Note: entries are made between the given dates, not exactly on a specific one.

**6 November 2016**

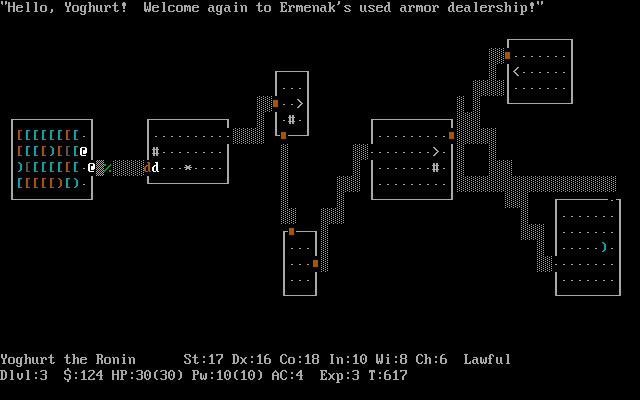
Begin project, the assignment was received on 3/11/16. Will begin considering ideas soon.

**20 November 2016**

Continued thinking about concepts for the game. I came up with a few ideas.

1. A 2D rogue-like

These games allow the player to control a person which moves around the dungeon, fighting monsters and attempting to go as far into the dungeon as possible. It would be quite easy to fulfil the requirements, and would be interesting. There are several good examples of these games made before, e.g. NetHack



NetHack, a very old rogue-like

1. A programming game

In this, the player need to write code (possibly in a fictional language) to create algorithms to solve puzzles. Zachtronics has made a few of these games (e.g. TIS-100), and this kind of game would be interesting to make and play. However, it does not have as distinct ‘lose’ condition (though this may be discussed).



TIS-100, a recent assembly programming game

1. A game similar to ‘gyro’

Gyro is a mobile game which is quite old (2 years), where you control a central circle with different coloured sectors. The player needs to rotate it so incoming balls land in the correct coloured area. I quite enjoyed playing this game before, but it no longer works, so I want to remake it. It is also a simple concept,



Gyro Gameplay

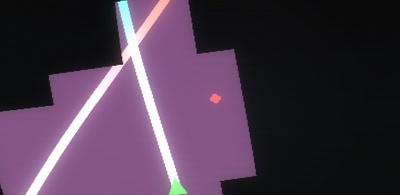
1. Orbital Mechanics

The main idea behind this game is to use gravity in space to achieve some goal. I have done something similar before (using JavaScript), but the game will need to be much more developed.

*[ no image available ]*

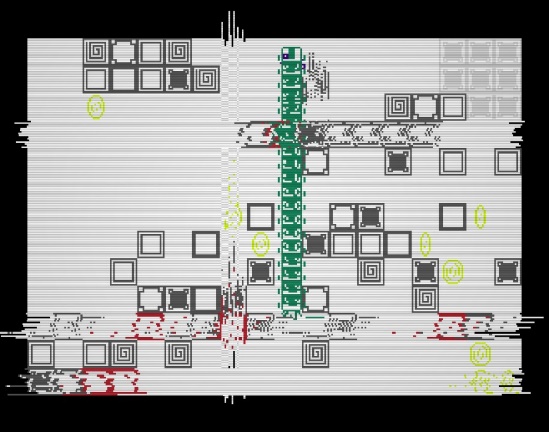
**4 December 2016**

I found a game that would likely be better to base off. It is called “jet/lag” (<https://svblm.itch.io/jet-lag>), and has fast-paced gameplay similar to other games I like. As a result, the game would probably be based off this.



Jet/lag gameplay. It’s hard to show in an image

Another game that would inspire the project is “glitchhiker” (<http://www.glitchhiker.com>).



*GlitchHiker, rather chaotic*

I will probably base the final game off these, with modification of course.

After thinking about this idea, I worked on the documentation. Specifically, the Gantt chart.

**18 December** **2016**

This is the first week of the holidays. I looked at various frameworks and libraries that could be used for the engine. I plan on using SFML for input/output (inc. display & sound). I previously also have made a vector library which I plan on using (unless another math library succeeds it). An entity system will also be needed – EntityX or anax will be used (though not currently decided).

During the second week, I went to Melbourne. As a result, I was unable to work on the project.

**1 January 2016**

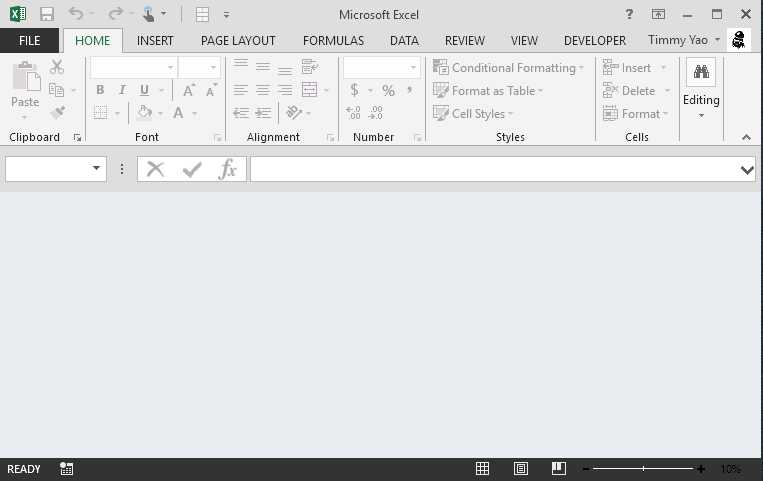
During the first week, I made a git repo for the project, which I will use to store the files created for the project (e.g. documentation, source code). I move the documentation over to that location, but did not move the source code (which stayed as a folder simply named ‘test’ on my desktop).

I experimented with using SFML, since I haven’t used it in a while. I found a decent vector library named CxxSwizzle (<https://github.com/gwiazdorrr/CxxSwizzle>), which will replace my own library. I didn’t try using either entity system yet – I don’t have the framework (e.g. a graphics system) necessary to use them.

**15 January 2017**

I made a simple implementation of signals and slots, which used a priority queue. Using this, I developed a few utilities to help with developing the game (runtime.cpp). I then continued working on the Gantt chart.

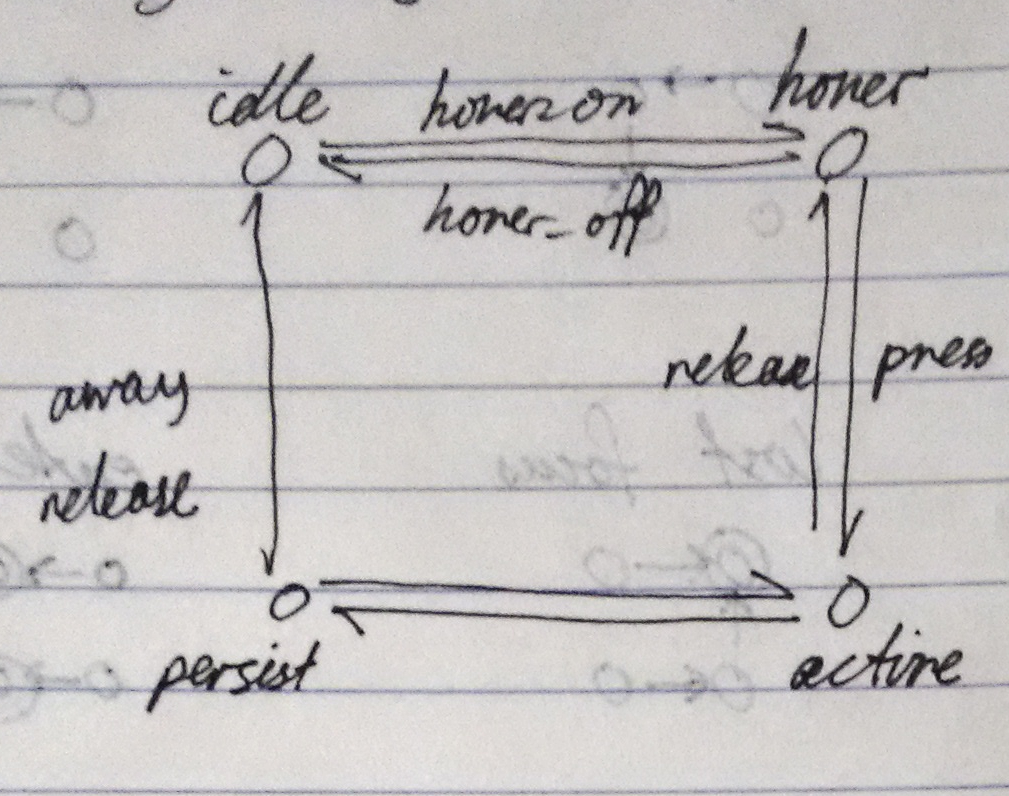
While working on the Gantt chart, Excel suddenly crashed, leaving the chart corrupted. Following this, Excel reported that closing another file I had open would close the corrupted file (?!). Even more surprising was that, instead of offering the option to recover, I was presented with a blank screen (shown below). I was finally able to recover after explicitly telling Excel to try to recover the data (press the arrow next to “open” in the open file selection dialog).



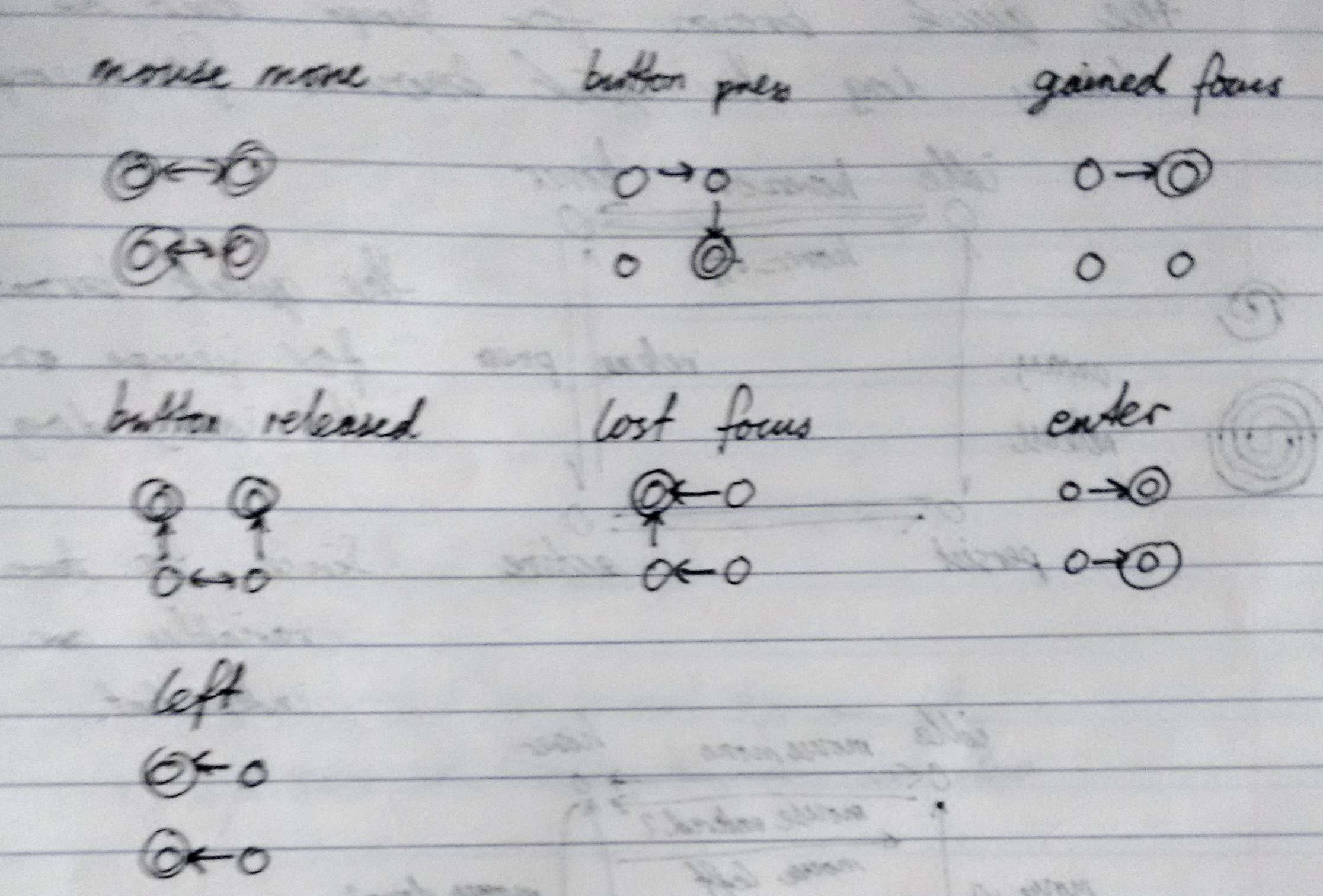
*A broken and confused Excel*

This broken file is still available as “broken-Gantt-original”, unless I make changes later. After this incident, I paused progress on the Gantt chart, and continued working on the code.

I finally moved the source code over to the repo. For the next few days, I worked on a button system for easily making buttons. I soon discovered that it was effectively a finite state machine, and spent about a day figuring out the transitions (no code yet).



*State representation, and transition names*

**

*Transitions when reacting to events*

From this, I implemented the button manager (as input/button.cpp). Using a test program, I verified that I implemented the machine correctly (it was). I then went back to working on the Gantt chart, and finished it.

**29 January 2017**

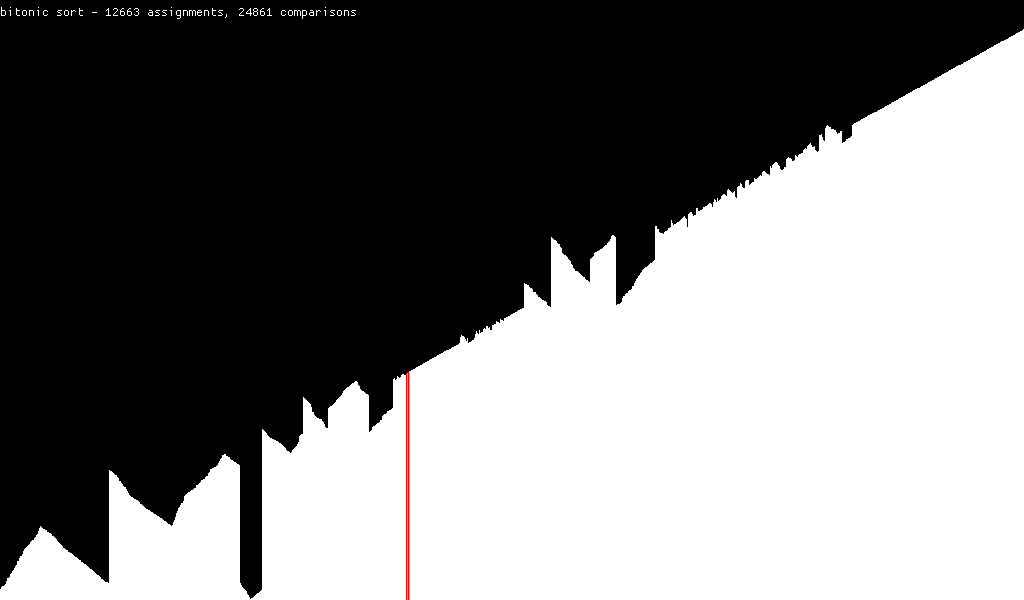
I got a new laptop (yay)! This meant moving my files from the old one to the new one, and installing various programs and such. This was rather time consuming, and the lack of a programming environment effectively stopped progress (I didn’t want issues from working on the project on 2 computers).

However, I did eventually set it up and could resume programming. I began implementing the necessary components to enable CMake to build the project. This would make compiling and testing programs much easier.

**12 February 2017**

I made a sorting algorithm visualiser as an “example” project. I got a bit carried away with it, but it looks great. From this, I learned the capabilities and limitations of the framework:

1. I needed to suppress warnings from external libraries (e.g. CxxSwizzle, which had several). These warnings made the compiler output too verbose, obscuring actual issues with my code.
2. There needed to be a way to convert from CxxSwizzle vectors to the SFML vectors. In my code, I mainly used CxxSwizzle vectors, as they have a better API. SFML API’s, however, require sf::Vector2f, which are not implicitly convertible from CxxSwizzle vectors. Since both are external libraries, there is no easy fix. The façade design pattern could be used.
3. RandUtils is a great library for random number generation.
4. The event-driven runtime encouraged modularisation. This made the program easier to implement (as separate modules) and made it more maintainable (since each module is independent).
5. Resource loading was annoying. Since the location of the executables is different from that of the resources (e.g. fonts, images), it was difficult to ensure that they were always accessible. I resorted to hard-coding them as byte arrays (since SFML can load files from memory), but these are nasty to create and maintain.



*Later stages of Bitonic sort.*

I also added a command line option parser (core/opts.cpp) to allow easy modification of some internal settings (e.g. window size and framerate). Since the sorting algorithm performed one action per frame, this was quite beneficial to ensure the slower sorts (e.g. bubble sort) actually finished in a reasonable amount of time (still a few minutes).

**26 February 2017**

Looking back on my prelim SDD game, I found that I should separate the implementation (.cpp files) from the interface (.hpp files). Then, the runtime library can be compiled separately and only once, improving build speed.

I fixed issue #1 by adding code to the wrapper headers to suppress these errors. Since the external libraries are only accessed through these headers, this fix was simple. See include/vector.hpp, include/randutils.hpp, etc.

I also began creating a solution to resource loading (issue 5 above). I figured that I could create a consistent interface for both on-disk file resources and hard-coded ones by having both accessible from memory. Hard-coded resources are simple, but on-disk resources required memory-mapped files (and all the issues of win32api). With a unified interface, I created 2 utilities:

1. A program to create hard-coded resources. This involved writing out the file as a byte array, and adding the headings to make it valid C++.
2. A program to generate the resource interfaces. Whether on-disk files or hard-coded data was used was decided by a switch, and the program generated the header and source files from the names of the files.

After adding CMake support, this was much better than before.

I also started experimenting with EntityX, the entity component system library I was going to use. After seeing that anax had a critical bug (see <http://tilemapkit.com/2015/10/entity-component-systems-compared-benchmarked-entityx-anax-artemis/>), I decided to use EntityX.

After some fiddling with CMake to build EntityX correctly, I created a starfield example. It was much simpler than the one without EntityX (1.5 screens vs 2.5 screens of code), and permitted running at any framerate. The lack of dependence on the framerate meant that I could run without FPS limiting, and got over 2000 fps (but it stressed the CPU quite a bit). Hilariously, it also works at extremely low framerates (like 1fps), and appears as if you have a bad computer.

**12 March 2017**

HSC exams are coming up, I’ll be taking a break. Not much done.

**26 March 2017**

Exams are over. Spent time squashing bugs.

I switched from using the original EntityX library to my fork, which fixed a few minor bugs. I also switched from using CxxSwizzle to another vector library named Velm, which allowed easier use of SFML vectors together with Velm’s own vectors.

Since I had a level of indirection (through include/vector.hpp), and did not directly use the libraries (but via another interface), I was able to easily swap the libraries by only modifying the include interface, avoiding much of the hassle of search and replacing all uses.

**9 April 2017**

I basically made various improvements to the API. I did, however, rework the button system, removing some redundant parts.

I had another idea for the game, which I think I might use. (you can tell I’m not that decisive on this). It’s similar to the others, and involves dashing. However, it is in a fixed grid and you can only stop when you hit a wall. No enemies, but that OK. Anyways, I’ll need to start working on content soon.

Considering the graphics style, I’m thinking of something that’s ‘retro’, with more pixelated graphics (See glitchhiker above).

I know it’s the holidays, but I’ve lost quite a bit of interest in this, and am definitely behind schedule (according to the Gantt chart). Hopefully I can catch up some time soon.

**23 April 2017**

After thinking about it a lot, I made a ‘delayed execution’ system, which allowed functions to be registered to be called at a later time (e.g. after 1s) or to be continually executed until a point in time. This allowed many time-wise operations (e.g. fading in/out with looping) which would be essential for the game.

I also made a “sprite set” class, which loaded spritesheets (with multiple NxM sprites), split it up into the individual frames, and provided an sf∷Sprite which used these. A method to change frames was also added. This made animating sprites quite easy.

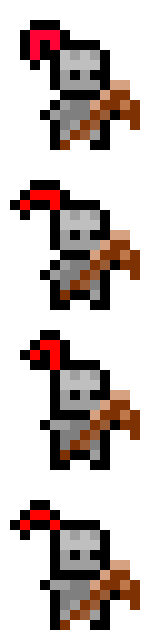
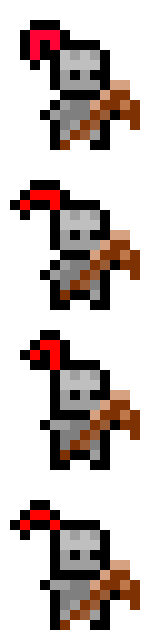
**7 May 2017**

I completely backflipped on the game idea. I’m making a chess-like rogue-like, not very fast-paced. However, the game’s only a concept at this point.

It’s half yearly period, so less time was spent on this project, and more on studying (I want to do well). Mostly past papers. Interestingly, I didn’t consider this in the original Gantt chart, so we’re going a bit further behind schedule.

**21 May 2017**

Post half-yearlies, pre HSC tests, so also not so much work. I made a sprite for a knight, which has a fake “horse” (a head with a stick):



*Knight*

It didn’t take too long, but it’s in a similar style to my prelim SDD game, with roughly 16x16 sprites and the default colour pallet in paint.

C:\Users\timmy\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ss.png

*Old style – not actually used*

**4 June 2017**

HSC test week! Only the first week though; I can work on the project more in the second.

I did some basic implementation stuff, like the chess board and the piece.



*Basic Chess Board*

The buttons I made at the beginning of the year were very useful handling individual squares. Additionally, the way I designed it allowed for a pressed tile to be highlighted.

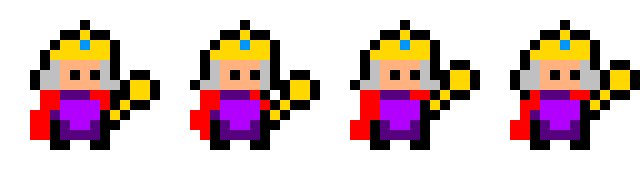
The board seemed small, so I increased the size to 16x9 later on. I also added some test data for clicking on tiles.

**18 June 2017**

Ms Chen has extended the due date for the assignment to around the HSC trials (in Term 3). This also meant I had the holidays to work on the game much more.

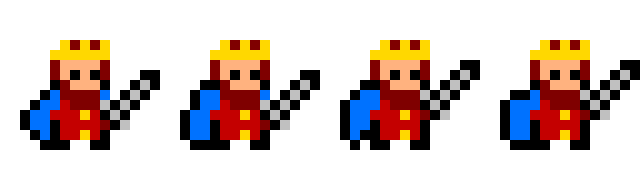
I was originally using a single file for the game’s source code. I split the subcomponents, making it easier to work with sections of the game. Additionally, I began using EntityX to represent the pieces. A feature to disable tiles (i.e. so nothing can move there) was also added, as this would allow me to design more interesting levels in the future, when I do implement that.

I also designed some more sprites. First the queen:



*Queen sprites*

And then the king:

**

*King sprites*

I also added an AI system, as well as the ability to capture enemies. The AI worked by assigning a score to every position they could move, and randomly picking based on this score. This score was determined by (with descending importance):

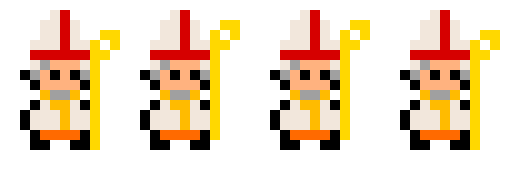
1. Whether it would capture the player (a very high score)
2. If the move would avoid capture
3. If the next move could capture the player (if the player didn’t move)
4. The number of potential moves that the piece would prevent
5. Proximity to the player

The final score after this was squared, and a weighted random choice was made using the moves. The use of a weighted random instead of always picking the highest score allowed the pieces to be captured once in a while. I originally considered using multiple player pieces at once, but this solution was simpler.

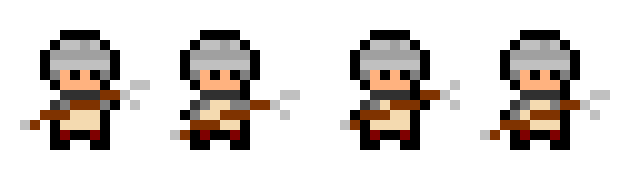
This was also not included in the pseudocode as it was too complex, and simplified to just “moving the ai”.

**2 July 2017**

More sprites were also made:



*A Bishop*



*A pawn*

I was also going to make a rook, but I didn’t know what to make it look like, even after researching a bit about the history of the rook (piece).

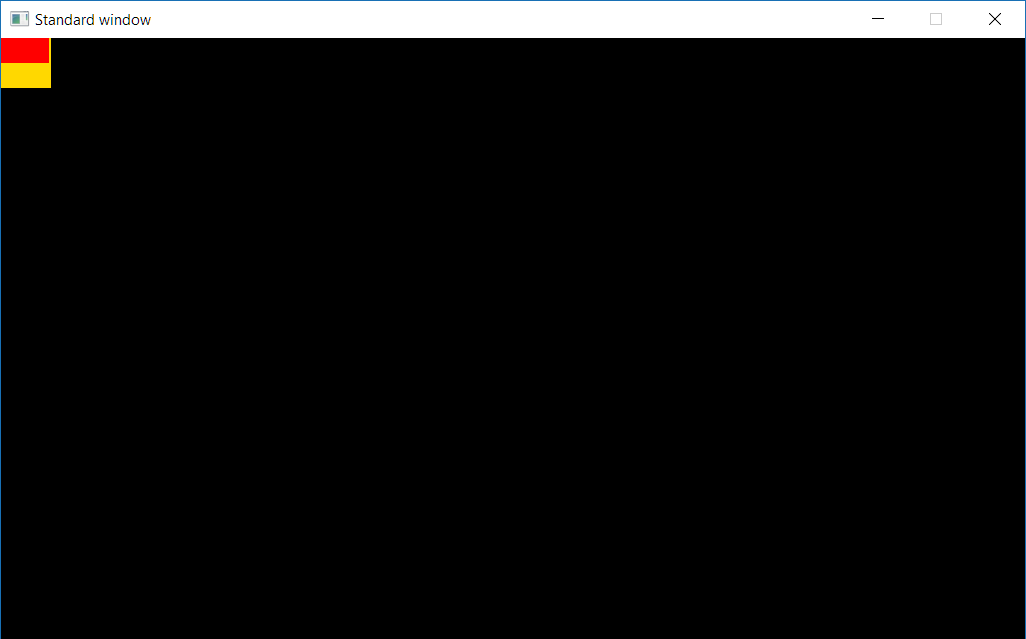
* An actual castle or siege tower (what the rook was supposed to look like) did not seem to “belong” with the other pieces.
* A chariot (origin of the rook) was too big to fit in the required space, and looked weird without the horses pulling it.
* An elephant (possibly another origin for the rook) was also weird and too big.

So I never made one (even though all other features for it were implemented).

I also made a level system, which refreshed the level when you reached the exit. With the help of the delay system I made earlier, I was able to add a fade effect, instead of “jumping” to the next level. I also used this to make the pieces “jump” to the target square, by offsetting the vertical position.

I discovered a bug in the delay system which caused actions after a task repeated over time to occur at the wrong time. This issue was soon fixed.

Around Wednesday of the first week, I encountered a strange bug, which affected all my examples (even if I didn’t recompile them). A strange offset was applied to the screen, hiding a section of the top and side of it.



*Effect of bug. Red is rendered “square”, yellow is what it should be.*

After troubleshooting a bit more, I discovered that this was from the graphics driver (!). If I used the Intel HD 620 driver, the weird offset was applied. If I instead used the Microsoft Basic Display Adapter, the bug disappeared! This is very strange, as it did not affect any other program that did not use SFML. This issue fixed itself after a few days, so everything was fine.

**16 July 2017**

Trials coming up, doing a small bit on the game, which is due after the trials.

A score system was added, with 100 points gained from a new level, and varying amounts for the pieces.

Added level loading using a special “syntax” similar to the format in text-only rogue-like games (e.g. nethack above):

...##......##...

.@.##......##.>.

...##.k..k.##...

...##..##..##...

...##..##..##...

...##..##..##...

.......##.......

...pp..##..pp...

.......##.......

*Actual format of a level*

Each character corresponded to a tile, with # marking blocked tiles, @ marking the start position, > marking the exit and other letters being the pieces. Support for multiple levels was also added, as well as a feature called “letter classes”, where a group of levels can be assigned a letter. Letter classes was used primarily for difficulty, with levels ranging from ‘a’ (easy) to ‘c’ (difficult). The letter is picked based on the score, which ensures that new players aren’t immediately presented with levels with queens.

I also implemented the pause and lose menu, which were simple overlays over the game.

I also worked on the user manual and test data.

**30 July 2017**

Finishing up the game. I added the main menu, as well as the help screens. Music and sound were also added (quite simple once I looked into it).

The driver bug I encountered during the holidays strikes again! Same deal as last time; it disappeared after a few days.

I found that a difficulty selector had to be available at the beginning, so I added one. It starts you out with a different score modifier depending on the starting difficulty. This selector has 5 options – one for each category.

I made the final Gantt chart and finished up the documentation, such as the pseudocode.