Levi Johnson

2022 Summer Research Under Dr. Anil Shende

https://github.com/lwjohnson/WFDS-Unity-Simulator

**Summer Research Final Report/Reflection**

**Introduction**

The focus of my summer research was Wildfire Spread VR Visualization using the FDS and WFDS Fire Simulators. This project is intended to explore the details of using the external fire spread simulator FDS/WFDS to simulate a fire spread and then visually show the spread in a VR environment. More specific to my time working on it, the project is also exploring the implementation of interaction into the VR environment that then affects the spread from the external simulation.

**Review of Past Work**

When the project was handed down to me, the project was close to ready for working on interaction. The project had terrain generation and initial fire placement from the input file. There was also the ability to move around using keyboard inputs, and you could look around using the VR headset.

There was also the ability to add fires in VR that would be added to the input file prior to starting the simulation, which does technically count as interaction, though this form does not have any difference from manually adding the fires to the input file itself outside of VR. Once started, Unity would then start a thread for WFDS to run in and wait for it to finish, then read in the fires from the lstoa file generated by WFDS and display them in the VR environment. There was no interaction made at any time after the initial run of WFDS, but it did have the beginnings of a restarting system in which it would modify the input file to prepare it for simulating the next interval of time, with some bugs.

**My Goals**

When beginning work on the project, my goals were a bit ambitious with the understanding that I would not achieve all of them, but at least make progress towards determining the feasibility of their implementation, particularly when it came to the several types of interactions being included, a piece that was one of the last things I would work on if time provided. My main goals were to:

1. Clean up and improve the project in its current state to make it easier to work with in the future
2. Fix the inconsistencies of the current Unity and Simulation back and forth so that it would continually run without any interaction
3. Implement a way to pause the visualization
4. Implement the ability to make interactions during this paused time
5. Restart the simulation with the new interactions
6. Have the visualization return to the normal back and forth post-interaction
7. Have this all occur in VR and using the VR hand controllers instead of keyboard and mouse input

Once these main goals were accomplished, my secondary goals were:

1. Switch from WFDS to FDS as the external simulation, as WFDS is likely to not be supported/updated in the future to match the FDS support/updates
2. Switch the process of running the simulation to use the restart capabilities of FDS that were not present in WFDS instead of starting from 0 on every simulation run
3. Again, make the pausing and interaction possible with FDS

Finally, some goals that were a bit more ambitious:

1. Increase the level set mode of the simulation so that the simulation produced more details and accurate fire spreads
2. Adding support for more complex obstacles, such as tress or trenches
3. Add several types of interaction, including trench digging, tree cutting, and water spray

**Preliminary Research**

Before diving deep into the project, I did spend time learning about the Unity game engine as well as the WFDS simulation. I spent about a week of my early time learning how to use Unity (which included the UI, understanding the structure of the game object system, exploring the tools provided by Unity, brushing up on my C#, and more small stuff like that) as well as learning the file structure of WFDS’s output, which mainly consisted of learning about the structure of the main input file as well as the lstoa file where fire spread information was stored.

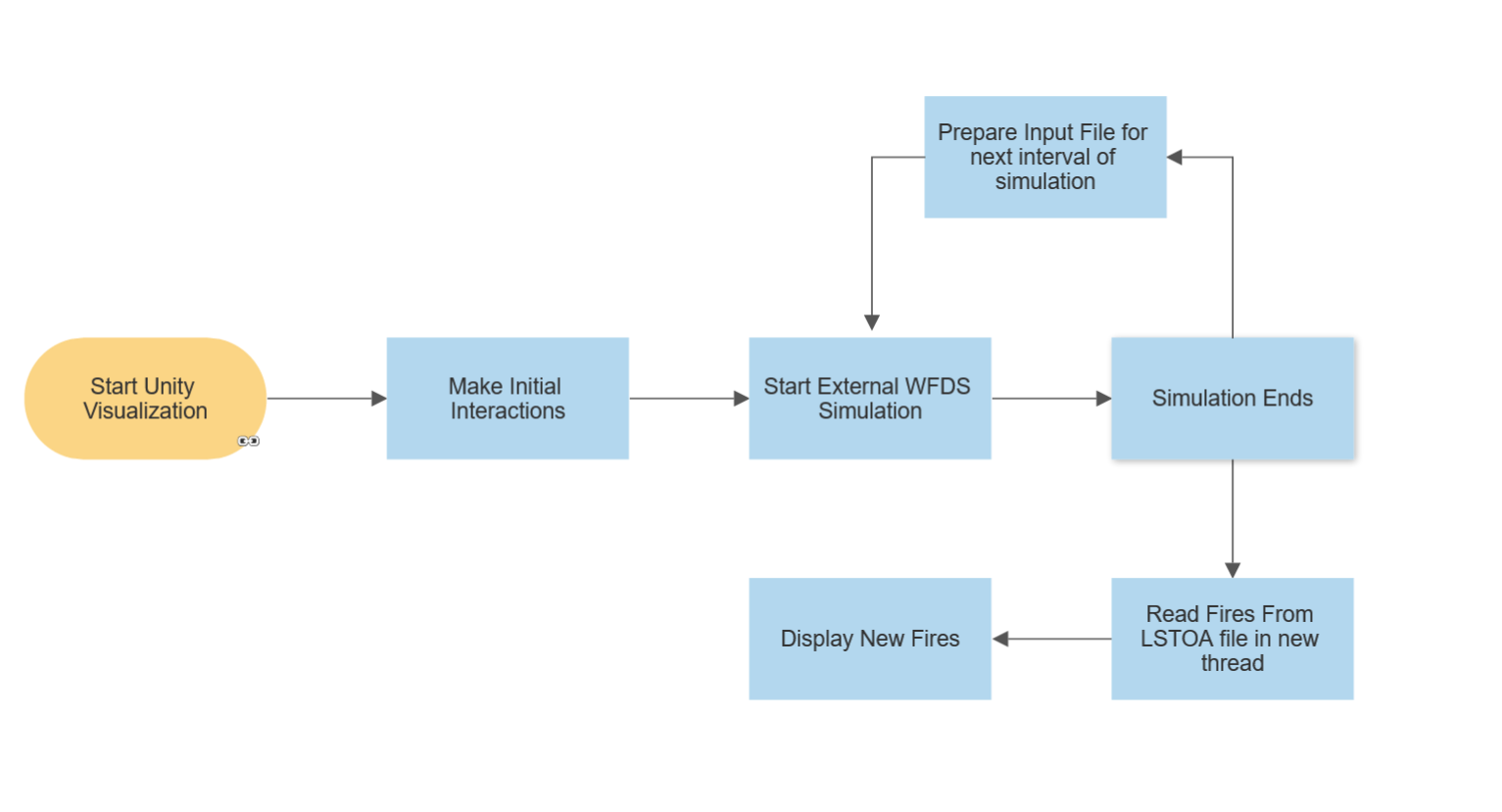
**My Work**

The start of my work in June was focused on goals 1 through 6 of the primary goals listed above. The cleanup and improvement of the code to begin with was short, as Cameron had done an excellent job keeping his code clean which made it easy to work with. Once I began working on the bugs with the flow between Unity and WFDS was when I first experienced having to write code that was new (by new I just mean things that were not already in place from Cameron or basic Unity functionality).

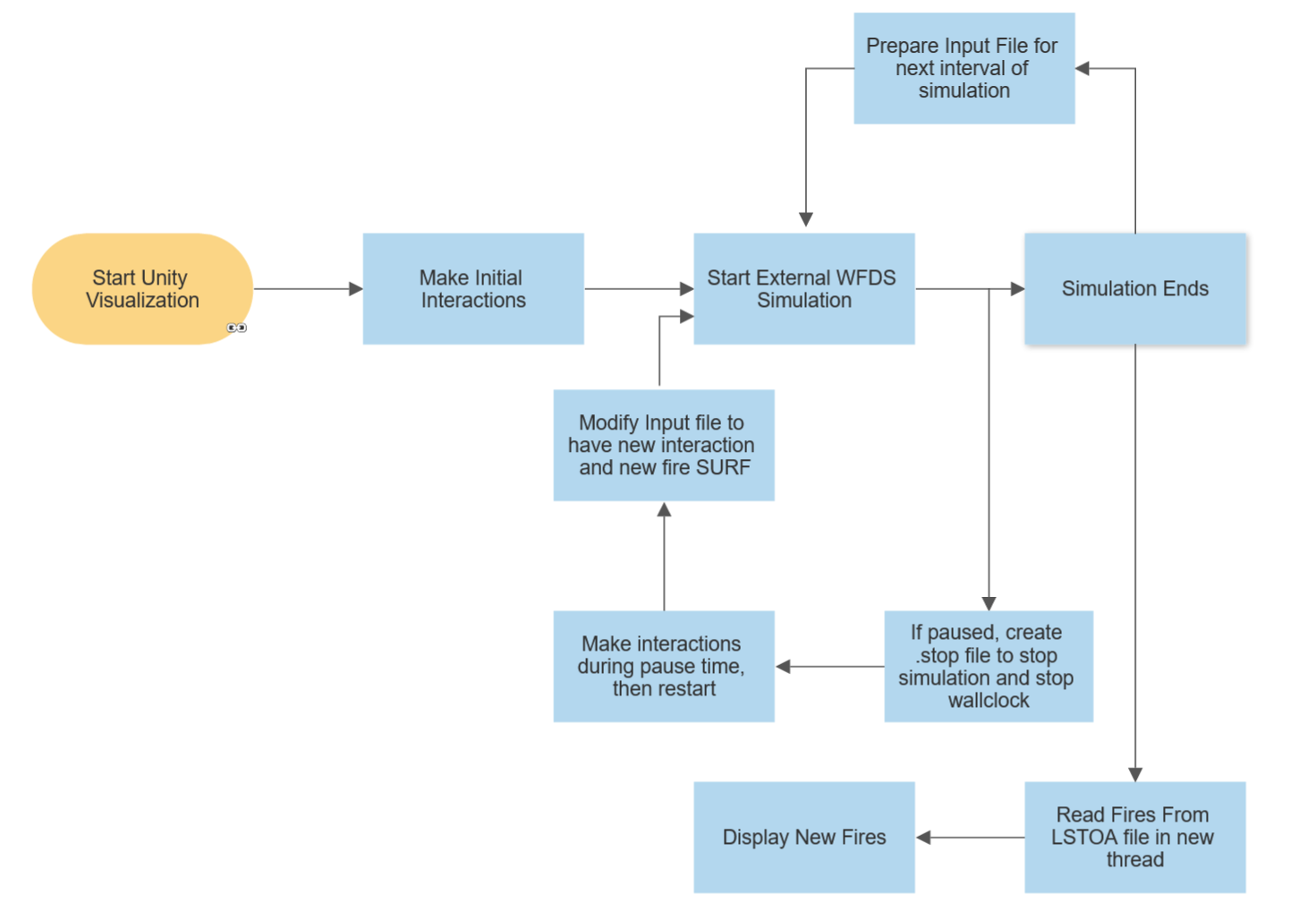
The process that was in place at the time was two threads that ran side by side, with one thread being the visualization in Unity and the reading in of fires and the other being the running of WFDS in the background. At first this system was fine, but it did produce lag spikes in the visualization due to the reading in fires piece, since Unity would stop to do that reading. My solution to this was to expand the reading fires piece to a new third thread that would trigger upon completion of a WFDS run. This new thread would be able to read in the fires without stopping the visualization, which made the visual aspect of the project much cleaner and freeze or lag less often.

This solution came with its own problems as well though. Since WFDS is external from the Unity project, the only communication they can have is through the output files from WFDS, which meant that only one of the two could be using a file at a given time. The simple solution was to simply make a copy of the lstoa file I needed to read from and read from that file while WFDS would continue to use the original lstoa file to write to.

After the flow between Unity and WFDS was working consistently, I moved on to the pausing and interaction piece. The pausing was simple, and simply used a boolean variable that I could switch back and forth via keyboard input to stop and resume the timer. One tricky problem was that since WFDS was running in a separate thread, it would have to be stopped as well once a pause was made. Alongside this I would need to prevent the user from resuming the visualization while WFDS was running, since the .stop file would still exist in the directory from trying to stop it when I paused to begin with. This problem had multiple timing aspects that had to be accounted for, and the timing for WFDS stopping could also vary depending on how far along into the simulation it was when the pause occurred. I ended up having a few simple boolean variables to keep up with the status of each piece and would block certain user interactions depending on the status of certain pieces.



Once I had the timings down, I now needed to add the interaction during that paused time. This interaction used the same system as the interaction at the beginning of the visualization used, so it was not complex to add. What was more complicated was to now add these interactions into the input file. I am only working with adding fires at this point, so I needed these fires to trigger at a certain time in the simulation. This meant that I could not use the same fire SURF (WFDS’s term for surface, such as a surface could be ‘grass’ or ‘stone’, this surface indicated a fire on the ground) as the initial fires did, since then it would have appeared at time 0 instead of the paused time. This required the creation of a new SURF with an ignition time equal to the current pause time so that it would appear to ignite at that time instead of at time 0 (This addition of a new SURF presented problems later once I switched over to FDS, but more about that later). Once the new SURF was added and the appropriate locations in the input file were marked with as this new SURF, I could then start WFDS again and after it finished simulating some into the future, the visualization and Unity to WFDS flow would now resume.



With this interaction complete, I had finished my main goals as listed prior, so now I moved on to making the switch to FDS. The two main reasons we wanted to move to FDS instead of WFDS as the external simulation were that WFDS may no longer be getting updates to match the FDS updates and that FDS had a restarting functionality that we wanted to take advantage of that WFDS did not have. In theory, the restart functionality would have made our visualization more efficient as far as less pause time because the restart functionality would not simulate back form 0 to our end time but could instead start from about where we paused, meaning less time spent simulating.

Most of the switch to FDS was not super complex, as the reading and executing portions of the project were only pulling a different input file designed for FDS and calling a FDS run command instead of a WFDS command. There is some extra information in the FDS input files as compared to the WFDS input files which must be parsed and used, such as a boolean variable for marking whether to restart or start from 0 or from where you stopped, but upgrades to the parsing we already had was simple.

The hard part for the switch to FDS was the SURF adding that I described above for my WFDS interactions. A problem arose where adding a new SURF to the input file and then trying to restart using the FDS restart functionality would cause a memory leak error for some reason I am unsure of, but regardless this meant I would need to modify my process flow to take this into account. The new process would now have to take some slightly modified steps when ending an interaction. This meant that when no interactions were made, FDS would use the restart functionality to save time over the restarting at 0 strategy, but when an interaction was made and a new SURF was added, I would restart from 0 and not use the restart from pause time functionality. This would of course mean that pauses directly after interaction would be longer, but in theory if FDS was as fast as WFDS then this would not be much of a problem.

Which brings me to the discovered downsides of FDS that threw our plan for a loop. As mentioned before, we were speculating around the idea that FDS would be as fast as WFDS, but we were unfortunately very mistaken. For comparison, from some data I collected using a stopwatch, my 200 by 200 cell input, and a 60-second interval to simulate, WFDS would take about 15 seconds to simulate it, while FDS would take upwards of 50 to 60 seconds. These times are for the average of simulation the first 0 to 60 second interval, and the 60 to 120 second interval took WFDS about 17 seconds and FDS about 60 to 70 seconds using the “restart from 0 every time” strategy. What is even worse though is that once I switched from that strategy to using the FDS restart functionality, the 60 to 120 second interval would now take upwards of 150 to 160 seconds to simulate.

This development made us keenly aware that this would present huge issues for the simulation, since it would be guaranteed that when using FDS, there would be pauses at the end of every interval in the visualization to wait for FDS to catch up. This development halted my progress in its tracks and is where I have concluded my research.

Since the inspiration behind the project is to be a training simulation that felt like a VR game, these pauses, in my opinion, would ruin the immersion that people would experience and fail to give people a feeling for the cause-effect relationships that their interactions with the fire created. While I personally did not feel this would be beneficial, it may still prove useful to the Forest Service in some way, which is a topic that requires further discussion with them about their goals for this project from Dr. Shende and whomever takes over the project from here.

**Future Directions and Work**

As this project continues forward, I do see some directions that could be taken and some future work that can be done. One direction I see is the potential for a different external fire simulation besides FDS or WFDS. The desired traits of a simulation that I think would work well for a project like this would be:

1. A fast simulation time, even if that means it may be less accurate than FDS is (though accuracy is a goal of the project, so it should still have accuracy that follows real-world spread as opposed to a fire spread engine you might see in a video game)
2. Optimized to handle larger, less detailed areas such as a large wilderness environment as opposed to more detailed areas like the inside of a room or building
3. A restart functionality like FDS that does not have the large simulation time that FDS’s restart has.
4. Written in C# (for Unity) or C++ (for Unreal Engine) so that it can be integrated into the project itself instead of being an external simulation.

While a new simulation might not be desired, as the continued development and accuracy of FDS are desirable traits themselves, it could open doors to a smoother experience.

If we continue to use FDS, there is also potential for more interaction that can be added in future work. Some ideas we have produced are digging trenches, cutting trees, and dousing fires, but more interactions could be possible in FDS.