Change Report

ENG1 Team 9

Jacob Dicken
Bertie Cartwright
William Croft
James Dovener
Henry Chan

Introduction

After inheriting our new project from another team we had to review their deliverables and documentation and test their code before making any of our own changes.

We firstly looked through their risk assessment deliverable and made sure to note any risks we felt were missing and others we felt no longer had any affect. Secondly we looked through the requirements document and compared it to the product brief and made sure that all parts of the brief were covered as users requirements and that there were functional and nonfunctional requirements that would lead to these users requirements to be met. If this was not the case this would be noted down to be changed. Next we looked through the architecture which helped us understand the game but realised that this would need changing as the code itself changed and instead made a rule that if we are changing the code in a way which alters architecture of the code in a significant way we should note this down and update the architecture accordingly. We also analysed the code and made note of any changes we would need to make so that it could match the rough idea of the new requirements. After looking at the requirements and the code, we then had a rough idea of what we needed to do and then went through the planning document noting down every major task, including changes to the deliverables, that needed to be added to the existing plan.
These notes that we made would be added to our weekly plans document.

To keep track of changes made to the code and documentation we used github which records changes as commits which record all the information about each change, the time it was made and the relative order of changes made. This not only allows us to keep track of the changes made but also allows us to revert to a previous version if we feel that a change is wrong or not needed. Proposed changes were recorded using GitHub Issues, which allowed us to assign team members to work on each issue and comment on issues to keep everyone in the loop and to propose further changes.
To keep track of changes made to the deliverables we used google docs which has an inbuilt change management - this tracks each change in the document's revision history and allows for reversion if needed. Google docs also allows for change suggestions which also helps keep track of proposed changes.

The changes made to the code and documentation would use github's pull system where after a team member made a change to either the code or its documentation they could then send a pull request to merge it with the main branch of the code. This pull request could be reviewed by other team members who would check if the code/documentation was correct, wouldn't conflict with other changes and corresponds with what we planned to change.
The changes made to the deliverables would be reviewed in our bi-weekly meetings so that we could make sure the group agrees that the change is correct and corresponds to what we planned to change.

After reviewing all the changes that had been made we would then check to see if all planned changes had been made and remove those that had been made.
Finally in these meetings we would again analyse all our deliverables and code and plan for any new changes we felt we needed to make.

Requirements

URL of the original document: https://jd760.github.io/deliverables/assessment1/Req1.pdf
URL of the changed document: https://jd760.github.io/deliverables/assessment2/Req2.pdf

Requirements Elicitation & Negotiation

The first thing that we changed in the requirements document was adding a bit about our research into sensible system requirements for the game. This wasn't previously included in the requirement document and would help us have a target specification that our game should run on that could be referenced in the requirements tables.

User Requirements

In the user requirements table we started by changing some of the preexisting requirements descriptions. Firstly **UR_STATIC_MAP** renamed **UR_MAP** to more closely represent its description, was changed so that it no longer limited us to having the map be static as we felt that to facilitate events the map may end up being changed, for example a snow event may change the map to a snowy map. We also added the line about having obstacles blocking the placement of buildings as this is a core part of the product brief which was missing from the user requirements.
In the **UR_GAME_PERSPECTIVE** requirement only a simple change was needed where we changed the requirement that the game was top-down to it instead being isometric as we felt this better matched how the game was and is meant to be.
In **UR_BUILDING_PLACEMENT** we changed the requirement of at least 1 recreational building to at least 2 as this is what is specified in the product brief.
In the **UR_GAME_CLOCK** requirement we added a line stating that the timer should start from 5 minutes as this wasn't explicitly stated and we felt it was important that it was as this was a key part of the product brief.
The final user requirement that was changed was **UR_MINIMUM_HARDWARE** which we changed to instead require our game to run comfortably on a machine with the minimum specification which we researched as prior to this there were no defined minimum hardware requirements that we could use for other requirements and testing.

After changing some of the preexisting requirements we then added some new ones. These new ones are as follows: **UR_EVENTS**, **UR_LEADERBOARD**, **UR_NAME_INPUT**, **UR_SATISFACTION**, **UR_ACHIEVEMENTS**, **UR_USABILITY** and **UR_EXTENDABILITY**.
UR_EVENTS, **UR_LEADERBOARD**, **UR_SATISFACTION** and **UR_ACHIEVEMENTS** were added as they are requirements linking to core parts of the product brief which didn't already have requirements which were events and reacting to events , a leaderboard, student satisfaction and achievements.
We added **UR_NAME_INPUT** as the game needed a way to obtain the players name at the end of the game so that it could be displayed on the leaderboard if needed.
UR_USABILITY was added as when testing the game we were given we found the game to be functional but hard to control and not very user friendly which would definitely hurt the enjoyability of our game. So we added a requirement to make sure that we made the game as user friendly as possible.

Finally **UR_EXTENDABILITY** was added as a low priority requirement as we felt that this would not only help debug and add features but also help future development teams to do the same which we believed to be useful and semi-important to our stakeholders.

Functional Requirements

In the functional requirements table we again started by editing some of the preexisting requirements which began with a small change to **FR_MAP** where the wording was changed from “static map” to “visual map” to better match the new changed **UR_MAP** requirement.
To go along with the changed **UR_MAP** requirement we also added the **FR_OBSTACLES** which together with **FR_MAP** will satisfy the requirements of **UR_MAP**.
Next we split **FR_GAME_CONTROL** into **FR_ISOMETRIC_CAMERA** and **FR_CAMERA_CONTROL** as we had moved the information about controlling the camera from **UR_GAME_PERSPECTIVE** as we felt it linked more into **UR_USABILITY** so we did the same here where **FR_ISOMETRIC_CAMERA** is a functional requirement that satisfied **UR_GAME_PERSPECTIVE**. And **FR_CAMERA_CONTROL**, which now refers to the game being controlled using a mouse which was found to be much easier than using the keyboard to move around the map, contributes to **UR_USABILITY**.
FR_PLACE_BUILDING was split and expanded into multiple function requirements as we felt the original **FR_PLACE_BUILDING** didn't contain all the necessary functional requirements to implement **UR_BUILDING_PLACEMENT**. These new requirements are: **FR_BUILDING_VARIETY**, **FR_BUILDING_SELECT** and, **FR_BUILDING_PLACEMENT** where **FR_BUILDING_VARIETY** implements the building types part of **UR_BUILDING_PLACEMENT**. **FR_BUILDING_SELECT** also implements part of **UR_BUILDING_MANAGER** as it is to do with selecting buildings from the building menu which is also important in the functionality of placing buildings.
FR_BUILDING_PLACEMENT is practically the same as the old **FR_PLACE_BUILDING**.
FR_GAME_PAUSE was kept the same but a new functional requirement **FR_GAME_PAUSE_EFFECT** was added which defines what should actually happen when the game is paused which is important to **UR_PAUSE_GAME**’s functionality.
FR_GAME_END was kept the same but a 2 new functional requirements were added which were **FR_TIMER_COUNTDOWN**, which was important to add as it specifies that the timer should count down in real time, and **FR_TIMER_START**, which simply specifies that the timer should begin a 5 minutes and start counting down when the play button is pressed which again is key to implementing **UR_GAME_CLOCK**.
Then we created 3 new function requirements for satisfaction which were: **FR_SATISFACTION_COUNTER**, **FR_SATISCATION_BUILDING_DISTANCE**, and **FR_BUILDING_RATIOS** which are all needed to implement **UR_SATISFACTON** as this user requirement didn't previously exist and therefore had no functional requirements for it. This reasoning also applies to **FR_EVENT_TYPES**, and **FR_EVENT_FREQUENCY** which were added for **UR_EVENTS** as well as **FR_ACHIEVEMENT_EARN**, **FR_ACHIEVEMENT_CRITERIA**, **FR_ACHIEVEMENT_NOTIFICATION**, and **FR_ACHIEVEMENT_REWARDS** for **UR_ACHIEVEMENTS** and **FR_LEADERBOARD**, and **FR_ADD_TO_BOARD**, for **UR_LEADER_BOARD** with **FR_NAME_INPUT** being a functional requirement for both **UR_LEADER_BOARD** and **UR_NAME_INPUT**.
FR_USER_INTERFACE is a new functional requirement for **UR_USABILITY** which refers to how the UI should be visually implemented which was added as this is an important factor in making the game as useable as possible.

FR_INTERACTIVE_ELEMENTS is another new functional requirement for **UR_USABILITY** which refers to how interactive elements like buttons should behave after being used by the user which is very important in making the game as user friendly as possible.
Lastly **FR_MINIMUM_HARDWARE** was removed as “Optimise the game for standard laptops or desktops” is not a functional requirement of the game and rather a task our team should complete.

Non-functional Requirements

For the Non-functional requirements table we firstly added a new requirement **NFR_GAME_ENGAGEMENT** which is to do with how much the user enjoys the game and was added to satisfy the new second piece of criteria on **UR_GAME_ENJOYABILITY**.
Next on **NFR_GAME_DIFFICULTY** we added **UR_USABILITY** as one of the user requirements it contributed to as we thought there was some overlap there and we also changed the fit criteria of “Player completion rate > 50%” to a more meaningful statement about satisfaction score as we felt that completion rate didn't really have a mean anything in our game as the game would start and end irrespective of what the user did while the game was running.
NFR_MINIMUM_HARDWARE was changed to reference the minimum requirements we set for our game which didn't previously exist. The fit criteria was changed to 58 fps from 30 with a 60 second average as we felt 30 fps was too low for such a simple game and our target should instead be 60 fps.
NFR_INTERACTIVE_ELEMENTS was added so that we could have a fit criteria for how our buttons should work within **UR_USABILITY**.
NFR_OPERABILITY was added as we felt it was important to determine how comfortable new players were with the game. The time limit was set to 2 minutes as we expect players to spend only a couple minutes to learn the controls and become comfortable with the simple concept of the game.
NFR_DOCUMENTATION and **NFR_CODE_MODULARITY** were added as needed ways to measure how useful and understandable our documentation was as well as how easy our code was to extend so that we could satisfy **UR_EXTENDABILITY**.

Architecture
URL of the original: https://jd760.github.io/deliverables/assessment1/Arch1.pdf
URL of the changed document: https://jd760.github.io/deliverables/assessment2/Arch2.pdf

The tools and processes used by the previous team to design the architecture were very similar to our own, largely relying on PlantUML for the diagrams. We did not use Graphviz or Draw.io in our architecture design process or any of the remaining diagrams so references to these were removed.

The class diagrams for GridPosition, GridArea and CollisionDetection are unchanged due to them being suitable for the new requirements with no modification.

We created a high-level structure diagram showing the state of the architecture before the new requirements were added. This structure was then changed dramatically in order to accommodate the new requirements. Mainly this involved dividing up the responsibilities of the main components in the original architecture into many smaller classes responsible for specific requirements such as the BuildingMenu class. An updated diagram showing the final structure has been added to show the evolution of the architecture due to the additional requirements.

The original document goes into little detail on the methods used to devise the architecture for the project, so we had to start our development of the architecture from the point at which we inherited the project.

Many of the diagrams and descriptions for various parts of the project needed to be adjusted as the architecture has changed significantly from inheriting the project, due to the introduction of additional requirements. The original versions of all diagrams have been preserved on the website on the diagrams page.

We added several new class diagrams to address new requirements:
- StudentSatisfaction
- Notifications
- Leaderboard
- Achievements
- Events

We also added a section at the start of the document describing and justifying the architectural style as this was missing from the original report we inherited.

Sequence & State Diagrams
While the existing sequence and state diagrams are still relevant to our final architecture, we decided to create more detailed diagrams that better represent the actual processes, particularly with the sequence diagram. The initial diagrams have been placed on the website and replaced with new versions and additional sequence diagrams.

Method selection and planning
URL of the original: https://jd760.github.io/deliverables/assessment1/Plan1.pdf
URL of the changed document: https://jd760.github.io/deliverables/assessment2/Plan2.pdf

Due to the nature of this deliverable, a large portion of the document was rewritten rather than directly changed, as sections such as team organisation have changed completely as a new team is now working on the project.

Software Engineering Methods and Development/Collaboration Tools
As we are inheriting a project from another team, it made sense for our team to extend the previous systems rather than reinventing the wheel. For example, we initially considered changing the build system to Gradle, as we are more familiar with this system and our research suggests LibGDX is most commonly used with Gradle. However, we decided to keep the previous team's build system as migrating from Maven appears to be a larger task than simply extending the existing solution.

The majority of the previous team's choices for development tools aligned with our own choices in the previous assessment, notably their use of GitHub for version control and use of feature-driven development.

One update to this section was to clarify the development environments used by our team members and choice of collaboration software.

Team Organisation
Before starting work on the inherited project, we first created a plan and task allocation, with a focus on team organisation for this assessment. From assessment 1 we had an awareness of which team members perform best at certain tasks, so used this to inform our changed team organisation system. This follows a similar system to our own team's structure in assessment 1, with slightly changed areas of focus. Our system was notably different to the previous team's organisation structure, which split the team into smaller working groups that tried to work independently on a subset of tasks each.

Planning
The only updates needed to this section were to add weekly snapshots of our plan since inheriting the project. No other changes to this section are justified as the plan used by the previous team has no bearing on our development process.

Risk assessment and mitigation
URL of the original: https://jd760.github.io/deliverables/assessment1/Risk1.pdf
URL of the changed document: https://jd760.github.io/deliverables/assessment2/Risk2.pdf

The first thing that was changed about the risk assessment and mitigation document was the format of the risk register itself. Here we added an ID column which helps us keep track of and reference each risk if needed. We also added a type column for each risk as assigning a type to each risk helps us keep track of what part of the project each risk affects. The Risk column was changed to a description column as we felt the prior risk names were not descriptive enough leading us to not know what each risk actually was and its effects; this new column also contains the description from the former impact column which is now a severity column. The severity column follows the rating system of the likelihood column which will help us quickly discern the severity of each risk and if needed easily change them by changing the rating. Finally the mitigation column was renamed to Avoidance/Mitigation/Contingency Plan(s) as this column doesn't only contain ways to mitigate risks but the other two methods too. To help distinguish between the 3 methods of risk management when there were multiple for one risk, we split them each into their own sub row with a title of what method they were using. The last minor change to the risk register as a whole is that the severity and likelihood columns are now colour coded so that the information contained within those columns is even clearer to our team.

All these changes to the risk register are reflected in the Risk Management Process part of the document which has a paragraph at the bottom that describes what each column is for and what information they should contain. In the analysis stage we added a line about how we placed each risk into types and how we went about deciding on the likelihood and severity ratings. This was added so that we will know how the types and ratings should be decided by our group.

The format change of the risk register led to most of the contents of the risk register being changed to some extent. With firstly all owners of every risk being changed from the prior group members to our group members.
Then each of the preexisting risks were reformatted with "Miscommunication" now being "**R11**" in the new risk register. "Uneven Workload" is now "**R15**" and "Creative Differences" is now "**R16**" with both of these getting a severity rating of medium as we felt that this risk wouldn't be too severe but could start to slow down the pace of development if it persisted. "Overambition" became "**R13**" where we changed the likelihood to medium as we felt it that it was less likely that our team would drastically increase the scope of the project this late in as we all knew what to do and would make sure that the core requirements were implemented first; however the severity rating was set to high because if for whatever reason this was to occur it would severely hurt our project to not be able to implement all the core requirements in time. "Lack of Feedback" is now "**R5**" with its description referring to the requirements not corresponding to what the customer wants, and the mitigation of meeting frequently with the customer. The changed "**R5**" is now not specifically being about a lack of feedback but about how the requirements may not be correct which can be remedied by having lots of feedback from the customer. Finally for the preexisting risks "Bugs and other performance issues" is now "**R17**" which was given a medium severity as while some bugs may be very severe, most especially the ones that are missed during development should be mostly minor and hopefully easy to fix.

Next we added some new risks to the risk register that were either totally new risks we considered or risks that we had previously found and felt still applied to this new project. Firstly “**R1**” which is the risk that one or more of our team members are unable to participate was put as a high severity risk as if this was to occur it could lead to more pressure on fewer group members and eventually lead to lots of work being incomplete or missing. “**R2**” (Use of poor quality libraries) and “**R3**” (Code structure and readability reduce as the project progresses) were added as we felt these were risks that may be unlikely to occur but if they did could have an impact on our ability to add new features to and test our project. “**R4**” (Product is unable to be built/run on the required hardware or Operating Systems) was added as even though it is a low likelihood risk, if it was to occur our customer wouldn't be able to use the product meaning it may as well have met none of the requirements. “**R6**” which refers to the risk of the requirements being changed was added as while it was deemed to have a low likelihood, if it was to happen it could be very severe as it could lead to us not meeting all the requirements for the project. “**R7**” (Misalignment between member’s strength and role) was added as a low likelihood risk, as by now we mostly know the strengths of each team member, but with a medium severity as if this was to happen it could slow down the project. “**R8**” (Unclear or ambiguous requirements) we felt was important to add as we had to analyse the new requirement tables and also add the new requirements for project 2 into them so we had to make sure that these requirements were also clear and unambiguous. “**R9**” (Use of tools that are overcomplicated or only one team member understands) was added as a risk with a low likelihood and a medium severity as while we would now be familiar with the tools for our original project, the tools used by the other team in this new project may be unfamiliar which could slow down our development as we have to learn the new tools. “**R10**” referring to the risk of poor time management was added as we felt there would be highly severe consequences if we poorly managed our time e.g. we missed the deadline to hand the project in with all the requirements met. “**R12**” (Project schedule is not defined or understood) was added with a medium likelihood and severity as we would need to create a new project schedule for the new project which could be misunderstood leading to team members not knowing what pieces of work they need to complete. “**R14**” (Parts of the writeup are poorly formatted or hard to understand) was added as we inherited a whole new set of writeup/deliverables which we need to edit and maybe or end up being poorly formatted leading the information to be hard to understand. Finally “**R18**” (The project that we take over has poor libraries, coding practices, documentation or deliverables) was added as this risk heavily applies to this second project as we have just taken over another project and need to make sure that there are no prior faults within it that will harm the future development of the project.