Agent-Based Crowd Evacuation using Boids

Ishpinder Kaur
University of Waterloo
20911296

Abstract: **During crowd evacuation in emergency situations, there are several physical and psychological factors that affects the safe escape as well as injuries caused in those situations. It is important to consider those factors and perform proper analysis to figure out best strategies for safe escape and also, to know the other behavioral parameters that can provide more efficiency to this process. In this paper, an idea is proposed which shows the behavior of multiple agents in crowd, in different environments considering their physical and psychological factors, especially panic. An algorithm of boids is implemented which is modified with the addition of some rules to deploy real life simulation of emergency-based crowd evacuation strategy.**

***Index Terms*— agents, boids, crowd behavior, emergency evacuation, panic, simulation.**

I. INTRODUCTION

Crowd evacuation is evacuating large proportion of people in less amount of time and is essential in several emergency situations such as earthquakes, fire, chemical exposure, terrorist attacks, and so on. Depending upon the level of threat, panic arises which causes an irrational behavior that leads to time consuming evacuation and in worse cases, mass fatalities and stampede [1] [2].

Behavior that is commonly visible in crowd is herding behavior that is following the neighboring people. Human response to emergency is quite different, unpredictable, and uncoordinated, triggered due fear and panic. This all points to immediate need of strategies which comply with human behavior. The 'what-if' behavior of the crowd is handled quite well by implementing agent-based models. Real life experiments and their respective data in emergency situation is a vital component in planning the modelling techniques of safety procedures.

Netlogo platform has been used to perform simulation in this project. Netlogo is a programming language and Integrated Development Environment that is built to perform multi-agent based simulations. Simulation of different scenarios are performed having different permutations of obstacles and exit doors, where agents will try to get to the exit doors in the minimum time possible. In summary, this report will evaluate the following:

1) Evaluating the agent-based modelling strategies for evacuation considering psychological and physical factors.

2) Herding behavior will be analyzed between agents using boids algorithm.

3) Emergent properties will be reflected from crowd behavior following the predefined rules.

4) By deploying different strategies following this rule-based approach and real-life behavioral traits, various what-if situations will be analyzed in case emergencies.

This report holds seven sections where, Section 1 is the introduction, Section 2 will reflect relevant papers and research implemented in this field, Section 3 will show the proposed model architect, Section 4 shows the experiments, Section 5 includes the results and discussion of the implemented simulation, Section 6 will be the future work and Section 7 has the conclusion.

II. RELATED WORK

Reynold [3] proposed theory of boids which is great achievement in the field of behavioral studies. Through observing flocking behavior in birds, it is stated that application of some local rules results in emergent property or global complex behavior. A leader-follower model is then presented by Ji and Gao [4] where each leader is assigned an exit door and followers will follow their leader using path finding A* algorithm [5] in dancing hall environment. This model is then mathematically optimized by Yang et al. [6]

Even in the studies of robotics control, flocking behavior has great impact. Using rule-based learning technique, a genetic algorithm was proposed by Schultz et al [7]. Bayazit et al [8] stated that global information from environment roadmap can provide more sophisticated herding property, which can be used for traversing, goal searching and so on.

A strong algorithm of social model force is then introduced by Helbing et al [9] which considered socio-psychological factors such as panic. Here, velocity of panic agents in altered and some force arises among agents while running and pushing. Following this, Murphy [10] studied the reaction of agents in panic situation and suggested voice communication to decrease panic.

III. MODEL ARCHITECTURE

The proposed model [11] is multi-agent based system where agents behaves as a particle (like, in other multi-particles models) with its motion direction and physical dimensions. Each agent shows its motion in 2-D Euclidean plane. Each agent displays certain behavior and has various parameters which also consists of refinement factor. Various environments simulated in this model are human agent environment, panic environment, obstacle environment, and visibility environment.

A. Boids Model

An artificial life algorithm called 'Boids' was proposed by Craig Reynolds [3] in 1986, for simulating flocking behavior in birds. It carries certain set of rules to show this flocking behavior which will be used in this model to show herding behavior of human crowd. The rules are mentioned as follows:

1) Cohesion: Every agent steers towards the average position or center of mass of local agents. Here, position vector is considered, ignoring the velocity.

2) *Separation: An* agent maintains a certain minimum distance and moves away from neighboring agent to avoid overcrowding and collision among them. The considerable factor here is position.

3) Alignment: Every agent steers towards the average heading of their neighboring agents. In this velocity factor is considered while ignoring the position factor and thus, can also be called as velocity matching.

B. Panic Model

Panic plays important role in this model. In panic situation, individual makes irrational decisions which will affect the path planning. Based on some related studies [12] [13] [14] , there are certain factors that leads to panic situation are described below.

1) The distance of an agent from exit door, as if distance is more, it will take agent more time to exit that room.
2) The velocity of the neighboring agents can increase panic among agents.
3) The alignment of crowd, as if crowd is moving in wrong direction, it will affect the evacuation of agent and thus, increase panic.
4) The number of neighboring agents present, as more are the agents, more will be the time to evacuate and hence, more panic.

The panic among agents at time t can be defined by $\zeta(t)$ which is either 0 or 1, where 0 means no panic and 1 means panic. It can also be called as combination of all the predefined panic factors of agent. The factors can be represented by $\delta_k(t)$, where k defines the above-mentioned factors.

Distance of current agent from its exit door can be calculated by taking euclidean distance of position vector agent from mid-point of position vector of exit door as follows,

$$D_i = \left(\vec{m_d} - \vec{p_i}\right) \quad (1)$$

Where, D_i is the distance between agent and mid-point of door, $\vec{m_d}$ is the position vector of mid-point of exit door and, $\vec{p_i}$ is position vector of agent i.

Hence, δ_d will calculated as,

$$\delta_d = \frac{D_i}{l} \quad (2)$$

Where, l is maximum distance of any agent from the door.

Another factor is velocity of agents. For this, first average velocity of the neighbouring agents will be calculated as,

$$v^{\rightarrow} = \frac{1}{N_c} \sum \vec{v_j} \quad (3)$$

Where, $v^{\rightarrow}$ represents the average velocity of all neighbouring agents, $\vec{v_j}$ represents the velocity of different neighbouring agents and N_c is the total count of close neighbouring agents.

Now, the velocity of agent i will be compared and calculated with neighbouring agents as,

$$\delta_v = \frac{1}{\|\vec{v_{max}}\|} . (\| \vec{v} \| - \| \vec{v_i} \|) \quad (4)$$

Here, $\vec{v_{max}}$ is the maximum velocity

Also, in case of lagging velocity of agent in crowd, panic will occur.

$$\delta_l = \frac{1}{\|\vec{v_{max}}\|} . (\| \vec{v_{min}} \| - \| \vec{v_i} \|) \quad (5)$$

Where, $\vec{v_{min}}$ is minimum velocity of agent when there is no panic.

Therefore, panic factor can be defined as combining these distance and velocity factors,

$$\zeta(t) = \frac{\left(\delta_d(t) + \delta_v(t) + \delta_l(t)\right)}{3} \quad (6)$$

This combination is regarded as one of the best strategies to define panic situations.

Also, for every $\zeta(t) > 0.5$, will be replaced by 1 i.e., $\zeta(t) = 1$.

Similarly, for every $\zeta(t) < 0.5$ will be replaced by 0 i.e., $\zeta(t) = 0$

In this way, the presence or absence of panic can be identified.

C. Model of Motion

This algorithm will define the movements of agents towards exit doors, away from the flames. It will show certain components [15] [16] that are considered to set agents in motion such as position and velocity. Various attributes with their respective symbols at any time t, used in this model are explained below.

a) r_i : Radius of particular agent i.

b) $\vec{c_i}$: Weight of agent *i*, which is used to calculate the physical force experienced by agent. It can be also stated as cohesion component.

c) $v^{\rightarrow}$: A vector which has both magnitude and direction and represents the instantaneous velocity of an agent *i*, at time *t*.

d) $\vec{p_i}$: Position vector of agent i.

e) ζ_i: It identifies the presence of panic in agent where, 0 means no panic and 1 means panic exists.

f) l_i: It gives the maximum acceptable distance to exit doors, visible to agents.

g) N_c : Count of close agents in group.

The radial distances that define three basic rules of boids and their behavioral scope, associated with an agent are defined as follows:

a) d_i^c*:* Radial distance required for cohesion behavior, and it represents certain boundary within a group.

b) d_i^s*:* Minimum distance required for separation among agents. It is used for displaying the separation factor.

c) d_i^a : Radial distance and a certain boundary of group, required for alignment factor.

Furthermore, refinement factors of agents varies by socio-psychological factor like panic and other physical factors such as velocity, position and its dimensions are defined below,

a) m_i^g: This factor shows the 'getting to the goal behavior of agent. Here, goal is the exit door which is decided and approached by agent.

b) m_i^c : This factor shows the herding behavior of agents.

c) m_i^s: This factor reflects the separation behavior of crowd, where an agent can separate itself from other agents.

d) m_i^a: This factor is related to alignment behavior where they show movement in a disciplined manner. By combining both motion factors of separation and cohesion, proper alignment can be achieved.

e) m_i^o: This factor shows the obstacle avoidance behavior of the agents, where there is a presence of obstacle in environment and agents avoids the obstacle to reach goal.

An agent i will first decide its goal and then determine the velocity of weighted goal, where m_i^g will be multiplied by goal velocity. Similarly, all rest components of boids components including separation, alignment and obstacle avoidance will be updated and multiplied by m_i^s, m_i^a and m_i^o , respectively. Steps to calculate motion component $v^{\rightarrow}$ are formalized below:

1) Initialize $c^{\rightarrow}$, $s^{\rightarrow}$, $a^{\rightarrow}$, $o^{\rightarrow}$ and vector $v^{\rightarrow}$ to 0.

2) Update *Cohesion* component: For each agent $j$in neighbouring distance d_i^c with condition $i \neq j$, the centre of mass $c^{\rightarrow}$ is calculated which is then added to the position of agents.

$$c^{\rightarrow} = c^{\rightarrow} + p_j^{\rightarrow} \quad (7)$$

The cohesion vector $c^{\rightarrow}$ is then updated and mean position will be calculated as follows,

$$c^{\rightarrow} = \frac{c^{\rightarrow}}{(N_c-1)} \quad (8)$$

3) Update *Separation* component: As each agent tries to match velocity of neighboring agent, then for each agent j within avoid distance $d_s^{\rightarrow}$,where $i \neq j$, the resultant repulsion vector is calculated as,

$$s^{\rightarrow} = s^{\rightarrow} - \left(p_j^{\rightarrow} - p_i^{\rightarrow}\right) \quad (9)$$

4) Update *Alignment* component: Now, the velocities of all agents is calculated and added to the alignment vector $a^{\rightarrow}$.The mean velocity with which agents align can be given within distance $d_a^{\rightarrow}$(where $i \neq j$) as,

$$a^{\rightarrow} = a^{\rightarrow} + v_j^{\rightarrow} \quad (10)$$

It will be updated for every agent and finally, give the following equation of mean velocity,

$$a^{\rightarrow} = \frac{a^{\rightarrow}}{(N_c-1)} \quad (11)$$

5) Update *Obstacle Avoidance* component: Now, for each agent j within avoid distance d_i^s, obstacle avoidance component can be given as,

$$o^{\rightarrow} = o^{\rightarrow} - \left(p_o^{\rightarrow} - p_j^{\rightarrow}\right) \quad (12)$$

6) Finally, all these components displaying rules of boids will be multiplied by their refinement factors and combined to update the final motion vector $v^{\rightarrow}$.

$$v^{\rightarrow} = (c^{\rightarrow} - p_i^{\rightarrow}).m_i^c + s^{\rightarrow}.m_i^s + (a^{\rightarrow} - v_i).m_i^a + o^{\rightarrow}.m_i^o \quad (13)$$

In this motion algorithm, the complexity is of order $O(n)$.

D. Simulation Algorithm

In this algorithm, flow of model will be presented. Simulation will start with first calculating the motion vector. It is then followed by adding the panic factor. Steps of evacuation procedure for each agent i in a room can be defined as:

1) Initialize goal velocity $v_g^{\to}$, agent velocity $v_i^{\to}$ and motion velocity $v_m^{\to}$ to zero.

2) Goal velocity can be obtained by multiplying the goal vector by m_i^g as,

$$v_g^{\to} = g^{\to}.m_i^g \quad (14)$$

3) Motion velocity $v_m^{\to}$can be obtained as,

$$v_m^{\to} = \left(c^{\to} - p_i^{\to}\right).m_i^c + s^{\to}.m_i^s + (a^{\to} - v_i).m_i^a + o^{\to}.m_i^o \quad (15)$$

4) Velocity of agent $i$is calculated by adding the goal and motion velocity.

$$v_i^{\to} = v_g^{\to} + v_m^{\to} \quad (16)$$

5) Now panic factor will be determined as, panicking agents will follow herding behavior of boids so, all other factor (except cohesion m_i^c) will be set to zero. Therefore, if ζ_i =1, then $v_i^{\to}$, m_i^s, m_i^a and m_i^o will be initialized again to zero, where $v_i^{\to}$ will be updated as $v_m^{\to}$

6) Position vector $p_i^{\to}(t+1)$ at time $(t+1)$ will be updated by adding position vector $p_i^{\to}(t)$ and velocity vector $v_i^{\to}(t)$ of agent i at time t as,

$$p_i^{\to}(t+1) = p_i^{\to}(t) + v_i^{\to}(t) \quad (17)$$

7) Velocity vector at time $(t+1)$ will remain same as velocity vector $v_i^{\to}(t)$ at time t.

$$v_i^{\to}(t+1) = v_i^{\to}(t) \quad (18)$$

In this simulation algorithm, the complexity is of order $O(n^2)$.

IV. Experiments

Experiments conducted in this model has parameters can be altered, but for analysis and evaluation identical setting of parameters is done as,

population size = 332

vision = 6 patches

minimum-separation distance = 2 patches

maximum-align-turn = 5 degrees

maximum-cohere-turn = 5.25 degrees

maximum-separate-turn = 6 degrees

Simulation will be performed in a black colored rectangular grid, whose distance is defined by patches. The yellow agents represent the pedestrians without panic and red agents are the pedestrians with panic. Exit doors are represented by green color and obstacles by red color. The scaling of the given parameters is done using Netlogo.

Case 1) Evacuation without obstacles

In this the agents are spread in room and will move towards the nearby available exit door. Different scenarios are considered here. This case describes the movement of agents in a room without any obstacle.

a) One exit door: Here, the number of door present is one. It provided evacuation time of 73.8 seconds with predefined fixed parameters.

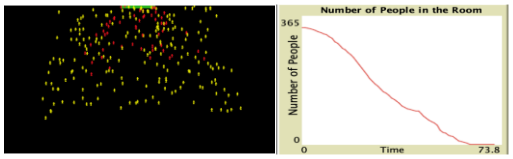

Fig. 1. Simulation of One exit doors without obstacle

b) Two exit doors: It has two doors on upper and lower sides. Due to visibility of agents in crowd to the nearest door, the crowd will get divided and hence, result in evacuation time of 58.8 seconds.

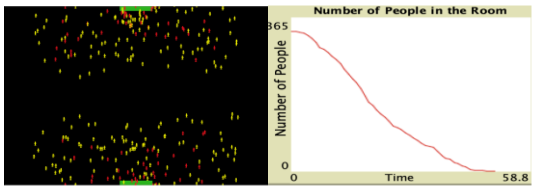

Fig. 2. Simulation of Two exit doors without obstacle

c) Two-Corner exit doors: It has two doors but in opposite corners. Agents have less visibility of the exit doors due to its design. Therefore, the evacuation times takes 184 seconds. Some agents get stuck near to the walls when most of the population escaped. So, they show no alignment motion and takes more time to find the nearest exit.

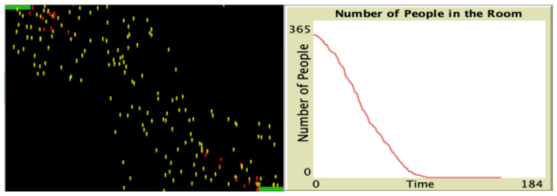

Fig. 3. Simulation of Two-Corner exit doors without obstacle

Case 2) Evacuation with obstacles

This case displays different orientation of doors in room in presence of red colored obstacle

a) One exit door and obstacle: This simulation has one door and one obstacle. Due to presence of obstacle, time of evacuation will get affected and will provide resultant time as 116 seconds.

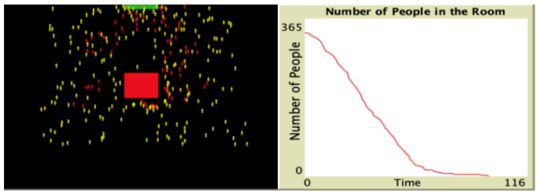

Fig. 6. Simulation of One exit doors with obstacle

b) Two exit doors with obstacle: This has two exit doors on opposite walls and one obstacle. The visibility is divided by vertical plane. Therefore, the evacuation time taken will be 73.8 seconds.

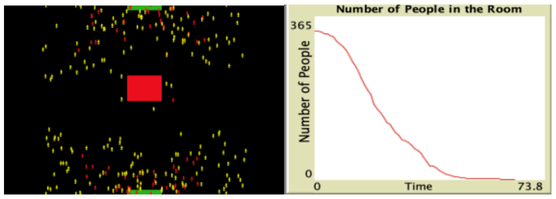

Fig. 5. Simulation of Two exit doors with obstacle

c) Two-Corner exit doors with obstacle: This orientation has two exit doors located at two opposite corners of room and in presence of one obstacle in center. It takes 231 seconds to evacuate the room.

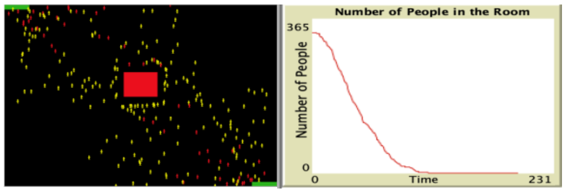

Fig. 6. Simulation of Two-Corner exit doors with obstacle

Case 3) Evacuation in complex design

This design is complex and has four random doors with five violet-colored obstacles which are randomly located as well. Moreover, the red spot in the upper right corner represents fire. Therefore, in this case, the agents will move towards the nearest exit door following the boid behavior, they will avoid obstacles as well as fire flames. The evacuation time taken by this complex scenario is 92.5 seconds.

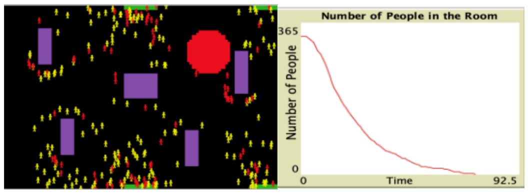

Fig. 7. Simulation of Complex design

V. Results

The evaluation is done to find the best orientation of room. The results are observed with four perspectives depending upon the number of doors, presence of obstacle, orientation of doors. In simulation, it can be noted that the speed of red agents is more than yellow due to panic factor, as they will try to escape as early as possible.

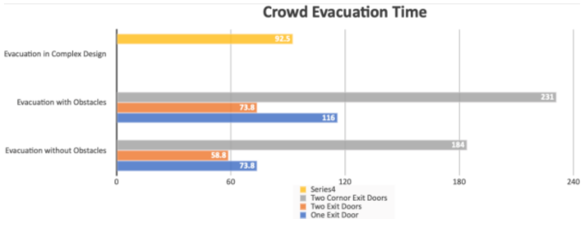

Fig. 8. Graph displaying evacuation time in different cases.

A. Effect of Obstacle:

Fig. 8. shows comparison of with and without obstacle between different number doors. It is visible that having obstacle in the room increased the evacuation time in all three scenarios with one, two and two-corner exit doors.

B. Effect of door location:

From Figure (8), it can be said that even when two doors are present but their positions are different, the evacuation time will vary. As in part b) of case 1), where the doors are present on opposite walls and agents have more visibility, the evacuation time is noted to be 58.8 which is the least among all. However, in part c) of case 1) where, doors are located at two opposite corners, the visibility of agents decreases which further increased the evacuation time by great extent. Similar is the result provided by part a) and part b) of case 2).

C. Effect of door count

In Figure (8), the difference between number of doors is displayed. For example, in part a) and b) of case 1), it is noted that a greater number of doors decrease the evacuation time, when varying visibility of nearest door to the agent is provided. Similarly, in case of part a) and b) of case 2), the one door exit took 116 seconds, and two door exits took around only 73.8 seconds. Therefore, having variable visibility with a greater number of doors provides better evacuation time. Part c) of both case 1) and 2) took more time even having more doors, because they had less visibility of door and some agents stuck during the process. Case 3) with complex design, also proves that having more doors will deteriorate the evacuation time.

D. The complex design

This simulation took less time in comparison to various above implemented models as, it has more visibility and a greater number of doors. The obstacles cause delay but with proper visibility of agent, number and orientation of doors, delay is handled.

VI. Future work

More complex designs need to be implemented which will figure out more emergent properties and interesting parameters that affect the evacuation. Implementing hyperparameter tuning of boids parameters, where best combination of cohesion, alignment and separation factor will be computed in order to provide less evacuation time and safe escape from hazardous situations. The variation in panic should also be considered as panic level reduces with distance to the exit door. At last, visibility of the agents must be improved in all cases.

VII. Conclusion

The implemented model with different conducted experiments succeeded in performing quantitative and qualitative analysis of model. Quantitative analysis provides a glance at the physical factors like time and position and provides good results but combining it with qualitative analysis will give real-life results. Different orientation of doors and visibility of agents affect the evacuation time, even though same doors or obstacles are present. Panic also alters the performance of evacuation model; the panic agents try to escape faster due to fear of injury or fatality. During this process, they might injure themselves or other agents, but their fast evacuation decrease the evacuation time by providing space to other agents. Emergency situations such as fire, terrorist attack, chemical exposure earthquake and so on., can arise anytime. More improvements can be made as discussed above, for more accurate results. Therefore, better evacuation strategy is need of the hour and will help save lives in real-life scenarios. Applying artificial life *Boids* algorithm, the human behavior in crowd is efficiently achieved.

References

[1] E. L. Quarantelli, "The nature and conditions of panic," vol. 60, no. 3, pp. 267-275, 1954.

[2] A. Mintz, "Non-adaptive group behavior," *J. Abnormal Social Psychol,* vol. 46, no. 2, p. 150, 1951.

[3] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model,”, vol. 21, no. 4, pp. 25–34, Jul. 1987.," *ACM SIG-GRAPH Comput. Graph.,* vol. 21, no. 4, pp. 24-34, 1987.

[4] C. G. Q. Ji, "Simulating crowd evacuation with a leader-follower model," *Int. J. Comput. Sci. Eng. Syst.,* vol. 1, no. 4, pp. 249-252, 2007.

[5] N. J. N. B. R. P. E. Hart, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.,* Vols. SSC-4, no. 2, pp. 100-107, 1968.

[6] D. V. D. X. H. Y. Yang, "Optimal leader-follower control for crowd evacuation," in *Proc. 52nd IEEE Conf. Decision Control (CDC),* Florence, Italy, 2013.

[7] J. J. G. W. A.-a. A. C. Schultz, "Roboshepherd: Learning a complex behavior," *Proc. Robot. Manuf., Recent Trends Res. Appl.,* vol. 6, pp. 763-768, 1996.

[8] J.-M. L. N. M. A. O. B. Bayazit, "Better group behaviors using rule-based roadmaps," in *Algorithmic Foundations of Robotics*, Berlin, Germany, Springer, 2004, pp. 95-112.

[9] I. F. T. V. D. Helbing, "Simulating dynamical features of escape panic," *Nature,* vol. 407, no. 6803, pp. 487-490, 2000.

[10] R. R. Murphy, "Human-robot interaction in rescue robotics," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev,* vol. 34, no. 2, pp. 138-153, 2004.

[11] S. R. A. Trivedi, "Agent-Based Modeling of Emergency Evacuations Considering Human Panic Behavior," *IEEE Transactions on Computational Social Systems,* vol. 5, no. 1, pp. 277-288, 2018.

[12] N. R. Johnson, "Panic at ‘The Who concert stampede’: An empirical assessment," *Social Problems,* vol. 34, no. 4, pp. 362-373, 1987.

[13] E. L. Quarantelli, "The nature and conditions of panic," *Amer. J. Sociol.,* vol. 60, no. 3, pp. 267-275, 1954.

[14] J. P. Keating, "The myth of panic," *Fire J.,* vol. 76, no. 3, pp. 57-61, 1982.

[15] I. J. F. P. M. T. V. D. Helbing, "Simulation of pedestrian crowds in normal and evacuation situations," *Pedestrian Evacuation Dyn.,* vol. 21, no. 2, pp. 21-58, 2002.

[16] C. E. L. M. H. L. S. A. C. P. M. S. V. Viswanathan, "Quantitative comparison between crowd models for evacuation planning and evaluation," *Eur. Phys. J. B,* vol. 87, no. 2, pp. 1-11, 2014.

APPENDIX

To run the simulation and produce results, Netlogo is required. Netlogo is Programming language and Integrated development environment, used to implement the simulation of agent-based crowd evacuation using boids. The link to download the software is provided below:

https://ccl.northwestern.edu/netlogo/download.shtml

A. EVACUATION ONE DOOR

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 39 and pycor <=
40] [
    set pcolor green
    set room "exit"
    ;set pcolor yellow
  ]

  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 2.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        set target-door one-of patches with [room = one-of ["exit"] ]
     ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]

  reset-ticks
end

to go
```

```
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]


  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee

      set heading towards target-door + random-float 5
end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
end

;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
  set color red
  set speed speed + 0.3
  set fear? true
end

to fear
  ask n-of 100 turtles [panic]
end

to spread
  ask turtles with [fear? = true and color != red]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
end
```

B. EVACUATION TWO DOOR

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 39 and pycor <=
40] [
    set pcolor green
    set room "exit1"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= -40 and pycor <=
-39] [
    set pcolor green
    set room "exit2"
    ;set pcolor yellow
  ]



  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 2.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        set target-door one-of patches with [room = one-of ["exit1" "exit2"]
]
     ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]

  reset-ticks
end

to go
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit1" or [room] of patch-here = "exit2" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]


  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee

      set heading towards target-door + random-float 5

end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
end

;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
  set color red
  set speed speed + 0.3
  set fear? true
end

to fear
  ask n-of (floor(population / 3)) turtles [panic]
end

to spread
  ask turtles with [fear? = true and color != red]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
end
```

C. Evacuation Two Cornor Door

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
  ask patches with [pxcor >= -60 and pxcor <= -50 and pycor >= 39 and pycor
<= 40] [
    set pcolor green
    set room "exit1"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= 50 and pxcor <= 60 and pycor >= -40 and pycor <=
-39]  [
    set pcolor green
    set room "exit2"
    ;set pcolor yellow
  ]



  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 2.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        set target-door one-of patches with [room = one-of ["exit1" "exit2"]
]
     ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]

  reset-ticks
end

to go
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit1" or [room] of patch-here = "exit2" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]


  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee

      set heading towards target-door + random-float 5

end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
end

;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
  set color red
  set speed speed + 0.3
  set fear? true
end

to fear
  ask n-of (floor(population / 3)) turtles [panic]
end

to spread
  ask turtles with [fear? = true and color != red]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
end
```

D. Evacuation one Door with Obstacle

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 39 and pycor <=
40] [
    set pcolor green
    set room "exit"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 0 and pycor <=
10] [
    set pcolor red
    set room "wall"
    ;set pcolor yellow
  ]

  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 2.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        set target-door one-of patches with [room = one-of ["exit"] ]
     ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]

  reset-ticks
end

to go
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]


  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee
    ifelse count patches in-radius 5 with [pcolor = red ] > 0 [
      set heading (towards min-one-of patches with [pcolor = red ][distance
myself] + 90   )
      if [pcolor] of patch-ahead 1 != black [
        fd 2
      ]
    ]
    [
      set heading towards target-door + random-float 5
    ]
end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
end

;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
```

```
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
  set color red
  set speed speed + 0.3
  set fear? true
end

to fear
  ask n-of 100 turtles [panic]
end

to spread
  ask turtles with [fear? = true and color != red]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
End
```

E. Evacuation Two Doors with Obstacle

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 39 and pycor <=
40] [
    set pcolor green
    set room "exit1"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= -40 and pycor <=
-39] [
    set pcolor green
    set room "exit2"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 0 and pycor <=
10] [
    set pcolor red
    set room "Wall"
    ;set pcolor yellow
  ]

  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 2.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        set target-door one-of patches with [room = one-of ["exit1" "exit2"]
]
     ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]

  reset-ticks
```

```
end

to go
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit1" or [room] of patch-here = "exit2" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]


  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee
    ifelse count patches in-radius 5 with [pcolor = red ] > 0 [
      set heading (towards min-one-of patches with [pcolor = red ][distance
myself] + 90   )
      if [pcolor] of patch-ahead 1 != black [
        fd 2
      ]
    ]
    [
      set heading towards target-door + random-float 5
    ]
end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
```

```
end

;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
  set color red
  set speed speed + 0.3
  set fear? true
end

to fear
  ask n-of (floor(population / 3)) turtles [panic]
end
```

```
to spread
  ask turtles with [fear? = true and color != red]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
end
```

F. Evacuation Two Doors with Obstacle

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
   ask patches with [pxcor >= -60 and pxcor <= -50 and pycor >= 39 and pycor
<= 40] [
    set pcolor green
    set room "exit1"
    ;set pcolor yellow
  ]
   ask patches with [pxcor >= 50 and pxcor <= 60 and pycor >= -40 and pycor
<= -39] [
    set pcolor green
    set room "exit2"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 0 and pycor <=
10] [
    set pcolor red
    set room "Wall"
    ;set pcolor yellow
  ]

  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 2.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        set target-door one-of patches with [room = one-of ["exit1" "exit2"]
]
     ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]

  reset-ticks
end
```

```
to go
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit1" or [room] of patch-here = "exit2" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]


  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee
    ifelse count patches in-radius 5 with [pcolor = red ] > 0 [
      set heading (towards min-one-of patches with [pcolor = red ][distance
myself] + 90   )
      if [pcolor] of patch-ahead 1 != black [
        fd 2
      ]
    ]
    [
      set heading towards target-door + random-float 5
    ]
end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
end
```

```
;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
  set color red
  set speed speed + 0.3
  set fear? true
end

to fear
  ask n-of (floor(population / 3)) turtles [panic]
end

to spread
```

```
  ask turtles with [fear? = true and color != red]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
end
```

G. Evacuation in Complex Design

```
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  target-door
  speed
  fear?

]

patches-own [
  room
]

to setup
  clear-all
  ask patches [
  ; creating fire
    let X max-pxcor * 0.5
    let Y max-pycor * 0.5
    if distancexy X Y < 10 [set pcolor red]
  ]
  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 39 and pycor <=
40] [
    set pcolor green
    set room "exit1"
    ;set pcolor yellow
  ]
  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= -40 and pycor <=
-39] [
    set pcolor green
    set room "exit2"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= 50 and pxcor <= 60 and pycor >= -40 and pycor <=
-39] [
    set pcolor green
    set room "exit3"
    ;set pcolor yellow
  ]
  ask patches with [pxcor >= 50 and pxcor <= 60 and pycor >= 39 and pycor <=
40] [
    set pcolor green
    set room "exit4"
    ;set pcolor yellow
  ]

  ask patches with [pxcor >= -7 and pxcor <= 7 and pycor >= 0 and pycor <=
10] [
    set pcolor violet
    set room "Wall1"
    ;set pcolor yellow
  ]
  ask patches with [pxcor >= 42 and pxcor <= 47 and pycor >= 2 and pycor <=
20] [
    set pcolor violet
    set room "Wall2"
    ;set pcolor yellow
  ]
  ask patches with [pxcor >= -45 and pxcor <= -40 and pycor >= 15 and pycor
<= 30] [
    set pcolor violet
    set room "Wall3"
    ;set pcolor yellow
  ]
  ask patches with [pxcor >= 20 and pxcor <= 25 and pycor >= -30 and pycor <=
-15] [
    set pcolor violet
    set room "Wall4"
    ;set pcolor yellow
  ]
  ask patches with [pxcor >= -35 and pxcor <= -30 and pycor >= -25 and pycor
<= -10] [
    set pcolor violet
    set room "Wall5"
    ;set pcolor yellow
  ]

  ask n-of population patches with [pcolor = black][
      sprout 1   [
        set shape "person"
        set color yellow   ;; random shades look nice
        set size 3.0  ;; easier to see
        set speed 0.3
;        setxy random-xcor random-ycor
        set flockmates no-turtles
        let d1 [ distance myself ] of patches with [room = "exit1"]
        set d1 min d1
        let d2 [ distance myself ] of patches with [room = "exit2"]
        set d2 min d2
        let d3 [ distance myself ] of patches with [room = "exit3"]
        set d3 min d3
        let d4 [ distance myself ] of patches with [room = "exit4"]
        set d4 min d4

        if d1 < d2 and d1 < d3 and d1 < d4 [
          set target-door one-of patches with [room = "exit1"]
        ]
        if d2 < d1 and d2 < d3 and d2 < d4 [
          set target-door one-of patches with [room = "exit2"]
        ]
        if d3 < d1 and d3 < d2 and d3 < d4 [
          set target-door one-of patches with [room = "exit3"]
        ]
        if d4 < d1 and d4 < d2 and d4 < d3 [
          set target-door one-of patches with [room = "exit4"]
        ]
        if target-door = 0 [
           set target-door one-of patches with [room = one-of ["exit1"
"exit2" "exit3" "exit4" ] ]
        ]
    ]
  ]
  fear
;  create-turtles population
;  [
;      if pcolor = black [
;        set color yellow   ;; random shades look nice
;        set size 1.5  ;; easier to see
;        setxy random-xcor random-ycor
;        set flockmates no-turtles
;        set target-door one-of patches with [room = one-of ["exit"] ]
;      ]
;  ]
```

```
  reset-ticks
end

to go
  ask turtles [
    flock
    flee
    if [room] of patch-here = "exit1" or [room] of patch-here = "exit2" or
[room] of patch-here = "exit3" or [room] of patch-here = "exit4" [
      if any? turtles-here or any? turtles-here in-radius 5 [
        ask turtles-here [
          die
        ] ; make turtles disappeared when there are evacuated
      ]
    ]
    spread
  ]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd speed ] display ]

  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  let pop count turtles
  if pop = 0 [stop]
  tick
end

to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere
        ]
    ]
end

to flee
  if count patches in-radius 5 with [pcolor = red ] > 0 [
    panic
    set heading (towards min-one-of patches with [pcolor = violet ][distance
myself] + 90 )
    if [pcolor] of patch-ahead 2 != black [
        fd 2
    ]
  ]
  ifelse count patches in-radius 5 with [pcolor = violet  ] > 0 [
      set heading (towards min-one-of patches with [pcolor = violet ][dis-
tance myself] + 90 + random-float 5 )
      if [pcolor] of patch-ahead 2 != black [
        fd 2
      ]
    ]
    [
      set heading towards target-door + random-float 5
    ]
end

to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
```

```
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-turn
end

;;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of flockmates
  let y-component mean [cos (towards myself + 180)] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to panic
```

```
  if  color != red [
   set color red
   set speed speed + 0.2
   set fear? true
  ]
end

to fear
  ask n-of floor(population / 3) turtles [panic]
end

to spread
  ask turtles with [fear? = true and shape = "person"]
  [ask other turtles-here with [ fear? = false ]
    [if (random-float 100) < 50
      [panic]
    ]
  ]
end
```