Architecture

Group 15

HesHus

Adam Robinson

Tsveta Ivanova

April Jack

Oliver Goodwin

Luke Jackson

Luca Hammond

Chris Rolfe

Previous versions of all architectural diagrams can be found on the website, under 'Interim Architecture Versions'

Structural Diagrams

Class Diagrams

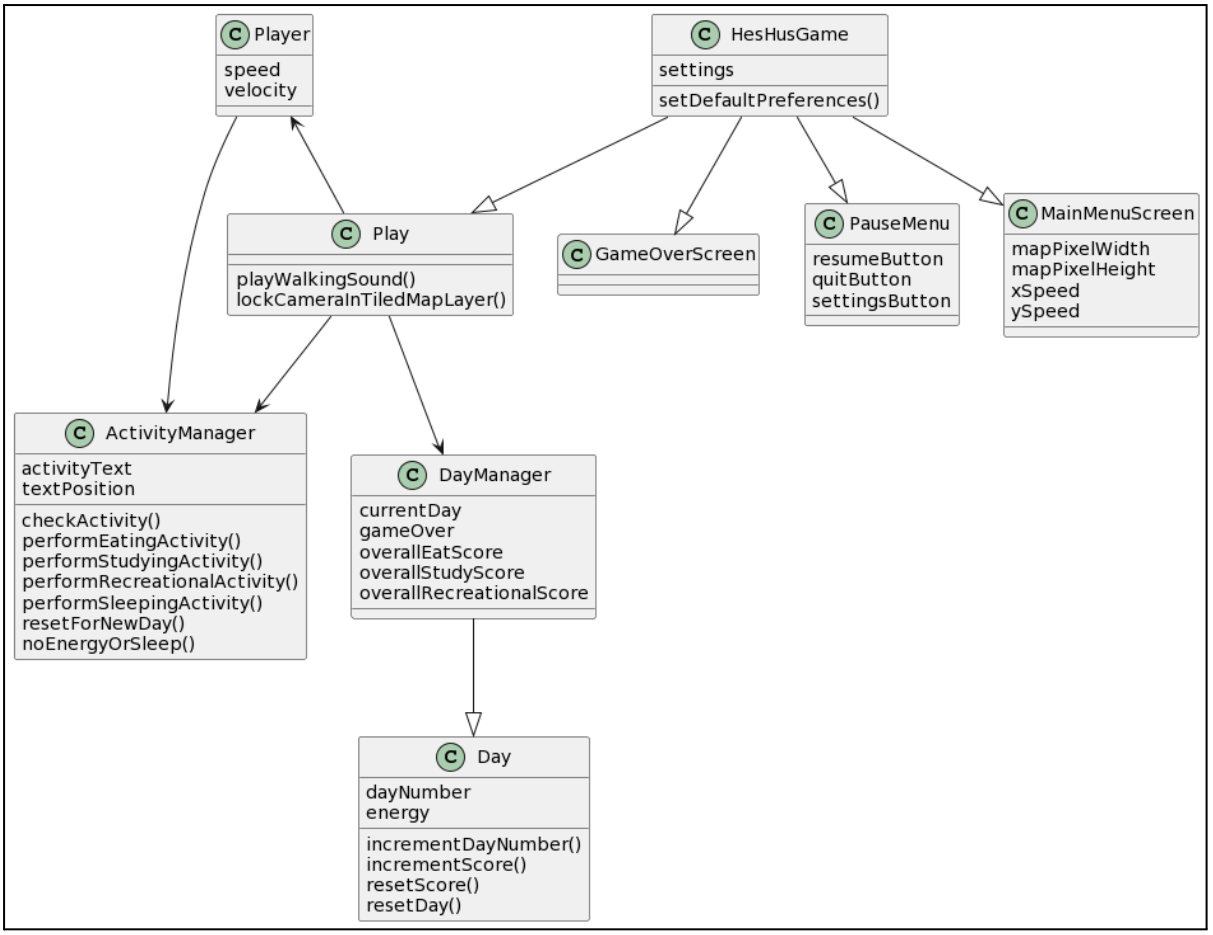
To create the class diagrams we used the PlantUML extension and constructed the diagrams using UML code.

The original class diagram was how a simple game would be structured with little thought about the requirements in order to get an understanding of how the skeleton of the system should look. It contained many classes that later became redundant, as the libGDX library contained prebuilt versions of the classes, for example the sprite class was unnecessary due to libGDX's Sprite class.

This led to the development of a new class diagram, primarily focused around game states, which were later replaced with screens due to simplicity. This class diagram also contained redundant classes such as the SoundManager, due to sound also being managed in libGDX.

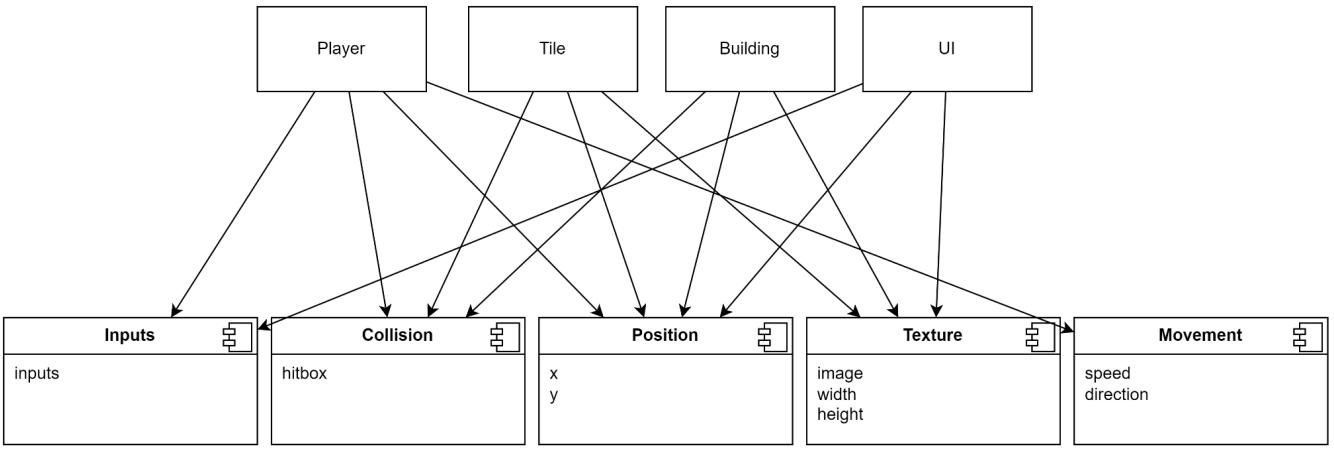
The final class diagram shown below contains all the classes needed for the system to work, without making the system too complex with small unnecessary classes. Since the game only needs very basic functionality, we limited the classes to controlling the flow of the game (all of which are functioning under the HesHusGame class) as well as decomposing the Play class into multiple related classes for easier management.

In relation to the requirements, the ActivityManager was created to control the different activities, the DayManager provides a place to track the different days and the Play/DayManager classes control the collection and displaying of the activity counters. FR_ENERGY_TIME_MANAGEMENT, FR_STATIC_TIME and FR_ACTIVITY_COUNTER are accounted for in the DayManager class, which implements a Day class allowing the game to progress through days, as well as play again if the player wants to. FR_FINISH, FR_PAUSE_MENU, FR_MAIN_MENU and UR_MAIN_MENU_NAVIGATION are all represented through different screens, controlled in classes extending the HesHusGame class. The ActivityManager uses the method checkActivity() in order to determine which activity is being interacted with, and therefore UR_CHOOSE_TASKS is complete.



Component Diagrams

This Entity-Component diagram was created with draw.io. Multiple entities have shared needs, and developing components would allow us to experiment more adding features to different things and reuse more code which would help with NFR_MAINTAINABLITY.



As the game is relatively simple and isn't performance intensive we did not end up using an entity component system, opting to use an OOP architecture instead.

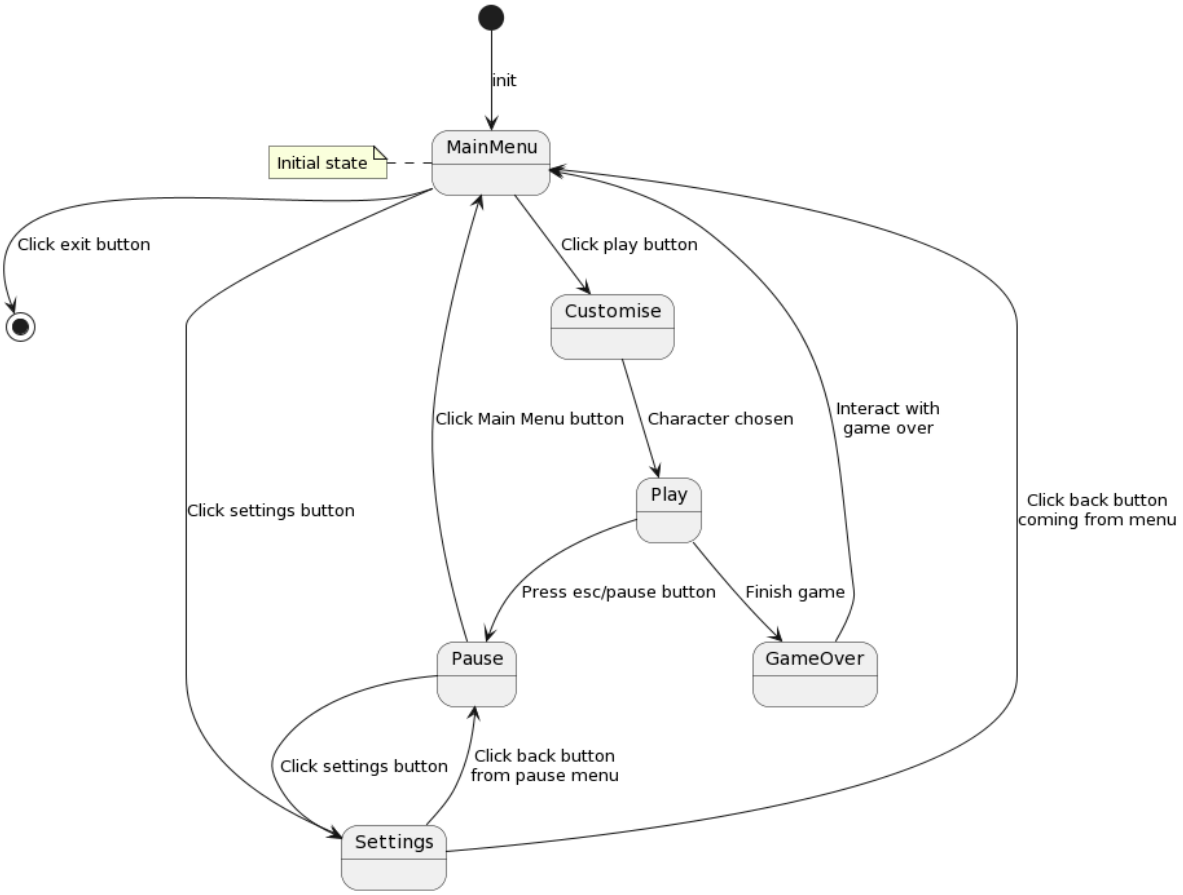
Behavioural Diagrams

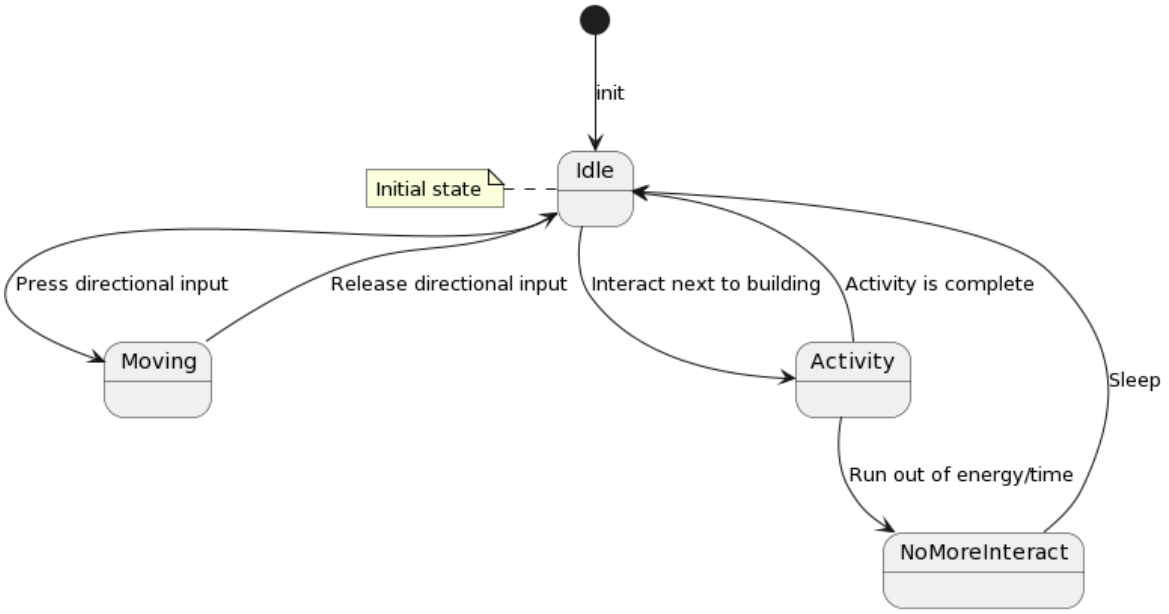
State Diagrams

The game being produced has 2 main flows to indicate how the game works:

- Flow of screens
- Flow of gameplay

These diagrams have been created using PlantUML and have mostly stayed the same throughout the different stages of development.
The only changes to the state diagrams are an additional GameOver branching off of play to enable the game to be replayed, in line with requirement UR_COMPLETE_GAME_SCREEN, as well as an intermediate state Customise to accommodate for UR_CUSTOMISE_AVATAR. The pause state allows the player to change their settings, even while the game is running (UR_PAUSE_MENU_VISIBILITY). The second state diagram describes what will happen during the play state of the original state diagram, which includes the description of a moving player (UR_CONTROL). The inclusion of the state NoMoreInteract falls in line with the requirement UR_SLEEP_FEATURE_TASK. We decided to use multiple state diagrams because understanding the flow of the entire system was key to implementing the required classes; knowing what needed to happen at each stage of the game simplified this process.


init
MainMenu
Initial state
Click exit button
Click play button
Customise
Click Main Menu button
Character chosen
Interact with
game over
Click settings button
Play
Click back button
coming from menu
Press esc/pause button
Finish game
Pause
GameOver
Click settings button
Click back button
from pause menu
Settings


init
Idle
Initial state
Press directional input
Release directional input
Interact next to building
Activity is complete
Moving
Activity
Sleep
Run out of energy/time
NoMoreInteract

Sequence Diagrams

The sequence diagram for the main interaction logic is shown below. This is the most significant sequence of interactions between different classes. Our first iteration of this was based largely on the initial class diagrams, and requirements like UR_ENERGY_TIME_FEATURE and UR_CONTROL, which governed how this logic should work. UR_CHOOSE_TASKS governs a lot of the alternative cases, where the player runs out of energy or time left in the day.

As development continued and this logic was implemented, the structure of the interactions changed: some classes like ActivityManager were responsible for modifying their own internal variables, rather than passing this off as a method call to another Manager class like we initially planned. However, the core functionality and sequence of events in the diagram below stayed consistent throughout development.