Architecture

Group 1
Assessment 2

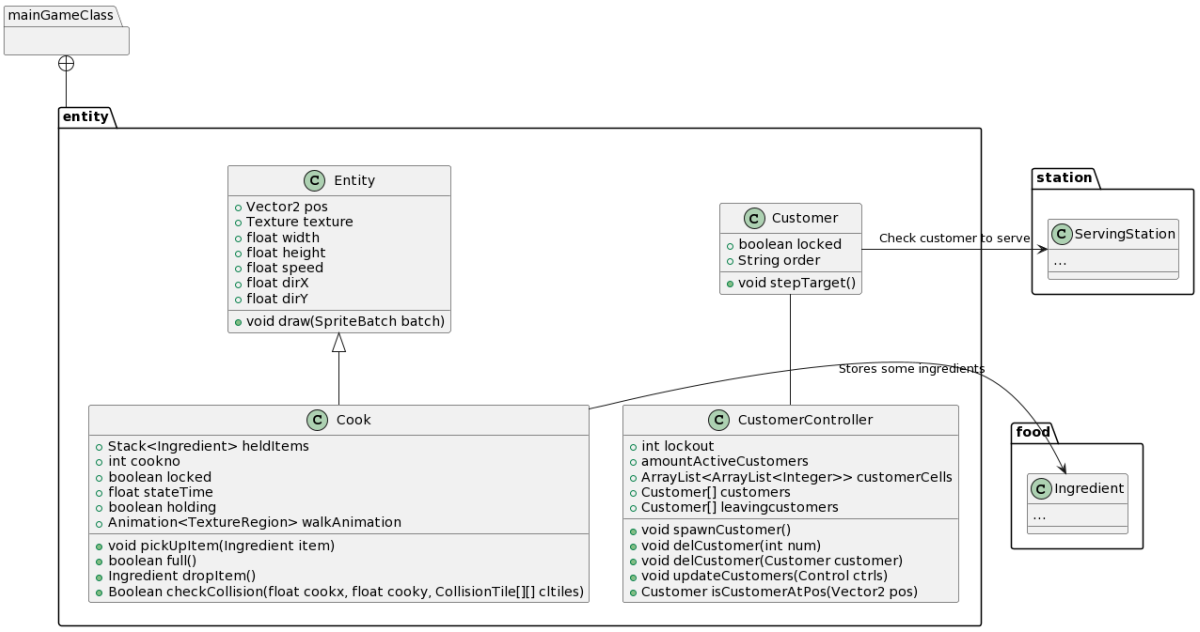
Emma Hogg
Isaac Rhodes
Isioma Offokansi
Toby Meehan
Seungyoon Lee
Ali Yildirim

Architecture Diagrams

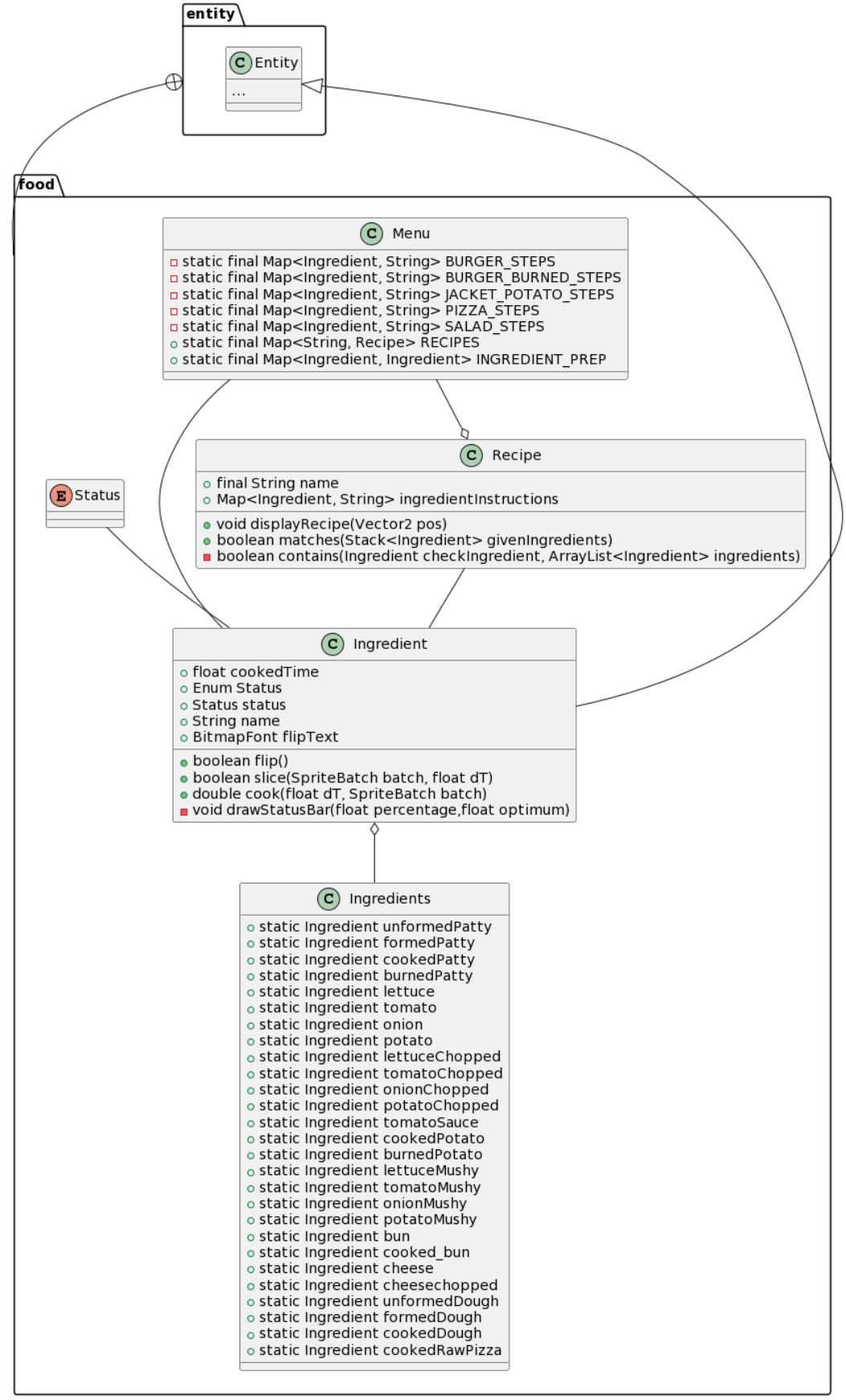
All diagrams were made using the PlantUML extension in VSCode. We used the extension because the PlantUML language produces clear diagrams with a large variety of symbols so we could easily get our intentions across to anyone viewing the diagrams. We found that
VSCode was the best editor for PlantUML because it was widely used by our group and had many features that would streamline the creation of our diagrams.
Higher quality images of the diagrams can be found on our website.

Class Diagrams

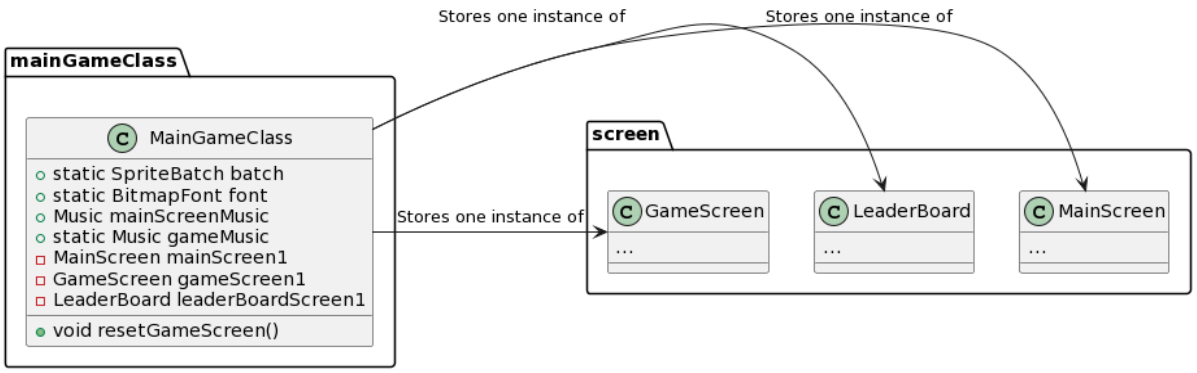
A full image of our class diagram can be found on our website.
entity package



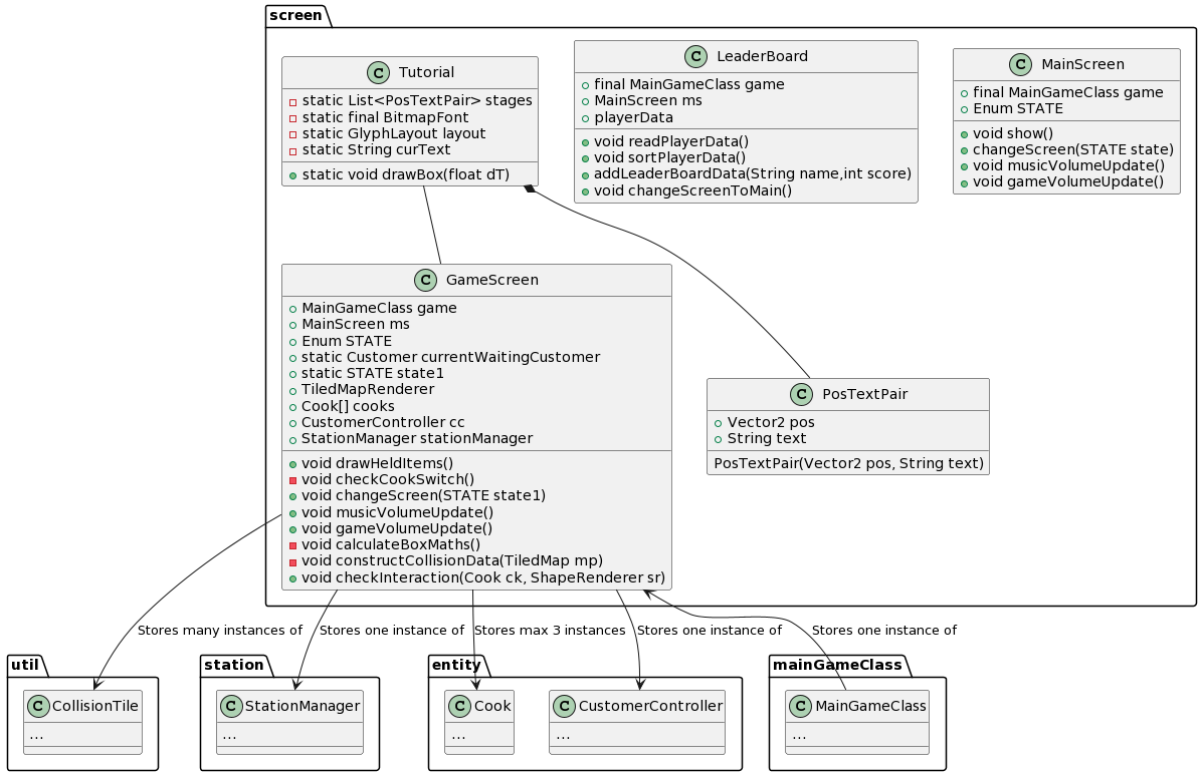
food package



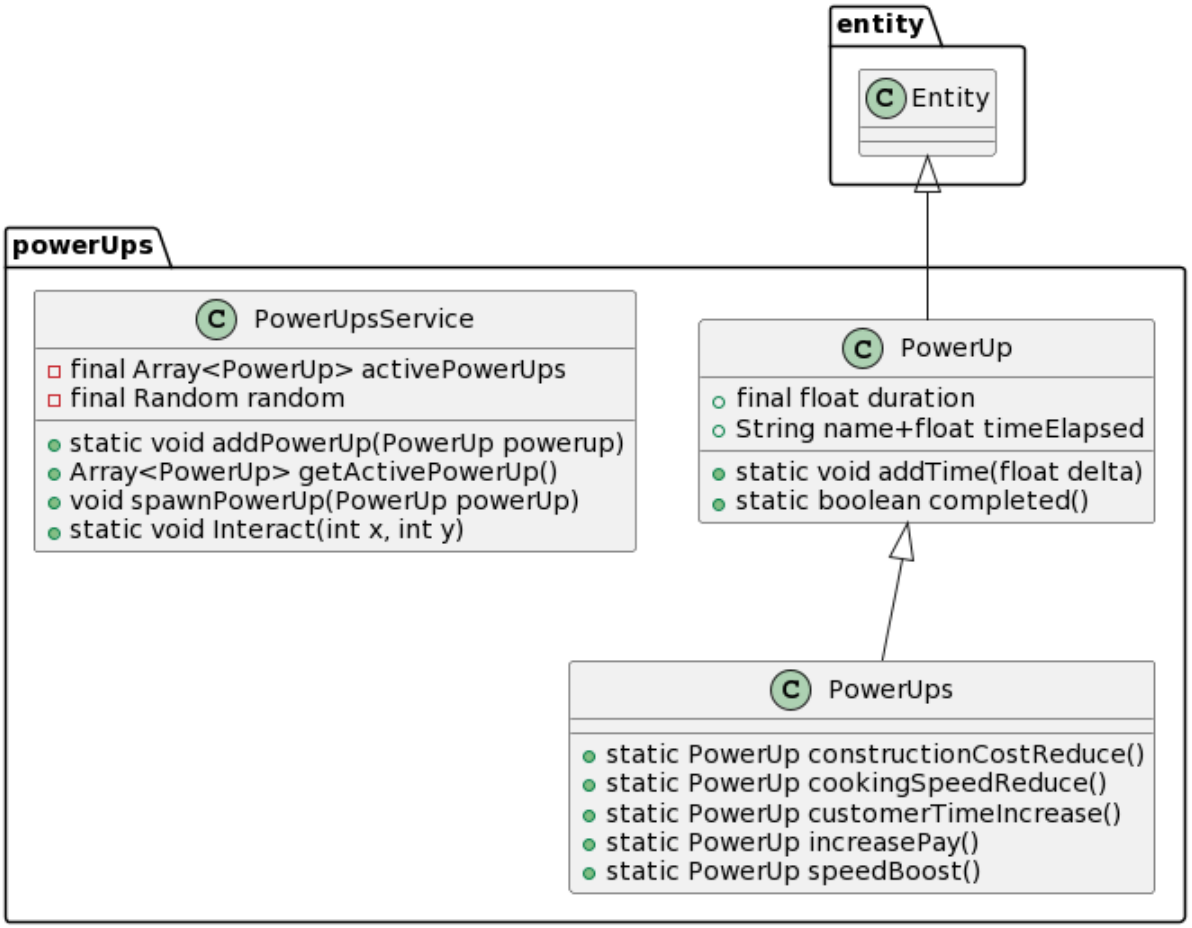
mainGameClass package



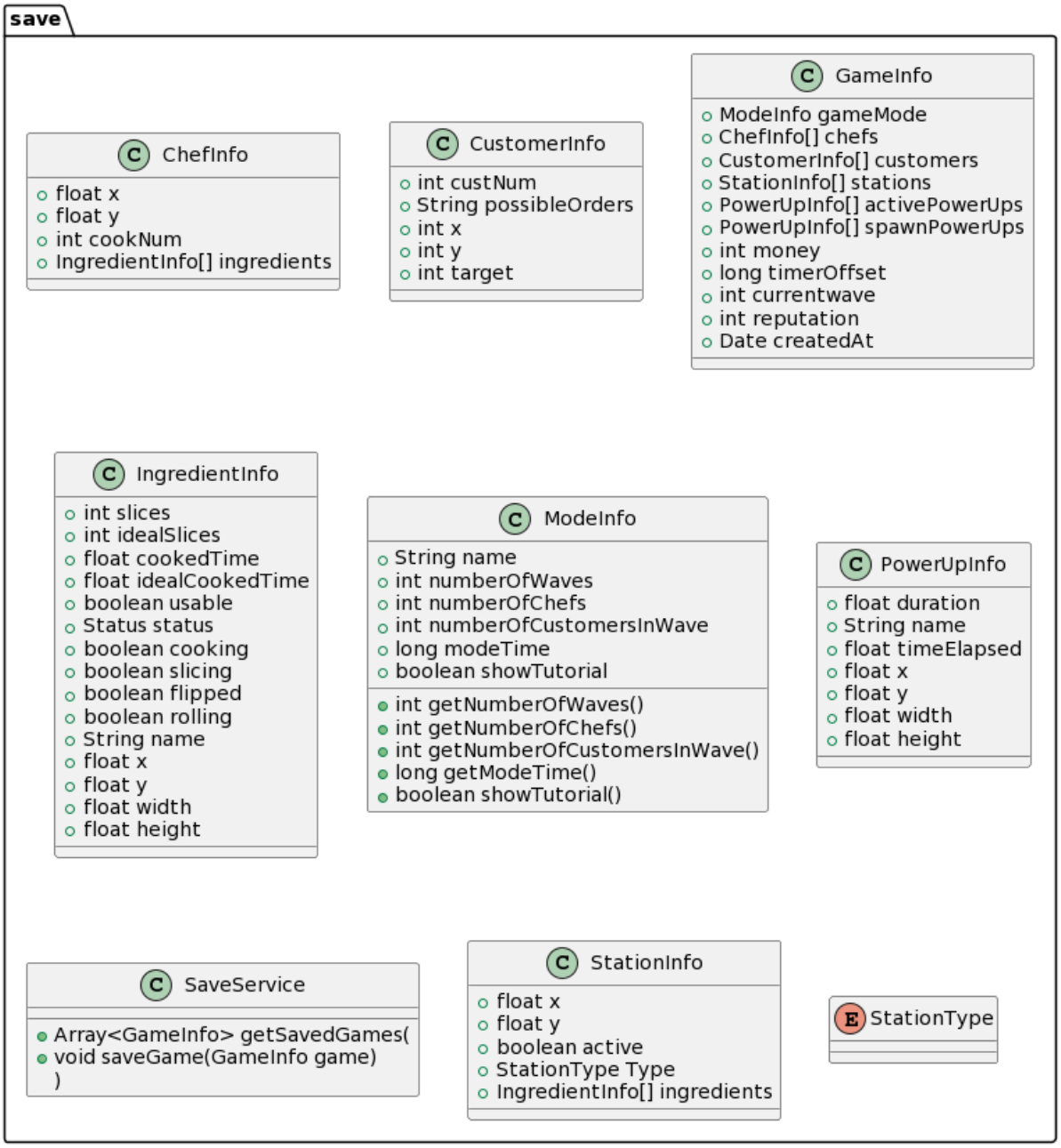
screen package



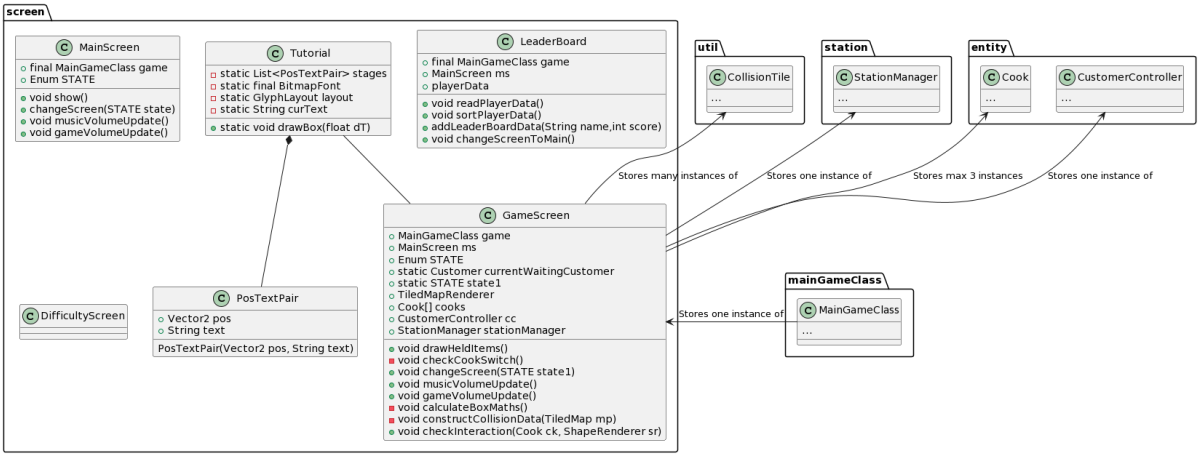
powerUps package



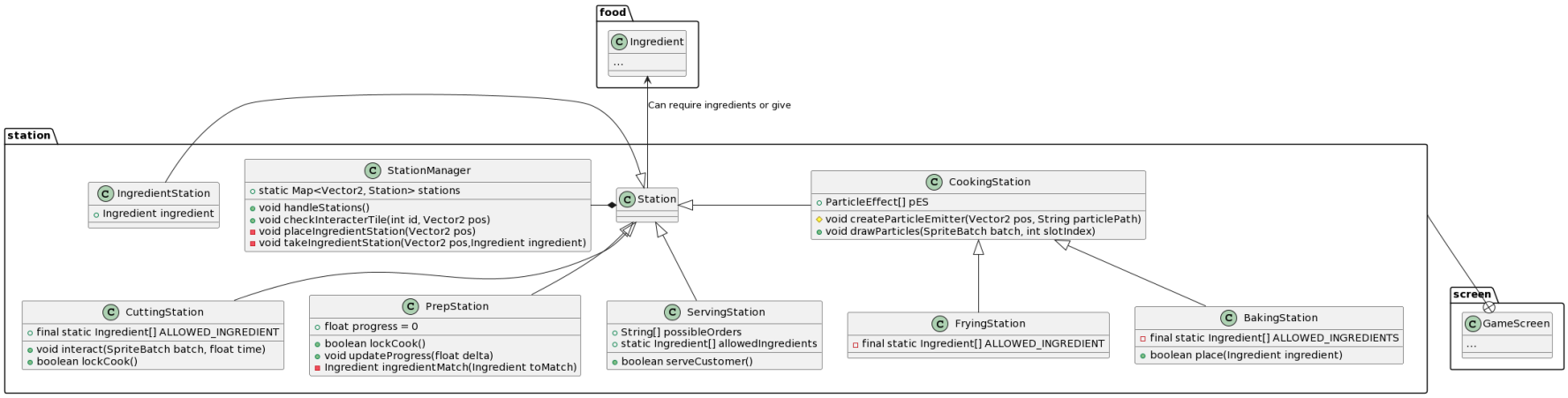
save package



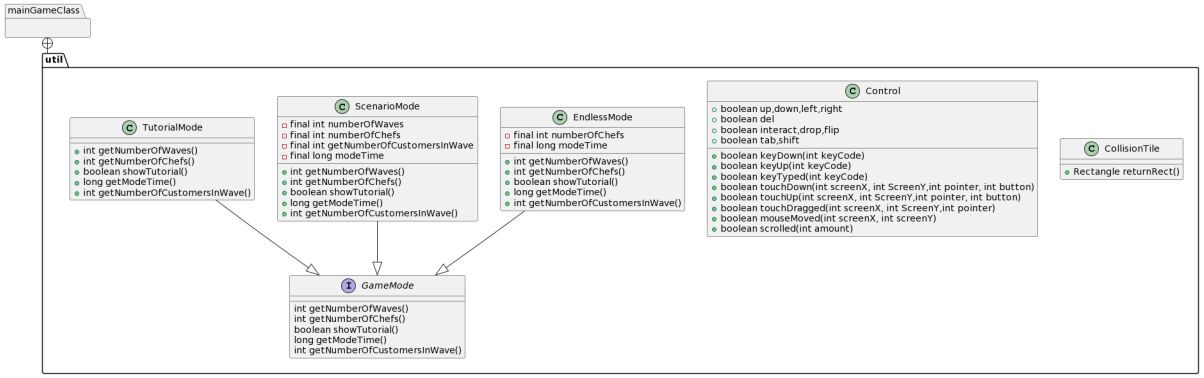
screen package



station package

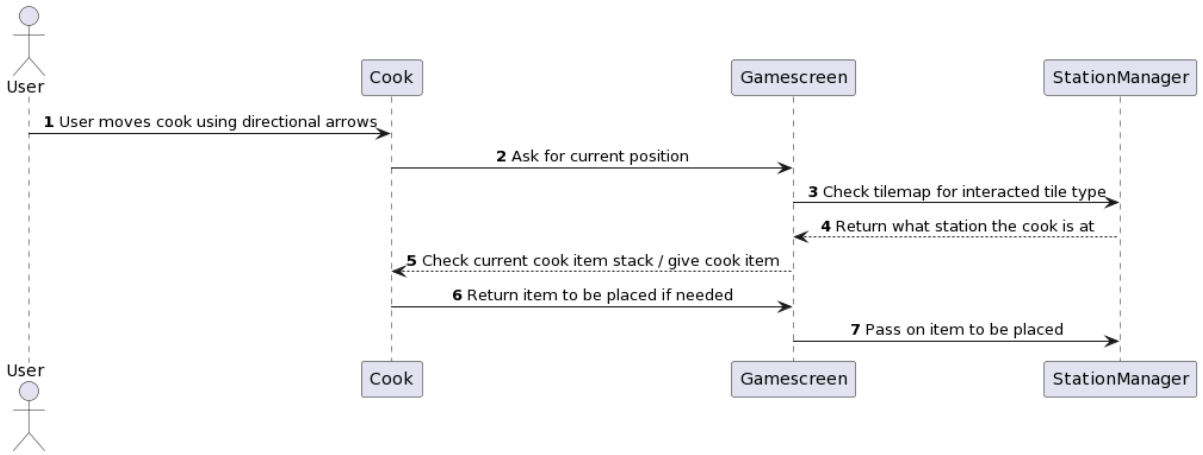


util package

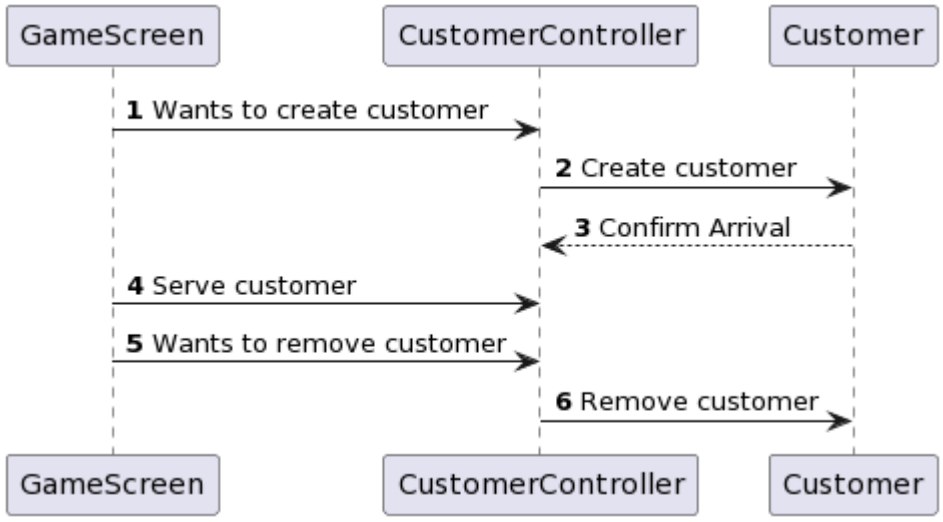


Sequence Diagrams

PlantUML sequence diagram for player interactions

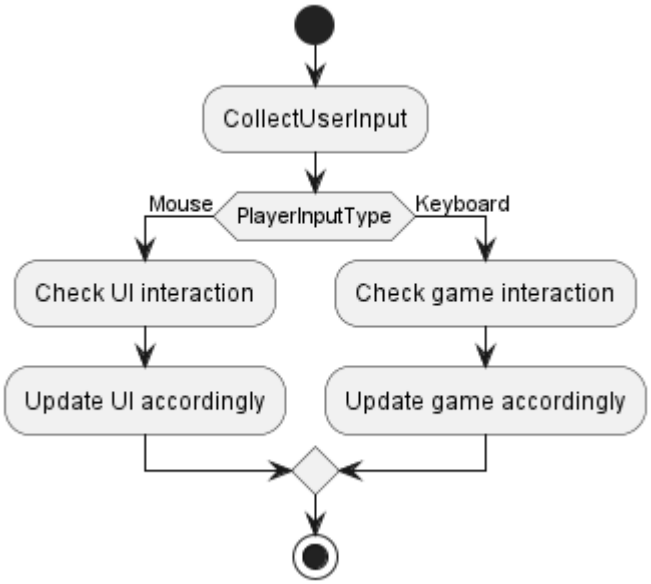


PlantUML sequence diagram for customer interactions



Activity Diagram

PlantUML Activity diagram for user input handling


CollectUserInput
Mouse
PlayerInputType
Keyboard
Check UI interaction
Check game interaction
Update UI accordingly
Update game accordingly

Architecture Justification

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 under the architecture section on our website).

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.

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

ID	Description
UR_SCENARIO_MODE	Customers will arrive one-by-one or in groups depending on which difficulty is selected. The game will finish when all have been served (currentWave >= MAX_WAVE).
UR_ENDLESS_MODE	Customers will arrive in groups that vary in size. The game ends when all reputation points are lost.
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_COOKS	Two instances of the cook class shall be created and rendered with a current cook being controlled directly.
UR_REPUTATION_POINTS	Game begins with 3 reputation points. Reputation points are lost when an order is not completed in time. The game is over if all three reputation points are lost.
UR_MONEY	Successfully fulfilling customer orders rewards should add the cost for the order into a variable storing the total amount of money.
UR_BUY_ITEMS	Additional stations and cooks should be added to the game when purchased using in-game currency.
UR_TOOLTIP	When approaching an intractable station, the appropriate key will be shown (e.g. ‘f’ to flip)..
UR_LEADERBOARD	When the game has been finished, the time will be uploaded to a leaderboard and sorted through that class.
UR_MINIGAME	Preparing food using stations should display a bar that shows the progress. Food will need to be removed in a set timeframe to avoid burning/over-chopping.
UR_PREPARATION_FAIL	Failing in the minigame should produce an unusable item
UR_POWER_UPS	Spawn randomly in areas accessible to the cooks. They should impact gameplay in some way (greater movement speed, cheaper construction cost, etc.)
UR_SAVE	Completing scenario or endless mode should save the game. Information that is saved should be: leaderboard information, money, what items have been purchased etc.