ARCHITECTURE

Group 3

Liam Martin

Aaliya Williams

Lucy Crabtree

Kai Nichol

Sammy Hori

Tim Gorst

Zac Ribbins

Introduction To System Architecture:

A key part of visualising the system of the game was done via Unified Modelling Language (UML) and PlantUML due to its robust and straightforward nature, as this allowed for detailed diagrams of specific components of the system, acting as a basis for the architecture. For example, the sequence diagram aided with providing a rigid structure for how the system should behave based on user input, this aided in the subsequent creation of the class diagram and further implementation of the code.

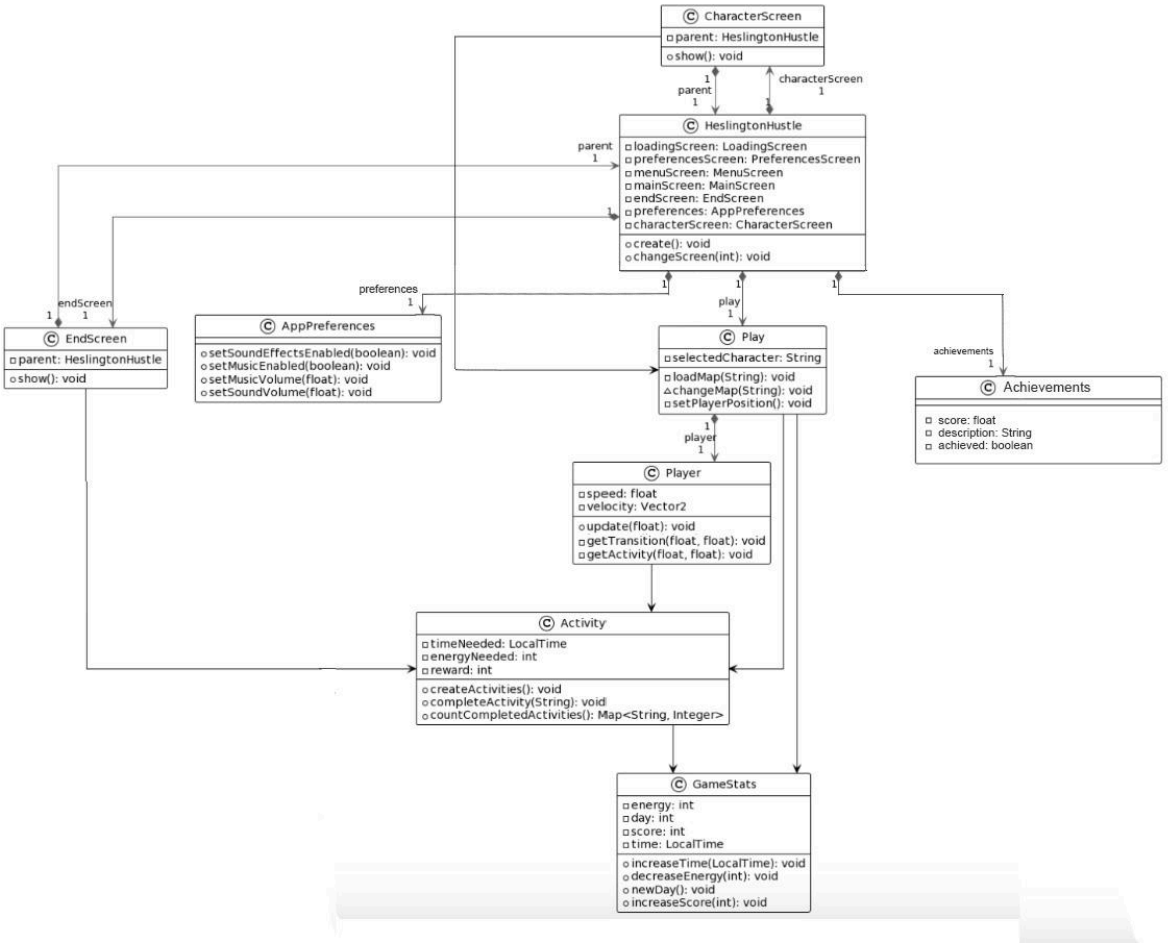
Class Responsibility Collaborator (CRC) Cards

One of the first courses of action was to determine what classes were needed 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.

The CRC cards (Figure 5) and our original class diagram (Figure 4) 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.

Class Diagram

As the it was predetermined that an Object Oriented Programing (OOP) would be used during implementation, 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; including inheritance between parent-child classes in conjunction with the key attributes and methods contained within each class.



New Class Diagram *(Available on website)*

Linking the Requirements to the Architecture

The following table contains individual classes, a brief description of said class and how it links to the pre established requirements:

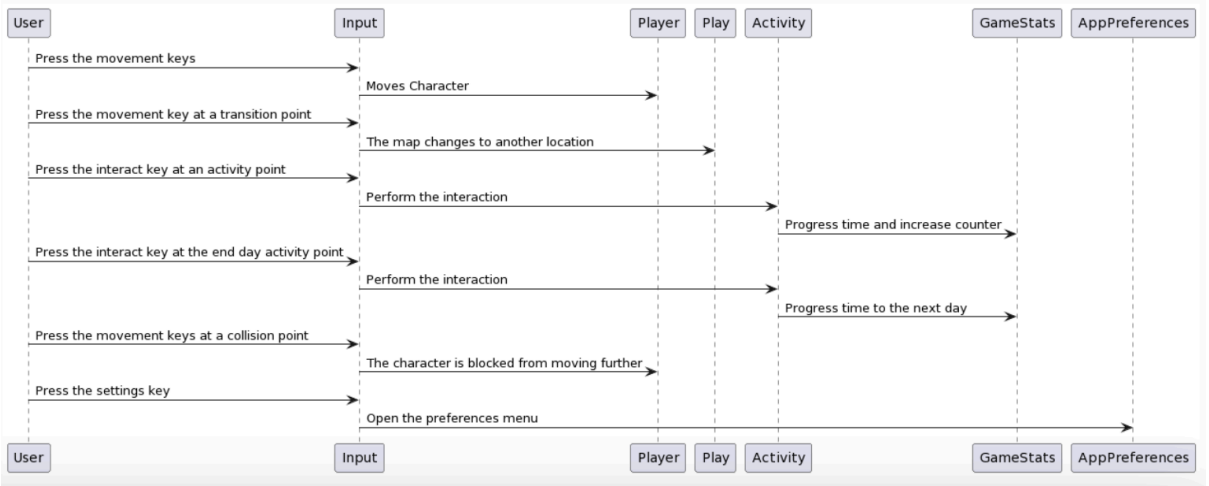
CLASS	DESCRIPTION	REQUIREMENT LINK

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.	UR_SCORE, FR_GAME_END, and FR_GAME_END_STATS
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.	UR_PREFERENCES
Player	The Player class is the class the player of the game will use while they play the game. The player moves using the WASD and the arrow keys, this is done through the keyUp() and keyDown() methods. They will be able to interact with the map through special tile blocks allowing the player to perform activities such as sleeping and studying.	UR_MOVEMENT, UR_CONTROLS, FR_CHARACTER_MOVEMENT, FR_CHARACTER_COLLISION and FR_CHARACTER_INTERACTION
LeaderboardScreen	The LeaderboardScreen class allows the player to see the scores and rankings of the top ten players.	UR_LEADERSHIP and FR_LEADERBOARD
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	UR_OBJECTIVE and UR_CHOICES

	a different amount. It tracks the number of activities completed using the countCompletedActivities() method.	
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.	FR_STATS, FR_STATS_UPDATE, FR_STATS_RESET and FR_STATS_SHOW
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	UR_PREFENCES
Score	The Score class keeps track of the points the player achieved and allocates them a score based on the standard British university grading system, i.e. 40-49 for a 'Third Class', 50-59 for a 'Second Class' and so forth.	UR_SCORE and FR_STATS
Achievements	The Achievements class stores the streaks the player has achieved throughout the game, for example, the achievement: 'Fitness_Fanatic' which players are rewarded with if they engaged in physical activity at least 3 times in the week.	UR_ACHIEVEMENTS, FR_ACTIVITY_COUNTER and FR_ACTIVITY_STREAK

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

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.

Justification of System Architecture

The architectural style used followed an Object Oriented Programming approach as it was deemed the best fit in order to meet the preexisting requirements. Using OOP allowed for inheritance and encapsulation, two characteristics that were used extensively during the implementation of the code and this can be seen via the class diagram, detailing the class hierarchy of the system and thus the inherited methods within these specific classes.

Evolution of the System:

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 to convert the GameScreen class into multiple separate classes, each with a singular responsibility. This change allowed 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 achieved once they reached the end of the game.

With the introduction of the second assessment brief, there was a need to change certain components of the system as thus, these changes occurred in the form of updating the class diagram to meet the new requirements identified as well as reflecting the changes made to reflect the code itself, in particular, classes HighScoreEntryScreen and LeaderBoardScreen were added. Accompanying these changes, alterations were made to the structure of the ActivityMapObject class include the dynamic accessing of the type, length of each activity and the effects on the stats overall, improving the structure and quality of the code. Another significant improvement involved modifying the way the different activities and their effects on player stats were stored as this was originally in the form of a hash mapping that was limited to three activities; no longer meeting the requirements.

One final noteworthy addition involved incorporating the InstructionScreen class in which sets the scene for the player, in combination with explaining the aim of the game and controls needed in order to play. This was added as not only a stylistic improvement to the game but also to maintain common gaming convention.

In general, as the time progressed, a combination of major and minor changes were made to the architecture of the system in order to adhere to the requirements, the product brief and overall make the system more robust, with the aim of keeping the gameplay simple and enjoyable for the player.

References:

[1] C. Piper, *eng1*. 2024. Accessed: May 06, 2024. [Figure 4]. Available: https://charliepiper.github.io/documents

[2] M. Richards and N. Ford, *Fundamentals of Software Architecture*. Ascent Audio, 2020.