Architecture

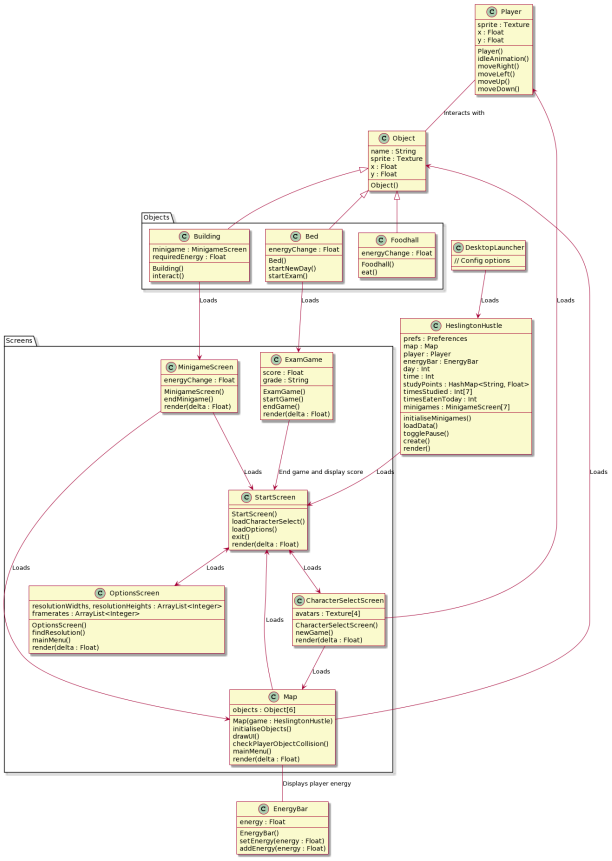
Cohort 1 Group 8

Roy Asiku, Tianqi Feng, Andrew Jenkins, Nicholas Lambert, Tom Byron

To represent the architecture of our game, we decided to use UML via the PlantUML editor to make structural and behavioural diagrams.We chose to do this as unlike image based editors, UML allows for easy adaptability and consistency between diagram formatting with specific tools to create class and state diagrams.We discovered as a group that UML has a very low overhead for beginners, being quite easy to pick up making it the best suited diagram creating tool for this project

Our structural diagram represents the different classes and packages within our game and how they interact with each other in a more technical sense. Our behavioural diagrams, on the other hand, represent the core game loop for the user, including both their interactions with the game's menus and the actual gameplay itself. We made sure to refer to our requirements when creating these diagrams to ensure that our game's architecture matched what the client desired. In this document when a design decision is made, its relevant requirement ID (see requirements document for reference) is given next to it.

Structural Diagram
https://nicholaslambert03.github.io/ENG1Website.io/class_diagram_5.png



Our structural diagram begins with the DesktopLauncher class, which, in the game engine libGDX, is responsible for starting up the application. It creates a new instance of the HeslingtonHustle class, which upon creation calls the create() method, loading up the game's main menu. The HeslingtonHustle class also stores vital game information such as screen size, the player object, energy and our minigames.

The game's main scenes work via screens, which is one of our main packages. Screens have the ability to call the render() method once per frame, which is responsible for drawing objects on the display. Our menus will have separate screens for easier editing, with the user being able to quickly navigate between all of our start menus. Our other main screens include the map and minigame screens, with each minigame having its own screen.

The map screen is the main view of the game, which allows the player to move between locations (FR_MOVMENT) and interact (FR_INTERACT) with objects to start up minigames or progress the game in other ways. The map also contains an array of all of the game's interactable objects, which it is responsible for drawing and checking, via its render() method, whether a player interacts with them.

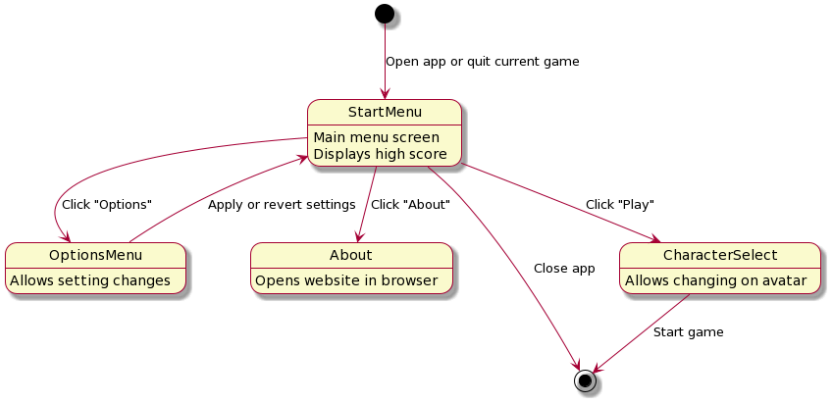
Our core gameplay is via minigames (FR_MINIGAMES), which can be accessed via interacting with Building objects (FR_STUDY and FR_RECREATION). Each minigame is unique and has different outcomes depending on the user's performance. The minigame's difficulty can also be scaled via the DifficultyScalar variable (NFR_DIFFICULTY), which will be provided when loading up the minigame.

Other important classes include the Player class and EnergyBar class (FR_ENERGY), each of which is instantiated in the Map class and stored, per run, in the HeslingtonHustle class. The Player class stores the player's position and avatar, while the EnergyBar class stores the player's energy.

Our object class has 3 separate child classes: the Bed (FR_REST), Building (FR_STUDY and FR_RECREATION) and Foodhall (FR_EAT_PLACE) classes. The Building class represents objects that have a minigame associated with them, which would be for either study or recreational activities. Each building has a required energy value (FR_RESOURCES) which is required to interact with it, in the case of recreation this value is set to 0, whereas for studying the player requires a certain amount of energy to continue. The Foodhall class is used for eating and has an energyChange value, which is the amount of energy the player will gain by eating there. The Bed class is interacted with to end the current day and progress the game forward (FR_TIME_CYCLE).

Behavioural Diagram - Menus

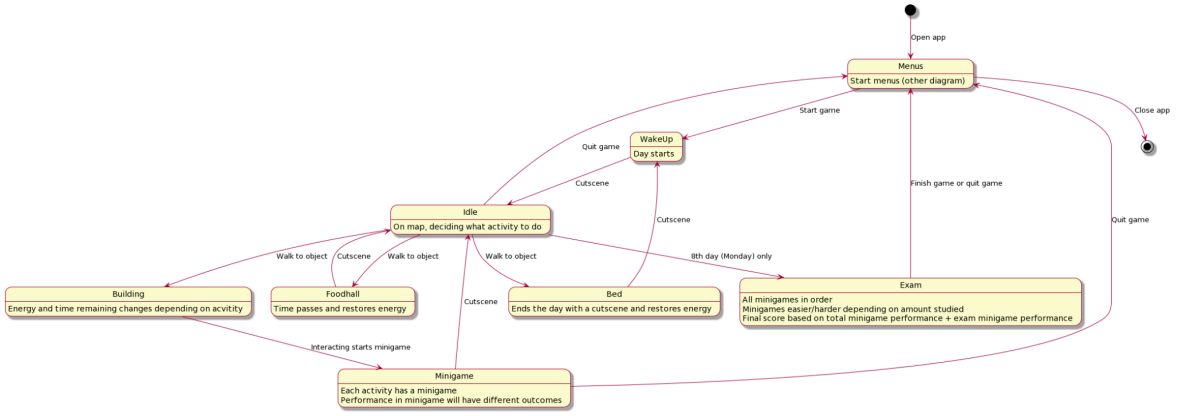
https://nicholaslambert03.github.io/ENG1Website.io/state_diagram_menus_1.png



Upon starting up the application, the user is immediately shown the start menu, which allows them to select either “Play”, “Options”, “About” or “Exit”. The “Play” button takes them to character selection (FR_CUSTOMISATION_MENU and FR_SKINS), which allows them to select an avatar from a small array of preset sprites. The “Options” button will take them to the options menu, in which they can change key settings such as volume , resolution (FR_SCREEN_SIZE) and framerate. The character selection and options menu will also allow the user to return to the start menu. The “About” button will open up our game’s website in the user’s browser, and the “Exit” button will close the application.

Behavioural Diagram - Game

https://nicholaslambert03.github.io/ENG1Website.io/state_diagram_game_1.png



After starting the game from the character selection menu, the player wakes up to start a new day (FR_TIME_CYCLE) and is taken to being idle on our map. From this idle state, the player can choose to interact (FR_INTERACT) with either of our objects by walking to them (FR_MOVEMENT) and hitting the interact button. If the player chooses to study (FR_STUDY) or perform a recreational activity (FR_RECREATION), they are taken to that activity’s minigame (FR_MINIGAMES). Upon completion of that minigame, they go back to the idle state, from which they can continue choosing activities or end the day (FR_REST).

Ending the day via interacting with the Bed will do a quick loop to the wake up state (FR_TIME_CYCLE), which will play a short cutscene and then take them back to the idle state, with the game’s time now progressed by one day. After completion of the 7th day, the player is taken to the exam state, in which they must complete all of the study minigames in a row to help determine their final score (FR_GAME_END). After completing the exam, the game ends and returns the player to the main menu after showing them their final score (FR_SCORE). Additionally, the player can choose to quit and return to the main menu at any time while in the idle, minigame or exam states.

Minigame Diagrams

All of our minigames (FR_MINIGAMES) ramp up very quickly in difficulty to ensure that the game does not last too long (NFR_GAME_TIME). All of our study minigames get harder the more the player studies (FR_OVERSTUDY) in a given day. Doing well in a study minigame rewards the player with study points, which both contribute to their overall score and make that minigame in the final exam easier, which also promotes variety (FR_STUDY_VARIETY).

Our recreational minigames reward the player with more energy (FR_RECREATION) if they perform well, and reward less energy if the game has already been played that day to promote variety (FR_RECREATION_PLACE).

BugFixer Minigame
https://nicholaslambert03.github.io/ENG1Website.io/bugfixer_diagram.png
This minigame, made by Andrew, is a study minigame in a “bullet hell” style, where the player (a mouse cursor) moves around the screen and shoots bugs to accumulate points, while also attempting to dodge the bugs’ own attacks. The more points you get and the longer you survive determine how many study points you gain. The BugFixer class is responsible for spawning SniperBugs and ScatterBugs, which, along with the player, spawn Bullets. This minigame makes use of libGDX’s Box2D library for handling physics.

BookStacker Minigame
https://nicholaslambert03.github.io/ENG1Website.io/bookstacker_diagram.png
This minigame, made by Nicholas, is a study minigame, where the player must click at correct intervals to drop books onto each other. If part of a book does not align with the book below, it is cut off and the game becomes harder. The taller the player’s stack of books, the more study points they gain. The BookStacker class handles most game logic and spawns BookSegments.

ColourMatch Minigame
https://nicholaslambert03.github.io/ENG1Website.io/colourmatch_diagram.png
This minigame, made by Nicholas, is a study minigame, where the player must memorise a sequence of colours that is displayed on the screen and input them in the same order, with the sequence increasing in length as the game progresses. The player gets points when they correctly input a sequence. The ColourMatch class handles most game logic and spawns ColourBlocks

DrunkDancer Minigame
https://nicholaslambert03.github.io/ENG1Website.io/drunkdancer_diagram.png
This minigame, made by Andrew, is a recreation minigame, like “Guitar Hero”, where the player must click directional keys when they are located in certain spots on the screen. The DrunkDancer class handles game logic and generates instances of the QuickTimeEvent class over time. The player gains points by pressing keys at correct times.

Squash Minigame
https://nicholaslambert03.github.io/ENG1Website.io/squash_diagram.png
This minigame, made by Roy, is a recreation minigame, where the player controls a paddle, which they should use to rebound a moving ball into a wall. Hitting the ball gives them points, and missing the ball will end the game. This minigame also uses the Box2D library to handle physics.

SwiftSwimmer Minigame
https://nicholaslambert03.github.io/ENG1Website.io/swiftswimmer_diagram.png
This minigame, made by Nicholas, is a recreation minigame, where the player must click as many times as possible in a given period of time to move their player across a lake. When the player reaches the edge of the lake, they turn around, and their score increases. When

time runs out, the game ends. The SwiftSwimmer class handles rendering and loading the game, and the Swimmer class handles most game logic.

Diagram Evolution

All versions of our architecture diagrams can be found on our website, designed by Nicholas: https://nicholaslambert03.github.io/ENG1Website.io/architecture.html

Version 1 (28/02/24):
https://nicholaslambert03.github.io/ENG1Website.io/class_diagram_1.png
Our first structural diagram was by no means definitive or complete, it was both a learning experience with UML as we were unfamiliar with it, and a basic layout of the game for which at the time we didn't have a solid plan for. We immediately saw the potential of minigames (FR_MINIGAMES) as a core gameplay loop and decided that we wanted to implement them somehow, with some basic ideas for minigames inheriting from the Minigame class. We also knew that we wanted each minigame to scale its difficulty depending on different factors, so we added difficultyScalar early on as a variable for the Minigame class (NFR_DIFFICULTY).

We added some basic classes that we knew were definitely going to be implemented, such as the player (FR_PLAYER_COUNT), objects (FR_STUDY and FR_RECREATION) and menus, with ObjectType being an example of a child of the Object class.

Version 2 (29/02/24)
https://nicholaslambert03.github.io/ENG1Website.io/class_diagram_2.png
Our second version of the structural diagram was a lot more fleshed out. We updated the menu into a StartScreen, as we knew there were going to be further menu types that we would need to add. From the StartScreen, we figured out that we would want the user to be sent to a map scene, on which they could move around (FR_MOVEMENT) and interact (FR_INTERACT) with objects placed around the map. Bed (FR_REST) and Building (FR_STUDY and FR_RECREATION) being our 2 current object types.

We knew that we wanted Player and EnergyBar (FR_ENERGY) classes as well in order to store important information to be used throughout the game. We also set up some minigame child (FR_MINIGAMES) classes to be filled out when we come to implementing them.

Version 3 (03/03/24)
https://nicholaslambert03.github.io/ENG1Website.io/class_diagram_3.png
The third version of the structural diagram was made after deciding on libGDX as our game engine and looking into how it functioned. After a few days of research, we cleared up how libGDX would render scenes and transfer between them.

To start, we added the interaction between the DesktopLauncher and HeslingtonHustle classes as the start of our game, as that is how the application is loaded. Since the HeslingtonHustle class was used as a parameter for most of the screen classes, we decided to also store important information such as the player, energy and day in it. We also decided to declare and use one spritebatch and bitmapfont across the whole project to save on resources, which is also stored in this class.

We fleshed out the starting menus a little bit, adding an options menu (FR_SCREEN_SIZE) and character select menu (FR_CUSTOMISATION and FR_SKINS). We also changed the minigames to be screens rather than objects so that they would be able to use the libGDX Screen interface's methods such as render() to be able to display things on the screen. In addition, we added constructors to classes where it was relevant to make it clear how they would be created.

We also added some methods and variables to the Map to enable it to create and draw objects for the player to interact with (FR_INTERACT). Next, the object child classes were added into their own package for clarity, and we added the Foodhall class (FR_EAT_PLACE), which the player can interact with to eat and restore energy (FR_EATING). Furthermore, we made some changes to each class's variables, trying to think ahead to what would be useful, for example the tooltip variable in the Object class to display to the user what each object does and how to interact with it (NFR_SIMPLICITY).

Version 4 (11/03/24)
https://nicholaslambert03.github.io/ENG1Website.io/class_diagram_4.png
We made the fourth version of our structural diagram after finishing our basic implementation. Apart from adding various variables and methods to make sure everything worked correctly, notable changes include the additions of the PopUpText and InformationScreen classes. Both were added to better communicate information to the player (NFR_SIMPLICITY). PopUpText is used to tell the player something quickly, such as when they have insufficient energy to study. InformationScreen is used when displaying a tutorial or a final score, and requires an input to continue to ensure that the player takes in the information. Other notable additions include sound effects (FR_SOUND), music and some art and animations for the game (FR_SKINS). The sound effects, music and tile maps were acquired from asset stores with appropriate copyright permissions, and other art was made by Andrew in a pixel art application called Aseprite.

Another important addition is "study points" (FR_STUDY), which are stored in a hash map in the HeslingtonHustle class. These are acquired via studying and doing well in study minigames, and will contribute to the player's final score as well as determine the difficulty of each minigame in the final exam.

Version 5 (19/03/24)
https://nicholaslambert03.github.io/ENG1Website.io/class_diagram_5.png
This is the final version of our structural diagram. It doesn't differ massively to version 4 as the main game was already mostly done, it was mainly the minigames that needed work. However some notable changes include the addition of inaccessible region arrays in the Player class, which came with the map redesign and prevent the player from straying from intended paths (FR_MOVEMENT). The main HeslingtonHustle class was also updated with a hashmap to allow us to track recreational activities and reward the player for interacting with different activities rather than the same one (FR_RECREATION_PLACE). Some variables used for outputting the console log for playtesting and debugging were also added.