Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Burning Man:
Simulating the behaviour of people in the event of fire

Project Report

Nico Hauser, Andri Horat, Elias Schmid, Jonas Spieler

Zurich
December 2019

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Nico Hauser

Andri Horat

Elias Schmid

Jonas Spieler

Contents

1 Abstract

This paper describes a model for the behaviour of people in the event of fire and it's simulation. The model tries to incorporate the general repulsion of people standing too close, the formation of groups in panic situations, the placement of exit signs and many smaller factor such as different weights and ages. In order to make observations on the defined model, it is simulated using a physics engine using javascript that makes it accessible from any place at any time and allows for reproducible and verifiable results. The paper also defines a general format for describing a ground plan of a building that can directly be imported into the model simulation. Combined with the right simulation parameters, this features allows the evaluation of the safety in buildings in the case of fire.

2 Introduction and Motivations

This paper was created during the course *Agent-Based Modeling and Social System Simulation* held by Dr. Nino Antulov-Fantulin and Thomas Asikis at ETH Zurich. The goal was to create a project matching the title of the course meaning the modelling of a complex system where humans are the agents. A complex system is in contrast to complicated system not per se hard to implement but rather consists of many simple small parts that on themselves act based on simple rules. The fact that there are a lot of these so-called agents gives rise to behaviours of the whole system that are not always easy to predict or even understand even if the simple rules of a single agent are well-known. As computers keep getting faster and new technology is developed every day, simulations are a good way of understanding these complex systems by playing with the available parameters while looking for emerging patterns and applying them to the real world.

In particular this paper describes and simulates the event of fire in a building that tries to incorporate many parameters an evacuation in real life depends on. When developing the model and simulation it was a primary goal to have every constant that was either empirically chosen or arbitrarily set to be an adjustable parameter. This allows the simulation of a much larger set of situations that may be needed in real life. Even the building is dynamic and can be changed by creating an appropriate file describing it. In the future it may even be possible to automate this process and directly parse a pdf ground plan which would allow even more people to take advantage of this model by testing their building for possible issues in the event of fire.

We also would like to add a side note about the project title *burning man*. When thinking about a project idea, it was clear quickly that an emergency situation would

be rather interesting to all of us but we didn't immediately know where the evacuation should take place, in a small house, a large building or an event such as a festival. As burning man is the name of a famous festival in Nevada that takes place in the middle of a desert. After some discussions we settled for the evacuation of a building in the case of fire and when thinking about a new name, we realized the name is still kind of fitting and even funny if you have a little bit of dark humour. As software projects often have similar names we thought we'd just leave it that way though it should be made clear that the aim of this project is of course to reduce the number of 'burning men'.

3 Description of the Model

The model is separated into different pieces. First of all we had to create a model for the building in which the evacuation takes place. Next a model for the individual agents, here representing humans, had to be created and evaluated.

3.1 Building model

Before agents can react to a fire inside a building, a building has to be defined and created. As the simulation in the end will take place on a computer screen, we first of all have to give the building or the *map* a `width` and a `height`.

The next obvious property any building has are walls. Walls define rooms, corridors and give the map its shape. We define the walls to be lines drawn from some coordinate (x_1, y_1) to some other coordinate (x_2, y_2). As the simulation takes place in a physics engine, the walls can't be just line but must be a rectangle of some sort and thus the model also requires a `wallThickness` property. For now this property is the same for all walls, even though this may not be accurate it almost wont't affect the result of the simulation so it is a appropriate simplification. Such a setting would already be enough for the agents to be randomly moving around and interact with each other but such a simulation isn't even remotely interesting for our purpose as the general behaviour of agents in an emergency is pretty clear. First we want to indicate the location of the doors so that the agents can find the next exit without having us to place safety signs into the rooms themselves. Outside the rooms, safety signs should be used for orientation. Both, the doors and the signs, have the property `orientation` that gives a normal vector for the direction the door or sign is visible from. In `tables` we can define additional obstacles which in our case are tables, thus also the naming. The definition consists of two points of a rectangle, the top left and the bottom right corner. These obstacles are different from walls as do not obstract the line of sight when looking for a safety signs. When following the safety signs,

at some point a safe zone or escape zone is or at least should be reached where the agents are removed from the simulation. These `despawnZones` are again described by the top left and the bottom right corner of a rectangle.

```
{
    width  : 1900,
    height : 1100,
    wallThickness: 10,
    walls  : [
       [{ x: 0, y: 0 },    { x: 1900, y: 0 }],
       [{ x: 0, y: 1100 }, { x: 1900, y: 1100 }],
       ...
    ],
    doors: [
        {
            position    : { x: 85, y: 425 },
            orientation : { x: 0, y: -1 }
        },
        {
            position    : { x: 445, y: 425 },
            orientation : { x: 0, y: -1 }
        },
        ...
    ],
    signs: [
        {
            position: { x: 550, y: 538 },
            orientation: { x: -1, y: 0 }
        },
        {
            position: { x: 1125, y: 538 },
            orientation: { x: -1, y: 0 }
        },
        ...
    ],
    tables: [
        [{ x: 70, y: 70 },  { x: 110, y: 170 }],
        [{ x: 70, y: 225 }, { x: 110, y: 325 }],
        ...
    ],
    despawnZone: [
        [{ x: 1275, y: 115 }, { x: 1585, y: 420 }]
    ],
    spawnPoints: [
        { x: 200, y: 538 },
        { x: 900, y: 520 },
        ...
    ],
    fires: [{position: { x: 20, y: 450 }}, {position: { x: 20, y: 500 }}],
}
```

3.2 Agent model

The agents representing the people are approximated as a circle in the physics engine. They have the following properties, most of which are generated randomly based ond a normal distribution with selected mean and standard deviation values. To have more realistic values we cap the random distribution within the standard deviation as the actual probability of some value far away would otherwise not be zero, just very small.

First we calculate the basic intrinsic properties `weight, fitness` and `age` based on the described capped normal distribution. The radius or size of the agent is then also chosen from the capped normal distrubution but is then also multiplied with the *normalized* `weight` and divided by the *normalized* `fitness`. By *normalized* we refer to the proportion of the random value to the mean for the given value. This factor $\frac{1}{|\texttt{agility}|} = \frac{|\texttt{weight}|}{|fitness|}$ (the $|\cdot|$ refers to the *normalized* value) or rather its inverse is also used when generating the `maxVelocity` and the `maxAcceleration`, both of these values are multipled with this factor `agility` after being randomly chosen from the respective capped normal distrubution. As visual sight is a very important in emergency situations for finding safety signs and can also be distrubed by for example smoke, we also generate the property `visualRange` from another capped normal distrubution whose result is then divided by the *normalized* age.

4 Implementation

The implementation of this model is done in javascript using a game library called phaser that already includes a physics engine and has means of drawing the result onto a `HTML` canvas. Phaser uses so called *scenes* that themself contain so called `GameObjects`. These `GameObjects` can either be just drawn onto the canvas or also be added to the physics engine which allows them to interact with the world. The walls are simple rectangles with enabled physics that are grouped together in a `Phaser.GameObjects.Group`. As the agents are approximated by circles, the extend the class `Phaser.GameObjects.Arc`, add themselves directly to the scene and enable physics. All agents are also grouped together which now allows us to tell the phaser physics engine to calculate collisions between the the groups and also within the group of our agents as they can run into each other. In contrast the doors and signs are simply drawn and not added to the physics engine as they are just there for interpretating the simulation. The tables, despawnzones and fires are added analogously.

4.1 Agent behaviour

As phaser includes a physics engine we can make use of this by determening the behaviour of the agents based on forces acting on them where the final movement is the vector addition of all these forces. Phaser accepts a `update` function that is called every time a new frame is calculated. In this callback function the following forces are calculated:

- Wall repulsion

  To model human behaviour, agents should in general not like to stand too close to walls which means we have to find the closest wall and check whether this distance is acceptable. If that's not the case, a force perpendicular to the wall and proportional to the distance is applied to the agent, forcing him to move away. The tricky part here is that the walls are line segments and not infinite lines, so the general math for calculating distances had to be changed and a check whether an agent is in front of the wall had to be added.

- Agent repulsion

  In general people do not like to stand close to each other, especially not in emergency situations. To simulate this behaviour an force exponential in the inverse of the distance is applied to agents standing close to each other.

- Agent attraction

  Even tough people do not like to stand close to each other, they still tend to form groups and do not like to act on their own. As the exponential repulsion only kicks in for small distances we can also add a linear attraction between the agents, forcing them to form groups. As soon as the linear attraction and the exponential repulsion balance out, the two affected agents will try to keep this distance.

- Target attraction

  Last but definitely not least, the agents have to do something and not just stand around. In the best case, the agents should try to reach one of the defined escape zones. In real life, safety signs guide people to these safe zones so our simulations tries to copy this behaviour by raytracing to all signs and check whether the sight is obstructed by a wall. Additionally it has to be checked wheter the sign can be seen from a given position as they should only be followed in one direction. A force in the direction of the closest not-obstructed sign is then applied to the agent hopefully guiding him in the direction of the escape zone. In the implementation section it was mentioned that there are also doors

having the same behaviour as signs but they are differently marked as they do not represent a safety sign but rather the intuitive behaviour of people leaving the current room they're in. Both of these forces are made visible by drawing a line to the target the raytracing algorithm selected.

5 Simulation Results and Discussion

5.1 Choosing start parameters

5.2 Interpretation of the simulation result

5.2.1 Clogging

When running the simulation, one might notice that the agents competing for a door are closer together. This is due to the fact, that all agents are pushing towards the door, and hence pushing them together. It worsens the situation, as the friction between the agents increases, which slows down the escape.

5.2.2 Keeping the distance

Interestingly, we can observe how agents are pushed back into rooms. Sometimes one could even witness how some agents use another path to avoid the crowd.

5.2.3 'Faster-is-Slower effect'

There are already many published papers regarding the social-force-model. Of which many describe the 'Faster-is-Slower effect' [1, 2]. The 'Faster-is-Slower effect' describes the phenomenon where the escape through an exit door is slower if the agents are pushing harder. When running our simulation on our default map, we did not observe this effect. The chances are, that this is due to the long corridor, where the agents make up time with a higher speed.

[1] Dirk Helbing, Ills Farkas, and Tams Vicsek. Simulating dynamic features of escape panic. *Nature*, 407:487–490, 09 2000.

[2] Peng Wang. *Understanding Social-Force Model in Psychological Principles of Collective Behavior*. PhD thesis, 05 2016.