

Report TSBK07

By Oscar Magnusson & Tobias Martinsson

1. Introduction and what was implemented

What we wanted to achieve was to be able to create a labyrinth specified in a text file. To navigate the labyrinth the user should control the camera with typical FPS-controls, moving in a 2D-plane with the WASD-keys and rotating the “camera” with the mouse. To make the labyrinth more interesting to navigate through we wanted to create an interesting effect on the walls and floor. We took inspiration from the movie *The Matrix* and it's moving wall effect, that is supposed to look like a visualization of a data flow. The final mandatory feature we had in our project was for the “player” or the “camera” to have some sort of collision with the walls, so the player could not cheat.

We also had some optional requirements, some of which we implemented in the final version of the project. The ones which we implemented were “Humans”(or agents from the movie) which chases the player when they have line of sight. We also implemented a goal in the labyrinth that when reached takes the player to the next map. The goal was illustrated by, keeping with the theme of the movie, a phone booth.

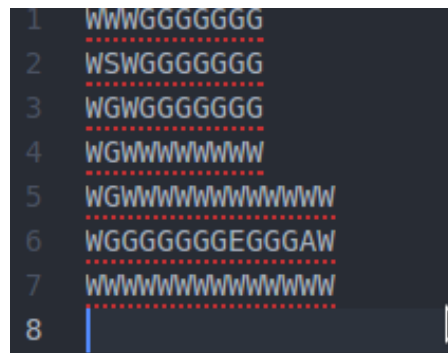
The optional tasks that we did not implement were that more agents appeared in the map and that they could fire bullets at the player. Having more than one agent is achievable but did not seems like a good use of time since we did not take multiple agents into account when structuring the code until the end of the project. The final optional features that was not implemented was a randomly generated labyrinth and the ability to stop or slow down time for the bullets.

2. Implementation: details and problems

What we started with was the structure of the labyrinth and its walls. We wanted 5 different types of “wall”: the floor, top wall, bottom wall, left wall and right wall. We created these by creating a square, consisting of 2 triangles, in OpenGL and rotating and moving it around. Basically we used one base square and transformed it to our desired square type.

The problem that came up after we created the base squares was how to represent them in a text file. We tried having different characters for the 5 different types. We tried to have special types for corners that was assembled with two squares. This resulted in many problems when designing a map. How should we represent the floor in a text file? How should we handle the grid? How should we create open spaces? At first the floor was created with regards to the width and height of the map and later we had one floor piece attached to every wall piece. Neither of these solutions was optimal and made it really hard to design maps in an efficient way. What we finally settled on was to have 2 main types: “W” for walls and “G” for floor. The walls consisted of four squares, the top, bottom, left and right. Basically a hollow box that took up one square in our grid. The floor was one square on the ground in the grid. This made it much easier to design maps since one

character in the text file was now exactly on box in our map grid. An example of how a map can be described in a text file can be seen in the image below.



Example of map in text file

2.1 Texture effect

To create the desired moving texture effect we started out by applying the texture “rutor.tga”(image below) to get a sense of how we wanted to cut the vertical and horizontal lines (basically how big the numbers where suppose to be). After that we separated the texture into columns that moved at different speeds. What we did was to move the different columns up and down based on a time variant variable.

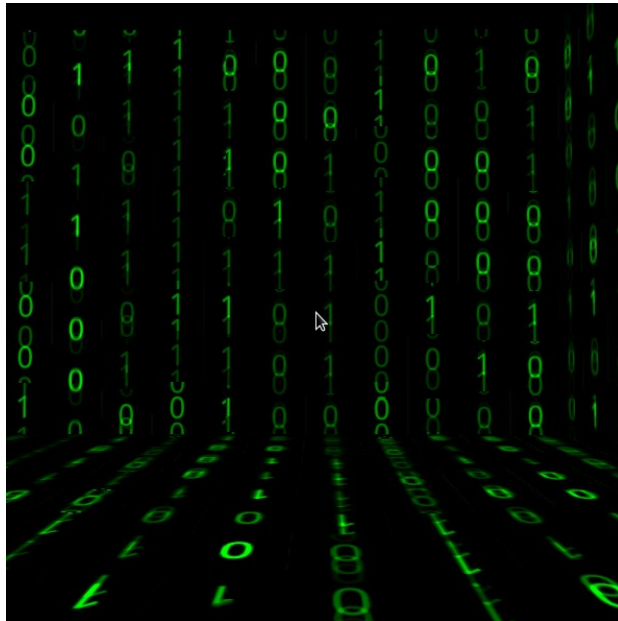
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

rutor.tga

After we got the texture moving the next problem was to assign 0s and 1s to the different position in our horizontally and vertically chopped up wall. We did this by creating 2 different texture files one for the 1 and one for the 0. We then decided for the wall to have 8x8 zeroes or ones, we then randomized a list (containing the information of which number is supposed to be in which position) for each of the walls. Now we had a wall with numbers that moved, but the moved at the same speed. So what we did was to have two different random speeds for each of the walls and multiplied these two with different values to have seemingly random varying speeds and directions(up and down) for the different columns.

To finalize the effect we added 2 more identical layers to the wall on top of the previous layer. These 2 layers moved at different speeds, their speed was a multiplication of the

speed of the first layer, ensuring that the speeds would not match. An image of the final wall and floor effect can be seen in the image below.



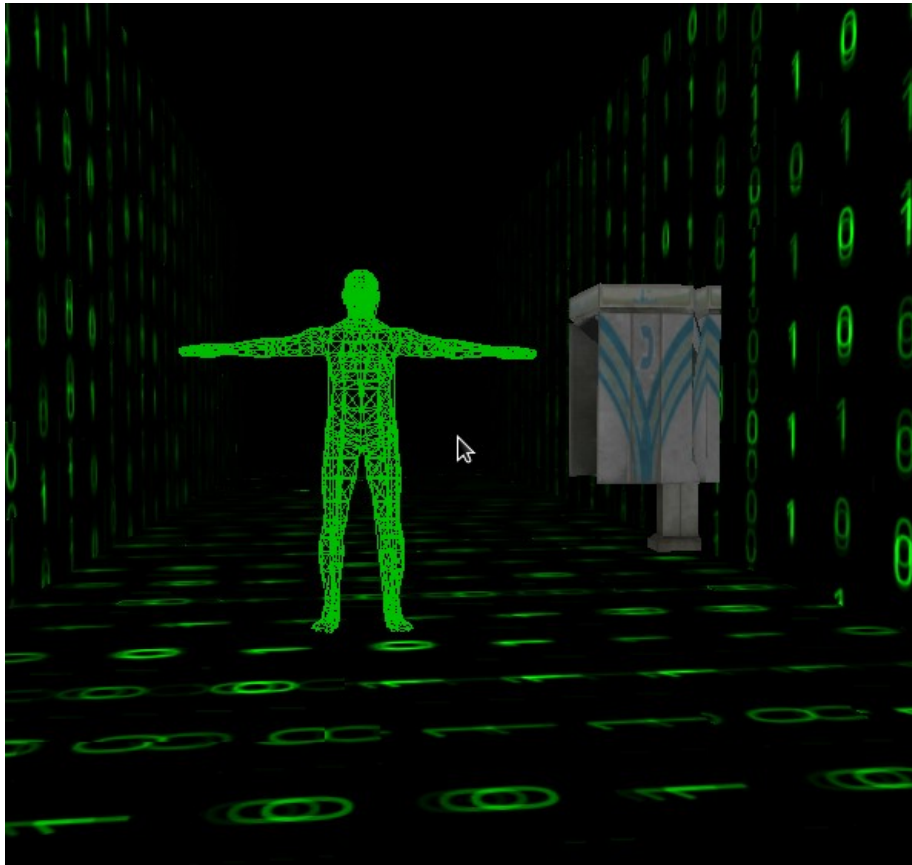
Wall and floor effect

2.2 Wall collisions, enemy and goal

Now that we had the walls in place, we wanted to add collisions to prevent our player from running through the walls. Our first thoughts were to find out what coordinates the walls had in X and Z axis, and make a comparison of the players X and Z coordinates. This part was a bit of trial and error, as we adjusted the bounds-distance variables to make the camera not “clip” into the wall. Since our walls are axis-aligned we only needed to determine what type of wall we were dealing with and check the distance along the axis that was orthogonal to the alignment axis (X- or Z-axis).

This created a problem at first since we now were checking the distance all along the alignment axis. This was solved by limiting the distance check to the part of the axis that the wall was on.

We added a human model to represent our enemy or agent in the labyrinth. Since we wanted to keep with the movie's theme we drew the model as a wire frame model and colored it green. We also added a phone booth model, with texture, to represent our goal in the labyrinth. Both of the models can be seen in the image below. To check if the player was colliding with the phone booth or the enemy we calculated the distance between the camera and the two objects, in XZ-plane and triggered our reset or goal event based on the distance.

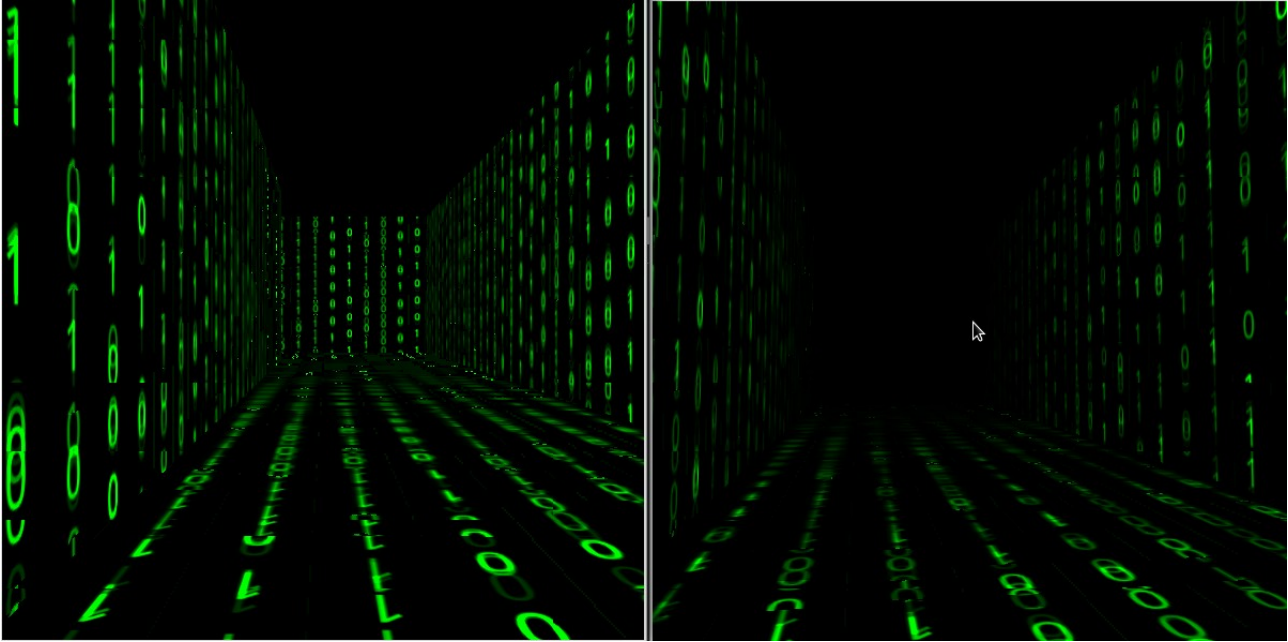


Agent model(LEFT) and phone booth(RIGHT)

2.3 Light and shading

The world in which the labyrinth is contained is mostly black. Therefore we realised we could achieve our desired light effect with rather simple means. What we did was to randomly generate a list which contained data that represent the brightness of the numbers. Each number in the 8x8 grid has a light value in the list.

To further enhance the experience of navigating the labyrinth we wanted to create a sort of fog or shade effect. This implies that you as a player can't see further than a given distance. The implementation causes a effect that makes it look like there is a light bulb on the camera. This effect is applied to all of the objects in the world. The effect is created by assuming that closest to player is bright(value of 1) and subtracting a value that is proportional to the distance from the player to the object(wall/agent/phone booth). A comparison image of a corridor in the labyrinth with and without the effect can be seen in the image below.



Comparison of the shade effect

3. Conclusions and what could be improved

We believe that our final project is a good representation of our initial ideas. We are very satisfied with the outcome.

We still have some issues that remains in the project. One of our more noticeable problems or bugs is when the player moves directly into a outward facing corner at about 45 degree angle. The player would then “peak” through the wall, seeing what is on the other side. This is of course not intended, but it is rarely occurring.

Another problem is our two imported models, which we did not create ourselves. The phone booth has some issues with mapping the texture, which is noticeable when up close. The agent models has a similar issue with how the vertices are linked, which creates some errors in the model. The agent model could also be animated to give the effect of moving, in the current project it is in a T-pose.