Architecture Document

Group 28

Piazza Panic

By OuseWorks

Ben Harris
Joshua Gill
Niamh Hanratty
Amy Raymond
Matthew Czyzewski
Matt Rohatynskyj

Part A
Each of the diagrams are listed below and are found on our website here: [insert url for architecture page].

Figure 1: CRC diagram

Figure 2: Class Diagram

Figure 3: Entity-Component Diagrams

Figure 4: Sequence Diagram

Figure 5: State Diagram

- Our structural diagrams are our class and entity-component diagrams and our behavioural diagrams are our sequence and state diagrams.
- Our UML [1] structural diagrams are useful for showing the high-level decomposition of our game; they focus on the composition of entities and systems rather than showing the relationships between them.
- The behavioural diagrams focus on the communication between system states and show how the game will play out, helping with our understanding of the implementation.
- We were able to effectively grain our components without the model becoming too fine-grained making the whole system highly coupled.
- We used PlantUML [2] to make these diagrams, as it facilitates creating efficient, maintainable code which helps keep development simple for everyone on the team.

Part B

Class-Responsibility-Collaborator Diagram

The first architecture we made was our Class-responsibility-collaborator (CRC) diagram (*Figure 1)*. We brainstormed all the classes we needed for our implementation and made a rough note of them on cards. These cards focus on what the classes do (their responsibilities) and which other classes they communicate with (collaborators). As you can see between *Figure 1* and *Figure 3 v2*, we decided not to keep all of these as classes but instead changed them to instances of classes. For example, our CRC diagrams included separate classes for each of the stations, which we changed to one 'Station' class in our entity diagram, with 'StationType' as a component. This fulfils our requirements FR_COOKING_STATIONS and FR_INGREDIENT_STATION.

There are other specific classes that we decided to include, in order to meet our requirements. One of these is the 'Timer' class which would help implement the functional requirements FR_COMPLETE_GAMEMODE and FR_TIMER as the brief says the system should inform the user how long it took them to complete the game. The brief also includes reputation points which we made a user requirement, FR_ REPUTATION. We have made sure to include this in our CRC cards as a gameplay class.

The CRC cards didn't evolve much as they are only the first view of our architecture. Instead, we concentrated on arranging and grouping together classes which share functionalities or features, adding any classes we missed out as we went.

Class Diagram

We created a class diagram *(Figure 2)* as the first structural UML diagram that we planned on making as it gave us an overview of how the whole system's content would link together.

The main structure of the game involves a Game class which manages all scenes in the game and some extra details. This fulfils the user requirement UR_INTERFACE and the functional requirement FR_MAIN_PAGE which covers all of the game interfaces. The active scene is updated regularly, while the others are not (this allows the game to be paused by selecting the PauseScreen, stopping GameScreen from being updated). However, we didn't have time to implement the PauseScreen. The GameScreen contains the player object, as well as lists of all stations and customers. The player, the stations, and the customers are all entities, and possess classes that control their characteristics. Examples of classes demonstrating the entity-component system include:

- Position
- Sprite
- Collision
- Input.

Our implementation differs from this class diagram: food entities were changed to entity types. We also decided it wasn't necessary to add both the help page and the tutorial as only one is needed to satisfy the user requirement (UR_UX) they are associated with.

Entity-Component Diagrams

From our initial class diagram, we thought it necessary to create a more specific Entity-Component diagram *(figure 3)* as this is more focused towards the game content rather than functionality. We originally made two separate diagrams, 'Figure 3 v1 edible' and 'Figure 3 v1 non-edible', found on the website. The UML diagram for the edible entities diagram consists of the ingredients and dishes used throughout the gameplay. Without these

diagrams, our initial class diagram*(Figure 2)* would be incomplete, as the Ingredient class is at the core of the game brief. Furthermore, it is helpful for future development purposes if there is a need to add more dishes or ingredients as the diagram clearly states the components which are needed by the entities. Components found in this diagram cover the different variables which we have used during development and in the implementation i.e. image png is the component which will contain the sprite image used for an entity. We then decided that it would be more appropriate to combine the two diagrams and create two packages within the diagram *(figure 3 v2)*, as components like 'Sprite' and 'Position' appeared in both.

The entity-component diagrams fulfil many of our requirements. The customer class has an inventory; this will be compared to their order and the customer will wait until their inventory is equivalent to the order, which implements the functional requirement FR_INFINITE_WAIT.

When implementing, we changed all of the food entities to entity types instead as this was easier to implement. We also added chopped and cooked versions of the required foods so we knew whether they were ready to add to the completed dish. This is shown on *Figure 3 v3.* In the final version of the entity-component diagram, we have changed the entity 'Station' into three different entity types: 'Cooker', 'PreparationStation' and 'IngredientStation'. This is instead of having 'StationType' which was a component of 'Station' in *Figure 3 v2.*

Sequence Diagrams

It was important to create a sequence diagram as it was the first behavioural architecture we made so it gave us a good sense of how the gameplay would roughly play out. Our initial sequence diagram *(Figure 4 v1)* shows the basic flow of how actors interact with participants in the game, however it is flawed as it only shows the cook preparing one ingredient. This made us discuss in more detail how the system would handle the cook preparing multiple ingredients before giving them all to the prep station to assemble the completed dish, therefore introducing the idea of loops within our updated sequence diagram *(Figure4 v2)*.

In making this diagram, we were able to decide what the functionality of each station was, which would heavily aid in our implementation. This helped us fulfil the functional requirement FR_INGREDIENT_STATION, which is where the cook will pick up the ingredients from. Our functional requirement FR_COOKING_STATIONS has been adapted to be a single cooking station, where patties are fried, and a prep station, where ingredients can be chopped and the completed dish is assembled.

As this sequence diagram shows the functionalities of the ordering system, many functional requirements regarding the customers are met. The main section, in the second version of this diagram, loops five times, as there are five customers (FR_SERVE_5) which each arrive one at a time (FR_ARRIVE_ALONE). The customer then places their order at the counter (FR_SERVE_COUNTER) and therefore all of these fulfil the user requirement UR_SCENARIO_GAMEMODE, which sums up assessment one.

An aspect of the brief that this diagram fails to capture is that the player can switch between two cooks, shown by requirement FR_MULTIPLE_COOKS. However, we believe that rather than being a part of the sequential system, it is covered in the states of the system.

State Diagrams

Our state diagram *(Figure 5)* follows on from our sequence diagram and showcases the interactions between classes in more detail. It shows what the system expects from the player and how the system responds to different choices made. The state diagram follows through the game from the idle screen to the end of the game. It presents how features like

switching between cooks would work, which fulfils our functional requirement FR_MULTIPLE_COOKS. The state diagram shows links between requirements, for example when the player uses the keyboard and mouse (FR_CONTROLS) it leads to the cook being controlled (UR_CONTROL_COOK).

We used forks and joins in the diagram to show how entities change state (e.g. the ingredient ‘is done’, ‘needs chopping’ or needs ‘cooking’) and shows that only the expected inputs and decisions will actually have an impact on the system (NFR_UNEXPECTED_INPUTS). While the state diagram doesn’t explicitly show that we will fulfil the requirements NFR_IDLE and FR_STANDBY, which suggests that the game will go back to the main page when left idle for more than 1 minute, we have shown that it will just remain in the idle state until some input is made from the user. We ended up not implementing the standby functionality, the only time it shows the idle state is at the start of the game.

This diagram is really important for our implementation as while programming the game, we kept looking back at the diagram if we were unsure on things; for example, whether the completed dish gets assembled at the pre station or the counter. During our implementation, we realised some parts of this state diagram won’t be implemented in the game, these were updated on Figure 5 v2. These include the fact that the cook can take and not give ingredients from the ingredient station. We also updated which inputs the player will need to use to play the game, which are as follows:

- Ctrl key - switches cooks
- Spacebar - interacts with entity types
- M key - mutes background music
- Mouse click - to select ingredient

References:

[1] Richards, M. and Ford, N. (2020) *Fundamentals of Software Architecture: An Engineering Approach*. Beijing: O'Reilly.

[2] *Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams.* (no date) *PlantUML.com*. Available at: https://plantuml.com/.