Change Report

Hannah Thompson
Kyla Kirilov
Ben Hayter-Dalgliesh
Matthew Graham
Callum MacDonald
Chak Chiu Tsang

1.1 Processes, Tools and Conventions for Change Management

When we received the project from Team 21, we conducted an initial assessment of the deliverables, code and supporting documentation. We met as a team to thoroughly assess the inherited files, making notes of any sections that needed further investigation, clarification, or that we thought would need updating in the Change Report. At this point, we created a checklist of tasks to be completed and assigned them to each team member, colour-coding them by priority and grouping them into related workflows. Alongside our Gantt charts (on the website under '*Methods and Planning*') this formed the backbone of the plan for our approach to the second part of the assessment.

To track our changes to the deliverables, we first downloaded the original documents and copied them to make 'updated-' versions. We compared the old versions with our notes, as well as discussing areas which we felt could be improved. By making a checklist of all the areas we needed to update, and marking when we were working on them and when they were completed, the documentation team was able to systematically work through changes to the files as the project progressed.

The website was also updated at this stage - we requested the original html code from Team 21 and made slight changes to the structure to accommodate the new files that would need to be uploaded. We prepared the website by adding filler sections for deliverables that were still being worked on, so that we could come back and easily replace them when completed.

To make changes to the existing diagrams, we requested access to the LucidChart files used by Team 21 and added our updated architectural structures. Similarly with the Gantt charts, we used PlantUML to create charts in the same style as the previous team and seamlessly integrated them into the website.

At the same time, the development team was adding functions to satisfy our updated requirements. They employed a peer-review process to review changes, where any updates to the code were reviewed by at least one other development team member before committing to the main branch. Similarly, changes were made incrementally by each team member, ensuring that each push only updated a small section of the code at a time. This ensured that changes could be easily tracked by all members, and any errors could be fixed by reverting to a recent version.

Throughout the second half of the project, the team continued to meet at least twice a week to discuss that week's changes and plan for the next week. This allowed us to keep up-to-date on what everyone was working on at any given time, ensuring that the whole team understood all the elements of the project, including areas that they were not actively working on.

Changes to the deliverables are presented in this report, separated by the document in which they were made and numbered for traceability. We felt that these changes were necessary to cover the full criteria for both assessments, and they have been reviewed by the entire development team.

2.1 Updates to ‘Requirements’

Original Document: Req1.pdf **Updated Document:** updated-Req1.pdf

We expanded upon the description of how the requirements were presented to include our own research into relevant literature [1] and IEEE standards for software requirements specification [2]. The existing description was limited and did not explain why the requirements were presented in this way.

The elicitation that Team 21 took was similar to our own approach, of consulting with module leaders and discussing the needs of the project within the group. For assessment 2, we had further discussions with the module leaders and GTAs, and met as a group to review the changes to the product brief and the ways in which these changes should be implemented.

We felt that there were some requirements to be added from our own elicitation process. The requirements that we added are in **bold text** - we believe these should have been included in the original documents, as they cover some of the key aspects of the game such as movement and the interface. There were also some user requirements that were not supplemented by functional requirements, so we added these where necessary. Additionally, we included the new requirements for assessment 2, as these were not available in the original project brief. The specific requirements that we added or edited are listed below, along with a brief explanation:

1. **Removed UR_CRASHING.**

   Removed UR_CRASHING from the user requirements table as it is a non-functional requirement (characterises the operation of the system rather than the behaviours that the user sees). The same criteria are covered by NFR_UPTIME.

2. **Changed UR_SCALABLE to FR_SCALABLE.**

   This is a functional requirement rather than a user requirement because it describes the behaviour of the system rather than expressing a need of the user - it has been copied into the functional requirements table and linked to UR_INTERFACE.

3. **Changed UR_SCORE_LEADERBOARD to reflect the updated requirements.**

   The team already had a definition for UR_SCORE_LEADERBOARD which originated from their assessment 1 client interview. We updated it to match the constraints of the new assessment 2 requirement, in line with the updated project brief.

4. **Added UR_ACHIEVEMENTS and FR_ACHIEVMENT_TRACKER.**

   The original requirements included no mention of achievements as they were not required for assessment 1. We added these requirements and their specifications for assessment 2, in line with the updated project brief.

5. **Updated non-functional requirements.**

   New non-functional requirements were added to cover the performance, reliability and efficiency of the updated system, which will be larger in size and require adapted fit criteria to constrain the finished game.

6. **Added requirement for multi-OS support.**

   The original requirements contained no information about which operating systems the game would support, so we added NFR_COMPATIBILITY covering support for Linux, MacOS and Windows.

7. **Edited UR_TUTORIAL and UR_INTRO.**

   The descriptions of these requirements originally seemed like duplicates as they involved very similar scopes (both being about an ‘introduction’ screen to welcome the player to the game). We changed the wording to focus UR_INTRO on describing the context and story of the game, while UR_TUTORIAL covers the controls and instructions for playing the game.

2.2 Updates to 'Architecture'

Original Document: Arch1.pdf **Updated Document:** updated-Arch1.pdf

Although our version of the game inherits the code from the original team, we spent the first few weeks of the project refactoring it to our own specifications, and adding the new classes we would need for assessment 2.

We chose not to update the CRC cards or sequence diagrams for the code because these were used in the earlier stages of planning the game, and were not relevant to our plans or implementation. The class diagrams presented lacked a significant amount of detail and did not show the extent of existing relationships. Since the overall structure of the code changed significantly, we created updated class diagrams to show the final architecture of the game. These can be found under the *'Architecture'* tab of the website.

The steps taken to update the document, along with the specific changes made, are as follows:

1. **Created an updated final class diagram for the inherited code.**

   This was done during the planning process of the project, and helped us to understand the structure of the code at the point we inherited it. To improve on the original class diagram, we used IntelliJ's built-in diagrams tool to help create the diagrams, as this gives a well-rounded description of the classes without leaving room for human error. We manually adjusted the diagram, and presented it both on the website (*'Architecture, Fig. 1'*) and in the updated-Arch1.pdf document. Some of the key changes we made were:

   - Added more associations between existing classes.
   - Re-defined the subclasses of GameObject as they were vague.
   - Renamed 'Anim' class with the full name 'Animation' to improve clarity.
   - Removed any getter and setter methods as these do not add to the definition of the class.

2. **Created a new final class diagram for our finished product.**

   As the developed game contains new classes and relationships, it was necessary to update the original class diagram to reflect the new structure. We did this using IntelliJ's built-in 'Diagrams' tool, which helped us to capture an exact overview of the final class structure and allowed us to edit the diagram to our needs. We present three separate class diagrams for clarity, which can be viewed on the website (*'Architecture, Fig. 2-5'*) and in the updated document. These are as follows:

   - Fig. 2: Screens package, including HesHustle and Server.
   - Fig. 3: Utils package, including HesHustle and Server.
   - Fig. 4: Objects package, including HesHustle and Server.
   - Fig. 5: Final system overview.

   A description of each diagram can be found in the updated document.

The final system overview diagram is shown on the website, although it may be challenging to read. To improve readability, we removed the methods and properties of each class to show only the class name and its relationships.

The full set of methods and properties are included in these diagrams, mitigating the need for CRC diagrams or similar.

3. **Added discussion on our list of languages and tools used.**

   The original document contained little to no discussion or justification for the languages and tools used in their implementation of the project. We have added a short paragraph to the document, detailing why the tools that Team 21 used were appropriate for the project, and adding justification for the tools that we have used in our own work.

2.3 Updates to ‘Method Selection and Planning’

Original Document: Plan1.pdf **Updated Document:** updated-Plan1.pdf

Changes made:

1. **Updated Team Structure**
   We reviewed the roles that we assigned to our team members at the beginning of the project, and reassigned them to mirror team 21’s structure. By restructuring our team to match their methods and planning, we were able to seamlessly integrate their workflow into our team, such that each member was able to continue the work of their counterpart. Their Gantt charts and planning could then be compared with our plans for assessment 2, to see which team members should be responsible for different tasks.

   The choice of team structure is also justified in this section, as the original team had minimal explanation or evidence for their choices.

2. **Added engineering method for assessment 2.**
   The original team did not discuss their engineering method, so we added a section describing our agile approach to software engineering for assessment 2 and how this benefitted us, both as a team and in terms of the quality of our deliverables.

3. **Added weekly snapshots for weeks 7-13(?).**
   As a continuation of the existing weekly snapshots, we added our own for weeks 7-13 and uploaded the corresponding Gantt charts to the updated website. This helped us plan for each week and visualise the time scale in which different parts of the project would be completed.

There were no further changes that we made to this document. This is because the majority of the report describes the methods and approaches that Team 21 took for assessment 1, and they don’t apply to our team for this assessment.

Due to the fact that we are using the same software for collaboration (Google Drive, GitHub, IntelliJ) and coding (LibGDX, Gradle, Tiled) as the original team, this part of the documentation does not need updating. This is beneficial to us because we should encounter fewer problems with integrating the existing assets and code.

Additionally, we took the PlantUML code mentioned in this document and used it as a basis for our own planning. The Gantt charts and weekly snapshots helped us to consider the time frame in which to plan and implement assessment 2.

2.4 Updates to ‘Risk Assessment and Mitigation’

Original Document: Risk1.pdf **Updated Document:** updated-Risk1.pdf

Changes Made:

Changes were made to multiple Risks as the mitigation explanations were not suitable enough. The owner names were updated to ensure that these risks are now managed by our new group members.

The Introduction and description of the risk management process was also introduced in order to more accurately reflect our group's expectations and understanding of the risk assessment.

Risks Added:

Risk ID	Description	Justification for addition
R15	Lack of comments in the inherited code	Upon receiving the new code there is a likelihood that code is poorly commented on and therefore would need someone to oversee the addition of Comments.
R16	Difficulty in understanding the architecture and design patterns used in the existing code	The existing code may utilise complex or unfamiliar architecture and design patterns. This could lead to misunderstandings and incorrect implementations, potentially increasing the time and effort required for code integration and modification.
R17	Integration challenges arise when new features need to interact with existing code	If the architecture of the existing code is not well-understood or documented, integration of new features can become challenging. This could lead to bugs, increased development time, and potential failure of the new features to work as expected.
R18	Compatibility issues with legacy dependencies and libraries used in the inherited code	The inherited code may rely on outdated or legacy dependencies and libraries. These could pose compatibility issues when integrating with newer technologies or systems. Therefore, it is crucial to identify and update these dependencies as early as possible in the project lifecycle.
R19	Unforeseen legal or intellectual property issues associated with the inherited code	Code and assets may have legal or intellectual property constraints that were not initially apparent. These could lead to legal disputes, project delays, or even the need to redesign certain aspects of the project.
R20	Absence of comprehensive test coverage for the inherited codebase, leading to uncertainty in the stability and reliability of new features	There may be a lack of comprehensive test coverage, which can lead to uncertainty about the stability and reliability of new features. This could result in undetected bugs, causing potential system failures or incorrect functionality.
R21	Risk of introducing unintended gameplay mechanics or breaking existing game features during the integration of new code	Integrating new code into an existing game can inadvertently alter gameplay mechanics or disrupt existing features. This could degrade the player experience or require extensive debugging.

References:

[1] I. Sommerville, “Requirements engineering”, in *Software engineering*, 10th ed. Boston, Mass. Amsterdam Cape Town Pearson Education Limited, 2016,

[2] "IEEE Recommended Practice for Software Requirements Specifications," in IEEE Std 830-1998 , vol., no., pp.1-40, 20 Oct. 1998, doi: 10.1109/IEEESTD.1998.88286.