Architecture

Cohort 3 Team 1: Pixels of Promise

Movitz Aaron, Jake Adams, Oliver Belam, Anna Burt, Emily Foran, Sky Haines-bass, Job Young

University of York

ENG1: Software & Systems Engineering

November 11, 2024

3.1 Introduction

3.1.1 Brief introduction to diagrams

Below are diagrams showing the architecture of our game, UniSim. There are a variety of different types of UML diagrams, mainly divided into two types: structural and behavioural. These diagrams have changed over the history of the project, so old versions, and initial designs like CRC cards have been added to the website; they can be found in the architecture sub-page.

3.1.2 Brief statement about UML and tools used

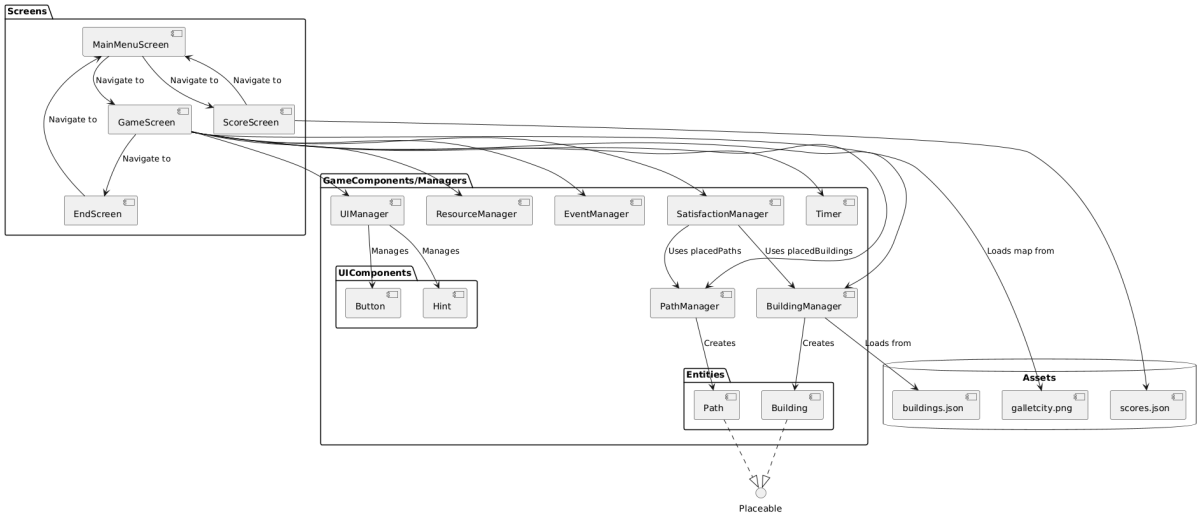
These diagrams were primarily created using PlantUML, a versatile tool that allows for the generation of UML diagrams from plain text descriptions. Below, we detail the specific languages, tools, and processes employed in creating and maintaining these architectural representations.

Tools and Languages Used

- **PlantUML**: We employed PlantUML to create our UML diagrams due to its ability to produce clear and maintainable textual descriptions of complex diagrams. This facilitated easy version control and collaborative editing among team members.
- **IntelliJ IDEA with PlantUML Integration**: During the development phase, all PlantUML diagrams were authored within IntelliJ IDEA using the PlantUML Integration plugin. This setup provided a seamless environment for writing and previewing UML diagrams alongside our codebase.
- **PlantUML Gizmo for Google Docs**: For our documentation, we integrated PlantUML diagrams into our Google Docs using the PlantUML Gizmo plugin. This enabled dynamic updating of diagrams within the document whenever the underlying PlantUML code was modified, ensuring that our documentation remained up-to-date with the latest architectural changes.

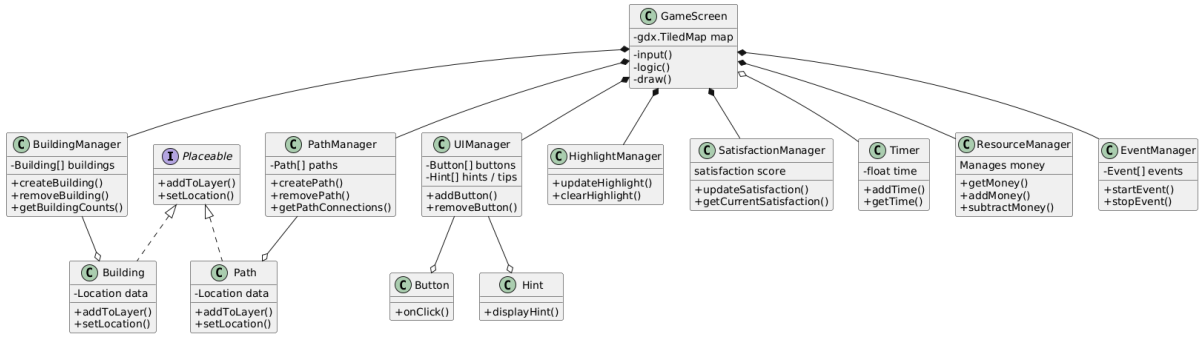
3.2 Structural diagrams

3.2.1 Package diagram for the entire game



Package diagram for the game. Here is a diagram for the packages in the game. We are mainly using screens, interacting with managers, that then further interact with entities and other components.
Here is a package diagram covering all the components of the entire game. It is split into 3 main components, screens, managers and assets. The screens handle all the input, logic and drawing to the screen, some of this logic might be passed onto a manger, this is to simplify the screen and allow a more modular approach. For example, some logic can be passed on from the screen to the BuildingManager that will then create a new object, placing it on the map, to then be rendered next frame. There is only one interface in the game, this is the Placeable interface. Multiple “databases” are used, these are all stored as assets. These are just files that have data we need to use in.

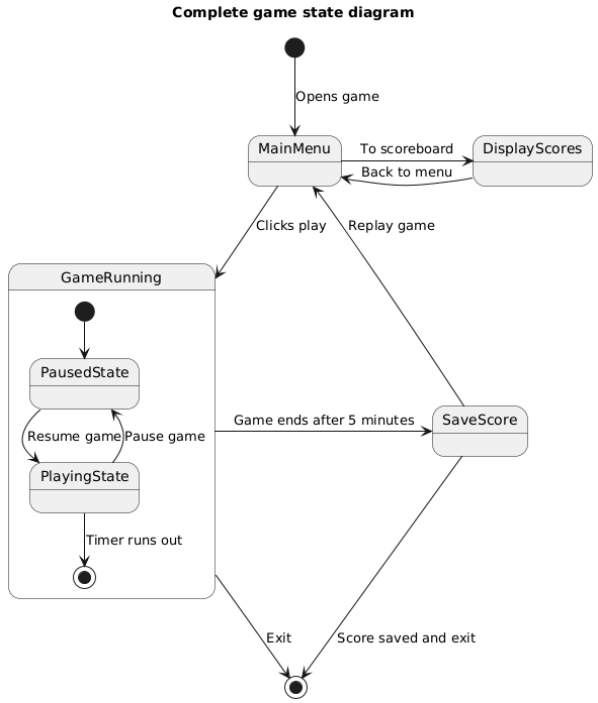
3.2.2 Class diagram for game screen and its components



This is a class diagram that shows the basics of the classes within our program structure, as well as how they interact with each other. The attributes and methods listed for each class are not exhaustive, they simply represent a high-level overview of the functionality of each class. GameScreen communicates with all of the different managers, giving the program structure a microkernel-like feel. There are some classes omitted from the diagram, like the other screens, although they will share the same functionality.

3.3 Behavioural diagrams

3.3.1 Complete overview game state diagram



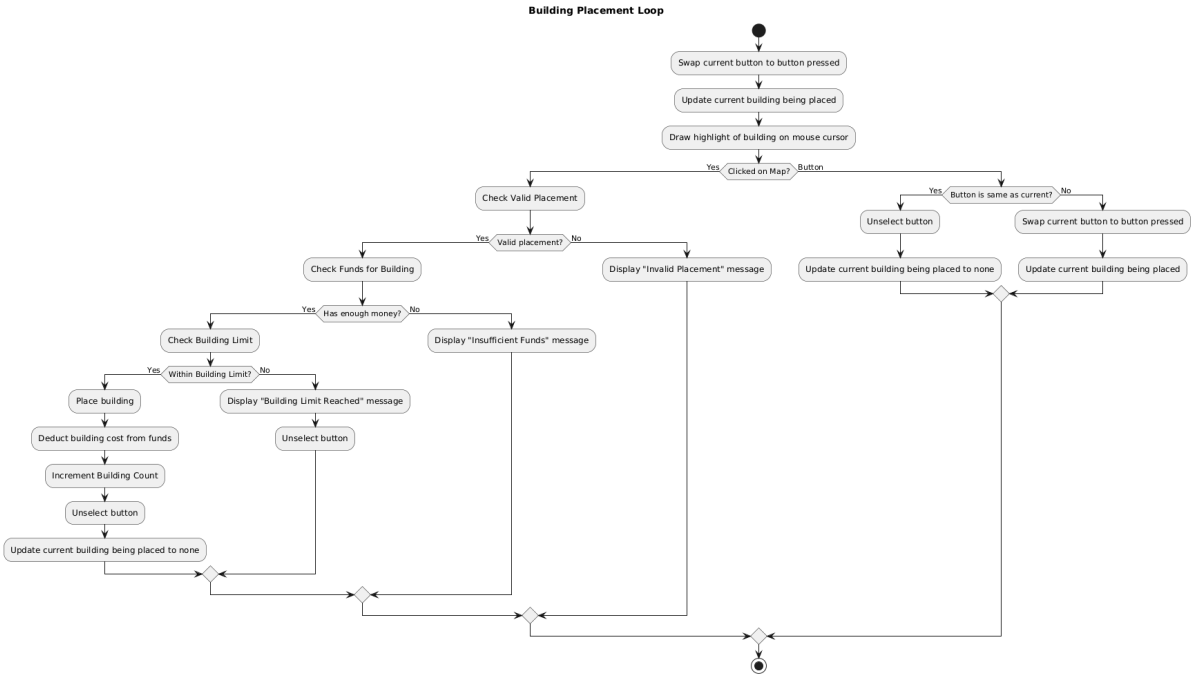
This state diagram in plantUML simulates the most fundamental behaviours for the game. The functional requirements demonstrated here are the FR_TIME_SIMULATION requirement (where the game shall simulate progression throughout the whole 5 minutes) and the FR_GAME_PAUSE requirement (the game shall allow players to pause and resume gameplay whenever they wish). Both FR_TIME_SIMULATION and FR_GAME_PAUSE link to the user requirement UR_USABILITY_PROMPT (the game shall contain an instruction/prompt to aid with usability).
It could also be argued that this diagram links

to NFR_ERROR_HANDLING, as, the fit criteria to fulfil this states that “The system should crash <5% of the time..”, and, the state diagram shows the system successfully handling the whole 5 minutes of runtime and terminating appropriately.

3.3.2 Detailed activity diagram for the game

You can find this diagram on the website, in the architecture sub-page. Under the heading UML diagrams not included on deliverable, subheading 3.3.2.
This activity diagram provides an overview of the whole system, including setup before the game is started, and score saving afterwards.
To begin, this links to NFR_PERFORMANCE, as it references initialisation of game resources, textures, assets, buttons and the game screen. The system then outlines the activities undertaken during game runtime, showing clear links to FR_GAME_PAUSE, FR_SATISFACTION_CALC, UR_SATISFACTION_METRIC and UR_BUILD. It also shows when the system will handle planned/unplanned events, clearly linking to FR_EVENT_TRIGGER.
Finally, this diagram also has implicit links to NFR_ADAPT, as this states that the “system should handle the increasing complexity of the campus design”, and, the diagram details the systems activity and thus, links to how it handles runtime.
The activities that start with “Handle” are described in more detail in the following diagrams, to maintain a simple overview for the game.

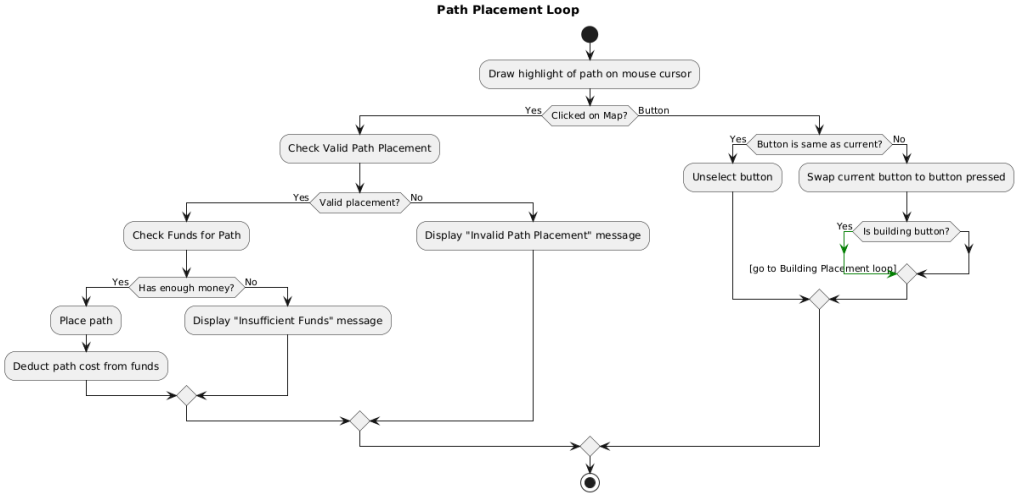
3.3.3 Handle building placement



This activity diagram details the process that the system takes to facilitate building placement in the game. It addresses key requirements such as UR_BUILD (enabling users to add buildings to the map) and FR_COST (requires users to have a minimum amount of currency to place a building). This ensures that each building placement follows the pre-established rules and limitations.

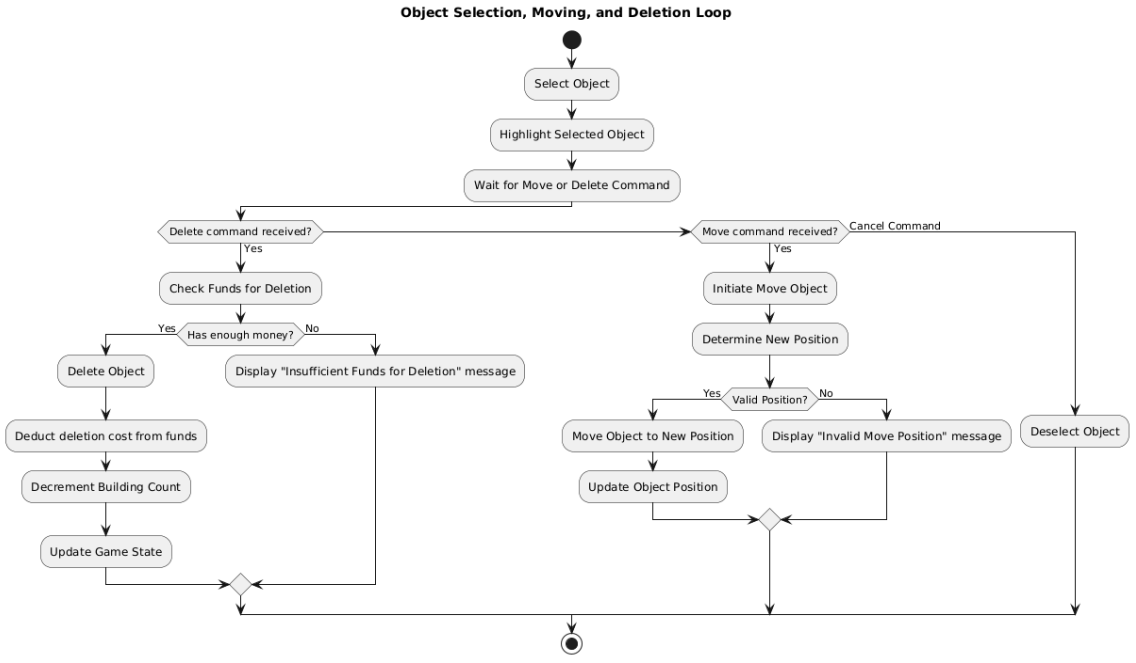
Additionally, this diagram shows links to UR_UX and consequentially, NFR_INTUITIVE. This is because it details how the UI will respond to user interaction, via button highlighting and displaying appropriate ‘error’ messages to the user if they attempt to do something that is not allowed (Eg, place a building on a lake).

3.3.4 Handle path placement



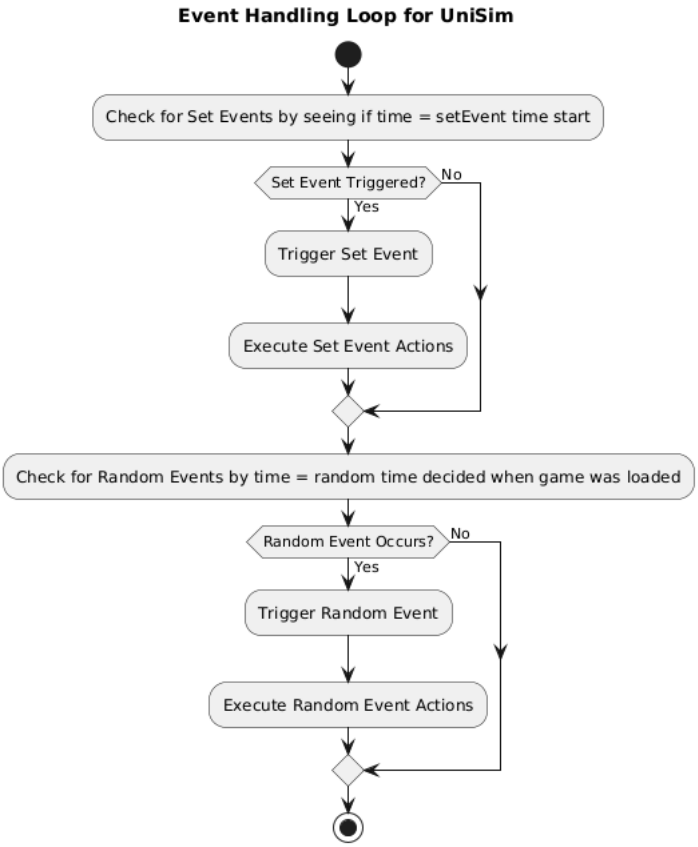
This activity diagram outlines the process for handling user input and validating the placement of paths on the map. This diagram links to the UR_BUILD and FR_COST requirements, as it displays how the system allows paths to be placed, whilst adhering to budget and map constraints.
It also displays a way that the system fulfils the NFR_INTUITIVE non functional requirement. This is due to the diagram detailing specific feedback messages that shall be displayed to the user upon an appropriate scenario occurring, thus avoiding further confusion of the user.

3.3.5 Handle object selection, movement and deletion



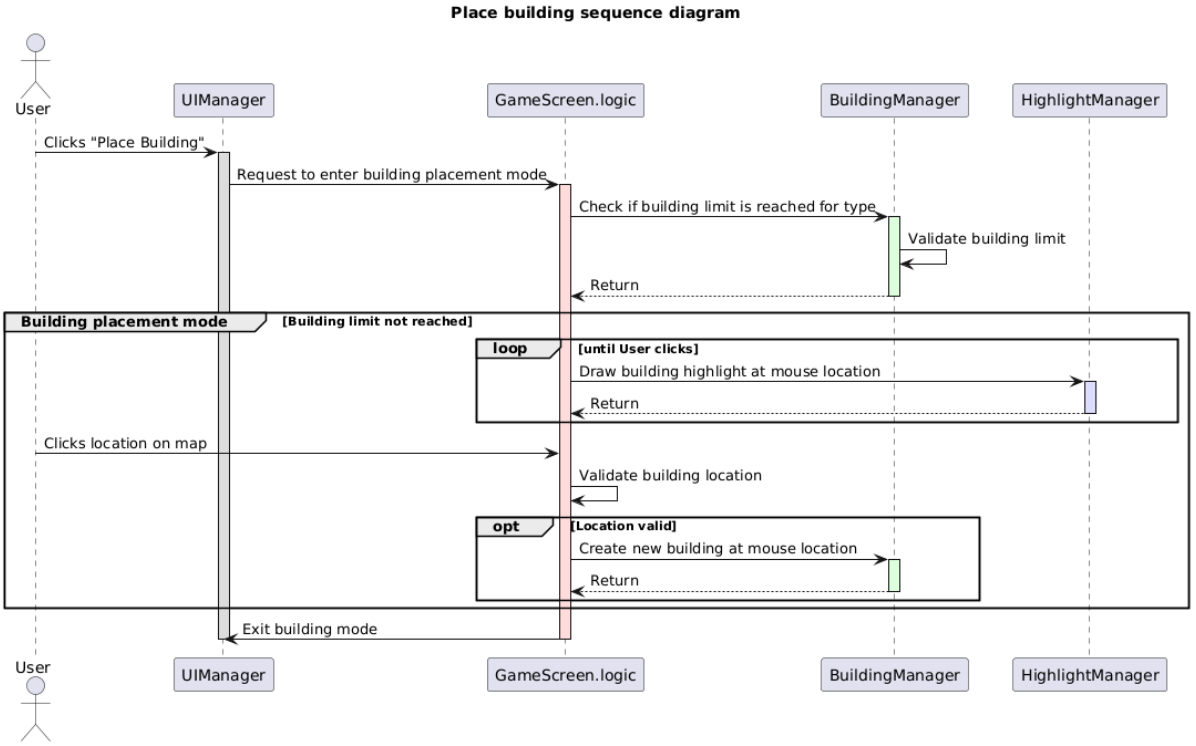
This activity diagram explores and expands upon the delete mode established in the delete_mode_sequence_diagram.puml, whilst also introducing a similar mode that the user can use to move building objects. This links to FR_BUILDING_DELETION for the same reasons as delete_mode_sequence_diagram.puml, whilst also building upon the ‘managing’ aspect mentioned in UR_BUILD, as moving buildings is a further managing feature. This also has ties to the requirement FR_BUILDING_PLACEMENT, as this details the user should be able to change building position during the game providing map constraints, and, in the activity diagram, a validity check is shown before placement is confirmed, fully satisfying this requirement. Additionally, the diagram links to FR_COST as it shows the system checking if the user has enough currency to complete the desired action.

3.3.6 Handle events



This is a simple activity diagram, detailing the process in which the system undertakes to ‘decide’ when to trigger a special event. This feature is to be implemented in the second assessment phase, and thus, does not explicitly link to any of the user or functional requirements established for the first assessment phase. However, it can be argued that it does implicitly link to FR_TIME_SIMULATION, as events are implemented to be triggered at set times.

3.3.7 Place building sequence diagram



This sequence diagram outlines the process of placing a new building on the map. The user requirements demonstrated here are the UR_BUILD requirement (the game shall allow the user to place and manage several different building types), and the UR_LIMIT requirement (The system shall limit the number of buildings the user is able to place).
The functional requirement FR_BUILDING_PLACEMENT (the system will allow the player to place buildings on the map and change their position throughout the game, providing map constraints) is also exhibited in this diagram.

3.4 Closing remarks

The structural diagrams, including the Package and Class diagrams, elucidate the modular organisation of our system, highlighting the interaction between screens, managers, and assets. This modularity not only simplifies the complexity of the game but also enhances scalability and maintainability. On the behavioural side, the State, Activity, and Sequence diagrams provide detailed insights into the dynamic processes and user interactions within the game, ensuring that all functional and non-functional requirements are addressed.

Throughout the project, our architecture evolved iteratively, informed by continuous feedback and refinement. Initial designs such as CRC cards served as foundational blueprints, guiding the transition to more sophisticated UML diagrams that accurately reflect the system's growing complexity. This evolution can be seen on our website, although there are not a lot of old diagrams, there are some.