Gam 320 Evaluation

Over the past seven weeks the group game project has been making steady progress and is starting to come together. However there have been multiple challenges and problems that have arisen, both personally and for the team as a whole. Although we he overcome the challenges we’ve faced, they have slowed development at times and on a personal level, reduced the quality of my work. During this evaluation I discuss some of these problems and set personal goals for next semester in order to avoid these issues in the future.

Flexibility

The team had decided to use Unreal from the beginning of the project, although we initially prototyped in Unity for speed. Due to technical issues with many of our computers that prevented us from using Unreal, we delayed the switch and continued working on the prototype even into week 3. Although this was largely caused by technical issues out of our control, I could have begun working on the project from home, on a working version of Unreal, even if the rest of the team were unable to. My reluctance to do so was in large part due to my familiarity with Unity, and the uncertainty of switching to a new engine.

This was clearly not the correct attitude, as although we fleshed out our prototype, we had already decided to move forwards with it. Instead, this just delayed the transition and more importantly, delayed the learning process of using Unreal. Switching between tools and being flexible enough to learn new things is a vital skill in the games industry, so during next semester, I will focus on learning and using Unreal’s new tools, rather than simply trying to use C++ to solve everything. I have already refactored a large portion of AI code to instead use the Behaviour Trees which has been extremely useful. Next semester I will spend an hour each week looking at Unreal’s example projects to see how they work and give me a better understanding of the tools available.

Design and Planning

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 - 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.

Version Control Guidance

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 comprehensive 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.

To help solidify the team’s understanding and willingness to work in version control, the coming semester I will be checking each week to see if people are using version control properly and offer assistance if they are not. If necessary, I will also make a short guide to using Sourcetree so people can reference that when unsure of what to do.

Overuse of Prototyping

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.

This would be a problem in industry, as the game is expected to meet standards of quality, not just in the finished result, but also in the code behind it. A desire to see results is a more selfish motivation, rather than thinking of the overall benefit of creating a more reliable product.

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.

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.

This error, and its solution, showed me the value of understanding the underlying mechanisms of the engine you’re working in, as it allows a much greater level of troubleshooting when something goes wrong. It particularly highlighted my lack of knowledge about how Unreal handles code, so I plan to spend half an hour every other week examining the UE4 source code to become even a little more familiar with how the engine works, as recommended to me by the programmer who helped solve the issue.

Conclusion

The project has been a success so far, moving forwards at a steady pace. As more systems and features are added to the game, the biggest challenge for me will be maintaining a logical and robust framework of C++ classes to support them, while planning ahead to make it expandable. I aim to learn from the problems I have encountered this semester and use the goals outlined in this evaluation to continue to improve the development process for next semester.