ENG1 Architecture

Cohort 3 - Group 28

“Team 28”

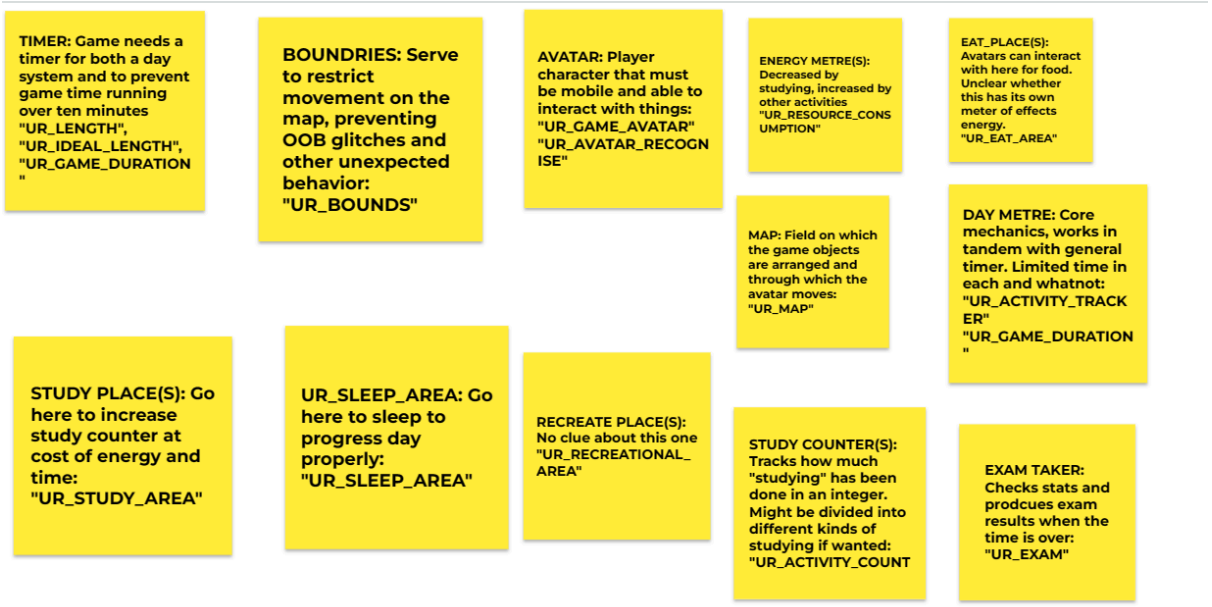
Muhammed Salahudheen
Joel McBride
Jamie Rogers
Maciek Zaweracz
Rhys Yeaxlee
Alex Spencer
Alex Firth

3. Architecture [22 marks]:

   Give diagrammatic representations (structural and behavioural diagrams) of the architecture of the team’s product, with a brief statement of the specific languages (for instance, relevant parts of UML) and the tool(s) used to create these representations. Include a systematic justification for this architecture and describe how it was initially designed and how it evolved over the course of the project. Provide evidence of the design process followed (e.g. interim versions of architectural diagrams, CRC cards) on your team’s website and link to them from your report. Relate the architecture clearly to the requirements, using your requirements referencing for identification, and consistent naming of constructs to provide traceability (22 marks, $\leq 6\ pages$).

We largely followed the methodology discussed in Fundamentals of Software Architecture : An Engineering Approach Part 1 Chapter 8. A Google Jamboard was used to arrange the earlier concepts, while a UML diagram on Draw.io was used for the later class diagrams.

CRC CARDS

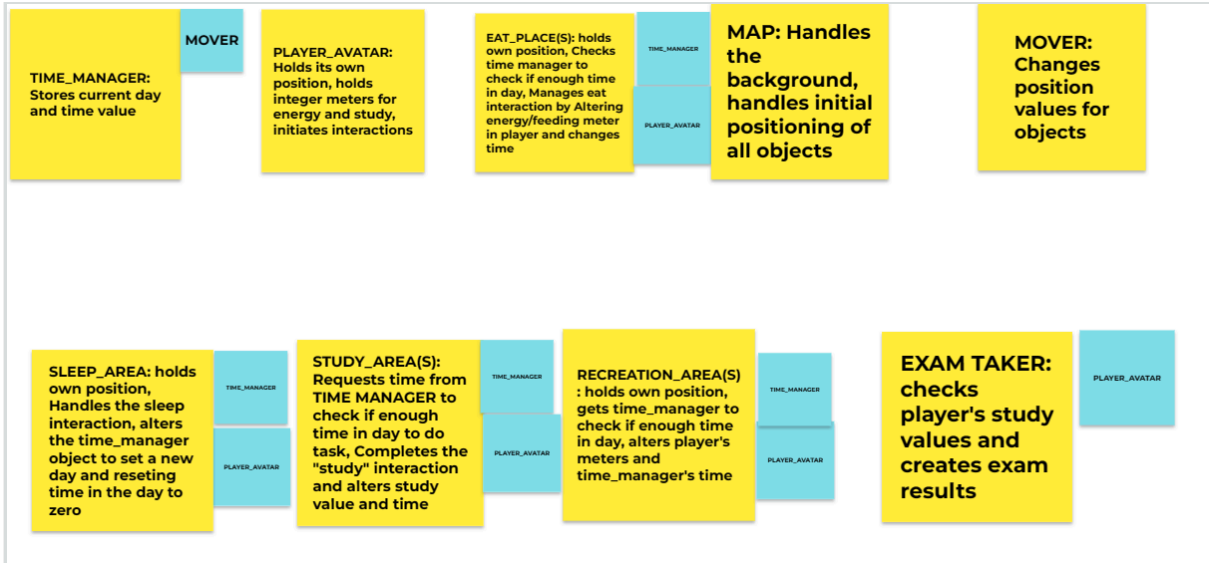


Candidates were created to closely follow the requirements, describing the intended function and listing the User Requirement within. Other team members clarified about how certain mechanics would work within the libgdx (i.e. how bounds should be handled and how movement should be thought about), resulting in the next stage of cards where there was no boundary candidate and a separate movement object was used to move objects like the avatar.

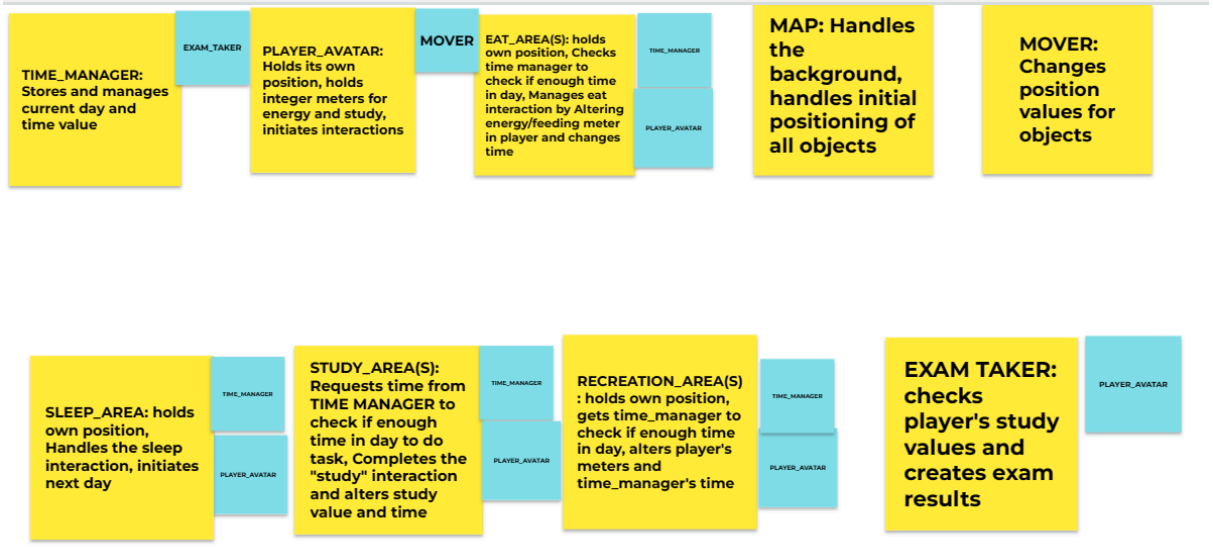
Stereotypes:

At this stage the stereotypes were relatively simple to assign.
Feedback from teammates indicates that the study counters and energy meter should be part of the avatar object as they all relate directly to the player and there's no reason to have to fetch that between separate objects. Timer and day meter are also merged as they effectively operate as two parts of the same mechanic.

These simplified cards will be used as the basis for the next step of defining responsibilities. Names have been changed to reflect new roles or to better identify them.

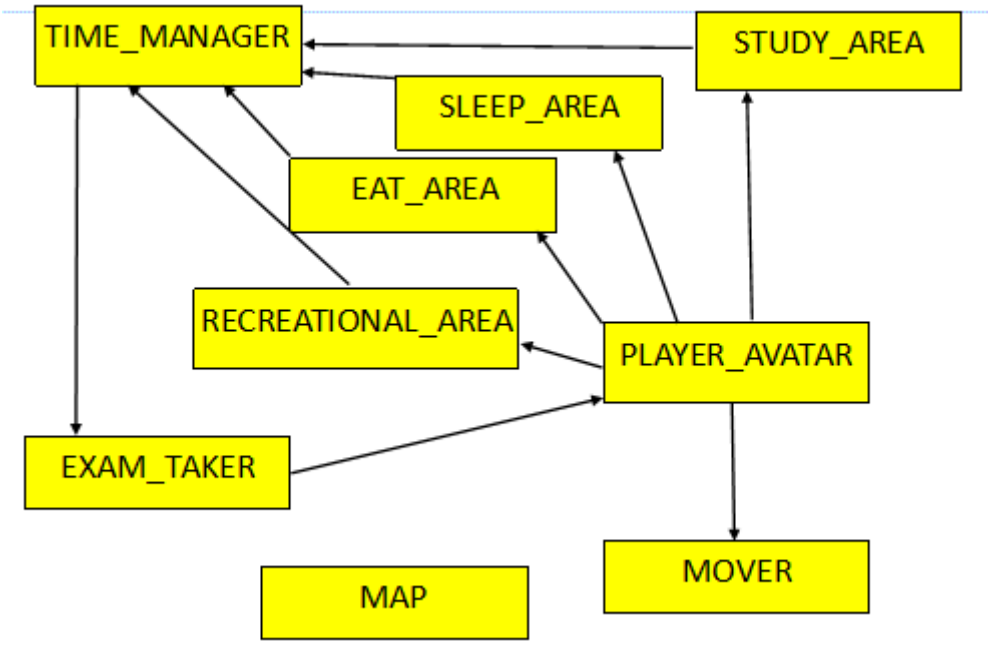


Adding responsibilities was simple as the connections in logic were not complex.



Upon further discussion with the team this was simplified to keep options open. The sleep area only initiates the day transfer, time_manager performs the specifics, meaning that it could be possible for other objects to transition days if needed.

Control Style



This basic logic map shows the logic flow at this stage. A dispersed control style is used as the game is too basic for a central smart object to be of much use.

This was the initial prototype for the game architecture, this quickly began to evolve as we did more research into the libraries we were using, in this case libGDX. Now although libGDX does support entity-system architecture we chose not to make use of this as the brief specifies a game that is quite small in scale and it is likely that most systems will not be used repeatedly. As such we chose to remain using a class system. We first used a class table so we could start to explore our class structure, linking the requirements to the classes, and marking their potential implementation. We then moved on to creating a UML class diagram, since many of the classes interact without inheritance this initially did not contain a whole lot of connections. At this point we also decided to have a central class to manage any aspects of the game that were not in their own separate class. This was called game manager, and also managed the time and the interactable objects.

At this point we began to make a simple UML diagram, we had not yet decided on our full implementation and instead chose to begin creating class representations and inheritance diagrams. All of the UML diagrams can be seen at https://team28.tech/2024/03/20/architecture-process.html the initial UML diagram is small and only contains one parent class, at this point we were just figuring out what processes and classes we might need to develop the game. We realised that we needed to split up the game manager into a screen portion and a game manager, this ensured that we did not have a monolithic class, as we wanted to compartmentalise our code. At this point we had done the most work looking at interactable areas, as we identified that this would probably be the segment to cause us the most trouble.

The second UML diagram in our series was when it started to develop slightly more. The inheritance got more complicated at this point as we added in the entity class to act as a superclass between entities we were adding to the game, such as the interactable areas and the player itself. Additionally we created the main class for the game itself, HeslingtonHustle, this is the activation class, from which the rest of the script is run. Furthermore, a separate class was made for movement, separate to the player class, we also added a separate class to hold the counters for the number of interactions with the different areas.

The third UML diagram is much more fleshed out, containing information about logical links between classes, these

CLASS TABLE

Class	**Requirements**	**Implementation**
GameManager	UR_IDEAL_LENGTH	Once timer reaches 10 min, do X
	UR_LENGTH	Once timer reaches 15 min, do X (force quit game?)
	UR_GAME_DURATION	
UIManager	UR_ACTIVITY_TRACKER	
	UR_ACTIVITY_COUNTER	
	UR_RESOURCE_CONSUMPTION	Visual Representation of resource consumption
-	UR_MAP	
	UR_BOUNDS	
Player	UR_GAME_AVATAR	
	UR_AVATAR_RECOGNISE	
	UR_RESOURCE_CONSUMPTION	Internal resource counters
MoveComponent	Part of UR_GAME_AVATAR	Allows an object to move
Interactable	-	
StudyArea implements Interactable	UR_STUDY_AREA	
SleepArea implements Interactable	UR_SLEEP_AREA	
RecreationalArea implements Interactable	UR_RECREATIONAL_AREA	
EatArea implements Interactable	UR_EAT_AREA	
Exam Manager	UR_Exam	