Gam 320 Evaluation

Intro

Analysis

Moving from Unity to Unreal

Designing classes and hierarchy.

When we started work on the Unreal version of the game, we had a fairly clear idea of what we needed to implement, as we had the Unity prototype to guide us. However, due my lack of experience with the Unreal Engine, I initially attempted to construct a very similar class structure to the one we had in Unity. This caused me a lot of frustration as I grappled with Unreal’s Blueprint/C++ division and the huge number of classes already available in UE4. It took me the first week working in Unreal to really start to grasp how to separate functionality into C++ classes and blueprints derived from them.

Though this was in part caused by my lack of experience with Unreal, something that I continually tried to improve by reading the documentation and watching tutorials. However, my difficulty in designing the class architecture was compounded by essentially planning on the fly, with minimal use of UML or class flow diagrams to help. Although I made some attempts at using UML, I mostly opted to “Code and fix”, refactoring multiple times to eventually reach the desired outcome.

The biggest lesson I’ve taken from this is the importance of spending time to plan out the architecture of the project – especially the division between blueprints and C++ - in order to avoid refactoring and redundant work. Although we’ve managed so far, as we start development next semester, I will draw up a UML diagram each sprint to help visualise how the new features for that sprint will fit into the project, with the help of our other programmer.

Blueprints

Four or five weeks into the project, when I had started to become familiar with Unreal, I felt very comfortable using blueprints to quickly prototype systems, especially since many tutorials and UE4 documentation was designed for blueprints. However, this has made it very easy to fall into using blueprints for everything, and I have found it difficult to motivate myself to spend time refactoring blueprints into C++ when everything already works. This is almost certainly also influenced by the lack of planning that has gone into the class structure, but I feel it’s also due to wanting to see results and feeling like I’ve done something useful for the game. After spending so long on the Unity prototype, I’ve been eager to progress in Unreal, which has made me impatient, rather than spending time improving the foundations of the game.

Next semester, I will dedicate one day each week refactoring any features prototyped in blueprints that should be moved into C++ and ensuring that they still work correctly. I will suggest that my fellow programmer does something similar, so we both work on preventing a mess of quickly-made blueprints that are hard to maintain. If we keep this up each week, it should help make the code maintainable and more cohesive, as we’ll be tidying up and refactoring consistently.

Teaching version control to team.

During the first two weeks of development, many of the team weren’t fully comfortable using version control, which led to both work-in-progress and completed assets being absent from the project repository. To help with this, I led a short demonstration of using version control, assisted by our other programmer. Although the demonstration was a significant success, there were still a lot of questions and uncertainty around versioning that suggest that my demonstration could have been clearer.

Due to my familiarity with version control, I moved quite quickly and there used some terminology, such as Branches, Remotes etc, that may not have been obvious to those watching. I also only wrote a few brief bullet points to remind me what to talk about, which meant I occasionally had to backtrack to point out an important point that I had forgotten to talk about. Writing a more compressive overview of what I wanted to talk about would have helped me present in a more consistent manner, a written guide with pictures could also have been a helpful tool for the team.

Unreal compilation issues

One of the major setbacks we’ve experienced this semester has been an Unreal error where the project fails to compile and closes before loading. It has appeared seemingly at random and prevented many team members from working in the project for several weeks. With assistance from another programmer on the course we found a workaround where we compiled the project files in Visual Studio, on a computer where the project worked, then uploaded the compiled files to version control, so that the Unreal project is pre-compiled and doesn’t have to do so on the computers where it did not work.

Being able to prevent this error will save a lot of wasted time and allow the team to make more progress. Throughout both the current and next semesters, I will be doing a visual studio compile of the project and pushing it to version control to ensure that the project is always accessible. This has the added benefit of removing the need for the non-programmers to compile the project in Unreal, which can save a bit of time whenever they pull an updated version of the project.

Outro