Evac Sim: Fall 2020 CSS600

Justin Downes and Chris Smith

December 2020

Abstract

Simulating large scale evacuation is cost-prohibitive in terms of realism and required man-hours. This model, creatively called "Evac Sim", provides an Agent-based simulation that supports choosing a floor plan, modeling agent behaviors, and capturing some flavor of how a real evacuation might play out. This research uses NetLogo and its BehaviorSpace facility to create our evacuation environment and execute the experiments. Our simulation environment should provide a foundation for further exploration of pedestrian movement mechanics.

1 Introduction

In order to understand pedestrian evacuation dynamics, one must have the flexibility to not only run multiple experiments in a variety of physical layouts but to control those pedestrian's behaviors in that environment. These requirements make running such experiments in the real world prohibitive. With the emergence of agent based simulation frameworks and the parallel increase of computation power, not only are digital simulations a viable alternative, they are becoming the main mechanism to explore these problems.

Our work utilizes the NetLogo framework to create an environment where we can simulate pedestrian evacuation. NetLogo itself is meant for non technical practitioners or educational cases and we have developed a number of key features, from loading various maps to developing an experimentation framework, that we hope are transferable to other NetLogo simulation environments. Due to this simplified nature of NetLogo we had to make certain trade-offs with our implementation which this paper goes over in depth. The

constraint of our modeling environment drove the experiments that we chose in that the simulations are meant to explore discreet components or behaviors encountered with pedestrian evacuations instead of compound ones such as dynamic pedestrian decision making.

1.1 Previous Work

The literature abounds with previous efforts in this area. This project incorporated ideas from a variety of sources in the literature, from general crowd modeling reminders, to requirements, and fully implemented models. A good overview is [1], which makes the point that there are three motivations for simulating: to generate observable phenomena, to test theories, and to test design strategies. They note that people tend to follow the path of least resistance while moving. The stress of emergencies leads to herding and flocking behavior, to the point of missing optimal escape routes, or engaging in inefficient "arching" to get through an escape, which is one of our avenues of inquiry. Layouts matter too, as brought out by [5], which points out that real-time generation of floor plans that are realistic is vital for systems such as games.

Moving on to fire-specific models, additional fire-specific requirements were discussed by [2], which outlines a requirement for a comprehensive fire evacuation model design capturing human behavior. The effort here was to capture the various theories and "behavioral facts" suitable for embedding in modeling tools, and then argued for including behavioral models in evacuation models. Also, [7] provides an overview of major evacuation factors and is helpful for developing Agent-based simulations such as this one. This includes a Fire Dynamics Simulator (FDS) employing computational fluid dynamics and Geographic Information System (GIS) for modeling human responses. This is far more realistic than NetLogo can support. Beyond the physical aspects, [3] emphasizes that behavioral aspects of crowds are also important, including behavior, locomotion, and navigation. These had been modeled previously. This research relied upon graph-based methods to plot the exit course, whereas this model uses values attached to patches to drive the movement decisions for the Agents. In both this and our model, the presence of other agents affects movement calculations.

Two final models of note were one that was also implemented in NetLogo, [8] PrioritEvac. This is an Agent-based model that explores social science effects in evacuation response. Agents maintain distinct priorities supporting

granular investigation of reactions. This is also a NetLogo model, and was validated against the Station fire, where pyrotechnics ended the career of the rock band Great White.

Finally, one pertaining to public space, and not fires, was [9]. This study begins with thirty-six hours of video from a public square in Ningbo, China, which were used to develop a Large-Scale Public Place (LPS) evacuation model, building upon the Social Force Model (SFM). This was expressed in five strategies. Attention was paid to walking speed and diameter, and a the model was used to study the efficiency of evacuating the area. The idea of starting with the human movement and not a map of an enclosed space.

1.2 Approaches

All models are abstractions of the real world and human behavior is an extremely complex parameter to account for. An annual fire drill in an office setting is nearly a pleasant affair in good weather. Drills are announced beforehand, alarms sound, people cease affecting work, an orderly progression to assigned locations occurs, rolls are taken, the fire department arrives, equipment is checked and reset, the drill concludes, and people resume affecting work.

The smooth orderly progression of events as planned rarely survives the real world. [1] refers to the "least effort principle" that succumbs to the entropy of the moment when humans encounter pressure. "As nervousness increases there is less concern about comfort zone and finding the most convenient and shortest way" [1]. Without good knowledge of the floor plan around them, people lose the ability to orient themselves. Panic sets in, visibility conditions worsen, breathing becomes an issue, and heat effects decision making all leading to a degraded ability to behave rationally during an emergency.

Under pressure, people seek leaders, and "... tend to follow others in the assumption they could get them out of the dangerous area," which leads to "flocking" behaviors, until the arrival at an exit, where arching and clogging are common. Attempts to capture these emergent behaviors come in three flavors: flow-based, cellular automata, and Agent based. In a stock-and-flow model, individuals are implicit. People are homogeneous and uniformly distributed, so that they move like current in an electrical system. This is appropriate for, say, an evacuation model, where a large area is under consideration, and individual foibles would bog the analysis down.

In cellular automata, we have a more detailed map of a smaller area, but the people are really little more than ants, lacking the kind of emergent behavior that is of real-world interest. Agent-based models, then, go beyond cellular automata and keep some state for each agent, along with modeling behavior based upon that individual state. Our model was not as expressive as the one in [1], focusing mainly on agent mobility. Even then, we seemed to encounter scalability issues with NetLogo.

We captured the basic components of the building geometry and path selection, but did not model as many granular aspects as could be sought. While we capture speed and position, we do not model visibility, reaction time, collaboration, insistence, or knowledge. Also not captured from the reference model are intrinsic attributes such as gender, age, experience, nervousness, or any organizational role.

2 Methodology

Our main goal was to hone in on the narrow situations in which we can detect patterns given our constrained simulation environment. Due to the constraints of the NetLogo environment, described further below, we were forced to simplify various aspects of our model. These simplifications though, provide us the opportunity to explore where emergent properties may arise in our agent's behavior3.5 by reducing complex problems to their foundational components.

2.1 Environment

Our simulations utilized the NetLogo modeling environment [1]. NetLogo was developed to provide an easy environment to simulate multi-agent models for educators and researchers with non programming backgrounds [4]. The environment is made up of patches and agents. The patches can be thought of as the properties of the world which are accessed per cell in a grid layout. The agents are objects that can interact with the world as well as each other. While both agents and patches can maintain their own states, only agents can move around through the world and so can change their relationship to patches.

[1] https://ccl.northwestern.edu/netlogo/

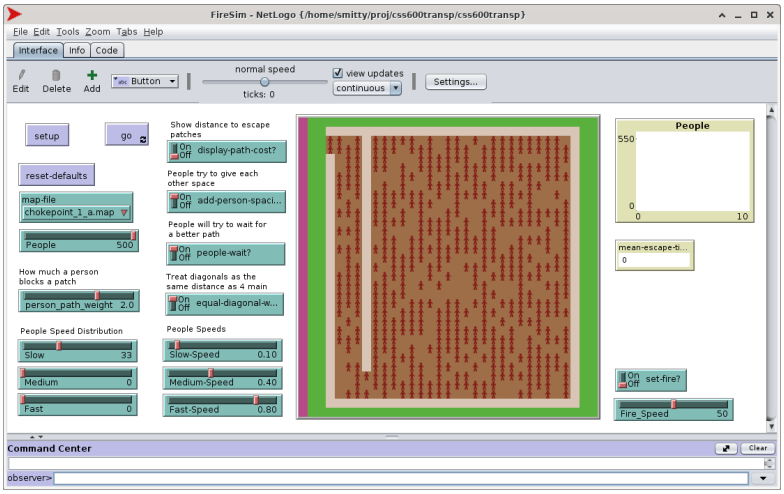


Figure 1: Evac Sim UI implemented in NetLogo

One of the foundations of our simulation effort was the ability to run experiments on a variety of maps, which mostly incrementally changed a key feature. In order to accomplish this we implemented a custom map loader in NetLogo so that we could quickly develop custom layouts externally and procedurally load them at experiment time 3.1. The method of loading maps involves reading a file in line by line where each line was a textual representation of a NetLogo list. This list was then used to sequentially set the patch to the corresponding state, be it safety patch, grass, floor, or wall. The code for loading a map is provided below.

Users may create map files in a separate text editor. Each cell reflects the desired patch state in the world, where 3 = safety, 0 = grass, 1 = floor, and 2 = wall. An important note is the beginning and ending brackets '[ ]'. These are necessary for NetLogo to correctly load each line as a list structure from the line's string representation. Below is an example portion of a map file.

```
to load-map
  file-open map-file
  ; xdim and ydim are half the dimensions
  ; in width and height
  let row  ydim
  while [not file-at-end?]
  [
    ; read each line in
    let linestr file-read-line
    ; load line into list variable
    let line read-from-string linestr
    let col  0 - xdim
    ; for each element in list set the patch color
    ; to a value from a table of color mappings
    foreach line [
      [x] ->
      ask patch col row [
            set pcolor table:get map_table x
      ]
      set col (col + 1)
    ]
  set row (row - 1)
  ]
  file-close
end
```

Listing 1: Map loading procedure in NetLogo

```
[3 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0]
[3 0 0 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0]
[3 0 0 2 1 1 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 0 0]
```

Listing 2: Example map file, 3 rows

Since our map creation needs were constrained to the few experiments that we planned to execute, we did not implement a procedural way to generate maps. This functionality, while one of our stretch goals, had to be dropped in the interests of deadlines. If one were to expand upon our work then there are number of mechanisms to generate layouts [5] such that further experiments could be devised without the human intensive effort of generating maps by hand. Such possibilities include evaluations of optimal floor

plans through a guided search or ablative studies on the subtle modifications of ideal floor plans.

Our simulation grid world is governed by the patch type. The safety patch (pink) is used as the foundation for our pathing algorithm described here 2.2. The grass and floor patches (green and brown) are areas in which agents can move freely. The two different types of patches are not just for aesthetics but are a residual from original plans to implement a fire aspect which would have burnt differently on different patches. The final patch type is the wall (light brown) which blocks agents from moving through it.

The agent's themselves are also defined fairly simply. They are principally defined by the speed at which they move, a value between 0 and 1. They also have an internal counter for how long they have been on the map which is used to calculate the mean escape time. This value increments on each global tick unless the agent has reached a safety patch. In order to allow for ease of experimentation there are 3 groups of agents; labeled slow, medium, and fast. Despite the label semantics, each group can be assigned its own speed depending on the wishes of the experimenter. During each movement step the agent moves the speed of the group that they are assigned to. The experimenter can also set the number of agents in the environment from 0 to 500 before each simulation as expressed by the parameter P below. A helper function that we provide is the ability to maintain the ratio of numbers of slow, medium, and fast agents no matter the total number of agents. This ratio is calculated in the following algorithm where S, M, and F are the values specified by the experimenter in sliding values between 0 and 100.

$$\text{count of slow people} = \frac{S}{S + M + F} * P \tag{1}$$

While they can each be assigned as if they are a percentage totaling 100, in actuality the percentage is the ratio of a single group's value to the sum of group values, multiplied by the total desired agents. This is a common algorithm to determine load balancing settings for computers.

2.2 Movement Mechanisms

In this section we will go over the mechanisms which dictate how our agents move within the simulation environment. Traditionally, in an environment where an agent is moving towards a goal, each agent's path to that goal will

be calculated individually through the environment[2]. In our situation we implemented a simplified pathing algorithm which instead calculates global weights for each environmental patch in which all agents flow to the lowest weight. These weights grow in relation to their distance to the safety patches and as agents move over them by imparting their own weight to the patch they are on. This simplified pathing allows for much quicker computation of routes and therefor quicker executions of simulations for our experiments.

The following will provide an overview of how the weighting mechanisms work and the choices agents make on whether to move. The foundation of movement is the weight of a given patch, or the cost as described below. The cost metric is defined as the distance from the patch to be considered P and the nearest safety patch P_s. While there may be many safety patches, for the sake of brevity, we assume that P_s is resolved *a priori* as the closest. We can confidently declare this since the weighting implementation iteratively grows from each safety patch such that when it reaches a candidate patch it has been reached by the closest one. This distance cost is also modified by the presence of an agent, $Agent(P)$, on that patch. This agent's weight, A_w, is configurable by the user. Of important note is that the weight is calculated independently of agents that may reside on patches between a patch and the safety patch. So the presence of additional agents on patches between the current patch and the safety patch has no impact on the cost. This is the simplified pathing mechanism at work in that only local environment calculations are conducted.

$$cost(P) = distance(P_s, P) + Agent(P) * A_w$$
$$Agent(P) = \begin{cases} 1, & \text{Agent Present} \\ 0, & \text{Agent Not Present} \end{cases} \tag{2}$$

The distance measurement is not as simple as calculating the Euclidean distance. Since our environment is a grid world that is inhabited by blocking obstacles, we can only consider movement in either the 4 cardinal directions or the 8 neighboring directions. These two options, which are additionally configurable by the user, determine how the distance between patches are measured. The distance between 2 points P_1 & P_2 is optionally defined by the flag $equal - diagonal - weight?$. If false the distance is the Manhattan

[2]http://www.cs.us.es/ fsancho/?e=131

distance [3]. If true the distance is the Chebyshev distance [4]. While this algorithm for distance works in an open world it doesn't account for the blocking obstacles. This is another side effect of how the actual implementation works. By growing outward from the safety patches and around obstacles the actual distance calculation is only ever considered for patches that are next to each other. This makes our implementation a narrow case of the algorithm below in that all distances we calculate equal 1 and the real choice is how we choose the next patches as either the Manhattan or the Chebyshev neighborhood.

$$distance(P_1, P_2) =$$
$$equal - diagonal - weight? = \begin{cases} max(|x_1 - x_2|, |y_1 - y_2|), & True \\ |x_1 - x_2| + |y_1 - y_2|, & False \end{cases} \quad (3)$$

Now that we have our patch weights to inform our movement decisions, it is time to actually make the decision of where to move. For this choice we simply choose the neighbor patch that has the lowest cost, where the neighbors are defined as the 4 patches above, below, right, and left of our current patch. Once we know which patch is lowest we take the vector to that patch and multiply it by our agent's speed. So, for a given agent A the vector to move is given by the vector of the lowest cost neighbor patch times our agent's speed. We use the 4 patch neighborhood for movements due to some limitations with the grid world and the 8 patch neighborhood causing agents to get caught in corners of walls. This was caused by an agent attempting to move in a diagonal while next to a wall but not having enough speed to actually clear the wall and move onto the diagonal floor patch.

$$move(A) = V(min(cost(neighbors_4(A_p)))) * S_a \quad (4)$$

$$neighbors_4(P) = \{patch(P_x - 1, P_y - 1), patch(P_x + 1, P_y - 1), patch(P_x - 1, P_y + 1), patch(P_x + 1, P_y + 1)\} \quad (5)$$

We have an additional parameter to account for situations where the cost of all neighboring patches is greater than the current patch. The flag $people - wait?$ allows simulators to decide whether or not agents stay put

[3] https://www.sciencedirect.com/topics/mathematics/manhattan-distance
[4] https://en.wikipedia.org/wiki/Chebyshev_distance

and wait for a better patch or to always move even if the new patch has a higher cost. This flag redefines the $move(A)$ function as $move(A)$ if the flag is false and if the flag is true then the agent only moves if the cost of the new patch is less than their current patch.

$$people - wait? = \begin{cases} min(cost(A_p), move(A)), & True \\ move(A), & False \end{cases} \tag{6}$$

Another phenomena we wished to capture is the ability for agent's to give each other space. This is configurable through the $add - person - spacing?$ flag. When the flag is true the cost of a patch is redefined to be the normal cost plus the sum of all the neighbors with agents, multiplied by the agent's weight divided by 10. This scaling factor of 1/10 has been determined through trial and error and is something that could be exposed to user control in the future. For ease of implementation and increased computation time the actual calculation is done from the perspective of the agent in that each agent has their scaled weight applied to its patch neighbors.

$$add - person - spacing? = \\ \begin{cases} cost(P) + \sum Agent(neighbors_4(P)) * A_w/10, & True \\ cost(P), & False \end{cases} \tag{7}$$

The grid world, while maybe too much of a simplification of real world dynamics, enables us to implement complex behaviors through simple straightforward algorithms. These complex behaviors have enabled us to conduct a few interesting experiments described in the following sections.

3 Experiments

The range of experiments we have decided to conduct seeks to explore specific phenomena based on layouts 3.3, agent settings 3.4, and replicating human behavior 3.5. In order to hone in on the most important features we have attempted to reduce each problem to its foundations through meticulous experiment design. To facilitate these experiments we developed a robust experimentation harness 3.1 to rapidly execute large and varied sets of experiments in NetLogo. It is our goal to generate reference examples of easily

understandable experiments that can help in exploring agent-based simulations in NetLogo. These experiments are designed to be the starting point for future and more complex evaluations of agent based evacuation models.

3.1 Experimental Environment

This section describes our experiment harness that we built using NetLogo's Behavior Space functionality. Behavior Space supports supplying simulation arguments via an XML document, invoking the NetLogo engine via a script pointing to the XML, and then capturing the results via the standard output.

This lends itself to scripting via Python [5]. The goal had been to extract the initial Behavior Space XML content directly from the .nlogo file, and then craft a SQLAlchemy [6] model on the fly that would support storing results in an RDBMS, e.g. SQLite [7]. That proved out of reach due to the advanced nature of SQLAlchemy, so a hand-crafted model was used, which makes alterations to the underlying model more maintenance intensive. While we stored the results in a local SQLAlchemy file, a mere update to the connection string would allow storing results in an enterprise RDBMS to good effect.

Another Open Source tool that was used extensively was PyTest [8]. Billed as a unit testing framework, PyTest supports breaking the problem down into granular fixtures and then combining them in a Lego-like fashion that lends itself to the problem space. For example, while Behavior Space allows stepped alteration of numerical parameters in a model, swapping out map file names is not directly supported. Implementing a Python function to generate the map file names and then re-writing the XML was far more convenient than having distinct experiments for each map.

Generating parameter inputs via functions also supports enforcing constants across multiple ones. For example, the number of slow/medium/fast people, these were three discrete integers, but were really percentage chunks of the people variable.

SQLite proved really helpful when we discovered some scalability issues with NetLogo itself. While there was no time to light off a Java debugger and delve into the root cause, we saw decreasing ability of the system to complete the ordered number of runs as the number of Agents increased and

[5] https://www.python.org/
[6] https://www.sqlalchemy.org/
[7] https://sqlite.org/index.html
[8] https://docs.pytest.org/en/stable/

the distribution shifted from slow to medium. It became useful to make a single SQLite file for each increment of the people variable and then merge the results in a script after the fact.

Finally, Python's data science facilities are well-known [9], supporting arbitrary visualization pipelines. Where visualizations were required to show results we leveraged the Matplotlib library to generate those graphs [10].

3.2 Experimental Parameters

To facilitate the various experiments that we wished to conduct we exposed a number of the variables that drive the models behavior to the user interface as well as the Behavior Space environment. Since most of our simulations were exploring a single feature we would only tweak a single parameter such as the map file or the Slow and Medium agent counts.

Variable	Description	Default Value
map-file	Name of map file to set up	N/A
People	Number of Agents	500
person_path_weight	Agent blockage	2.0
Slow	Agents at Slow-Speed	100%
Medium	Agents at Medium-Speed	0%
Fast	Agents at Slow-Speed	0%
Slow-Speed	Agent movement rate	0.3 patches
Medium-Speed	Agent movement rate	0.75 patches
Fast-Speed	Agent movement rate	0.99 patches
add-person-spacing?	Add person path wt buffer	True
equal-diagonal-weight?	Calc diag move cost	True
people-wait?	If people tarried	True
mean-escape-time	Average agent escape time	Output

Table 1: Model parameters exposed to experimenters

To account for the fact that agents are randomly positioned at the beginning of the simulation we ran multiple iterations of each experiment for a given set of parameters. Generally this count was 10 which enabled is to

[9] https://www.scipy.org/index.html
[10] https://matplotlib.org/

be confident in our output as well as to run the simulations in a reasonable amount of time. Due to the extensive experimentation harness that we developed we had the ability to control the parameter input at a much higher level than NetLogo's built in Behavior Space. This is especially evident in situations where we wanted to modify pairs of parameters in conjunction with each other instead of all pairwise combinations.

3.3 Experiments Based on Layouts

The layout of an environment is probably the largest factor that determines evacuation behavior. These layouts often have compounding features that work together to impact an evacuation[7]. For our set of experiments based on layouts, we attempted to isolate those discreet features and exaggerate them such that the actual impact on evacuation can be seen. These experiments build off of a base map showcasing that feature, and then replicate it with variations plotting that impact on average evacuation time.

3.3.1 Choke Point Experiment

Our choke point experiments seek to understand how choke points impact evacuation. For our purposes a choke point is a narrowing of the map that people must travel through. This is a subset of the often used definition of a choke point as a critical point at which people must move through[6] regardless of its physical dimensions. We begin our map design with a map with one choke point close to the exit. We expand upon this by adding more choke points to that map as well as starting new families of maps that have their choke point start further away from the exit.

For each map we executed a number of runs and captured the mean escape time, with results stated below 6. By increasing the number of choke points we sought to understand the initial impact of a choke point as well as the impact of sequential additions, which we expected to have decreasing impact. The intuition behind this is that even with more choke points, agents will start to align to the most narrow width. We also increased the distance from the first choke point to the exit to evaluate if the ability of agents to spread back out can negate some of the negatives of the choke point.

3.3.2 Exit Dimensions

Our next layout based experiment evaluates how exit dimension size and placement impacts evacuation speeds. We varied the size of the exits as well as their placements; alternating between states of one exit on one side, exits evenly split to opposite sides, and exits evenly split into 1 width wide exits all on one side. As shown below, each column of maps has the same overall exit size while each row shows the various placement strategies.

Similar to the choke points experiment, we captured the mean escape time for a number of runs on each map, with the results provided below 7. In these experiments we hoped to gain some insight, when dimensions are held the same, of how exit placements can be beneficial to escaping pedestrians. This is why the map itself is an empty room, so that the only layout impact we have is the size and placement of the exits.

3.4 Experiments Based on Agent Features

Experiments based on agent features explore how the intrinsic properties of pedestrians affect escape times. These properties may be simple measures of speed of movement to complex behaviors such as knowledge of the layout. In general, for all of our experiments, all agents have knowledge of the complete layout as derived from our global pathing algorithm which impacts each agent the same. For our specific experiment we evaluated how speed distribution between agents impacts the escape times.

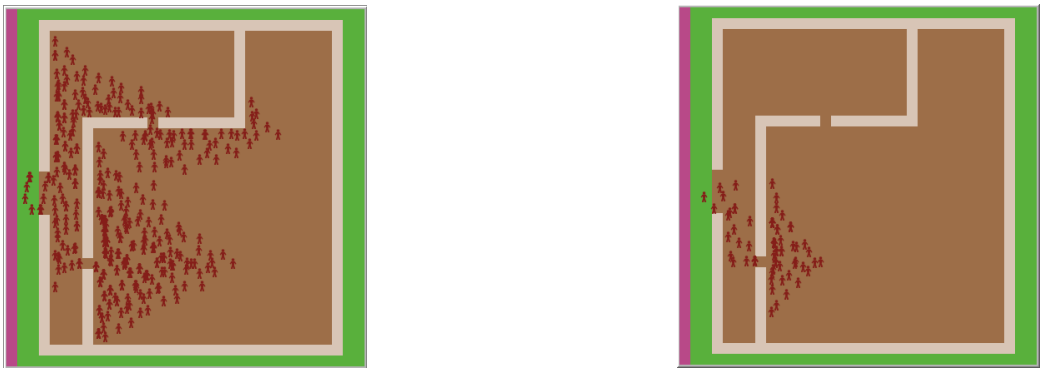
The goal of varying the distribution of speeds among the agents is to see if that people moving at different speeds introduces a negative factor, *a la* a turbulence similar to fluid dynamics. While we expect the average escape time to decrease while we move from more slow agents to faster agents, we are looking for a deviation from that expected decrease as shown below 8. If various movement speed distributions do introduce this turbulence into the system we can begin further discussions and experiments on how to mitigate these impacts through more efficient designs.

3.5 Experiments Replicating Emergent Crowd Behaviors

The final group of experiments that we evaluated were situations where we attempted to replicate real world human phenomena in our model. Since

our model uses a very simple environment and pathing algorithm it could be beneficial to recreate this behavior in our setup in that the underlying principals that the behavior derives from do not require complex models or computation. Our experiments worked backwards from attempting to recreate the behavior and then looking at the environment dynamics once that behavior is observed.

The first behavior we sought to mimic was the herding behavior. Herding is caused by irrational behavior in emergency situations that causes people to flock together [1]. For our situation, since we don't have the complex agent behavior to recreate this, we had to rely on map layouts that may evoke a similar pattern to herding.



(a) Evacuation with no herding

(b) Evacuation with herding (i.e. ignoring viable exit)

Figure 4: Herding behavior replicated through residual escape pathing from agents that never considered the alternative route.

The layout that we found to replicate herding behavior is one that has multiple exits with different distances to the safety patch. As a residual of the pathing algorithm, at a certain point every agent who who would flow out of one of those exits has left while the other one still has evacuating agents. While these agents could use the alternative exit, the flow of patch weights dictate that they continue on their current path. This commitment to a path, even if detrimental, is the same observed behavior in herding. But, since this only arises in specific map layouts in our simulation, this is not a viable area for further study with our models.

The next behavior we tried to replicate was that of clogging. This phe-

nomena arises when people block a passageway from each other while waiting for others to pass through [1]. We sought to replicate this behavior by adding parameters that makes agents increase distance between each other as described here 7 and here 3. The weight of a given agent is the cost that is directly added to a patches weight. This by itself causes a majority of avoidance behavior that we see. The person spacing parameter, when turned on, increases a patches weight if it is next to a patch with a neighbor. This makes patches next to people slightly more expensive to move to than other empty patches.

(a) No clogging due to people not blocking paths

(b) Clogging due to increased weight per person on a patch

Figure 5: Clogging behavior replicated through a weight attribute calculated for each patch an agent is on as well as a configurable additional weight for each agent's neighboring patch.

As seen in 5 when there is no person weight, agents can follow the same path and move on top of each other. As we turn the weight up we start to get clogging behavior. At a certain point agents too near each other on a patch increase the weight to a point where agents must seek other paths. This is analogous to the clogging behavior in that agents must weight for people to clear out when constrained from other alternative paths. Of interesting side note, usually with clogging comes arching behavior, or the formation of a semi circle around the exit as people evenly distribute. Our model shows the formation of a triangle which is most likely the side effect of the distance calculations in a grid world versus Euclidean distance in a point world.

By exploring where we can and cannot replicate these emergent behaviors

we can also explore the limitations of our modeling environment. While herding highlights a limitation of our model, clogging is replicated fairly well in our simulations. Other behaviors may necessitate the expansion of stateful information that agents maintain or the addition of more complex pathing solutions.

4 Results

Overall the results we observed were in line with our expectations. While our experiments had a random element due to the random initial placement of agents, we are confident that we performed enough tests to properly capture error ranges. These error bars are included with each result graph. The following detailed discussions will explore not only the results we observed but some of the caveats that our modeling environment causes.

4.1 Choke Point Results

As described in 3.3.1 this experiment sought to understand the impact that choke points have on evacuations. The maps for these experiments **??** are broken into three lines as plotted below. Line 1 shows the family of maps with the first choke point closest to the exit, with line 2 slightly further, and line 3 the furthest. The a through f designation describe maps with increasing numbers of choke points.

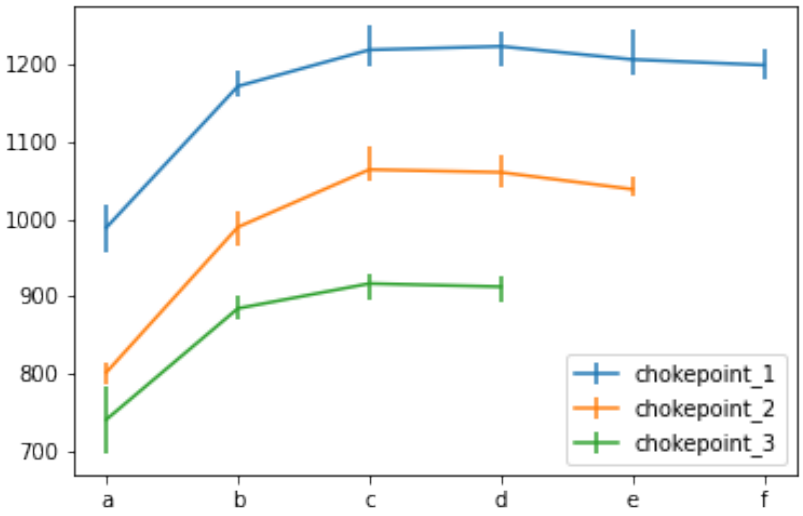


Figure 6: Number of choke points plotted against mean escape time. Choke point 1 has its first choke point closest to exit and choke point 3 the furthest. a through f denote increasing number of choke points with a the fewest and f the most.

There are few important observations from this results plot. The most obvious is that as we add choke points the escape time increases. This is expected behavior, but what may not be intuitive is that after a certain point adding more choke points no longer increases escape time. This is most likely due to people forming orderly queues in which an adverse narrowing has already been achieved by preceding choke points. Another point of interest is to be seen as the gaps between the lines. Remember that each line is a family of maps that has the first choke point further from the exit. We can see that having a choke point further from the exit increases mean escape time and that this increase does not scale linearly with the distance, at least initially.

4.2 Exit Dimensions Results

The exit dimensions experiments 3.3.2 sought to understand how placement of exits impacts the escape times. Each family of maps had the same dimen-

sions of exits (2, 4, 6, & 8) but placed them differently. The a group had one exit, the b group had the dimensions evenly split between 2 exits on opposite sides of the map, and the c group had the exits split into 1 unit exits evenly placed on 1 edge of the map **??**.

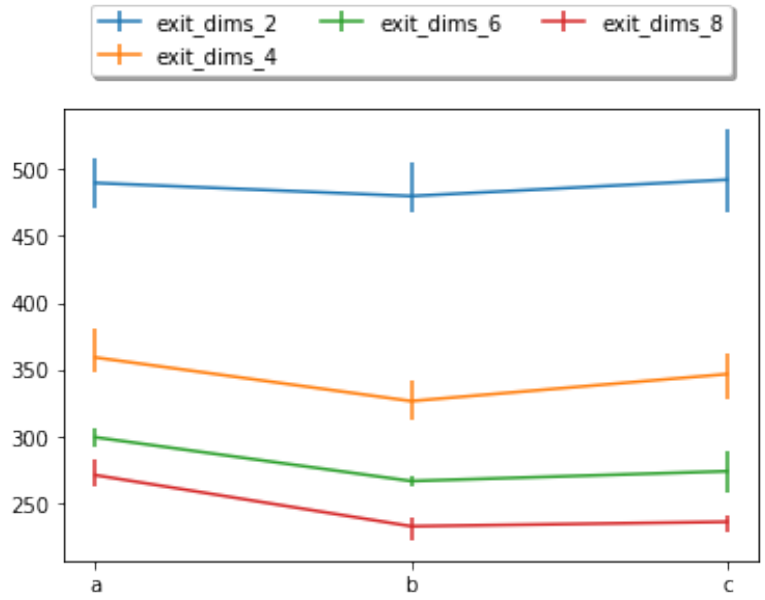


Figure 7: Exit placement strategies plotted against mean escape time. exit_dims_n are a family of maps with total width of exits equal to n. a group has 1 exit of width n, b has 2 exits on opposite sides with width $n/2$, and c has n exits all on one edge.

From this plot we can clearly see a trend where b and c exit placements work better. These are the placements that break up exits to either 2 on opposing sides or 1 width exits evenly spaced on one side. This trend also becomes more prevalent as we increase the overall exit dimensions. This may seem a little counter intuitive in that large exit dimensions should negate some of the clogging effects we see elsewhere. Further analysis would be needed to understand fully the principals driving this conclusion though. Similar to the choke point study, we also see diminishing returns from increasing the total exit dimension widths. This is denoted by the shrinking grap between each line going from top to bottom.

4.3 Agent Speed Distribution Results

As described in detail here 3.4 our agent speed experiment studied the impact of various agent speed distributions in the population. We steadily started with a population of all medium speed agents and incrementally moved that population to the slow category. We then ran each experiment multiple times plotted below as the measured escape line and its associated error bars. We compared this to what we expected to see, which would be a smooth transition in escape times from faster to slower, as noted in the predicted escape line.

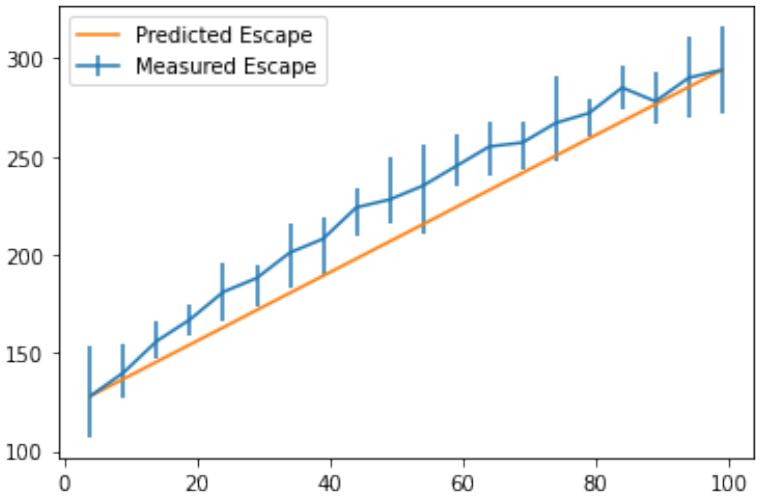


Figure 8: Agent speed distribution results

As can be seen in the graph there is a deviation from the predicted results that grows as the distribution of speeds becomes 50%. This implies that the maximum interference between 2 groups moving at different speeds comes when those groups are evenly divided. This follows intuition, but what doesn't follow is why this would emerge. One possible explanation is that the faster group displaces the slower as they weave through the environment. These perturbations cause minor delays for the slower agents that aggregates to a measurable increase in the mean delay. Further evaluation of agent escape times, specifically broken down by agent speed group, would be needed to validate these assumptions.

5 Future Work

This sort of model never reaches an end-state. Every aspect can be improved. Starting with the agents, per [1], additional features can be modeled to make their reactions to stress more realistic. Age, gender, experience, role, specific challenges, e.g. wheel chairs, blindness, deafness, and the attendant behaviors are ripe for investigation. Along with the intrinsic agent dynamics there are also other agent decision behaviors that can be modeled. Avoiding crowds and following crowds are opposite behaviors that can be beneficial in different scenarios. Modeling behaviors such as these can lead to more realistic simulations.

While we intended to model the aspects of fire, smoke, and their associated side effects, we realized it doesn't contribute to the key evacuation components that we were studying. But, to add these features, can be beneficial for studying other aspects such as path degradation, fire spread, and more dynamic layouts. To compliment the fire's behavior other features could be added such as the fires reaction to materials, smoke visibility effects on agents, and so forth.

There are also improvements that can be made with how the maps were generated, currently the extremely limited strategy of by hand. The layouts could be procedurally generated within some sort of constraints to open up the space that is experimented with. We could also include multi floor maps within NetLogo through some clever world manipulation and the introduction of patches that transport agents to different floors.

While NetLogo was developed in part to aid in teaching about simulations, its use of Lisp like programming dialect can make development difficult for modern practitioners. Porting to Mason [11] or Mesa [12] and their more modern development environments could increase the speed at which new capabilities could be developed. This porting effort would be beneficial in spreading these models and techniques to a broader audience.

6 Conclusion

The demand that our buildings and public spaces are designed with safety in mind is becoming increasingly loud. Before large investments are made in

[11] https://cs.gmu.edu/ eclab/projects/mason/

[12] https://mesa.readthedocs.io/en/master/tutorials/intro_tutorial.html

these projects planners need to have robust tools to understand the impact of their design choices. Every model of the world loses some amount of information in its abstraction. Our models seek to explore where these trade-offs impact realism and where simplified representations can stay relevant. This paper described a first step to realizing the capability of running these types of simulations in the NetLogo environment.

Our experiments attacked small discrete problems in the hopes that this could be the foundation to more complicated experiments built on top of our framework. Understanding that each aspect of our model has room to grow we hope that these experiments and models inspire future work with NetLogo in this field. While there exists other modeling environments with more capabilities, NetLogo remains a key tool to communicate and educate the agent based approach. Our study should show that there are areas that can be modeled effectively in NetLogo if you are prepared to make certain trade-offs . Agent based simulations will remain as one of the few efficient ways to conduct experiments that explore problems of these magnitudes for the foreseeable future.

References

[1] João E. Almeida, Rosaldo J. F. Rosseti, and António Leça Coelho. Crowd Simulation Modeling Applied to Emergency and Evacuation Simulations using Multi-Agent Systems.

[2] E.D.Kuligowski and S.M.V.Gwynne. The need for behavioral theory in evacuation modeling.

[3] Angelika Kniedl, Dirk Hartmann, and Andre Borrmann. Using a multi-scale model for simulating pedestrian behavior.

[4] D Kornhauser, W Rand, and U Wilensky. Visualization tools for agent-based modeling in netlogo.

[5] Maysam Mirahmadi and Abdallah Shami. A Novel Algorithm for Real-time Procedural Generation of Building Floor Plans.

[6] Carol Rice, Ronny Coleman, and Mike Price. Clarifying evacuation options through fire behavior and traffic modeling.

[7] Fangqin Tang and Aizhu Ren. Agent-based evacuation model incorporating fire scene and building geometry.

[8] Eileen Young. Prioritevac: An agent-based model of evacuation from building fires.

[9] Jibiao Zhou, Yanyong Guo, Sheng Dong, Minjie Zhang, and Tianqi Mao. Simulation of pedestrian evacuation route choice using social force model in large-scale public space: Comparison of five evacuation strategies. 14(9):e0221872–e0221872.