Architecture

Group Number: 10

Team Name: Decassociation

Group Member Names:

Mohammad Abdullah

Tom Broadbent

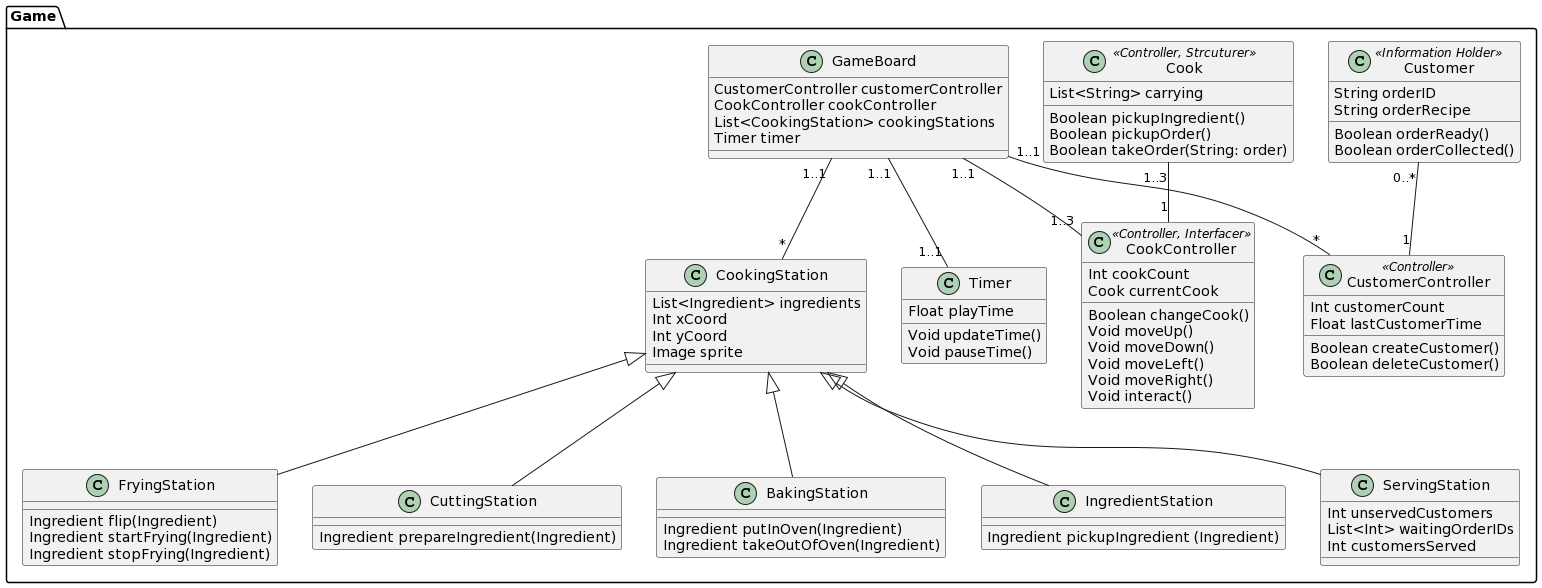
Poppy Fynes

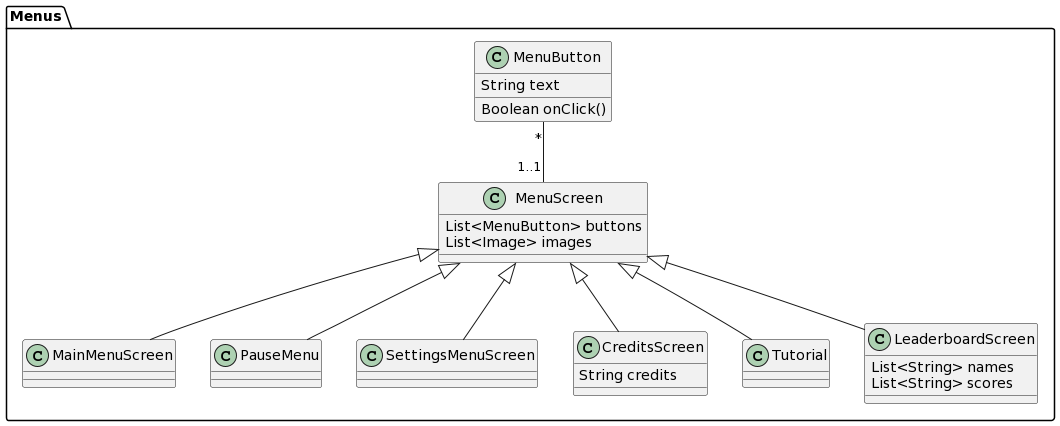
Owen Lister

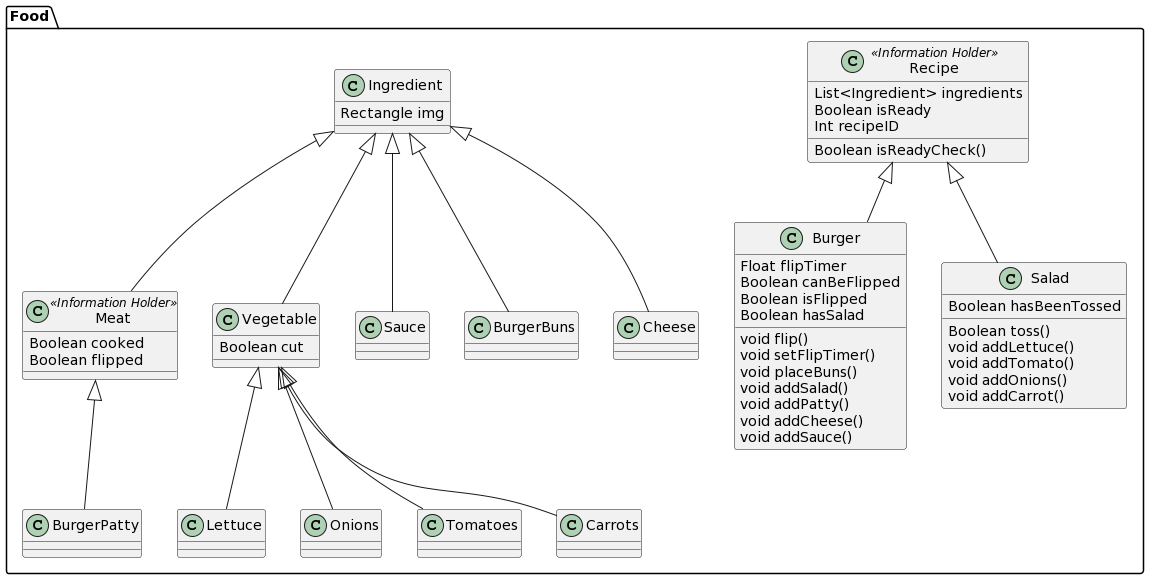
Michael Marples

Lucy Walsh

The architecture for our Piazza Panic game is displayed as UML class diagrams. To build these class diagrams, we used the PlantUML open-source tool and because we created these in Google Docs we used the PlantUML Gizmo extension/add-on tool. We chose PlantUML because it made the creation of the diagrams easier as the software uses a straightforward plain text language that was quite quick and easy to learn.

[](https://www.plantuml.com/plantuml/img/XLNRQXin47tNLmW_jI4n3NqhnWGxf3K4377RTw4TEyAxaf16JesRVwyktfNaI_Ec6PsvdAFPsGUBpA3hMa8qurks0JhwpZeOqGEPEukg0qD7NyRZcn6zkwEZ-n7jtNCbqQYs1KF8NAcjlnoFlsQOTvr2XDoiaA5Gaax-UjoJsMVNHSJS8Nu0cR7j1uY5t1Xe14Zy7se5PbSa8YJeC5FCD97PVM9savWLm8km9CM91bSJdYjwBQa__HHTq9EtpDfa7CXJgnXItR9zS93VIZJKwOOX1FlJv-JHpDdUyTvd65vvCbd8jJ9TakI7QXimq-c1hD1ujbGPRoyUA_C5kD10Pagrm6JlOytU5wYSyy2A8pINAcVQleTiXDeBYPGFC2VnoDJ7OnyU6VVLk05FkhygscYWXUeYjn5cjonyIpr7xj2PrCIpi3X9jAUKCnE7TaYi1TywdKUig9ckba60meji2ycNSdvB4frqT4MKMp_C26RDU25GX0Tn8XNkZF5zX7EMvPN93GHNFmkTsi4lNLgFwasMzZEiiRHVnEPrS8ZG1UEOX2Hbv_dJE4gNjPdwkFviO-z_V8Hfud5_F7PX2Lbj148vA-KU2BpxF5GRqCn0yGBvM8XQhOawWSE5NEv0Le458BwOm-Nw33HahpPPbNtT2dqjSLo-AVGwHEaBYA5mlRNIG3XflHUQVjfj_mx-0TwO248kqzTigo_Chc9OJdwsE6lTJYQVNkvmM6NvZr8CDp_VVIVZp-ib6gdKSpQDc6nR4jSYnRWftqCh3o0R_vVx1m00)

[](https://www.plantuml.com/plantuml/img/VLBBQiCm4BphAnQTsa82kGSJabCXWOBx0njv2IAsLBGhA3ZvzyfnQik8zhHY7YlDe1qB1ebTgzGNcYjU2FI9N6JTApKSQXE876nlgnMSqBePA-WFZ4m3M30rYLXtuN_CXq2D5Vw3_OpYWyMs88w43OMpnz0yhG_9Fefu1_fDGnBfpNgzqP3ThaoBp31BUrLBI4y4eMzHU-zRGWVU7LfhhY-lwfvRnYMzEbgMxRoZWlDZyaYyTwdD2kmmE5kmA36rF36_8NDiwZG7opfpTj7abCkCw22Rb4LvpqYZfGA77V4IOkF3c6P7had_vWS0)

[](https://www.plantuml.com/plantuml/img/TLJBQiCm4BphAqHExI5_449mYcaWfIK9lM-jZIEYI4PQ5uBRVwzisPPYK_31E-kTcNtWZIEmLDyKOnKKLoYHBt9ZnA9Xx825h92lVfPBdjMsHFi07K61O6odIuj2egO0lo4Gdw6VM2B1by9PvWXrCKU3NbPhDqjjBuWE6Uk4aey_WEZEsAWKSdiaQWKcyBkMndDFq9Ev0HcSuLkmrf1Zh53W71-ciDhfix6-n7FnLwC4slMwONlfQ1LThxaStntBZ548cajtG11ttnrns_7jNgP9tsbnVNfcluDm96tQLH3eqbkKjp9-4gRoZxE1lp3cYc8CpqfMbOz7ah7zfAIccEzM7tCNS1cYFXdd4bRoeJV_RQJW84I_Y0G9qqw0i9G427FtG9GEksvOhenlj7L-aZT_cQDFq1dcVKDndbF4MmxtsqcrD8EiGye7rW6h51JO7cFYhIjFukxyKk_TcJuKXFPzAm-zT8NDfC5MB6Bzx28G9XNZuN0Zqfzia-piWI7OQsF_R50B_m_u0m00)

We began designing the architecture by doing Responsibility Driven Design (RDD). We first came up with the themes for the game. Then we made CRC cards and on them we wrote the names of some candidate objects. Then on each card we wrote the purposes of the object and its role stereotypes. After that, we grouped the objects into customer, cook, menus, stations, food and general. We then removed objects which were too similar to another, and then on each card we wrote the responsibilities of the object. Once we were happy with the cards, this was all compiled into a Google Drive folder to make it easier to reference, and then from this we started to write the object diagrams in PlantUML. We made sure each object had the right attributes and methods to satisfy the responsibilities we had given it, as well as making small changes where it made sense. For our CRC cards, see our group’s website at: https://decassociation.github.io

One significant aspect of our class diagrams is that they are not 100% made from our CRC cards. RDD and from that our CRC cards gave us a base for our game architecture however once we started actually developing our game and converting the CRC cards to class diagrams we realised that we were missing a lot, for example each of the ingredients and an ‘Ingredient’ superclass for all of the ingredients. It was such a simple thing but something that we completely missed. So, from our initial plan/base, the class diagrams have evolved a lot as we realised we were missing classes, functions and variables because we can’t predict everything and often key details would arise during development of our game implementation. To see any intermediate class diagrams, see our group’s website: https://decassociation.github.io

A key and noticeable aspect of our class diagrams is that they are split into packages. Each package is a separate diagram because it makes the display of the classes easier as each package contains related classes, as well as making it easier for multiple team members to be working on the diagrams simultaneously. All our classes and packages are not only based on our CRC cards from RDD but they’re also heavily based on the requirements for our Piazza Panic game. Our requirements gave us a strong guideline because the requirements are of course paramount in our development, they act as a sort of checklist for us to always refer back to, to ensure we meet our customer’s needs.

**To Add:**

* NEED LINK TO A PAGE ON WEBSITE WITH CLASS DIAGRAMS AND **CRC CARDS**
* Component-Entity-Systems description?

***PROPOSED TABLE WITH CHANGES:***

| Classes | Requirement ID | Reasoning |
| --- | --- | --- |
| Timer, GameBoard, CustomerController, Customer | UR\_GAME\_TIME, UR\_TIMER, FR\_TIMER, UR\_SCENARIO\_MODE | The Timer class works with the GameBoard class to store the time the player takes to complete the scenario because the GameBoard works with the CustomerController so it can check when all customers have been served and the timer can be stopped. CustomerController also works with the Customer class as each customer will know how long they have waited. |
| PauseMenu, MainMenuScreen, CreditsScreen, LeaderboardScreen, SettingsMenuScreen, Tutorial | UR\_PAUSE, FR\_PAUSE\_SCREEN, UR\_MAIN\_MENU, FR\_MAIN\_MENU, UR\_CREDITS, FR\_CREDITS\_SCREEN, UR\_LEADERBOARD, FR\_LEADERBOARD, UR\_CONTROL\_CHANGE, UR\_TUTORIAL, FR\_TUTORIAL | All menu screens inherit from MenuScreen (superclass) since they all have the same purposes with only slight differences really. Even the tutorial is just a screen with the controls displayed. This meets all user and functional requirements around the screens, menus and tutorials. |
| Tutorial, SettingsMenuScreen | FR\_TUTORIAL\_CONTROL\_CHANGE, UR\_CONTROL\_CHANGE | Tutorial will read the settings which have been set by SettingsMenuScreen. The tutorial is just a screen/menu with the controls on as it is meant to be simple. If the controls are changed in the settings menu then the Tutorial class should be updated with this information. |
| GameBoard, MenuButton, MenuScreen, SettingsMenuScreen | UR\_SOUND, NFR\_COLOUR\_BLIND | GameBoard may control sounds played during gameplay, MenuScreen and MenuButton may control sounds played in menus and SettingsMenuScreen will allow customisation of things like volume. MenuScreen is the superclass of all of the different screens because they all have very similar functions and just link to different screens etc really to having one class they all inherit from will save duplicated code. A MenuButton will also have one MenuScreen but a MenuScreen can have many buttons. We will also design sprites with colour blindness in mind so that colour is not the only distinguishing factor, try and avoid common colourblind clashes like red and green, and colours may be changed in the settings menu, which would change the GameBoard class. |
| Cook, CookController | UR\_COOKS, UR\_CARRY, FR\_COOK\_INTERACT, FR\_MOVE\_COOK, FR\_DROP\_INGREDIENTS, FR\_SWITCH\_COOK | The CookController class defines the controls and general logic for all of the cooks that the player can play as, so it acts as a controller for them. However, each cook must store the ingredients they are carrying, that is not the job of the CookController. Therefore, the two interact with one another. Together they meet all of the project requirements around the cooks, how they move, cook and interact. |
| Customer, CustomerController | UR\_SCENARIO\_MODE | CustomerController will control how customers appear, in this case 5 customers total |
| Recipe, Burger, Salad | UR\_RECIPES | The recipe class is the superclass for all recipes we are implementing in Piazza Panic (burger and salad) which reduces duplicate code. For example, all recipes need a function to check if the recipe/meal is ready to give to the corresponding customer and a boolean variable is required to store this. |
| Ingredient, Vegetable, Meat, Sauce, Cheese, Burger Buns, Carrot, Lettuce, Tomatoes, Onions, BurgerPatty | UR\_INGREDIENTS | There is a class for every ingredient because every ingredient will of course have its own sprite. Some ingredients have a superclass of either Vegetable or Meat depending on whether they are a meat, a vegetable or neither. Meat, Vegetable and the classes that are neither, have the Ingredient superclass. There is a hierarchy here because all ingredients have a sprite but vegetables can be cut and then meat ingredients can be cooked and flipped (the vegetable ingredients are always raw so are just cut and put into a salad for example). |
| CookingStation | UR\_COOKING\_STATIONS | The CookingStation class is the superclass of all the different cooking stations. Each type of station has its own, specific functions but they all have some of the same variables such as a list of the ingredients that they can use, cook, cut, etc.There would be a lot of repeated, redundant data if it wasn’t a superclass. |
| CookingStation | NFR\_QUICK\_COOK, FR\_COOKING\_STATIONS, UR\_COOKING\_STATIONS | CookingStations may change an ingredient’s state after a period of time, set this time to be appropriate, likely around 30 seconds. Some cooking stations (since CookingStation is the superclass of all stations) will take different times to perform certain actions. For example, cutting vegetables will take a shorter time than frying a burger patty. |
| ServingStation, BakingStation, FryingStation, CuttingStation | UR\_SERVE, FR\_COOKING\_STATIONS, FR\_COOK\_INTERACT | As described above, all cooking stations inherit from CookingStation, but they each have different functions and purposes. All cooks will be able to interact in specific ways with each cooking station. For example, cooks will cut a vegetable at a CuttingStation and put something in the oven at a BakingStation. This also includes cooks being able to serve customers at a ServingStation - taking their order and giving it to them once made. |
| IngredientStation | FR\_INGREDIENT\_STATIONS, FR\_UNLIMITED\_INGREDIENT, UR\_INGREDIENTS, UR\_PANTRY, UR\_RECIPES, UR\_CARRY | IngredientStation inherits from CookingStation, however it works a bit different from the other cooking stations. The only thing a cook can do with an ingredient station is pick up an ingredient and add it to the stack of ingredients they are carrying, and the ingredients should never run out. These are the ingredients used to make the recipes so IngredientStation is fundamental to the recipe and ingredient requirements. All ingredient stations must be in the pantry area to meet the UR\_PANTRY requirement but that isn’t covered explicitly by the architecture. |
| Tutorial, MenuScreen, SettingsMenuScreen, GameBoard, CookController, CustomerController, CookingStation | NFR\_EASE\_OF\_USE | All aspects of the game will be designed with ease of use in mind, but these classes are most likely to have an effect on ease of use through clear and simple navigation. Each screen class inherits from the MenuScreen class so they will all have a common, standard structure that will make the menus easier to use. |

***ORIGINAL TABLE:***

| **Requirement ID** | **Class** | **Reasoning** |
| --- | --- | --- |
| UR\_GAME\_TIME | Timer |  |
| UR\_PAUSE, FR\_PAUSE\_SCREEN | PauseMenu | The PauseMenu class is used to pause the game at the press of a button which then takes the player to the pause menu. |
| UR\_MAIN\_MENU, FR\_MAIN\_MENU | MainMenuScreen |  |
| UR\_CREDITS, FR\_CREDITS\_SCREEN | CreditsScreen |  |
| UR\_LEADERBOARD, FR\_LEADERBOARD | LeaderboardScreen |  |
| UR\_COLOUR\_BLIND | Cook, Customer, Ingredient, CookingStation, MenuScreen | Will design sprites with colour blindness in mind so that colour is not the only distinguishing factor, try and avoid common colourblind clashes like red and green |
| UR\_CONTROL\_CHANGE | SettingsMenuScreen | Controls will be changed in the settings menu |
| UR\_TUTORIAL, FR\_TUTORIAL | Tutorial |  |
| UR\_MOBILE\_SUPPORT |  | We are not planning on implementing mobile support at this time |
| UR\_SYSTEM\_SPEC |  | All aspects of the game will be designed with low power systems in mind, no specific class |
| UR\_SOUND | GameBoard, MenuButton, MenuScreen, SettingsMenuScreen | GameBoard may control sounds played during gameplay, MenuScreen and MenuButton may control sounds played in menus and SettingsMenuScreen will allow customisation of things like volume. MenuScreen is the superclass of all of the different screens because they all have very similar functions and just link to different screens etc really to having one class they all inherit from will save duplicated code. |
| UR\_COOKS | Cook, CookController |  |
| UR\_SCENARIO\_MODE | Customer, CustomerController | CustomerController will control how customers appear, in this case 5 customers total |
| UR\_RECIPES | Recipe, Burger, Salad | The recipe class is the superclass for all recipes we are implementing in Piazza Panic (burger and salad) which reduces duplicate code. For example, all recipes need a function to check if the recipe/meal is ready to give to the corresponding customer and a boolean variable is required to store this. |
| UR\_INGREDIENTS | Ingredient, Vegetable, Meat, Sauce, Cheese, Burger Buns, Carrot, Lettuce, Tomatoes, Onions, BurgerPatty | There is a class for every ingredient because every ingredient will of course have its own sprite. Some ingredients have a superclass of either Vegetable or Meat depending on whether they are a meat, a vegetable or neither. Meat, Vegetable and the classes that are neither, have the Ingredient superclass. There is a hierarchy here because all ingredients have a sprite but vegetables can be cut and then meat ingredients can be cooked and flipped (the vegetable ingredients are always raw so are just cut and put into a salad for example). |
| UR\_COOKING\_STATIONS | CookingStation | The CookingStation class is the superclass of all the different cooking stations. Each type of station has its own, specific functions but they all have some of the same variables such as a list of the ingredients that they can use, cook, cut, etc.There would be a lot of repeated, redundant data if it wasn’t a superclass. |
| UR\_PANTRY | IngredientStation |  |
| UR\_CARRY | CookController | CookController will control which cook interacts with an ingredient and how |
| UR\_SERVE | ServingStation |  |
| UR\_TIMER | Customer, CustomerController, Timer | Customers know how long they have been waiting to time each individual serving, Timer will track overall game time |
| FR\_SWITCH\_COOK | CookController |  |
| FR\_MOVE\_COOK | CookController |  |
| FR\_COOK\_INTERACT | CookController |  |
| FR\_DROP\_INGREDIENTS | CookController |  |
| FR\_COOKING\_STATIONS | CookingStation |  |
| FR\_INGREDIENT\_STATIONS | IngredientStation |  |
| FR\_UNLIMITED\_INGREDIENTS | IngredientStation |  |
| FR\_CONTROL\_CHANGE | SettingsMenuScreen | SettingsMenuScreen will allow users to change their controls |
| FR\_TIMER | Customer, CustomerController, Timer |  |
| FR\_TUTORIAL\_CONTROL\_CHANGE | Tutorial, SettingsMenuScreen | Tutorial will read the settings which have been set by SettingsMenuScreen |
| NFR\_COLOUR\_BLIND | Cook, Customer, Ingredient, CookingStation, MenuScreen | Will design sprites with colour blindness in mind so that colour is not the only distinguishing factor, try and avoid common colourblind clashes like red and green |
| NFR\_QUICK\_COOK | CookingStation | CookingStations may change an ingredient’s state after a period of time, set this time to be appropriate, likely around 30 seconds |
| NFR\_LOW\_SPEC |  | All aspects of the game will be designed with low power systems in mind, no specific class |
| NFR\_EASE\_OF\_USE | Tutorial, MenuScreen, SettingsMenuScreen, GameBoard, CookController, CustomerController, CookingStation | All aspects of the game will be designed with ease of use in mind, but these classes are more likely to have an effect on ease of use |