Evacuation Simulation: EvacX

Members: Diana Lysova, Andrew Chen, Noah Kochavi, Conrad French

Context

In our modern world, we unfortunately have to deal with many unprecedented events that may happen in our lives. From flooding in coastal cities, tornadoes in the midwest, to even the horrible intruder threats in our schools. In many cases, local governments have been caught off-guard, with many not even mounting a proper response until hours later. Denizens of affected or at risk communities may wonder, "is there anything that can be done?"

Our group says that there is. EvacX is a simulator that will be able to mount a much better response than past measures. In this project, we will use active intruder threats as an example to display the suite of functionality EvacX is capable of. Although, EvacX should ideally be able to mount a response against a variety of threats.

Goal

The goal of the simulation is to guide as many victims to the exits as possible. Intruders will be present and will try to eliminate as many victims as possible, so it is important for the system to make the best decisions in both the short and long term.

To achieve this we will be using several graph algorithms as well as levels of optimization to allow the system to run in an efficient and timely manner

Floor-Plan

To represent the floor-plan of the establishment we were running the simulation in, we decided on a 2D matrix consisting of equal blocks.

There are 2 types of blocks with danger levels:

- Room: Requires doors to enter/exit; Safest
- Hall: Does not require doors to enter/exit; Only needs door to access adjacent rooms; Least safe

Algorithms

1. **Breadth-First Search** finds the shortest path from two vertices in an unweighted graph or 2D matrix
2. **Edmonds Karp** find the maximum amount of flow from a source to a sink (destination). In this context, the algorithm is finding the maximum amount of people that can be evacuated from a source to the exits.
3. **K-Means** is an unsupervised machine learning algorithms that classifies inputs based on some metrics. In this context, the metrics are the proximity from the intruder and the location of each victim. From these metrics, K-Means will generate *clusters* of the danger levels of victims. The higher a danger level of a victim, the more priority the system will give to them.

Data Structures

1. **Quad-Tree** is a tree data structure whose nodes are positions within a display. A node is only present if there is at least one item within a quadrant. In this context, only quadrants that have victims or the intruder and stored, saving memory and allowing for fast collision detection.

2. **Union-Find** is a tree data structure whose nodes represent positions within a building. Lookups - is one hallway connected to another - are very fast, O(LogN)

Ethical Problems

There were several ethical issues that are present in this project

1. Similar to the trolley problem, if forced, should the system sacrifice a smaller group of people voluntarily to save a larger group or do nothing but take the risk of losing more people overall?

2. Between two people who are in immediate danger, how should the system choose whom to save? Should it be a toss of a coin or would it be moral to not choose but risk losing both?

3. For a group of people who are in no immediate danger, should the system bring them into more danger to bring another group out of danger? This may save both but may also cause more losses than if the system were to simply focus on one group.

4. As an intruder is by no means rational, how should the system make decisions without knowing what the intruder may do?

5. If a victim distrusts the system and places the safety of others at risk, should the system try to protect them less?

Possible solutions to these issues

1. The system should never sacrifice one group over another, as this would lead to distrust among victims. Instead, the system should always try to save the most people in the present moment; Even if it may not be the most mathematically optimal.

2. A victim's trust is vital to the functioning of a system. Thus, the system will never choose to bring a person into more danger.

3. A group should never be brought into more danger as once again, the system should not sacrifice safety of any individual or group

4. This is where machine learning comes in. The system should monitor the intruder and try to form a model based off their actions. While this may be a waste sometimes, in the long run, this will save more victims.

5. No, as the system should not make moral decisions like this. The system does its best in these situations, but a victim's safety ultimately lies within him/her.

Since the project is published under the MIT license, others have the freedom to fork and implement it themselves. But my approach will use the above. (Andrew Y Chen)

General Idea

1. The program draws the floor-plan as a 2D matrix
2. Victims are randomly distributed in classrooms & hallways
3. An intruder is introduced in a random classroom or hallway
4. The simulation starts; all algorithms are executed and run until the end
5. The system operates by sending real-time ens notifications to all victims
6. The system classifies victims into danger zones based on their proximity and location within their building (K-Means)
7. The system starts to build a model of the intruder (Library)
8. The system locks all doors adjacent to the intruder
9. The system notifies danger zones 4 & 5 to proceed to the nearest exits (BFS to find path to exit & Edmonds-Karp for maximum rate of evacuation to exit)
10. The system instructs dangers zones 0 & 1 to run away; It'll unlock & lock doors immediately to assist in evacuation
11. The system instructs danger zones 2 to hide & move to safer locations (danger levels 3, 4, and 5)
12. The simulation operates until all living victims make it to the nearest exits

Pseudocode Logic

```
# Classes
class Person:
        def __init__(self, name, id):
                self.name
                self.id = id
        def move(self, (x, y), (x, y)):
                pass

class Victim(Person):
        def __init__(self, name, id, status):
                super(name, id)
                self.status = status
        def listen(self): # listen to instructions from system
                pass

class Intruder(Person):
        def __init__(self, name, id):
                super(name, id)
        def listen(self): # listen to sounds of footsteps
                pass
```

```
        def shoot(self, victim): # shoot at a victim
                pass

class Building:
        def _init_(self, filename):
                self.layout = load(filename) #2D matrix of Blocks

class Block:
        def _init(self, cap, cur):
                self.capacity = cap
                self.current = cur

class Room(Block):
        def _init_(self, cap, cur, doors):
                super(cap, cur)
                self.doors = doors
                self.soundlevel = 0
                self.soundFactor = 3

class Hall(Block):
        def _init_(self, cap, cur, doors):
                super(cap, cur)
                self.doors = doors
                self.soundlevel = 0
                self.soundFactor = 7

# Algorithms
def simulation(building, victims, intruders):
        evac_system(building, victims, intruders)
        return len(victims)

def evac_system(building: Room[][], victims: int[], intruders: int[]):
        paths, model, zones, alive, dead
        while alive in building:
                refine(model)
                paths = [] * alive
                zones = kmeans(alive, 5) # classify victims
                for V in alive: # generate path via BFS
                        paths[V] = BFS((x, y), exit)
                for Z in range(zones) and V in zones[Z]:
                        move(paths[V], exit)

def move((x, y): Position, (x, y): Exit) -> move:


def BFS((x, y): Start, (x, y): End) -> moves[]:
```

```
def kmeans(alive: [], K: int) -> zones[]:

def kill(victim):
        victim.status = dead # victim can be alive or dead
```