Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Computer Games Development Project Report Year IV

[Seán Whelan]
[C00250016]
[26-04-2023]

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) SEÁN WHELAN

Student Number(s):	C00250016
Signature(s):	Seán Whelan
Date:	26 April 2023

Contents

Acknowledgements

I would like to thank the following people who assisted in completing this project including;

Martin Harrigan, who was my project supervisor throughout the development cycle.

I would also like to thank all of my lecturers for helping develop my programming and game development abilities throughout the years.

Project Abstract

The main goal of my project is to create a no-code game editor which allows users without any coding experience to create a 2d, top-down, "shootemup" game. In the landscape of today's gaming industry allowing users to create their own game modes and environments seems essential to the continued popularity of a game. With the industry shifting further every week towards a game's as a "live service" model, the expectation of players is a consistent amount of new content. Very few developers can create content which they can put their name behind fast enough to meet this demand. This is where a game editor comes into play. Outsource the creation and distribution of new content to the players themselves. This has proven to be a great addition to many of the major games which have tried it.

Project Introduction and Research Question

"Is it possible to create a 2D top-down shootemup game editor which has no code involved whatsoever?"

"Will it be easy to use with no experience in Game Development and no tutorial?"

"Will the editor allow games created to vary enough to allow for creation of a multitude of games each with their own novel gameplay while still being powerful in that specific area?"

"How does no code development stack up to a traditional learning curve of code-based development?"

In the last couple of years most games which have had continued success and a thriving player base beyond the initial launch phase of the game have had this to some capacity. Roblox, Fortnite, Minecraft and GTA 5 are the best examples of this. Halo Infinite for example, launched without this and while it saw a massive player base initially, this dwindled off as 343 Industries were unable to add new content fast enough. Now they have released their in-game editor "Forge Mode" and the player base is beginning to return.

In my opinion, having some form of level editor in a lot of games, is near essential in order to sustain a large player base that keeps coming back for more. Furthermore, how your players are able to distribute these levels to each other is vital.

No game has gone further in this regard than Fortnite Battle Royale, and low and behold consistently one of the most played games in world since its release 5 years ago. Not just does Fortnite have a game editor called "Creative" mode which allows you to create your own maps and minigames, but it also has a way to get your game viewed by the masses, distributed to them and played. Lastly, Fortnite incorporates referral codes called “creator codes” which allow the builders of these game modes to get paid. A player who plays their game, will see their creator code and opt to put in in their item shop, then when they buy something from the shop, a portion (5%) of that money goes to the creator. This is genius as it incentivises creating content more than even the enjoyment of being creative, but also the excitement of others enjoying your game and lastly the bonus of a pay check for doing a good job.

While it’s good for a game editor to have options, as it allows the player/creator to be as creative as they care to be, it’s also essential for an editor to have clear barriers to what can’t be done and a direction/ use case which the editor is for.

In this instance the use case of my game editor is specifically to create 2d, top-down, “shootemups”. This ensures that people who come looking to create that, won’t be overwhelmed by a million things that don’t relate to what they are specifically trying to create. More options are good to a point, where it crosses over and makes you less likely to create.

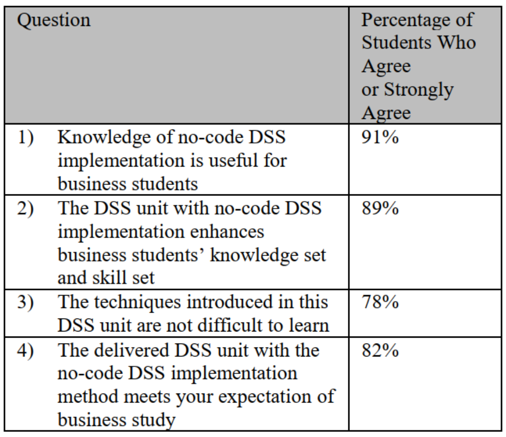
The 2 main focuses of my editor are a smooth level designer and an accessible UI. The level designer allows you to create the world you want (within reason) that your game takes place in. An easy to use user interface is essential to the editor as players will only stay creating if their user experience is enjoyable.

Literature Review

“Improving Student Performance by Introducing a No-code Approach: A Course Unit of Decision Support Systems”

[1] (Hai Wang and Shouhong Wang 2022)

In recent years no code tools have been gaining traction in educational settings as a way to teach both coding and overarching concepts that relate to it. This is primarily useful for students who may not come from a computer science background but who are capable to understanding the material if the initial barrier of not understanding code is taken away. An example of this which I found particularly pertinent to my project is a paper titled “Improving Student Performance by Introducing a No-code Approach: A Course Unit of Decision Support Systems” which was written by Hai Wang and Shouhong Wang (2022). The course which the authors experimented with was a decision support systems (DSS) college course. In the conclusion they noted how the students which completed the no-code unit had overall higher grades and in their opinion a better understanding of the DSS concepts which were being taught than those who had not completed that unit.



Question	Percentage of Students Who Agree or Strongly Agree
1) Knowledge of no-code DSS implementation is useful for business students	91%
2) The DSS unit with no-code DSS implementation enhances business students' knowledge set and skill set	89%
3) The techniques introduced in this DSS unit are not difficult to learn	78%
4) The delivered DSS unit with the no-code DSS implementation method meets your expectation of business study	82%

Table 2. Summary of Students' Feedback

Business students feedback studying DSS on how they felt about the no-code unit.

This is particularly relevant to SlayerMaker as a no-code game editor as in simple terms it should make the development process easier for the end user than achieving the same goal with traditional programming. This is amplified by the fact that SlayerMaker is targeted primarily at the target audience which gets the most benefit from no-code tools, people who do not come from a computer science background.

"No-Code Video Game Development Using Unity and Playmaker"

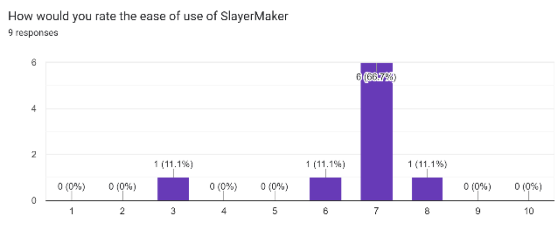
[2] (Mike Kelley 2016)

The book "No-Code Video Game Development Using Unity and Playmaker" by Mike Kelley covers various aspects of no code development. In particular it focuses on using Unity and Playmaker to achieve this but much of the concepts are not subjective to just this environment and actually are very useful when it comes to understanding no-code game development as a whole. In particular for SlayerMaker Chapter 7 is pertinent as it focuses primarily on user interfaces (UI) again, in Unity. The chapter is a UI 101 class which involves step-by-step instructions on creating various UI elements, some more useful than others. These include panels, text, images, buttons, modal windows and animations. The author also goes into detail about best practices in regard to UI design. These included ensuring UI elements are easy to interact with, using appropriate font sizes and colours and keeping the layout simple and consistent. The chapter lastly covers how to make user interfaces more responsive as well as making them adaptable to different screen sizes and resolutions.

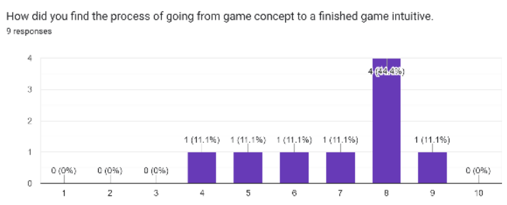
As SlayerMaker allows the user to write no code at all the design principles and best practices that are discussed in Chapter 7 will be useful in helping create a user-friendly UI that makes the entire application easy to use and to navigate. Particularly when it comes to starting with a concept of a game, and seeing that through the entire development process all the way until the end and creation of said game. As the GUI is the number one thing the user will be interacting with, it is pertinent to the user experience (UX) of SlayerMaker that the UI follows these principles and practices, in particular for the target audience of non-technical people with the lack of programming or game development experience.

Evaluation and Discussion

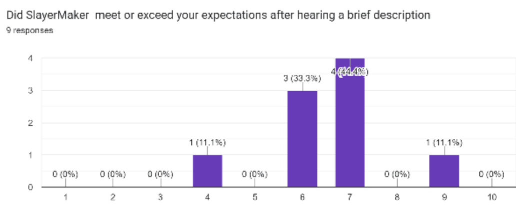
I had a good few of my friends who are not in our course use SlayerMaker towards the back half of the development cycle, when most of the functionality was implemented to some degree. My friends work and study jobs and courses in a wide variety of fields, none of which were coding or gaming related. Overall, I got 9 responses to the google form which was quite positive, given I asked around 12 people total. Additionally, I have since added the link to answer the google form to the readme for SlayerMaker.



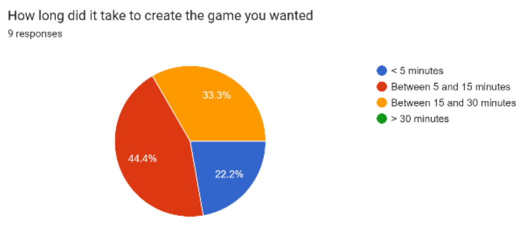
There was a small bit of spread but the response to ease of use was mostly positive



There was a wider spread for intuitive process but the response was mostly positive with 8 being the median.



The deviation for exceeding expectations was mixed, but majorly positive. In retrospect, the brief description I gave was not exactly the same and therefore could have affected the data.



It took no one more than 30 minutes to create a game. Other than that, a good spread.

Overall the responses were positive. It seems the UI was relatively intuitive. Additionally, in comparison to traditional code-based techniques the length of time it takes to create a game is night and day. While I do think this is good evidence that the existence of no-code options is a good thing it isn't the be all and end all. Game Engines and IDEs exist for a reason, they can build a much broader type of game.

Project Milestones

*While I did not have any fixed milestones from the start my supervisor and I regularly came up with what was worth working on next at every project meeting. On the weeks where our meeting couldn't go ahead, I came up with them myself. I feel we followed a logical process when deciding what to work on each week which resulted in small one- or two-week sprints, regular commits and finished features, similar to an agile development process. This means that most of my features which were small and manageable in size were achieved on time. Please note, some gaps exist for Halloween, Christmas and parts of ARGO.

Feature / Milestone / Component	Date to be completed	Achieved on time?
Refined the proposal. Narrowed the scope to two aspects of a no-code editor: Map Creator and User Interface	21 October	Y
Simple version of wall placement	28 October	Y
Click and place walls	28 October	Y
Simple Menu	14 November	Y
State Management	14 November	Y
Research Document Draft	14 November	Y
Other wall placement forms, brushing multiple cells	25 November	N
Path from high to low level game creation: Stages	25 November	N
Finish better menu	25 November	Y
Cell and Grid Rewrite	25 November	Y
Literature Review Draft	30 November	Y
Finish Multiple Brush types	10 December	Y
Finish Stages of game creation from high to low level. Modular, can be added to	10 December	Y
SRS Feature list at current time	20 January	Y

Wall Rewrite	20 January	N
Rubber tool for removing placements	20 January	Y
Simple Player	27 January	Y
Test / Demo Level functionality	27 January	Y
Fixed Toolbar UI	10 February	N
Choice of sprites or walls, curated set	10 February	Y
Enemies, Spawn points, weapon for player	10 February	Y
Wall change again, more efficient	14 March	Y
Add crosshair	14 March	Y
Add shooting towards crosshair	14 March	Y
Enemy Spawners implementation	14 March	Y
Removal / Deletion of spawners with rubber	24 March	Y
Enemy implementation	24 March	Y
Saving to CSV for walls and Enemies	24 March	Y
Loading from CSV to create Games	24 March	Y
Choose which game to play	24 March	Y
Blood Splatter when enemies shot	24 March	Y
Goal/ Sub goal design. High level game types Possible options	31 March	Y
Game types fully implemented	7 April	Y
Ui for game play	7 April	Y
Customisation options	7 April	Y
Items to make games more varied	14 April	Y
Powerups to make the user feel powerful	14 April	Y
Simple network manager to move csv's to database	21 April	Y
Fill tool	21 April	Y
Finished Documentation	26 April	Y
Finished Network Manager	26 April	Y
Simple Sound Manager	26 April	N

Major Technical Achievements

I believe I had a good few major technical achievements during the development of SlayerMaker, all of which contributed, to a great end product.

Game Creation, Saving, Loading:

A user can, with no development experience create a game from start to finish, this is done in an intuitive way, starting with a higher-level look at what the game will be all the way to low level, small changes. Upon finishing this the user can then save their creations to a file. When returning to the play screen a dynamically created set of buttons for the control of that game will now exist, without ever restarting the program. When clicking play on a game, the file which contains that game is loaded efficiently and the game begins. Additionally, the user interface will dynamically change to other changes to the screen. The delete button which deletes a game file will change the UI, the same for downloaded files from a database.

Game Objects:

The way objects are handled in SlayerMaker involves using dynamically created vectors of unique pointers to objects. This is an extremely efficient and useful way to work with so many objects. Additionally, the textures which these game objects use are loaded in using a texture manager just once, these are then saved to a map and reused by each object instead of being reloaded. All of this contributes to a really smooth experience when placing and using objects in the creation process of the game, something which I think makes the user experiencer really enjoyable.

Network Manager:

During my Internship I used a fair bit of SQL for database interactions, all be it in a very different tech stack. During this project I tasked myself with being able to implement something similar. I settled on using SQLite as it seemed lightweight. I managed in the end to get the CRU principles out of CRUD (Create, Read, Update, Delete) when interacting with the database included in the end product.

Game Variety:

Both my supervisor and I agreed early on that in order to make the editor easy to use a curated list of items, enemies', walls, and other game objects made sense. The main concern with making this decision was we would sacrifice the diversity of the shootemups that could be made in favour of the editor being easy to use. However, this is not the case. For how simple the editor is to use the variance in end product is rather surprising.

Project Review

I think there was a lot that went right with my project. I think consistently keeping on top of work was why I get to say that. Additionally, the final product, while lacking some polish, still achieves the major things I set out to achieve. When I started I had the ambition of creating a no code game editor, I wanted it to be easy to use and dynamic, I wanted it to be able to save and load from a file, and I wanted all of the games created to have their own novel gameplay. I believe all of this has been done to a great standard and more. I tested the project on some of my non-technical friends towards the back end of the development process and it seemed like everyone could achieve what they wanted in the game they created.

I think a few things went wrong, although there's probably some recency bias, so I may have forgot some of the earlier things. I was unable to add sound for the games section of the editor because the lab PC just will not give me an audio output. My PC broke during the development process too. This meant I lost about 1.5 weeks of working time. Something I tried was to keep the grid for the gameplay as well and have the enemies use a search algorithm to path find instead of just directly seeking. However, I felt that to have it work with the number of enemies I wanted to be able to spawn (to make the games really be shootemups) the algorithm would have to be extremely efficient. I felt this would be massively too time consuming if possible at all and was probably out of the scope of the project. I couldn't get the animation class I wrote to work as well for some reason. I need to debug it more. Lastly Python is installed twice (different versions) from the windows store on the lab PC. Which means you cannot delete them as we do not have access to the windows store. So, I could not configure a newer version of Python without them conflicting with each other. I was originally going to use Python and Anvil from last year instead of SQLite for the server actions

Sound is still missing, although the files are there, and the sound manager is set up. I wanted to have the build button work, which would instantiate the compiler again at run time and create an exe of the specific game, creating a separate folder for this and then pulling in all the necessary assets. I could not get this to work in time, all the code is still there but the created exe has errors.

If starting again I would look into setting up an Entity Component System. This seems like a good approach but I started looking into it much too late into the development process. I also think I would have maybe made my project less large in scope. It has around 28 classes, some of which are quite large in size as I didn't have as much time to refactor as I thought I would. This is a massive project, seeing others project scopes and sizes, I think I would have narrowed mine

For advice to someone doing a similar project in the future I would suggest using a library for user interface. Other than making the interface dynamically change when something happens to the project there is very little skill involved in creating user interface from scratch in SFML. The skill comes in placing it well and in the overall design. This would save a lot of time and energy to be spent on things that effect the user experience more, things that might get forgotten about otherwise.

I think for this type of project it was a decent technology stack to use. I did toy with the idea of doing the entire project in JavaScript for a while and have it hosted on a website. This would have meant that the user wouldn't even have to install anything or run an executable which is scary to non-technical people. This would have been a good choice given the target market. However, I think many of "SlayerMaker's" qualities would have suffered

Conclusions

In conclusion, I feel it is possible to create a game editor in which the user uses no code whatsoever. SlayerMaker provides intuitive system of a toolbar and choicenbar to create games, eliminating the barrier to entry for game development that typically exists, coding.

I also think SlayerMaker's UI and UX make it east to use, this was particularly important given the target audience that it was aimed at. It seems to provide just about all the tools required to make little shootemup games, something that was goal from the get go.

In saying that, the editor allows for a surprising versatility of games to be created in a multitude of different ways, each with their own novel gameplay. Despite this it is still not overwhelming due to the use of the initial template "Game Type" system which was used.

Finally, a no-code approach to development of all kinds but particularly in games wildly reduces the learning curve associated with it typically. This is at least the case for non-technical users who had no prior related experience. This is not to say game engines don't have a in learning development, they definitely do. Just that perhaps a great stepping stone when making the push from no game development experience to using a game engine like unity would be a no-code alternative like SlayerMaker.

Future Work

Given a database of created games now exists I think something I would like to add in the future would be a games browser. I would create login credentials and usernames, which would all be saved and encrypted etc. Then I would have a browser screen update regularly with the full database of games. The screen would then be a list of all these games, with the original creator and the number of plays or unique players that each game had.

Additionally, I would finish the build button to create executables of the different games.

Lastly, I think the game editor requires some more polish, so I would clean it and finish it off

References

[1] "Improving Student Performance by Introducing a No-code Approach: A Course Unit of Decision Support Systems" by Hai Wang and Shouhong Wang 2022

[2] "No-Code Video Game Development Using Unity and Playmaker" by Mike Kelley 2016 (Specifically Chapter 7)

Appendices

SlayerMaker Response Form