Agent-based Evacuation Simulation

Luka Marković-Đurin
University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
lm52473@fer.hr

***Abstract* – This paper presents an implementation for an agent-based evacuation simulation. The implementation is used to analyze different problematics related to autonomous agents such as obstacle avoidance and path following.**

***Keywords* – agents, simulation, evacuation, emergency, boids**

I. INTRODUCTION

In the face of increasing urbanization and the ever-present threat of natural or human-made disasters, the optimization of evacuation processes has become a critical aspect of emergency management. Consequently, it is vital to consider the evacuation configuration of the building to best organize the evacuation upstream.

Agent-based models are computational models used for simulating actions of autonomous agents and their interactions with the environment. They provide a natural description for modeling emergent phenomena such as traffic jams and flocking.

An autonomous agent is an entity that is capable of acting independently, without direct human control, to achieve specific goals or objectives.

This paper presents an agent-based model for simulating an event of evacuation from a building. The implementation of the model is then analyzed in different scenarios to highlight the advantages and flaws of the model's approach.

II. RELATED WORK

In the paper "Algorithm and Examples of an Agent-Based Evacuation Model" [1] authors have described an autonomous agent model based on real-world parameters such as field of view, reaction time and response time.

The paper "Social force model for pedestrian dynamics" [2] explores a computational model that simulates the movements of pedestrians by considering the interaction of social forces, providing insights into collective behavior in crowded environments.

III. IMPLEMENTATION

A. Implementation Framework

The simulation was implemented using the JavaScript programming language and the p5.js library [3].

B. Environment Representation

Simulation environment is comprised of three types of entities: wall, exit and obstacle.

A wall is described with two points, signifying its starting and end points. Walls are used to construct the general layout of the building.

Exit is defined by its center position and its length. Exits are defined on walls and are typically used as objective targets by autonomous agents.

Obstacle is defined by its center position and a radius. Agents are required to steer away from all obstacles in their path.

C. Agent Model

An agent is defined by its radius, position and velocity. In each simulation frame an agent evaluates its environment and calculates forces which are acting on it. The forces are then used to calculate the agent's acceleration. Acceleration is used to update the agent's velocity which is in turn used to update its position. The described algorithm is represented by equations (1) – (4).

$$\vec{F}_{total} = calculateForces \tag{1}$$

$$\vec{a} = \frac{\vec{F}_{total}}{m} \tag{2}$$

$$\vec{v} = \vec{v}_{prev} + \vec{a} \tag{3}$$

$$\vec{p} = \vec{p}_{prev} + \vec{v} \tag{4}$$

D. Modeling Behaviour

The underlying method for calculating forces acting on an agent is based on Reynolds's boid algorithm [4]. Rather than calculating forces that push the agent, Reynolds defines a steering force which directs the agent's movement towards its target. Reynolds's steering formula is stated in equation (5).

$$\vec{F}_{steer} = \vec{v}_{desired} - \vec{v}_{current} \tag{5}$$

As the agent already keeps track of its current velocity, it is only necessary to define the desired velocity.

E. Seeking a Target

$$\vec{n} = \vec{p}_{target} - \vec{p}_{agent} \tag{6}$$

Equation (6) describes a vector ***n*** which points from the agent towards a given target.

$$\vec{v}_{desired} = m \cdot \frac{\vec{n}}{|\vec{n}|} \tag{7}$$

Equation (7) describes a formula for calculating the desired velocity where *m* represents the maximum speed of the agent. The equation describes the desired velocity as moving towards the target at maximum speed.

F. Agents Separation

To avoid collisions with other agents, each agent should keep a minimum distance from its neighbors.

$$\vec{F}_{sep} = \sum_{i=1}^{N} \frac{\vec{p}-\vec{p}_i}{|\vec{p}-\vec{p}_i|^2} \quad (8)$$

Equation (8) describes a formula for a separation force between an agent and its neighbors. The formula is only applied for neighbors that are closer than the desired separation.

G. Obstacle Avoidance

Each frame of the simulation, an agent checks for obstacles and calculates a necessary steer angle to avoid them. An obstacle must meet two conditions to be deemed significant to an agent. First it must be in range of the agent, and second the agent must be moving towards it.

$$\vec{u} = \vec{p}_{obstacle} - \vec{p}_{agent} \quad (9)$$

$$\theta_1 = \theta_{tangent} = \sin^{-1}\left(\frac{r_{obstacle}+r_{agent}}{|\vec{u}|}\right) \quad (10)$$

$$\theta_2 = \theta_{direction} = \cos^{-1}\left(\frac{\vec{u}\cdot\vec{v}}{|\vec{u}|\cdot|\vec{v}|}\right) \quad (11)$$

$$\theta_{steer} = \theta_{tangent} - \theta_{direction} \quad (12)$$

Equation (9) describes a vector $\boldsymbol{u}$ pointing from the agent to a given obstacle. To check if an agent is moving towards the obstacle, the dot product of the agent's velocity and the vector $\boldsymbol{u}$ is calculated. If the dot product is positive the agent is moving towards the obstacle and if it is negative the agent is moving away from the obstacle. If the agent is moving towards the obstacle, it is necessary to check if the agent is on the collision course with the obstacle. Equation (10) describes the formula for an angle that the tangent from agent makes with the line joining the center of the obstacle. Equation (11) describes the formula for the angle between the direction of the agent and the line joining the center of the obstacle. If θ_2 is smaller than θ_1 the agent is on a collision course with the obstacle. Equation (12) describes the formula for the steer angle. Finally, the agent must decide which way to circumvent the obstacle. If the angle between the vector $\boldsymbol{n}$ and the vector $\boldsymbol{u}$ is positive the steer angle is negated. The calculated steer angle is used to rotate the velocity vector of the agent.

Obstacle avoidance does not include avoidance from the walls. However, collision detection algorithm has been implemented to ensure that the agents cannot move through walls.

H. Path Following

A path is a series of points defining line segments. Each path has a radius defining the width of the path. An agent can be assigned a path which it will then follow. To follow the path an agent will try to predict its future position by adding a scaled version of the velocity vector to its position. The agent then calculates the normal to the path segment it is currently on. The magnitude of the normal equals the distance between the predicted position and the path segment. If that distance is greater than the path radius the agent will look to steer towards the path. To do this, the agent calculates a point on the path that is some distance away from the normal and then uses it as a seek target.

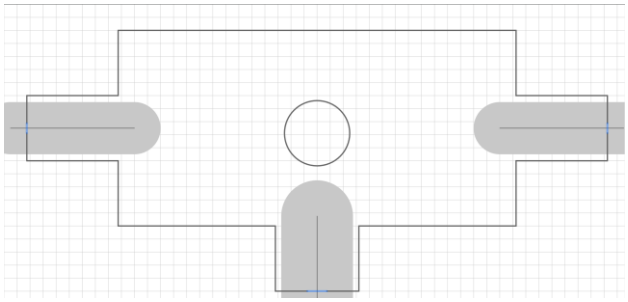

Figure 1. Scenario 1 layout

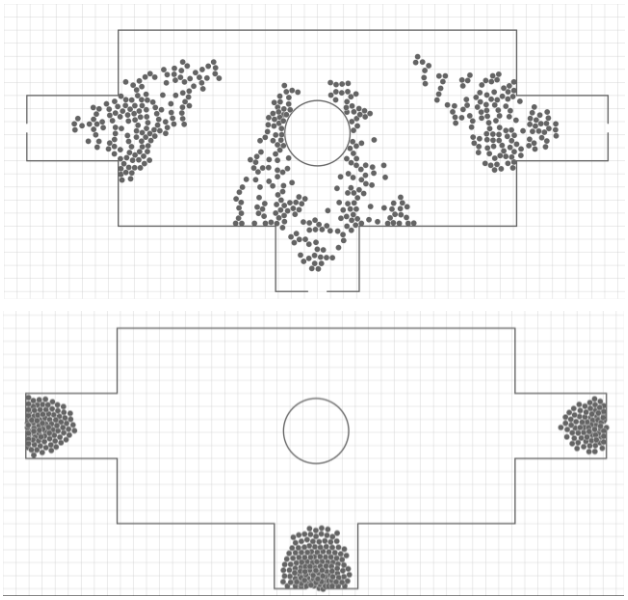

Figure 2. Screenshots at different timestemps of the simulation

I. Imporving Efficiency

To avoid the N^2 complexity of interaction checks in agent separation, a uniform grid was implemented. Each simulation frame, agents are stored in appropriate cells of the grid. The algorithm for agent separation performs checks only on the agent's cell and neighboring cells.

IV. Results and Analysis

To display some of the applications of the implemented algorithms, two scenarios were constructed.

A. Scenario 1

Scenario 1 layout can be seen in Figure 1. It contains 3 exits located in the left, right and bottom corridors. The layout also contains an obstacle in the middle, represented by a circle. To ensure better agent pathing each exit has been assigned a path, represented by a gray surface and a corresponding line. At the start of the simulation each agent was assigned the closest exit as a seek target and the corresponding path to follow. Figure 2 shows the simulation at different time steps.

Lack of paths would result in agents near the walls adjacent to the corridors experiencing a tendency to become trapped when attempting to seek the exit. Applying a path-following algorithm enabled agents to maneuver around the corners of the walls in a manner more reminiscent of human-like navigation. Similar results could potentially be achieved by defining a series of seek targets: around the corners of the corridors and in the middle of the corridors and applying the seek behavior on those points before seeking the exit.

B. Scenario 2

The layout of scenario 2 can be seen in Figure 3. It contains a single exit located in the center area and a series of corridors and paths. The purpose of this scenario was to test the application of paths for directing agent pathing through a complex building layout. When spawned the agents would determine the closest path based on their distance to the path's starting point. After an agent reaches the end of the path it would select the next path to follow in the predefined order. Figure 4 shows a screenshot of the simulation.

Rather than defining one path that spans throughout the layout, multiple paths were used. When a single path was defined, agents would focus too much on following the middle of the path which resulted in less human-like navigation. This was especially evident in the bottom right corner of the layout. Breaking the path into multiple smaller paths allowed agents more freedom while navigating the layout. However, this approach required more involvement when defining the layout and would be more challenging to incorporate with path finding algorithms.

V. Potential Improvements

The main outlier of the presented model is the lack of intelligent navigation. The alternative presented by applying a path following algorithm for navigating complex layouts could be upgraded in several ways. For example, the paths could be described as a curve instead of a collection of line segments, to better achieve navigation around corners. Instead of manually defining the path, path-finding algorithms could be implemented. The path properties such as its points and radius could be dynamically adjusted to accommodate a changing environment.

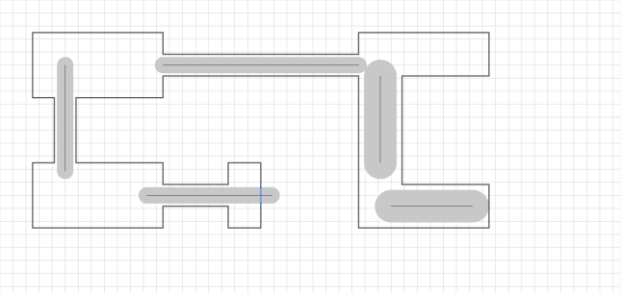

Figure 3. Scenario 2 layout

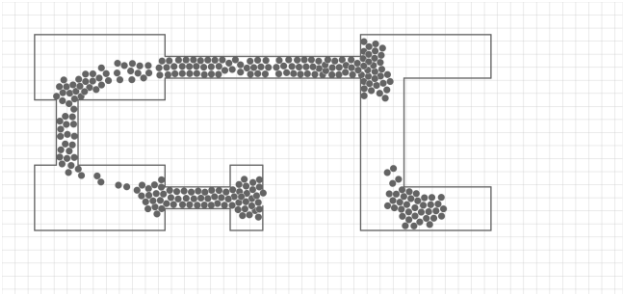

Figure 4. Screenshot of the simulation

Additionally, although the achieved results exhibit human like navigation during an emergency, the model lacks foundation in reality. A more realistic simulation could be achieved by implementing findings of other papers such as those listed in the Related Works section.

VI. Conclusion

This paper presented some of the methods used when modeling an agent-based simulation. Reynolds's boid behavior modeling was applied to different scenarios pertaining to the evacuation context. While the achieved results resemble human-like behavior during an emergency evacuation, the autonomous agents lack autonomy and individuality. More work is needed to achieve realistic modeling of human behavior during an emergency.

References

[1] X. Cui, J. Ji, X. Bai, "Algorithm and Examples of an Agent-Based Evacuation Model", 2023

[2] D. Helbing, P. Molnar, "Social force model for pedestrian dynamics", 1998

[3] https://p5js.org/

[4] C. W. Reynolds, "Steering Behaviors for Autonomous Characters", 1980