Change Report

Cohort 2 Team 5

Harry Beaumont-Smith
Tom Nolan
Will Punt
Ruth Russell
Mimi Shorthouse
Lottie Silverton
Stanley Thompson

Introduction

When initially switching projects we started by evaluating the documents the previous team had in place and discussed what needed to be immediately changed. We then divided the documents between our group, sticking to our roles from the previous half of the project. At first we edited the parts of the documents that we thought could be improved upon, trying to make them similar to, or add parts to the documents to make them more similar to our previous project. This would help us to understand and improve the new project, in a way that was more similar to our previous project.
This time, when updating the documents, in order to abide by conventions of the documents we used textbooks and websites. This helped ensure we made documents that conformed to the general standards of game/project creations.

When updating this document we made sure that if anything from the initial documents/project was changed, that you immediately update it within this report. This enabled us to not miss anything and keep accurate and up to date information on what we had improved/implemented with this new game.
To keep track and review we used our planning methods to verify if and when changes had been made in order to keep this document updated.

User Requirements:

Old URL: https://thomas-nolan.github.io/team5projectsite/docs/Req1.pdf
New URL: https://thomas-nolan.github.io/team5projectsite/docs/Req2.pdf

When modifying the requirements, we thought that the previous requirements already set up were suitable for the project, so they didn't need to be changed or altered too drastically. We mainly added and edited new requirements based on the new user functionalities and requirements added to the new, updated brief. To make sure our requirements accommodate these changes we edited preexisting requirements and added new ones for the new functionalities needed for the game.
We also added some requirements for parts of the original brief that we felt didn't have a specific enough requirement already, that needed a requirement specifically for them. As well as this we renamed some requirements to make more sense with our updated architecture.

New User Requirements

UR_Leaderboards:
Description: The game should have a leaderboard that contains high scores and the names of players who achieved them.
Priority: Shall
Original Req1.pdf didn't require leaderboards / saving, Assessment 2 does.
Simple file based implementation
Implemented in FileManager class with file based persistence
Added FR_Data_Storage: Leaderboard data (names and scores) stored in csv files.

UR_Achievements:
Description: The game should have an achievement system that acknowledges and rewards milestones achieved by the player.
Priority: Shall
Implemented in the AchievementManager class (7 achievements total)
Added achievement tracking logic
Modified FR_Data_Storage to include achievement persistence
Added UI elements to display achievements

UR_Invisible_Events:
renamed from UR_Hidden
Original Req1.pdf the game should have three hidden events.
Updated to the game should contain at least 3 events that are invisible to all players until they are triggered.
Priority: Shall
Implemented as BookshelfEvent, EventLongboi, TeleportEvent

Requirements Modified

UR_Negative:
Original Req1.pdf had 3 negative events; the full game should have five negative events that hinder the player from progressing.

UR_Lose_Time to UR_Lose_Condition:
Original Req1.pdf the player should lose if they haven't escaped after 5 minutes. Updated to the player should lose the game if they haven't escaped after 5 minutes or have been caught by the dean.
ID changed from UR_Lose_Time to UR_Lose_Condition and added second lose condition, caught by Dean.
Implemented in GameScreen collision detection: Dean AI must check collision and trigger game over.

UR_Dean renamed to UR_Dean_Object:
Original Req1.pdf the game should involve a Dean, who is chasing the player which was abstract
Priority: Shall
The original had two Dean requirements: UR_Dean (Shall). Dean could be abstract so players might not interact with Dean.
UR_Dean_Object (May): Dean could be visible.
Decided to implement visible Dean (UR_Dean_Object) and removed redundant to simplify requirements.

FR_Player_Displayed:
Original Req1.pdf had two separate requirements FR_Player_Displayed and FR_Player_Movement.
Updated to single requirement: FR_Player_Displayed to show player character displayed on the screen and the user to move the character in one requirement.

FR_Fast_Score renamed to FR_Fast.

New Functional Requirements

FR_Data_Storage:
Description: The high scores and achievements of players are saved and displayed in the game.
Implemented in FileManager class with file I/O.

New Non Functional Requirements

NFR_Compatibility:
Description: The system must be compatible with the user's computer and function on all computers.
Tested on multiple platforms (Windows, macOS, Linux).

Unchanged Requirements

Gameplay Requirements:
UR_Player, UR_Time_Tracker, UR_Pause, UR_Map_Limits, UR_Fast, UR_University, UR_Tutorial, UR_Consistent_Maze, UR_Dean_Object.
Very clear what these requirements do with no overlap with each other or redundancy. The specifications were detailed enough for implementation. They were all successfully implemented without trouble and allowed creative freedom.

Quality and Performance Requirements:
UR_Performance, UR_Licensed_Assets, UR_Standard_Computer, UR_Desktop_Inputs, UR_Accessibility.
Used the same hardware and third party resources so no need for change.
UR_Licensed_Assets is immutable legal constraint. UR_Accessibility is a May priority therefore was only implemented if seen as necessary.

User Experience Requirements:
UR_Audio, UR_Score, UR_Main_Menu, UR_Settings.

Target Audience Requirements:
UR_Audience, UR_Positive.
UR_Audience clearly defined and difficulty level implementation does not contradict requirement as allow easy, medium, hard.

Functional Requirements:
FR_Main_Menu, FR_Tutorial, FR_Audio, FR_Map_Limits, FR_Performance_Display, FR_Score_Counter, FR_Event_Counter, FR_Pause, FR_Timer, FR_Losing_Time, FR_Pause_Menu, FR_Settings, FR_Chasing_Dean.
Original functional specifications detailed and complete. Covered UI design, technical implementation, user interactions, and system behaviors. All specifications provided sufficient detail for implementation and were successfully implemented without modifications.

Non Functional Requirements:
NFR_Time_Constraint, NFR_Assets, NFR_Usability, NFR_Sys_Reqs, NFR_Accessibility
NFR_Time_Constraint brief stayed the same at 5 minutes. NFR_Usability kept the same to know how to implement for successful testing. Legal and technical constraints are immutable.

Method selection and planning:

Old URL: https://thomas-nolan.github.io/team5projectsite/docs/Plan1.pdf
New URL: https://thomas-nolan.github.io/team5projectsite/docs/Plan2.pdf

This section summarises and justifies the changes made to the original team's method selection and planning. The aim being to develop the documentation to more closely show our continued development practices, clearing up the planning process, and reflecting the iterative replanning that went on.

Major changes:

- The original deliverable implied a largely fixed plan despite it being agile and as such was changed to acknowledge the provisional nature of early estimates. The new deliverable in turn displayed the continuous need for revision as the length of task became clearer. This change was done in order to adhere to the ENG1 guidance, which states that plans change over time and as such require constant re-planning.
- The document also now explicitly presents Kanban as part of the agile framework, explained by the fundamental agile principles. These principles being: incremental changes - where work is delivered in small packets, adaptive planning - in which continuous replanning enables the team to adapt to changing developments and finally simplicity of process. The simplicity of the process is due to Kaban's straightforward approach, which requires minimal setup.
- The methodology choice is now clearly linked to the project's constraints of being a smaller team, the need to coordinate work remotely and the limited access to the customer. This is something which further reflects the ENG1 suitability, whilst being mindful of customer constraints.
- In addition there was also the reframing of Kanban as a planning and organisation tool, in which it is now described as supporting task level planning allocation. Furthermore it is also described as managing parallel work and reprioritising tasks when plans change.
- The new documentation also explicitly states the planning approach being constraints first. It states constraints prior to plan creation being the required technology of Java and LibGDX, as well as the required documentation tools. The documentation also reinforces the constraints of the remote nature of work and how we overcame it as a team. All of which help in strengthening the traceability between the planning constraints and planning decisions.
- UML diagrams and Gantt charts are also now explicitly used to show planning purposes instead of leaving open ended questions about our planning. These being WBS diagrams, which were used for decomposing the deliverables and Gantt charts to visualise timing constraints as well as deadlines.

- Finally a clearer definition of work packets now described the scope of deliverables and their tasks. In addition these packets were assigned to individuals or small groups and were given intentionally pessimistic time estimates allowing for extra time in case of unforeseen difficulties arising. Moreover the progress of these packets were monitored to ensure delivery time.

Minor and sequential changes:

- Improved integration of diagrams, which were now introduced contextually within the text rather than at the end. This in turn improved clarity, strengthening the link between the diagrams and the text explaining them, aligning with ENG1 expectations.
- In addition all planning diagrams were changed to reflect the work completed by the team, including new tasks and dependencies that arose from development. This in turn meant the diagrams created accurately reflected the workflow.

This strengthened section now explicitly ties Kanban and project planning to Agile principles whilst also integrating ENG1 guidance. Furthermore the new deliverable highlights the iterative and adaptive nature of planning, displaying the areas that need redevelopment as the project progresses.

Architecture

Old URL: https://thomas-nolan.github.io/team5projectsite/docs/Arch1.pdf
New URL: https://thomas-nolan.github.io/team5projectsite/docs/Arch2.pdf

This section summarises and justifies the major architectural changes made to the original team's Assessment 1 deliverable. The aim is to display and simplify the major changes that were undertaken to improve the architecture of the game.

Major Architectural Changes:

- The architectural diagram was updated to explicitly show packets and layers to clearly show the system as monolithic. In addition excess detail was removed as the original diagram was overly detailed and risked penalty under the assessment guidelines. The revised diagram focused on architectural intent rather than implementation detail, improving its clarity.
- Direct navigation between UI Screens was removed and in turn replaced with a UIController, to manage all UI transitions as well as interactions. This resulted in UI screens no longer depending on each other, this in turn resulted in a central navigation logic between screens, reducing coupling (Refactoring Guru, n.d.).
- The old RoomManager class was refactored into more manageable and cohesive subsystems such as DoorController and PlayerController as well as others. The original class acted as a God class and violated the Single Responsibility Principle by handling multiple different unrelated tasks. Refactoring this class improved cohesion, readability, and maintainability, aligning with the SOLID design principles (TheServiceSide, 2023).

- A central GameController was added to coordinate gameplay flow. Other systems now communicated through the GameController instead of directly to each other. This in turn prevents circular dependencies between managers (Refactoring Guru, n.d.).
- The Abstract event class was replaced with an interface, this being IEvent. This in turn eliminated dependencies on concrete classes and fixed inheritance hierarchies. As a result this promoted looser coupling, clearer boundaries and improved testability. This approach aligns with the Interface Segregation Principle by encouraging dependency on abstraction rather than implementations (TechTarget, 2023).
- New requirements meant new managers and classes had to be implemented. Some examples being UR_Achievements, UR_Leaderboards, UR_Invisible_Events, UR_Loose_Condition and UR_Dean_Object. This led to the addition of managers, such as AchievementManager and FileManager as well as updating existing systems. These changes were seen as necessary in order to maintain the requirements' traceability. Moreover these changes were implemented without major restructuring due to the modular nature of the architecture.

Minor and consequential changes:

- Changes were made to the sequence diagrams to reflect the new controller based architecture as well as the new requirements and a new game sequence. These new requirements included UR_Achievements as well as UR_Dean_Object. This was done to ensure sequence diagrams accurately reflected updated behaviour and traceability to the requirements.
- The state diagram remained largely unchanged, with minor updates to include losing via the dean. This change was done to maintain the accuracy of the previous diagrams while reflecting the additional loose condition
- The Component diagram was simplified to highlight the controllers as mediators and reduce visibility density. This in turn improved readability and demonstrated the modular structure of the architecture without losing the essential information.

As seen by the changes stated, the revised architecture has reduced coupling, improved modularity, and supported evolving requirements. Moreover, the major changes are consistently reflected across the structural, behavioural, and architectural diagrams. This consistency in turn demonstrates effective change management, ensuring traceability.

Risk Mitigation:

Old URL: https://thomas-nolan.github.io/team5projectsite/docs/Risk1.pdf
New URL: https://thomas-nolan.github.io/team5projectsite/docs/Risk2.pdf

From reading through the documentation of the previous team, there were two main sections to look into, the introduction, methodology and approach taken for the document and the risk register. When looking at the previous team's work there wasn't much to add due to it being more formatting and readability. We did not add any new risks due to the risks already covering a large area of our project.

Change (1)
We added an introduction to the document clearly laying out what approaches were taken. When reading through the document, they clearly had the four stages of Somerville's process due to the document and the risk register. The four stages of the process are risk identification, analysis, planning and monitoring. We added an explanation of why this process was used.

Change (2)
Formatting was changed to split up the paragraphs in regards to the separate stages of Sommerville's process. Each process was explained, something that was already present in the document, but we edited for readability.

Change (3)
We added an introduction to the risk register explaining what it is and how it's sorted.

Change (4)
We sorted the register table. It is now sorted primarily by type so now it is all grouped together in the correct groups. It is secondarily ranked by severity. It was important that it was ranked by severity so it could highlight risks we should look out for the most frequently in risk reviews. Thirdly it was ranked by likelihood. The higher the likelihood and the higher the severity the more serious the risk is.

Change (5)
We changed the assigned owners to our group members due to us overtaking the project. We had a group meeting and discussed what risks we were most confident in owning, we aimed to split these risks evenly.

Bibliography

- DeMarco, T. and Lister, T. (2003) Waltzing with Bears: Managing risk on software projects. New York: Dorset House publishing.
- Refactoring Guru (n.d.) Mediation. Available at https://refactoring.guru/design-patterns/mediator (Accessed : 18th November)
- Sommerville, I. (2016) Software Engineering. 10th edn. Harlow: Pearson Education.
- TechTarget (2023) An intro to the SOLID principles of object-orientated design. Available at : https://www.techtarget.com/searchapparchitecture/feature/An-intro-to-the-5-SOLID-principles-of-object-oriented-design (Accessed: 18th November)
- TheServiceSide (2023) How to Apply the single responsibility principle in Java. Available at : https://www.theserverside.com/tip/How-to-apply-the-single-responsibility-principle-in-Java (Accessed : 18th November)