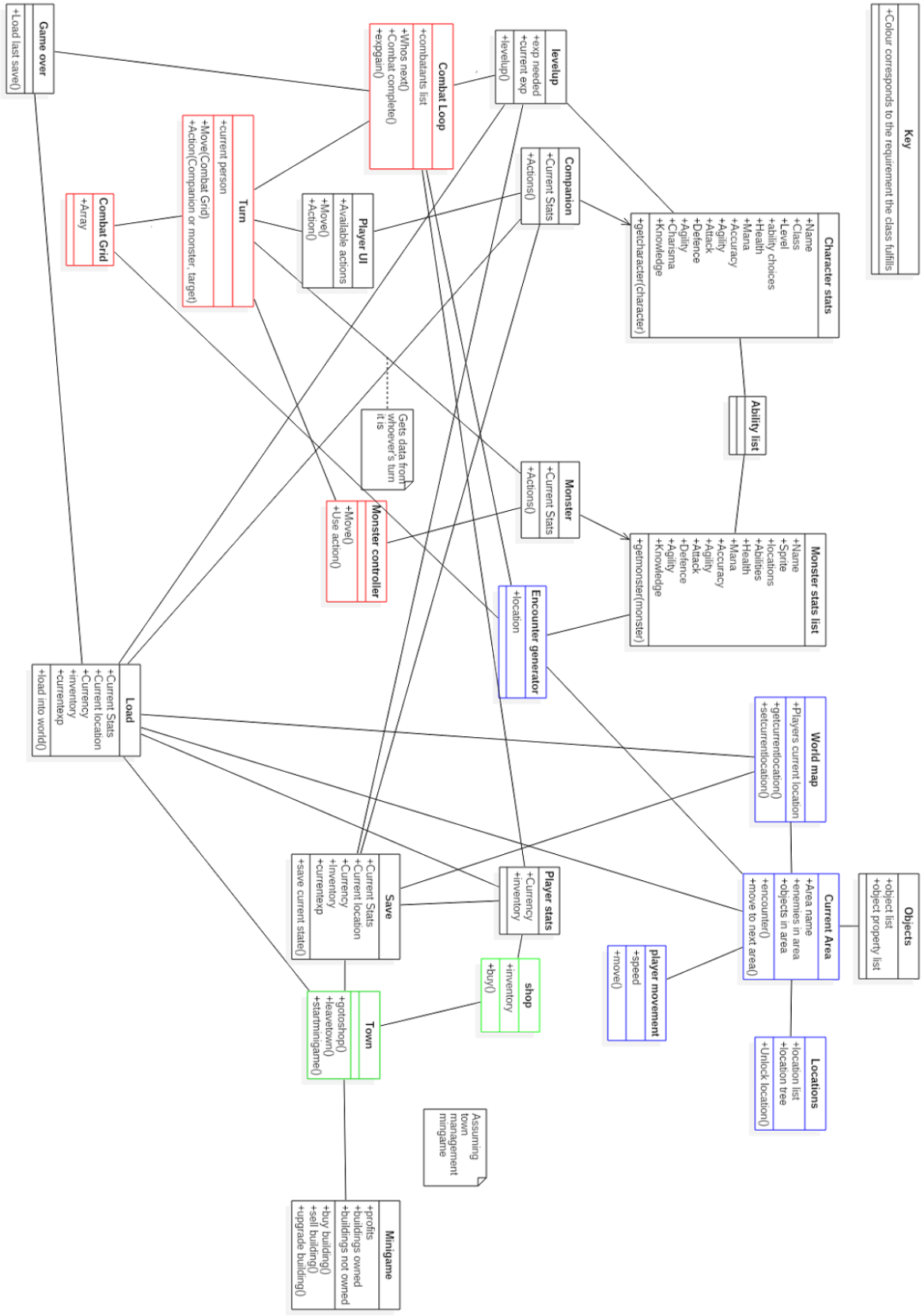
1.3 | Architecture

Our Proposed Architecture:



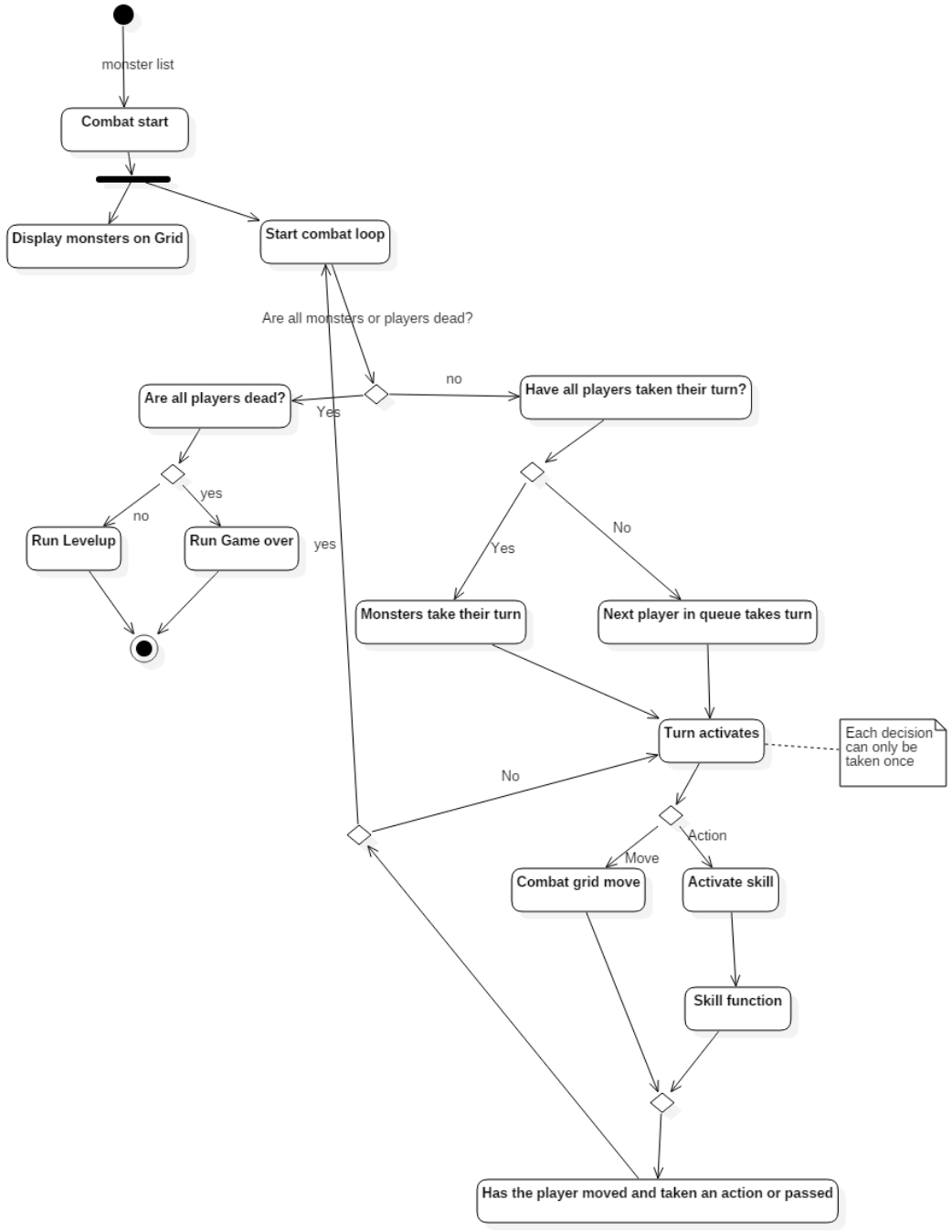
Architecture modelling tool

This was created in StarUML using the UML 2.0 standard. We did this so the model would be easy to understand for those familiar with UML, as it is the most widely used software modelling language. We decided to use StarUML as it was available for free, had a simple and easy to use interface that members of the team could quickly learn and, supported multiple export file types e.g. png and pdf so that diagrams could be quickly and easily uploaded to Google Drive (Our chosen collaboration platform) and viewed on Google Drive natively, without the need to download and then open in StarUML.

Architecture Elements

Our proposed Architecture is made of 24 classes which can be subdivided into five categories, with each category linking with requirements and once implemented fulfilling that requirement. The categories are as follows:

Combat system: (Requirement U3) Labelled on the diagram in red (figure 1). These classes will run the game's combat system. Centered around the Combat Loop class and interacting primarily with levelup and Encounter generator. Classes within the combat system are notable as they have no interaction with the Save and Load classes, this is because the player does not need to be able to save in the middle of combat. The Combat system is further defined in the behaviour diagram below.



Map: (Requirement U2) Shown in Blue (Figure 1). The Map classes handle the players movement through the world and their non-combat interactions, including their current location, available locations to move to and displaying their progress on the world map. These classes do interact with save and load to store the player's current location and any locations they have unlocked.

Town: (Requirement U4) Shown in Green(figure 1). The town classes handle the players shopping and spending. They interact with Load and Save as the player must be able to save while in town.

Load and Save: (Requirement U9) These two classes handle the player saving the game and loading previous saves. Load gets the player's current location, Current Stats, Currency, inventory and current exp. It then writes these to a file which is then read by save whenever the player reloads the game.

Support classes: These classes don't fit directly into the above categories, such as the minigame class, Character and monster stats and, the ability list. These classes are also primarily helper classes which provide functionality for use by the other classes.

Architecture Justification

Explaining colours: As mentioned above different parts of our architecture model are shown in different colours. These colours correspond to colours within the requirements model and will correspond to elements of our concrete architecture. This allows us to trace what the purpose of each class is and which requirement it fulfills.

Justification of classes:
Each class has a specific purpose and fulfills a vital task. Description of classes follows:

Class	Description/justification
Character stats	This class is the core class used to store data about a character such as in-game stats and abilities that belong to the character. It serves as the base class for companion which will store individual character information.
Monster stats list	This class will store the core data of what stats each monster in the game has and will serve as the base class of the instance class monster.
Ability List	This class is responsible for ability functionality, such as how much damage an ability deals and who the damage is dealt to. This will also store information about an ability.
Companion	This class is responsible for the stats of your current individual companion(s) and what actions they can perform. This inherits from character stats.
Monster	This class is responsible for the information about individual monsters and what actions they can perform as well as their current stats. This inherits from monster stats list.
Combat loop	This class is responsible for enacting the combat loop as shown in the combat loop diagram, it controls whose turn it is and also checking for the end of the combat instance.
Turn	Responsible for controlling a single characters move and actions in one turn.
Combat grid	Stores the state of the combat grid (10x3 area where the characters do battle) and also player movement
Encounter generator	The encounter generator is responsible for creating each encounter with monsters that a player character can face. It stores the location and generates the encounter.
Player UI	This class is what lets the player control their characters actions.
Monster controller	This class decides on what actions the monsters will take on their turns.
Level up	This is triggered when all monsters are slain, gives players xp and puts them back on the map
Game over	This is triggered when all the players die in combat (currently planned to reload the last save)
Current Area	Stores information about the current area that the player is located in and is responsible for checking if the player will move to the next area or if the player encounters some enemies in which case it invokes

	the encounter method.
World map	This stores the location of the player and can be used to change the location of the player, it will also be responsible for displaying the world map.
Locations	Responsible for keeping track of what locations are available and which locations have access to other locations.
Objects	This class stores information about in game objects and what properties they have. E.g. locked doors, npcs
Player movement	This class is responsible for the how/where the players moves and at what speed.
Player stats	Stores how much currency a player has and what is in a player's inventory
Load	This class is responsible for everything to do with loading from a save game such as where the player is, what the players current state is where to loads into the world etc.
Save	This class is responsible for creating save games and saving the current state of the world and characters in the area.
Town	This is the class which is responsible for town areas and entering in and out of buildings within the town
Shop	Responsible for the shop system, the inventory of the shop and controlling the buy/sell functionality when interacting with the shop
Minigame	This store everything to do with the minigame and will control how the minigame works and plays. This calls may need expanding if the minigame grows in complexity.

Justification of Structure

The structure is based around object oriented design and is an object oriented model. Each class has its own purpose and responsibilities. This is a good way to design the system so that it is modular and each part is independent of other parts, meaning that as we add functionality to one class, it does not break other functionality or the way that the classes interact with each other. We have used inheritance for the companion and monster classes as this lends itself to an inheritance model due to the fact that multiple characters with different attributes/actions will all need the stats present in the base classes. We have separated the classes by the categories mentioned in previous sections so it is clear how each class will interact with the other classes.