Change Report:

Team 14

Joshua Beaven (jb2825@york.ac.uk)

Calum Feenan (cf1218@york.ac.uk)

Harry Hillman (hth512@york.ac.uk)

Jonathon Snelgrove (js3336@york.ac.uk)

Omar Alhosani (oaa560@york.ac.uk)

Rayan Butt (rb1770@york.ac.uk)

REQUIREMENTS

Requirements change report:
URL to old requirements:
https://drive.google.com/file/d/1Zig19N6itzXcc-EWavvsFZvftuwG2XYS/view?usp=share_link
URL to new requirements:
https://docs.google.com/document/d/1cb_z7XnFRjg_WHMYKEqh-Q3YBQakfRb3IknLoAZPPJk/edit?usp=sharing
Updates to the old requirements are highlighted in Red(indicating a new requirement) or are highlighted in Yellow(where an old requirement has been edited. The list of new requirements was created by first checking the old requirements and seeing what was currently on there. We then consulted the product brief to find out what were new boundaries the project had to follow. From this, we first created and edited all the user requirements that the new constraints gave us.

The new user requirements we created were: UR_FAIL_PREP (the user could now fail prep at an ingredient station), UR_CUSTOMER_ARRIVE (customers for the users game could now enter in groups and expect to be served in a specific time), UR_NUM_CHEFS (The number of chefs has been increased from 2 to 3), UR_ENDLESS (a new game mode was to be added that allowed the user to play forever), UR_INVEST (the user is now allowed to invest money they have gathered into unlocking new stations and workers), UR_POWER(new powerups), UR_Difficulties(Multiple difficulties), UR_SAVESTATE(Allow the users to save and reload a game). User requirements that we also edited were: UR_WIN (now specific to “scenario” mode), UR_LOSS (Game can now be lost if customers are not served in time), UR_RECIPE (added new recipes for pizzas and jacket potatoes).

From these new user requirements, we created the new functional requirements. The new functional requirements created were: FR_INFINITE, FR_FAIL_ACTION, FR_INVESTING, FR_CUSTOMER_ARRIVE, FR_POWERUP, FR_DIFFICULTY, FR_SAVEGAME.

We did not introduce any new non-functional requirements as all of the new user requirements were not requirements that specified criteria for the project to meet. Nor did we introduce any new constraint requirements, as no new constraints were made.

Method Selection & Planning

The Document change is: https://drive.google.com/file/d/13THD8r1enXh_r5sJNZ_uuDehA_RcSTJF/view?usp=share_link

For the Team 13 method selection & planning report, I didn't feel that I needed to change much of the documentation as our team's structures are quite similar. There were a few sections that needed amending though, such as the specific software and technologies used, team management, and planning Gantt charts. See below for exact document changes. We also decided to use the same SCRUM approach as we found this approach most effective. We chose to use SCRUM, which is an Agile method, as it is good for small teams, short iteration cycles, and accommodating change with low documentation overhead which was beneficial to us due to our time constraint. Other engineering methods, such as plan-driven methods, were not as advantageous to us as they require a stable environment and extensive documentation in order to communicate and progress the project. Plan-driven methods are not suitable for this project as new requirements from the client will be produced, causing a need for change. Also, due to the time constraints, a large amount of paperwork would detract focus from development.

When using SCRUM, we would have one weekly SCRUM meeting, potentially more depending on deadlines, to see what work has been completed and delegate more tasks. This way, we were able to ensure requirements were met due to the regular meetings and changing any plans if tasks took longer to complete than anticipated.

Development & Collaboration Tools (Implementation):

Team Communication: No change: Discord

Version Control: No change: Git with GitHub

Documentation: No change: Google Drive

Task Management: In-Person Meetings & Discord We found that the best way to manage our tasks was to discuss our plan of action in an online or in-person meeting, list down what we all needed to do on either our own notebooks or the To-Do-List section on the discord page. We found that this method was most efficient as our tasks are discussed with the team beforehand, clearly laid out for us, and are noted without the need for more unnecessary use of technology or software, so less work overall. We also find that this method is better for an individual pace of work, as our team structure delegates one topic of around about one topic of the project per person, aside from implementation.

Team Organisation: For our team organisation, we mainly use in-person meetings to discuss weekly tasks and activities, allowing us all to keep updated with each other's completion and workload. As mentioned before, in these meetings, we would create a To-Do-List, and figure out what needs to be done and who needs to do it in the meeting. We also host regular Discord update calls to check each other's progress so that we stay up to date with all work at hand.

In terms of workload distribution, our team has split the work into two categories: documentation and implementation. Since the first part of the project, where we initially had Jonathon, and Harry work on implementation and Calum, Josh, Omar, and Rayan work on sections of the documentation each, we have altered the distribution more equally so that there is less overall stress and a more equally distributed workload. Josh now supports

Jonathon and Harry, working primarily on implementation and lending assistance when needed to software testing reports and architecture.

As previously mentioned, our chosen software engineering method is SCRUM. We met at least once a week, either in person or online, through Discord to have sprint planning meetings. These sprints would be a few days to a week in length. During each meeting, we created a list of tasks
that we needed to do from reviewing the requirements and the previous sprint's work. This way, we can keep up to date with issues in the code and unfinished work and ensure requirements are met. We were unable to fully keep to our meeting plan due to team members being unavailable, especially during the Christmas holiday when members were away seeing family or revising for upcoming exams.

Project Plan: The initial planning for several days of the project was mostly focused on sorting out the requirements documentation alongside delegating the assignment of work, deciding the best software methodology approach, the technologies we would use, and familiarising ourselves with the assignment. We found that deciding all of these preliminaries before we start the process of the project would be most beneficial as all group members would be on the same page.

Over the next month, we completed the interview with the customer and then started to work on implementation, having decided on all of the software used for communications, implementation, and design. We also started the production of the website at this point. At this stage, we had a few team meetings as the project was waiting on the customer interview to attain all information to proceed with the project.

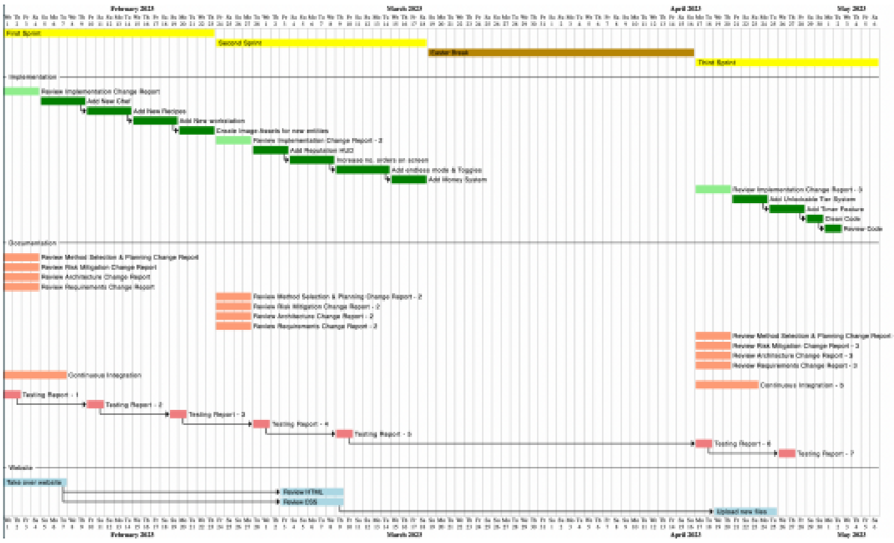
We didn't do a team meeting or much work over the Easter break. As a team, we believed it to be most productive to have an essential break in order to prepare correctly for our exams whilst also having the ability to relax and recover.

During the first week or so of the new project, we came to terms with and spent time understanding the new environment. This involved learning about the previous group's implementation, their method and planning approach, understanding their architecture, and reading their HTML & CSS for the website. We then decided to compare our project so far with this new project that we have taken over and create the change reports, starting with this as a beginning.

From here, we laid out a plan and approach for the new project and decided to create this plan on our chosen SCRUM methodology as the way forward. We chose this method as we found that it was easiest to tackle the task of taking over a large project in small, manageable pieces, adding our own features in increments. We decided to make a Gantt chart to showcase these incremental improvements by having multiple and regular testing reports, using continuous integration, and regularly reviewing and updating our change reports, all whilst regularly meeting.

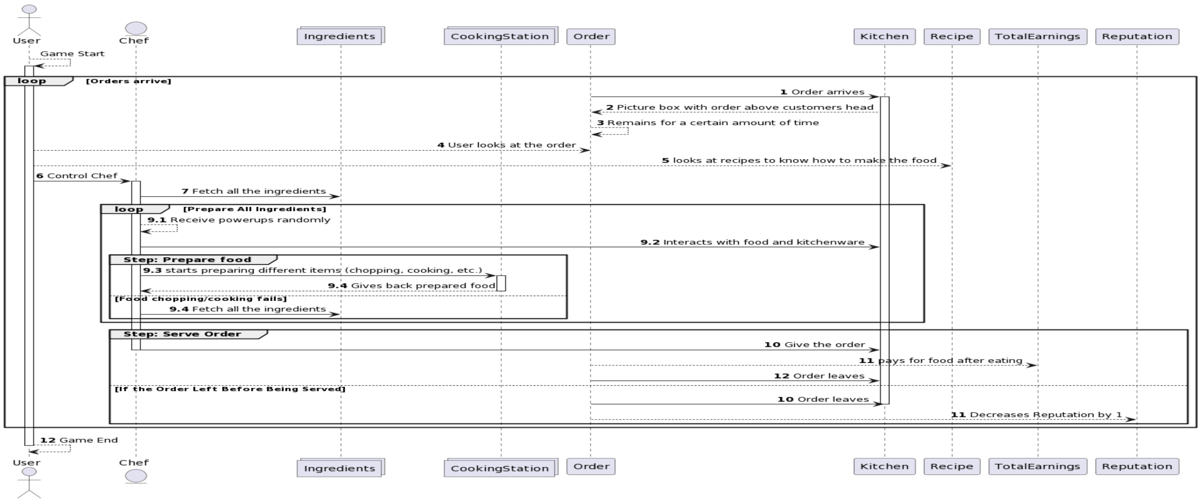
The use of the Gantt chart also helped us keep our project split into manageable 'sprints', allowing work to be equally divided up over a large amount of time rather than leaving all of the work until the last minute.

Our Final Gantt chart can be found below:


February 2023
March 2023
April 2023
May 2023
First Sprint
Second Sprint
Easter Break
Third Sprint
Implementation
Review Implementation Change Report
Add New Chef
Add New Recipes
Add New workstation
Create Image Assets for new entities
Review Implementation Change Report - 2
Add Reputation HUD
Increase no. orders on screen
Add endless mode & Toggles
Add Money System
Review Implementation Change Report - 3
Add Unlockable Tier System
Add Timer Feature
Clean Code
Review Code
Documentation
Review Method Selection & Planning Change Report
Review Risk Mitigation Change Report
Review Architecture Change Report
Review Requirements Change Report
Review Method Selection & Planning Change Report - 2
Review Risk Mitigation Change Report - 2
Review Architecture Change Report - 2
Review Requirements Change Report - 2
Review Method Selection & Planning Change Report
Review Risk Mitigation Change Report - 3
Review Architecture Change Report - 3
Review Requirements Change Report - 3
Continuous Integration
Continuous Integration - 5
Testing Report - 1
Testing Report - 2
Testing Report - 3
Testing Report - 4
Testing Report - 5
Testing Report - 6
Testing Report - 7
Website
Take over website
Review HTML
Review CSS
Upload new files

ARCHITECTURE

SEQUENCE DIAGRAM:


User
Chef
Ingredients
CookingStation
Order
Kitchen
Recipe
TotalEarnings
Reputation
Game Start
loop [Orders arrive]
1 Order arrives
2 Picture box with order above customers head
3 Remains for a certain amount of time
4 User looks at the order
5 looks at recipes to know how to make the food
6 Control Chef
7 Fetch all the ingredients
loop [Prepare All Ingredients]
9.1 Receive powerups randomly
9.2 Interacts with food and kitchenware
Step: Prepare food
9.3 starts preparing different items (chopping, cooking, etc.)
9.4 Gives back prepared food
[Food chopping/cooking fails]
9.4 Fetch all the ingredients
Step: Serve Order
10 Give the order
11 pays for food after eating
12 Order leaves
[If the Order Left Before Being Served]
10 Order leaves
11 Decreases Reputation by 1
12 Game End
User
Chef
Ingredients
CookingStation
Order
Kitchen
Recipe
TotalEarnings
Reputation

Sequential Diagram

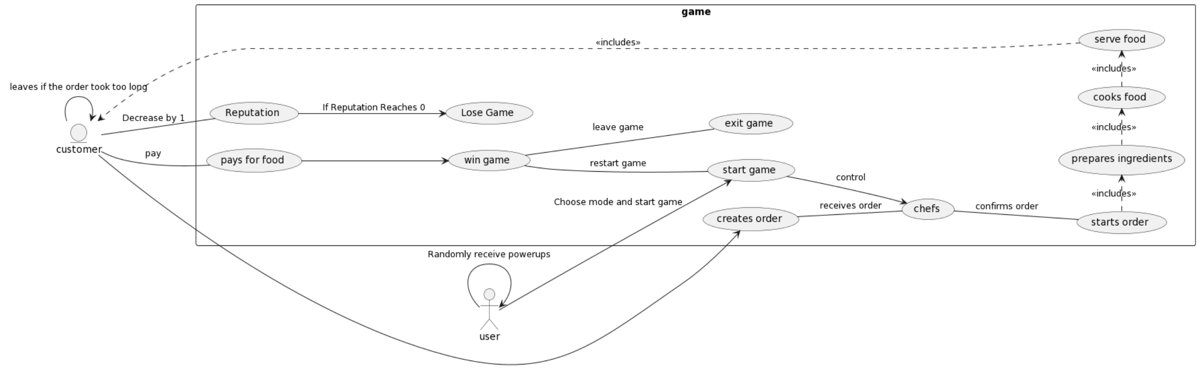
A great number of changes have been added to the sequential diagram. Prior to this, this diagram had a basic and simple implementation. Now, additions like loops, steps, and if else statements have been added to the diagram to give a more in-depth look into how the gameplay is likely to be carried out.

Two loops have been added here, with one being a nested loop. The “Orders arrive” is there to showcase how the game will have more than one order before the game is finished. This gives a general idea of how the game will be played. The inner loop is to demonstrate how the Chef will have to process multiple ingredients to get the finished product. The inner loop also has an “If Else” statement in case the ingredient processing fails, and the chef has to start over.

The Serve Order step finalises the process where the chef places the order on the counter, and the TotalEarnings is incremented. In the “If Else” statement, the Order leaves before receiving the food, thus losing a reputation point.

Chef is now an entity, Ingredients and CookingStation are a collection, and the Reputation class is added to the diagram.

Use Case Diagram:



Use Case Diagram

Previously, there was no scenario where the user lost a game. As illustrated above in the Use Case Diagram, a Reputation system is implemented into the Use Case diagram, which dictates the losing condition for the player. If the Reputation reaches 0, the game stops, and it would mean a loss.

And in this iteration of the game, the “customers” or orders remain for a certain amount of time before leaving or until the order is received. “Customer” and “Chefs” are now labelled as an Entity.

Elements such as Powerups have also been implemented and will be randomly added during the gameplay. These powerups will be extremely helpful when playing in the endless mode of the game.

STRUCTURAL/CLASS DIAGRAM:

CHEF CLASS: Chef Class Diagram
ORDER & RECIPE CLASS: Order & Recipe Class Diagram
INGREDIENTS CLASS: Ingredients Class Diagram
HUD CLASS: HUD Class Diagram
SPRITES CLASS: Sprites Class Diagram

The changes that were made to the classes include the addition of a pauseScreen class, multiple recipe/ingredient classes, oven classes, powerup classes, and an unlockables class. The pauseScreen class, which was added, leads to a similar screen to the start screen; however, the option to change the mode from SITUATION to ENDLESS and vice versa is not available, and neither is the difficulty changing option.
The recipes added include pizza and jacket potato, which add more variety to the game - as do the ingredients (cheese, dough, potato).
The Oven class is an extension of the InteractiveTileObject class, which takes a near-complete recipe such as UncookedPizza or JacketPotato and then cooks it so that it can be handed to the customer. Powerup classes have been implemented to extend Sprite, a LibGDX class - they appear on top of the worktop for the Chefs to approach, interact with and collect. PowerupSpawn is used to create a space for power-ups to appear.

Furthermore, Unlockable was added in order to create workstations in the game, which require the use of in-game money to be used.

RISK ASSESSMENT

When looking over the Risk Assessment and Mitigation deliverable created by Team 13, we didn't feel it needed to be amended as it continues to be a complete and accurate assessment of the risks involved in the second stage of the assessment.

After reading through the description of the process of risk assessment, we felt it was an accurate assessment of the ideology our team adopted towards risk assessment and described the definition of risk correctly. It also correctly identifies and describes each of the stages of risk management clearly and concisely.

The document then goes on to describe the risk register they used, which includes all the relevant information required for an effective analysis of the risks of the project; this includes ID, Type, Description, Likelihood, Severity, Mitigation, and Owner. We feel the most important section to be concise with is the description and mitigations of the potential risks, and this is because, without an accurate description, the problem could be difficult to identify, which can cause the team the struggle when dealing with a previously prepared problem. Mitigations are very important if the instructions to avoid these risks are not clear and concise, and it can result in the risk occurring even when it is believed that precautions were taken for said risk.

One thing we had to change was the Owners column in the risk register. This is because there are different members of my team that are now responsible for each risk as opposed to members of the previous team. This was generally a simple transfer as our roles and tasks were clearly laid out for us beforehand, so we checked who was in charge of each section and assigned them ownership of the appropriate risk. This meant, for example, all the risks that were related to coding went to Harry and Johnathon, all testing-related risks went to Josh and Callum, and all risk identification risks went to Rayan.