A Cellular Automata Model for Predicting Crowd Movement during Evacuation

ANMOL SINGH DHALIWAL, ANTARA GHOSH, NAMAN MANSUKHANI

> All the wonders of the universe can in effect be captured by simple rules, yet…there can be no way to know all the consequences of these rules, except, in effect, just to watch and see how they unfold.
>
> **– Stephen Wolfram**

Introduction

We usually go for outings to shopping malls, cinema halls, and amusement parks or go to our respective workplaces, such as offices or schools. But have you ever paid attention to that small evacuation map hanging on the side of the corridor of your building? We usually walk past these and take these for granted. But what if an actual earthquake or some other disaster were to happen? How would you manage the situation then?

Many public places attract large crowds of people and are susceptible to stampedes and other disasters. During a panic situation, say an earthquake, if people are inside a building, they would generally move in a random fashion causing crowding and clogging at exit points. Often an evacuation system exists, but it isn't the best option. Hence, having an effective evacuation plan is extremely crucial in such situations. Mathematical modeling can help us develop more efficient evacuation plans.

Researchers have been developing mathematical models for pedestrian movements and evacuation systems. Many of these models are based on differential equations [2]. Such models are highly complex and often difficult to interpret and understand. Numerical methods are used to solve such models and these may be computationally expensive and time consuming.

In this article we will describe a project in which the concept of cellular automata (CA) has been used to develop a simulation model

Keywords: modeling, cellular automata, simulation, math applications

for an evacuation system. Cellular automata is a mathematical tool which can govern complex systems through simple rules. It has applications in traffic flow modeling, structural design, neural networks, forest fires, ant colony activity, crystal growth and in many other fields [1]. Cellular automaton models can be one, two or even three dimensional. They can be generated and explored through simulation and their outputs are very conducive for exploring real life crisis situations.

The aim of the project was to

1. Apply the two dimensional Cellular Automata to simulate the movement of people in a crowded room, hall or corridor.
2. Develop a Python driven CA model to simulate crowd movement in times of a crisis.

Some mathematical preliminaries

In this section we shall explore some basic questions related to cellular automata. To begin with, what is a cellular automaton?

A cellular automaton (CA) is a collection of cells on a grid of a specified shape that evolves through discrete time steps according to a set of rules based on the state (or color) of the neighbouring cells.

- Each cell has a state – dead or alive. Pictorially the dead cells can be represented by white color and the live cells are coloured black. They can also be represented using 0s and 1s, where 0 represents a dead cell and 1 represents a live cell.
- Each cell has a neighbourhood. A neighbourhood of a given cell is a set of cells which are adjacent to it.
- Every CA must have a certain set of rules based on which it evolves through various time steps. These rules are referred to as the defining rules of a CA.

One-dimensional cellular automata

In a one-dimensional cellular automaton (also referred to as an elementary cellular automaton or ECA), the state of a cell can be either 0 or 1 (dead or alive). Such automata are represented on a linear grid of cells, in which each cell has two neighbours (the cell to its left and the one to its right). Each row of the automata represents a different generation or evolution in a different time step. A more detailed account of ECA may be found in the article titled *The Elementary Cellular Automata: A journey into the computational world* [3] published in the March 2018 issue of At Right Angles.

If we consider any three adjacent cells of a linear grid, each of these can be coloured either black or white. Hence, there are a total of 2 x 2 x 2 = 8 possibilities of colouring a set of three adjacent cells. See Figure 1, in which the top rows (of three cells each) represent these eight combinations. Further, each of these combinations can be assigned 0 or 1 (as indicated in Figure 1). Hence, the total number of defining rules in a one-dimensional CA is $2^8 = 256$ possibilities. These possibilities or ECA combinations, are numbered from 0 to 255. Each ECA combination is therefore identified by its *rule number* (from 0 to 255). They can also be represented as binary numbers. For example, Figure 1 represents the defining rule for ECA rule 30 which also has the binary representation 00011110.

The defining rule 00011110 may be treated as a binary number whose decimal representation (namely, 30) is obtained as follows

$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 30.$

The evolution of a one-dimensional CA starts with an initial state (generation 0), and evolves according to one of the 256 ECA rules. Different

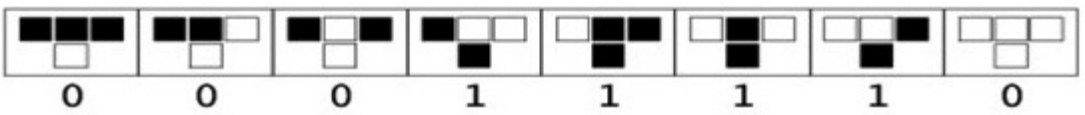
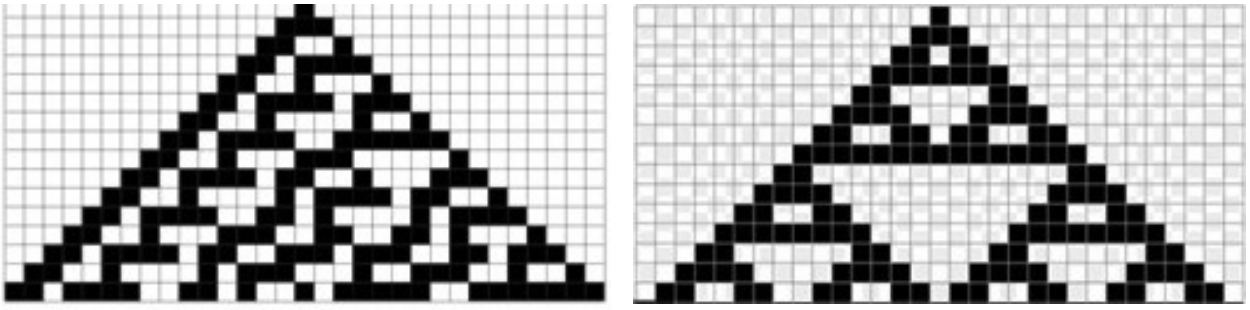


Figure 1. Defining rule for ECA rule 30.



Rule 30 Rule 126

Figure 2. Evolution of ECA rule 30 and rule 126 over 15 time steps starting with a single live cell in generation 0 (indicated by the topmost row).

rules generate different patterns. For example, rule 30 leads to a chaotic pattern, and rule 126 leads to a Sierpinski triangle like pattern. Figure 2 shows the first 20 generations (where each row of the grid represents a different generation) of the ECA rules 30 and 126 respectively.

Two-dimensional Cellular Automata

Like one-dimensional CA, the state of a cell in two-dimensional CA is also either dead or alive. A more detailed analysis of two – dimensional cellular automata may be found in [4]. However, a two-dimensional automaton evolves on a grid of square cells. One way to visualise a two-dimensional cellular automaton is to imagine an infinite sheet of squared paper and a set of rules according to which the cells evolve from one generation to the next. Each cell in a grid has neighbourhood cells which are the cells adjacent to it.

The two types of neighbourhoods widely used in two-dimensional CA are (a) the Von Neumann neighbourhood, and (b) the Moore neighbourhood. As shown in Figure 3, the

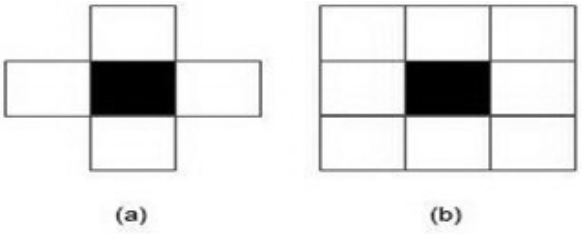


Figure 3. Two dimensional cellular automata neighbourhoods

Von Neumann neighbourhood of a given cell comprises the cells adjacent to it and positioned just above, below, to the left and to the right. In the case of the Moore neighbourhood, all the eight cells surrounding a given cell comprise the neighbourhood.

The state of the center cell in the next generation depends on all its neighbourhood cells in the current one. If we take the Moore neighbourhood, each of the nine cells can be either black or white. Hence, the total combinations would be 2^9 = 512 combinations. The central cell of each of these 512 possibilities can change in any way in the next time step according to defined rules. This leads to a total of 2^{512} possibilities which is a very huge number!

Game of Life

One of the very well-known and popular two dimensional cellular automata is the *Game of Life*. It is a game based on cellular automata that was invented by the British mathematician John Horton Conway [4] in the1960s. It is a one-player game, that is, only the initial configuration of cells is required and the game continues according to certain rules.

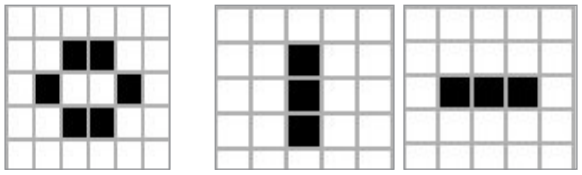
The rules of the Game of Life are as follows:

1. Any live cell with less than two live neighbours dies (due to lack of resources).
2. Any live cell with two or three live neighbours lives on to the next generation.

3. Any live cell with more than three live neighbours dies (due to excess resources and overcrowding).
4. Any dead cell with exactly three live neighbours becomes a live cell (by reproduction).

This implies that any initial configuration of cells (referred to as generation 0) will be subjected to the above four rules and will lead to a new configuration, that is, generation 1. Generation 1 will further undergo the rules (1 to 4) and so on. Interesting patterns are seen to emerge after a few generations. Some common pattern types include: *still lifes*, which do not change from one generation to the next; *oscillators*, which return to their initial state after a finite number of generations; and *spaceships*, which translate themselves across the grid. A more detailed account is available in [5].

Figure 4 illustrates one example of each type. Pattern (a) represents a still life called *beehive*, (b) represents an oscillator called *blinker* which oscillates between the two states and (c) represents a *glider*. The reader is urged to play the game online at https://playgameoflife.com/



(a) Beehive (still life) (b) Blinker (oscillator)

(c) Glider (spaceship)

Figure 4. Examples of patterns emerging from John Conway's Game of Life

A Two-dimensional Cellular Automata model to simulate the behaviour of people during evacuation

Let us imagine that the floor of a hall is a rectangular grid consisting of small square cells (Figure 5(a)). Each person in the hall occupies exactly one cell in a given time step. Thus a cell is *live* if it is occupied by a person and is *dead* if it is unoccupied (Figure 5 (b)). The state or configuration of the system at a given time step is represented by the black and white cells.

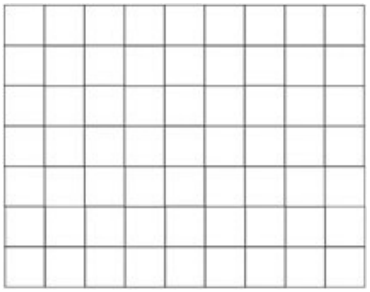


Figure 5(a). A grid of square cells representing the floor of a hall

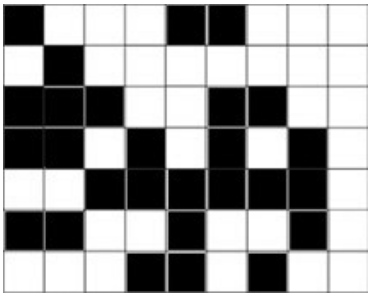


Figure 5(b). Black cells represent people standing inside the hall

The eight cells surrounding a given cell will be referred to as its neighbours (similar to Moore neighbourhood). In the next time step a person can move to any one of its 8 neighbouring cells.

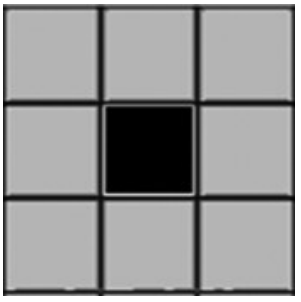


Figure 6. The black cell represents the central cell and the grey cells are its neighbours

Movement Rules

The rules determining the movement of the people occupying the room are as follows:

1. A person can move **only one cell at a time** and it can move only to its defined neighbourhood cells.
2. Logically, a person would want to take the **shortest route to the exit**. So, (s)he will move to one of its neighbourhood cells which is closest to the exit. If an obstacle is present, (s)he will move to the neighbourhood cell second closest to the exit and so on.
3. A person **cannot move in the direction opposite to the exit**. In case his/her movement is completely restricted by other neighbouring cells, (s)he does not move.
4. If there is a conflict between two people for one cell, the one **closer to the exit will prevail**, that is, will get to move.

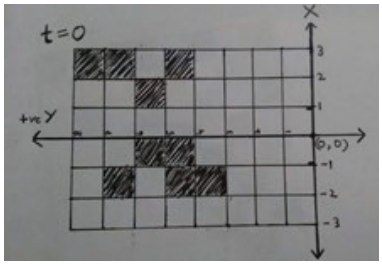


Figure 7. The state or configuration of the room at time t=0

In Figure 7 we have an 8 x 6 sized grid (representing a room) with the exit positioned at the origin. The black cells represent cells occupied by people at time t = 0.

Each cell has specific coordinates assigned to it. Figure 8 represents the configuration at time t = 1, after applying the movement rules 1 to 4.

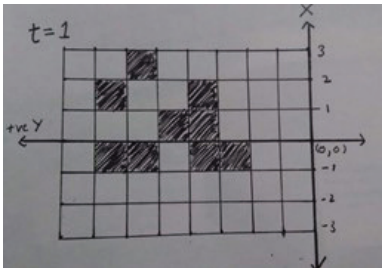


Figure 8. The configuration of the hall at time t=1

Movement rules (such as rule 1 to 4 mentioned above) can be used to simulate the movement behaviour of people during an evacuation.

The Python-driven CA Model for a single room

The goal of this model was to simulate the movement of people exiting a single room. The model was simulated using the Python mode for Processing, an open-source graphics library. The model is based on an m x n grid, with the exit at the cell corresponding to the mth row and the nth column (the bottom right corner). The model takes in another argument x, the number of people in the room.

Occupied and empty cells are represented by black and white colours respectively. For each iteration of the model, x random positions are coloured black at t = 0. The configuration of the grid at t = n is a function of the configuration at t = n – 1. In order to determine the configuration at the next step, the cells adopt a 'greedy' strategy. Each cell tries to move along the diagonal if possible; if this is not possible, then the cell moves randomly either to the bottom or to the right, depending on whether the desired cell is empty. For each iteration, the cell at the front gets the preference in movement, as is seen in real-world situations as well. Figure 9 illustrates the simulation where people are exiting a room (represented by an 8 by 8 grid enclosed within the red square) where the exit is located in the bottom right corner of the square.

The algorithmic complexity in order to determine the configuration at the next time-step is O(mn), where m = number of rows and n = number of columns.

The Python driven CA Model for a multi-room floor evacuation

The goal of the second simulation was to extend the first one and study floor-based evacuations rather than room-based ones. It was developed purely in Python using industry-standard addon libraries like 'numpy' and 'matplotlib', and the inbuilt libraries 'time' and 'random'. Typically

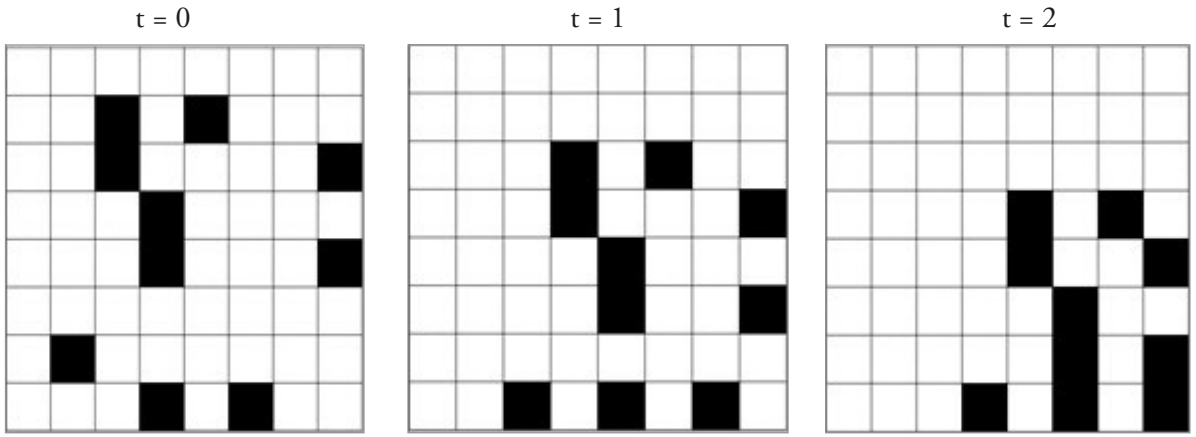


Figure 9: Model with m = n = 8 and k = 10 at time steps t = 0, t = 1 and t = 2

such problems would require the use of graph theory to find the shortest path for each evacuee. However, our model simplifies this by dividing the coordinate frame into unit cells, and using 2D Euclidean distance measurements between cells, binary matrices, and sorting algorithms to generate the movements at each time step.

Simple shifting of origin concepts are used to convert evacuee positions in each room to global coordinates for the floor, and a 'global' binary matrix is maintained to keep track of occupied cells for the entire floor.

In this model [x, y] denotes a matrix index, (x, y) denotes a 2D coordinate, and {cx, cy} denotes a movement pair of cx units horizontally, and cy units vertically.

Any closed space (a room or a corridor) is defined using (a) a bottom left coordinate, and (b) its size as a width and height pair. An example definition would be a bottom left coordinate of (1, 1) and a size of (20, 20), indicating a room/corridor that starts at (1, 1) in the global reference frame, and has a size of 20 x 20. Thus, it would be a square having a diagonal between the points (1, 1) and (21, 21).

Figures 10 and 11 are outputs of the Python simulation. Figure 10 shows three rooms of size 5 by 5 which open into a corridor while Figure 11 shows three rooms of varying sizes along with a corridor. The boundaries of the rooms and the corridor are indicated in red. Instead of narrowing down the frame to the smallest bounding box that would enclose all rooms and the corridor, five rows and five columns, respectively, have been padded around to improve aesthetic appeal.

The exits, represented by blue coloured lines, are also defined as coordinate pairs, in the frame of reference of each room, and then for the corridor as well. Exits of different lengths can be defined just by declaring their component edges of unit length. Rooms and corridors must have exactly one common edge, and the exit must lie on that edge. This is ensured while defining the model.

Steps of simulation

1. The number of evacuees in each room can be defined, and their locations are generated randomly without any overlaps. The locations are returned as coordinate pairs, local to each room. These occupied cell locations are assigned a value of 1 in the binary matrix, with empty cells having value 0.
2. A separate 'labels' array is also maintained for assigning numbers to each cell in each room based on order of random generation while in the room. The label for a cell is subsequently prefixed with a specific letter of the alphabet as soon as it exits the room. For example, in the frame of reference of room 'a', cell 1 refers to the first randomly generated coordinate cell. While it is in room 'a', it will remain as '1', and as soon as it exits the room, it will become 'a1' in the corridor.

- For example, visualise a unit square based grid, with a movement of [2, 2] → [1, 3] in a 3 x 3 room having bottom left corner as (5, 5). This will result in the transfer of colour black from the cell with bottom left corner as (6, 6) to the cell with bottom left corner as (7, 7) in the global frame. The underlying concept is just that matrix row indices increase in a top - bottom manner, whereas Cartesian y coordinates increase in a bottom - top manner, with row indices in matrices being indicative of the y coordinate, and column indices being indicative of the x coordinate.

The algorithm

The algorithm starts by calculating the euclidean distance between the center of each exit and the corresponding cell location for each cell. It has the capability of simulating two kinds of scenarios. Firstly, where each cell independently moves to its optimal cell in every timestep i.e. directly moving to the one that will take it closest to the exit (this may result in many collisions), and secondly, where each cell moves to a distinct cell depending on its closeness to the exit, factoring in neighbour awareness, which means this will result in no collisions.

In both scenarios, in every timestep, the cell may only move to any cell in its Moore neighbourhood (scenario 1), or additionally also have the capability of staying at its position (scenario 2). To exit, the cell must reach the unit cell having a common edge with the corridor and lying inside the particular room.

In the first scenario (Figure 10), a cell's final position is simply assigned as the one that will take it closest to the exit. The best step of the cell is therefore just a movement of +1, -1 or 0 in each of the two x and y directions, depending on the location of the exit. However, in this scenario, since it is effectively moving blindly i.e. simulating panic, it must move at least one unit in every timestep, so a {0, 0} i.e. static (no movement) step is not allowed.

In the following parts, when we say that an exit is located at a matrix index, we imply that that cell will have the exit(s) as its edge(s).

Consider an exit at [2, 5] in a room matrix. For an evacuee located at [5, 4], the best movement path will be [5, 4] → [4, 5] {-1, +1} → [3, 5] {-1, 0} → [2, 5] {-1, 0} → exit. However, if there is another cell located at [5, 5] in the first time step, it will also strive to move to [4, 5], and in this scenario, a *collision* will occur. The number of collisions are counted by determining the number of cells that have the same final coordinates at each time step, and then dividing by 2. Each collision has been given a one second penalty to simulate the real-world delay it causes, and in the plots, the labels for those colliding cells are overlaid on top of each other to also give a visual effect. The simulation continues until the last cell exits the corridor.

In the second scenario (Figure 11), the final positions are calculated for each cell in a hierarchical format depending on their closeness to the exit. This means that a cell closer to the exit will have the chance to move first and choose its preferred coordinate before the others. Once it chooses the final location, that location is invalidated for the other cells, implying an absolute *collision-free ideal evacuation*. If two cells are equidistant from the exit, e.g. lying in the left and right upper diagonals of the Moore neighbourhood of the target exit cell, the one having lower label value will get to move first. This also means that there is a possibility for a cell to remain at its current position in a timestep if all of its preferred locations have already been occupied by the cells that moved before it.

Screenshots of simulations

Scenario 1: Note the overlaps (in Figure 10 (a); an overlap can be seen in the lower room 'b', two units directly below the exit and in Figure 10 (b) one overlap can be seen in the cell lying in the middle row of the corridor just two units away from the long blue line center). The overlaps depict collisions and hence delays.

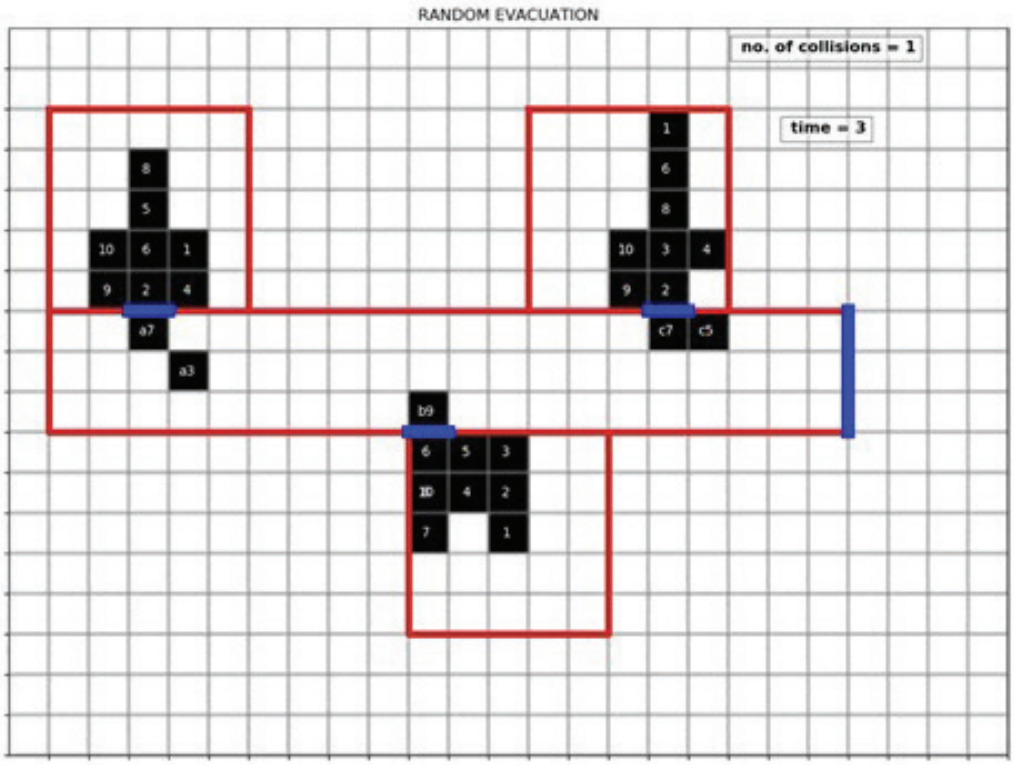


Figure 10(a). Scenario 1 simulation at t = 3 seconds

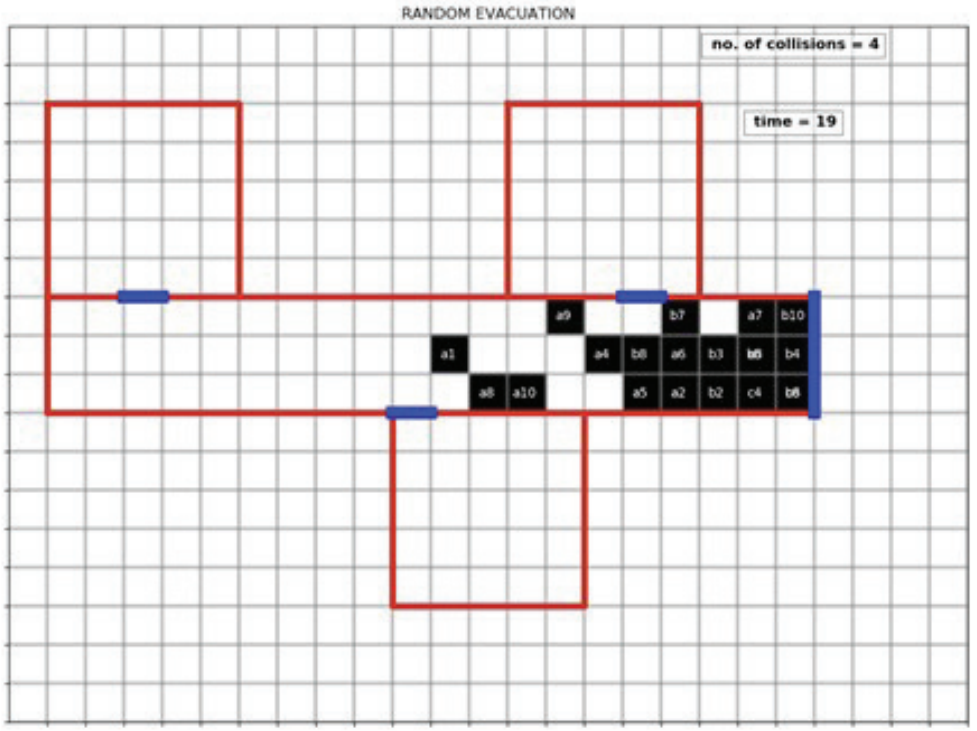


Figure 10(b). Scenario 1 simulation at t = 19 seconds

Scenario 2: A perfect orderly movement of the cells is visible. Also note the different room sizes and exit locations, showing the versatility of the algorithm.

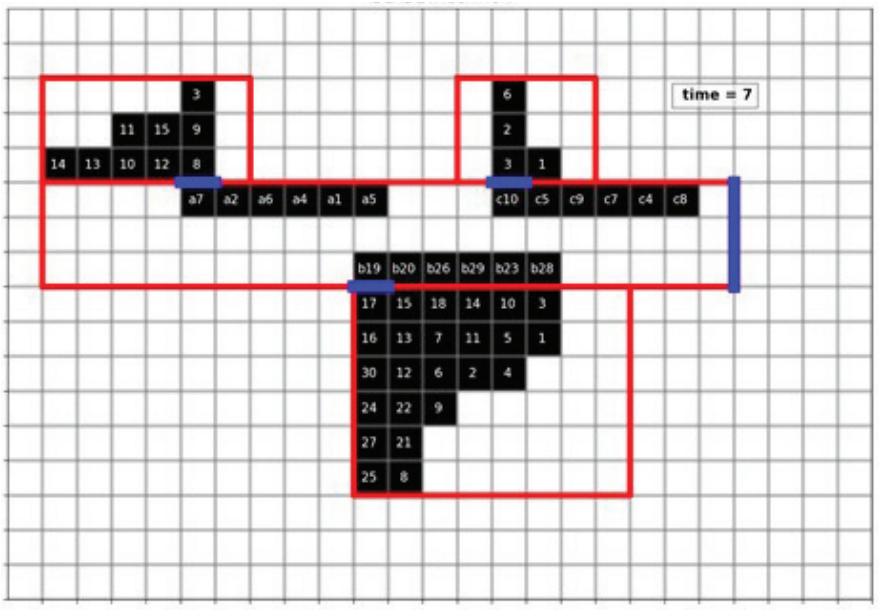


Fig 11(a): Scenario 2 simulation at t = 7 seconds

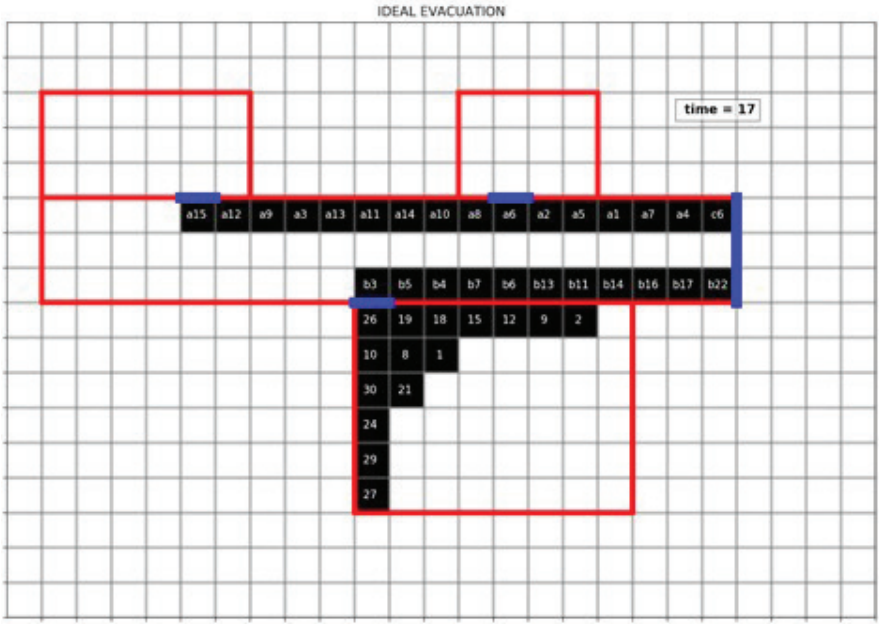


Figure 11(b). Scenario 2 simulation at t = 17 seconds

Supported by intuition, it was observed that the evacuation time for scenario 2 simulations was always less than scenario 1 simulations, for the same initial configuration of cells. Smaller rooms would have greater collisions, and thus the difference between the evacuation times between the two scenarios was more in those cases.

Conclusion

In this article we have tried to show that Mathematics and Computer Science can prove to be the ideal tools for studying phenomena such as evacuation during a crisis situation. Initially, we thought of using one dimensional ECA to model crowd behaviour in corridors but it proved

to be unrealistic and less efficient. The use of two dimensional CA was a necessity in this case.

The Python implementations verified that panic caused by random movements of people inevitably lead to longer evacuation times and can be harmful in the long run. The proposed model of rule-based evacuations with neighbour awareness is an example of a robust model that will lead to ideal, collision-free evacuations with least possibility for injuries and loss of life.

This project can be extended to simulation of people exiting a hall with two or more exits and identifying an ideal path for evacuation. We also plan to increase the complexity of the model by studying bidirectional crowd movements (such as at a subway or a metro station) and finding an efficient means of crowd management.

Finally, we would like to thank Mr. Mukesh Kumar, Head of Computer Science Department, Delhi Public School, R K Puram, for his support and guidance in developing the algorithms and Python implementations. We would also like to acknowledge Mr. Anil Kathuria, Head, Department of Mathematics, and our teachers Ms. Tandeep Kaur, Ms. Shalini Monga and Ms. Nagalakshmi for their encouragement and support.

References

[1] Gong, Yimin. (2017). A survey on the modeling and applications of cellular automata theory, IOP Conference series: Materials science and engineering 242 (2017) 012106 .

[2] Sarmady, S., Haron, F., Talib, Abdullah Z., (2014). Simulation of pedestrian movements using fine grid cellular automata model. Retrieved from https://arxiv.org/pdf/1406.3567.pdf

[3] Ghosh, J.B & Adsule, R. (2018). The Elementary Cellular Automata: A journey into the computational world, At Right Angles, Volume 7 (1), pp. 95-101.

[4] Huxley, T.H. Two dimensional automata, retrieved from http://psoup.math.wisc.edu/491/Schiff_4.pdf

[5] Conway's game of life retrieved from https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Anmol Singh Dhaliwal
anmolsinghdhaliwal03@gmail.com

Antara Ghosh
antaraghosh1966@gmail.com

Naman Mansukhani
naman.mansukh@gmail.com

The authors are students of class 12. Anmol Singh Dhaliwal has three published articles related to using Wolfram Mathematica in robotics and English language teaching to underprivileged children. His interests are coding, robotics and mathematical modelling. Antara Ghosh has interests varying from dancing and playing basketball to physics and mathematics and solving real life problems. Naman Mansukhani enjoys spending time on websites like Codeforces and Brilliant and is interested in machine learning and neural networks. He enjoys learning about other cultures and their languages.