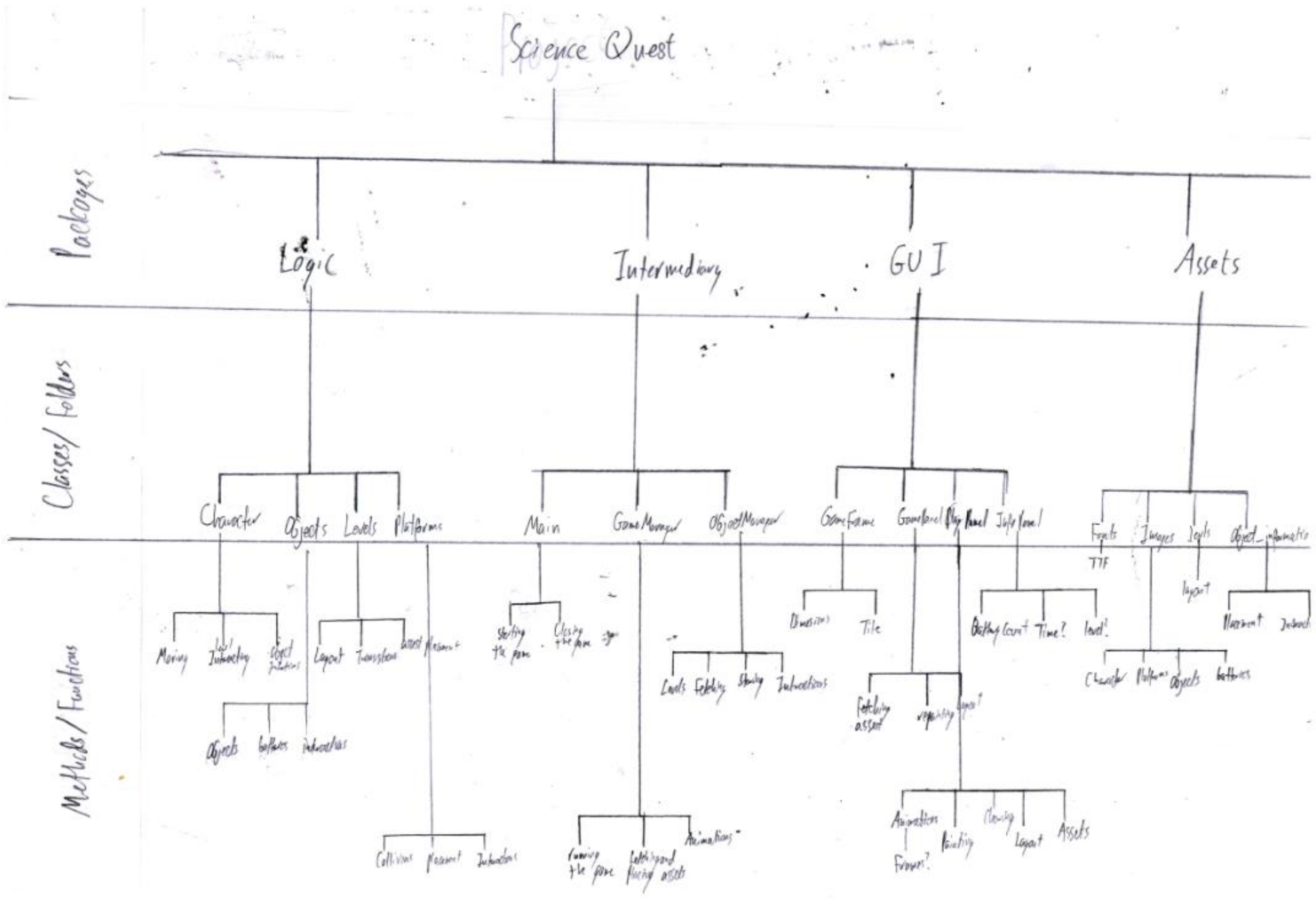
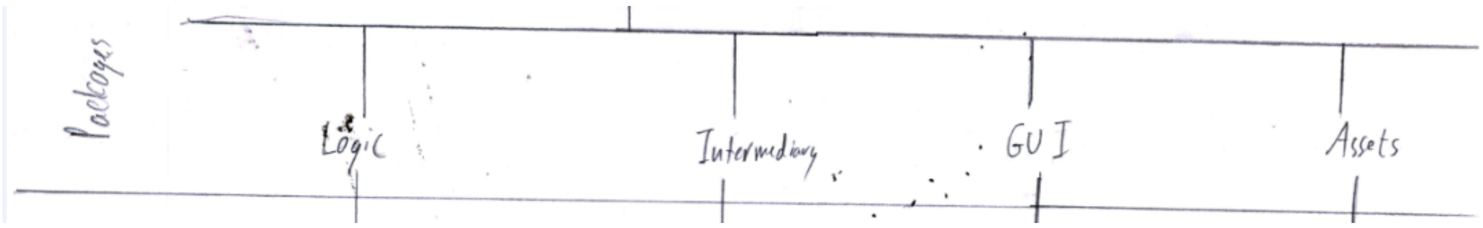
# Developing the Coded Solution- Modular Map

When faced with any programing task, it is important to think about the complexity of the solution. If the program begins to span over a few lines of code, it may become increasingly difficult to create a solution. At times like this it may become useful to use a modular approach. By breaking down the problem into feasible chunks, it becomes a lot easier to tackle. There is also the ability to have multiple coders working on a single program via different modules, although that will not be the case for this project.

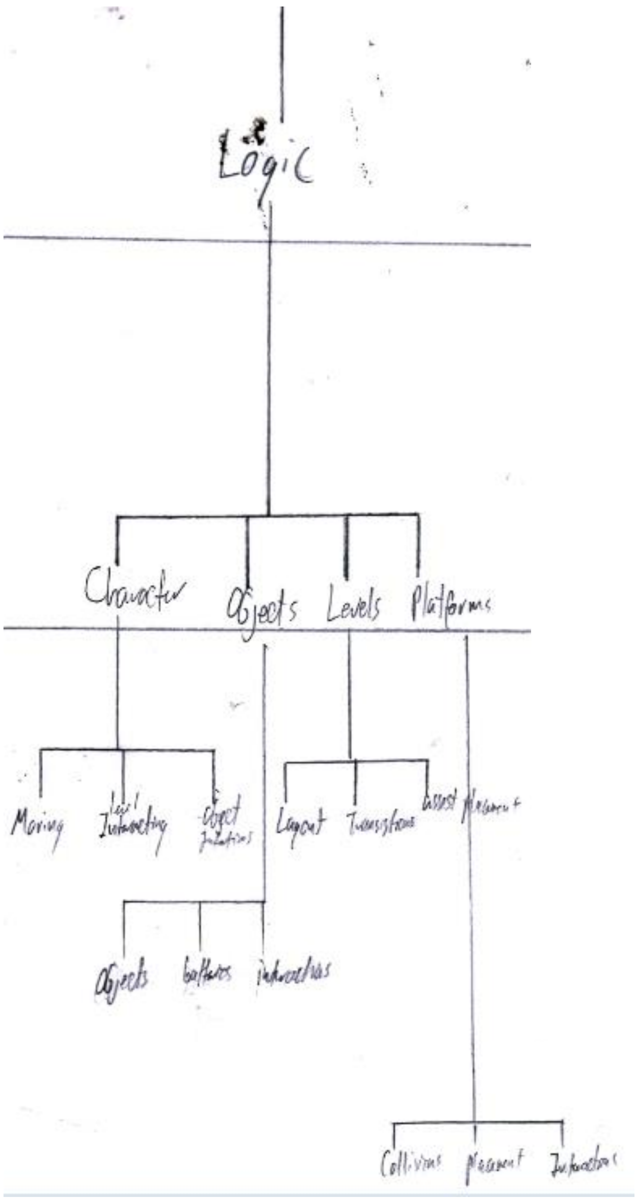


This is my modular map, it has taken my whole program and broken it down into sections. Not only does this allow me to structure the sections for easier implementation, but I can code each section at a time, allowing for a gradual build up to my final version by implementing sections according to function or importance. The modular approach allows me to also make small adjustments to one module without having to touch any of the others. In this document I will briefly go over the four packages that make up the first layer in detail, followed by showing and explaining other lower layers in less detail.



The first layer is dedicated to the packages of my program. I have split each aspect of my program into these four categories; Logic, Intermediary, GUI and Assets.

The first package is the Logic package. This package contains all the classes and folders that will deal with the logical processes that make up the whole game.



The second layer is dedicated to the potential classes that will make up the package. For the logic behind the game, I have split it into four main aspects; Character logic, Object logic, Level logic and Platform logic. Everything the user will do in the game will be governed by the logic behind these four classes.

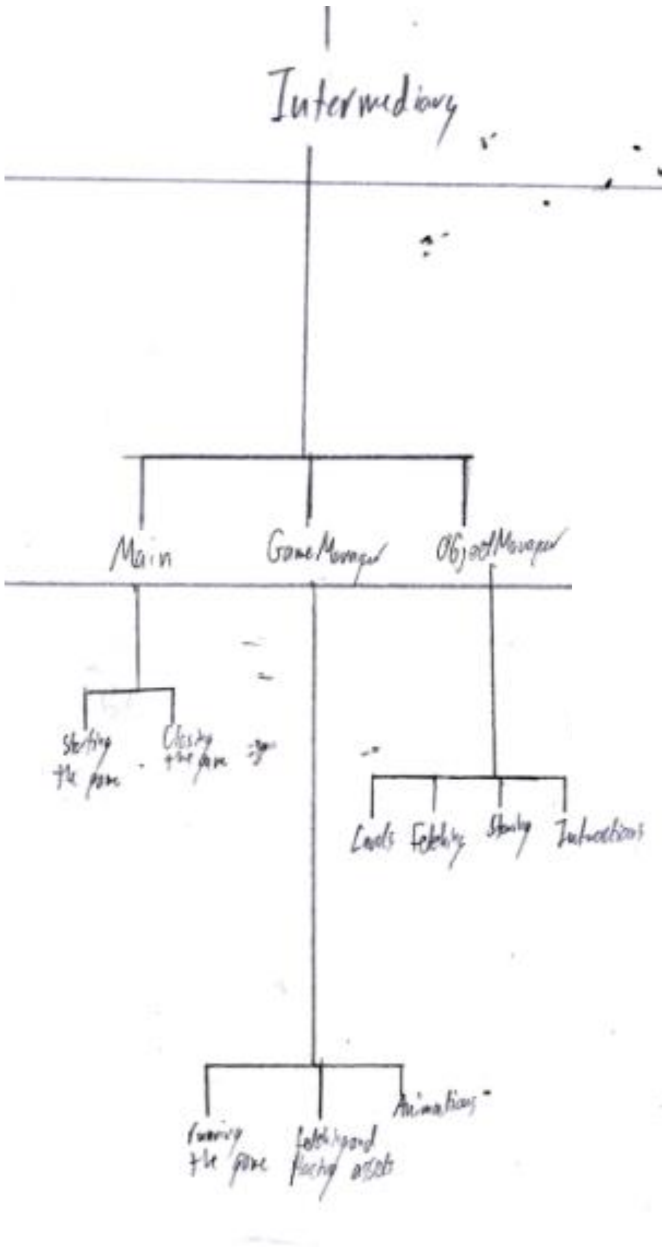
The last layer is given to the methods and functions in each class. Due to the detail of this layer, it is the most subject to change as development progresses. Each class breaks down as follows:

Character: moving, level interactions and object interactions.

Objects: objects/obstacles, batteries and their interactions.

Levels: layout, level transitions, the placement of assets in the level.

Platforms: collisions/ solid, placement method, interactions with non-player assets.



The second package is the intermediary package. The classes in this package deal with the logic created in the other package and acts as an interface between logic and GUI.

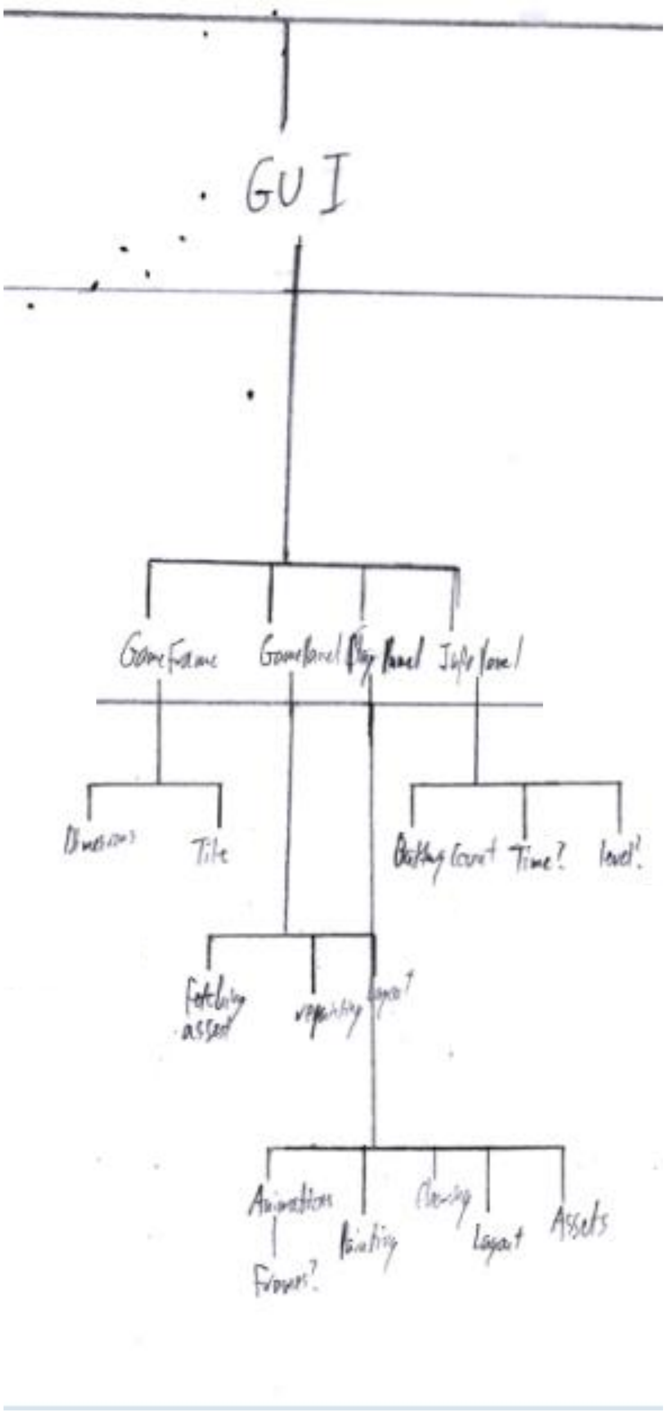
There are three classes under the intermediary package; Main, GameManager and ObjectManager.

Each class breaks down like the following:

Main: starting up the game, closing down the game.

GameManager: Running the game, fetching and placing assets and animations.

ObjectManager: Level placement, fetching objects, storage and categorisation and interactions with both the level and the character.



The Graphical User Interface (GUI) package deals with what the user will see when they play the game.

The whole experience is split into four classes: the GameFrame, GamePanel, PlayPanel and InfoPanel.

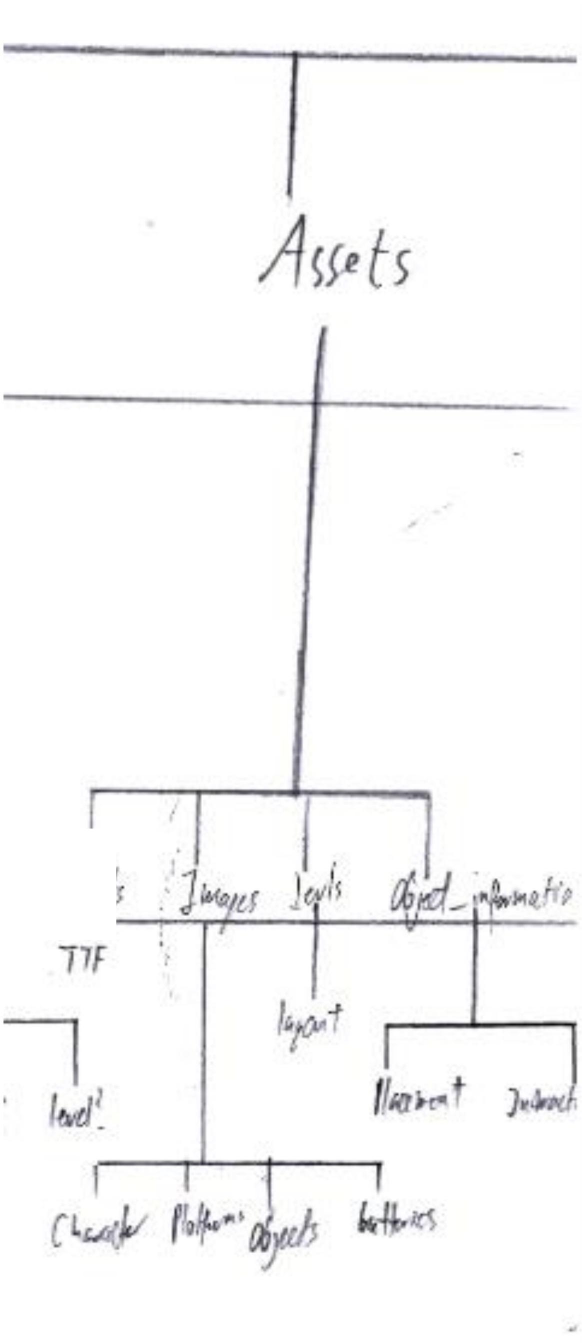
Each class controls part of the interface and graphics the user will interact with a see, as such there is a vast range of functions and methods to consider:

GameFrame: the dimensions of the frame, as it will affect the whole size of the game, how each space/tile will take up the frame.

GamePanel: Fetching assets, painting and repainting assets (animations), the placement and layout of assets.

PlayPanel: Animations and their frames, painting assets, clearing assets at the changing of a level, layout of assets in a level and how the assets will be placed/drawn on the panel.

InfoPanel: battery count, possible timer, possible level counter.



The final part of the first sections isn't actually a package. In a heading under which assets such as images and object information would be stored.

Rather than classes, there are four different aspects of the program that needs additional information supplied to it, these are as following:

Fonts in the form of True Text Files (TTF), seeing that I am going for a science theme, any text in the game (be it speech or instructions) should be in a font that shares a science theme.

Images are a major part of the game, objects, platforms, the character and batteries all will consist of images, either fetched or drawn onto the stage.

The levels information will be saved on text files to allow me to quickly implement them.

The final asset is object information. Both the placement of the objects in a level and the interactions with the character will have to be stored (probably in a text file), to allow me to quickly implement many objects.