**Make-Me-CryEngine**

This is a design overview of the engine I’ll be making, and the libraries (or rather, library) I’ve chosen to use to do so; named Make-Me-CryEngine because during the development of it I expect that’s what I’ll mostly be doing. It may or may not be retitled to something more reasonable as the project develops.

Make-Me-CryEngine will initially (and probably only) be targeted at the creation of single player top-down 2D racing games.

The basis of Make-Me-CryEngine is to have the user be able to:

* Create a grid that is customisable in size (that will be the scene)
* Create and place ‘track’ and ‘background’ tiles in grid squares
* Place waypoints on track tiles
* Place player and AI start positions on the track
* Place objects anywhere in the scene
* Modify variables on pre-built AI logic using steering behaviours
* Turn off pre-built AI and attach their own logic
* Include BMP, PNG, TGA and JPG image files
* Include 8 and 16-bit WAV files
* Attach scripts to objects

I have tried to keep the use multiple libraries to a minimum, ideally only having to use a single extra library that will be able to handle everything I need it to, filling in any gaps in functionality myself. I have listed options that I may use where I would currently intend to write things myself should writing them myself prove too challenging / time consuming to be realistically achievable.

Most of the libraries I have researched I discovered via <https://en.cppreference.com/w/cpp/links/libs> - a list of open source C++ libraries for various usages – and googling.

**2D Rendering**

The options I explored for a 2D rendering library were the Skia Graphics library (<https://skia.org/>), SIGIL (<http://www.libsigil.com/>), SDL (<http://www.libsdl.org/>), SFML (<https://www.sfml-dev.org/> ) , and cairomm (<https://www.cairographics.org/cairomm/>).

I initially during researching wanted to use the Skia Graphics library, due to it’s very easy to navigate API documentation and strong example sets to aid usability but have since changed my mind to use SIGIL instead. Marketed as “mind-bogglingly simple”, after reading through some of the documentation it really is super straight forward and looks like it will make my life a lot easier than some of the alternatives. It also can help me with handling input, can load images and audio, and contains some functionality for time. However there are 2 more very strong multimedia library options I am likely to use should I run in to any issues using SIGIL.

SDL also has fantastic documentation (and a certain fanboy-appeal factor know that many of Valve’s games have used it) and looks to be a very powerful library that I could also use for input, debugging/logging and audio. Should SIGIL not serve all the purposes I may need, SDL will likely be my next option.

I have seen SFML (‘Simple and Fast Multimedia Library’) mentioned and recommended a lot in discussions for both 2D and 3D game development, it looks well maintained, has very strong documentation and there’s the availability of 4 entire books (<https://www.sfml-dev.org/learn.php>) to familiarise myself with it; the same as SDL and SIGIL it would also help me with input and audio.

**User Input**

As above, I will also be using SIGIL to handle user input, but some options I have also explored include Gainput (<http://gainput.johanneskuhlmann.de/> / <https://github.com/jkuhlmann/gainput>) and SDL (<http://www.libsdl.org/>).

SIGIL still appeals the most, since I’m using it for rendering I can minimise the libraries I will be bringing in, and as above it seems very straight-forward.

Gainput calls itself easy to use and awesome, but what I didn’t find awesome was the API documentation, it doesn’t seem very detailed and it wasn’t easy to navigate or search through unlike SIGIL’s.

**Physics / Collision Detection**

From my research it appears there are two main contenders for 2D physics libraries – Box2D (<http://box2d.org/>) and Chipmunk (<https://chipmunk-physics.net/>, using the still maintained C++ bindings from <https://github.com/jhasse/chipmunkpp/>).

Seeing as all I want from the physics engine will be (in theory) fairly straight forward, I am considering the idea of attempting to write the physics and collision portions of the engine myself; I haven’t explored the option enough yet to say for certain that I will be doing it, but I will at least be attempting it.

If I decide that writing it myself is too much to take on, between the two library options they seem to accomplish very similar things, with very similar documentation, that I may as well flip a coin to decide between the two.

**AI**

The AI for games made in the engine should be very simple, and largely (if not only) require use of steering behaviours. There seems to be one library I can find that is intended to help with the creating of steering behaviours – OpenSteer (<http://opensteer.sourceforge.net/>) – but it has not been updated in over a decade and I do not believe I will need to use it; I should be able to use my knowledge gained in the AI for games module to create everything I need for AI in the engine, and I will include a basic driver AI in the engine that users can choose to use or replace with their own AI logic.

**Resource Management**

The engine will have basic runtime resource management that can load and unload JSON text, BMP, PNG, TGA and JPG images, and 8-bit and 16-bit WAV audio files.

SIGIL is able to handle the loading of the image and audio file types, and I will be trying to write my own file parser for JSON files, failing that I see the JsonCpp library (<https://github.com/open-source-parsers/jsoncpp>) as an option I can explore.

I see this as one of the potentially one of the larger challenges of the module, but looking at some basic resource manager examples it is a little less intimidating for a basic runtime implementation.

**Scene Management**

The scene manager for the engine will handle everything that is not a player or AI entity, including player and AI start-position objects; It will handle object parenting, positioning, depth sorting, and time.

The basis of the engine is to have the user create a track using a grid system and place individual track pieces in each grid square, which will mean a number of set grid positions that can be used by the scene manager to determine positions by using the offset value to the centre of the closest grid square.

I anticipate each scene will contain a very small number of objects, so the scene graphs will likely be stored in arrays, and render order will be determined using a painter’s algorithm to render from the parent down to child objects.

SIGIL contains some basic time functions which I should be able to use to get a game time that is able to be manipulated for things like pausing and slowing/speeding up time.