Change Report

ENG1 Team 4

Isabella Miles
Maciek Racis
Sally Finnon
Sophia Eaves
Wrijurekh Mukherjee
Xavi Murtagh Molina

Introduction:

As we made changes to the other team's Assessment 1 deliverables, our team had to prioritize carefully. Primarily, we aimed to make as few changes as possible while keeping the documentation up-to-date – making massive, unnecessary overhauls would limit our ability to work on other parts of the project. Part of the reason we selected Team 9 for the second half of the Assessment was their overall good deliverable quality. By limiting the amount of changes that were made, we made the most of the choice we had made. Most of the edits were for one of three purposes: to bring their documentation in-line with our previous methodology, to update based on new expectations and requirements, or to fix factual errors and inconsistencies. This list was, again, made to limit the edits made.

The process for changes mirrors the one we used during the initial creation of deliverables. Each item was assigned to the people best suited for the task – in this case, whoever had written our version of the original deliverable – to ensure quality updates were made. All Team 9's deliverables were shared with all those people, to be referenced as needed. The different writers then worked with each other, sharing their progress at group meetings, as they had done while producing the original documents. We took extra care to ensure that naming conventions were shared between documents, in a way that aligned with both our method and the previous team's as closely as possible.

After making changes, the various team members then wrote their change report and shared it with the group. This allowed us to collectively review the changes made, as well as allowing the member in charge of the change document, El, to collate and reformat them into the single Change2 document. Noticeable omissions from those sections were also corrected, as well as a few sections being edited to increase clarity and cohesion throughout.

As with all of our deliverables, our team primarily made use of Google Docs to share documents and collaborate on the provided documents. This choice is discussed in more detail through the Methods Selection and Planning section of our deliverables.

In order to maintain consistency and structure during the transition, we employed similar strategies in regards to changing our inherited code. The programming team worked closely with both the Requirements and Architecture team, ensuring all diagrams were up-to-date and all requirements were those that could realistically be achieved within our time. Communication with the Architecture team, in particular, ensured that code remained streamlined and clearly structured, re-using as much of both our code and the other team's code as possible to avoid wasted time. We also made use of the strategies outlined in the Continuous Integration deliverable to ensure smooth collaboration between these disparate teams – an incredibly helpful resource when editing an unfamiliar coder's programme.

Requirements:

While the design of the table in its various colours was aesthetically pleasing and made it easier to understand, the content of the tables needed an overhaul. This would allow our team to work on/understand it better, add new requirements easily and let others who may want to take on and edit our version understand it better while keeping the ideas of the original project intact. The original document (the link to which can be found at the bottom of this section as [1]), the changes are in [2] and were done so for the following reasons.

First we began with the User Requirements which we changed completely. Lots of the User requirements they had written down were a better fit for the functional requirements. If you take a look at their requirements [1] apart from the first 4, they talk about what the game should be. Because of this we found a lot of repeated information, irrelevant information and due to the lack of abstraction, we found the User requirements quite hard to digest. To fix this we moved some of the text to the other requirements for example we moved the relevant text from **UR_BUILDING_VARIETY**[1] to **FR_BUILDING_TYPE**[2], deleted a lot of repeated text such as the obstacles in **UR_CAMPUS_CREATION[1]** which was already in **FR_OBSTACLES** [1]. After this we tried to keep the User requirements simple and solely directed towards the user.

The Functional Requirements were far better written and thought out and as a result we kept a few of them such as **FR_TIME_LIMIT**, **FR_MAP**, **FR_BUILDING**. As in the User Requirements, we did find a fair bit of repeated text which we removed such as details about the effects of the difficulty which were repeated in others such as in **FR_BUILDING[1]**. There were a couple of requirements that were moved to other sections or removed entirely. **FR_USER_INTERFACE**[1] became **NFR_USER_INTERFACE**[2] and **FR_SATISFACTION_BUILDINGS[1]** was removed. What we spent most of the time doing in this section was adding new Requirements that we felt were missed out by the original team e.g. **FR_SCORE** [2], Requirements that were added in the product brief e.g. **FR_LEADERBOARD** [2], and Requirements we thought would add to our system such as **FR_SETTINGS[2]**.

The Non-Functional Requirements needed changes in the description due to repetition in the Fit Criteria such as **NFR_PERFORMANCE** [1]. A couple like **NFR_CODE_MODULARITY** [1] were scrapped altogether as they weren't relevant and then we added a couple for the same reasons we talked about in the Functional requirements. E.g. **NFR_ZOOM[2]**, **NFR_VOLUME** [2].

We kept the criteria requirements as we liked them and added **CR_FILE**[2] as mentioned in the product brief.

Due to the nature of the project, if we changed all the ID's, we would have to change all the references to them throughout. Despite this, we thought that in some cases that they were too long e.g. **NFR_INTERACTIVE_ELEMENTS_REACTION** [1] or did not reflect what the description originally or currently contained, e.g. **UR_BUILDING_VARIETY**[1] to **UR_BUILDING**[2]. Due to some being moved between requirement sections, it was inevitable that we did have to change some.

There were some ideas in the requirements that we decided not to pursue such as the option to upgrade buildings in **UR_BUILDING_VARIETY[1]** and so we removed them.

Based on the feedback from the shareholders after Assessment 1 we made sure that we kept the requirements portrait for readability and the fit criteria which we didn't do in our last requirements but agreed that it was better to have them this time round.

[1] - https://github.com/JD760/Team9-UniSim/blob/main/docs/deliverables/Req1.pdf
[2] - [insert pdf requirements 2]

Architecture:

We implemented various new features, as well as changed the implementation of a number of existing features, in the game we adopted for part two of the project. Before doing this, however, we edited the architectural diagrams we inherited to reflect these changes.

The diagram we made the most changes to was the class diagram. Two of the largest changes were adding Event and EventManager classes, as well as Achievement and AchievementManager classes, to fulfill the newly added requirements in the second phase of the project, such as **FR_LEADERBOARD** and **FR_ACHIEVEMENTS**.

Similarly in line with requirements, we added a Reputation class to handle the university's reputation, or student satisfaction. This is in line with **FR_SCORE**. Additionally, although this feature is not a requirement, we did not want the player to be able to place buildings and increase student satisfaction without any limits; this would make for a tediously simple game. Therefore, we have included a Money class in our design; the player would accumulate money over time, and then be required to use this money to pay for the buildings they want to place.

This money feature would also add an extra layer of interest to events; players would be able to pay to mitigate the effects of negative events, or to make the most of positive events. That is, if they feel they have the finances to be able to do so.

Moreover, once again to satisfy a range of requirements, we included three new screens: AchievementsScreen to satisfy **FR_ACHIEVEMENT_SCREEN**, LeaderboardScreen in line with **FR_LEADERBOARD**, and InstructionsScreen to satisfy **UR_HOW_TO_PLAY**. In addition to this, we changed GameOverMenu to GameOverScreen as we felt that a separate game over screen would look better than an overlay on the main game screen.

One class we altered the implementation of is the Timer class. We changed the design of this feature to what we had already implemented in our game in the first stage of the project, which will allow us to show the player both an in-game date, and how much time has passed in real life.

Finally, we combined the three separate diagrams that made up the original class diagram into one, as we felt this looked cleaner and reduced repetition of certain classes and interfaces.

We also reworked the state diagram to reflect these changes in implementation. This involved adding several new states for screens: a leaderboard screen, an achievements screen, and an instructions screen for usability and accessibility. These are in line with both the screens in our class diagram and the requirements discussed in relation to those.

Although not adding a state, we also changed GameOverMenu to GameOverScreen to demonstrate that we would be adding an extra game over screen rather than just having a menu overlaid. Additionally, we added an arrow from GameOverScreen to the final state to demonstrate that the player will be able to quit and close the application from the GameOverScreen.

One significant alteration we made was adding a PausedState, which is entered from StartMenu when the player begins the game, as well as at any point from the Gameplay state, and when exited goes to the Gameplay state. Originally, the ability to pause the game was only handled in the Timer class and would only pause the timer itself. This meant that a player would be able to place buildings on the map still whilst the game was paused. We created this PausedState to reflect the wider effects on the whole game, such as being unable to place buildings, whilst the game is paused.

One last change we made to the state diagram was to add descriptions of how the system would go from one state to another, to add more detail and make the diagram more explanatory.

With regards to the sequence diagrams, we replaced the panMap diagram with a diagram that shows how this sequence of actions would work when the user is using arrows or WASD to move around the map, rather than clicking and dragging it. We changed this implementation as we felt it was more intuitive to use arrows or WASD.

Other than this, we are keeping the behaviours described by the sequence diagrams the same, so we did not revise any of the other diagrams.

All the diagrams we modified, as well as interim versions of them, can be viewed on the website.

Method Selection and Planning:

Engineering methodology

With respect to the engineering methodology, for part one, both team nine and ourselves decided to use an agile approach. As a group we felt that this worked for us successfully and we are now familiar with this approach and its documentation. This is one reason why we have again chosen to use an agile methodology for part two of the project.

As pointed out by team nine in their methods and selection report, agile is particularly useful for short and collaborative projects. The regular checkpoints keep us on track, motivated and less distracted by inessential details. Given the similar time constraints for part two, agile is also applicable. Additionally, this allows for effective collaboration as deliverables can be fairly allocated to team members based on skill sets. Fair and targeted work allocation will also mitigate the risk of burn out that agile can cause, as mentioned by group nine. Furthermore, factors such as time constraints and feedback made the development process in part one changeable. Consequently, we really benefited from the adaptive nature of an agile project.

Source Control

For the second part of our game, we are choosing to use GitHub for version control and code sharing. This is not a change from team nine's report. Our group also used GitHub in part one and it is the industry standard. Consequently, the whole team is familiar with it and plenty of documentation is available if any problems are encountered. As pointed out by team nine, GitHub is great for allowing multiple team members to collaborate on a project. Code can be pulled, modified, and pushed into a side branch to be checked before merging back to the main branch. Allowing team members to work on different sections in parallel, without affecting each other. The verification process before merging also simplifies the debugging process further down the line. As a team, we felt this worked successfully for part one and we agree with team nines report on why they used GitHub, so no change is necessary.

Development Environment

Team nine chose to use both VScode and IntelliJ as their IDE. The reasoning given was that both support Java compilation and debugging via GitHub, so group members can use whichever they prefer. Although we agree that both are good examples of Java compatible IDEs, we do not feel it necessary to use more than one. Due to VScode being used in other modules at the university and by us in part one, all group members are familiar with it. Which is why we have chosen VScode over IntelliJ for our project.

Communication Tools

Since effective lines of communication between the team were established in part one, we did not feel the need to make changes to comply with team nine's chosen communication tools. To save time and make things easier, we will continue to communicate informally in our WhatsApp group chat. This is a change from team nine's use of Discord as not everyone in our team is very familiar with it.

However, we are sticking with team nine's use of GitHub and GoogleDocs for file sharing and more formal communication. We also chose to use these tools for part one. We encountered no major issues and the whole group is happy with these tools, so continuing to use them makes the most sense

Implementation Tools

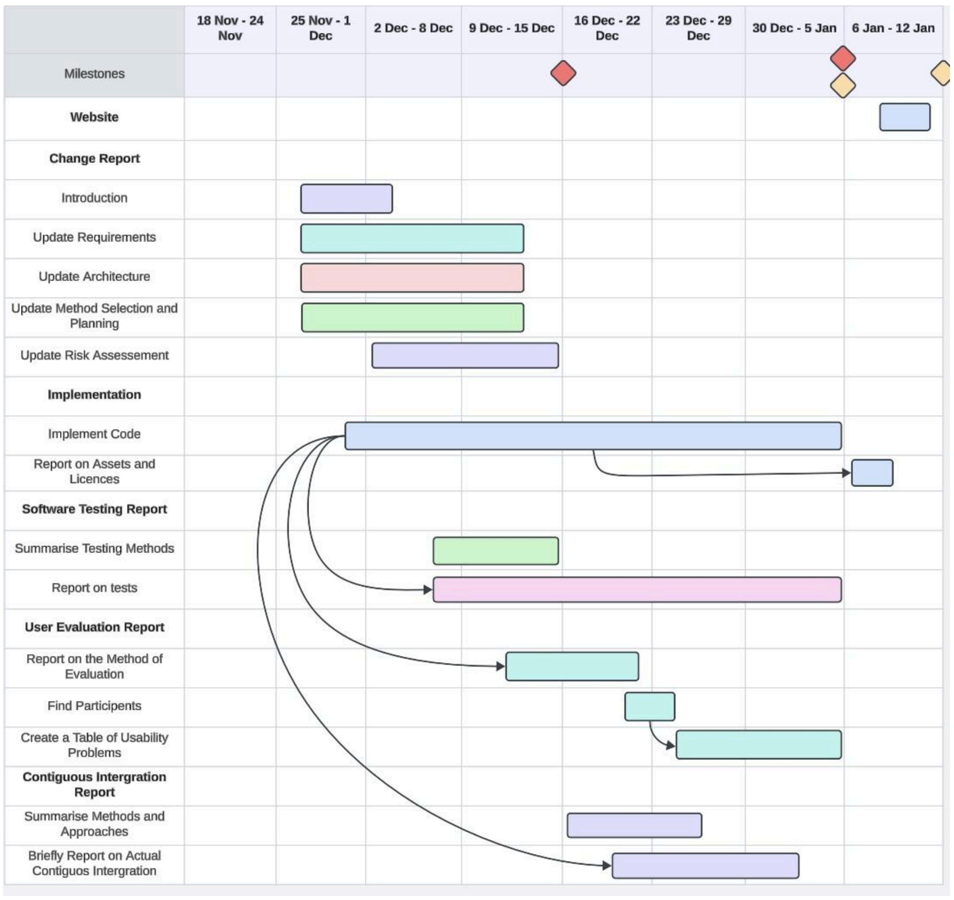
Both teams used LibGDX successfully. It is commonly used, well documented and Apache 2 licenced. Consequently, no changes will be made to this.

Work Allocation

We have taken the same approach to allocating work as team nine outlined in their report. That is to say, based on personal interests and skills. For example, Chek and Wri both have coding experience and prefer programming to writing heavy sections, such as a long report. A change from team nine however is that we are also influenced by our experience in part one. For example, each section of the change report is allocated to the person who was incharge of that section for part one. This saves time as other team members will not have to familiarise themselves with the content. Additionally, due to the software testing report being too big for one person, Sally and Sophia will collaborate on it. This based on them having successfully worked together on part one.

Work Breakdown

This section of the report outlines the work breakdown, structure and timelines of the project. Due to part two having a new set of deliverables all of this will be a change from team nines report.



Priorities (as outlined on the Initial Gantt chart above)

Implementation has been given the highest priority. This is because tests are to be carried out on new code. Thus the user evaluation report, software testing report, and contiguous integration report have been scheduled further down the line. As they are dependent on implementation. This gives the coding team an opportunity to get significant new sections of the implementation down before we begin.

The change report has been given the same priority as Implementation. This is for two reasons. First being that delaying the other reports until significant implementation has been completed leaves Sally, Xavi, Sophia and El free. So to maximise time efficiency, they will first complete the change report. Secondly, The change report provides lots of documents that are vital for the project. For instance, new architecture diagrams, the work breakdown, a risk assessment and new requirements.

This is our initial plan, although some changes are likely to occur. Deliverables are likely to take more or less time than we had expected. Plus there may be external factors such as illness that will alter our timeline. So we will periodically review our plan/Gantt chart and make necessary changes to ensure that all points are met and everyone is on track. These updates will be posted on our website.

Risk Assessment and Mitigation:

As the risks involved in this project have not changed significantly due to the change of program, the edits made to the Risk Assessment and Mitigation deliverable were mostly small quality-of-life changes. Clarifications were made to certain risks to accommodate the fact we are undertaking this project from its midpoint, and new risk owners were assigned from our own group. Beyond this, the only change was the inclusion of a new explanation section, elaborating on the process of these assignments and changes.

First, risks were removed or edited to more accurately reflect the current situation for the project. **R7** and **R9** were removed from the original table; **R7** was deleted as it was related to initial team-building, and **R9** was removed as it was determined to be redundant with **R1**. We also removed the Mitigation sections from two risks, **R4** and **R6**, as they were also related to initial decision making from before we joined the project. This reason also resulted in changes to the text of **R5** and **R11's** plan section, while **R8** was changed to specify the need for clear communication within the group.

We felt these changes were needed to avoid unnecessary inclusions within the risk assessment – as the document is designed to be updated based on the current risk environment, it no longer needed risks related to the initial group's development of their team and practices. Our team was already fully developed, and our code base was already written.

The other major update to the risk assessment was the change of risk ownership. We replaced the members of Team 9 included in the table with members of our own group, who would take ownership and responsibility for the risks created by the other team. Each member of the group was assigned two risks, and the table was updated to reflect this. Due to the necessity of this change – a group no longer working on this project cannot be responsible for managing the risks involved with it – no explanation is needed for the update.