Team 19: Hard Cubans

ENG1 Assessment 2 Deliverable:

Change Report

Team Members:

Thomas Feilden

Salman Khan

Ben Green

Charles Baroud

Jacob Williams

Meilir Lloyd

2a:

Process

Inherited Documents:
We reviewed Team 13's documents and identified areas where it could be improved and also made the necessary changes since some of the documentation is only relevant for Team 13 and not us. We then made a plan on how we can update the documents and ensure that the future development would be clear and easy to understand.

Inherited Code:
Reviewed Team 13's code to understand how it worked and identify any issues that need fixing. Once we had a good understanding of the code, we could come up with a plan to refactor and improve it based on the assessment's requirements.

Reviewing Changes:
We held regular meetings to review changes. Generally there was one meeting a week over discord and one in person for the practical. This is where we could discuss and review what the team has done in their own time and provide feedback on what is good and what needs improving.

Tools

Trello:
After evaluating all of Team 13's deliverables, we saw that they used Trello for their team management so we decided to have a look at it and were happy with what it could do. From there we made our own board and added all the tasks that we had identified to it.
GitHub:
We used GitHub for our version control because it can keep track of the changes to the code base and allows for easy collaboration between the team on the project.
Visual Studio Code:
We decided to use Visual Studio Code for our IDE since we are all familiar with it and it allows for easy access to GitHub. Unit tests are also easy to run with one click of a button and we can see the outcome visually using green ticks or red crosses.
Google Docs:
Using Google Docs meant that if a note hadn't been made of what changes had been made, we could go through the version history to identify what was changed. In the next meeting we would ask whoever made that change to justify it and then add it to this document.

Tracking Changes

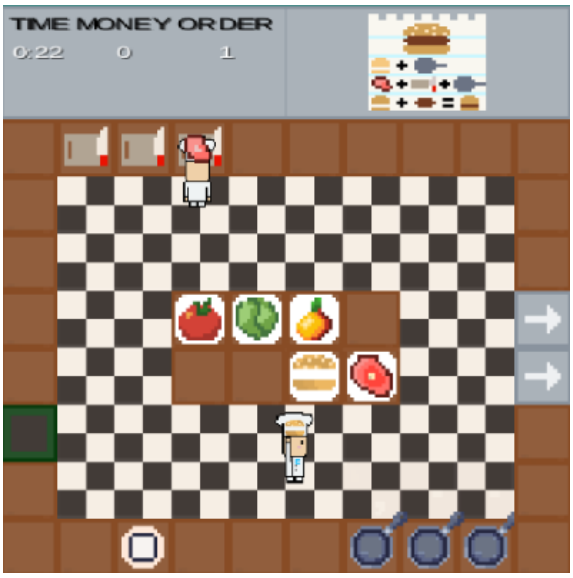
From the beginning, we made it very clear that any changes to the assessment 1 deliverables needed to be documented and have a note on why it was changed. This will be then added to this document where we can systematically show what we changed and justify why using the notes people made after each change. Using Google Docs and GitHub helped us in tracking these changes (as seen above).

2b:

Requirements:

The requirements document needed a moderate level of modification to fit the criteria for Assessment 2. In the introduction, we have made a couple of changes which included some research on existing solutions for requirements and added some details on how the tables are structured.

Furthermore, the document also made no mention of two elements that were required in Assessment 1. That being the stack feature for a chef carrying items and the customers that arrive into the restaurant to be served, recipes would simply appear at the top of the screen and there was no class for customers as interactable objects within the game. Note the image below where there are no customers and each chef can only carry one item at a time.



The requirements also had no mention of the two game modes that were required, endless and scenario. These had to be added as criteria to be met in order to meet the final brief. The deliverable was adequate but focused too much on implementing the recipes, chef collision and switching between chefs, all of which are essential to the game running but showed the team's narrow view on features.

The list of added requirements are as follows:

<u>User Requirements:</u>

- UR_CUSTOMERS: The game should have customers that arrive with unique requests that wait to be served and react depending on whether their order was correct or not
- UR_GAMEMODE: The game can be played endlessly or in a scenario mode with a limited number of customers
- UR_DIFFICULTY: The game should have different levels of difficulty for the user to choose from (easy, normal, hard)
- UR_STACK: The cooks should be able to carry a 'stack' of ingredients, dropping the top item on the station they interact with
- UR_SAVING: The game can be saved at any point and resumed later
- UR_POWERUP: The cooks should be able to pick up five power ups that improve their efficiency in some way (e.g. speed increase, shorter cooking times)
- UR_PROGRESSION: At the start of the game, some of the cooking stations will be locked which can be enabled by using some of the earnings. Earnings can also be used to call more kitchen staff back from leave.

System Requirements - Functional:

- FR_REACTION: The system will show a reaction from the customer indicating if their order was correct or not
- FR_DIFFICULTY: The system will make customers arrive by themselves at first and after some time, in pairs or groups of 3
- FR_SAVING: The system will allow the player to save the state of the game at any point and resume a saved game later
- FR_POWERUP: The system will spawn power ups at set locations on the map at random time intervals
- FR_GAMEMODE: Before the game starts, there should be a menu allowing the player to choose either “scenario” or “endless” mode

System Requirements- Non Functional:

- NFR_DATA_STORAGE: The system should only store data for the purpose of a scoreboard or a saved game. The names should not exceed a reasonable length.

The additional requirements were added in order to meet the full design brief for Assessment 2, implement any features missing from Assessment 1, or to meet the three new additional requirements that were added (random power ups, difficulty selection, game saving). Finally, the SSON (Simple Statement of Need) was added to the start of the deliverable to introduce what the project and document will be about.

Architecture:

The architecture document needed a major change because the current architecture is inefficient and repetitive. For example, there were individual objects for each station and ingredient which meant that implementing assessment 2 would have been more time consuming and difficult to implement. In addition, the current architecture meant that we could not perform unit tests since all of the code was run in a game loop.The document also isn't very clear how the code works, we had to go through all the files to understand how things work (which is where we found some of the issues relating to the architecture).

Changed Ingredient class:

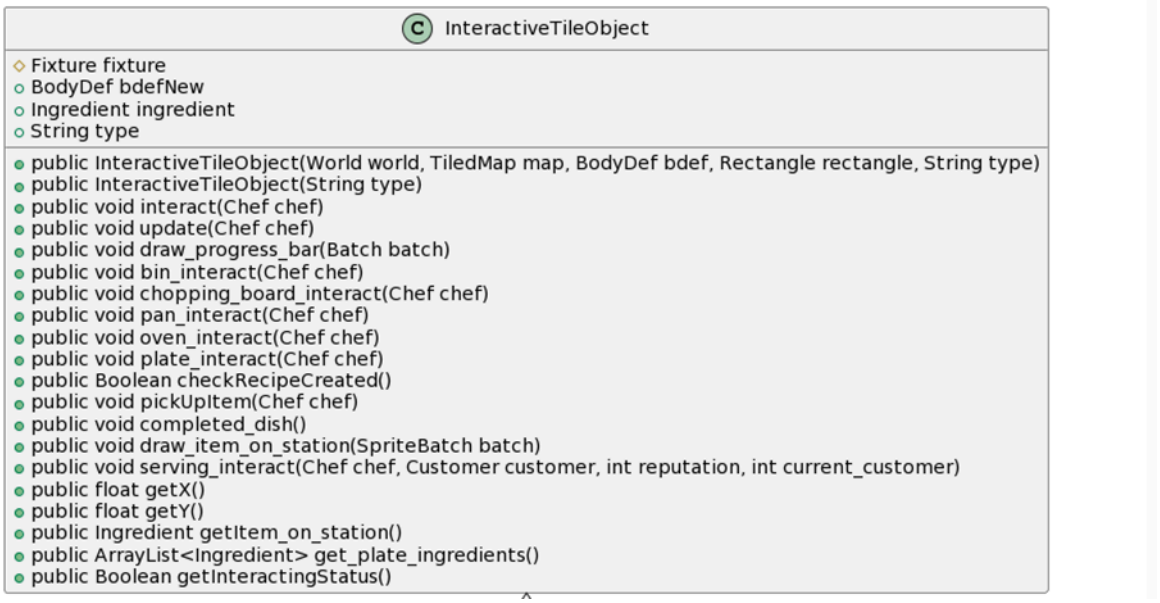
Instead of having an individual class for every ingredient, we simplified the solution to have just one ingredient class. We added a name to the ingredient and also a status that is responsible for showing a texture corresponding to the current condition of the ingredient. We also had to add the boolean attribute "burnt" that is used for the burger to store if it is burnt or not - meaning that you can't assemble a burnt burger. The new UML diagram can be seen in the architecture document.

Implemented IngredientStation:

An IngredientStation was implemented which inherits from InteractiveTileObject except its only purpose is to allow the chef to pick up ingredients.

Changed InteractiveTileObject:

Similar to all the ingredients, there was an individual class for each station and all the code for each station was running in the PlayScreen. We decided to move all the code and update some of the functions to the InteractiveTileObject class. This not only made the code easier to understand, but is essential for testing. In order to reduce the size of individual functions, which meant readability was easier, we split up the interact function into different interactions depending on the type of the InteractiveTileObject. For example, if the type was "Chopping", then the chopping_board_interact() function would be called.



Implemented ProgressBar:
We felt that implementing a progress bar for the game would both help the user determine how long is left for each ingredient to be prepared/cooked but also show when an ingredient is burning (a requirement that wasn't implemented).

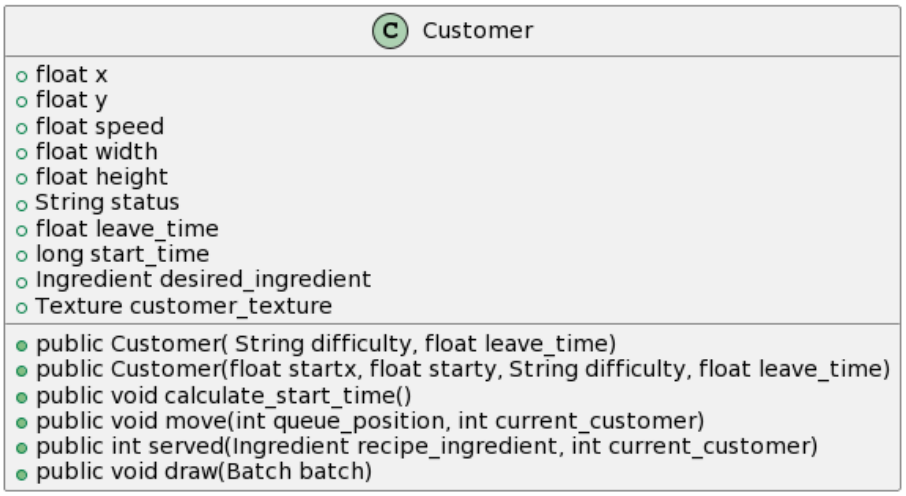
Changed Chef:
The previous implementation had individual sprites for every item the chef is carrying which would mean we would have to make lots of new sprites. We decided to change this implementation and had a sprite for a chef not holding anything and then with their arms up, then the sprite of the chef's current ingredients will be placed on top.

Some of the functions that were in the PlayScreen class were moved into the Chef class to make readability better and used for testing.

In addition, the requirements stated that the chef needed to have a stack of items which Team 13 did not implement.

Implemented Customer:

The previous code did not have any customers but just orders. We decided that to improve the playability of the game we would add customers that move to the serving area, wait for their order, form a queue and leave if it takes too long to be served. This makes the game more interactive as you can see if you are taking too long due the size of the queue and with customers leaving.

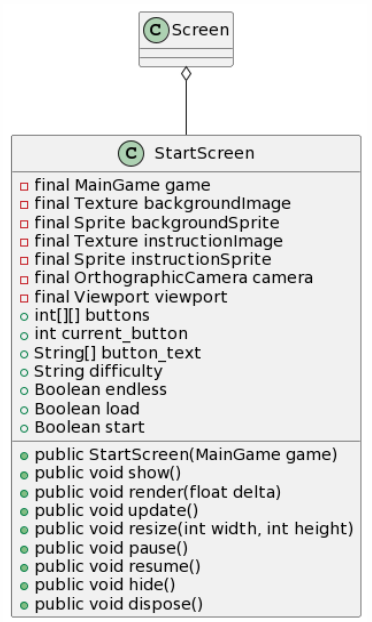


Changes to main game loop:

The main game loop is intended to show all the possible states that the game can be, however previously, it only displayed the states for the chef making the wrong decisions and failing to serve the customer. There was no possible path to serving the customer correctly, them accepting the order and the chef receiving earnings as payments. Furthermore, it is possible for the chef to correct themselves after visiting the wrong cooking station shown by the arrow from wrongStation to correctStation, something that was also missing from the previous iteration of the state diagram. The correct diagram can be found in the architecture document.

Changes to start screen:

The previous implementation of the StartScreen class simply showed the instructions for the game. There needed to be a way to change the difficulty of the game, change between endless and scenario mode and load a previous save of the game. Below is the architecture we aimed to implement:



Method selection and planning:

Software Engineering Method Outlined and Justification

We kept this section unchanged, as frequent scrum meetings and rapid reassignment of tasks worked fine for us during Assessment 1. We have similar time constraints as during Assessment 1, so a plan-driven approach remains unnecessary - especially given the relative lack of experience in the team that may lead to us poorly predicting how long particular tasks might take. Our client/customer remains unchanged, and there are few qualitative changes in the requirements that would warrant a switch to a different software engineering approach.

Changes to Development and Collaboration Tools

We removed Smart Draw from the UMl tools since we did not use it for any of the uml diagrams as we found that PlantUML was sufficient for all of our needs.

We also updated the document in the Trello section since we are using Trello slightly different to Team 13. We are using a single board (compared to their two separate boards for documentation and implementation).

Changes to Team Organisation

We have updated the team organisation slightly but we have decided to change how we were previously organised because we believed that Team 13 had a good approach to team organisation. How we used Trello slightly changed but that was due to us having a different Trello board setup.

We have assigned new projects and product owners as well. There was a minor change to how we used SCRUM for our team which was changing when we organised meetings.

Changes to Project Plan:

We will need to update the project plan deliverable as the plan only accounts for assessment 1 and we need to change it for assessment 2. This section has been nearly completely re-written because it is our plan and not Team 13’s plan and evolution. However, it did follow the general structure of the previous project plan.

Risk assessment and mitigation:

The risk assessment and mitigation document only needed a few small changes which were to the risk register as we are confident in their risk management process and believe it to be the appropriate approach**.**

The first thing we did was update the owner for each risk since we now own the risk register and not Team 13. We did this by having a team meeting to decide who will be responsible. The other change we made was the likelihood of a few of the risks since we believed they were too high. The changes we made are below:

- **R13** - We have changed the likelihood to low because we believe that we have a strong enough programming ability to add all of the features for assessment 2.
- **R15** - We have changed the likelihood from high to low because after updating the requirements from Team13, it is unlikely that they will be changing again.
- **R16** - We have changed the likelihood from high to low because we are all familiar with the requirements and what needs to be done to complete the project. Therefore this will be unlikely to happen

After some evaluation we found two risks that seemed unnecessary to the risk register. These were the following:

- **R6 -** There should be no risk in compiling the game since Gradle can sort it and GitHub actions can also automatically compile the game for us
- **R11 -** This risk is unnecessary for the risk register since it can be continuously updated if any risks have been identified
- **R18 -** The project won't have any features removed since the brief will not be changing. Therefore, we decided to remove this risk from the risk register