Team PAD: Fire Evacuation Simulation – An Algorithmic Approach

Devesh Sharma, 19103
Data Science and Engineering
Indian Institute of Science Education and Research, Bhopal

Praharsh Nanavati, 19222
Data Science and Engineering
Indian Institute of Science Education and Research, Bhopal

R S Anudeep, 19230
Data Science and Engineering
Indian Institute of Science Education and Research, Bhopal

***Abstract*—With the development of science and technology, the design of modern architecture is becoming more and more complicated. Large buildings are densely populated, with various structures and complex functions.**

In case of sudden disasters (fire, earthquake, gas leakage, etc.), the evacuation is inefficient due to the lack of effective evacuation guidance and panic psychological instructions. On the one hand, large buildings usually have a larger internal area, their channel setting and internal structure design are relatively complex, and the overloaded electricity may easily cause fire. On the other hand, the fire smoke and fire in large buildings spread over a wide range of areas, and it is easy to produce chimney effect in elevator well, exhaust pipe, etc.

Therefore, efficient and safe evacuation of people in building fires has become the focus of research, and it is of great significance to protect the safety of life and property in large public buildings.

This project aims to provide the best path for evacuation for multiple agents on a given floor plan for multiple fire origination points. We approach this problem through a maze solver algorithm [10] which uses the Dijkstra Algorithm to solve for mazes. We simply extend this idea to floor plans, where the furniture and walls act as static obstacles and the spreading fire acts as a dynamic obstacle.

We are able to provide decent results using this novel idea for various floor plans which are part of the ROBIN dataset.

I. Introduction

Automation has been an active field of research in recent times. Applying automation techniques to real-world problems is the need of the hour. Nowadays, Artificial Intelligence is playing a major role in redefining solutions to disaster management through various algorithms and optimization techniques. These algorithms are widely implemented and deployed as they are easy to use, optimal in nature, less chaotic and most importantly, effective.

Managing evacuation on a large scale in confined spaces can prove to be a challenging problem. Things can go haywire due to lack of escape organisation and structured planning. Agents involved in such environments often don't behave rationally and are in a state of chaos and panic. At such points in time, it is beneficial to have a robust evacuation plan.

We have considered the case of breaking out of fires in residential buildings as there has been an increase in such mishaps lately due to exponential growth in infrastructure.

We discovered few papers and projects which seemed to be relevant in our context and they provided major insights on the whole area of real-time optimization. We will list a few of them here:

In their paper [1], the authors build a novel evacuation plan algorithm for multi exit buildings based on an indoor route network model. Evacuees are grouped by their location proximity, classified into several evacuation zones and are sorted by their shortest path length and optimized accordingly.

The objectives of the proposed algorithm include minimizing the total evacuation time of all evacuees, the travel time of each evacuee, avoiding traffic congestion, balancing traffic loads among different exits, and achieving high computational efficiency. The influences of group number, group size, evacuation speed on the total evacuation time are discussed on a single-exit network, and that of partitioning methods and evacuation density on the performance and applicability in different congestion levels are also discussed on a multi-exit network. Results demonstrate that their algorithm has a higher efficiency and performs better for evacuations with a large occupant density.

In this paper [2], the authors build a Risk-based Model of Evacuation Route Optimization in fire situations. They take into account various factors like the variation laws of risk indicators such as temperature, thermal radiation, the concentration of toxic gas etc.

The scheme of the best route for an evacuee at each location is the one along with the shortest time and minimal risk and suggested based on the Dijkstra algorithm. And then, one case is presented and result indicates that this model can aid people to avoid crowdedness and evacuate as soon as possible under fire accident. The risk-based model is also useful for the evacuation planning.

In this piece of research, [3], the authors build a 3D model which uses a greedy algorithm which identifies and analyzes buildings, roads, vehicles, pedestrians, trees and other elements in the region, and proposes an intelligent route planning method based on the specified 3D model. This method provides an effective and feasible intelligent route planning scheme for the rescue and decision commanders and greatly improves the efficiency of the rescue work in path selection.

The position of the evacuation doors is ruled by design specifications and building structures, in this study, a constraint-based model which includes space constraints and design constraints is proposed to generate the optimal door positions which minimize evacuation distance.

The optimal combinations of the evacuation doors positions are obtained with the branch and bound algorithm in the model. Compared with the existing work, this approach avoids repeated modification of the design drawings and has significantly reduced the evacuation time by automatically generating optimal door positions.

In order to find the optimal path for emergency evacuation, this paper [4] proposes a dynamic path optimization algorithm based on real-time information to search the optimal path. It uses the Dijkstra algorithm to get the prior evacuation network which includes evacuation paths from each node to the exit port.

The proposed method uses the breadth-first search strategy to solve the path optimization problem based on the prior evacuation network. Because the prior evacuation network includes global optimal evacuation paths from each node to the exit port, the breadth-first search algorithm only searches local optimal paths to avoid the blockage node or dangerous area. Because the online optimization solves a local pathfinding problem and the entire topology optimization is an offline calculation, the proposed method can find the optimal path in a short time when the accident situation changes.

The simulation tests the performances of the proposed algorithm with different situations based on the topology of a building, and the results show that the proposed algorithm is effective to get the optimal path in a short time when it faces changes caused by the factors such as evacuee size, people distribution, blockage location, and accident points.

After getting exposed to the kind of work these papers have showcased, we thought of using the ROBIN dataset by vectorizing it for various objectives such as image processing, obstacle detection etc. The dataset contains about 500 images of generalised floor plans. We processed 70 images at random from this dataset to test our algorithm.

The original dataset which we used can be found at: https://github.com/gesstalt/ROBIN.git

We started looking at the manipulations of floor plans and came across a different flavour of papers. These papers helped us in setting up our problem. We explore these papers in the subsequent sections.

II. Contributions

Our project provides the best path for evacuation for multiple agents on a given floor plan for multiple fire origination points.

We approach this problem through a maze solver algorithm [10] which uses the Dijkstra Algorithm to solve for mazes. We simply extend this idea to floor plans, where the furniture and walls act as static obstacles and the spreading fire acts as a dynamic obstacle.

The team members have contributed equally to the working environment. All the modules of the running code were programmed in the presence of all team members.

The work was divided in the following manner:

1) Devesh Sharma provided the input which includes the processed images including binarification along with the dictionary with exits of each floor plan.

   Originally, the input images would be processed using image segmentation and object detection techniques for classifying each room and obstacles into unreachable nodes.

   Other ideas consisted of automating the process of detection of end points in each image and automatic removal of doors using template matching. These techniques are now included in our future work.

2) Praharsh Nanavati referred to maze solving modules to apply the working algorithm. The coding module [10] provides an environment for maze solving.

   Our working algorithm extends this idea to floor plans, where the optimized path avoids all obstacles in the floor plan including the spreading fire (radially outward direction). The algorithm does not discriminate between the walls, fire and obstacles in its path. Our algorithm works for evacuating multiple agents where the fire is spreading in multiple areas.

3) R S Anudeep processed the results of the algorithm into the desired output using imageio and pyglet. Originally, we thought of using Motion Planning and Path planning for our desired output. However, after looking at the output of the algorithm, the original simulation idea did not seem feasible.

III. Background

The crux of our work is mainly covered by the Dijkstra Algorithm, which is one of the more popular basic graph theory algorithms. It is used to find the shortest path between nodes on a directed graph. We define this in Algorithm 1.

After understanding papers mentioned in the Introduction, we realised that manipulations of floor plans had become a necessity. For this, we explored a different flavour of papers which focused more on how to manipulate floor plan images and tweak them according to our needs.

In this paper [5], a novel synthetic workflow is presented for procedural generation of room relation graphs of floor plans from structured architectural datasets.

Different from classical floor plan generation models, which are based on strong heuristics or

Algorithm 1 The Dijkstra Algorithm

Required: Node x, Node y.
(1) Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

(2) Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.

(3) For the current node, consider all of its unvisited neighbours and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. Otherwise, the current value will be kept.

(4) When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

(5) If the destination node has been marked visited when planning a route between two specific nodes or if the smallest tentative distance among the nodes in the unvisited set is infinity, then stop. The algorithm has finished.

(6) Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Output: Shortest path between nodes x,y.

low-level pixel operations, their method relies on parsing vectorized floor plans to generate their intended organizational graph for further graph-based deep learning.

This research work presents the schema for the organizational graphs, describes the generation algorithms, and analyzes its time/space complexity. As a demonstration, a new dataset called CubiGraph5K is presented. This dataset is a collection of graph representations generated by the proposed algorithms, using the floor plans in the popular CubiCasa5K dataset as inputs. The aim of this contribution is to provide a matching dataset that could be used to train neural networks on enhanced floor plan parsing, analysis and generation in future research.

However, the previous paper focused a bit too much on vectorizing floor plans efficiently and hence the cost of applying it was a bit high and hard to achieve. This led us to the paper [6], which seemed relevant, where there was no need to vectorize images but involved image segmentation.

In this paper, their first method develops an original graph construction method. They also analyze different types of graph construction method as well as the influence of various feature descriptors. The proposed graph, called a local/global graph, encodes adaptively the local and global image structure information.

Their second method derives a discriminative affinity graph that plays an essential role in graph-based image segmentation. A new feature descriptor, called weighted color patch, is developed to compute the weight of edges in an affinity graph.

This new feature is able to incorporate both color and neighborhood information by representing pixels with color patches. Furthermore, they assign both local and global weights adaptively for each pixel in a patch in order to alleviate the over-smooth effect of using patches. The extensive experiments show that our method is competitive compared to the other standard methods with multiple evaluation metrics.

The third approach combines superpixels, sparse representation, and a new midlevel feature to describe superpixels. The new mid-level feature not only carries the same information as the initial low-level features, but also carries additional contextual cue.

The authors validate the proposed mid-level feature framework on the MSRC dataset, and the segmented results show improvements from both

qualitative and quantitative viewpoints compared with other state-of-the-art methods.

Now, to implement the methods mentioned in the previous paper, we needed to convert our floor plan into a directed graph. Their second approach, which was to derive a discriminative affinity graph by adding a new feature descriptor, called weighted color patch was interesting.

We came across another paper [7], which deals with dividing the floor plan into segments. In this paper the authors concentrate on indoor navigation considering obstacles represented as polygons.

They introduce a specific space subdivision based on a simplified floor plan to build the indoor navigation network. The experiments demonstrate that they are able to navigate around the obstacles using the proposed network.

Considering to well-known path-finding approaches based on Medial Axis Transform (MAT) or Visibility Graph (VG), the approach in this paper provides a quick subdivision of space and routes, which are compatible with the results of VG. However, it seems the authors achieved this manually and hence was a little different from what we had anticipated.

However, this paper gave us the idea of using each pixel as a node of a graph where regional subdivision would no longer be required.

Another paper [8] detects objects on the instances of the ROBIN dataset and provides a pathway from door-to-door for blind people. This provides the path from a room to another. This paper synthesizes textual description from a given floor plan image for the visually impaired.

With the help of a text reader software the target user can understand the rooms within the building and arrangement of furniture to navigate. This paper is the first framework for describing a floor plan and giving direction for obstacle-free movement within a building.

The paper describes learning 5 classes of room categories from 1355 room image samples under a supervised learning paradigm. These learned annotations are fed into a description synthesis framework to yield a holistic description of a floor plan image. We demonstrate the performance of various supervised classifiers on room learning. The authors also provide a comparative analysis of system generated and human written descriptions.

This paper gives state of the art performance on challenging, real-world floor plan images. This work can be applied to areas like understanding floor plans of historical monuments, stability analysis of buildings, and retrieval. The output of this paper was especially inspiring to us and our output resonates with the results of this paper.

After reading and exploring all methods discussed in all the papers cited above, we came up with a different, seemingly novel idea. The floor plans provided in the ROBIN dataset could be converted into binary images and be treated as mazes. None of these papers mentioned above perceive floor plans as mazes.

By doing so, finding the shortest way out from a given point is easily calculable and not computationally expensive. Using the modules present here [9] [10], we treat our floor plans as mazes and have defined dynamic obstacles for representing spreading fires. We explain our idea further in the Methodology section.

IV. Materials and Methods

A. Format

1) Input:

- Image Number: For a Preloaded image from our curated dataset.
- Number of agents to be evacuated.
- X-Y coordinates of each agent's starting point.
- Number of fire origination points.
- X-Y coordinates of each fire point's starting point.

2) Output:

- A simulation of all agents being evacuated while the multi-point fire spreads using path planning on the floor plan image.

B. Methodology

The methodology of the working environment can be broadly classified into four categories, namely:

```python
img_no = int(input("Enter image number (0-69): "))                    # Index to access image from dataset

img = cv2.imread('../image_'+str(img_no)+'.png')     # Read image using opencv
plt.figure(figsize=(5,5))
plt.imshow(img)                                                       # Show the image on the screen
plt.show()

start_lst = []                                                        # List to store starting points of agents
agent_no = int(input("Enter number of agents: "))
for i in range(agent_no):
    startx = int(input("Enter the X-coordinate of the "+str(i+1)+"th start node: "))
    starty = int(input("Enter the Y-coordinate of the "+str(i+1)+"th start node: "))
    start_lst.append((startx,starty))

fire_no = int(input("Enter number of fire origination points: "))
fire_orig = []                                                        # List to store starting points of fire
for i in range(fire_no):
    fire_origx = int(input("Enter the X-coordinate of the "+str(i+1)+"th fire origination point: "))
    fire_origy = int(input("Enter the Y-coordinate of the "+str(i+1)+"th fire origination point: "))
    fire_orig.append((fire_origx,fire_origy))
```

Fig. 1. The code for taking all required attributes as input.

1) Data cleansing and wrangling.

2) Algorithm formation

3) Algorithm application.

4) Output Generation.

1) Data cleansing and wrangling: The images in the raw data required some manipulation for turning it into suitable formats. Firstly, all dataset instances were converted to binary images and then the physical doors on the floor plan needed to be manually removed.

The end point of all images are then stored in a dictionary. This could be done in more efficient ways as discussed in further sections.

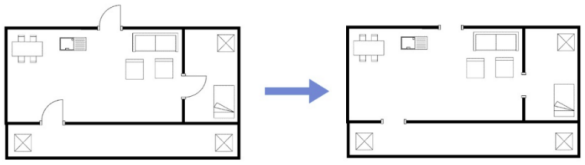


Fig. 2. A sample image from the ROBIN dataset before and after manual processing. We tried automating this process using the Template Matching feature from OpenCV. We were only partially successful in doing so. The code for this is provided in the Code Trials Folder.

2) Algorithm formation: Now that all images are processed, we needed to form an efficient algorithm which provides the most optimal route for multiple agents through spreading fire. We approach this problem by treating floor plans as mazes where the spreading fire will also be treated as a dynamic obstacle.

Mazes present a programming problem that we may solve using shortest-path techniques like Dijkstra's algorithm. Therefore, extension of this idea to floor plans is viable.

We use the code provided in [10] as a function module. The author creates a Vertex class which helps in keeping track of the coordinates. It is necessary to keep track of the parent node to reconstruct the entire path once the distances are calculated.

The author creates a matrix of Vertices, representing the 2D layout of pixels in an image. This will be the basis for our Dijkstra's algorithm graph. The author also maintains a min-heap priority queue to keep track of unprocessed nodes.

There are also a few helper functions defined within the code:

1) $get\_neighbors$: Function to return neighbors.
2) $bubble\_up$: For min-heap priority queue.
3) $bubble\_down$: For min-heap priority queue.
4) $get\_distance$: Gives the distance between two nodes on an image.
5) $drawPath$: Draws path between two nodes on images.
6) $find\_shortest\_path$: Returns optimal path.

```python
#Return neighbor directly above, below, right, and left
def get_neighbors(mat,r,c):
    shape=mat.shape
    neighbors=[]
    #ensure neighbors are within image boundaries
    if r > 0 and not mat[r-1][c].processed:
        neighbors.append(mat[r-1][c])
    if r < shape[0] - 1 and not mat[r+1][c].processed:
            neighbors.append(mat[r+1][c])
    if c > 0 and not mat[r][c-1].processed:
        neighbors.append(mat[r][c-1])
    if c < shape[1] - 1 and not mat[r][c+1].processed:
            neighbors.append(mat[r][c+1])
    return neighbors
```

We can think of an image as a matrix of pixels. Each pixel (for simplicity's sake) has an RGB value of 0, 0, 0 (black) or 255, 255, 255 (white). Our goal is to create a shortest path which starts in the white and does not cross into the black boundaries.

To represent this goal we can treat each pixel as a node and draw edges between neighboring pixels with edge lengths based on RGB value differences.

```
def bubble_up(queue, index):
    if index <= 0:
        return queue
    p_index=(index-1)//2
    if queue[index].d < queue[p_index].d:
            queue[index], queue[p_index]=queue[p_index], queue[index]
            queue[index].index_in_queue=index
            queue[p_index].index_in_queue=p_index
            quque = bubble_up(queue, p_index)
    return queue
```

Fig. 4. This function is used for bubbling up an element in a min heap priority queue

```
def bubble_down(queue, index):
    length=len(queue)
    lc_index=2*index+1
    rc_index=lc_index+1
    if lc_index >= length:
        return queue
    if lc_index < length and rc_index >= length: #just left child
        if queue[index].d > queue[lc_index].d:
            queue[index], queue[lc_index]=queue[lc_index], queue[index]
            queue[index].index_in_queue=index
            queue[lc_index].index_in_queue=lc_index
            queue = bubble_down(queue, lc_index)
    else:
        small = lc_index
        if queue[lc_index].d > queue[rc_index].d:
            small = rc_index
        if queue[small].d < queue[index].d:
            queue[index],queue[small]=queue[small],queue[index]
            queue[index].index_in_queue=index
            queue[small].index_in_queue=small
            queue = bubble_down(queue, small)
    return queue
```

Fig. 5. This function is used for bubbling down an element in a min heap priority queue

```
def drawPath(img,path, thickness=2):
    '''path is a list of (x,y) tuples'''
    x0,y0=path[0]
    for vertex in path[1:]:
        x1,y1=vertex
        cv2.line(img,(x0,y0),(x1,y1),(255,0,0),thickness)
        x0,y0=vertex
```

Fig. 6. This function helps in drawing the evacuation path on the image for an agent.

We will use the Euclidean squared distance formula and add 0.1 to ensure no 0-distance path lengths which is a requirement for the Dijkstra Algorithm.

This formula makes the distance of crossing through the floor plan boundary prohibitively large. As we can see, the shortest path from source to destination will clearly be around the barrier, not through it.

3) Algorithm applications:

1) We take the Image Number, Number of agents, Start node coordinates, number of fire origination points and coordinates of fire points as input.

2) The end point for the selected image is extracted from the manually constructed dictionary.

```
def find_shortest_path(img,src,dst):
    pq=[] #min-heap priority queue
    source_x=src[0]
    source_y=src[1]
    dest_x=dst[0]
    dest_y=dst[1]
    imagerows,imagecols=img.shape[0],img.shape[1]
    matrix = np.full((imagerows, imagecols), None) #access by matrix[row][col]
    for r in range(imagerows):
        for c in range(imagecols):
            matrix[r][c]=Vertex(c,r)
            matrix[r][c].index_in_queue=len(pq)
            pq.append(matrix[r][c])
    matrix[source_y][source_x].d=0
    pq=bubble_up(pq, matrix[source_y][source_x].index_in_queue)

    while len(pq) > 0:
        u=pq[0]
        u.processed=True
        pq[0]=pq[-1]
        pq[0].index_in_queue=0
        pq.pop()
        pq=bubble_down(pq,0)
        neighbors = get_neighbors(matrix,u.y,u.x)
        for v in neighbors:
            dist=get_distance(img,(u.y,u.x),(v.y,v.x))
            if u.d + dist < v.d:
                v.d = u.d+dist
                v.parent_x=u.x
                v.parent_y=u.y
                idx=v.index_in_queue
                pq=bubble_down(pq,idx)
                pq=bubble_up(pq,idx)

    path=[]
    iter_v=matrix[dest_y][dest_x]
    path.append((dest_x,dest_y))
    while(iter_v.y!=source_y or iter_v.x!=source_x):
        path.append((iter_v.x,iter_v.y))
        iter_v=matrix[iter_v.parent_y][iter_v.parent_x]


    path.append((source_x,source_y))
    return path
```

Fig. 7. This is the function which helps us in calculating the shortest path for each agent while avoiding all black nodes.

3) For each agent, we calculate the shortest path from its starting point to the end point so that the agents can start moving. We consider the spreading fire as a dynamic obstacle and simultaneously calculate the path to reach the end point.

4) Instances of a gif are formed by slicing for simulation and output generation.

4) Output Generation: Imageio is a Python library that provides an easy interface to read and write a wide range of image data. We have used it to save the final output of our working environment.

Pyglet is a cross-platform windowing and multimedia library for Python, intended for developing games and other visually rich applications. However, we have used it to show only the output of our code in a gif format.

V. Results

We were successfully able to run our simulations on all images we processed, with multiple agents and fire points.

Our working environment will also work for floor plans that are binary images and not a part of the ROBIN dataset regardless of what it's components look like. This is because we use RGB mapping to avoid the blackened nodes.

Our simulation was two-dimensional in nature with the path for each agent shown in red and all obstacles including the spreading fire in black.

By treating the floor plan images as a maze, applying the gridding technique was a straightforward process and there was no need to perform motion or path planning.

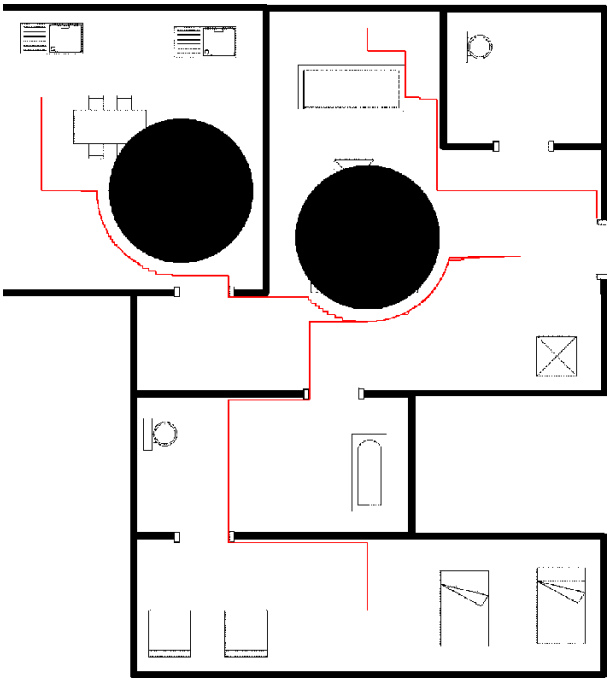
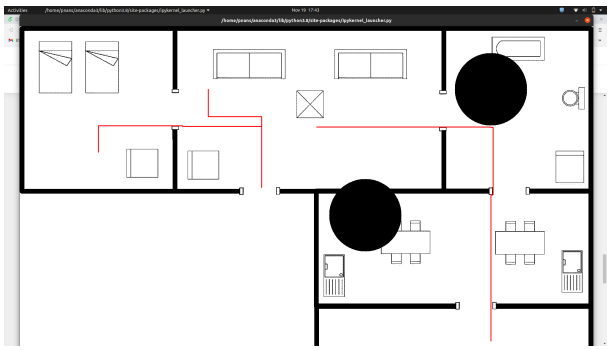


Fig. 8. This is how the output of our code looks like. In this particular case, three agents are getting evacuated and the fire originates and spreads from two points. In our working environment, this output is in the form a gif. By treating the floor plan images as a maze, applying the gridding technique was a straightforward process. Our algorithm is sufficiently accurate and computationally very inexpensive.

Our results are not the most optimal, which is visible in many outputs as there is hardly any diagonal movement of the agents across the floor. However, our algorithm is sufficiently accurate and computationally very inexpensive, making it effective.



VI. Discussions

Our execution environment provides a path for evacuation for multiple agents on a given floor plan for multiple fire origination points. We approached this problem through a maze solver algorithm which uses the Dijkstra Algorithm to solve for mazes.

We simply extended this idea to floor plans, where the furniture and walls acted as static obstacles and the spreading fire acted as a dynamic obstacle.

There were a few things which we would want to improve in our execution of the project. Here, we list a few of them:

- We removed the doors from each processed floor plan manually, but we would have wanted to automate the process by using template matching, a popular OpenCV feature. We tried this but were able to achieve only partial results.
- We manually compiled a dictionary of exits for each floor plan. But we would like to create a model to detect the exits.
- If one of the fire origination points is at the exit of the floor plan, then in this situation, our algorithm fails.
- We made a simple assumption that the spreading fire or the dynamic obstacle will move

radially outward. This is an oversimplification of the real situation. However, our code would work even if the fire spread as it does in the natural world.

VII. Conclusion

Artificial Intelligence is playing a major role in redefining solutions to disaster management through various algorithms and optimization techniques.

We have considered the case of breaking out of fires in residential buildings as there has been an increase in such mishaps lately due to exponential growth in infrastructure.

We believe using maze solvers on floor plan images is a simple yet seemingly novel idea which is not computationally expensive and provides decent accuracy even though it does not provide the most optimal path.

We would like to automate the whole process regardless of the floor plan images provided and would also like to extend our work to multi-exit and multi-floored buildings.

We were successfully able to run our simulations on all images we processed, with multiple agents and fire points. In conclusion, we learnt many new techniques while implementing this module and it was a good learning experience.

References

[1] Han, L., Guo, H., Zhang, H., Kong, Q., Zhang, A., Gong, C. (2020). An Efficient Staged Evacuation Planning Algorithm Applied to Multi-Exit Buildings. ISPRS International Journal of Geo-Information, 9(1), 46.

[2] Li, Jing-jing, and Hong-ya Zhu. "A risk-based model of evacuation route optimization under fire." Procedia engineering 211 (2018): 365-371.

[3] Gao, Han, et al. "Building evacuation time optimization using constraint-based design approach." Sustainable cities and society 52 (2020): 101839.

[4] Huajun Zhang, Qin Zhao, Zihui Cheng, Linfan Liu, and Yixin Su. Dynamic path optimization with real-time information for emergency evacuation.Mathematical Problems in Engineering, 2021, 2021.

[5] Lu, Yueheng, et al. "CubiGraph5K-Organizational Graph Generation for Structured Architectural Floor Plan Dataset." (2021).

[6] Wang, Xiaofang. Graph based approaches for image segmentation and object tracking. Diss. Ecully, Ecole centrale de Lyon, 2015.

[7] Xu, Man, Shuangfeng Wei, and Sisi Zlatanova. "An indoor navigation approach considering obstacles and space subdivision of 2D plan." Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci 41 (2016): 339.

[8] Goyal, Shreya, et al. "SUGAMAN: describing floor plans for visually impaired by annotation learning and proximity-based grammar." IET Image Processing 13.13 (2019): 2623-2635.

[9] Maxwellreynolds. "Maxwellreynolds/Maze." GitHub, https://github.com/maxwellreynolds/Maze.

[10] Reynolds, Max. "Solving Mazes with Python." Medium, Towards Data Science, 29 June 2020, https://towardsdatascience.com/solving-mazes-with-python-f7a412f2493f.