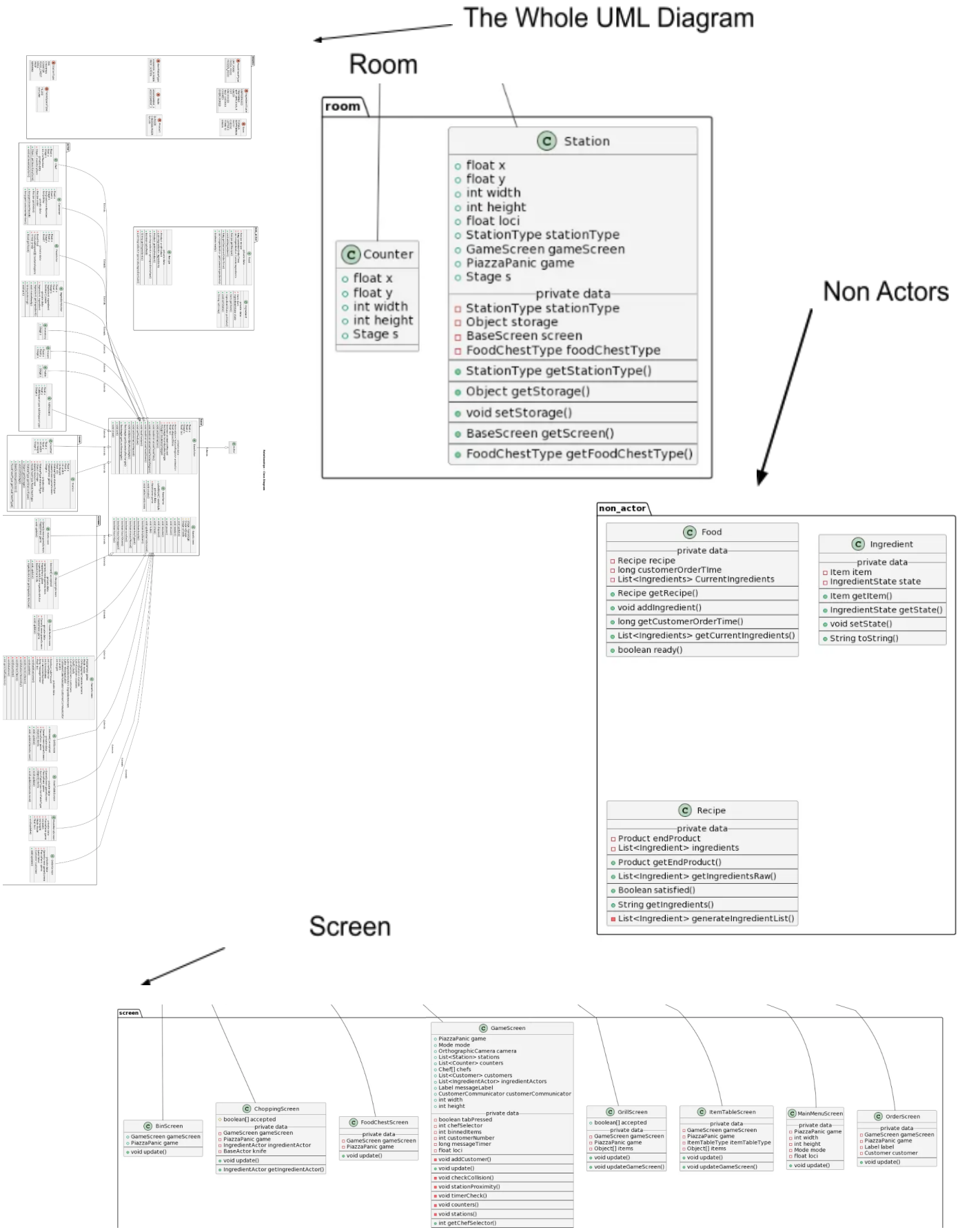
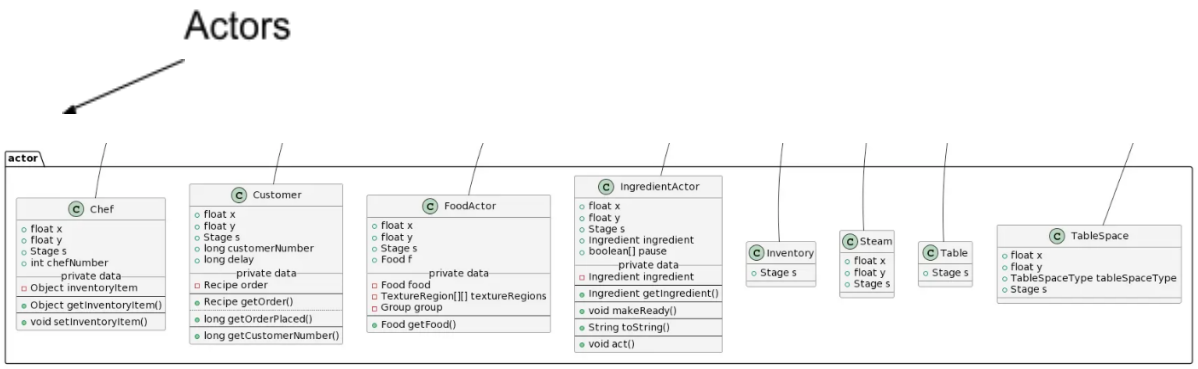
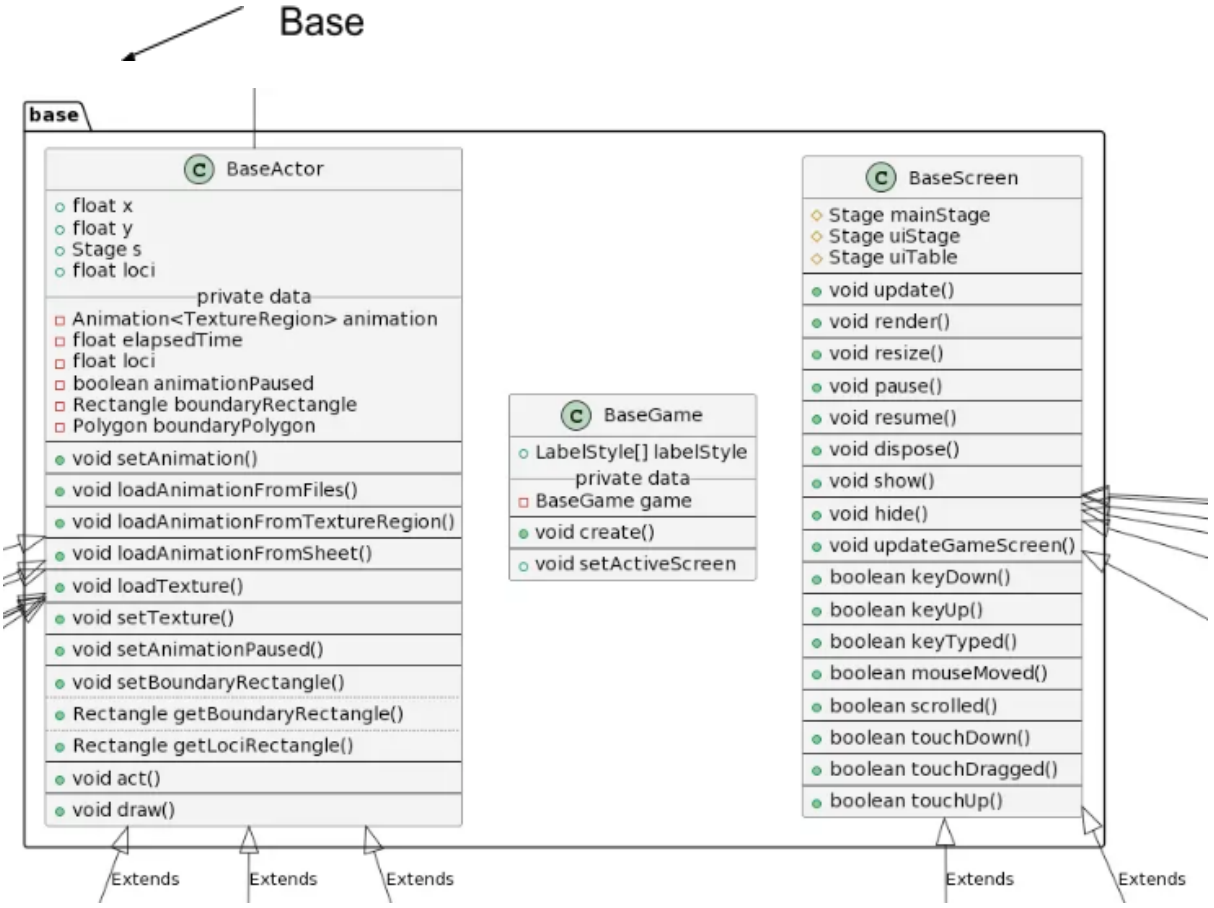
ENG1 Group 9
Architecture

Cameron Fox
Liam Burnand
Jack Cameron
Eoin O’Connor
Adam Penny
Ben Brown

PART A


The Whole UML Diagram
Room
room
Station
float x
float y
int width
int height
float loci
StationType stationType
GameScreen gameScreen
PiazzaPanic game
Stage s
private data
StationType stationType
Object storage
BaseScreen screen
FoodChestType foodChestType
StationType getStationType()
Object getStorage()
void setStorage()
BaseScreen getScreen()
FoodChestType getFoodChestType()
Counter
float x
float y
int width
int height
Stage s
Non Actors
non_actor
Food
private data
Recipe recipe
long customerOrderTime
List<Ingredients> CurrentIngredients
Recipe getRecipe()
void addIngredient()
long getCustomerOrderTime()
List<Ingredients> getCurrentIngredients()
boolean ready()
Ingredient
private data
Item item
IngredientState state
Item getItem()
IngredientState getState()
void setState()
String toString()
Recipe
private data
Product endProduct
List<Ingredient> ingredients
Product getEndProduct()
List<Ingredient> getIngredientsRaw()
Boolean satisfied()
String getIngredients()
List<Ingredient> generateIngredientList()
Screen
screen
BinScreen
GameScreen gameScreen
PiazzaPanic game
void update()
ChoppingScreen
boolean[] accepted
private data
GameScreen gameScreen
PiazzaPanic game
IngredientActor ingredientActor
BaseActor knife
void update()
IngredientActor getIngredientActor()
FoodChestScreen
private data
GameScreen gameScreen
PiazzaPanic game
void update()
GameScreen
PiazzaPanic game
Mode mode
OrthographicCamera camera
List<Station> stations
List<Counter> counters
Chef[] chefs
List<Customer> customers
List<IngredientActor> ingredientActors
Label messageLabel
CustomerCommunicator customerCommunicator
int width
int height
private data
boolean tabPressed
int chefSelector
int binnedItems
int customerNumber
long messageTimer
float loci
void addCustomer()
void update()
void checkCollision()
void stationProximity()
void timerCheck()
void counters()
void stations()
int getChefSelector()
GrillScreen
boolean[] accepted
private data
GameScreen gameScreen
PiazzaPanic game
Object[] items
void update()
void updateGameScreen()
ItemTableScreen
private data
GameScreen gameScreen
PiazzaPanic game
ItemTableType itemTableType
Object[] items
void update()
void updateGameScreen()
MainMenuScreen
private data
PiazzaPanic game
int width
int height
Mode mode
float loci
void update()
OrderScreen
private data
GameScreen gameScreen
PiazzaPanic game
Label label
Customer customer
void update()





We have used a diagrammatic representation in the form of a UML Diagram to represent the structural and behavioural aspects of the games architecture. Due to the size of the diagram, we have provided screenshots of each class for a more detailed view, with the overall diagram giving a general view of the program's structure. To build this we used a collaborative approach in PlantUML. Sending iterations of the textual representation over Slack allowed us to maintain backups, as well as make any changes

as necessary when there are changes made in the code. We could have used alternative tools such as LucidChart which uses a more graphical building UI however we felt the fine control over the diagram was beneficial. The architecture design has allowed for a very logical approach that allows new features to be added by extending the relevant base class for example both Chefs and Customers are Actors so both can benefit from the Actor class, this keeps the majority of classes short and allows for the reuse of code. The use of PlantUML to create diagrams allowed for very rapid additions to be made as whilst not identical to java classes the diagramming language allows for almost direct copying of classes compared to other tools which would require manually retyping all class information. The structure of the architecture maintains that very similar to the diagram. Besides the files required by libGDX the Java class files are stored in the eng1 directory within core and are stored in folders grouped by function or likeness e.g. actor,base, screens etc. LibGDX allows for a dedicated assets directory which allows for all graphical and audio aspects of the game to be placed together and are easily accessible by code without the full path of the component being required. Further to this it also prevents different file types cluttering the same directory thus making development more efficient.

PART B

The first part of the system architecture developed was the first commit to our GitHub repository. This set-up our VCS system and also included the required licensing for the project. Next the libGDX base project was set up using the libGDX launcher. This created a build.gradle file along with other required libraries for us to begin developing our game. These are commit code bd621dc and feedcee respectively.

After this, the first screens and text characters were introduced in commits d11e782 (text) and ab6b957 (screens). This included the window containing the main title screen and also the screen in which the game will be played. This was a significant step as it provided a way for us to visualise how the code would be working and if it was doing what we were intending it to do in later commits. One such commit was a0f8891 which introduced new classes including baseActor, chef and counter. BaseActor allows all the components of the game (such as chef and counter) to exist on screen, and then a texture to be set to the actors via image files or sprite maps. It also adds simplicity to the individual actor classes as these previous components are removed from those, allowing us to solely focus on developing their functions (Requirements UR_CHEF, .

The next significant commit to the repository was ef972678. This added the station object class to the game, and all of its functions, the map and boundaries for the stations. The commit also added station components to the GameScreen class in order for it to be usable (Requirements FR_PLACE_ITEM, FR_CHOP_ITEM, FR_COOK_ITEM, FR_DISPOSE_ITEM, FR_GRAB_ITEM).

A smaller yet important commit was be7704ea. This removed the objects package and created a new class called enum to contain that code. This was placed in the com.eng1 package in order to create a more defined structure for the code in order for it to be more navigable and easier to use.

In the next development stage we split the BaseScreen into BaseGame and BaseScreen. This allowed easier switching between them and also improved the functionality of both the screen and the game. Also, some functions were removed from these screens as after they were split, they were no longer necessary to have (commit 4407b3b9) (Requirements UR_UI, UR_UX).

Next, commit 30ccd929 added a centralised loci value. This was assigned at the beginning of the code and was used to control the size of the counters and stations used in the game. This allowed us to map the playing area and choose where and how many of each utility would be added.

Commit d68b13f1 was pushed in order to fix an error discovered during the development process when we found we were using the wrong java version. This was rectified by changing it and it changed records to normal classes, meaning to supplement this we needed to add getters to these classes. Then, commit 79c96912 was incredibly important and significant as it added support to the collision logic and prevented players from leaving the area of play and subsequently the game if a counter was removed during testing (Requirements UR_INTUITIVE, UR_CHEF). Commit bbe8d47b was arguably the most important commit as it provided the inventory functionality by adding all of the screens and implementing stored screens in the stations (Requirement UR_UI).

Later in development, we changed how the recipes functioned in the game and also changed the ingredient types from strings to enums in order to ease the process of adding new ingredients in the future (commit 1ae077c6) (Requirements UR_UX).
The next major and significant commit we implemented was to add a reset function to the oldest order timer, allowing the game to sufficiently track the next one from 0 seconds (Requirement UR_FINISH_LEVEL).

After, commit 46296a4 added a huge amount of functionality to the code, including drag and drop mechanics and interactions between each station and the chef. It also introduced the ingredients as objects to be interacted with. This followed requirements FR_GRAB_ITEM, FR_PLACE_ITEM, FR_CHOP_ITEM, FR_COOK_ITEM and FR_DISPOSE_ITEM.

Commit 6c8a532 built on the previous commit with another large amount of code introduced. Further drag and drop mechanics, states for the ingredients whilst being prepared (cooked, uncooked etc). It also implemented messages for the status of things, such as whether a chef has an item compatible with certain stations (Requirements: FR_COOK_ITEM, FR_CHOP_ITEM, FR_SERVE_CUSTOMER).

Finally, commit 7970498 added the timers for each order placed as well as priority for them (first come first served). It also built further upon the compatibility functionality, such as not being able to grill lettuce (Requirements: FR_COOK_ITEM, FR_DISPOSE_ITEM, FR_CHOP_ITEM).

Architecture is instrumental in software engineering and development, which is why we put a big emphasis on documenting it and how it evolved during our design process. From starting as a basic libGDX library, the software underwent massive change throughout the assessment period. One such example was the changes the ingredients went through. From starting as strings, their type consistently changed until finishing as

enums. This is supplemented by the sheer number of commits made to the GitHub repository, showing how the architecture was constantly being tweaked and changed to better support our development goals.

Whilst most changes were made to do this, some were for separate reasons. Some changes, such as commit d68b13f1, were made to repair previously unspotted errors (such as the incorrect java version in the example commit) and others like commit 1ae077c6 were made to make further implementations easier to complete and less drawn out.

Overall, the way the architecture was constructed allowed our project to evolve along with it, to the point of completion. It also meant that we could track how the compilation of the project's components was going and integrate them together.