**Implementation:**

As mentioned above are we using the code from a previous project because of its high efficiency and therefore good performance.

Of course we had to make several changes to the code so that it is suitable for our simulation in order to answer our question setting.

In this chapter we will explain these changes. For the exact understanding of the original code and the process of optimization, please refer to the documentation, “Modeling of Evacuation Siutations in a multi-level building”, chapter 5 and 9 **[2].**

**1. exits are not on lowest floor**

Reason:

The first change which was necessary was due to the fact, that the exits in our simulation are not simply on the lowest floor.

Assumption:

Here we assume, that the agents above the exit floor only move down, and those under it only upstairs.

Function: applyForcesAndMove.m,

new varables:

🡪 To define the floor in which the exits are we introduce a new variable „floor\_exit“ in the config file.

Modifications:

Since we know now in which floor the exits are, we can modify the code easily by splitting the loop, in which we calculate the forces and move the agents, in two parts.

First we loop over all floors higher than the exit floor, in which all agents only have to use the stairs down, or in case they reached the exit floor, the exits.

And second, we do the same for all the agents in floors lower than the exit floor, now only checking if an agent reached a staircase up or an exit.

This modification keeps the code fast, since we do not have to loop twice over all floors. We can also keep the simple concept of a vector of booleans.

For each agent there is one number:

🡪if agent reached a staircase and therefore changes floor: 1

🡪else 0

The assumption also is a great simplification for the pictures since we don’t have to mess with the problem of having „overlapping“ stairs.

**2.** **having different exits**

Reason:Since the exits model lifeboats, which can only hold a limited number of agents, we had to find a way, to differ the exits from each other and to assign a specific number to every exit.

Assumption:

Function:loadConfig.m

new varables:

🡪exit\_count, to define the number of exits

🡪for each exit: exit\_1\_nr, to define the number of agents it can hold

🡪To know how many agents can exit in one specific exit, we introduced a matrix exit\_nr matrix, where the number of agents that can exit is indicated in each pixel

Modifications:

🡪We had to make some changed in the decoding of the pictures. The aim was to change as little as possible to the original code. It was clear that we’re going to nee das much different colours as we have different exits, to be able to distinguish them during the simulation. By defining every pixel that has red value=0, blue value=0 and green value not zero, we can have a lot of different colored exits.

-->how we implement the matrix…different green values for different exits

**3. closing exits during simulation**

Reason:

Assumption:

Function:

new varables:

Modifications: