Architecture

Assessment 2

Group 5 :

Callum Watson

Jack Guan

Craig Slomski

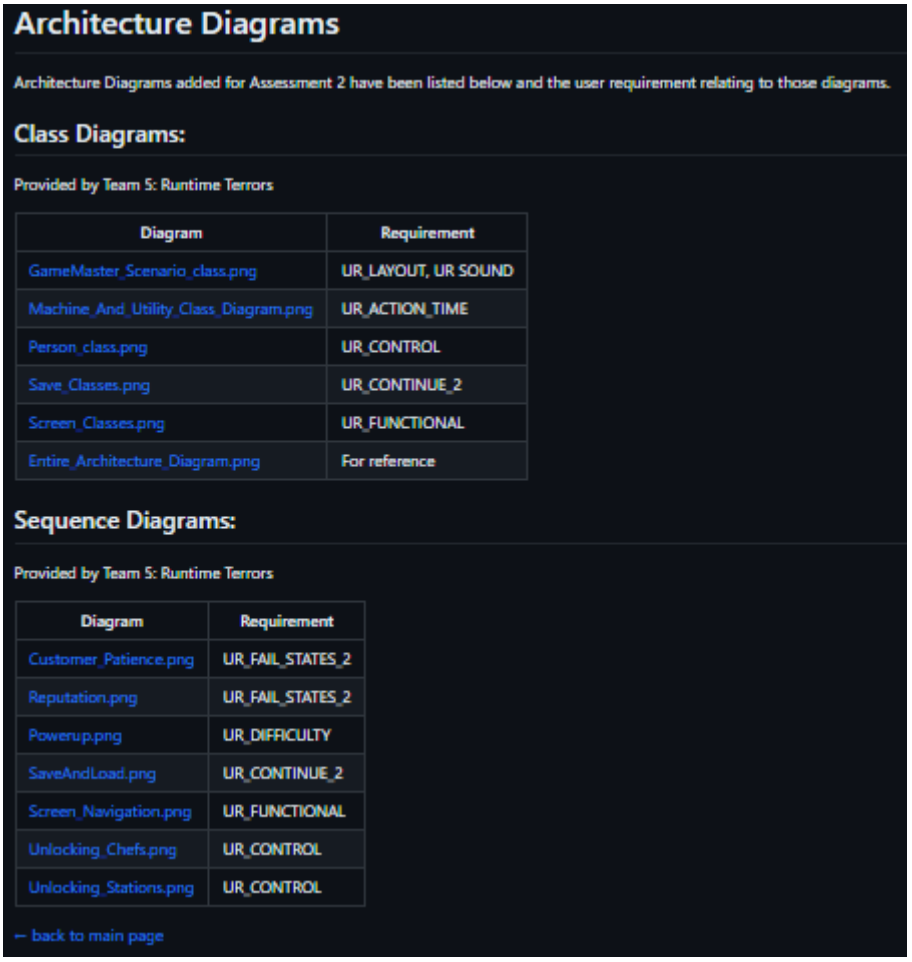
Chase Mo

Kamrul Islam

Yaseen Khan

Assessment 2 Architecture:

Following the new updated requirements, we have inherited the methods used here by Team 6 to create new diagrams to extend the later requirements. These can all be found on our website.



Architecture Diagrams

Architecture Diagrams added for Assessment 2 have been listed below and the user requirement relating to those diagrams.

Class Diagrams:

Provided by Team 5: Runtime Terrors

Diagram	Requirement
GameMaster_Scenario_class.png	UR_LAYOUT, UR SOUND
Machine_And_Utility_Class_Diagram.png	UR_ACTION_TIME
Person_class.png	UR_CONTROL
Save_Classes.png	UR_CONTINUE_2
Screen_Classes.png	UR_FUNCTIONAL
Entire_Architecture_Diagram.png	For reference

Sequence Diagrams:

Provided by Team 5: Runtime Terrors

Diagram	Requirement
Customer_Patience.png	UR_FAIL_STATES_2
Reputation.png	UR_FAIL_STATES_2
Powerup.png	UR_DIFFICULTY
SaveAndLoad.png	UR_CONTINUE_2
Screen_Navigation.png	UR_FUNCTIONAL
Unlocking_Chefs.png	UR_CONTROL
Unlocking_Stations.png	UR_CONTROL

← back to main page

Justification:

We have extended the existing methods used by the previous team as it allowed them to create effective Sequence and Class UML Diagrams to represent the range of user requirements presented to them. We aim to replicate this by introducing new UML diagrams to represent those features and clearly link them to the related requirements.

We wanted to trail the generated UML diagram post development in order to display the similarities and differences in the code base and show the growth and development of the project overtime.

We did however, opt for PlantUML to create both Sequence and Class diagrams, whereas the previous team used a mixture of sequencediagram.org and PlantUML. This is because our team was already familiar with PlantUML and can efficiently create both Sequence and Class diagrams, therefore there was no need to learn additional software tools.

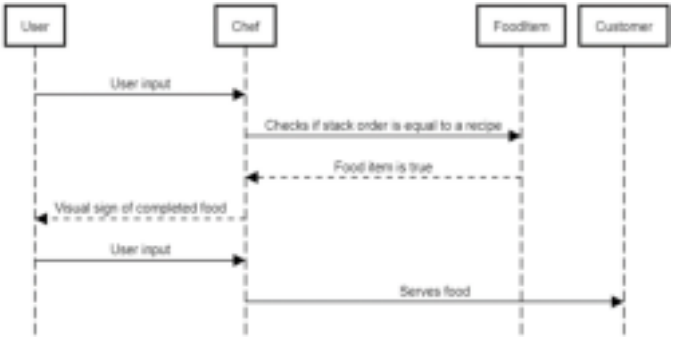
Additional CRC Cards were not needed as the new features (such as adding a new station) are already represented by the existing CRC Cards.

Assessment 1 Architecture:

For the architecture section of this project we are opting to use an iterative process for designing and advancing the architecture of the game. We will be using a mixture of visual methods to create the architecture such as Sequence diagrams and UML diagrams. We aim to create multiple versions of the architecture diagrams as we progress through the planning and development of the project in order to show the discussion and growth of our project.

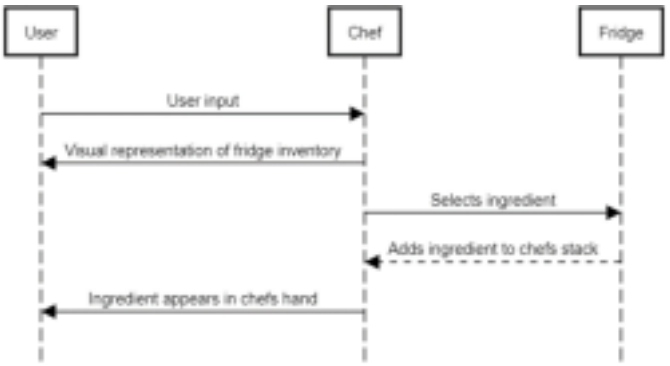
Sequence Diagrams

To get a grasp on how users will interact with the system and more specifically how different classes will interact with each other we developed a set of sequence diagrams. A sequence diagram is a method of showing how classes and actors will interact so that when we begin to implement not only do we have an idea of how the classes should be constructed from the UML diagram but we will know how these classes function together using the sequence diagrams which will also help us to develop and improve the UML diagram. We used a web tool on the website sequencediagram.org to develop the sequence diagrams.

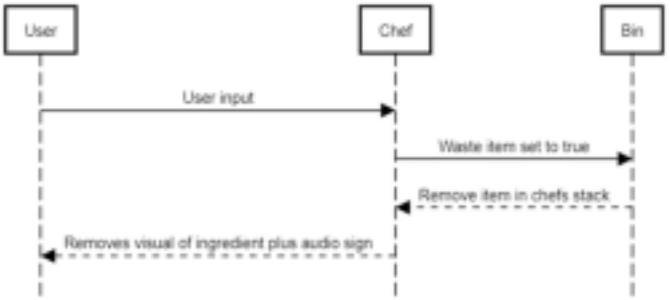


The above sequence diagram uses User, Chef, FoodItem and Customer. The diagram represents if a user is holding a correct recipe and then serving that food. On a step by step basis, the user will input which interacts with the chef which then goes ahead and checks if the stack of food the chef is holding is a valid recipe, it checks this against the FoodItem. If it is a valid recipe it returns as true to the chef and displays a visual signal showing a food item has been created. Then to serve the user will input to serve the food item to the customer.

The above two diagrams represent how a user will interact with the grill to cook and the chopping station to chop. The user will input which is received by the chef, if the chef is in the correct place/next to the grill or chopping station then it will set the grilling/chopping value to true on the chopping station or grill unit, this in response displays a visual sign that the food is being grilled. When the grilling/chopping value is set to true it will begin a timer, once this timer is complete the value is set back to false implying that grilling/chopping has completed and the food is cooked. This is shown to the user using a visual and audio queue.



The diagram shown above shows that the user will get the ingredients from the ‘fridge’. The user will input to the chef and return a visual representation of the fridges inventory to the user. The ingredient is then selected from the fridge. It is then added to the chef's stack/what it's holding; this will then be shown visually by displaying it in the chef's hand.



The diagram shown above represents how a user will discard an item. The user will input to the chef, if the chef is in the correct position, next to the bin, then the waste item is set to true. This will remove the item from the stack and will remove the visual representation of the food in the hand and will also return an audio queue to show the user it has been wasted.

Using these sequence diagrams we can craft a UML diagram. We can devise this diagram using the actors/objects in the diagrams e.g user, chef, etc. and how they interact to give these classes the necessary variables and functions they require to implement the sequences in the diagrams.

CRC Cards

After creating the above set of sequence diagrams we moved onto making CRC cards. Class, Responsibilities, Collaborators cards is a method of pre planning classes, what responsibilities the classes will have and how they will collaborate with other classes. It is a great stepping stone between the sequence diagrams and the UML diagrams as it allows the actions shown in the sequence diagrams to be condensed into more specific classes that can then be used as the basis for the UML diagram. After some discussion the below CRC cards are what we devised.

Character/Player	
Move around map	Unit
Switch Chef	Food
Interact with units	

Unit	
Grill food	Food
Chop food	Character/Player
Store ingredients	
Discard ingredients	Bake Food

Food	
Recipes	Unit
Ingredients	

The above CRC cards describe the key classes of Food, Character and Unit. They can all be seen being used in the sequence diagrams which is where we based the ideas for these classes around. In them we describe its key responsibilities/assets and what other classes they related to. From here now we can go ahead and begin developing a UML diagram that will allow us to see the overall structure and how all these classes will weave together.

UML Diagrams

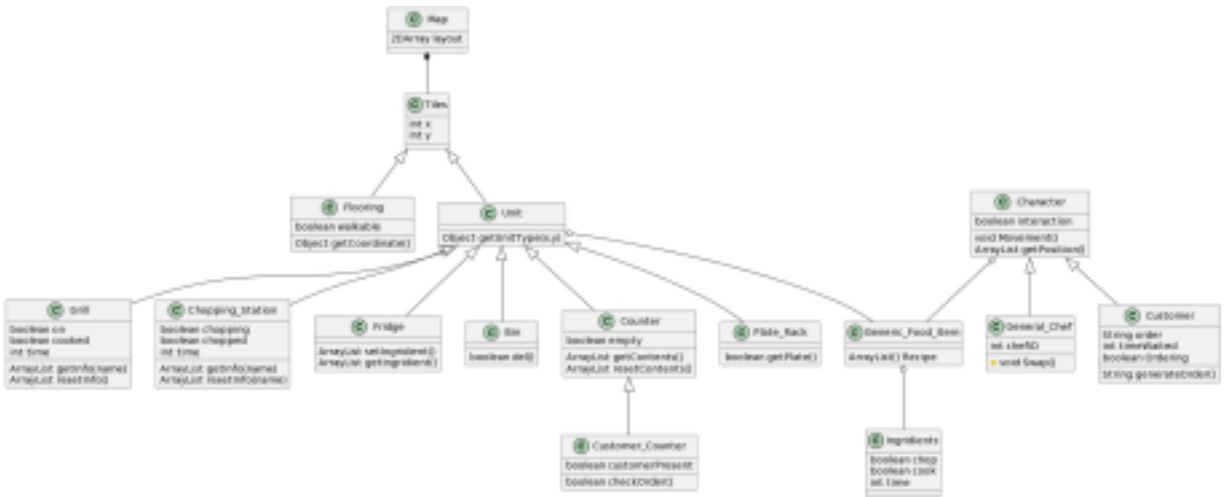
We used the webtool found on plantuml.com to create the UML diagrams. We also used MS Paint to highlight any changes to be made in the UML iterations as seen by the yellow highlights on some of the diagrams below.

For our first draft of the UML diagram we just wanted to get the fundamentals basics of the codes architecture down these were the basic classes of:

- Map
- Tiles
- Unit
- Generic_Food_Item
- Character (later changed to mob)

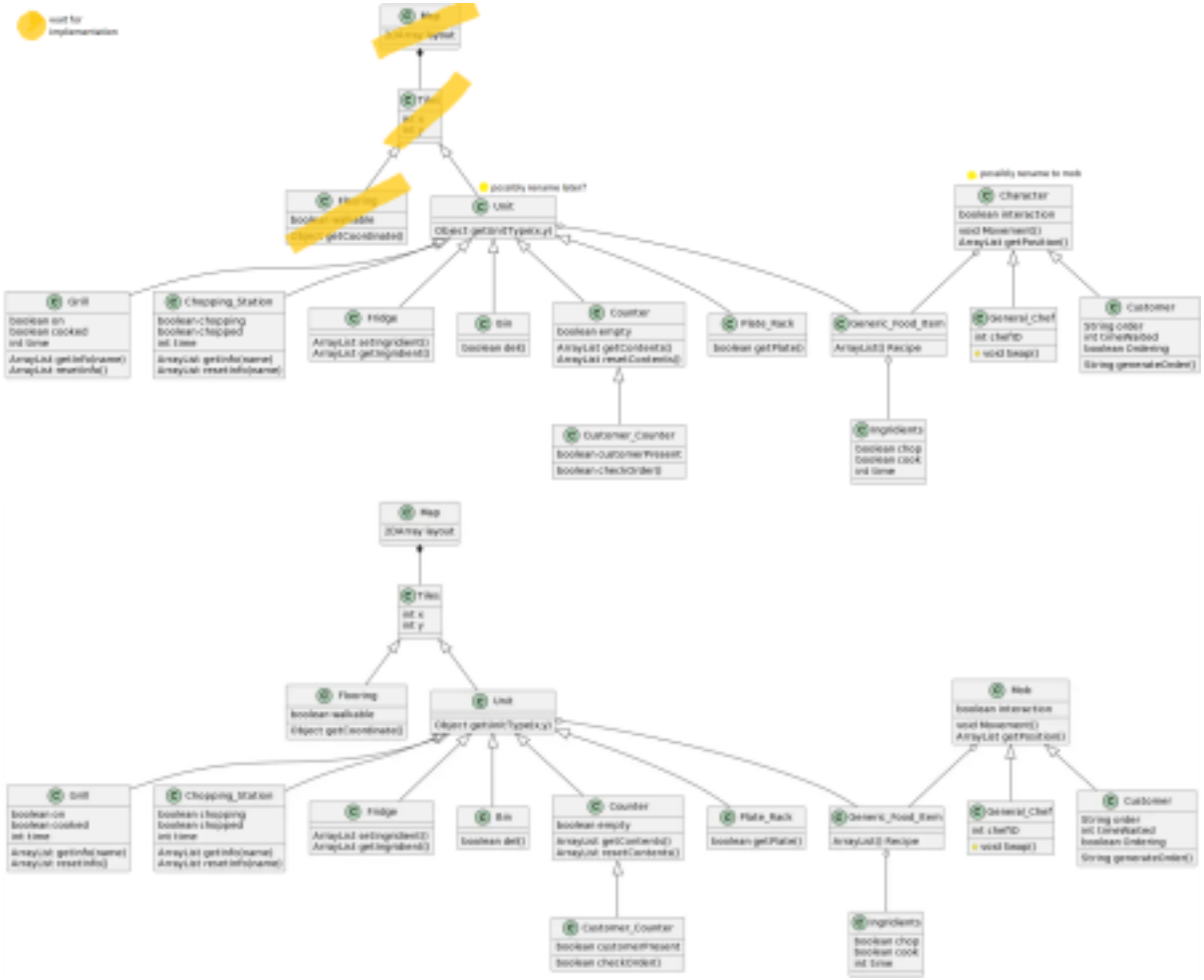
All with their respective subclasses as shown in our first draft below:

With the next draft we wanted to change to architecture to be a little more cohesive and modular while maintaining the same idea for our original first draft. In addition to this we also made sure to add the methods and fields to each class where necessary giving the second draft of our UML shown below.



Our final draft before implementation was just to change a few names of classes such as making characters to mob and having the idea that the diagram may change after implementation where our final draft of the UML diagram would be done. Here are the

following small edits we wanted to make followed by them in action.



As you can see there were very minimal changes and it was just done as a more functional change than anything else so that there would be as little changes needed as possible for the final UML diagram after the classes had been implemented.

The final UML diagram was made during the implementation phase rather than the other diagrams which were made pre-development. The reason for this is that the IDE we are using for development (IntelliJ IDEA Ultimate) has a built in toll to construct a UML diagram complete with attributes and methods for each class allowing us to more easily create the UML diagram as well as have it closely reflect how the actual implementation looks.

Between the previous version of the UML diagram and the final version shown above a lot has been altered. Initially when the architecture was being crafted pre-implementation we opted for a largely interconnected set of classes with a lot of classes extending from other classes. When we came to the creation of the final UML diagram we simply noticed in implementation that it was not necessary to have this level of complexity. If we had implemented it the way we had initially devised it would have been accomplishable however it would have resulted in a much more convoluted and bloated project with a lot of classes connecting to so many other ones. It would have also made debugging a lot more challenging. As a result we came to this much more disconnected diagram. It still contains a lot of similar classes and some have been condensed down. One thing we didn't initially consider was the game/screen classes that are necessary for the mechanics of the game to function properly so as a result they have been included here.

Overall as you can see from the sequence diagrams to the first UML diagram all the way to the final UML diagram the architecture changed a fair bit. Its evolution was due to many factors such as discussion as a group, considering implementation, considering the way the user interacts with the system and by observing what requirements would be necessary from the system. This iterative evolution happened over a number of weeks and allowed us to view the project dynamically changing as we discussed further.

Relation to requirements / Systematic justification:

- UR_DISCARD: “Allows players to discard ingredients when they go wrong in the middle of the recipe.” - In the final UML diagram you can see there is a bin class under units that is to delete items in the chef’s inventory if something is done incorrectly.
- UR_RECIPE: “List of recipes to be displayed on screen and taken off screen when the player has completed them. The priority of the recipes needs to be specified. Each recipe should be limited to three ingredients.” - On the UML diagram you can see the use of Generic_Food_Item which allows the creation and display of what items are in the inventory, the priority of the orders, etc as they can be called and displayed.
- UR_ACTION_TIME: “Timers should be displayed above work stations indicating the time left for the current action, i.e. cooking the dish, indicating if it's overcooked” - In the UML the ScenarioGameMaster class handles the active timer on the chef for when the chef is performing an action as suggested by the requirement
- FR_CONTROL_MOVEMENT: “The system should allow the player chiefs to move on screen with the WASD keys.” - The Person class shown in the final UML diagram is what overall dictates and controls the players movements using the specific keys stated in the requirement.
- FR_CONTROL_ACTION_KEY: “The action key to switch between chefs is 1,2,3 and the action key to grab , work on and discard an item is E key.” - The Person class shown in the final UML diagram is what overall dictates and controls the switching of the chefs and how they interact with the map as described in the requirement and it uses the specific key that the requirements asks for
- FR_ITEM_REMOVAL: “The system must allow the player to remove items from the game if they accidentally ruin an order or want to discard an item in the trash station using the action key E. This item needs to be removed from the playable area” - In the final UML the bin class under the units class is there to allow a chef to discard their inventory when something has been done incorrectly