SDD **Major Project Log**

*Seb Holzapfel*

*8/10/2012*

We were notified of the project, what it would involve, it’s weighting, and due date. One of the most essential components of the assessment was the project log; hence I thought it an idea to begin this as early as possible.

I’ve failed to come up with some form of idea as yet; however I am thinking using C++ to create some sort of 2D application might be the way to go. This is especially the case as I have written 2D applications before, increasingly adding to a generic application framework which has proven to be extremely useful. As this sort of framework would probably exist as a library in other languages (rather than having to create it from scratch), I don’t believe it detracts from the complexity of the project I will be completing in the following terms in using some [heavily modified] code I developed prior to being notified of this project.

*17/10/2012*

Recent topics in physics and the observation of a classmate playing a Java game (Death Star Battles) have inspired me to create some sort of game which involves gravity in a point-mass form. Hopefully the maths isn’t too complex!

*18/10/2012*

Talks on malicious software reminded our teacher of an old computer game (’97) named Virus, which involved the player piloting a mech-like contraption through the computer’s **actual filesystem**, with pictures being displayed on walls, text files showing their contents etc. I have never seen nor played something with a similar mechanic so I think this would be an interesting element to explore!

*19/10/2012*

Elaborated on some of the ideas I had in mind and now have an outline of what my project will be:

A real-time 2D space-shooter type game, whereby a directory is fed to the program (Maybe via a shell extension, a command-line argument or an in-game browser ). The contents of the directory will seed the level generated; each file and its size accounted for in a probabilistic level generator, whereby each entity will be labelled with the filename and the largest sized files tend to be planets or black holes, the smallest tend to be bombs etc. Some files will be randomly designated ‘enemies’ which fight the player. The program will keep count (percentage-wise) of how much of the user’s computer has been ‘defeated’.

A nice graphical ‘universe-type’ visualisation of the user’s file system for level exploration between levels would probably be the best way to go, however I believe this would require a large amount of artistic skill and time; and Software isn’t my only subject! (Even if it is my favourite)

On second thoughts, an easier way to implement this may be to simply have folders represented as wormholes, their entry loading a new level in that folder (There will always be a wormhole to move up a directory). I should be able to achieve this in reasonable time.

I have dubbed a (maybe temporary) name for the project: **Metanact**; which I arrived at after much time abusing techno-babble generators and thesauri, a simple combination of the words Meta and Enact. It is quite a unique name which would be important were the project marketed – having only 6 total results on Google (all of which aren’t perfect matches).

Also, I’m starting to deconstruct the engine I wrote for my other applications so I can document its structure (maybe recalling my thought processes) and determine which components need to be modified or removed.

*20/10/2012*

Wrote the *Engine Architecture – Pre Modification* document, also found the old tag list.

Completed a little prototype (command line C++) that shows the contents of a directory, determines which items are folders, and sorts files by their size. This will become a component of the level generator eventually.

Managed to get the engine running without a network server (Projectiles don’t work yet though), still have to strip out some more networking code.

*22/10/2012*

We were instructed to begin work on a Gantt chart, I should probably do that before making any more progress!

This afternoon I incorporated the CLI file directory program into the engine’s level loading routine. It works quite nicely, at the moment all I am doing is creating a warper with the name of each folder and a piece of scrap with the name of each file, putting them in random positions. I think the size of each level would have to change dynamically though depending on the number of files and folders.

*23/10/2012*

I managed to pull directory exploration together – it was actually a little simpler than I expected. I had to pull the level-loading routines into a re-callable method (which involved some changes to the entity destruction mechanism) so that a new level can be loaded while the application is running and already in a level. Then all I did was change the code inside the ‘Warp’ tag to check if it was warping the player, in which case it would re-load the level using the current directory plus a slash and the Warper’s display name. I also changed the level loading code to incorporate the number of files and folders into the resultant level size (it’s [500 + 50\*files + 150\*folders] pixels to be exact)

I was having some performance problems, achieving frame rates around 80FPS on my desktop, especially in folders with a large amount of files (which would probably correspond to about 20FPS at school) (Oh, and VSync wasn’t enabled), so I decided to hack in a little culling routine into the function that calls each entities’ draw function – if the entity is more than 50 pixels offscreen, don’t draw it. This brought the framerate up to about 350FPS. I’m thinking that when I begin to use images instead of line graphics the framerate will go up further as there will be less function calls on the CPU and more processing done on the GPU.

I put some notification text at the bottom left of the screen informing the player of the directory they are in and the size of the current level in ByteParsecs ( About 1 for every 10 folders/files).

I also created an extremely educational TRACSDD wikipage involving myself and a potato.

*24/10/2012*

I created a rough Gantt chart (‘SDD Gantt’) illustrating how long I think each development process will take and the order in which each task will be performed. I probably won’t be sticking to it too strictly, but it still serves as a guide if I’m not sure what I’m up to.

I also updated my wikipage, it now shows my Gantt chart.

I’m not sure whether to put my log on the wikipage because it will end up being so long – might have to though depending on whether we are judged/marked partially by our wikipage.

*27/10/2012*

After a morning of maths homework I didn’t really feel like doing much else so I decided to get some creative work done as that’s a little more productive than just playing games (Which is what I’d be doing otherwise). I wrote the main theme music track and rendered it out – I think it turned out alright – a sort of electronic-meets-ethnic type style with a recognisable melody (An interesting [personal] change – usually when I write videogame soundtracks they tend to be more textural than melodic).

I also fiddled around in Photoshop a bit and chugged out the Metanact logo and modified a ‘Schnommus presents’ logo. Lookin’ good I think!

The next thing I want to do implementation-wise is allow entities to have an image assigned to them rather than (or maybe in addition to?) line graphics. I can then code up a couple of view-space entities for the game menu system and get the splashes I created into the game. It would also allow me to start transitioning the old line-based Fluxan graphics into nicer images.

I was bored in the afternoon and so ended up creating the IMAGE tag that can be used to assign images to entities – currently this works alongside the line graphics while they are transitioned over. Next I have to create some sort of way for the engine to keep track of view-entities (which are drawn in the view-space without camera displacement).

After some trial and error I hacked together a way for splash images to be displayed on startup.

I also made a way for the ship to move up a directory – It was pretty easy, just had to add a warper with it’s directory set to ‘..’ – the universal ‘up a directory’ command (Brought to my attention by Brendan Joseph Roy © 2012 All Rights Reserved. Unless explicitly expressed, all writings within this document are the opinion of Sebastian Holzapfel. While Brendan Roy endorses Sebastian Holzapfel, he cannot guarantee the accuracy and integrity of his writings. EG: ‘..’ is NOT a command, Brendan Roy understands this. On further research, Seb agrees – it is a parameter). I ran into problems with this though as the engine was working, but spitting out paths like ‘C:\Windows\CLR\..\..\Program Files\..\’ which only got longer as the game progressed – it was an easy fix though, turns out Boost has an inbuilt function ‘canonical’ which reduces a path to its simplest syntax, so I just call this to my current path every time a directory change is made. At the moment this ‘special’ warper looks the same as all others only with a different name. I’ll probably make a separate entity for this and change it’s colour and size so it’s easier for the player to recognise.

*28/10/2012*

The splash screen images weren’t displaying nicely so I turned linear interpolation off which fixed it.

I also photo-shopped some menu sprites together so I can begin coding that part soon. I’m thinking the Metanact logo should move up and the menu should fade in underneath it.

I created a couple of resource management methods, for images and fonts so that they are only loaded from files once and kept in memory. This increased the framerate a little as well, probably because each entity used to keep its own copy of an image/font whereas now all images/fonts are shared. It works by keeping track of all the loaded files, and when the method is called it returns a file already loaded – if it isn’t loaded, it is, and then returned. I’ve changed a couple of the game fonts so they look a little more original.

The new font manager will probably handle the menu, might ditch the sprites I created earlier – besides a ‘selection’ sprite.

Documentation-wise I’m thinking I should probably finalise my formal requirements before working on the game too much.

*29/10/2012*

We were told in software today to recognise copyrighted materials – I’m thinking of initiating some sort of document which will describe the third-party libraries and resources that I am using.

Spent some of today’s lesson creating logos for other people – serves as good project motivation for them I think!

This afternoon I implemented a couple of features Dr. Willsher thought would be a good idea – fading in of folder names when they are approached, and a limiting of the number of files + folders that will be loaded into a level. The list of files/folders kept is randomized so that each time the game is played, larger folders are slightly different and the player eventually has the opportunity to see every file that was in the larger folder.

I tried to introduce a ‘loading screen’ of sorts, but it isn’t proving to be a simple task – the game freezes while it analyses folders that it enters. Either I make the level-loading routine multi-threaded, or I put a time-lock on folder traversal so that the loading screen fades in nicely before the game freezes. (At the moment, it is difficult to draw a loading screen BEFORE the game freezes, because the level loading routine takes place outside the draw loop [Preventing me from putting a ‘draw sprite’ instruction at the top of the level loading routine]).

My code is beginning to look a bit piece-meal; might have to start a refactor job soon.

*30/10/2012*

I wrote a document (‘SDD Third Party Resources’) which contains all the resources I have used (programs/tools as well) and their licenses. I’ll be adding more to it as the project progresses – but it is already almost 1 and a half pages!

I cleaned up the rest of the networking code, and as far as I’m concerned there is none left now. I also felt like doing some nice visual coding so I created a little fading binary ‘star’ effect that takes place in the background of the game – it gives it a nice 2.5D depth I think.

*31/10/2012*

I had a little brainwave after looking up crossfire systems – what if I stagger the gravity calculation? After all, it doesn’t *have* to be computed every single frame and at the moment it is probably the single biggest CPU hog.

This wasn’t as simple as each warper keeping a value and incrementing it, taking the modulus to perform gravity every n frames; because that would mean that *all* of the warpers would perform gravity calculation in the same frame (every third) because they are all created at the same time– not beneficial as that would just make the game jittery. After thinking for a few minutes I decided that using the [entities’ ID mod 3] as an initialization for the modulus counter would be the easiest way to go, as this is unique for every entity, and will stagger gravity calculations across frames without jitter.

So I implemented a little staggering algorithm in the gravity code, which gave me ~100FPS boost, from about 400 to 500 FPS on my desktop while exploring drive C:\ (Staggering workload across 3 frames). Might be of relevance were the program to run on a less capable computer.

*2/11/2012*

We got the ‘official’ assignment brief today so I should probably do a reprioritization of all tasks so everything is ready by the first due date. (There are 3 stages as I see it, first half of project due term 1, second half due term 2, presentation due term 3)

*3/11/2012*

I rearranged the project filesystem so it looks nicer; media files are now in subfolders within a ‘media’ folder instead of all over the place, source files are in their own folder, and compiled binaries also have their own folder. Consequently I had to change some of the source code to retrieve assets from the new directories, which now works.

I also added some code which adds ‘loading…’ text to the screen while a directory change is being made.

*7/11/2012*

Started writing the formal specifications – I’ve finished the ‘problem’ – now I’ll have to move onto the ‘solution’ section and some sort of feasibility study.

*12/11/2012*

Hand-drew 3 storyboard diagrams (Menu screen, options screen and in-game screen), and will probably scan them in later for the wiki. I also created another slide in the Gantt PowerPoint illustrating the changes in schedule thus far.

*13/11/2012*

Created a little engine effect for the player’s ship – which creates decaying circles with a starting transparency dependant on the speed of the ship so they become more visible the faster the player flies. Thinking I might hard-code a particle system into the engine (tag list) to avoid recoding specific particle types every time I need one – maybe an IS\_PARTICLE tag.

*18/11/2012*

Had a crack at creating the first enemy – the ‘grunt’ (Alongside an internal ‘IS\_ENEMY’ tag). All it does currently is follow the player and starts shooting as soon as it gets in range. I halved gravity’s effect on the enemy so that it’s harder for them to be sucked into warpers while persuing the player.

In Monday’s double I’m planning to make some more progress on the formal specifications, and maybe envision some more game play elements.

*23/2/2013*

I finished off the long-delayed first page of my problem specification/feasibility document – Lots of fluffy high-level objectives and excessively formal language is to be found within. It’s now stuck on my wikipage as ‘SDD Formal Specifications’.

Finally started working on the much-needed menu system; deciding to opt for an OO modular approach to its design as it needs to be extensible and easy to add options too. I thought I might document my thought process a little more for this component of the engine just to illustrate how my mind works when encountering a problem (I like to say that my project consists of many smaller ‘modules’ of which each can be considered as their own problem).

Details in the specific ‘SDD Menu System’ document. In brief though, today I got menu items to show up in the game and they highlighted when the mouse runs over them.

*24/2/2013*

Continued working on the menu system, each item now has virtual ‘Click’ callbacks to perform their respective functions and there are specialised ‘Option’ classes for menu items that are able to be toggled. Escape also brings up the pause menu now. Again, more info on the menu system is in ‘SDD Menu System’.

*3/2/2013*

Stuck the project files in a proper VCS; Mercurial hosted on bitbucket. Spent a couple of hours on the program code, ticking off a couple of TODO items – changed the music encoding from .wav to .ogg which brought the project size down from 30mb to 6mb, added a particle density option to the menu for less-able computers (I’m talking *extremely* old), and wrote routines to load & save game options to a configuration file instead of them being wiped at every startup.

I also removed a couple of dependencies on images that were placeholder-only and streamlined the exception mechanism a little better so that when the program crashes it doesn’t just freeze, it leaves a console window open stating what went wrong.

*4/2/2013*

Looked at a couple more TODO items; made it so that the visible sizes of wormholes actually correspond to the underlying directory size (which meant I had to re-write the directory traversal algorithm to be semi-recursive). This code also helped me fix a bug that’s been around for a while; making it so that inaccessible directories don’t spawn wormholes. I also coded some fixes for the minimap code so that wormholes don’t display themselves on the minimap before a level is complete (they used to be invisible in the game-view but not on the minimap).