University Of Plymouth

School of Engineering,
Computing, and Mathematics

A Co-Operation Level Editor

Osbourne Laud Abraham Clark

10777267

BSc (Hons) Computer Science

Plymouth University
United Kingdom

$??^{th}$ April 2025

Acknowledgements

Abstract

Extra

Word Count:
Code Link:

Contents

List of Figures

1 introduction ¡TODO LOOK AT SOFTWAREDESIGN DOC IN REPO¿

1.1 Background

this dissertation idea came about through a problem identified in my second academic year while developing a mod for the game Co-Operation created by MINDFEAST. the issue found was the tedious nature of creating levels. once levels became big or contain a lot of objects they start to become a mess, and attempts to quickly change a single object can become a massive effort. there are multiple method used for finding and changing object positioning:

- counting rows and columns in the game
- crawling through the file until you have found the correct object
- make continuous mental logs of where things have been placed

even these with these methods, level designing and iteration becomes slow and tedious. my aim with this project is to create a visual, easy to use tool to be used along side Co-Operation making level creation more fluid.

1.2 Objectives

the objective of this project is to create an easy to use visual level editing tool for both experience and inexperienced. it will be made for individuals in the Co-Operation community wanting to create levels easier. It will be aimed at people who have little to no understanding in modding and/or YAML. more experienced modders can utilise this tool to help create levels as they wont need to manually edit YAML files. it will also open up the creativity of modding to those who wouldn't normal attempt to make one.

1.3 Deliverables

the deliverable will be a piece of software that can display a close to 1:1 level representation, while facilitating a live tile manipulation solution to be able to edit the level directly. the users will be able to export their created levels to their respective folders, as well as import existing level files so that older levels can also be easily modified with the tool.

for object manipulation features that are to be required are CUD(Create, Update, Delete) on object with in the scene, as well as be able to import object in GLB format from the correct file location scene manipulation is the zooming and panning of the camera. fo zooming the maouse wheel is used with minumum and maximum limits to zooming, and for panning, middle mouse click is used as well as the speed changin based on zoom distance.

when importing and exporting of levels it uses the yaml fomat described by the game documents. the exporting requirements are that it has the same level layout and can be played in the game, and core level features are supported for importing. this includes the main features of 2d art objects as well as 3d art objects and the structure of object definition them selves. the only feature that cant be supported without integration, of this tool into the game, would be ”mapObject” with values that aren't ”custom” since these rely on built in objects that arn't avaliable. if the object uses previous described object it woud use ”base” instead of ”mapObject” quality of life

- have the object highlight when mouse is hovering over it
- know which object you are currently interacting with
- know which object is the next for placement
- middle mouse click to select currently hovering over object for next placement(creates new object not modifies new object)
-

2 Legal, Social, Ethical and Professional Issues

3 Method Of Approach

3.1 technologies

3.1.1 unity

to create this project i utilised unity for its easy to use graphics manipulation methods. From the beginning it was clear a 3D solution was required for the project as it would allwo for easy understanding of visuals, easy cell identification, as well as allow for more intuitive use of the software. If 2D was chosen it would not have given the same feedback methods or ease of usability to the user. By going 3D i could give the user a closer idea to what the levels will look like in game.

3.1.2 C#

i was required to code within C# as this is unity primary coding language . this was beneficial as i am well versed in it and have used it before(both unity and C#) and so didnt have a learning curve of a new language.

3.1.3 Libraries

importing GLB objects

i have chosen to use UnityGLTF to import GLB object, which are compatible with Co-Operation, into unity at runtime. this also allows for hot reloading object definitions without the need of re compilation of the entire unity project meaning further object definition defining could be performed.
https://github.com/KhronosGroup/UnityGLTF

parsing YAML

YamlDotNet is used be able to import and export file in YAML format without having to write a parser from scratch. This is a library built for C# and so worked easily with unity. this made importing and export much easier as there was no need to manually parse the file.
https://github.com/aaubry/YamlDotNet

3.1.4 Co-Operation game

Through out the development the actual game Co-Operation was used to valid my progress. without the use of the game validation coudl not be made for features such as:

- true object alignment from editor to game
- exported file compatible with the game

access was gained to this during comp2003, where the client MindFeast ask for a mod to be created for their game Co-Operation.

3.1.5 GitHub

GitHub has been used for version control and storage of the project files and code. currently only two branches have been made and have now been merged, with the later branch being made more towards the end of development. while version control wasn't heavily used, it was extensively used for transporting the code.

3.2 functional requirements

functional requirements include details such as object CRUD as well as map panning and the importing and exporting of levels. these are the core vital aspects for the application to work. even with the bare bones of these features the tool would have made level creation easier in both the visual aspect as well as the object placement.

3.3 non-functional requirements

non-functional requirements where the quality of life feature like:

- changing of pan speed based on zoom distance
- the on screen hints as to the current state of selected and next placements
- quick middle mouse click to select object for next placement

3.4 UML

3.4.1 classes

3.4.2 control flow

4 project managment

5 Preparation

there are three main aspect to preparation that i took. the first two are my previous attempts at a solution when i first witnessed the issue and attempted to come up with a fix. the third is the research of the libraries to be used with in the project. this was vital as i aimed for a full 3d application, requiring positioning information out of the YAML as well as the import of the GLB objects them selves.

5.1 libraries in use

importing GLB objects

i am using UnityGLTF to import GLB object as this is a library that i have previously used and understand easily.
https://github.com/KhronosGroup/UnityGLTF

parsing YAML

to be able to import and export file in YAML format i am useing YamlDotNet. in my previous attempts i only need the name and was able to make a simple parser. in this version (the third attempt), i need positional and rotation data from the object definition and so need a more robust parser.
https://github.com/aaubry/YamlDotNet

5.2 previous attempts

the previous attempts were hacks. they were quickly made in an attempt to make the editing at that point easier. only one works, which happens to be the 2nd attempt, due to my shortcuts used used when programming win32.

5.2.1 1st attempt

add images for the first attempt
the first attempt used windows win32 header to create and show information to the users. it was entirely text based, only using buttons and input boxes, which no advance visual features. the point of its creation was to quickly create a application that could be used by the mod creation team in second year to help make levels easier. understandably it was never used as it was made after most of the levels had been developed and did see a vast amount bugs due to its two day creation. it did however attempt to tackle the issue of cell identification. by using images of all letter combinations from AA to ZZ, the application placed those images in the cell. within the game view the cells could be easily identified and then modified through the tool. the only issues with this design was that it required at least 2 windows open and the one for each cell that was getting edited and so the screen could become a mess of windows if multiple cells where getting edited at a time.

5.2.2 2nd attempt

add images for the second attempt
after the first attempt, while it was not used, testers gave feed back and with it came the second variation. the second attempt used a 2d pixel based engine called ”olcPixelGAmeEngine”. it worked through the idea that instead of the cells displaying all of its current items, it would show a colour based on if a specific item was in the cell. this was done by creating layers so each item had it own map. this attempt did not see it implementation of easier to identify cells, instead its aim was to make level editing quicker. this was done by only needing the user to find the object in a searchable list and clicking the cell with feedback though colouring telling the user the item was now in that cell. identification could have been added through displaying of the images, but at this point i was also wanting little user setup so it was more user friendly instead of requiring images to be copied into the working folder of each mod.

5.2.3 3rd attempt

the third attempt is this dissertation. its aim is to have easy identifiable cells as well as make them easily editable, merging both major features from the previous attempts into one. it was understood early on that a completely 3D solution would be best, but due to time restraint on other project this was not feasible.

6 implementation

6.1 unity

6.2 YAML

6.2.1 importing

6.2.2 exporting

6.3 Reverse Engineering of Level Layout

6.3.1 object placement

6.3.2 object orientation

6.4 cacheing for faster loading

it was found through testing that when placing new objects there was a slight lag spike. this was identified as a resource loading issue as it only occurred when placing detail heavy objects. once the issue was identified it was remedied though a cacheing system that used C# dictionary class to store already created object.

the method is as follows. when an object is about to be created it is first checked to see if it has already been created, using the files name as a look up reference in the dictionary. if it hasn't, it get imported into the scene through the GLFT importer library i am using. once in it is moved to an off screen location, this is so it cant be seen, and is then copied to be placed at the correct position where the mouse pointer.

this method has been done for both the gltf object and the images as both could cause lag spikes each time they were getting imported into the scene. by doing this, it considerably improved the speed meaning levels formed a lot quicker and are almost instant when reloading them.
see if you can get timeings for both cached loading and uncached loading

7 Evaluation

• What went well and what went badly? • Why was this the case? • To what extent was the aspect under consideration responsible (vs. other contributing factors, e.g., your own performance). • Was your experience in line with what might have been expected given the body of knowledge within the literature? • To what extent does the above cause you to reconsider the choices that you made in relation to the given aspect?

8 Conclusion

It is a brief summary of the project and its achievements. Therefore, you should relist your project's objectives and critically (and ruthlessly) evaluate whether you met the objectives

8.1 further Work