International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 29, No. 1, February, 2024, pp. 83-93
Journal homepage: http://iieta.org/journals/isi

Agent-Based Simulation of Crowd Evacuation Through Complex Spaces

Mohamed Chatra[1,2*], Mustapha Bourahla[1,2]

[1] Computer Science Department, University of M'Sila, M'Sila 28200, Algeria
[2] Laboratory of Informatics and its Application of M'Sila, Computer Science Department, University of M'Sila, M'Sila 28000, Algeria

Corresponding Author Email: mohamed.chatra@univ-msila.dz

https://doi.org/10.18280/isi.290110

Received: 13 July 2023
Revised: 6 December 2023
Accepted: 31 December 2023
Available online: 27 February 2024

Keywords:
crowd evacuation behavior, agent-based model simulation, artificial intelligence, behavior animation, effectiveness, efficiency evaluation

ABSTRACT

In this paper, we have developed a description of an agent-based model for simulating the evacuation of crowds from complex physical spaces for escaping dangerous situations. The model describes a physical space containing a set of differently shaped fences, and obstacles, and an exit door. The pedestrians comprising the crowd and moving in this space in order to be evacuated are described as intelligent agents with supervised machine learning using perception-based data to perceive a particular environment differently. The description of this model is developed with the Python language where its execution represents its simulation. Before the simulation, the model can be validated using an animation written with the same language to fix possible problems in the model description. A model performance evaluation is presented using an analysis of simulation results, showing that these results are very encouraging.

1. INTRODUCTION

It has become commonplace for people to congregate in public spaces like retail malls, theme parks, and subway stations. This assembly presents difficulties for planners and administrators as well as a variety of risks to people's lives (such as trample and overcrowding). When an emergency occurs, crowds' irrational conduct brought on by fear or even panic during the evacuation process may create immeasurable losses. Because of this, worries about comprehending the dynamics of crowds in both routine and emergency evacuation situations have developed rapidly [1, 2].

Model-based simulations are often the main research methodologies used to analyze crowd evacuation instead of real-world experiments, which are highly challenging to examine crowd behavior during an emergency evacuation [3-5]. The simulation models in this case are driven by environmental information, which should be qualitative instead of quantitative.

The main elements of the models are humans, which obtain their environmental information via perception rather than measured values. Because each pedestrian perceives a particular environment differently, it is challenging to estimate the degree to which actual environmental stimuli are present. In light of this, it makes sense to model and study human actions and features using linguistic data (words) rather than numerical quantities [6-8].

In this research, we will develop a crowd evacuation model, where a pedestrian is considered as a distinct individual. This model will incorporate intricate interactions between people and their settings. The behaviors of a pedestrian are influenced by both the individual's consciousness and the environment. Given the intricate connections with the environment, it is challenging to develop a model that effectively describes and predicts a pedestrian's activities.

In this model, the perception-based data is fully utilized, as well as human experience and expertise to act sanely in the event of a crowd evacuation while taking the impact of the environment into account. The precise values of the intricate interactions with the environment, such as speeds, directions, and distances, have statistical evaluation effects on a pedestrian's behaviors.

We will classify information obtained from settings that are considered measurement-based to be perception-based using the Artificial Intelligence (AI) technique [9]. Thus, we need to create a usable model that can fully utilize perception-based data and capture the connection between environmental design and pedestrian perception.

We develop an AI-based model to achieve these objectives. a pedestrian's perceptions of their immediate surroundings are typically expressed using natural language, which is by nature ambiguous and imprecise. This AI-based approach is capable of handling the inherent imprecision and unpredictability of perception information.

The reasoning and decision-making processes of pedestrians can be articulated using a set of straightforward inference rules that have the advantages of conveniently available input data and intelligible output [10]. The innovative aspect of this study is the suggestion of a pedestrian model based on AI techniques that can effectively account for human knowledge and experience as well as pedestrian perceptions of the surrounding environment.

During the modeling process, the impacts of intricate interactions with the environment on pedestrian dynamics are taken into account qualitatively. The model uses a few straightforward inference rules to represent various factors that may have an impact on certain pedestrian motions. As an AI inference system with predetermined input and output variables, the pedestrian has two combined behaviors: avoiding obstacles and seeking the goal.

These actions are taken to direct pedestrians to travel in the direction of their objectives, choose the path with the least amount of negative energy, and avoid the front obstacles, respectively. The two behaviors are integrated using the priority approach, where the decisions of turning angle and movement speed are made at each step of the model simulation.

The structure of this work is as follows. In Section 2, related works are presented. The description of the crowd evacuation model based on artificial intelligence is presented in Section 3. Section 4 describes how to validate the proposed model using model animation and how to produce simulation results. Performance evaluation of the proposed model is presented in Section 5 by analysis of simulation results. At the end, conclusions and perspectives are given in Section 6.

2. RELATED WORKS

Various simulation models have been developed over the past few decades to analyze the dynamics of crowd evacuation in both regular and emergencies. We will organize the discussion by modeling methodologies: social force, cellular automaton, fuzzy logic, and artificial intelligence.

Helbing and Molnar [11] have suggested a social force model that views pedestrians as force-driven particles and addresses the challenge of evacuating frightened pedestrians. A cellular automaton model was used by Varas et al. [12] to mimic the evacuation of pedestrians from a room with fixed obstacles. In order to imitate an evacuation experiment conducted in a classroom with barriers, Liu et al. [13] modified the cellular automaton model.

Lattice gas models [14, 15], game theory models [16, 17], optimization-based feedback controls [18], and others are frequently used to analyze evacuation issues in public spaces. Additionally, multiple methodologies have been used to study evacuation behaviors in various circumstances, including smoke-filled road tunnels [19], burning hotels [20], high-rise buildings [21], and bio-terrorism in micro-spatial contexts [22].

These findings show that crowd behavior influence types and magnitudes vary widely with situations and associated surroundings. In the meantime, numerous useful tips for ensuring a safe evacuation are also provided by study accomplishments that have been published in the literature.

Nasir et al. [23] presented a genetic fuzzy system. Fuzzy perception and fear are ingrained in human thinking, and Dell'Orco et al. [24] proposed a behavioral model for crowd evacuation based on fuzzy logic and accounting for these aspects. Furthermore, several fuzzy inference systems are made to provide escape, egress delay, and motion direction [7, 8].

Notably, studies of pedestrian dynamical behaviors in both calm and frenzied situations have been carried out through modeling and simulation based on artificial intelligence [25, 26], whereby artificial intelligence techniques can be used to predict human spirit and perception.

To mimic and model the guiding behavior of pedestrians in constructed environments, Wang et al. [27] presented a study on pedestrian movement dynamics under emergency evacuation using machine learning. In the work presented in the study by Yao et al. [28], a reinforcement learning method is used to produce a data-driven model for crowd evacuation. Li et al. [29] have combined the techniques of deep learning and social to develop a simulation model for crowd evacuation.

Each of these approaches has advantages and disadvantages. Some of them concentrate on the quantitative component of information, but they may be limited in their ability to perceive the environment. However, various approaches have limits in depicting actual space.

3. DESCRIPTION OF CROWD EVACUATION MODEL

The description of the crowd evacuation model is based on the description of the physical space, which is composed of obstacles, borders, and an exit door (the goal), and pedestrians, which are represented by physical positions (indicated by location coordinates) and behaviours.

We use Python [30], a high-level all-purpose programming language running on the Jupyter Notebook (a web-based interactive computing platform) to develop a simulation model for analysis of crowd evacuation using artificial intelligence (the complete Python code is available upon request).

3.1 Description of physical space

We first introduce a method for representing physical space "s", which plays a central role in the modeling and simulation. The suggested pedestrian model, with a goal_width exit in the center of the wall and a scale of SpaceWidth × SpaceLength, will be used to simulate in a square hall. The physical space s is composed of borders (walls) and internal obstacles with different shapes and an exit. The following is a part of the Python code to generate a model of the physical space "s".

First, we generate the exit (goal) position by the execution of Goal, goal_position = generate_goal(goal_width), where Goal is the exit polygon (it is a rectangle). The call to the function positions = generate_positions(pnumber) will generate randomly a set of pnumber positions (points) within the space to represent the initial positions of pedestrians in s. The execution of the function obstacles = generate_obstacles(positions, nobstacles_vertexes) will generate randomly a set of obstacles, which are filled polygons, each one is specified with several vertexes. The list nobstacles_vertexes contains the vertexes number of each obstacle. The borders (which are external obstacles, with the shape of rectangles) are generated by the execution of the call to the function borders = generate_borders(goal_position, goal_width).

```
def generate_physical_space(pnumber = 200,
        nobstacles_vertexes = [], goal_width = 500):
  Goal, goal_position = generate_goal(goal_width)
  positions = generate_positions(pnumber)
  obstacles = generate_obstacles(positions,
                                nobstacles_vertexes)
  borders = generate_borders(goal_position, goal_width)
  return positions, obstacles, borders, Goal, goal_position
```

3.2 Description of pedestrians

In addition to the physical space *s*, the model is composed of a crowd of pedestrians to be evacuated from a dangerous situation. The evacuation is essentially based on the behavior of pedestrians in response to the environment. This human behavior makes our model intelligent, where the knowledge used to describe the intelligent model relies on human experience in categorizing values for distances, directions (angles), and velocities.

Their classifications result in several linguistic classes (categories), which are employed to cover the universe's conversation. For this structural intelligent model, the accuracy and time complexity rise exponentially as the class number increases. Consequently, we select multiple language classes to describe state variables, balancing correctness and computational efficiency.

From human experience, distances are represented by two categories (knowledge sets): 'Near' and 'Far'. A pedestrian will perceive any position in the physical space as a near distance if the distance from his position to that position is between 0 and 80 units, and it is a far distance if it is in the interval [100, VisualDistance].

The following Python code will build a Multi-Layer Perceptron Classifier, named *DistanceModel*, which is trained with specified data to make each pedestrian percive a distance value d by calling the method *perceiveDistance(self, d)*, where *self* is the reference to the pedestrian object.

```
Near = np.aran<wge(0, 80, 0.1)
Far = np.arange(100, VisualDistance, 0.1)
DistanceX = np.reshape(np.concatenate([Near, Far]),
                                                        (-1, 1))

DistanceY =
np.reshape(np.concatenate([np.full((len(Near), 1), [0]),
np.full((len(Far), 1), [1])]), len(Near)+len(Far), ))
DistanceModel = MLPClassifier(solver='lbfgs', alpha=1e-
5, hidden_layer_sizes=(15, ),
                  random_state=1).fit(DistanceX, DistanceY)
```

Direction angle (the angle between the pedestrian's position and the goal's (exit) position) can be perceived by the pedestrian (decision maker) as one of the classes (knowledge sets): “Zero”, “SmallPos”, “LargePos”, “SmallNeg”, or “LargeNeg” , where pedestrians are turned left or right by the commands “Neg” and “Pos”, respectively. In the same way, their intervals are defined in the following Python code.

```
Zero = np.arange(-10, 10, 1)
SmallPos = np.arange(15, 110, 1)
LargePos = np.arange(120, 180, 1)
SmallNeg = np.arange(-110, -15, 1)
LargeNeg = np.arange(-180, -120, 1)
DirectionX = np.reshape(np.concatenate([Zero,
    SmallPos, SmallNeg, LargePos, LargeNeg]), (-1, 1))
DirectionY = np.concatenate([np.full((len(Zero), 3),
                                                     [0,0,0]),
                           np.full((len(SmallPos), 3), [0,0,1]),
                           np.full((len(SmallNeg), 3), [0,1,0]),
                           np.full((len(LargePos), 3), [0,1,1]),
                           np.full((len(LargeNeg), 3), [1,0,0])])
DirectionModel = MLPClassifier(solver='lbfgs',
                 alpha=1e-5, max_iter=1000,
                 hidden_layer_sizes=(15, ),
                 random_state=1).fit(DirectionX, DirectionY)
```

A Multi-Layer Perceptron Classifier, called *DirectionModel*, is built and trained with this data, where *DirectionX* is a concatenation of all direction value intervals and *DirectionY* is the concatenation of category (class) codes.

Pedestrians can perceive the movement speed of a kind coming in the opposite direction to be one of the following categories “Stop”, “Slow” and “Fast”. A Multi-Layer Perceptron Classifier, called *SpeedModel*, is built and trained with the given data, where *SpeedX* is a concatenation of all speed value intervals and *SpeedY* is the concatenation of category (class) codes.

```
Stop = np.array([0])
Slow = np.arange(1, 20, 0.1)
Fast = np.arange(30, 50, 0.1)
SpeedX = np.reshape(np.concatenate([Stop,Slow,Fast]),
                                                        (-1, 1))
SpeedY = np.concatenate([np.full((len(Stop), 2), [0,0]),
                     np.full((len(Slow), 2), [0,1]),
                     np.full((len(Fast), 2), [1,0])])
SpeedModel=MLPClassifier(solver='lbfgs',alpha=1e-5,
                hidden_layer_sizes=(15,),
                random_state=1).fit(SpeedX,SpeedY)
```

During the simulation, in addition to the prediction functions of the direction model *DirectionModel.predict()* and the speed model *SpeedModel.predict()*, it is necessary to define the *direction_crisp()* and *speed_crisp()*functions to calculate the apparent (crisp) value of the corresponding class direction angle and velocity, respectively using the random selection method to update the pedestrian’s state.

```
def direction_crisp(d):
   if d == "Zero": return random.choice(Zero)
   if d == "SmallPos": return random.choice(SmallPos)
   if d == "SmallNeg": return random.choice(SmallNeg)
   if d == "LargePos": return random.choice(LargePos)
   return random.choice(LargeNeg)

def speed_crisp(s):
   if s == "Stop": return random.choice(Stop)
   if s == "Slow": return random.choice(Slow)
   return random.choice(Fast)
```

A pedestrian *p* moving in the physical space *s* is an object of the Python class *“Pedestrian”*, which is characterized by location information, which is a point with coordinates (x_p, y_p), to track its movements, direction information, which is the angle between the pedestrian’s position and the goal’s (exit) position, and movement speed to track its speed to move from one position to the next.

MovementsNbre, MovementsDistance and MovementsSpeed information are used to track the number of movements (a movement is a change in direction, speed, or both), the distance of the pedestrian movements, and their total velocities, respectively, to reach the arrival (goal) position. They are used to calculate the pedestrian energy to exit from the physical space. The data member of the “Blocked” information will indicate if the pedestrian is bloking due to physical obstacles preventing him from moving and the “ReachedGoal” information will indicate if the pedestrian has already reached the goal.

```
class Pedestrian:
   def __init__(self, p):
      self.position = p
      self.direction = self.perceiveDirection(
               self.get_angle(p, goal_position))
      self.speed = "Stop"
      self.MovementsNbre = 0
      self.MovementsDistance = 0
      self.MovementsSpeed = 0
      self.Blocked = False
      self.GoalReached = False
```

Pedestrian objects (we also call them decsion-makers) can update their data members during the simulation steps of the model by calling the method member *update(self, sp, speed)* to change their positions, directions, and speeds, where the parameter self is the decision maker, sp is the position of the visual sector to establish the direction of movement towards the goal and the speed is the specific movement speed of the pedestrian.

Pedestrians can have perceptions of distance, direction, and velocity data using their member methods that perceive distance *(perceiveDistance(self, d)),* direction *(perceiveDirection(self, a))* and velocity *(perceiveSpeed(self, s)),* which call corresponding model prediction functions, respectively.

To complete the model description, we need to create a set of pedestrians without overlapping each other. For a simulation, we call the function *generate_pedestrians(n)* to generate n pedestrians, where the i-th pedestrian occupies the position *positions[i]* in the space *s*.

```
def generate_pedestrians(n):
   Pedestrians = []
   if n > len(positions): n = len(positions)
   for i in range(n):
      Pedestrians.append(Pedestrian(positions[i]))
   return Pedestrians
```

A pedestrian's visual field is defined as an area in the form of a circle with a central point, which is the pedestrian's position, radius R units, and central angle ($Central\_Angle = 360°$). This region is divided into n similar but not congruent sectors occupying the central angles of $Sector\_Angle_i, i = 1, \dots, n$ ($\sum_{i=1,\dots,n} Sector\_Angle_i = Central\_Angle$).

A trade-off between model complexity and accuracy determines how many sectors make up the visual field. By integrating goal details and environmental information already received in all sectors of the visual field, pedestrian motion states can be updated in conjunction with a predefined space representation approach.

3.3 Pedestrian's behavior

Pedestrians want to survive and get away from harmful situations. In Section 3.1 and Section 3.2, we proposed an AI-based structural model of pedestrian behavior in a crowd. To make the suggested model clearer and easier to understand, the following presumptions are made:

(1) Every pedestrian knows the intricacies of their goals and the regional information within their field of vision, but they are not aware of the global knowledge about their surroundings.

(2) In an ignored rest time, the pedestrian may choose to transition between any two predefined states.

The general structure of the artificial intelligence-based model consists of a prediction system, which is used to describe a combination of obstacle-avoiding behavior and goal-seeking behavior. The input information mainly includes obstacles information, pedestrians' information, and goal information.

For example, the decision maker's (the pedestrian's) closest distance from barriers determines the decision maker's behavior of avoiding them, whereas the decision maker's path (or goal)-searching behavior is greatly influenced by the distance and speed of the person walking in the opposite direction. Turning angles, or directions, and movement speeds are the system's output data that are utilized to calculate the final motion states.

Local goal-seeking and obstacle-avoiding behaviors combine to establish crowd evacuation behaviors. A pedestrian must travel in the direction of the goal at the proper speed, avoiding collisions with other pedestrians as well as obstacles and borders that come into view.

It is clear that pedestrian behavior is mostly influenced by the goal's location, the distribution of pedestrians of the same kind, and the presence of impediments. The location of the goal should be known in advance to all pedestrians. The visual field is defined as follows: it is a circle with a radius of Radius units (50 units by default).

The central angle of the pedestrian's visual field Central_Angle is subdivided into identical sector angles with the same value as the angle $Sector\_Angle$, then we have $n = Central\_Angle/Sector\_Angle$ sectors that make up the visual field. Each sector defines a direction from the pedestrian position to the sector position sp.

In general, the decision-making process of pedestrians can be described as follows. First, the decision maker (the pedestrian) scans his visual field, sector by sector, and chooses which sector to pass through to reach the exit (goal) based on personal consideration. Then, in order to accomplish the aim, he moves in the direction of the chosen sector position sp while maintaining the proper pace and direction of travel to avoid colliding with oncoming traffic and frontal obstructions.

The closest pedestrian-obstacle distances in each sector are the algorithm's inputs, and its outputs are the turning angle and movement speed. A combination of goal-seeking and obstacle-avoiding behavior is used to determine the decision maker's final turning angle and movement speed. The decision maker will avoid the obstacle in front of it before pursuing the goal. Therefore, avoiding obstacles is more vital than pursuing goals. As a result, the pedestrians may accomplish their objective and stay clear of any impediments and other pedestrians that may cross their path.

To control pedestrians' motion, the following information must be known. The position (x_p, y_p) of the decision maker (pedestrian) p, a goal g, which is located at position (x_g, y_g) representing the place where pedestrians want to reach in s and is located at goal angle γg (the angle between the pedestrian position p and goal position g, which is called the direction angle of the pedestrian) and is at goal distance *dg* from the position of the pedestrian p.

During model simulation, the method *get_distance_perceptions(self)* (self is the pedestrian's reference), will be called by the method *behavior(self)* defined below, to get pedestrian perception of minimum distances between his location and his surrounding obstacles, borders, and other pedestrians coming in opposite directions in different sectors of his visual field.

These distance perceptions are saved to a list called *distance_perceptions*. Its maximum length is the number of sectors, which equals $Central\_Angle/Sector\_Angle$ (the sector's angle $Sector\_Angle$ is equal, by default, 45 degrees).

The list *distance_perceptions* after executing *get_distance_perceptions(self)* will contain information about the distances that the subject perceives himself in each sector of his visual field.

```
def get_distance_perceptions(self):
    sector_angle = 0
    obstacles_borders = obstacles + borders
    distance_perceptions = []
    while sector_angle < Central_Angle:
        sector_pos = Point((
            self.position.x+Radius*math.sin(sector_angle),
            self.position.y+Radius*math.cos(sector_angle)))
        if (sector_pos.x < -SpaceWidth+Radius or
             sector_pos.x > SpaceWidth-Radius or
             sector_pos.y < -SpaceLength+Radius or
             sector_pos.y > SpaceLength-Radius):
              sector_angle += Sector_Angle
              continue
        MinDistance = VisualDistance
        for ob in range(len(obstacles_borders)):
            d = Poly(obstacles_borders[ob]
                    ).boundary.distance(sector_pos)
            if d >  Poly(obstacles_borders[ob]
                      ).boundary.distance(self.position):
                continue
            if d < MinDistance: MinDistance, kind = d, -1
        for k in range(len(Pedestrians)):
            if self != Pedestrians[k]:
                d =sector_pos.distance(Pedestrians[k].position)
                if d > self.position.distance(
                          Pedestrians[k].position): continue
                if d < MinDistance: MinDistance, kind = d, k
        if MinDistance < VisualDistance:
            distance_perceptions.append((sector_pos,
                  self.perceiveDistance(MinDistance), kind))
        else:
          distance_perceptions.append((sector_pos,"Far",-1))
        sector_angle += Sector_Angle
    return distance_perceptions
```

All sector information are triples, the first element being the position of the sector, and the second element being the minimum perceived distance from the pedestrian's position to all obstacles, borders, and other pedestrians observed through the sector's window of the visual field.

The third information item in the triple, is used to determine whether the obstacle is a pedestrian walking in the opposite direction or another obstacle. When this information are collected for the current pedestrian (decision maker), he will make a decide to change position with appropriate speed and angle of direction in relation to his strategy.

The method *behavior(self)* is defined below to change the state of the pedestrian to make him move from one location to another until it reaches the exit (goal). The current pedestrian can also go into a state of blocking (stop moving) if he is in front of physical obstacles in all sectors of the visual field.

To determine the movement of each pedestrian, we call the function *get_distance_perceptions(self)* defined above to get decision-making information for the pedestrian self. The method *behavior(self)* combines the obstacle-avoiding behavior and the goal-seeking behavior to get new positions.

These new positions are candidates for moving the pedestrian (decision-maker). The selection of the best new position is based on a formula giving the small weighting between the distance and dierction angle from the goal using two weighting parameters alpha and beta to define the preference between them.

First, the pedestrian self (the decision maker) checks whether he is close to a kind, who has already reached the goal, and then his state changes to as he has reached the goal (exit).

If not, he will check if all obstacles/borders/pedestrians are far from him to decide to follow the path for seeking the goal by calling the method *goal_seeking_behavior(self, distance_perceptions)*.

```
def behavior(self):
   if not self.GoalReached and not self.Blocked:
     distance_perceptions =
                           self.get_distance_perceptions()
     if [k for (x, (sp, od, k)) in
                     enumerate(distance_perceptions)
       if od == "Near" and
          k != -1 and
         Pedestrians[k].GoalReached] != []:
              self.GoalReached = True
       else:
          # The goal-seeking behavior
          new_positions =
                 self.goal_seeking_behavior(
                         distance_perceptions)
          if new_positions == []:
             # The obstacle-avoiding behavior
             new_positions =
                      self.obstacle_avoiding_behavior(
                                    distance_perceptions)
          if new_positions != []:
             new_positions.sort(
                     key = lambda x:x.get('index'))
             if new_positions[0].get('speed') != "Stop":
                 self.update(new_positions[0].get('sp'),
                              new_positions[0].get('speed'))
          else: self.Blocked = True
```

One sort of global conduct known as "goal-seeking behavior" is the propensity of the decision-maker to constantly move in the direction of his goal, regardless of the surroundings around him. It is defined by the goal angle γ_g (which can belong to one of the classes LargeNeg, SmallNeg, Zero, SmallPos, and LargePos) and the goal distance *dg* (which can belong to one of the classes Near or Far).

Formulating global goal-seeking behavior motivates pedestrians to walk in the direction of the goal. The decision maker reduces his speed and turns sharply towards the target without missing it when the pedestrian approaches the exit but does not face it. On the contrary, facing the target, he moves freely in the direction of the target at a high (fast) speed.

With this method, the decision maker first scans the sectors, which indicate distant internal obstacles, external obstacles (boundaries), and blocked pedestrians, in which case he adds to the list of decisions "gsb" (goal-seeking behavior) to move to a direction specified by the expression *abs(self.get_angle(self.position,sp)-self.get_angle* *(self.position, goal_position))* with fast movement speed.

If the obstacle is of some kind, the decision maker checks if that kind is going in the opposite direction and then adds to the decisions the same angle of movement direction but stops moving otherwise he will add fast as the movement speed.

```
def goal_seeking_behavior(self, distance_perceptions):
    gsb = [(sp, sp.distance(goal_position),
             abs(self.get_angle(self.position, sp) -
             self.get_angle(self.position, goal_position)),
            "Fast")  for (x, (sp, od, k)) in
             enumerate(distance_perceptions)
             if od == "Far" and (k == -1 or (k != -1 and
            Pedestrians[k].Blocked))]
    gsb = gsb + [(sp, sp.distance(goal_position),
                  abs(self.get_angle(self.position, sp)-
                  self.get_angle(self.position,
                  goal_position)), "Slow")
                  for (x, (sp, od, k)) in enumerate
                  (distance_perceptions)
                  if od == "Far" and k != -1 and not
                    Pedestrians[k]. Blocked and
                    self.opposite(self.perceiveDirection(
                          self.get_angle(sp, goal_position)),
                    Pedestrians[k].direction)]
    gsb = gsb + [(sp, sp.distance(goal_position),
                  abs(self.get_angle(self.position, sp)-
                  self.get_angle(self.position,
                                  goal_position)), "Fast")
                  for (x, (sp, od, k)) in
                          enumerate(distance_perceptions)
                   if od == "Far" and k != -1 and not
                      Pedestrians[k].Blocked and not
                      self.opposite(self.perceiveDirection(
                                         self.get_angle(sp,
                                         goal_position)),
                                 Pedestrians[k].direction)]
    return [{'index':(alpha * d + beta * a)/
                     (alpha+beta), 'sp':sp, 'speed':speed}
```

If the decision maker cannot find a sector position (the list *new_positions* is empty) to seek the goal, which means he is surrounded by obstacles from all the directions defined by the sectors of his visual field, then the obstacle-avoiding behavior must be invoked, which is defined with the method *obstacle_avoiding_behavior(self, distance_perceptions),* to avoid frontal obstacles because the distances between decision maker and the obstacles are close at the local scope.

This method is called because all obstacles are close to pedestrians, in which case he checks if one of the obstacles is some kind. In the code, the decision maker adds to the list "pab" (pedestrian (obstacle)-avoiding behavior), the decision to follow the direction, which is calculated by the following expression. The absolute value of the angle between the sector position (from which the decision maker has observed a kind walking in his direction) and the goal minus the angle between the decision maker and the goal, at a slow speed in motion.

If the kind is going in the opposite direction, the resolution to add has the same direction as above but with a stop moving until future simulation steps.

If no kind is an obstacle, then all the obstacles are internal or external obstacles, which puts the decision maker in a state of blocking (see the method of behavior) and then cannot be evacuated (i.e., access to the exit).

```
def obstacle_avoiding_behavior(self,
                              distance_perceptions):
    oab = [(sp, k) for (x, (sp, od, k)) in enumerate
           (distance_ perceptions)
    if od == "Near" and k != -1 and not
      Pedestrians[k].Blocked]
    new_positions = []
    if oab != []:
       pab = [(sp, sp.distance(goal_position),
               abs(self.get_angle
             (self.position, sp) self.get_angle(self.position,
              goal_position)), "Slow")  for (x, (sp, k)) in
                 enumerate(oab) if not Pedestrians[k].
                     GoalReached and  not self.opposite(
                     self.perceiveDirection(
                        self.get_angle(sp, goal_position)),
                     Pedestrians[k].direction)]
       pab = pab + [(sp, sp.distance(goal_position),
                   abs(self.get_angle(self.position, sp) –
                   self.get_angle (self.position,
                                  goal_position)), "Stop")
     for (x, (sp, k)) in enumerate(oab) if not
                         Pedestrians[k]. GoalReached and
                       self.opposite(self.perceiveDirection(
                         self.get_angle(sp, goal_position)),
                         Pedestrians[k].direction)]
       new_positions = [{'index':(alpha * d + beta * a)/
                        (alpha+beta), 'sp':sp, 'speed':speed}
                for (i, (sp, d, a, speed)) in enumerate(pab)]
                      return new_positions
```

The definition of obstacle-avoiding behavior is the tendency of a pedestrian to shift direction gradually and smoothly as opposed to abruptly. Because of this, if there is an identical pedestrian-obstacle distance in every sector, the code is built so that pedestrians will generally go in the same direction.

This crowd evacuation model is described by a way that can help analysis of its simulation results using a method by which we can change parameters like space dimensions, number of sectors of visual field, weighting parameters, numbers of obstacles, shape of each obstacle, number of pedestrians, etc. Some values of these parameters can make the model more complex and then its simulation more time and space consuming.

4. MODEL VALIDATION AND SIMULATION

This model of crowd evacuation must be validated before running the simulation (i.e. executing the Python code). The model is validated by animating the behavior of a small number of pedestrians in the physical space to understand their movements.

The function do_animation() below, can be called for generating animation data from the model simulation, then generating animation by calling the partial function animate() using the Python functional tools.

The result of the animation is saved as an animated GIF (Graphics Interchange Format) file to be displayed to check pedestrian behaviors.

```
def animate(i, ax, animation_data):
    ax.clear()
    for p in range(len(Pedestrians)):
        ax.plot(animation_data[p][i].x,
                    animation_data[p][i].y, color='green',
                    label='original', marker='o')
        ax.annotate(f"({p})", (animation_data[p][i].x,
                            animation_data[p][i].y), textcoords =
                            "offset points", xytext=(0,10),
                             ha='center')
    plot_physical_space(ax)
```

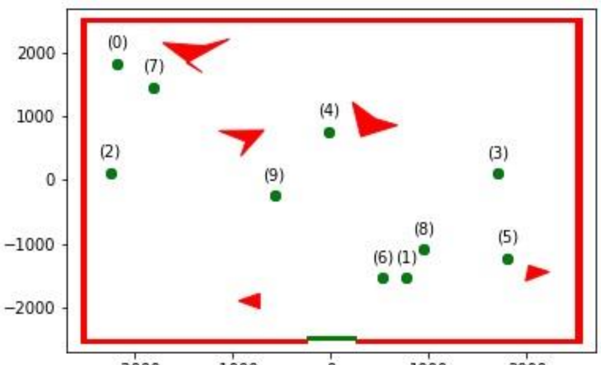
The *generate_animation_data(n)* function is used to generate animation data from the initial state of a number of pedestrians to their last state, in which all pedestrians have reached the goal (exit) position or entered the blocking state.

```
  def do_animation():
    fig, ax = plt.subplots(1,1)
    plt.ylim(-SpaceLength/2-200,SpaceLength/2+200)
    plt.xlim(-SpaceWidth/2-200,SpaceWidth/2+200)
    animation_data = generate_animation_data()
    ani = FuncAnimation(fig, functools.partial(animate,
                        ax = ax,
                        animation_data = animation_data),
                frames =  len(animation_data[0]),
                interval=500, repeat=False)
    # Save the animation as an animated GIF
    ani.save("animation.gif", dpi=100, writer=
                 PillowWriter(fps=1))
                     fig.savefig("animationEnd.png")
```

The animation function below can be called to clear the previous state and plot the new state after one step to move all the pedestrians. In the plots, all pedestrians represented by their location points are annotated with the associated numbers.

```
  def generate_animation_data():
    animation_data = [[] for _ in Pedestrians]
    for p in range(len(Pedestrians)):
        animation_data[p].append(Pedestrians[p].position)
    plot_for_animation(animation_data,
                            "animationStart.png")
    while not pedestrians_behavior_done():
        for p in range(len(Pedestrians)):
           Pedestrians[p].behavior()
           animation_data[p].append(Pedestrians[p].position)
    return animation_data
```

Figure 1(a) shows the initial state of the distribution of all pedestrians (in green) annotated by their numbers in the physical space. The obstacles are colored red and the exit (goal) is a green line at the bottom.



(a) Initial state

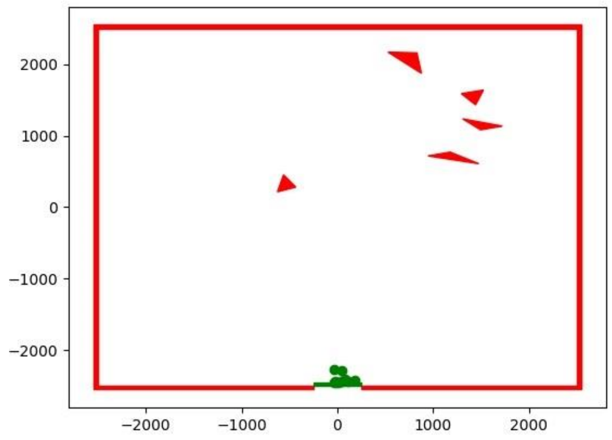
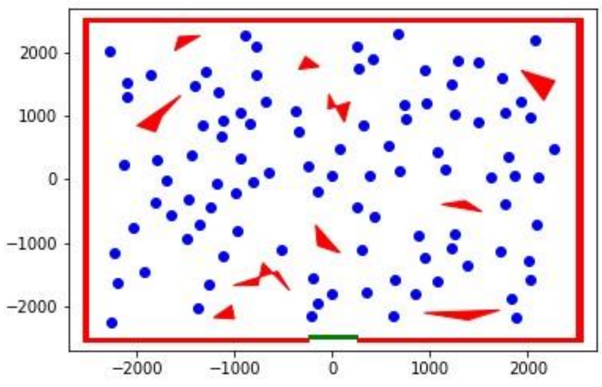

(b) Final state

Figure 1. Animation with graphical information format file

When we open the GIF file, we see the animation of the pedestrians' movements from this initial state to the last state shown in Figure 1(b). In this latter case, we observe that all pedestrians have reached the goal.


(a) Initial distribution

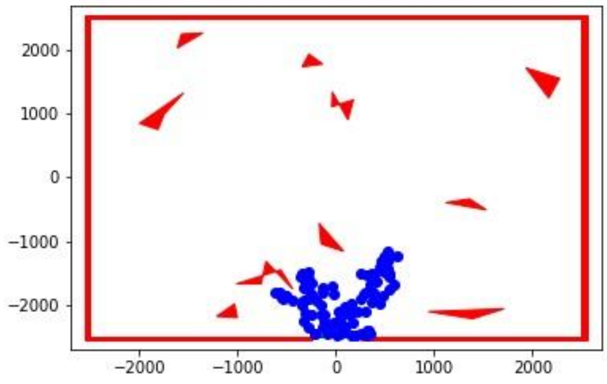

(b) Final state

Figure 2. Simulation of the crowd evacuation model

This model of crowd evacuation is being tested extensively to verify how pedestrians avoid obstacles and borders, their kinds, and how they change direction and update their speeds to reach the goal. After this verification process, which helped us fix many bugs in particular errors of pedestrians behaviour, a simulation can be launched with a large number of pedestrians and obstacles.

Python code has been written to run the simulation, in which several results can be tracked for performance evaluation in Section 5. Figure 2(a) shows the initial randomization of the 100-pedestrian crowd in blue and the inner and outer obstacles in red. For this simulation, every pedestrian is moving in the direction of the exit at the required speed until the final stop shown in subgraph (b) of Figure 2 after the simulation is terminated. For performance evaluation, this crowd evacuation model was simulated on different parameter values.

5. EXPERIMENTS AND PERFORMANCE EVALUATION

The evaluation of the performance of this crowd evacuation analysis using the simulation of the model described in Section 3, is based on an assessment of the scalability over crowds of large sizes, which must adhere to the pedestrian randomization composed of crowds and obstacles. Therefore, the evaluation will measure trade-offs between scalability and pedestrian randomization.

The performance evaluation is based on a set of metrics. The first metric is the effectiveness of the crowd evacuation, which is the ability to evacuate as many pedestrians as possible. The degree of effectiveness of crowd evacuation is measured in terms of the percentage of pedestrians R who reach goal by model simulation on the total number of pedestrians N ($Effectiveness = R/N * 100\%$).

The evacuation of the crowd is effective if all the pedestrians that make up the crowd have reached the goal (exit).

The second metric is the efficiency of crowd evacuation, which is the ability to evacuate an intended number of pedestrians with the least pedestrian energy, which is expressed as follows:

If we express the total number of pedestrian movements by NM (pedestrian movement is the movement after updating its state in direction and speed, used to measure the pedestrian dynamics and velocity-density relationship), the total distance traveled by all pedestrians to reach the goal by D.

Then the efficiency degree is $Efficiency = \big((N/NM + ID/D) * Density\big) * 100\%$, where ID is the ideal total distance from the initial positions of the pedestrians to the goal position, and $Density$ is the model density, which is $Density = (N + \sum PO)/NP$, where $\sum PO$ is the total number of positions occupied by internal obstacles and NP is the total number of associated positions by physical space.

The third metric is the total time T spent by all pedestrians to reach the goal or stop moving. In addition to these metrics, there is a common metric of evacuation performance that measures evacuation time and effectiveness/efficiency [31] together to help assess potential trade-offs between evacuation time and evacuation capacity, as well as overall crowd evacuation performance.

To measure these performance metrics, several experiments must be done using model simulation for evacuating large crowds. So, we need to be able to vary the number of pedestrians, and see how it scales. We randomly created a physical space (a square hall), which is composed of 10 internal obstacles that have different shapes.

We have achieved eight experiments ($e = 1, \dots, 8$), in which the eight diverse crowds were randomly distributed in the hall consisting of 20, 40, 60, 80, 100, 120, 140, and 160 pedestrians, respectively, to be evacuated from the single exit (goal) with a specified width.

The model simulations were run on a laptop running Windows 10 (64 bits) with an Intel (R) Core (TM) i5-4210U CPU clocked at 1.70 GHz and 8 GB of RAM.

The experiments by model simulations are to see if we can evacuate these large-scale crowds from the physical space and to measure its effectiveness and efficiency.

Table 1 shows for each experiment e, the number of pedestrians N composing the crowd, the model density $Density$, the total number R of the pedestrians reached the goal, the total number of pedestrians' movements NM, the total distance D traveled by the pedestrians before reaching the goal or stopping the movement, the ideal distance ID, the total time spent by all the pedestrians during their movements T in seconds, crowd evacuation effectiveness ES and efficiency EY.

The results showed that almost all pedestrians reached the goal (exit) location and there was a proportional increase in the number of simulation steps (number of movements) and simulation time with increasing number of pedestrians (model intensity). These results are consistent with those of previous researchers [32, 33].

Figure 3 shows that the total number of movements and the total distance traveled obviously increase with the increase in density (the number of pedestrians in the physical space), causing the average velocity to decrease until the crowd almost reaches a steady state when the density exceeds a critical condition.

However, if the density is at an acceptable level, the pedestrian will move at the desired speed due to sufficient space and slight influences from other pedestrians. These results are consistent with what was expected from the model description.

Table 1. Experimental results

Experiment	N. of Pedestrians	Density	N. of reached Goal	N. of Movements	Travelled Distance	Ideal T. Distance	T (seconds)	Effectiveness	Efficiency
e=1	20	12.0%	20	359	71554.5	61784.67	7.65	100.0%	85.94%
e=2	40	24.0%	39	680	135547.5	109938.96	18.52	97.5%	80.73%
e=3	60	36.0%	60	1015	198974.0	150560.28	31.31	100.0%	75.31%
e=4	80	48.0%	80	1260	245035.0	193382.25	44.46	100.0%	78.55%
e=5	100	60.0%	100	1600	315699.5	236088.38	67.29	100.0%	74.44%
e=6	120	72.0%	119	1825	360660.0	256611.76	80.78	99.17%	70.83%
e=7	140	84.0%	138	1985	390559.0	291358.11	99.03	98.57%	74.26%
e=8	160	96.0%	159	2258	441845.0	334681.13	121.38	99.38%	75.4%

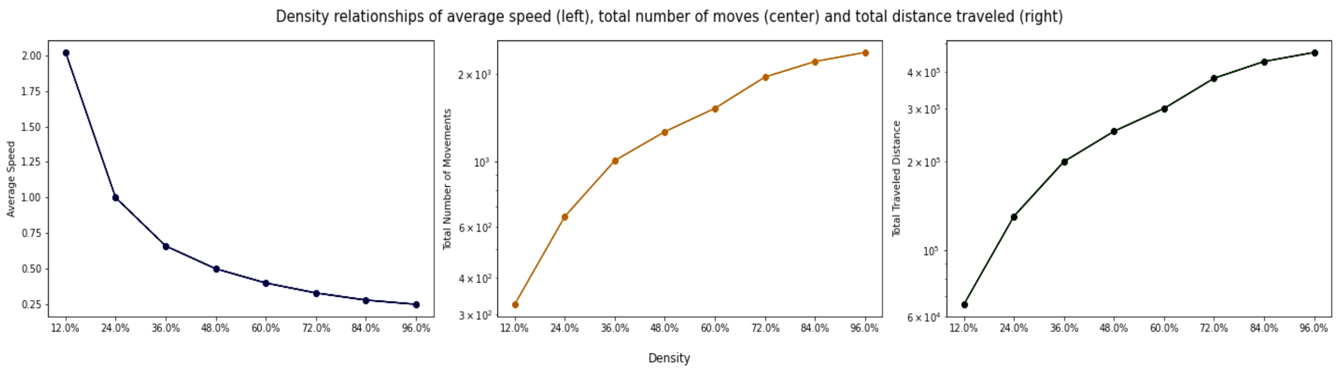


Figure 3. Density relationships of average speed (left), total number of moves (center) and total distance traveled (right)

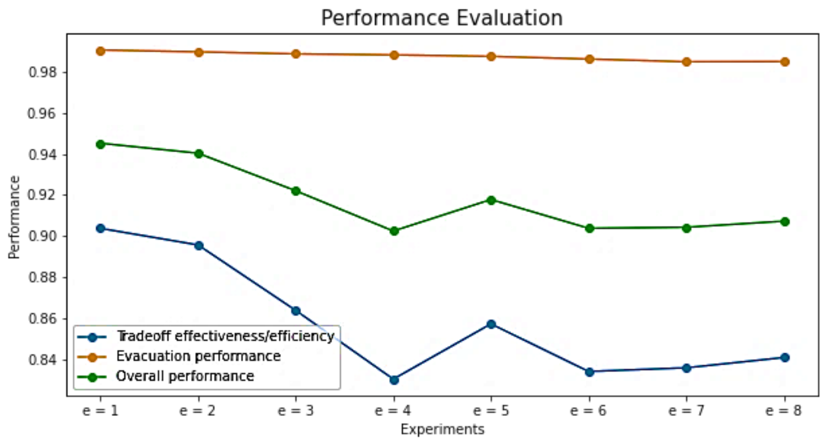


Figure 4. Performance evaluation

It is important to evaluate the overall performance of the crowd evacuation model by combining the metric results above. The metric F_e (F-Score of combining effectiveness and efficiency) is used to calculate the trade-off between crowd evacuation effectiveness ES and efficiency EY (Eq. (1)).

$$F_e = \frac{(\beta^2+1)\times EY_e \times ES_e}{\beta^2 \times EY_e \times ES_e} \tag{1}$$

In the preceding formula, EY_e and ES_e ($\in [0,1]$) represent the experiment (e) model simulation efficiency and effectiveness, respectively. Consequently, the relative weighting between them is ascertained using this formula. We employ the metric in addition to computing the crowd evacuation model performance P_e (Precision of experiment) for the experiment e (Eq. (2)).

$$P_e = \frac{1}{1+\exp^{\gamma\times T_e/N_e-\delta}} \tag{2}$$

In the formula, T_e is the simulation time (seconds) for the experiment e. The number N_e is the total number of pedestrians. To allow for comparison of the metric values across experiments of different sizes, we use the simulation time per pedestrians number (i.e. T_e/N_e) in the calculation.

To distinguish between the experiment results over different sizes, two control parameters γ (called the slope) and δ (called the shift) are used. Their values are chosen to make P_e (the fast model simulation with respect to N_e) close enough to one (above 0.99). For these experiments, $\gamma = 2$ and $\delta = 5$.

The overall performance metric OP (F-Score of P and F combination) rewards the model simulation when it makes pedestrians reach the goal faster, more efficiency and more effectively and it is defined as a composite metric of simulation time, efficiency and effectiveness for each experiment e, where $e = 1, \cdots, 8$ as in the following (Eq. (3)).

$$OP_e = \frac{(\alpha^2+1)\times P_e \times F_e}{\alpha^2 \times P_e + F_e} \tag{3}$$

The values of β and α are set to 1, which means the weights of crowd evacuation efficiency and effectiveness are equal, and the weight of model simulation time against the weight of efficiency and effectiveness are also equal.

Figure 4 shows the results of the overall performance metric OP of the model simulation that is calculated for each experiment. The trade-off between the efficiency and effectiveness of crowd evacuation, is more than 80%, with an overall performance of more than 90%, and the evacuation performance is almost perfect, which is more than 99%.

The aggregate results show that the model simulation runs extremely smoothly across the board for every experiment. These numerical findings, displayed in Figure 4, aid in our comprehension of our model's overall effectiveness in crowd evacuation.

6. CONCLUSIONS AND PERSPECTIVES

We have presented a description with the Python language of a model for the simulation of the evacuation of crowds located in physical spaces with dangerous situations. The behavior of the pedestrians composing this model is described using the technique of artificial intelligence, especially the perception of numerical quantities such as distances, angles of direction and speeds of movement which transform by nature into qualitative quantities for human reasoning.

Before the simulation, this model was validated using an animation (an animated execution) which helped us to fix several errors, especially in the description part of pedestrian behavior. The simulation results are analyzed to evaluate the performance of this model. We obtained a good compromise between its effectiveness and its efficiency. These performance analysis results are very promising for future prospects.

The first perspective is to vary the behavior of pedestrians according to characteristics such as gender, age, whether they have diseases, etc. The second perspective is to automate the description of physical spaces. The input to this automation will be images on the physical spaces and through the object detection technique we can have their descriptions. The third perspective is to use artificial intelligence to predict pedestrian behavior.

REFERENCES

[1] Shin, Y., Moon, I. (2023). Robust building evacuation planning in a dynamic network flow model under collapsible nodes and arcs. Socio-Economic Planning Sciences, 86: 101455. https://doi.org/10.1016/j.seps.2022.101455

[2] Adrian, J., Seyfried, A., Sieben, A. (2020). Crowds in front of bottlenecks at entrances from the perspective of physics and social psychology. Journal of the Royal Society Interface, 17(165): 20190871. https://doi.org/10.1098/rsif.2019.0871

[3] Bakar, N.A.A., Majid, M.A., Ismail, K.A. (2017). An overview of crowd evacuation simulation. Advanced Science Letters, 23(11): 11428-11431. https://doi.org/10.1166/asl.2017.10298

[4] Zheng, L., Peng, X., Wang, L., Sun, D. (2019). Simulation of pedestrian evacuation considering emergency spread and pedestrian panic. Physica A: Statistical Mechanics and its Applications, 522: 167-181. https://doi.org/10.1016/j.physa.2019.01.128

[5] Zhang, J., Zhu, J., Dang, P., Wu, J., Zhou, Y., Li, W., Fu, L., Guo, Y., You, J. (2023). An improved social force model (ISFM)-based crowd evacuation simulation method in virtual reality with a subway fire as a case study. International Journal of Digital Earth, 16(1): 1186-1204. https://doi.org/10.1080/17538947.2023.2197261

[6] Zhang, D., Li, W., Gong, J., Zhang, G., Liu, J., Huang, L., Liu, H., Ma, H. (2023). Deep reinforcement learning and 3D physical environments applied to crowd evacuation in congested scenarios. International Journal of Digital Earth, 16(1): 691-714. https://doi.org/10.1080/17538947.2023.2182376

[7] Zhou, M., Dong, H., Wang, F.Y., Wang, Q., Yang, X. (2016). Modeling and simulation of pedestrian dynamical behavior based on a fuzzy logic approach. Information Sciences, 360: 112-130. https://doi.org/10.1016/j.ins.2016.04.018

[8] Zhou, M., Dong, H., Wen, D., Yao, X., Sun, X. (2016). Modeling of crowd evacuation with assailants via a fuzzy logic approach. IEEE Transactions on Intelligent Transportation Systems, 17(9): 2395-2407. https://doi.org/10.1109/TITS.2016.2521783

[9] Baker, C. (2019). Artificial Intelligence: Learning Automation Skills with Python (2 books in 1: Artificial Intelligence a Modern Approach & Artificial Intelligence Business Applications). Independently published.

[10] Teso, S., Alkan, Ö., Stammer, W., Daly, E. (2023). Leveraging explanations in interactive machine learning: An overview. Frontiers in Artificial Intelligence, 6: 1066049. https://doi.org/10.3389/frai.2023.1066049

[11] Helbing, D., Molnar, P. (1995). Social force model for pedestrian dynamics. Physical Review E, 51(5): 4282. https://doi.org/10.1103/PhysRevE.51.4282

[12] Varas, A., Cornejo, M.D., Mainemer, D., Toledo, B., Rogan, J., Munoz, V., Valdivia, J.A. (2007). Cellular automaton model for evacuation process with obstacles. Physica A: Statistical Mechanics and Its Applications, 382(2): 631-642. https://doi.org/10.1016/j.physa.2007.04.006

[13] Liu, S., Yang, L., Fang, T., Li, J. (2009). Evacuation from a classroom considering the occupant density around exits. Physica A: Statistical Mechanics and its Applications, 388(9): 1921-1928. https://doi.org/10.1016/j.physa.2009.01.008

[14] Helbing, D., Isobe, M., Nagatani, T., Takimoto, K. (2003). Lattice gas simulation of experimentally studied evacuation dynamics. Physical review E, 67(6): 067101. https://doi.org/10.1103/PhysRevE.67.067101

[15] Guo, R.Y., Huang, H.J. (2008). A mobile lattice gas model for simulating pedestrian evacuation. Physica A: Statistical Mechanics and its Applications, 387(2-3): 580-586. https://doi.org/10.1016/j.physa.2007.10.001

[16] Shi, D.M., Wang, B.H. (2013). Evacuation of pedestrians from a single room by using snowdrift game theories. Physical Review E, 87(2): 022802. https://doi.org/10.1103/PhysRevE.87.022802

[17] Bouzat, S., Kuperman, M.N. (2014). Game theory in models of pedestrian room evacuation. Physical Review E, 89(3): 032806. https://doi.org/10.1103/PhysRevE.89.032806

[18] Shende, A., Singh, M.P., Kachroo, P. (2011). Optimization-based feedback control for pedestrian evacuation from an exit corridor. IEEE Transactions on Intelligent Transportation Systems, 12(4): 1167-1176. https://doi.org/10.1109/TITS.2011.2146251

[19] Frantzich, H., Nilsson, D. (2004). Evacuation experiments in a smoke filled tunnel. In 3rd International Symposium on Human Behaviour in Fire, Interscience Communications Ltd, United Kingdom, pp. 229-238.

[20] Kobes, M., Helsloot, I., De Vries, B., Post, J.G., Oberijé, N., Groenewegen, K. (2010). Way finding during fire evacuation; An analysis of unannounced fire drills in a hotel at night. Building and Environment, 45(3): 537-548. https://doi.org/10.1016/j.buildenv.2009.07.004

[21] Ma, J., Lo, S.M., Song, W.G. (2012). Cellular automaton modeling approach for optimum ultra high-rise building evacuation design. Fire Safety Journal, 54: 57-66. https://doi.org/10.1016/j.firesaf.2012.07.008

[22] Song, Y., Gong, J., Li, Y., Cui, T., Fang, L., Cao, W. (2013). Crowd evacuation simulation for bioterrorism in micro-spatial environments based on virtual geographic environments. Safety Science, 53: 105-113. https://doi.org/10.1016/j.ssci.2012.08.011

[23] Nasir, M., Lim, C.P., Nahavandi, S., Creighton, D. (2014). A genetic fuzzy system to model pedestrian walking path in a built environment. Simulation Modelling Practice and Theory, 45: 18-34. https://doi.org/10.1016/j.simpat.2014.03.002

[24] Dell'Orco, M., Marinelli, M., Ottomanelli, M. (2014). Simulation of crowd dynamics in panic situations using a fuzzy logic-based behavioural model. In: de Sousa, J., Rossi, R. (eds) Computer-based Modelling and Optimization in Transportation. Advances in Intelligent Systems and Computing, Springer, Cham, 262. https://doi.org/10.1007/978-3-319-04630-3_18

[25] Bahamid, A., Mohd Ibrahim, A. (2022). A review on crowd analysis of evacuation and abnormality detection based on machine learning systems. Neural Computing and Applications, 34(24): 21641-21655. https://doi.org/10.1007/s00521-022-07758-5

[26] Lee, J., Won, J., Lee, J. (2018). Crowd simulation by deep reinforcement learning. In Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games, pp. 1-7. https://doi.org/10.1145/3274247.3274510

[27] Wang, K., Shi, X., Goh, A.P.X., Qian, S. (2019). A machine learning based study on pedestrian movement dynamics under emergency evacuation. Fire Safety Journal, 106: 163-176. https://doi.org/10.1016/j.firesaf.2019.04.008

[28] Yao, Z., Zhang, G., Lu, D., Liu, H. (2019). Data-driven crowd evacuation: A reinforcement learning method. Neurocomputing, 366: 314-327. https://doi.org/10.1016/j.neucom.2019.08.021

[29] Li, X., Liang, Y., Zhao, M., Wang, C., Bai, H., Jiang, Y. (2019). Simulation of evacuating crowd based on deep learning and social force model. IEEE Access, 7: 155361-155371. https://doi.org/10.1109/ACCESS.2019.2949106

[30] Raschka, S., Patterson, J., Nolet, C. (2020). Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information, 11(4): 193. https://doi.org/10.3390/info11040193

[31] Zidane, Y.J.T., Olsson, N.O. (2017). Defining project efficiency, effectiveness and efficacy. International Journal of Managing Projects in Business, 10(3): 621-641. https://doi.org/10.1108/IJMPB-10-2016-0085

[32] Helbing, D., Johansson, A., Al-Abideen, H.Z. (2007). Dynamics of crowd disasters: An empirical study. Physical Review E, 75(4): 046109. https://doi.org/10.1103/PhysRevE.75.046109

[33] Helbing, D., Farkas, I., Vicsek, T. (2000). Simulating dynamical features of escape panic. Nature, 407(6803): 487-490. https://doi.org/10.1038/35035023