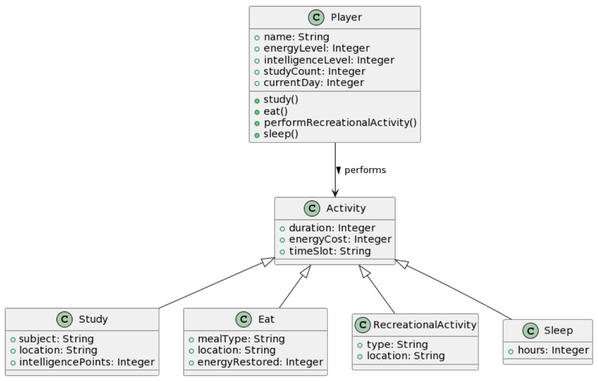
Architecture

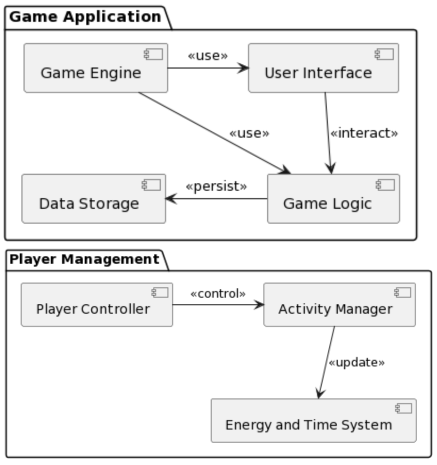
Group Number: 4
Group Name: THEEMD
Members: Mikaella Loppnow, Tom Daly, Harriet Kirby, Ethan Buss, Dillon Pandya, Ereife Odusi

To detail the architecture of our 2D game developed in Java, we've constructed both structural and behavioural diagrams. These diagrams aim to provide clarity on the game's design, component interaction, and operational dynamics, offering a comprehensive architectural overview.

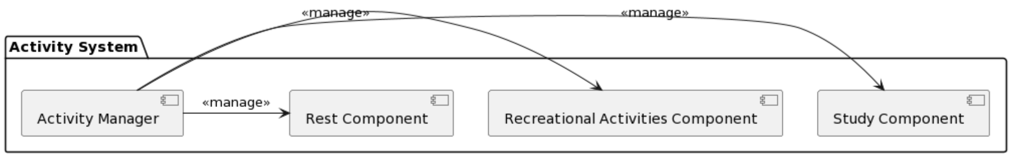
Structural Diagrams:

Class Diagram: This diagram serves as a general overview of our game's architecture, showcasing the classes that make up the game, their attributes, methods, as well as the relationships between them. It includes classes for game objects (e.g. player, studying, eating), game states (e.g. menu, playing, paused, game over), and everything else. This structural view allows understanding of the game's data and functional organisation.
Component Diagram: The Component Diagram "is a collection of vertices and arcs and commonly contains components, interfaces and dependency, aggregation, constraint, generalisation, association, and realisation relationships."- Visual paradigm [1]. It may also contain notes and constraints. It complements the Class Diagram by illustrating the larger structural organisation of the game, highlighting how major game components work together. This diagram emphasises modularity and the separation of concerns, key principles that support maintainability and scalability.





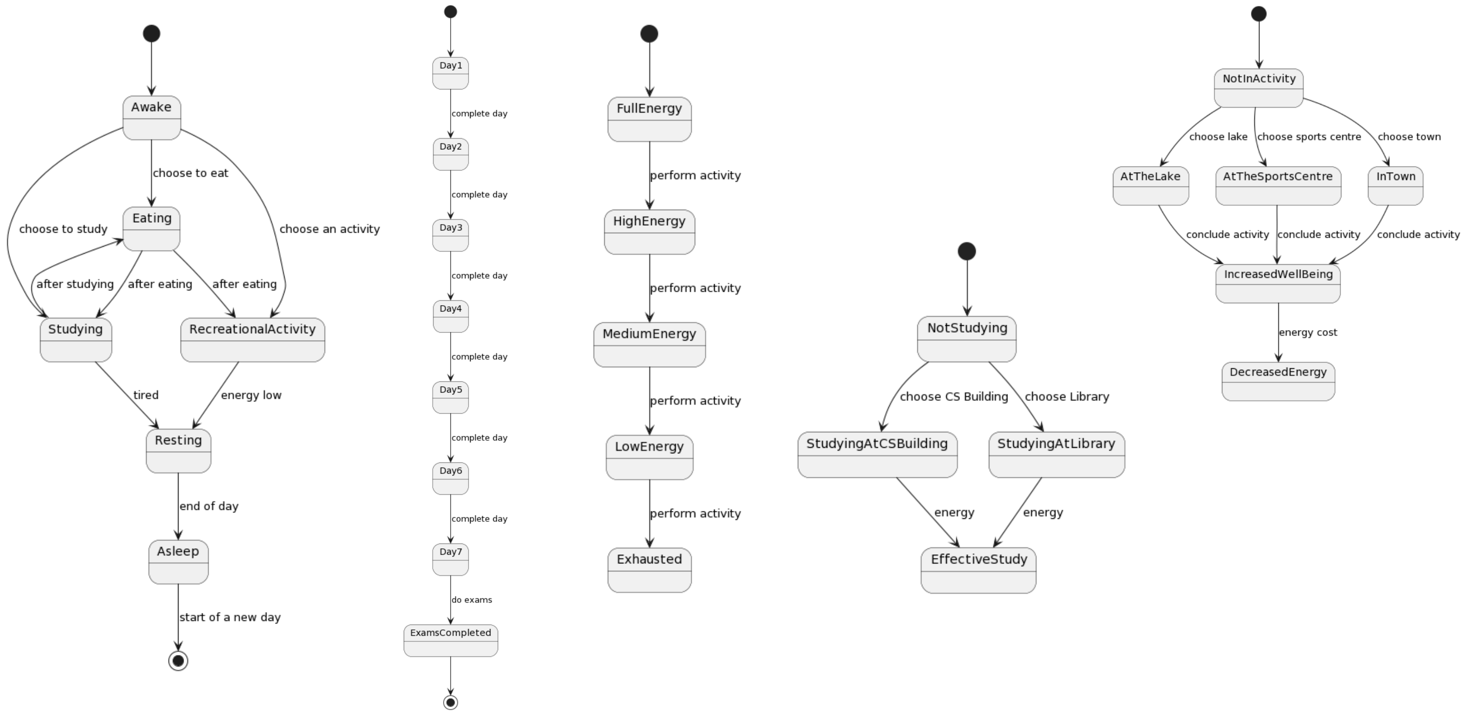
Behavioural Diagrams:



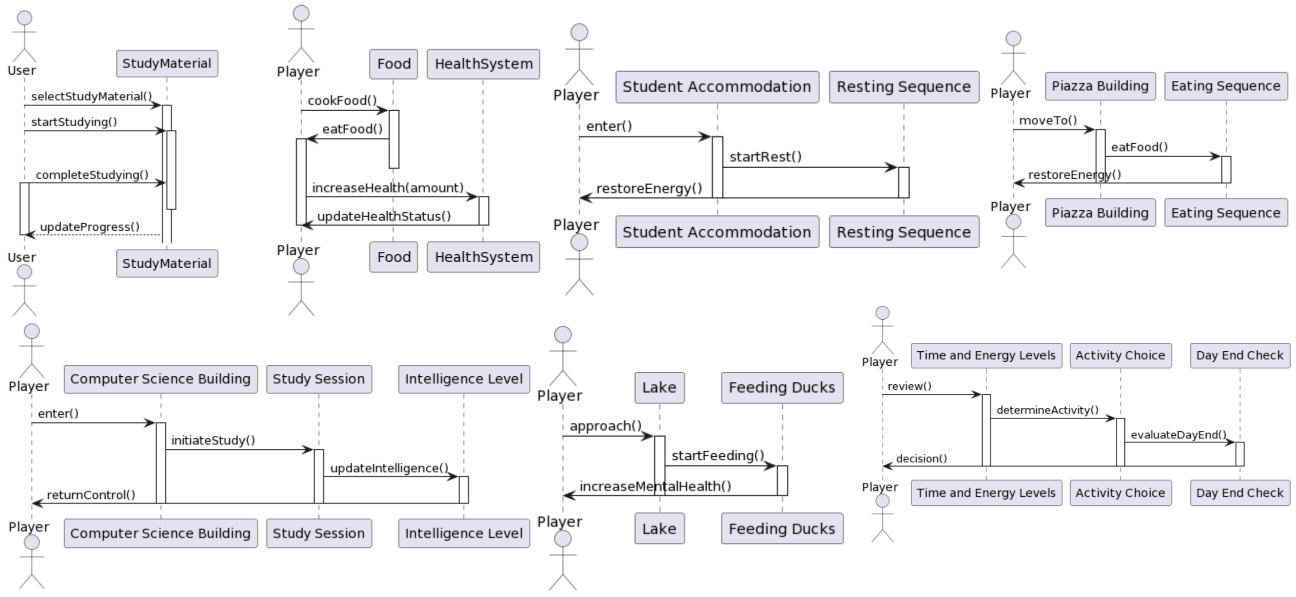
State Diagram: Essential for game development, the State Diagram represents the various states our game can be in (e.g, start screen, in-game, pause menu) and the transitions between these states triggered by the actions of a user. This diagram is crucial for planning the game and ensuring a good overall user experience. "State diagrams require that the system

1
V. Paradigm, "What Is Component Diagram?," *Visual Paradigm*. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/. [Accessed: 12-Mar-2024].

described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction." - Wikipedia[2]

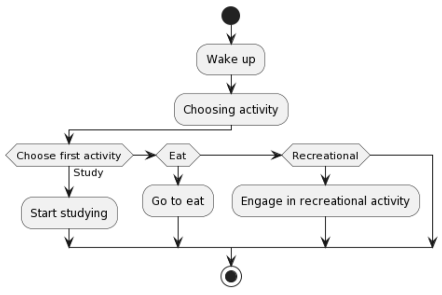


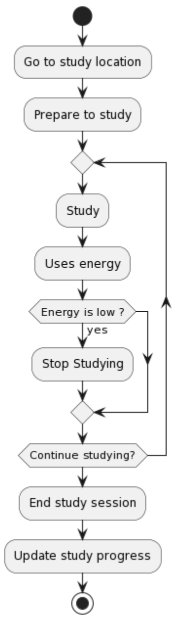
Sequence Diagram: To detail the interactions that occur during a typical game loop (e.g. processing input, updating game state, rendering), Sequence Diagrams are used. These diagrams illustrate how game objects communicate and the sequence of these interactions and how the user can interact with the system and what steps the user does to do this and in what order they are required to do them in. Such as updating the process of when a user is studying or eating as shown below.

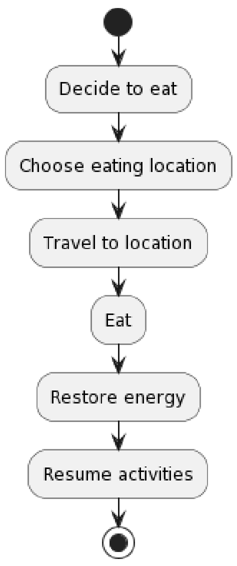


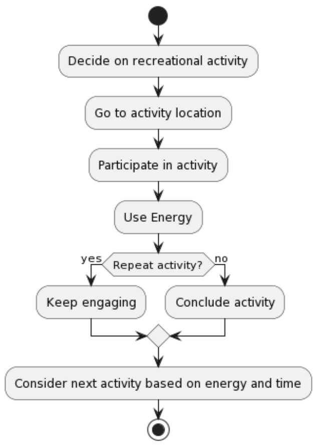
Activity Diagram: Focusing on specific game features, such as study progression or eating, Activity Diagrams map out the flow of actions and decisions within these features. This provides insights into the logic behind game functionalities, aiding in the development and debugging processes. "They show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities." Sparx Systems

2 Wikipedia, "State Diagram," *Wikipedia*, 02-Mar-2024. [Online]. Available: https://en.wikipedia.org/wiki/State_diagram. [Accessed: 12-Mar-2024].









Languages and Tools Used:

The diagrams were created following the Unified Modeling Language (UML) standards. As a group we selected it for its ability to model both the static structure and dynamic behaviour of our system, making it ideal for our project. For diagram creation, we utilised *Plantuml*, chosen for its support of UML and features that facilitate the diagramming of software architecture and it implements well with Google Docs where we chose to write our documentation in making it easier as a team to collaborate on it.

We chose to implement a Gantt chart for our project, which was done by utilising the tool '*Monday.com'*. This decision came from our team's want to establish a structured schedule, ensuring we stay on track and effectively manage all deliverables. Initially, we considered PlantUML for this task, but after a thorough discussion, we found its presentation unsatisfactory for our needs, leading us to opt for *'Monday.com'* instead which is free and accessible for students, offering us plenty of features for task management. It allowed us to assign specific tasks to team members, incorporating task dependencies to clarify what we needed to do and what order they needed to be done in (e.g., Requirement B could start after the completion of Requirement A). This functionality allowed tracking of responsibilities and progress. To enhance organisation and visibility, we maintained a separate document for recording and sorting tasks. The Gantt chart was updated at the end of each meeting, ensuring our plan remained current. Tasks were also assigned priorities ranging from low to critical, enabling us to focus on urgent matters first while gradually escalating the priority of lesser tasks as the project advanced. Additionally, we colour-coded the Gantt chart, improving the distinction between tasks and making the overall project timeline clearer. This not only helped in understanding individual responsibilities but also provided a comprehensive view of the project's timeframe and our self-imposed deadlines, thanks to the timeline feature.

This systematic approach to project management has been instrumental in keeping our team aligned, organised, and focused on meeting our project milestones.

Justification

For our proposed 2D game project developed in Java, the chosen architectural framework was meticulously selected from the set of architecture styles presented to us in the lectures. After going through all the options as a team we landed on Entity-Component Systems (ECS). We thought it would be good for game development and so the coding team is able to separate all the entities and connect them in an object-oriented way. It is also used for its ability to be scalable, flexible and for its performance in game development, making it the overall ideal choice. To clearly relate the ECS architecture to the requirements outlined in the tables provided in Req1, we will use consistent naming of constructs and reference the requirement IDs.

Scalability:

The ECS architecture supports scalability due to its approach. By breaking down game objects into entities, components, and systems, we can easily add new features, activities, or behaviours without modifying existing entities or systems. This modularity allows for the seamless introduction of new game mechanics, such as additional student activities or locations on the campus map, supporting the game's ability to grow and evolve over time which supports the Agile design method we as a group chose earlier in the project.

Maintainability:

The separation of data from logic in ECS makes the game codebase more maintainable. This means the coding team can work on different systems or introduce new components without the risk of creating complex interdependencies that are hard to debug and maintain. This clear separation also makes for easier testing on individual aspects of the game, such as tweaking the logic for energy consumption during activities or modifying the attributes of specific activities without affecting unrelated systems.

Performance:

ECS architecture typically offers improved performance over traditional object-oriented approaches, especially in systems where there are many entities with varying sets of components. Since systems in ECS operate on components of entities in a data-oriented manner, this leads to faster execution of game logic. This is particularly useful for creating and maintaining smooth gameplay.

Team's Familiarity with the Technology:

Our team has a rough foundation in Java, but as specified in the product brief the client requires the game to be coded in Java 11. After the coding team took some time to research Java and the use of libGDX they became a lot more familiar with it as a concept and started making progress with the game. The time spent researching and developing their skills accelerated the development time. Additionally, Java's and libGDX's robust set of libraries and frameworks for game development provided us with lots of documentation to look through and use in order to create a fundamental base for our game meeting our requirements. By using ECS it gave the coding teams a clear structure to follow making it easy for other members of the team to contribute and get onboard with the project as a whole without having to understand the systems all in all.

Design Process

You can see the early layout of the system we as a team created through the use of CRC cards:

Class Name

Responsibilities	Collaborators

Character	
Responsibilities: - Moves around map - Interacts with buildings / NPCs	**Collaborators:** Map InteractableLocations

Map	
Responsibilities: - Contains InteractableLocations at different locations - Contains the Character - Provides methods to check energy of character	**Collaborators:** Character InteractableLocations

InteractableLocation	
Responsibilities: - Character can interact with this to complete a task - Resource cost for task displayed before interaction confirmed by player - Performs a check of energy cost and time consumption vs the Character’s energy and time remaining, only allowing interaction to proceed if sufficient resources remain for both. - Contains the values for resource modification (e.g., cost 10 energy, 2 hours time, score + 5)	**Collaborators:** Map Character

CharacterSelectScreen	
Responsibilities: - Provides clear visual indication of character selection - Allows user to select an avatar which will be used as the characters sprite throughout the game - Moves into GameScreen once a character is selected (by clicking on them).	**Collaborators:** Character GameScreen

GameScreen	
Responsibilities: - Contains and runs the game - Takes inputs from the user - Changes map when player moves to a certain location / interacts with a building - Displays available resources to the player - Opens help screen when a specific button is pressed - Keeps track of the time and days and displays to player	**Collaborators:** Map Character HelpScreen ScoreScreen

- Tracks and displays how many interactions have occurred with each InteractableLocation - Goes black and displays “Day: X” at the beginning of each new day. - Shows the value of health and intelligence	

ScoreScreen	
Responsibilities: - Displays final score - Allows user to restart the game from CharacterSelectScreen	**Collaborators:** GameScreen CharacterSelectScreen

HelpScreen	
Responsibilities: - Clearly displays a simple explanation of how to play the game. - Closes and returns to the GameScreen on input.	**Collaborators:** GameScreen

This then evolved throughout the implementation phase of the project as the coding team learnt more about the system and how it works. The coding team took these cards as a basis to work off of the system. We kept to the same variable names so it made it easier to implement.

1. **Character Control and Interaction** (UR_CHAR_CONTROL, FR_CHAR_CONTROL):
    - CRC Card - *Character*: Initially envisioned as a simple entity able to move and interact, it evolved into a more complex set of components managed by ECS, enabling smooth control and interaction within the game environment.
2. **Character Avatar Selection** (UR_CHAR_SELECTION, FR_AVATAR):
    - CRC Card - *CharacterSelectScreen*: From the simple selection mechanics, we introduced an ECS-driven avatar system that allows players to choose their character at the start.
3. **User Experience and Error Feedback** (UR_UX, NFR_ERROR_FEEDBACK):
    - CRC Card - *HelpScreen* & *GameScreen*: The HelpScreen and GameScreen were refined to offer tutorials and real-time error feedback.
4. **Game Time Management** (UR_GAME_TIME, FR_DAY_END):
    - CRC Card - *GameScreen*: The GameScreen component's responsibilities expanded to manage in-game time, aligning with the '16 hours' gameplay requirement and ensuring the day ends appropriately.
5. **Activities and Score** (UR_ACTIVITIES, UR_SCORE, FR_COUNTER, FR_USER_SCORE):
    - CRC Card - *InteractableLocation* & *ScoreScreen*: We developed components that track and display activity completion and scoring, linking these actions to the player's ability to interact with game objects and see their score, satisfying the requirements for activities and scoring.
6. **Game Replay and Termination** (UR_REPLAY, UR_END_GAME, NFR_TERMINATION):
    - CRC Card - *ScoreScreen*: The ScoreScreen allows the user to replay the game and shows proper game termination, ensuring the game adheres to the requirements for replayability and termination.

Throughout the project, we maintained consistent naming conventions and clear references to requirement IDs, ensuring traceability from our CRC cards and architectural components to the specific user and functional requirements. This approach not only streamlined the development process but also helped with communication within the team by providing a clear idea of where and how each requirement was addressed within our game's overall architecture.