

## Software Architecture

Figures 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 and 1.7 represent an abstract view of the software architecture.

Figure 1.1 shows a component diagram describing how different components of the system communicate. Figures 1.2, 1.4 and 1.6 show state diagrams capturing the states of the system as an event occurs. Figures 1.3, 1.5 and 1.7 show interactions between entities in the system with a focus on the user. These were created with use of PlantUML. (<https://plantuml.com/>)

Figure 2 represents a concrete view of the software architecture. It shows a class diagram of the core entities, their features and relationships reflecting how the system was implemented. This was created using UML Star. (<https://staruml.io/>)

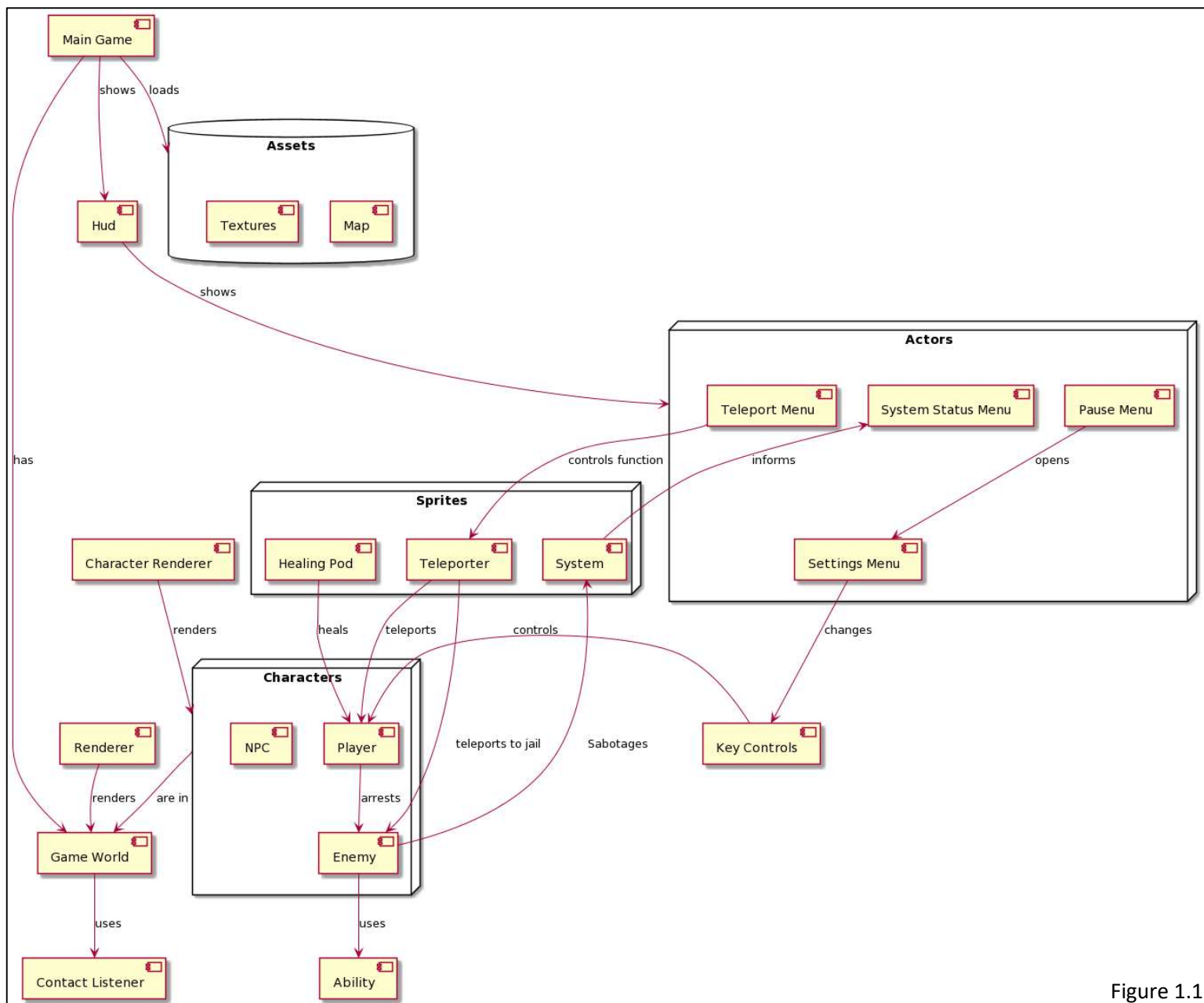
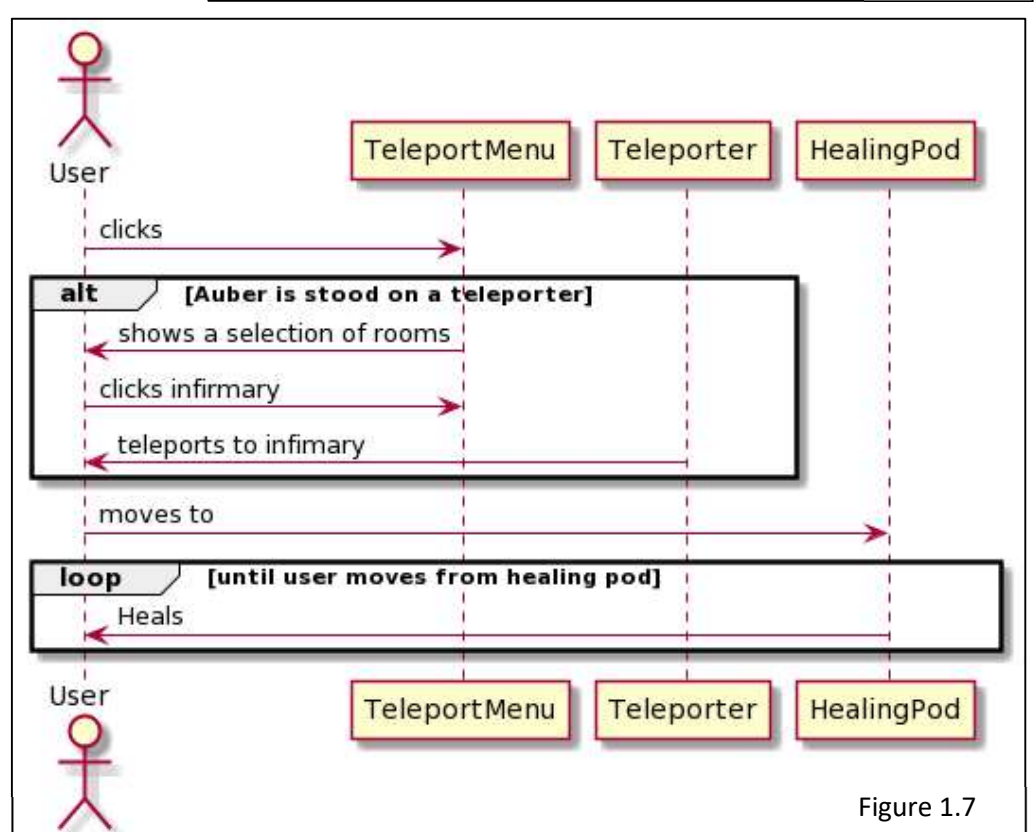
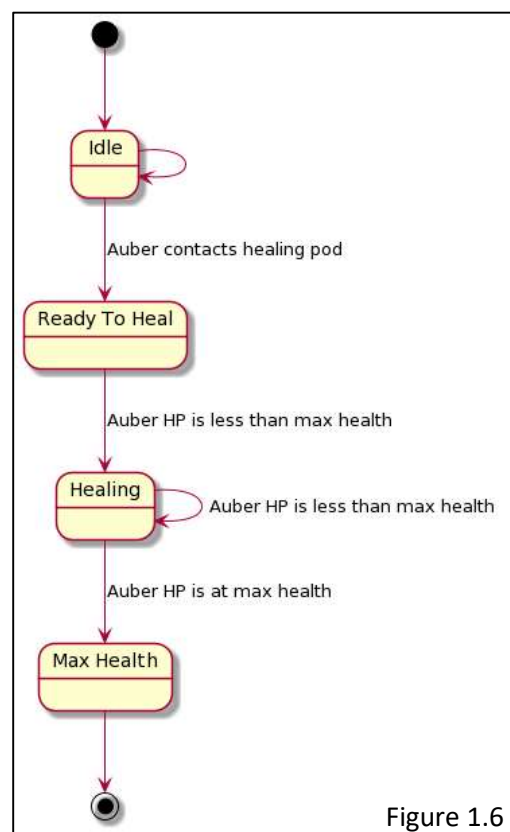
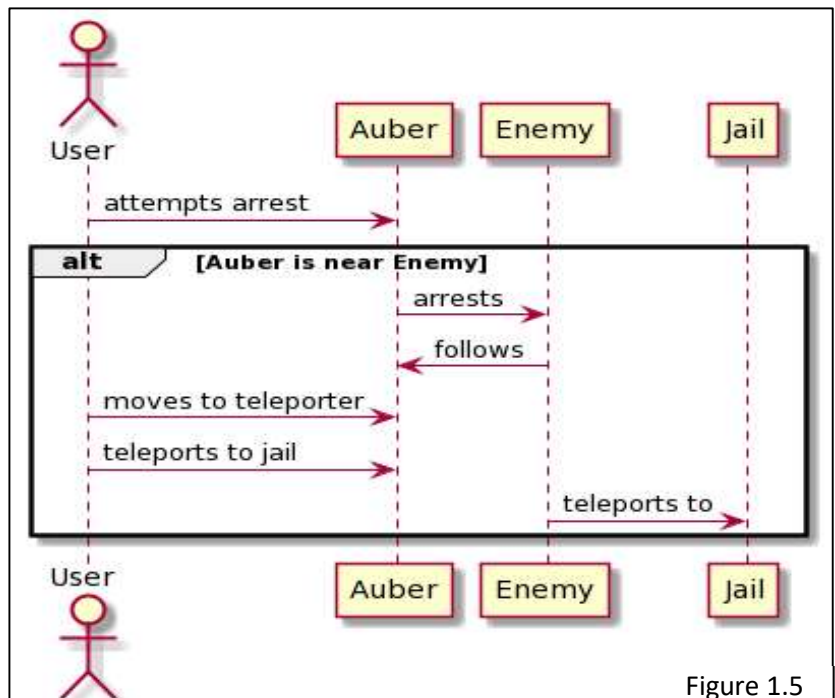
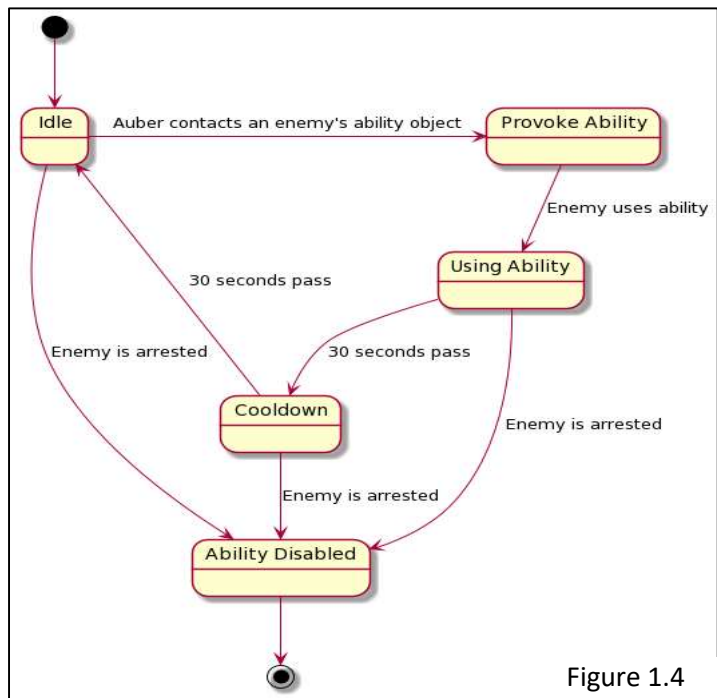
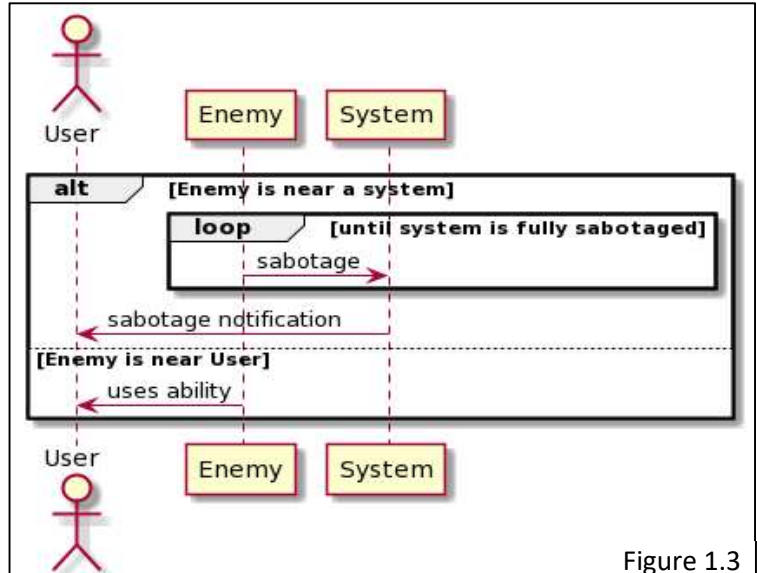
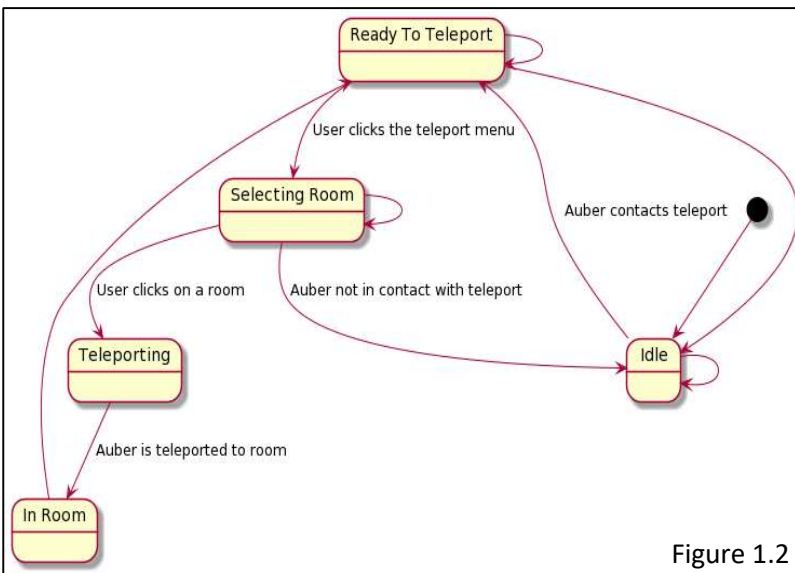


Figure 1.1





The architecture employs an object-oriented paradigm where objects encapsulate attributes and behaviours of entities. This was favoured over an entity component system where entities are composed of components which are used by systems to dictate interactions. This decision was based on the scope of the project. An object-oriented model provides a more rigid structure favoured as entities fit an inheritance model well and aspects of entities in this system do not change much at run time. Entity component systems work well for systems with many unique entities, however our system has relatively few. Not only this, but team members are more experienced using an object-oriented approach, so using this style allowed for better collaboration.

The concrete class diagram shows a much more detailed version of the abstract architecture with a few additions to better reflect the implementation. Figure 2 shows the inheritance relationships of entities which were added to reuse code. It also shows aggregation and composition relationships that represent how communication between objects was implemented. Methods, attributes and their visibility are shown to represent objects behaviour and what can be accessed by other objects. Multiplicity of objects in relationships are also shown, highlighting the number of objects interacting in the system. Some additional objects were created to help facilitate interactions such as the Teleport\_process class which performs some of the teleporting tasks.

The Main Game is the main entity of our whole system, which is represented in an abstract form in Figure 1.1. This encompasses the whole game, and then further abstraction is shown in figures 1.2-1.7. It is implemented in our concrete representation by the Gameplay class.

Figure 1.2:

- Satisfies requirement UR\_Teleport
- Incorporates Teleport Menu, Teleporter and Player from Figure 1.1
- Implemented by Player, TeleportMenu, Teleporter, Teleport\_Process, ObjectContactListener in the concrete, Player
- The idle state in the abstract is represented by ObjectContactListener in the concrete to determine when the Auber can teleport, effectively replacing two parts of the abstract diagram

Figure 1.3:

- Satisfies requirement UR\_Sabotage, makes use of UR\_Abilities
- Incorporates System Status Menu, System and Enemy from Figure 1.1
- Implemented by SystemStatusMenu, Hud, System, Enemy and EnemyManager in Figure 2
- The concrete representation is more complex due to the number of classes involved in this requirement, as it borrows from another large requirement. The abstract diagram represents the process involved in the sabotage process and an enemy using their ability upon being found in a picture format, but the concrete shows the classes involved and specifically which classes do what.

Figure 1.4:

- Satisfies requirement UR\_Abilities
- Incorporates Enemy and Ability from figure 1.1
- Implemented by Enemy, EnemyManger, Ability, Player and ObjectContactListener
- This requirement lends itself to the above Figure 1.3 and closely follows the abstract diagram.

Figure 1.5:

- Satisfies requirement UR\_Arrest
- Incorporates Player, Enemy, Teleporter and TeleportMenu.
- Implemented by Player, Enemy, EnemyManager, Teleporter, Teleport\_Process, TeleportMenu, ObjectContactListener and jail in Figure 2

Figure 1.6:

- Satisfies requirement FR\_Healing
- Incorporates Healing Pod and Player from Figure 1.1
- Implemented by System, Player and ObjectContactListener from Figure 2
- This figure demonstrates the healing feature that is used in UR\_Teleport, which encompasses both the teleport and the healing systems of the game.

Figure 1.7:

- Satisfies requirement UR\_Teleport, makes use of FR\_Healing
- The teleport and healing system are grouped together as you teleport to the infirmary to heal: the abstract and concrete architecture are the same in this respect
- Incorporates TeleportMenu, Teleporter, Healing Pod, Player from Figure 1.1
- Implemented by Player, ObjectContactListener, Teleport\_process, Teleporter, TeleportMenu in the concrete