
# Crowd Egress Simulation

CS166 - Spring 2020

Ahmed Abdelrahman

# Table of Content

# Section-1 Two-dimensional Cellular Automaton Model

## Introduction:

Crowd characteristics and movements are an essential field of research due to their many real-life applications and benefits. This paper provides a detailed simulation analysis of Crowd Egress using Python implementation and MontoCarlo principles. Utilizing cellular automaton modeling principles, the paper builds and analyzes multiple experiments to different realistic scenarios. The paper concludes by illustrating the best actions for room egress by explaining the relationship between simulation parameters and the time taken to clear the room completely.

## Description of the model:

The model is using a two-dimensional CA simulation to model the movement of people as they evacuate a room, responding to a fire-alarm. This dissertation aims to define and validate a model to measure the time taken for evacuation and the factors involved in the process.

**Rules:**

Observing the architecture of buildings and rooms in real life, we have seen that many of them have only front and back exits- doors in two opposite edges of the building. We have built the model to simulate a living room or office with exits only in the front and the back of the room. The room is set as a two-dimensional grid, where each cell can have no more than one

person. The room is occupied with a number of people with random and unique positions, where each person can move in one of four directions (up, down, right, and left). Exits of the room can be randomly placed or placed with a predetermined strategy.

As time proceeds, people update their positions in the grid to reach the nearest exit to them as fast as possible. All the decisions are taken at the same time via a synchronous updating. The model is using the current state and updates a new state to make sure all people are using the information from the previous time step to make their decisions.

**Assumptions:**

1. The room dynamics can be represented by a cellular automaton, e.g., a spatially and temporally discrete model, as there is finite reaction time.

2. People are rational. People will always try to leave the room and move toward the nearest exit to them.

3. People have similar physical and mental characteristics. People can only move one step at a time to any of their neighbors or stay at the same place based on the faced situation.

This assumption may not hold in real-life situations, where people are diverse in their abilities. The assumption was added to the model to simplify the interactions between all agents. To consider people's differences, more parameters can be added to specify people's speeds and moving abilities.

4. People live in a safe space. Everyone is trying to escape the room, but they don't hurt or intervene with each other.

**Parameters:**

1. *Number of exits*

This parameter represents the total number of exits in the room. The parameter takes an integer with a default value of 6 exits. In a fixed-size room, as we increase the number of exits, people can find closer exits to their locations, which will be directly reflected in the total evacuation time.

2. *Number of people*

This parameter is responsible for the total number of people placed in the room. It takes an integer with a default value of 25 people. The parameter will directly affect the total time needed to clear the room. I am assuming that as we increase the number of people in a room, the total evacuation time will increase as well. The number of people can be implemented as a ratio of the room size, but I decided to make it as an integer to be easier for the user to use and interpret.

3. *Exit size*

The parameter is sitting the size of each exit (the number of cells that the exit covers). It takes an integer with a default value of 1 cell-size. This parameter is important as it controls how many people can use the exit at the same time. I am assuming that as we increase the size of exit, the less time it will take to evacuate.

4. *Exit randomness*

The parameter determines the chosen way of placing exits inside the room. The parameter takes a boolean value (True or False) with a default value as True.
If True, a random side of the two prespecified sides (right and left) will be chosen. Then, a random index along the chosen edge will be picked to place the exit with the right size.
If False, exits will be placed along the two edges with a deliberate strategy:

- Exits will be placed along each side with equal spaces between them to cover as much area as possible of the side.
- Exits can NOT be attached. Exits will have at least one cell between them. The rule is added to represent real-life situations where there is always space for the exit door to open.
- Sides will have a similar number of exits. At most, an edge will have +1 exit more than others.

E.g, if we are placing 6 exits of size 1 in a (10x10) room, then exits will have these coordinates: [[0, 0], [3, 0], [6, 0],[0, 9], [3, 9], [6, 9]]

I am assuming that placing exits this way will help people leave the room faster, as I am placing exits to cover the entire edge instead of concentrating them in one place. Detailed analysis and comparison of the parameter affected will be discussed later.

5. *Room height and width*

These two parameters determine the two dimensions of the room (height, width). They can be chosen to be any integer with a default value of 10 cells each. The bigger the room, the more time it takes people in the middle of the room to evacuate.

6. *Wall size and direction*

Two parameters to provide the option of placing a random wall in the room with a vertical or horizontal direction. Wall size takes an integer with a default value of 0, Wall directions takes a string with a default letter as “v”. Adding walls and obstacles in the room can increase the time needed to evacuate the room.

7. *Panic ratio:*

A stochastic parameter to add randomness and indeterministic behavior to the evacuation process. It takes a ratio between 0 and 1, with a default value of 0.05. If a person is panicked they will not move.

**Output/Measurements:**

In evacuation scenarios, the most important results to think about is getting people out safe and as fast as possible. Considering a non-dangerous situation where people will not get hurt or injured, the simulation will focus on the time taken to evacuate the room. The simulation will measure evacuation time (simulation steps) on various scenarios by varying parameters' inputs. In the following sections, I will perform different experiments to identify parameters effects on the evacuation time using simulation and statistical tools.

**Monte-Carlo Simulation:**

As the evacuation process is a real-life event that includes much randomness, the proposed simulation method is designed to follow Monte Carlo simulation principles. It is very hard to predict or come up with a mathematical equation to describe people's behaviors during room egress situations. The model generates its initial configuration by randomly placing people and exits in the room utilizing random python generators. Also, it has a stochastic parameter that influences people's movements in the room and captures the indeterministic behavior of the event. In real-life, there are many people involved in room egress, which make the outcome of the event not a single time, but rather a probability distribution of all times. The results of the simulation will be presented as a distribution; then, it will be analyzed to illustrate the effects of the parameters on the evacuation process.

## Simulation Results Interpretation:

I have performed four experiments on the simulation to evaluate the effect of the discussed parameters on evacuation time. In each experiment, I have varied the values of only one parameter while fixing the rest of the parameters at their default values. The simulation iteration values were determined based on my device computational ability and time constraints, with a total amount of 1400 runs in each experiment. The experiments' method is designed to identify and generalize the effect of each parameter on the evacuation time.

*Running the experiment multiple times can have slightly different results due to the randomness of Monte-Carlo simulations. However, the obtained distributions will have similar behaviors and statistical characteristics.

### Experiment-1 ( Number of Exits Effect) :

While fixing the simulation parameters at their default values, I have varied the number of exits between (1-14) with 1000 iterations for each of these values.

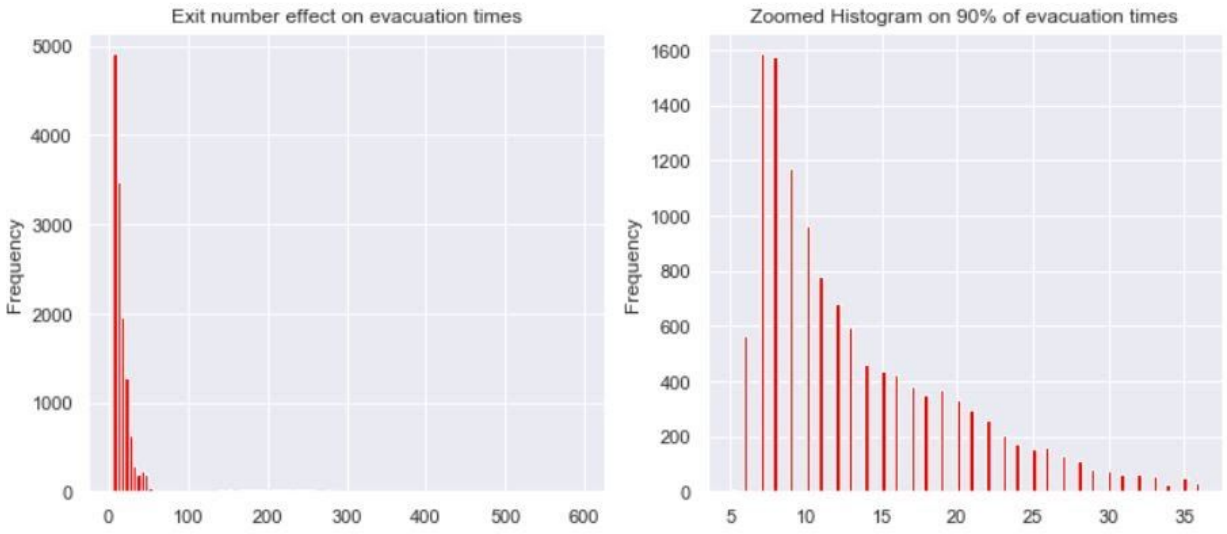



Figure 1: A distribution showing the effect of number of exits on evacuation time.

The obtained histogram of the simulation results (times) has a minimum value of 5 and a maximum value of 595; it's heavily skewed to the right with a skewness value of 4.58. This is reflected in the mean of the data as 27 time-steps and the mode as 7 time-steps. Although the maximum value is huge, we can see that 90% of the results fall in the range of [0:36], as shown in the zoomed histogram. Rare but very large time results mainly cause the wide range of the data, due to a small number of exits in the room.

For each value between (1-14), I have calculated the measures of central tendency (Mean, Mode, Median). The results are shown below:

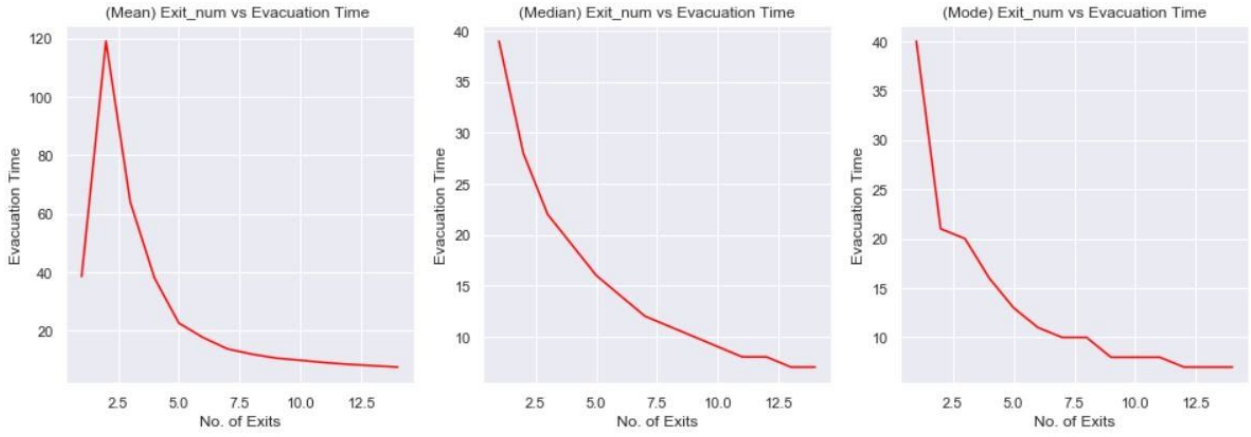



Figure 2: Graphs showing the effect of number of exits on evacuation time.

The mode and the median of the data are smaller than the mean, which illustrates the right skewness of the data distribution. As shown in the graphs, there is an inverse relationship between the number of exits and the time taken to evacuate the room. This can be observed in real-life scenarios, as the door number decreases, more people need to wait for others to leave first. In the (Mean) graph, there are very high time values when the number of exits is small due to the rarely observed values in the distribution.

**Experiment-2 ( Number of People Effect) :**

While fixing the simulation parameters at their default values, I have varied the number of people between (1-28) with 500 iterations for each of these values.

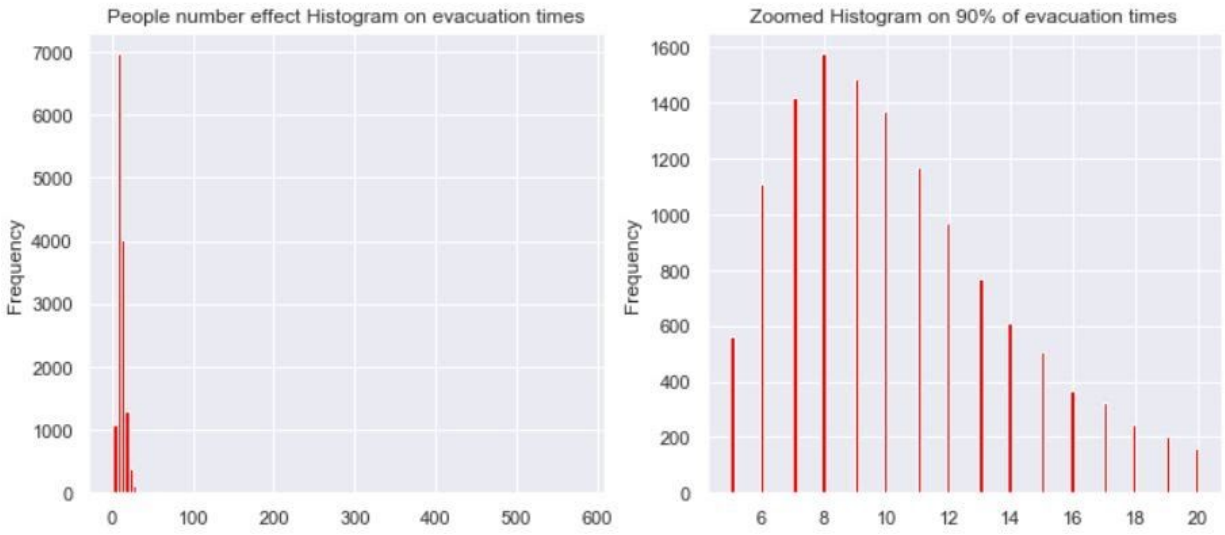



Figure 3: A distribution showing the effect of number of People on evacuation time.

The obtained histogram of the simulation results (times) has a minimum value of 1 and a maximum value of 579; it's heavily skewed to the right with a skewness value of 12.3. This was reflected in the mean of the data as 12.5 and the mode as 8 time-steps. Although the maximum value is huge, we can see that the 95th percentiles of the data is 20 which means that 95% of the data are less than 20 time-steps. The wide range of the data was mainly caused by rare, but very large, time results due to a large number of people at the same place.

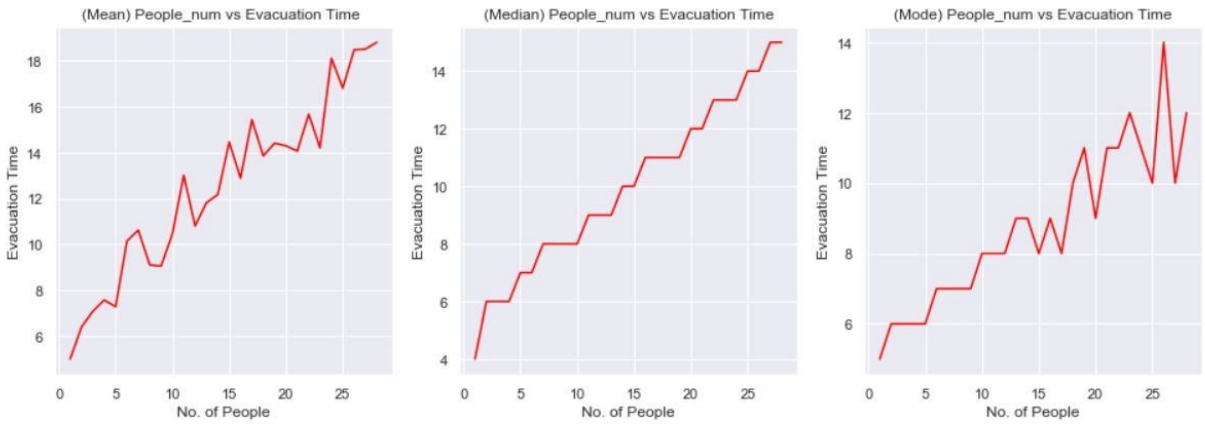



Figure 4: Graphs showing the effect of number of People on evacuation time.

The mode and the median of the data are smaller than the mean, which illustrates the right skewness of the data distribution. As shown in the graphs, there is a direct relationship between the number of people and the time taken for them to evacuate the room. As the number of people increases, it will take them more time to evacuate the room with a fixed number and size of exits. The fluctuation of the data can be explained by the randomness of people's original location in the room and how it affects the evacuation time.

**Experiment-3 (Exit Size Effect) :**

In a predetermined room size (10x10) and a fixed number of exits, the exit size can't exceed a specific value. I have designed this experiment to be a (10x10) room with only two exits, one exit on each edge to be able to capture the exit size effect on the evacuation time without considering other factors. Exit size values vary between (1-10) with 1400 trails for each exit.

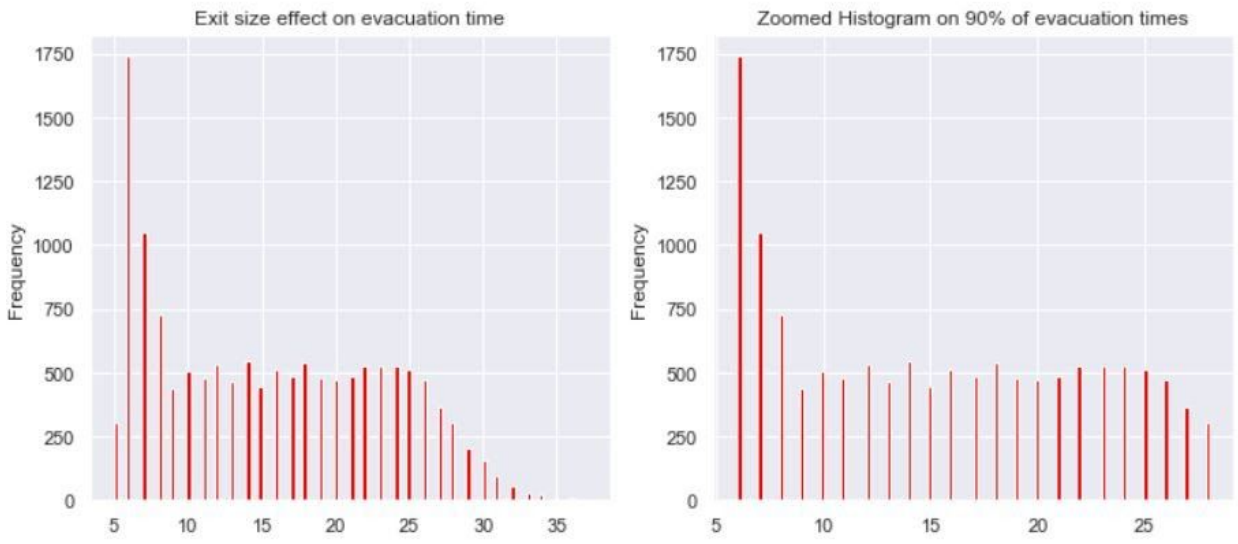



Figure 5: A distribution showing the effect of exit size on evacuation time.

The obtained histogram has a minimum value of 5 and a maximum value of 37; it's slightly skewed to the right with a skewness value of 0.2. The mean of the data is 15.5, and the mode is 6 time-steps. The data are not widely spread, which means that it follows a stable behavior that prevents it from random rare values.

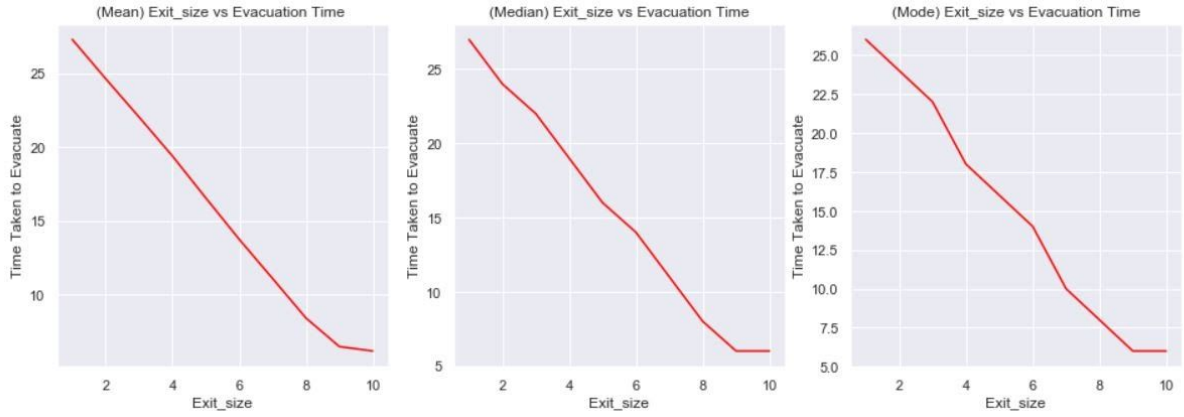



Figure 6: Graphs showing the effect of Exit size on evacuation time.

The mode and the median of the data are almost matching the mean, which illustrates the small skewness of the data distribution. As shown in the graphs, there is a very strong inverse relationship between the exit size and the time taken for them to evacuate the room. As the size of the room gets bigger, more people can leave at the same time. The smoothness of the data is due to the experiment method, where randomness is almost eliminated by pre-determining exit locations.

**Experiment-4 (Room Dimensions Effect) :**

While fixing the simulation parameters at their default values, I have varied the height and the width of the room between (8-22) with 1000 iterations for each of these values. In the experiment, I am assuming scenarios in squared rooms where (height=width), but other configurations can be applied as well.

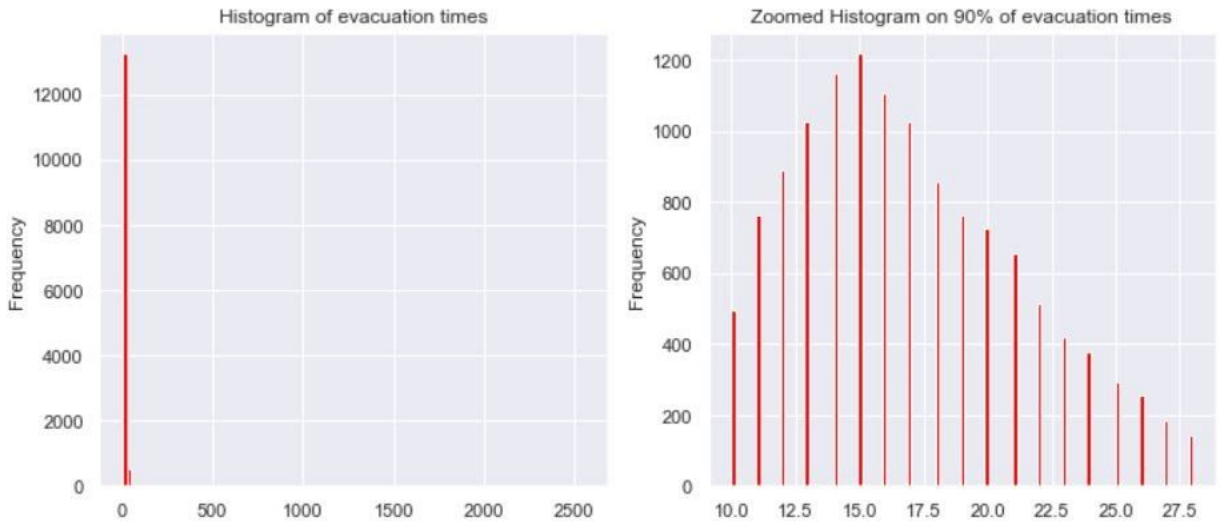



Figure 7: A distribution showing the effect of room size on evacuation time.

The obtained histogram has a minimum value of 6 and a maximum value of 2551; it's skewed to the right with a skewness value of 11.75. The mean of the data is 28.1, and the mode is 15 time-steps. Although the maximum value is enormous, we can see that the 95th percentiles of the data have a value of 28, which means that 95% of the data are smaller than this value. The data are widely spread with some very high results, which explains the distribution behavior.

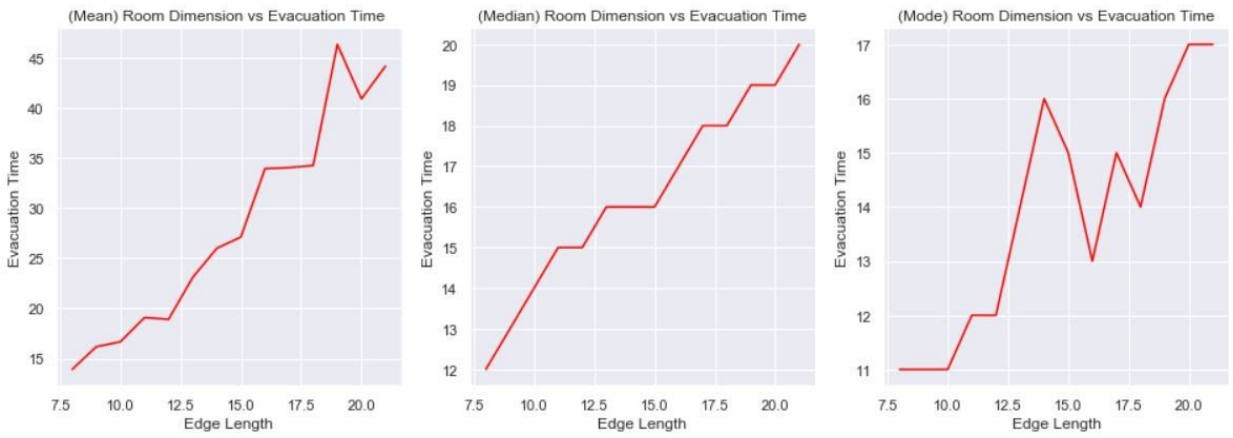



Figure 8: Graphs showing the effect of room size on evacuation time.

The mode and the median of the data are smaller than the mean, which illustrates the right skewness of the data distribution. There is a direct relationship between the number of exits and the time taken to evacuate the room. As the room size increases, people will take more time to reach exits that are placed on the edges of the room. The (Mode) graph shows that when the Edge length is almost 14, there are many observations with high evacuation time.

**Experiments Uncertainties:**

Running Monte Carlo simulations can help to understand and generalize the model results due to the high number of iterations. I have calculated the 95% confidence interval on the experiments' results to quantify the certainty of the obtained results. A 95% confidence interval is a range of values that we can be 95% certain it contains the true mean of the data. This is not the same as a range that includes 95% of the values. It is a measure of how confident I am that the result is within the specified range. For a 95% confidence interval if we repeated an experiment 100 times and checked to make sure there were no gross errors, 95 of the repeats would be expected to have results within the range of the confidence interval.The uncertainty range of each experiment is calculated as the difference between upper and lower bounds of its 95% confidence intervals. The 95% confidence intervals are calculated using this equation: $\bar{x} \pm Z\frac{S}{\sqrt{n}}$ , where x is the mean, z is the chosen Z-value (1.96) from its table, S is the standard deviation and n is the number of observations.

| | Mean | 95% CI | Uncertainty-Width |
|---|---|---|---|
| Number of Exits exp | 27 | (26.1,27.9) | 1.8 |
| Number of People exp | 12.5 | (12.2,12.9) | 0.7 |
| Exit Size exp | 15.6 | (15.5,15.7) | 0.2 |
| Room Dimensions exp | 28 | (26.4,29.6) | 3.2 |

Table-1. A 95% confidence interval table, where all values are rounded to hundredths place

The experiments' confidence intervals are narrow with low uncertainty compared to the data variance. The low uncertainty is due to the large number of simulation steps used to obtain the results and generalize the data. The widths of the confidence intervals can be even reduced more by increasing the number of simulation steps, but this will require more computational power. In generals, larger studies tend to give more better estimates of effects than smaller studies

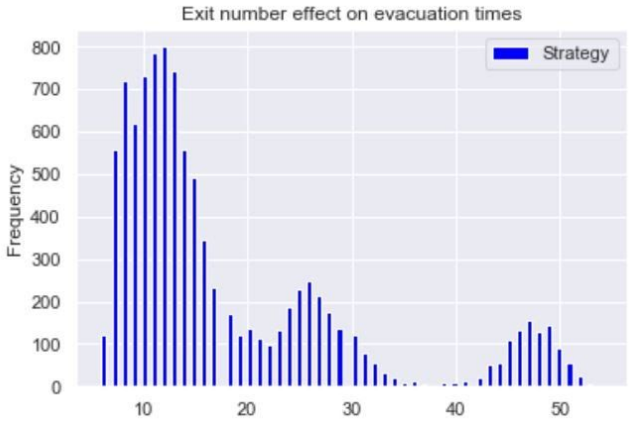
# Section-2 Exits Placement Strategies Comparison

In this section, I will compare two strategies of placing exits inside the room. The first strategy is randomly placing exits, and the second one is following the strategy mentioned in the description of the exit_randomness parameter. The deliberate strategy aims to optimize the distance between exists along each edge and improve the evacuation process. The parameters' values of each experiment will be designed to reflect the predetermined strategy rules. E.g., we can't place more than 5 exits in a 10-meter long edge if we are leaving a distance of at least one cell between exits.

## Simulation Results Interpretation:

**Experiment-5 (Number of Exits Effect) :**

While fixing the simulation parameters at their default values, I have varied the number of exits between (1-10), with 1000 iterations for each of these values, and compared the proposed and the random strategies of exit placement.



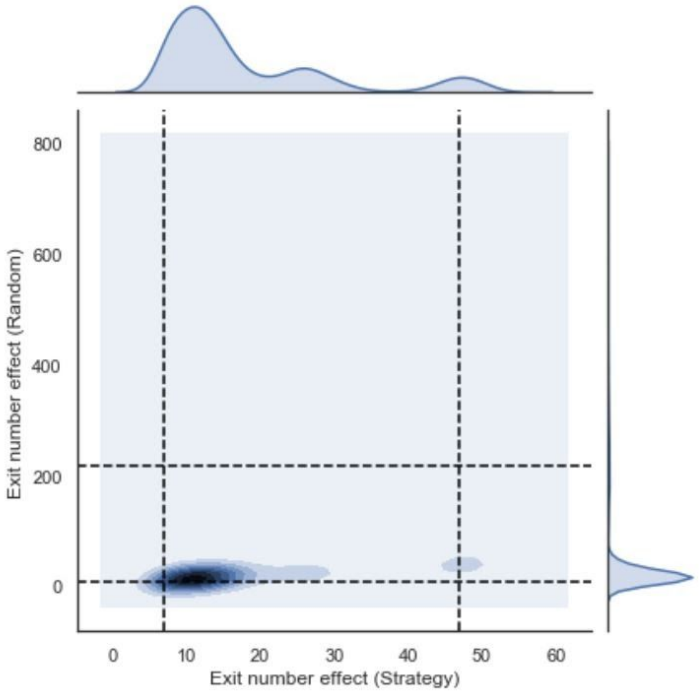




Figure-9 showing the distribution of the time obtained by exp-5. Second graph is a density estimate of the joint probability of exp-1 and exp-5. The quantiles (0.05 and 0.95) are pointed in the dashed black lines.

The obtained histogram has a minimum value of 6 and a maximum value of 54; it's skewed to the right with a skewness value of 1.5. The mean of the data is 18, and the mode is 12 time-steps. The data is concentrated in a small range compared to data generated from random exits placement (exp-1). In the joint probability figure, we can see how the random strategy has a long and thin tale compared to the proposed strategy. Randomly placing exits can result in high evacuation time which was illustrated in the distribution and the large area between (0.05 and 0.95) quantiles. This illustrates the effectiveness of the proposed strategy to avoid hazardous cases of slow evacuation time.

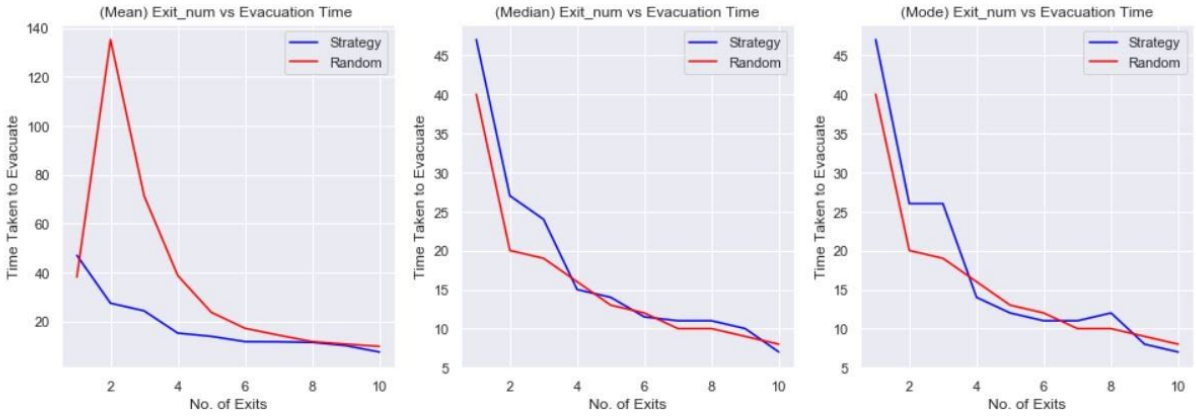
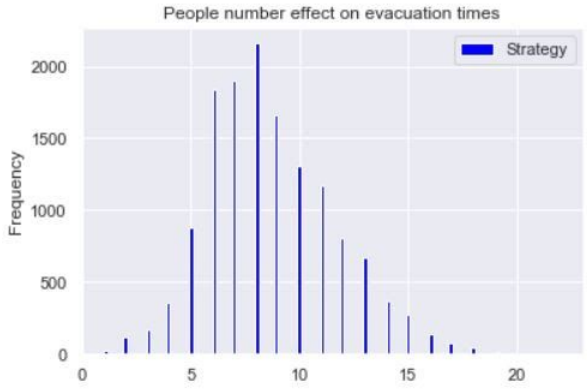



Figure10 : Graphs showing the effect of exits number on evacuation time.

As shown in the graph, the proposed strategy of placing exits has a smaller evacuation meantime compared to randomly placing exits in the room. Also, the strategy has a more stable inverse relationship with evacuation time compared to the random placing in experiment 1. The median and the mode of both strategies are similar to the similarity of the settings of the experiments.

**Experiment-6 ( Number of People Effect) :**

While fixing the simulation parameters at their default values, I have strategically placed exits in the room and varied the number of people between (1-28). Then, I have compared the obtained results with the one from experiment 2.



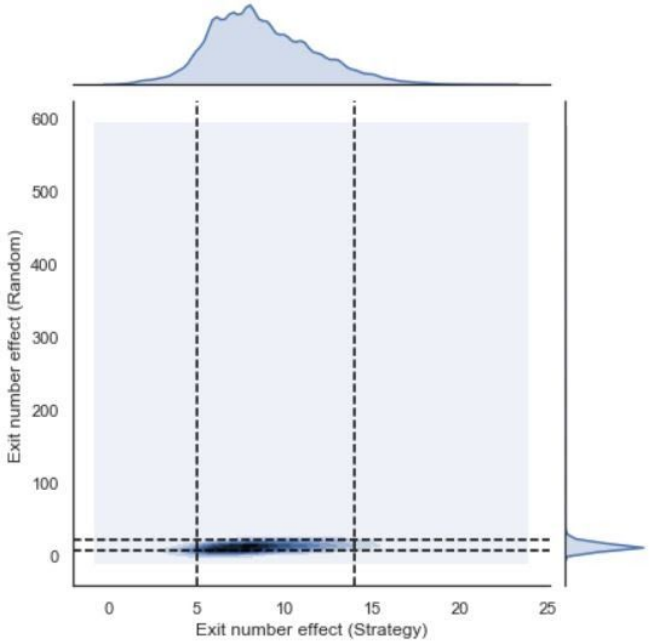




Figure-11 showing the distribution of the time obtained by exp-6. Second graph is a density estimate of the joint probability of exp-2 and exp-6. The quantiles (0.05 and 0.95) are pointed in the dashed black lines.

The obtained histogram has a minimum value of 1 and a maximum value of 22; it's slightly skewed to the right with a skewness value of 0.5. The mean of the data is 8.7, and it is concentrated in a small range compared to data generated from random exit placement (exp-2). In the joint probability figure, we can see how both strategies have 95% of their data below 25 time-steps. However, in the random placement strategy, there are some extreme cases with massive time steps, which increased its mean value. This illustrates the effectiveness of the proposed strategy of maintaining stable results and avoiding extreme cases of very slow evacuation time.

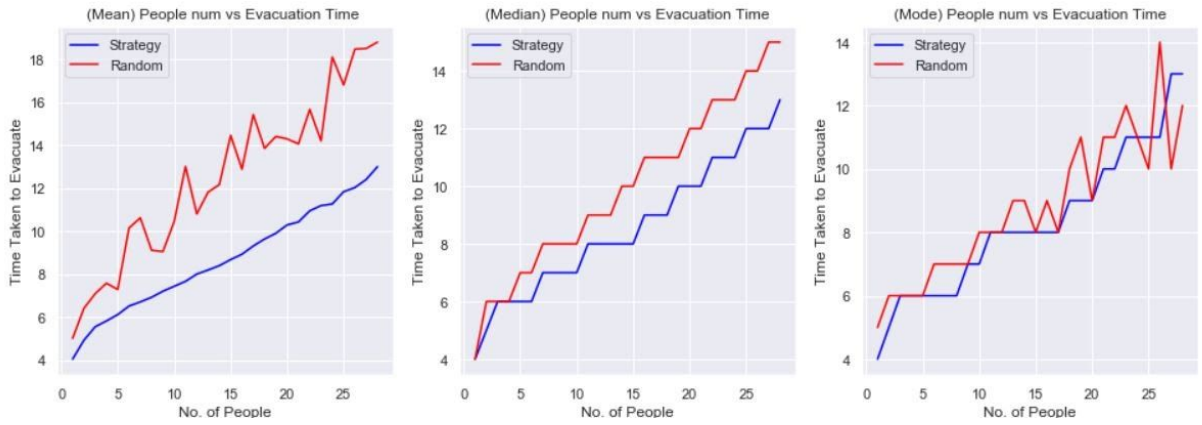



Figure-12 : Graphs showing the number of people effect on evacuation time.

The proposed strategy of placing exits has a smaller positive slope with the evacuation time. The strategy has a more stable direct relationship with evacuation time compared to the random placing in experiment 2. The smoothness of the line is due to the lack of randomness with the new strategy. The graph shows how strategically placing exits in the room helps reduce the time needed for evacuation.

**Experiment-7 (Room Dimensions Effect) :**

I have repeated experiment-4 but with placing the room exits based on a specific strategy rather randomly. This a crucial experiment to illustrate the effectiveness and the need for the proposed strategy to cover the room edges and optimize the time needed for evacuation.

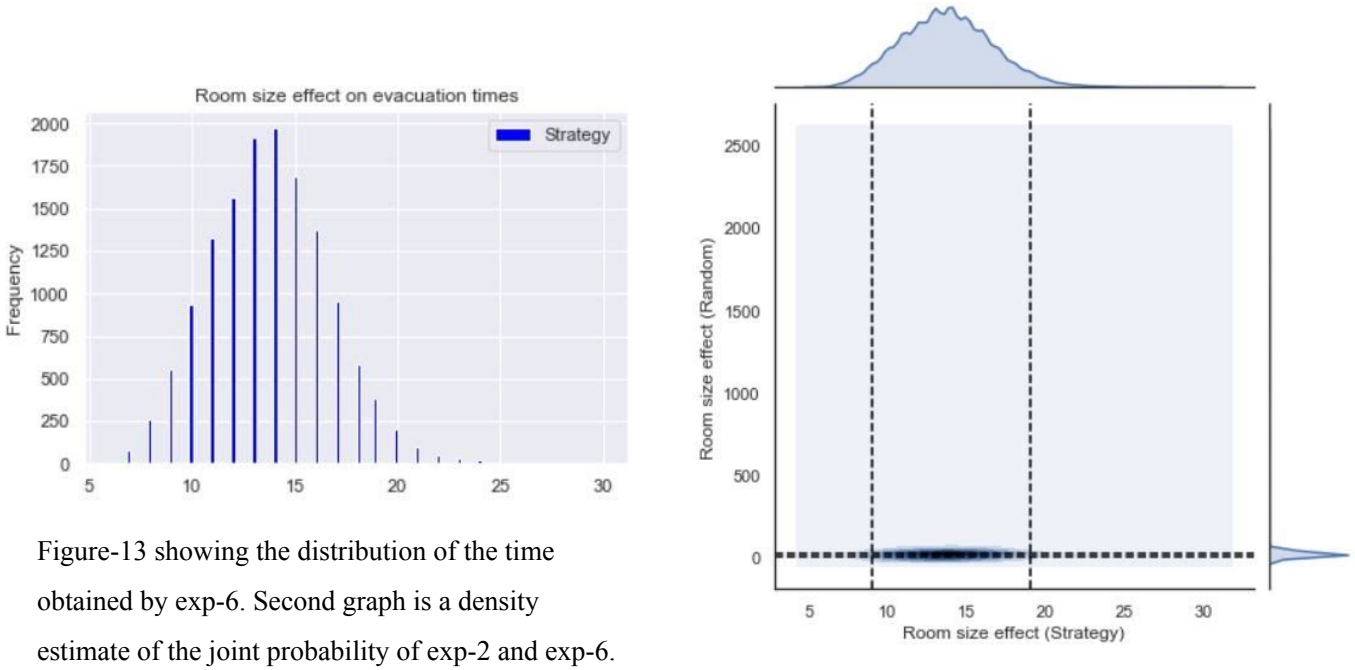



Figure-13 showing the distribution of the time obtained by exp-6. Second graph is a density estimate of the joint probability of exp-2 and exp-6. The quantiles (0.05 and 0.95) are pointed in the dashed black lines.

The obtained histogram has a minimum value of 6 and a maximum value of 30. It has a bell curve shape, but with a small skew to the right with a skewness value of 0.2. The mean of the data is 13.8, and it is concentrated in a small range compared to the histogram from experiment-4. Both strategies have 95% of their data below 35 time-steps. However, in the random placement strategy, there are some extreme cases with huge time steps, which increases its mean value. This illustrates the effectiveness of the proposed strategy of maintaining stable results and avoiding extreme cases of prolonged evacuation time. As we increase the room size with a fixed number of exits, it's crucial to have a specified strategy to optimize exit placements along with the room.

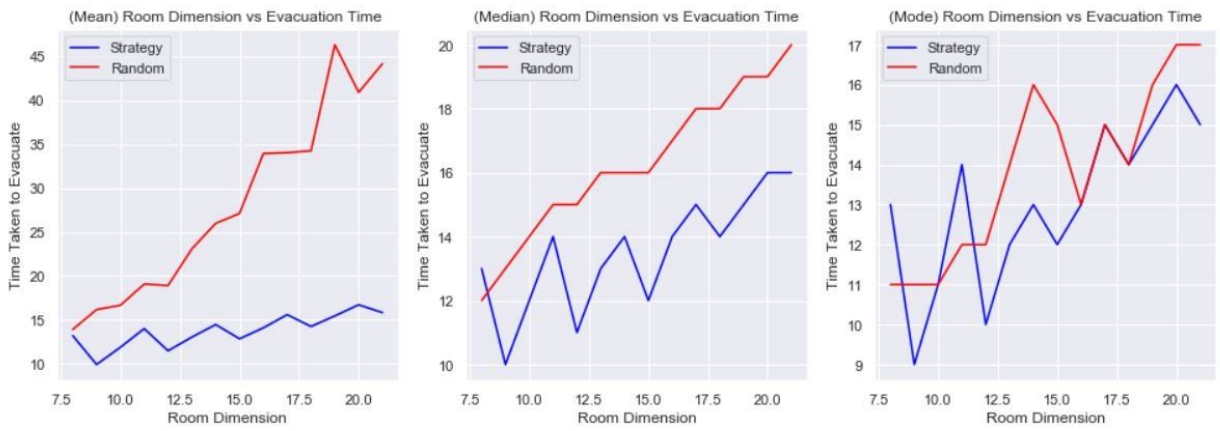



Figure-14 : Graphs showing room size effect on evacuation time.

The random strategy has a linear relationship that increases the evacuation time by increasing the room dimension. The proposed strategy has a similar relationship but a smaller slope. The plots show the effectiveness of the proposed strategy in optimizing the evacuation time and being resistant to changes and more able to be generalizable.

## Strategies Comparison:

| | Strategic Exit Placements (Mean) | Strategic Exit Placements (95% CI) | Random Exit Placements (Mean) | Random Exit Placements (95% CI) |
|---|---|---|---|---|
| Number of Exits exp | 18 | (17.8, 18.2) | 37 | (35.7, 38.5) |
| Number of People exp | 8.7 | (8.64, 8.74) | 12.5 | (12.15, 12.9) |
| Room Dimensions exp | 13.8 | (13.71, 13.81) | 28 | (26.46, 29.86) |

The table illustrates the effectiveness of strategically placing exits instead of placing them randomly. The strategy has outperformed randomness by having a shorter time to evacuate in all of the experiments shown above. This is due to the method of equally placing the exits between the edges and maximizing the distance between them so that they can cover more space of the side.

# Section-3 Model Extension

Rooms in real-life contain furniture, walls, or even obstacles that hinder the movement of people during the evacuation process. By varying the number of people in the room, I am going to explore the effect of having a vertical wall, versus not having one, on the evacuation time of the room.

## Simulation Results Interpretation:

**Experiment-8 ( Wall Effect) :**

While fixing the simulation parameters at their default values, I have varied the number of people between (1-28) with 500 iterations for each of these values. I have randomly added a vertical wall with a size of 5 cells in the room.

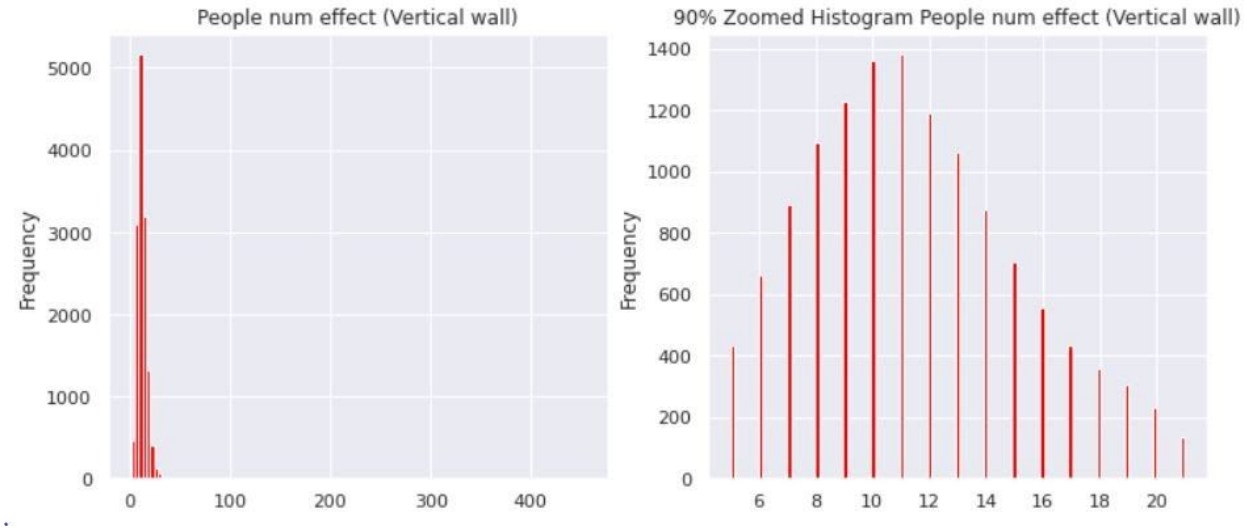



Figure 15: A distribution showing the effect of number of People on evacuation time, while having a vertical wall.

The obtained histogram of the simulation results has a minimum value of 1 and a maximum value of 454; it's heavily skewed to the right with a skewness value of 11.8. Although the

maximum value is huge, we can see that the 95th percentiles of the data is 21 which means that 95% of the data are less than 21 time-steps. The wide range of the data was mainly caused by rare, but very large, time results due to a large number of people at the same place.

I have varied the number of people in the room and compared having a wall vs not having a wall.

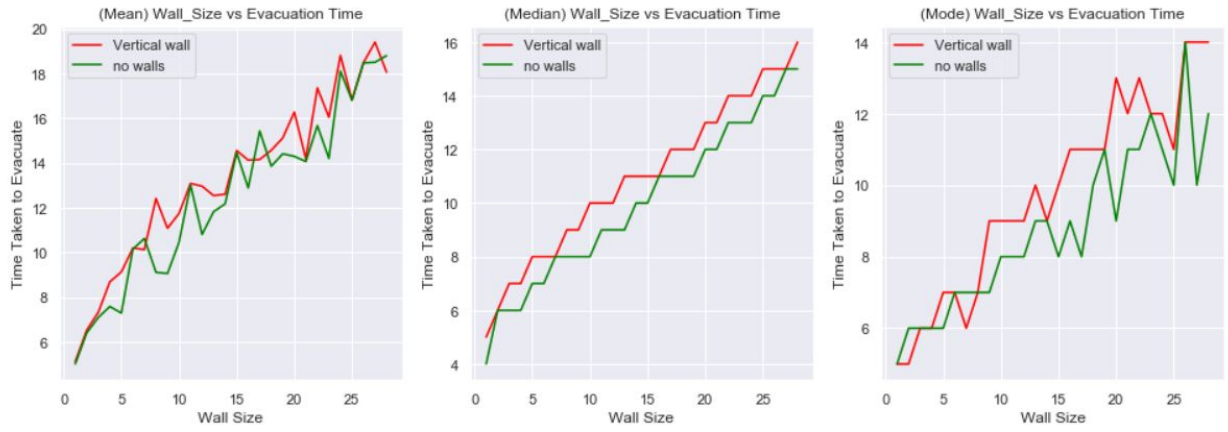



Figure-16 : Graphs showing wall effect on evacuation time.

As shown in the graph, having a vertical wall increases the time needed to evacuate the room. To be specific, the mean of evacuation time in a room with a wall is 12.2 versus 13 for a room without one. Having obstacles or walls in the room will make people take more time to evacuate. The effect of having a wall varies with the location, the randomness, and the size of it, which can be explored in further experiments in the future.

## Conclusion and Recommendation:

As illustrated in section-1, each parameter plays an important rule in determining the time needed to evacuate the room. Increasing the number of exits can help people leave in a faster way, as it allows them to find more exits closer to their initial locations. In cases when they don't, this is because the locations of exits are not optimized to be close to everyone. Instead, they are concentrated in one place. Also, as the size of exits increase, more people will be able to leave the room simultaneously without the need to wait for others. The dimensions of the room should be proportional to the size and the number of exits. Based on experiment-4, increasing room dimensions without increasing exits' sizes and numbers will only increase the time needed for evacuation. In big rooms, it's harder for the people to evacuate, especially for those in the middle of the room that needs to walk long distances to reach the exit. Finally, the number of people in the room plays a scaling factor for the rest of the parameters. More people in the room means more people are getting affected by the design of the room, which will be reflected in the evacuation time. In general, the time needed to clear the room increases by increasing the number of people in it.

The location of the exit plays an important rule in the evacuation process. As shown in section-2, strategically placing exits along the room edges helps to reduce evacuation time. We suggest following the strategy rules of maximizing the distance between exits instead of placing them randomly, which have proven to have less evacuation time in experiments (5,6,7) compared to experiments (1,2,4). In random exits allocation, the simulation has shown some extremely long evacuation times scenarios, which can be dangerous for the people inside the room. This can be avoided by optimizing exits allocation between the room edges and between the exits themselves as well.

Section-3 has demonstrated how walls, furniture, and obstacles can hinder the evacuation process and increase its time (Figure-16). Not only the size of the obstacle affects the evacuation time, but also the direction (Horizontal-Vertical) of it. Thus, the content and the purpose of the room needs to be considered in designing it.

Finally, the model is designed as a simplified simulation of the room egress scenarios in real-life. While interpreting the simulation results, the reader should consider the confidence interval and the uncertainty of the obtained results before making decisions. The model can be more realistic by considering more human characteristics such as speed, age, and competitiveness. These characteristics will make the model more capable of capturing human interactions in emergencies. Moreover, the model can explore different mechanisms and shapes of obstacles inside the room and how will this affect the evacuation process. Also, it can consider the dynamical and physiological behavior of the fire and smoke, which can influence people to take specific exit routes and movements. Nevertheless, the model simulates and analyzes the evacuation process and the relationships of various parameters with evacuation time

# Resources:

A Cellular Automaton Model for Crowd Movement and Egress ... (n.d.). Retrieved from https://duepublico2.uni-due.de/servlets/MCRFileNodeServlet/duepublico_derivate_00005477/Disskluepfel.pdf

Andy Connelly, Andy Connelly, & GLP Consulting Singapore. (2017, July 18). Uncertainty. Retrieved from https://andyjconnelly.wordpress.com/2017/05/16/uncertainty-and-repeats/

Bmayer0122Bmayer0122 1, shasanshasan 1, Ulrich SternUlrich Stern 7, bogatronbogatron 14.1k33 gold badges4343 silver badges4343 bronze badges, & Xavier GuihotXavier Guihot 18.2k1111 gold badges102102 silver badges7171 bronze badges. (1962, November 1). Compute a confidence interval from sample data. Retrieved from https://stackoverflow.com/questions/15033511/compute-a-confidence-interval-from-sample-data

# Appendix

**#confidenceintervals**: I have correctly calculated the confidence interval of the obtained distributions for my experiments using mathematical notations and python programing language with clear and detailed steps. I have used confidence intervals to explain and illustrate my certainty about the obtained results.

**#analogies:** The assignment was a great opportunity to apply this HC. I have considered room egress simulation to be a problem, then I started observing similar scenarios around me. By considering how humans think and move in emergencies, I have built an algorithm to simulate their movements. The reverse engineering of human movements in room egress, I was able to build an effective model and provide valuable advice.

**#modeling**: I have used the HC to correctly identify and build a model to capture the realistic characteristics of room egress. I accurately determined the relevant aspects of a model in a given context and provides a clear justification for all my parameter choices.

**#organization, #professionalism:** I have organized a well written and visualized communication using a sophisticated organizational structure to deliver a message in a highly effective way. I have followed the professor's advice in providing an organized report and python notebook.

**Python Code is attached below:**

# CS166_FP (1)

April 24, 2020

## 1 Room Egress FP - Ahmed Abdelrahman

### Section-1 Two-dimensional Cellular Automaton Model

***Python Implementation***

[1]:
```
#Importing Libraries

import random
import time
import numpy as np
import pandas as pd
from copy import deepcopy
from random import shuffle
import scipy.stats as sts
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline
```

The model below follows an algorithm that can be described as:

1. Place people and exits randomly in a two-dimensional grid, where exits are always on the edges of the room.
2. Check and store the location of people and exits in the grid.
3. Iterate through people's locations in the current state and update them synchronously.
4. If a person's location is the same as an exit location, then delete the person' locations from the grid in the next update (as they will leave the room).
5. If people are on the room sides, update their locations to move vertically to their nearest exit to them. If they are in the middle of the room, then move them horizontally to their nearest exits. When people see fixed objects (Walls) in the room, they turn around them and try to evacuate the room as fast as possible.
6. There is a stochastic parameter to capture the panicking behavior of people. If a person is panicked, they will not move for a one-time stamp. The following code illustrates the implementation of these rules.

[2]:
```
class crowd_egress:

    def __init__(self, room_w=10, room_h=10,people_num=25, exit_num=6,
                 exit_random=True ,exit_size=1, wall_size=0, wall_direction='v'
                 , panic_ratio=0.05):


        '''
        Create a new crowd egress object. People are distributed randomly
        in the room; exits can be randomly distributed or distributed with
        a pre-determined rule

        Inputs:

            room_w,room_h (int): The width and the hight of the room.
            Default: 10

            people_num (int): The number of people in the room. Default:10

            exit_num (int): The number of exits in the room. Default: 6

            exit_random (boolean): The parameter controlling if doors will
             be randomly distributed in the room or not. Default: True

            exit_size (int): The size of each exit. Default: 1

            wall_size, wall_direction: The size of the wall and
            the direction of it

            panic_ratio: a stochastic parameter to capture the panicking
            behavior of people. If a person is panicked, they will not move.
        '''

        self.room_w=room_w
        self.room_h=room_h
        self.people_num=people_num
        self.exit_num=exit_num
        self.exit_size=exit_size
        self.exit_random=exit_random
        self.wall_size=wall_size
        self.wall_direction=wall_direction
        self.panic_ratio=panic_ratio
        # A list with all exits' positions in the room
        self.exit_coordinates=[]
        self.r_exit_coordinates=[] #for right edge exits
        self.l_exit_coordinates=[] #for left edge exits
```

```
    #Initializing an empty room with the given size
    self.current_state = scipy.zeros([self.room_h,self.room_w])
    #storingthe time steps and total number of people that left the room
    self.time = 0
    self.people_out=0
    #storing the location of people in the room
    self.wall_loc=[]



def initialize(self):

    """Adding people, exits and walls to the room """


    #Adding a random wall to the room
    #with a specified

    if self.wall_size!=0:
        if self.wall_direction=='v':
          #specifying random coloumn and row for the wall location
            col=random.randint(1,self.room_w-2)
            row=random.randint(1,self.room_h-self.wall_size-1)

    #the visualization of the room is reflected
    #where the [0] index of the object is the row number of it (y-axis)
            for i in range(self.wall_size):
                self.wall_loc.append([row+i,col])

        else:
            col=random.randint(1,self.room_w-self.wall_size-1)
            row=random.randint(1,self.room_h-2)

            for i in range(self.wall_size):
                self.wall_loc.append([row,col+i])

        for loc in self.wall_loc:
            self.current_state[loc[0],loc[1]]=2



    #Adding people to the room by specifying
    #unique and random column/rows indexes

    counter=0
```

```
        while counter < self.people_num:
            c_indx=random.randint(0,self.room_w-1)
            r_indx=random.randint(0,self.room_h-1)

            #checking that no position has more than 1 person
            if (self.current_state[r_indx,c_indx] !=1 and
                self.current_state[r_indx,c_indx] !=2):
                self.current_state[r_indx,c_indx]=1
                counter+= 1



        #if exit_random is true, we add random exits
        #on the right or the left sides of the room
        if self.exit_random:

            for _ in range(self.exit_num):

                placed= False
                #looping to ensure placing the right number
                #of unique exits in the room
                while placed==False:

                    edge = random.choice([0,self.room_w-1])
                    exit_proposed=[]
                    exit_start= random.randint(0,self.room_h-1)
                    edge_end= [*range(self.room_h-self.exit_size+1,
                                      self.room_h)]

                    #Adding proposed exits to a list and ensuring
                    #that exits are placed with the right size

                    if exit_start in edge_end:
                        #Increasing the exit size based on the user input
                        for i in range((self.exit_size)):
                            exit_proposed.append([exit_start-i,edge])
                    else:
                        for i in range((self.exit_size)):
                            exit_proposed.append([exit_start+i,edge])

                    #checking if the proposed exits are already
                    #located in the room. If so, iterate and propose new ones
                    check= True
                    for _ in exit_proposed:
                        if _ in self.exit_coordinates:
                            check=False
```

```
                if check:
                    for i in exit_proposed:
                        self.exit_coordinates.append(i)
                    placed=True

        #iterating through all exits and then
        #putting them in the right lists
        for exit in self.exit_coordinates:
            if exit[1]== 0:
                self.l_exit_coordinates.append(exit)
            if exit[1]== self.room_h-1:
                self.r_exit_coordinates.append(exit)


    #Placing exits with pre-determined rules
    #by optimizing the distances between exits

    else:
        left_exit_num=self.exit_num//2
        right_exit_num=self.exit_num-left_exit_num

        for i in range(left_exit_num):
            for s in range(self.exit_size):
                #placing exits based on this rule below
                self.exit_coordinates.append(
                    [(i*((self.room_h//left_exit_num))+s),0])
                self.l_exit_coordinates.append(
                    [(i*((self.room_h//left_exit_num))+s),0])

        for i in range(right_exit_num):
            for s in range(self.exit_size):
                self.exit_coordinates.append(
                    [(i*((self.room_h//right_exit_num))+s),self.room_w-1])
                self.r_exit_coordinates.append(
                    [(i*((self.room_h//right_exit_num))+s),self.room_w-1])




def draw(self):
    """Plotting the current state of the simulation"""

    plt.cla()
    plt.pcolor(self.current_state, cmap="Blues")
    plt.axis('image')
    plt.title('t = ' + str(self.time))
```

```
def update(self):

    """
    Updating the simulation synchronously by reading
    from the current state and updating the next one.
    The function egresses people in front of the exits
    and move people to the nearst exit to them

    """

    #Initializing an empty next_state
    self.next_state = scipy.zeros([self.room_h,self.room_w])
    #Adding walls to the room if any
    if self.wall_size != 0:
        for loc in self.wall_loc:
                self.next_state[loc[0],loc[1]]=2

    #Locating each person in the room, so we can update their
    #positions accordingly
    people= np.where(np.array(self.current_state)==1)
    self.people_loc=[]
    for i in range(len(people[0])):
        self.people_loc.append([people[0][i],people[1][i]])


    #Egress people in front of exits

    people_out=[]
    for person in self.people_loc:
        if person in self.exit_coordinates:
            self.current_state[person[0],person[1]]=0
            people_out.append([person[0],person[1]])
            # A metric to calculate how many people have left the room
            # the metric is used to validate the simulation success
            self.people_out+=1

    #Removing people who left the room  from people's locations list
    for i in people_out:
        self.people_loc.remove(i)
```

```
        #iterating through people list
        #to update their location based on the simulation rules

        for person in self.people_loc:

            #Updating all people locations at the same time
            #through two functions. if the person is on the edges
            #upadte_edge will be activated, otherwide update
            #center will do the work

            self.update_edges(person)
            self.update_center(person)

        #Updating the current_state synchronously
        self.current_state, self.next_state = self.next_state,self.current_state
        self.time +=1




    def update_edges(self, person):

    #if the person is on the left vertical edge
    #calculate the shortest path to the exit and
    #follow it (vertical movement)

        if person[1]==0:

            if len(self.l_exit_coordinates)>=1:
                v_differences=[]
                move_direction=[]

                for exit in self.l_exit_coordinates:
                    v_differences.append(abs(exit[0]-person[0]))
                    if (exit[0]-person[0])>0:
                        move_direction.append(1)
                    else:
                        move_direction.append(-1)

                #shortest move
                move=move_direction[v_differences.index(min(v_differences))]

                #if we have two moves with the same
                #distance, we pick randomly between them
                if (v_differences.count(min(v_differences))>1
                    and (random.random()<=0.5)):
```

```
                move=-(move)

            #update the next_state
            if self.current_state[person[0]+move,person[1]]==1:
                self.next_state[person[0],person[1]]=1
            else:
                self.next_state[person[0]+move,person[1]]=1
                self.current_state[person[0]+move,person[1]]=1
                #self.current_state[person[0],person[1]]=0
        else:

            #update the next_state
            move=1
            if self.current_state[person[0],person[1]+move]==1:
                self.next_state[person[0],person[1]]=1
            else:
                self.next_state[person[0],person[1]+move]=1
                self.current_state[person[0],person[1]+move]=1

#if the person is on the right vertical edge
#calculate the shortest path to the exit and
#follow it(vertical movement)

    if person[1]==self.room_w-1:

        if len(self.r_exit_coordinates)>=1:

            v_differences=[]
            move_direction=[]

            for exit in self.r_exit_coordinates:
                v_differences.append(abs(exit[0]-person[0]))
                if (exit[0]-person[0])>0:
                    move_direction.append(1)
                else:
                    move_direction.append(-1)

            move=move_direction[v_differences.index(min(v_differences))]

            #if we have two moves with the same
            #distance, we pick randomly between them
            if (v_differences.count(min(v_differences))>1
                and (random.random()<=0.5)):
                move=-move

            #update the next_state
            if self.current_state[person[0]+move,person[1]]==1:
```

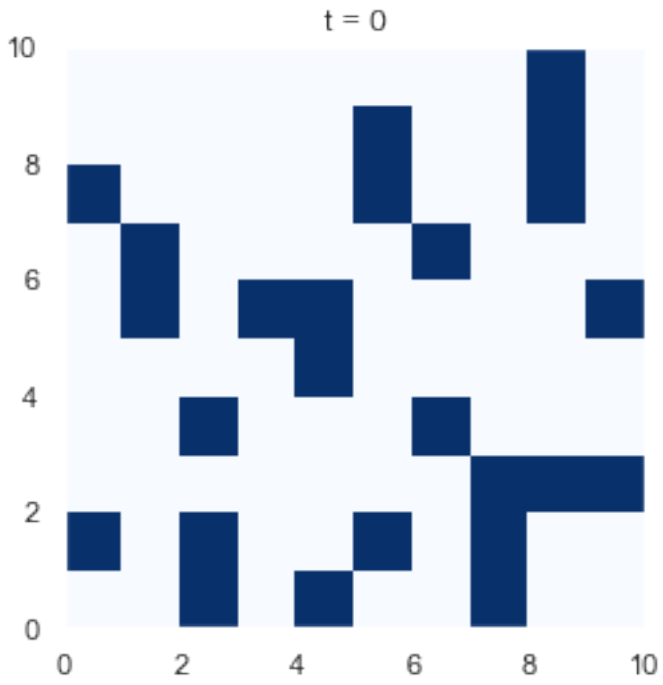
```
                self.next_state[person[0],person[1]]=1
            else:
                self.next_state[person[0]+move,person[1]]=1
                self.current_state[person[0]+move,person[1]]=1
                #self.current_state[person[0],person[1]]=0
        else:

            #update the next_state
            move=-1
            if self.current_state[person[0],person[1]+move]==1:
                self.next_state[person[0],person[1]]=1
            else:
                self.next_state[person[0],person[1]+move]=1
                self.current_state[person[0],person[1]+move]=1




def update_center(self,person):


#if the person is not along the edges
#update their location to move to the nearst edge
#to them. (Horizontal Movements)
    if (person[1] != self.room_w-1) and (person[1] != 0):

        if self.panic_ratio >= random.random():
            #if a person is panicking, they will not move
            #and update their position
            self.next_state[person[0],person[1]]=1

        else:

            if (len(self.l_exit_coordinates)>=1
                and len(self.r_exit_coordinates)>=1):


                #checking the direction of the movement
                move=0

                if person[1]> (self.room_w-1)/2:
                    move=1
                else:
                    move=-1
```

```
                    #moving people

                    if self.current_state[person[0],person[1]+move]==2:
                        #checking which vertical move to take (up or down)

                        if person[0]> (self.room_h-1)/2:
                            v_move=1
                        else:
                            v_move=-1

                        if self.current_state[person[0]+move,person[1]]==1:
                            self.next_state[person[0],person[1]]=1
                        else:
                            self.next_state[person[0]+move,person[1]]=1
                            self.current_state[person[0]+move,person[1]]=1

                    elif self.current_state[person[0],person[1]+move]==1:
                        self.next_state[person[0],person[1]]=1
                    else:
                        self.next_state[person[0],person[1]+move]=1
                        self.current_state[person[0],person[1]+move]=1

                else:

                  #Calculating the horizontal distance between people and exits

                    h_differences=[]
                    move_direction=[]

                    for exit in self.exit_coordinates:
                        h_differences.append(abs(exit[1]-person[1]))
                        if (exit[1]-person[1])>0:
                            move_direction.append(1)
                        else:
                            move_direction.append(-1)

                    #shortest move
                    move=move_direction[h_differences.index(min(h_differences))]

                    #if we have two moves with the same
                    #distance, we pick randomly between them
                    if (h_differences.count(min(h_differences))>1 and
                        (random.random()<=0.5)):
                        move=-(move)

                    #update the next_state
                    #moving people
```

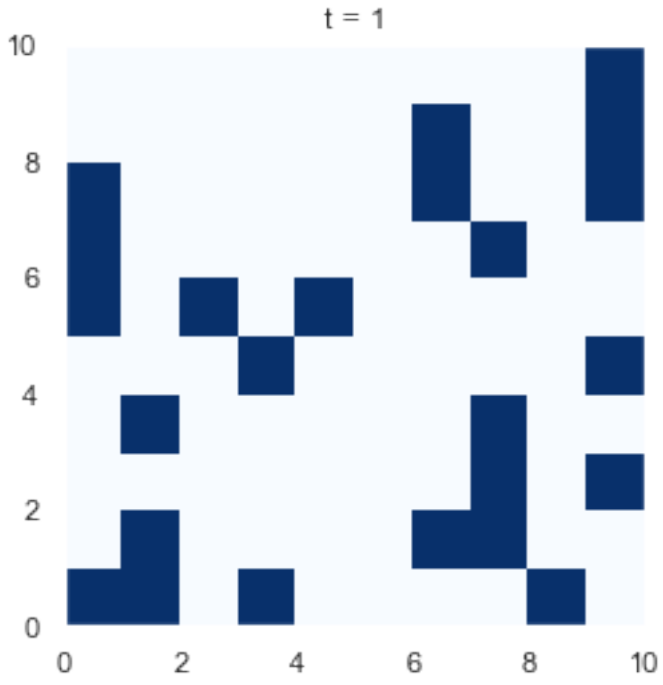
```
                    if self.current_state[person[0],person[1]+move]==1:
                        self.next_state[person[0],person[1]]=1
                    else:
                        self.next_state[person[0],person[1]+move]=1
                        self.current_state[person[0],person[1]+move]=1
```

[3]:
```
#Visualize and testing the state of the model over time
#With the default parameters

sim=crowd_egress(people_num=25,exit_num=6, room_h=10)
sim.initialize()

print("People Out:",sim.people_out)
print("People remaining in the Room:",
      np.count_nonzero(sim.current_state)-sim.wall_size)
sim.draw()
plt.show()

#A while loop to stop when the room is empty
while (np.count_nonzero(sim.current_state)-sim.wall_size) != 0:
    sim.update()

    #Two metrics to help debuging and
    #validating the simulation success
    print("People out",sim.people_out)
    print("People remaining in the Room",len(sim.people_loc))
    sim.draw()
    plt.show()
```

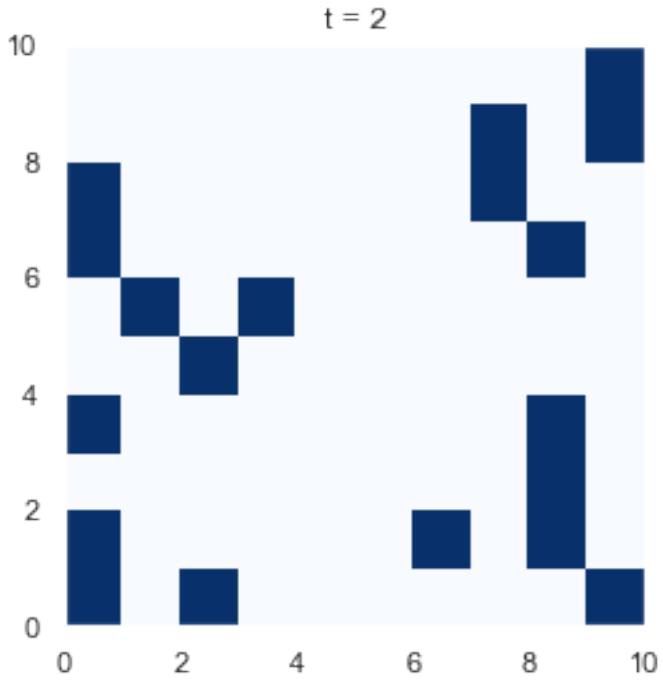
```
People Out: 0
People remaining in the Room: 25
```



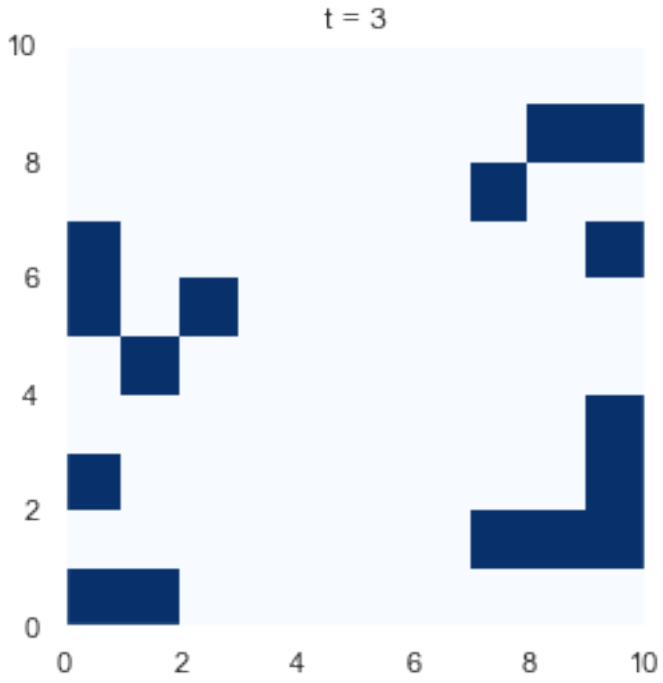

```
People out 1
People remaining in the Room 24
```



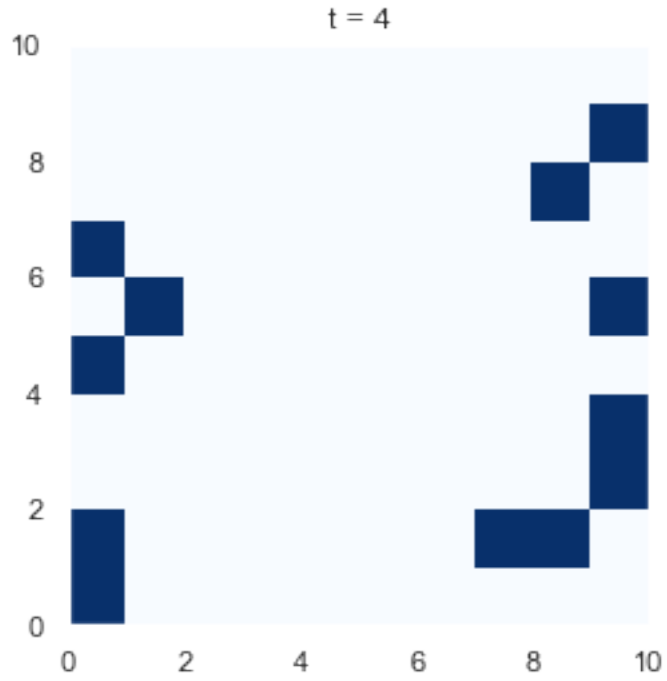
```
People out 6
People remaining in the Room 19
```



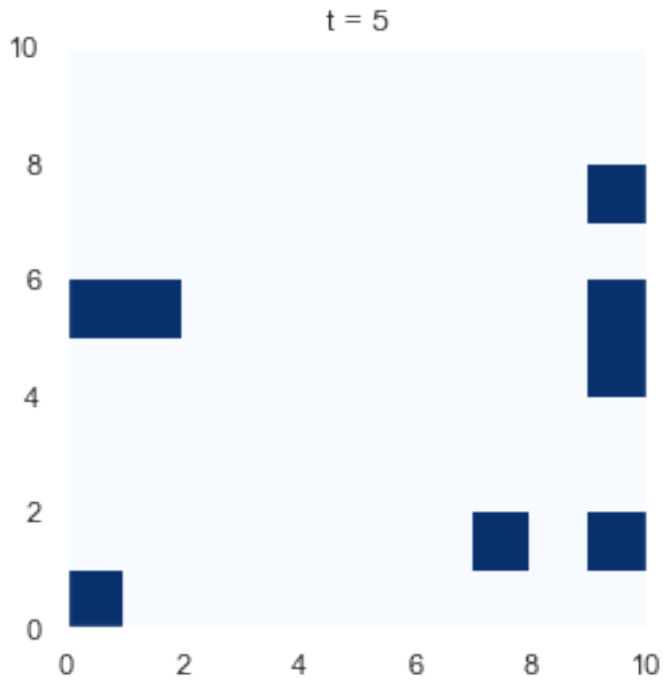

```
People out 9
People remaining in the Room 16
```



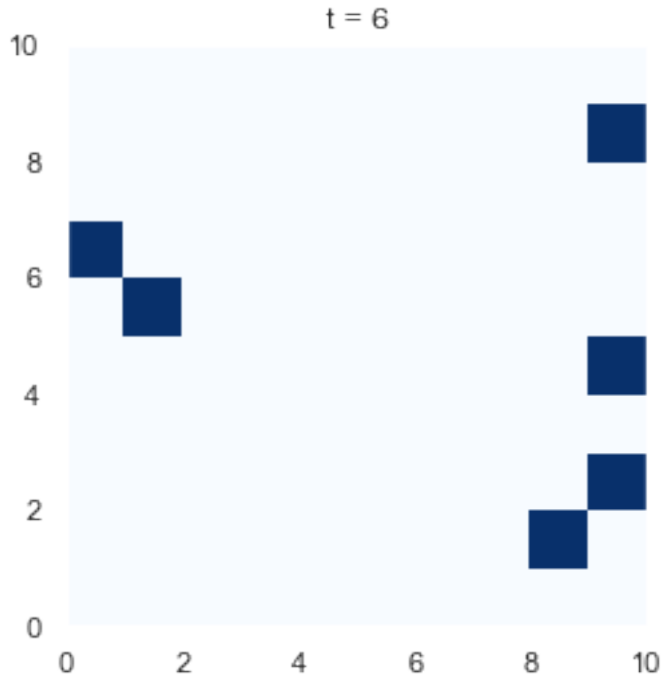
```
People out 13
People remaining in the Room 12
```



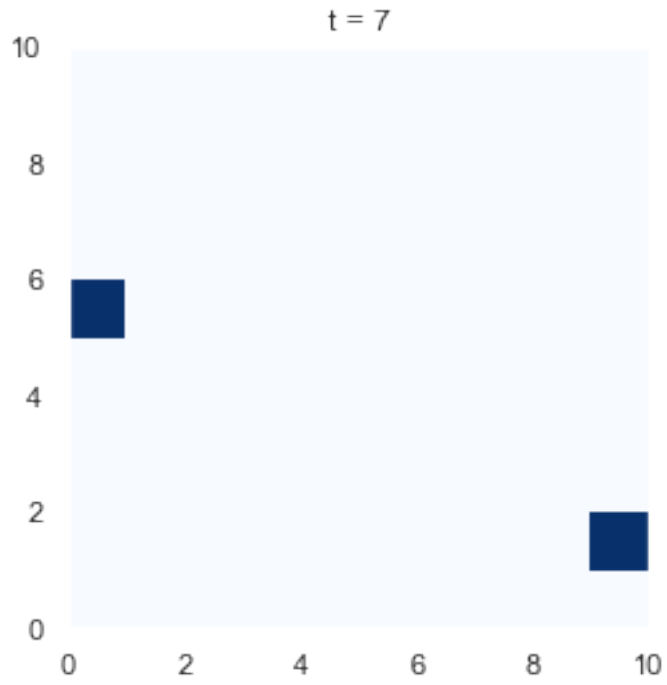

```
People out 17
People remaining in the Room 8
```



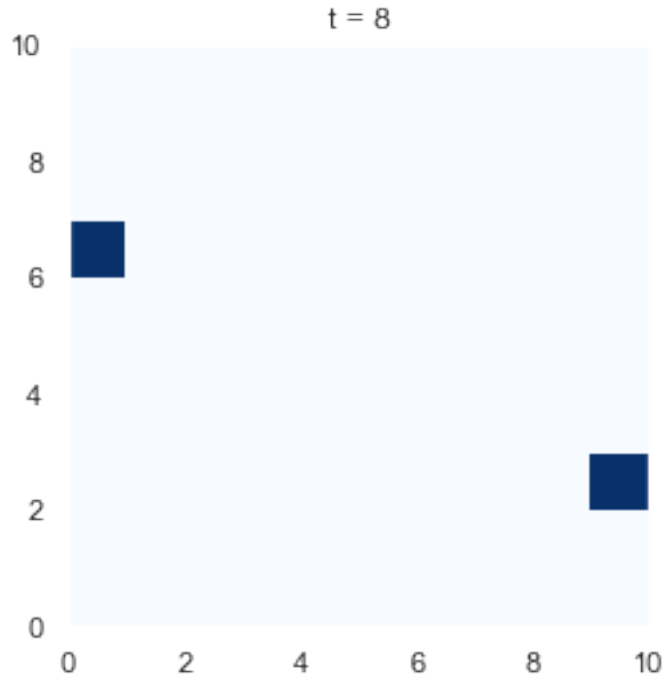

```
People out 19
People remaining in the Room 6
```



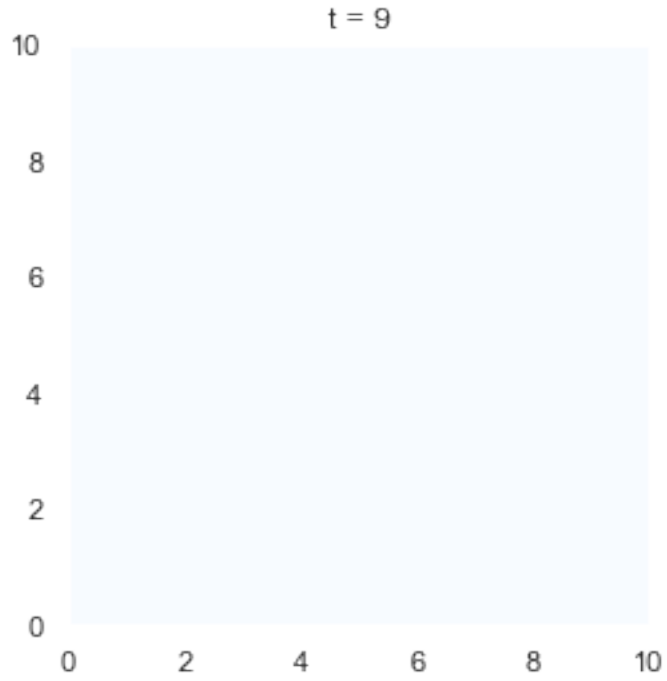
```
People out 23
People remaining in the Room 2
```




```
People out 23
People remaining in the Room 2
```



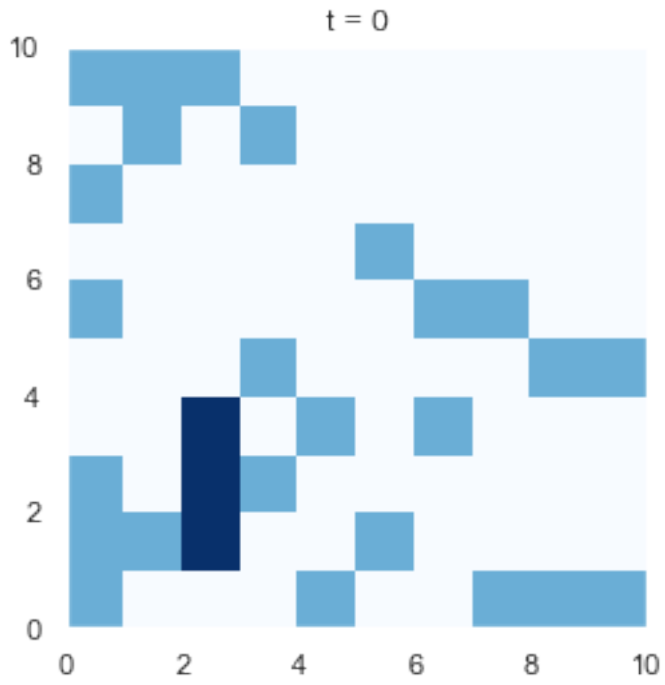
```
People out 25
People remaining in the Room 0
```



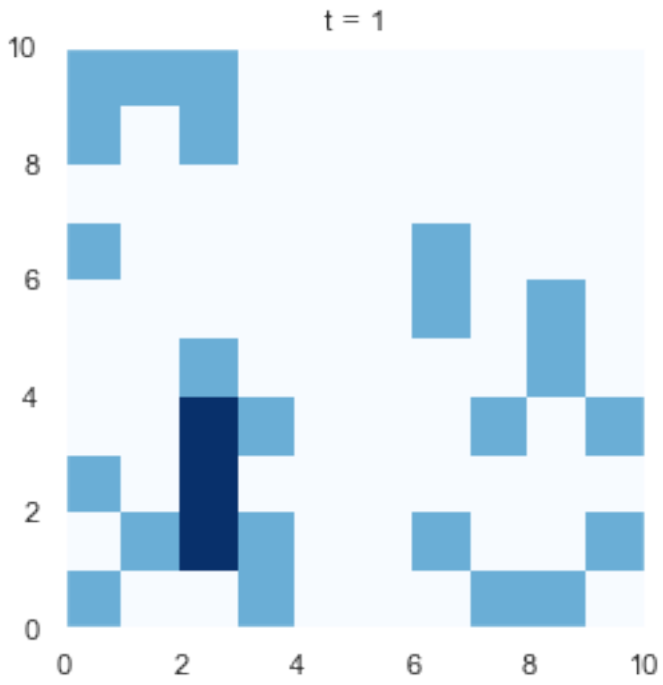

[4]:
```
##Visualize and testing the state of the model over time
#By adding a vertical edge with with a size of 3

sim=crowd_egress(people_num=25,wall_size=3, wall_direction='v')
sim.initialize()

print("People Out",sim.people_out)
print("People In the Room",np.count_nonzero(sim.current_state)-sim.wall_size)
sim.draw()
plt.show()

#A while loop to stop when the room is empty
while (np.count_nonzero(sim.current_state)-sim.wall_size) != 0:
    sim.update()

    #Two metrics to help debuging and
    #validating the simulation success
    print("People out",sim.people_out)
    print("People remaining",len(sim.people_loc))
```

```
    sim.draw()
    plt.show()
```

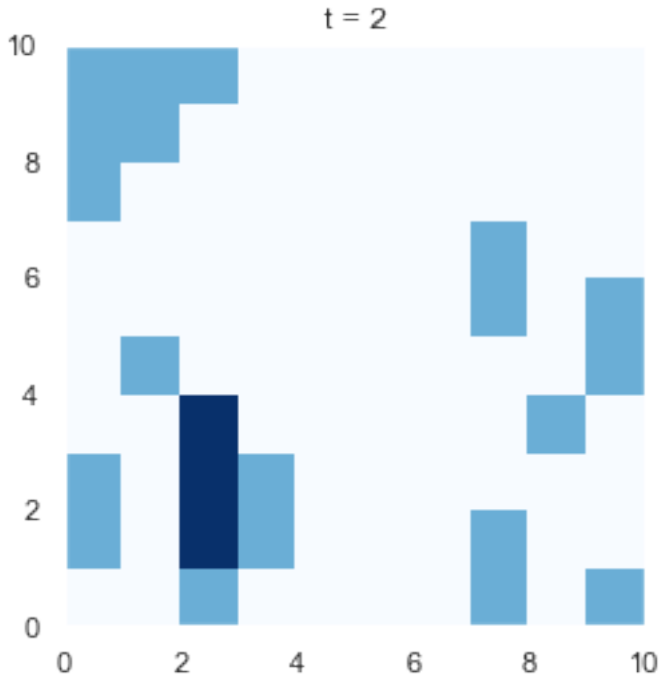
```
People Out 0
People In the Room 25
```



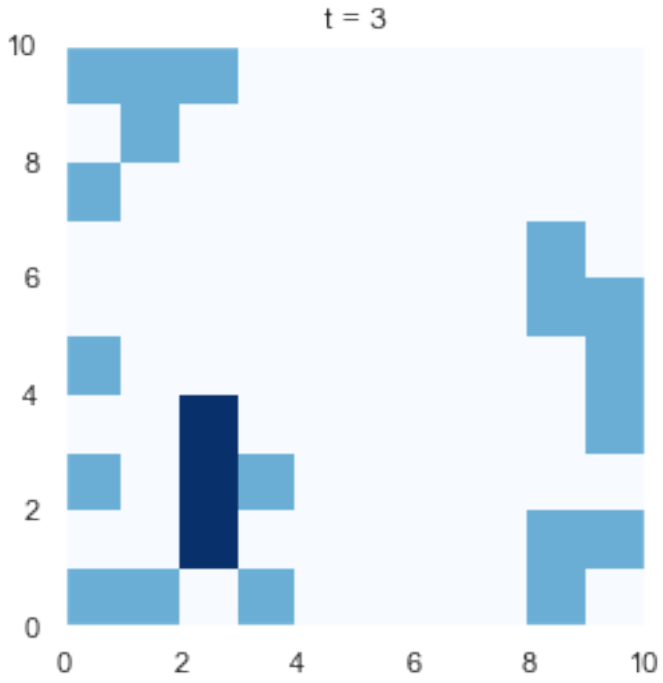

```
People out 2
People remaining 23
```



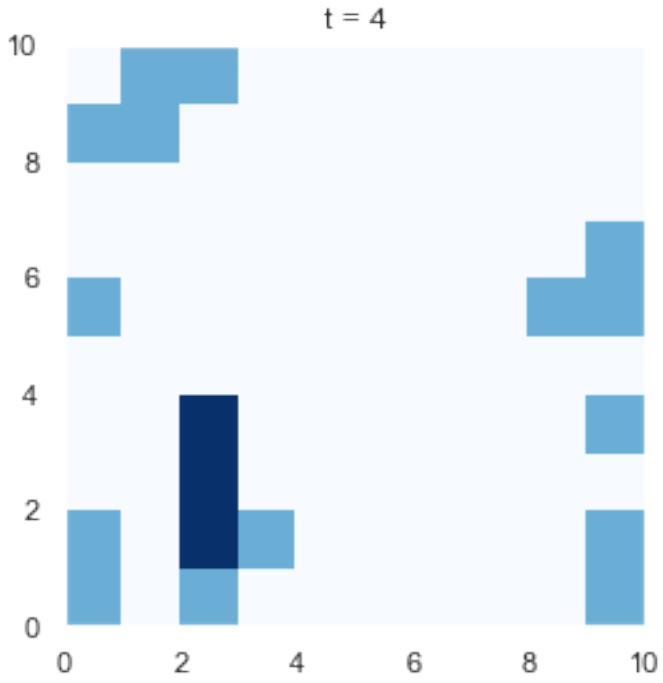

```
People out 5
People remaining 20
```



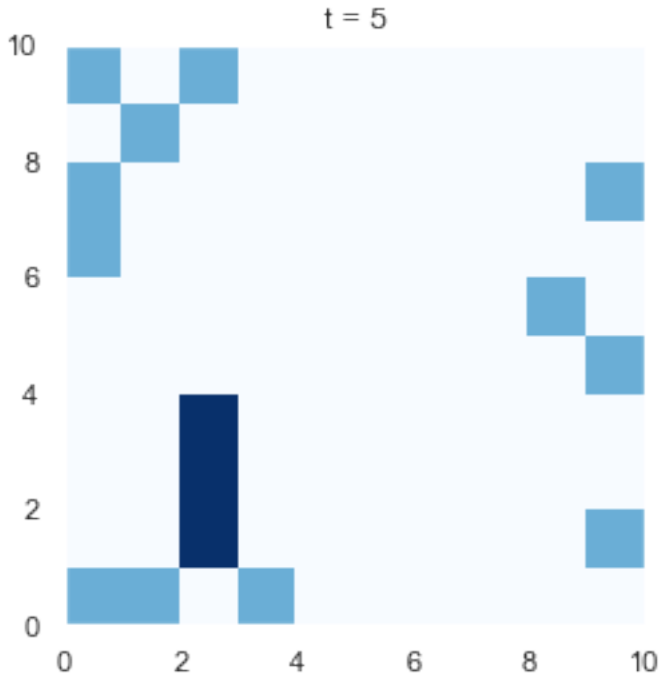
```
People out 6
People remaining 19
```



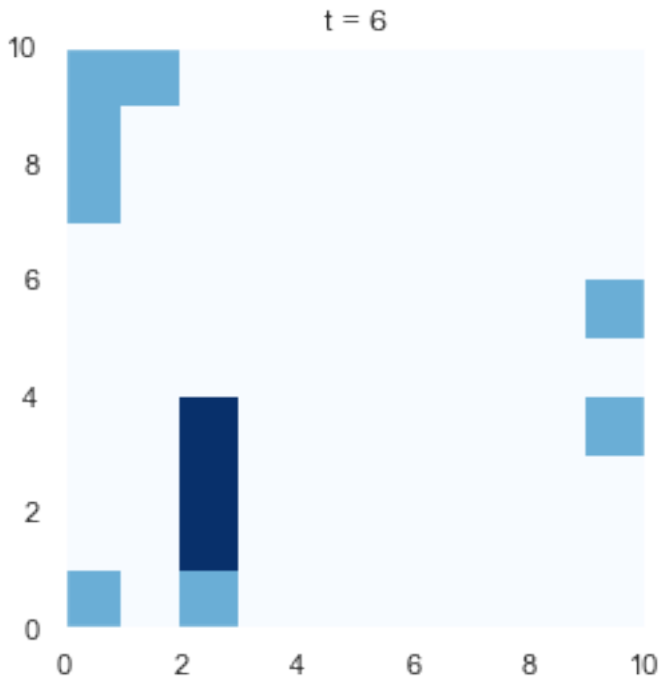

```
People out 10
People remaining 15
```



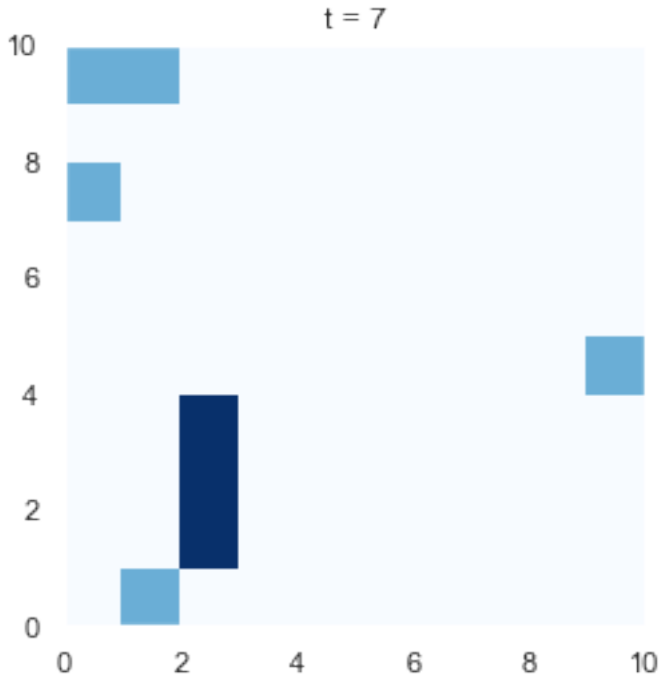
```
People out 13
People remaining 12
```



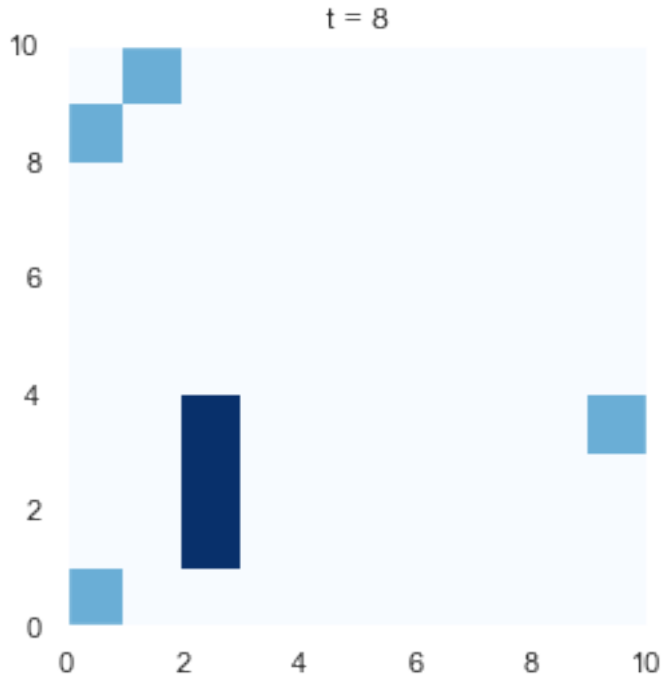

```
People out 17
People remaining 8
```



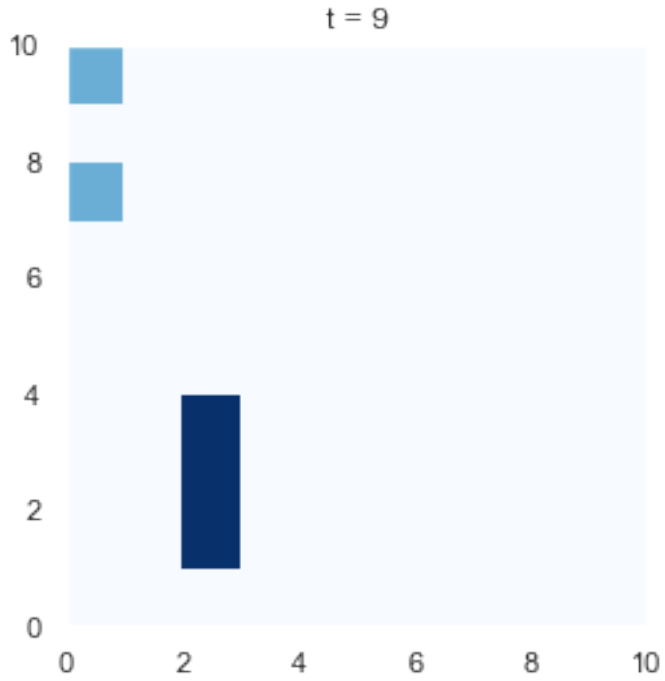

```
People out 20
People remaining 5
```



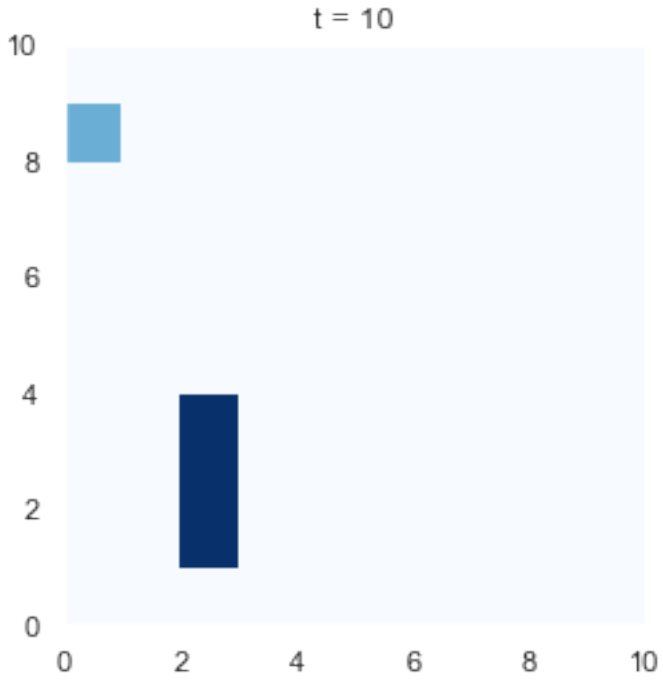
```
People out 21
People remaining 4
```



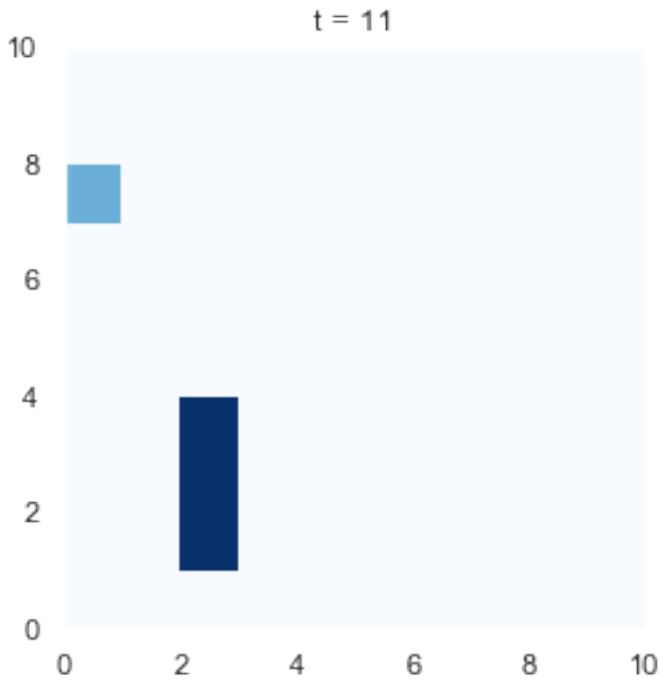

```
People out 23
People remaining 2
```



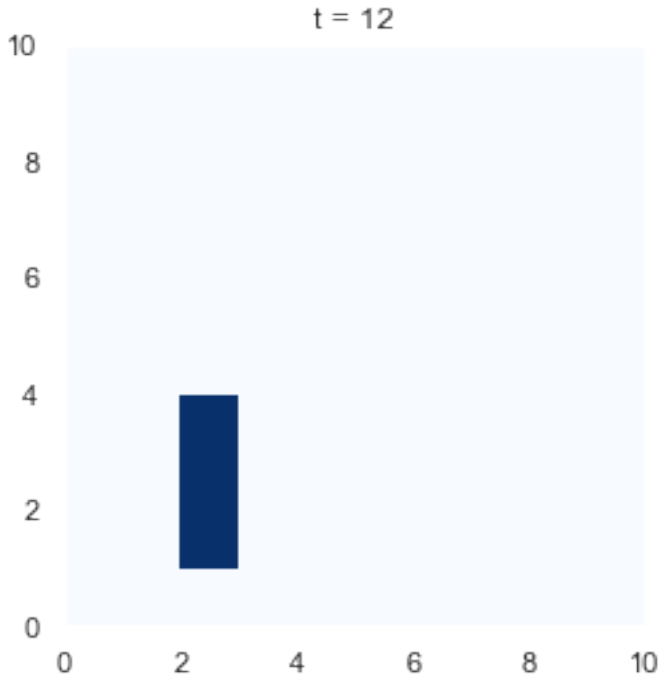
```
People out 24
People remaining 1
```




```
People out 24
People remaining 1
```




```
People out 25
People remaining 0
```



[5]:
```
#A function resposible for making animation
#of the simulation

def make_animation(draw_function, frames, interval, filename=None,
                   verbose=False, figure_params=None):
    """
    Write an animated GIF to file using matplotlib and ImageMagick.

    Inputs:

        draw_function (callable) The function to call, once per frame,
          to render to the current matplotlib figure. This function's call
          signature must be

            draw_function(current_frame_number, total_frame_count)

          The first frame is 0 and the last frame is total_frame_count-1.

        frames (int) The number of frames to generate.

        interval (float) The number of milliseconds to wait between frames
          in the animation. Set this to 40 for a fairly standard frame rate
          of 25fps (1 second / 40 (milliseconds per frame) = 25 frames per
          second).

        filename (str) The path to the file to write to. If none is provided,
          the animation will be written to a random filename, prefixed with
          "animate_" in the current directory. Default: None.

        verbose (bool) Whether to print the current frame number to the
          console as the animation is being created. This will clear other
          console output. Default: False.

        figure_params (dict) The keyword arguments to pass to matplotlib
          when creating a new figure for this animation. Use it to set the
          figure size and other figure properties. Default: None.

    Returns: A display object that will inject HTML for displaying the
      animated GIF into the notebook.
    """
    from matplotlib import pyplot, animation
    from IPython.display import HTML, display, clear_output
    import random

    if filename is None:
        filename = 'animate_%06i.gif' % random.randint(0, 999999)
    # Create figure
```

```
    if figure_params is None:
        figure_params = {}
    figure = pyplot.figure(**figure_params)
    # Wrap draw_function if we need to print to console
    if verbose:
        old_draw_function = draw_function
        def draw_function(current_frame_number, total_frame_count):
            old_draw_function(current_frame_number, total_frame_count)
            print('Processed frame', current_frame_number + 1, '/',
                  total_frame_count)
            clear_output(wait=True)
            if current_frame_number + 1 == total_frame_count:
                print('Writing animation to file...')
                clear_output(wait=True)
    # Generate animation
    anim = animation.FuncAnimation(
        figure, draw_function, frames=frames, interval=interval,
        init_func=lambda: None, fargs=(frames,))
    anim.save(filename, writer='imagemagick')
    # Close the animation figure so the last frame does not get displayed
    # in the notebook.
    pyplot.close()
    # Return display object for the animated GIF
    return display(HTML('<img src="' + filename + '">'))
```

[6]:
```
#An animation of the simulation
#for better visualization and results
sim=crowd_egress()
sim.initialize()

def animate_simulation(frame, total_frames):
    sim.update()
    sim.draw()


while (np.count_nonzero(sim.current_state)-sim.wall_size) != 0:
    make_animation(
        animate_simulation, frames=1, interval=1,
        figure_params={'figsize': (5, 4)},
        verbose=True)
    time.sleep(0.7)
```

```
<IPython.core.display.HTML object>
```

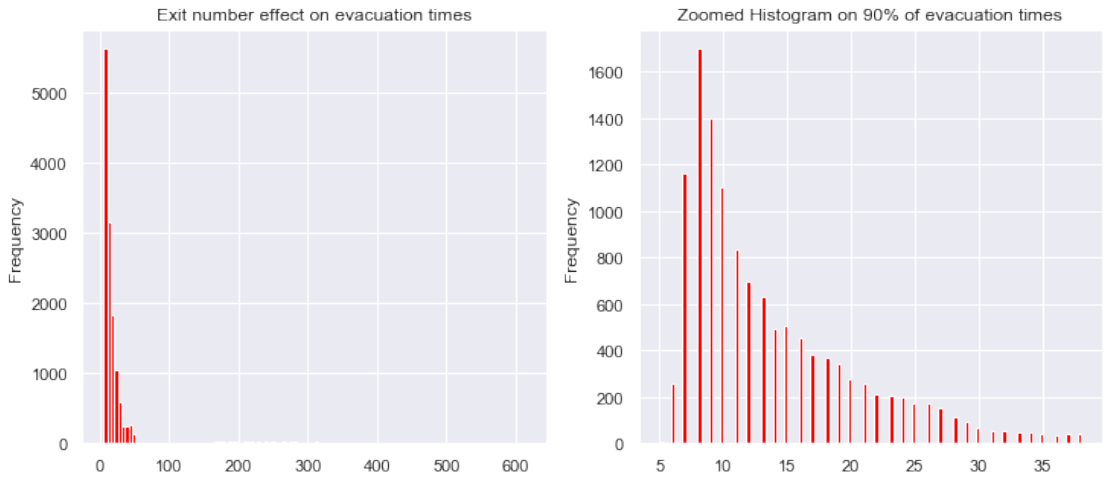
***Part-2 Parameters Experiments***

[7]:
```
"""
Experiment-1

Exit Number vs Evacuation Time

A simulation experiment of different number of
exits on Evacuation time
The number of exits are varied from 1 to 14
with the fixed default parameters

Each step is counted as one interval in this scenario.

"""

exits_num= [*range(1,15)]
trails=1000

E1_steps = []
E1_median_time=[]
E1_mode_time=[]
E1_mean_time=[]

for num in exits_num:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(exit_num=num)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E1_steps.append([num,sim.time])

    E1_median_time.append(np.median(counts))
    E1_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E1_mode_time.append(max(set(counts), key=counts.count))

#E1_input = [_[0] for _ in E1_steps]
E1_time = [_[1] for _ in E1_steps]
```

[8]:
```
#Histogrm of the expriment times with
#their frequencies


print ("95% Confidence Interval:", sts.t.interval(0.95, len(E1_time)-1,loc=np.
 ↪mean(E1_time), scale=sts.sem(E1_time)))
```

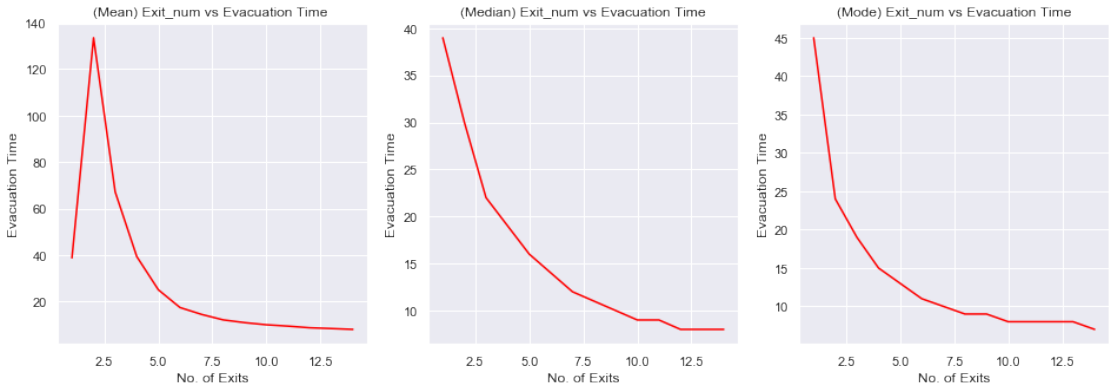
```
print ("05th-95th percentiles of the data:", (np.quantile(E1_time, 0.05),np.
 ↪quantile(E1_time, 0.95)))
print(sts.describe(E1_time))


#Focusing on 90% of the data
E1_zoomed_times=[]
for i in E1_time:
    if (i<=np.quantile(E1_time, 0.90)):
        E1_zoomed_times.append(i)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)

plt.hist(E1_time,bins = int(np.sqrt(len(E1_time))), color = "red")
plt.title("Exit number effect on evacuation times")
plt.ylabel("Frequency")


plt.subplot(1, 2, 2)

plt.hist(E1_zoomed_times,bins = int(np.sqrt(len(E1_zoomed_times))), color =␣
 ↪"red")
plt.title("Zoomed Histogram on 90% of evacuation times")
plt.ylabel("Frequency")

plt.show()
```

```
95% Confidence Interval: (27.833986250083605, 29.831728035630682)
05th-95th percentiles of the data: (7.0, 167.0)
DescribeResult(nobs=14000, minmax=(5, 613), mean=28.832857142857144,
variance=3635.5947486044915, skewness=4.533343223274561,
kurtosis=22.228262368498626)
```




[9]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(exits_num,E1_mean_time, color = "red")
plt.title("(Mean) Exit_num vs Evacuation Time ")
plt.xlabel("No. of Exits")
plt.ylabel("Evacuation Time")



plt.subplot(1, 3, 2)

plt.plot(exits_num,E1_median_time, color = "red")
plt.title("(Median) Exit_num vs Evacuation Time")
plt.xlabel("No. of Exits")
plt.ylabel("Evacuation Time")

plt.subplot(1, 3, 3)

plt.plot(exits_num,E1_mode_time, color = "red")
plt.title("(Mode) Exit_num vs Evacuation Time")
plt.xlabel("No. of Exits")
plt.ylabel("Evacuation Time")

plt.show
```

[9]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



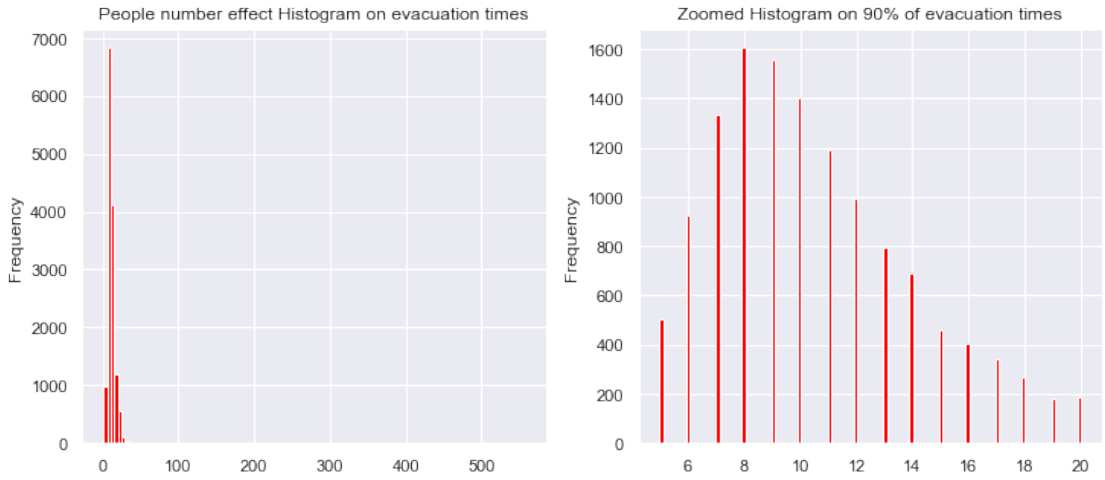

[10]:
```
"""
Experiment-2

Number of People vs Evacuation Time

A simulation experiment of different numbers of people on
Evacuation time
The number of people are varied from 1 to 28
with the fixed default parameters


Each step is counted as one interval in this scenario.

"""

people_num= [*range(1,29)]
trails=500
E2_steps = []
E2_median_time=[]
E2_mode_time=[]
E2_mean_time=[]

for num in people_num:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(people_num=num)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E2_steps.append([num,sim.time])

    E2_median_time.append(np.median(counts))
```

```
    E2_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E2_mode_time.append(max(set(counts), key=counts.count))

#E2_input = [_[0] for _ in E2_steps]
E2_time = [_[1] for _ in E2_steps]
```

[11]:
```
#Histogrm of the expriment times with
#their frequencies


print ("95% Confidence Interval:", sts.t.interval(0.95, len(E2_time)-1, loc=np.
 ↪mean(E2_time), scale=sts.sem(E2_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E2_time, 0.05),np.
 ↪quantile(E2_time, 0.95)))
print(sts.describe(E2_time))


#Zooming on the 90% of the data
E2_zoomed_times=[]
for i in E2_time:
    if (i>=np.quantile(E2_time, 0.05)) and (i<=np.quantile(E2_time, 0.95)):
        E2_zoomed_times.append(i)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)

plt.hist(E2_time,bins = int(np.sqrt(len(E2_time))), color = "red")
plt.title("People number effect Histogram on evacuation times")
plt.ylabel("Frequency")


plt.subplot(1, 2, 2)

plt.hist(E2_zoomed_times,bins = int(np.sqrt(len(E2_zoomed_times))), color =␣
 ↪"red")
plt.title("Zoomed Histogram on 90% of evacuation times")
plt.ylabel("Frequency")

plt.show()
```

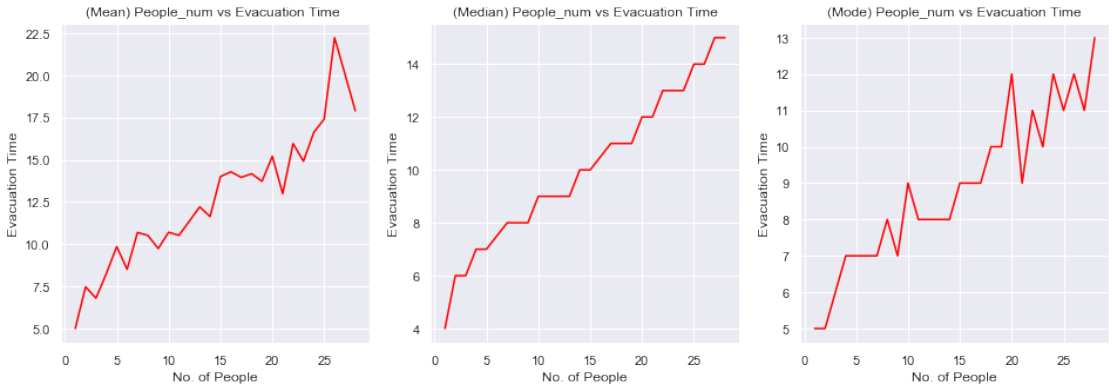
```
95% Confidence Interval: (12.359987096571222, 13.117727189143062)
05th-95th percentiles of the data: (5.0, 20.0)
DescribeResult(nobs=14000, minmax=(1, 557), mean=12.738857142857142,
variance=523.0428788995133, skewness=12.553436133449555,
```

kurtosis=185.65060304415334)




[12]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(people_num,E2_mean_time, color = "red")
plt.title("(Mean) People_num vs Evacuation Time ")
plt.xlabel("No. of People")
plt.ylabel("Evacuation Time")


plt.subplot(1, 3, 2)

plt.plot(people_num,E2_median_time, color = "red")
plt.title("(Median) People_num vs Evacuation Time")
plt.xlabel("No. of People")
plt.ylabel("Evacuation Time")

plt.subplot(1, 3, 3)

plt.plot(people_num,E2_mode_time, color = "red")
plt.title("(Mode) People_num vs Evacuation Time")
plt.xlabel("No. of People")
plt.ylabel("Evacuation Time")

plt.show
```

[12]: <function matplotlib.pyplot.show(*args, **kw)>

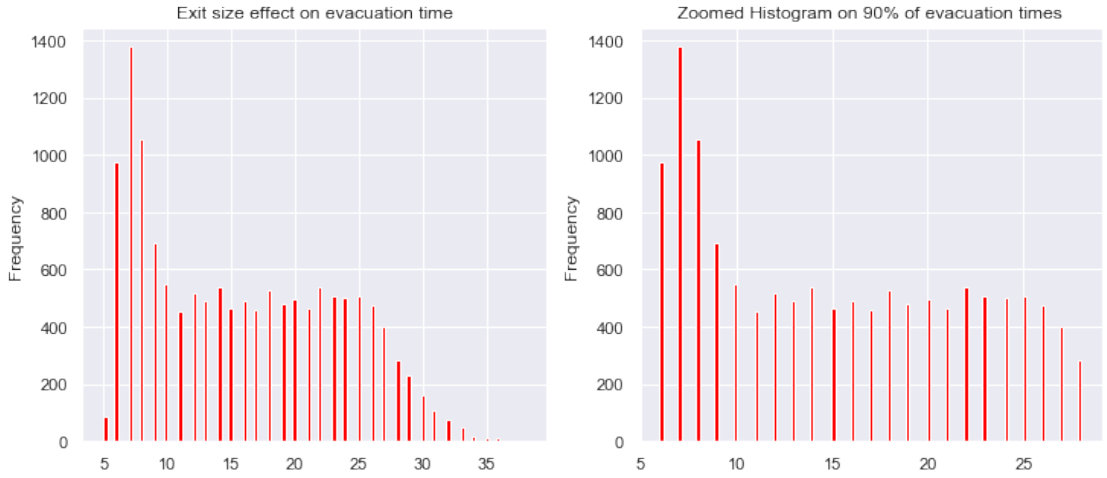



[13]:
```
"""
Experiment-3

Exit Size vs Evacuation Time

A simulation experiment of different Exit sizes
on Evacuation time
The Exit sizes are varied from 1 to 10
and of two Exits and the rest of
the fixed default parameters


"""

exit_size= [*range(1,11)]
trails=1400
E3_steps = []
E3_median_time=[]
E3_mode_time=[]
E3_mean_time=[]

for num in exit_size:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(exit_num=2, exit_size=num, exit_random=False)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E3_steps.append([num,sim.time])

    E3_median_time.append(np.median(counts))
    E3_mean_time.append(np.mean(counts))
```

```
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E3_mode_time.append(max(set(counts), key=counts.count))

#E3_input = [_[0] for _ in E3_steps]
E3_time = [_[1] for _ in E3_steps]
```

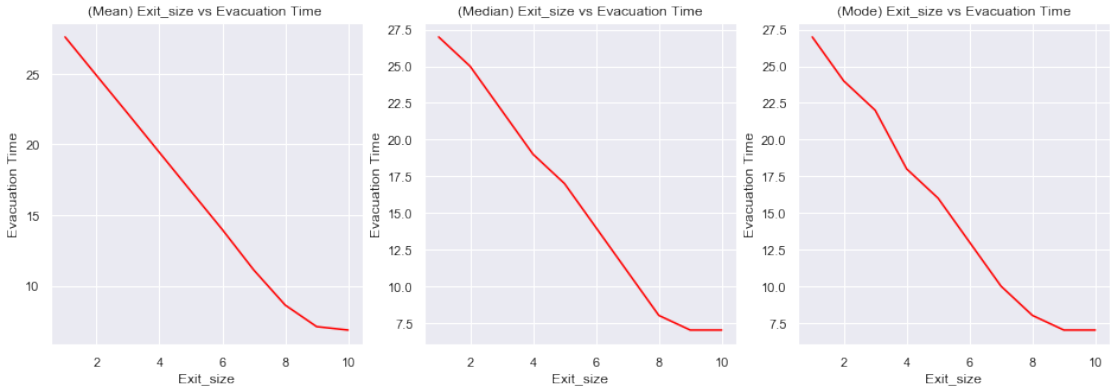
[14]:
```
#Histogrm of the expriment times with
#their frequencies


print ("95% Confidence Interval:", sts.t.interval(0.95, len(E3_time)-1, loc=np.
 ↪mean(E3_time), scale=sts.sem(E3_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E3_time, 0.05),np.
 ↪quantile(E3_time, 0.95)))
print(sts.describe(E3_time))


#finding the 90% percentile of the data
E3_zoomed_times=[]
for i in E3_time:
    if (i>=np.quantile(E3_time, 0.05)) and (i<=np.quantile(E3_time, 0.95)):
        E3_zoomed_times.append(i)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)

plt.hist(E3_time,bins = int(np.sqrt(len(E3_time))), color = "red")
plt.title("Exit size effect on evacuation time")
plt.ylabel("Frequency")


plt.subplot(1, 2, 2)

plt.hist(E3_zoomed_times,bins = int(np.sqrt(len(E3_zoomed_times))), color =␣
 ↪"red")
plt.title("Zoomed Histogram on 90% of evacuation times")
plt.ylabel("Frequency")

plt.show()
```

```
95% Confidence Interval: (15.722552351488108, 15.971733362797607)
05th-95th percentiles of the data: (6.0, 28.0)
DescribeResult(nobs=14000, minmax=(5, 38), mean=15.847142857142858,
variance=56.56224628289775, skewness=0.32627367306894833,
kurtosis=-1.0912165843992174)
```




[15]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(exit_size,E3_mean_time, color = "red")
plt.title("(Mean) Exit_size vs Evacuation Time ")
plt.xlabel("Exit_size")
plt.ylabel("Evacuation Time")


plt.subplot(1, 3, 2)

plt.plot(exit_size,E3_median_time, color = "red")
plt.title("(Median) Exit_size vs Evacuation Time")
plt.xlabel("Exit_size")
plt.ylabel("Evacuation Time")

plt.subplot(1, 3, 3)

plt.plot(exit_size,E3_mode_time, color = "red")
plt.title("(Mode) Exit_size vs Evacuation Time")
plt.xlabel("Exit_size")
plt.ylabel("Evacuation Time")

plt.show
```

[15]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

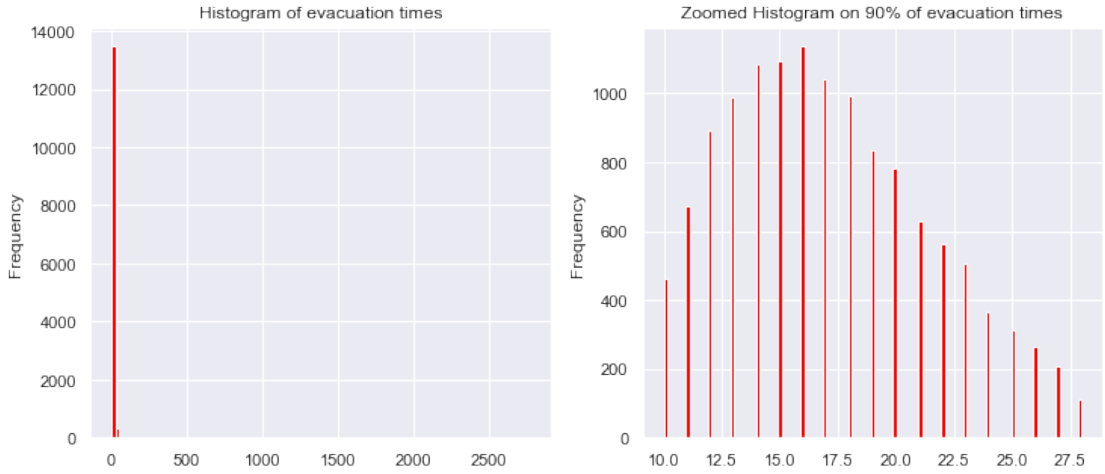



[16]:
```
"""
Experiment-4

Room Size vs Evacuation Time

A simulation experiment of different Room
dimensions on Evacuation time
The Room h and w are varied from 8 to 22
with the rest of fixed default parameters

"""
room_size= [*range(8,22)]
trails=1000

E4_steps = []
E4_median_time=[]
E4_mode_time=[]
E4_mean_time=[]

for num in room_size:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(room_h=num, room_w=num)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E4_steps.append([num,sim.time])

    E4_median_time.append(np.median(counts))
    E4_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
```

```
    E4_mode_time.append(max(set(counts), key=counts.count))

#E4_input = [_[0] for _ in E4_steps]
E4_time = [_[1] for _ in E4_steps]
```

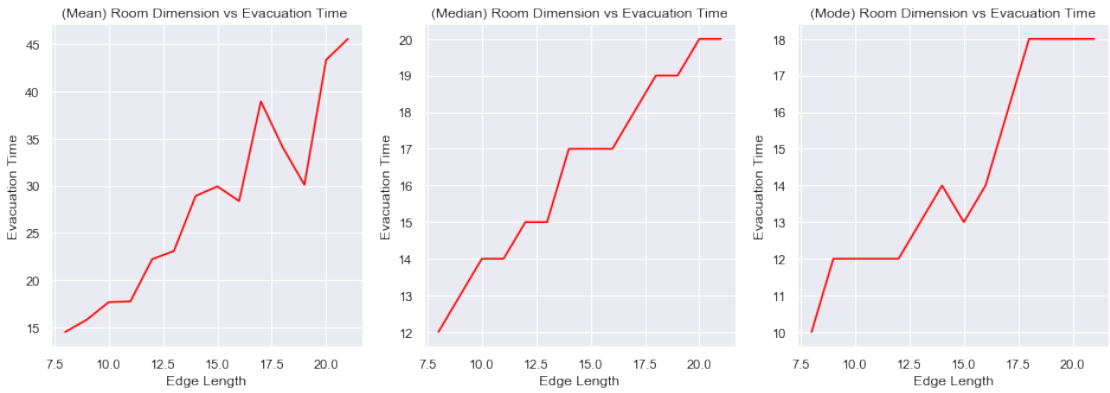
[17]:
```
#Histogrm of the expriment times with
#their frequencies


print ("95% Confidence Interval:", sts.t.interval(0.95, len(E4_time)-1, loc=np.
 ↪mean(E4_time), scale=sts.sem(E4_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E4_time, 0.05),np.
 ↪quantile(E4_time, 0.95)))
print(sts.describe(E4_time))
print("Data mode:",max(set(E4_time), key=E4_time.count))


#finding the 90% percentile of the data
E4_zoomed_times=[]
for i in E4_time:
    if (i>=np.quantile(E4_time, 0.05)) and (i<=np.quantile(E4_time, 0.95)):
        E4_zoomed_times.append(i)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)

plt.hist(E4_time,bins = int(np.sqrt(len(E4_time))), color = "red")
plt.title("Histogram of evacuation times")
plt.ylabel("Frequency")


plt.subplot(1, 2, 2)

plt.hist(E4_zoomed_times,bins = int(np.sqrt(len(E4_zoomed_times))), color =␣
 ↪"red")
plt.title("Zoomed Histogram on 90% of evacuation times")
plt.ylabel("Frequency")

plt.show()
```

```
95% Confidence Interval: (26.237506012189062, 29.52120827352522)
05th-95th percentiles of the data: (10.0, 28.0)
DescribeResult(nobs=14000, minmax=(6, 2773), mean=27.879357142857142,
variance=9822.551270391761, skewness=11.976010573397875,
kurtosis=176.1314612707289)
Data mode: 16
```




[18]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(room_size,E4_mean_time, color = "red")
plt.title("(Mean) Room Dimension vs Evacuation Time ")
plt.xlabel("Edge Length")
plt.ylabel("Evacuation Time")


plt.subplot(1, 3, 2)

plt.plot(room_size,E4_median_time, color = "red")
plt.title("(Median) Room Dimension vs Evacuation Time")
plt.xlabel("Edge Length")
plt.ylabel("Evacuation Time")

plt.subplot(1, 3, 3)

plt.plot(room_size,E4_mode_time, color = "red")
plt.title("(Mode) Room Dimension vs Evacuation Time")
plt.xlabel("Edge Length")
plt.ylabel("Evacuation Time")

plt.show
```

[18]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```




### Section-2 Comparing Exit Placement strategies

#### *Part-1 Experiments*

[19]:
```
"""
Experiment-5

strategic non-random exit placement
Exit Number vs Evacuation Time

A simulation experiment of different number of exits on
Evacuation time
The number of exits are varied from 1 to 11
with the fixed default parameters


Each step is counted as one interval in this scenario.

"""

exits_num= [*range(1,11)]
trails=1000

E5_steps = []
E5_median_time=[]
E5_mode_time=[]
E5_mean_time=[]

for num in exits_num:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(exit_num=num, exit_random=False)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
```

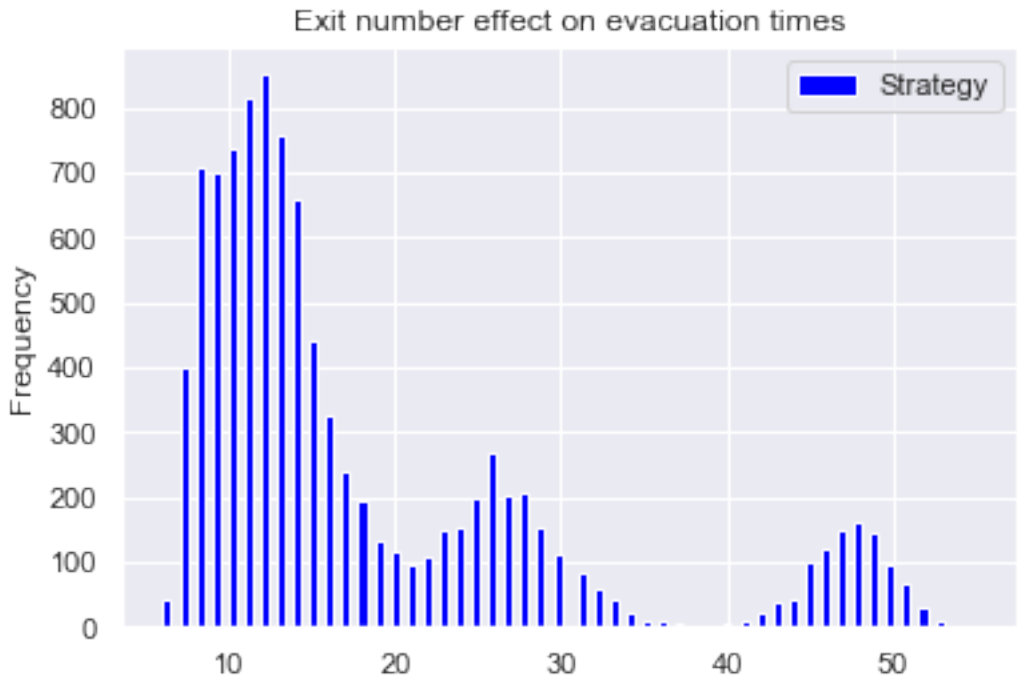
```
            sim.update()
        counts.append(sim.time)
        E5_steps.append([num,sim.time])

    E5_median_time.append(np.median(counts))
    E5_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E5_mode_time.append(max(set(counts), key=counts.count))

#E5_input = [_[0] for _ in E5_steps]
E5_time = [_[1] for _ in E5_steps]


#-------------------------------------------------
#randomly placing exits
E6_steps = []
E6_median_time=[]
E6_mode_time=[]
E6_mean_time=[]

for num in exits_num:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(exit_num=num, exit_random=True)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E6_steps.append([num,sim.time])

    E6_median_time.append(np.median(counts))
    E6_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E6_mode_time.append(max(set(counts), key=counts.count))

#E5_input = [_[0] for _ in E5_steps]
E6_time = [_[1] for _ in E6_steps]
```

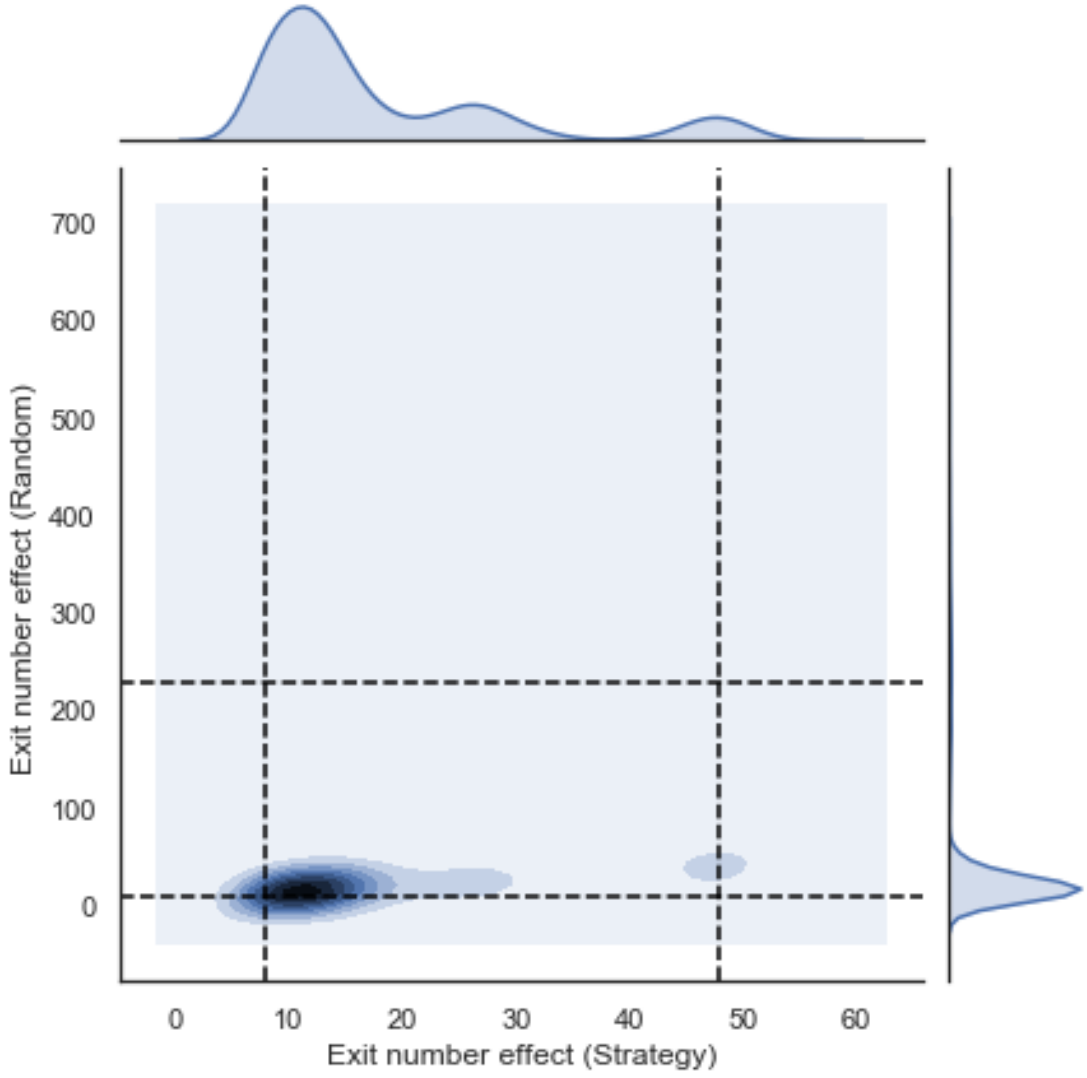
[20]:
```
#Histogrm of the expriment times with
#their frequencies

print ("95% CI of Strategic Placement:", sts.t.interval(0.95, len(E5_time)-1,
                                                        loc=np.
 ↪mean(E5_time), scale=sts.sem(E5_time)))
```

```
print ("05th-95th percentiles of the data:", (np.quantile(E5_time, 0.05),np.
 ↪quantile(E5_time, 0.95)))
print(sts.describe(E5_time))
print("--------------------------------")
print ("95% CI of Random Placement:", sts.t.interval(0.95, len(E6_time)-1,␣
 ↪loc=np.mean(E6_time),
                                                    scale=sts.sem(E6_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E6_time, 0.05),np.
 ↪quantile(E6_time, 0.95)))
print(sts.describe(E6_time))



# #finding the 90% percentile of the data

# E5_zoomed_times=[]
# for i in E5_time:
#     if (i<=np.quantile(E5_time, 0.90)):
#         E5_zoomed_times.append(i)


# E6_zoomed_times=[]
# for i in E6_time:
#     if (i<=np.quantile(E6_time, 0.90)):
#         E6_zoomed_times.append(i)



plt.hist(E5_time,bins = int(np.sqrt(len(E5_time))), color =␣
 ↪"blue",label='Strategy')
#plt.hist(E6_time,bins = int(np.sqrt(len(E6_time))), color =␣
 ↪"red",label='Random',alpha=0.65)
plt.title("Exit number effect on evacuation times")
plt.ylabel("Frequency")
plt.legend()



with sns.axes_style("white"):
    #sns.jointplot(x=E5_time, y=E6_time, kind="kde", ax=axes[0, 1]);
    fig = sns.jointplot(x=E5_time, y=E6_time, kind='kde', n_levels=10)
    fig.ax_joint.axvline(x=np.quantile(E5_time, 0.05), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axvline(x=np.quantile(E5_time, 0.95), linestyle='--', color =␣
 ↪'black')
```

```
    fig.ax_joint.axhline(y=np.quantile(E6_time, 0.05), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axhline(y=np.quantile(E6_time, 0.95), linestyle='--', color =␣
 ↪'black')
    fig.set_axis_labels('Exit number effect (Strategy)', 'Exit number effect␣
 ↪(Random)')

plt.show()
```

```
95% CI of Strategic Placement: (18.007421798192176, 18.465578201807823)
05th-95th percentiles of the data: (8.0, 48.0)
DescribeResult(nobs=10000, minmax=(6, 55), mean=18.2365,
variance=136.57342509250927, skewness=1.5011515927023742,
kurtosis=1.3085798049454391)
--------------------------------
95% CI of Random Placement: (36.141534301111484, 38.92546569888851)
05th-95th percentiles of the data: (8.0, 227.0)
DescribeResult(nobs=10000, minmax=(5, 671), mean=37.5335,
variance=5042.614139163915, skewness=3.666121473564117,
kurtosis=13.90975444607616)
```

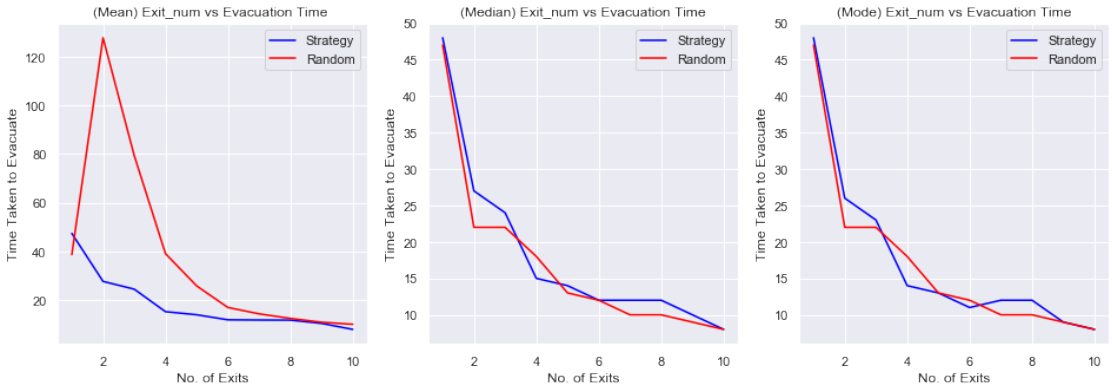





[21]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(exits_num,E5_mean_time, color = "blue",label='Strategy')
plt.plot(exits_num,E6_mean_time, color = "red",label='Random')

plt.title("(Mean) Exit_num vs Evacuation Time ")
plt.xlabel("No. of Exits")
plt.ylabel("Time Taken to Evacuate")
plt.legend()
```

```
plt.subplot(1, 3, 2)

plt.plot(exits_num,E5_median_time, color = "blue",label='Strategy')
plt.plot(exits_num,E6_mode_time, color = "red",label='Random')
plt.title("(Median) Exit_num vs Evacuation Time")
plt.xlabel("No. of Exits")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

plt.subplot(1, 3, 3)

plt.plot(exits_num,E5_mode_time, color = "blue",label='Strategy')
plt.plot(exits_num,E6_mode_time, color = "red",label='Random')
plt.title("(Mode) Exit_num vs Evacuation Time")
plt.xlabel("No. of Exits")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

#plt.show
```

[21]: <matplotlib.legend.Legend at 0x1b3ecc50>




[22]:
```
"""
Experiment-6

Non-random Exit placing
People Number vs Evacuation Time

A simulation experiment of different numbers of people on
Evacuation time
The number of people are varied from 1 to 28
with the fixed default parameters
```

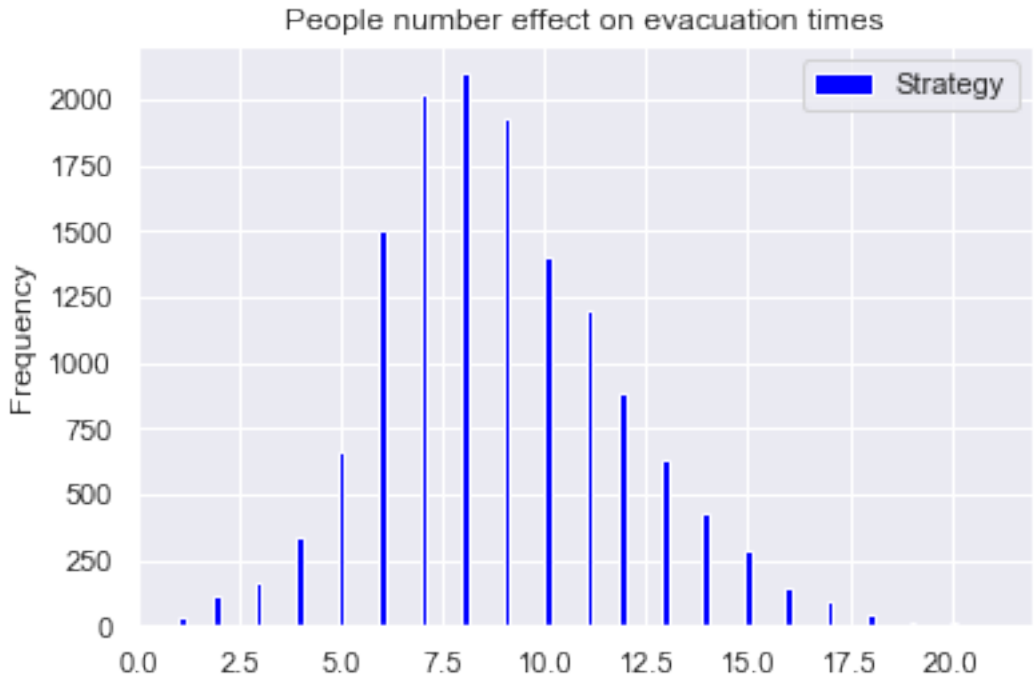
```
Each step is counted as one interval in this scenario.

"""

people_num= [*range(1,29)]
trails=500
E7_steps = []
E7_median_time=[]
E7_mode_time=[]
E7_mean_time=[]

for num in people_num:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(people_num=num, exit_random=False)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E7_steps.append([num,sim.time])

    E7_median_time.append(np.median(counts))
    E7_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E7_mode_time.append(max(set(counts), key=counts.count))

E7_input = [_[0] for _ in E7_steps]
E7_time = [_[1] for _ in E7_steps]
```

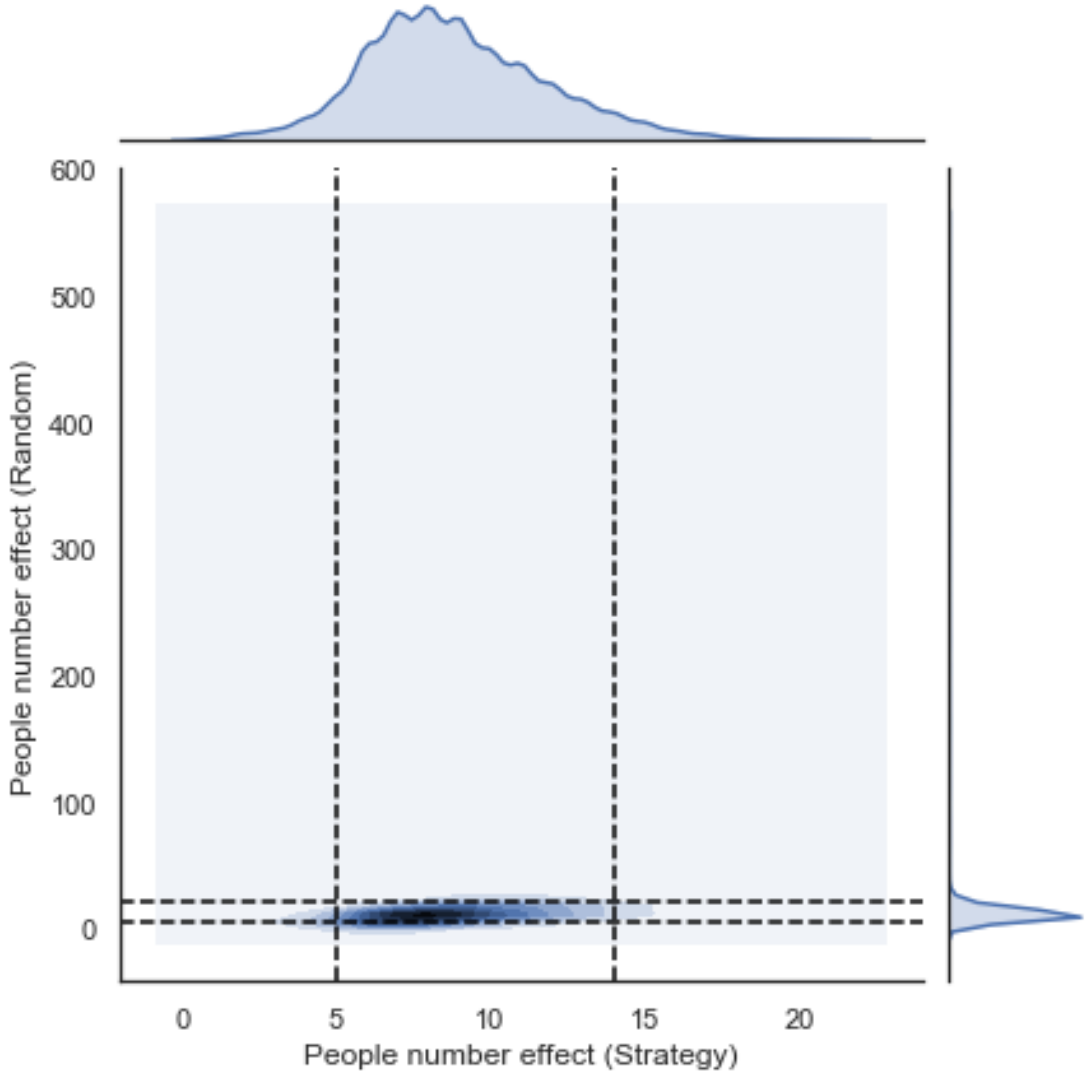
[23]:
```
#Histogrm of the expriment times with
#their frequencies


print ("95% CI of Strategic Placement:", sts.t.interval(0.95, len(E7_time)-1,
                                                        loc=np.
 ↪mean(E7_time), scale=sts.sem(E7_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E7_time, 0.05),np.
 ↪quantile(E7_time, 0.95)))
print(sts.describe(E7_time))
print("---------------------------------")
print ("95% CI of Random Placement:", sts.t.interval(0.95, len(E2_time)-1,␣
 ↪loc=np.mean(E2_time),
                                                    scale=sts.sem(E2_time)))
```

```
print ("05th-95th percentiles of the data:", (np.quantile(E2_time, 0.05),np.
 ↪quantile(E2_time, 0.95)))
print(sts.describe(E2_time))



# #finding the 90% percentile of the data

# E7_zoomed_times=[]
# for i in E7_time:
#     if (i>=np.quantile(E7_time, 0.05)) and (i<=np.quantile(E7_time, 0.95)):
#         E7_zoomed_times.append(i)



plt.hist(E7_time,bins = int(np.sqrt(len(E7_time))), color =␣
 ↪"blue",label='Strategy')
#plt.hist(E2_time,bins = int(np.sqrt(len(E2_time))), color =␣
 ↪"red",label='Random',alpha=0.5)
plt.title("People number effect on evacuation times")
plt.ylabel("Frequency")
plt.legend()


with sns.axes_style("white"):
    #sns.jointplot(x=E5_time, y=E6_time, kind="kde", ax=axes[0, 1]);
    fig = sns.jointplot(x=E7_time, y=E2_time, kind='kde', n_levels=10)
    fig.ax_joint.axvline(x=np.quantile(E7_time, 0.05), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axvline(x=np.quantile(E7_time, 0.95), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axhline(y=np.quantile(E2_time, 0.05), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axhline(y=np.quantile(E2_time, 0.95), linestyle='--', color =␣
 ↪'black')
    fig.set_axis_labels('People number effect (Strategy)', 'People number effect␣
 ↪(Random)')

plt.show()
```

```
95% CI of Strategic Placement: (8.83410998450558, 8.931604301208704)
05th-95th percentiles of the data: (5.0, 14.0)
DescribeResult(nobs=14000, minmax=(1, 21), mean=8.882857142857143,
variance=8.658753176247282, skewness=0.4665694029462451,
kurtosis=0.3509679728087911)
--------------------------------
95% CI of Random Placement: (12.359987096571222, 13.117727189143062)
```

```
05th-95th percentiles of the data: (5.0, 20.0)
DescribeResult(nobs=14000, minmax=(1, 557), mean=12.738857142857142,
variance=523.0428788995133, skewness=12.553436133449555,
kurtosis=185.65060304415334)
```

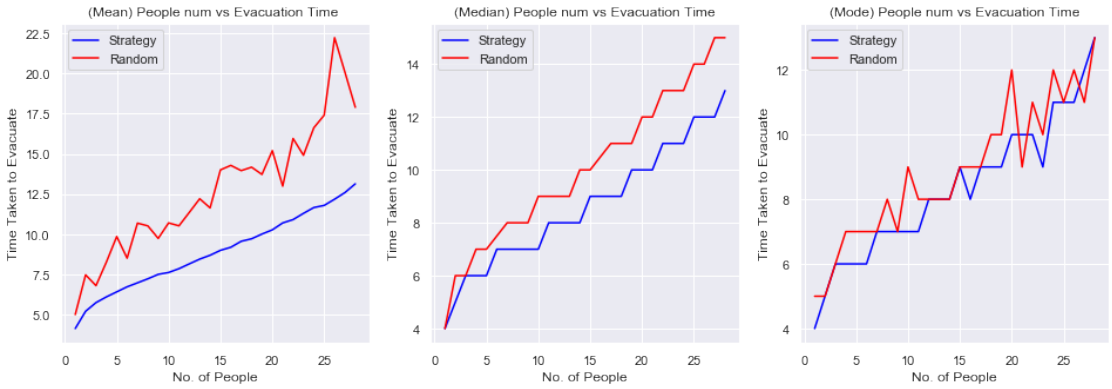





[24]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(people_num,E7_mean_time, color = "blue",label='Strategy')
plt.plot(people_num,E2_mean_time, color = "red",label='Random')

plt.title("(Mean) People num vs Evacuation Time ")
plt.xlabel("No. of People")
plt.ylabel("Time Taken to Evacuate")
plt.legend()
```

```
plt.subplot(1, 3, 2)

plt.plot(people_num,E7_median_time, color = "blue",label='Strategy')
plt.plot(people_num,E2_median_time, color = "red",label='Random')
plt.title("(Median) People num vs Evacuation Time")
plt.xlabel("No. of People")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

plt.subplot(1, 3, 3)

plt.plot(people_num,E7_mode_time, color = "blue",label='Strategy')
plt.plot(people_num,E2_mode_time, color = "red",label='Random')
plt.title("(Mode) People num vs Evacuation Time")
plt.xlabel("No. of People")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

#plt.show
```

[24]: <matplotlib.legend.Legend at 0x1b45f970>



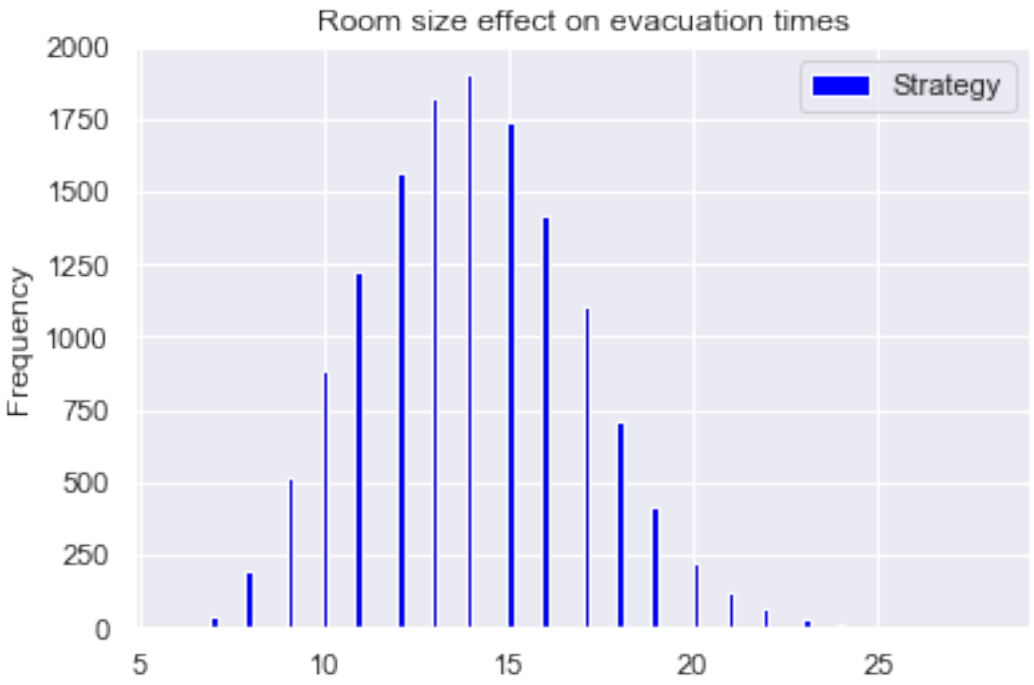

[25]:
```
"""
Experiment-7

Room Size vs Evacuation Time

A simulation experiment of different Room
dimensions on Evacuation time
The Room h and w are varied from 8 to 22
with the rest of fixed default parameters
```

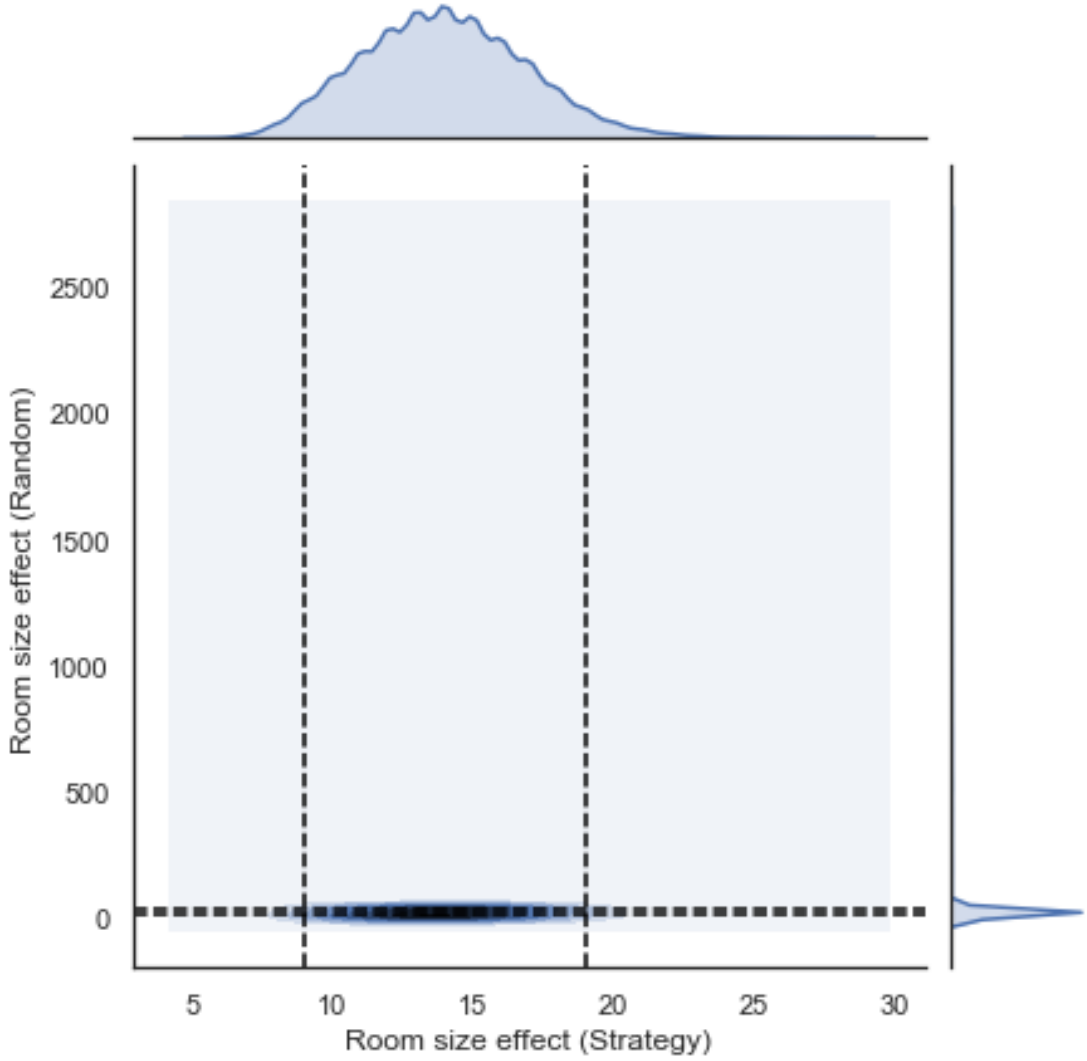
```
"""
room_size= [*range(8,22)]
trails=1000

E8_steps = []
E8_median_time=[]
E8_mode_time=[]
E8_mean_time=[]

for num in room_size:
    counts=[]
    for i in range(trails):
        sim =crowd_egress(room_h=num, room_w=num, exit_random=False)
        sim.initialize()
        while np.count_nonzero(sim.current_state) != 0:
            sim.update()
        counts.append(sim.time)
        E8_steps.append([num,sim.time])

    E8_median_time.append(np.median(counts))
    E8_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E8_mode_time.append(max(set(counts), key=counts.count))

#E8_input = [_[0] for _ in E8_steps]
E8_time = [_[1] for _ in E8_steps]
```

[26]:
```
#Histogrm of the expriment times with
#their frequencies


print ("95% CI of Strategic Placement:", sts.t.interval(0.95, len(E8_time)-1,
                                                        loc=np.
 ↪mean(E8_time), scale=sts.sem(E8_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E8_time, 0.05),np.
 ↪quantile(E8_time, 0.95)))
print(sts.describe(E8_time))
print("--------------------------------")
print ("95% CI of Random Placement:", sts.t.interval(0.95, len(E4_time)-1,␣
 ↪loc=np.mean(E4_time),
                                                    scale=sts.sem(E4_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E4_time, 0.05),np.
 ↪quantile(E4_time, 0.95)))
print(sts.describe(E4_time))
```

```
# #finding the 90% percentile of the data

# E8_zoomed_times=[]
# for i in E8_time:
#     if (i>=np.quantile(E8_time, 0.05)) and (i<=np.quantile(E8_time, 0.95)):
#         E8_zoomed_times.append(i)

# plt.figure(figsize=(12, 5))
# plt.subplot(1, 2, 1)

plt.hist(E8_time,bins = int(np.sqrt(len(E8_time))), color =␣
 ↪"blue",label='Strategy')
#plt.hist(E4_time,bins = int(np.sqrt(len(E4_time))), color =␣
 ↪"red",label='Random',alpha=0.5)
plt.title("Room size effect on evacuation times")
plt.ylabel("Frequency")
plt.legend()


with sns.axes_style("white"):
    fig = sns.jointplot(x=E8_time, y=E4_time, kind='kde', n_levels=10)
    fig.ax_joint.axvline(x=np.quantile(E8_time, 0.05), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axvline(x=np.quantile(E8_time, 0.95), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axhline(y=np.quantile(E4_time, 0.05), linestyle='--', color =␣
 ↪'black')
    fig.ax_joint.axhline(y=np.quantile(E4_time, 0.95), linestyle='--', color =␣
 ↪'black')
    fig.set_axis_labels('Room size effect (Strategy)', 'Room size effect␣
 ↪(Random)')
plt.show()
```

```
95% CI of Strategic Placement: (13.942147754032137, 14.038137960253577)
05th-95th percentiles of the data: (9.0, 19.0)
DescribeResult(nobs=14000, minmax=(6, 28), mean=13.990142857142857,
variance=8.393645239966121, skewness=0.26926840092858934,
kurtosis=-0.01860205887575983)
--------------------------------
95% CI of Random Placement: (26.237506012189062, 29.52120827352522)
05th-95th percentiles of the data: (10.0, 28.0)
DescribeResult(nobs=14000, minmax=(6, 2773), mean=27.879357142857142,
variance=9822.551270391761, skewness=11.976010573397875,
kurtosis=176.1314612707289)
```

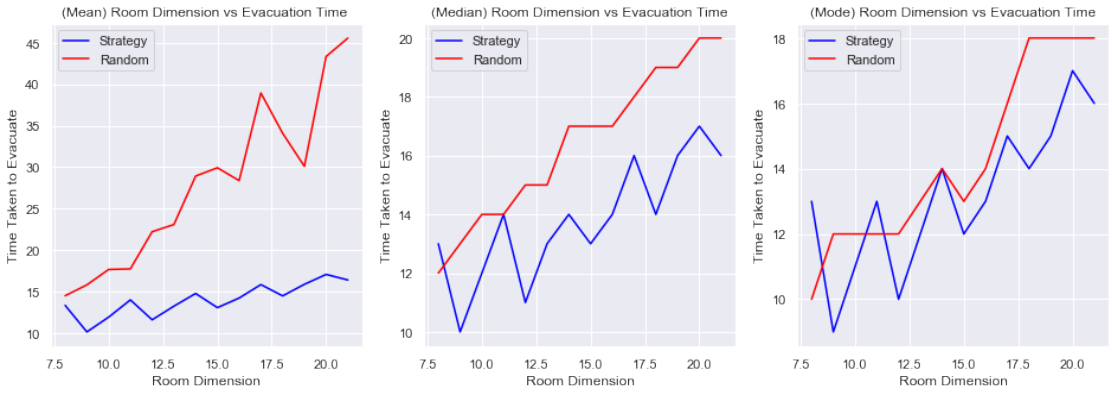

Room size effect on evacuation times
Strategy
Frequency
2000
1750
1500
1250
1000
750
500
250
0
5
10
15
20
25




[27]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)

plt.plot(room_size,E8_mean_time, color = "blue",label='Strategy')
plt.plot(room_size,E4_mean_time, color = "red",label='Random')

plt.title("(Mean) Room Dimension vs Evacuation Time ")
plt.xlabel("Room Dimension")
plt.ylabel("Time Taken to Evacuate")
plt.legend()
```

```
plt.subplot(1, 3, 2)

plt.plot(room_size,E8_median_time, color = "blue",label='Strategy')
plt.plot(room_size,E4_median_time, color = "red",label='Random')
plt.title("(Median) Room Dimension vs Evacuation Time")
plt.xlabel("Room Dimension")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

plt.subplot(1, 3, 3)

plt.plot(room_size,E8_mode_time, color = "blue",label='Strategy')
plt.plot(room_size,E4_mode_time, color = "red",label='Random')
plt.title("(Mode) Room Dimension vs Evacuation Time")
plt.xlabel("Room Dimension")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

#plt.show
```

[27]: <matplotlib.legend.Legend at 0x58c3dd0>




### Section-3 Model Extension

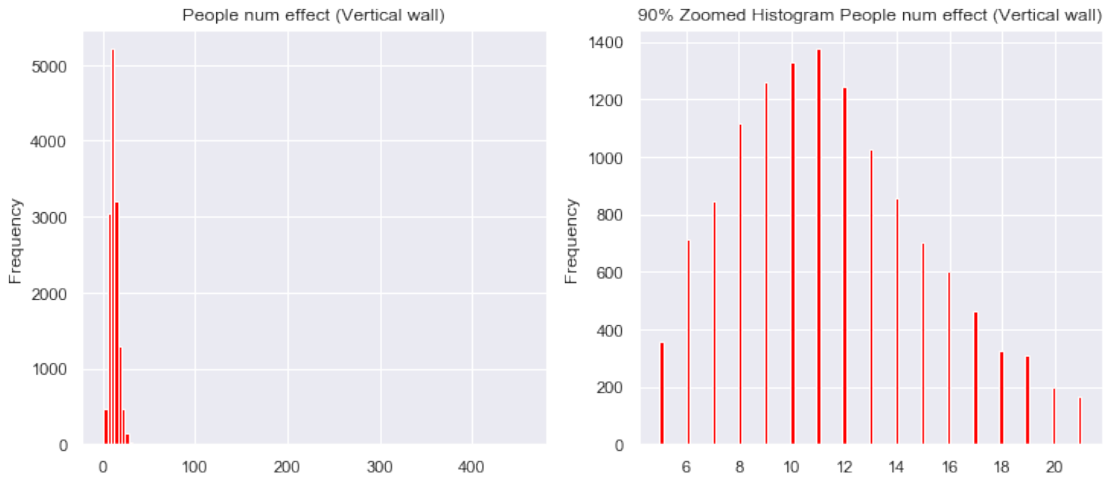
[28]:
```
"""
Experiment-8

Having Wall vs Not Having Wall
on the evacuation time

The number of people are varied from 1 to 28
with the fixed default parameters
```

```
Each step is counted as one interval in this scenario.

"""


people_num= [*range(1,29)]
trails=500


E9_steps = []
E9_median_time=[]
E9_mode_time=[]
E9_mean_time=[]

for num in people_num:

    #vertical wall
    counts=[]
    for i in range(trails):
        sim =crowd_egress(people_num=num,wall_size=5,wall_direction='v')
        sim.initialize()
        while (np.count_nonzero(sim.current_state)-sim.wall_size) != 0:
            sim.update()
        counts.append(sim.time)
        E9_steps.append([num,sim.time])

    E9_median_time.append(np.median(counts))
    E9_mean_time.append(np.mean(counts))
    #calculating the mode of hist by finding
    #the max number of counts in the list
    E9_mode_time.append(max(set(counts), key=counts.count))
E9_time = [_[1] for _ in E9_steps]
```

[29]:
```
E9_time = [_[1] for _ in E9_steps]
#Histogrm of the expriment times with
#their frequencies

print ("95% CI Vertical Wall:", sts.t.interval(0.95, len(E9_time)-1, loc=np.
 ↪mean(E9_time), scale=sts.sem(E9_time)))
print ("05th-95th percentiles of the data:", (np.quantile(E9_time, 0.05),np.
 ↪quantile(E9_time, 0.95)))
print(sts.describe(E9_time))


#finding the 90% percentile of the data
E9_zoomed_times=[]
```

```
for i in E9_time:
    if (i>=np.quantile(E9_time, 0.05)) and (i<=np.quantile(E9_time, 0.95)):
        E9_zoomed_times.append(i)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)

plt.hist(E9_time,bins = int(np.sqrt(len(E9_time))), color = "red")
plt.title("People num effect (Vertical wall)")
plt.ylabel("Frequency")


plt.subplot(1, 2, 2)

plt.hist(E9_zoomed_times,bins = int(np.sqrt(len(E9_zoomed_times))), color =␣
 ↪"red")
plt.title("90% Zoomed Histogram People num effect (Vertical wall)")
plt.ylabel("Frequency")

plt.show()
```

```
95% CI Vertical Wall: (13.010429297879787, 13.640856416405926)
05th-95th percentiles of the data: (5.0, 21.0)
DescribeResult(nobs=14000, minmax=(1, 457), mean=13.325642857142856,
variance=362.0483173236864, skewness=12.022603451154671,
kurtosis=174.81909355206852)
```




[30]:
```
plt.figure(figsize=(16, 5))

plt.subplot(1, 3, 1)
```

```
plt.plot(people_num,E9_mean_time, color = "red",label='Vertical wall')
plt.plot(people_num,E2_mean_time, color = "green",label='no walls')

plt.title("(Mean) Wall_Size vs Evacuation Time")
plt.xlabel("Wall Size")
plt.ylabel("Time Taken to Evacuate")
plt.legend()



plt.subplot(1, 3, 2)

plt.plot(people_num,E9_median_time, color = "red",label='Vertical wall')
plt.plot(people_num,E2_median_time, color = "green",label='no walls')
plt.title("(Median) Wall_Size vs Evacuation Time")
plt.xlabel("Wall Size")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

plt.subplot(1, 3, 3)

plt.plot(people_num,E9_mode_time, color = "red",label='Vertical wall')
plt.plot(people_num,E2_mode_time, color = "green",label='no walls')
plt.title("(Mode) Wall_Size vs Evacuation Time")
plt.xlabel("Wall Size")
plt.ylabel("Time Taken to Evacuate")
plt.legend()

plt.show
```

[30]: <function matplotlib.pyplot.show(*args, **kw)>