

**Architecture**

**“Lucky” Team 13**

**Team 13**

**Yuxin Wu**

**Bailey Findlay**

**Elizabeth Edwards**

**Chenxi Wu**

**Alex McRobie**


**Lawrence Li**

# Architecture - Class diagram, Sequence diagram, State diagram, Component diagram and Use-Case diagram

## Part A


### CLASS DIAGRAM

Class diagrams are box-arrow diagrams that can capture the structure of a system. Within the diagram it includes the classes, attributes, operations and relationships between classes of the game that is being developed. In class diagrams, each box has three compartments which can include name, attribute (optional) and operations (optional). These show all the classes that are used in the system.

CHEF CLASS:  Chef Class Diagram

ORDER CLASS:  Order Class Diagram

INGREDIENTS CLASS:  Ingredients Class Diagram

HUD CLASS:  HUD Class Diagram

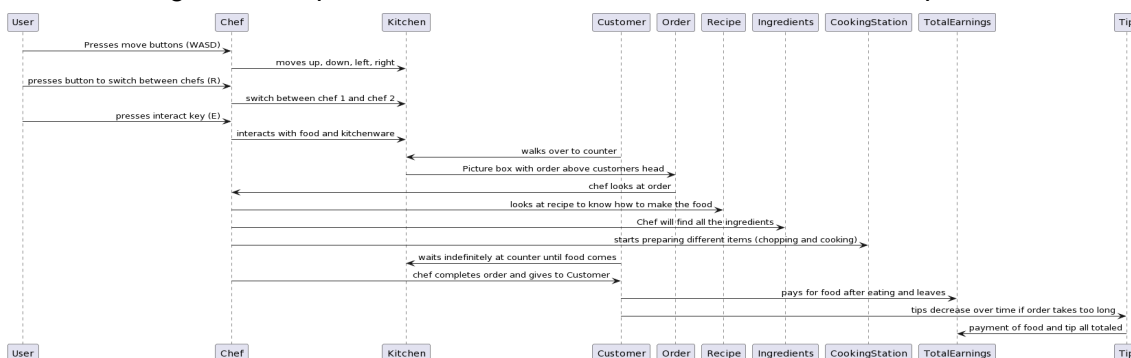
SPRITES CLASS:  Sprites Class Diagram

Each diagram is created using plantUML, in total there are 5 different class diagrams all showing the classes which we have chosen to implement into the code. Within each diagram, it shows the attributes and methods for each class and what the visibility of the code is (shown by the different coloured boxes and circles before each attribute and method). These diagrams give us enough information on the different classes that we need to develop the game and further improve the code.

Note: In some of the class diagrams that we have drawn, we have not included all of the classes which we used in our final project. This is because there are just too many different attributes and methods for each class that plantUML would not render the diagrams properly. For example, in the sprites diagram we could only fit the classes: OnionStation, Pan, ChoppingBoard and Bin. There are many more sprites that we did not include such as: BunsStation, Chef, CompletedDishStation, LettuceStation, PlateStation, SteakStation, TomatoStation and worktop.

### SEQUENCE DIAGRAM

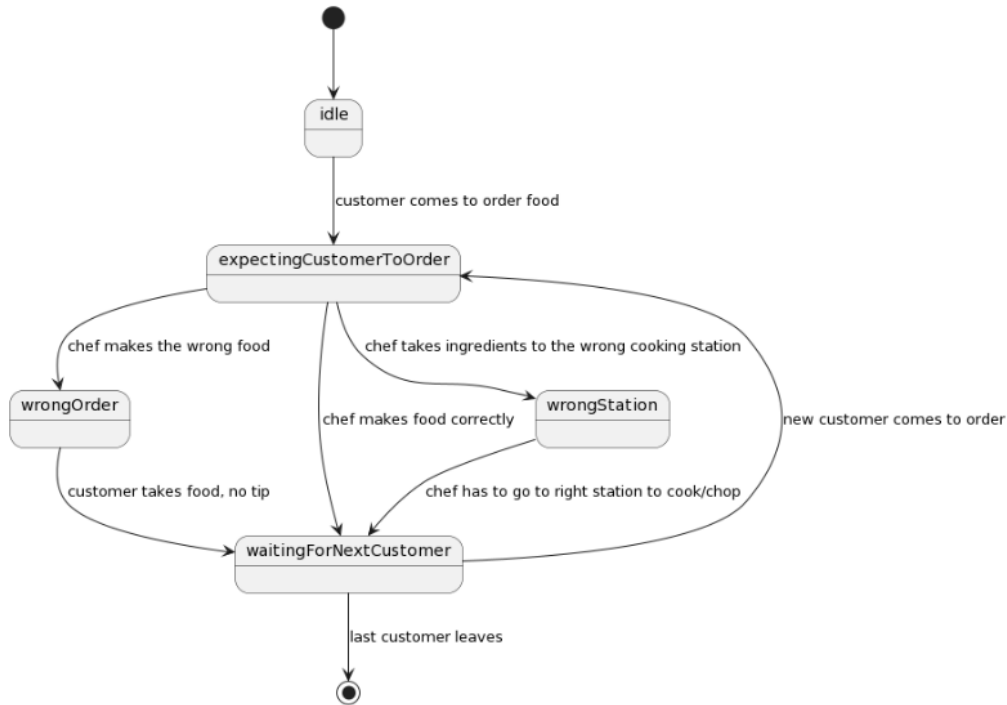
Sequence diagrams capture the interactions between entities in the system in a sequential order. The final sequence diagrams will show the interactions between the user and all of the different entities of the entire game. For example the interaction between the user controlling the chefs by making orders that have come in from customers. This will give a better understanding of the sequential order in which each action should take place.



The final version of the sequence diagram representing the whole of the game shows each and every interaction between the user and the characters and other entities in the game. This gives a better understanding of the sequential order of what happens at what time throughout the game. This diagram is made using plantUML.

### **STATE DIAGRAM**

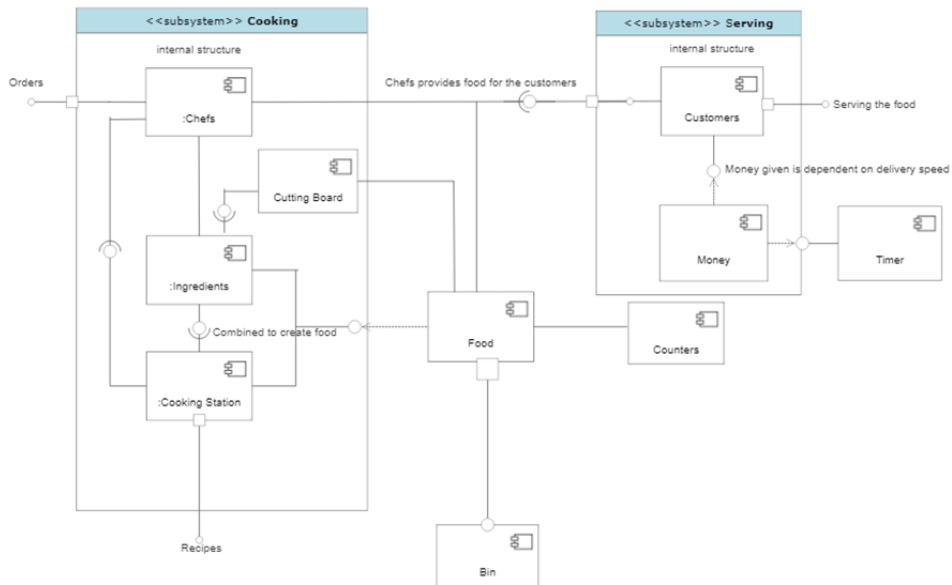
State diagrams are diagrams that capture the states of the system and the events that trigger transitions between them. For example, version 1 shows what happens when the chef makes the wrong order for the customer. The different versions of the state diagram below show the improvements that take place as we develop the game further.



The final version is made on plantUML, showing what would happen when a user controlling the chef makes a mistake during either the prep stage or the cooking stage. The diagram gives the paths of different stages that the user could take.

### **COMPONENT DIAGRAM**

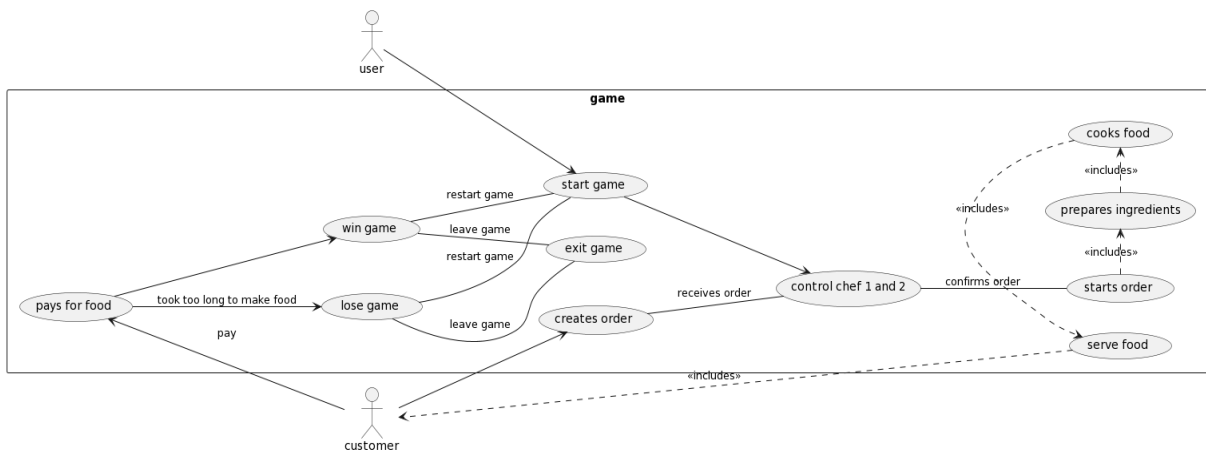
The aim of the component diagram is to visually illustrate how each of the individual components in a system connect and work together. It also helps with breaking down the whole system into smaller parts and therefore, better demonstrates their individual purpose.



The final version of the component diagram was created following the meeting with the customer, which is made using SmartDraw (web-based diagramming tool). Several additions were added to increase the accessibility of the game for those who need it.

### USE CASE DIAGRAM

Use-case diagrams model the behaviour of a system and help them capture the requirements for the system as well as identifying the interactions between the system and its actors. The main goal of a use case diagram is to describe a function that a system performs to achieve the users goals.



The final version of the use case diagram, made using plantUML, is a more improved and refined version with more interactions between use cases. This makes it easier to see the overall behaviour of the system. For example, in this version, the diagram includes a win/lose game use case. This gives a way for the user to restart the game for a second try or exit the game to the main menu.

## **Architecture - Class diagram, Sequence diagram, State diagram, Component diagram and Use-Case diagram**

### **Part B**

#### **CLASS DIAGRAM**

The first class diagram that we have created is the chef class diagram. This links together chef and sprites as within the chef class we have implemented, chef extends sprite, which shows the relationship between objects. This can be seen in the diagram by the hollow diamond arrow pointing from chef to sprite. Within the chef there are many different attributes, some original attributes and some added during the implementation stage to improve the code. However there are too many to physically be able to draw onto plantUML. These include `All attributes in chef`. The user requirements that are related to the chef class are UR\_CHEF, UR\_COLLISION and UR\_PLAYABLE. These requirements allow the game to run as efficiently as possible for the user.

The second diagram is the order class diagram. This extends order and recipe with sprite and then burgerRecipe and saladRecipe with recipe. This makes it much easier to implement as there all of the attributes are clearly labelled and ready to be used. In this diagram, there are no attributes or methods that are missing. Within our final projects code, we have used the exact same attributes and methods showing consistent naming of constructs. The functional system requirement for this class would be FR\_RECIPES which uses the user requirements UR\_RECIPES, UR\_PLAYABLE and UR\_CONVENTIONS to allow the system to show the user what they are currently making.

The third diagram is the Ingredients class diagram which includes the classes: Bun, Lettuce, Onion, Steak and Tomato. These are extended from Ingredients which extend sprite. This diagram contains many different methods and attributes for the class ingredients. All of which are needed in the final implementation. This makes it easier for us to see what each object does and what services they provide.

The fourth diagram is the HUD class diagram, this shows how WorldContactListener is linked to playScreen and playScreen is linked to MainGame and MainGame is linked to HUD. For some of these classes, there are too many attributes and methods that plantUML would not be able to create diagrams efficiently, such as playScreen. These include:

`All attributes in playScreen`. The user requirements that are linked to the HUD class diagram are UR\_PLAYABLE and UR\_CONVENTIONS. Meaning that the game would need to follow standard conventions so it can be playable and enjoyable for the users.

The last diagram is the sprite class diagram which shows the connections between interactiveTileObject, OnionStation, Pan, Bin and ChoppingBoard. All of which extends from interactiveTileObject. These are all related to the user requirements of UR\_KITCHEN\_UNIT which includes every workbench located inside the kitchen.

All of the class diagrams that we have created for this project are all implemented into the final program with the same names, attributes and methods, making it much easier to code as we already have a list of all the things that we need.

#### **SEQUENCE DIAGRAM**

The first version of the sequence diagram ( [Sequence Diagram v1](#) ) is to show the basic sequence of interactions between the user and the game. This version contained the user pressing different inputs to control the chefs allowing them to navigate through the game. These can be seen in the functional system requirements document that we have written, where the requirement ID is FR\_CONTROLS. This requirement is to make the controls as accessible and easy to understand for people who aren't familiar with video games. For example: the user would press WASD keys to control their chef to allow them to move up, down, left and right in the kitchen area. These are universal keys that are used for all purposes, therefore making the controls as easy as possible. Another key the user could press is R, to switch between chefs. This is also seen in the requirements as FR\_SWITCH, which allows the user to switch chefs easily. The last input key for this game is E, which is included in the requirements as FR\_INTERACT, meaning the system must let the user interact with the kitchen to navigate through the game.

Version two of this sequence diagram ( [Sequence Diagram v2](#) ) is a diagram containing the interactions between all the objects within the game. For example: the Customer, Kitchen, Order, Chef, Tip and TotalEarnings. These are included in the requirements as UR\_KITCHEN\_UNIT, UR\_CHEF and UR\_MONEY. UR\_KITCHEN\_UNIT includes all of the different functions within the kitchen, such as chopping boards, frying pans etc. UR\_CHEF is to make both chefs unique through the shape of their hat allowing users to know who they are controlling at each point in the game. Lastly UR\_MONEY is the money the user makes for each order based on the speed of completion. As the customer will wait indefinitely, the earnings for the order will be added to the tip to become the TotalEarnings. Version 2 and version 1 will be merged together to form the final version.

The final version of the sequence diagram that we have designed ( [Sequence Diagram v3](#) ) is a mix of both previous versions with slight additions, which include: Recipe, Ingredients and CookingStation. These are all classes that have been coded into the final implementation of the game. This version now shows all of the interactions between the user and the game, from beginning to end. For example: a user can control the chef around the kitchen by using the WASD keys, allowing the user to prepare and cook orders (from the customer), and then serving to the customer for a set price plus tip (TotalEarnings). This now gives a better understanding of the sequential order in which each action takes place.


## **STATE DIAGRAM**


Version 1 of the state diagram that we have drawn ( [State Diagram v1](#) ) captures the states of the user and chef and the events which trigger the transitions between them. In this case, the event was if the chef would correctly prepare and cook the food, depending on the order. Which would trigger transitions along the diagram, for example: if the chef makes the wrong order, the transition would take the leftmost route else it would take the centre route. However, as nothing can go wrong, the customer will still take the wrong order, but leave no tip.

The final version of our state diagram ( [State Diagram v2](#) ) is improved upon version 1. Now we have included more details on the events that can trigger transitions, which can be shown by "wrongStation". This now takes into account if the user controlling the chef takes the ingredients being prepared to the wrong cooking station, for example: taking steak to be


chopped at a chopping station (steak can only be cooked on the pans). This now gives an abstract description of the behaviour of the system. In the end, when the user has finished cooking and serving all of the customers, they will win the game. The user requirement used for this is UR\_WIN and the functional system requirement of FR\_FINISH. However, if the level is made too difficult, it risks the possibility that the level could be impossible to win.


## **COMPONENT DIAGRAM**

Version 1 of the component diagram (  Component diagram v1 ) only shows the basic components that are present. This includes the essential components needed for the game such as “chef” and “customer”. In component diagrams, several components combine together to form subsystems. In this case “chef”, “ingredients” and “cooking station” forms the subsystem “cooking”, which is an essential system in the game. These all relate to FR\_RECIPE functional system requirements as the system will show the user what recipes they are making and what ingredients they would need to use.

The final version of the diagram (  Component diagram v2 ) was created following the meeting with the customer. Several additions were made following the changes in the game. ‘Ingredients’ can now form an assembly with ‘Cutting Board’ and not just ‘Cooking Station’ this is due to the importance of following conventions and therefore, increasing the accessibility of the game for those who might need it. This follows the non functional system requirements of NFR\_ACCESSIBLE, as this feature adds accessibility features to improve the users experiences. ‘Bin’ has now become an actual component and no longer requires ‘Food’ to provide it the interface needed. ‘Timer’ was added so that the amount of ‘Money’ earned by the player is required to depend on the speed of the food served. This also acts as a timer to keep the game fast paced and to keep order completion times short, which uses the user requirements of UR\_TIME.

## **USE-CASE DIAGRAM**

Version 1 of the use case diagram (  Use-Case Diagram v1 ) shows the basic requirements of the system and the interactions between the system and its actors. The actor in this diagram is the user, who is able to start the game and control the chefs to navigate through the game. This shows the main user goals that the user needs to achieve when playing the game. The main user requirement used for this version was UR\_PLAYABLE, because for the game to be playable it requires different cases such as start game and exit game.

The final version that we have created (  Use-Case Diagram v2 ) is extended upon version 1, which includes a win case. This diagram can now show the next actions they can take after winning the game, for example: if the user wins the game they have an option to either start the game again to try and beat their score or exit the game. This is included in the user requirements as UR\_WIN and FR\_FINISH, where the game must be won when the user finishes serving all the customers. Starting the game again would traverse through the diagram from the start to the end, allowing us to see the overall behaviour of the system. This version of the use-case diagram also includes the <<include>> relationship which means that the base case explicitly incorporates the behaviour of another use case which shows further improvement from the first version of this diagram.