RESEARCH REPORT

Aaron O' Neill

C00241596@itcalrow.ie

Contents

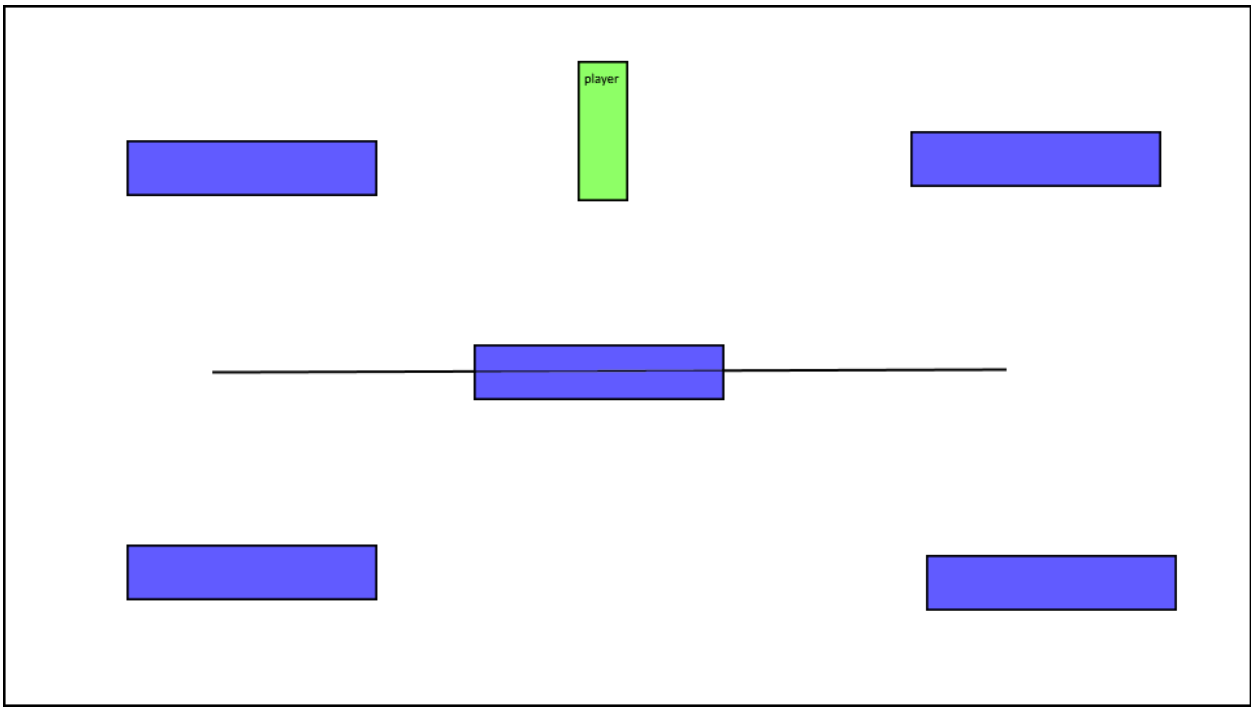
Abstract

Today, many indie games are created simply by a developer getting a small idea for a game that they like, jotting down the idea, and then making a very simple version of the game to test the idea. This means that the faster they can make a prototype for the game idea the more game ideas they can test. To achieve the prototype people will usually use Game Engines like Unity, Unreal, and Godot just to name a few. It doesn't only apply to indie games, all big AAA companies that are making games use game engines the only difference is they use game engines with their own touch put on them, so if there's a problem with the engine, they can easily fix it as they are the ones who created a lot of it. When using Graphics libraries such as SFML or SDL for rendering the game, you don't get a preview of the level before compiling and running like game engines give you, this makes it tougher to tinker with levels and get a preview without compiling and running the code. This can result in having to sink a lot of man-hours into the level design if you wish to make the level without using a game engine, this can be a big turn off if you are looking to make games for fun, or make games in a professional sense, the time you spend making a level could be used refining some other aspect of the game. I aimed to make software that would allow the user to dynamically create a level, by giving them an editor that would support the creation, modification, and joining of all theshapes that they could see on the screen.

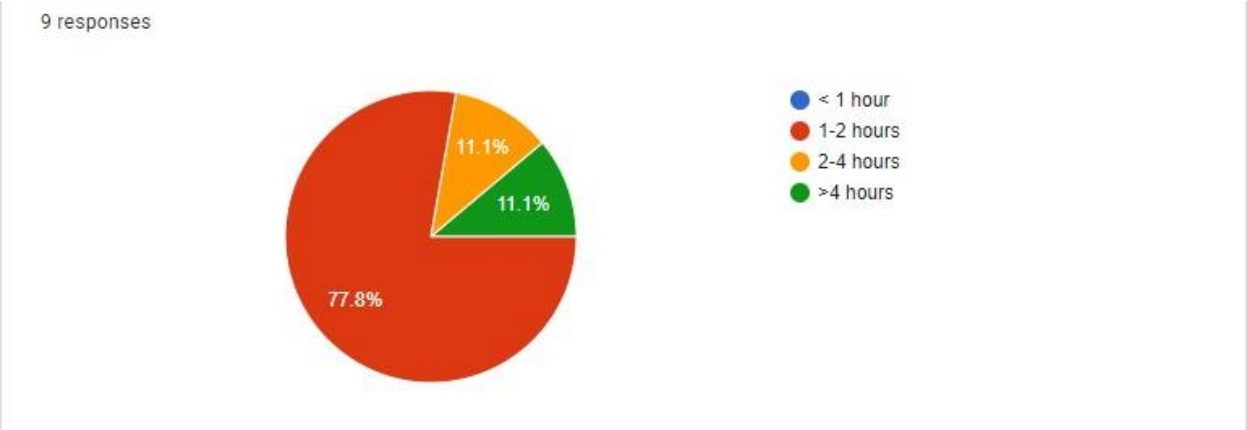
Introduction

Why I wanted to solve this problem

For all the games I've created using SFML I have always found the most tedious aspect to be the level creation, you need to use hundreds of lines of code to create all the shapes and then if you're using some external lib like Box2D for the physics end of it, it can be very difficult to make sure that the two are staying linked. If one falls out of sync with the other the physics doesn't match what the player sees and then the immersion
into the game is completely broken. It can also be very frustrating when you bump into something you can't see inside a game. To confirm that the level creation is very tedious I decided to ask my classmates who have a lot of experience using the tools over the last 4 years. I asked them how long they would predict it would take to create a level by following thisdiagram:



They were informed that the middle platform needs to be able to move along the line (The line should not be visible).
The results I got were as follows:



If you were given, 3-to 4 prototype levels to create and mess with to see which one ‘feels’ the best to play assuming they are all around the same level of complexity that would be at best a total of 4 man-hours and near worst-case scenario 16 man-hours. That would be a daunting number of resources to be given away to testing different-level designs. If the level needed to change a little the small tweaks, recompiling, and checking to see if it looks better, which may need to be done quite a few times, can be very time- consuming and very tedious.

Research

Game Engines

As my project is loosely inspired by the famous Game Engine Unity, I will be researching Game Engine's in relation to my program, and the impact that game engines have had in the production of games in recent years.
'Why games can be made faster and easier using an external tool like a game engine?'
'Are game engines worth using?'
'Should I make a game engine?'
These are the kinds of questions I hope to answer so I can draw inspiration from multiple game engines and create the best level manager I can.

Why Games can be made Faster and easier using Game Engines?

The reason that game engines are so much more convenient for making games is because nearly all of the back-end work is done for you. Without using a game engine, you need to manually write all the code yourself or use external libraries and tie them all together. Game engines will already have this done for you.

'Due to a large number of benefits, the majority of game developers are opting for game development engines to develop class apart games like the web and mobile gaming apps.' (Anon., 2021)

Game engines come with a vast array of tools built in that the user can avail of in order to get the game or level they are making, done way faster. In comparison to using SFML, when you wish to show something on the screen you have to load in a texture and draw it to the screen. Only when you compile the code and run it will you know if the texture has loaded correctly. When using a game engine, you would be able to see a preview image of the texture so you can be assured that it is loaded in correctly, then when you go and add it to the scene you will be able to see it before ever having to run the scene.
Game engines also handle managing memory for you, Unity for example uses c# for scripting so the user doesn't have to worry about managing memory at all. They can create shapes to their hearts content and once they delete the shape it will be cleaned up.

Are Game engines worth it?

Game engines are widely used around the world, there are a countless number of game engines that exist today, game engines can be very powerful as they do a lot of the 'busy' work developers dread repeating as they have done so many countless times.
The question are game engines worth it is easy, yes, and no, game engines make life way easier for the developer and remove the shackles of limitations that they might get using something smaller like SFML or SDL. All game engines have slightly different things to offer, some may offer better shader integrations, some may offer better AI abilities, but it all depends on what you are trying to make. If you simply want to make a little doodle jumper clone to release and run on your phone you couldn't use most modern-day game engines, even though they can build the app to run on your phone, the executables they produce are just too large. Some also have a license so if you are making the game in a professional sense then it may cost you money, whereas SFML and SDL are both free to use. (James, n.d.)

Should I make a game engine?

‘A lot of big studios do use pre-existing game engines as it simply isn’t feasible to create their own. It can take a lot of time and money’ (Maxwell, n.d.)
Game engines are extremely useful but creating one from scratch can take an extremely long amount of time, especially if you are trying to create one yourself. It can be a very good learning opportunity to make one, but you must be willing to put the time and effort into it.
As there are so many features in most game engines it can be tough to know where to start, but game engines doesn’t have to have any set number of features, if you only want to learn a certain set of them like my project you can choose a subset, in my case I wanted to learn more about memory management and level editors.

UI Library

One of the key components inside my software is the GUI, as this is how the user is going to be able to interact with everything inside the level. I need a GUI that would preferably be as lightweight as possible while also offering all the functionality I neededinside the software.

TGUI

TGUI is currently only being maintained by one person and that is ‘Texus’ the designer and creator of TGUI. Texus has a discord for TGUI, and it is very active within the discord if anyone has problems.

Benefits

TGUI comes with a GUI builder which can make it very easy to set up the layout for everything. This will come in very handy if I wish to change how the GUI is laid out. TGUI also seems relatively small for what it adds to the project.
The UI library can be built to render using many ways, like SFML, Vulkan, and OpenGL. This would come in very handy if I ever wanted to port my project to work with other rendering mechanics.

Drawbacks

It seems to have a bit of a learning curve to be able to use it as they use their own
"Signals" for the elements and these signals control what callbacks get called at whatpoint.

SFGUI

SFGUI is a 'Simple Fast GUI' that was created specifically for games created with SFML. It is a little simple for what I hope to have inside the project.

Benefits

It was specifically created for SFML so it is most likely very compatible with everything that is already being used inside SFML.

Drawbacks

I have no experience with SFGUI and the documentation about it is relatively scarce.

ImGUI

Benefits
ImGUI is a very well know and well-developed UI library that is used within many different a list of which can be found here. It is a very extensive UI Library.

Drawbacks
It seems to have a lot of features that I don't need in my project.

Conclusion
I will be going with TGUI as the GUI library for this project.

Physics Engine
All the physics engines that I researched fell into my criteria, that is a 2D physics engine that was programmed in cpp.

Liquid Fun
Liquid Fun is a very popular 2D physics game engine and is an extension of Box2D.

Box2D
Box2D is a very popular and very broad 2D physics library that has been used in many popular games. A link to the engine can be found here.

Bullet
Bullet is a very extensive Physics engine that supports lots of different physics collisions, it also handles Robotics.
A link to the Github can be found here.

Conclusion
After researching these 3 physics engines I decided to use Box2D.

Critical Analysis

At the beginning the biggest blocker for the project was figuring out a way to make sure that the SFML points and the Box2D points stayed in sync, I was running into loads of errors where shapes would go flying off the screen or fall through the floor then rotate around and be miles above the ground, this was happening as the Box2D world was 32 times smaller than the screen showed. After months of debugging and tweaking values I finally got to a point where the shapes were syncing up and I was free to mess with how things were being drawn, that's when I happily swapped to a vertex array as they can be resized to have the number of points that was equal to the sides I wanted.
If I was to go back and start again with the knowledge, I know now I would incorporate more programming patterns, I added a few like the factory pattern for the screens and the states, the way some of the shapes are created are kind of hacked together.
I would also loved to have made an edge shape instead of adding the edge shape functionality into the polygon shape class.
I would have liked to put the time into the callback function, now the function it calls has to accept a const IShape&, I would have liked it would be able to accept no arguments either, just to give the user some more customizability.
Overall, I'm happy with how it all worked together.

Conclusion

At the beginning I asked my classmates how long it would take them to design a platformer game, using the rough sketch of a simple level I had drawn. The results were pretty strong in the 1–2-hour range, I thought that was quite a modest figure, so I aimed to beat it once my level manager was created.
Once I got the level manager created and the level loader setup, I created a brand new project and setup the basics for it, simple 60 fps game loop, and linked in SFML and Physexe. I then started a timer and created the level in Physexe, saved the final version of the level to a file in just under 10 minutes, I then imported the level into the project I had created and used Physexe's complimentary Level loader to import the level in, I then added the functionality of the bullets, player and platforms in just under 15 minutes for a grand total of 25 minutes from start to finish. I feel like that is quite an improvement on time, from a 1-2-hour time frame down to a simple 25 minutes. If I then needed to change any aspect of the level it would be as simple as firing Physexe back up, importing the level and making any modifications that I felt were needed within the level.
Overall, I am very glad about the final version of Physexe. It was a very good learning experience creating a level manager and I hope I will be able to expand upon it and make some more games in the future using it.

Bibliography

Anon., 2021. *Advantages and Disadvantages of Best Game Development Tools.* [Online]
Available at: https://www.brsoftech.com/blog/advantages-and-disadvantages-of-best-game-development-tools/
[Accessed 23 September 2021].
James, S., n.d. *Stack Exchange | Game Development.* [Online]
Available at: https://gamedev.stackexchange.com/questions/859/what-are-the-advantages-and-disadvantages-to-using-a-game-engine
[Accessed 21 July 2010].

Maxwell, W., n.d. *What is a Game Engine and Do You Need One.* [Online]
Available at: https://cgobsession.com/what-is-a-game-engine/
[Accessed 05 December 2021].