Game Development Tool Design Document

Type of software

This type of software tool will be a standalone desktop application built using unity that is meant to be used in combination with MindFeasts CoOpertion game, with the aim of allowing easier creation of levels for the game.

Product Vision

This product is aimed at people who are wanting to create mods and custom levels for Co-Operation but either don't have the knowledge to create YAML or don't want to manually create the YAML themselves. Co-operation Level Editor is a piece of standalone software that will allow for users/modders/developers to be able to easily construct level without sifting through YAML code just to create a simple level as well as easily creating object definitions with specific tags, and Lua scripts. This project is unique due to Co-Operation being in a closed alpha stage and not already having a Level editor. This Meaning the market for this (free) product is quite sparse and a gap that can be filled.

Core mechanics

Interactable Map

There will be an interactable map. The user will be able to pan around and zoom in and out to facility full movability around the map. There won't be any functionality that allows for rotating as in Co-Operation the camera is static and by allowing rotating the user could accidentally produce a map that isn't viewable from the camera in game. This will be achieved with a script attached to Unity's main camera and will be based on the mouses screen coordinates instead of world coords, so implementation is simpler.

Object CRUD

CRUD operations for objects will be implemented so they can be manipulated on the map. Such functionality will be:

- creating/placing objects within the world
- Viewing and the ability to read the object attributes so the user can understand what is going on with the object if bugs occur like accidentally scaling too large.
- Updating the objects so that both the model and the attributes attached to it can change. Attributes will be Lua scripts, tags, and object world position data. This will allow for correction of mistakes and modifying of the object's current functionality
- removing/deleting the objects from the map, so that objects wont piles up if too many are made. This will be a permanent delete and no history will be kept so once an object is deleted the user will have to recreate the object if they want it back

Export and import YAML

To allow for the level editor to work properly and so the levels can be played, the importing and exporting of YAML is needed. There will be 2 scripts used one for import and the other for exporting, and to achieve this I will use an already made C# YAML parser to be able to both serialise and deserialise the YAML for easier digesting.
(https://github.com/aaubry/YamlDotNet)

Would Like mechanics

Open and edit Lua Scripts

A feature I would like to add would be to let the user be able to open Lua files in the editor, so they don’t need to edit them in an external file. This is more quality of life as it would keep everything and stop the need for switching applications. But this is if i have the time to implement it and the current goal it to just make the level editor instead of a texted editor too.

Readable YAML output

An issue that could occur is that the exported YAML is unreadable. This can come from the definitions of the objects used in the level, as it is very unlikely that 2 objects will be the same so cant share the object definition. A way this could be combated is to separate out the definitions and the levels (e.g. Level1.yaml && level1_obj.yaml)

Users

The users will be people who own the game and want to make mods for the game, while it won't overly help with the Lua it will allow for easier creation of levels. I aim to target people who have been in similar situation as mine, where I wanted to create levels but didn't want to keep tracking and counting rows and columns just to update one thing, or getting confused when not being able to figure out which tile was just above another as its 12 tiles back in the list.

Technical specs

The types of models that will be used will be 3D as the game objects are in glb format. This type of format has multiple unity library that allows for dynamic loading of glb files so they won't need to be compiled into the software and can be updated runtime. (https://docs.unity3d.com/Packages/com.unity.cloud.gltfast@6.8/manual/index.html)(https://github.com/KhronosGroup/UnityGLTF)

The viewing angle for the game will be isometric to closey mimic the angle in the game. A panning and zooming mechanic will also be added to move around the world so levels can be as big or as small as the user wants. Due to having only one viewing angle in the game there will be no rotating as the user might end up creating a map or section that is completely unviewable when exported and used in game.

Target Platform

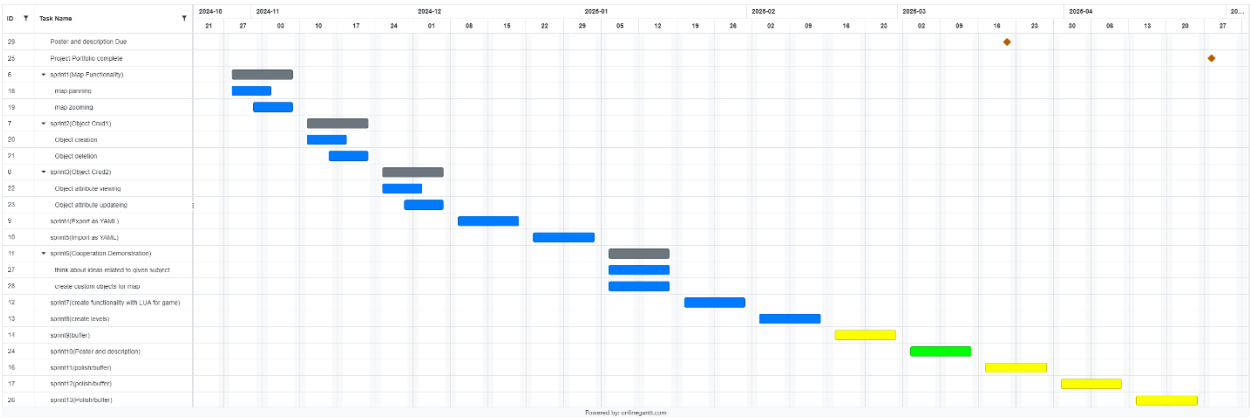
The target platform will first be windows due to development being primarily done on a windows desktop but won't mean that it will be limited to windows as unity provides different compile target meaning Linux is an option that can be thought about later.

Risk Plan

Risk Plan

risks	how to solve
Broken/partially working code	Create tests to make sure function run correctly
Burnout	Slow down, think and do the most important thing. Go outside for walk
Scope creep	Document the initial functional and nonfunctional requirements list, and rank each by how important. If new features are thought up add to a separate list and decide where it stand with the current list
Inaccurate features for software	Gather user requests and information on what should be added. Carry out user testing when possible and appropriate
Team member leaving	This can't happen unless I leave
Ill understanding on method of Implementation	Extensive design on how systems will work together, front end menu to back and translating to YAML
procrastination	Create short time goals based around current development section, so I don't get lost and confused on what to do next

Gantt chart



Design documents

Figure 1 shows the functions and methods that will be use in the objects class. Each object will contain an object script that stores all the attributes associated with the object on the map as well as have defined functions to allow for modification of the attributes and to output individual objects as their YAML representation.

Figure 2 are the basic UI mock ups to show the interactive panels for the user to use. It shows the object selection and placement panel, overall attribute reader that will allows for modification of single values such as position and rotation, and two separate popup panels for editing list variables which are the tags and Lua script lists.

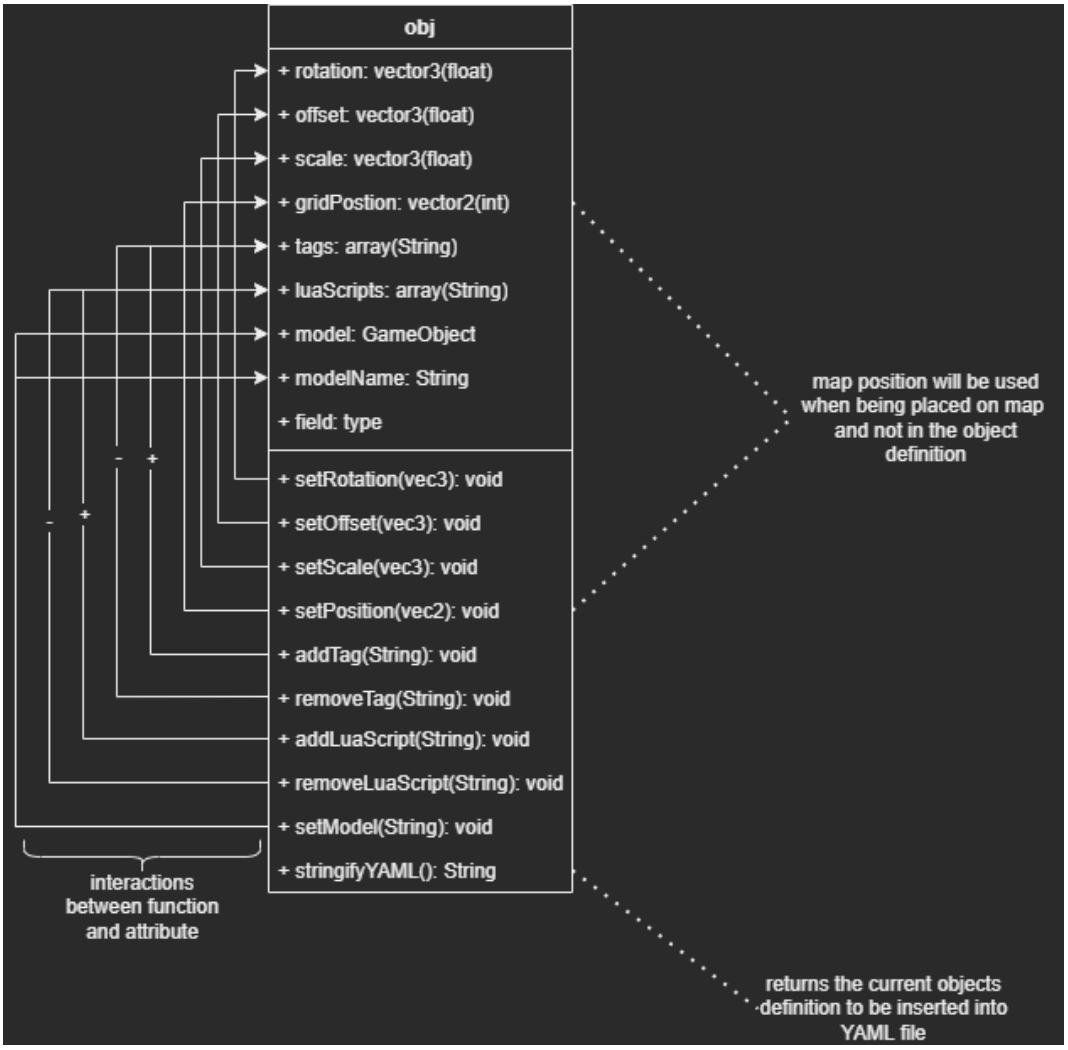


Figure 1- design for object class

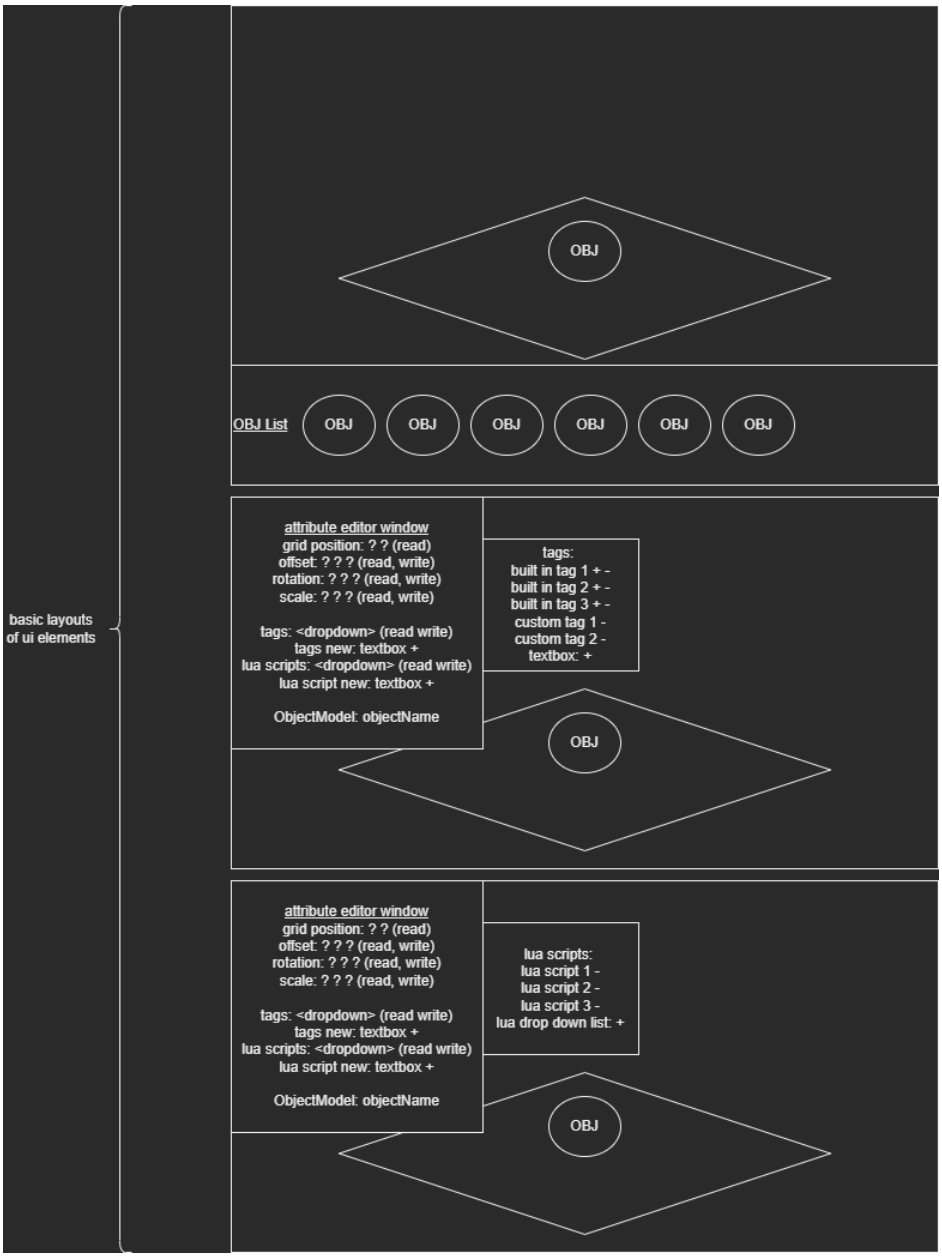


Figure 2- mock ups of UI screens