnecessary to the gameplay, because the tasks and the Kanban board were not heavily related in carrying out one or the other.
- Users found it difficult to identify/find tasks where there was not a clear identifier for them.
- Comments on the map were that the map was incredibly large and hard to identify which area they were in.
- Tasks didn’t make sense, stating the images seemed interactable when they weren't.

Based on feedback and criticism gathered from user testing, we modified functional areas of our game for a better user experience.

- The instructions to be provided to the player was added. This consisted of the menu-screen when the game was first loaded as a how-to-play button, as well as a question mark button in the game room. This provides first time users with an understanding of how to carry out the game.
- Implementation of buttons to interact with the kanban board and various tasks around the map were added. This provided clear visual indicators that were intuitive to players, as well as those who prefer mouse based gameplay.
- Limited the size of the map, to make only the top area playable from our prototype, which helped provide more focus in creating a more fun and interactive environment in a limited area.
- Provided more interactive tasks (plan sprint task and plan project task) which required the player to interact with in task elements.

Testing

Regarding evaluating the project by the metric of code testing, our team has performed poorly in this regard. We do not have tests to check the performance of our code. In an ideal project the intention is that, every requirement should have a test associated with it, but our team struggled with this for a variety of reasons:

- The nature of the game being multiplayer and run on a server made it difficult to test. Furthermore, we struggled to find a built-in testing framework for Phaser that we understood, within the time constraints. Our team has familiarity with certain testing frameworks in other languages such as Junit but this does not relate to the way our code was implemented.
- Based on these findings regarding game testing [24] our team also felt that with the nature of a multiplayer

game, the priority in regards to testing should be user testing (which we did do) as opposed to automated testing.

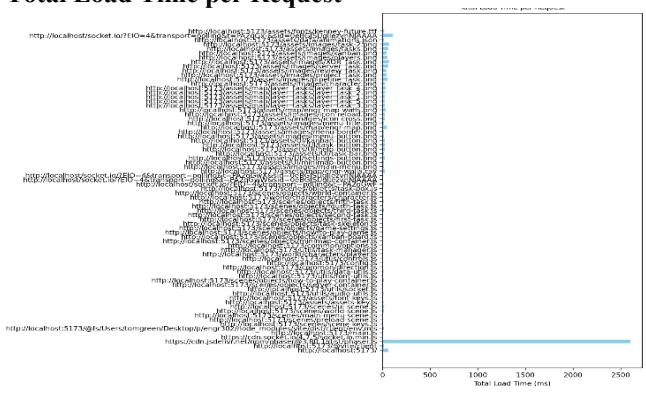
Thus, automated testing was not a priority for us, in favour of user testing. Furthermore, we were constrained by time, functionality and lack of understanding regarding the test framework. We were focusing on having a functioning game by the deadline as opposed to testing. The general opinion of the team was that our project goals were more focused on user experience and education as opposed to having the most quality-assured product. In hindsight what we could have done was explicitly assign a member of the team to handle testing. They could have become familiar with testing frameworks in Phaser. For example Mocha [25] or JEST [26].

Measuring Performance

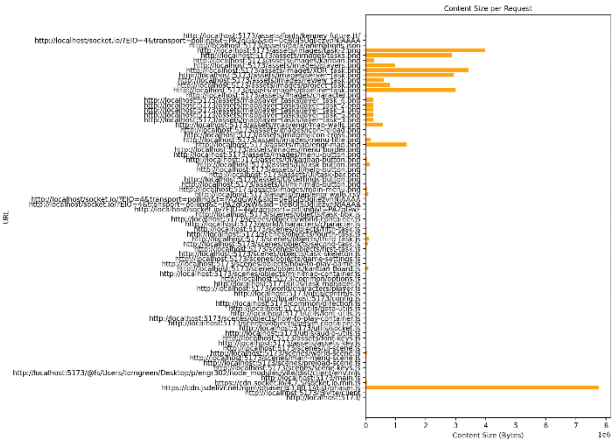
Being critical regarding our requirements, we did not explicitly outline how we were going to measure various components of performance. This has led to two issues. The first was that when implementing we did not explicitly think about how to garner an optimal performance, rather we just wanted something that functioned in some regard. Secondly, when trying to evaluate our product it is difficult as we do not have specific requirements laid out to test against.

However we did still try to evaluate the performance of our product, this evaluation came about as we were developing. One thing we observed once we had an implementation was how long it took to perform a GET request on the various necessary files and scripts needed to load the game.

Total Load Time per Request



Content Size per Request



Analysis of graphs

Our implementation involved preloading all assets into the cache. This meant that during gameplay, there are no GET or POST requests to acquire assets (having these requests in gameplay would result in the overall performance and frame-rate of the game worsening). The graphs provided represent the time and size of the assets during the pre-loading stage of the game startup. We can notice from the ‘Content Size per Request’ graph, that some of the images used in the game are very large and take an extensive amount of time to load. Based on the way that Phaser works with/handled assets and images, this would hinder the performance. Using this set-up process of loading all images beforehand, achieves the non-functional COULD HAVE requirements regarding game performance and a smooth, maintained frame rate.

Stakeholder Satisfaction

When evaluating how well our product satisfies the needs of our stakeholder, we feel that we have performed highly in this regard. We consistently met with the stakeholder throughout the project lifecycle. We were in a constant feedback loop that shaped our requirements and design. The stakeholder was not satisfied with our initial requirements, reasons being that there was confusion between functional and non-functional requirements, too large a scope and confusing ‘qualities’ with ‘requirements’.

Following on from this experience, we not only refined the requirements but also made sure to consistently update the stakeholder more in-depth about the status of our project at the current stage. This was to ensure that he was satisfied and make sure that the scope of our game had not changed.

We also took on board feedback from the stakeholder to improve our design choices. For example, on 04/10/2024 we demonstrated the game (as existed at that time) to our stakeholder. General feedback on the game was good. He deemed our result a prototype and clarified that we had met the core requirements of the brief. The game was not deemed the

most ‘exciting’. A point of confusion was the task popup, a lot was going on visually, so it was unclear what part of the popup was the actual task. The stakeholder suggested adding a transparent overlay to our task popup to make the mini-game tasks more visually intuitive for users. We took this feedback into account.

Following Engineering Processes

Our team's initial plan was to follow an iterative development process. This is where we would cycle between the steps of planning, design, implementation, testing and evaluation.

Our team struggled to follow this cyclic process. We spent the majority of the project timeline on planning, then switched our focus towards design and implementation in the last few weeks. In regards to testing, we have done user testing, but besides this, our team did not spend enough time prioritising testing. We were also lacking in a proper evaluation. Iterative development intends to cycle through this process multiple times during a project lifecycle, but we did not do this due to time constraints.

Limitations

In terms of evaluating aspects that are missing from the product we produced.

Firstly as discussed in the Design and Implementation sections the tasks that were designed and implemented were much more scaled-down than what was originally envisioned for the game. Furthermore, there is a lack of synchronisation between the kanban board and the tasks. This means that when a task is completed by a player there is no prompting from the game itself to reflect those changes in the kanban board. The user has full control over modifying the Kanban board. They could move tasks that have not been completed into the COMPLETED section of the Kanban board if they wished. So essentially there is a lack of communication between these two features. Had this been more synchronised the product would have better achieved the intention of teaching agile.

Also as mentioned in our Literature Review, there were components from existing solutions that we were intending to add into our product. For example, features such as Reviewing and Continuous Improvement similar to what exists in the Kanban Pizza game were never added to our game due to lack of time, despite them being considered advantageous to our project goal of educating.

VI. Conclusions and Future Work

To conclude, our game does somewhat achieve the main goal established in the introduction to ‘*improve first-year students' understanding of Agile principles, specifically Kanban boards, through an interactive and engaging multiplayer game*’. However, our team does acknowledge that due to the nature of the project (the limited timeframe, initial uncertainty about the objectives, balancing other university courses, etc.) the game is much more scaled down than what we initially envisioned. We have achieved our MUST HAVE requirements. However some of the SHOULD HAVE and COULD HAVE requirements are lacking. Our finished result can be considered more of a ‘prototype’ than a final product. When showing the result to the stakeholder in Week 11, he referred to it himself as a prototype but was satisfied that the product brief had been met.

In the context of what could a ENGR489 student do with our result to expand upon it, they can improve upon the fact that our game is currently operating on a local host. A project that could be started from this point is, figuring out how best to deploy this game to make it more accessible to a broader target audience (e.g. those outside Victoria University of Wellington). So a project could be integrating the game with a web server infrastructure, to increase scalability. This would further help achieve our sustainability goal of making this educational product more equitable and accessible.

Some of the limitations discussed in the Evaluation section serve as good points to upgrade this project. To improve upon the product a straightforward idea is simply implementing what we intended to but couldn’t because of constraints. Further project ideas include making a game that has a greater synchronisation between the kanban board and the tasks, or a game that has reviewing and continuous improvement features. Even just taking the product we have now but expanding upon the design and implementation of tasks so that tasks are of the wider scope and complexity we initially envisioned.

In a general regard, another project could be using the literature review as a starting point and doing a more in-depth investigation into existing solutions. ENGR489 students can spend more time on this and incorporate more valuable features of the existing solutions into a new and improved project.

Furthermore, our finished result lacks the security features deemed appropriate for a game. It currently uses tracking information to make the game function. An ENGR489 project could be to develop upon this game to make it still function, whilst adhering to security principles. The use of the Phaser framework in JavaScript could potentially be extended to use more advanced security features. If this game were to be deployed these security principles must be met.

References

[1] Opentext, “What is Agile Development and why is it important?,” OpenText. https://www.opentext.com/what-is/agile-development

[2] Ahmad, M. O., Markkula, J., and Oivo, M., "Kanban in software development: a systematic literature review," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Sep. 2013, doi: 10.1109/SEAA.2013.28.

[3] Kropp, M., Meier, A., Mateescu, M., and Zahn, C., "Teaching and learning agile collaboration," *IEEE Xplore*, Apr. 1, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6816791. Accessed: Mar. 29, 2022.

[4] United Nations, "Transforming Our World: the 2030 Agenda for Sustainable Development," United Nations, 2015. https://sdgs.un.org/2030agenda

[5] Mohnish, "Among Us — Skills That are Used in the Real World - Mohnish - Medium," Medium, Apr. 18, 2024. https://medium.com/@mohnishde/among-us-skills-for-the-real-world-0d617b3e70c1 (accessed Oct. 13, 2024).

[6] E. El-Badry, "A few years ago I was introduced to the squarely-named 'Kanban Pizza' game. It has 'kanban' and 'pizza' in the name - so I was immediately intrigued," *LinkedIn*, 2020. [Online]. Available: https://www.linkedin.com/pulse/kanban-pizza-game-keeps-giving-ehab-el-badry/. Accessed: Oct. 13, 2024.

[7] Lean Production, "Theory of Constraints (TOC) | Lean Production," leanproduction, 2021. https://www.leanproduction.com/theory-of-constraints/

[8] "Kanban Simulator - Software development," www.kanbanboardgame.com. http://www.kanbanboardgame.com/

[9] "Retropoly What is it?" Accessed: Oct. 13, 2024. [Online]. Available: https://www.agilepractice.ro/wp-content/uploads/2017/04/Retropoly-game-instructions.pdf

[10] "getKanban Board Game," getKanban, 2015. https://getkanban.com/

[11]"Trello," trello.com. https://trello.com/templates/engineering/kanban-template-LGHXvZNL

[12] Atlassian, "Jira kanban boards | Atlassian," Atlassian. https://www.atlassian.com/software/jira/features/kanban-boards

[13] "(PDF) Unity Game Development Engine: A Technical Survey," *ResearchGate*. https://www.researchgate.net/publication/348917348_Unity_Game_Development_Engine_A_Technical_Survey (accessed Feb. 03, 2022).

[14] Fransson, E., and Hermansson, J., "Performance comparison of WebGPU and WebGL in the Godot game engine," *DIVA*, 2023. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1762429&dswid=-9859. Accessed: Oct. 13, 2024.

[15] Foxman, M., "United we stand: Platforms, tools and innovation with the Unity game engine," *Social Media + Society*, vol. 5, no. 4, pp. 1–10, Oct. 2019, doi: 10.1177/2056305119880177.

[16] Schratz, C., "Getting started with Phaser 3 and Node.js," *DEV Community*, Aug. 31, 2020. [Online]. Available: https://dev.to/cschratz/getting-started-with-phaser-3-and-node-js-4mbp. Accessed: Oct. 13, 2024.

[17] Tomasetti, M., "Websocket performance comparison," *Medium*, Feb. 8, 2021. [Online]. Available: https://matttomasetti.medium.com/websocket-performance-comparison-10dc89367055.

[18] Lucas_DevSamurai_, "Understanding the MoSCoW prioritization | How to implement it into your project," *Atlassian Community*, Aug. 30, 2023. https://community.atlassian.com/t5/App-Central-articles/Understanding-the-MoSCoW-prioritization-How-to-implement-it-into/ba-p/2463999

[19] "CRC Cards: Product Management & Operations Explained," www.launchnotes.com. https://www.launchnotes.com/glossary/crc-cards-in-product-management-and-operations

[20] Mozilla, "Express/Node introduction," *MDN Web Docs*. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

[21] "What is CORS? - Cross-Origin Resource Sharing Explained - AWS," *Amazon Web Services, Inc.* https://aws.amazon.com/what-is/cross-origin-resource-sharing/

[22] "Moscow Prioritization: Definition, Examples, and Benefits," Roadmunk. https://roadmunk.com/glossary/moscow-prioritization/

[23] Victoria University of Wellington, "Human Ethics Application", 2024. [Online]. Available: [URL https://nuku.wgtn.ac.nz/courses/18231/files/3256924?module_item_id=657599]

[24] R. Ramadan and B. Hendradjaya, "Development of game testing method for measuring game quality," *IEEE Xplore*,

Nov. 1, 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7062694. Accessed: Nov. 26, 2022.

[25] Mocha, “Mocha - the fun, simple, flexible JavaScript test framework,” *Mochajs.org*, Oct. 18, 2019. https://mochajs.org/

[26] Jest, “Jest Delightful JavaScript Testing,” *Jestjs.io*, 2017. https://jestjs.io/