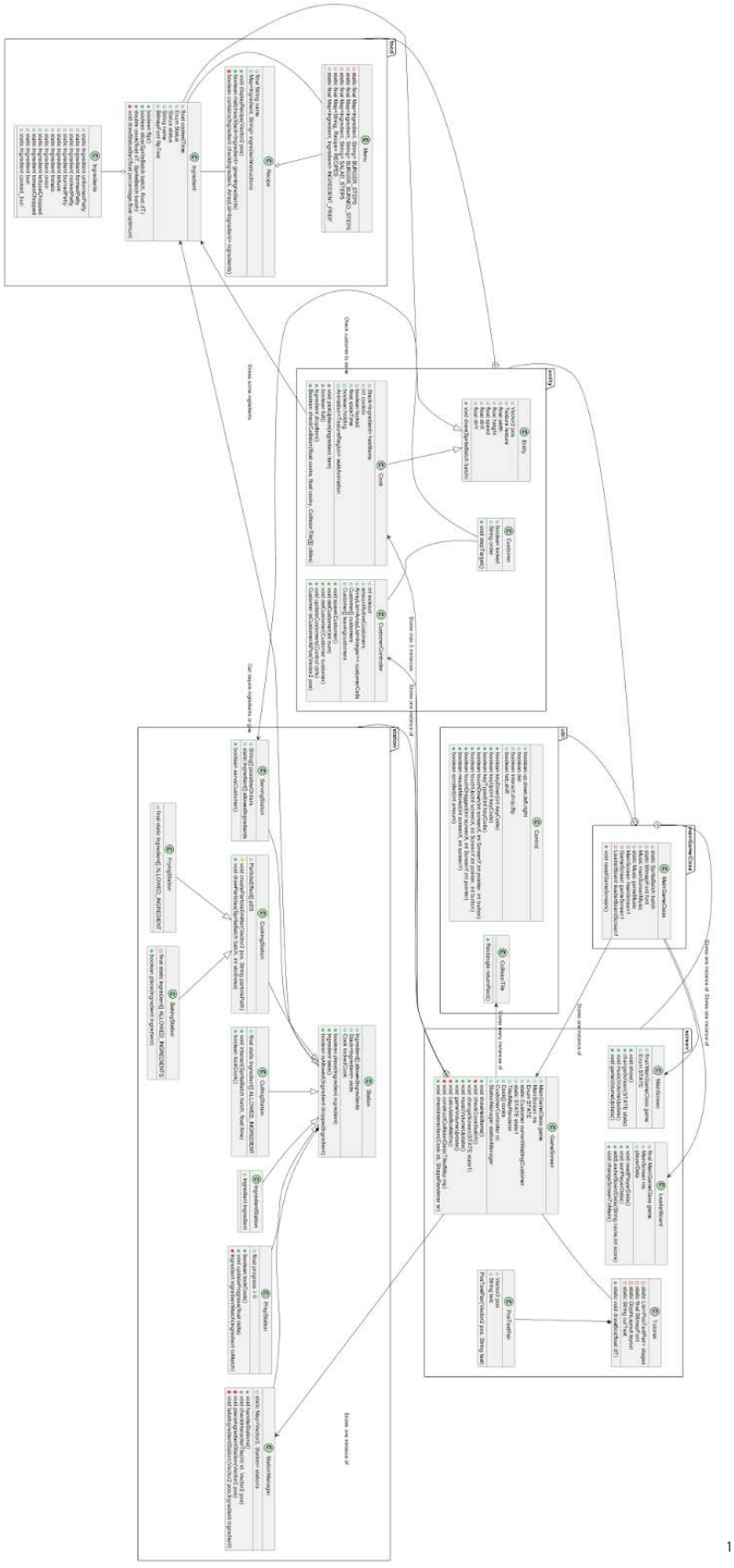
Architecture

Group 3
Assessment 1

Ben Howard <bh1219@york.ac.uk>
Cai Hughes <cabh500@york.ac.uk>
Harry Richardson <hr1040@york.ac.uk>
Ivan Ndahiro <in597@york.ac.uk>
James Sutton <jis509@york.ac.uk>
Yuzhao Liu <yl5164@york.ac.uk>

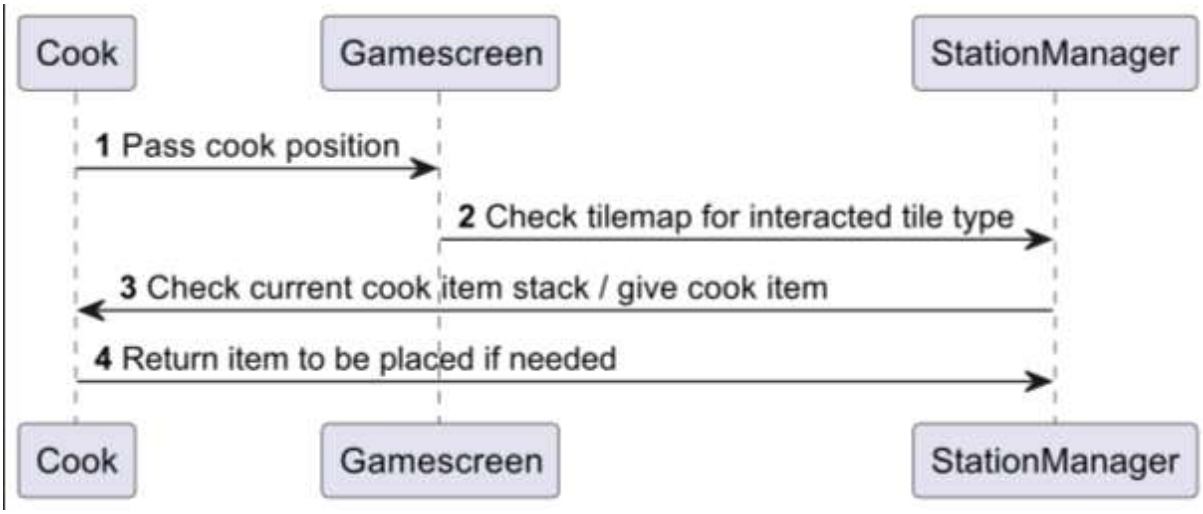
Class structure diagram - Generated using PlantUML



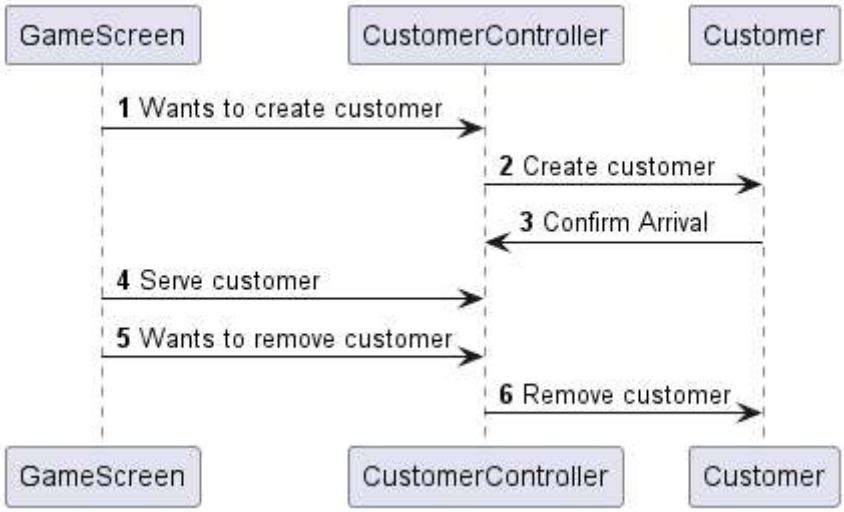
[1]

[1] Full Resolution Image: https://drive.google.com/file/d/1gdHToT4J1ciCABP7xcGOS8B-CiJ0IuZ8/view?usp=sharing

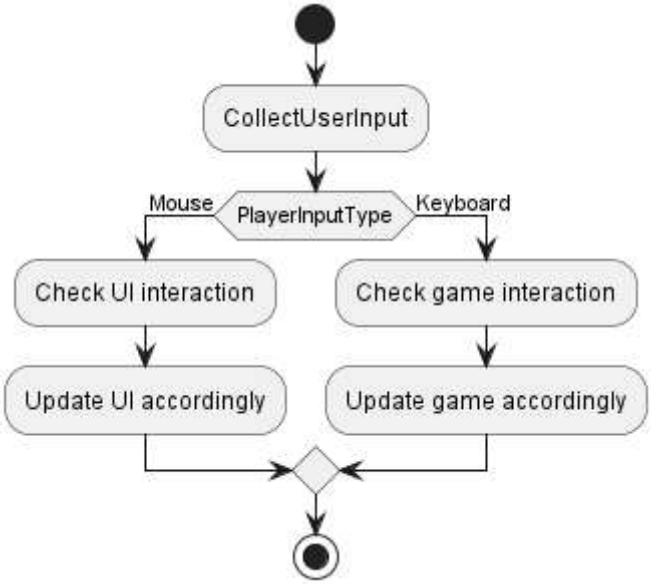
PlantUML sequence diagram for player interactions



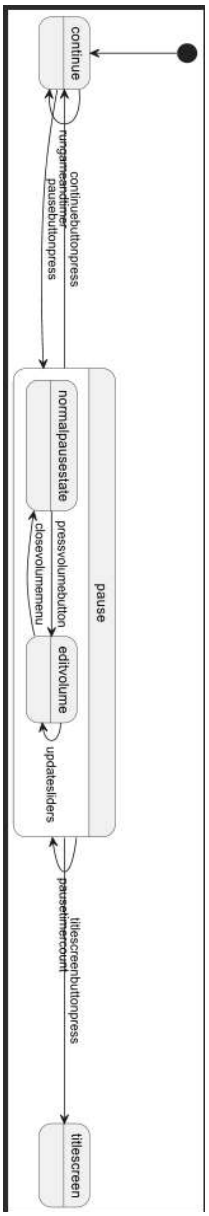
PlantUML sequence diagram for customer interactions

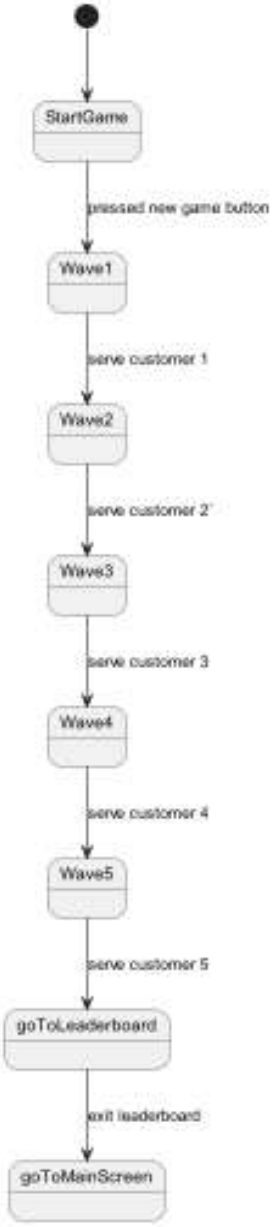


PlantUML Activity diagram for user input handling



PlantUML state diagram for gamewindow + PlantUML state diagram for gameplay state





Initially, we looked at the user requirements and from this, we developed a list of potential classes and structures we would need in order to fulfil them. This included linking how the user would interact with the system and how it would respond. For instance, the user would use a keyboard's controls (FR_COOK_CONTROLLER) to move the cooks and perform the various actions. Additionally, they would have to use the mouse cursor to navigate the menu. We made the assumption that most users would be familiar with the typical WASD controls since the majority of games utilise these controls. We also accounted for the closeness of keys, for instance the keys q, e, tab, and shift are quite accessible from the standard position of the left hand on the keyboard. This would meet the UR_CONTROL_SYSTEM, UR_UX user requirement and NFR_OPERABILITY non-functional requirement..
Another consideration that was important to the architecture of our system was similarities between functions. For instance, all stations would behave in a similar manner with the player controlled cook being able to take and drop items when appropriate from these stations. Similarly, connections were made between cooks, customers and ingredients since all would require similar properties of a position, texture, dimensions (width and height), and motion. Consequently, it would make sense for likewise objects to inherit from a shared parent. We developed CRC cards to make this clearer (See Figure 1).

One key functionality which would affect the overall feel to the game would be how actions would be performed on stations and consequently ingredients. We decided that the best course of action would be for all ingredients to have an internal 'cookTime' and 'slices' state which would then be updated if the ingredient was on the appropriate station with the corresponding action being performed. This would meet the FR_COOK_ACTIONS, FR_CUTTING, FR_FRYING functional requirements. Additionally, one gameplay differentiation between the two would be that cooking wouldn't need the cook to be at the station, only there to initiate it, whereas if the user wanted the cook to cut, they would have to lock them at that station. This made more sense since you could quite easily leave something in a pan to cook while slicing is an action directly performed by a person. Additionally, if the user had moved the cook(s) away from the station and for some time, they'd be punished for leaving the ingredient(s) unattended.

In the case of the stations, each would have a number of slots, allowed ingredients, and then appropriate methods to deal with entity interactions. This allowed for flexibility when creating a new station and preventing the user mistakenly placing the wrong ingredient on a station. When the customer arrives at the service station, the user would have to guide one of the cooks to that station to get the order and only when it is placed on the station will the customer leave.

As the project evolved over time, several changes from the initial design became apparent. One such change was in the tutorial menu and the other screens (main menu, pause menu). It felt more natural for the user to be guided through the game, to the exact locations of the stations, when they first played instead of having the tutorial on just another screen with different images. This lets the user understand where the locations are without having to remember each location. Over time, the code in our MainGameClass increased dramatically and it became apparent that we would have to separate certain

components and functionalities into different classes. This was especially the case when separating the main game code from the menu and other UI elements.

For creating recipes and menu items, a more modular system felt natural for our implementation. This included individual classes for ingredients and recipes along with a menu and list of ingredients to hold all final recipes (UR_RECIPES) and ingredients respectively. Then, through a station class, we could specify which ingredients the station takes or provides (depending on the type of station - UR_PANTRY_STATION, UR_ITEM_STATION).

Figure 1.

For some of the user requirements, we developed a table below with a brief description on how they would be implemented:

UR_SCENARIO_MODE	Our customers will arrive one-by-one and the game will finish when all have been served (currentWave >= MAX_WAVE).
UR_CONTROL_SYSTEM	A control class will contain boolean properties for every action which will be true when the key is activated.
UR_ITEMS	The cook will have a stack of held ingredients which can be dropped and added to through station interactions.
UR_DEMANDS	If the service station is interacted with by a customer, a demand will be made (which is an instance of recipe).
UR_NUMBER_OF_CUSTOMERS	There will be a global constant for the max number of customers (like a wave system).
UR_COOKS	Two instances of the cook class shall be created and rendered with a current cook being controlled directly.
UR_TOOLTIP	When approaching an intractable station, the appropriate key will be shown (e.g. ‘f’ to flip)..
UR_LEADERBORAD	When the game has been finished, the time will be uploaded to a leaderboard and sorted through that class.