Warehouse Escape – Post Mortem

Overview

This project has been a great learning experience for myself. At first I found it very overwhelming, due to have doing majority of my programming in Java. This project has been a massive learning curve for myself. Throughout the development of Warehouse Escape, I utilised the lecture slides, google, and the discussion forum to solve any programming issues that arose. During development I was fortunate enough to have my flatmate's test my game as it was being developed. This was very useful for finding bugs, and also great for getting general feedback.

I feel good with the end result. I do however regret not making a more fun side-scroller, rather than the top-down game that it is. The game has no memory leaks, bugs, crashes, warnings, and can be restarted, paused, and quit. The quality of the code could be improved, however due to time constraints, I felt the extra time I had would be better adding more features, and making a gold build because it ran fine. The goal of the game is still the same as mentioned in the design phase. However, some features were change during development, and some were not included.

What went well:

Memory management: Although I did not take Steffan's words seriously enough when he said "Manage your memory as you develop, rather than doing it all at the end.", I quickly regretted this when I finally came to removing all the leaks. Although there were many, I found patterns and trends shown by the memory leak debugger, and solved many of the leaks. I got to a point where I had 13 leaks left, and could not find them. I found an extension through the visual studio store called Visual Leak Detector, this pointed directly to the line of code which was creating the leak. From this I was able to resolve all remaining leaks. This has definitely been a great lesson learnt, and I will hopefully not have this horrid task again in the future.

Implementing middleware was very new to me. Although I struggled a lot understanding how each middleware is to be implemented, through developing the game, I eventually felt more comfortable with each middleware used. I was initially put off using middleware from the beginning as I had never used them before, but once I had them implemented, I had the help of google, and my peers to give me pointers in the right direction. I am definitely pleased that I got these in, as they made the game much more flued, gave it more character, and made it very easy for me to build a more complex solution.

I was initially a bit worried about what the game assets would consist of, and if they would be any good. A good friend of mine informed me of a great free game asset website, which houses most asset types, such as; music, sound effects, sprites, and more. This was opengameart.org. I used this for majority of my assets. I also created a few visual assets for my game, and got the music tracks from a non-copyright YouTube channel called NoCopyrightSounds. With the addition of these free external assets, it gave the game more character than I ever could, if I were to do them all.

What went wrong:

Data driven design was never implemented properly. I devoted majority of my development time into building the core gameplay. A particle emitter was not implemented either. This should have been an easy task, as I had previous experience with creating particles in computer graphics. As stated above, while building the game I did not manage my memory, this resulted in a big back lash at the end when I went to clean it up. The initial design mentioned that the game would have two controller types; mouse and keyboard, and Xbox controller. However, I only ended the build available to be played using an Xbox controller.

My project did suffer slightly from project creep; it doesn't contain every single detail which I stated in the design phase. This happened because, during development I would see the features which I wanted to implement, and find that they were either not going to be worth implementing due to my abilities, and amount of time I had, or because I did not think that they were worth the time and effort. An example of this was; stunning enemies. During development I found that stunning the enemies was not as fun as simply killing them, so I made this change.

Lessons learnt:

Throughout this project I have learnt a vast amount of things. Firstly, C++! I had never coded with C++ before in my life, and am now happy that I learnt it, and created something exciting and fun with it. Secondly, implementing middleware. The use of middleware made the whole development easier (Build more exciting things quicker) and harder (Having to learn how they all work). The struggles that the middleware gave me definitely paid itself off, as I was able to add so much to the framework with it. Finally, memory management. This was great part of programming that I'm glad I've learnt. I also learnt that I should manage my memory over a whole project, rather than leaving it closer to the end, as this cost me a lot of time which could have been spent on developing features.

Conclusion:

In conclusion, I am satisfied with the outcome of the game I have developed. I'm a bit disappointed that I did not get a map editor built, nor some sort of particles, but these will definitely be achieved in future projects. Project creep in any project is not a good thing, but in this case I feel like it made the end result far more exciting. During this project I have had ups and downs, but overall feel very accomplished, and glad I put the effort in. Look forward to applying the skills I have learnt over the course of this project, and apply them in future projects.