Computer Games Development
Project Report
Year IV

Patrick Donnelly
C00236160

10/11/2022

DECLARATION

Work submitted for assessment which does not include this declaration will not be assessed.

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Patrick Donnelly

Student Number(s): C00236160

Signature(s): *Patrick Donnelly*

Date: 24/04/2023

Please note:

a) * Individual declaration is required by each student for joint projects.
b) Where projects are submitted electronically, students are required to submit their student number.
c) The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.

Contents

Acknowledgements....3
Project Abstract....3
Project Introduction and/or Research Question....4
Literature Review....5
Evaluation and Discussion....7
Conclusions....20
References....24
Appendices....24

Acknowledgements

I would like to thank my supervisor Martin Harrigan for assisting me in structuring my workflow and managing my project milestones. I would also like to thank him for the advice and direction he provided throughout my project.

I would like to thank my lecturers from SETU for providing me with the materials necessary to develop my skills over my 4 years at SETU.

Project Abstract

In this Thesis, my aim was to design a No-Code Game Editor that would allow a user from a non-programming background to create game layouts without the need to write any code. Thus, removing technical skill barriers associated with traditional game design. The editor would also allow the user to save their creations without having to worry about managing data themselves.

Through the development of my Editor, I wished to gain the necessary knowledge to compare Game Development through a No-Code Game Editor with that of traditional Game Development processes, more specifically how it compares to developing a game using a Game Engine.

In order to achieve this, I choose to design the components of the editor using C++ as my foundation. To handle rendering and input I used SFML, a multi-media API designed to ease the development of games and applications. To manage game data, I chose to use YAML, as it is efficient in saving and loading data. It also has the ability to structure that data in a human-readable format, allowing me to assess the accuracy of the data being saved.

The No-Code Game Editor's audience are those from non-technical backgrounds who wish to experience game design without having to contend with existing skill barriers. Therefore, I focused on designing a simplistic user interface that a user could look at and recognise its intended function without the need for lengthy explanations or prior training.

Overall, the No-Code Game Editor provides a user-friendly and efficient way of creating game layouts whilst providing a convenient way to save data, load data and theoretically export game data through YAML files.

Project Introduction and/or Research Question

"To what extent does a No-Code game editor implemented in C++ and SFML empower users with no programming experience to create complex games, and how does it compare to traditional game development processes in terms of usability, accessibility, and flexibility?"

Level editors and content creation have become pillars within the gaming community.

Games such as "Dungeon Maker" on PC and "Super Mario Maker" on Nintendo Switch allow users to create entire levels by dragging and dropping items, enemies and other objects onto a grid.

Software like "RPG Maker Fes" on Nintendo DS or "Game Builder Garage" on Nintendo Switch allows entire games to be created using visual scripting, predefined events and entire toolkits.

The Mod community on Steam is often supported by game developers in the creation of content for their games post-release.

With these factors in mind, I wanted to create an editor that allows a user with no programming experience to create a complex game from scratch without the need to write any code. Examining how these game creations would compare to games created through conventional game development tools like Game Engines.

I achieved this by presenting the user with a simple Graphical User Interface, that allows them to create a game world by placing objects they have selected from the GUI, onto a Grid in the game world. I defined components in C++ code, processed input and rendered objects using libraries defined in SFML and stored game data from the game world using YAML.

In this report, I will discuss any relevant research that aided me in the production of the No-Code Editor. I will assess and analyse the advantages and disadvantages of development in a No-Code Game Editor compared to that of traditional development environments like Game Engines. I will cover any notable achievements I have made in developing the software that contributed to the usability, accessibility and flexibility of my editor, specifically for those from a non-technical background. I will also break down my project schedule and milestones as well as detail features, I would have liked to have implemented within the time frame of my project and beyond.

Literature Review

Stelios Xinogalos, Maya Satratzemi & Christos Malliarakis
Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment? [1]

In this paper, Xinogalos, Satratzemi and Malliarakis (2015) discuss what features their target audience identified as crucial for a successful introductory programming environment.

An introductory programming environment should engage its audience by incorporating their interests, in this case, games and mobile apps were identified [1]. A No-Code Game Editor engages its audience through the gamification of the process involved in developing a game, by allowing a user to combine components and test the results by playing their creation.

The Author's demographic insisted an ideal introductory environment would incorporate a straightforward Graphical User Interface that supports the visualisation of objects [1]. Incorporating objects and components into a GUI that a user can understand almost instantly is important for the design of a No Code Game Editor. I did this by depicting their functionality through images and labels.

Students surveyed identified the need for simple and straightforward error messages [1], as mistakes do happen regardless of how intuitive software may be. In a No-Code Game Editor, pop-ups could be used to display any errors the user may encounter as they design and create their game. I found it important to incorporate such messages on the Game save screen and text editor screen when the user attempted to save over an existing file, or to save a file that had no file name entered.

The survey participants indicated they would look for the ability to execute a program step by step in an introductory programming environment [1]. For my No-Code Game Editor, I decided to split the building process of creating a game into several phases so as to not overwhelm the user. I split the creation process into a building phase that focused on placing objects in the game world, a test phase that allowed the user to test their creation at their own pace and a save game phase where the user could name their game and save its data using a purpose-built GUI.

Toni Härkönen

ADVANTAGES AND IMPLEMENTATION OF

ENTITY-COMPONENT-SYSTEMS [2]

The author discusses the makeup of an ECS design and the separation of Entities, components and systems. Interestingly the author specifically details the modularity of an ECS design and the performance benefits that an ECS design can introduce. Härkönen (2019) points out this is due to "Data-Oriented Design (DOD), which ECS is a subset of" (p.13), and how DOD and ECS allow for cache optimisation. A process which focuses on ordering data in memory in an efficient manner, to avoid large hops while iterating through data for updates, rendering etc [2]. This kind of structure would benefit a No-Code Game Editor such as mine, as it would allow larger game worlds to be created with more objects placed in the game world with a minimum impact on performance if implemented correctly.

Härkönen (2019) discusses the "advantage of using Data-oriented composition is its modularity" (p.14). Härkönen (2019) goes on to explain how this modularity makes a project "easier to manage" "improves the readability of code" (p.19) and makes refactoring and updating code easier [2]. A No-Code Game Editor would benefit from the modularity of Data Orientated composition as it would allow game objects to be assigned behaviour at runtime, allowing multiple objects of the same type to have unique behaviour. Lastly, the shorter functions and code snippets associated with an ECS would have made managing my No-Code Game Editor easier and less time-consuming.

Evaluation and Discussion

Through the development of my No-Code Game Editor, I wished to assess the usability, accessibility and flexibility of a No-Code Game Editor compared to traditional methods of game development.

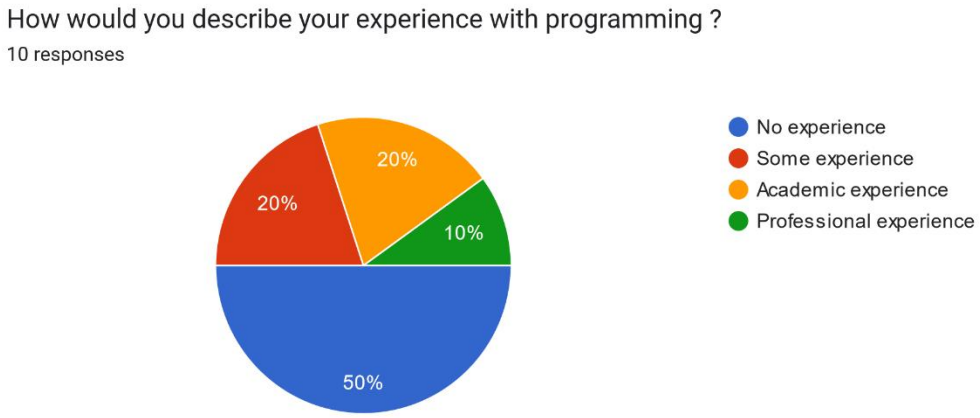
Traditionally games are developed using game engines - software frameworks that provide developers with a set of tools and libraries to help them create video games more efficiently. Although games can be created from scratch using IDEs, larger games tend to incorporate game engines to handle elements such as graphics rendering, physics simulation, sound processing, AI programming, networking, and user interface development.

Game engines are powerful tools that can help Game Developers create high-quality games efficiently and effectively while granting complete control to the developer in terms of modifying existing systems and components or creating entirely new systems and components.

In comparison, No-Code Game Editors aim to open up the development process to users who may not have experience or skills associated with traditional Game Development, by defining functionality through combinations of visual elements selected from a Graphical User Interface. In the process removing the ability to write custom scripts and create components or systems that have not been predefined in the editor by the developer of the software.

Thus there are impactful differences between Traditional Game Development Processes and No Code Game Editors in terms of Usability, Accessibility and Flexibility. I will be discussing these discrepancies based on the experience I have gained whilst developing my No-Code Game Editor, focusing on the differences between an Editor and a Game Engine specifically. I will also be analysing data from a limited playtest I undertook.

The background experience of those who participated in the playtest was as follows :



Each person that participated in the playtest was requested and helped in some cases to download an editor of their choice, Unity or Godot. A copy of my No-Code game editor was also provided. The participants were requested to use the Game Engine and the No-Code Game Editor for 10 minutes or more, unsupervised. Only one participant failed to commit 10

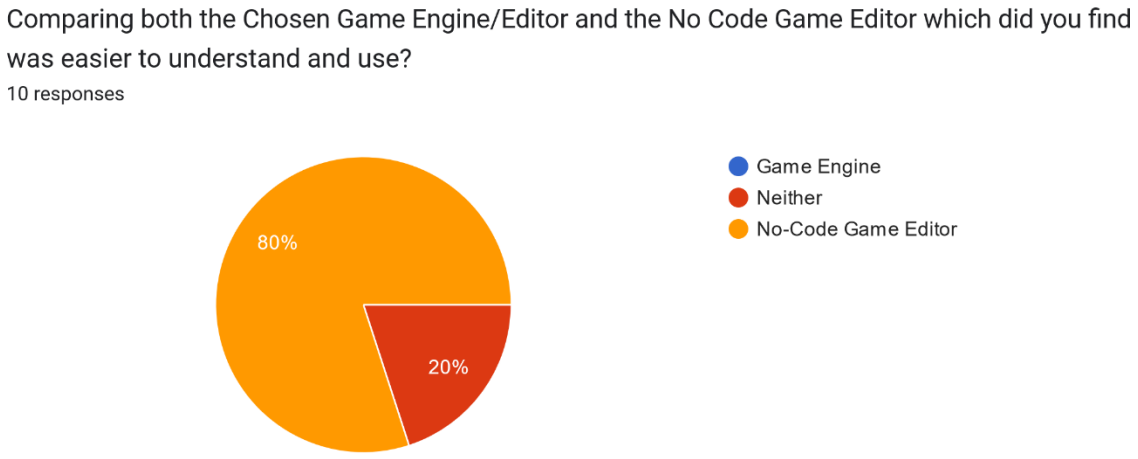
minutes or more to the Game Engine as they decided they “didn’t know how to use it”. Each participant was asked to fulfil the questionnaire, attached in the appendix, after their allocated time with the Game Engine and the No-Code Game Editor was Complete.

Usability:

No-code game editors are designed to be user-friendly and intuitive, with graphical interfaces enabling users to create games without writing code. This makes them ideal for beginners and non-technical users who want to create games quickly and easily. Game engines, on the other hand, have steeper learning curves and require knowledge of programming languages and software development tools. They would be more challenging for users who don't have experience in programming or game development.

For users with no technical skills, a no-code game editor is a more accessible and user-friendly option that allows them to create games without requiring any prior knowledge of programming or game development. For users with experience in programming and game development, a game engine provides a more powerful and flexible toolset that requires a higher degree of technical expertise but enables them to create more complex and advanced games.

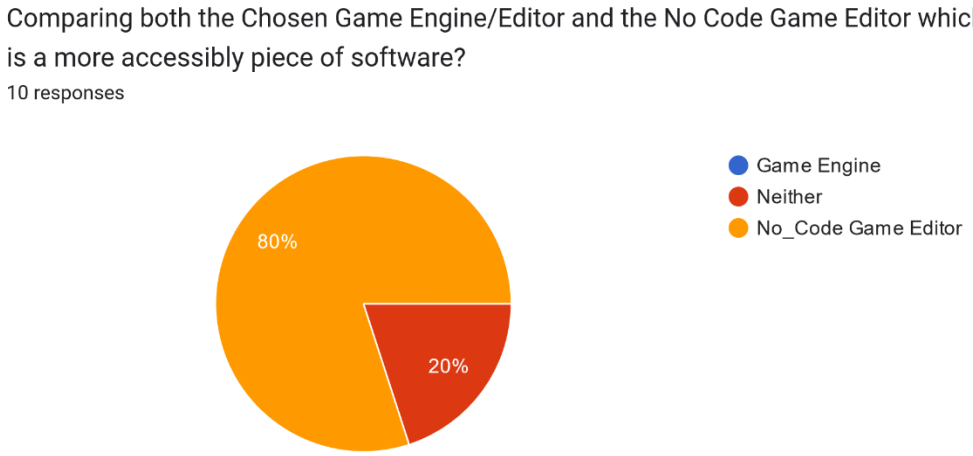
Traditional Game Development through the use of a Game Engine is better suited to large-scale games that need custom systems. No Code Game Editors would be better suited to tackling specific genres so that systems and game components traditionally associated with those genres can be focused on and refined to the point that a user would be able to develop a unique game that fits that genre without feeling like their vision was limited by the editors defined presets.



According to those surveyed 80% of individuals deemed the No Code Game Editor to be easier to understand and use than the Game Engines they worked with. 20% of those surveyed deemed that neither the Game Engine nor the No-Code Game Editor was easy to use or understand. This result was in direct correlation with the result I discuss in detail in the accessibility section below, as the participants answered the same way when analysing the accessibility of No-Code Game Editors and Game Engines.

Accessibility:

For users with no technical skills, a no-code game editor is more accessible for Game Development as it requires very little or no technical knowledge. Creating functionality in your game depends solely on your ability to logically link components programmed by the developer of the editor. This would normally be a straightforward process accommodated through a simple user interface and visual elements. Game engines by design are less accessible as they require some level of programming knowledge to create a finished game, automatically introducing a steeper learning curve to overcome. One which may take years of training to fully master.



The above pie chart indicated that 80% of the surveyed users found the No-Code game Editor to be more accessible. 20% found neither to be more accessible than the other. One such participant who identified themselves as having Academic experience in programming, commented that both the game engine and the No-Code Game Editor presented barriers to accessibility. This user specifically commented “The UI was straightforward forward but there was no system or prompts in place to guide the player in what they were aiming to achieve. It lacks direction and guidance which would be intimidating for someone who has no experience with game development.” Another user who described themselves as inexperienced when it came to programming and gaming in general commented “I didn’t know what to do in either” [Game Engine or the Editor] and decided that Neither option was accessible in their opinion.

Overall the response was positive with 80% describing the No-Code Game editor as more accessible, one user, described as having no programming experience commented on the lack of gameplay “Nothing moved when I tried to play my game” but this did not affect their decision when it came to accessibility. The same user simply answered “I stopped playing Unity after a couple of minutes. I didn’t know how to do anything.”

Flexibility:

Game engines typically offer more flexibility than no-code game editors, as they provide developers with full control over the underlying code and allow them to customize every aspect of the games they create. This makes game engines ideal for developers who want to create complex and highly customised games. No-code game editors, on the other hand, have limitations on what can be customised or modified and are more suitable for simpler games or prototypes. This is down to the fact that no code game editors rely on every system, entity and component being predefined. Although there is flexibility in terms of how these components can interact with one another, their flexibility is predetermined by the developer of the editor and their intentions on how these elements can interact.

For users with no technical skills, a no-code game editor provides a simplified approach to game development that doesn't require any coding knowledge. These users can create games using pre-built components and templates, which limits the amount of flexibility they have over the final product. For users with experience in programming and game development, a game engine provides the flexibility to implement custom functionality and optimize game performance to a higher degree.

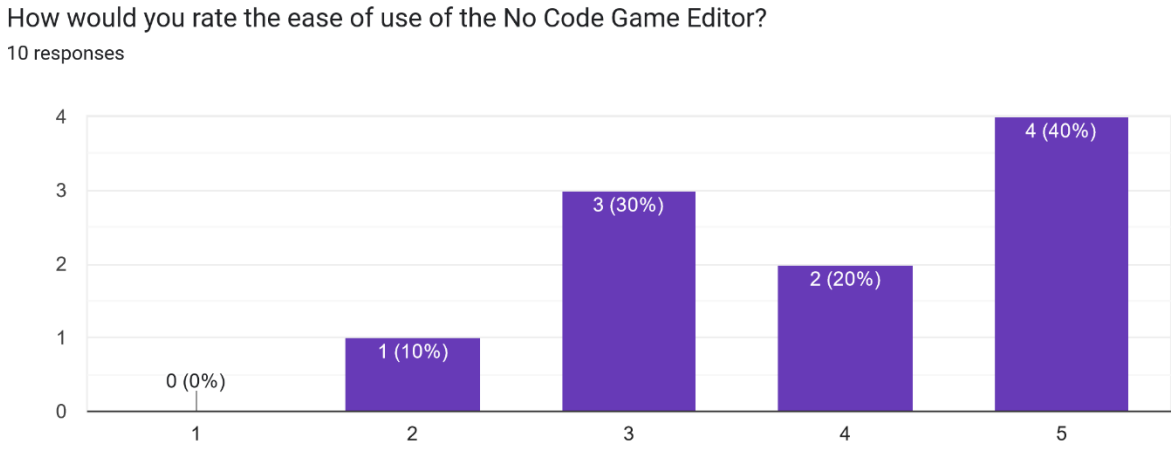
A no-code game editor could in theory offer great flexibility but at the cost of increased development time for the developer and a steeper learning curve for the user. Although the flexibility that can be achieved in a no-code game editor depends on the developer's implementation and design, it is undeniable that a no-code game editor can offer in-depth levels of flexibility. In saying this, the flexibility of a no-code game editor cannot exceed the flexibility of Game Engines where a developer has free reign over the creation of game mechanics, systems and elements through the implementation of custom code.

I have chosen not to include the results of the flexibility question as the majority of the people surveyed had little to no programming experience (80%). I think due to the lack of experience of the demographic surveyed the results were unfairly biased, as they do not have the necessary experience to compare and analyse flexibility in terms of Games Development.

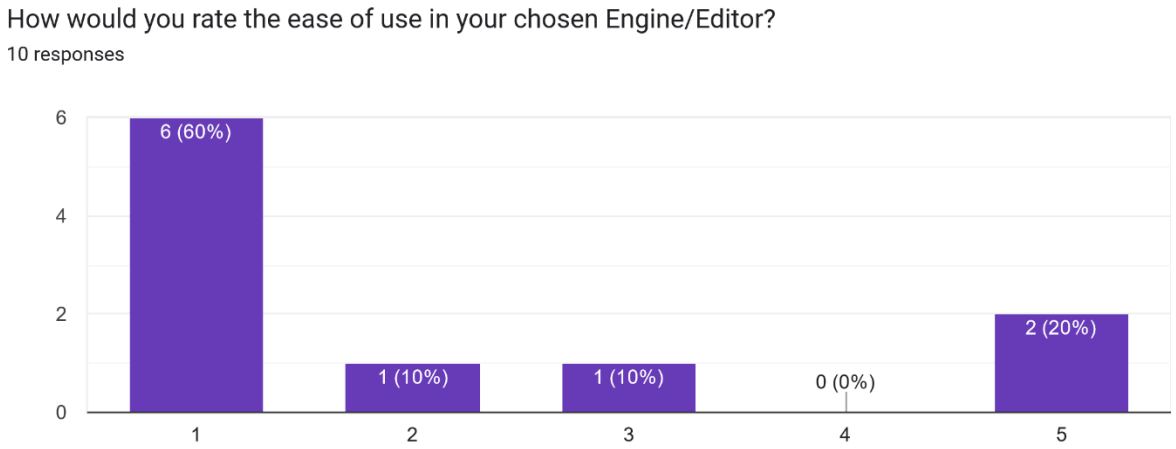
Conclusion:

No-code game editors are suitable for creating simple games or prototypes quickly and easily, without requiring extensive technical knowledge or programming experience. They are ideal for users who want to create games as a hobby or to take their first step into game development. Game engines, on the other hand, are suitable for creating more complex and advanced games that require custom functionality, optimisation, and high-performance graphics. They are ideal for users who want to create professional-grade games or pursue a career in game development.

The below graphs range from a value of 1 “Complex - I made no progress and did not know what I was doing” and 5 “Straight forward - made progress in adding objects to a scene”.



The No-Code Game Editor generally received positive feedback regarding its ease of use, with only one individual vying on the side of it being too complex. 40% deemed it to be straightforward however 10% of that figure had prior academic experience in programming. Interestingly 30% of users found the ease of use of the Editor to be down the middle between complex and straightforward. This 30% was made up of individuals who described themselves as having some experience to professional experience. As explained earlier in the paper one individual described the software as lacking in terms of direction and guidance.



In comparison, only 20% of the individuals surveyed agreed that the Game engine was straightforward to use. However, this consisted of individuals with academic and professional experience.
Every user that identified themselves as having no experience with programming prior to this survey described the Game engine as too complex to use. Most highlighted they did not know what to do. In general, the Game Engine was deemed too complex by those with no prior experience with programming.

In summary, the data shows a No-Code Game Editor provides a simpler and more accessible approach to game development that is suitable for users with no technical skills, while a game engine provides a more powerful and flexible toolset that requires a higher degree of technical expertise but enables users to create more complex and advanced games. Ultimately, the choice between a no-code game editor and a game engine depends on the specific requirements and skillset of the user.

Project Milestones

Other project commitments are the reason for the discrepancies in my sprint scheduling, particularly in February when all my time was dedicated to a month-long group project.

Sprint 1:

Date: 25/11/2022 TO 01 /12/2022

Objectives:

- Create a Player.
- Start to Check Collision.
- Start Check Room Validity.
- Create a grid that can be resized.

Sprint 2:

Date: 02/12/2022 TO 09 /12/2022

Objectives:

- Create Player movement.
- Create and assign Colliders to game objects.
- Create a method to validate rooms.
- Create buttons to use as UI in the game.
- Create a method to clear the grid of all objects.
- Create a method to assign colliders to sets of walls instead of individual wall tiles.
- Create a class that handles collision between game objects (AABB collision).
- Handle collision between player and walls.
- Handle collision between player and obstacles.
- Animate the player.
- Create a method to place floor tiles automatically in a valid room.
- Display to the user which wall tiles are invalid when they try to generate an invalid room.
- Create a class for obstacles and allow the user to place them on the grid.
- Give the player an attack.
- Check the collision between the player’s weapon and obstacles and destroy obstacles on contact.

Sprint 3:

Date: 13/01/2023 TO 19 /01/2023

Objectives:

- Tag system.
- Level loading and saving. Investigate file formats, and third-party libraries (YAML, CSV, JSON etc.)
- Add buttons to the UI for changing grid size, items, terrain, decorations etc.
- Update Player Collision with other objects.
- Add the ability to delete placed objects.
- Add Game States.
- Allow cleared cells to have objects placed in them again.
- Add a build phase and test phase.

Sprint 4:

Date: 20/01/2023 TO 27 /01/2023

Objectives:

- Expand on the palette of entities you can place.
- Reset object positions when going from the test phase back to the build phase.
- Allow grid and colliders to be toggled on and off. Create UI buttons to handle these functions.
- Create Ui Buttons for each object type and each object that can be placed.
- Manage how placed objects are stored.
- Automate the UI for placeable objects so sprites placed in folders in the project repo can be read automatically and have buttons assigned automatically as part of the UI.
- Create Categories of objects and create tab states to keep track of which category is selected for object placement.
- Create rows of selectable objects and create UI buttons to navigate through rows of objects.

Sprint 5:

Date: 27/01/2023 TO 03 /02/2023

Objectives:

- Create a dialogue box.
- Allow text to be read in from the user and displayed inside the dialogue box.
- Allow text to be deleted and removed from the dialogue box at run time.

Sprint 6:
Date: 03/03/2023 TO 09 /03/2023

Objectives:

- Create a text editor GUI for the input and output of text.
- Wrap text so that it fits the dialogue box regardless of length.

Sprint 7:

Date: 10/03/2023 TO 16/03/2023

Objectives:

- Create an inspector to attach pieces of text to triggers. Decide on what the GUI for the inspector will look like.
- Finish GUI for the text editor.
- Create an input field class to handle the input text for the body of the dialogue and the name (title) of the file.
- Allow the text to be saved to a text file with the text from the body being the dialogue to display and the text from the header being the name of the saved file.
- Allow saved text files to be loaded in.
- Allow loaded text files to be edited.
- Display all saved text files in a list to the side of the editor.
- Update the list every time a text file is saved.
- Create pop-ups to make the user aware if a file exists if they try to overwrite it.
- Create a button that will clear the text in the input fields.

Sprint 8:

Date: 17/03/2023 TO 23 /03/2023

Objectives:

- Allow text files to be completely deleted from inside the editor.
- Create a checkbox class. In this instance use a checkbox to preview any input text from the text editor in a dialogue box. Toggle on and off.
- Add multiline input fields.
- Create the GUI for an inspector.

Sprint 9:

Date: 24/03/2023 TO 30/03/2023

Objectives:

- Split placeable objects into their own classes.
- Create a texture manager to manage the textures being loaded in, to prevent duplicate textures in memory.
- Create a font manager.
- Allow an object to be selected by double-clicking.
- Display the inspector of a selected object.
- Allow objects to be relocated once selected.
- Create a GUI that will allow dialogue to be attached in the inspector.

Sprint 10:

Date: 31/03/2023 TO 06 /04/2023

Objectives:

- Create a drop-down menu for the inspector.
- Create an in-range condition that can be set in the inspector that will trigger dialogue if the player is within range of an NPC that has this enabled.
- Create an interacted condition that can be set in the inspector that if enabled will trigger dialogue any time the player interacts with an NPC.
- Allow dialogue to be loaded in and saved to a selected object.
- Add a main menu.
- Install YAML.
- Save Data to a YAML File
- Load data from a YAML File
- Add the ability to scroll the screen based on the mouse position.

Sprint 11:

Date: 07/04/2023 TO 13 /04/2023

Objectives:

- Increase Grid Size.
- Update YAML to save data for the increased grid size if necessary.
- Create a separate view for the game scene and User Interface.
- Add zoom-in and zoom-out functionality.
- Anchor UI regardless of game world position.
- Add functionality to highlight an area and fill that highlighted area with the currently selected object (multi-object placement).
- Create a save game GUI where the user can enter their game name.
- Create folders at runtime to store all game data.
- Create pop-ups for when the user is about to overwrite existing game data, when a game is saved successfully and when they try to save a game with no name entered.
- Add a colour picker so the user can customise their game names colour for display on a game menu at a later time.
- Add text formatting like italics and underlining for display on a game menu at a later time.

Sprint 12:

Date: 14/04/2023 TO 20 /04/2023

Objectives:

- Update data being saved in YAML files.
- Update the Game list at runtime every time a new game is saved.
- Make the view follow the player's movement.
- Allow colliders to be placed independently of objects.
- Allow terrain to be deleted.
- Allow multi-object deletion.
- Adjust UI.
- Various bug fixes.
- Develop a website to showcase the project.
- Create and edit a Video to showcase the features of the project.

Major Technical Achievements

So far, I have managed to create a No-Code Game Editor that allows the user to build, save and load their games without writing a single line of code. This is achieved through a friendly graphical user interface and pre-programmed game components and entities that the user can click on in the GUI and place onto a grid in the Game View.

Game Building:

Users can create a game world from the ground up. Various types of objects such as terrain, walls, enemies, items and more can be placed, to bring the user's vision to life.

Game Saving:

Users can save their game as they build, resuming the build at any time they wish.
Data is stored in directories containing unique YAML files that are created at runtime.

Game Loading:

Users can load their creations to continue editing or to play through their creations at the click of a button.

Built-in text editor:

Users can create dialogue scripts within the built-in Text Editor. These scripts can be attached to NPCS and Enemies through the inspector to create narrative experiences in their games.

Dynamic Loading:

User-created dialogue and games are instantly assigned UI buttons so that the player can access their files as soon as they are saved without having to restart the software.

Sprites placed in the Project Directory are automatically loaded in and given Buttons in the GUI so that instances of those sprites can be placed on the grid as objects.

QoL Features:

Features such as multi-object placement and deletion exist to make the user experience in building games as quick as possible. Users can place or delete multiple objects by highlighting an area in the Game View. Pop-ups will display in areas where the user might risk losing or overwriting data like on the Save Game screen.

Project Review

Although I am happy with the progress I have made to date on the No-Code Game Editor I freely acknowledge that there is much more work to be done to make it live up to the experience I envisioned.

The editor itself in terms of design and user interface has come together well. Saving and loading Game Data works as intended. The text editor and its functionality to create and save dialogue files work well. There are also many quality-of-life features that streamline the creation process such as multi-object placement and deletion, game world scrolling in the editor and the ability to zoom in and zoom out whilst creating your game.

I however acknowledge that it lacks essential features that I had planned to implement but due to time constraints fell beyond the scope of the project.

In hindsight, I would have liked to have used an external library such as “ImGui” to implement the UI, and I would have pushed harder for permission to do so. Not only would this have perfectly suited my vision for simple intuitive UI design, but it also would have allowed me to spend time developing gameplay features that would have enhanced the creative experience of my editor and greatly increased the quality of the products users could design within my editor. As things stand it would be fair to say that approximately a third (possibly more) of my project time was spent on creating a simple User Interface that functioned as expected from scratch, as each time a new feature was introduced, UI elements had to be created to accommodate that feature. Time, I think, could have been better spent focusing on gameplay elements that the user could incorporate into their games.

I also think I should have increased my initial research phase before I began implementing the code. I believe if I had done so I would have realised an ECS design would have perfectly fitted my project. I think this would have allowed my editor to be more flexible and efficient. As well as this I think implementing an ECS design would have cut development time and made the structure of my written code easier to navigate and behaviour more readily modifiable at runtime.

In retrospect, I should have spent more time researching basic game engines as I believe a No-Code Game Editor shares similar methodologies in terms of its aesthetic and how certain functionality could be implemented. Examples of this would be UUIDS and Serialisation.

I would have liked to have given my research and design phase greater focus as I may have been able to better foresee some of the delays and many complications I encountered.

I adopted a can-do attitude throughout the process that I think worked to my detriment in the long run. I allowed the scope of my project to go beyond the realities of what could be achieved within the allocated timeframe, possibly not fully understanding the measure of time I would have to invest in projects that ran in tandem with this one. I focused on features that I now see were less important to the overall project but took a lot of time to develop like the text editor. Although this is a neat feature it should not have taken priority over gameplay mechanics especially since it took a month to develop without altering the overall experience of the software much. Gameplay mechanics like enemy behaviour, weapons and unique items would have made more of an impact with a shorter turnover time in terms of development.

Conclusions

From my work, I can deduce that it is possible to make a flexible and accessible No-Code Game Editor using C++ and SFML. In doing so one must consider the rendering limitations of the library and the cost of rendering objects on the screen as that has the biggest negative impact on performance if handled incorrectly.

Creating a texture manager to limit duplicate textures being loaded greatly improved performance and reduced processing times when multiple objects were placed at the same time.

Batching multiple objects into a single draw call reduced the number of draw calls and increased performance.

Culling objects beyond screen space by simply not rendering them if they are not within the view increases performance.

Instancing by rendering multiple instances of the same object using a single call reduced rendering costs and again increased performance and reduced lag.

Other methods like Texture Atlasing could be used to reduce the number of textures being loaded and stored, although I did not incorporate that in my editor.

Some of the same principles can be applied to updating objects such as culling. Keeping track of whether an object is within the current view or not and updating it accordingly would greatly increase the efficiency of games created in the editor, especially when there are many updateable objects placed in the game world.

From the creation side, I have managed to streamline the process reducing any visible lag and inefficiencies.

I can confidently conclude that in theory C++ and SFML can be used to create a flexible, accessible and complex No-Code Game Editor for users from a non-technical background. Once rendering, audio and input are handled efficiently the incorporation of SFML will have no or very little negative impact on gameplay performance, that performance would rely on the game logic being designed in an efficient manner in C++ code as well as update calls being handled and optimised in an appropriate manner.

In terms of how a No-Code Game Editor compares to traditional development tools like Game Engines, I can conclude in terms of flexibility a Game Engine excels in that department due to the ability to create custom code that allows infinite possibilities in terms of systems and gameplay mechanics. No-Code Game Editors are limited by the implementation of their components and systems.

In terms of Usability, both No-Code Game Editors and Game Engines have their advantages. No-Code Game Editors are better suited for those coming from non-technical backgrounds who wish to focus on smaller projects or prototyping, however, complex games are definitely possible depending on the tools made available within the editor.

In terms of Accessibility No-Code Game Editors have no skill barriers and shorter learning curves in comparison to those you would associate with traditional Game Development processes and offer a better entry point for those wishing to experience game development for the first time.

To finish, when taking on a project like this it is necessary to manage the scope of the project effectively and to research design and implementation methods for No-Code Game Editors thoroughly. Depending on the time frame allocated to the development of the editor it may be important to focus on creating an editor that implements gameplay mechanics and design principles associated with a specific genre of game in order to keep the scope of the project reasonable.

Future Work

I have a lot of ideas I would like to build on in relation to my No-Code Game Editor, some of which are implemented to a certain degree already, albeit still a work in progress. I think with further development this No-Code Game Editor could be a robust piece of software that could be used to showcase the flexibility of a No-Code Game Editor.

Entity Component System:
An Entity Component System (ECS) would add extra flexibility, scalability and reusability to a No-Code Game Editor. ECS allows users to create and customize game objects and entities using various components, rather than relying on pre-built objects. It provides a scalable way to manage large amounts of data and behaviour, which can make it easier for users to create complex games without needing to understand complex programming concepts. By defining reusable components that can be used across multiple entities or systems, users can create complex games without needing to write custom code.

Serialisation:
Serialisation is a mechanic often implemented in game engines. However, I think it would have its benefits inside a No-Code Game editor. Serialising a game's objects would allow a user to test their creations thoroughly by allowing changes to be made to entities during a test phase and reverting them once the test phase has ended. They could experiment with collision, destroy objects, lose health, gain coins, interact with NPCs etc. in the test phase and have those changes reversed to their original state once they exit the test phase.

Dynamic Behaviour:
I would have liked to have implemented the ability to add unique behaviour to Entities. The ability to determine an enemy's behaviour at run time would have allowed users to assign different types of AI behaviour to them such as the ability to pursue the player, find a path to the player or even flee. Assigning entities with unique behaviour, stats and weapon load-outs would greatly increase the customisation options available. This could be achieved through an ECS or design patterns like the decorator pattern.

Multiple Grid Layers:
Allowing the user to add multiple layers to their Game would allow them to create bigger game worlds. They could separate interiors from an overworld layer to manage their game world more efficiently.

Individual Menu Customisation:
Giving the player the ability to customise a menu for their game would add to the game feel and the quality of the game a user creates.

Dynamic Audio:
Allowing the user to customise audio on various aspects of the games they create such as background tracks and overworld sounds would help with immersion and the uniqueness of a game. It would also serve as a nice introduction to sound design, a key element in games, that increases a game's emersion value.

Gameplay Mechanics:
Implementing more weapon types or interactive objects like keys, power-ups and switches would allow for creative scenarios unique to each game. The more gameplay mechanics available within a No-Code Game Editor the more diverse and flexible it can be.

Tutorial System:
A tutorial system would really benefit an editor like this, as the goal is to introduce a non-technical user to game design without introducing skill barriers. A system which explains components through pop-ups and prompts as the player navigates the software would be appropriate for software like this.

References

[1] Xinogalos, S., Satratzemi, M. & Malliarakis, C. Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment?. *Educ Inf Technol* **22**, 145–176 (2017). https://doi.org/10.1007/s10639-015-9433-1

[2] Toni Härkönen, ADVANTAGES AND IMPLEMENTATION OF ENTITY-COMPONENT-SYSTEMS (April 2019).

https://trepo.tuni.fi/bitstream/handle/123456789/27593/H%C3%A4rk%C3%B6nen.pdf?seq

Appendices

Play Testing Survey:
https://docs.google.com/forms/d/1Nhc0aXAc6I51Lje7Pndh39_SR8kW7Vnum_2pcFtYdlw/edit