UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Change Report

Group 20

Formerly Group 16

Group Members:

Group 16

Charlotte MacDonald
Hollie Shackley
Luis Benito
Kaustav Das
Sam Hartley
Owen Gilmore

Group 20

Leuay Dahhane
Max Irvine
Sam Butler
Flynn Gadsden
Jacob Wharton
Billy Moore

How we updated our deliverables

We kept track of every task that needed to be completed for the second assessment using a KAN board on Jira. This encompassed both work towards the new deliverables, code changes and changes to the assessment1 deliverables.Tasks were assigned to members, the status of each task was recorded. For example a given task would be assigned in the *To Do* section, move to *In Progress* once started, move to *Review* once complete, then moved to *Complete* once it has been reviewed by another team member and any feedback has been worked out. We tried to break down the work into fairly small tasks that could be completed within one week as we treated this as our “sprint” duration, meaning tasks would be assigned each week and ideally everyone would be able to complete at least one task in that time.

The lead for the change report was responsible for ensuring that someone was assigned to each deliverable. During the process of changing deliverables any notable changes were tracked in a separate document we called a change table. The change table had three columns: what change was made, where it was made and why the change was made. This allowed us to note down our changes whilst we worked in a shorthand but constructive way. In addition we made changes in a different colour where possible. We also made use of the change history of google docs to understand exactly what changes were made.This made it easier to write up the change report for each document, additionally we made sure that the change report was either written up by the same person who made the changes or in very close communication with them to ensure that changes are represented correctly.

For changes to the code, tasks on the KAN board were linked to the git branch. To make sure these changes within these branches were traceable and clear, we used atomic commits with clear concise explanations. We aimed to update docstrings and other comments as we went, we were weary of making changes without updating the related comments, as legacy comments that are incorrect may harm maintainability rather than help it.

Overall we tried to minimise the amount of changes made to the project we inherited, we wanted to ensure that any changes we made had a clear justifiable purpose. Most commonly changes were made to include assessment two concerns, for example adding gantt charts for assessment two and updating the requirements.

Requirements

Original Document: Req1.pdf
Updated Document: Req2.pdf
We updated the requirements to best reflect the additional requirements for the second assessment. We also made some minor modifications to the titles of the existing requirements to make them more usable. Whilst the IEEE Recommended practice for software requirements specifications [1] only specifies that all requirement names should be uniquely identifiable, as they were when we inherited the project. We felt some modifications would make the requirements more explicit and understandable.

The existing approach relied on specifying the broad area of the requirement and incrementing the title for each requirement in that area. This led to largely different requirements differing by one digit, for example NFR-DOCUMENTATION1 referred to the creation of an architecture plan with the associated diagrams and NFR-DOCUMENTATION2 referred to the commenting and documentation of code. To change this we switched instead to using titles that summarised the body of the requirement. For example FR-GAME-PLAY1 became FR-SLEEP-LOCATION, at a glance it can now be understood that the requirement relates to the provision of a location for the player to sleep and there is no longer a risk of confusion with FR-GAME-PLAY2, FR-GAME-PLAY3 and FR-GAME-PLAY4 now named FR-STUDY-LOCATION, FR-EATING-LOCATION and FR-RECREATION-LOCATIONS respectively.

Existing requirement title	**Updated title**
FR-GAME-PLAY1	FR-SLEEP-LOCATION
FR-GAME-PLAY2	FR-STUDY-LOCATION
FR-GAME-PLAY3	FR-EATING-LOCATION
FR-GAME-PLAY4	FR-RECREATION-LOCATIONS
FR-SLEEP1	FR-SLEEP
FR-SLEEP2	FR-ENFORCED-ENERGY
FR-ENERGY1	FR-ENERGY-DEPLETION
NFR-DOCUMENTATION1	NFR-ARCHITECTURE-PLAN
NFR-DOCUMENTATION2	NFR-CODE-DOCUMENTATION
NFR-RESILIENCE1	NFR-MAP-LOCATION-RESILIENCE
NFR-RESILIENCE2	NFR-START-UP-RESILIENCE
NFR-OPERABILITY1	NFR-NEW-USER-OPERABILITY
NFR-OPERABILITY2	NFR-SET-UP-OPERABILITY

NFR-OPERABILITY3	NFR-STAND-ALONE-OPERABILITY
NFR-ACCESSIBILITY1	NFR-COLOUR-BLIND-ACCESSIBLITY
NFR-ACCESSIBILITY2	NFR-SOUND-ACCESSIBILITY
NFR-USABILITY1	NFR-SIMPLE-ERROR-MESSAGES
NFR-USABILITY2	NFR-USABLE-GAME-INSTRUCTIONS
NFR-USABILITY3	NFR-UNIVERSITY-REPRESENTATION
NFR-TIMING1	NFR-ENFORCED-SLEEP
NFR-TIMING2	NFR-GAME-LENGTH
NFR-MAINTAINABILITY1	NFR-CODE-SIMPLICITY
NFR-MAINTAINABILITY2	NFR-CODE-MAINTAINABILITY

Additionally some requirements needed to be changed or added to bring the project in line with the updated brief. New requirements:

UR-ACHIEVEMENTS, UR-LEADERBOARD, UR-SCORING, FR-LEADERBOARD, FR-ACHIEVEMENTS, FR-SCORING

Updated requirements:

FR-STUDY-LOCATION, FR-EATING-LOCATION, FR-RECREATION-LOCATIONS

In order to meet the new requirement and in line with our initial conversations with the customer we decided to introduce a town map. This necessitated some additional functional and non-functional requirements and a change to the existing UR-WORLD requirement. Requirements added:

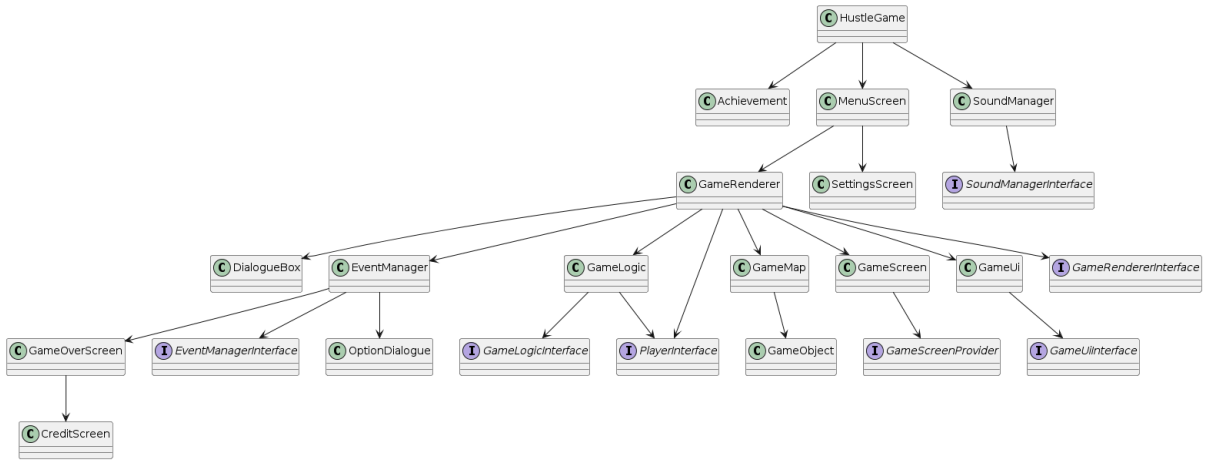
NFR-MOVEMENT-BETWEEN-MAPS-USABILITY, FR-MOVEMENT-BETWEEN-MAPS, FR-TOWN-MAP, FR-CAMPUS-MAP

Negative requirements are harder to verify as they are less specific and may have many possible outcomes [2]. To bring the project in accordance with this we changed FR-DEVICE to be rewritten as a positive requirement, in this case saying the game should work on windows rather than should raise an error if running on a non-windows machine.

We made small changes to the introduction to generalise over both assessments, for example changing the deadline to refer to both the first and second assessment deadlines as opposed to just the first.

Architecture

[Original Document: Arch1]
[Updated Document: Arch2]



Final UML class diagram^

For our architecture changes we Refactored overloaded classes and significantly improved our game's architecture, making it more modular, maintainable, and testable. This strategic approach laid out our brownfield foundation by giving us a clean scaliable codespace.

It allowed us to set up strong , focused continuous testing throughout which saved us time as we would encounter errors and bugs significantly quicker when implementing our new requirements , instead of us finding them once we have completed the product.

The architecture also improved scalability significantly, adding new features to the code was a lot simpler with the modularisation of code . We didn't have to worry about large classes with several responsibilities breaking down on small changes.

To conclude, the refactoring of our code and further implementations of interfaces allowed us to produce a modular, readable code with clear class uses.This set up saved us vast amounts of time in regards to testing,further implementing and debugging .

Method Selection And Planning

Original Document: Plan1.pdf
Updated Document: Plan2.pdf
We felt the *Outline and Justification of Software Engineering Methods* layed out a good approach to the project that was clearly used with success in the first assessment, as a result we chose to make no changes to this section and adopt this approach fully.

We made minor changes to the *Identification and Justification of Development and Collaboration Tools Used*, we made small changes to clear up that git and Github are different pieces of software. In addition we added the Jira KAN board that we have used as this wasn't one of the original tools used but we have found it to complement the agile approach we used well.

We updated the *Team Organisation* section of the deliverable to reflect the team organisation within the team, we largely adopted the approach of the previous team but updated it with the names and responsibilities of members of our team.We also made minor changes bourn from our experience in the first assessment for example we chose to further emphasise the importance of the bus factor as we had team members become unavailable in the first project and saw the impact of this first hand.

We switched to using a tabular approach to display responsibilities, these were previously included in the body text in the original document. We felt this was more clear at a glance and we were unconcerned about the extra space taken up as we are not limited by a page count.

We created a second work breakdown for the second assessment containing all tasks related to the second assessment specifically. Additionally we updated the deliverable and task tables to include the second assessment deliverables and tasks specifically.

We produced gantt charts for the second assessment. We discussed these in more detail than was provided in the original document, we decided this was appropriate as there were more significant changes between weeks in the gantt charts we produced than those produced by the original team. We also chose to include the diagrams in the document as we are no longer limited by page limits.

Risk assessment and mitigation

Original Document: Risk1
Updated Document: Risk2

Many of the existing risks proposed were still present in the brownfield project, we took apart the existing risk register and reassessed how probable and impactful they'll be in this part of the project. We also added many new risks, which were solely relevant to the brownfield part of the project.

Transition Efficiency: Likelihood that we will be able to implement adequate knowledge transfer , for example ensuring all team members understand critical legacy information.

Code quality: Likelihood that we will be able to implement high quality code , however this takes into account encountering challenging code that is poorly written or undocumented

Compatibility issues:Likelihood of encountering issues regarding integration with the existing development systems and our new implementation systems, ensuring we don't encounter any conflicts.

We scored these and rated them , giving them varied amounts of attention based on the score given(25- high priority, 7- low priority. By Identifying and communicating these risks early we were able to complete this project successfully without encountering any majorly time consuming issues ; making the project's transition significantly smoother.

[1] https://www.cse.msuedu/~cse870/IEEEXplore-SRS-template.pdf Section 5.3
[2] https://www.ibm.com/docs/en/erqa?topic=scores-negative-statement