Project Number: MQP-C23481

Modeling Pedestrian Flow: A Path-finding Approach

A Major Qualifying Project Report

Submitted to The Faculty

of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Victoria Mirecki

Ally Salvino

2021-22

Approved:

Professor George Heineman

Professor Sarah Olson

Abstract

Cellular automata is often used in modeling pedestrian movement, though this research took a path-finding approach, making use of the A* search algorithm for finding least cost paths. Our research team developed a simulation to model pedestrian exit from an enclosed space, and made comparisons between different room configurations. We evaluated these configurations by averaging an evaluation metric for each set of trials, where a lower evaluation metric is considered "better." The research revealed that the most efficient rooms were rooms with more exits, meaning that as the number of exits increases, exit from the given room becomes better. Additionally, our team found that as the number of adults with backpacks in a classroom increases, exit from that room becomes worse.

Executive Summary

Pedestrian flow modeling and simulations have many useful applications, which include assisting in crowd management and evacuation preparedness (such as in the case of fire or other emergency event). In instances such as these, fire marshals and other officials need to know how long it will take pedestrians to exit a building or enclosed space to ensure that there are enough appropriately placed exits to allow everyone to get out of the building in a timely manner and promote safety. By modeling pedestrian movement within a building, we can predict whether or not a safe exit is possible and make an effort to plan for maximum efficiency when mass exit must occur. Additionally, there is the potential to be able to determine maximum capacities of rooms and determine the best locations of fire exit doors when designing new buildings. This problem of modeling pedestrian flow served as motivation for our project.

The goal of this project was to develop an online simulation tool to model and analyze pedestrian flow using path-finding algorithms for global path decisions in conjunction with localized rules for local movement decisions. In an effort to do this, we began by analyzing existing tools that model pedestrian flow and determining what features we wanted our own simulation to have. Once we had decided on the main characteristics of our tool, we started from an existing code base, modifying existing functionalities of the code while implementing new features for our application. We have developed a simulation that models pedestrian flow using the global path-finding algorithm A* in conjunction with localized rules for conflict resolution in the event that pedestrians collide with one another. Our simulation has an existing initialization for four different types of pedestrians, all of whom take up different amounts of space on the grid: children, adults, adults with backpacks, and adults with bicycles. Our user interface additionally allows for user input for a variety of factors, and produces summary statistics during, and at the conclusion of each run. We then used these summary statistics to make comparisons between runs and to be able to characterize initial room layouts.

Through our research, we were able to simulate real-world examples of pedestrians leaving a room, and numerically evaluate them. For instance, our research revealed that if we keep all other parameters constant (including the number of people in the room) and increase the size of the room, the average exit time over all of the pedestrians increases. This makes sense; as a room gets larger, we expect it to take more time for pedestrians to leave the room. Our evaluation and analysis included many comparisons like this. For example, we compared different ways to resolve deadlock situations, different heuristics used in the A* algorithm, and a varying number of pedestrians in the room upon initialization. The simulation contains random placement of objects when creating the grid. Because of this, our team evaluated each parameter 3500 times to ensure that we were confident in our findings. Therefore, we were also able to show that the mean of the average exit time increases when the grid size increases, for example. In one of our more "real-world" examples in which we were testing on the number of adults with backpacks in a classroom set-up, we found that as the number of adults with backpacks in the rooms increases, so do the exit time and number of collisions, as expected. As mentioned before, this simulation has many potential implications for future work. One would be able to validate these results with some of the fire marshal's standards and then help others in the design and creation of new rooms in buildings. By modeling pedestrian flow out of a room, we were able to quantify the parameters that constitute a "good" room initialization and give recommendations for groups of people leaving a room.

Acknowledgements

We would like to thank our advisors, Professor George Heineman and Professor Sarah Olson, for their continued support, guidance, and encouragement throughout this project. Their passion and vision are invaluable assets to our future careers in mathematics and computer science.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Neural networks, data packets, ants moving in a line, vehicles stuck in traffic, and pedestrians trying to evacuate a building may not seem like they have much overlap, but they do have one key commonality: they are all instances of jamming phenomena [21]. Often times, rule-based modeling, and more specifically, cellular automata, are used to study the movements of these different types of particles [21]. Rule-based models are those in which the entire model is directed by a set of rules allowing it to be simulated and analyzed more easily than other models [8]. A cellular automaton is one such rule-based model that takes place on a grid of cells in which each cell takes on one of several defined states [46]. In many uses of cellular automata, such as Conway's Game of Life, these states include alive and dead, while in other cases, states may be defined by varying colors [13]. At each time step, or generation, a specific set of rules are applied to each cell that can cause the cell to switch states [46].

An additional use of cellular automata, and the one that is the main focus of this paper and research, is jamology, referring to the jamming of self-driven particles [21]. More specifically, we focus on pedestrian flow in this research. The importance of studying pedestrian flow is derived from crowd management and emergency evacuation preparedness. Consider the case of a large office building that houses many employees and visitors throughout the day. In the case of a fire, it is imperative that everyone is able to exit safely and in a timely manner. Creating models that allow researchers and developers to measure this and place a sufficient number of exits in appropriate locations throughout the building is key [22].

Rule-based models and cellular automata can be used in the modeling of pedestrian flow. In the basic approaches to modeling pedestrian flow, a two-dimensional grid of square cells is used and cells are designated as either empty or occupied by a pedestrian, with only one pedestrian able to occupy a given cell at a time [21]. At each discrete time step, pedestrians may either move to a neighboring cell if it is unoccupied or remain in their current position. They do so with a certain probability, one that changes when conflicts (two or more pedestrians attempting to move to the same cell) occur [22]. With such a probabilistic approach based on localized decisions, movement from one cell to another for each pedestrian is based primarily on the surrounding cells. Instead, our research team chose to take a path-finding approach, utilizing global decisions based on a distant target (an exit), in conjunction with localized decisions based on obstacles, other pedestrians, and collisions.

Several models and code repositories exist that can be used in the modeling of pedestrian movement, flow, and evacuation (see Section 3.1 for a summary of these approaches) [29, 48, 42, 11]. These existing tools possessed many of the features that we were interested in our own simulation tool having, though there were some additional complexities that our team was interested in modeling and exploring. The primary characteristic of interest that differs from current models is the incorporation of density and pedestrian occupancy levels into the movement analysis using a metric of movement inspired by Fruin's

Level of Service (LoS), referring to six different levels classifying free movement to restricted movement in crowds [12].

This forms the basis of and motivation for our project team's research. The goal of this project is to develop a graphical user interface (GUI) that models pedestrian flow using globalized path-finding algorithms in combination with supplemental localized rules. The model will additionally make use of occupancy level analysis inspired by Fruin's LoS, in addition to a global density analysis based on exit location in an effort to avoid high-density situations near the exit points. This occupancy analysis, along with the average amount of time it takes all pedestrians to exit through any designated exit and the average number of times that pedestrians collide with one another will be factored into the characterization of the evacuation. This will help ensure that pedestrians in the model are moving as quickly as possible to exit while still observing safe crowd area occupancy levels. Upon the completion of this model, we compared pedestrian exit times, collision counts, and other statistics between different runs of the simulation to determine what common initializations characterize the most efficient and effective exits. Our team found that the most efficient room exits occurred in rooms with more exits, indicating that as the number of exits in a room increases, exit from that room becomes better. We additionally found that as the number of adults with backpacks in a classroom increases, exit from that room becomes worse. These results, and others, are discussed in detail later in the paper.

Chapter 2

Background

2.1 Rule-Based Modeling

Ruled-based, or cell-based models, are mathematical models which are used to represent the behaviors of different systems [18]. For these models, a grid-like space is used where each individual grid-square, or cell, is given a set of rules to follow. These local rules are generally the same for every cell, and by following the rules, a behavior for the entire system emerges. Rule-based models follow a bottom-up approach, as each individual cell is represented and each cell follows the local rules given while the behavior of the whole system is observed [18]. Figure 2.1 is a visualization of some different example cell behaviors that may occur after one generation (also commonly referred to as an iteration or time-step). In the figure below, white blocks represent dead cells, black blocks represent living cells, and gray blocks represent a different type of living cell that has changed types. Here, the type of behavior that occurs is based on the predefined rules of a particular system. As you can see, from the same initial state, a cell can divide and create a new cell, migrate and change location, die and disappear, or differentiate and become a different cell.

These rule-based models are useful because of their ability to model individual cell behavior in conjunction with interactions between cells. Before these models were used, the traditional approach to representing systems was very limiting, especially in the field of biology, due to the complex behavior of cells. Cells consist of many parts, and have the ability to interact in many ways, which was part of the motivation behind the development of rule-based models. The traditional methods required explicit specification of the entire system being modeled which caused evaluation times to become too expensive. Therefore, rule-based models were developed in which the model is comprised of a set of rules which can be processed by simulation and analysis tools more efficiently [8].

Rule-based models are unique in the sense that they offer both structure and flexibility. The structure comes from the rules and is what allows for efficiency. In other words, rule-based models take in data, such as an initial condition with specific states assigned to each cell, and use a set of predefined rules dictating how these cells' state's will change with each iteration. In addition to taking in data and manipulating it, rule-based models may also use data for validation (to check that the rules are working in the expected manner), or may use data to inform the rules (different initial conditions leading to different rules). However, the creator of the model has the flexibility in choosing the rules, which can often be thought about as if-then statements. For example, if X condition holds, then Y happens [44]. Therefore, there are an infinite number of models that can be created to model a wide variety of phenomena. Rule-based modeling is used in various fields of study to represent a wide variety of different populations including ecosystems, traffic, people in a crowded room, or characters in animated games [18]. However, it is important to keep in mind the limitations of these models. It can be difficult

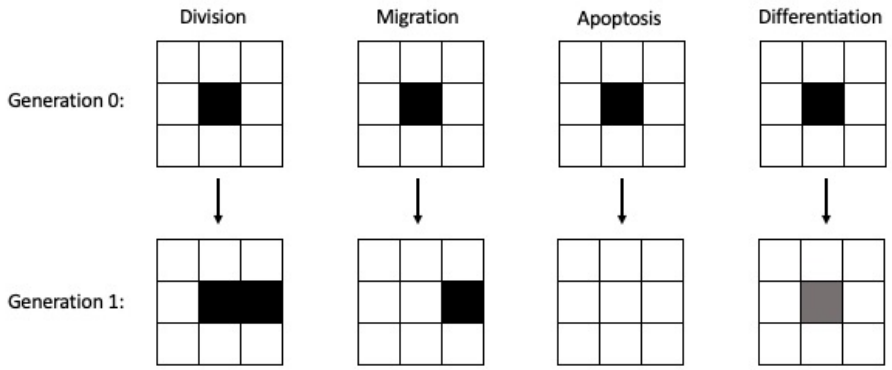


Figure 2.1: Different cell behaviors that can occur in a rule-based model. From one generation to another, a cell (shown in black) can divide and create a new cell, migrate and change location, die and disappear (shown in white), or differentiate and become a different cell (shown in gray). Adapted from [18].

to predict and develop a simple set of rules for living systems due to their complex and natural habit of unpredictability of how they perform in practice [18].

2.2 Cellular Automata

A cellular automaton (CA) is an example of rule-based modeling, also known as a cellular space, tessellation automaton, homogeneous structure, cellular structure, tessellation structure, or iterative array, and is a discrete model of computation. The model begins with a fixed-size grid-like structure of cells (also of fixed size) in which each cell operating on the grid has a certain number of possible states (represented on the grid) [46]. Cell states change with each time step, or generation, based on rules about how an individual cell should act given its neighborhood, or the designated cells surrounding it [38]. With an initial configuration of cells in starting states, the cells will follow the given rules, and patterns may be observed in the resulting configuration. In the traditional approach to modeling CA, the rules for each cell on a grid are the same, do not change over time, and are applied to the entire grid at once for a simultaneous update as opposed to a sequential one [35]. Although CA are defined more thoroughly by their set of rules, they can be divided into categories based on the results of those given rules. There are four common types of patterns that emerge in many CA models or simulations:

1. Patterns that generally stabilize into homogeneity.
2. Patterns that evolve into mostly stable or oscillating structures.
3. Patterns that evolve in a seemingly chaotic fashion.
4. Patterns that become extremely complex and may last for a long time with stable local structures.

Evaluating different cellular automata allows researchers to analyze the manner in which different rules, starting configurations, neighborhoods, numbers of states, and more affect the behavior of the cells as a whole [38]. CA have many interesting real-world applications that mathematicians, computer scientists, researchers, and others continue to investigate and take advantage of. These include applications in the fields of pedestrian flow, image generation, alteration, and recovery, tessellations, cryptography, and more [36, 33, 40]. We investigate pedestrian flow in detail later in the paper, but here we introduce some of the other common applications.

2.3 Applications and Uses of Cellular Automaton

2.3.1 Conway's Game of Life

One example of a popular cellular automaton is Conway's Game of Life. Developed in 1970 by John Conway, Conway's Game of Life is a type of CA on an infinite two-dimensional grid of squares in which each cell has two possible states: dead and alive [13]. Game of Life is based on three main rules of interaction between the cell of interest and the eight cells sharing a vertex with it, also known as the Moore neighborhood (refer to Figure 2.3) [20]. These rules are as follows, and are applied to all cells simultaneously at each generation:

1. A live cell with two or three live neighbors survives.
2. A dead cell with exactly three live neighbors becomes alive.
3. All other cells die.

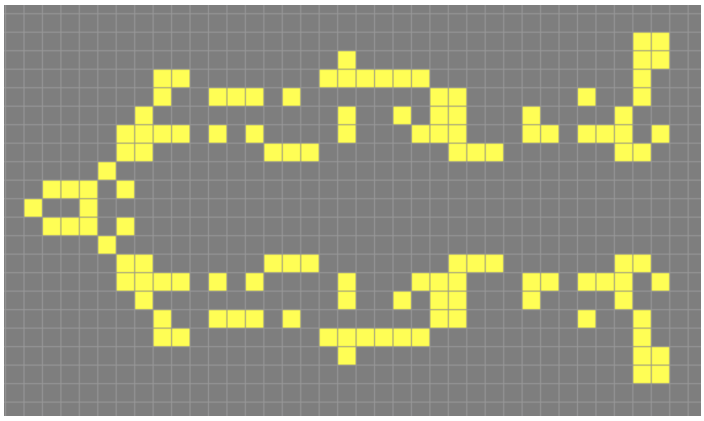


Figure 2.2: A spaceship pattern resulting from running Conway's Game of Life on an online program, lexicon, where yellow cells are alive and gray are dead [28]. This resulting pattern is based on an initial configuration, and, in this case, moves left across the board over generations.

Game of Life is a popular hobby for CA researchers and enthusiasts due to the unique patterns that have been shown to emerge. From an initial configuration, anything can happen. As such, at the onset of the game, new discoveries were constantly being made. For example, one common pattern emerging in the Game of Life is the glider, originally discovered by Richard Guy. Gliders travel diagonally across the grid and have four repeating generations. A glider is a type of spaceship, one of the three main types of patterns that can occur in Game of Life, with the other two being still lifes and oscillators of varying periods. Still lifes are those patterns that do not change from one generation to the next,

while oscillators return to their initial configuration after a certain number of generations (known as the period). Spaceships, on the other hand, translate across the grid. Figure 2.2 is an example of a spaceship configuration. This image is only a screenshot of a program running the Game of Life, and therefore only captures the board configuration at an instant, or generation, but this configuration moves across the board, hence the name spaceship. There are numerous simulations that have been created to visualize Conway's Game of Life and there are many interesting applications and variations to the original idea of the Game of Life, including those featuring alterations to the neighborhood, number of states, rules, shape of cells, and other characteristics [13].

Some variations on Conway's Game of Life include those resulting from changing the shape of cells, such as from squares to triangles, changing the number of states that a cell can take on, such as "dying" in addition to alive and dead, changing the rules that dictate whether a cell lives or dies (or takes on another state), such as including the condition that one live neighbor results in the birth of a new cell, and changing the way in which the neighborhood of a cell is defined, such as the four cells sharing an edge with the cell of interest (von Neumann neighborhood) or the eight cells sharing a vertex with the cell of interest (Moore neighborhood) [20]. Figure 2.3 shows these two primary different types of neighborhoods.

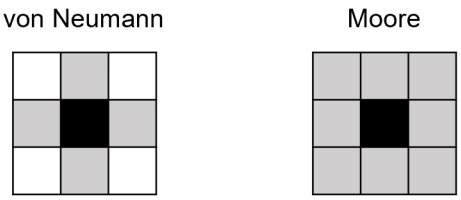


Figure 2.3: Two primary types of neighborhood used in rule-based automata, where the black cell is the cell of interest and the gray cells represent those belonging to the respective neighborhood (adapted from [9]).

In the Game of Symmetry, for example, cells still take on only two states, - alive and dead - but the rules that determine a cell's state at each time step are different; dead cells come back to life when their neighbors are symmetrically distributed, living cells with symmetrically distributed neighbors survive, and living cells with asymmetrically distributed neighbors and the inability to become symmetrically distributed with only one change die. Figure 2.4 shows the sixteen meta-configurations that make up the Game of Symmetry. These sixteen meta-configurations are the only possible patterns given any initial set-up and may appear as stand-alones or in combination with others. From an initial configuration in the Game of Symmetry, the grid generally evolves to become symmetrical and often gives way to still lifes (collections of isolated particles). Additionally, the Game of Symmetry is considered a relatively simple implementation for infinite growth (consider two particles directly adjacent to one another as a starting configuration; this will give way to infinite growth in either adjacent direction) [25].

In another variation on Conway's Game of Life, known as Life 1133, a cell remains alive only if it has exactly one live neighbor cell, and it becomes alive only if it has exactly three live neighbors. In this CA, research has found that an initial configuration must eventually stabilize and cannot grow without limit. Additionally, a glider must exist and occur naturally [15]. The final variation on Conway's Game of Life mentioned here is that of triangular tessellations. In this version of Life, the Game of Life is played on a triangular, or isometric, grid in which the neighborhood for a cell consists of the three cells sharing an edge with it. Figure 2.5 shows an example of such a triangular grid and neighborhood. Significant research has been done into the results of various rules on the triangular grid. One such

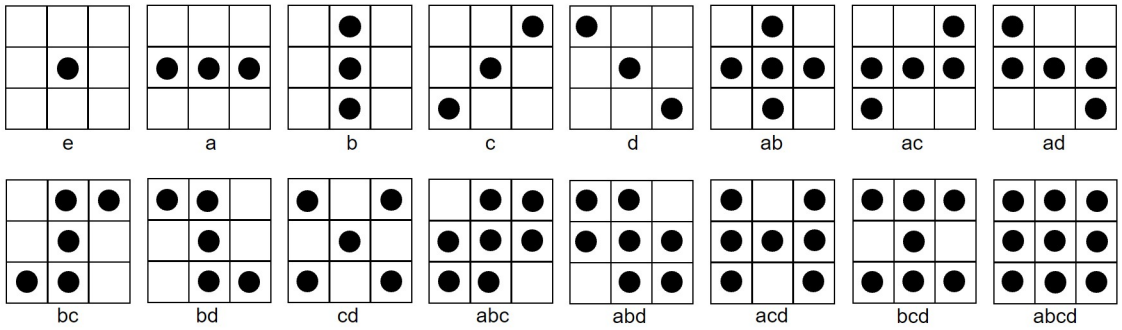


Figure 2.4: The 16 possible patterns that can appear in Game of Symmetry at a specific time step (may change at the next generation) given any initial state where the black dots represent live cells and the white boxes are dead cells (adapted from [25]).

example is that any birth/survival rule in which having exactly one live neighbor is one of the conditions for birth leads to unbounded growth [33].

2.3.2 Cryptography

Cryptography refers to the practice and study of methods for secure communication, and is most often used to defend and protect against adversarial behavior. Typically, the process takes a string of plaintext as input and applies an algorithm to it, creating an encrypted output (also known as ciphertext). The ciphertext can then only be deciphered by someone with the correct key [45]. Many different algorithms exist for the encryption of these initial inputs, and an implementation of Conway's Game of Life is among them.

Conway's Game of Life can be used as an encryption technique and has been chosen due to its reduced run time and complexity, as well as that it is less memory-intensive than many other methods [40]. Conway's Game of Life was also implemented as an encryption algorithm for cryptography due, in part, to its irreversible nature, meaning that given the ending configuration, it is nearly impossible to get back to the initial state, even knowing the rules. If there were a way to reverse the steps of the game and, knowing the ending configuration and the rules, get back to the initial configuration, this could additionally have applications to solving the halting problem in computer science theory. The halting problem refers to the computability problem that given an input and the knowledge of what a

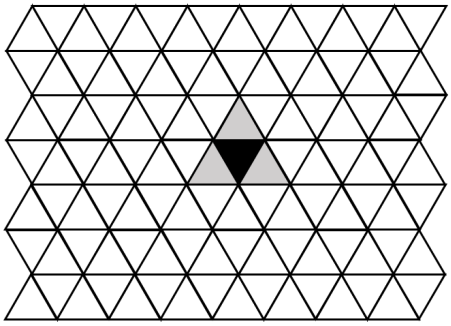


Figure 2.5: Triangular grid with sample triangular grid neighborhood in which the black cell is the cell of interest and the gray cells correspond to the neighborhood (adapted from [33]).

computer program does, there is not a way to determine whether it will finish running or continue to run forever [3]. The same problem occurs when implementing Conway's Game of Life, as there is no way to know whether an initial configuration will run forever, develop into a stable pattern, or die off.

2.4 Jamology and Pedestrian Flow

Another application of cellular automata, and the one that is the focus of this research and paper, is within the field of jamology. Jamology refers to the jamming of self-driven particles, including the jamming of data packets in the internet, neurons firing in neural networks in the brains of people with Alzheimer's disease, and cars in traffic [21]. The primary focus of this research is the jamming of pedestrians and pedestrian flow with respect to the movement of large quantities of pedestrians, though we begin with a discussion of autonomous vehicles and jamology.

2.4.1 Modeling Autonomous Vehicles Using Cellular Automata

Recent technological advancements in sensors, camera, and radar, along with software advancements, have allowed for the development of cars with built in safety features like collision warning, collision intervention, and driving control assistance [10]. Continued advancements with these technologies will allow for companies like Tesla, Apple, Ford, and Waymo to develop fully automated vehicles [39].

In addition to autonomous vehicles having the ability to increase our safety, they also have the potential to increase traffic flow and decrease traffic jams. In fact, Yang, Lui, Zhao, and Wu used cellular automata to simulate traffic flow with different proportions of self-driving and non self-driving vehicles [47]. Overall, as the proportion of self-driving cars to non self-driving cars increased, traffic flow also increased. The rules for this CA were based on vehicle safety distance (since it is assumed to take longer for non self-driving cars to react) and lane-changing probability. Additionally, this model was used under different conditions, including single lane roads, multi-lane roads, number of lanes constantly changing, density of vehicles, and including dedicated lanes for self-driving cars. All in all, the results showed that when all vehicles on the road were self-driving, traffic flow was the highest. In addition, they also found that the dedicated self-driving lane has no influence on traffic flow and that when the proportion was higher, that meant that traffic flow was better when the road density was high (peak travel hours). They also found that the vehicle merging improved when the proportion of self-driving cars to non-self-driving cars was .63. As seen in this study, CA has the ability to not only simulate interactions, but also allow for analysis into these scenarios that can be applicable to the real-world [47].

Existing Autonomous Vehicle Modeling Tools

Modeling tools are useful as they allow for the representation and testing of scenarios that take too much time and resources to test in the real world. One such scenario relates to self-driving cars and their effectiveness on traffic flow. Some traffic flow simulators have all the cars on the road being autonomous, while others have the ability to represent various proportions of self-driving to non-self-driving cars on the road [48, 30]. Figure 2.6 is a screenshot of a simulation, React, that was created to address the research question, "If all cars were driverless, would we need traffic lights?" In Figure 2.6, the letters represent cars, the black strips are the roads, and the green blocks are barriers where the cars cannot go, like buildings. Each car is either labeled "n," "e," "s," or "w", and it is used to represent their desired cardinal direction (north, south, east, or west of the board). When the simulation runs,

all of the cars begin moving and eventually reach their desired location, stopping as needed to avoid collisions with other cars. The research answer from this simulation is no, we would not need traffic lights if all cars were driverless [30]. This is because the cars are able to recognize when to move, and more importantly, when not to move. This simulation is able to model driverless cars due to the rules created for the model, and by defining the rules that each cell follows, the cars are able to move to their desired goal destinations.

Not only does React visualize driverless cars, it also gives a numerical summary of the simulation. This is helpful to see the number of times the cars have to stop, and it also allows the user to alter other parameters that might vary in real time, like courteousness of the drivers. By giving the numerical analysis and allowing the parameters to change, one can compare the effectiveness of driverless cars with other models with different parameters. However, this simulation only incorporates roads where all cars are driverless, and does not allow modeling of non-self-driving cars.

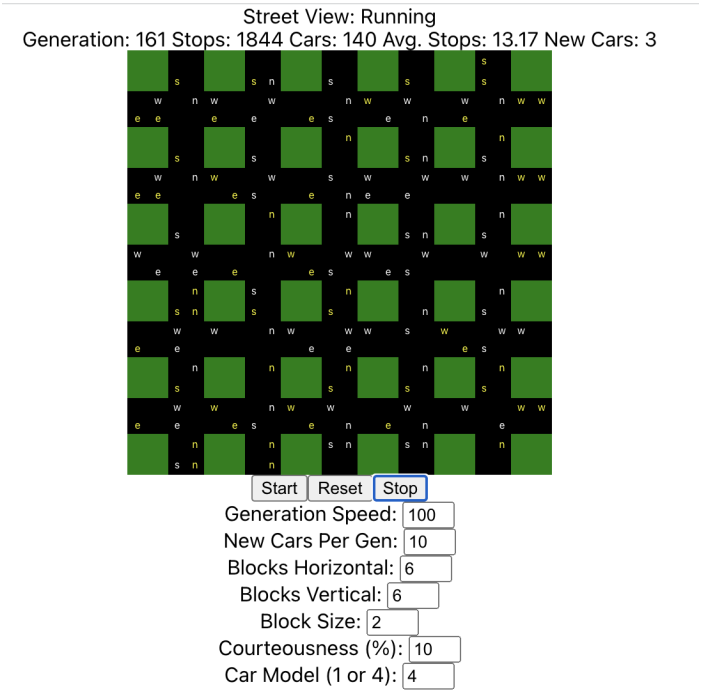
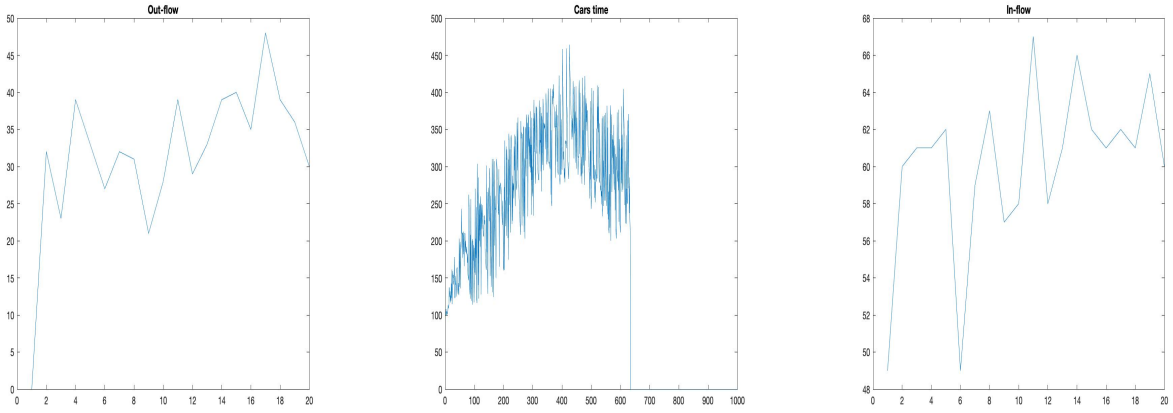


Figure 2.6: Screenshot of the simulation React that models the traffic flow of cars moving through streets and intersections where the black portions are the roads and "n," "e," "s," and "w" denote a car moving in the specified direction [30].

One simulation tool that does model both non-self-driving and self-driving cars on a highway is Auto-Car. This simulation allows the user to alter the length of the path, number of cars on the grid, acceleration, deceleration, speed limit, time step, self-driving lanes, probability to generate a car somewhere on the grid, proportion of self-driving to non-self-driving cars, probability of random slow cars, speed of random slow cars, length of a car, traffic counts, and time of simulation. It also produces three graphs after the simulation ends as shown in Figure 2.7. These three graphs correspond to the inflow of the cars on the highway, the outflow of cars on the highway, and the time taken for the cars to travel across the highway [30]. While this simulation allows for the alteration of many parameters, the graphs that are produced do not have the x and y-axis labeled, so it is difficult to tell exactly what is being represented from the graphs. In addition, this simulation only models one-way highway traffic, while React simulates turning onto different streets and having varying destinations. These two tools

are useful for future work, especially since many believe that our future contains driverless vehicles. Further in this report, we discuss more in depth the pros and cons of these models, and how we plan to implement some of their features into our project.



(a) Out-flow graph **(b)** Cars Time graph **(c)** In-flow graph

Figure 2.7: The three graphs produced after running the spacing continuous cellular automaton simulation, Auto-Car [48].

2.4.2 Pedestrian Flow

Pedestrian flow can be modeled using cellular automata and is used for crowd management, as well as to analyze and predict evacuation and the placement of emergency exits. There are several hazards related to crowd movement that result in injuries such as suffocation, crushing, trampling, and in some extreme cases, even death [36]. Modeling pedestrian flow and pedestrian movement can offer insight that helps ensure that buildings have sufficient exits to allow for safe evacuation in case of fire or other emergencies. It also encourages efficient exit from mass gatherings and large events, such as football games and concerts. For modeling pedestrian flow using CA, the asymmetric simple exclusion principle, a concept that is used in the modeling of self-driven particles (such as pedestrians), explains that only one particle may occupy a given cell at a time. In the realm of pedestrian flow, this means that no more than one person can occupy a particular cell at once [21].

Modeling Pedestrian Flow Using Cellular Automata

The asymmetric simple exclusion principle, named asymmetric because the particles tend to move in one direction towards a goal, applies to all versions of pedestrian flow modeling using CA [21]. However, beyond that consistency, there are many different modeling variations. The base model takes place on a two-dimensional grid made up of squares that are approximately 40 cm by 40 cm [34]. In this approach, each pedestrian occupies only one square cell at a time, with each cell designated as either empty or occupied [22]. At each discrete time step, pedestrians have the ability to remain in their current cell, or move to a neighboring cell, assuming that it is available [34]. Pedestrians will move to a new cell with a certain probability, one that changes when conflicts occur (two or more pedestrians trying to move to the same cell) in order to avoid collisions.

In an alternative model, a two-dimensional grid of squares is used, with each square 5 cm by 5 cm or 8 cm by 8 cm, significantly smaller than the base model. This is referred to as fine grid cellular

automata, and is used to more accurately model the varying sizes and movements of a pedestrian. For example, intuitively, it is known that people are different sizes (child vs. grown adult) and can take up different amounts of space, including the additional space needed when wearing a backpack, walking with a bike, etc. [34]. In this approach using fine grid CA, pedestrians still have the ability to move at each discrete time step, though they are driven forward by the cell corresponding to their center of mass [34].

As CA approaches for modeling pedestrian flow become more complex, they additionally expand to account for several other characteristics of pedestrians and their interactions. One such characteristic is the happiness of pedestrians. Some research reveals that "happy" pedestrians tend to move in a preferred direction, while "unhappy" pedestrians move in a more random way [7]. It is additionally of note that there are different types of interactions and relationships that can determine interpersonal distances between pedestrians:

1. Mutually positive, such as friends and couples.
2. Mutually negative.
3. Asymmetrical, such as parents and children or workplace superior and inferior.
4. Neutral.

These four different types of relationships affect interpersonal distances between pedestrians (as space allows) such that individuals in mutually positive relationships will tend to position themselves the closest together, individuals in asymmetrical and neutral relationships will leave space between themselves, and individuals in mutually negative relationships may actively avoid standing close together or crossing paths [24]. Pedestrians can also move with different speeds. These differences in speed can occur between people of varying age and mobility ranges (consider again the example of a child vs. a grown adult) [24]. Pedestrian walking speed can additionally vary based on proximity to the center of attraction or destination, timing of the day, and other event-based characteristics [27].

Some models also take into account varying crowd densities. For example, in more dense crowds that often occur during mass exit from large gatherings or during emergency evacuation and panic situations, pedestrians are positioned more closely together and are likely required to move slower than they would in an open space. Alternatively, when given the ability to space themselves out in a less dense environment, pedestrians typically will, also moving at a more normal walking pace [12]. Fruin's Level of Service (LoS) is one tool that can describe this phenomenon, depicted in Figure 2.8 [12]. Fruin's LoS consists of six levels, A through F, describing the different ranges of area occupancy that a given pedestrian may experience:

1. Level A: More than 35 sq. ft./pedestrian; pedestrians have enough space to choose their own walking speed, pass other pedestrians, and avoid crossing conflicts.
2. Level B: 25-35 sq. ft./pedestrian; pedestrians have enough space to choose a normal walking speed and bypass others, while minor crossing and reverse direction conflicts may occur.
3. Level C: 15-25 sq. ft./pedestrian; pedestrians have a restricted ability to freely select their walking speed and pass others and there is a high probability of conflict requiring frequent adjustment.
4. Level D: 10-15 sq. ft./pedestrian; most pedestrians experience restricted walking speeds due to difficulty passing slower walkers and avoiding conflicts.

5. Level E: 5-10 sq. ft./pedestrian; nearly all pedestrians experience restricted walking speeds, as well as extreme difficulty in cross-flow and reverse-direction.

6. Level F: Less than 5 sq. ft./pedestrian; all pedestrians have extremely restricted walking speeds and progress occurs via shuffling. Conflict is likely unavoidable.

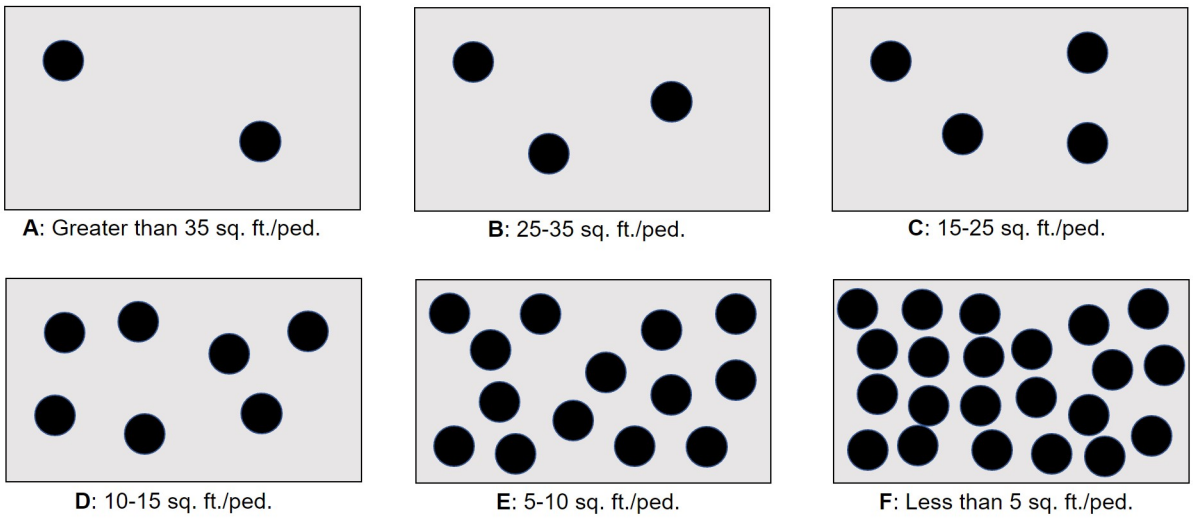


Figure 2.8: Image depicting the occupancy levels at each of Fruin's six levels of service where a black dot represents a pedestrian (adapted from [12]).

These ranges and descriptions for Fruin's LoS can be used to identify crowd occupancies that will and won't support safe or optimal exit from mass gatherings or in the case of emergency evacuation [12]. It is of note that the research reveals that pedestrians in different environments and parts of the world are comfortable with different crowd densities. For example, research has found that pedestrians in India are less concerned with bumping into people (pedestrian conflict) compared to pedestrians in Germany. Pedestrians in India are additionally reported to walk faster than those in Germany in high density situations [14].

Existing Pedestrian Flow Modeling Tools

There are currently several existing tools that assist in the monitoring of pedestrian flow and evacuation. Some, such as PTV Viswalk, are commercial products available for purchase. PTV Viswalk is a pedestrian or crowd flow simulator meant to assist in reproducing human movement behavior in a virtual environment [29]. AnyLogic is another commercial tool that can be used for simulating pedestrian flow (see Figure 2.9). In this model, users can simulate an evacuation due to fire from an office building. This tool also features time data, telling users how long the evacuation took for a given number of pedestrians inside at a certain time of day [11]. There are additional github repositories available that can be used to simulate pedestrian flow and movement using cellular automata [48, 42].

As one of the early steps in developing our own simulation tool, we used and analyzed all four of the above modeling tools, making note of the advantages and disadvantages of each, as well as the features that we wanted our own implementation to have. These pros and cons and a list of the desired features are presented in Chapter 3 on the project design.

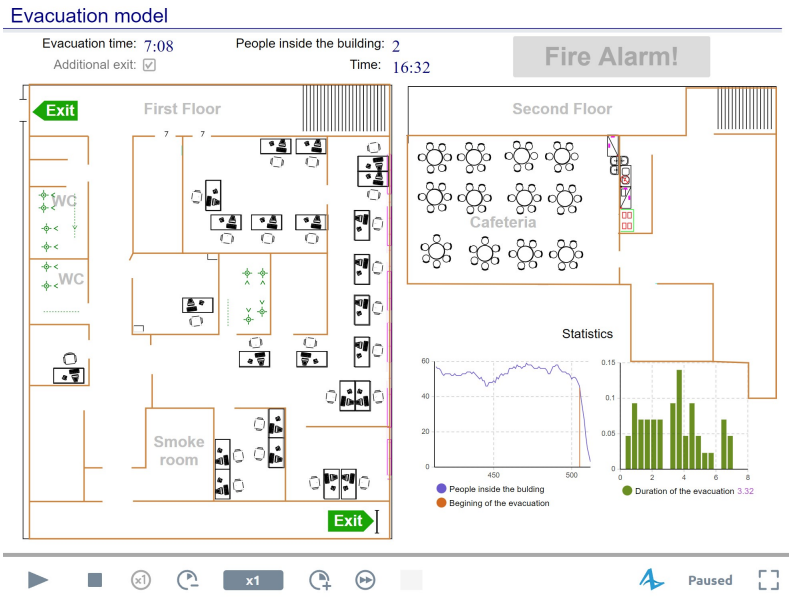


Figure 2.9: A screenshot from AnyLogic, one of the pedestrian evacuation modeling tools [11]. AnyLogic allows the user to choose the speed and when the "fire alarm" is triggered and inputs statistics including a graph of the number of people inside the building over time and the duration of the evacuation.

Chapter 3

Project Design

The goal of this project was to develop a graphical user interface (GUI) that would act as a simulation tool to model pedestrian flow and movement within a given space. As discussed in previous sections, such a tool would prove useful in assessing and simulating crowd management tactics, such as specific room layouts and exit assignments based on certain characteristics. In addition to modeling pedestrian flow using such a tool, we also wanted to provide on-screen analytics based on the simulation and offer the ability to save and utilize data for further statistical analysis between runs of the simulation.

3.1 Existing Simulation Tools

After significant background research that helped us determine that we wanted to create such a simulator, we first investigated what tools already existed for modeling pedestrian flow. In doing so, we discovered four main simulators and determined what features we liked that they had that we wanted to include in our own simulation, as well as what additional features we wanted our own modeling tool to have (if any) in an effort to decide on the features and characteristics that we wanted our own tool to possess.

1. PTV Viswalk [29]:
    - Description: Commercial product available for purchase that serves as a pedestrian or crowd flow simulator and is meant to assist in reproducing human movement behavior in a virtual environment
    - Advantages: Calculates summary statistics such as number of pedestrians and run times at the end of a run, already set up as a user interface
    - Disadvantages: Not an open-source code base, interface is a bit confusing, some functionality (like drawing areas) are not intuitive and are difficult to use (at least in the demo mode)
2. AnyLogic [11]:
    - Description: Commercial product in which users can simulate pedestrians exiting a building after a fire alarm (or other emergency alert)
    - Advantages: User has the ability to trigger the "fire alarm" to begin exit, simulates regular pedestrian movement throughout the day in an office building (busiest during the core hours of the work day), provides some counts and statistics of how many pedestrians are in the building and how long it takes all pedestrians to exit once the alarm is set off

- Disadvantages: User can't place pedestrians or obstacles, user has minimal control (can only adjust speed and pull alarm)

3. Pedestrian Flow Simulator Github [48]:

   - Description: Open-source pedestrian flow simulator based on cellular automata that can be used to dynamically simulate pedestrian flow in an enclosed space and generate a real-time animation
   - Advantages: Creates graphs of summary statistics at the end of a run, allows for altering many parameters, relatively straightforward code
   - Disadvantages: Not well documented, graphs are unlabeled, difficult to understand the meanings in their evaluation

4. Pedestrian Flow Automata Github [42]:

   - Description: Open-source pedestrian flow simulator that utilizes cellular automata and objective matrices to probabilistically group and move pedestrians
   - Advantages: Although not currently functional, this tool does seem like it would have the ability to add exits, entrances, obstacles, and pedestrians, has a nice UI set up already
   - Disadvantages: Not well documented, some functionality doesn't work (adding exits, entrances, obstacles, pedestrians)

After conducting this analysis, we were able to decide on the features that we wanted our own simulation tool to have in addition to the basic characteristics of the tool. These features included:

- Modeling pedestrian flow using path-finding algorithms
- Providing pedestrian exit times, pedestrian collision counts, and other statistics (and supporting their visualization)
- Allowing users to create their own grid initializations (and have some of our own loaded in, such as a classroom layout)
- Allowing users to save or export a gif of the run
- Allowing users to change the speed of the simulation, in terms of the amount of time between grid updates, along with other parameters

In order to accomplish this, we began with an existing code base that makes use of rule-based modeling: Javascript Ants from Sim4edu [1]. The simulation works to model ant behavior by mimicking the situation in which ants find food and bring it back to the nest. The basic idea is that ants leave a chemical pheromone trail behind them as they move. Thus, once a few ants have found the food pile, other ants begin becoming attracted to the trail, eventually locating the food and returning to the nest with it. See Figure 3.1 for an image of the original ant simulation [1]. While this code did serve as our starting point, we made several modifications and changes outlined in the following sections and chapter.

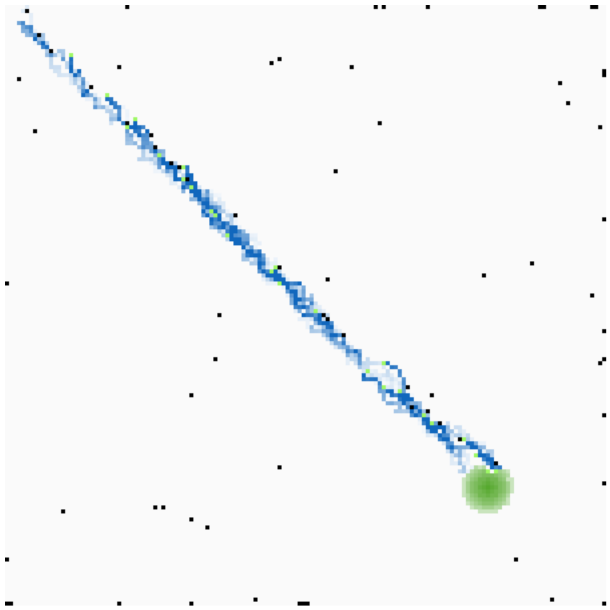


Figure 3.1: Screenshot from the motivating simulation, showing the ants (black) locating the green food source and returning to the nest while leaving their fading blue pheromone trail [1].

3.2 Initialization of the Model

In order for our simulation to run, it requires several initial parameters, all of which have preset default values if the user makes no adjustments. In particular, these initial parameters relate to the pedestrians and the grid used in the simulation.

3.2.1 The Pedestrians and State Space

In day to day life, we see and encounter a wide variety of people and pedestrians. We see children, who are small and take up less space than adults. We see adults of all different shapes and sizes. We see adults and children alike wearing backpacks, wheeling suitcases, carrying boxes, and anything else you can imagine. We see people skateboarding, bicycling, rollerblading, and much more. All of these distinct pedestrians take up different amounts of space, an important fact of life that we didn't want to overlook in our simulation. For that reason, we decided to make use of fine grid CA in our simulation and develop a model in which, although no more than one pedestrian can occupy a cell on the grid, each pedestrian may take up more than one cell on the grid, depending on their age, size, means of transportation, objects they're carrying, etc.

To be precise, we begin by acknowledging that each cell on the two-dimensional grid is a discrete entity represented as a point. Each cell can be either occupied or unoccupied, though it can not be partially occupied, meaning that each cell is identifiable by a single state. The cell state space is $\mathcal{S} = \{\mathcal{P}, \mathcal{O}, \mathcal{E}\}$, corresponding to the three possible states:

1. $\{\mathcal{P}\}$: Pedestrian
2. $\{\mathcal{O}\}$: Obstacle
3. $\{\mathcal{E}\}$: Exit

If a cell does not belong to one of these states at a given time step, it is unoccupied. Two of the three states are static and do not change between generations of the simulation: obstacles, $\{\mathcal{O}\}$, and exits, $\{\mathcal{E}\}$. The location of an object belonging to one of these two states is represented as an x and y-coordinate, (x, y). For the case of an obstacle, the (x, y) location corresponds to the single obstacle cell. For the case of an exit, however, the (x, y) location identifies the location of the anchor point for the given exit, while the exit itself has a length of four cells. The anchor point is either the location of the topmost cell (if the exit is vertical and on the left-hand or right-hand edge of the grid) or the location of the leftmost cell (if the exit is horizontal and on the top or bottom edge of the grid).

The remaining state for pedestrians, $\{\mathcal{P}\}$ has many dynamic attributes that will likely change over the course of the simulation. Despite the changing attributes throughout each run, there is a fixed number of pedestrians that are initialized at time t_0 and tracked over time, although there are four different classifications of pedestrians (discussed below). The number of each type of pedestrian only changes when one of the pedestrians exits the board, at which point the number of pedestrians of the specified type decreases by one. When a pedestrian reaches an exit, they leave the simulation and are no longer being tracked, so information about them is no longer being saved. The total number of pedestrians currently on the grid at any given time step is N^t, where t represents the current generation.

The attributes that we save for each pedestrian include their location. A pedestrian's location is represented as (x_t, y_t) where x_t is the x-coordinate location at the t^{th} iteration, and y_t is the y-coordinate location at the t^{th} iteration of the simulation. Similar to the case of the exit, the location, (x_t, y_t), of one of these non-static entities denotes the location of the anchor point of the object at a given time step. For each type of pedestrian, the location of the anchor point on the pedestrian is the same (for example, for the adult, which is made up of two cells directly next to one another, the left-most cell is always the anchor point). Each of the four types of pedestrians takes up a different amount of space on the grid, discussed in the next subsection, Section 3.2.2. Additionally, each non-static pedestrian has several characteristics (attributes in the code). These include:

- $PT = \{\mathcal{C}, \mathcal{A}, \mathcal{B}, \mathcal{K}\}$ represents the type of the pedestrian where $\mathcal{C}$ represents a child, $\mathcal{A}$ represents an adult, $\mathcal{B}$ represents an adult with a backpack, and $\mathcal{K}$ represents an adult with a bike

- Orientation in one of the eight possible directions that is chosen based on the designated path (discussed more in Chapter 4) and may change at each iteration of the simulation; saved as a direction, θ

- N_{PT}^t represents the number of pedestrians of the specified type on the grid at time step t, N^0 represents the total number of pedestrians at the beginning of the simulation, and N_{PT}^0 represents the initial number of pedestrians of the given type at time $t = 0$

- $A_{PT_i}^t = (x, y)$ represents the grid location of the anchor point for each pedestrian of the specified type on the grid at time step t and $i = 0, 1, 2, ...$ is the specific pedestrian

- $E_{PT_i} = (x, y)$ represents the grid location of the anchor point of the nearest exit for each pedestrian of the specified type on the grid upon initialization based on the user-selected calculation metric (discussed in Section 3.2.3 and computed between the anchor point of the pedestrian and the anchor point of the exit without accounting for obstacles or other pedestrians in the way) and $i = 0, 1, 2, ...$ is the specific pedestrian

- $P_{PT_i}^t = [(x_0, y_0), (x_1, y_1)..., (x_n, y_n)]$ represents the x and y-coordinates of the profile of the specified pedestrian PT_i at time step t and $i = 0, 1, 2, ...$ is the specific pedestrian, where the profile serves to form the shape of the pedestrian

- $W^t_{PT_i}$ represents the number of steps that a specified pedestrian has been waiting to be able to make their move at time step t, where the value resets each time the pedestrian is able to make a move, and $i = 0, 1, 2, ...$ is the specific pedestrian
- C_{PT_i} represents the number of collisions that a specified pedestrian has been involved in where $i = 0, 1, 2, ...$ is the specific pedestrian
- X_{PT_i} represents the time that it takes the pedestrian to exit the grid from their starting position and $i = 0, 1, 2, ...$ is the specific pedestrian
- Path corresponding to the location of each cell that the pedestrian occupies over the duration of their time on the grid; saved as a list of x and y-coordinates $[(x_1, y_1), (x_2, y_2), ...]$ (note that this path also accounts for repeated steps in the same location which are possible if the pedestrian has experienced a collision or is waiting to make their next move)
- $O^t_{PT_i}$ represents the average area occupancy of the pedestrian at each time step t and $i = 0, 1, 2, ...$ is the specific pedestrian

3.2.2 The Grid

Similar to the approach used for CA, we decided to use a two-dimensional grid made up of square cells. However, our simulation differs from those utilizing CA in a number of ways. Each square cell on the grid represents a space of one foot by one foot. This decision was made based on the average size and width of an adult and to allow pedestrians of different ages, sizes, and carrying different objects to be represented. As mentioned in the previous subsection, we implemented four different types of pedestrians in our simulation.

The child is the smallest and takes up only one grid cell (an area of one foot by one foot), the adult is the next smallest and takes up two grid cells (an area of one foot by two feet), the adult with backpack is slightly larger and takes up four cells (an area of two feet by two feet), and finally, the adult with bicycle is the largest and takes up fourteen cells (an area of one foot by two feet for the adult directly adjacent to an area of six feet by two feet for the bike). One limitation to note about the adult is that it is currently only implemented in the horizontal direction, meaning that the two cells making up the adult are always next to each other horizontally rather than vertically, although the pedestrian can still move in any of the eight previously mentioned directions. Additionally, we have a similar limitation to the bicycle, as it is also only implemented in one direction, meaning the person is always on the left with the bicycle directly to the right of them, although again, they can still move in any direction. Also on the grid are obstacles which default to taking up one cell (an area of one foot by one foot) and exits which take up four grid cells (an area of one foot by four feet). In order to align with a realistic room layout, exits appear only on the borders of the grid to simulate exits appearing only along the walls of a given enclosed space. See Figure 3.2 for a visualization of what each type of pedestrian looks like. Despite our similar use with CA of a two-dimensional grid, in our case the population of entities exists outside of the grid (on a temp grid) and is projected onto the grid based upon that entity's location as well as its orientation and exit destination.

3.2.3 The Input

One of the features that we knew we wanted our simulation to offer was the ability to specify the initial parameters, as shown in Figure 3.4. These include the initial counts for each type of pedestrian

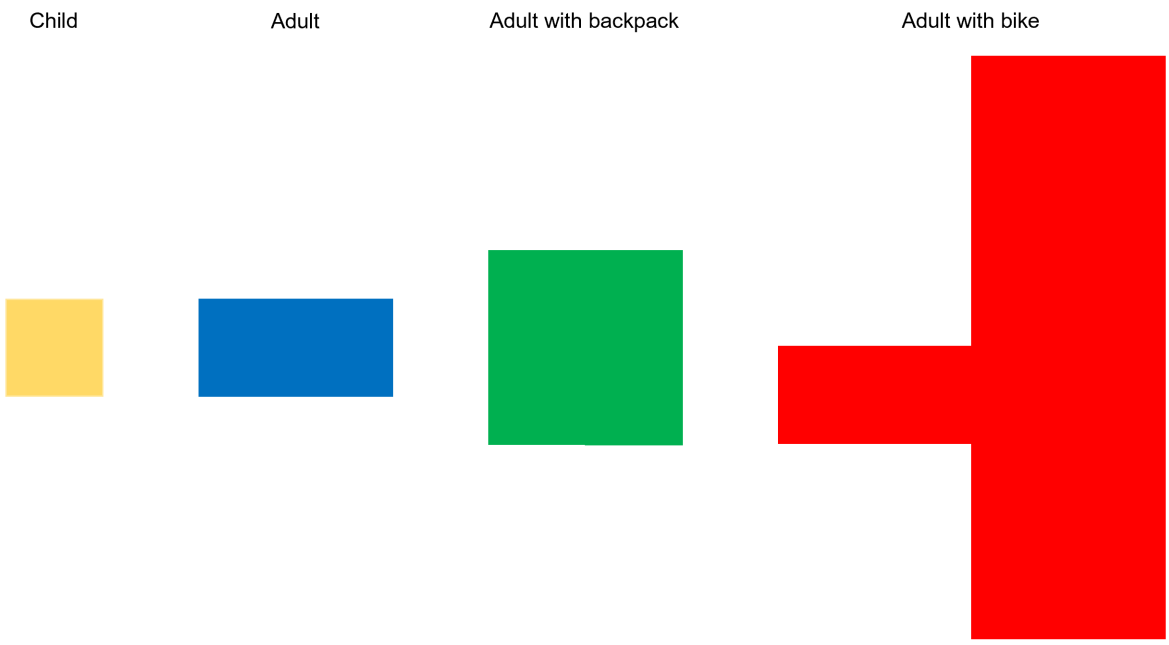


Figure 3.2: Screenshot depicting what each of the four pedestrian types looks like.

(children, adults, adults with backpacks, and adults with bicycles), the amount of time between updates (this can be thought of as the apparent speed of the simulation in the user interface; e.g.: all of the same steps will happen, but a value of 100 will take longer to run than a value of 10, though this amount of time between updates can take on any value based on user input), the time to wait (meaning the amount of time that a pedestrian will wait without movement before seeking an alternative route), the size of the grid (width and height), and the number of exits and obstacles on the grid. See Figure 3.3 below for an example initial configuration of the simulation.

Additionally, our model has several checkbox options that can be selected prior to starting the simulation. One of the features that we knew we wanted our simulation to have was the ability for the user to save or export the simulation run. The "Take Snapshot Images" checkbox allows the user to do just that. When the user selects the "Take Snapshot Images" checkbox, the simulation will take a screenshot of the grid at each iteration and download them to the user's computer. These screenshots can then be easily fed into an online gif creator to generate a gif image of the simulation. We also knew that we wanted the user to be able to design the room that their pedestrians were in, so our interface allows the user to choose an initial layout. The user can select from a generic lecture hall, a model of Fuller Lower at Worcester Polytechnic Institute, and a generic classroom (modeled after a classroom in Stratton Hall at Worcester Polytechnic Institute). In addition to these three pre-loaded layouts that the user has to choose from, they can additionally upload their own layout in the form of a text file. See Figure 4.3 for an example of what this text file should look like.

We also wanted the user to have a say in how the pedestrians move. For this reason, the user has the option to choose the heuristic function (see Section 4.2 for a description of heuristic functions). The GUI additionally gives the user the ability to choose which graphs they would like to appear upon the completion of the simulation. Finally, the user can select the order in which they would like the conflict resolution strategies to be utilized (see the next chapter, Chapter 4 for an explanation of these strategies), along with the threshold value at which each conflict resolution strategy will be utilized.

All of these options that allow user input also have limitations. For instance, the user cannot select

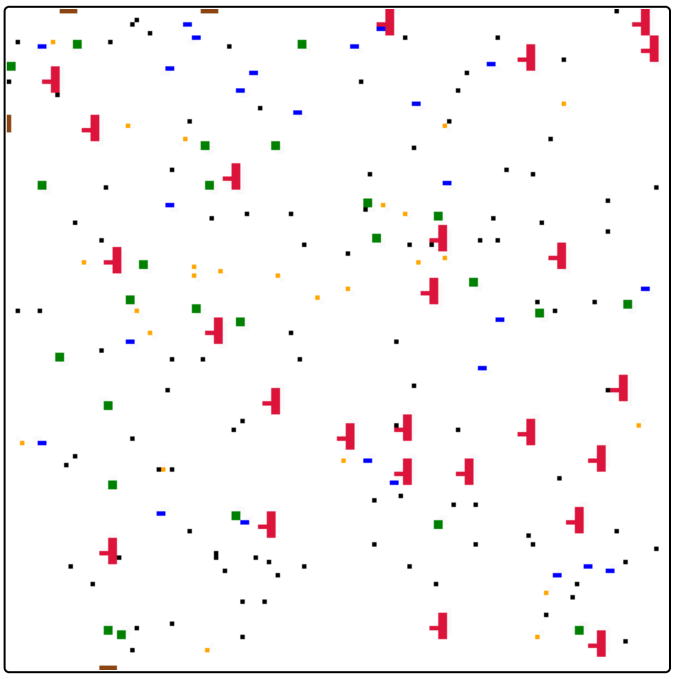


Figure 3.3: Screenshot from our simulation tool depicting a sample initial configuration with four exits (brown), 100 obstacles (black), 25 children (yellow), 25 adults (blue), 25 adults with backpacks (green), and 25 adults with bicycles (red).

more than one heuristic, or more than one layout. The user additionally cannot enter more pedestrians than will fit on the given grid. Additionally, the minimum grid size is set to be 25 by 25, so the interface does not allow input below these values. In the event that the user tries to enter invalid values for any of these, they will receive an error pop-up dialog box telling them to adjust their input. See Figure 3.4 for an example of what this pop-up box looks like.

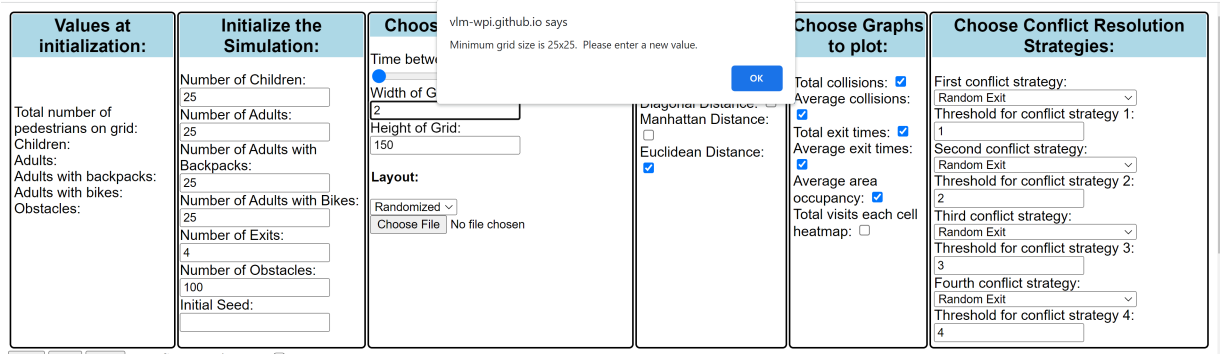


Figure 3.4: Screenshot from our simulation tool depicting a sample dialog box that appears in case of invalid user input. In this case, the user set the grid size to be too small.

3.3 Statistics and Calculations

As discussed previously, we calculate and save several different statistics throughout the run of our simulation. We then use different visualization tools to display these statistics. Firstly, we save the number of children, adults, adults with backpacks, and adults with bikes at each iteration of the

simulation by decrementing a counter by one each time one of them exits the grid. We then convert this value to be a percentage out of 100 for graphing purposes and display this data as a time series plot depicting the percentage of pedestrians (total and of each type) over time. To differentiate between the four different types of pedestrians shown on the plot, we use the color that aligns with the color used in the simulation itself. For example, the yellow line on the graph represents the percentage of children over time, and the green line on the graph represents the percentage of adults with backpacks over time. See Figure 3.5 for an example of this graph.

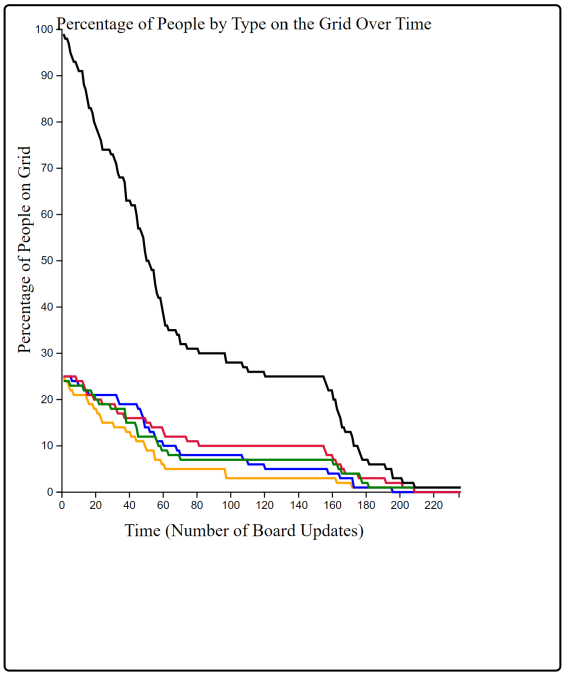


Figure 3.5: Screenshot from our simulation tool depicting a sample graph showing the percentage of pedestrians on the grid over time where the line color corresponds to the pedestrian color on the grid.

We additionally keep track of the exit time for each individual pedestrian, and then use these values to calculate the average exit time for all of that type of pedestrian (children, adults, adults with backpacks, and adults with bicycles). For children, we calculate this by dividing the sum of all of the children's exit times by the total number of children on the grid at initialization, where the exit times are based on the number of iterations that the child was present on the grid before exiting (a counter is also used to keep track of this value). The same process is used to calculate the average exit times for adults, adults with backpacks, and adults with bicycles. In keeping track of this data, we additionally determine the maximum exit time for each type of pedestrian by getting the maximum from the list of exit times. In addition to computing these values for each type of pedestrian, we also do them overall to obtain the total average exit time and the total exit time (from the maximum exit time).

Our third main calculation is of the number of collisions. Similar to the exit time data, we compute and save values for the average and total number of collisions overall, as well as for each type of pedestrian. We use a counter that is incremented with each collision, and then divide the number of collisions of each type by the respective number of pedestrians of that type to get the average.

An additional calculation that we do is for the local area occupancy. The local area occupancy is representative of the localized density around each pedestrian, and it's calculation is slightly more complicated than those we've presented so far. We calculate the occupancy for each pedestrian at each time step. In order to do so, we begin by determining the pedestrian's direction of motion (up, down,

left, right, diagonally up and left, diagonally up and right, diagonally down and left, or diagonally down and right). We then project a seven by seven grid in the given pedestrian's direction of motion based on the location of their anchor point. If the pedestrian is moving to the left, right, up, or down, the pedestrian is positioned at the back center of the square in their respective direction of motion. If the pedestrian is moving in a diagonal direction, they are positioned at the vertex of the square, again in their respective direction of motion. See Figure 3.6 for a depiction of the different square projections. Additionally, Figure 3.7 shows an example of a pedestrian (gray) who is moving upwards and whose local area occupancy at this particular time step is 23, as there are 23 unoccupied (white) cells in their seven by seven projected square. We chose a seven by seven grid based on Fruin's LoS because the "best" case for Fruin's LoS is level A in which each pedestrian has 35 sq. ft. or more to themselves to move about freely. As such, we knew that we wanted our square projection to be larger than this 35 sq. ft., and a six by six square grid projection would have still been relatively small (at only 36 sq. ft.), therefore directing our choice of a seven by seven projection. It is of note that this seven by seven projection size does not change based on grid size. Once the projected square is aligned correctly, the code loops through each cell within the seven by seven area, checking if it is occupied, and if so, by what (child, adult, adult with backpack, adult with bicycle, or none of these; note that obstacles are not counted). Counts are kept for both of these values and are eventually used to calculate the local area occupancy for each pedestrian at each time step by dividing the number of unoccupied cells by the number of pedestrians. Since only the anchor points will be accounted for by using this counting method, we then multiply the count for each type of pedestrian in the seven by seven projection by the number of cells that it takes up. For example, if a count revealed that there were 2 bikes, 2 adults with backpacks, and 3 adults, this would be calculated as $(2 \times 14) + (2 \times 4) + (3 \times 2)$. Additionally, when close to a boundary (wall), we count the cells outside of the simulation as their closest cell inside of the simulation (directly above, below, left, or right). The occupancies are then saved in four ways:

1. As a separate list for each individual pedestrian with their area occupancy at each time step of the simulation.

2. As four separate lists (one for each type of pedestrian) with the average area occupancy for all pedestrians of that type at each iteration of the simulation.

3. As an overall list for the total average occupancy across the entire grid at each generation of the simulation.

4. As a single calculation for the overall average area occupancy for all pedestrians across the entire grid and throughout the entire simulation.

With this single calculation for the overall average area occupancy, it can additionally be used to identify the average LoS according to Fruin [12].

We also perform a calculation related to the density over different regions of the grid. As our code and simulation progressed, we knew that we wanted a way to represent the density near each exit. For that reason, we decided to compute the density about each exit. To do so, we simulate the use of a Voronoi diagram with the exits as the reference points [5]. In doing this, the Voronoi diagram splits the grid into the same number of regions as there are exits in which each region contains all points closest to its corresponding exit. To calculate this, we adapted an existing code base that simulates the construction of a Voronoi diagram from [43]. See Figure 3.8a for a general example of a Voronoi diagram and see Figure 3.8b for an example of what the Voronoi diagram looks like in our simulation [5]. In Figure 3.8a, each of the black dots represents one of the points of interest that are used to construct the diagram. With the given black dots, the diagram is then divided into regions of the

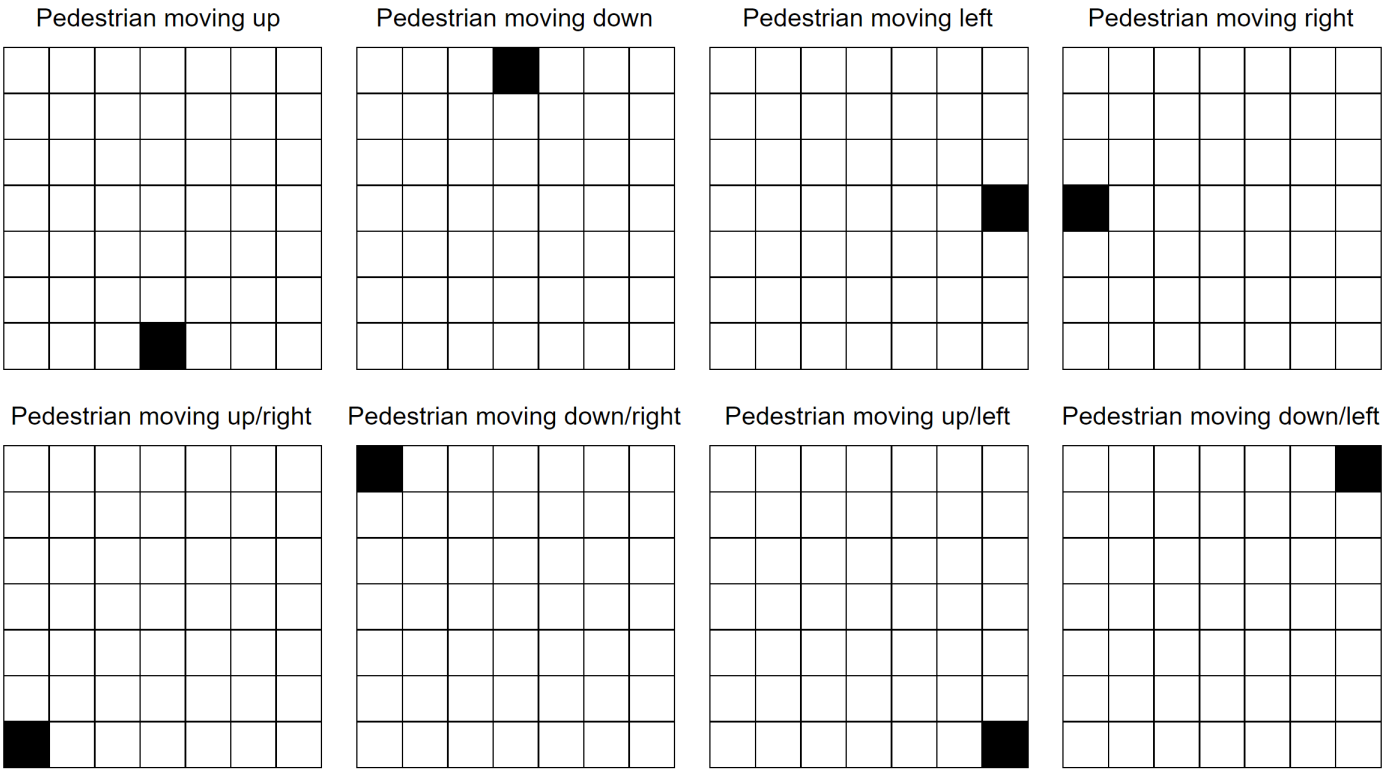


Figure 3.6: Depiction showing the set up for the square projections for the area occupancy in the eight possible directions of motion where the black cell represents the anchor point of the given pedestrian.

domain that are closest to each of the points, and then filled with color to create the Voronoi diagram. After constructing our own Voronoi diagram using the exits as these points, we then calculate the density within each of these regions by dividing the number of pedestrian-occupied cells by the number of total cells in the region. The density of each region is then used in one of the conflict resolution strategies, discussed in Section 4.3.

The final two calculations that we perform are closely related, and both involve keeping track of how often part of the grid is visited. First, in keeping track of each pedestrian's path over the course of the simulation, we can then go through these paths to find out how many times each individual cell is visited, including which cell is visited the most. From this path list for each pedestrian, we can also get the exit that each person used, and keep track of how many pedestrians went through each exit to leave the grid. We then save both of these sets of values.

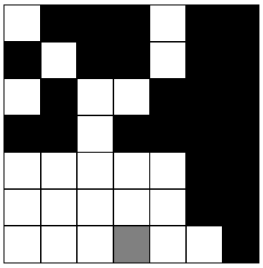
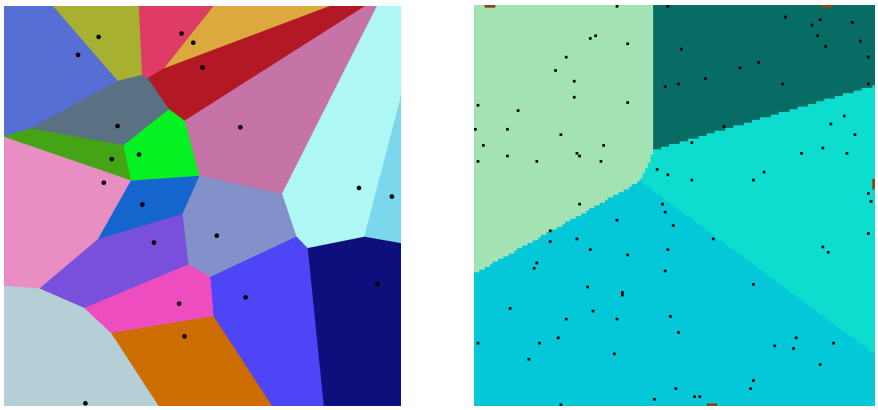


Figure 3.7: Depiction showing an example of when the local area occupancy for a given pedestrian (gray) is 23 and other pedestrians are shown as black cells.



(a) General Voronoi diagram **(b)** Our Voronoi diagram Distance

Figure 3.8: Figure 3.8a shows a Voronoi diagram where each color corresponds to a different region, constructed such that all points within a colored region are closer to the black dot in their region than any other point in the domain [43] and Figure 3.8b shows a Voronoi diagram from our simulation where each colored region corresponds to a region where all points have the closest distance to the specified exit (depicted in brown) and obstacles are shown in black.

Purpose	**Computation**	**Computation storage**
Global density of each region	$\frac{N_e^t}{V_e}$ where e is the specified exit corresponding to the Voronoi region, N_e^t is the number of pedestrians in the given region, and V_e is the area of the Voronoi region	Single value for each exit

Table 3.1: Table showing the computations we perform at the start of the simulation, in addition to how the values are saved (named according to the purpose column).

With all of these calculations computed and saved, we decided to use some of them in comparisons between different runs of the simulation. In particular, we use the exit data, collision data, and area occupancy data. See Section 4.4 for more information about between-run comparisons.

Additionally, we have several computations that take place, as shown in Table 3.1 (showing the calculations that take place at the start of the simulation), Table 3.2 (showing the calculations that take place at each time step of the simulation), and Table 3.3 (showing the calculations that take place at the end of the simulation), some of which have been described previously.

It is of note that "End of simulation" in Table 3.3 refers to the point at which all pedestrians have exited the grid ($N^t = 0$), referred to as t_f. A simulation is considered to have failed if not all pedestrians are able to exit the grid or if the simulation reaches the maximum number (set to 1000) of iterations and there are still pedestrians stuck on the grid. In the case that a simulation run fails, the statistics are still calculated, but they are calculated based only on the pedestrians that have left the grid so far.

Purpose	**Computation**	**Computation storage**
Number of pedestrians of each type on grid	$N^t_{PT} = \sum_i N^t_{PT_i}$	Four single values
Total number of pedestrians on grid	$N^t = \sum_{PT}(N^t_{PT})$	Single value
Exit time for each pedestrian	X_{PT_i}	List of values
Area occupancy for each pedestrian	$O^t_{PT_i} = \frac{WS(N^t_{PT})}{S_{OA}}$ where $WS(N^t_{PT})$ is the weighted number of pedestrians in the square, calculated by summing the number of each type of pedestrian multiplied by the number of cells that a pedestrian of that type occupies in general (even if they only occupy one or two cells inside the square), and S_{OA} is the number of open cells in the 7x7 square projection	List of values
Average area occupancy for each type of pedestrian	$O^t_{PT} = \frac{\sum_i (O^t_{PT_i})}{N_{PT_i}}$	Four separate lists
Average area occupancy overall	$O^t = \frac{\sum_{PT}(O^t_{PT})}{4}$	List of values

Table 3.2: Table showing the computations we perform at each time step of the simulation, in addition to how the values are saved (named according to the purpose column).

Purpose	**Computation**	**Computation storage**
Exit time for each pedestrian	X_{PT_i}	List of values
Total exit time for each type of pedestrian	$max_{i \in PT}(X_{PT_i})$	Four single values
Total exit time overall	$max_{PT}(max(X_{PT_i}))$	Single value
Average exit time for each type of pedestrian	$\frac{\sum_i (X_{PT_i})}{N_{PT}}$	Four single values
Average exit time overall	$\frac{\sum_{PT}(\sum_i (X))}{N^0}$	Single value
Total number of collisions for each type of pedestrian	$\sum_i (C_{PT_i})$	Four single values
Total number of collisions overall	$\sum_{PT}(\sum_i (C_{PT_i}))$	Single value
Average number of collisions for each type of pedestrian	$\frac{\sum_i (C_{PT_i})}{N_{PT}}$	Four single values
Average number of collisions overall	$\frac{\sum_{PT}(\sum_i (C_{PT}))}{N^0}$	Single value
Total average area occupancy overall	$\frac{\sum_{t=1}^{t_f} (O^t)}{t_f}$, where t_f is final time-point	Single value

Table 3.3: Table showing the computations we perform at the end of the simulation, in addition to how the values are saved (named according to the purpose column).

Chapter 4

Methods

4.1 Background Research

Throughout the entirety of our research project, we conducted continuous background research. At the beginning of the project, we began by researching Conway's Game of Life, as that was our initial research interest. As we dove deeper into the topic, we found papers discussing other applications for the underlying structure of Conway's Game of Life, which is rule-based modeling on a two-dimensional grid. Eventually, this led us to the topic of jamology and pedestrian flow. Throughout the research process, we took notes on the papers and topics of interest that we found, as well as keeping track of our sources. Additionally throughout the research process, we routinely used several of the same or similar search terms to obtain our results. Some of these included cellular automata, rule-based modeling, pedestrian movement, modeling pedestrian flow, pedestrian evacuation, and crowd management in addition to others and combinations of these.

Once we knew the goal of our research project, we began looking into how to code it. This portion of our background research led us to the existing modeling tools discussed in Section 3.1. As discussed in that section, we started with the code from Javascript Ants from Sim4edu [1]. In order to adapt the code to our needs and to have the features we desired, we quickly began making modifications to the code, particularly with respect to the search algorithm involved.

4.2 Search Algorithm: A*

As mentioned in Chapter 3, the primary purpose of the GUI we designed is to model pedestrian movement. In order for pedestrians to move within our model, they require a goal destination that they are trying to reach. We have chosen exits as the desired destination for our model. When a pedestrian is in a room, their goal destination is the exit that they are closest to upon initialization of the simulation. One feature of our model is that each pedestrian is able to calculate the shortest distance to the closest exit in a room, including accounting for the presence of obstacles, and move along that path to the desired destination. The algorithm behind this ability is the A* algorithm. A* is a search algorithm that uses a heuristic function to find the shortest possible path from the pedestrian's current position to the exit. In our model, there are three main ways that we calculate the "shortest" path with the default heuristic function making use of euclidean distance. In addition to euclidean distance, there are options for manhattan distance and diagonal distance to be used in the calculation of the heuristic function. See below for formulas for the calculation of these three types of distance. For each of the three heuristics, we calculate the distance from the anchor point of the given pedestrian

to the goal cell of the exit and choose the minimum value over all exits.

Diagonal Distance:

$$D = |x - X_E| + |y - Y_E| + ((\sqrt{2} - 2) * min(|x - X_E|, |y - Y_E|)) \tag{4.1}$$

Manhattan Distance:

$$D = |x - X_E| + |y - Y_E| \tag{4.2}$$

Euclidean Distance:

$$D = \sqrt{(x - X_E)^2 + (y - Y_E)^2} \tag{4.3}$$

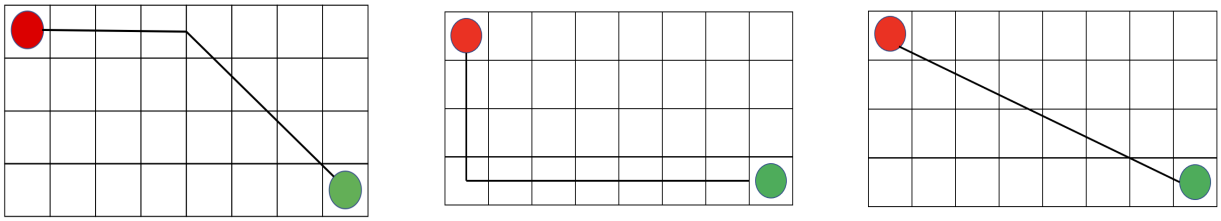
where x and y refer to the x and y coordinates of the anchor point of the pedestrian, and X_E and Y_E correspond to the x and y coordinates of the exit, respectively.

The effectiveness of A* depends on how well this heuristic function is chosen. A heuristic function is a calculation that is used to estimate the cost of a potential next move. The A* algorithm finds the path with the lowest overall heuristic value, and therefore the lowest cost path, and chooses the first move in that path to make as its next move. The heuristic that A* uses is a calculation of both the cost from the original starting spot to the potential next move, called g, and the estimated cost from the potential next move to the goal, called h. After each move calculation, the pedestrian keeps track of the cost from its previous spot to its next move and adds this value to g, the cost from the original starting spot to the current one. Therefore, our g value continuously updates when investigating a given potential path. The other value that is calculated, h, can be done in many ways and again, it is the estimated cost from the potential next move to the goal.

Our simulation allows the user to choose either the diagonal distance, euclidean distance, or the manhattan distance method when calculating the heuristic. These methods calculate the distance between a pedestrian's current position, and their exit goal location. The chosen heuristic is then used uniformly in the calculation for each pedestrian over the course of the simulation. Each of these calculations are similar, although there are important differences to note. The diagonal distance is calculated by taking the maximum of the absolute value of the differences between the current and goal x coordinates and the current and goal y coordinates. This means that one computes the number of steps one would take without moving diagonal, and then subtracting the the steps that are saved by using the diagonal [16]. The manhattan distance is calculated by taking the sum of the absolute value of the differences between the current and goal x coordinates and the current and goal y coordinates. Here, the difference between the diagonal and manhattan distance is taking the maximum versus taking the sum of these differences. The euclidean distance is calculated by taking the difference between the current cell and the goal cell. A visual comparison of these three calculations can be seen in Figure 4.1. One of our hypotheses is that the diagonal distance will provide our simulation with better results, using our evaluation metrics. These results and evaluation metrics are further discussed in Section 5. We believe the diagonal distance will give better results than the manhattan and euclidean distance because the diagonal distance is best for when paths can be made using all eight directions (such as north, south, east, west, northeast, northwest, southeast, southwest) [2].

Once these calculations of g and h are made, the values are added together to get an overall heuristic value for a move, we call this value f. As mentioned before, A*'s success is all based upon how well the

heuristic value is chosen, meaning how well one can estimate the cost of a path. Sometimes it is difficult to estimate the cost from one move to the next, but in our case we have a pretty straightforward task at hand. Since we are working with an $n \times m$ grid, the distance is very easy to estimate because we know the structure of the grid and it does not change over time (except for the movement of other pedestrians, though these problems are addressed with localized rules when an area gets particularly dense).



(a) Diagonal Distance **(b)** Manhattan Distance **(c)** Euclidean Distance

Figure 4.1: Examples of different distance (black line) calculations from the starting position (red circle) to the goal (green circle). Adapted from [2].

A* was chosen because it has the capability to find an optimal path with the chosen heuristic. And, in our case, since we are working on an $n \times m$ grid, there are very effective and simple heuristics available, as previously mentioned. Not only does A* guarantee a cost-optimal solution, it is efficient because it prioritizes paths that have low heuristic values, meaning it prunes away other paths that would not lead to an optimal solution [32]. The pruning capability is done by keeping a queue of the possible paths and their heuristic values. After evaluating a potential move and obtaining its heuristic value, the queue is reordered so that the path with the lowest heuristic value is located at the front of the queue. By doing this, the path that gets explored is always the path with the lowest cost so far. Then, the rest of the queue has the estimated-cost-remaining nodes. In addition, if there are multiple paths that reach the same goal, only the path with the lowest cost-so-far is kept, therefore pruning away paths that will not lead to an optimal solution [31].

There are other search algorithms that are also capable of finding an optimal path, but these are not as efficient as A*. For example, breadth first search is an example of a search algorithm that guarantees finding an optimal path, however its run time is longer than A*. Similarly to A*, breadth first search works by handling a queue as well. It first checks if the node that it is looking at is the destination and if not, it generates all of the potential moves, and then adds these potential moves to the end of the queue. This queue never gets ordered, because there is no way of numerically evaluating the cost of the path, since heuristics are not involved. Therefore, breadth first search branches out every single possible path until it reaches the goal. Breadth first search is sometimes referred to as a "blind" search because it always begins by expanding all the nodes at the current level before going to the next level [31]. Since it expands all of the nodes, breadth first search is generally slow. In our pedestrian flow model, we must call a search algorithm on every pedestrian each time the board is updated, so A* is called for each pedestrian at each time step. Therefore, it is important to use an efficient algorithm like A*, since slow algorithms will exponentially decrease our model's efficiency and increase the amount of time that it takes to run.

One thing that our A* algorithm on its own does not take into account when finding a path is the location of the other pedestrians in the room (on the grid). When finding a path, A* picks the shortest path as if they were the only pedestrian in the room with the obstacles. Because of this, collisions may occur in which one pedestrian is trying to access the same cell as another one, or one pedestrian is currently occupying the cell that another pedestrian wishes to move into. When pedestrians get into

these deadlock situations, we then have to resort to local rules and can no longer use A* to address such conflicts. One of these local rules involves counting the number of times a pedestrian has not moved. If a pedestrian has been waiting for a certain amount of updates of the board, the pedestrian makes a move (if it can) in a random direction and then it tries to find a path to the desired exit from there. This "certain amount of updates" is a variable that can be set each time the model runs. By having the pedestrians make a random move, it simulates how people in actual rooms will move out of the way for one another to exit. Therefore, we can evaluate how well our model runs with various settings based, in part, on how many iterations each person waits to move out of the way for another pedestrian on average. Another collision strategy we have implemented is having a pedestrian find a path to another random exit if it is waiting, or not moving, for more than a certain amount of board updates. A user has the option of choosing these different collision resolution strategies along with the threshold each pedestrian has to be waiting in order to resort to the specified resolution strategy. Other collision strategies that are implemented are choosing to go to the exit with the lowest density based on the Voronoi diagram, resetting one's wait time after choosing another random exit to leave from, and performing A* while taking other pedestrians into account as obstacles. The way that these local rules are implemented are important, and are further discussed in Section 5. Without the implementation of these local rules at all, the simulation has the potential to stay in a deadlock situation. Deadlock situations are when multiple pedestrians cannot move on the grid. Figure 4.2 is an example of a deadlock situation. As you can see, the blue cell cannot move because the yellow cell is blocking it and the yellow cell cannot move because the blue cell is blocking it. Because these two pedestrians are blocking each other, none can move. These local rules combined with the A* search algorithm allow our user interface to model pedestrian flow.

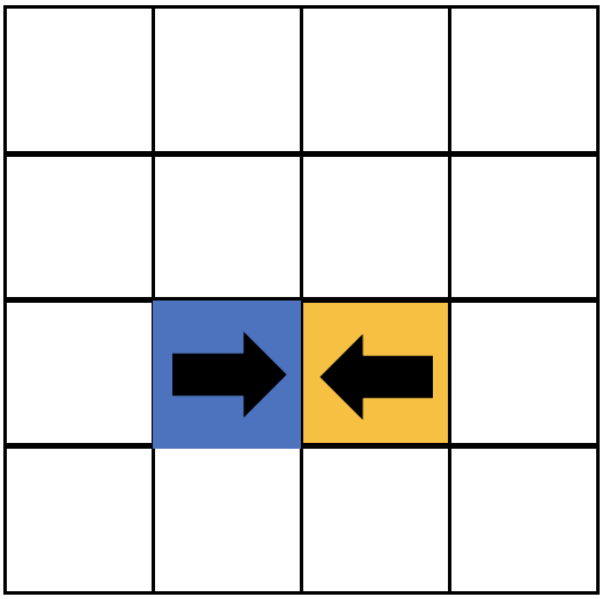


Figure 4.2: Example of a deadlock situation. The blue cell cannot move because the yellow cell is blocking it and the yellow cell is cannot move because the blue cell is blocking it.

As just mentioned, we also have a variation of A* as one of our local rules. If a person cannot find a path to the exit, we then take into account the locations of the other people and run A* to try to find a path to the exit. However, one weakness of the A* algorithm is that if a path is not found, the algorithm

fails. This means that all of the possible paths were searched, but no path was found and the algorithm returns nothing. In this case, the algorithm is not helpful, and will actually slow the simulation down. This is because every single path is explored before deciding there is no path, instead of stopping when a path is found (which most often does not result in the worst case scenario of exploring all paths). This may happen in the simulation when the local rule of running the A* algorithm and taking into account the location of the other pedestrians in the room (on the grid) is implemented in an effort to find a path. This is because if there are a lot of pedestrians on the grid all moving toward an exit, the A* algorithm may not find a path from a pedestrian's current position to its goal exit. This is due to the fact that the algorithm sees pedestrians as places on the grid that others cannot move to. So, the A* algorithm does not allow a pedestrian to move to another pedestrian's location, even though a pedestrian might have moved away from that cell by the time another pedestrian reaches that location. However, with fewer pedestrians on the grid, when two people are in deadlock, using this collision resolution strategy is helpful since these pedestrians in deadlock have a chance to find a path around the other one. We evaluated these collision resolution strategies and compared them to each other in order to decide which strategies are more useful with our simulation. These evaluations are further discussed in Section 5.

4.3 The Code and the Modules within it

When creating and developing our simulation, we knew that we wanted our code to be understandable and usable to someone outside of our research group. Ultimately, we wanted someone outside of our team to have the ability to pick up on our project and make any edits or changes as they see fit without much difficulty. The primary way that we decided to address this is by making our code publicly available online as a GitHub repository at https://github.com/vlm-wpi/MQP and hosting the GUI simulation online at https://vlm-wpi.github.io/MQP/app/final.html for easy user access. We also developed a ReadMe file explaining how to run our code and aimed to keep our code well-documented and simple to follow.

We additionally decided to tackle this problem by creating separate modules for sections of the code. Each module has a separate JavaScript file that contains the necessary information to perform the required tasks. We have a module for A*, the conflict resolution strategies, the data, the debugging mode, the graphs, the GUI (graphical user interface), the population, the heap structure, the layouts, the metrics, headless mode, random numbers, the voronoi computation, and the final resulting simulation. By separating our code and categorizing it, one can understand the different parts to our code more easily. In addition, each module has the capability of using variables and functions of other modules. By separating the code into modules, another researcher could also understand which parts of the code rely on one another. When the simulation is actually run, the final.js file is the only file that the html uses to run the code when the simulation is run using the GUI, though this is slightly different in headless mode when the html doesn't get used (only the final.js file is called). This is because in the final file, all of the parts are placed together so that the simulation can run.

The heap structures are contained inside of the "heap" module. These are the structures that have the capability to keep a min heap, so that when the A* algorithm is run, the minimum path is always at the beginning of the heap. Going off of this, the "astar" module depends on the heap structures and contains all of the information regrading the heuristic A* algorithm used to find the shortest path for each pedestrian. Our "gui" module takes care of all of the data that we rely on user input for. GUI elements are only allowed to be used in a browser, so this module is not used when running the simulation on the command line, as further discussed in this section. Inside of the "data" module, all of

the global variables are defined, and these variables are used in the calculation of the output statistics. In the "layout" module, we have a structure that defines the exits and obstacles, along with the layout for the grid. Depending on which layout the user chooses, the layout module contains the structure for the specified layout. Originally, the code for our layouts was structured with many for loops and (x, y) coordinates. One of the advances in our code was to give users the flexibility of inputting their own layout. Our simulation allows the user of inputting their own text file layout. The reasoning behind this is that it allows users to pass in a file to the simulation and the simulation generates a layout matching the desired layout. This allows anyone to easily create their own room initialization, and was one of our other simulation goals, as further discussed. The module that defines all of our pedestrians is "pop". Here, "pop" creates an array of all of the different types of people so that it can be used in the final module. Our "graph" module handles evaluating the statistics and visualizing them in the desired way. There are some real-time statistics (updated during each iteration of the simulation), such as the current number of each type of person on the grid, and final statistics (computed only at the end of the simulation), like the average exit time. The "conflict" module contains the computations for the five different possible conflict resolution strategies that can be used. The "voronoi" module contains the calculation of the distinct Voronoi regions (cells) based on the number of and locations of the exits on the grid. This module additionally calculates the density within each of these regions. The "metrics" module contains information for the calculation of the diagonal, manhattan, and euclidean distances. The "final" module brings all of the other modules together and is where all of the functions that actually perform the board updates are called. Here, all of the people, decisions about where the people will move, and evaluation of the simulation run are brought together.

4.4 Evaluating Each Simulation

Another goal of our project was to be able to compare simulations and give evaluations and recommendations based off of the data our simulation produced. In order to do this accurately, we needed our simulation to run many times and to be able to keep track of the data. So, we created a feature called headless mode in which we can run our simulation using only a terminal or command prompt. The module "nodeApp" is used to run the simulation in headless mode. When using headless mode, nothing actually gets created in a browser window, but the simulation still runs and produces statistics, giving us accurate data to evaluate the simulations with. Headless mode is made possible by using Node.js. "Node.js is a run-time environment which includes everything you need to execute a program written in JavaScript. It's used for running scripts on the server to render content before it is delivered to a web browser" [17]. With this capability, our data is not altered by outliers, and we have the capability to see trends and test different layouts, numbers of people, types of people, heuristics and combinations of heuristics, conflict resolution strategies, and more. In addition to the "nodeApp" module, "random" is used to calculate all of the random numbers we call throughout the simulation, and it also creates a random seed for every trial we run in headless mode. We implemented the seed capability so that we could recreate any run we wanted to, by calling the seed and using the same initializations. Finally, we have a "debug" module which is used to debug the simulation by allowing for console messages to appear on the console.

As mentioned before, we also needed to create a metric to compare simulations to each other. When creating this metric, we thought about the important things that people consider when leaving a room. Certainly, most people would like to leave the room as fast as possible, so we include the average exit time for everyone to leave in this metric. Not only do people want to leave the room fast, they also want to keep moving and not keep bumping into others when trying to leave a room. Therefore, we also

use the average number of collisions over the entire simulation in the equation. Finally, we decided that most people do not walk towards an exit that is very dense; instead, they will go towards an emptier exit, even if it is a little further away than the closest exit. So, we use the average area occupancy during the simulation. The final evaluation equation thus becomes

$$Evaluation = \frac{\sum_{PT}(\sum_{i}(X))}{N^0} + \frac{\sum_{PT}(\sum_{i}(C_{PT}))}{N^0} - \frac{\sum_{t=1}^{t_f}(O^t)}{t_f} \tag{4.4}$$

4.5 Adaptability of the Simulation

We also accomplished creating a code base that would be straightforward if someone wanted to change the simulation or make any feature additions to it. For example, we only establish four types of people: a child, an adult, an adult with a backpack, and an adult with a bike. However, it is not unlikely that someone would want to use the simulation with the addition of a new type of person, such as a pedestrian taking up five grid cells. The modules feature helps with this. If someone wants to add a different type of person, all they have to do is go into the pop.js file and add onto the features of that new person they want to create. Once this is added here, all the other files will recognize the new person and still be able to have the same capabilities as all the other people. This part of the code was changed from when we originally created this and had a bunch of if statements, checking the type of each person, to having much more simple for loops. Since every operation is almost the same for every type of person, we condensed the code so that you go through a list of types of people and perform the same operations on them.

Similarly, with the layouts, one can create their own layout with their own decided size and placement and numbers of obstacles and exits. The layouts were originally created by many tedious for loops needed to place everything. However, there is an option for someone to add their own layout by uploading a text file to the simulation using the choose file button. The file should have the "v" symbol to represent a vertical exit, the "h" symbol to represent a horizontal exit, the "." symbol to represent empty cells, and the "o" symbol to represent obstacles. Note that for the exits, the "h" or "v" represents the anchor point of the given exit. Additionally, the grid size for the board is determined by the number of characters in each row and the number of columns. An example layout text file can be seen in Figure 4.3.

4.6 Visualizing the Simulation Using Graphs

Another part of our simulation is the statistics that we produce. As mentioned above, there are a lot of statistics we want to show. In order to do this, we use the visualization tool D3 which includes a library of code that makes it capable of producing graphs and real-time statistics on the same page as our simulation [4]. One of these real-time statistics is the percentage of people on the grid over time. This is visualized as a line graph and each line represents each type of person, except the black line, which represents the total percentage of pedestrians on the grid. The x-axis is time, and is in the units of board updates. The y-axis is the percentage of people. Here, each line is a different color and corresponds to the color that the type of person represents on the grid of the simulation. As time increases, the x-axis adjusts and adds time so that all of the data stays on the graph. See Section 3.3 for an example of what this graph may look like after a run of the simulation.

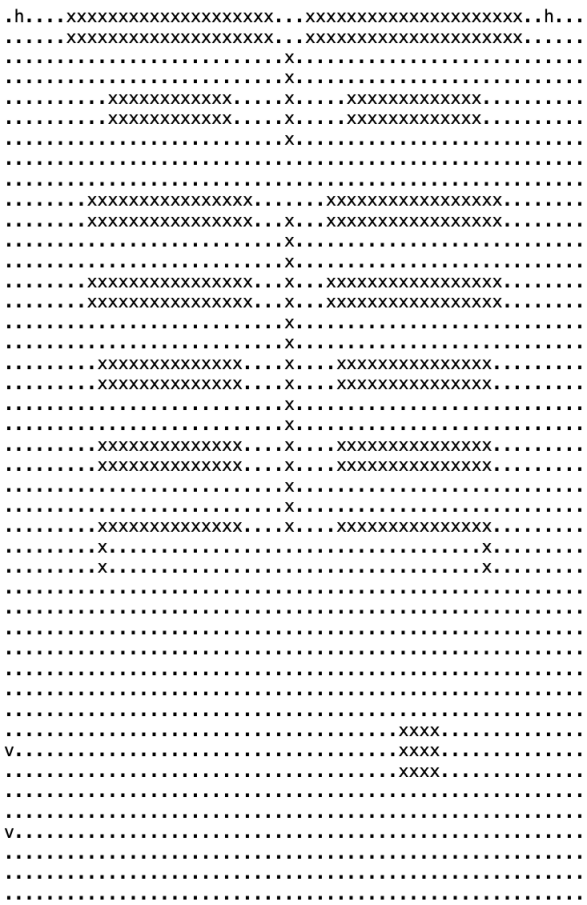


Figure 4.3: Example of a text file to be uploaded as a layout for the simulation.

We also have statistics that are calculated once the simulation has finished. These statistics are shown using bar graphs and summarized in Table 3.3. We have a graph for the average number of collisions for each type of pedestrian. Here the bars are color coded based on the color of the pedestrians in the simulation. Additionally, this graph contains a mark for the minimum and maximum number of collisions for each person, and one standard deviation from the mean for each bar. We also have the same type of graph for the average exit time, as shown in Figure 4.4. Similarly, there are graphs for the exit times and area occupancy's for each type of pedestrian. Like the time series plot, if you scroll your mouse over a bar, a box pops up that shows the exact value for that bar.

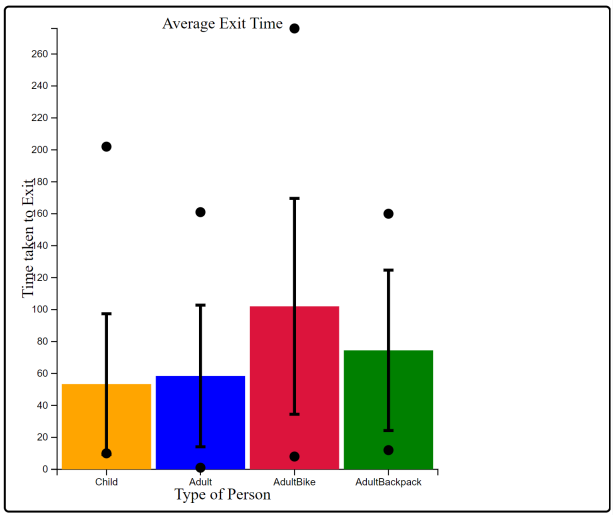


Figure 4.4: Screenshot from our simulation tool depicting a sample graph showing the average number of collisions for each type of pedestrian.

Chapter 5

Results and Evaluation

5.1 Setting Up for Evaluation

As discussed earlier, when running the simulation in headless mode, we allowed for several user inputs. While many of these inputs related back to the the same input options that the GUI supports, we additionally allowed the user to indicate which of the statistics they would like to print to an output file at the conclusion of the simulation trial(s), as well as what they would like to name that output file. Thus, our first step when running in headless mode in an effort to make comparisons between trials was to determine what information the user wanted to save for each trial run. For our purposes, we decided that we wanted to save the following:

- All values provided as initial parameters, including:
  - Seed number
  - Size of the grid (width and height)
  - Count for each type of pedestrian upon initialization
  - Number of exits and obstacles
  - Heuristic that was used (euclidean, diagonal, or manhattan)
  - Layout (randomized or one of the pre-loaded ones)
  - Up to four conflict resolution strategies (each with thresholds)
  - Name of output file
- Total exit time overall
- Average exit time overall
- Average number of collisions overall
- Total average area occupancy overall
- Evaluation metric, given in Equation 4.4
- Boolean indicating whether or not a deadlock occurred

Note that these calculated values are detailed in Tables 3.1, 3.2, and 3.3.

After determining which statistics we wanted to print to an output text file at the conclusion of the set of trials running, we developed a python script to write the information from this text file to a csv file. This helped make the post-processing and analysis described in the following paragraphs significantly easier.

5.1.1 Determining the Number of Trials

In addition to figuring out what values to store, we also needed to determine how many trials were sufficient for a given room initialization, noting that, when not specified by a layout file, pedestrian, obstacle, and exit locations are random. In the case of exits, this does mean that it is possible for all exits to be placed along the same wall or for them to be evenly dispersed with one along each wall, emphasizing that the initial distance to each exit from a given pedestrian varies even in the same size room and with the same number of obstacles. Since the main goal of headless mode was to run large quantities of trials of the same initializations to be able to make comparisons between different room configurations, we knew that just one trial per initialization wouldn't be sufficient. As such, we developed a python script, titled "k-fold" that makes use of the ideas behind k-fold cross validation, an algorithm often used in machine learning, as well as the pandas and numpy packages, and runs on the csv file containing the information from the output file. For a given list of sets of trials, $[[a_1, b_1], [a_2, b_2]...]$, and a given k, the script loops through each set of trials, splitting the individual set of trials $[a_i, b_i]$ into k random groups 100 times. Then, for each of the k groups, it calculates the average evaluation metric for all trials within the $k-th$ group, and all trials still within the set $[a_i, b_i]$, but outside of the $k-th$ group [6]. The "k-fold" script then proceeds to calculate the percent difference between these two averages, counting it as a "good" set of trials if the percent difference is less than or equal to 5%, and counting it as a "bad" trial otherwise. Finally, we use the python script to calculate the total percentage of "good" trials, and consider the number of trials to be sufficient if this percentage is 95% or greater. We tried this with $k = 3, k = 5$, and $k = 10$ for several different initializations, with the results shown in Table 5.1 below. From this information, we were able to conclude that 3500 trials is sufficient, and we proceeded with running 3500 trials for each of our chosen initializations, discussed later. We believe that these 3500 trials are sufficient because they afford a large enough sample size to allow us to be confident in our results for the average values that are being used in the calculation of our evaluation metric.

Run Description	**Number of Trials**	$k = 3$	$k = 5$	$k = 10$
Size: 50x50, 10 children, 10 adults, 10 adults with backpacks, 0 adults with bicycles, 0 obstacles, 1 exit	100	27%	22%	20%
Size: 50x50, 10 children, 10 adults, 10 adults with backpacks, 0 adults with bicycles, 0 obstacles, 1 exit	1000	80%	70%	57%
Size: 50x50, 10 children, 10 adults, 10 adults with backpacks, 0 adults with bicycles, 0 obstacles, 1 exit	1500	86%	81%	66%
Size: 50x50, 10 children, 10 adults, 10 adults with backpacks, 0 adults with bicycles, 0 obstacles, 1 exit	2500	97%	89%	80%

Size: 50x50, 10 children, 10 adults, 10 adults with backpacks, 0 adults with bicycles, 0 obstacles, 1 exit	3500	98%	98%	85%
Size: 150x150, 25 children, 25 adults, 25 adults with backpacks, 15 adults with bicycles, 0 obstacles, 4 exit	3500	99.67%	98%	96%

Table 5.1: Table showing the percentages of good trials calculated using k-fold for a varying number of trials, leading us to conclude that 3500 should be sufficient for each initialization.

5.1.2 Determining the Trials

Once we knew that 3500 would provide adequate data, the next step became determining what initial room configurations we wanted to run the simulation on, presented in Table 5.2.

Run Description	**Grid Size or Layout**	**Num Exits**	**Num Obstacles**	**Heuristic Function**	**Conflict Resolution and Thresholds**	**Num Pedestrians**
3. Conflict Resolution Strategy	100 × 100	4	100	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 10	-25 children -25 adults -25 adults with backpacks -10 adults with bicycles
4. Conflict Resolution Strategy	100 × 100	4	100	euclidean	1. Random move - 3 2. Random exit - 6	-25 children -25 adults -25 adults with backpacks -10 adults with bicycles
5. Number of People	Classroom	N/A	N/A	euclidean	1. Random move - 3 2. Random move - 6 3. Random move - 9 4. Random move - 12	-0 children -0 adults -5 to 50 adults with backpacks in increments of 5 -0 adults with bicycles
6. Number of Obstacles	100 × 100	4	0-500 in increments of 50	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-25 children -25 adults -25 adults with backpacks -10 adults with bicycles

7. Number of Exits	100 × 100	1-8	100	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-25 children -25 adults -25 adults with backpacks -10 adults with bicycles
8. Conflict Thresholds	100 × 100	4	100	euclidean	1. Random move - 2-20 in increments of 2	-25 children -25 adults -25 adults with backpacks -10 adults with bicycles
9. Heuristics	100 × 100	4	100	euclidean	1. Random move - 3 2. Take others into account - 6 3. Random Exit - 9	-25 children -25 adults -25 adults with backpacks -15 adults with bicycles
10. Heuristics	100 × 100	4	100	diagonal	1. Random move - 3 2. Take others into account - 6 3. Random Exit - 9	-25 children -25 adults -25 adults with backpacks -15 adults with bicycles
11. Heuristics	100 × 100	4	100	manhattan	1. Random move - 3 2. Take others into account - 6 3. Random Exit - 9	-25 children -25 adults -25 adults with backpacks -15 adults with bicycles
12. No Conflict Resolution	100 × 100	4	100	euclidean	N/A	-25 children -25 adults -25 adults with backpacks -15 adults with bicycles
13. Number of People	Fuller Lower	N/A	N/A	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-0 children -0 adults -20-200 adults with backpacks -0 adults with bicycles

14. Number of Bikes	Fuller Lower	N/A	N/A	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-25 children -25 adults -25 adults with backpacks -0-10 adults with bicycles
15-18. Small Grid Size	50 × 50	4	0	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-10, 25, 50, 100 children -10, 25, 50, 100 adults -10, 25, 50, 100 adults with backpacks -0, 15, 40, 75 adults with bicycles
19-22. Medium Grid Size	150 × 150	4	0	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-10, 25, 50, 100 children -10, 25, 50, 100 adults -10, 25, 50, 100 adults with backpacks -0, 15, 40, 75 adults with bicycles
23-26. Large Grid Size	300 × 300	4	0	euclidean	1. Random move - 3 2. Random exit - 6 3. Take others into account - 9	-10, 25, 50, 100 children -10, 25, 50, 100 adults -10, 25, 50, 100 adults with backpacks -0, 15, 40, 75 adults with bicycles

Table 5.2: Table showing the different room initializations that we decided to run our simulation on.

5.1.3 Initial Comparisons Across Trials

As the above trials finished running, we took some post-processing steps prior to our more general across-initialization comparisons. Due to time constraints on our project, the last run, number 26 did not finish running because of the large board size along with the large initial number of people on the board. If we had more time, we would include this run in our evaluation, but we were not able to fully get all of the data from the run. First, we ran our previously mentioned python script to write all of

the data from the output files to csv files so that they would be easier to work with using the pandas, numpy, and matplotlib packages in python [19, 26, 23]. Once we had these csv files, we decided to do a deep dive into some of the data as it finished running to ensure that our outputs made sense and aligned with our expectations, and to begin developing our findings. To do this, we first grouped the runs from the table above, Table 5.2, when necessary based on which ones we wanted to compare:

1. Comparing conflict resolution strategies: 1, 3, 4, 12
2. Comparing heuristic functions: 9, 10, 11
3. Comparing the number of pedestrians on a small grid: 15, 16, 17, 18
4. Comparing the number of pedestrians on a medium grid: 19, 20, 21, 22
5. Comparing the number of pedestrians on a large grid: 23, 24, 25, 26
6. Comparing the grid size for a small number of pedestrians: 15, 19, 23
7. Comparing the grid size for a medium number of pedestrians: 16, 20, 24
8. Comparing the grid size for a large number of pedestrians: 17, 21, 25
9. Comparing the grid size for an extra large number of pedestrians: 18, 22, 26
10. Comparing the number of adults with backpacks in a classroom: 5
11. Comparing the number of obstacles: 6
12. Comparing the number of exits: 7
13. Comparing the value for the conflict threshold: 8
14. Comparing the number of adults with backpacks in Fuller Lower: 13
15. Comparing the number of adults with bicycles in Fuller Lower: 14

As the above runs completed, we started looking into the data and developing python scripts to make comparisons. For each of the above, we compared the total exit times, average exit times, average collisions, average area occupancy, and evaluation metric. Here, we highlight some of the above comparisons.

The first set of runs that we will discuss is for the comparison of the second item in the list above in which we are comparing across the three different heuristic functions. Our team found this comparison to be of interest because of our expectations of what would happen versus what actually did. We hypothesized that the diagonal heuristic would perform best as indicated by our background research and discussed earlier. Instead, the results showed that there was minimal difference between the three heuristics, as seen in the graphs below in Figure 5.1. In particular, the average evaluation metrics were 23.09 for the euclidean heuristic function, 24.05 for diagonal, and 24.10 for manhattan, a range of just over one.

In contrast to this negligible difference between the three heuristic functions, number ten in the list above in which we were comparing a varying number of adults with backpacks in the classroom layout (from five to fifty in increments of five) revealed a relationship between the number of adults with backpacks and each of the statistics mentioned previously. Figure 5.2 shows these relationships.

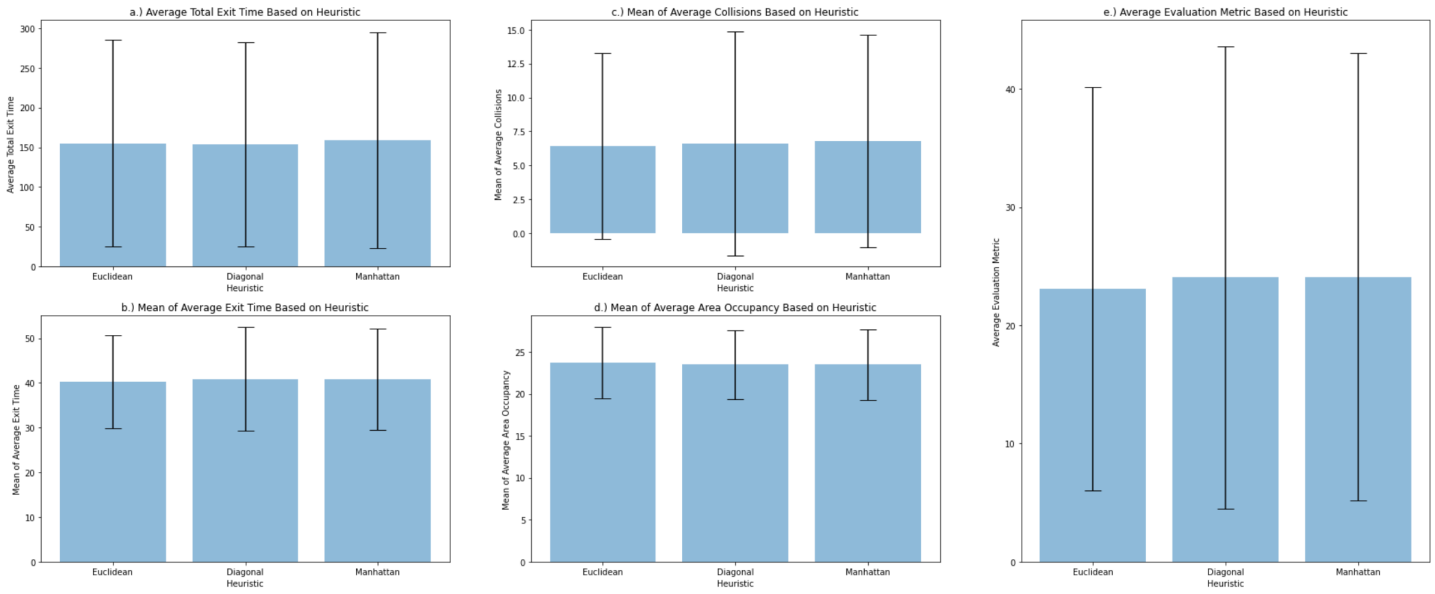


Figure 5.1: Graphs comparing the average of five statistics across 3500 trials for each of the three heuristic functions with all other parameters held constant. Graph a.) shows the average total exit time, graph b.) shows the mean of the average exit time, graph c.) shows the mean of the average number of collisions, graph d.) shows the mean of the average area occupancy, and graph e.) shows the average evaluation metric.

The graphs in this figure reveal that as the number of adults with backpacks in the classroom layout increase, the average exit time, total exit time, and average number of collisions all increase while the average area occupancy decreases. Subsequently, this causes the evaluation metric to increase as the number of adults with backpacks in the classroom increase with all other parameters kept constant. This result tells us that the fewer adults with backpacks there are in a classroom, the more efficient their exit from their classroom will be, which intuitively makes sense.

Similar to the case of the varying number of adults with backpacks in the classroom, we also found that there is a relationship between the number of exits in an enclosed space and the specified statistics (the comparison for number 12 in the above list). As seen in Figure 5.3, our team determined that as the number of exits in a room increase, the average exit time, total exit time, and average number of collisions all decrease, while the average area occupancy increases. Additionally, the evaluation metric decreases as the number of exits increase while all other parameters are held constant. The comparison between the number of exits and between the number of adults with backpacks, while both revealing a relationship with the given statistics, did show different types of relationships, as the comparison for the number of exits appeared to be more non-linear. Regardless, this relationship again makes sense intuitively, as since pedestrians have more exits to choose from (and thus less crowding around any given exit), it follows that they will be able to leave the space in a more efficient manner.

Additionally, we were interested in the results from running our simulation on the same size grid while increasing the number of pedestrians (as seen in numbers 3, 4, and 5 in the list above), as well as the results from keeping the number of pedestrians the same and increasing the grid size (as seen in numbers 6, 7, 8, and 9 in the list above). We found that, in general, if the grid is kept the same and the number of pedestrians is increased, average exit time, total exit time, and average number of collisions all increase, while the average local area occupancy tends to decrease. This subsequently caused the evaluation metric to increase as the number of pedestrians did, indicating that, for the same size room, as the number of pedestrians increases, exit from that room becomes worse. In the case of increasing

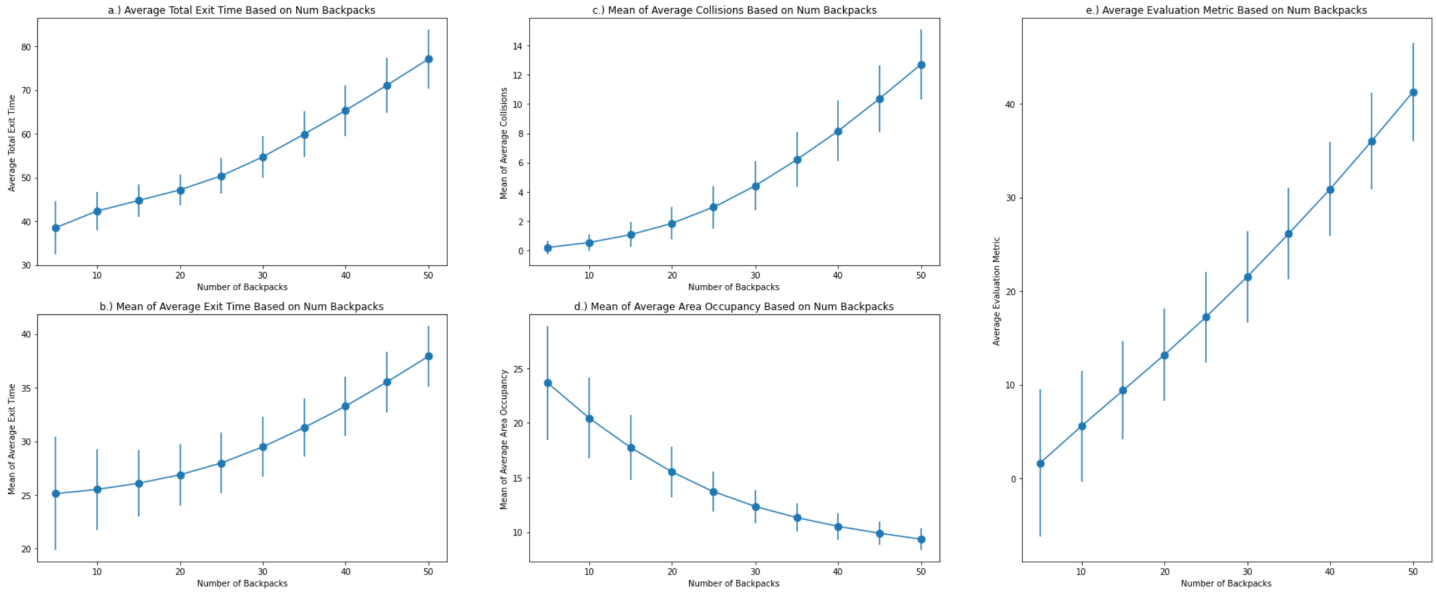


Figure 5.2: Graphs comparing the average of five statistics across 3500 trials for each of the given increments of adults with backpacks with all other parameters held constant. Graph a.) shows the average total exit time, graph b.) shows the mean of the average exit time, graph c.) shows the mean of the average number of collisions, graph d.) shows the mean of the average area occupancy, and graph e.) shows the average evaluation metric.

room size, we found that, with the same number of pedestrians, the average exit time, total exit time, and average local area occupancy all increase, while the average number of collisions tends to decrease. Additionally, the evaluation metric increases as the grid size does. From this we were able to conclude that in general, for the given number of pedestrians and other initial parameters, we expect the exit from a room to become worse as that room gets larger.

5.1.4 Implementing Bayesian Weighted Averages

In addition to the initial comparisons between sets of trials detailed in Section 5.1.3, we knew that we also wanted a more concrete way to make comparisons across all of our different sets of trials. As previously discussed, we developed our evaluation metric, Equation 4.4 in order to do this, but realized that this evaluation metric value would vary widely based on some of our initialization parameters, such as grid size and number of pedestrians. To combat this issue, we decided to incorporate Bayes weighted average in the calculation of a new, standardized, variable that could more easily be used to make accurate comparisons across sets of trials.

In order to perform this calculation, we developed another post-processing python script that reads in the data from a csv file containing the data for all of our runs, does the computation, and then writes the result back to the same file in a new column. Generally, Bayes weighted average is computed using the following equations:

$$S = wR + (1 - w)C \tag{5.1}$$

$$w = \frac{v}{v + m} \tag{5.2}$$

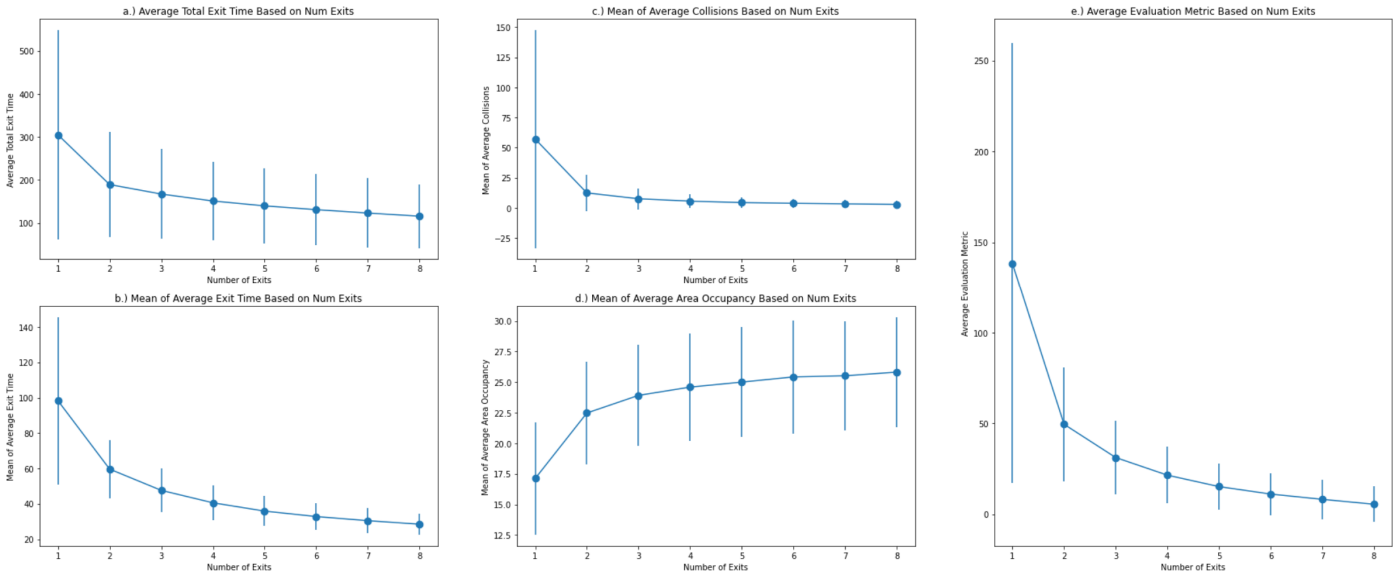


Figure 5.3: Graphs comparing the average of five statistics across 3500 trials for varying numbers of exits (one to eight) with all other parameters held constant. Graph a.) shows the average total exit time, graph b.) shows the mean of the average exit time, graph c.) shows the mean of the average number of collisions, graph d.) shows the mean of the average area occupancy, and graph e.) shows the average evaluation metric.

where S is the overall score (now scaled), w is the weighting, v is number of ratings for the current item, m is average number of ratings for all items, R is the average of ratings for the current item, and C is the average of ratings for all items [41]. Applying this to our own simulation and values, we got the following equation:

$$S = wR + (1 - w)C \tag{5.3}$$

$$w = w_1 w_2 \tag{5.4}$$

$$w_1 = \frac{v_1}{v_1 + m_1} \tag{5.5}$$

$$w_2 = \frac{v_2}{v_2 + m_2} \tag{5.6}$$

where S is a single standardized value for the evaluation metric that we can use to make comparisons across different sets of trials, w is the overall weighting, w_1 is the weighting derived from the number of pedestrian occupied cells on the grid at the start of the simulation, v_1 is the number of cells used by pedestrians in the given set of trials, m_1 is the average number of cells used by pedestrians across all sets of trials, w_2 is the weighting derived from the size of the grid, v_2 is the area of the grid for that

set of trials (calculated by multiplying the grid width by the grid height), m_2 is the average area of the grid across all sets of trials, R is the average evaluation metric for that trial set (sum of all evaluation metrics for that set of trials divided by the number of trials), and C is the average evaluation metric across all sets of trials. Similar to the case of the evaluation metric, we have that a lower Bayes weighted average score is better. The next section, Section 5.1.5 will discuss our results from implementing Bayes weighted average.

5.1.5 Bayes Weighted Average Results

By running the post-processing python script detailed in the previous section, we were able to create an updated csv file containing the information for each of our initializations, and the associated Bayes weighted average for it. We then produced several summary statistics using Excel, including the mean, median, standard deviation, range, minimum, maximum, and skewness of the Bayes averages. These values, as well as our ranges for values falling within one, two, and three standard deviations are given in Table 5.3 below.

Bayes Statistic	**Value**
Mean	34.50
Median	30.53
Standard Deviation	15.66
Range	106.95
Minimum	24.92
Maximum	131.88
Skewness	4.77
Within 1 Standard Deviation (using median)	(14.99,46.06)
Within 2 Standard Deviations (using median)	(-.54,61.60)
Within 3 Standard Deviations (using median)	(-16.07,77.14)

Table 5.3: Table showing the summary statistics calculated for Bayes weighted average scores.

As seen in the table, our data was relatively skewed to the right, so we decided to use the median as our measure of center instead of the mean to combat this [37].

Using Bayes weighted average, we found that the four lowest scores, meaning the four most efficient and effective room initializations, were those with eight, seven, six, and five exits respectively. This indicates to us that the number of exits appears to have the largest impact on the evaluation metric, and therefore, exit from a given room. We additionally found that the fourth worse Bayes score came from the set of runs with only one exit on the grid, further emphasizing that the number of exits in a room has a large impact on the exit from that room. The other three highest Bayes scores, meaning the other three worst exits, came from specific grid sizes and numbers of pedestrians. This indicates that the given numbers of pedestrians in combination with the given grid sizes didn't afford pedestrians enough space to move around freely.

In addition to these global results, we also have results for the comparisons of interest listed in Section 5.1.3. For the first item in that list in which we were interested in comparing across different resolution strategies, we found that they could be ranked in the following order from best to worst:

1. Run 12: No conflict resolution strategies

2. Run 1: Random move, random exit, taking others into account

3. Run 4: Random move, random exit

4. Run 3: Random move, random exit and reset wait time, taking others into account

As mentioned previously, we found a near negligible difference between the trials run with the varying heuristic functions (the second item in the list). The third item in the list was meant to compare between the number of pedestrians on a small grid (50×50) and yielded the best to worst ranking below:

1. Run 15: Small number of pedestrians (10 children, adults, and adults with backpacks, and 0 adults with bicycles)

2. Run 16: Medium number of pedestrians (25 children, adults, and adults with backpacks, and 15 adults with bicycles)

3. Run 17: Large number of pedestrians (50 children, adults, and adults with backpacks, and 40 adults with bicycles)

The sixth item in the list made comparisons across different grid sizes for a small number of pedestrians and had the following best to worst ranking:

1. Run 19: Medium grid size (150×150)

2. Run 15: Small grid size (50×50)

3. Run 23: Large grid size (300×300)

The eleventh item in the list, and the last one that we present the Bayes results for here, compared trials with varying numbers of obstacles. These trials resulted in the following best to worst ranking:

1. 0 obstacles on the grid

2. 50 obstacles on the grid

3. 100 obstacles on the grid

4. 150 obstacles on the grid

5. 200 obstacles on the grid

6. 250 obstacles on the grid

7. 300 obstacles on the grid

Similar rankings and comparisons could also be performed for the other items in the list.

Chapter 6

Conclusion

Originally, this project started out with us looking at rule-based models and Conway's Game of life. We were very intrigued by how patterns could appear to emerge by only global rules being applied to each cell. For this project, we wanted to find real world applications to these phenomena and apply them. During our research, we came across pedestrian flow which discusses how people move in an enclosed space. We were particularly interested in Fruin's level of service which describes the different ranges of area occupancy that a given pedestrian may experience. When people are free to move around, they are in level A and in a very minimally dense area. When they are in level F, people can barely move because they are in an extremely dense area. We were able to capture this through our evaluation metric which includes the exit time, number of collisions, and area occupancy. Furthermore, Figure 6.1 is an example of some initial results looking at the cells that pedestrians occupied over time. Our project focused on replicating and modeling pedestrian flow of people leaving a room, while minimizing a pedestrian's area density and avoiding level F of Fruin's level of service. Ultimately, we were able to code an online simulation that represents different sizes of people with different objects, leaving different types of rooms.

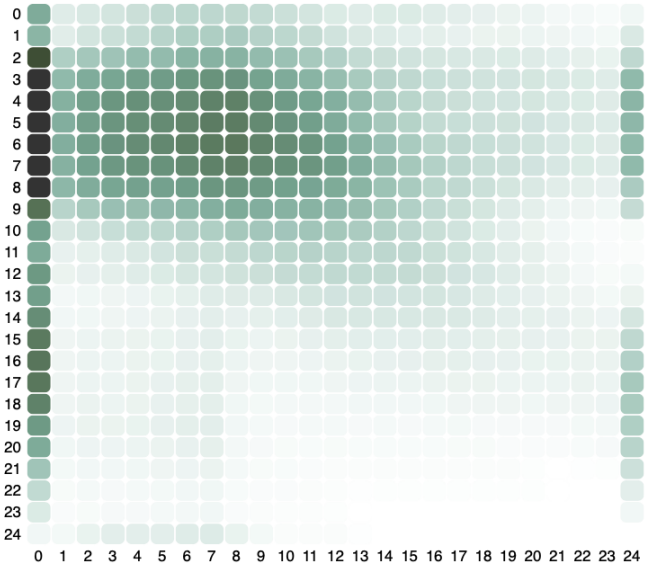


Figure 6.1: An example heatmap that is produced at the end of a simulation run. The darker the color, the more times that cell was visited during the duration of the run. This specific simulation was initialized on a 25 by 25 grid, with 50 children, and 1 exit.

When developing our simulation, we decided to use the A* algorithm as our global rule. While traditional global rules apply to all of the cells no matter where they are located (like in Conway's Game of Life), we call A* our global rule even though it takes into account the layout of the board and makes a move depending on its position, because we were motivated by rule-based models. Also, we implemented local rules which our pedestrians use when they find themselves in deadlock positions. These rules were developed in order to replicate how people actually move in different situations. For example, in order to avoid high levels of Fruin's level of service, people would not move towards a dense exit, but otherwise towards a less dense one, even if it might be a little bit further away.

There were many important factors taken into account when creating our simulation, and we wanted to make our simulation different from the other ones that already exist. What makes our simulation different than existing simulations, is that the user has total control on changing the parameters of the simulation. Users can change the room layout, number of people initially in the grid, and can alter all of the different strategies that are implemented which try to accurately model pedestrians leaving a room. Additionally, we were able to evaluate and compare these different initializations and strategies. We found some "common sense" outcomes that help us verify that the simulation can model how people actually act when leaving a room. For example, as the number of people carrying backpacks in a room increases, the overall time taken for everyone to leave the board increases. We also created an evaluation metric that allowed us to rank all of the runs we tested. Overall, the worst initialization was a 150x150 grid with 100 children, 100 adults, 100 adults with backpacks, 75 adults with bikes, 0 obstacles, and 4 exits.

We were able to successfully model and evaluate pedestrian flow, while allowing users the flexibility and freedom in interacting with the simulation. While our work validated current pedestrian flow simulations, there are some applications that this simulation could be used for, one of which includes working with fire departments in evaluating the number of people safely allowed to occupy a room. Additionally, our project has the potential to continue to grow by implementing different strategies that could model pedestrians even better. In conclusion, we were able to accomplish the goals we set for this project, and are excited to see the different applications and future work that our project could continue to grow to.

Chapter 7

Future Work

During the construction of our simulation, there were certain restrictions on our project, the main one being time. There are some features and project ideas that we just did not have time to implement, but feel as though they are important to discuss for future work on this project.

Firstly, the scope of our project takes into account people leaving a room. However, we never take into consideration if there are people who are trying to enter a room at the same time people are getting out. This situation would happen for example during the change of class, when students are leaving the classroom, and others are trying to enter for their class coming up. We decided on first focusing on people leaving the room because in most situations, people do not enter a classroom until the previous class is over or there is no longer anyone coming out of a room. However, it would be interesting to evaluate and compare pedestrian movement when taking into account the line of people crowding a hallway, waiting to get into a room for the next class.

Since one of the goals of our project was to allow the user the flexibility of varying different inputs and initializations, there are a multitude of different initializations. One can vary the number of people on the grid, size of the grid, heuristic functions used, conflict resolution strategies and their corresponding thresholds. Therefore, the amount of different initializations is pretty much infinite. We had to decide on the specific initial board configurations we wanted to test, and we tried to test a variety of different initial configurations. However, there are many more features and combinations we would have liked to test if we had more time. One of these different initializations that is particularly interesting is expanding the layout to be a whole building, and using obstacles to construct different rooms. It would be interesting to see how our evaluation metric changes when evaluating entire buildings. Also, we decided on three different heuristic functions and five different conflict resolution strategies to test. However, there are many more heuristics and conflict resolution strategies that one would be able to come up with and test. For example, a heuristic function could take into account the length of the path from the current to goal cell, but also add the number of obstacles one would be directly next to during that path. A* could also be altered to only find the first five or so moves a person should take. This would decrease run time and it would be interesting to see if the accuracy of the model would differ.

Additionally, our simulation could be tested to determine the maximum number of people allowed in a room. By looking up fire codes, we would be able to evaluate a simulation and if the exit time exceeds the allowed limit, then there would be too many people initially in the room. This room limit would be different than existing room limits, because they do no take into account objects, like bikes and backpacks. Ideally, our simulation evaluation could be even more accurate in predicting the amount of people who could safely leave a building or room in a given amount of time. This is due to the fact that we take into account objects that take up more space in a room than just a single person.

Something else that we did not have the time to look into is the possibility of pedestrians moving at different speeds. What happens when someone in a room has to run all the way across campus and get to their next class, but most other people have their next class in the just the building over? It is possible that some people would be rushing out of the room, and therefore moving at a greater pace than people who casually walk to their next class. Currently, our simulation has people moving at two different speeds: moving and standing still. It would be very interesting to look into and evaluate the results based on people moving at different speeds. This could also be added as another parameter that a user would be able to change the speed, and vary the speeds of the pedestrians in the simulation.

Another feature that we are leaving for future work is allowing pedestrians to move in more than 8 directions. While 8 directions gives the pedestrian a decent amount of variety in their movements, it is still not totally realistic. Ideally, we would give pedestrians the option of moving at 360 different angles in order for their movements to be even more realistic. However, implementing this would cause the A* algorithm's run time to increase exponentially, something that we were not interested in doing for our project. In addition, we did not have time to implement a bike, or anyone turning when they change their direction. Realistically, when someone chooses to move to a different exit, they turn and face the direction they are walking in and do not walk backwards to the exit. But, when a person turns around, their orientation changes and causes the person to occupy other cells, either in the turning process, or both in the turning process and once the object has turned. Because of this, we did not decide to implement this into our simulation. We would have to check the availability of other cells to see if a pedestrian would have the capability of turning around before being allowed to move in a different direction.

Currently, the simulation removes a pedestrian from the board when any part of their profile is touching any part of an exit. However, in reality, a person's whole body and belongings need to fit out of the front of a door at a time. Originally we were going to implement this into our simulation, and is the reasoning why our exits are four cells wide. However, we ran out of time in implementing this feature, and decided that there were more important features that we wanted to test our simulation on. Implementing this type of pedestrian exiting would make the simulation more realistic and would be interesting to see if results change based on this feature. Additionally, as mentioned before, a whole person and all of its cells are removed once any part of its body is touching an exit. It would also make the simulation more realistic if only one cell or row of cells of the person was removed at a time, and for each consecutive board generation, only the cells that are touching an exit from the front gets removed from the board.

When choosing the exit locations for a given simulation trail, one cell is randomly chosen along the perimeter of the board and either a horizontal exit is formed with the given cell and the three cells to the right, or a vertical exit is formed with the given cell and three cells below it. Vertical and horizontal exits are chosen depending on the side of the board that the random cell chosen is on. If the position of the chosen cell in the x direction is zero or the length of the board, a vertical exit is formed. This is to ensure that exits are randomly placed on the perimeter of the board, and not in the middle of a room. However, due to this logic, there are two boundary cases that need to be addressed. Since there is a vertical exit when the x position is zero, the cell in the top left corner will always be vertical, and there will never be a horizontal exit from that cell when randomly placing exits on a grid. This affects the randomness of exit placement, and it is important to note that a horizontal exit will never be randomly placed in the upper left corner of the board. The second case is the bottom left corner of the board. If the position of the chosen cell in the y direction is 0 or the length of the board, a horizontal exit is formed. But due to the logic in our code, vertical exits are checked first so a horizontal exit will never be formed from the bottom left hand corner of the simulation. Instead, if that cell is picked, it will be an exit cell of one cell wide. We discuss these two cases in future work because if we were to change

the code now, the random seed will no longer align with previous runs. We leave these edge cases for future work.

The last feature we discuss in future work is an interesting idea about obstacles. When creating the simulation, we wanted to include obstacles to act as objects like chairs or desks that take up space in a room, therefore people cannot occupy the space of an obstacle. We also assume that obstacles can never move during the simulation. However, in real life, people can move a chair if it is in their way, or other obstacles that are easy enough to carry. So, it would be very interesting to see how the simulation would work if people could actually move certain obstacles and place then somewhere else instead of always avoided the obstacles. The ideas and features discussed are all very interesting concepts, and provide new and different ways to expand and improve this project so that the simulation could be even more realistic.

Bibliography

[1] *A Javascript Implementation of Ants*. URL: http://www.natureincode.com/code/various/ants.html (visited on 01/29/2022).

[2] *A* Search Algorithm*. en-us. Section: Algorithms. June 2016. URL: https://www.geeksforgeeks.org/a-search-algorithm/ (visited on 12/15/2021).

[3] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. "Winning Ways for Your Mathematical Plays". In: *Journal of Multivariate Analysis* (1982).

[4] Mike Bostock. *D3.js - Data-Driven Documents*. D3. URL: https://d3js.org/ (visited on 02/14/2022).

[5] University of Bristol. *What is a Voronoi diagram?* Last Modified: 2021-01-15 Publisher: University of Bristol. URL: https://www.bristol.ac.uk/maths/fry-building/public-art-strategy/what-is-a-voronoi-diagram/ (visited on 02/14/2022).

[6] Jason Brownlee. *A Gentle Introduction to k-fold Cross-Validation*. Machine Learning Mastery. May 22, 2018. URL: https://machinelearningmastery.com/k-fold-cross-validation/ (visited on 04/25/2022).

[7] C. Burstedde et al. "Simulation of pedestrian dynamics using a two-dimensional cellular automaton". In: *Physica A: Statistical Mechanics and its Applications* 295.3 (2001), pp. 507–525. ISSN: 0378-4371. DOI: https://doi.org/10.1016/S0378-4371(01)00141-8. URL: https://www.sciencedirect.com/science/article/pii/S0378437101001418.

[8] Lily A. Chylek et al. "Rule-based modeling: a computational approach for studying biomolecular site dynamics in cell signaling systems". In: *Wiley Interdisciplinary Reviews: Systems Biology and Medicine* 6 (2014).

[9] Debasis Das. "A Survey on Cellular Automata and Its Applications". In: *Communications in Computer and Information Science*. Vol. 269. Dec. 2011. ISBN: 978-3-642-29218-7. DOI: 10.1007/978-3-642-29219-4_84.

[10] *Driver Assistance Technologies — NHTSA*. en. Text. URL: https://www.nhtsa.gov/equipment/driver-assistance-technologies (visited on 10/07/2021).

[11] *Evacuation - Simulation Models in AnyLogic Cloud*. URL: https://cloud.anylogic.com/model/90677dc1-15c5-4bb9-917e-7f6547feceb7?mode=SETTINGS&tab=GENERAL (visited on 10/05/2021).

[12] John J. Fruin. "DESIGNING FOR PEDESTRIANS: A LEVEL-OF-SERVICE CONCEPT". In: *Highway Research Record* (1971).

[13] Master Gardener. "Mathematical games: the fantastic combinations of john conway's new solitaire game "life". In: 1970.

[14] H. Gayathri, P. M. Aparna, and Ashish Verma. "A review of studies on understanding crowd dynamics in the context of crowd safety in mass religious gatherings". In: *International journal of disaster risk reduction* 25 (2017), pp. 82–91.

[15] J. Heudin. "A New Candidate Rule for the Game of Two-dimensional Life". In: *Complex Syst.* 10 (1996).

[16] *Heuristics*. URL: https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html (visited on 04/27/2022).

[17] *How to Install Node.js and NPM on Your Windows System*. Knowledge Base by phoenixNAP. Oct. 28, 2019. URL: https://phoenixnap.com/kb/install-node-js-npm-on-windows (visited on 01/30/2022).

[18] Minki Hwang et al. "Rule-Based Simulation of Multi-Cellular Biological Systems—A Review of Modeling Techniques". en. In: *Cellular and Molecular Bioengineering* 2.3 (Sept. 2009), pp. 285–294. ISSN: 1865-5033. DOI: 10.1007/s12195-009-0078-2. URL: https://doi.org/10.1007/s12195-009-0078-2 (visited on 10/07/2021).

[19] *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/ (visited on 04/25/2022).

[20] Melanie Mitchell et al. "Computation in Cellular Automata: A Selected Review". In: *Nonstandard Computation*. 2005.

[21] Katsuhiro Nishinari. "Jamology: Physics of Self-driven Particles and Toward Solution of All Jams". In: *Distributed Autonomous Robotic Systems 8*. Ed. by Hajime Asama et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 175–184. ISBN: 978-3-642-00644-9. DOI: 10.1007/978-3-642-00644-9_15. URL: https://doi.org/10.1007/978-3-642-00644-9_15.

[22] Katsuhiro Nishinari et al. "Modelling of self-driven particles: Foraging ants and pedestrians". In: *Physica A: Statistical Mechanics and its Applications* 372.1 (2006), pp. 132–141. ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2006.05.016. URL: https://www.sciencedirect.com/science/article/pii/S0378437106005917.

[23] *NumPy*. URL: https://numpy.org/ (visited on 04/25/2022).

[24] Dr Dirk Oberhagemann. "Static and Dynamic Crowd Densities at Major Public Events". In: *Technical Report* (2012), p. 46.

[25] Zan Pan. "Emergence from Symmetry: A New Type of Cellular Automata". In: *arXiv: Cellular Automata and Lattice Gases* (2010).

[26] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (visited on 04/25/2022).

[27] Angshuman Pandit, Anuj Budhkar, and Soumya Dey. "Study of Pedestrian Stream Behaviour at Mass Gathering". In: Dec. 2020.

[28] *Play John Conway's Game of Life*. en. URL: https://playgameoflife.com/ (visited on 10/13/2021).

[29] *PTV Viswalk new*. PTV-Group. Sept. 20, 2021. URL: https://www.ptvgroup.com/en/solutions/products/ptv-viswalk/ (visited on 10/05/2021).

[30] Rob Rennie. *auto-car*. original-date: 2018-12-29T05:37:43Z. Mar. 2020. URL: https://github.com/robrennie/auto-car (visited on 10/13/2021).

[31] *Ruiz's Summary of all Search Methods*. en-us. (Visited on 12/15/2021).

[32] S.J. Russell, S. Russell, and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2020. ISBN: 978-0-13-461099-3. URL: https://books.google.com/books?id=koFptAEACAAJ.

[33] MohammadReza Saadat. "Cellular Automata in the Triangular Grid". eng. In: (Feb. 2016). Accepted: 2018-04-13T07:37:15Z Publisher: Eastern Mediterranean University (EMU) - Doğu Akdeniz Üniversitesi (DAÜ). URL: http://i-rep.emu.edu.tr:8080/xmlui/handle/11129/3618 (visited on 04/27/2022).

[34] Siamak Sarmady, Fazilah Haron, and Abdullah Zawawi Talib. "Simulation of Pedestrian Movements Using Fine Grid Cellular Automata Model". In: *CoRR* abs/1406.3567 (2014). arXiv: 1406.3567. URL: http://arxiv.org/abs/1406.3567.

[35] J.L. Schiff. *Cellular Automata: A Discrete View of the World*. Wiley Series in Discrete Mathematics & Optimization. Wiley, 2011. ISBN: 978-1-118-03063-9. URL: https://books.google.com/books?id=uXJC2C2sRbIC.

[36] Oksana Severiukhina et al. "The study of the influence of obstacles on crowd dynamics". In: *Procedia Computer Science* 108 (2017), pp. 215–224. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.05.162. URL: https://www.sciencedirect.com/science/article/pii/S187705091730738X.

[37] *Skewness and the Mean, Median, and Mode – Introductory Business Statistics*. URL: https://opentextbc.ca/introbusinessstatopenstax/chapter/skewness-and-the-mean-median-and-mode/ (visited on 04/25/2022).

[38] T. Toffoli and N. Margolus. *Cellular Automata Machines: A New Environment for Modeling*. MIT Press series in scientific computation. Cambridge, 1987. ISBN: 978-0-262-20060-8. URL: https://books.google.com/books?id=HBlJzrBKUTEC.

[39] *Top Self Driving Car Companies In 2022 — Analytics Steps*. URL: https://www.analyticssteps.com/blogs/top-self-driving-car-companies-2022 (visited on 10/07/2021).

[40] Patrick Tu. *Conway's Game of Life meets the Simple Encrypted Arithmetic Library (SEAL)*. Patrick Tu — M.Sc. Data Science Student. URL: https://www.patrick-tu.com/blog-backend/2018/7/30/homomorphic-encryption-conways-game-of-life-meets-the-simple-encrypted-arithmetic-library-seal (visited on 10/05/2021).

[41] *Using the Bayesian average in ranking*. Algolia. URL: https://www.algolia.com/doc/guides/solutions/ecommerce/relevance-optimization/tutorials/bayesian-average/index.html (visited on 04/23/2022).

[42] Yosel Del Valle. *Automata Based Crowd Simulation*. original-date: 2017-09-25T23:36:52Z. Nov. 23, 2017. URL: https://github.com/lmydayuldd/pedestrian-flow-automata (visited on 10/05/2021).

[43] *Voronoi diagram*. In: *Wikipedia*. Page Version ID: 1065491669. Jan. 13, 2022. URL: https://en.wikipedia.org/w/index.php?title=Voronoi_diagram&oldid=1065491669 (visited on 02/14/2022).

[44] *What is a rule-based system? What is it not?* en-GB. Jan. 2019. URL: https://www.thinkautomation.com/eli5/what-is-a-rule-based-system-what-is-it-not/ (visited on 10/07/2021).

[45] *What Is Cryptography and How Does It Work? — Synopsys*. URL: https://www.synopsys.com/glossary/what-is-cryptography.html (visited on 10/05/2021).

[46] Stephen Wolfram. "Statistical mechanics of cellular automata". In: *Rev. Mod. Phys.* 55 (3 July 1983), pp. 601–644. DOI: 10.1103/RevModPhys.55.601. URL: https://link.aps.org/doi/10.1103/RevModPhys.55.601.

[47] Lanqing Yang et al. "Analysis of the Impact of Self-driving Cars on Traffic Flow". en-US. In: *DEStech Transactions on Computer Science and Engineering* 0.aiea (2017). Number: aiea. ISSN: 2475-8841. DOI: 10.12783/dtcse/aiea2017/14946. URL: http://www.dpi-proceedings.com/index.php/dtcse/article/view/14946 (visited on 10/07/2021).

[48] Xiaowen(Shawn) Ying. *Pedestrian Flow Simulator*. original-date: 2016-11-20T21:05:00Z. Sept. 2, 2019. URL: https://github.com/xwying/Pedestrian_Flow_Simulator (visited on 10/05/2021).