Architecture

Group 5 :
Callum Watson
Jack Guan
Craig Slomski
Helene Dima
Chase Mo
Kamrul Islam

3.A Diagrammatic Representations

The following diagrams were created using PlantUML. See more about how this came into use in Section 3.B

Structural Class Diagrams:

Fig 1.0: Chef and Customer OOP inheritance diagram:

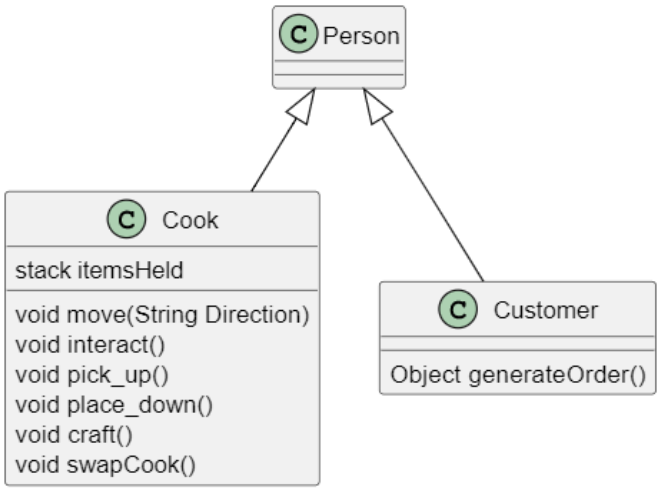


Figure 1.1: Order queue and customer relationship:

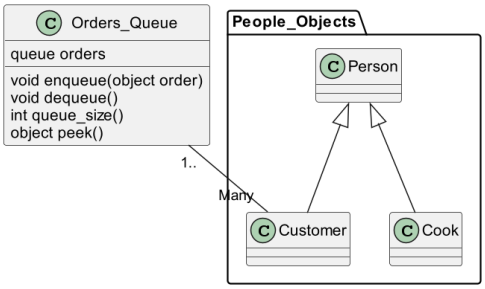


Figure 1.2: Ingredient classes inheritance and class relationship design:

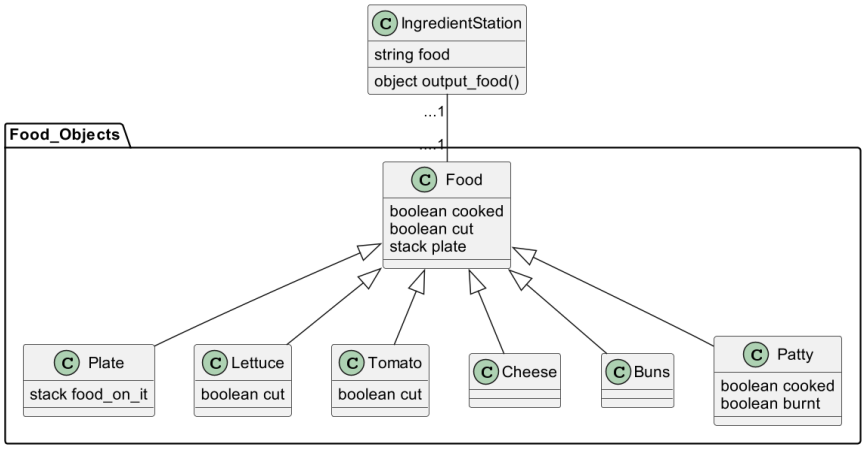
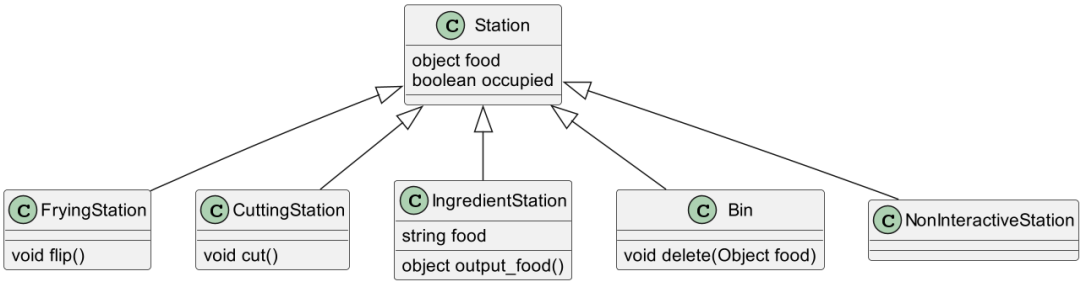


Figure 1.3: Station class OOP design:



See Figure 1.4 on our website, illustrating the class diagram for the different game screens

Behavioural Sequence Diagrams:

Figures 2.0-2.5: Chef interactions (between Chef's and stations):

Figure 2.0: Counter Storage

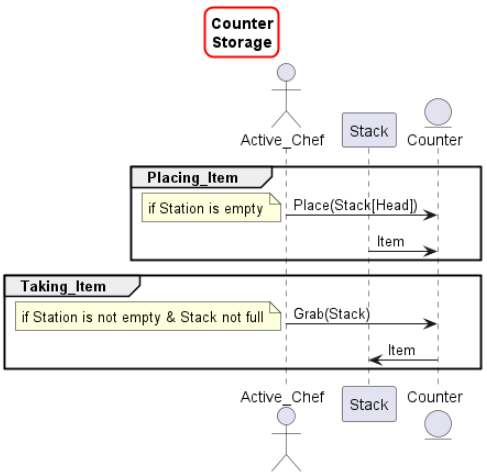


Figure 2.1: Disposing Items

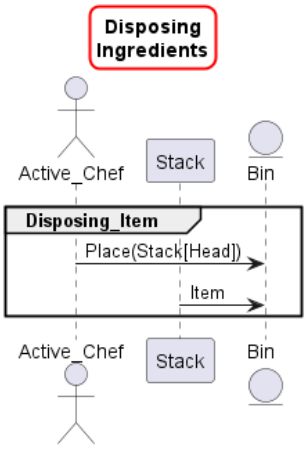


Figure 2.2: Serving Meals

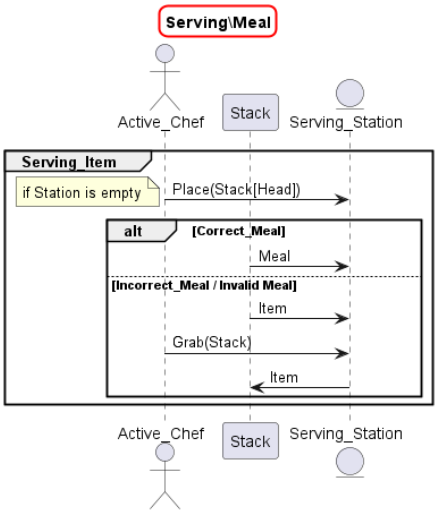


Figure 2.3: Preparing Ingredients

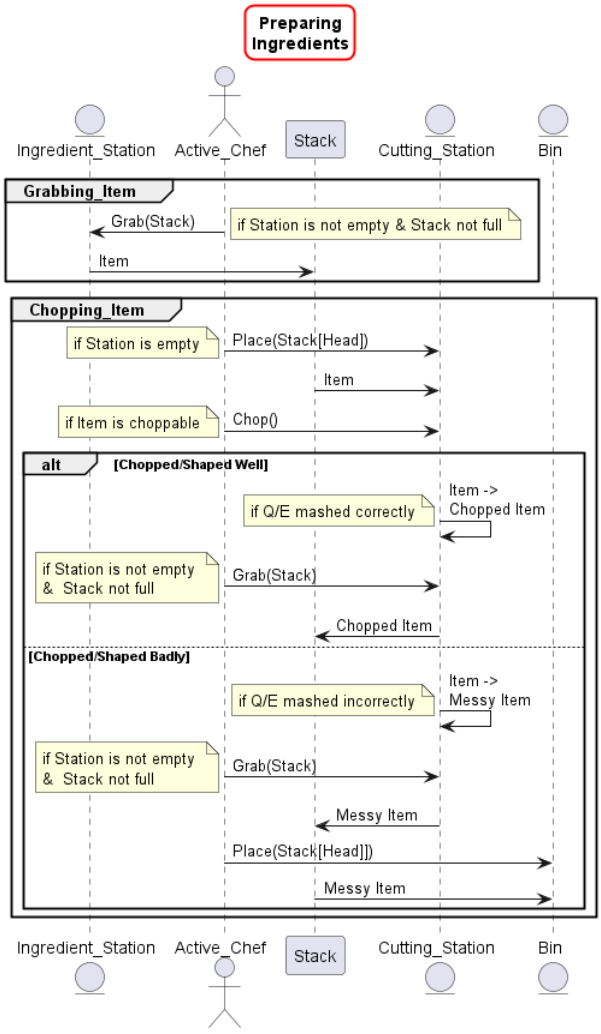


Figure 2.4 Frying Ingredients

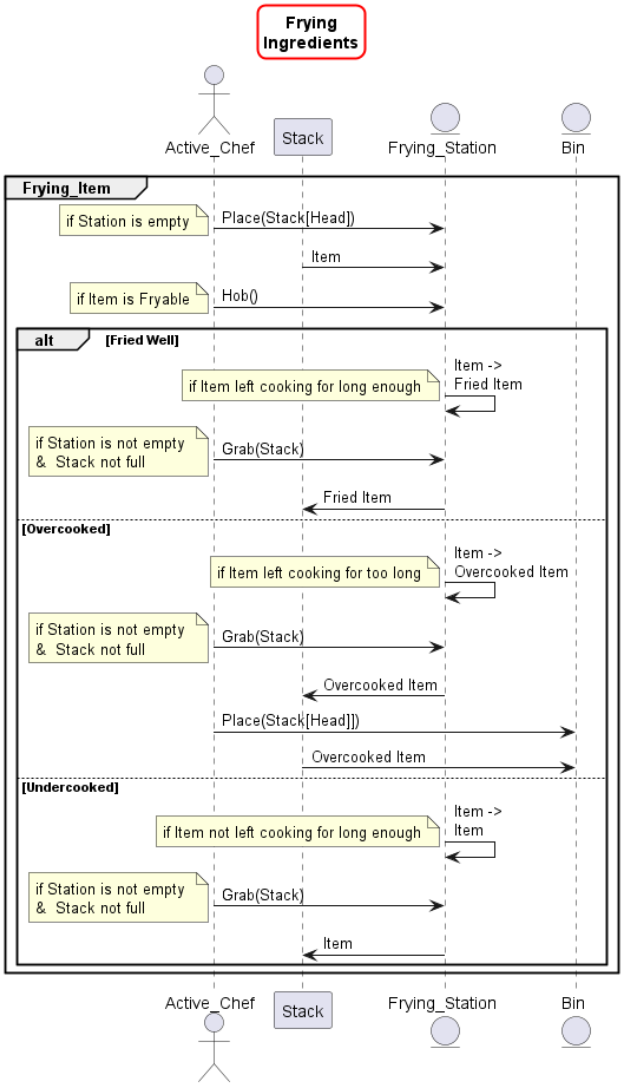


Figure 2.5: Crafting Meals from Ingredients using Recipes

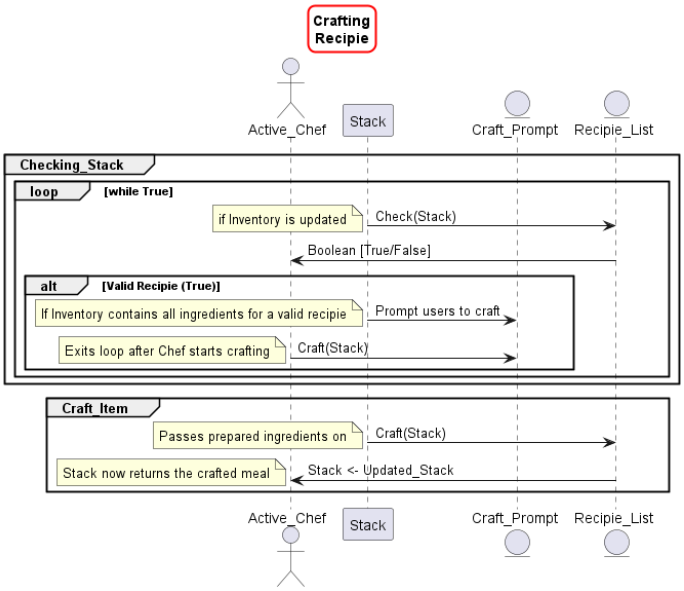
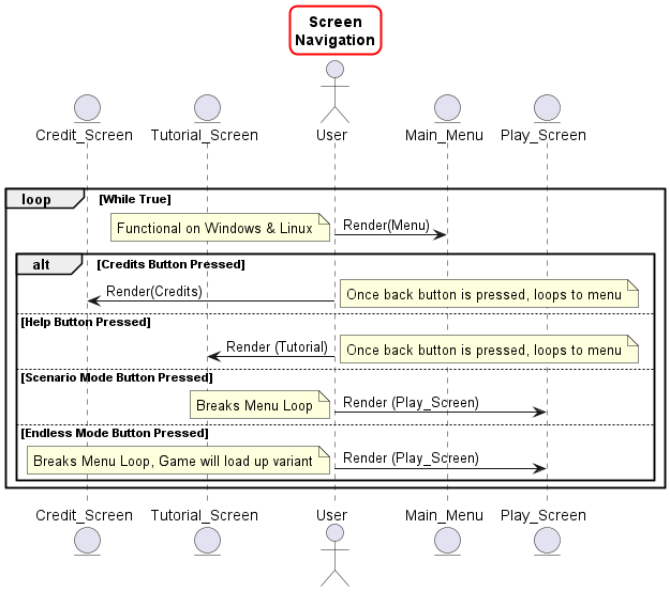


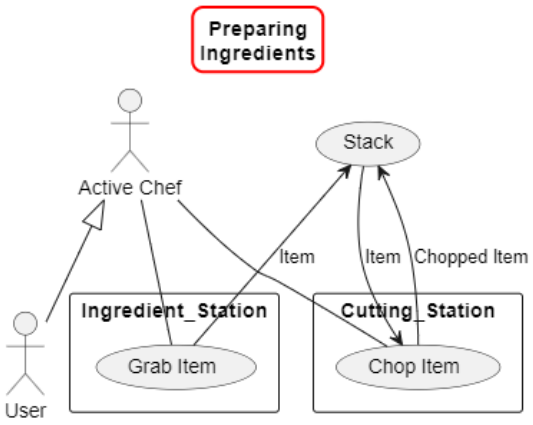
Figure 2.6: User interactions (User and Menu Interactions):



Graphical Use Cases:

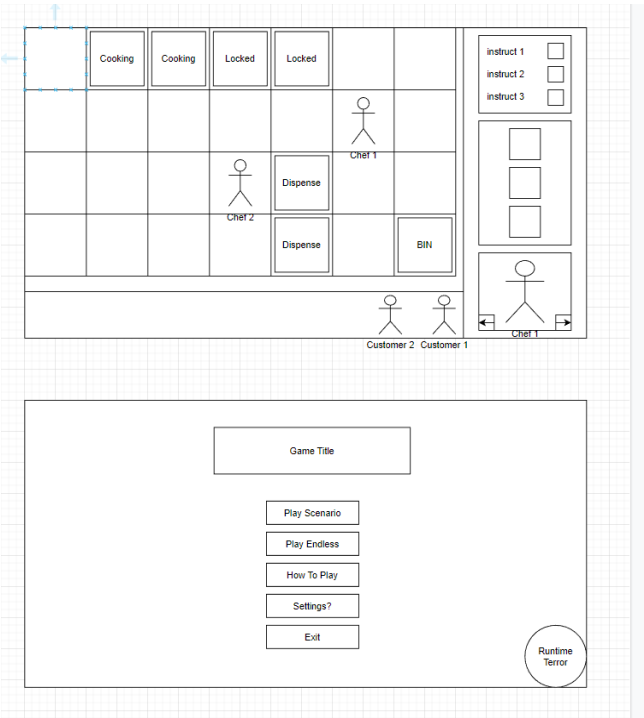
No longer used, see reasoning in 3.B

Figure 3.0: Preparing Ingredients Use Case



Additional Architectures:

Figure 4.0: Initial map layout was made in Draw.io and was purely to get an idea of how our menu and kitchen were intended to look



3.B Architecture evolution:

Introduction:
The original architecture represented a logical representation of the (OOP) game classes/entities and their relationships with one another without the later insight of how they would be implemented with libGDX. After working with libGDX the class design was adapted so it could be practically implemented, however the logical design still remained.

Formatting and software used:
In order to ensure that our diagrams are consistent and formatted clearly (as described in Risk 10, R10) we are using PlantUML to design: Graphical Use cases, Class Diagrams and Sequence Diagrams. Using PlantUML ensures that our design is clean and VScode extensions for PlantUML are readily available to all team members making it clear and easy to use. In addition to ensuring that no architecture is limited to one device and inaccessible to the team (as described in Risk 5, R5) these will be pushed to Github.

Original Plan:
Initially we started using use-cases as shown in Figure 3.0 as a form of interim architectural diagrams, which showed an overview of relationships between actors and use cases. This was helpful initially to get an understanding of how the game would function at a non-technical level. However, since we did not go back to the stakeholder with the architecture side of things, it meant that they were no longer very useful and instead the focus shifted over to Class Diagrams and Sequence Diagrams.

Current Plan:
In the end, we decided to stick with Class Diagrams and Sequence Diagrams.
Class Diagrams to show the structure of the project and used as a model as blueprints for the implementation. So the Class diagrams guide the implementation instead of a direct mapping. This meant that our Class Diagrams could be more abstract and clear for future team members. To build these Class Diagrams we first made Class Responsibility Collaborator diagrams shown later in order to abstract the technicality further to improve understanding and prevent missing features from the finalised Class Diagrams.
Sequence Diagrams to show the intended behaviour of the project and used as model as sketches, so therefore there is no strict notation but are designed to accurately map the user requirements to a behavioural model with an idea of how it could be implemented (for example in Figure 2.1 the buttons to chop were Q/E alternating, but in practice the implementation involved different interact buttons entirely)

Architecture diagrams with no mapping to requirements:
Some architecture diagrams such as Figure 4.0 Kitchen Layout, does not have a requirement relating to it but as developers we felt it was important that we had the same vision and is a key element into the overall game design. In addition, it is easier to show to a non-technical stakeholder what we have in mind and to check we are on the same page.

Requirements with no mapping to an architecture diagram:
Some of the requirements in the eliciting requirements section were not mapped onto an architecture diagram and we decided not to for one of four reasons:

- Architecture diagram does not support/not relevant to the implementation of the requirement. For example UR_VISUAL_STYLE which is implemented through the use of accessible assets and is not supported by an architecture diagram.
- The requirement is likely to not be fully implemented due to time constraints and the requirement having a lower priority in comparison to other requirements. For example, UR_CUSTOM_CONTROL which we decided would take too much of our implementation time and was highlighted by the stakeholder that it would be a 'nice to have feature' but by no means required.
- The requirement is to be implemented in a later version of the software and is not dependent on the current implementation. For example, UR_EARNING_REWARD is reserved for a later implementation of the game and was not a priority for this version.
- The requirement is a Non Functional Requirement which means it cannot be represented clearly in most architectural diagrams. For example, NFR_MAINTAINABILITY which cannot be clearly shown by an architectural diagram and is better represented by the user requirement UR_COMPATIBILITY. Similarly, Functional requirements have a direct mapping to another user requirement thus are not mentioned in the following table.

Mapping Architectures to Requirements:

Architecture	**Mapped Requirement(s)**	**Architecture**	**Mapped Requirement(s)**
Figure 1.0 Chef & Customers	UR_COOK_CONTROL UR_COOK_INVENTORY UR_CUSTOMER_ARRIVAL UR_CONTROLS	Figure 2.2 Serving Meals	UR_COOK_INVENTORY UR_COOK_CONTROL
Figure 1.1 Order Queue	UR_CUSTOMER_ARRIVAL	Figure 2.3 Preparing Ingredients	UR_COOK_INVENTORY UR_COOK_CONTROL
Figure 1.2 Ingredients	UR_RECIPES	Figure 2.4 Frying Ingredients	UR_COOK_INVENTORY UR_COOKING_STATION UR_COOK_CONTROL
Figure 1.3 Stations	UR_COOKING_STATION UR_PLACEHOLDER_STATION	Figure 2.5 Crafting Meals	UR_COOK_INVENTORY UR_COOK_CONTROL UR_RECIPES
Figure 1.4 Game Screens	UR_MAIN_MENU UR_TUTORIAL UR_GAME_MODE UR_CONTROLS	Figure 2.6 User & Menu Interactions	UR_MAIN_MENU UR_TUTORIAL UR_GAME_MODE UR_CONTROLS
Figure 2.0 Counter Storage	UR_COOK_INVENTORY UR_PLACEHOLDER_STATION UR_COOK_CONTROL	Figure 3.0 Preparing Ingredients Use-Case	N/A
Figure 2.1 Disposing Items	UR_COOK_INVENTORY UR_COOK_CONTROL	Figure 4.0 Kitchen Layout	N/A

Original CRC Diagrams:

Figures 1.0-1.1 CRC Diagrams:

Order Queue	
Enqueues Order Dequeues Knows Order_Queue	-

Cook	
Moves Interacts Crafts Swaps Cooks Knows Inventory	-

Customer	
Orders Knows Order Knows Patience	Order Queue

Person	
-	Cook Customer

Figure 1.2 CRC Diagrams:

Frying Station	
Cooks Flips	-

Station	
Knows if occupied Knows object on station	Frying station Cutting Station Ingredient Station Bin Counter

Cutting Station	
Cuts	-

Ingredient Station	
Provides Ingredient	Food

Bin	
Disposes Ingredient	-

Counter	
-	-

Figure 1.3 CRC Diagrams:

Food	
Knows if cooked Knows if cut	Lettuce Tomato … Other Ingredients

Lettuce, Tomato, … Other Ingredients	
-	-