Technical Design Document

Version 1.4 – 22/03/26

Changelog

Version 1.0 - 03/03/26 - Add sections for GitHub usage, naming conventions, commenting practices, tech stack, platform targets, risks and challenges, folder directory structure.

Version 1.1 - 04/03/26 - System breakdown flowcharts, refactoring folder directory system, merging on GitHub and added more player systems breakdown detail.

Version 1.2 - 05/03/26 - minor tweaks to have the universities Unreal Engine version and system requirements, added screenshots to that. Bibliography for references when needed

Version 1.3 - 12/03/26 - Added detailed system breakdown information and more flowcharts. Also updated the folder directory structure, improved the naming conventions section and added a sentence about functions to the documentation section.

Version 1.4 - 22/03/26 – Updated documentation to include guidance about easy node organisation.

Project Introduction

The game is an isometric puzzle game focused on environmental problem solving and navigation. Players must interact with objects and avoid hazards while using logic and spatial awareness to solve puzzles and progress through each level.

The player controls two characters: a fallen celestial being and a humanoid companion who are trying to climb a tower so the celestial can return to the sky. The game is a single-player, grid-based puzzle game where the player switches between the two characters to solve environmental puzzles.

Risks and Challenges

TODO

- Multiple characters
- Lighting system updating BP's may be non-performant – each light source must scan for all effected BP's and update them every time they move.

Folder Directory Structure

Everyone must have their own folder directory. Please do not modify files outside of your personal folder. Whenever you want to add or modify a file in the main content folder you should copy the file and folders to your own directory. This version of the file is the one you should make modifications to.

For example, if I wanted to change a file in Content/Assets/UI I would copy all of Content/Assets into my folder and then delete anything not currently being modified. This means I would then still have all Content/etc in a separate folder and my folder would exclusively have Developers/Tech/Jake/Content/Assets/UI unless I needed to access anything else.

When multiple people need to simultaneously work on the same blueprint or asset then the file should be duplicated and placed in the respective folders. This is to make merging easier.

Upon a successful merge, that personal folder will be wiped as it can now be found on the main branch. For instance, if the design branch was merged with the main branch, then everyone on the design team would have the contents of their folders deleted.

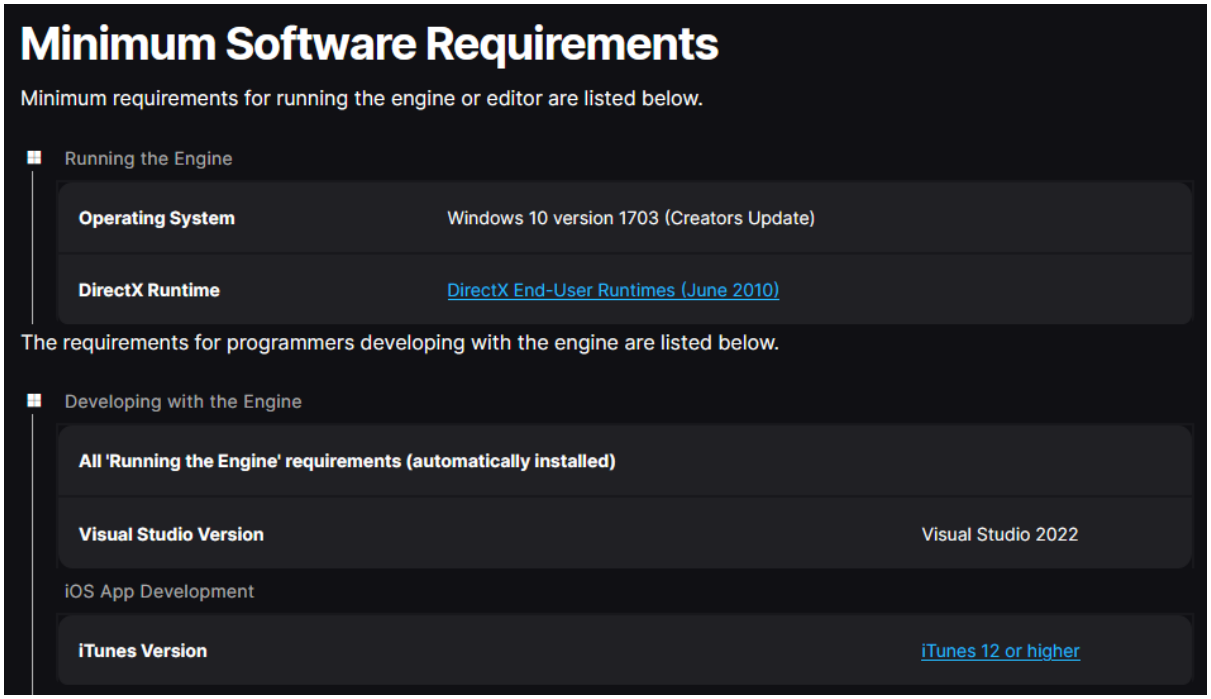
- Developers
    - Team (Art/Design/Tech)
        - Person in team
- Content
    - Imported Assets (separate from functional assets for artists ease of use)
        - Soundwaves
        - Textures
            - VFX
            - UI
            - Meshes
        - Meshes
        - Animation (sequences/skeletons)
    - Blueprints
        - Characters
            - Player (Player BP, animation BP & controller BP)
        - Input (input actions and input mappings)
        - Data (structs, enums, tables, persistent data etc))
            - Components
            - Structures
        - Systems
            - Game modes
            - Managers
            - Utilities
            - Interfaces
        - UI
    - Levels
    - SFX
        - Metasounds
        - Sound Concurrency
        - Sound Attenuation
    - Materials
        - Material (Contains material and its instances)
        - Post Process
        - VFX
    - VFX

    - Niagara Systems
    - Niagara Emitters

Tech Stack

- JetBrains Rider
- Visual Studio 2022
- GitHub Desktop
- Unreal Engine 5.6.1
- Adobe Suite (Substance Painter, Substance Designer & Photoshop)
- Autodesk Suite (Maya & 3DSMax)
- Blender
- ZBrush

Unreal Engine 5.6.1 System Requirements



Minimum Software Requirements

Minimum requirements for running the engine or editor are listed below.

Running the Engine

Operating System	Windows 10 version 1703 (Creators Update)
DirectX Runtime	DirectX End-User Runtimes (June 2010)

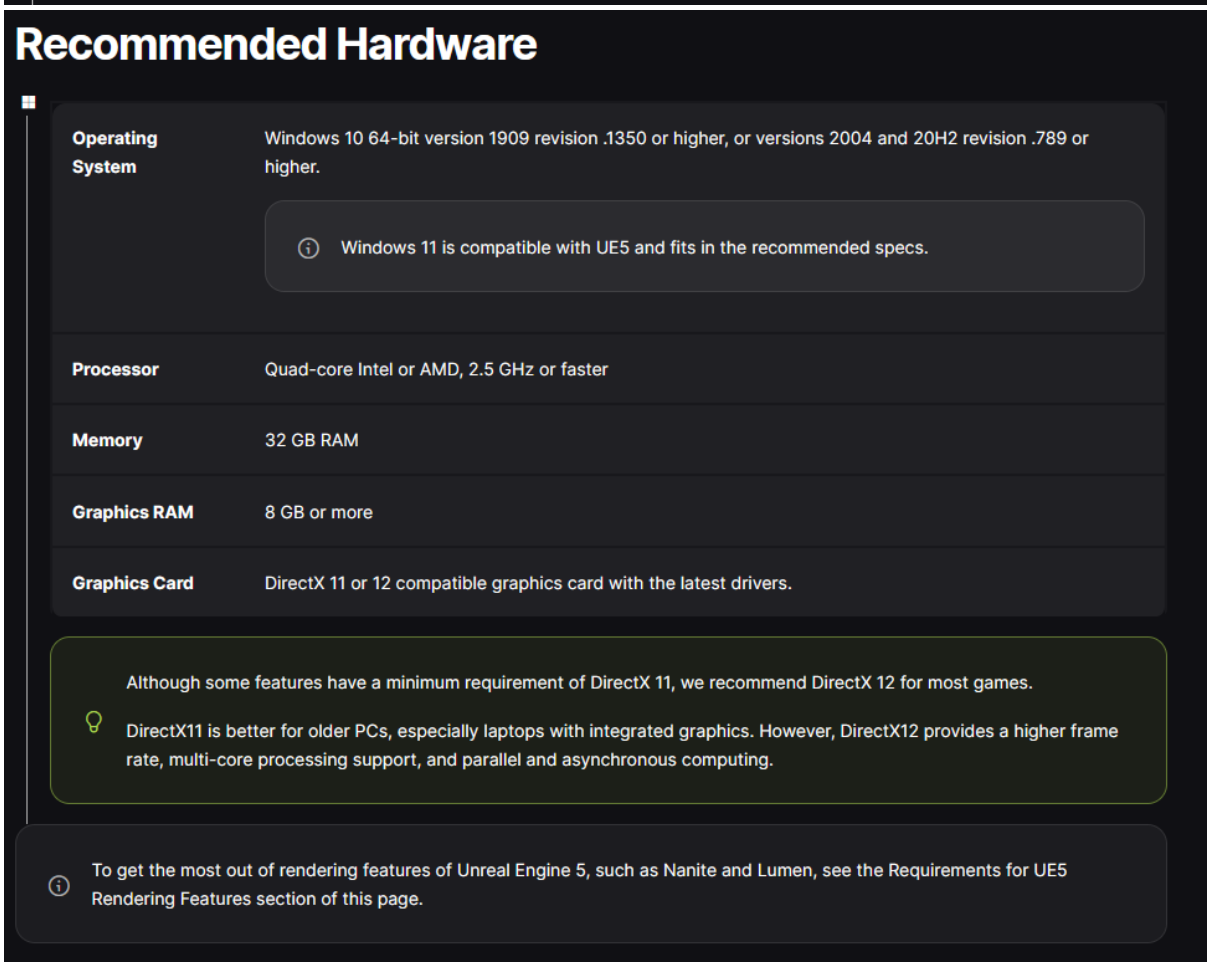
The requirements for programmers developing with the engine are listed below.

Developing with the Engine

All 'Running the Engine' requirements (automatically installed)	
Visual Studio Version	Visual Studio 2022

iOS App Development

iTunes Version	iTunes 12 or higher



Recommended Hardware

Operating System	Windows 10 64-bit version 1909 revision .1350 or higher, or versions 2004 and 20H2 revision .789 or higher.
Processor	Quad-core Intel or AMD, 2.5 GHz or faster
Memory	32 GB RAM
Graphics RAM	8 GB or more
Graphics Card	DirectX 11 or 12 compatible graphics card with the latest drivers.

Windows 11 is compatible with UE5 and fits in the recommended specs.

Although some features have a minimum requirement of DirectX 11, we recommend DirectX 12 for most games.

DirectX11 is better for older PCs, especially laptops with integrated graphics. However, DirectX12 provides a higher frame rate, multi-core processing support, and parallel and asynchronous computing.

To get the most out of rendering features of Unreal Engine 5, such as Nanite and Lumen, see the Requirements for UE5 Rendering Features section of this page.

Platform Targets

Minimum:

Windows 11 | 60 FPS | 1080p

Ideal:

Windows 11 | 120 FPS | 2140p

GitHub – Technical Basics

To start using the GitHub repo, send Jake (Tech Lead) your GitHub username, if you have not already. You should then clone the repo to your local machine using GitHub Desktop. This should be on the lab machines already. Every team will have their own branch. You should also make a branch in the format team-name. This will be deleted after you have finished your feature and committed it to the repo.

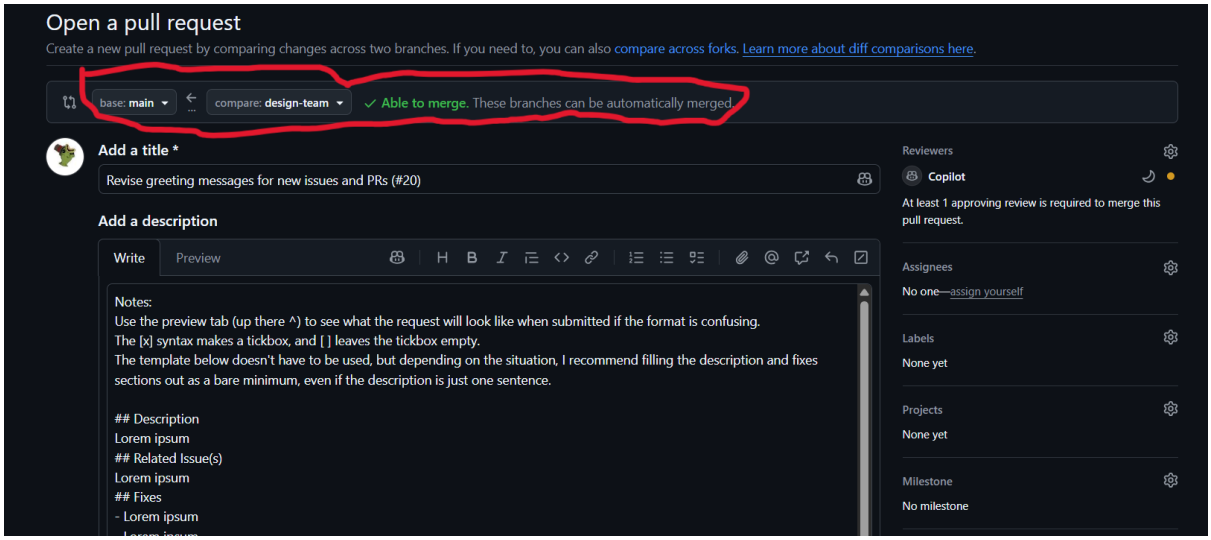
Committing

If you want to commit the work you've done, then commit it to your own branch. When switching branches, you should ensure that Unreal/code editors are closed and you have no unsaved changes otherwise it may break things and try to stage changes on the new branch. I strongly recommend you always keep your branch up to date with your team's branch, as they will always have the most recent features. This ensures you always have access to the most recent changes/features and aren't trying to work on something in your own branch that has already been uploaded to the repository.

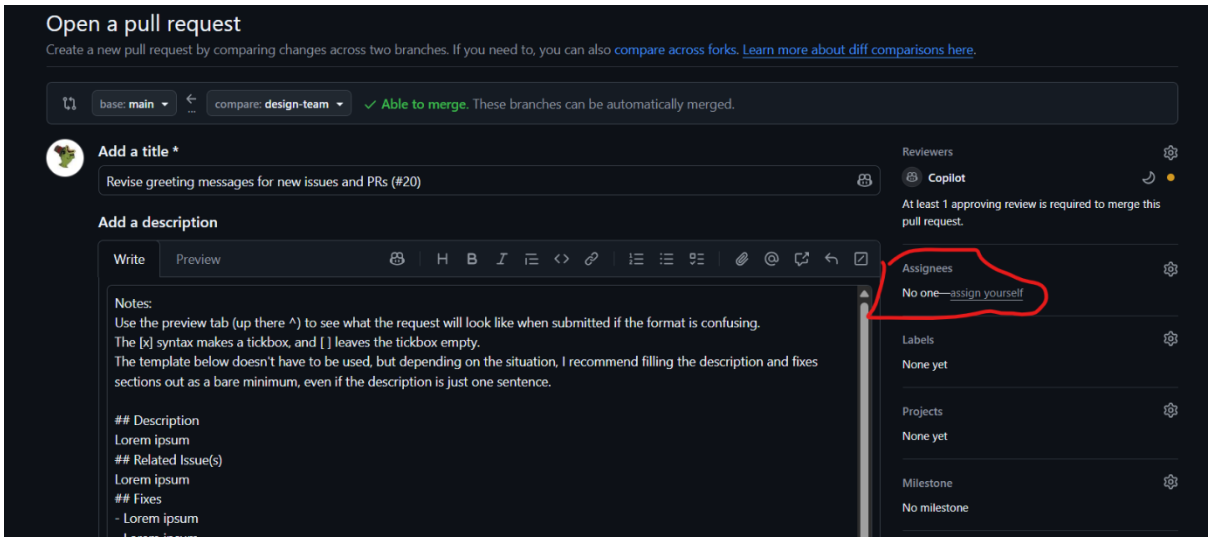
Merging

If you have finished an entire feature, need to update your branch or need to give your work to the rest of your department; then you should create a pull request on your departments branch. Ensure the branch you're merging to is in the left box.

Branch destination (where you want changes to go) ← branch source (where things have been changed).



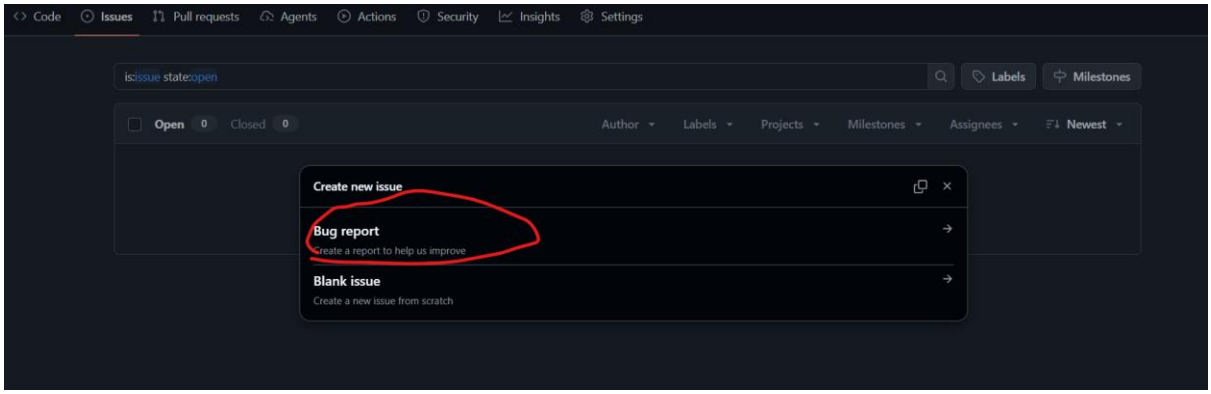
After that you can fill out the description section and follow the instructions on the custom template. This can then then be reviewed and approved by someone on the tech team (so assign someone). The branch you were working on will then be merged and deleted, to ensure everything stays up to date.



Issues

These are a great way to communicate any bugs to the tech team. When you try to create an issue, it will ask you to select which template to use. Generally, you should choose bug

report because it has a default assignee and tag set.

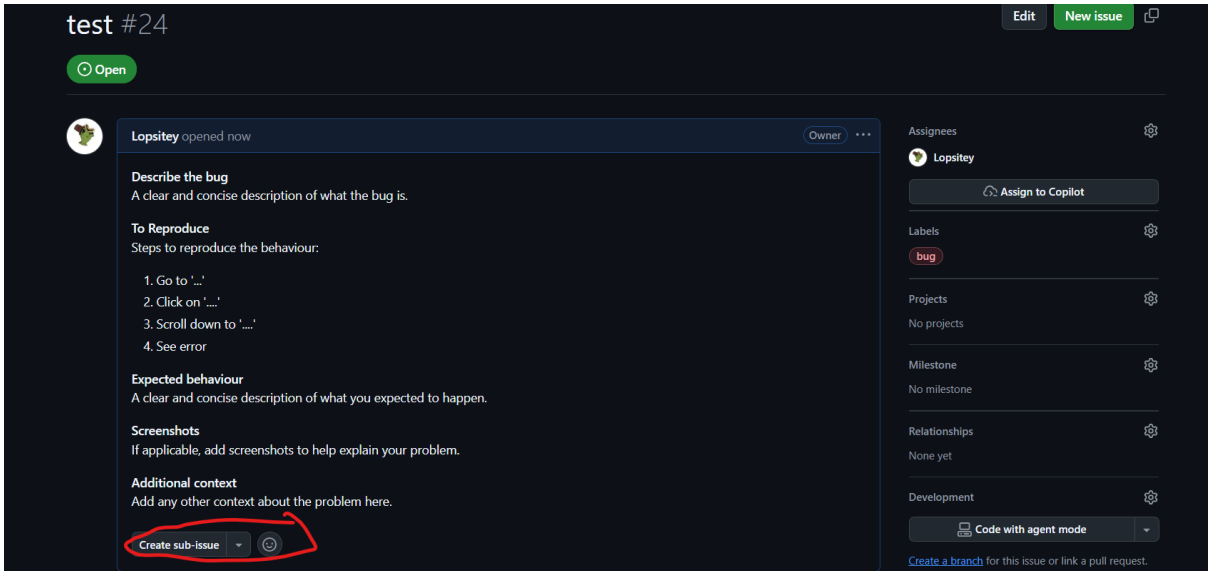


However, if you have a specific query, you can select blank issue and fill it out from scratch. If you forget to add an assignee to a blank issue or remove the default on the bug report, then an action will be run to automatically add Jake to the assignees list.

Upon being added, an event is activated in Microsoft Power Automate which adds the new issue to the Microsoft planner bugs column. This also means that if you change a pre-existing issue on GitHub; you can then remove and re-add Jake as an assignee, so the MS plan gets updated.

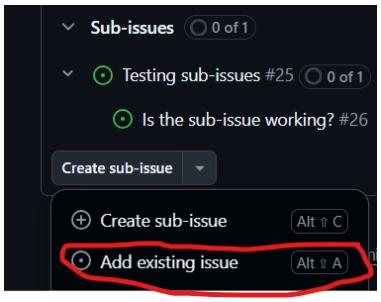
Sub-Issues

You can also create sub-issues. This is great for organising closely related issues.

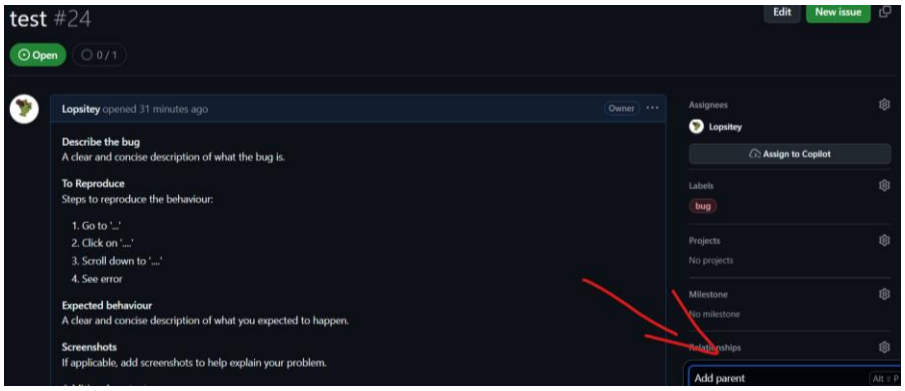


You can make sub-issues of sub-issues which can be nested up to eight times.

This dropdown makes a pre-existing issue a child of the current.



This allows you to make a pre-existing issue a parent of the current.



Organising issues like this is great because, for instance, you could have one large issue with two sub-issues. The large issue could be ammo, which says something like: ammo doesn't work properly, the first sub-issue could say: lightning ammo not loading into mag and the second could say: fire ammo tick rate too high.

Testing and Profiling

TODO

Issues submitted through GitHub can be ticked off on MS planner when testing.

Tests should follow the pattern on the main testing sheet.

When optimising for the highest performance Unreal profiler should be used along with the reference tree and memory graphs.

Class Architecture

TODO

Naming Conventions

- Classes – snake case: BP_player_character
- Functions – pascal case: FireWeapon
- Variables – camel case: currentHealth
- Other cases – ask Jake (more info)

Asset Type	Prefix
Materials	
Material	M_
Material Instance	MI_
Post Process Material	PPM_
Texture	T_
Animation	
Animation Blueprint	ABP_
Skeleton	SK_
Skeletal Mesh	SKM_
Rig	RIG_
Animation Sequence	AS_
Animation Montage	AM_
Blend Space	BS_
Meshes	
Static Mesh	SM_
Blueprints	
Blueprint	BP_
Blueprint Interface	BI_
Widget Blueprint	WBP_
Editor Utility Widget	EUW_
Blueprint Function Library	BFL_
Blueprint Component	BPC_
Actor Component	AC_
Levels	
Level	LVL_
Level Sequence	LS_
Data	
Curve Table	CT_
Data Asset	DA_
Data Table	DT_
Enum	E_
Structure	F_

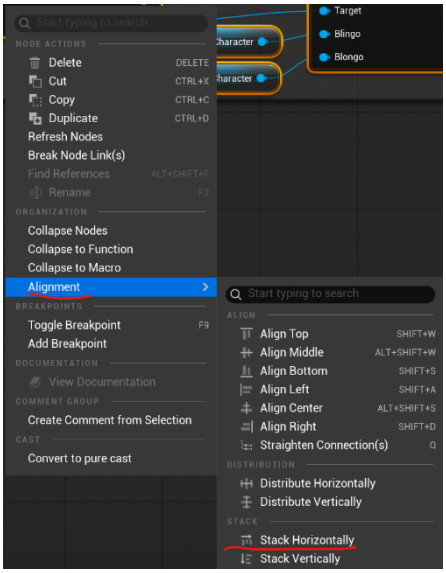
VFX	
Niagara Emitter	NE_
Niagara System	NS_
SFX	
Metasound	MS_
Sound Wave	SW_
Sound Concurrency	SC_
Sound Attenuation	SA_
Inputs	
Input Action	IA_
Input Mapping Context	IMC_

Documentation

Variables inside blueprints should be placed in relevant categories. This makes them easier to find in the editor and details panels.

In blueprint and C++, there should be a main comment for every function and extra comments for ambiguous or hard to understand sections. Additionally, anything that requires attention or is a work in progress should be commented. If a blueprint contains nodes, it should contain at least one comment. A function's main comment should also be pasted into the description box in the details panel so you can hover over it to see the functionality.

For easy blueprint organisation, please select all your nodes and then right click to open the menu. Click alignment and then select **Stack Horizontally**. It takes five seconds and makes complete systems far easier to read and less like spaghetti.



Do not use comments for communicating identified bugs, this should be done using GitHub and MS Planner. This way they are documented in one place to be viewed by the whole team.

Systems Breakdown

This is how every system has been implemented, not the design or purpose behind any mechanics.

CharacterBase (Abstract class)

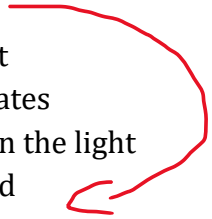
Summary

Checks WASD input and moves tile position based on that.

Movement uses timeline lerp for each position not player movement system

Input should be turn-based. An example:

1. Input pressed
2. Player pushes object
3. Pressure plate activates
4. Checks if Blongo is in the light
5. Next input is allowed



Action delay makes the animation look smooth and peaceful.

Functions

- GridMove
- CheckGrid (Blueprint function library or subsystem)
- CheckShadow (Actor Comp or Function)

Children

- Blingo
- Blongo

Blingo (Child class)

Interfaces

- Weighted
- Moveable

Moveable Object (Class)

Summary

If player attempts to move into pushable block, check the tile it will be pushed into, if empty, allow movement

Interfaces

- Moveable
- Weighted

Functions

- CanPush

Children

- Block

TriggerBase (Abstract class)

Interfaces

- Interactable (Interface)

Functions

- Interact

Children

- Pressure Plate
- Button (Interactible)

TriggerableBase (Abstract class)

Functions

- Activate

Children

- Door
- Moving Block

StartLevel

Summary

Two separate entrances to the level specified as one each for Blingo and Blongo.

EndLevel

Summary

Two separate exits to the level specified as one each for Blingo and Blongo. The level is only left when both characters are on their exits.

Movement Component

Summary

Converts world locations to grid tiles.

CheckGrid - Checks occupancy of tile at location.

MoveActor – Would be called by the pathfinding function in the player and used when moveable objects are pushed. This would lerp player/object between locations in the grid.

Functions

- CheckGrid
- MoveActor

Command History

Summary

Handles the data for the players inputs so that they can be undone and redone.

AddToStack -

Functions

- CanUndo
- CanRedo
- Undo
- Redo
- AddToStack
- CommitStack

Level Generator

Summary

The lights would store the static and dynamic layers. This would be done using an interface. When a tile is updated, an event is run which the light caster would then use to update the dynamic layer. At the start of the level the static layer is ran to initialise the shadows. Moveable objects would invoke the update event to affect the dynamic layer.

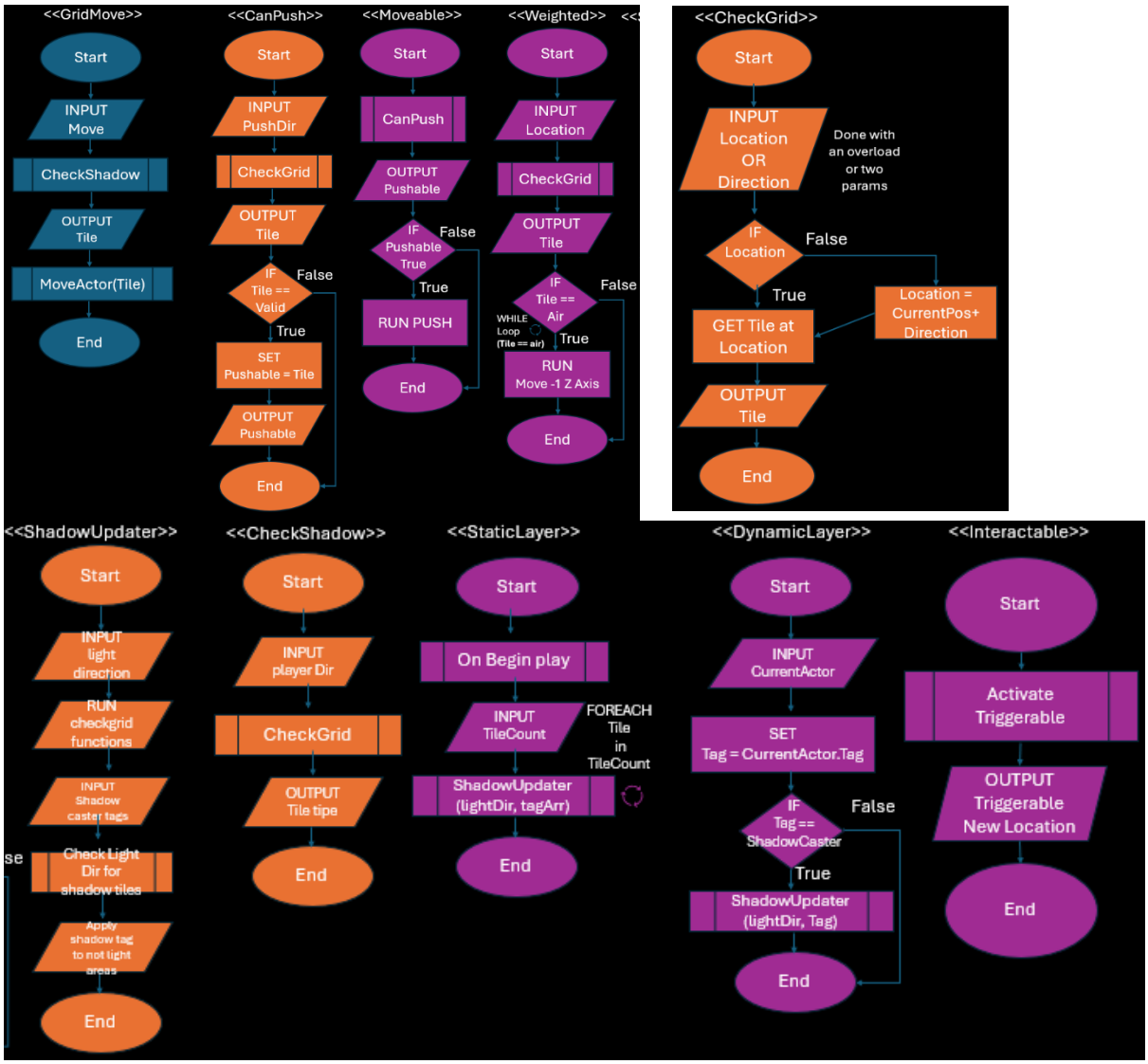
The event update should send a reference to the actor that invoked it so the light can check that to find the tag.

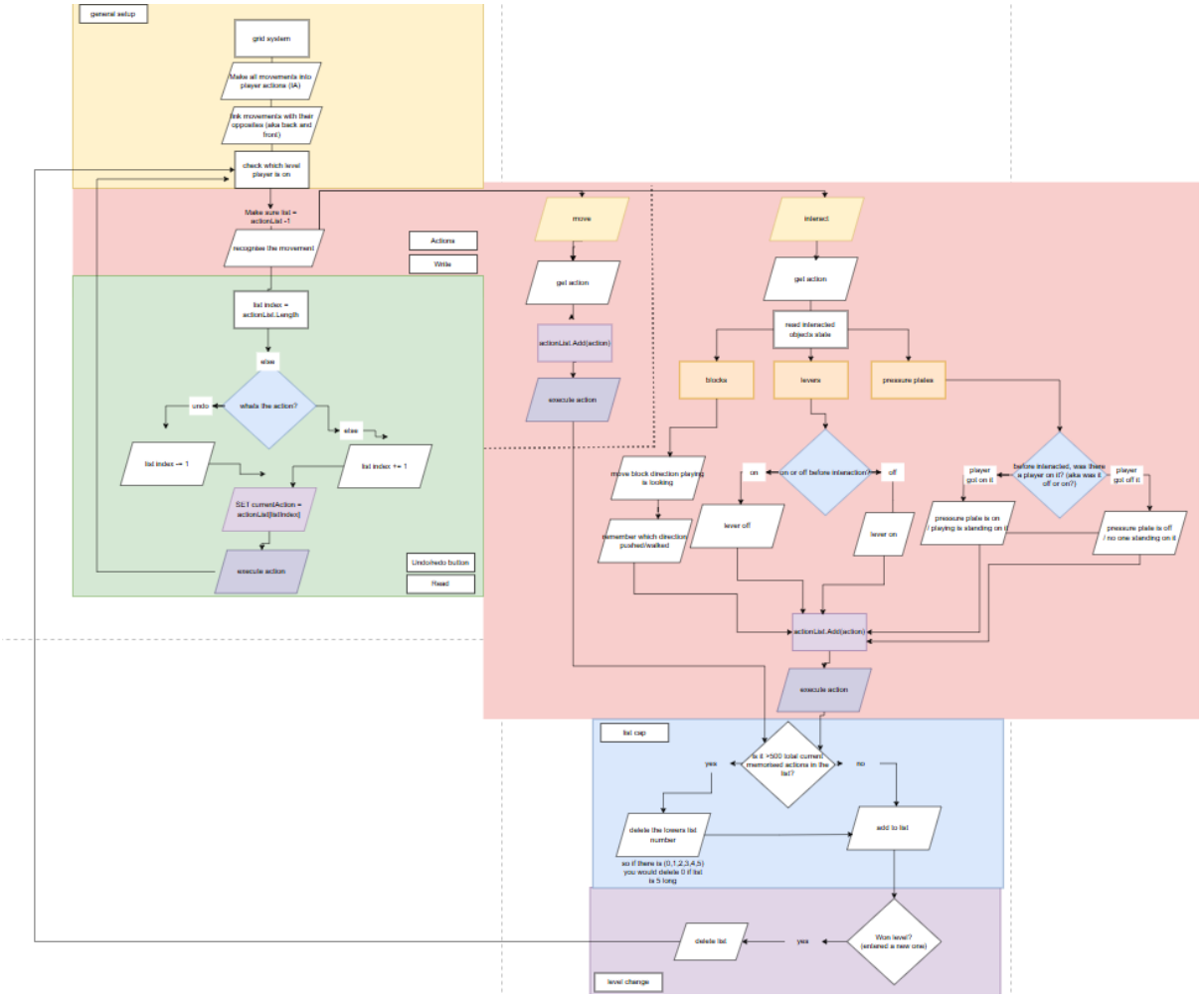
Add a "blocked" or "wall" query tag?

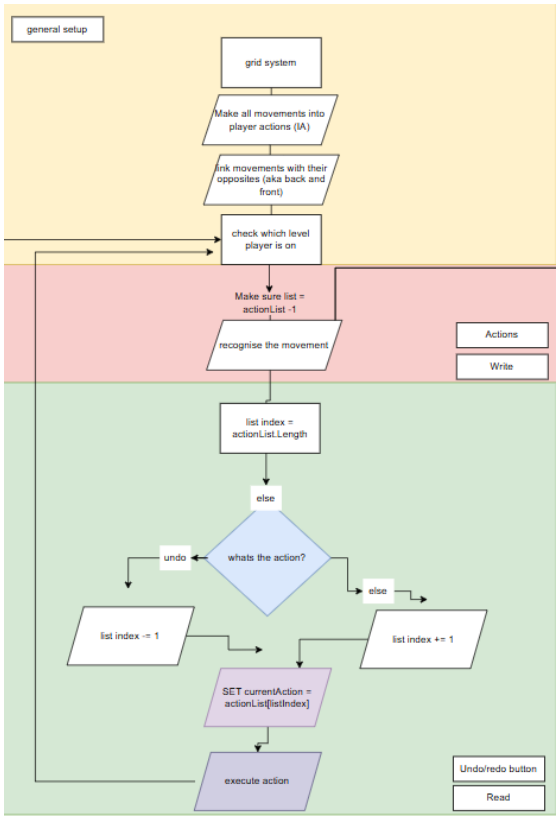
System Flowcharts

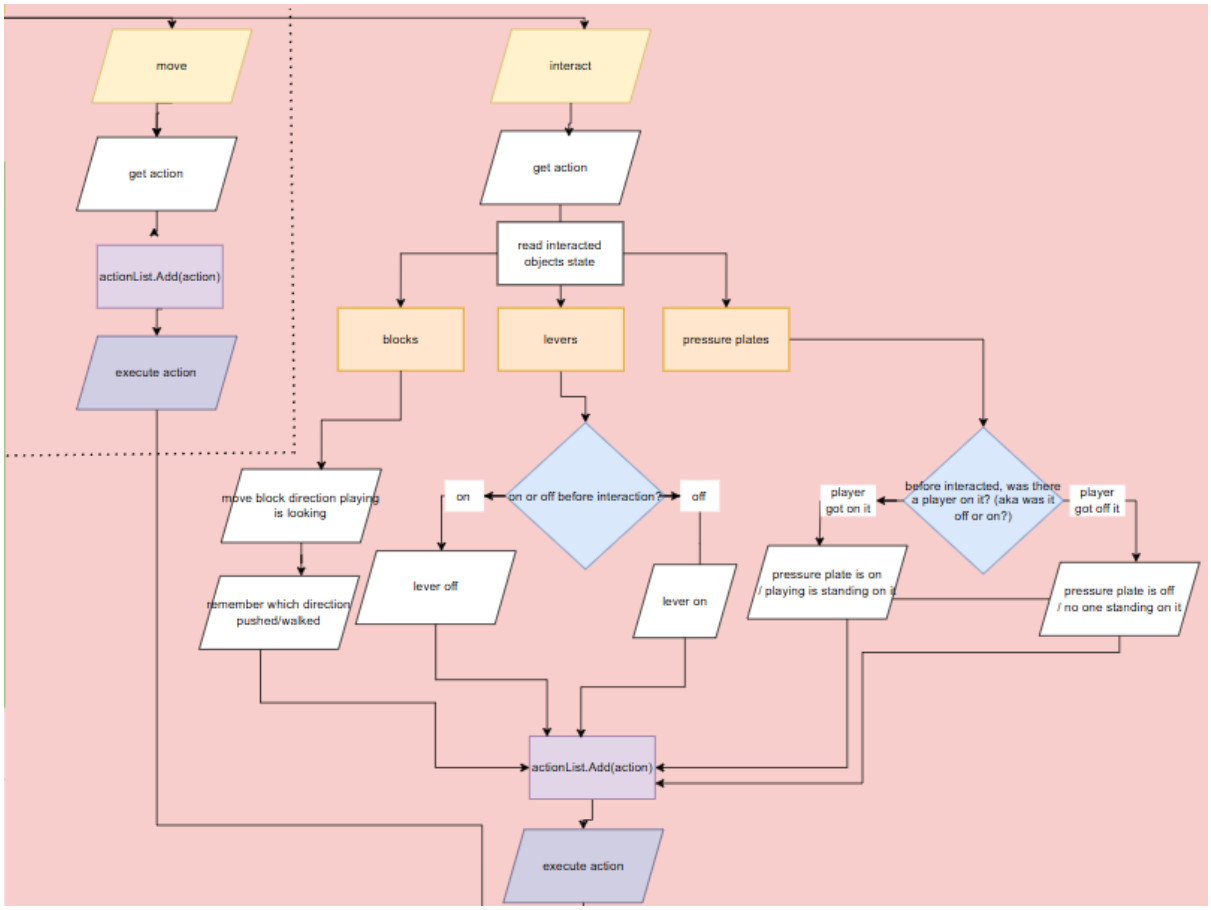
Legend

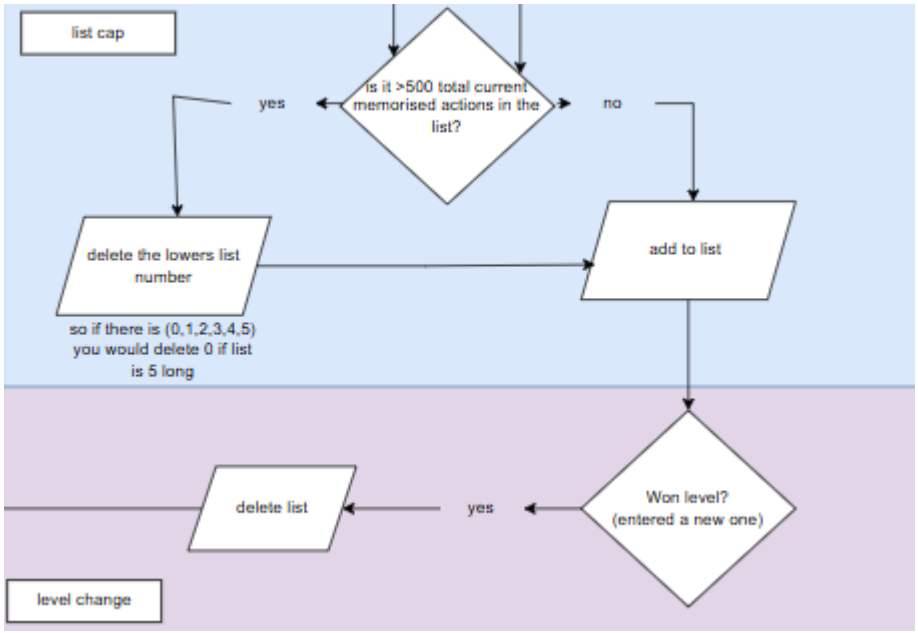
Colour	Meaning
Blue	Nodes in a blueprint
Orange	Function
Purple	Interface




general setup
grid system
Make all movements into player actions (IA)
link movements with their opposites (aka back and front)
check which level player is on
Make sure list = actionList -1
recognise the movement
Actions
Write
list index = actionList.Length
else
whats the action?
undo
else
list index -= 1
list index += 1
SET currentAction = actionList[listIndex]
execute action
Undo/redo button
Read
move
get action
actionList.Add(action)
execute action
interact
get action
read interacted objects state
blocks
levers
pressure plates
move block direction playing is looking
remember which direction pushed/walked
on
on or off before interaction
off
lever off
lever on
player got on it
before interacted, was there a player on it? (aka was it off or on?)
player got off it
pressure plate is on / playing is standing on it
pressure plate is off / no one standing on it
actionList.Add(action)
execute action
list cap
is it >500 total current memorised actions in the list?
yes
no
delete the lowers list number
so if there is (0,1,2,3,4,5) you would delete 0 if list is 5 long
add to list
delete list
yes
Won level? (entered a new one)
level change


general setup
grid system
Make all movements into player actions (IA)
link movements with their opposites (aka back and front)
check which level player is on
Make sure list = actionList -1
recognise the movement
Actions
Write
list index = actionList.Length
else
whats the action?
undo
else
list index -= 1
list index += 1
SET currentAction = actionList[listIndex]
execute action
Undo/redo button
Read


move
get action
actionList.Add(action)
execute action
interact
get action
read interacted objects state
blocks
levers
pressure plates
move block direction playing is looking
remember which direction pushed/walked
on
on or off before interaction?
off
lever off
lever on
before interacted, was there a player on it? (aka was it off or on?)
player got on it
player got off it
pressure plate is on / playing is standing on it
pressure plate is off / no one standing on it
actionList.Add(action)
execute action


list cap
Is it >500 total current memorised actions in the list?
yes
no
delete the lowers list number
so if there is (0,1,2,3,4,5) you would delete 0 if list is 5 long
add to list
Won level? (entered a new one)
yes
delete list
level change

DynamicLayer

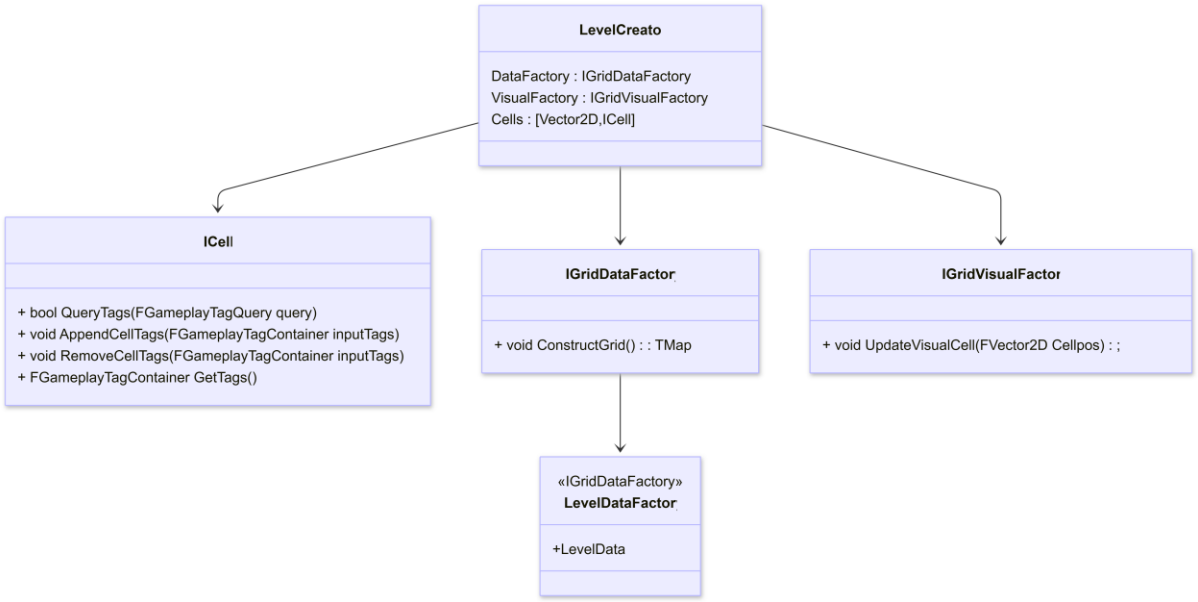
Objects on this layer are modify the data layer.

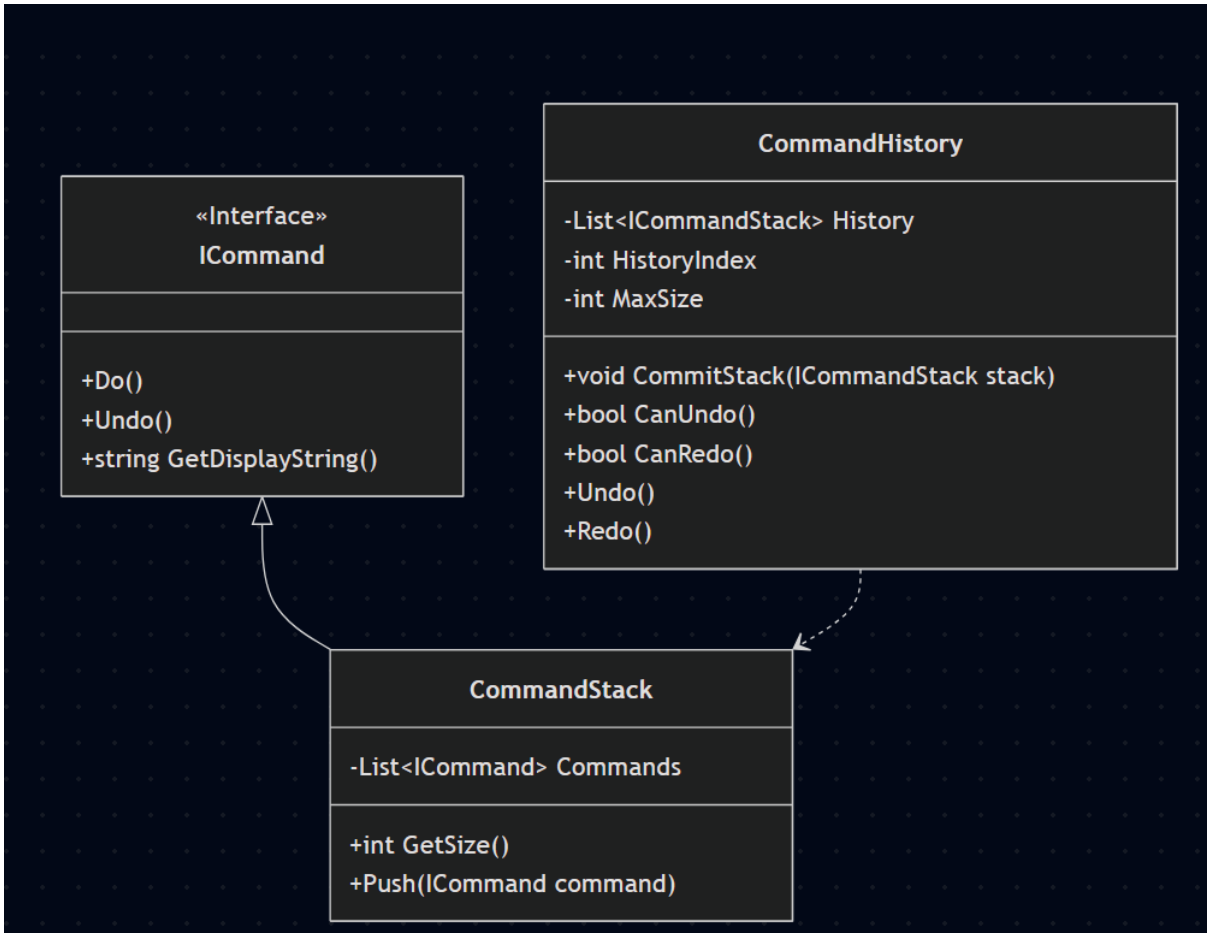
Interactions happen here and are passed onto the data layer.

DataLayer

The data layer is used for quickly querying the state of the grid.

It is used to determine cell data such as: "shadow", "wall", "box" etc.





Bibliography

Epic Developer Community. (n.d.). *Hardware and Software Specifications for Unreal Engine*. [online] Available at: https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-and-software-specifications-for-unreal-engine.