Post-mortem

Overview

My overall feeling is that the development of my individual game went well. The game looks and functions, for the most part, according to my vision. It includes several technical features that were personally challenging for me, but turned out nicely, for example the GameState Finite State Machine and rotating exhaust smoke sprite.

However, given it was my first game development project from 'scratch', much of the development was done on the fly without a lot of prior planning. While a Technical Design Document was completed prior to development, I found throughout the process that it was lacking in detail and in some cases missing mention of important features altogether. Whereas the creative documents (Pitch and Game Design Document) were perhaps slightly ambitious due to a mismatch in my creative goals and technical abilities. This meant that by the time the Gold Milestone was due, many features that I had wanted and listed in the Game Design Document were not complete.

The experience gained from this development project will help a lot in the planning and implementation of my next game project, and will inform future milestones and deadlines. I now have a better idea of some of the common challenges in game development and have a better appreciation of what it takes to implement a variety of common features. For my next game project I'd like to have improved my knowledge in fundamental math and general programming principles so that 'simple' technical features pose less of a barrier and I can graduate to tackling more complex game elements.

What Went Right

The game achieved an aesthetic close to what was envisaged, especially considering the time constraints. The game provides a good approximation of the experience I wanted it to give and with more time it could be brought even closer to the original vision. The two technical aspects I am most satisfied with are the Finite State Machine (FSM) and rotating exhaust smoke sprite.

FSM:

As above, I feel that the finite state machine (FSM) used for my game states turned out well. It simplified my game immensely which in turn paved the way for many additional features. At first I found the FSM concept overwhelming, so after reading several articles and a few tentative attempts, I decided to copy my solution and slowly refactor my code into the FSM design pattern. I believe that the FSM is an example of a development problem that I personally could not have planned a solution for entirely upfront. The reason I chose to copy my solution was that I was not yet fully confident with SVN. Next time I would just utilise the repository rather than copy the solution.

Having a backup to my solution allowed me to implement the state machine without the fear that I might break what was already working. I found that this strategy allowed me to fully commit to the changes that were required in my game structure, because I knew I could just go back to the old state if need be. The ability to experiment with alternative structures in this way, knowing that I could go back to a previous working state, unlocked my capacity for exploring other implementations. This was a useful learning in terms of the wider development process.

The FSM development and integration was gradual. I broke the concept down to small understandable and achievable increments that I recognised, such as identifying which elements within the game.cpp file represented typical GameEngine functionality and which represented typical PlayState functionality. This helped me to recognise what the base class GameState should do and consist of. Then I slowly started to separate the components and refactor my code. During this time I would build and compile frequently and utilise Visual Studio to address errors that I otherwise hadn't noticed.

While implementing the FSM, I found that I did not fully understand the concept until I was almost complete with its development. Once completed I found that the game suddenly got much easier to manage. During the process I also got a better understanding of pointers and variable addresses and how they can be used. Additionally I got to understand the utility and function of static variables and virtual methods, of which I hadn't used a lot of up until now.

In the future I hope to learn about and use more of these types of well-established programming patterns, tools and techniques. This expanded toolset will save me from reinventing the wheel when I come up against common programming problems – such as what the FSM pattern seeks to solve.

Rotating Exhaust Smoke Sprite:

I was surprised by many challenges during development associated to features I thought would be easily implemented. One example of this, was the rotating exhaust sprite which was an absolute odyssey of the mind to create and get working. This was mostly due to poor prior planning and a lack of basic mathematical knowledge. In the end the mathematics proved to be fairly simple, but it was the overall complexity associated to the elements of the wider problem that made it difficult for me, such as:

- identifying the mathematical functions required to achieve the result I wanted;
- identifying the appropriate SDL2 functions to use;
- distinguishing between the smoke sprite's own rotation and its overall rotation around the ship sprite; and
- identifying the origin of rotation for each of the sprites.

In the end I undertook the following steps:

1. Calculated the radius of the rotation i.e. the distance between the centre of the ship sprite and the upper left corner of the smoke sprite. This required Pythagoras'

theorem using half the height of the ship sprite and half the width of the smoke sprite.
2. Given the opposite and adjacent sides of the resulting right angle triangle were equal, the angle was obviously 45 degrees, however being naive I initially used the arctan function to get this angle.
3. Having both the angle and radius, I calculated the relative position of the smoke sprite in relation to the centre of the ship.
4. This allowed me to determine the smoke's absolute position in relation to the ship's absolute position, and therefore the absolute location of the smoke sprite's centre.
5. This then gave me the values required to implement the standard equation for rotating a point around another point.
6. The draw methods could then rotate each of the sprites around their individual origins based on how much the ship had been rotated.

The above is probably quite obvious for anyone familiar to the associated math, and likely a very an inefficient way of solving the problem. However, I found it to be an incredibly tough and frustrating process. The sheer amount of variables in play and the nature of Sine, Cosine and Tangent meant that buggy results were often very hard to decipher and solve.

I think that if I had been familiar with trigonometry and vectors this would have been a trivial task. I aim to upskill in these areas, as so far it seems like a small amount of basic mathematics knowledge helps solves a large proportion of the different problems encountered in game development. I also found that sketching the desired outcome during planning led to useful insights and often revealed missing variables that I wasn't currently considering. In the future I will be doing much more of this.

The rotating exhaust animated sprite turned out to be one of the most challenging parts of the development. Had I had a better grasp and familiarity with basic trigonometry this process I think would have been much smoother. With all that said, this was actually the most satisfying problem to solve.

Finally, personally, I found the order in which I developed was quite good given my level of experience in C++ and game development. I found it very hard to plan implementations upfront without knowing something about the challenges I'd face. Iterating and planning as I discovered specific problems helped me move forward without getting paralysed by the scope and unknowns. However, now that I have more experience, I think I will be able to do better upfront planning for future projects.

What Went Wrong

As mentioned above, planning was hard to do upfront without enough experience in game development. I didn't fully appreciate what it would take to solve common game development problems. For instance, it wasn't until after I had implemented my high score board that I realised I hadn't considered how I might save these values for reloading at the game's start. This likely demonstrates poor planning, but I think it also shows that I lacked real world experience in developing games and that I hadn't yet appreciated the full scope of game development.

Similarly, I think the main reason I found the technical aspects of preproduction hard to complete was due to my inexperience in game development and programming in general. I lacked an awareness of the typical problems encountered throughout a game's development. In the end, a large part of my technical planning consisted of encountering a problem during development and then planning the solution, as opposed to identifying all problems and planning solutions upfront. This iterative approach actually worked reasonably well, but ultimately led to me running out of time to implement all the features I wanted. Next time I hope to take a more methodical approach, now that I better appreciate some of the common problems in game development. This will allow me to scope the game more appropriately and create a more reliable development timeline.

I also found that timeframe constraints and uncertainly regarding certain problem solutions caused me to program in a messy, haphazard way. My code lacked comments, appropriate debug logging, appropriate access modifiers and had many magic numbers. With further experience in C++ and game development I'm hoping to get rid of these poor habits and get to a point where best practice takes less deliberate effort to apply. I think that more experience under less stressful circumstances, along with some generalised reading on computer science and programming, will help a lot with understanding and adopting best practice ideas. Similarly, while using C++ in this game project has helped me better appreciate the data lifecycle, I think I could still do some work in this area. I don't think I yet fully appreciate the function, role and utility of pointers and references. The same applies to static variables and various standard design patterns etc.

In the end, my lack of experience and upfront planning meant that the final product is missing numerous features that would have been great for achieving a full 'vertical slice' of my game. For instance, I didn't get a chance to implement Data Driven Design or AI which I was disappointed about. Additionally, I underestimated what it would take to implement some features to my satisfaction and therefore failed to deliver on them, such as the screen wrapping draw function. The controls were also disappointing in that while the game can be played with the XBOX 360 controller, it still needs a lot of refinements to sensitivity and function. The game also lacked the depth I envisaged due to it missing variety in music, sound effects, game entities and animations etc. Finally, there are several bugs remaining associated to the bullets and collisions which still need minor fixes. Again, all time related issues due to poor planning.

Lessons Learnt

During the entire production process I learned several important lessons. Firstly, I think that being invested in the game idea and/or features really helps propel the project. I found it very satisfying to bring a vision to life. This along with milestones and deadlines provided the impetus required to keep development progressing steadily.

Secondly, I found that there is a balance to be achieved between planning and implementing. Overall, planning plays a very important part in the development process. It helps one understand and appreciate scope and gives a path forward to solving problems. In the future I would like to plan more upfront using my new experiences to inform this. I

would also like to adopt simple techniques and tools into my workflow, such as sketching solutions, to help aid the problem solving process.

With that said, I also think there is a point where one needs to act in order to reveal enough information to fuel that planning process. Too much emphasis on forward planning without the requisite knowledge can lead to a certain kind of paralysis. I think this is where an iterative process comes into play. Throughout development, I found that the sheer scope of possible optimisations and best practice ideas often left me unable to act because I was looking for the best solution straight away. For a large part of the development, I was under the illusion that there was a single “right” way of implementing the game. In the future I would like to operate under slightly different assumptions so that I can maximise both the benefits of thorough upfront planning and an iterative design process.

Finally, I discovered during this process that I could personally benefit a lot from some generalised learning in both mathematics (Trigonometry, Linear Algebra and Newtonian Physics) and fundamental programming principles and best practice. This knowledge would supplement my current position well, giving me the tools to solve common and frequent problems associated to game programming and programming in general.

Conclusion

In conclusion, overall I am in very satisfied with the outcome of my game. While it is not perfect, it has come close to what my initial vision was and I am proud of the sheer amount of time and effort I put in. However, next time I would like to approach the process in a more structured way with more thorough upfront planning, while also leaving room to iterate and discover new things throughout development. I would like to approach development with an open mind and a drive for best practice, while not letting this stunt the overall development process. Ideally, I would also bring a more comprehensive basic knowledge of programming and math to the project, so that my problem solving toolset is more equipped for facing many of the common challenges associated to game programming.