Architecture

Cohort 1 Group 1

Charlie Piper, Chris Oulton, Ella Daramola, Dillon Anthony, Kevin Thomas,
Shirin Sitara Alok Kumar, Tom Haslam

Class Responsibility Collaborator (CRC) Cards
We need to determine what classes we need for the game, so we made CRC cards to help figure out what classes are needed. Each card shows what actions the class must be able to perform and what other classes that class can interact with.

Class: HeslingtonHustle	
Responsibility: - Creates an instance of each of the GameScreens class and is responsible for navigating between the screens - Creates an instance of the AppPreferences class	Collaboration: - GameScreen - AppPreferences

Class: GameScreen	
Responsibility: - Responsible for all the screens such as character selection and the ending screen - Creates an instance of the Play class	Collaboration: - HeslingtonHustle - Play - Player - Activity

Class: Play	
Responsibility: - Represents the screen where main gameplay takes place - Allow for transitions between maps	Collaboration: - GameScreen - Player - Activity - GameStats

Class: Player	
Responsibility: - Represents the player character in the game - Allows movement around the map	Collaboration: - Play - Activity

Class: Activity	
Responsibility: - Knows the amount of time taken for each activity - Knows the score added for each activity - Has the various types of activities that can be performed	Collaboration: - GameScreen - Play - Player - GameStats

Class: GameStats	
Responsibility: - Stores and updates all the stats such as time, energy, score, and day	Collaboration: - Play - Activity

Class: AppPreferences	
Responsibility: - Manages the preferences of the game such volume	Collaboration: - HeslingtonHustle

The CRC cards and our original class diagram [1] were made after the supervisor meeting and were created with the user and system requirements in mind and how each one that was set out would be achieved, this allowed us to get a broad idea of how the system should work.

Once development began and our game began to evolve overtime we made a few adjustments to our classes set out before in the CRC cards and the original class diagram, our biggest adjustment being converting the GameScreen classes into multiple separate classes each with a singular responsibility, this is because it allows for code that is easier to change, test, extend, and understand. An example of one of the new screen classes that were added is the EndScreen class which is responsible for showing the user the score they got at the end of the game. The new class diagram can be seen below.

<u>Class Diagram</u>
After creating classes, determining their roles and how they interact with other classes we created a UML class diagram using Plantuml to show the classes that will be included in the implementation, the relationships between the classes, and key attributes and methods that each class has.

HeslingtonHustle
The HeslingtonHustle class is responsible for managing all the screens, it switches between the screens using the changeScreen() method.

CharacterScreen
The CharacterScreen class is the class that represents the screen the player of the game will use to select the character sprite that they want to play using the show() method. This meets the UR_CUSTOMISATION and FR_CHARACTER_SELECTION requirements.

EndScreen
The EndScreen class is the class that represents the screen the player will receive at the end of the game, it will display the final score they got using the show() method. This meets the UR_SCORE, FR_GAME_END, and FR_GAME_END_STATS requirements.

LoadingScreen
The LoadingScreen class is the class that represents the screen that is used while loading resources or preparing for the main menu.

MainScreen
The MainScreen class is the class that represents the main screen of the game, it handles all the user interactions using the MainScreen() method and creates an instance of the Play class using the show() method. This meets the FR_GAME_START and FR_GAME_QUIT requirements.

MenuScreen
The MenuScreen class is the class that represents the main menu screen used in the game, it provides the option for starting a new game, accessing preferences, and exiting the game. It does this through the show() method.

PreferencesScreen
The PreferencesScreen class is the class that represents the preferences menu section of the game, it allows the player to adjust game settings using the show() method. This meets the UR_PREFERENCES requirement.

Play
The Play class is used show the gameplay, it is responsible for loading the current map you are on using the loadMap() method, swapping you between maps using changeMap(), and setting the initial player position on the map when the map is first rendered using the setPlayerPosition() method.

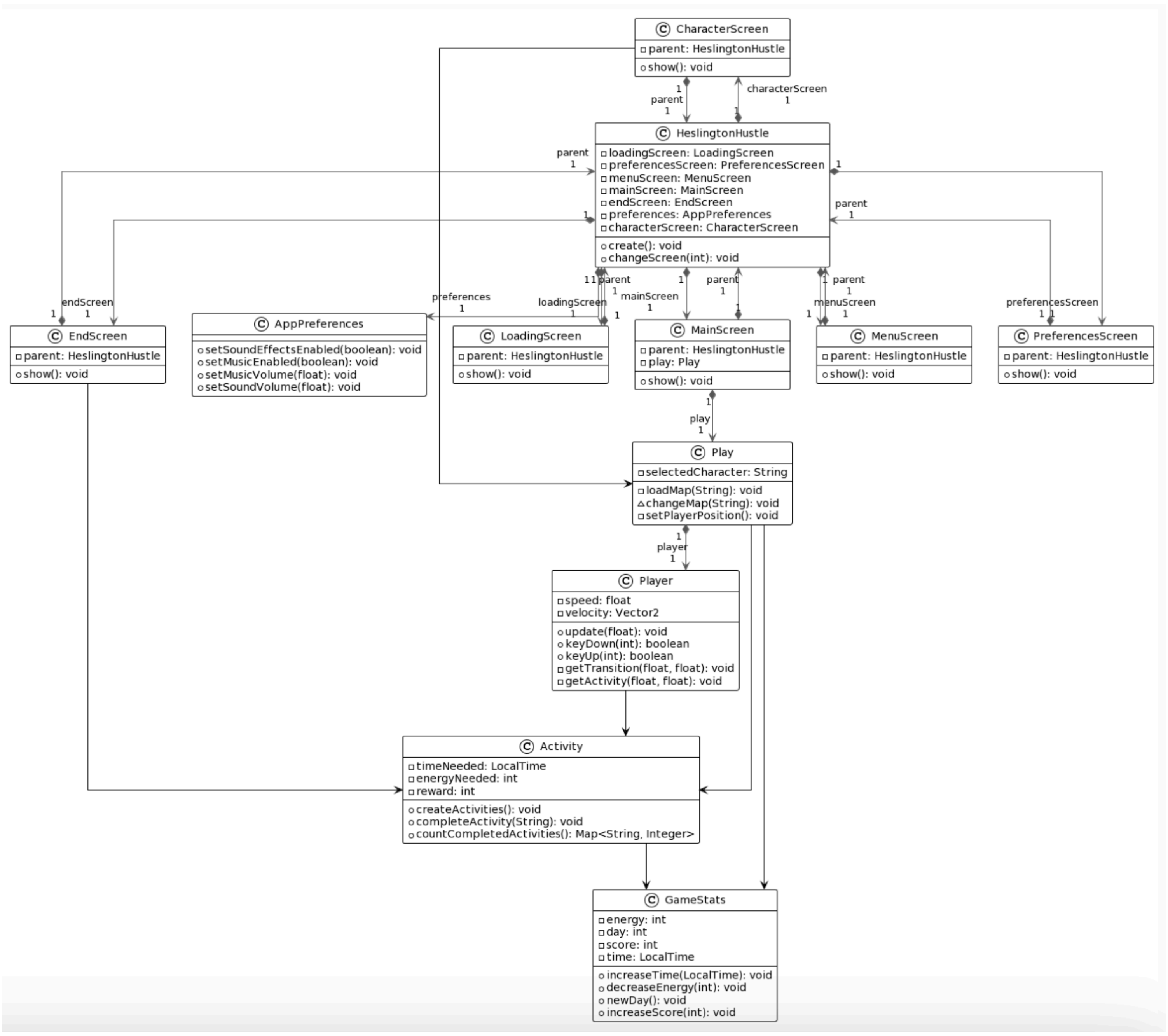
Player
The Player class is the class the player of the game will use while he plays the game, he is controlled and moved using the WASD and the arrow keys this is done through the keyUp() and keyDown() methods, he will be able to interact with the map through special tile blocks that will allow him to perform activities such as sleeping and studying. This meets the UR_MOVEMENT, UR_CONTROLS, FR_CHARACTER_MOVEMENT, FR_CHARACTER_COLLISION and FR_CHARACTER_INTERACTION requirements.

Activity
The Activity class is the class that the player will interact with to perform activities around the map they interact via the completeActivity() method, there are four activities that the player can perform: study, relax, eat, and sleep each activity will progress time and expend energy by a different amount. It tracks the number of activities completed using the countCompletedActivities() method. This meets the UR_OBJECTIVE and UR_CHOICES requirements.

GameStats
The GameStats class is the class that stores all the game counters such as energy, score, day, and time. Each score is also updated in the class each time an activity is completed. This meets the FR_STATS, FR_STATS_UPDATE, FR_STATS_RESET and FR_STATS_SHOW requirements.

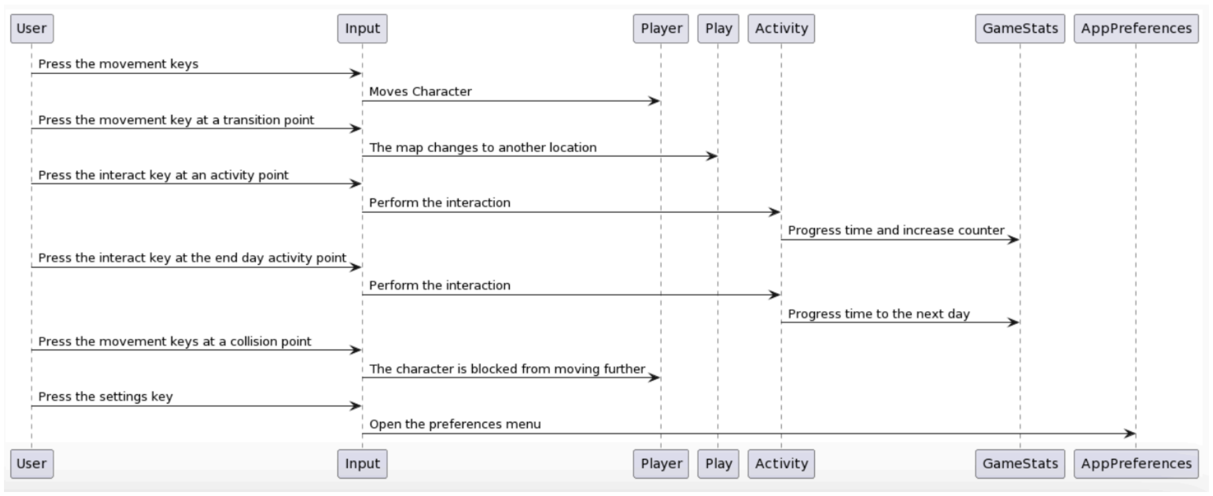
AppPreferences

The AppPreferences call will allow the players of the game to control the volume of the sounds he hears in the game such as music and other sound effects. This meets the UR_PREFENCES requirement.



Sequence Diagram

A sequence diagram was also created to show the expected interactions between the objects in the system during runtime. It was created using Plantuml and shows the expected flow between the user doing an action and the effect it has on the game; these sequences should achieve the goals that were set out in the user and system requirements section.


User
Input
Player
Play
Activity
GameStats
AppPreferences
Press the movement keys
Moves Character
Press the movement key at a transition point
The map changes to another location
Press the interact key at an activity point
Perform the interaction
Progress time and increase counter
Press the interact key at the end day activity point
Perform the interaction
Progress time to the next day
Press the movement keys at a collision point
The character is blocked from moving further
Press the settings key
Open the preferences menu

User

The user can move around the map using the movement keys (WASD or arrows), and can interact with the map at the activity points.

Input

Depending on the input from the user a different method will be called, for example moving the player, transitioning the maps, and opening the preferences menu.

Activities

Every time an activity is completed the game should be progressed in this case this will be increasing the time and counter by a varying amount depending on what activity was completed.

Requirements completed

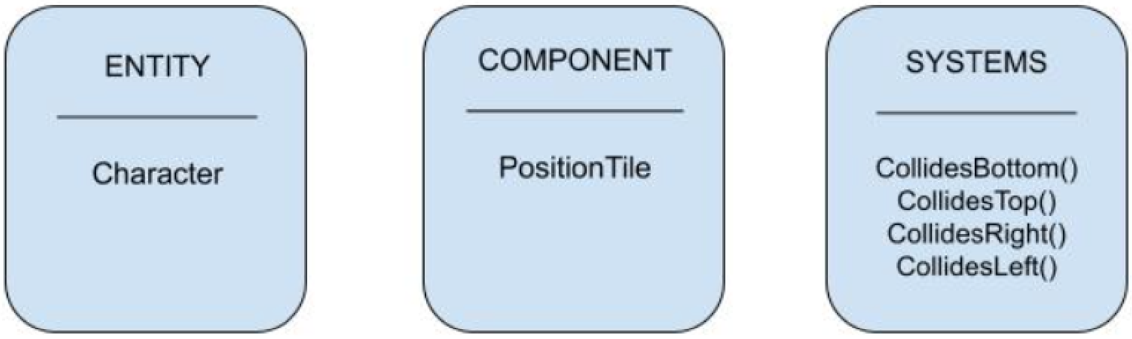
- The ‘Press the movement keys’ sequence fulfils the UR_MOVEMENT, UR_CONTROLS, and FR_CHARACTER_MOVEMENT requirements.
- The ‘Press the movement key at a transition point’ sequence fulfils the UR_MOVEMENT, FR_CHARACTER_INTERACTION, and FR_CHARACTER_MOVEMENT requirements.
- The ‘Press the interact key at an activity point’ sequence fulfils the UR_CHOICES and FR_CHARACTER_INTERACTION requirements.
- The ‘Press the movement key at a collision point’ sequence fulfils the UR_MOVEMENT, FR_CHARACTER_INTERACTION, and FR_CHARACTER_MOVEMENT requirements.
- The ‘Press the settings key’ sequence fulfils the UR_PREFERENCES requirement.

System Architecture

We followed the Component Entity Style of system architecture for the development of our game [2]. This style is an extension to OOP (object-oriented programming) which is a concept the team was already familiar with. We used 3 main software tools:- IntelliJ, LibGDX and Tiled.

Based on the requirements - user and system, the main nouns/noun phrases were used to build entities such as - characters, maps and status bars. Components are data types consisting of a unique behaviour assigned to an entity. They are reusable modules that programmers attach to the entities, providing behaviour, functionality, and appearance, forming an entity [3]. These are then implemented by giving them logical systems to adhere to.

A simple way to showcase the architecture is with an example:



An entity - Player has multiple components such as the position of the character, the appearance, speed and velocity of the character, etc. Different components of an entity are implemented using different systems, for example, the movements are controlled by the - KeyDown(), and KeyUp() functions, and the collision and transition tracking system is implemented using functions such as - isCellBlocked(), isCellTransition(), getTransition(), etc.

In this manner, a sprite from LibGDX gets moved around maps made using Tiled by rules written in Java (IntelliJ).

References:

[1]: https://charliepiper.github.io/documents Figure 4

[2] : Fundamentals of Software Architecture by Mark Richards and Neal Ford (O’Reilly)
.
Copyright 2020 Mark Richards, Neal Ford, 978-1-492-04345-4.
https://ebookcentral.proquest.com/lib/york-ebooks/detail.action?pq-origsite=primo&docID=60
29037

[3] : https://www.simplilearn.com/entity-component-system-introductory-guide-article