Victor Haskins

Game Programming 2

Lab 12 – Multithreading

Introduction

This last lab for our class was interesting as we didn't have an assisted section, but had instead a lesson on delegates and lambda functions on delegates to use in an on-your-own section, should we so choose. The meat of our coding this time was up to our discretion as we set up scripts to move objects with threading.

Methods

We went over the issue of delegates as an empty method to be filled in later by the coder. This ties in well with threads, as that is exactly what happens when a thread is created; the method to fill it is described elsewhere and after initialization, the code can be run with varying results or removed entirely to be replaced with other code. We then move onto lambda expressions, which are basically code instructions that have no method to contain them, but are inserted into the blank space of the thread.

With this new knowledge we are left to our own devices to create the code for our lab. We are given a few simple instructions and the rest is left to us. In a new Unity project, we place three cubes, and a sphere. The sphere gets a script to keep track of health that is decremented every time it makes contact with a cube and display this to the user on the screen. Then all objects receive a movement script that will move it in a direction and randomly change the direction every five seconds. The catch is that the movement calculation needed to be carried out by a thread.

I started out with the health issue by creating a UI canvas and placing a text object in the corner to display the current health and change it whenever it was altered. I then set up an OnTriggerEnter function on the movement script so I only had to add the rigidbody and tag to one object as I thought that was where one of my problems was. I later found out I was wrong, but I explain when I get there.

The movement script overall was the most confusing. I couldn't have the threads created modify transforms or access the time differentials so I was forced to find workarounds. I was also hung up on the language of the instructions for a while too, as it said the movement translating needed to be in the thread and took that to literally mean the translate function. I later figured out that the function needed to be given the start, stop, and delta time for a Lerp function within the thread and be monitored by a boolean value so that it works only when the main thread updates the position.

This is where it really got weird to me because even after figuring out all of these other issues, my game consistenly broke on me and crashed Unity, requiring a restart. After hours of research and trial and error involving trying to force abort sequences when the game was quit that didn't work, I noticed that all the abort sequences were trying to work on finished threads and mine were started as infinite loops. At this point I added another boolean value initialized to true that would govern the thread's run status and would be set to false as soon as the application was quit--right before the threads were aborted. This fixed my crash issues.

Conclusion

I had a devil of a time with this lab. I also knew that, not only would the lesson be important, I would have to work really hard to know this for use in the final project. The threading is pretty messed up in conjunction with Unity as the engine itself cannot handle threading. This means that any and all instances of Unity specific information, such as gathering information from a transform in-game or getting the time passage from the engine is prohibited by anything other than the main thread. All other threads need to wait for this information to be gathered or pushed for them and cannot directly affect the sources. Jeez, this was a lot to wrap my head around.

Questions

Post-Lab

* Explain what is displayed in the console and why it is displayed as it is:

|  |
| --- |
| void Start**()**  **{**  string text **=** "t1"**;**  Thread t1 **=** **new** Thread**(** **()** **=>** Debug**.**Log **(**text**));**  text **=** "t2"**;**  Thread t2 **=** **new** Thread **(()** **=>** Debug**.**Log **(**text**));**  t1**.**Start **();**  t2**.**Start **();**  **}** |

The text would be displayed as

t2

t2

The two different threads are made to print the information from the string "text" but are both started after the text string is altered. They would both be looking to the same string after the change instead of finding and holding the current information at the time of creation.