Change Report

Cohort 2 Team 1 (Assessment 1)
Ahmet Abdulhamit
Zoey Ahmed
Tomisin Bankole
Alanah Bell
Sasha Heer
Oscar Meadowcroft
Alric Thilak

Cohort 2 Team 2 (Assessment 2)
Bader Albeadeeni
Dan Hemsley
Jennifer Bryant
Mathilde Couturier-Dale
Oliver Elliott
Rosie-Mae Connolly
William Mutch

Change Report

Processes, Tools, and Conventions

In order to ensure a smooth transition from assessment one to two, we have taken several steps as a team. This includes:

- Every team member must have a full in-depth look at all of the documentation and code used in team one's project, taking the time to read and understand what is going on.

- We have created fixed team roles, ensuring everyone has an even amount of marks. Each team member chose a role and tasks were adjusted to ensure everyone was happy and in a role suited to them and their skills.

- Deadlines have been set earlier in advance, and meetings have been scheduled weekly to ensure we are all on the same page and on track in the lead up to Christmas. A Gantt chart has been created to show this evolution.

- A log book has been created to record our progress and any concerns we have each week. See here (Weekly Log). We will also use this to track attendance and who is falling behind with their section of marks, so we can work quickly and efficiently to resolve it.

- GitHub was used to manage all the code and documentation changes through version control. The development was done using IntelliJ IDEA, and Discord was used for regular team meetings. WhatsApp was also used for continuous updates from all group members as well as routine peer review, allowing all team members to question and clarify each other's work, ensuring transparency and accountability

- Assessment two comes with an updated product brief. This includes features like a leaderboard that shows the name and score of the top five player attempts. In total, we need to implement eight new features as well as 'achievements' that can affect the score positively or negatively. We have created a list of modifications to be made to the code before implementing these new features (see here).

- It is important for the coding team that we rebuild the game using our own platforms and tools to make sure the integration is correct and working before we move any further. From there we can move ahead quickly with coding and testing.

Change Report

i. Requirements

Summary of Changes

The project requirements were updated following the transition from team 1 to team 2. The original format and structure of the requirements document was maintained while additional requirements were introduced to meet the Assessment 2 specifications. Specifically, these new requirements introduced the leaderboard and achievement functionalities as well as various other inclusions which were required to make these changes possible such as the JSONHandler and events classes. Like in the original document, requirements were divided into User Requirements, Functional Requirements, and Non-Functional Requirements. Some requirements were removed as they were no longer applicable to the final game. For example, the requirements to pause the tutorial were not necessary as the fully-functioning tutorial was implemented as a singular non-interactive screen.

Requirements removed:
NFR_TUTORIAL_LOADING,NFR_TUTORIAL_PROGRESSION, FR_TUTORIAL_PAUSING

Documentation Challenges

When inheriting the project, the original requirements document was provided in a PDF format. Converting this back into an editable google doc caused some problems with the formatting of the various tables. As a result, the document required significant clean up before it was ready to incorporate the new content. All major changes were written in red text to clearly distinguish them from the original document.

Scope of Requirements

Requirements related to user-visible features were added to the game with additions primarily focused on gameplay events and scoring. Changes were also made to the internal behaviour during development, such as changing the map layout and the introduction of diagonal movement for the player. However, these changes were not included in the user/functional requirements because they were more geared towards refining implementation than adding new system functionality.

Team 1 Original Document: Req1.pdf
Team 2 Extended Document: Req1Edited.docx

User Requirements Table

ID	Description	Priority
UR_LEADERBOARD	The system shall display a leaderboard showing the top 5 scores and the user's final score.	Shall
UR_LEADERBOARD_NAV	The user shall be able to return to the main menu from the leaderboard screen.	Shall
UR_ACHIEVEMENTS	The user should be notified when an achievement is unlocked.	Should

UR_BOOK_EVENT	The system should include an interactable book event which takes the player to a quiz.	Shall
UR_QUIZ_EVENT	The system shall present the player with a quiz screen with multiple choice questions.	Shall
UR_BUS_EVENT	The user shall be able to find and collect a bus ticket in order to complete the game.	Shall
UR_REPELLENT_EVENT	The user shall be able to collect an item which temporarily stops the dean from catching them.	Shall
UR_LOCKER_EVENT	The user shall be able to interact with a locker which gives the player a temporary speed boost.	Shall
UR_NPC_EVENT	The user shall be able to interact with an NPC and see dialogue.	Shall
UR_BIRDSEEDS	The user shall be able to obtain birdseeds and see an on screen indicator.	Shall
UR_GOOSE	The system shall include a goose which follows the player until fed.	Shall
UR_SAFE_EVENT	The user shall be able to interact with a safe and open it with a code.	Shall
UR_CODEPAGE	The user should be able to view a code page to obtain part of safe code.	Should

Functional Requirements Table

ID	**Description**	**User Requirements**
FR_LEADERBOARD	The system shall record a final score to a local leaderboard, load the top 5 entries, and display them. It should then allow the user to return to the menu.	UR_LEADERBOARD UR_LEADERBOARD_NAV
FR_ACHIEVEMENT_DISPLAY	The system displays an achievement notification which can be dismissed	UR_ACHIEVEMENTS
FR_BOOK_INTERACT	The system shall allow the player to interact with a book when within range by pressing the interaction key, taking them to a quiz screen.	UR_BOOK_EVENT
FR_QUIZ_DISPLAY	The system shall display a quiz question with four answers which can be selected with the mouse input.	UR_QUIZ_EVENT
FR_QUIZ_RETURN	The system shall return the user to the game after an answer is selected.	UR_QUIZ_EVENT
FR_TICKET_COLLECT	The system shall allow the player to collect a bus ticket once they have found it.	UR_BUS_EVENT
FR_TICKET_UI	The system shall display a bus ticket icon once it is collected.	UR_BUS_EVENT

FR_DEAN_REPEL	The system shall allow the player to collect an item when nearby which will alter the dean's behaviour.	UR_REPELLENT_EVENT
FR_REPEL_RENDER	The system shall remove the repellent from the game once it has been collected.	UR_REPELLENT_EVENT
FR_SPEED_BOOST	The system shall allow the player to interact with a locker when nearby and apply a temporary speed boost.	UR_LOCKER_EVENT
FR_LOCKER_MESSAGE	The system shall display a message when the locker is searched.	UR_LOCKER_EVENT
FR_NPC_INTERACT	The system shall allow the player to interact with an npc when nearby which displays text for dialogue.	UR_NPC_EVENT
FR_NPC_HIDE	The system shall hide the npc's dialogue when the player moves away.	UR_NPC_DIALOGUE
FR_BIRDSEEDS_UI	The system shall display a birdseeds icon when seeds have been obtained.	UR_BIRDSEEDS
FR_GOOSE_FOLLOW	The system shall update the goose position so that it follows the player while it is not fed.	UR_GOOSE
FR_GOOSE_STOP	The goose shall stop following the player once it has been fed or once the player is far enough away.	UR_GOOSE
FR_SAFE_INTERACTION	The system shall allow to interact with a safe, enter a code, and receive feedback based on the result.	UR_SAFE_EVENT
FR_CODEPAGE_VIEW	The system shall allow the user with a codepage to view the text.	UR_CODEPAGE

ii. Architecture

Team 1 Original Document: **Arch1.pdf**
Team 2 Extended Document: **Arch1Edited.docx**

Architectural Extensions

Following the transition from team 1 to team 2, the existing game architecture was reviewed and extended to include the new functionality. Multiple new events and a leaderboard system were added by introducing new classes without heavily modifying the core structure of the original code. These additions were integrated with the existing structure through pre-existing classes like GameScreen and WinScreen. This was done in order to ensure continuity and consistency as well as for the ease of continued development.

Documentation Challenges

Like the requirements documentation, the original architecture document was provided to the inheriting team only in PDF format. When converted into an editable word document, the formatting became unusable, requiring it to be recreated so that it matched the original PDF

exactly. Furthermore, architectural diagrams were not included directly in the original document and were instead provided as screenshots on the website. As these were not editable, the team contacted the original developers via email to obtain the original diagram files. When we did implement changes, they were recorded in red text to clearly distinguish post-inheritance updates from the original document.

Event Extensions

Multiple 'events' were added including a book with a quiz, a bus ticket required to escape, a locker containing an item, and a dean repellent for protection. Existing features were also re-modelled into events such as the NPC interaction. Each event was implemented as an independent class responsible for managing its own logic. This is consistent with the original design of the code by having intractable objects manage their own behaviour, while being initiated by the GameScreen.

Interaction with these events is generally triggered when two conditions are met. Firstly, there is a proximity check (the player must be close enough to interact) and secondly, there must be a correct user input (pressing the E key). This is kept the same for all events in order to maintain consistency with how the player interacts with the game. While the interaction model is the same, each interaction has a distinct outcome such as a quiz, speed boost, revealing of a hidden item, or ability to alter the dean's behaviour.

Leaderboard and Persistence

The leaderboard system was implemented with the introduction of a JSONHandler class. This class is responsible for retrieving data for the leaderboard from a local file (leaderboard.json). This ensures that data is maintained when the system is offline.

Like with the events classes, the leaderboard logic is kept isolated. Insertion and retrieval of the scores is handled by the JSONHandler while the presentation of the leaderboard onto the screen is handled by the relevant screen class. This allows for the leaderboard to be modified independently making it easy for continued development.

Screen Integration

Some events integrate with the already established screen management system. For example, the book event triggers a transition to a dedicated quiz screen using the MyGame.setScreen() component. The quiz functionality is held completely within its own screen rather than being made part of the main game logic.

The new QuizScreen class maintains the same screen lifecycle approach which the original team took by managing its own input handling, rendering, and transitioning back to GameScreen. This is aligned with the other screens, thus seamlessly integrating the event.

Separation of Game Logic and Rendering

The original codebase tightly coupled game logic with LibGDX rendering, asset loading, and input handling, which made automated testing impractical without a graphical runtime environment. To fix this, core decision and state-management logic was extracted into dedicated logic classes, separating it from rendering and UI code, e.g. DeanLogic.java for Dean.java. This architectural change makes deterministic unit and integration testing possible without requiring LibGDX screens or assets. Rendering and UI behaviour is

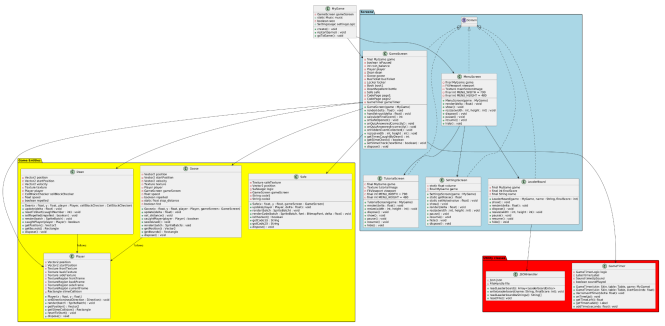
validated through manual testing. Separating these components improves test coverage and maintainability of the code base.

CRC Cards Table

Class	Responsibilities	Collaborators
Leaderboard	Screen which writes name and final score to leaderboard if the score is greater than 0. Also loads top scores and returns to menu when space is pressed.	MyGame, MenuScreen, JSONHandler, Screen
JSONHandler	Reads and writes leaderboard entries to leaderboard.json. Sorts the scores numerically.	LeaderboardEntry, Gdx.files.local, Json, Leaderboard
LeaderboardEntry	Represents a single leaderboard record. Contains a position, name, and score.	JSONHandler, Json
Achievement	Manages achievement rendering.	SpriteBatch
Book	Detects a player interaction and transitions to the quiz screen.	Player, QuizScreen, MyGame, GameScreen
QuizScreen	Displays a quiz question, takes a user input and shows whether it was correct or incorrect. Returns to the game screen.	MyGame, GameScreen, Gdx.input
BusTicket	Controls discovery and collection of the bus ticket required to complete the game as well as the rendering of the UI icon.	GameScreen, OrthographicCamera, SpriteBatch
DeanRepellent	Handles the player interaction, collection, and usage of the dean repellent.	Player, Dean, GameScreen
Locker	Handles the player interaction with the locker, speed boost, and displaying of message.	Player, GameScreen
NPC	Handles the interaction with an npc which is a proximity based text display.	Player, GameScreen
BirdSeeds	Stored birdseed state using isBought and isUsed and renders a UI icon when bought.	GameScreen, OrthographicCamera. SpriteCamera
Goose	Follows the player and stops when fed.	Player, GameScreen
Safe	Manages player interaction, code entry, and validation when using the safe.	Player, GameScreen
CodePage	Handles interaction with rendering an enlarged page with text displaying a code.	Player, OrthographicCamera, CodePageLogic, GameScreen

Structural Diagram

We created a new structural class diagram in UML to show the main classes in the system and to show the additional classes that were added after taking over the project. The structural diagram shows all the screens in the game, some game entities and utility classes. Not every game entity is included in the diagram as this would make the diagram cluttered and less readable. As a result, the game entities shown are the player and dean as they are major classes and the as well as the goose and the safe as these were new classes added to the game in assessment 2. The class diagram added is an extension of the final OOP component diagram created by



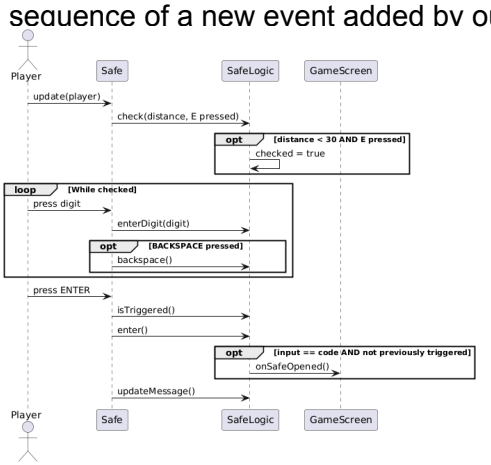
the previous team, keeping the structure of grouping the screens, game entities and utility classes together.

Behavioural State Machine Diagrams

We determined that not all events and other classes required a state machine diagram to be represented. The two classes in particular which we decided to include (Player.java and QuizScreen.java) were chosen due to their logic consisting of multiple changes of state.

Sequence Diagrams

The original document referenced 3 use cases for sequence diagrams of player movement, dean's pursuit and player searching objects. The sequence diagram we added was the safe event which models the interaction between the player and the safe with the logic used. The diagram captures the chronological order of steps when the player tries to interact with the safe object. This was a needed change to the architectural document as it shows the sequence of a new event added by our team after inheriting the project.



(All diagrams can be found in Arch1Edited.docx)

iii. Method Selection and Planning

Team 1 Original Document: Plan1.pdf
Team 2 Extended Document: Plan1Edited

Methodology Extension

We extended the chosen methodology section and the justification to account for our team taking over the project. We continued with the same agile approach as the previous team so the additions to this section were explaining how this approach suited us and the specifics of the Scrum methodology we used. Furthermore, an 'alternatives considered' section was added to explain why we didn't continue with the waterfall style approach is assessment 1 and how the iterative approach benefits the implementation heavy assessment 2.

Tools Extension

The tools section had limited additions as many of the tools used by the previous team in assessment 1 are the same tools we are using to extend the project. We mentioned draw.io that was used to create some of the architectural diagrams and we explained the use of LibGDX to develop the project.

Team Organisation & Communication Extension

The main additions to this section were what roles each person had and how the workload was divided up after we took over the project. In the dividing roles extension we explained the use of splitting the team in half and having one half focus on implementation while the other half focuses on documentation. This was an essential change in the documentation as it explains how our team's approach was different from the previous teams. Instead of tasks being divided evenly among members there was a clear divide between documentation and development and we explained our justification. Another addition in this section was a list of our Scrum roles and the tasks that accompanied each role. This addition was needed as it shows what roles we implemented after taking over the project and how these differed from the previous teams roles but still followed the Scrum methodology framework.

Timeline Extension

The table below is an extract of the timeline table showing which tasks are being completed each week, their priority and their dependencies on other tasks. These include the newly added tasks for assessment 2:

iv. Risk Assessment and Mitigation

Team 1 Original Document: **Risk1.pdf**
Team 2 Extended Document: **Risk1Edited**

Risk Management Extension

As the project was inherited from a previous team, the risk assessment management process was extended to account for additional risks introduced with the transition. A dedicated period was allocated at the beginning of the development to reviewing, understanding, and documenting the inherited code and existing documentation in order to identify risks. These risks were recorded by maintaining the same 4-step process (identification, analysis, planning/mitigation, monitoring) and risk table format to ensure consistency across the project. Most of the identified risks related to unfamiliarity with the project as well as potential issues with the original submission. 4 types of risks were identified: technical risks; organisational and team risks; scheduling and delivery risks; and quality and requirement risks.

This approach was chosen to minimise disruption to the original management process. By keeping the original structure, it was easier to directly compare new risks to those previously identified. Like with the architecture and requirements document, the original Risk Assessment document was provided in the form of a PDF which, when converted into an editable google doc, became unusable. Once again, much of the document had to be recreated from scratch to match the uncorrupted PDF. This was done during the mentioned familiarisation period at the beginning alongside reviewing and understanding the original group's submission.

Likelihood Key:	Impact Key:
Likely = 5/5	Very High = 5/5
Possible = 4/5	High = 4/5
Unlikely = 3/5	Moderate = 3/5
Rare = 2/5	Low = 2/5
Very Rare = 1/5	Very Low = 1/5

Change Report

Updated Risk Table

The tables below are extracts from the risk assessment document which include the newly identified risks relating to inheriting the project and was produced using the same format as the original risk assessment:

Technical Risk ID:	**Risk**	**Likelihood**	**Impact and Severity**	**Mitigation**	**Ownership**
TR7	Inherited codebase complexity	Likely	4 - Unfamiliar or poorly structured code from the first team may slow new implementation with the second team.	The team can review original code and document classes and systems.	Developers analyse the code. Scrum master monitors progress.
TR8	Hidden dependencies	Likely	3 - Undocumented dependencies between classes might cause unexpected behaviour.	Class dependencies and relationships should be reviewed and documented before making changes. May require debugging.	Coders and architecture documentors should review, document, and share any useful information.

Organisational and Team Risk ID:	**Risk**	**Likelihood**	**Impact and Severity**	**Mitigation**	**Ownership**
OTR5	Lack of knowledge transfer from previous team.	Likely	3 - Missing knowledge about design could cause mistakes due to misunderstandings.	Team will go through code and documentation together and contact the previous team for information if needed.	All members should participate.
OTR6	Inconsistent code practises	Possible	2 - Differences between the first team's code and second team's code could damage readability and maintainability.	Coding standard can be agreed on by the inheriting team based on the original code. Exceptions can be made.	Developers can follow the agreed standards.

Scheduling and Delivery Risk ID:	**Risk**	**Likelihood**	**Impact and Severity**	**Mitigation**	**Ownership**
SDR3	Slow familiarisation time	Likely	3 - Could take more time than expected to understand the inherited project, delaying development.	Time for familiarisation should be accounted for when sprint planning.	Members of coding and documentation teams should report progress to the entire group. Scrum master ensures that timing is on track.
SDR4	Defects in inherited code	Possible	3 - Pre-existing bugs may require unplanned fixes which could disrupt development.	Prioritise early testing and address bugs before implementing new features.	Coders can identify and fix bugs in the code.

Quality and Requirement Risk ID:	**Risk**	**Likelihood**	**Impact and Severity**	**Mitigation**	**Ownership**
QRR4	Inherited project not meeting current requirements	Possible	4 - Inherited project might not meet all requirements which will take time to go back and implement before continuing with further development.	Requirements should be reviewed so that gaps can be documented and prioritised.	All members should bring light to gaps in their designated areas so that they can be prioritised.

Reference

Yuan,T(2025). Change Management. Department of Computer Science, University of York