Architecture

Cai Hughes<cabh500@york.ac.uk>
Ben Slater<bs1463@york.ac.uk>
Adeola Adeniji<aa3098@york.ac.uk>
Mathew Riedy<mr1723@york.ac.uk>
Riad Kasmi<rmk526@york.ac.uk>
Simon Konieczny<sk2144@york.ac.uk>

Architecture

The implementation of the product will require the use of object-oriented design techniques to create the program. This entails setting up multiple classes for each component of the game which can be utilised by a main java file. The modularity of this architecture design will assist in making troubleshooting and collaboration easier, as well as making it simpler to add additional features if the customer requests them or if their requirements change.

To start, we needed to understand what classes we might need for the design, so we created Class Responsibility Collaborator (CRC) Cards to identify what classes we may need, the information they need to know, actions they must be able to complete and other classes they may need to interact with.

Initial Designs

Initially, our CRC cards and subsequent class diagram only considered a generic class, "**Event**" the types for all map locations that didn't have any way to specify what type of location it was and had no way to differentiate what type of actions should occur when interacting with different locations.

In an attempt to remedy this, we created a second design with a new abstract class, **MapInteractable**. We also changed the Event class to be an abstract class, and created 3 new classes, **Activity**, **StudyArea** and **RestArea**. Activity and StudyArea are subclasses of Event and RestArea is a subclass of MapInteractable. This allows each of these subclasses to override their methods, allowing different interactions between each interactable map location.

This approach, although better, would force the scoring to be decided by the activity and not the player, which would not allow the points the player earned to change based on other factors such as the facilities they used, how many times they used each type of facility or the order in which they used each type of facility.

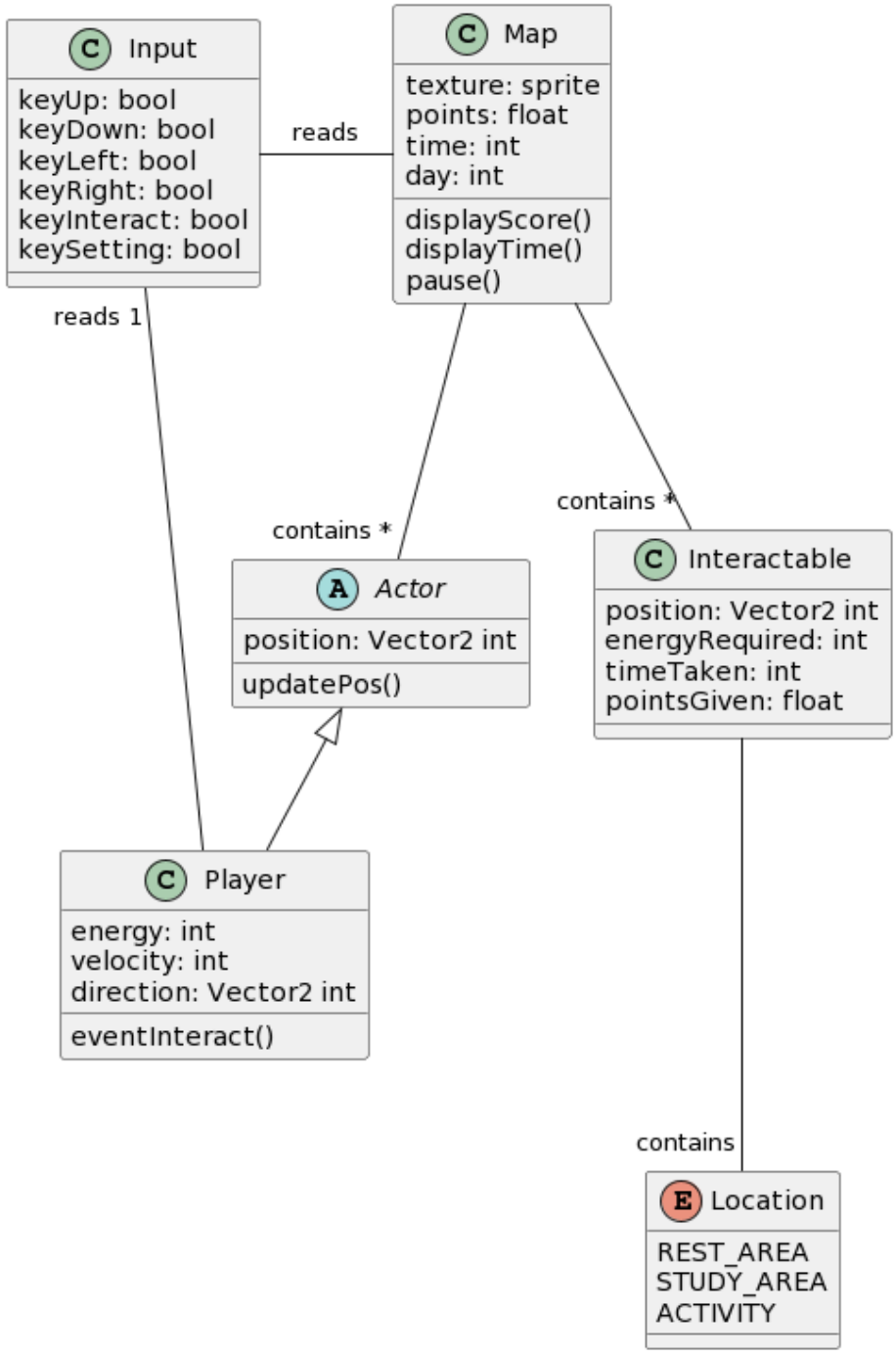
This led us to create another design, similar to the first one, where there is only a generic class "**Interactables**", however this time, we created an enum, "**Location**", containing constants describing the different types of locations the game contains. This allows us to assign a constant from the enum to each instance of the Interactable class, which allows differentiation between the different types of interactable locations and also allows the player to be in control of the result of these interactions allowing for better tracking and scoring variability.

Below are links to the CRC cards in interim versions of the class diagram:

- CRC cards
- Initial class diagram
- Second class diagram

Class Diagram

After considering the classes we will use for our design and their responsibilities, we created a UML class diagram using PlantUML to show the classes that will be included in the implementation, the relationships between them and key attributes and methods that the classes require..



Some of the classes contain sprite and vector data types. These are implemented by LibGDX, the Java game development framework the project will be using.

Map

The Map class contains and displays everything that exists within the game world.

Attributes

- ➢ **sprite** texture - The textures used for the map.
- ➢ **float** points - The score for the current game.
- ➢ **int** time - Stores the time of day for the current.
- ➢ **int** day - Stores the current day in the week.

Methods

- ➢ displayScore() - displays the score.
- ➢ displayTime() - displays the current date and time left in the game.
- ➢ pause() - pauses the game.

Interactable

The interactable class is used for the activities, study areas and rest areas placed throughout the map.

Attributes

- ➢ **Vector2 int** position - A 2D vector storing the position of the interactable.
- ➢ **int** energyRequired - The amount of energy required to use the interactable.
- ➢ **int** timeTaken - The amount of time required to use the interactable.
- ➢ **int** pointsGiven - The amount of points the interactable gives.

Location

The location enum stores values defining the type of interactable locations the map can have. These are to be used with the interactable class to differentiate between the type of interactables and their functions when the player interacts with them.

Values

- **REST_AREA** - used for rest areas
- **STUDY_AREA** - used for study areas
- **ACTIVITY -** used for activities

Actor

The Actor class is used for all the characters in the game. This class can be built out to include the player character as well as other non-playable characters in the game if the requirements are updated to be in need of them.

Attributes

- **Vector2 int** position - A 2D vector storing the position of the actor in the world.

Methods

- updatePos(**Vector2 float** position) - Updates the position of the actor.

Player

The player class is a subclass of the Actor class, it represents the class the player of the game will play as. It has limited energy per the user requirements that can be expended through events.

Attributes

- **int** energy - The amount of energy the player has.
- **float** speed - The speed the player moves at.
- **Vector2 float** velocity - A 2D vector storing the velocity of the player.
- **Vector2 int** direction - A 2D vector storing the facing direction of the player.

Methods

- eventInteract() - performs an interaction between the player and Interactable object in the game world.

Input

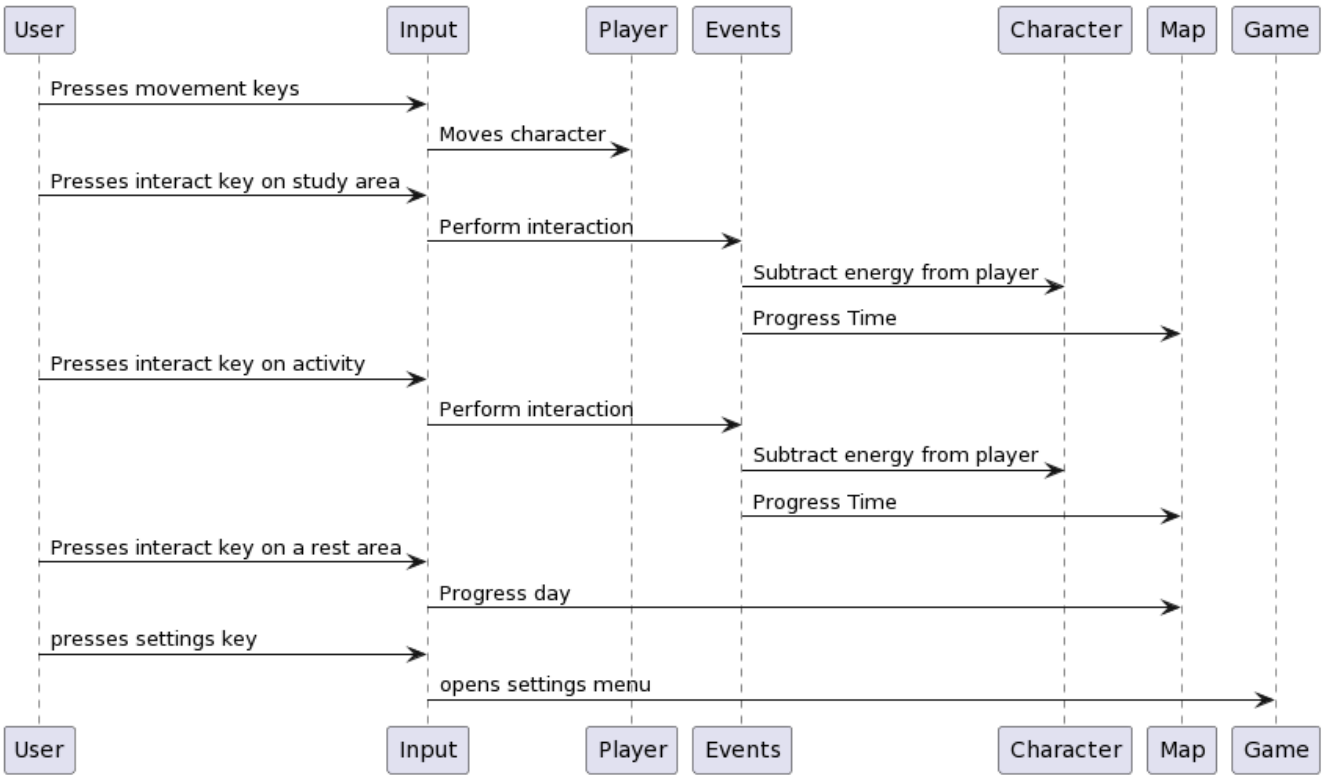
The input class defines the ways in which the user can interact with the game, With keypresses being represented as a boolean value.

Attributes

- **bool** keyUp - Stores the press state of the up movement key.
- **bool** keyDown - Stores the press state of the down movement key.
- **bool** keyLeft - Stores the press state of the left movement key.
- **bool** keyRight - Stores the press state of the right movement key.
- **bool** keyInteract - Stores the press state of the interact key.
- **bool** keySetting - Stores the press state of the pause key.

Sequence Diagram

A sequence diagram was additionally created to show some of the expected interactions between the objects in the system during runtime. It was created using PlantUML and shows our expected implementation of gameplay in order to adhere to our collected User Requirements.



User

The user has the ability to move around the map using the arrow keys, and can interact with areas in the environment. These interactions can be multiple different events each with their own effects on the user character.

Input

On input from the user, different methods are called which progress the game. This can be moving the player character, interacting with events, and opening the settings menu.

Events

The requirements for the game are to have multiple different types of events. There needs to be space to study, places to exercise/socialise, and one place to sleep. These events update both the character and map classes.