Change Report

Cohort 3 Team 4

Kiran Kang

Hannah Rooke

Ben Slater

Abualhassan Alrady

Cassie Dalrymple

Charles MacLeod

Dash Ratcliffe

Harley Donger

Change Report Overview
Our team used an organised agile approach when reviewing group 7's deliverables and code and making any necessary changes to the existing project, as well as when developing new features in the documentation and game itself. Our goal was to make sure we completed the product brief for Assessment 2 by making high quality changes. To support this process all major changes were given traceable identifiers in the updated report allowing people reviewing the report to easily locate each change. The change report explains what modifications are made and the justification behind them. Implementation changes were tracked using a shared google doc on what was changed/added to the game.

Processes
Our team decided to continue using an agile methodology, using scrum principles. This is because we had already used this in Assessment 1, and are therefore not only familiar with this structure, but also understand that it worked in terms of meeting goals we had set to stay on track and meet deadlines.We made weekly meetings where our main goal was to produce a new piece of code. This meant we were able to make gradual progress by coming together to review the new features that had been developed and discuss how we can further improve these. By using these short sprints, we were able to build up to the final game that incorporated all the new events efficiently.

Tools used
GitHub - used for pull requests, merging branches and managing the inherited code.

IntelliJ - IDE for Java using LibGDX, helped with debugging and code structure.

GitHub issues + projects - used for assigning tasks and tracking headway in terms of code progression

Google Docs/Drive - used to collaboratively edit/structure files and write up deliverables.

Conventions (change management)
We used Lewin's 3 stage model of change as a convention when planning how to develop Group 7's code and deliverables. The 3 stages of this framework are listed below, along with an explanation of how we applied each stage to our project.

Unfreeze: We came together as a group to review the Assessment 1 deliverables, code and documentation; we discussed what changes we could make to the existing changes and how we could develop pre-made features.

Change: Our team focused on implementing new changes using the GitHub branches, IntelliJ and Google Docs. This was iteratively processed and refined based on feedback that followed from using the agile methodology.

Refreeze: When everything was reviewed and approved by all members of the team, all deliverables and code were finalised in their correct file format to hand in for the assessment to prevent further modifications.

Software Maintenance:
When inheriting the group's code, we decided to take on a software maintenance approach that combines corrective, perfective and preventive maintenance. Corrective maintenance prioritises fixing errors/bugs in the software system, led by Ben and Charles. Perfective maintenance prioritises on functionality, performance and reliability of the game. Preventive maintenance involves preventing future issues; this is done through optimisation, updating documentation and testing the system, led by Dash and Abualhassan. Continuous integration via GitHub actions which supports all three maintenance is led by Cassie.

REQUIREMENTS:
All changes made in the updated report will be under *EDITS* alongside with its traceable identifier so it'll be easy to find.
Report with no edits: https://glitchgobblers.github.io/public/deliverables/Req1.pdf
Final report with edits: https://glitchgobblers.github.io/public/deliverables2/Req2.pdf

The requirements deliverable from Assessment 1 began with a brief explanation of how the requirements are presented, followed by a description of the process the group went through to arrive at their final list of requirements.

When we looked through this, we decided that it needed more detail, and that it lacked sufficient evidence of research into requirement elicitation and specification. Further detail in regards to this had been added in the *EDITS* section at the end of the Req1_group7 deliverable; arriving at a conclusion of what to add was based on our experience when coming up with new requirements to consider, as well as just a general expansion of the material we were given to work with.

When we got around to looking at the tables of requirements, we thought they were relevant to the project, as they covered many of the features that the customer required. On top of this, the traceability aspect of this deliverable was helpful for us, as it meant we were easily able to identify the links between both the system and the requirements - we decided to keep the image that shows this for that reason.

The *'Non-functional requirements'* table in this deliverable had a few areas we needed to tweak. Firstly, the '*NFR_RUNTIME*' requirement should be renamed to *'NFR_RELIABILITY'*, as the previous name seemed slightly misleading - the old name was not really to do with the running time of the game, more so how well the game runs.
Secondly, the *'NFR_GAME_TIME*' requirement is about the system functionality/capabilities - this is a functional requirement, so it would make sense to switch the table it is a part of.
Both of these changes are reflected in the new tables at the end of the original Req1_group7 deliverable under the *EDITS* section.

In terms of details we thought were missing, there were a few requirements we came up with as a team that were not present in the tables from Assessment 1.
After looking through the tables, it seemed there were no requirements to do with player movement or collisions in general. These seemed extremely important given the nature of the game, so we made sure to include these in the updated tables.
We also considered areas like maintainability and accessibility which again did not seem to be covered within the original requirements. These factors seemed quite broad, so including them would apply to a range of game factors.
All of the changes listed above can be found in the tables in the *EDITS* section, but some more specific requirements we added are explained below.

'*UR_ACHIEVEMENTS'* and '*UR_PLAYER_NAME'* were added, since player naming and achievements are explicitly described in the product brief for Assessment 2 as main gameplay features. The achievements are earned by meeting a certain criteria depending on the player's actions to reward the player, such as exploring or interacting with events. Likewise, a player name is required to associate scores with individual players and to display that score on the leaderboard if the player has one of the top five highest scores. These are necessary as user requirements because these features directly affect the player's experience and therefore it is imperative for them to be captured as user requirements.

To support these UR requirements, we included corresponding functional requirements - '*FR_ACHIEVEMENT_TRACKING'* and *'FR_LEADERBOARD_STORAGE'*. These two requirements specify that it's the system's responsibility to track the player's actions and

unlock these achievements if the player has met a certain condition on top of storing and displaying the top player scores. These functional requirements needed to be present in the implemented features in Assessment 2 so they could be traced back to the defined system behaviour, which was not something that was explicitly stated in the Assessment 1 requirements.

A new non-functional requirement, '*NFR_DATA_LEADERBOARD'* was added to address the data persistence and reliability, as the leaderboard data is needed to be stored across gameplay sessions even if the player quits the game session. This ensures that the system's behaviour stays consistent and aligns with the user's expectations.

Overall, the original Req1_group7 deliverable gave us a good basis to work with, as the general requirements had already been established; this meant we were able to focus on identifying areas that had not previously been thought about. In addition to this, we were able to ensure full coverage of the product's brief features by adding new requirements to improve traceability between requirements, architecture, software testing, continuous integration and implementation. This strengthens the requirements report as a solid foundation for system evolution, maintenance and evaluation. Once we had considered this, we felt confident that the essential requirements were now a part of the updated deliverable.

ARCHITECTURE:
All changes made in the updated report will be under *EDITS* alongside with its traceable identifier so it'll be easy to find.
Report with no edits: https://glitchgobblers.github.io/public/deliverables/Arch1.pdf
Final report with edits: https://glitchgobblers.github.io/public/deliverables2/Arch2.pdf

*MAJOR CHANGE - NEW CLASS DIAGRAM (**MJ_CDIAGRAM**)*
In Assessment 1, the final class diagram we inherited specified that they followed an OOA approach using OOP principles. This approach separated its responsibilities between the core game logic, entities and the UI components. For Assessment 2, this architecture style was retained by continuing to use PlantUML from Assessment 1 to ensure consistency and to avoid altering the system's structure unless necessary to add new features or fix bugs.

Rather than redesigning the system, the final class diagram was extended to introduce aggregation, composition and inheritance to accurately show the relationships of how new and existing classes interact with the evolved system. For example, in the Architecture doc the new different event types from the product brief now inherit from the abstract class Event. Whereas, the AchievementSystem or Leaderboard are composed or aggregated within the Game class to represent ownership and lifecycle dependencies.

These changes were needed to improve the traceability, clarity and maintainability of the new architecture structure. The class diagram in assessment 1 only showed simple associations for not many classes meaning that there were no associations shown to represent ownership or specialisation as the system evolved. By introducing a more suitable association and inheritance. The new updated diagram better reflects the current implementation and supports software maintenance and aligns with Assessment 2's change management and how the architecture is evolved.

This change is reflected in the new Architecture report in the *EDITS* section below MJ_CDIAGRAM showing the new classes AchievementsScreen, AchievementSystem, HiddenEvent2,PositiveEvent2,NegativeEvent2,Event and Leaderboard added. Including new types of associations and how some classes has been inherited from.

*MAJOR CHANGE - EXTENDING ARCHITECTURE'S EVOLUTION (**MJ_EVOLVE**)*
In Assessment 1, the previous group explained how their architecture evolved through a series of processes that was shown in their website. For Assessment 2, this approach was extended to include the evolution of the architecture during our own development period.

This change was important to present how the inherited architecture was incrementally extended rather than redesigned. Which was done by documenting the process with dates, images and brief explanation. (Shown in the website page provided in the change report). Where the evolution in the change report demonstrates how it's changed over time and why these changes were necessary. Without this, only presenting a final class diagram would obscure the design decision made throughout Assessment 2 and make it harder to understand how the system evolved.

Including the architecture evolution process will improve traceability between the requirements, design decisions and implementation. This supports the team's agile process and its change management for its system evolution. (word better). It provides clear justification for the final class diagram by showing how it emerged through structural and incremental changes rather than a single redesign.

This change is reflected in the new Architecture report in the *EDITS* section below MJ_EVOLVE. This section explains how the architecture has evolved through a series of processes

*MAJOR CHANGE - NEW STATES AND SEQUENCE DIAGRAMS (**MJ_STATES**)*
r - add sequence diagrams
In Assessment 1, the previous group only added a limited number of state diagrams, such as the Timer state and the Score state. While these diagrams were correct they were simple and were not able to fully capture the dynamic and interactive behaviours in the system. However, the other states, Dean and Player state, could've been more interesting by providing a sequence diagram instead to be more explanatory.

For Assessment 2, the report was extended by introducing more meaningful state and sequence diagrams that better reflect the system's runtime behaviour. Instead of using simple state diagrams, sequence diagrams were used to explain more complex interactions between core components. These interactions are more effectively demonstrated through sequences as they show the order of method calls and the system's response to the player actions.

This change was important to make sure there was better clarity of the architecture report. By replacing less informative state diagrams with sequence diagrams. This provides a more clear understanding of gameplay flow such as an event being triggered, how score increases and decreases and achievement unlocking through the Dean and player state. This supports the comprehension of the system's behaviour and aligns with Assessment 2's evolution of architecture and its documentation.

This change is reflected in the new Architecture report in the *EDITS* section below MJ_STATES. Where new sequence diagrams are shown and explained for the Player and Event interaction, Player and Dean interaction, Achievement unlocking and the Leaderboard.

METHOD SELECTION AND PLANNING:
All changes made in the updated report will be under *EDITS* alongside with its traceable identifier so it'll be easy to find.
Report with no edits: https://glitchgobblers.github.io/public/deliverables/Plan1.pdf
Final report with edits: https://glitchgobblers.github.io/public/deliverables/Plan2.pdf
*MAJOR CHANGE: - METHODOLOGY APPROACH: (**MJ_METHODOLOGY**)*

In Assessment 1, the method and planning report does not mention any methodology approach like waterfall, agile, scrum etc… Instead, the method and planning report highlighted tools used and a good discussion of how the team's organisation worked.While this is a good foundation to work on, the absence of a defined methodology made it hard to see how changes would be prioritised, reviewed and integrated when inheriting this project.

For Assessment 2, Our team decided to introduce an Agile methodology approach using scrums principles as it will help our team to provide a structured framework for iterative development and continuous feedback. This is important for maintaining and extending inherited code and this change aligns with Assessment 2's focus on software maintenance and change management.

This change is updated and reflected in the method and planning section of the Assessment 2 deliverables in the *EDITS* section at the end of the report under **MJ_METHODOLOGY**. Where agile and scrum practices such as iterations, backlog management and weekly sprints are described. The methodology is also referenced in part(a) of this change report when describing our process and conventions and in the updated systematic project plan for Assessment 2.

*MAJOR CHANGE: - SYSTEMATIC APPROACH (**MJ_SYSTEMATIC**)*

In Assessment 1 in the group's systematic planning they defined the deliverables, general tasks, the dates but there was a limited visibility of the priority and who was responsible for the tasks. This made it difficult to trace changes, see progress and understand how the workload was distributed across all team members.

For Assessment 2 the systematic plan was extended and refined by adding some new changes. This will be done in a table for each deliverable to make things more clear. The updated systematic plan will explicitly define who is responsible for each task, with a priority ranging from high to low, with justification. The tasks from each deliverable will be broken down into smaller tasks and assigned to specific team members. This makes it more clearer of how many people were involved in what area of work.
Moreover, the tasks were expanded and refined to align with the increased scope of Assessment 2. I.e including change management, software maintenance and progress updates. Extending the systematic plan allows work to be distributed evenly across the team, tracked who's done what easily and remains consistent with the Agile methodology using scrum principles. (**MJ_METHODOLOGY**)

This change is reflected in the method and planning section of the Assessment 2 deliverables at the end of the report in the *EDITS* section under **MJ_SYSTEMATIC**. Where each deliverable has its own structured table detailing with ID names, tasks, dates, output, priorities and who was responsible for each task. This plan will follow the agile methodology's **MJ_METHODOLOGY** process.

Overall, introducing an explicit agile process and an extended systematic plan will improve how changes, tasks, responsibilities and its output are managed and defined in Assessment 2. These changes will provide a clear structure and good traceability when maintaining and

extending group 7’s project we’ve inherited from while also remaining consistent with the original tools and its constraints that were defined in Assessment 1.

RISK ASSESSMENT AND MITIGATION:

All changes made in the updated report will be under *EDITS* alongside with its traceable identifier so it'll be easy to find.
Report with no edits: https://glitchgobblers.github.io/public/deliverables/Risk1.pdf
Final report with edits: https://glitchgobblers.github.io/public/deliverables2/Risk2.pdf

The risk assessment and mitigation deliverable we were presented with at the beginning of Assessment 2 had a brief overview of why creating a risk assessment is important for a project. It explained how identifying potential risks early can minimise damage and improve overall project delivery. The process the group carried out to evaluate potential risks was also presented, as well as the reasoning behind the format of their risk register - this risk register was also exhibited.
In terms of steps the group took, it would be clearer if there was more explanation in regards to what they did in practice for risk identification and analysis. To include this within our updated deliverable, we made sure to note the process we followed when identifying new risks that weren't already in the register. This was then explained in the *EDITS* section at the end of the Risk_1 deliverable - this section shows all the changes made to the original deliverable.

When reviewing the risk register, we needed to look at its overall format, as well as the risks themselves that were included.
The explanation at the beginning of the deliverable did not mention the chosen format of the risk register. This again is something we felt necessary to include, so a description of this is also included in the *EDITS* of Risk_1.

When continuing with the analysis of the format, we noticed a few things we needed to tweak/add. Firstly, the register included both a mitigation and contingency plan column - we only required one of these, as the risk mitigation is our contingency plan.

Something we felt was missing was a 'Type' column. When creating our risk assessment in Assessment 1, the inclusion of this meant we were easily able to group risks together in terms of a common factor, such as whether the risk was to do with the user, technical difficulties, testing etc. However, this column was not present in this risk register, so we decided to add it, since this was useful during implementation in Assessment 1, as it meant we were able to refer back to this in case of a problem and quickly identify the associated mitigation strategy. The inclusion of this also meant a more organised, clearer register in general.

The column Risk(ID) included both the name of the risk and its associated ID. We understood that these pieces of information should be split, as referring to a risk by its ID is useful, but with the description of the risk alongside this, it could get confusing to remember. Because of this, we created two separate columns containing this information, as they both needed to be present, just not together.

The last thing we changed in terms of format of the register was to do with the 'Owner' column. Although some risks related to all responsibilities within the project, having 'Everyone' as an owner seemed impractical, as having multiple people overlooking multiple risks could result in quite a bit of uncertainty within the group. Instead, it would be ideal to have one specific member of the group who is responsible for managing and mitigating a certain risk if needed; if this was a lot to handle, others could get involved to help, but we felt having a general overseer of a single risk made it easier to manage.

All of the changes to the format register described above are presented in a new table in the *EDITS* section. This table is the same as the original, just with the changes we deemed it necessary to include.

After the new format was decided, we looked through each potential risk the previous group has included and compared this to our experience in Assessment 1. A lot of these risks were ones we had encountered throughout the earlier stages of game development. These included testing problems, technical difficulties and problems related to the project not meeting customer requirements. Since these risks seemed relevant, given our prior experience, we decided to keep them as part of the Assessment 2 risk register.

Given this, we also decided to add some new risks to the register - we grouped these into timing and product risks. The timing risks we felt were missing were to do with getting familiar with new technologies, as well as completing the required specification on time. For product risks, we needed to consider if the general gameplay was fun and enjoyable for the player, on top of factors like the difficulty of the game itself.
These new risks have also been added to the new table in the *EDITS* section.

Overall, the original risk assessment we were given provided us with a firm foundation for the continuation of implementation of the game. This, combined with the inclusion of some new risks, helped to support a smoother development process, as we felt better prepared to mitigate issues as they arose in terms of adding new functionality.

References:

[1] Priyanka Malik “Lewin’s 3-Stage Model of Change Theory: Overview”
https://whatfix.com/blog/lewins-change-model/
[2] Claire Drumond “What is Scrum? Breaking down the Agile framework”
https://www.atlassian.com/agile/scrum