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 Environment model

Before agents can react to a fire an environment has to be created which in our case is a building but is in general not restricted to buildings and could thus also be applied to other settings such as a festival. As the environment is one of the parameters that in a successful model should be easy to change, a subset of a preexisting format definition is used. The environment consists of the so-called object layers `signs`, `agents`, `doors`, `physical-walls`, `navmesh` and `despawn-zones`. The `signs` and `doors` layers contain points indicating the position of the safety signs and the doors. They are separate as they represent different things. The safety signs represent actual signs that the agents follow should they be visible. The location of the doors on the other hand are remembered by the agents and thus do not have to be visible for the agent to know it's location. In addition both signs can have the two properties `orientationX` and `orientationY` indicating the direction of a 180 degree angle from which the sign is visible. Another property signs can have is `radius` which, if defined, overrides the default radius for marking a sign as visited. The reasons will be made clear in the description of the agent model. The `agents` layer also consists of points indicating the spawn points of the agents. `physical-walls`, as the name might suggest, defines the location of the walls using rectangles. This layer is used for giving the environment a shape as the agents cannot walk through these walls. In addition this layer is used for the raytracing algorithm as the agents should not be able to see signs if there's a wall in between but more to that in the agent model description. The `navmesh` layer defines the navigation mesh mode of rectangles. This mesh defines the points

an agent can use for navigation when walking into the direction of a sign or a door. The details of this navigation mesh are also described in the next section about the agent model. Last but not least there's the `despawn-zones` layer where rectangles are used to define areas that will, when colliding with an agent, remove it from the simulation and count them as in safety.

3.2 Agent model

The agents representing the people are approximated as a circle in the physics engine. Each agent has different intrinsic properties, most of which are generated randomly based on 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 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 distribution whose result is then divided by the *normalized* age.

After the generation of these random values, the agents are a sole product of their environment, their behaviour is based on a set of simple rules.

- Wall repulsion

  To model human behaviour, agents should in general prefer to move away from walls as most humans prefer to have some space around them.

- Agent repulsion

  For the same reason people do not like to stand close to each other, especially not in emergency situations. To model this, the agents will thus move away from each other if their distance is small.

- 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. Thus the agents are instructed to move closer together as long as the distance between them is in the acceptable range.

- 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 the agents are instructed to always be on the lookout for visible signs guiding them to safety. In real life not all small rooms do have safety signs in them as for humans the only viable option in an emergency is to first leave the room they're in. As the agents should follow this behaviour, doors function similar to signs except that they do not have to be visible. This allows to simulate some kind of memory of the agents and enables them to leave the room even without have seen a safety sign. Another type of memory that is simulated is the remembrance of already visited signs and doors as real humans will not follow the same sign twice if the direction the sign points in didn't lead to an exit the first time around. An exception is made to this, when an agent doesn't have any target anymore they "forget" everything they visited before and start from scratch. This allows for some more complex behaviours such as an agent being pushed back into the room he just was in because of other agents rushing the hallway. If the memory wasn't cleared, the agent now wouldn't leave the room a second time which of course would be a real bad behaviour.

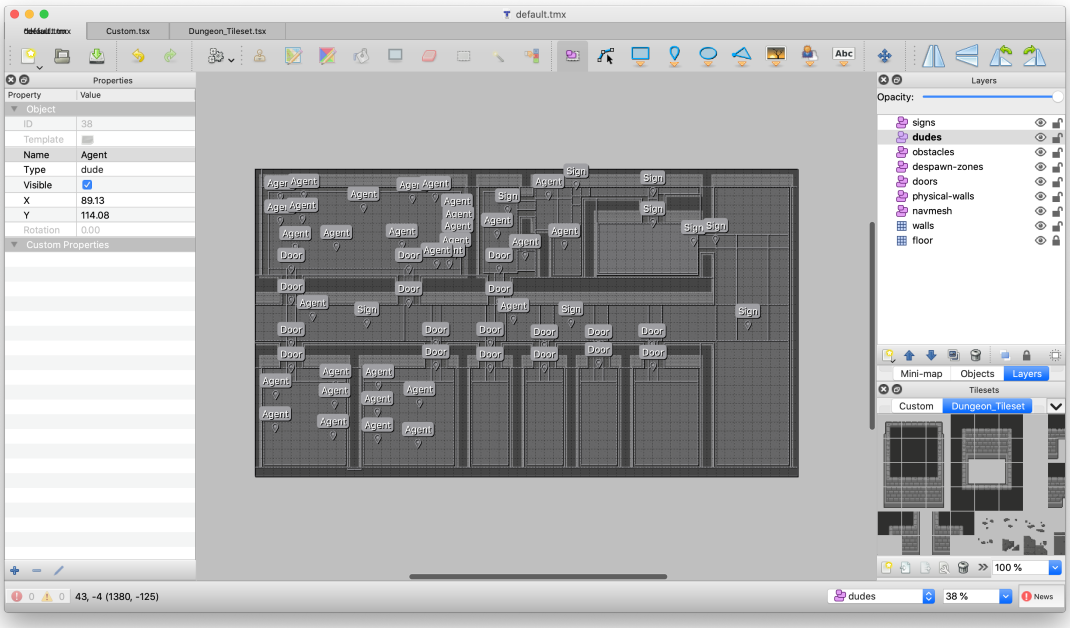
4 Implementation

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 meaning the whole simulation can easily be rendered on a webpage and doesn't require any setup.

4.1 Environment construction

As people are at least to some extent the product of their environment, it is very important that the simulations does not have a fixed environment in which all simulations take place. In fact it is one of the most important parameters when looking at the results, i.e. how many people survived a fire emergency. People in a totally enclosed room have no chance of escaping whereas agents sitting next to an escape zone will have no issues. To make the change of this parameter as accessible as

possible, the implementation uses an open source format called `Tiled's map format` (TMX).



The editing screen of Tiled

Using this editor anyone with basic computer skills is able to adjust the given examples or of course define their own settings enabling people using this model as a basic evaluation method of the safety in a case of fire.

4.2 Agent behaviour

As phaser includes a physics engine we can make use of this by simulating the behaviour of the agents based on forces acting on them where the final movement is the vector addition of all these forces. Phaser has an `update` function that is called every time a new frame is calculated. which is perfectly suited for such a task. The different behavioural rules are implemented as follows

- Wall repulsion

  -

- Agent repulsion

Agent repulsion can be implemented rather easily: for every pair of agents a force

$$f_r = c_1 \cdot e^{-\frac{d}{c_2}}$$

where c_1 and c_2 are adjustable parameters but constant for the two agents and d is the distance between the two agents. As the force is proportional to e^{-d}, the force is exponentially decreasing when linearly increasing the distance, i.e. exponentially increasing when linearly decreasing it. The initial values for c_1 and c_2 were empirically determined and are dependant on phasers units and on the agent attraction force.

- Agent attraction

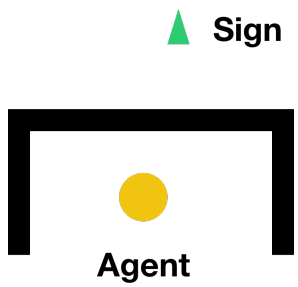
  The agent attraction force is, in contrast to the agent repulsion force, linearly dependant on the distance d

$$f_a = c_3/d$$

  but of course directed in the opposite direction of f_r. As f_r is exponentially dependant on d and f_a just linearly there is a distance d_0 for which the two forces are in balance and the two agents will stay in place if no other force would act on them.

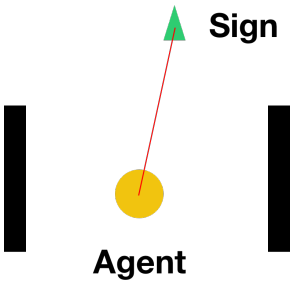
- Target attraction

  For checking what signs are visible, we iterate over all of them and use a technique called raytracing. This means we draw a line from the agent's position to the sign's position and check whether this line intersects any rectangle defined in the environments `physical-walls` layer.



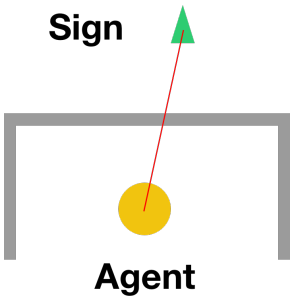
An agent's vision blocked by a wall

In the figure above the agent is surrounded by walls meaning a line between the agent and the sign would intersect the front wall and thus in this case the agent wouldn't move towards the sign.



An agent's sight of a sign visualized by a red "ray"

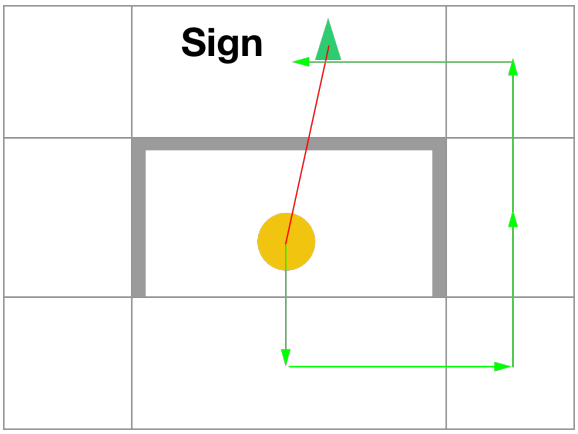
If that's not the case, the sign is indeed visible and we can apply a force into the direction of the sign. This works fine until you introduce obstacles drawn in gray instead of black which stands for a wall. Obstacles collide with the agent as de walls, meaning the agent won't "climb" over it, the difference is that the agent can see "through" or over an obstacle. A good example for obstacles are tables.



An agent stuck within obstacles as the direct force
in the direction of the sign doensn't help

Now the issue with directly applying a force to the agent into the direction of the ray becomes obvious, the agent will get stuck as it cannot move through the wall. To circumvent this issue, an so called pathfinding algorithm is used, once the agent sees a sign. Now the agent doesn't just move into the targets direction but rather finds a way to the target and this way is able to walk around obstacles. The way this pathfinding algorithm works is by using the `navmesh` layer in the environment definition that is made up of rectangles which the agent can use as a path. It then creates an ordered list of points that the agent

must follow in order to get to the target.



An agent boxed in obstacles while seeing a sign
who's using a pathfinding algorithm to find a way

5 Simulation Results and Discussion

Some Ideas for experiments: One door for example 100 dudes. Vary the desired velocity and observe how long it takes until all have passed the door. With obstactle in front of the door versus without, observe how long it takes.

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. This is actually what we would expect because of the mutual repulsion of the agents.

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.

5.3 Adapting the parameters for better results

5.4 Finding the model's limits

5.4.1 Evaluation of the building model

5.4.2 Evaluation of the agent model

5.5 Evaluation of the simulation

6 Summary and Outlook

7 Bibliography

[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.