Method Selection and Planning

Group 27 - WaddleWare Studios

James Smith

Amanda Ling

Fran Medland

Hannah Vas

James Kellett

Malik Tremain

Mischa Zaynchkovsky

(Design Process)

From the project brief and from our requirements gathering, we found key themes regarding:
1. How the player interacts with the avatar and the buildings.
2. How the player's actions affect the time and scoring of the game.
3. How the game is displayed to the player and what the player can see.
4. What the player experiences whilst playing the game.

Theme	User Requirements
1	UR_Interract/ UR_Movement
2	UR_Activity/ UR_Time/ UR_Resting (time) UR_Activity/ UR_Skill/ UR_Scoring/ UR_Win (scoring)
3	UR_Map/ UR_Appearance/ UR_Easy_Play
4	UR_User/ UR_Access/ UR_Enjoy

From these themes we discussed the potential objects needed to fulfil the functional and non-functional requirements of the game, these requirements stem from the main user requirements above, and from those objects, we modelled our CRC cards. To create the CRC cards we used the tool Miro, an online sticky notes board. We chose to use this tool because it allowed us to visualise and collaborate together more easily as opposed to physical cards. We iterated our CRC card model twice as the first version we found did not satisfy most of our requirements when we tried implementing. For our final CRC cards version, we have 6 groups: main, buildings (within main), input, screens, clock and scoring. The roles and responsibilities of each object were influenced by the majority of the requirements we had established earlier and are shown in the table below.

| Groups and their objects:	Functional/ Non-Functional Requirements:
Main ● Avatar ● Buildings ● Map	FR_Sprite (UR_Movement) FR_Map (UR_Map)
Buildings ● Study building ● Food building ● Recreation building ● Sleep building	FR_Map_Locations (UR_Map) FR_Buildings/ FR_Building_Activities (UR_Activity)
Input ● Keyboard (wasd) ● Keyboard (enter/space) ● Keyboard (esc)	FR_Movement (UR_Movement) FR_Interact (UR_Interract) FR_Interaction (UR_Access)
Screens ● Display game ● Interaction pop-up ● Outcome ● Menu ● Tutorial	FR_Whole_Map/ FR_Time_Display (UR_Map) FR_Energy_bar/ FR_In_Game_Time/ FR_Scalability (UR_Appearance) NF_Display/ NF_Map (UR_Map) FR_Icon (UR_Interract)

● Settings	FR_End_Message (UR_Win) FR_Score_Display (UR_Win) FR_Menu_Screen (UR_Easy_Play) FR_Tutorial (UR_Easy_Play) FR_Music/ NF_Music/ NF_SFX (UR_Enjoy) NF_Colour (UR_Access)
Clock ● Time ● Day	FR_Smooth_Time/ FR_Time_Drain/ FR_Time_Cut (UR_Time) FR_Time_Sleep/ FR_Sleep (UR_Resting) FR_Study (UR_Activity) NF_Game_Time (UR_Time)
Scoring ● Energy bar ● Score	FR_Energy_Amount/ FR_Energy_Bar (UR_Activity) FR_Score_Calc/ FR_Score_Dec/ FR_Study_Score/ FR_Recreation_Score/ FR_Overstudy/ FR_Forget_to_Eat/ FR_No_Recreation (UR_Scoring) NF_Difficulty/ NF_Lose (UR_Skill) NF_Win (UR_Win)

(Structural)

The main tool we have used is the UML tool PlantUML, after considering the following tools: PlantUML, Microsoft Visio and LucidChart. The main advantage of using PlantUML was the tool support, such as those of google docs gizmo and its great documentation, and plain text encoding and definition system which made it the most simplistic and robust to use.

From the first CRC cards version, the groups: main, time and screen and input were our initial class plans and the group: scoring was a system within the game. And from that model, we created our first UML class diagram to establish an initial OOP structure along with our first entity component system for scoring:
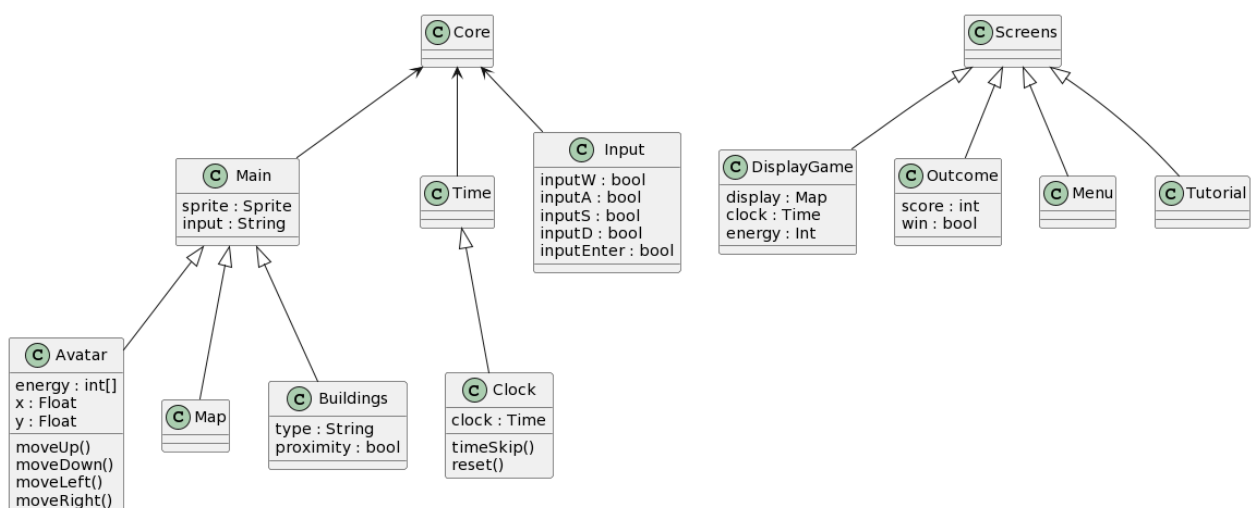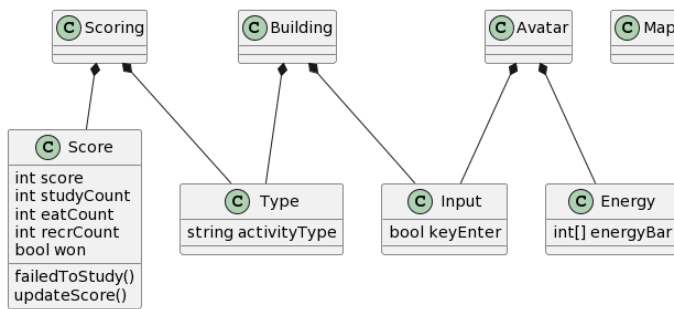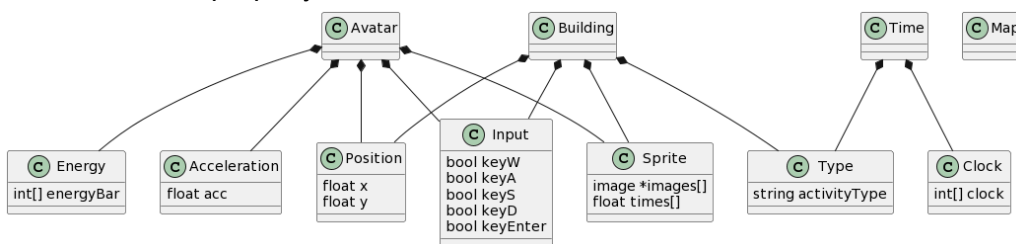
The way the scoring system works is that the component Score contains attributes which count the individual no. activities the entity Avatar has done (interaction) via the component Type) under the entity Building. Then, it will update the score based on the counts and if the avatar fails to study then the player cannot win (scoring), relating to theme 1 and theme 2 respectively. Additionally, the avatar can only do tasks based on energy level.

The second CRC cards model helped to establish a more detailed class diagram along with the input group now being a system alongside scoring:
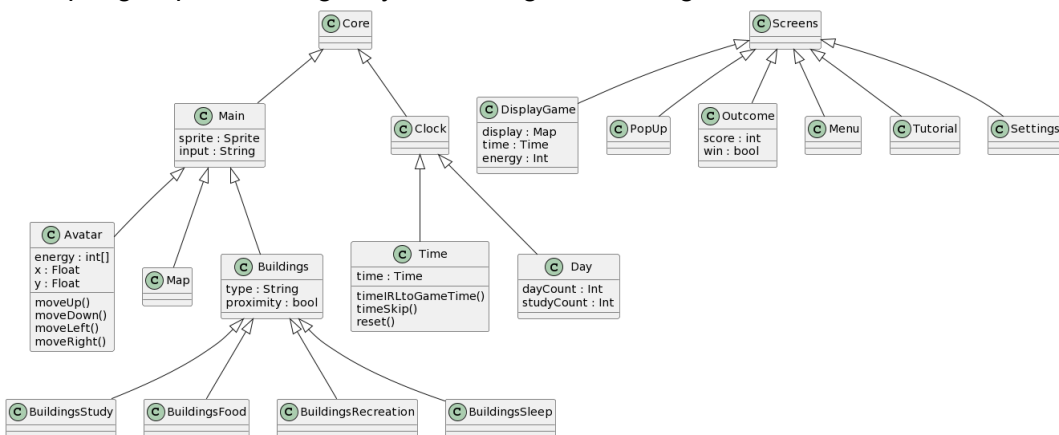

Through the addition of objects PopUp and Setting under the Screens class, the themes 3 and 4 were now properly met.

The way the input system works is that the component Input will affect the related components of: the entity Avatar (movements) and the entity Time from the entity Building (time skip), relating to theme 1 and theme 2 respectively.

Upon further implementation, we developed our class diagram again to be more thorough, with changes to class names, addition of classes and changes in some class relationships. to be more relevant to the game structure.
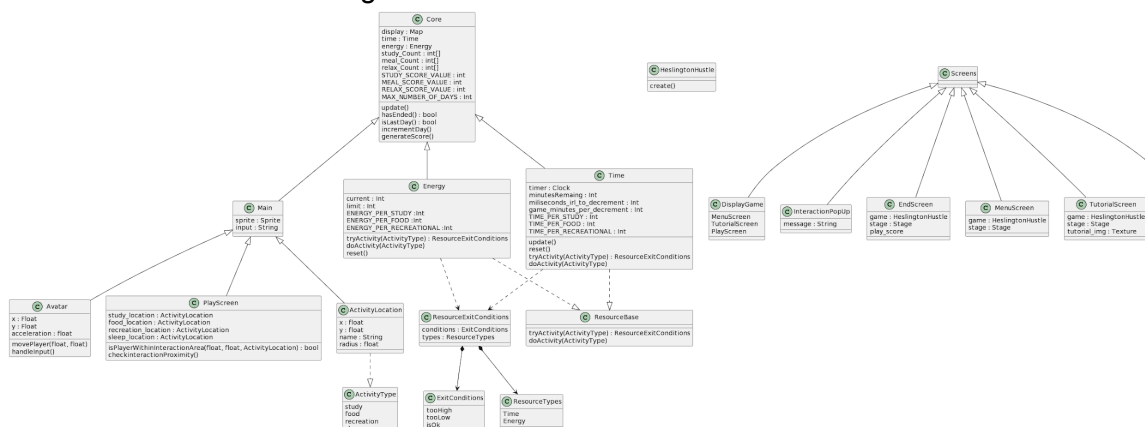
The final class diagram is taken from the built-in tool within the IntelliJ IDE. The generative tool allowed us to simply display the final class structure of the game more easily.
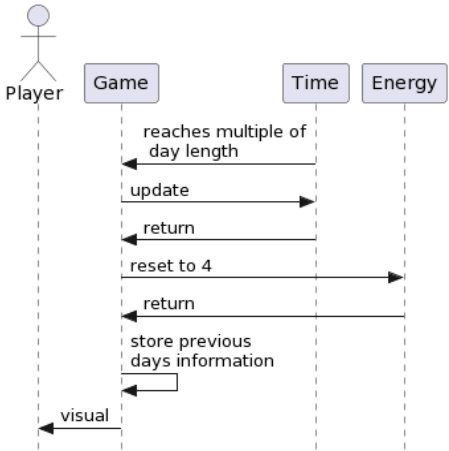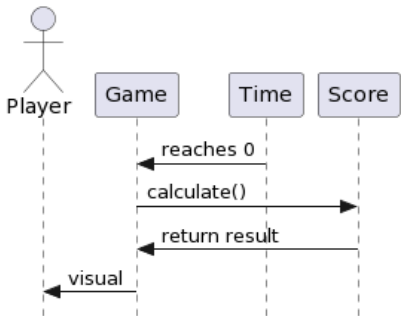
(Behavioural)

We created a set of initial sequence diagrams to represent a subset of the major interactions in the game. These were there to give us a feel for the interactions that will end up performed to progress towards the final goals of the game.

Entities acting in the following sequence diagrams, and their justification as entities:
- Energy: a core mechanic of the game given to us by the specification
- Time: a core mechanic of the game given to us by the specification
- Player: the player represents the user interacting with the game system
- Building: the building is an entity representing an abstract version of any building the the game such as those for sleeping, eating, recreational activities and studying. Each of those buildings is included as part of the specification of the game
- Game: Game in this context is an entity for the game itself where it will perform some functions and be able to display visuals to the user
- Score: Score is an entity for the scoring subsystem of the game to be used when the 7 days have passed

Scenarios Table:

Scenarios	Code
1. Player opens game 	skinparam style strictuml Actor Player Player -> Game: Start Game -> Energy:init Energy->Game:return 4 blocks/day Game -> Time:init Time -> Game:return game length Game->Player:visual
2. Interact with building 	skinparam style strictuml Actor Player Player -> Game: walk Game -> Building: proximity Building -> Game: interactable Game -> Player: visual Player ->Game: interact Game ->Building: interact Building ->Energy: --building energy Energy ->Building: return new value Building ->Time: --building time Time ->Building: return new value Building -> Game: return new values Game -> Player: visual

3. Day ends 	skinparam style strictuml Actor Player participant Game Time-> Game: reaches multiple of\n day length Game ->Time:update Time -> Game: return Game->Energy:reset to 4 Energy-> Game:return Game->Game:store previous\ndays information Game->Player: visual
4. Game ends 	skinparam style strictuml Actor Player Game <- Time: reaches 0 Game -> Score: calculate() Score -> Game: return result Game->Player:visual

These first diagrams were designed with the intentions of covering some of the most important functional and nonfunctional requirements, the following table links them to the requirements they cover:

Diagram	Requirements
1 Player Opens Game	FR_Interaction, FR_Energy_Bar, FR_In_Game_Time, FR_Menu_Screen
2 Interact With Building	FR_Interaction, FR_Energy_Bar, FR_Interact, FR_In_Game_Time, FR_Map, FR_Map_Locations, FR_Score_Calc, FR_Energy_Bar, FR_Time_Display, FR_Movement, FR_Icon, FR_Sprite, FR_Time_Cut
3 Day Ends	FR_Energy_Bar, FR_In_game_Time, FR_Scoring, FR_Score_Calc, FR_Energy_Bar, NF_Time_Display
4 Game Ends	FR_Scoring, FR_End_Message, FR_Sleep, FR_Study, FR_Score_Calc, FR_Score_Dec, FR_Score_Display, NF_Game_Time, NF_Win, NF_Lose, NF_Difficulty

Further to this we made the following new diagrams or changes to existing ones. These changes were justified by discussion among the group about the intended interactions of the game system, such as in the case of diagram 3 where we decided that it would be better for the Player actor to initiate the end of the day via the sleep system, and otherwise for coverage of the user/functional/non functional requirements.

Scenario	Code

1 Player Opens Game	skinparam style strictuml
	Actor Player
	participant Game
	alt start
	   Player -> Game: Start
	   Game -> Energy:init
	   Energy->Game:return 4 blocks/day
	   Game -> Time:init
	   Time -> Game:return game length
	else tutorial
	   Player->Game: Tutorial
	   Game->Game: show tutorial screen
	else exit
	   Player->Game: Exit
	   Game->Game: quit
	end
	Game->Player:visual

3 Day Ends	skinparam style strictuml Actor Player participant Game Player->Game: Goes to sleep Game ->Time:update Time -> Game: return Game->Energy:reset to 4 Energy-> Game:return Game->Game:store previous\ndays information Game->Player: visual

Sequence	Changes to Requirements
1 Player Opens Game	FR_Controls, FR_Tutorial, NF_Availibility
3 Day Ends	FR_Interaction, FR_Controls, FR_Study, FR_Buildings, FR_Time_Sleep, FR_Sleep,

These sequence diagrams are high level overviews of sequential interactions between entities acting in the game ecosystem, intended to give an insight into the intended behaviour of the system. They have been created by interactive team discussion and have been designed in a way that they are justified by the associated requirements.

References:
Enhanced/ Enlarged versions of all diagrams are available in their original formats and sizing on the GitHub website in the appropriate order.