Version 0.1 2022/23

[Tool Development]

Technical Design Document

Kieran Cooksley – C018249L

Kieran Cooksley

C018249L@student.staffs.ac.uk

Contents

Project Introduction

The project aims to create 3 sets of tools for use in Unreal Engine 5 that utilise Editor Utilities to generate and manipulate assets and data outside of runtime. Assets will be visualised in real-time as the tool is in use and will aim to speed up and optimise the users' workflow.

Project Goals

The goal of each tool will be to speed up the user's workflow, optimising tasks and solving challenges faced in the discipline of the target demographic.

Toolset 1: Level Kit

Useful for anyone who works extensively with the world editor/outliner in Unreal Engine, especially Level Designers producing block-outs. This toolset allows for easier organisation of level assets along with tools to more efficiently manipulate transform settings.

Toolset 2: Building Generator

This tool aims to quickly produce building assets using procedural generation of walls, doors, windows, and props. Procedural building generation can speed up the workflow of artists for use as a foundation to build existing features, as well as other disciplines such as designers or programmers to easily generate structures to populate their levels and worlds.

Toolset 3: Ship Configurator

This tool will allow users to quickly generate different variations of pirate ships. Meshes can be customised including toggling the masts, sails, and cannons, as well as material customisation on the ship. Ship data can be modified in the widget and initialised into the actor at run-time. This allows the end user to quickly generate ships of different appearance and in-game behaviour.

Challenges and Risks

The main challenge of this project is having no prior experience with Editor Utilities in Unreal Engine or manipulating objects outside of runtime. This project will require extensive independent research and experimentation that expands on the class content to achieve a satisfactory outcome with all brief requirements for each tool implemented and functional.

Project & time management will also come into consideration. Building 3 toolsets in 9 weeks is a fairly short time frame and I run the risk of failing to meet the deadline if projects are over-scoped or time is used ineffectively. To resolve this, each tool will be dedicated a certain timeframe, and a cohesive plan will be drawn up to make sure all required features are built and functional within the allotted time.

Hardware Requirements

Below are the hardware requirements to run Unreal Engine 5 that is recommended on the Unreal Engine documentation, as well as the hardware specifications for Epic Games' systems as well as the University Lab PCs

	UE recommended*	**Epic Systems****	**University Lab PCs**
Operating System	Windows 10 64-bit	Windows 10 64-bit	Windows 10 64-bit
Processor	Quad Core, >2.5GHz	6 Core Xeon E5-2643 3.4GHz	8 Core i7-11700 2.5GHz
Memory	8GB	64GB	32GB
Graphics	DirectX 11/12 compatible	NVIDIA RTX 2080 super	NVIDIA RTX 3080

***Recommended hardware**

**** Performance Notes**

https://docs.unrealengine.com/5.0/en-US/hardware-and-software-specifications-for-unreal-engine/

Platforms

Target Platform

The toolsets created are designed to be used on PC.

Engine Summary & Specifications

Unreal Engine version 5.2 will be used for the project.

Refer to Hardware Requirements for a table documenting the recommended specifications to run the engine and toolsets.

The Water System Plugin is needed for the Ship Configurator toolset.

https://docs.unrealengine.com/5.0/en-US/water-system-in-unreal-engine/

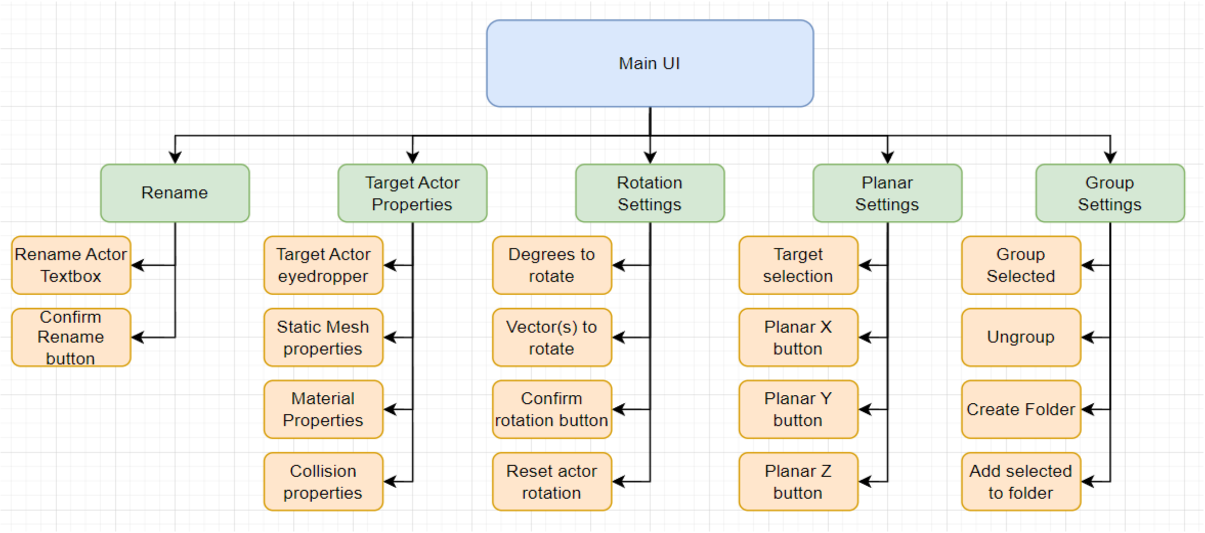
Systems and Diagrams

[Document the planning of your systems here, each system should have it's own section and include a range of diagrams, tables and charts where applicable]

System 1 – Level Kit

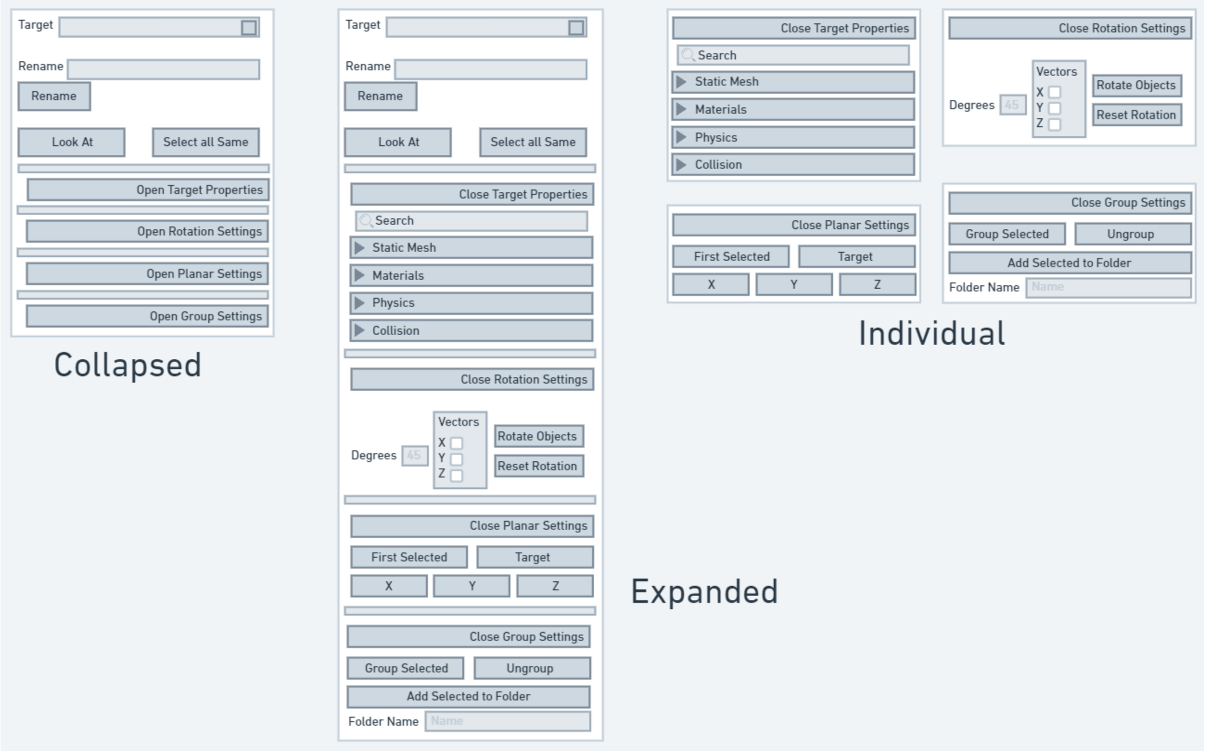
IA Diagrams

This diagram depicts the architecture and structure of the tool's widget. Nested into the main UI of the tool are the various dropdown menus and settings. From there, each specific action is grouped together with similar actions, allowing the user to collapse and show only specific tools they need at any given moment.



UI Wireframes

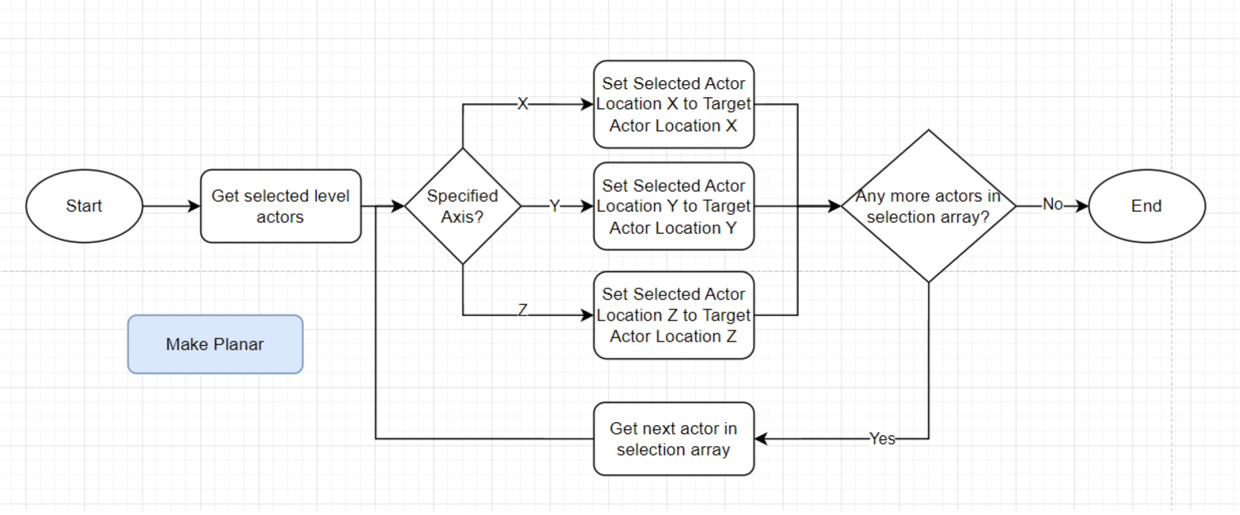
This diagram depicts various UI wireframes for the tool. Each menu setting is displayed in its individual group, as well as how the entire tool looks when all settings are expanded and collapsed.



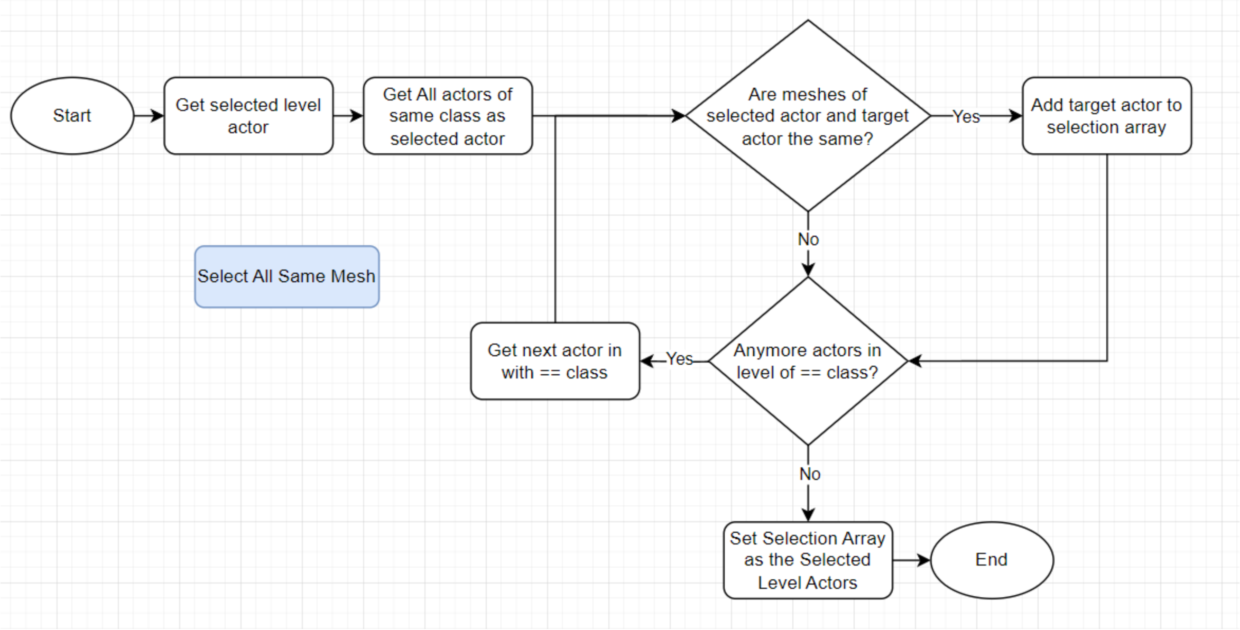
System Flowcharts

A series of flowcharts have been created to visualise the flow of execution of various features.

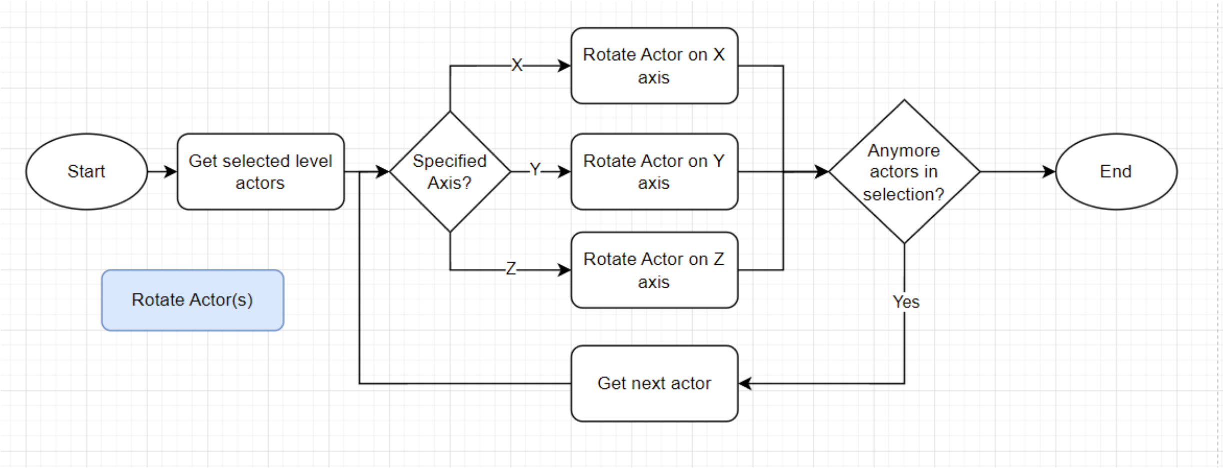
Make Planar sets the selected actors' X, Y, or Z translation to the corresponding axis of the target actor.


Start
Get selected level actors
Specified Axis?
X
Y
Z
Set Selected Actor Location X to Target Actor Location X
Set Selected Actor Location Y to Target Actor Location Y
Set Selected Actor Location Z to Target Actor Location Z
Any more actors in selection array?
No
End
Yes
Get next actor in selection array
Make Planar

Select All Same finds all actors in the level of the same class as the selected actor, adding them to the selection.


Start
Get selected level actor
Get All actors of same class as selected actor
Are meshes of selected actor and target actor the same?
Yes
Add target actor to selection array
No
Anymore actors in level of == class?
Yes
Get next actor in with == class
No
Set Selection Array as the Selected Level Actors
End
Select All Same Mesh

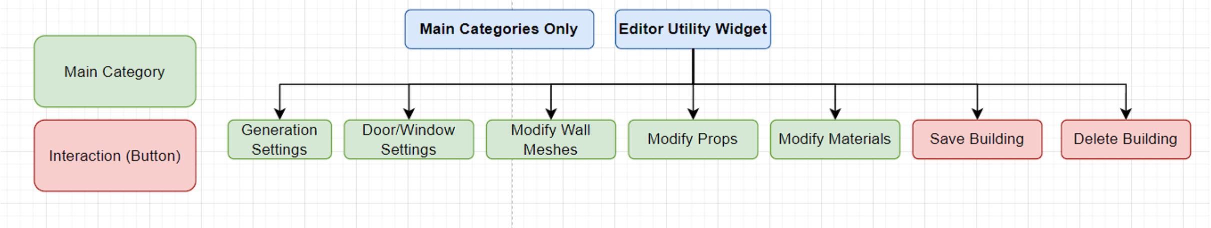
Finally, Rotate Actors gets all selected actors, and based on a specified Axis, rotates them a certain number of degrees.



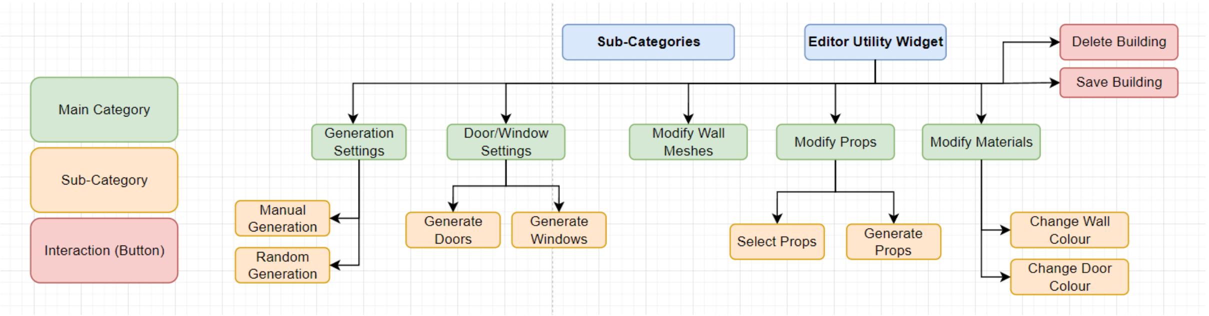
System 2 – Building Generator

IA Diagrams

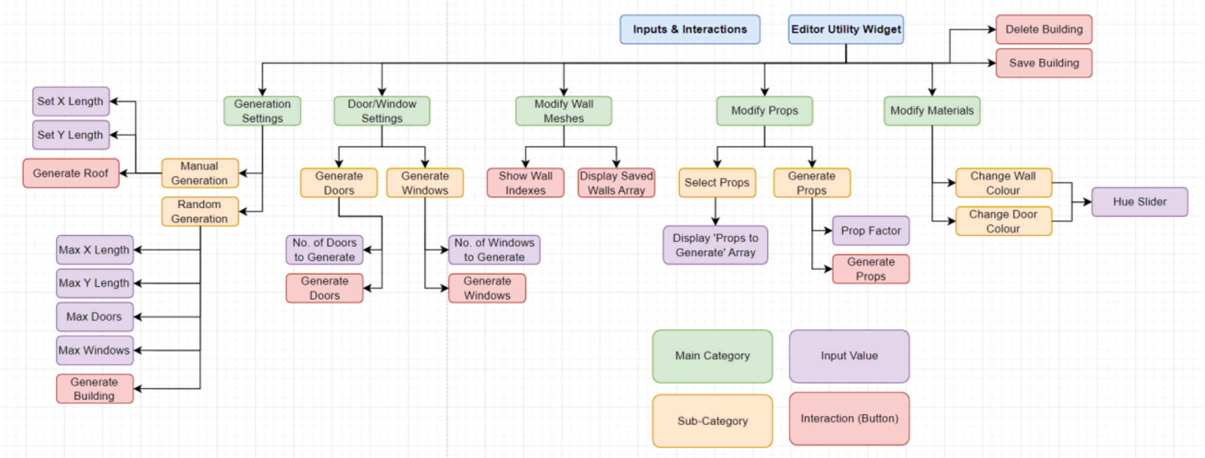
The following diagrams are a set of iterative Architecture plans to plot out the structure of the Widget interface. The first diagram simply displays each main category of features on the widget.



This diagram aims to specify a little more what's in each category by displaying its sub-categories (if it has sub-categories)

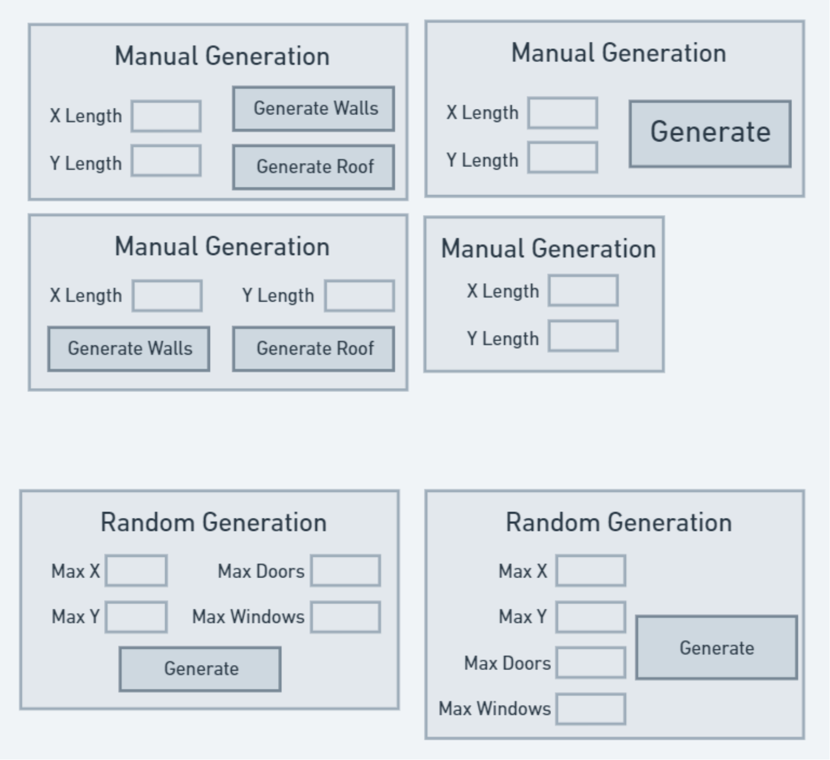


The final IA diagram iteration aims to show each features' inputs (purple) and interactions (red). Inputs are the values the user will have to specify, such as X/Y Length, Number of Doors/Windows, and Prop Factor. Interactions are simply buttons & checkboxes the user presses to perform an action.



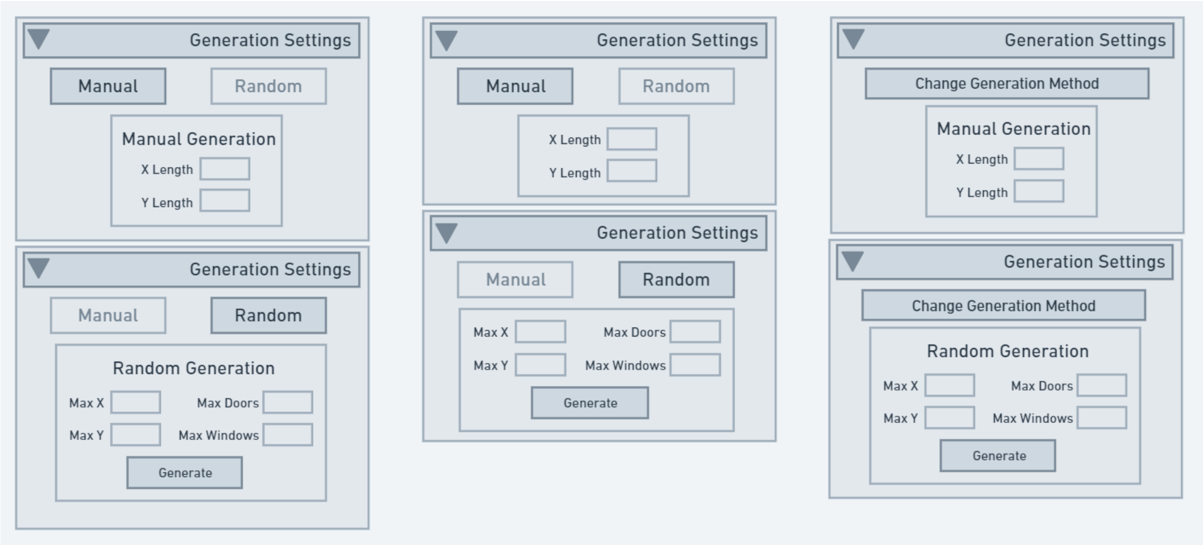
UI Wireframes

The first wireframe experiments with the layout of the Wall Generation setting. Grouping widgets by type (inputs on the left and buttons on the right) provides a better User Experience however considering walls are generated in real-time for Manual Generation, buttons aren't required at all.

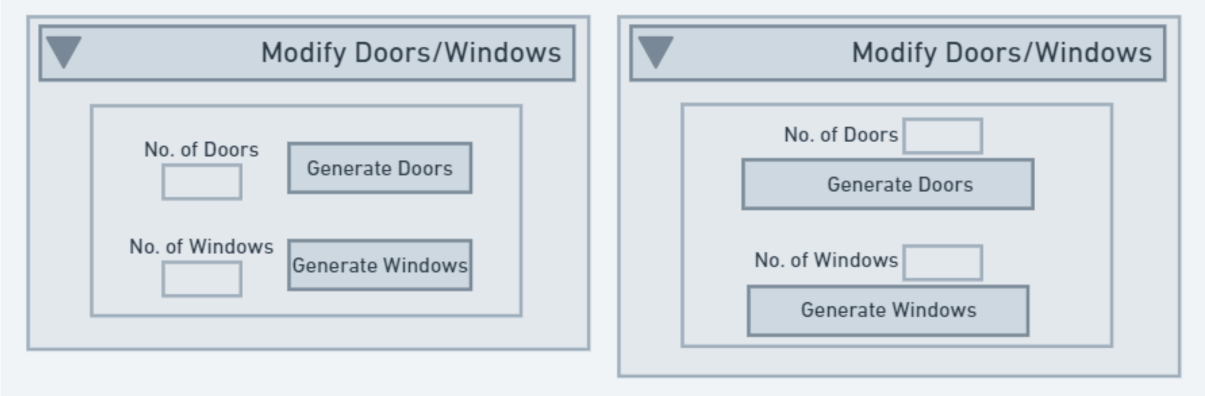


Iterations of the Wall Generation Panel are shown below. The main experimentation was having either two buttons for each method, or one button to switch between them. I chose the latter as this provides a better User Experience. A single, larger button is easier to navigate to and interact with, adhering to Fitt's Law better than the former option of two smaller buttons.

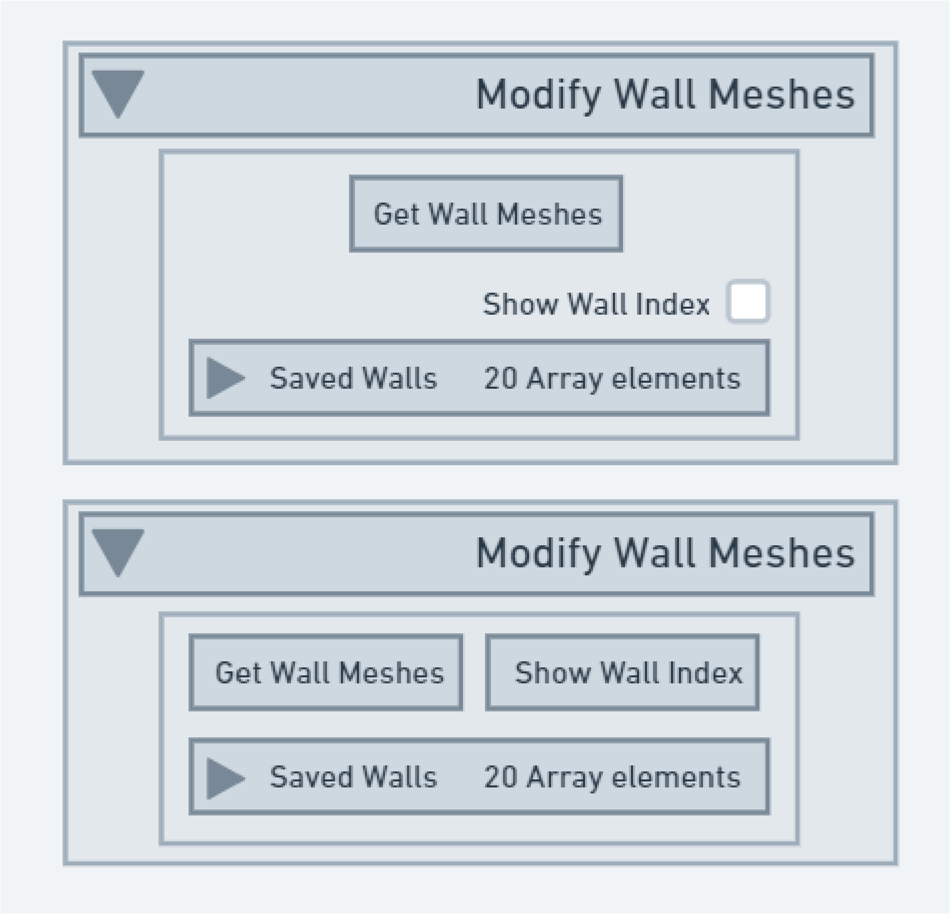
(Fitt's Law = The time required to move to a target is a function of the distance to the target and size of the target.)



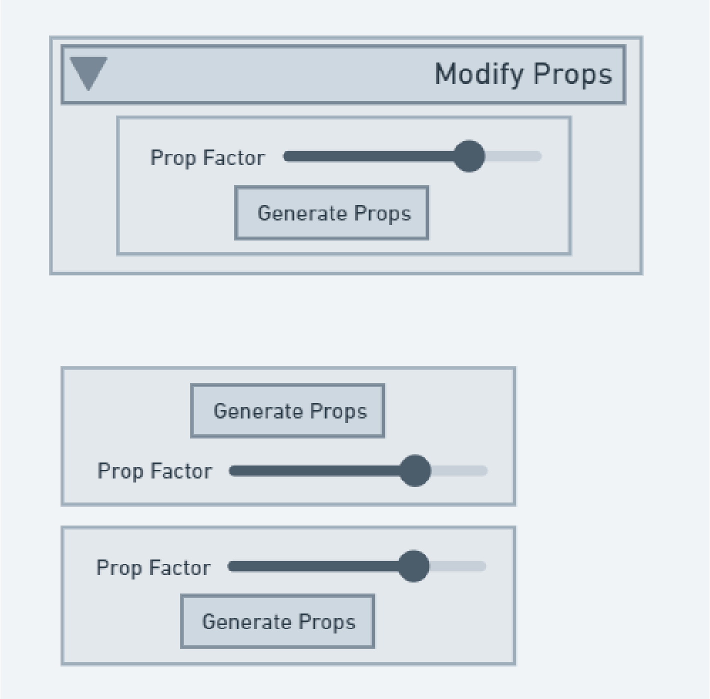
Each subsequent feature required less iteration due to the limited options in the feature. The Doors/Windows Panel simply experimented with layout of the inputs & buttons. I chose the second iteration in the final widget as the buttons are closer to their corresponding input, better grouping the relevant features together.



Modify Wall Meshes makes use of a checkbox instead of a button to break up the visual monotony of the panel, providing the user alternative methods of interacting with the tool rather than just buttons that may be harder to distinguish from each other.



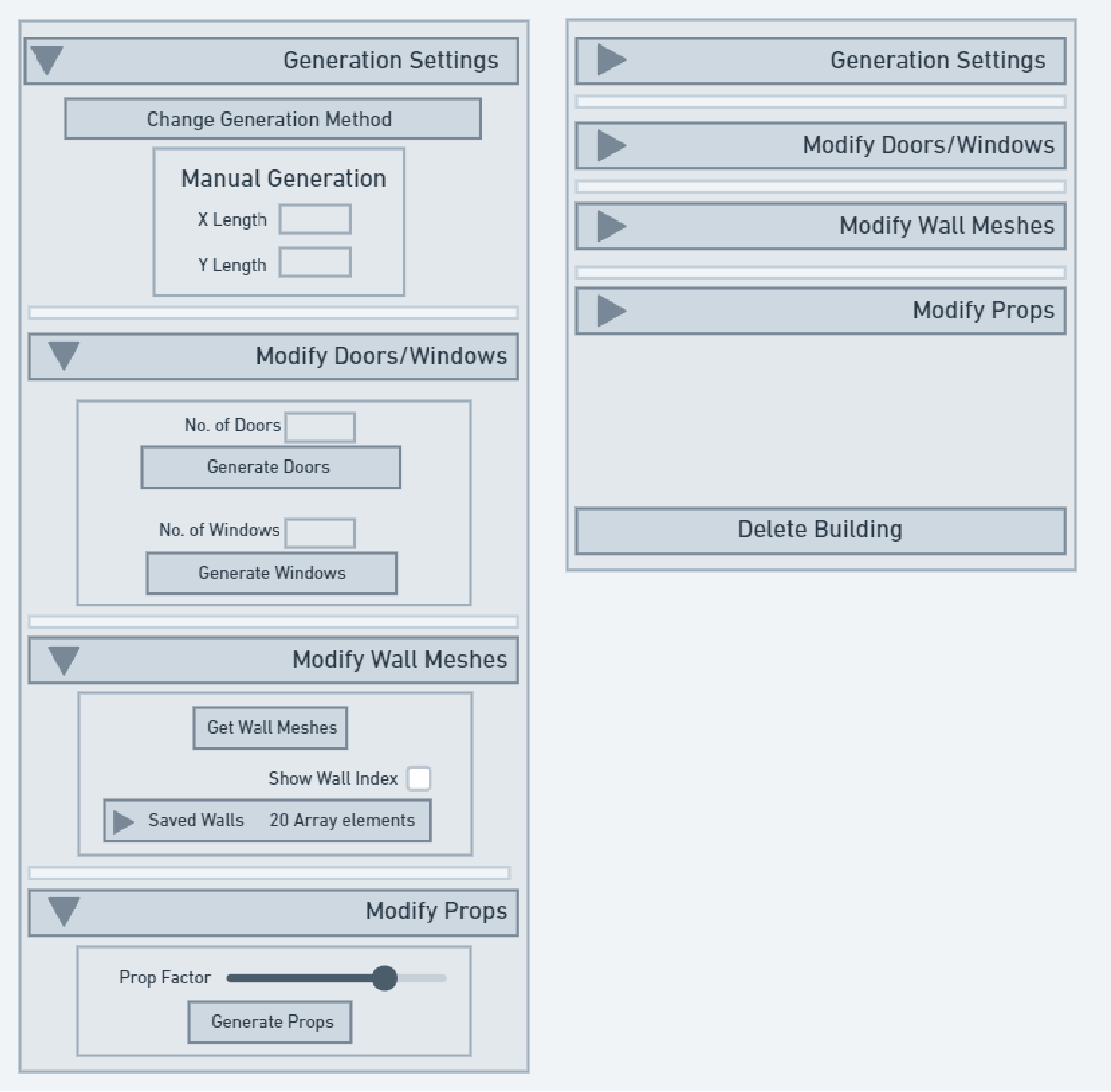
The last feature is Modify Props. Experimenting with the placement of features, I found having the button below the Prop Factor was more suitable. This is because the tool flows in a more natural state as the user will start by modifying the prop factor then clicking the button, resulting in working from top to bottom which provides a more natural and intuitive flow.



The final wireframe compiled all these panels together to make the final widget. Each panel begins collapsed, allowing the user to pick which information they have access to at any point reducing the cognitive overload.

The order of the features also adheres to the concept of hierarchy. Generation Settings is the most fundamental feature that every user will need to modify first. The widget then flows down in a rough assumed order of priority.

Finally, the Delete Building button is located at the bottom of the widget after a sizeable gap. This button will take longer to navigate to and interact with. This is preferred as it reduces the chance that the button is pressed accidentally, and the user has the time to make sure that this is the action they wish to perform.

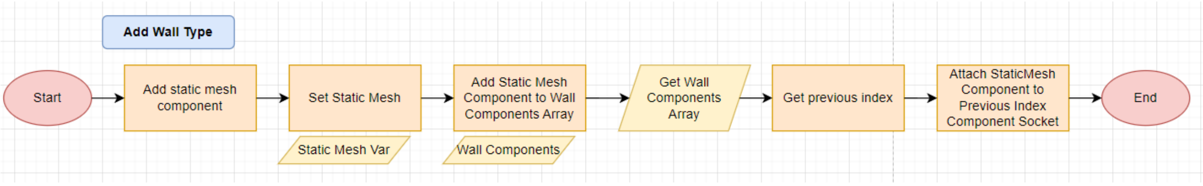


System Flowcharts

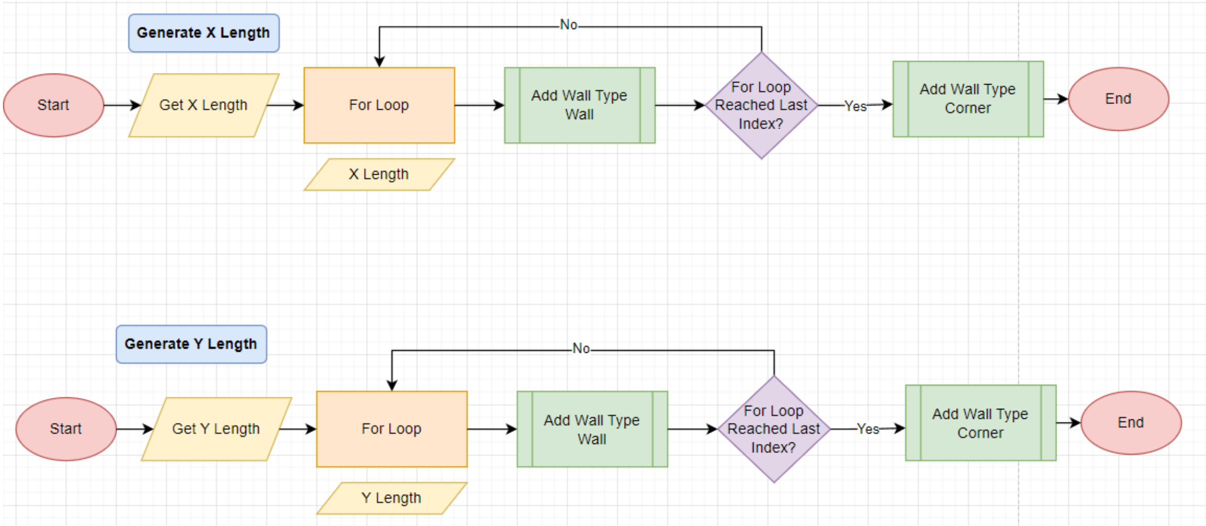
A comprehensive number of flowcharts have been created for the tool to visualise and display the flow of execution for the variety of features in the Building Generator.

Manual Wall Generation

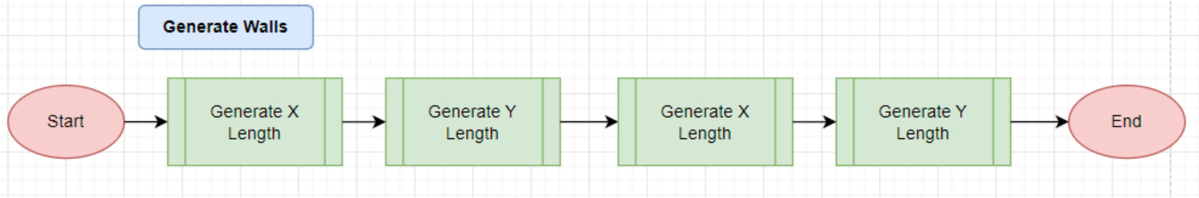
The function to add wall meshes and attach them to the current structure is shown below.


Add Wall Type
Start
Add static mesh component
Set Static Mesh
Static Mesh Var
Add Static Mesh Component to Wall Components Array
Wall Components
Get Wall Components Array
Get previous index
Attach StaticMesh Component to Previous Index Component Socket
End

Generating the X and Y Lengths repeats the Add Wall Type function based on the integer the user inputs.

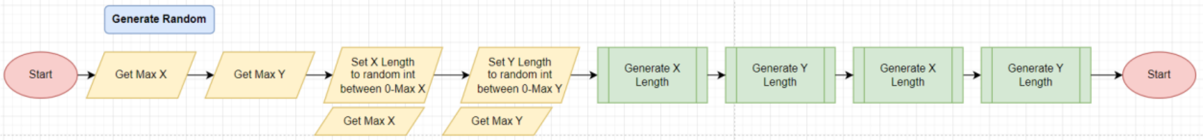


This function is called when the user wishes to generate the building. It calls the Generate X and Y functions which in turn executes AddWallType.



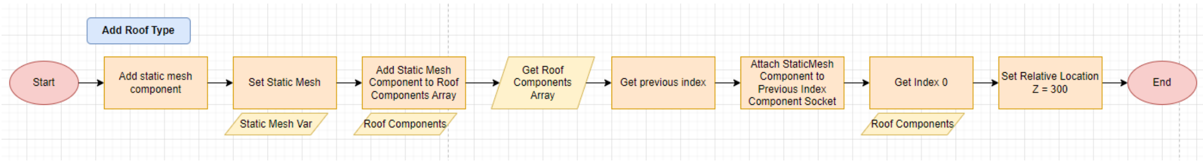
Random Wall Generation

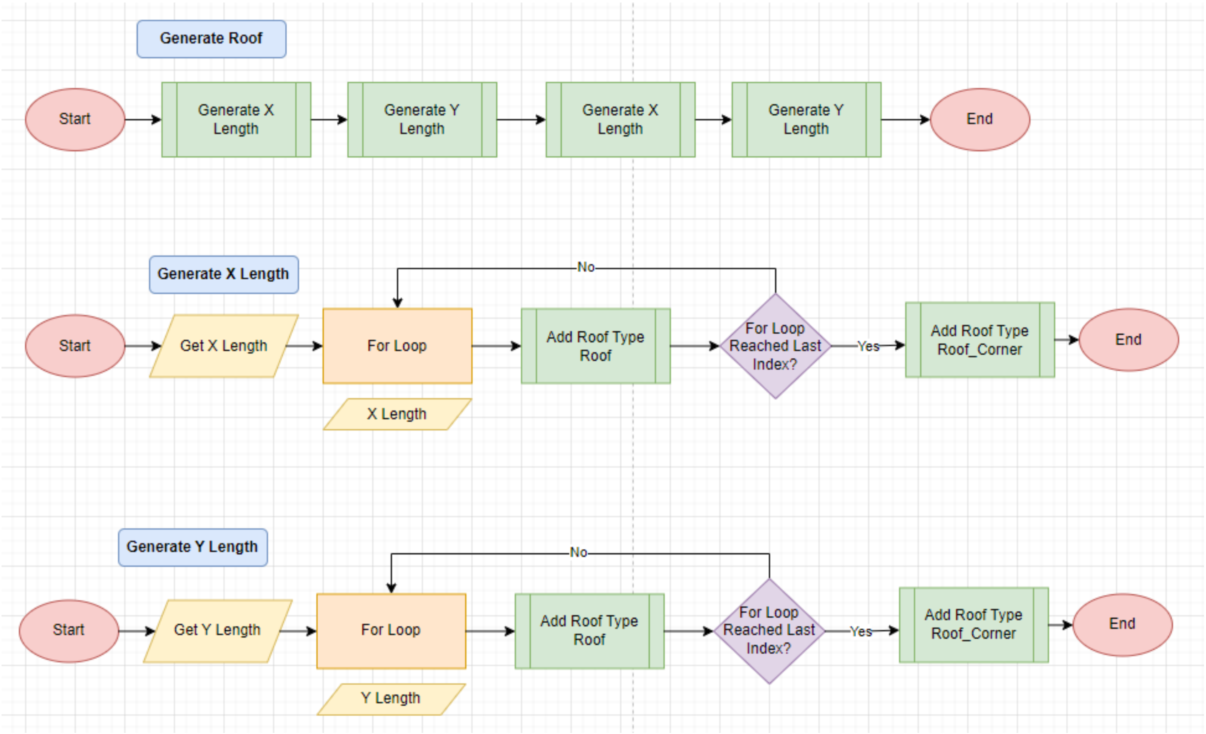
Random Generation utilises the same functions as Manual, except taking Max X and Y lengths from the user input and returning a random integer between 0 and this value.



Roof Generation

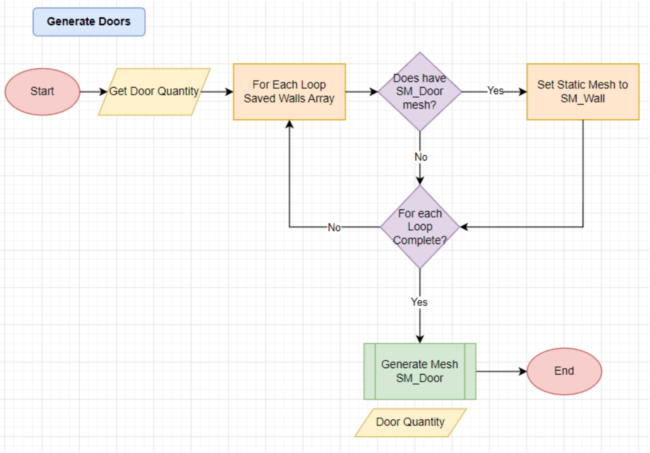
The roof is automatically generated along with the walls, but the functions to generate the roof are much the same logic as Wall Generation.

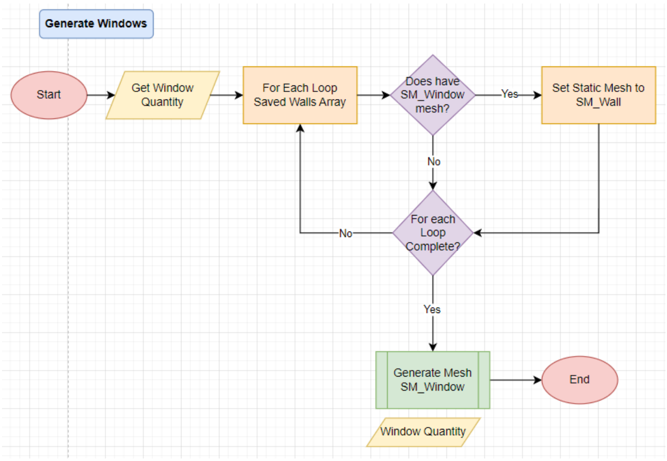

Add Roof Type
Start
Add static mesh component
Set Static Mesh
Static Mesh Var
Add Static Mesh Component to Roof Components Array
Roof Components
Get Roof Components Array
Get previous index
Attach StaticMesh Component to Previous Index Component Socket
Get Index 0
Roof Components
Set Relative Location Z = 300
End



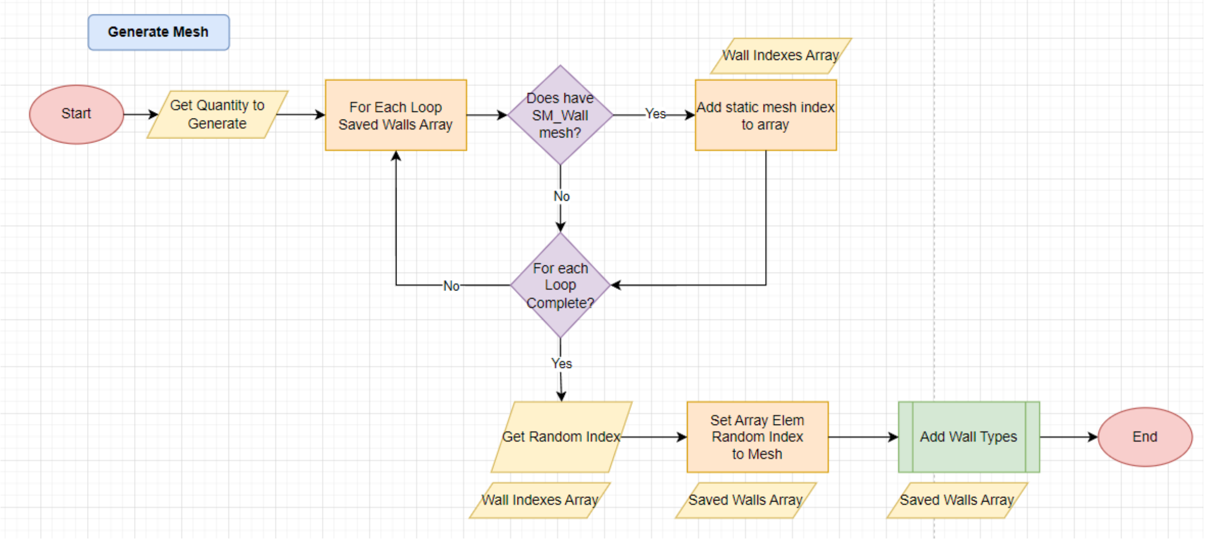
Generating Doors & Windows

These functions reset all wall types back to SM_Wall, allowing new doors & windows to be regenerated using the GenerateMesh function.



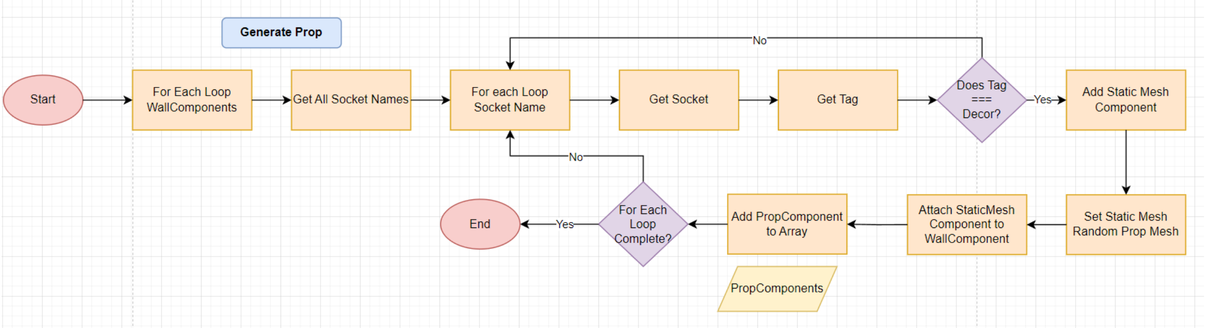


GenerateMesh gets the quantity to generate along with what mesh should be generated. It picks out random wall pieces and sets their mesh to the desired door/window mesh.



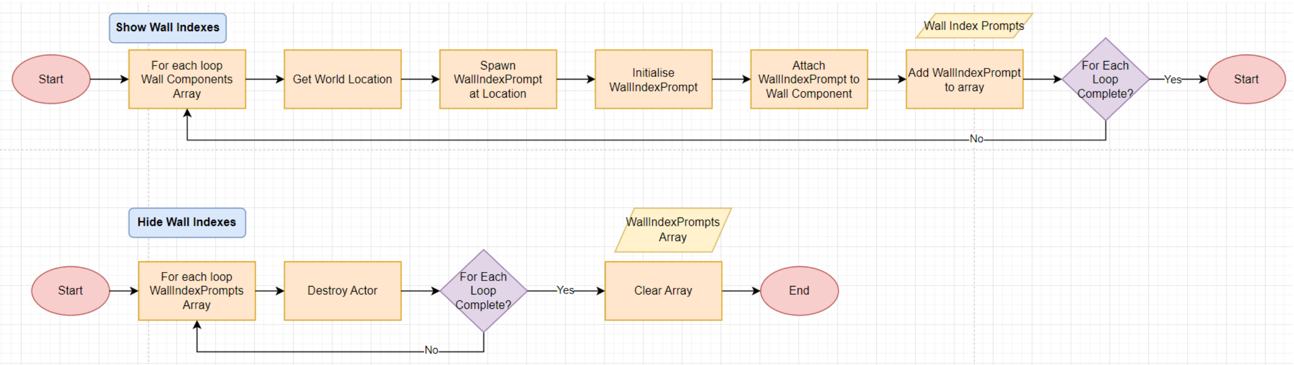
Prop Generation

Prop Generation gets the Decor sockets of each wall piece, adding a mesh at that location and attaching it to the corresponding wall component.


Generate Prop
Start
For Each Loop WallComponents
Get All Socket Names
For each Loop Socket Name
Get Socket
Get Tag
Does Tag === Decor?
No
Yes
Add Static Mesh Component
Set Static Mesh Random Prop Mesh
Attach StaticMesh Component to WallComponent
Add PropComponent to Array
PropComponents
For Each Loop Complete?
No
Yes
End

Display Wall Indexes

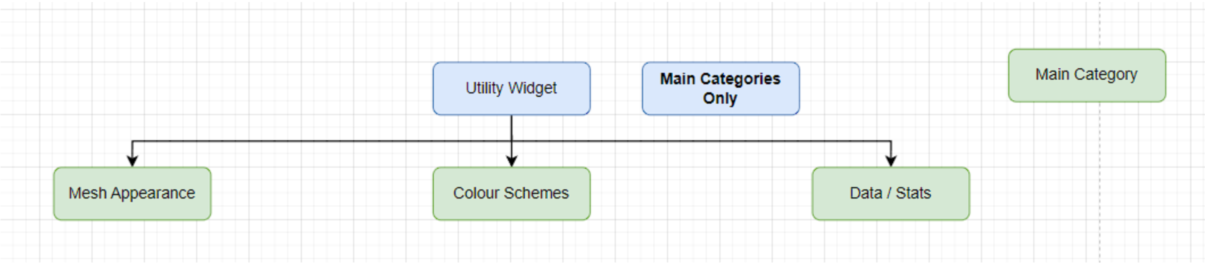
This function allows the user to identify the index of a wall piece. It simply spawns an actor at the location of each wall piece in the WallComponents array. Hiding them destroys all the index prompt actors.


Show Wall Indexes
Start
For each loop Wall Components Array
Get World Location
Spawn WallIndexPrompt at Location
Initialise WallIndexPrompt
Attach WallIndexPrompt to Wall Component
Wall Index Prompts
Add WallIndexPrompt to array
For Each Loop Complete?
Yes
Start
No
Hide Wall Indexes
Start
For each loop WallIndexPrompts Array
Destroy Actor
For Each Loop Complete?
Yes
WallIndexPrompts Array
Clear Array
End
No

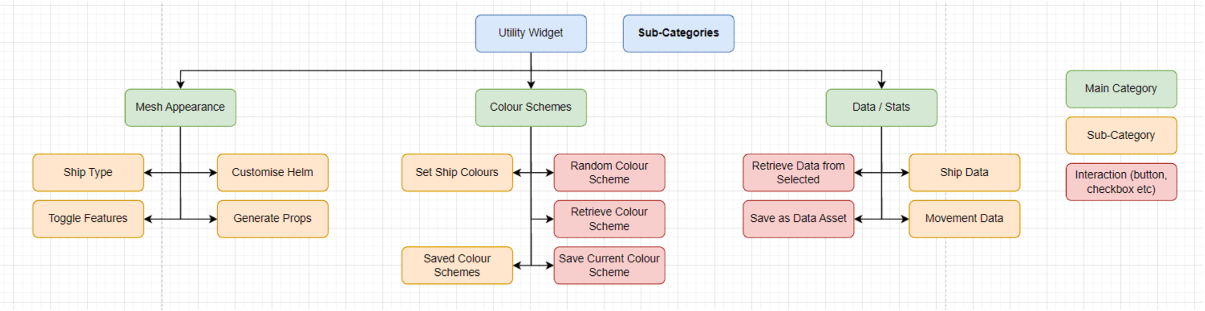
System 3 – Ship Configurator

IA Diagrams

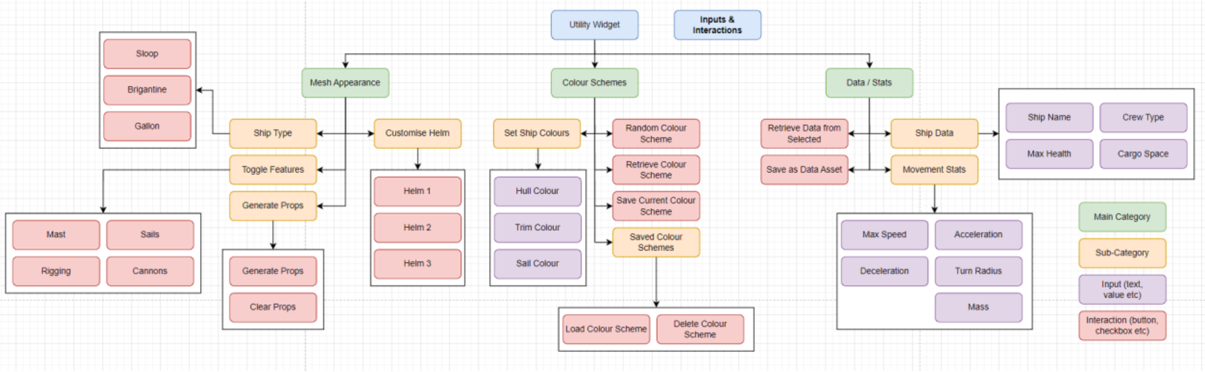
The first Architecture simply outlines the main areas of customisation that can be accessed from the Editor Utility Widget.



These categories are then expanded on in the second IA diagram going into more detail regarding the specific features that can be found under each main category.



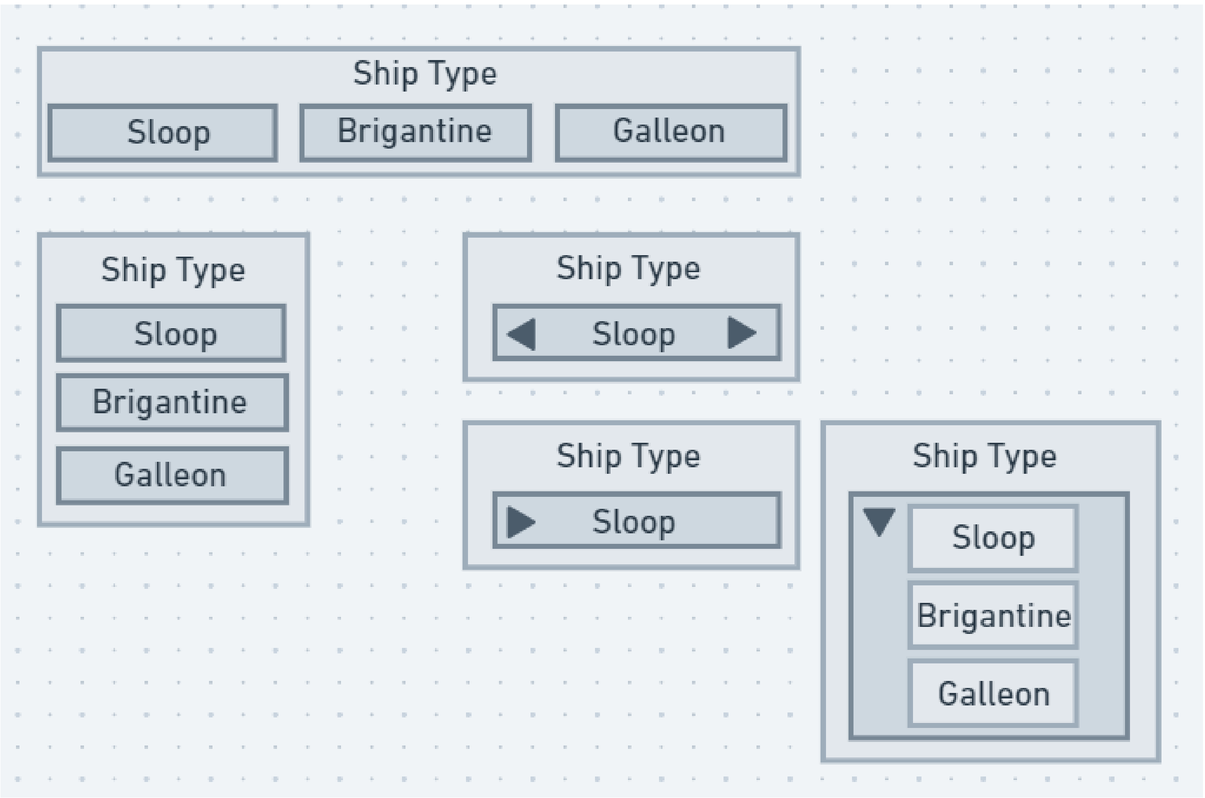
The final diagram breaks down every feature to its smallest state, aiming to depict what specific aspects and details of the ship can be modified in each category and sub-category.



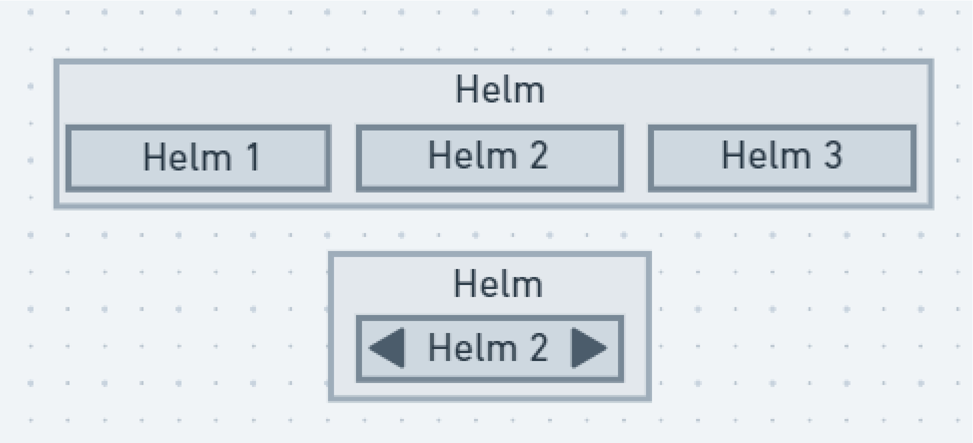
UI Wireframes

The first set of wireframes experiments with the layout and structure of the Mesh Appearance panel.

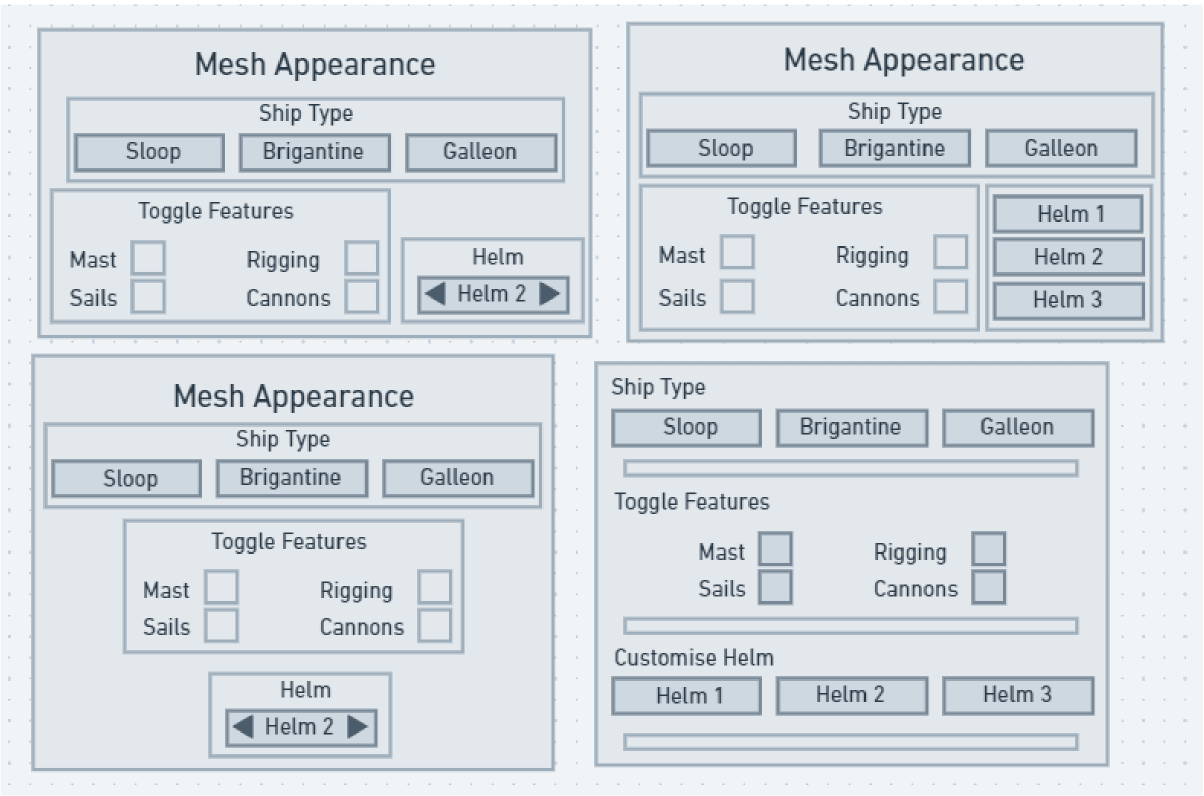
Several layouts for Select Ship Type buttons were wireframed. I chose the horizontal layout as this enforces the principle of Hierarchy. This feature is placed at the top of the mesh appearance panel, spanning the entire width of the panel taking up the most space. As this is the most fundamental feature of the configurator, this should be regarded as the most important.



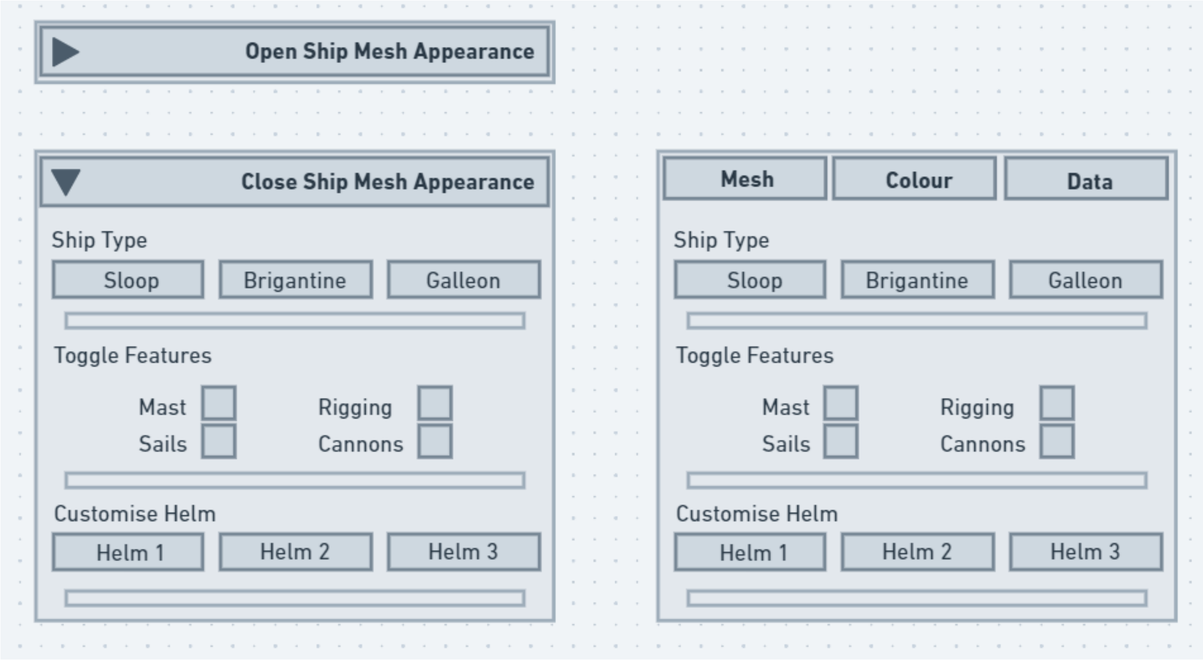
Selecting a helm uses either a button or scroll box. Scroll boxes wouldn’t be suitable if a larger number of helms are available in the tool and buttons give the user more control of which helm they want to select. Because of this, buttons were chosen for the design.



These wireframes experiment with the structure of the entire Mesh Appearance panel. I selected the bottom right option. It's less crowded and cluttered than the first two options and the line breaks between features naturally groups together each feature without having to contain them in a visible border.

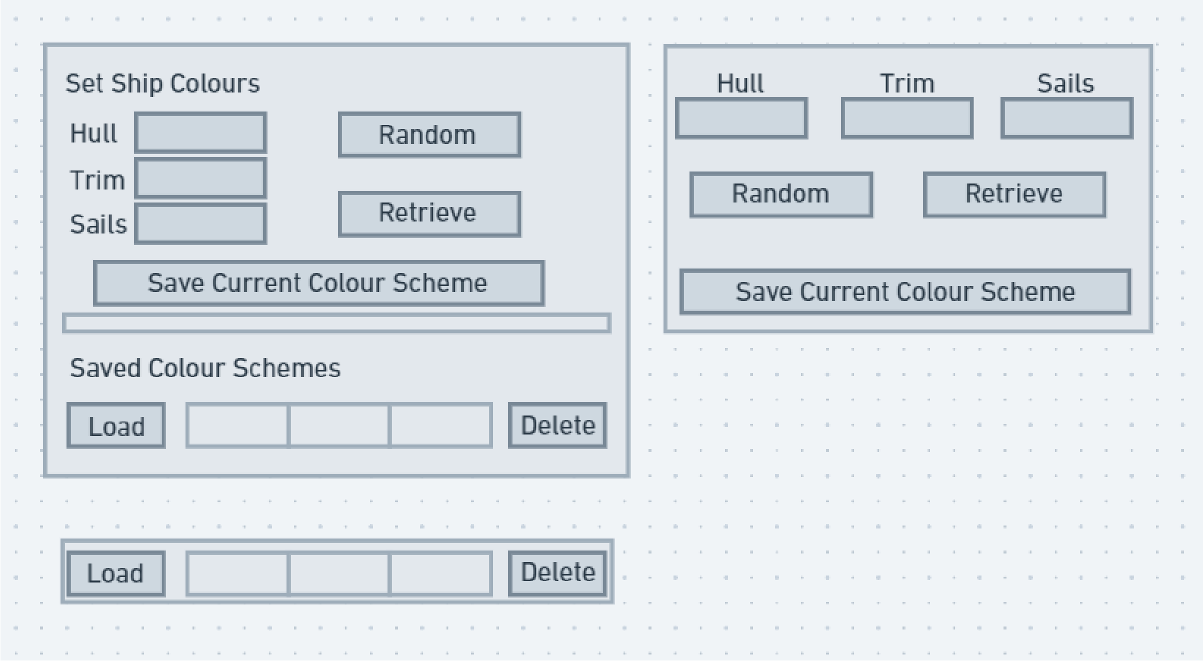


For this tool I decided to use a widget switcher to display each panel. The Widget Switcher hides panels the user no longer needs, and the buttons at the top act as headings for each panel making it easier for the user to know which panel is selected. The widget switcher also reduces the entire vertical space required for the widget, meaning the user won't have to spend time scrolling up/down to find a certain feature.

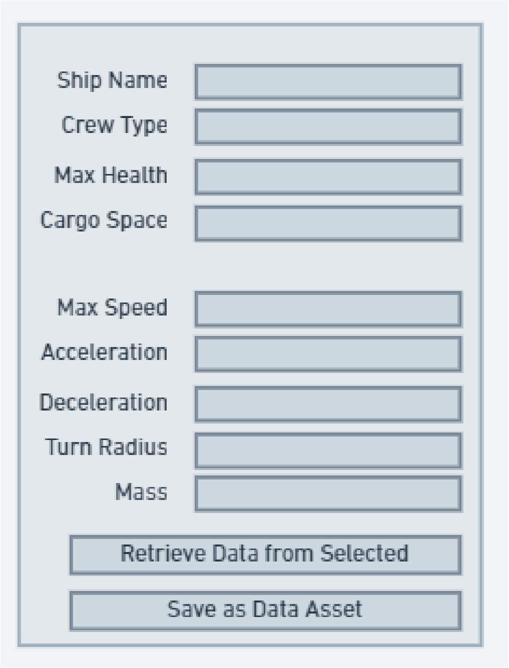


Within the colour panel, saved colour schemes are contained in a horizontal box with the load and delete buttons on either side, spacing them far apart to reduce the likelihood of any button being pressed accidentally.

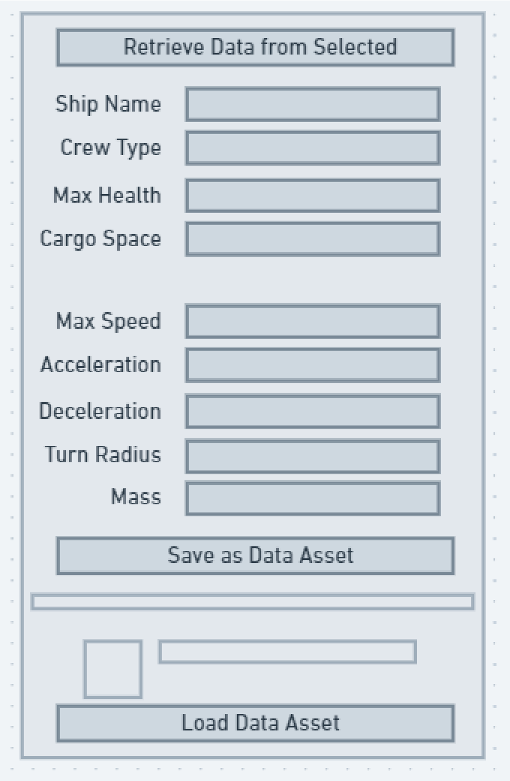
The first Wireframe uses grouping as each colour selection box is closer to each other when compared to other features. The Random and Retrieve buttons have a larger amount of spacing between them to reduce the likelihood of accidental presses.



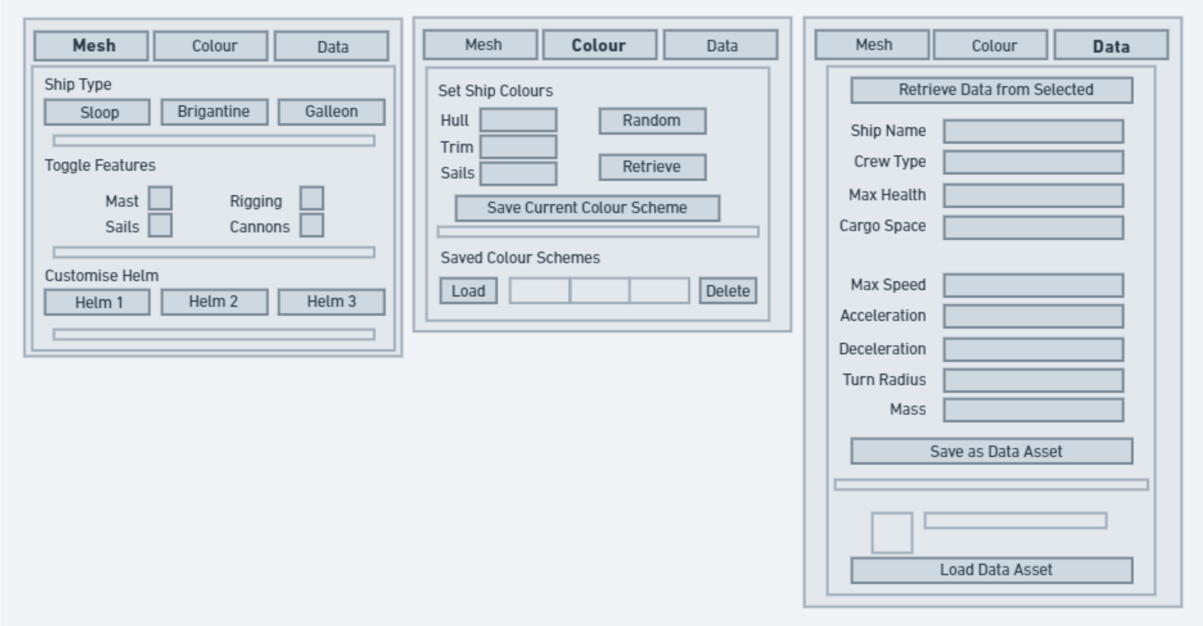
The data panel is a vertical list of the customisable options for the ship. There's spacing to separate the two groups of settings, the top being general ship settings and the bottom containing the more specific data that affects control and movement. Without needing any dividers or headings, the user knows these settings are in two groups.



This wireframe was changed slightly to accommodate the Data Asset feature. Retrieve Data from Selected has been moved to the top to allow the user to more quickly retrieve the selected data. The new Load feature has been added to the bottom and separated by a line break to more clearly indicate this is a separate feature from the regular setting & saving of data.



Each panel is then contained in a Widget Switcher with buttons to determine which panel is selected, doubling up as the panel heading by increasing the size and boldness of the text when selected.

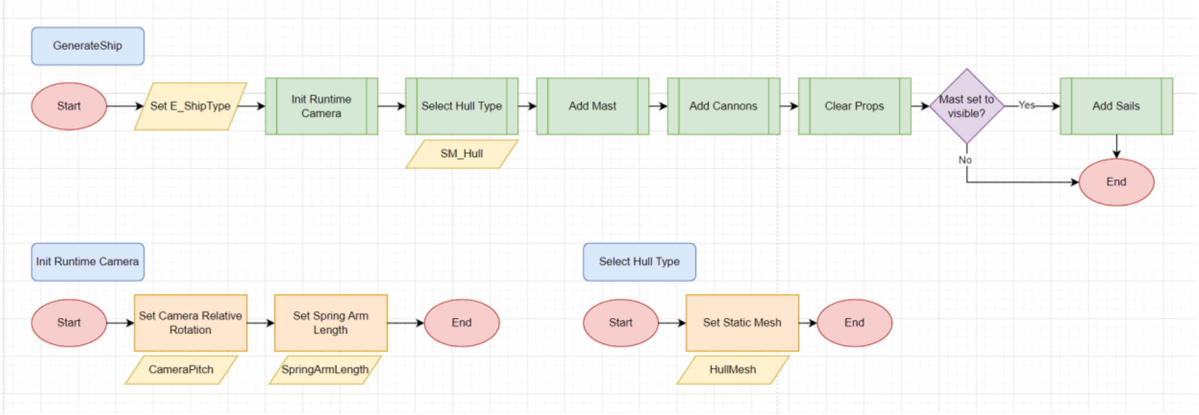


System Flowcharts

Several flowcharts have been created to show the flow of execution and logic within each feature.

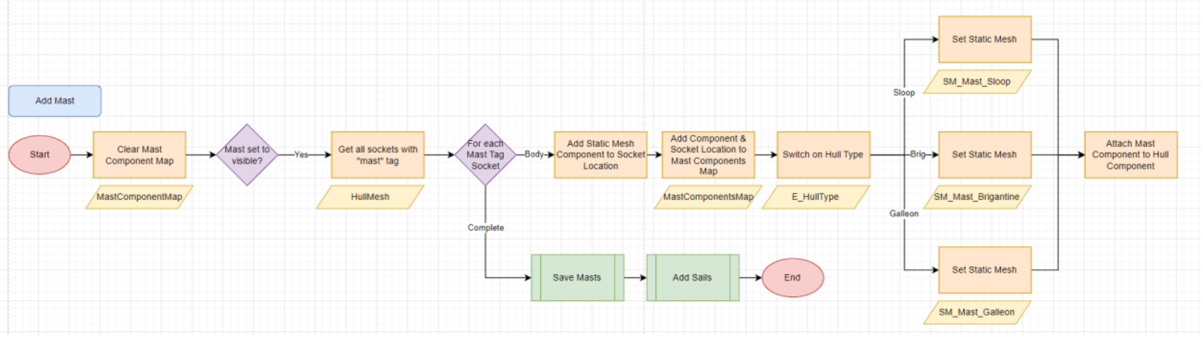
Generate Ship / Runtime Camera

This flowchart is the main initialisation chain of the ship. It generates each feature along with Initialising the runtime camera with a suitable spring arm length and camera angle for the ship.



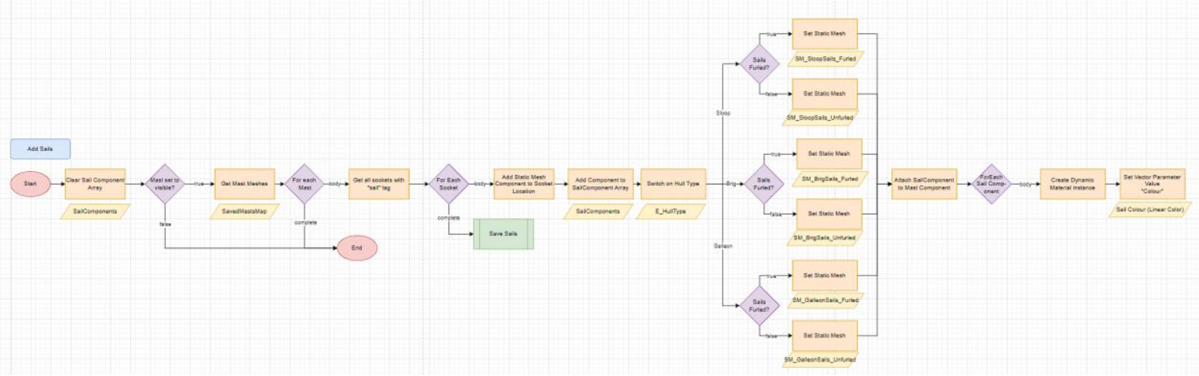
Add Mast

This function creates a static mesh component at the location of the mast socket on the hull. The static mesh set is on a switch of the enum ShipType.



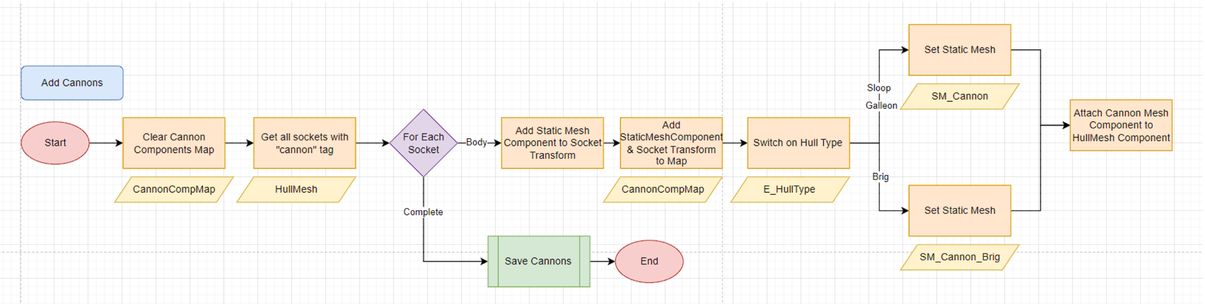
Add Sails

This is a similar function with a couple of extra checks. If the masts are visible, the sail sockets are returned, adding a component and mesh. Different meshes will be set depending on the checked state of the sails.



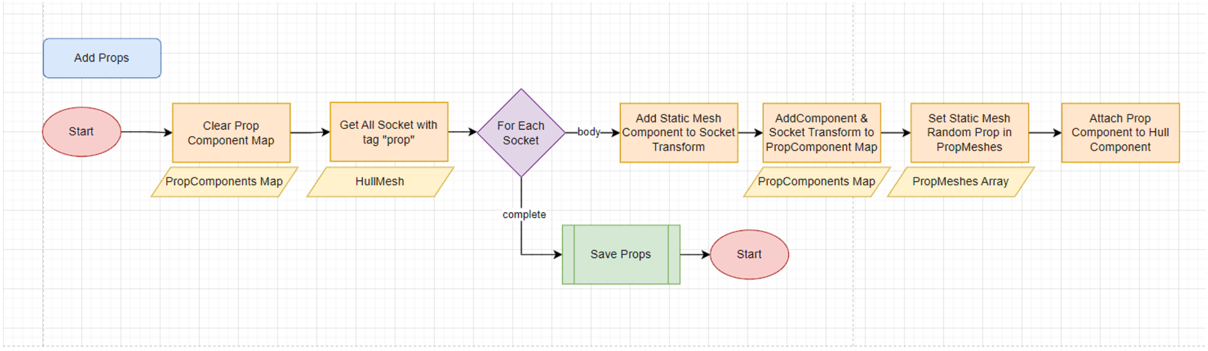
Add Cannons

A very similar structure of the previous addition of components. The Brigantine Ship type has different cannon meshes to the other ship types, so a switch is needed in this function.


Add Cannons
Start
Clear Cannon Components Map
CannonCompMap
Get all sockets with "cannon" tag
HullMesh
For Each Socket
Body
Complete
Add Static Mesh Component to Socket Transform
Add StaticMeshComponent & Socket Transform to Map
CannonCompMap
Switch on Hull Type
E_HullType
Sloop
Galleon
Brig
Set Static Mesh
SM_Cannon
Set Static Mesh
SM_Cannon_Brig
Attach Cannon Mesh Component to HullMesh Component
Save Cannons
End

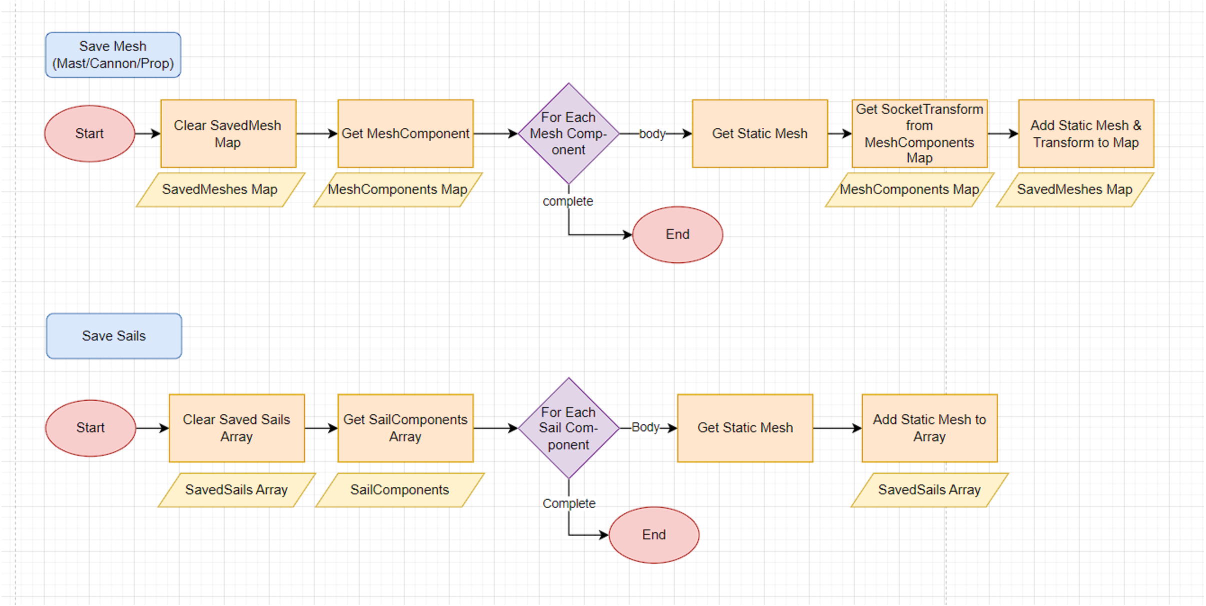
Add Props

Adding props use the same technique. This time, the mesh is selected as a random index from an array of possible prop meshes.


Add Props
Start
Clear Prop Component Map
PropComponents Map
Get All Socket with tag "prop"
HullMesh
For Each Socket
body
Add Static Mesh Component to Socket Transform
AddComponent & Socket Transform to PropComponent Map
PropComponents Map
Set Static Mesh Random Prop in PropMeshes
PropMeshes Array
Attach Prop Component to Hull Component
complete
Save Props
Start

Saving Meshes

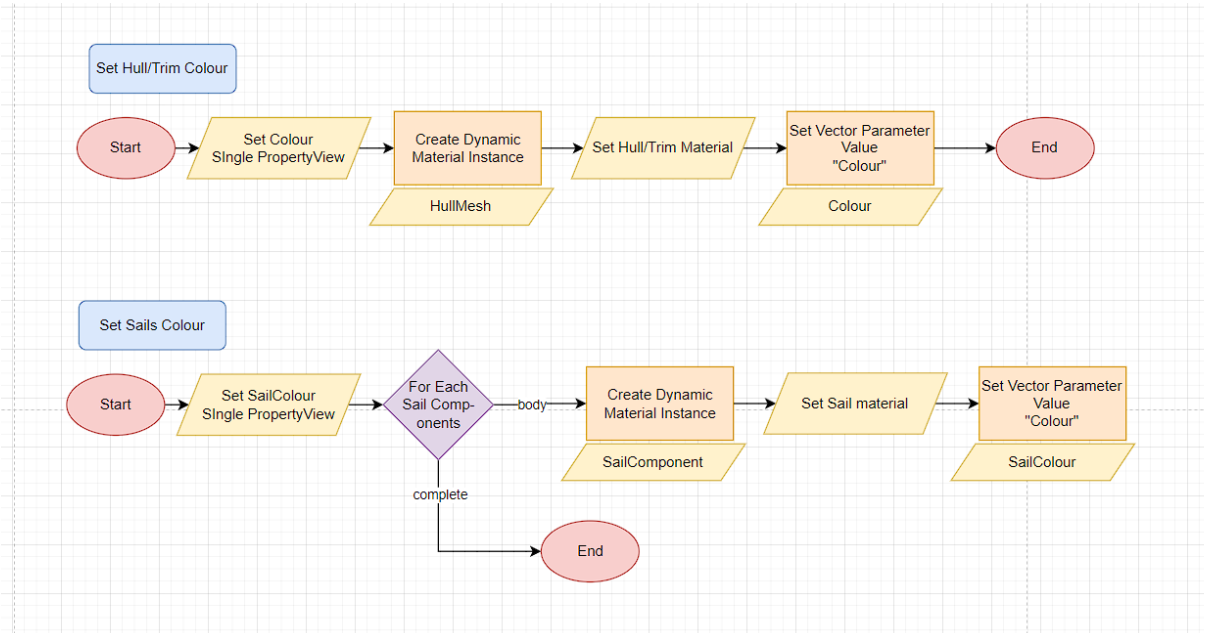
Each mesh needs to be saved in order to be generated properly on the construction script. Where sockets are used to place static meshes, the transform of the socket is mapped against the static mesh and saved in a Map Variable to be used in the construction script.



In the construction script, very similar logic is used when generating assets. It simply creates a static mesh component, using the transform data in the Map Variable for the transform of the component. The Static Mesh to be set is retrieved from the Map.

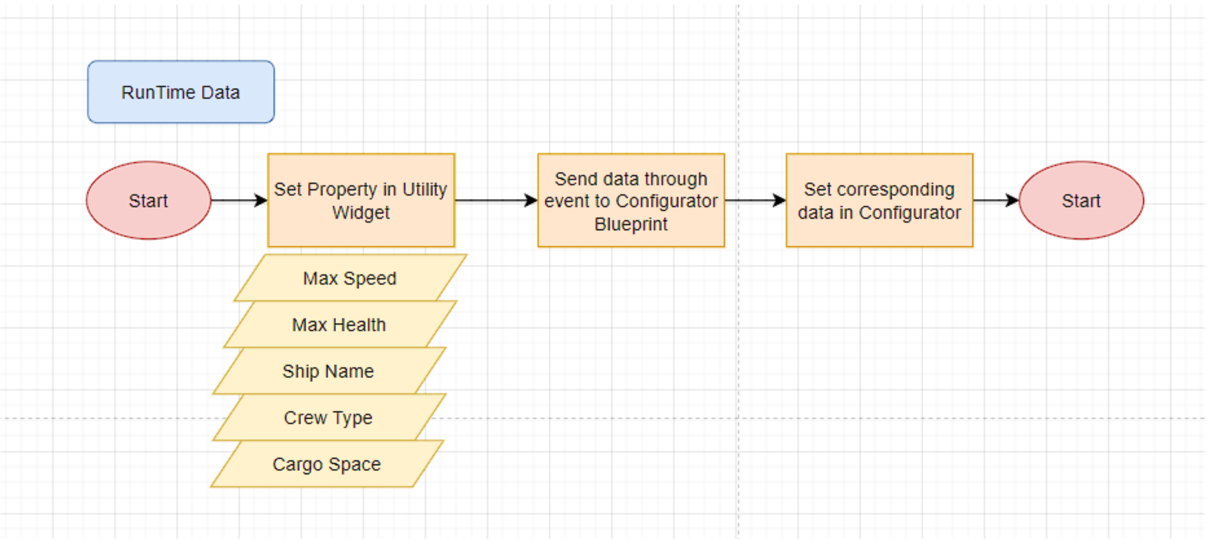
Set Colours

Each Linear Colour variable is contained in a Single Property View. Dynamic Material Instances are created for each mesh that has a customisable colour, setting their Colour parameter as the Linear Colour property.


Set Hull/Trim Colour
Start
Set Colour SIngle PropertyView
Create Dynamic Material Instance
HullMesh
Set Hull/Trim Material
Set Vector Parameter Value "Colour"
Colour
End
Set Sails Colour
Start
Set SailColour SIngle PropertyView
For Each Sail Comp- onents
body
complete
Create Dynamic Material Instance
SailComponent
Set Sail material
Set Vector Parameter Value "Colour"
SailColour
End

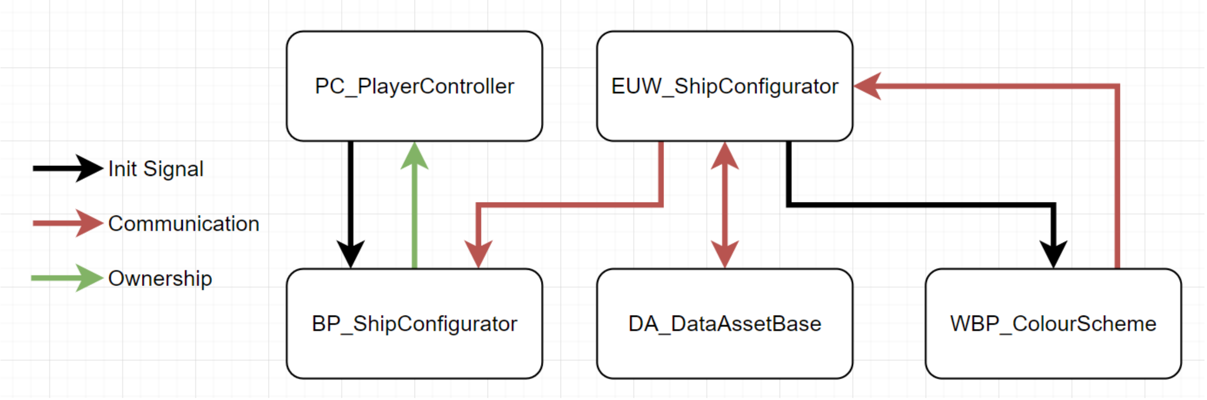
Runtime Data

Each piece of data is contained in a Single Property View editable on the Utility Widget. Editing this property sets the variable, which is sent through an event to the configurator, where the corresponding variables are set.


RunTime Data
Start
Set Property in Utility Widget
Max Speed
Max Health
Ship Name
Crew Type
Cargo Space
Send data through event to Configurator Blueprint
Set corresponding data in Configurator
Start

Communication Diagram

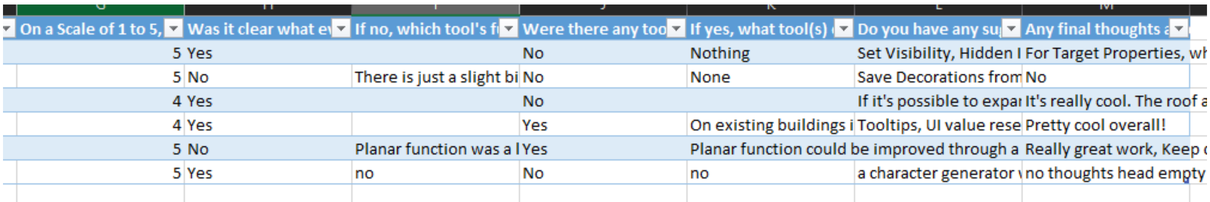
The Ship Configurator requires a couple of additional blueprints to initialise and communicate between each other. The Editor Utility Widget communicates to the Ship Configurator Pawn as well as back and forth to the Data Asset storing its data. It initialises the widget storing each colour scheme which communicates back to the Utility widget when loading and deleting colour schemes.



Optimisation and Profiling

Profiling Systems

Functionality has been thoroughly tested throughout the project using Unreal Engine’s built-in debugging tools such as breakpoints and print strings to ensure the successful execution of logic. User testing has also been carried out to test the usability and experience of the tool as well as responding to feedback by improving and amending the tool where necessary.

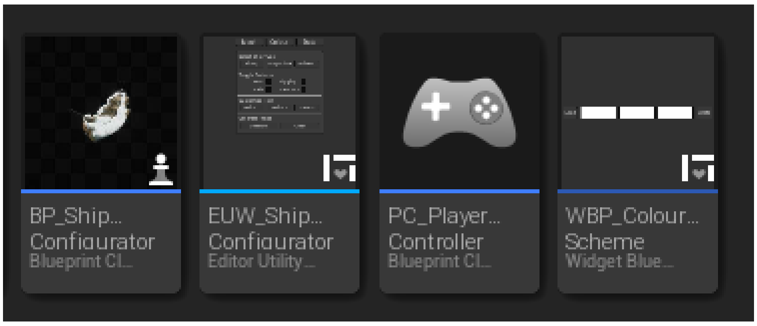


On a Scale of 1 to 5,	Was it clear what ev	If no, which tool's f	Were there any too	If yes, what tool(s)	Do you have any su	Any final thoughts a
5	Yes		No	Nothing	Set Visibility, Hidden I	For Target Properties, wh
5	No	There is just a slight bi	No	None	Save Decorations from	No
4	Yes		No		If it's possible to expar	It's really cool. The roof a
4	Yes		Yes	On existing buildings i	Tooltips, UI value rese	Pretty cool overall!
5	No	Planar function was a l	Yes	Planar function could be improved through a		Really great work, Keep c
5	Yes	no	No	no	a character generator v	no thoughts head empty

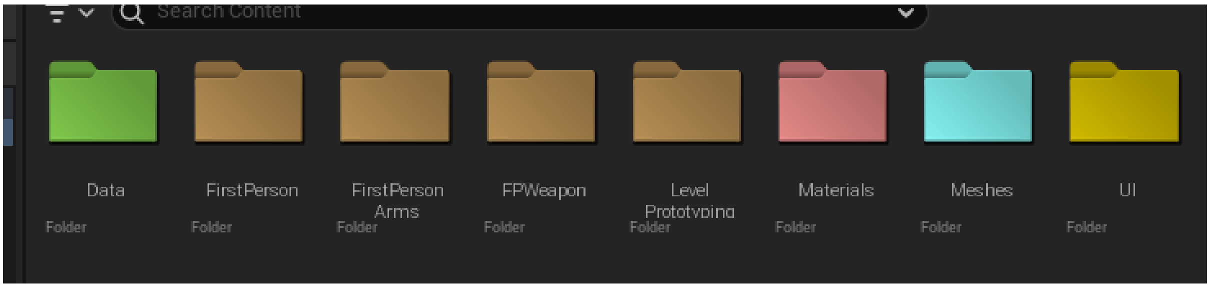
Coding Standards

Programming Standards

Suitable coding standards have been adhered to during development. Appropriate naming conventions and prefixes have been used to ensure readability and understanding for other developers viewing the project.

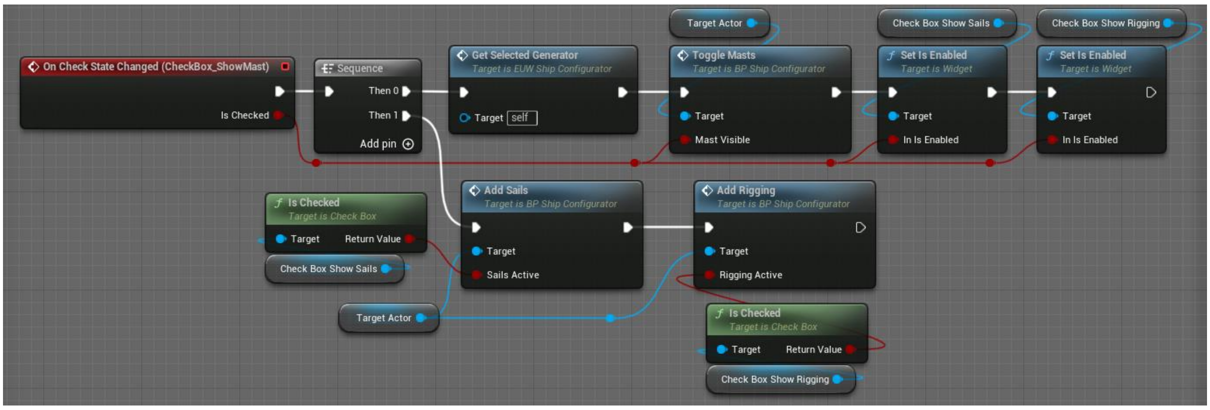

BP_Ship... Configurator Blueprint Cl...
EUW_Ship... Configurator Editor Utility...
PC_Player... Controller Blueprint Cl...
WBP_Colour... Scheme Widget Blue...

The project has been organised into suitably named folders to group relevant blueprints and allow blueprints to be found and stored more easily.

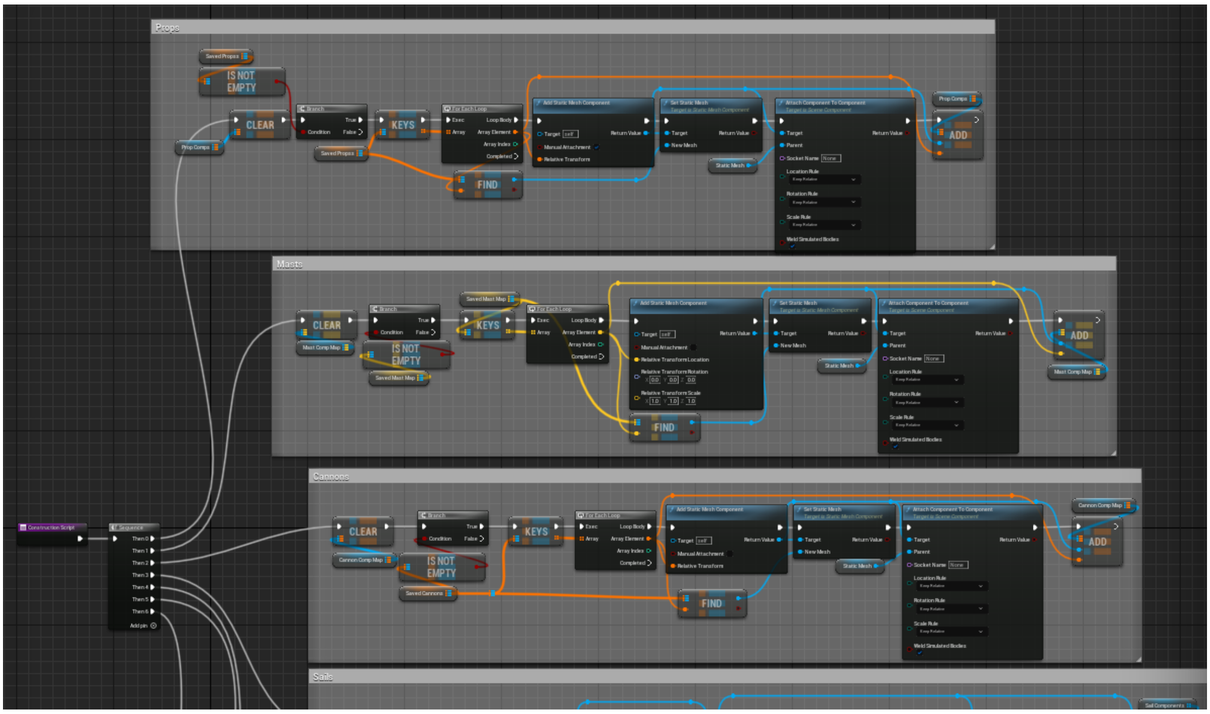


Style Guide

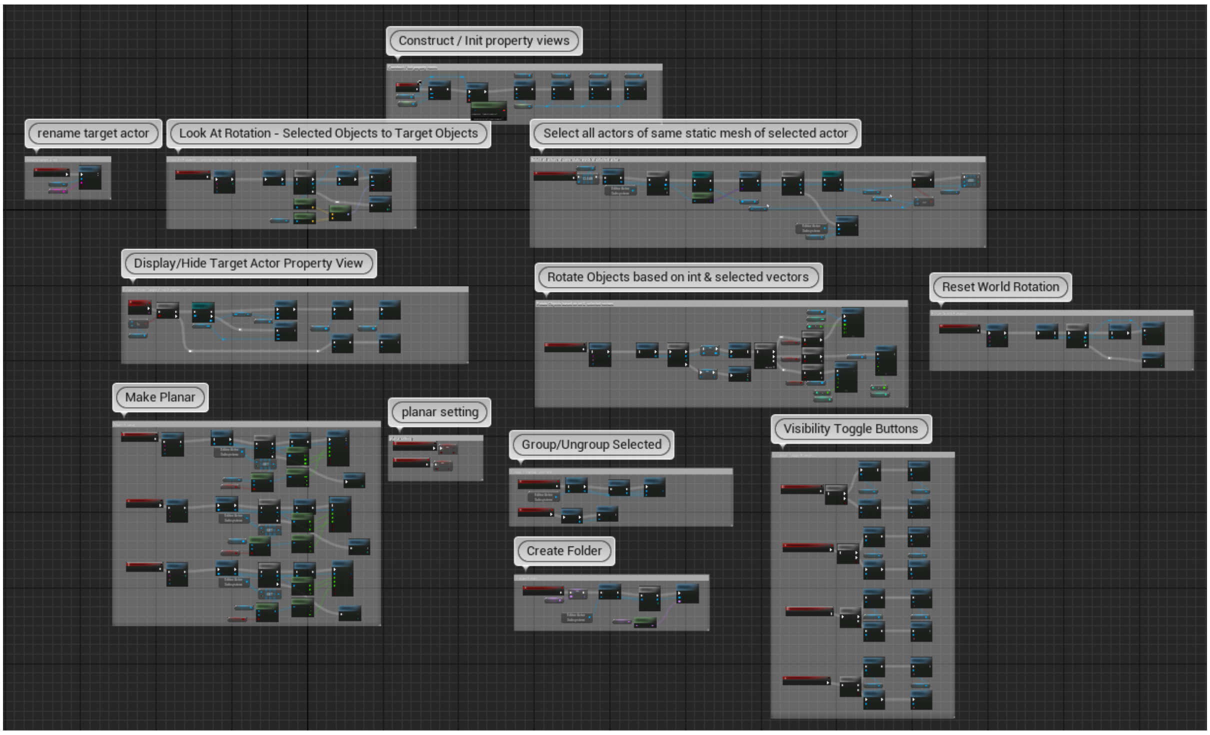
Where possible, script has been neatly organised within event graphs using reroute nodes and alignment tools to ensure script is readable.



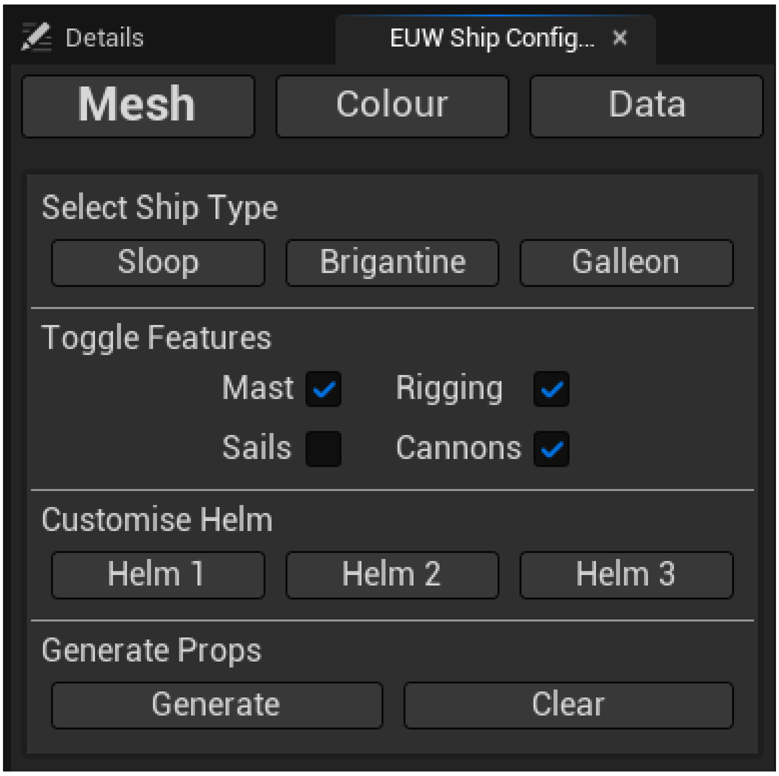
Script has been commented and organised within the event graph to explain the functionality of the script to anyone viewing the project.



Comment Bubbles allow the viewer to very quickly navigate to any piece of script by displaying the comment text when zoomed out to see the entire event graph.


Construct / Init property views
rename target actor
Look At Rotation - Selected Objects to Target Objects
Select all actors of same static mesh of selected actor
Display/Hide Target Actor Property View
Rotate Objects based on int & selected vectors
Reset World Rotation
Make Planar
planar setting
Group/Ungroup Selected
Visibility Toggle Buttons
Create Folder

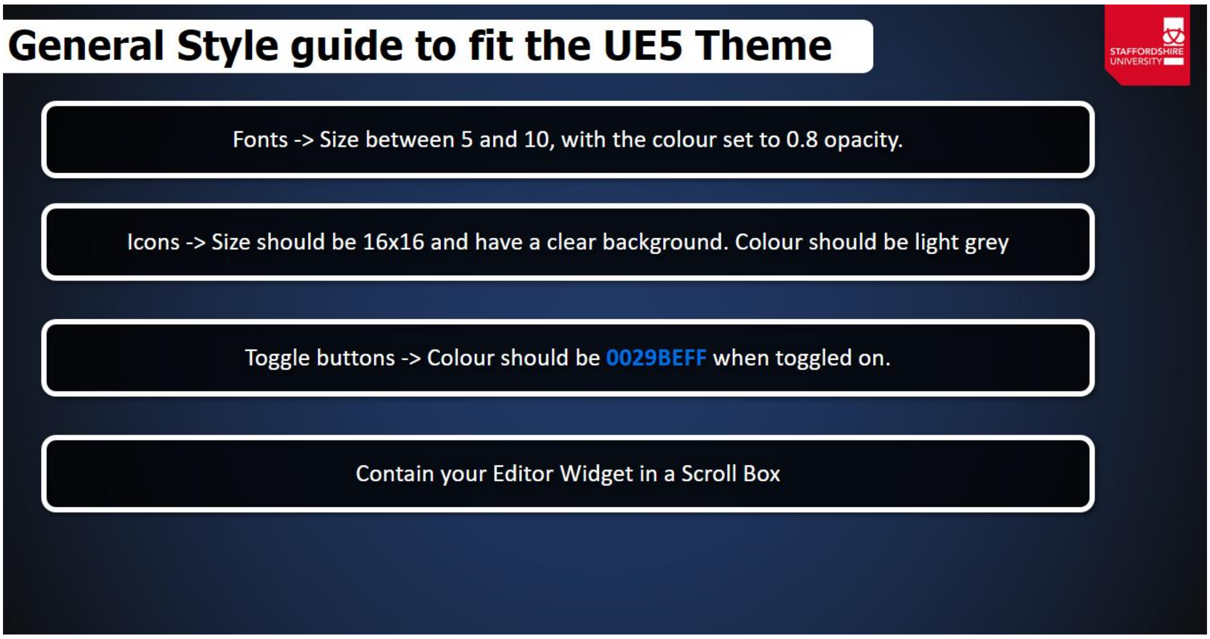
The use of an Editor Utility Widget ensures that the default styling of the tool – for widgets such as buttons and checkboxes - is consistent with Unreal Engine's theme and User Interface.



Mast ☑ Rigging ☑

Sails ☐ Cannons ☑

A style guide for other widget elements has also been adhered to, to ensure further consistency to the Engine style.



General Style guide to fit the UE5 Theme

Fonts -> Size between 5 and 10, with the colour set to 0.8 opacity.

Icons -> Size should be 16x16 and have a clear background. Colour should be light grey

Toggle buttons -> Colour should be 0029BEFF when toggled on.

Contain your Editor Widget in a Scroll Box

Production Overview

Moscow

Tool 1 – Level Kit

Must	Should	Could	Won't
Rename Objects in the outliner	Manipulate actor transform (location, rotation, scale)	Add to/Remove from Groups	Manipulate Materials/Textures
Sort Objects into Folders within the outliner	Select a Target Actor to show its properties	Planar Settings to match X, Y, Z axis of selected actors to target actor	Any save options including save the level or saving data into assets etc
	Utility Widget to interact with rather than using Details Panel/Construction Script	Select all actors in level with same mesh as target actor	

Tool 2 – Building Generator

Must	Should	Could	Won't
Generate walls dynamically	Roof Generation	Manual & Random options for each individual feature	Generate additional floors
Placement of doors & windows	Saving the building to a static mesh	Saving the Level to a new Map	Interior generation
Material & Texture Variation	Generate all features randomly in one button (walls, doors, windows, props)		Spline based wall generation
Mesh Variation	Use of modular components to generate shapes		
Generate decoration & exterior props			

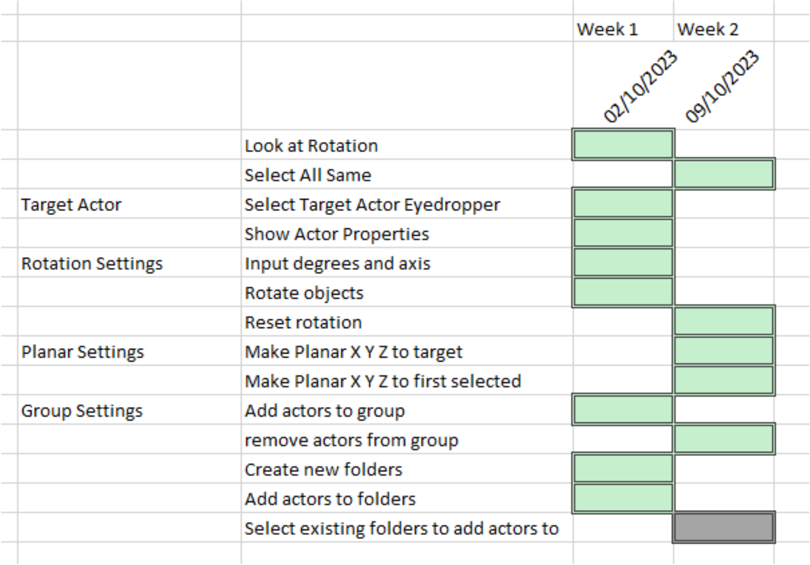
Tool 3 Ship Configurator

Must	Should	Could	Won't
Customise Ship Appearance	Possess and Control ship at runtime	Load Data from other Data Assets	Functional Health & Damage System
Customise Materials	Customise Crew Type	Basic Buoyancy Simulation	Advanced Water Buoyancy Simulation
Modify Runtime Data		Initialise runtime HUD widget with ship data	
Save Data to an Asset			

Timeline

Tool 1 – Level Kit

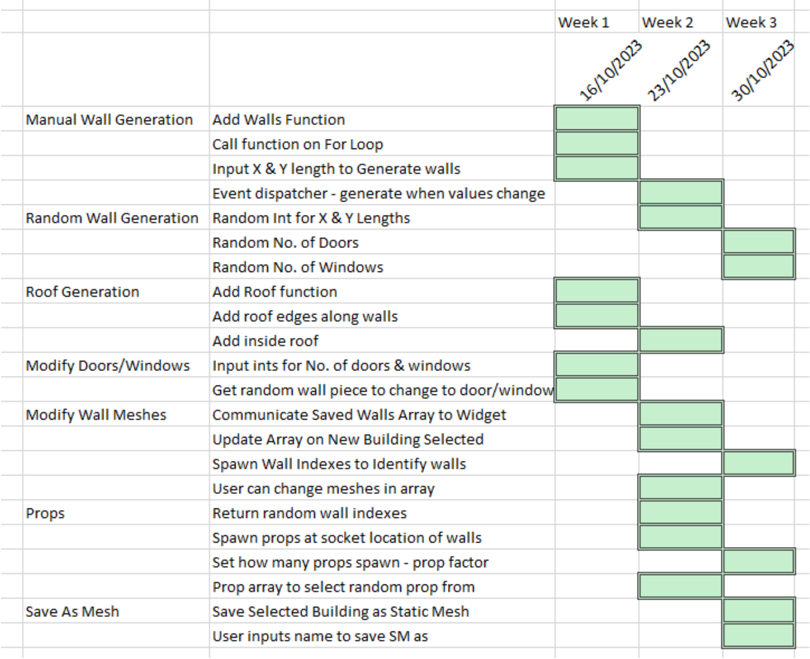
The Level Kit had a short time span of 2 weeks with a simple Gantt Chart to organise the development of the tool. The only task unable to make it into the tool is selecting existing folders to add actors to. This can still be done by typing in the name of the folder, however a list selection to pick from would've improved usability.



		Week 1	Week 2
		02/10/2023	09/10/2023
	Look at Rotation	🟩	
	Select All Same		🟩
Target Actor	Select Target Actor Eyedropper	🟩	
	Show Actor Properties	🟩	
Rotation Settings	Input degrees and axis	🟩	
	Rotate objects	🟩	
	Reset rotation		🟩
Planar Settings	Make Planar X Y Z to target		🟩
	Make Planar X Y Z to first selected		🟩
Group Settings	Add actors to group	🟩	
	remove actors from group		🟩
	Create new folders	🟩	
	Add actors to folders	🟩	
	Select existing folders to add actors to		⬛

Tool 2 – Building Generator

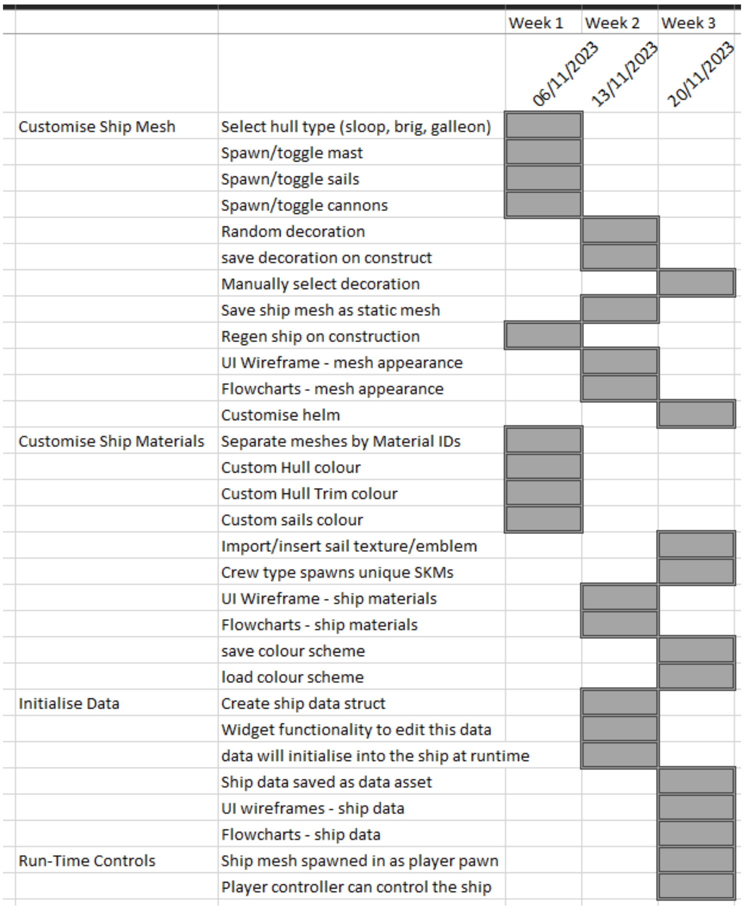
The Building Generator had a time span of 3 weeks during weeks 3/4/5 of the module. The fundamentals of each feature were created first to produce a Minimum Viable Product while later weeks were used for further development and expansion on these features.



		Week 1	Week 2	Week 3
		16/10/2023	23/10/2023	30/10/2023
Manual Wall Generation	Add Walls Function	🟩		
	Call function on For Loop	🟩		
	Input X & Y length to Generate walls	🟩		
	Event dispatcher - generate when values change		🟩	
Random Wall Generation	Random Int for X & Y Lengths		🟩	
	Random No. of Doors			🟩
	Random No. of Windows			🟩
Roof Generation	Add Roof function	🟩		
	Add roof edges along walls	🟩		
	Add inside roof		🟩	
Modify Doors/Windows	Input ints for No. of doors & windows	🟩		
	Get random wall piece to change to door/window	🟩		
Modify Wall Meshes	Communicate Saved Walls Array to Widget		🟩	
	Update Array on New Building Selected		🟩	
	Spawn Wall Indexes to Identify walls			🟩
	User can change meshes in array		🟩	
Props	Return random wall indexes		🟩	
	Spawn props at socket location of walls		🟩	
	Set how many props spawn - prop factor			🟩
	Prop array to select random prop from		🟩	
Save As Mesh	Save Selected Building as Static Mesh			🟩
	User inputs name to save SM as			🟩

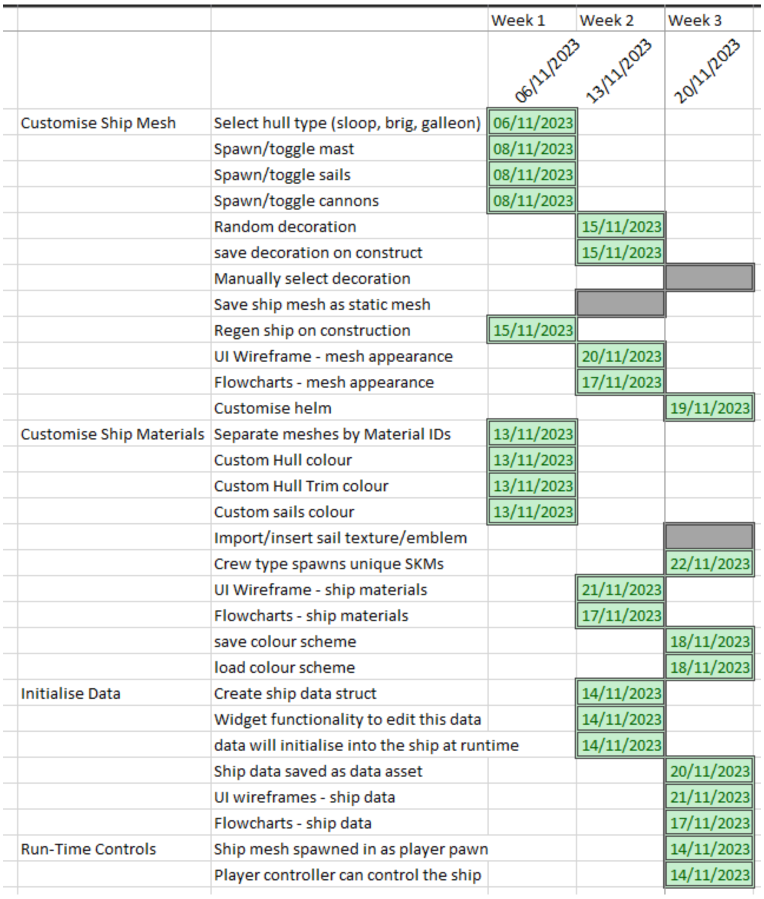
Tool 3 – Ship Configurator

This tool had a time span of 3 weeks during weeks 6/7/8 of the module. Like the Building Generator, the fundamentals of each feature were prioritised to create a basic prototype of the tool. These features were then expanded upon in later weeks to create more depth and customisation in the tool.



		Week 1	Week 2	Week 3
		06/11/2023	13/11/2023	20/11/2023
Customise Ship Mesh	Select hull type (sloop, brig, galleon)	▨		
	Spawn/toggle mast	▨		
	Spawn/toggle sails	▨		
	Spawn/toggle cannons	▨		
	Random decoration		▨	
	save decoration on construct		▨	
	Manually select decoration			▨
	Save ship mesh as static mesh		▨	
	Regen ship on construction	▨		
	UI Wireframe - mesh appearance		▨	
	Flowcharts - mesh appearance		▨	
	Customise helm			▨
Customise Ship Materials	Separate meshes by Material IDs	▨		
	Custom Hull colour	▨		
	Custom Hull Trim colour	▨		
	Custom sails colour	▨		
	Import/insert sail texture/emblem			▨
	Crew type spawns unique SKMs			▨
	UI Wireframe - ship materials		▨	
	Flowcharts - ship materials		▨	
	save colour scheme			▨
	load colour scheme			▨
Initialise Data	Create ship data struct		▨	
	Widget functionality to edit this data		▨	
	data will initialise into the ship at runtime		▨	
	Ship data saved as data asset			▨
	UI wireframes - ship data			▨
	Flowcharts - ship data			▨
Run-Time Controls	Ship mesh spawned in as player pawn			▨
	Player controller can control the ship			▨

A couple of features planned on the Gantt Chart have not been implemented. This was due to other features and mechanics being added during development taking a higher priority. These features added a small amount of extra customisation to the ships but are not completely required to make the tool functional.



		Week 1	Week 2	Week 3
		06/11/2023	13/11/2023	20/11/2023
Customise Ship Mesh	Select hull type (sloop, brig, galleon)	06/11/2023		
	Spawn/toggle mast	08/11/2023		
	Spawn/toggle sails	08/11/2023		
	Spawn/toggle cannons	08/11/2023		
	Random decoration		15/11/2023	
	save decoration on construct		15/11/2023	
	Manually select decoration			▨
	Save ship mesh as static mesh		▨	
	Regen ship on construction	15/11/2023		
	UI Wireframe - mesh appearance		20/11/2023	
	Flowcharts - mesh appearance		17/11/2023	
	Customise helm			19/11/2023
Customise Ship Materials	Separate meshes by Material IDs	13/11/2023		
	Custom Hull colour	13/11/2023		
	Custom Hull Trim colour	13/11/2023		
	Custom sails colour	13/11/2023		
	Import/insert sail texture/emblem			▨
	Crew type spawns unique SKMs			22/11/2023
	UI Wireframe - ship materials		21/11/2023	
	Flowcharts - ship materials		17/11/2023	
	save colour scheme			18/11/2023
	load colour scheme			18/11/2023
Initialise Data	Create ship data struct		14/11/2023	
	Widget functionality to edit this data		14/11/2023	
	data will initialise into the ship at runtime		14/11/2023	
	Ship data saved as data asset			20/11/2023
	UI wireframes - ship data			21/11/2023
	Flowcharts - ship data			17/11/2023
Run-Time Controls	Ship mesh spawned in as player pawn			14/11/2023
	Player controller can control the ship			14/11/2023

Budgeting

Tool	Feature	Time Budget (hours)	Financial Budget (£)
Level Kit	Look at Rotation	2	0
	Select All Same	4	0
	Target Actor Properties	2	0
	Rotation Settings	4	0
	Planar Settings	4	0
	Group Settings	6	0
	Subtotal:	22	0
Building Generator	Manual Wall Generation	6	0
	Random Wall Generation	4	0
	Roof Generation	4	0
	Door/Window Generation	6	0
	Modify Wall Meshes	4	0
	Prop Generation	12	0
	Save As Mesh	8	0
	Subtotal:	44	0
Ship Configurator	Select Hull Type	2	0
	Add Masts & Sails	8	0
	Add Rigging & Cannons	8	0
	Prop Generation	6	0
	Customise Ship Materials	10	0
	Initialise Run-Time Data	8	0
	Create Runtime Controls	8	0
			0
	Subtotal:	50	0
	Project Total:	116	0