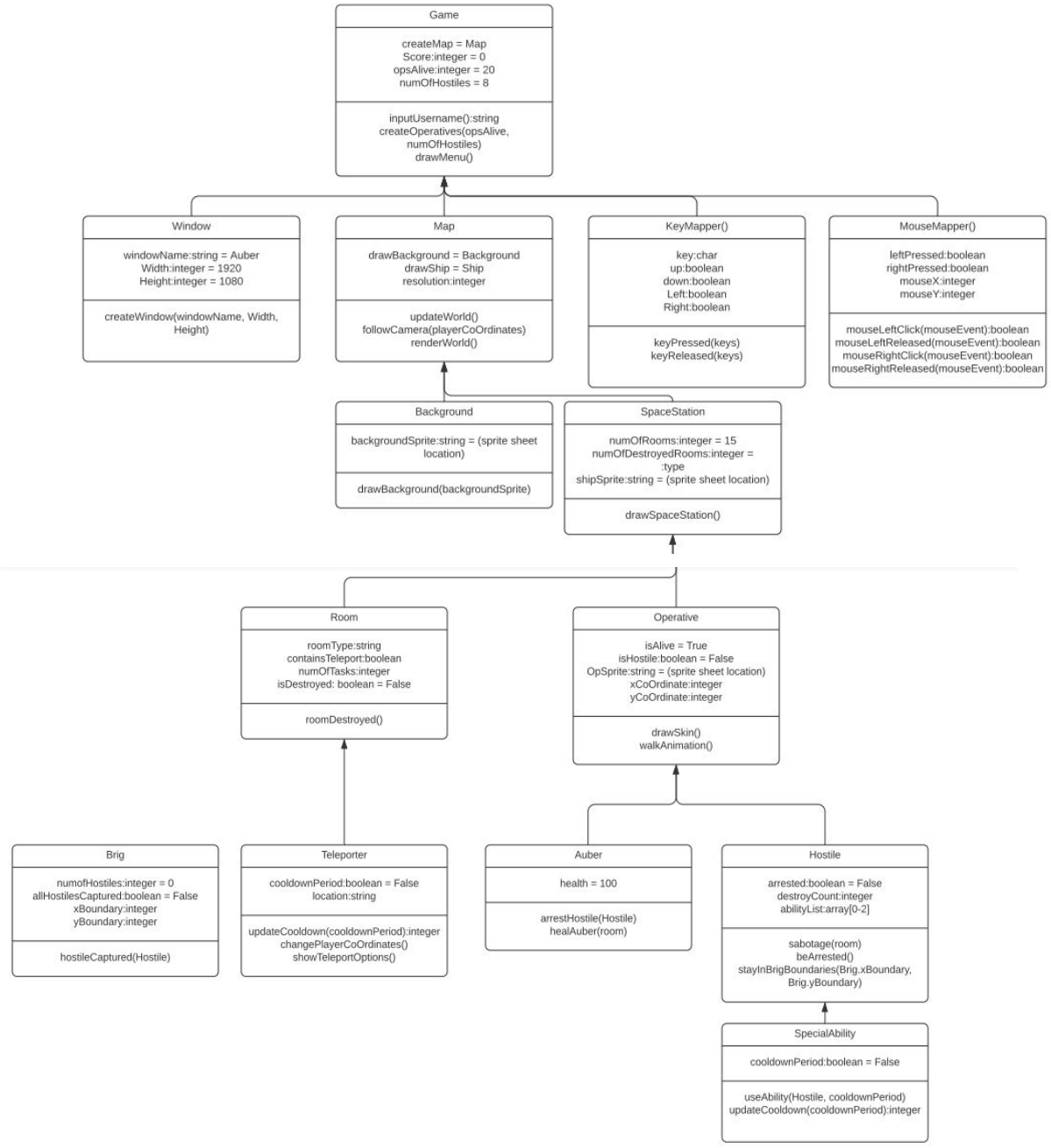
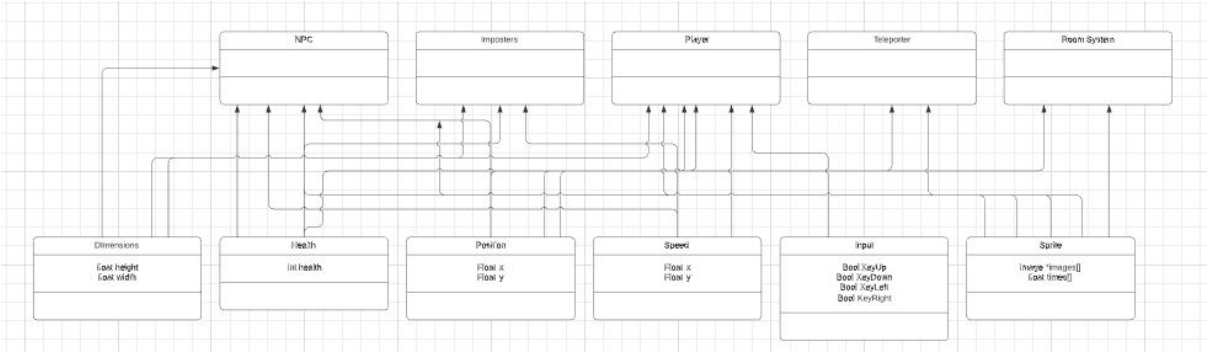
| | |
|---|---|
| **Module** | ENG1 |
| **Year** | 20/21 |
| **Team** | 33 (Short Circuits) |
| **Members** | Jack Lord, Neo Metcalfe, Sam Rodgers, Mohammad Abdullah, Qi Tang |
| **Deliverable** | Architecture |

## Class Diagram




As a team, we created a class diagram to portray and demonstrate how we could create the classes within Auber. This diagram allows us to see the entities that are within our game, the classes that we'll have to create for these entities, and the attributes and methods that they will require. This diagram was created using LucidChart, which is a free website that allows users to create diagrams and representations utilising UML.

## Entity-Component-System Diagram




This is the second diagram that we created as a team, in order to portray the entities in our game, and every component that these entities will contain within their respective classes. This diagram was also created in lucidchart and utilises UML to demonstrate the relationships between entities and components.

## Justification

Our Abstract representation of the Architecture was created in the form of a class diagram, and helped us to visualise how Auber would work early into the project. This diagram clearly and concisely prescribes how we will create our game, and more importantly the classes within our game. However, even though this diagram was useful for portraying the classes in the game, it was not useful for visualising the systems within the game, and how the entities will interact with each other. In order to show this, we built on our class diagram and created an entity-component-system diagram. This diagram was very useful to us, as not only did it allow us to visualise the entities and their respective components, it also allowed for us to see the systems within our game, and how the different components interact with each other in order to allow these systems to function. For example, one of the most important systems is moving the character, this is achieved by input, speed and position components. These systems are not easily traceable using the class diagram, but are when using the entity-component-system diagram, and therefore assisted with programming the game, whilst also being a more representative diagram of the software.

This entity-component diagram contains the entity ‘Player’. This is in line with the user requirements of FR_CHARACTERS, and FR_NAME. The Auber entity is the most important entity within the game, as this is the entity that the user interacts with and controls, therefore it was important for us to document it within this diagram perfectly so that it would meet the requirements of the project and the customer. Another benefit of documenting it so that the requirements are met in the diagram is that this allowed for an easy transmission between the diagram and the actual implementation of the entity and system, therefore allowing our software to be in line with the requirements. Another requirement we ensured was shown in our diagram was FR_HEAL, this is shown by the connection between the entity ‘Player’ and the component ‘Health’. It was important to include this within our game so that the infiltrators could damage Auber with their special abilities, and therefore require the user to travel to the infirmary so that they can heal.

Another entity in our diagram is ‘Teleporter’. This entity signifies the teleportation pads that will be located within the rooms of our game, which is also shown in the class diagram, and are used to instantaneously move the player from one room to another. The requirements linking to these are FR_TELEPORT and FR_TELEPORTATION_PADS. One important factor of these teleportation pads, is that they should only teleport Auber, the main player, and should not be interacted with by the npc’s or the infiltrators, as according to the requirement of FR_NOT_TELEPORT.

Our diagrams also contain an entity called ‘Room System’ which is important as these are a functionality of the game that determines the score of the player and whether they win or lose. The game contains at least 15 room systems, and these are interacted with by the infiltrators, as shown by requirement FR_DESTROY. The rooms within our game are different, but they all inherit from the same class, this is shown in the Class diagram with the Brig inheriting from the Room class.

By showing these requirements clearly in a visual representation, it allowed us to produce a piece of software that was heavily focused on the requirements and expectation of the customer. This is because the abstractions of the game ,created by these diagrams,

influenced how we programmed our game and provided us with blueprints to assist and ease the coding experience.