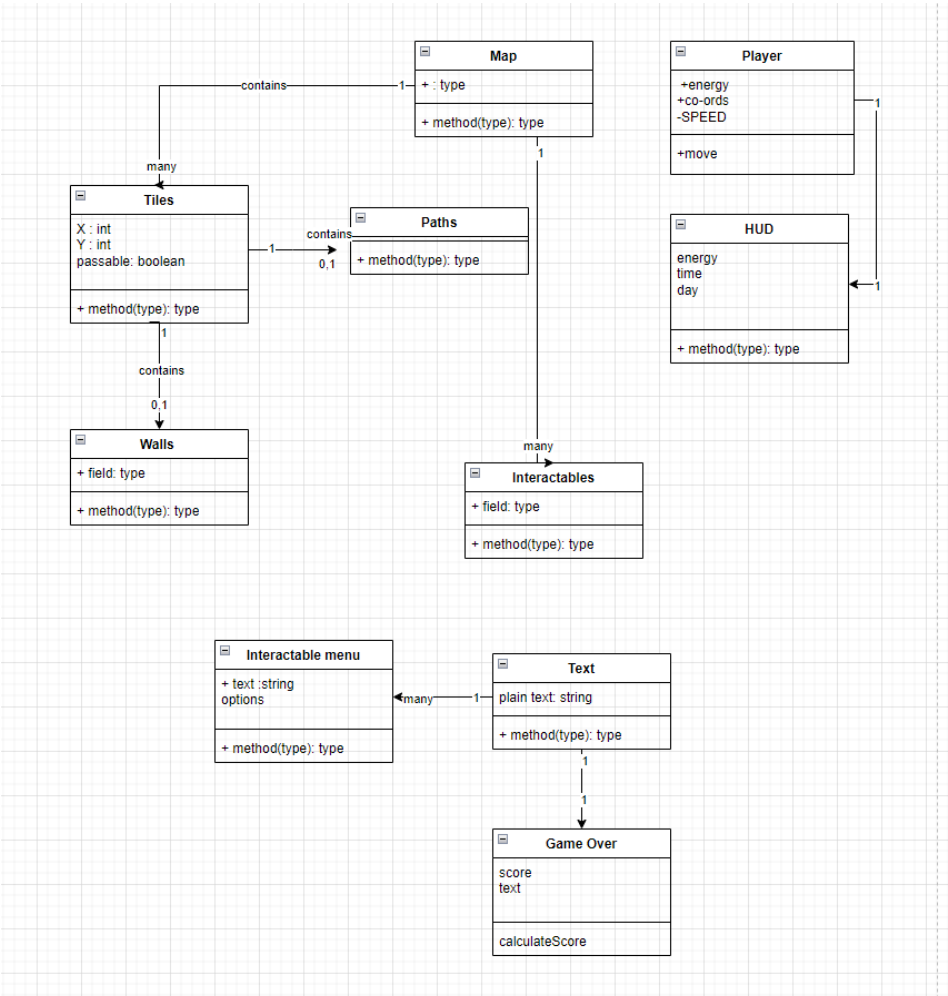
Architecture

Group 7

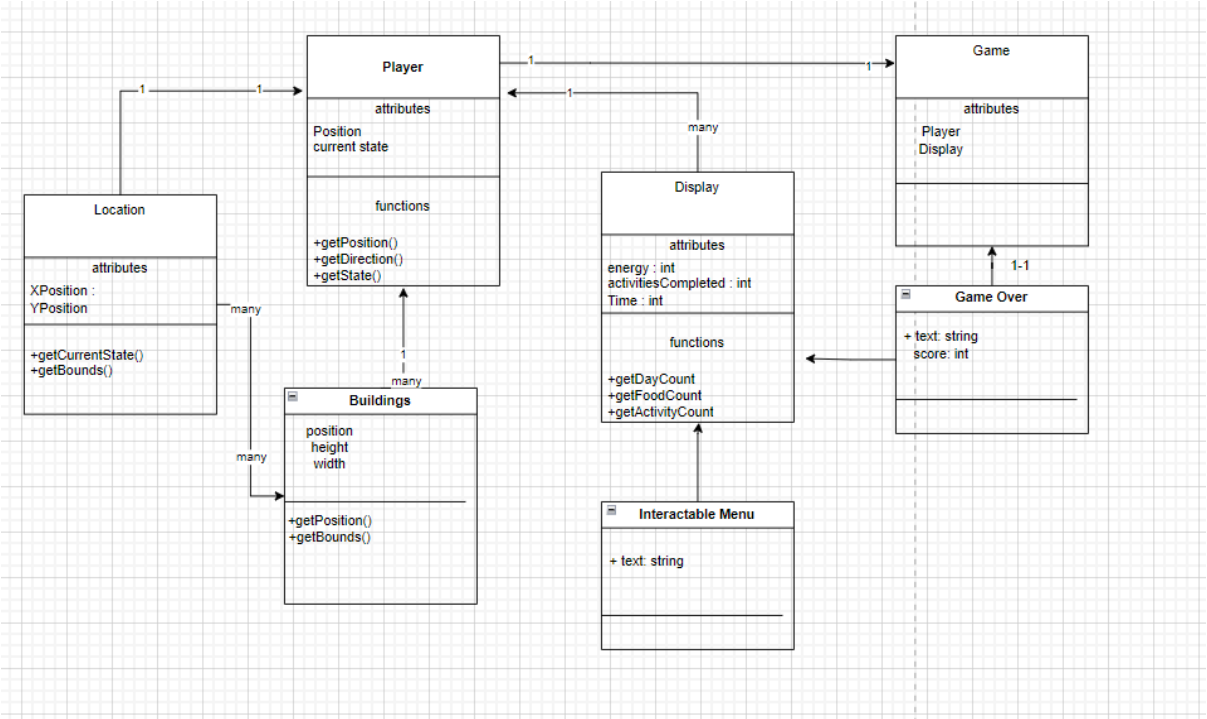
Charlie Meader, Sylvia Bulch, Emilija Dudko, Sam Laljee,Eisvinas Daujotas,Max Sweetman

For the architecture of our product, our team has used a combination of structural and behavioural diagrams to provide visual representations. The UML diagrams provided below have been created using the website ‘Draw.io’ and these diagrams will be translated into Java during implementation.



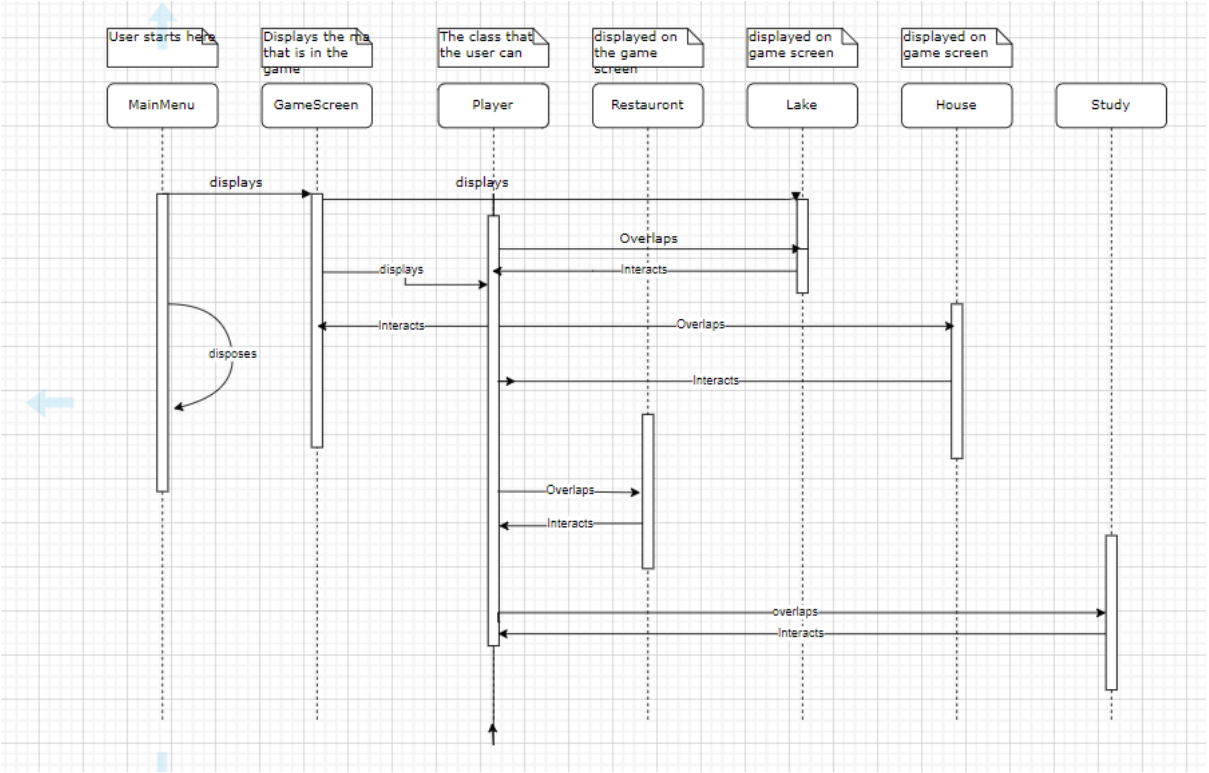
The diagram above is the display of our initial UML diagram, this shows how we initially organised the game into 3 categories, the map, the player and pop-ups. The map being all the world that the game would take place in, including the activities which the player would complete and walkable surfaces. The player which is the character that the user would use as well as the information about the game that would change: energy, time and day. Finally, pop-ups which is how we would choose the activity that the user wanted in the game as well as their final score at the end.

This initial design was a good beginning for our UML diagram however we needed to further connect these categories as this initial design needs to display the connections between these 3 categories and would be completely different when we started our implementation. This made us create a second design for our UML diagram, where we connected the categories and made the diagram easier to understand.



Our second diagram is a more advanced version of the original. As it is also grouped into three main categories. One for the display, one for the player and one for the player's location. Any relation information can most probably be found within these categories. As we develop the code, it's almost certain that the code will stray from the UML blueprints, however, it is a great starting point we can and will utilise.

Structural Diagram:



The structural diagram was created using the ‘Draw.io’ website.

This diagram was our initial thoughts on how our classes would interact with each other, this was done by creating a screen that displays the whole map with the buildings and the player, this is shown by the diagram by the game screen displaying the player, lake, restaurant, house and study. This allows us to see if the player is overlapping with each of these building classes and if so the building class allows interaction between them and the player, this also shows that we will be able to create some inheritance in our program allowing us to create multiple buildings more easily.

Whilst implementing our architecture into runnable code we encountered problems with using text and the interactable menu, this is shown in our structural diagram as we have removed any mention of the player interacting with a text menu, this is because we decided that the building classes themselves interact with the player. This allows us to fulfil the requirement of allowing the user to complete the game by using a keyboard.

For a clearer understanding of the link between the architecture and the requirements, for the following examples, the requirement code [in brackets] will be provided for each reference that we make. These can be linked to the requirements document for a better understanding.

For example, a functional requirement mentioned the need for the game to last seven days [1]. In our UML diagram, we have introduced the component of a HUD, where we have time

and days as components of it. The HUD is responsible for managing how long the player has played, directly addressing the functional requirement of time management, as well as allowing the game to not require a pause menu as the game progresses when the player interacts with the world which would change the HUD.

Another set of requirements was that there has to be at least 1 place to sleep, 1 place to study, 1 place to eat and at least 3 places to complete activities [4.1 - 4.4], to solve this we designed the interactables class, this allows us to easily separate each of these buildings on the map much easier as well as reducing the game's complexity in design allowing us to complete the implementation of the game faster

To show this directly, we can look at the structural diagram provided above. In the structural diagram, we have a parent class of map that is inherited by the player, interactables and tiles, this is deemed necessary to associate all the objects that a user will see in-game. The Parent class contains the coordinates so we can locate them in-game. This is important as it will allow us to easily find where they are in functions that require the player to move as well as functions that require the player to interact with individual objects.

The three child classes have been separated due to their individuality and significance to a player. Adding a layer of readability and understanding for all. For example, the 'player' is the only class that is required to move, 'interactables' are the only class that allows the user to study or complete activities, and 'tiles' are the class responsible for the creation of traceable paths and impassable walls. The 'Hud' was created to display the main attributes that the user needs during the game's playtime: energy, time and day.