I developed the entirety of this project, but the main important parts of this project are the game world model, and the GUI.

Since I have developed the entire project, I am responsible for the entirety of the project. However, The two main components and packages of my program are adventure.model, which stores and is responsible for the logic behind the game components, and adventure.view which is responsible for the graphical components of this project.
This project is designed in a MVC manner, with my main controller class being instantiated in this project by World, as it holds and creates both event listeners, and references to both the model and the view. adventure.model is the model component, and adventure.view is the view

I believe that my grouping and splitting of packages into separate components, as well as what they are responsible for in this design is well laid out, allows for a large amount of customisation, and creation of many different types of game worlds. Specifically: I believe the split between Room, RoomSection, And the GameObject inheritance tree is an example of a well-designed split of responsibilities, and makes clear the definitions used to define all objects present in the game.

One of the ways I could improve this design is by making the controller separate from World. World serves as a decent enough implementation of the controller, but should really be used only for world creation, instead, this should be merged into world creation, and the world should be selectable from a higher controller class. Secondly, I could improve the design of the model by implementing more individual class logic classes, and expanding on the use of the player's inventory. For example, being able to access the inventory could allow weapons, more devices etc.

I was considering making the inventory system a composite inherited system, and making inventory an interface. But I realised that inventory both needed a concrete implementation, as well as the items contained in one inventory were not necessarily inventories. I could still separate the logic into a interface with a bit of extra work!