

CrowdEgress: A Multi-Agent Simulation Platform for Pedestrian Crowd

Peng Wang Xiaoda Wang Peter Luh

Neal Olderman Christian Wilkie Timo Korhonen

This manual introduces a simulation tool to study complex crowd behavior in social context. The agent-based model is extended based on the well-known social force model, and it mainly describes how agents interact with each other, and also with surrounding facilities such as walls, doors and exits. The simulation platform is compatible to FDS+Evac, and the input data in FDS+Evac could be imported into our simulation platform to create single-floor compartment geometry, and a flow solver is used to generate the road map towards exits. Most importantly, we plan to integrate advanced social and psychological theory into our simulation platform, especially investigating human behavior in emergency evacuation, such as pre-evacuation behavior, exit-selection activities, social group and herding effect and so forth.

1. Introduction

The agent-based model (ABM) is a computational research method to study social systems. This model-driven approach partly origins from statistical physics, investigating how individuals in free space or in lattice interact with each other and whether there is any converged pattern emerged at the macroscopic level. Since a society composed of many people is a typical system of many-particle, it is possible to apply the principles of statistical physics to study social behavior of many individuals. Recently, there has been a growing interest in using agent-based model and simulation to understand social phenomena such as economic market or political opinions [Peralta, Kertesz, Iniguez, 2022; Quang et. al., 2018].

The agent-based model refers to a system of many-particle that exhibits emergent characteristics when autonomous agents interact with each other. Basically, the ABM consists of agent, system space, and external environment. The agent is autonomous and decides his/her behavior by interacting with the neighbors or the external environment with the rules of behavior. In our simulation platform, for example, the system space is a 2D planar space, and the environment is given as a structural layout that consists of obstructions (e.g., walls) and passageways (e.g., doors or exits), and other environmental stimuli may be imported such as gas temperature or smoke density in the future. Agents are interacting with each other and moving within this structural layout.

The simulation is mainly implemented by a component as packed in a class called simulation class (simulation.py), and it computes interaction of agents with surrounding entities including walls, doors and exits. The agent model is described in agent.py, whereas walls, doors and exits are coded in obst.py. The agent-based model is an extension of the well-known social force model (Helbing and Monlar 1995; Helbing, Farkas, Vicsek 2000; Lakoba, Kaup

and Finkelstein 2005). This model has been applied in many existing pedestrian simulators such as PTV Viswalk, MassMotion, FDS+Evac and so forth [Santos and Aguirre 2005; Ronchi, R. Lovreglio, M. Kinsey, 2020]. The model aims at investigating prototypes of pedestrian behavior in crowd evacuation. The core algorithm is still being developed and improved. This is an inter-discipline study topic, which refers to Newton particles, statistical physics, dynamic systems as well as social and behavioral science. Your contributions or comments are much welcome. The program source code and numerical test cases are mainly uploaded online at <https://github.com/godisreal/crowdEgress>.

The program also consists of several functional components such as User Interface and Data/Visualization Tool.

User Interface: The user interface is written in tkinter in ui.py. Please run ui.py to enable a graphic user interface (GUI) where one selects the input files, initialize compartment geometry, and configure or start a simulation. An alternative method is using main.py to directly start a simulation without GUI. Currently there is a simple version of GUI and it needs to be improved in several aspects.

Data Tool: This component is packed in data_func.py, and it reads in data from input files, and write data to output files. The input data is written by users in either csv files or fds input files [McGrattan et. al., 2021]. Agents must be specified in csv file while walls, doors or exits can be described either in csv file or read from standard fds input file. The simulation output is written into a binary file, which is compatible to the fds output data (fds prt5 data format). In the future we plan to visualize such data by smokeview [Forney, 2022], which is the standard tool to visualize fds output data.

Visualization Tool: The visualization component is packed in draw_func.py and currently pygame (SDL for python) is used to develop this component. Users can select to visualize the simulation as it runs, or visualize the output data after the simulation is complete. If anyone is interested, please feel free to extend the module or try other graphic libraries to write a visulization component.

2. About Simulation Models

Agent-based model (ABM) describes interactions among individual agents and their surroundings. In the simulation there are four types of entities: agents, walls, doors and exits. As below we will introduce how to specify these entities.

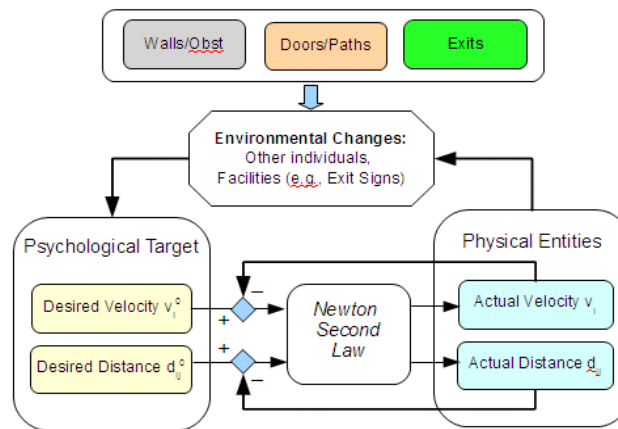


Figure 1. Agent-based model with Environmental Facilities.

The walls, doors and exits are alternatively specified by FDS input files. Users are welcome to use existing FDS input files to create compartment geometries. In current version only one-

floor crowd simulation is supported. So if there are multiple evacuation meshes in FDS input files, they should all belong to the same z interval in the vertical direction (z axis). By using FDS input files the walls are created by &OBST, and the doors are specified by &HOLE or &DOOR. The exits are obtained from &EXIT in FDS input files. If users want to find more about how FDS define a compartment area, please refer to FDS UserGuide for more information [McGrattan et. al., 2021]. If users do not use FDS input files, the above entities can alternatively be specified by using csv files as introduced below, and users need to write data blocks in csv file, and such data blocks are identified by &Door, &Exit, &Wall and &Ped as the first element in the data block.

Walls: Walls are obstruction in a compartment geometry that confine agent movement, and they set up the boundary of a room or certain space that agents cannot go through. &Wall is the identifier for the data block of walls. In other words, &Wall claims that this data block describes walls in the simulation, and thus &Wall should be written as the first element in the data block, namely, the most upper left element in the array-like data sheet. Please refer to examples for more details.

In our program walls are either lines or rectangular areas. If any users are interested, please feel free to extend the wall types to circular or polyangular areas. If users import walls from a FDS input file, the walls are created as a rectangular type and it corresponds to &OBST in FDS input file. If users specify a line obstruction, it is expected to input the position of starting point and ending point of a line. If users specify a rectangular obstruction, it is expected to input the diagonal position of two points to shape a rectangular area.

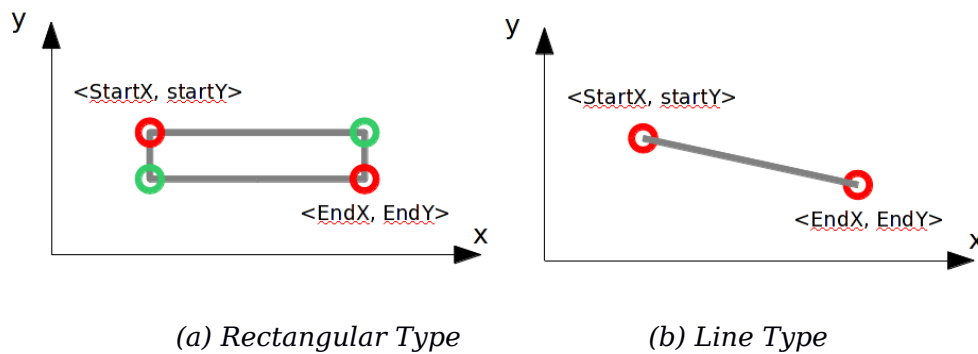


Figure 2. Create Walls in Rectangular Type or Line Type.

<startX, startY>: One diagonal point for rectangular obstruction; Or starting point for line obstruction.

<endX, endY>: The other diagonal point for rectangular obstruction; Or ending point for line obstruction.

<arrow>: Direction assigned to the obstruction so that agents will be guided when seeing this obstruction, especially when they do not have any target door or exit. The direction implies if the obstruction provides evacuees with any egress information such as exit signs on the walls or not. The value could be +1 for positive x direction, -1 for negative x direction, +2 for positive y direction and -2 for negative y direction. If no direction is given, the value is 0.

<id>: id number assigned to this obstruction; id number is optionally shown in the pygame screen so that users can easily modify the obstruction.

<inComp>: a boolean variable to indicate if the obstruction is in computation loop or not. Normally it is given true/1. If users want to quickly remove a obstruction in simulation, it is assigned be to false/0.

<mode>: Either rectangular or line obstruction in current program; the default mode is rectangular model.

Table1. Data Array of Wall

&Wall	0/startX	1/startY	2/endX	3/endY	4/arrow	5/id	6/inComp	7/mode
Wall Down	0	0	10	0.5	0	0	1	rect
Wall Top	0	9.5	10	10	0	1	1	rect
WallLeft	0	0	0.5	10	0	2	1	rect
WallRight	9.5	0	10	10	0	3	1	rect

Doors and Exit: Doors are passageways that direct agents toward certain areas, and they may be placed over a wall so that agents can get through the wall by the door. Doors can also be placed as a waypoint if not attached to any walls, and they can be considered as arrows or markers on the ground that guide agent egress movement. In brief doors affect agent way-finding activities and they help agents to form a roadmap to exits. In current program doors are only specified as rectangular object, and the data block of doors is claimed by &Door, which should be written as the first element in the data block. Please refer to examples for more details.

Exits are a special types of doors which represent paths to the safety. Thus they may be deemed as final path to reach safety areas, and computation of an agent is complete when the agent reaches an exit. An exit is usually placed over a wall like doors, but it can also be put anywhere independently without walls. In the program exits are only defined as rectangular areas. The data block of exits are claimed by &Exit, which should be written as the first element in the data block. Please refer to examples for more details. The specific features of doors and exits are given as below.

<startX, startY>: One diagonal point for rectangular door/ exit.

<endX, endY>: The other diagonal point for rectangular door/exit.

<arrow>: Direction assigned to the door or exit so that agents will be guided when seeing this entity, especially when they do not have any target door or exit. The direction implies if the door or exit provides evacuees with any egress information such as exit signs or not. The value could be +1 for positive x direction, -1 for negative x direction, +2 for positive y direction and -2 for negative y direction. If no direction is given, the value is zero. Please refer to FDS+Evac manual to better understand the direction setting.

<id>: id number assigned to the door/exit; id number is optionally shown in the pygame screen so that users can easily identify the door information.

<inComp>: a boolean variable to indicate if the door/exit is in computation loop or not. Normally it is given true/1. If users want to quickly remove a door/exit in simulation, they could assign it be to false/0 for a quick test.

<exitSign>: a boolean variable to indicate if the door/exit is attached with an exit sign or not. If there is an exit sign the boolean variable is given true/1. Actually it is not that useful in existing door selection algorithm. So users may omit this feature currently.

Table2. Data Array of Door

&Door	0/startX	1/startY	2/endX	3/endY	4/arrow	5/id	6/inComp
Door Down	4.49	-0.1	5.51	0.6	0	0	1
Door Top	4.49	9.4	5.51	10.1	0	1	1

Table3. Data Array of Exit

&Exit	0/startX	1/startY	2/endX	3/endY	4/arrow	5/id	6/inComp
Exit Down	4.5	-0.3	5.5	0.3	0	0	1
Exit Top	4.5	9.7	5.5	10.3	0	1	1
Exit Right	9.3	4.3	10.3	5.9	0	2	1

Agents: Finally and most importantly, agents are the core entity in computation process. They interact with each other to form collective behavior of crowd. They also interact with above types of entities to form egress motion toward exits. The resulting program is essentially a multi-agent simulation of pedestrian crowd. Each agent is modeled by extending the well-known social force model. The model is further advanced by integrating several features including pre-evacuation behavior, group behavior, way-finding behavior and so forth. In current program the data block of agents are claimed by &Ped or &Agent, and they are written as the first element in the data block.

<InitalX, InitialY>: Initial position of an agent in 2D planar space.

<DestX, DestY>: Destination position in 2D planar space. This value is almost not used in current computational loop because the destination position is automatically determined by the exit selection algorithm. When the exit is selected by an agent, the destination position is given by the exit position.

<interRange>: The range when agents have herding effect, which means they may exchange opinions by talking.

<tau>: Tau parameter in the social force model, or as usually called relaxation time in many-particle systems or statistical physics, and it critically affects how fast the actual velocity converges to the desired velocity.

<tpre>: Time period for pre-evacuation stage. Within this time period agents do not select and move towards an exit.

<pD>: Parameter p in opinion dynamics, and it indicates how an agent's opinion/decision is impacted by surrounding others, and it critically affects herding effect in collective behavior. The measurement of this parameter is within [0, 1], and an agent's opinion/decision completely follow others if pD=1.

<pDMode>: This parameter affects how parameter p is dynamically changing. Currently it is not used in computational loop.

<aType>: The type of way-finding behaviors. Some agents may actively search for exits while others may just follow the crowd. In current simulation all agents follow the egress flow field, and thus this parameter is not actually used in existing version of code.

<ID>: ID number assigned to this agent. ID number is optionally shown in the pygame screen.

<inComp>: a boolean variable to indicate if the agent is in computation loop or not. Normally it is given true. If users want to remove an agent in simulation, they could assign it be to false for test.

<mass>: The mass of agents.

Table4. Data Array of Agents

	00/ &Ped	01/ IniX	02/ IniY	03/ DestX	04/ DestY	05/ tau	06/ tpre	07/ p/p	08/ pMode	09/ inter Range	10/ ID	11/ inComp	...
Ped0		6.33	2	66	23	0.3	3	0	fixed	30	0	1	...
Ped1		7.33	3	66	23	0.6	2	0.6	random	31	1	1	...

Important Notice:

(1) The sequence of &Wall &Door &Exit &Agent could be arbitrarily changed within an input file. Users may either first specify walls, or doors or agents. However, data feature inside a data array could not be changed in sequence. For example, you must first give <startX>, and next <startY> for walls, doors and exit, and cannot change the sequence to be <startY> and <startX>, or move <endX> or <endY> before them. All the features for walls, doors, exits and agents are read in certain order, and such order cannot be altered in current version (version 2.2). New contributors are welcome to improve this functional setting in

data_func.py such that this sequence can also be adjusted by users.

(2) As above we exemplify all the input data by table-like data sheet, and users can select any table processing software such as Excel or GNUmeric to show such data. However, please remember that csv (comma separated values) is the basic format of the above data file, namely all the data cell in the input file are separated by comma. Those comma are omitted in Excel or GNUmeric, but are well shown in any text editor.

Exit Selection Probability: Way finding refers to how individual agents orientate themselves towards exits, especially within a multi-compartment layout. Several factors influence their decision making in way-finding activities such as guidance information (e.g., exit signs) or other individuals' choice.

If users select the default setup of using the nearest-exit strategy, it implies that each agent will select the exit nearest to their current locations, and no exit-selection probability is much involved. This is indicated as solver 1 in our program, and it is the basic setup for a multi-agent simulation.

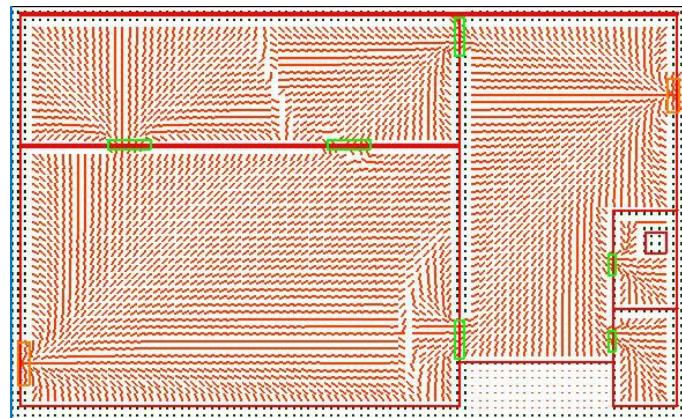


Figure 3. Flow field of using the nearest exit.

Another solver deals with a more complicated scenario, where each individual's choice is determined by exit-selection probability distribution. This process is indicated as solver 2 in our program, which can be seen as the result of a hierarchical decision making process entailing two stages: (1) tactical choice of exits and routes; and (2) operational short range choices concerning the interaction with obstacles and other evacuees [Lovreglio, Fonzone, and dell'Olio, 2016]. In other words we suppose that there are several candidate exits known by evacuees and simulate how they select one among these exits as well as choose a proper way to reach them.

The tactical choice is described by a data array as exemplified above, where each row corresponds to each agent and the column refers to exit selection probability. The entire data array is claimed by &Ped2Exit or &Agent2Exit, which is written in the portion of the first element. All the agents select exits base on the discrete probability distribution as described by the array.

Table 1. Data Array of Exit-Selection Probability

&Ped2Exit	exit0	exit1	exit2
agent0	0.7	0.2	0.1
agent1	0.3	0.3	0.1

The operational choice is to generate the evacuation route based on the exit selected. There are a number of methods used in existing egress simulators. Our algorithm is partly learned from FDS+Evac, where the route is calculated by a two-dimensional flow solver. The computation result is a 2D flow field that guides agents to the exit selected. The flow field can be better explained as a social field related to social norms or other behavioral characteristics (Lewin, 1951; Helbing et. al., 2000; Wang, 2022), and we will further elaborate the idea in future. In this algorithm each exit is a sink point, and solver calculates the route as the crowd flow move to the sink (Korhonen et. al., 2008; Korhonen, 2016). The detailed discussion of the flow solver will not be included in this article, but we emphasize that this method is more suitable to describe human collective behavior on the background of social science and psychological theory.

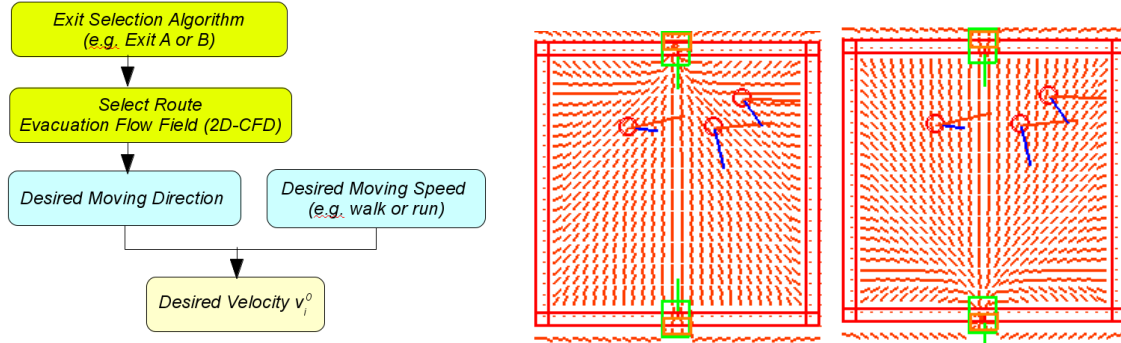


Figure 4. Simulation of crowd egress with exit-selection algorithm.

In the above setting of solver2 the exit-selection probability may be time-invariant for each agent. A more realistic and complicated scenario assumes that each individual's choice is made by integrating three major factors: (1) historical knowledge of building layout or prior information such as habit of using different path; (2) Judgment on current situation in egress (e.g., the distance to an exit) (3) Decisions of other people. Because each individual's decision is interacting with others, the collective decision-making is critically studies, and several social and economic models are extended to describe this process with integration of various social-psychological findings in evacuation research [Ozel, 2001; Staal, 2004; Santos and Aguirre 2005]. In fully describe this scenario users should well define parameter $\langle pD \rangle$ for each agent in csv file. As a result, the data array claimed by $\&agent2exit$ only gives the initial value of exit-selection probability distribution, and $\langle pD \rangle$ critically determines how it evolves dynamically in the simulation.

3. How-To

As above we briefly introduce how to write csv file to describe agents, walls, doors and exits. In this section we will explain how to use GUI (tkinter window) and set up the simulation parameters.

In Tkinter Screen:

When tkinter window (GUI) is activated, users will see several tabs including $\langle QuickStart \rangle$, $\langle Parameters \rangle$ and so forth. Next, we will briefly introduce how to set up the simulation in the tabs.

The default tab is called $\langle QuickStart \rangle$. In this tab users select the input files to get a quick run of the simulation. Choose csv file to specify agent data. Users can optionally use fds file to create the compartment geometry, and the pedestrian features are described in csv file. If both csv and fds files are presented, the compartment structure will be created by fds file. If fds file is omitted, the compartment geometry should be described in csv file. The agents must

be specified in csv file currently while the walls, doors and exits can either described by csv file or fds file. Please take a look at the examples for details.

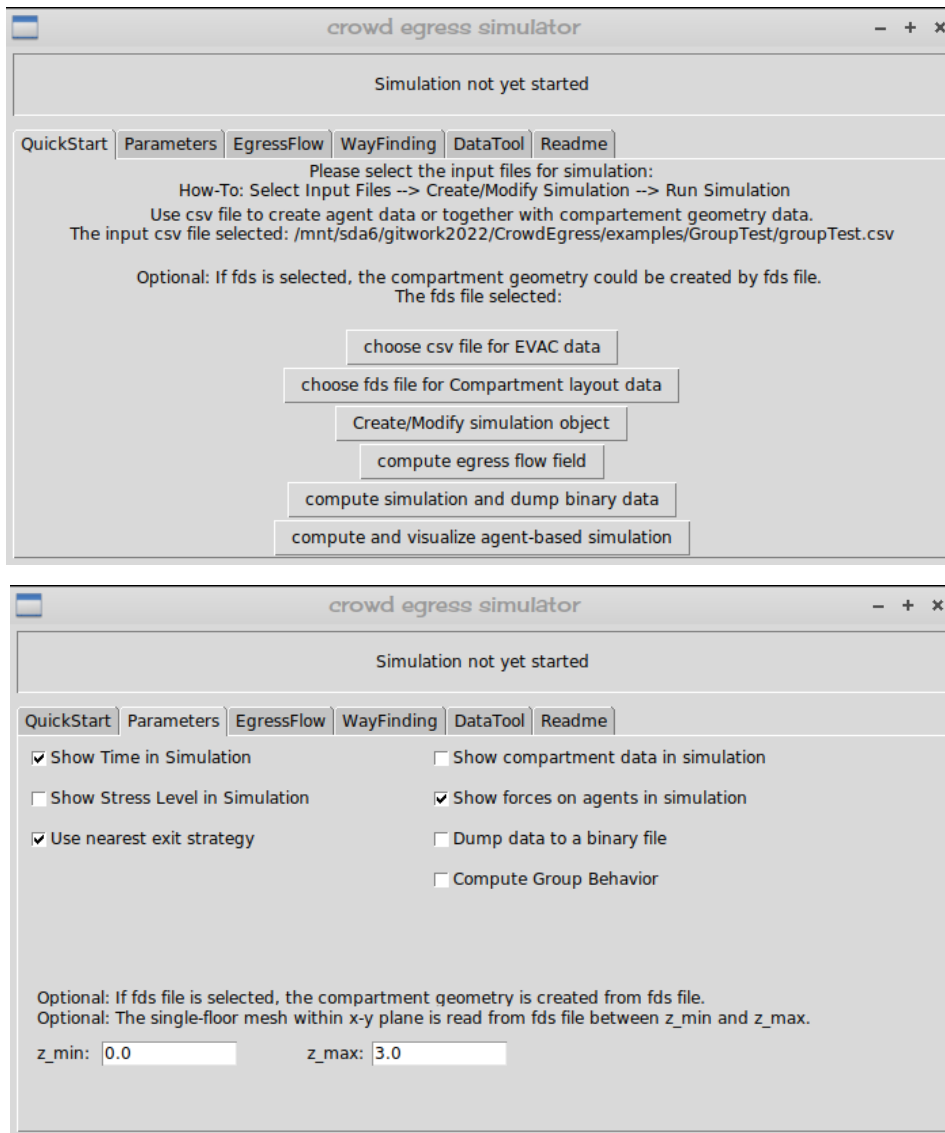


Figure 5. User Interface of CrowdEgress

After the input files are well selected, it is suggested that users first create the simulation object to see if the structural layout is created as expected. Another screen is next displayed by pygame, where users could check all the entities in the simulation, such as wall, doors, exits and initial positions of each agent. We call this screen by TestGeom in this manual, and it means testing the geometric settings of compartment layout. In TestGeom user could add walls, doors or exits, or change the initial position of each agent. We will introduce how to do such things soon later.

After the simulation object is created or modified, the next step is running the simulation. There are several options for users.

<Compute Egress Flow Field>: The first option is to compute the egress flow field and display it on the pygame screen, and agent-based simulation is not started in this option. In the pygame screen users will find how the mesh is created such as the number of x points and y points, the boundary value of mesh (i.e., min_x, max_x, min_y, max_y). Users will also see the flow field towards each exits. If solver 1 is selected, there is only one flow field computed, which is simply using the nearest-exit strategy (See Figure 3). If solver 2 is selected, there are several flow field computed, which corresponds to the roadmap towards each exit (See

Figure 4). The nearest-exit strategy is also additionally computed in solver 2. Users could press <o> and <p> in the keyboard to switch from different flow field.

<Compute Simulation and Dump Binary Data>: The second option is to compute the agent-based simulation with the flow field. The numerical result will be written into a binary file, but not displayed timely in pygame screen.

<Compute and Visualize Agent-Based Simulation>: The third option is to compute the agent-based simulation and visualize the result timely in pygame screen. This is the most common and important option such that users are able to directly observe how agents interact and move towards a selected exit in the compartment layout. Users can show the forces or movement trace of each agents, pause the simulation, and we will introduce this issue soon later.

The second tab is called Parameters, where users specify the basic parameters before the simulation starts. An important issue is that z-interval (min_z and max_z) should be specified when a FDS+Evac input is used to create the compartment geometry setting. The reason is that FDS+Evac is a 3D simulator, and its input file may include several computation meshes as several floors in a building. However, our simulator CrowdEgress is a 2D simulator only for single-floor layout. Thus, if a FDS+Evac input file is used with several compartment floors, sometimes users should check or modify the min_z and max_z to determine which floor should be computed in the simulation. The default value in CrowdEgress is given as min_z=0.0 and max_z=3.0, and it is commonly useful for most FDS+Evac file with only one compartment floor.

Other parameters are briefly introduced as below.

<Show Time in Simulation>: Show the computational time and simulation time when simulation starts.

<Show Stress Level in Simulation>: Indicate the difference between actual velocity and desired velocity. Please refers to Wang, 2021 for more details.

<Use nearest exit strategy>: Solver 1 is used if selected. Namely, each agent is guided to the nearest exit and exit-selection algorithm is not involved.

<Show compartment data in simulation>: Show the geometric data of doors and exits in the pygame screen such that users can identify each door and exit by their indice and default directions.

<Show forces on agents in simulation>: Visualize forces in the pygame screen. The forces are measured by several lines shown on each agent. The line in purple color is the wall force; line in green color represents the door force. The line in pink color is the social force.

<Dump data to a binary file>: Dump simulation data into a binary file which is compatible to fds prt5 data format. The data file is used to visualize the numerical result after simulation stops.

<Group behavior>: Use group social force in simulation. This is a little complicated issue and we will elaborate it in future.

There are other tabs including EgressFlow, WayFinding, DataTool, and the GUI is under development. Thus, we will introduce them in the future.

In Pygame Screen:

When pygame screen is displayed, press keys to adjust the display features and visualize entity data. There are several sections in pygame screen. One section is called TestGeom, where users can visualize compartment geometric settings and modify them manually.

In TestGeom users can add walls, doors or exits by selecting the corresponding items in the menu bar. Sometimes it is necessary to adjust the exit-selection probability if new exits are added in TestGeom. Namely, the number of column in data array &Agent2Exit (See Table 1) should be equal to the total number of exits.

Users can also dump geometric data (e.g., wall data and door data) into csv file by selecting

the item <OutputData> in the menu bar. The output file is created as bldDataRev.csv for any modification of the compartment geometric in TestGeom. The data can also be briefly shown in the screen by selecting the item <ShowData> in the menu bar. If users click <Simulation>, then the simulation starts and the program goes to the second section called RunSimulation.

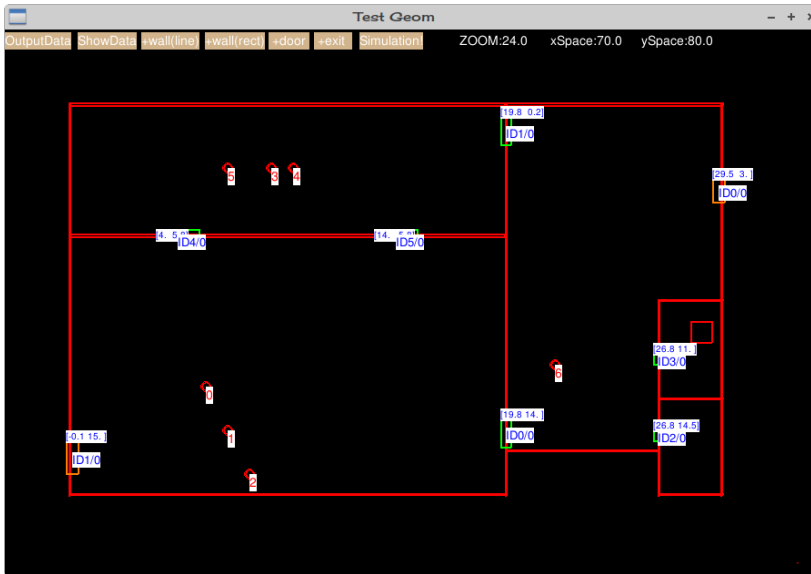


Figure 6. Pygame screen of TestGeom.

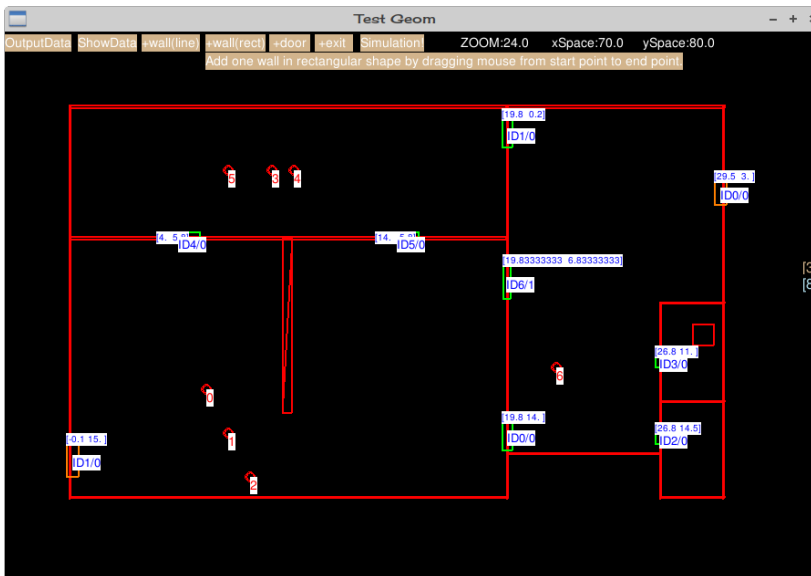


Figure 7. Add walls and doors in TestGeom.

The other section is RunSimulation, in which the agent-based simulation is started and visualized on the pygame screen. User can pause the simulation, but cannot rewind it in current version (Version 2.2). A binary data file is optionally created when the simulation runs and users can also visualize the data after the simulation stops.

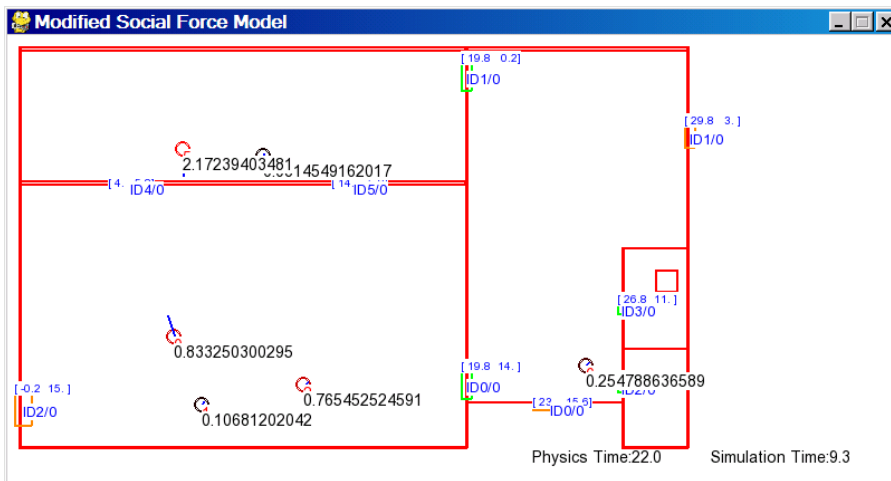


Figure 8. Agents move with stress level indicated.

In both phases of *TestGeom* and *RunSimulation*, there are hot keys defined. Use *pageup/pagedown* to zoom in or zoom out the entities in screen. Use *space* key to pause the simulation. Use *arrows* to move the entities vertically or horizontally in screen. Use *1/2/3* in number panel (Right side in the keyboard) to display the door or exit data on the screen.

Try Examples:

There are currently several examples in the repo <https://github.com/godisreal/crowdEgress>. For instance there are standard example of a single room with two exits or three exits, which are basically used to test how agents socially interact and select different exits. The example of a room with one exit is also widely used to test crowd behavior at bottlenecks such as the faster-is-slower effect (Helbing, Farkas, Vicsek, 2000).

There are also some more complicated examples. A typical one is learned and extended from *MassEgress* project (Pan, 2006), and some *FDS+Evac* test cases are also included. Users can also learn how to write the csv files from the examples. In order to run such examples it is suggested that users should first check the mesh parameters. Especially, the total number of *x* point and *y* points are to be properly given to build a mesh for flow computation. The flow solver may take some computational time to generate the egress flow field, especially if *x* point and *y* point are relatively large to generate a refined mesh.

Acknowledgments

The author is thankful to Timo Korhonen, Kerry Marsh and Vivek Kant for helpful comments on earlier work in University of Connecticut. The author appreciates the research program funded by NSF Grant # CMMI-1000495 (NSF Program Name: Building Emergency Evacuation - Innovative Modeling and Optimization). The program source code and numerical test cases are mainly included online at <https://github.com/godisreal/crowdEgress>. If you have any comment or inquiry about the testing result, please feel free to contact me at wp2204@gmail.com or start an issue on the repository.

References

- [1] D. Helbing, I. Farkas, T. Vicsek, "Simulating Dynamical Features of Escape Panic." *Nature*, Vol. 407, pp. 487- 490, 2000.
- [2] D. Helbing, P. Molnar, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, pp. 4282-4286, 1995.
- [3] G. P. Forney, "Smokeview, A Tool for Visualizing Fire Dynamics Simulation Data, Volume I: User's Guide", NIST Special Publication 1017-1 6th Edition, National Institute of Standards and Technology, Gaithersburg, MA, June 2022, 188 p.
- [4] T. Korhonen, "Technical Reference and User's Guide for Fire Dynamics Simulator with Evacuation," (FDS+Evac, FDS 6.5.3, Evac 2.5.2), VTT Technical Research Center of Finland, 2016.
- [5] T. Korhonen, S. Hostikka, S. Heliovaara, H. Ehtamo, "FDS+Evac: Modelling Social Interactions in Fire Evacuation," *Proceedings of the 4th International Conference on Pedestrian and Evacuation Dynamics*, February 27-29m 2008, Wuppertal, Germany.
- [6] T. I. Lakoba, D. J. Kaup, N. M. Finkelstein, "Modifications of the Helbing-Molnar-Farkas-Vicsek Social Force Model for Pedestrian Evolution, Simulation, Vol. 81, Issue 5, pp. 339-352, May 2005.
- [7] K. Lewin, *Field Theory in Social Science*, New York, Harper, 1951.
- [8] K. McGrattan, S. Hostikka, R. McDermott, J. Floyd, C. Weinschenk and K. Overholt, "Fire Dynamics Simulator, User's Guide", NIST Special Publication 1019 6th Ed., National Institute of Standards and Technology, Gaithersburg, MA, 2022, 367 p.
- [9] F. Ozel, "Time Pressure and Stress as a Factor During Emergency Egress," *Safety Science*, Vol. 38, pp. 95-107, 2001.
- [10] R. Lovreglio, A. Fonzone, L. dell'Olio, *A Mixed Logit Model for Predicting Exit Choice during Building Evacuations, Transportation Research Part A*, 2016. DOI: 10.1016/j.tra.2016.06.018.
- [11] X. Pan, C. S. Han, K. Dauber, and K. H. Law, "Human and Social Behavior in Computational Modeling and Analysis of Egress," *Automation in Construction*, Vol. 15, pp. 448-461, 2006.
- [12] A. F. Peralta , J. Kertesz, G. Iniguez, *Opinion dynamics in social networks: From models to data*, 2022.
- [13] L. A. Quang, N. Jung, E. S. Cho, J. H. Choi and J. W. Lee, *Agent-Based Models in Social Physics, Journal-Korean Physical Society*, 2018. DOI: 10:3938/jkps.72.1272.
- [14] E. Ronchi, R. Lovreglio, M. Kinsey, *A Survey on Evacuation Model Awareness, Usage and Users*, 2020.
- [15] G. Santos and B. E. Aguirre, *A Critical Review of Emergency Evacuation Simulation Models*, NIST Workshop on Building Occupant Movement during Fire Emergencies, June 9-10, 2004.
- [16] M. A. Staal, "Stress, Cognition, and Human Performance: A Literature Review and Conceptual Framework (NASA/TM - 204-212824)," August 2004, Hanover, MD: NASA Scientific and Technical Information Program Office.
- [17] P. Wang, "Understanding Social-Force Model in Psychological Principle of Collective Behavior," *arXiv:1605.05146v10 [physics.soc-ph]*, 2021.