Shooty Space Rocks

Post-Mortem

By Sam Ward

Overview

This project was an individual effort, carried out over roughly 7 weeks. It was my first attempt at a game project and my first time working with c++. There were a few criteria that we had to meet, but for the most part were allowed creative freedom. Shooty Space Rocks came about because I wanted to work with something familiar, but at the same time a little bit different. Mostly I wanted to develop a game that allowed me to practice the various c++ techniques learned throughout the semester and, at a glance, Asteroids seemed to tick those boxes.

It's been a great experience and development went relatively smoothly.

What Went Right

Flexible, modular code

Something I try to put a lot of effort into is code flexibility, and I think I did quite well with the use of things like relative positioning of objects to the screen size (game can be played in full screen) and adaptable classes e.g. particle emitter. I value modularity and code reuse quite highly and have written some code with this in mind, using virtual methods etc. I could have broken things down further into separate classes/methods, and a few pieces of code I have recognised as needing a change but didn't get around to, something I am looking to improve upon for the next project. There is definitely room for improvement here, but overall, I'm pretty pleased with it so it goes in the win column.

FMOD

Took me a while to get around to doing this, I think initially the idea of game sounds didn't really interest me that much so I put it off, and I thought it might be difficult to implement. Turns out it was really easy and quite fun, following the FMOD documents and creating the sounds with sfxr. In the end quite pleased with the sounds and seems it does add value to the game after all, could possibly have gone a bit further with a victory or defeat jingle.

State Machine

This is a 50/50 right/wrong. While it was a lot of effort and wasn't short of issues, I'm pleased with my achievement of the state machine. There is a small bit of confusion between what state should own what objects which I will try to improve for the next project, but generally it works pretty well.

Vector Mathematics

I quite enjoyed playing with the mathematics for getting the rotation, directions, angles for the game entities. Everything works well and the best result of this would be the enemy AI.

Enemy AI

Seeing the enemy ship move out of the way of an asteroid for the first time was a proud moment for me. I'm really stoked with how the enemy AI turned out; it actually looks quite natural. Slower and weaker than the playership so still gets destroyed by the asteroids but I had my fair share of

dogfights during the play testing. The direction from enemy to player and rotation of the gun, reversing direction angle to avoid asteroids, was an enjoyable part of the project for me.

What Went Wrong

General C++ learning curve

Being my first c++ project, the general learning curve and debugging could be frustrating at times. Every now and then I would spend a long time trying to figure out how to do something that seems so simple, the feedback from the compiler can sometimes be misleading (vs other languages) so I would end up on the wrong track. These are all things that will get ironed out with experience, especially learning your way around the debugger and debugging techniques.

Particle Emitter

While this ended up in the right section under flexible code, getting it to that stage was not so much. The process of trying to make a flexible piece of code causes untold issues, I think I went back and forward about four iterations before ending up with what I have now. Memory leaks and out of bounds exceptions were common, but we got through them and I think the result is something quite useable.

Magic Numbers

As happens when a product is a learning experience, there are still a lot of magic numbers spread throughout the code. I'm not proud of this but I understand how it happens with experimenting, will definitely look at fixing this in the future.

SVN Version Control

I would quite often forget about SVN and go long periods without any commits. While this is not so bad for an individual project, it most definitely will cause issues for a group project. I think ideally you would be doing commits for almost every method you write, and I plan to get into the habit of very regular commits for the group project (and future).

Data Driven Design

I didn't see that data driven design was a requirement until very late in the project, so did not get around to it. It would have been good to give it a go even for simple things like playership initiation (speed, sprite, maxBullets). I quite like the idea of DDD, being able to design aspects of a game in plain text, nice way to easily visually represent things. I will be giving this a go for the group project.

Issues and resolving

Int to Char*

Trying to convert an int (numOfAsteroids) to a const char* to be used in the BackBuffer DrawText() method. This issue absolutely did my head in, taking hours to resolve, and to this day still can not figure out the correct way to make it work. I thought it would be a simple conversion, std::to_string(int).s_str(), as works for the LogManager, but no. A ton of googling and a ton of different methods, I ended up getting it to work with itoa, only for it to give a warning saying to use _itoa. Well _itoa then warns me to try _itoa_s, as it's not secure, but this returns an errno_t rather than a char* so back to square one. I ended up going with _itoa and suppressing the warnings with #define _CRT_SECURE_NO_WARNINGS in the DebugState class.

Asteroid Collisions

When implementing the asteroid collisions, I had an issue where the entire asteroid would be destroyed on collision rather than being split into two. Turns out the issue was the collision was processing many times as the asteroids passed over each other, essentially split and destroying the new asteroids in quick succession. Once I found the issue it was an easy fix, I added an immunity timer to the asteroids similar to PlayerShip, to give them time to pass over without triggering another collision.

Mid-Project Changes

Ditching the Rocket

While the rocket would of provide some good learning aspects, after looking into it further it wouldn't really work for this game. This game is quite fast paced, essentially requiring thumbs on joysticks the entire time, so decided it wasn't feasible to then press a button and independently aim a rocket. Possibly could have implemented a second weapon with the left bumper would have given it a nice feel, but unsure what that weapon would be.

Entity Immunity

Wasn't in the TDD but looking back it's pretty obvious it should have been. Implementing this was a necessity for resolving collisions, as well as for the player invincibility debug feature. Mostly used for the playership when you take damage, you are granted a short period of immunity.

Spawn Mode (Debug Feature)

This feature was added as a great way to test the gameplay. It allows you to switch off the normal game scenario and have control over asteroids and enemy ships by spawning them as required. This coupled with the ability to set the number of asteroids required to win the game, and the invincibility feature, really allowed for some fast tracked play testing.

Things I Would Add/Change

Box2d

Would have liked to implement box2d and given the game a more physics feel with player acceleration and the like. Not paramount, just a nice to have as the current gameplay still feels natural without it

Ship Regeneration

Would look at adding some kind of damage regeneration over time, to reward the player for good play and an extra edge to win the game.

Colours Enum

Would have been good to put all the colours the used into an enum for easy identification and reusability. I've done this in the past in Java and it's a great way to easily access the colour required without having to copy/paste from somewhere else.

Lessons Learnt

Game Concept

I wasn't too stoked with my initial game concept, just another boring asteroid game, so was hard to find the motivation for development. I tend to get caught up on small things, such as whether I had made the right choice for player movement (dwelled on this for a long time). Once I got into it though I had a blast, and the game turned out to be quite fun, with player movement adding to this by being a bit different. Just shows that you don't have to think something all the way through in your head, just start working on it and watch it evolve.

Prioritisation

I have the idea that I will be able to work on something from start to finish and complete every detail to 100%. This causes me to get tunnel vision on a specific feature or focus too deeply on a minor detail. I should have realised early on in the project that it's not realistic to complete every detail, and perhaps started to look at the big picture and prioritise things. Get all the big stuff working roughly and then do the fine tuning, rather than getting caught up with the fine tuning as I go. This is a big takeaway for me and something I will need to work on.

Conclusion

I'm pleased with the outcome of this project, the game functions well and is enjoyable to play. On a personal level, I'm happy with the progress I am making in game development and c++. My c++ skills are getting better and a lot faster, I've noticed already that I'm able to identify errors much quicker than early on in the semester. I do have the bad habits as mentioned above but will work hard to rectify these. This project was a very enjoyable experience and I am looking forward to seeing what we can achieve as a group in the next project.