Modeling Pedestrian Evacuation Strategy with Cell-DEVS and Cadmium Simulator

*Harshita Ghushe Gabriel Wainer*

Department of Systems and Computer Engineering Carleton University

1125 Colonel By Drive

Ottawa, ON. K1S-5B6 Canada

{harshitaghushe, [gwainer}@sce.carleton.ca](mailto:gwainer%7d@sce.carleton.ca)

**ABSTRACT**: *Cell-DEVS formalism is a way of describing the behavior, structure, and dynamics of natural systems with spatial features using discrete-event cellular models. However, it has limitations in defining scenarios with irregular cell shapes. To address this, the asymmetric Cell-DEVS formalism was introduced to support cellular models with irregular topologies and complex relationships between cells. A prototype implementation of this formalism using the Cadmium simulator was created, and different cellular models were combined with geographical data to produce realistic results. The paper discusses the use of Cell-DEVS with the Cadmium simulator to simulate pedestrian evacuation strategies in a room, taking into account the visual field restrictions. The simulation involves modeling people's evacuation reactions using cellular automata with different categories for choosing evacuation strategies based on the reference of the wall until they find an exit. In the new version of Cadmium, each person is represented as a cell implemented in C++, and the cell space is defined using a JSON configuration file, enabling the study of multiple setups by modifying the configuration file without recompilation. The model involves updating people's states to evacuate the room through the exit(s).*

# Introduction

The study of pedestrian dynamics is essential in managing pedestrian evacuations and designing pedestrian facilities. An extended cost potential field cellular automaton model is a fundamental tool for this purpose. It enables the simulation of pedestrian movement during emergencies and helps in the development of appropriate evacuation protocols and regulations. Various physical methods have been used to observe and replicate the dynamic properties of pedestrian crowds, including their self-organizing behaviors. However, simulating pedestrian movement during emergencies is challenging when visibility is affected, which can happen due to factors such as smoke, fire, power outages, or panic. Most simulations assume ideal visibility conditions, but, visibility can be reduced, which affects pedestrian behavior. Therefore, understanding the behavior of pedestrians in adverse sight conditions is crucial for accurately simulating pedestrian evacuation in such situations. The conceptual model involves simulating the evacuation reactions of people in a room using cellular automata (CA) and Cadmium Simulator. The CA is equipped with different categories that determine the evacuation strategy of the pedestrians, such as choosing to go in multiple directions based on the reference of the wall until they find an exit, which helps in effectively managing emergency situations.

# Background

**CELL-DEVS:**

Cell DEVS is extended DEVS formalism, in which we have implementation of cellular automata to improvise the execution performance of cellular models using a discrete event approach. Here, each cell is defined as atomic model using timing delays, which is later merged to a coupled model representing a cell space.

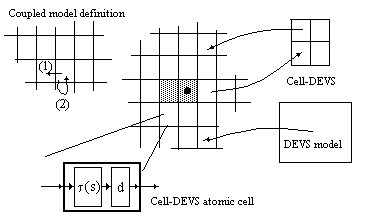
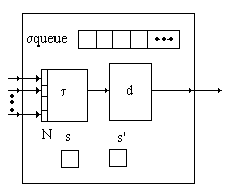


Fig 1: Cell-DEVS Coupled model Fig 2: Cell-DEVS Atomic Model

**CD++ Toolkit:**

CD++ (Wainer, 2002) is a modelling and simulation toolkit that implements DEVS and Cell-DEVS theory. Atomic models can be defined using a state-based approach (coded in C++ or an interpreted graphical notation), while coupled and Cell-DEVS models are defined using a built-in specification language. We will show the basic features of the tool through an example of application. CD++ also includes an interpreter for Cell-DEVS models. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighbourhood and borders.

The cell’s local computing function is defined using a set of rules with the form:

{POSTCONDITION}{DELAY}{PRECONDITION}.

These indicate that when the PRECONDITION is satisfied, the state of the cell will change to the designated POSTCONDITION, whose computed value will be transmitted to other components after consuming the DELAY.

**CELLULAR AUTOMATA:**

To simulate cellular automata, there is a neighbourhood defined for each cell. Each cell has its 2 possible states that is whether it is empty or full. The neighbourhood is generally the adjacent residing cells. We classify the neighborhoods into 2 parts :  [von Neumann neighborhood](https://en.wikipedia.org/wiki/Von_Neumann_neighborhood) and the [Moore neighborhood](https://en.wikipedia.org/wiki/Moore_neighborhood)

A red and white flag

Description automatically generated with medium confidence

Fig 3: The red cells are the [neighborhood](https://en.wikipedia.org/wiki/Moore_neighborhood) for the blue cell

**Cell DEVS Formal Specification:**

To specify the model, we are utilizing the following Moore neighborhood

A picture containing text, crossword puzzle

Description automatically generated

As the above figure shows, the neighborhood consists of 9 cells:

{(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0, 1), (1, -1), (1,0), (1, 1)}

The core cell in the model is represented by (0,0), and pedestrians are allowed to move to adjacent cells in the north (-1,0), south (1,0), east (0,1), or west (0,-1) directions according to the rules specified above.

M = <Xlist, Ylist, I, X, Y, ƞ, N, {r,c}, C, B, Z, SELECT>

* Xlist = { Ø };
* Ylist = { Ø };
* I = { Ø };
* X = Y = {0,1,2,3,4,5};

The different states of the cells and their corresponding meanings are:

* + 0: Empty cell
  + 1: Cell/pedestrian occupied by an up walker
  + 2: Cell/ pedestrian occupied by a down walker
  + 3: Cell/pedestrian with state 1 has reached a wall and is moving left
  + 4: Cell/ pedestrian with state 2 has reached a wall and is moving right
  + 5: Wall cell
* Ƞ = 9;
* N = {(0,0), (0,1), (0,-1), (1,-1), (1,0), (1,-1), (-1,-1), (-1,0), (-1,1)};
* r = 20; c = 20;
* C = {Cij | i ϵ [0,19], j ϵ [0,19]};
* B = {Ø}; % wrapped
* Z = Inverse neighborhood of N
* SELECT = {(0,0), (1,0), (0,1), (0,-1), (-1,0)};

# Cell DEVS Simulation with Cadmium:

# This section introduces the new version of the Cadmium simulator that supports classic and asymmetric Cell-DEVS models. Cadmium is a header-only library written in C++, allowing the modeling and simulation of computational models based on the DEVS formalism. In this new version, cells are implemented in C++ while the cell space is defined using a JSON configuration file. This approach enables studying various setups by modifying the configuration file, thereby eliminating the need for recompilations, and reducing overall exploration time. Additionally, this version enables modelers to integrate Cell-DEVS models with other DEVS models implemented with Cadmium. The figure below illustrates the simulation lifecycle for exploring cellular models with the asymmetric Cell-DEVS formalism using the Cadmium simulator.

# Diagram Description automatically generated

# *Figure. Simulation Life cycle of Cell DEVS model with Cadmium*

# 

# 

# To use Cadmium, one first needs to define a conceptual cellular model describing the system under study following the asymmetric Cell-DEVS formalism. The conceptual model is then translated into a computational model using the tools provided by the Cadmium library. After implementing the computational model, one can run simulations over different scenarios by modifying the JSON configuration file. Finally, the simulation results are analyzed to gain insight into the system under study.

# Model Definition

**Rules Implementation:**

|  |  |  |  |
| --- | --- | --- | --- |
| State Names | Values | Colors | Description |
|  |
| Blank Cell | 0 | Silver | Empty cell. |  |
| Pedestrian Evacuating from Upwards direction | 1 | Red | This cell indicates individuals moving upwards on the grid. |  |
|  |
| Pedestrian Evacuating from downwards direction | 2 | Green | This cell indicates individuals moving downwards on the grid. |  |
|  |
| The Pedestrian occupying the cell with state 1 has now reached a wall and is moving towards the left. | 3 | Yellow | This cell represents a pedestrian who was previously moving up on the grid and has now reached a wall. The person is now changing direction and moving towards the left, relative to the wall. |  |
| The Pedestrian occupying the cell with state 2 has now reached a wall and is moving towards the right. | 4 | Blue | This cell represents a pedestrian who was previously moving down on the grid and has now reached a wall. The person is now changing direction and moving towards the right, relative to the wall. |  |
| Wall | 5 | Brown | Denotes a wall. |  |

**Model Specification:**

[top]

components : pedestrianevacuation

[pedestrianevacuation]

type : cell

width : 20

height : 20

delay : transport

defaultDelayTime : 100

border : wrapped

neighbors : pedestrianevacuation(-1,-1) pedestrianevacuation(-1,0) pedestrianevacuation(-1,1)

neighbors : pedestrianevacuation(0,-1) pedestrianevacuation(0,0) pedestrianevacuation(0,1)

neighbors : pedestrianevacuation(1,-1) pedestrianevacuation(1,0) pedestrianevacuation(1,1)

%%%% state values %%%%

% 0 - Empty Cell

% 1 - cell occupied by an up walker

% 2 - cell occupied by a down walker

% 3 - cell with state 1 has reached a wall and is going left

% 4 - cell with state 2 has reached a wall and is going right

% 5 - wall cell

initialvalue : 0

initialrowvalue : 0 55555555555555555555

initialrowvalue : 1 50000000000000000105

initialrowvalue : 2 50000000000000000005

initialrowvalue : 3 50000000000000000005

initialrowvalue : 4 50010000000000000005

initialrowvalue : 5 50000000000000000005

initialrowvalue : 6 00000000000000000005

initialrowvalue : 7 50000000000000000005

initialrowvalue : 8 50000000000000000005

initialrowvalue : 9 50000000000020000005

initialrowvalue : 10 50000000000012000005

initialrowvalue : 11 50000000000000000005

initialrowvalue : 12 50100000000000000005

initialrowvalue : 13 00000000000000200005

initialrowvalue : 14 50000000000000000005

initialrowvalue : 15 50000012000000000005

initialrowvalue : 16 50000000000000000005

initialrowvalue : 17 50000000000000000005

initialrowvalue : 18 51000000000000000025

initialrowvalue : 19 55555555555555555555

localtransition : pedestrianevacuation-rule

[pedestrianevacuation-rule]

%Pedestrian 1 - Going up

rule : 0 100 {(0,0)=1 and (-1,0)=0 and (1,0)!=5 and (-1,1)!=3}

rule : 1 100 {(0,0)=0 and (1,0)=1 and (0,-1)!=5 and (0,1)!=5 and (1,1)!=5 and (0,1)!=3}

%Pedestrian 1 has reached the top wall. State changes to 3 and going left

rule : 3 0 {(0,0)=1 and (statecount(5)>=1)}

%Pedestrian reached a wall at the top and cell to the left is empty

rule : 0 100 {(0,0)=3 and (0,-1)=0 and (-1,0)=5}

%Pedestrian is in current state 0, its right cell is in state 3 with wall at the top (Top to Left top)

rule : 3 100 {(0,0)=0 and (0,1)=3 and (-1,0)=5}

%Pedestrian has reached a wall to left top (0,-1) , cell below is empty

rule : 0 100 {(0,0)=3 and (1,0)=0 and (0,-1)=5}

%Pedestrian is in current state 0, its up cell is in state 3 with wall to the left top (Left top to Left bottom)

rule : 3 100 {(0,0)=0 and (-1,0)=3 and (0,-1)=5}

%Pedestrian has reached a wall to the left bottom(1,0), cell to the right is empty

rule : 0 100 {(0,0)=3 and (0,1)=0 and (1,0)=5}

%Pedestrian is in current state 0, the person to the left is in state 3 with wall to left bottom (Left bottom to right bottom)

rule : 3 100 {(0,0)=0 and (0,-1)=3 and (1,0)=5}

%Pedestrian has reached a wall to right bottom (0,1) , cell above is empty

rule : 0 100 {(0,0)=3 and (-1,0)=0 and (0,1)=5}

%Pedestrian is in current state 0, the person to the bottom is in state 3 with wall to right bottom (Right bottom to right top)

rule : 3 100 {(0,0)=0 and (1,0)=3 and (0,1)=5}

%Pedestrian 2 - Going down

rule : 0 100 {(0,0)=2 and (1,0)=0 and (-1,0)!=5 and (1,1)!=4}

rule : 2 100 {(0,0)=0 and (-1,0)=2 and (0,1)!=5 and (0,-1)!=5 and (-1,-1)!=5 and (0,1)!=4}

%Pedestrian 1 has reached the bottom wall. State changes to 4 and going right

rule : 4 0 {(0,0)=2 and (statecount(5)>=1)}

% Pedestrian has reached to bottom wall (1,0) and cell to the left is empty

rule : 0 100 {(0,0)=4 and (0,-1)=0 and (1,0)=5}

% Pedestrian is in current state 0 and person the right is in state 4 with wall at the bottom (bottom to left bottom)

rule : 4 100 {(0,0)=0 and (0,1)=4 and (1,0)=5}

%Pedestrian has reached the left bottom wall (0,-1) and cell above is empty

rule : 0 100 {(0,0)=4 and (-1,0)=0 and (0,-1)=5}

%Pedestrian is in state 0 and cell to the bottom is in state 4 with wall at the left bottom (Left bottom to left top)

rule : 4 100 {(0,0)=0 and (1,0)=4 and (0,-1)=5}

%Pedestrian has reached the left top wall (-1,0) and cell to the right is empty

rule : 0 100 {(0,0)=4 and (0,1)=0 and (-1,0)=5}

%Pedestrian is in state 0 and the cell to the right is in state 4 with wall the left top (Left top to right top)

rule : 4 100 {(0,0)=0 and (0,-1)=4 and (-1,0)=5}

%Pedestrian has reached the right top and the cell below is empty

rule : 0 100 {(0,0)=4 and (1,0)=0 and (0,1)=5}

%Pedestrian is in current state 0, the top cell is in state 4 with wall at the right top (Right top to right bottom)

rule : 4 100 {(0,0)=0 and (-1,0)=4 and (0,1)=5}

%Prevent collisions in the open area of the cells

%Pedestrian is in current state 2(down) and a cell is in state 1(up), dont change the position

rule : 0 100 {(0,0)=2 and (1,0)=1}

%Pedestrian is in current state 0, the cell to top right is in state 2(down) and either of right and bottom cell is in state 1(up)

rule : 2 100 {(0,0)=0 and (-1,1)=2 and ((0,1)=1 or (1,1)=1)}

%avoid clash in bottom right to bottom left

rule : 4 100 {(0,0)=3 and (0,1)=4 and (1,0)=5}

rule : 3 100 {(0,0)=0 and (0,-1)=3 and (0,1)=4 and (-1,-1)=0 and (-1,0)!=2 and (1,0)=5}

**%New rules are added to handle collisions around the walls**

**%avoid clash in Bottom Left to Top Left**

**rule : 4 100 {(0,0)=3 and (1,0)=4 and (0,-1)=5}**

**rule : 4 100 {(0,0)=0 and (-1,0)=3 and (1,0)=4 and (0,-1)=5}**

**%avoid clash in the Top Left to Top Right**

**rule : 4 100 {(0,0)=3 and (0,-1)=4 and (-1,0)=5}**

**rule : 4 100 {(0,0)=0 and (0,1)=3 and (0,-1)=4 and (-1,0)=5}**

**%avoid clash in the Top Right to Bottom Right**

**rule : 4 100 {(0,0)=3 and (-1,0)=4 and (0,1)=5}**

**rule : 4 100 {(0,0)=0 and (1,0)=3 and (-1,0)=4 and (0,1)=5}**

% Default - don't move

rule : {(0,0)} 100 {t}

**Implementation of Cell DEVS using Cadmium Simulator-**

Each cell Ci ∈ C of the scenario is formally defined as follows:

Ci = 〈Xi^N, Xi^E,Yi, Si, Ni, τi, di, delayi〉,

where:

• Xi^N is the neighborhood input set.

• Xi^E is the external input set.

• Yi is the cell output set.

• Si is the cell state set.

> **{cellstates : 1,2,3,4}**

• Ni is the cell neighborhood set.

>**neighborhood unordered\_map<cell\_id C, vicinity V>**

• τi: Si × Ni × Xi → Si is the local computation function of the cell. When triggered,

it computes the new state of cell Ci from its previous state, the neighborhood set,

and inputs.

>**localComputation()**

• di: Si → ℝ≥0 is the cell delay function.

> **Inertial Delay Buffer:**  **OutputDelay(PedestrianCell state)**

inertial(qi, si′, di(si′)) = {〈si′, di(si′)〉}.

Diagram

Description automatically generated

**Figure. UML Class Diagram- Pedestrian Evacuation model**

**JSON file –**

The JSON file has two main sections - "**scenario**" and "**cells**".

The "**scenario**" section describes the properties of the simulation environment. It includes the "shape" of the grid, which in this case is a 5x5 grid, the "origin" of the grid (0,0), and whether the grid is "wrapped" or not, which means whether the edges of the grid wrap around to the other side. For the project, “wrapped” is set to true.

The "**cells**" section contains information about the individual cells in the grid. In this case, there is only one type of cell defined, called "default". The "default" cell has several properties:

"**delay**": This specifies the delay behavior of the cell, which in this case is "inertial". The inertial

delay function presents a preemptive behavior, as the output queue can only contain one message at a given time. Therefore, if the cell changes its state, all the previously scheduled messages are dropped from the queue.

"**cell\_type**": This specifies the type of the cell, which is "pedestrianEvacuation".

"**state**": This defines the initial state of the cell. It includes the current "cellstate" (which in this case is set to 2), the cell's "x" and "y" coordinates on the grid, the locations of the exits that the pedestrians will try to reach, the location of the walls that pedestrians will avoid, and whether the cell has reached an exit or not.

The "**neighborhood**" property of the cell defines the neighborhood of the cell, which determines which other cells are considered neighbors of this cell.

In this case, there are two neighborhoods defined - a "moore" neighborhood with a vicinity of 1 and a range of 1, which includes all cells within one cell distance of the current cell, and a "relative" neighborhood with a range of 1 and a single neighbor at (0,0), which means only the current cell itself will be considered a neighbor.

**State.hpp –**

This code defines the state of the benchmark cell in a pedestrian evacuation simulation using the Cell-DEVS modeling and simulation framework.

The PedestrianCell struct defines the state of the cell and includes the following properties:

**cellstate**: The state of the cell, which in this case is an integer value.

**x and y**: The coordinates of the cell in the grid.

**reached\_exit**: A flag indicating whether the cell has reached an exit or not.

**exit\_locations**: A vector of vectors containing the locations of the exits that pedestrians will try to reach.

**wall**: A vector of vectors containing the locations of the walls that pedestrians will avoid.

The PedestrianCell struct also includes a default constructor that initializes the state of the cell to some default values.

If a value is not specified in the JSON file, the default values will be used.

The operator != function is defined to compare two PedestrianCell objects and detect any change. This is required for detecting state changes during the simulation.

The operator << function is defined to print the state of the PedestrianCell object.

The from\_json function is defined to parse a JSON file and generate the corresponding PedestrianCell object. This function takes a JSON object j and a PedestrianCell object s as input. It uses the get\_to function of the json library to extract the values of the different properties from the JSON object and set them in the PedestrianCell object. The **[[maybe\_unused]]** attribute is used to indicate that the from\_json function might not be used in all cases, and to suppress any warnings related to this.

**pedestrianEvacuation\_grid\_cell.hpp –**

This code defines a synthetic cell class for a pedestrian evacuation simulation using the Cell-DEVS modeling and simulation framework.

The GridPedestrianCell class inherits from the GridCell class and represents a single cell in the grid. It includes several properties:

**cellstate**: A double value representing the state of the cell.

**exit\_locations**: A double value representing the locations of the exits that pedestrians will try to reach.

**wall**: A double value representing the locations of the walls that pedestrians will avoid.

**x and y**: The coordinates of the cell in the grid.

The **localComputation** function is defined to compute the new state of the cell based on its current state and the states of its neighboring cells. It switches on the current cellstate and updates the position of the cell based on its direction of movement. If the cell reaches an exit, its reached\_exit flag is set to true and its cellstate is set to 0 (which represents a cell that has exited the simulation). If the cell hits a wall, its cellstate is updated to reflect its new direction of movement.

The **hasReachedExit** function is defined to check if a pedestrian has reached an exit. It loops through all of the exit\_locations and checks if the cell's x and y coordinates match the coordinates of any of the exits.

The **hasHitWall** function is defined to check if a cell has hit a wall. It loops through all of the wall locations and checks if the cell's x and y coordinates match the coordinates of any of the walls.

The **outputDelay** function is defined to return the output delay of the cell, which in this case is a constant value of 1.

Overall, this code defines the behavior of a simple pedestrian in a pedestrian evacuation simulation, where the pedestrian moves towards an exit and turns left or right if it encounters a wall.

**main\_grid\_pedestrianEvacuation.cpp –**

This code defines the main function of a pedestrian evacuation simulation using the Cell-DEVS modeling and simulation framework.

The program takes two command line arguments:

**SCENARIO\_CONFIG.json**: A JSON file that contains the configuration for the simulation scenario.

**MAX\_SIMULATION\_TIME**: An optional argument that specifies the maximum simulation time in time units. If this argument is not provided, the default value is 500 time units.

The main function first checks that the correct number of arguments have been provided, and then reads the file path and simulation time from the command line arguments.

The **addGridCell** function is defined to create a new GridPedestrianCell object based on the cellModel specified in the configuration. If the **cellModel** is not recognized, a bad\_typeid exception is thrown.

The main function then creates a **GridCellDEVSCoupled** object with the name "pedestrianEvacuation", the addGridCell function, and the configuration file path. The **buildModel** function is called to construct the model.

The **RootCoordinator** object is created with the GridCellDEVSCoupled object as an argument. A **CSVLogger** object is created to log the simulation data to a CSV file named "**pedestrian\_grid\_log.csv**" with ; as the delimiter. The logger is set for the RootCoordinator using the setLogger function.

The simulation is started using the start function of the **RootCoordinator**. The simulate function is called to run the simulation for the specified simulation time. Finally, the **stop()** function is called to stop the simulation.

The execution time for each step of the simulation is also recorded using the chrono library and printed to the console.

# Simulation Results

# Addition of new rules:

# *Handling collisions around walls during evacuation –*

Figures below have 1 evacuation path and consist of 16 pedestrians, 4 moving upwards [Red cells], 4 moving downwards [Green cells], 4 up walkers who have reached the wall and following it[Yellow cells], 3 down walkers who have reached the wall and following it[Blue cell].

# Chart, waterfall chart Description automatically generated Chart, waterfall chart Description automatically generated

# Chart, waterfall chart Description automatically generatedChart, waterfall chart Description automatically generated Chart, waterfall chart Description automatically generated Chart, waterfall chart Description automatically generated

The figures demonstrate that during the evacuation, there is a collision between the yellow and blue cells on the three sides of the wall(from bottom left to right, top left to right and top right to bottom right positions). To handle such collisions, yellow cells turn to blue cells and follow the path with the blue cell.

Chart, waterfall chart

Description automatically generated Chart

Description automatically generated with low confidence

From the last figure, it is understood that cells (Pedestrians) moved smoothly and naturally throughout the simulations, indicating the effectiveness of the rules

**Draw File Logs –**

The draw file shows different stages of pedestrian evacuation in states 1,2, 3 and 4:

Chart

Description automatically generated

Graphical user interface, chart

Description automatically generated

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated with medium confidence

# Based on the results, the simulation was completed at 00:00:07:600 with 16 pedestrians in a room that had only one exit for evacuation.

# Results with Cadmium Simulator:

1. **Test Scenario 1 –** This JSON file describes a pedestrian evacuation scenario where pedestrian cells have a cellstate of 2. Pedestrians with cellstate 2 are programmed to move downwards in the grid and to move towards the right if they encounter a wall, continuing in this direction until they reach an exit. The simulation takes place on a grid with wrapped edges, meaning that pedestrians can continue moving from one edge of the grid to the opposite edge. The state of each pedestrian cell is defined by its current location, whether it has reached an exit yet, and the locations of the exits and walls in the grid. The neighborhood of each pedestrian cell is defined by a Moore neighborhood and a relative neighborhood with a range of 1, which includes the pedestrian's own cell.

**JSON File –**

# Text Description automatically generated

**CSV Log File –**

# time;model\_id;model\_name;port\_name;data

# 0;1;(3,4);;<2, 5, 0, 0>

# 0;2;(4,3);;<2, 5, 0, 0>

# …

# 0;25;(1,2);;<2, 5, 0, 0>

# 0;1;(3,4);outputNeighborhood;<2, 5, 0, 0>

# 0;1;(3,4);;<2, 5, 1, 0>

# ….

# 1;10;(3,0);outputNeighborhood;<2, 5, 1, 0>

# 1;10;(3,0);;<2, 5, 2, 0>

# 1;11;(1,1);outputNeighborhood;<2, 5, 1, 0>

# 1;11;(1,1);;<2, 5, 2, 0>

# ……..

# 3;25;(1,2);;<2, 5, 4, 0>

# 4;1;(3,4);outputNeighborhood;<2, 5, 4, 0>

# 4;1;(3,4);;<4, 5, 5, 0> // at [5,5], there is a wall, so cellstate 2 is changed to 4

# 4;2;(4,3);outputNeighborhood;<2, 5, 4, 0>

# 4;2;(4,3);;<4, 5, 5, 0>

# ………

# 4;25;(1,2);outputNeighborhood;<2, 5, 4, 0>

# 4;25;(1,2);;<4, 5, 5, 0>

# 5;1;(3,4);outputNeighborhood;<4, 5, 5, 0>

# 5;1;(3,4);;<2, 6, 5, 0>

# 5;2;(4,3);outputNeighborhood;<4, 5, 5, 0>

# 5;2;(4,3);;<2, 6, 5, 0>

# ……

# 9;24;(3,2);;<2, 6, 9, 0>

# 9;25;(1,2);outputNeighborhood;<2, 6, 8, 0>

# 9;25;(1,2);;<2, 6, 9, 0>

# 10;1;(3,4);outputNeighborhood;<2, 6, 9, 0>

# 10;1;(3,4);;<0, 6, 10, 1> // at [6,10], there is an exit, so cellstate is changed to 0 {Pedestrian is evacuated)

# 10;2;(4,3);outputNeighborhood;<2, 6, 9, 0>

# 10;2;(4,3);;<0, 6, 10, 1>

# …

# 11;1;(3,4);;<0, 6, 10, 1>

# ….

# 11;25;(1,2);;<0, 6, 10, 1>

1. **Test Scenario 2 –** In this scenario, the pedestrian cells with cellstate 1 will move upwards in the grid and will move towards the left if they encounter a wall, continuing in this direction until they reach an exit.

**JSON File –**

Text

Description automatically generated

**CSV Log File –**

time;model\_id;model\_name;port\_name;data

0;1;(3,4);;<1, 5, 8, 0>

0;2;(4,3);;<1, 5, 8, 0>

…

1;25;(1,2);;<1, 5, 6, 0>

2;1;(3,4);outputNeighborhood;<1, 5, 6, 0>

2;1;(3,4);;<**3, 5, 5, 0**> //at [5,5] there is a wall, so the cellstate is changed from 1 to 3

2;2;(4,3);outputNeighborhood;<1, 5, 6, 0>

2;2;(4,3);;<3, 5, 5, 0>

..

2;25;(1,2);outputNeighborhood;<1, 5, 6, 0>

2;25;(1,2);;<3, 5, 5, 0>

3;1;(3,4);outputNeighborhood;<3, 5, 5, 0>

3;1;(3,4);;<**1, 4, 5, 0**> // continued moving upwards after turning left

3;2;(4,3);outputNeighborhood;<3, 5, 5, 0>

3;2;(4,3);;<1, 4, 5, 0>

..

3;25;(1,2);outputNeighborhood;<3, 5, 5, 0>

3;25;(1,2);;<1, 4, 5, 0>

4;1;(3,4);outputNeighborhood;<1, 4, 5, 0>

4;1;(3,4);;<**0, 4, 4, 1**> // at [4,4] there is an exit, so cellstate is changed to 0 {Pedestrian is evacuated}

4;2;(4,3);outputNeighborhood;<1, 4, 5, 0>

..

5;25;(1,2);;<0, 4, 4, 1>

# Conclusion

The cadmium simulator and Cell DEVS are simulation tools that can be used to model and analyze pedestrian evacuation strategies in each scenario. The Cadmium simulator is a discrete-event simulation framework that provides a modular and hierarchical modeling environment, which can help to organize and simplify the development of the model. This can make it easier to create, modify, and test different scenarios and evacuation strategies. Moreover, Cell DEVS is a modeling language that allows for the representation of complex systems as a collection of discrete, interconnected cells. This allows the modeling of each pedestrian's behavior and movement in a detailed manner, while still being able to simulate the entire system.

Using the Cadmium simulator and Cell DEVS, we modeled the movement of pedestrians in the evacuation grid and analyzed various evacuation strategies. For example, we simulated different exit routes and evaluated the time it takes for all pedestrians to evacuate, and we analyzed the effect of different obstacles or barriers on the evacuation process.

The benefit of using these simulation tools is that it allows you to test and evaluate different evacuation strategies without having to physically implement them. This can save time and resources, as well as potentially identify issues or problems with the strategy before it is implemented in real life. Additionally, using simulation tools allows for a more detailed analysis of the evacuation process, such as the movement of individual pedestrians, which can help improve the efficiency and safety of the evacuation plan.

Overall, using a Cadmium simulator and Cell DEVS can provide a powerful tool for designing and testing evacuation strategies in a controlled, virtual environment. This can help to improve the safety and effectiveness of real-world evacuation procedures in the future.

# Reference

[1] An extended cost potential field cellular automaton model for pedestrian evacuation considering the restriction of visual field, 2019. By Xingli Li, Fang Guo, Hua Kuang, Zhongfei Geng, Yanhong Fan.

# 