UNIVERSITY OF YORK
DEPARTMENT OF COMPUTER SCIENCE

ENG 1
GROUP 9 - Kitchen Tossups

Report - Architecture

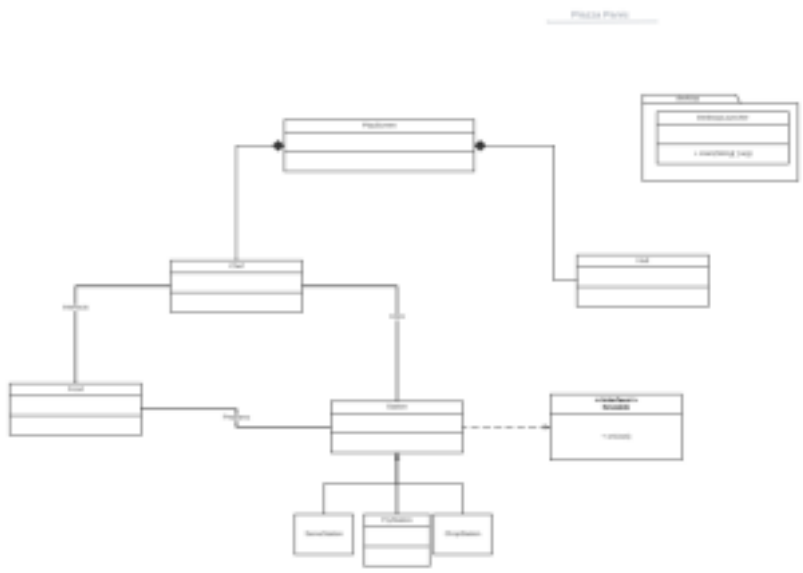
Team Members:

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

3. Architecture [22 marks]:
a) Give diagrammatic representations (structural and behavioural diagrams) of the architecture of the team's product, with a brief statement of the specific languages (for instance, relevant parts of UML) and the tool(s) used to create these representations (10 marks, ≤ 3 pages).

To generate these diagrams we used an online tool called 'Lucid'. This tool allowed us to generate a UML diagram graphically through a process of dragging and dropping each class and connection between them. Using this tool we also developed an Entity Component Diagram and a Use Case Diagram, which describes the function of the system in broader strokes. This diagram allows us to model how a user interacts with the system while maintaining the simplicity allowed without such a granular model. The implementation of a use case diagram to map the general functionality of the application in a way that is readable at a glance.

Our initial design without the attributes and methods:



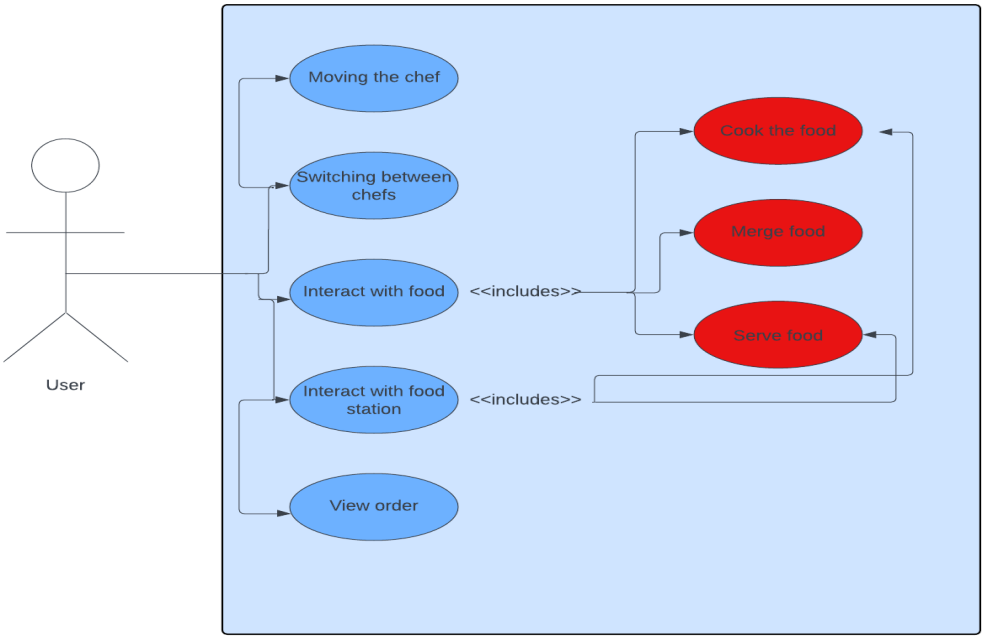
Our received architecture:

- Received Diagram

Our final architecture:

- Final Diagram

Use Case Diagram:



The Use Case Diagram does not conform to a style of code, but describes the use of the system from a user perspective. This diagram shows how the user would interact with the system to complete the goals of the game, which are to cook and serve food. It shows how the system is meant to function in order to distribute new orders and receive complete orders in a way that is understandable to a typical user.

b) Give a systematic justification for the architecture and describe how it was initially designed and how it evolved over the course of the project. Provide evidence of the design process followed (e.g. interim versions of architectural diagrams, CRC cards) on your team's website and link to them from your report. Relate the architecture clearly to the requirements, using your requirements referencing for identification, and consistent naming of constructs to provide traceability (12 marks, ≤ 3 pages).

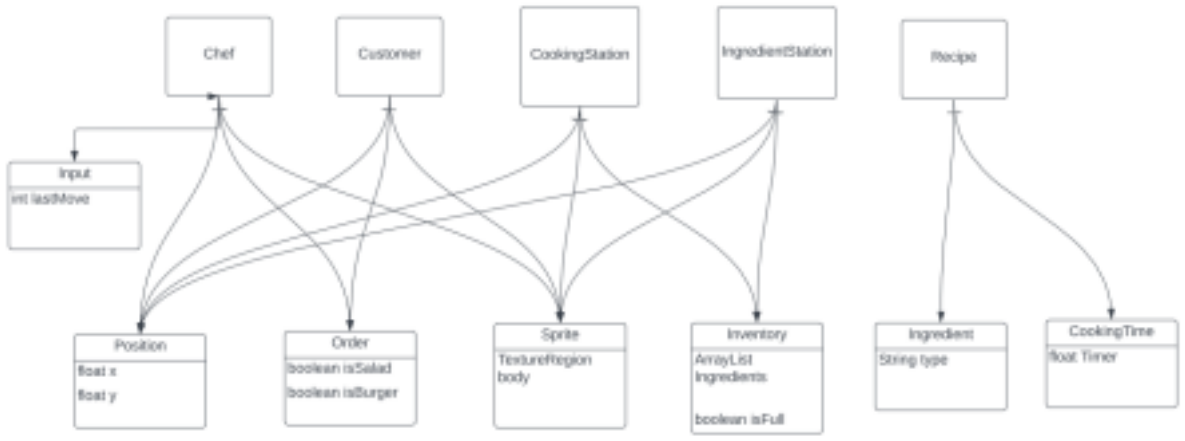
Greenfield Architecture:

When we first began to design the architecture of Piazza Panic, our main goal was to create a fun and challenging game (UR_FUN_GAME) with good performance, scalability, and flexibility. We started by identifying the key requirements for the game such as UR_SWITCH_COOKS, UR_MOVE_COOK, and UR_SERVE_DISH. These requirements were about the player's ability to move, their position, and serving the order. Another example of a requirement is (UR_RESTAURANT_DESIGN) which we had done long ago since we made a simple demo in the first week that we improved throughout the game .

In order to meet the requirements and achieve the goal of an engaging game, a systematic approach was taken to design the architecture of the game, and this approach is centred on the Entity Component System (ECS) pattern.

The ECS is a commonly recognized pattern in the field of game development and it emphasises the separation of concerns by decomposing a game into entities, components, and systems. After careful consideration of several potential architectures, we decided to use ECS in order to ensure good performance with a dynamic coding style. This pattern would enable easier modification and extension without disrupting the rest of the system, as well as efficient management of entities and the components that are linked with them.

To start, we created high-level architectural diagrams to illustrate the system's layout and how the various components would interact. The initial design focused on identifying core entities such as Chef, Customer, IngredientStation and CookingStation. Each of these entities had unique characteristics and behaviours, so for each one, a set of components were identified to fully describe their behaviour and attributes. For instance, the Chef entity required a position, a sprite, an input and an order component, and these components were attached to the diagram along with others.
The entity-component diagram:



As the development process continued, the architecture of the game evolved and additional entities were added as needed. The Recipe entity and its components, for example, were added later as the team realised that it was a necessity for the game because when the customer orders, the chef needs to know the recipe. Additionally, to map out the programming process and demonstrate the relationships between the various classes and their attributes, we created a class diagram.

When we first drew the class diagram, we only laid out classes, and it was quite basic since it was missing the attributes and methods. Then we enhanced our design and added many classes that we thought we would be using for the implementation, such as PiazzaPanic, PlayScreen, Station, Hud, Chef and Food. Then we also implemented InteractiveTileObject and WolrdCreator to the diagram. The ECS helped us to build our future game and class diagram since the components of entities then shaped into methods of our classes.

However, the design of the class diagram kept evolving as we came across bugs or realised we were missing some qualifications. Since it is so convenient to add or remove classes in ECS, it was not much difficult for us to evolve the implementation. For example, FoodCrate was added, which extends the Station, on the final design of the UML Class Diagram, as well as the CountdownTimer, InputHandler, Main Menu, Servery, PrepStation, OrderSystem,

OrderHud and Order.

Throughout the development process, interim versions of the architectural diagrams and CRC cards were attached to the team's website (github-wiki) to provide a clear overview of the evolution of the architecture. This documentation supports the explanation of how the architecture has evolved and provides evidence of the design process followed.

As a team we decided to use the LibGDX game development framework and the Box2D physics engine to implement the ECS architecture. The LibGDX framework provided a number of helpful tools that aided in the development process, and the Box2D physics engine was used to handle the physics simulation in the game.
In conclusion, the design of the architecture for PiazzaPanic was a crucial aspect of the development of the game. By adopting Entity Component System, we managed to produce a game that satisfies our requirements while also being highly flexible to modification. In order to avoid confusion and to provide traceability, we also used consistent naming for the requirements across the entire project. Our documentation of the architecture and its evolution on the website provided clear evidence for the design process that followed.

Brownfield Architecture:

After receiving the deliverables from assessment 1, we felt that a better representation of the architecture would be CRC cards going forward, and so we made some based upon the entity-component diagram and the base game code we received:

Chef	
Resposibilities	Collaborators
• Pick up and hold onto ingredients • Move around • interact with stations • Serve food to customers at counter	• Station • Customer • Ingredient • Order

Station	
Resposibilities	Collaborators
• Accessed by chef to cook/ prepare ingredients • Change states of ingredients	• Ingredient • Chef

Customer	
Resposibilities	Collaborators
• Be served orders • Place orders	• Order

Ingredient	
Resposibilities	Collaborators
• Combine to make orders • Can change state in stations • Carried and used by chefs	• Chef • Order • Station

Order	
Resposibilities	Collaborators
• Made up of ingredients • Servable to customers via the counter • Made by chef	• Customer • Chef • Ingredient

Power Ups	
Resposibilities	Collaborators
• Grant abilities to chefs • Spawn on the floor at random intervals	• Chef

After this, we went back to our requirements elicited from the customer meeting we held and noted them down, starting the proper software development thereafter. We started the further development of the architecture by refactoring and tidying the code (commit daa6dae) we had received in order to make it more manageable for us, such as removing unused imports and unnecessary spaces. After that, a third chef was added to correspond with requirement UR_SWITCH_COOKS (user should be able to switch between 3 cooks); this is shown by commit 8432e4c. This was one of the major components of the design as by adding a third chef we fully completed one of the main requirements we were elicited at the very beginning of the assessment. Later on, “mode” was added, satisfying requirements UR_SCENARIO_MODE and UR_ENDLESS as part of the brief (commit 49c508e). Later, as specified by the brief, a difficulty concept was added (UR_DIFFICULTY), which allows the user to select a difficulty for the game before starting it up (commit 63d2b0e). Furthermore, commit 651662f added the plain jacket potato item, making it available to be ordered, made and served. This fulfilled the FR_POTATO functional requirement we elicited via the brief, which stated that at least 4 recipes be included in the game. Commit 7de40dc introduced the power-ups class, including BasePowerUp as the baseline for all the “to-be” introduced power-ups and also the ExtraLife power-up (requirement UR_POWER_UPS).

Finally, due to necessity, we committed a bulk of code (commit 2304a88). This commit was the most important by far as it fulfilled numerous requirements we needed. One such requirement was UR_SAVE_GAME, which was accomplished during this commit and was a very important requirement to implement, although it was only partially fulfilled in the end. Another requirement completed was instructions (NFR_CONTROL_SCREEN). This was completed by adding the instructions to the main menu of the game.