# Architecture

Team 8
OctaGame

Lisandro Nascimento
Thomas Nash
Letam Patrick-Bienwi
Nathan Peacock
Will Peel
Harrison Pegg

Our architecture serves as a blueprint or guideline for implementation. However, there will be times when implementation may differ slightly from it. Reasons for this could be technical limitations, unforeseen risks or performance issues. Tools used within our architecture include: VS code PlantUML extension to draw the interim versions of Class Diagrams, Lucidchart to draw the Final UML Class Diagram and the Use-Case Diagrams.

The general aim of our architecture is to give a high level overview of our system to guide development. We focused on splitting the game structure into four groups in order to focus on the primary components that would make up the game, these include the buildings, the map, the GUI, and a group for resources. Through several meetings and discussion we made a plan about the architecture of these groups and how they would work together in a layered structure.

The GUI group represents the presentation layer, used to display the visuals of the game that the user can interact with. The inputs a user gives the GUI are controlled by the business layer, in our project the buildings and the map act as this, all logical elements of our game are controlled within these groups. The buildings and the map also act at the persistence layer, a constant loop runs throughout the game, constantly updating variables, and with each input and movement from the user the logic saves new values and gives them to the GUI group in order to display new information. Finally, our resources group acts as the data layer for our game, every required variable and asset are stored within the resources in order to make the structure of our game as maintainable as possible. However, in order to accomplish that we must first understand the product as well as user requirements.

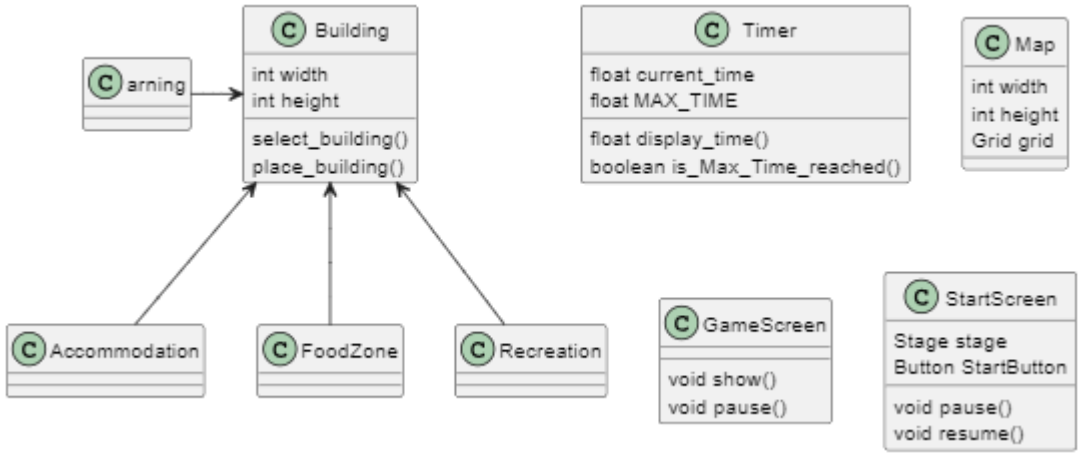
Our brief says: implement one of each building location (one place to sleep, one place to learn, one place to eat, one recreational activity), a tracker of the time the game lasts (up to 5 real-world minutes), and a simple counter denoting how many of each building have been placed so far.

Upon this brief, we extrapolated  various requirements, both user and functional into a structured document. However, these requirements don't show enough. So we combine user requirements and conversations with our stakeholders into a "story" that encapsulates our findings. This story maximises abstraction and allows us to start thinking about individual objects and components. From this we're able to create CRC cards for each component, we start to assign purpose, categorise components and see relations between components. Additionally we also include collaborations between components.

A link to the CRC cards can be found in the references section in this document.

We use these cards, with the distribution of responsibilities to map towards actual language concepts. From this, we create our first UML class diagram iteration, that has these building blocks but with things that we believe are essential.

**First iteration**




Within this iteration, a couple of deviations occur. For starters, “Building Manager” is not included while “Building” is included with 3 other classes that inherit from it. Our thought process was whilst we know the need for a building manager, we need a general class for actually creating building types. That is what “Building” does. The class allows you to select buildings and place them on the map. It does not require any instantiation. This class fulfils the requirement UR_BUILD. In addition, we include subclasses “Learning”,”Accomodation” “FoodZone”, and “Recreation” which act as each building type, each one can hold its own properties maximising deployability and scalability. These classes fulfil the requirements UR_SLEEP_PLACE, UR_LEARN_PLACE, UR_EAT_PLACE, UR_RECREATIONAL_ACTIVITY.

Considering the requirements UR_TIME_TRACKER, FR_CLOCK and UR_GAME_PAUSE we also include the “Timer” class. This class tracks the time of the game and continually checks if the current time has exceeded the allocated maximum time of 5 minutes.

We also include the “Map” class. This class provides an area where players can place buildings on. we wanted a certain level of simplicity which would restrict the user as to not overcomplicate the functionality and gameplay. The requirements UR_MAP and FR_MAP outline our intent for the map's design.

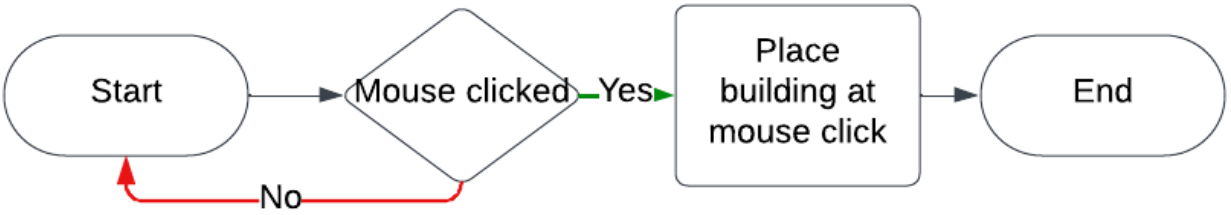
GameScreen simply is what the users can see, it is where we display our map, building, structure etc. It fulfils requirements UR_GRAPHICS. Start Screen represents the first screen you see and contains the start button to commence the game.

**Handling buildings:**

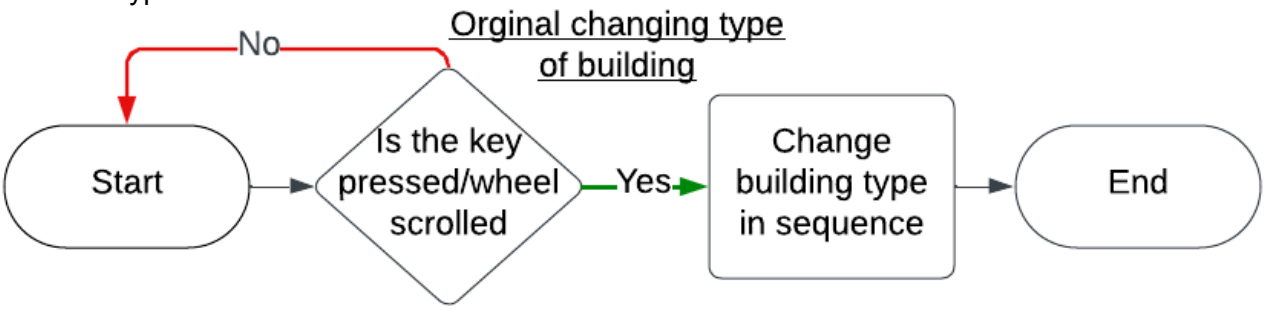
As the game revolves around the buildings in a campus our primary goals were to ensure the functionality for building the campus was efficient in terms of logic and how it would interact with other groups, and also to be intuitive for the user to interact with. The main requirements we wanted to focus on were UR_BUILD, FR_BUILDING_PLACING, NFR_BUILDING_INTERACTIONS, UR_GAMEPLAY, and NFR_GAME_INTERFACE.

First Ideas

In our group meetings we discussed how we would tackle the placing of buildings in the game, after some brainstorming we came up with a few ideas. Originally we tested having buildings be placed where the user clicked on the screen.

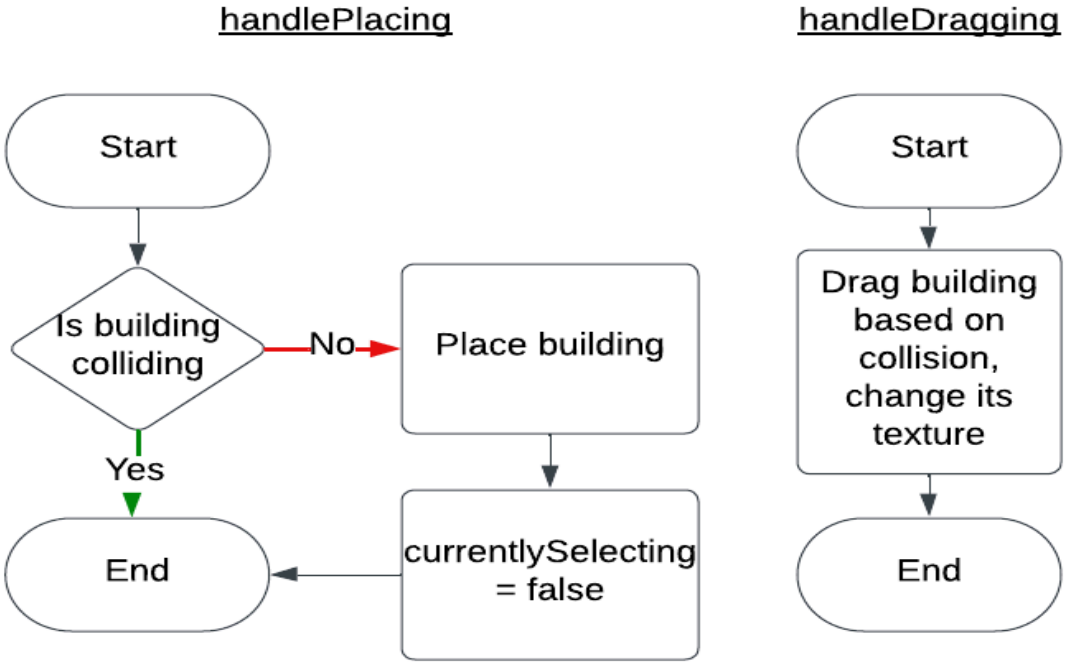



This posed a few problems. Firstly, there would be no restrictions to where the user can place a building such as only being allowed within grid cells, this would've added unnecessary complexity with ensuring we maintain the limits of the map. In addition, this idea affected how the user would select a specific type of building, we had ideas involving a set sequence of how you should place buildings, or having the user scroll through the different types.

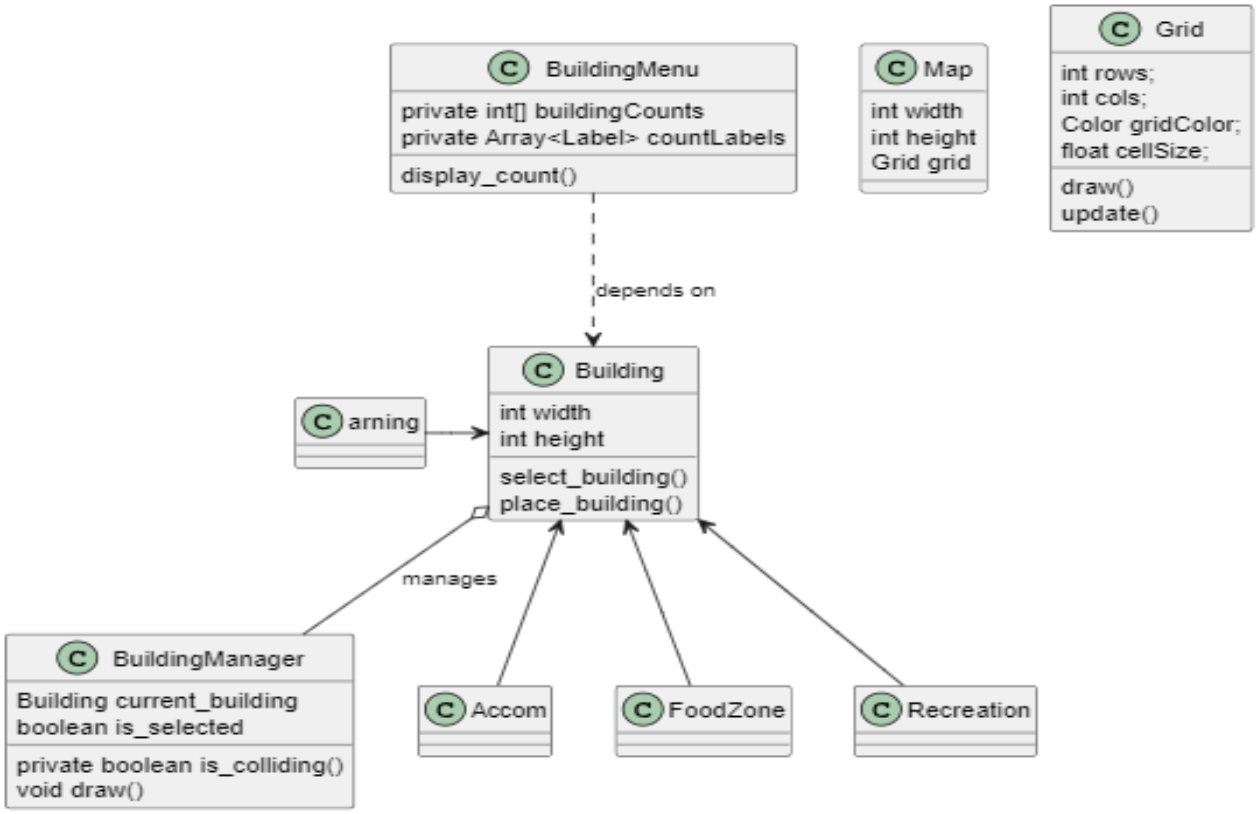



Final Idea

In order to ensure the system is easy to use and fun to interactive we opted for a drag and place approach involving three steps. Firstly, the user would click a button labelled with the desired building type, this adds a higher sense of choice for the user compared to a key or scroll wheel approach. The next step involves dragging the building across the map, and finally the user would click again to place it. When testing how this felt as a player we found it significantly more interactive and much easier to understand and use.



## Iteration 2



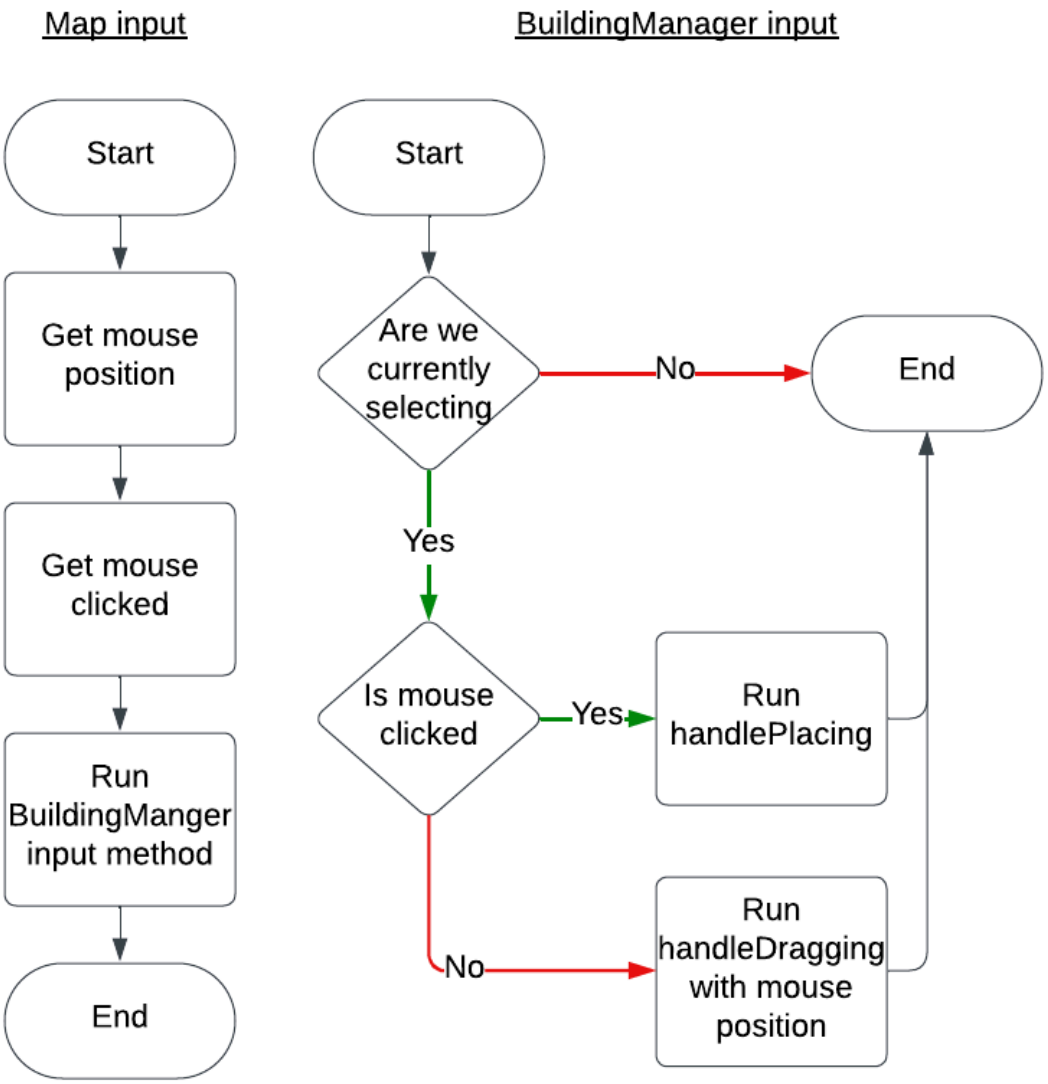

This iteration includes everything from iteration 1 but with a couple of new additions. For starters, 2 new classes are included: “BuildingManager” and “Building Menu”. We wanted to meet the requirement of placing  and actually drawing different building types. The new class aims to also handle collisions between buildings. That is if players decide to place buildings in the same position, it ensures it cannot happen.

We created the BuildingMenu class, which would contain the buttons that are used to create the buildings. We also added labels to the buttons to display the count for each building. This class helped meet the requirement of ensuring that every building type being placed was counted and displayed correctly. It covers FR_COUNTER_INCREMENT. The Building Manager class on the other hand is responsible for managing the general placement of building types. It keeps track of what building was being made, where it was placed and ensured that there was no collision in building placement. I.e buildings were not being placed on the same spot.  It covers requirements FR_BUILDING PLACE. It also took care of the process of actually drawing the building.

We show that Building Menu depends on the class Building. If there is no building placed, there is no building to be counted. Building Manager on the other hand deals with the tangible features of buildings and ensures appropriate placement.

Making a grid

After tackling the buildings our first thought was to implement a grid system into the map that would split it into cells rather than have unrestricted movement throughout the game. This was the obvious choice after analysing how we were going to handle buildings, especially when considering the issues regarding our first ideas and the requirements that they're detrimental to. In order to connect the structure of the buildings with the map we had two input methods, the map would constantly get the mouse position and check if the mouse has been clicked, and then pass this information through to buildings, independently a separate method would take control of the dragging and placing.

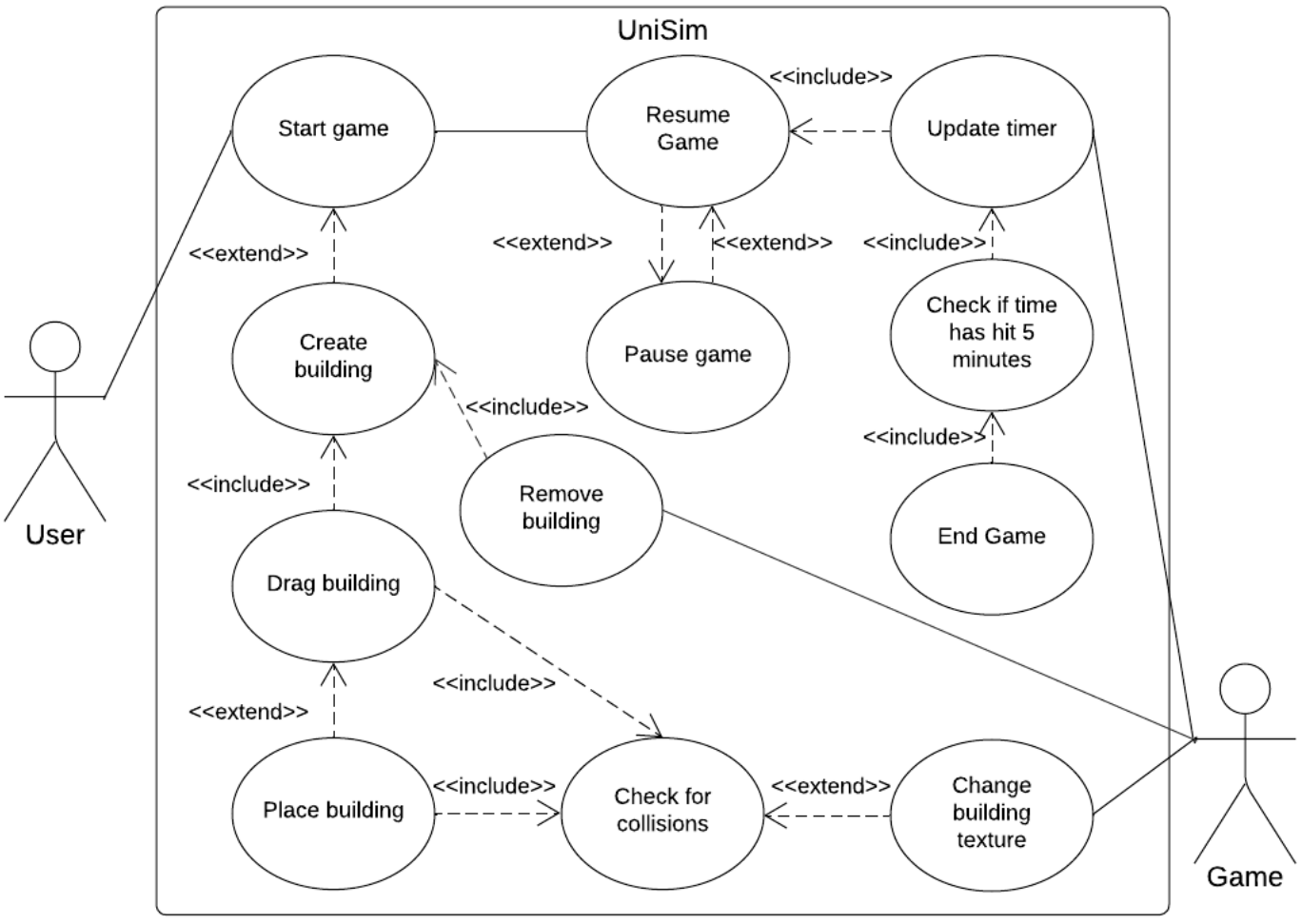



After having this system in place and conducting thorough testing we found that we required a remove system to allow a user to remove a building that they are currently selecting in order to compensate for user error. We created a new user requirement UR_REMOVE_BUILDING. This was a simple change due to the extensibility of our system, and we found it increased the sense of completeness with the game. A user can remove a building they are selecting by pressing backspace.

**GUI:**

Our focus was to make a simple 2D fun interface for the user, we wanted the visual aspects of the game to heighten the users experience and combine them with the logic of the map and the buildings for cohesiveness throughout the structure. We split the GUI group into three subsections that outline the process of how the game is played, start screen, game screen, and end screen. Together they describe the natural flow of the game and each section the user will interact with. The requirements NFR_GAME_INTERFACE, UR_GAMEPLAY, UR_GAME_PAUSE, UR_GRAPHICS, UR_TIME_TRACKER, UR_BUILDING_COUNTER, UR_BACKGROUND_MUSIC,UR_VISUAL_AND_SOUND_EFFECTS, FR_CLOCK, FR_COUNTER_INCRIMENT, NFR_GAME_END were all considered for the creation of the GUI.

Both the start and the end screen are simple but useful interfaces that anchor the structure of the game together creating a full sequential experience for a player. The main component of the GUI is the game screen, which initialises the map group and contains a game menu. The game menu itself consists of a building menu and a timer, in our group we discussed the layout of the menu and decided on a sidebar to display it. We decided on this as distinguishing information and interactive aspects of the game, and the map where the game takes place would benefit the usability and ease of use of the system. We built the menu one problem at a time creating different iterations, each one building on the next. The first iteration consisted of purely the game buttons which create the buildings, we then added the counter for each one, and combined that with a timer and pause functionality (the user can press space to pause and unpause the game), completing the structure of the GUI. The diagram below shows the flow of the game and how the layers interact.



**References**

CRC Cards Link:
https://6708f185-154c-4209-83c3-4f5273aa7fbf.filesusr.com/ugd/53bd28_6e8de265ed9e4558ad79d1913ca8236b.pdf

PlantUML Class Diagram Source-Code Link:
Iteration1:
https://6708f185-154c-4209-83c3-4f5273aa7fbf.filesusr.com/ugd/53bd28_e8417753a8974d6b9009ad9ef99d712f.txt?dn=iteration1.txt

Iteration2:
https://6708f185-154c-4209-83c3-4f5273aa7fbf.filesusr.com/ugd/53bd28_782a6fe4ce924cf18ff1b1684d45235b.txt?dn=iteration2.txt

Final UML Class Diagram Link:
https://6708f185-154c-4209-83c3-4f5273aa7fbf.filesusr.com/ugd/53bd28_ef31db3874b643fcba1a5141d1801bda.pdf