Engineering 1
Assessment 1
Architecture Document
Cohort 3
Team 23

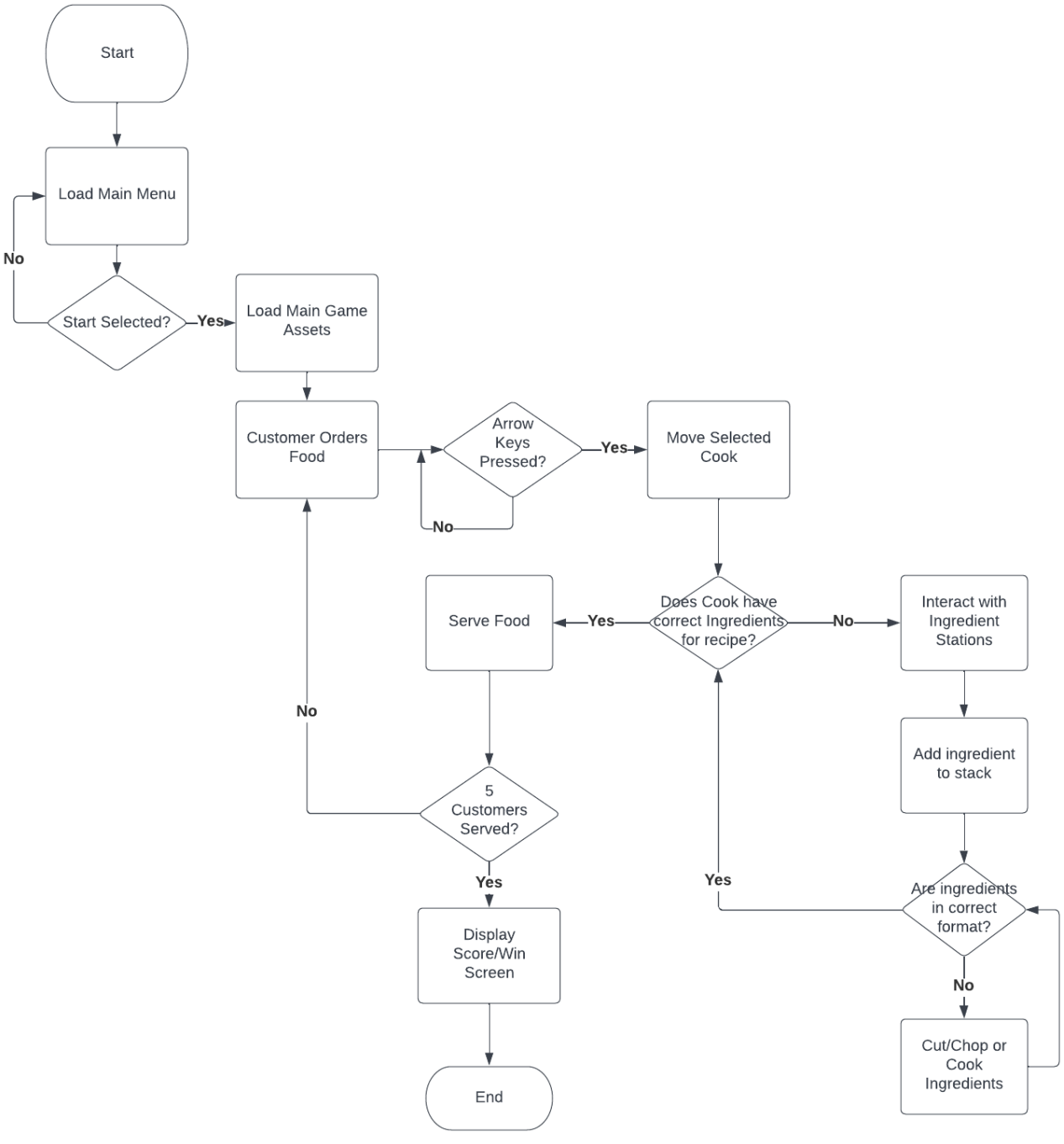
Seif Hussein
Joshua Mckean
Harry Draper
Sebastian Armstrong
Thomas Maalderink
Tikhon Likhachev

Architecture Development

UML class diagrams were used to represent the architecture we used. One tool that we used was PlantUML in the IntelliJ Idea IDE, this was used for the class diagram as it was simple to use and created a clearer diagram than other alternatives. Lucidchart was used to create the flow diagram to show the basic sequence of the game. Lucidchart was used because it allowed the creation of a very simple yet effective flowchart which is easy to understand.

Abstract Architecture

To enable an improved understanding of the game, a simple flow chart was created which is easy to follow. This gives a general representation at the high level of the game, displaying connections between different aspects and parts of the game. The flowchart is an abstract representation of the game as it doesn't go into the smaller details of the game for example the specific buttons used to interact or the timer which tracks how long it takes to complete the scenario, it instead just shows the base logic of the game. This helped to determine the classes for the UML class diagram.



CRC Cards

To help understand the classes needed for the game, CRC cards were created, this allowed us to determine the purpose of each class and the collaborators that each class will have in order to help make the implementation easier and give a better representation of the hierarchy of the game.

Entity
Responsibilities: - Update the entity's state - Render the entity on the screen
Collaborators: - GameScreen - Chef - Customer

GameScreen
Responsibilities: - Update the current screen state - Render the current screen
Collaborators: - MainMenu - Entity - PiazzaPanic

MainMenu
Responsibilities: - Update the main menu state - Render the main menu
Collaborators: - GameScreen

PiazzaPanic
Responsibilities: - Run the game - Switch between game screens
Collaborators: - GameScreen

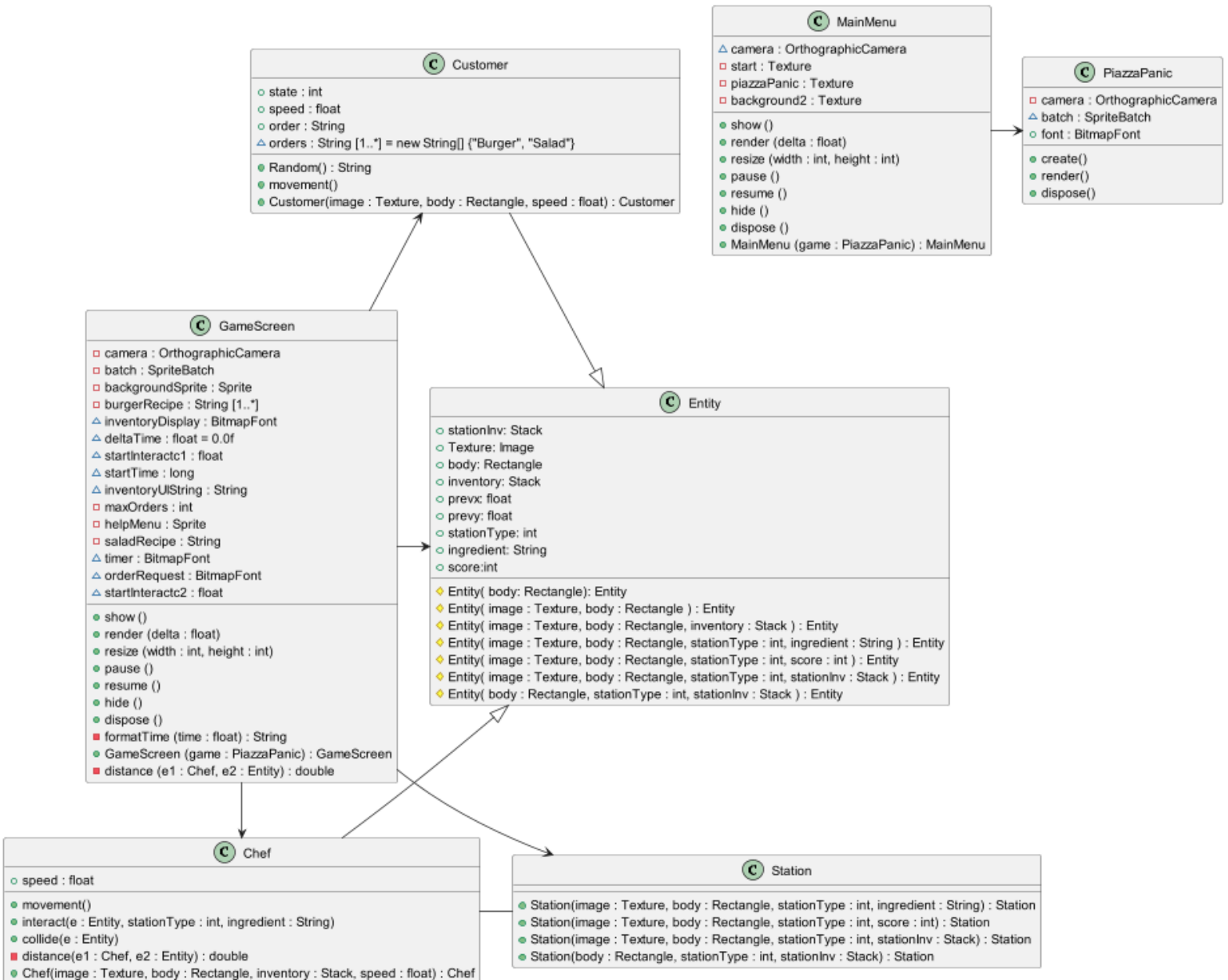
Chef
Responsibilities: - Cook Food - Collect Ingredients - Serve recipes
Collaborators: - Entity - Station

Station
Responsibilities: - Prepare recipes - Serve recipes to customers
Collaborators: - Chef - Customer

Customer
Responsibilities: - Order food
Collaborators: - Chef - Entity

Concrete Architecture

The class diagram below represents the implementation of the classes as well as the relationships and dependencies between them. This diagram gives a more detailed insight into the attributes of each class and which classes are extended by others. Furthermore, it shows which attributes are protected, private or public to make the classes easier to understand. The data type of each attribute is also displayed in the class diagram which makes the implementation of the game more efficient.



Justification Relating Architecture to User Requirements

The list below shows how each user requirement was satisfied by the architecture:

UR_SUPPORT_MULTIPLE_COOKS - The class “GameScreen” uses the third party library libGDX to enable keyboard inputs to change between cooks.

UR_CONTROL_COOKS - LibGDX allows keyboard inputs to provide a way for the user to move cooks around the kitchen. The “Chef” class contains all the attributes of the movement of the cooks and enables them to interact with elements to complete recipes

UR_SUPPORT_MULTIPLE_COOKING_STATIONS - The “Station” class is responsible for the functionality of the different cooking stations in order to complete recipes.

UR_INTERACT - Using libGDX the user can press keys on the keyboard to interact with cooking stations, ingredient stations and other interactable parts of the game. This is done in the GameScreen class.

UR_FLIP_PATTY - The “Chef” class allows the user to flip the patties on the correct cooking station by interacting with the keyboard.

UR_CUT_INGREDIENTS - The “Chef” class allows the user to cut/chop certain ingredients using the correct cooking station by interacting with the keyboard.

UR_HAVE_RECIPES - The “GameScreen” class contains the recipes within it, so that the user can put together the correct meals.

UR_HAVE_PANTRY - The “GameScreen” class contains the functionality to collect ingredients from the pantry to put together the recipes

UR_CUSTOMER_ORDER - The “Customer” class contains the attributes of the customer and selects a random recipe to order. It also creates a maximum of 5 customers which need to be served in order to complete the scenario.

Abstract Architecture Justification

For abstract architecture, OOP (object oriented programming) was used because it allowed the reuse of code through inheritance. The high level features of the program were decided and were split into certain classes, in order to make inheritance easier without the need for extra code throughout the implementation process.
One example is the decision for the Entity class, as a result of both the Customer class and the Chef class both being entities in the game it would be better to have an Entity class to reduce the amount of code needed.

However, a different approach was used for each screen because each screen has different attributes and functionality to display the screens. This kept the screens contained so that there would be a reduced chance of errors caused when switching screens.

Concrete Architecture Justification

LibGDX was used as it gave access to simple setup for user inputs and dimensions of the windows. Furthermore, as a result of the OOP approach LibGDX made it easier to add functionality and improve upon existing functionality already implemented. LibGDX is also a popular library for game development, meaning that the support for the future development and the performance make it the best choice for our game. The concrete architecture design allows the easy management of platforms and specific requirements in a combined way.

In addition LibGDX ensures that game logic and the visual aspects of the game are optimised for the target platform in order to make a better user experience.

Overall, the use of concrete architecture when using libGDX provides a solid foundation for game development, enabling the development of a high quality game that runs smoothly on the chosen platform.