8. Testing

All the math has been tested on implementation with confirmed values from apt references. I also made great use of the continuing running of the AnimationTimer, which provides with changing realtime values, which made it in most cases rather easy to find out the problem. I also made the file readers throw out useful exceptions when the input data is not suitable. Otherwise as there is very limited user input, I do not think making Unit Tests for the user is very useful. I did the testing I thought, more of it than I thought, but mostly in the form I thought.

9. Known faults and short-comings

The clipping of objects on the sides and behind the camera. As seen from the difference on how it looks with clipping of, it is certainly doing a lot, but just a couple of triangles glitching over to the wrong side of camera is a program-breaking bug in the amount of disruption it causes to the player. I couldn’t fix it and neither did the main assistant, not to any fault of either in my honest opinion. In the program using – and + keys it is possible to alter the “e” value of the projection which corresponds to a certain kind of plane. This in my testing has a surprisingly limited effect on the amount of unprompted clipping. Then when I got the intersection math to work, I couldn’t find a plane where it would clip the objects in a desired way, so I set it to the best I could and it’s there now. I think going through all the algorithms with a fine comb is the only possible way to try to fix this. If that wouldn’t work, the entire core of the project would have to be switched to a different projection, but that could be manageable as it is just a piece in the puzzle. Also the intersection algorithm is a bit iffy in itself. It is supposed to give a factorial of how long along the original line the intersection point is, but these values do not all the time be accurate. But as the intersection point has proven to be correct based on my tests and I’m not interested in the factorial, I’m not too conserned about this. And on a general rule of thumb for most of the maths I haven’t tested all the possible edge cases, but I’ve tried to combat this in some places doing the calculations up to a certain accuracy.

11. Differences from the plan, working order and schedule

Instead of 2 to 3 coordinates per row, I used 9 to form a single triangle per CSV row. AD-keys are used to move sideways instead of turn, which is entirely handled by the mouse. The class structure is vastly different as in plans to accommodate the different problems risen during development. Minimap wouldn’t have been a very good visualization of the environment as some objects are horizontal, some vertical and others somewhere in between, so visualizing the data in a way that a player could get something useful out seemed unlikely, so I ended up not implementing that. On the other hand during the development I implemented a lot of user customization and regarding the UI, I created a stamina bar to better visualize the amount of stamina left, instead of just trying to feel how fast the player was going, or how long did it take to recover. I’ve decided to also scrape all dependencies regarding calculations except for scala.math which provides some trigonometry, max/min and some others. The matrices runs entirely with my self-defined vector and matrix math. Only real dependency is Scala FX, which requires difficult installation, when not familiar with sbt, but a ready-made jar file is included for the user. Taken with a grain of salt, the original class structure witheld rather well, with the emphasis on the flow of the data from CSV through various methods to the ScalaFx app which draws it. There was a lot of additions, for example the menus, styling, and a general math file to package in one place a lot of the heavy calculations not inherently part of the triangle of VectorVer, in itself.

Then regarding the working order. I only include major progress events on the app, and completely skip all the parts not ending up on the final product.

I started of with getting the ScalaFx dependency to work. Then moved on to work on a bare-bones app, and basic matrix calculations to make the projection work, as well as CSVMaker to make something to test out the program. Got the file reading and writing to work, as well as the triangle sort based on their distance. In case you speak Finnish, please see the “Kaiken pahan alku ja juuri” under progress. Continuing when I got the projection to work, I focused on game mechanics, such as walking, turning, all the other stuff. Then I started doing the intersection math. That turned out to be a impossible task so I switched my focus on the menus. First the ScalaFx side of buttons and scene interactions, followed by styling and finalised by the screen size manipulations.

Regarding time usage.

Grosely underestimated the amount of time I would use on this project. First of all I also had far more of other schoolwork when I originally planned on doing this. After the exam week, I’ve mainly worked on this day in day out. Probably in the 300-400 hours slot right now. There has been certain major contributing factors in this. First of all, ScalaFx is not a pleasure to work with. The main documentation leaves a lot to be desired, and only after much sweat I fully understood the value of using both it and JavaFx docs, which still isnt perfect as I then cannot find the syntax for ScalaFx methods and so on. Secondly the projection has required a lot of care put into it. Things would have been much easier using 2D-fisheye projection in which I could have done all the math in 2D and just drawn walls in regard to a kind of blue print of lines. From there it is natural to jump to the third and the most important point. I wanted the project to be fully scalable to be able to draw any kind of 3D object, the player to move freely inside the world, with a lot of customization to make sure the experience is excellent for the user. These all came with a price tag of a lot of programming time and difficulty. I understood the value of limited project scope, but in this case as I just wanted to learn programming and try out and implement everything, I think could make the project better.

12. Conclusion

I think the project goes above and beyond the scope present by the project definition. The class structure is well defined with a minimal amount of anything not requiring ScalaFx done in the main file. Otherwise the different objects each serve their purpose and are well confined in their integral scope. The only failure on the project is the intersection math. To problem lies deep within the projection and its “e” plane, which I could not alter in any meaningful way to curtain the clipping. If I’d do this project all over again, I still wouldn’t do it just according to the scope with limited 2D coordinates, but I would do it on a different projection. Otherwise I would do everything the same. Using ScalaFx is the best option available, when the language must be Scala. Without this limitation I would have probably first looked at the possibility of just making it web-based, which of course I would have had to learn as well. CSV is the perfect file for this simple 3D application, but if I would be doing a much bigger world, I would format it to make use of duplicate points. In future there is no options for continuing development because of the problems with the projection and obj file loading, but if those got to work, I would try to optimise the running, to do less iterations over the triangles before they are ready to be displayed.