रक्षा अनुसंधान एवं विकास संगठन
रक्षा मंत्रालय, भारत सरकार
DEFENCE RESEARCH &
DEVELOPMENT ORGANISATION
Ministry of Defence, Government of India

Project Report
On

Detection of Stampede Using Discrete Event Simulation

Submitted By

Prakersh Kumar Arya
Bachelor of Technology-2023
Computer Science and Engineering Department
Rajkiya Engineering College Sonbhadra
(Churk Uttar Pradesh)

Under the Guidance
of

Vinay Verma
Scientist - E, Ministry of Defence,
Defence R&D Organisation
(Scientific Analysis Group)

रक्षा अनुसंधान एवं विकास संगठन
रक्षामंत्रालय, भारत सरकार
Defence Research and Development Organisation

Ministry of Defence, Government of India

Abstract

Cities are growing at a dizzying pace and they require improved methods to manage crowded areas. Crowd management stands for the decisions and actions taken to supervise and control densely populated spaces and it involves multiple challenges, from recognition and assessment to application of actions tailored to the current situation. To that end, Wi-Fi-based monitoring systems have emerged as a cost-effective solution for the former one. The key challenge that they impose is the requirement to handle large datasets and provide results in near real-time basis. However, traditional big data and event processing approaches have important shortcomings while dealing with crowd management information. In this paper, we describe a novel system architecture for real-time crowd recognition for smart cities and smart buildings that can be easily replicated. The described system proposes a privacy-aware platform that enables the application of artificial intelligence mechanisms to assess crowds' behavior in buildings employing sensed Wi-Fi traces. Furthermore, the present paper shows the implementation of the system in two buildings, an airport and a market, as well as the results of applying a set of classification algorithms to provide crowd management information.

ACKNOWLEDGEMENT

It is really a great opportunity for a student to work under one of the most prestigious organisations of India, DRDO. I am very grateful to almighty God, my teachers and SAG (Scientific Analysis Group, DRDO) for giving me this opportunity.

During this project completion I have faced a lot of confusion and I thank everyone who supported and helped me in solving all my doubts. I have learned a lot of new things from this project and I am really happy that due to this project , I learned something new and most importantly the project was completely based on a real life scenario which gave me an opportunity to work on a real life based project. I am very thankful to **Vinay Verma Sir** for guiding me during this project , he was always available and ready to support whenever I got any doubt. During the initial phase it was a little bit difficult to understand how to proceed , at that time sir guided me about the project.

I would also like to thank DRDO , our college (REC Sonbhadra), faculty members and all my colleagues who also helped me and encouraged me for the completion of this project.

Table of Contents

List of Figures

1.0 INTRODUCTION

1.1 What is Crowd Management?

By crowd, we mainly refer to the average number of individuals present in a par-
ticular place. A place becomes crowded if the total population of the certain area
becomes much more than the capacity. As a result of such crowd, various accidents
may take place. Extreme crowd results individuals in losing control and turning the
place into disaster. Often miscreants use such crowd to do various inhuman activities
like harassing women. Presently, crowd counting is of severe importance in order
to maintain human safety in crowded situations. One such occurrence hap-
pened on 14 April 2015. That day, several women have been sexually harassed at
Dhaka University premises while celebrating the first day of the Bengali year, Pohela
Boishakh. Again, old people may become uncomfortable in overly crowded places.
Another crowd catastrophe took place in the year 2015 during Hajj pilgrimage. That
time, 2236 hajjis lost their lives being rushed during the Hajj in Mecca on September
24. There are many such incidents of massive tragedy due to overcrowded situations.
The crowd may frequently occur in modern society.

1.2 Why is crowd management important?

Being in a crowd presents an unusual number of situations and behaviours in people. A lot of people act in crowds in ways that they wouldn’t act on the street by themselves, and this mentality can lead to disastrous situations if not kept in check by experienced crowd management. For instance, if a crowd is particularly raucous, others may be incited by the atmosphere and energy, and violence can soon follow.

There have been all too many stories about eager and impatient crowds pushing to get closer to the stage and creating a crush that leads to people dying in horrible circumstances, and to prevent this, a clear crowd control programme needs to be in place.

1.3 Benefits of Crowd Management?

First and foremost, effective crowd management helps to ensure the safety of those at an event, from the guests to the staff, and the performers. When an event is taking place, everyone in the venue should be able to enjoy themselves without worrying about their safety.

The consequences of a poorly managed crowd can be disastrous; people can be injured and lives can be lost. Even if an event is poorly managed but nobody is hurt, it may not bode well for future events; people may have less trust in a venue, and sanctions can be imposed for poor event management.

Everything from orderly queues to the crowd during a gig needs to be managed properly. If you’ve ever been in a poorly managed queue, one that seemingly never moves or one where people push in at will, you’ll know how frustrating that can be. This frustration can lead to tempers fraying and altercations happening, so effective management can help minimise the risk of these occurrences.

1.4 Purpose

The main purpose of this project is to detect whether in a crowded area , the crowd is normal or it is above the threshold level , which we will call stampede. By detection of this stampede we can prevent some dangerous event and we can save a major loss.

1.5 Method used here

As the name suggests we will be using a discrete event simulation method for the detection of stampedes over a crowded place. Here we mainly care about the situation at some discrete points or intervals. By using this method we will perform a simulation of a real life scenario , where people will be created randomly and with the help of simulation , we will detect whether the stampede is there or not.

1.6 Scope

So far , we have understood that it is very important to detect stampedes for a better crowd management system , so in all the places where there are high chances of stampedes we can use this project in a generalized way for better crowd management.

1.7 Summary

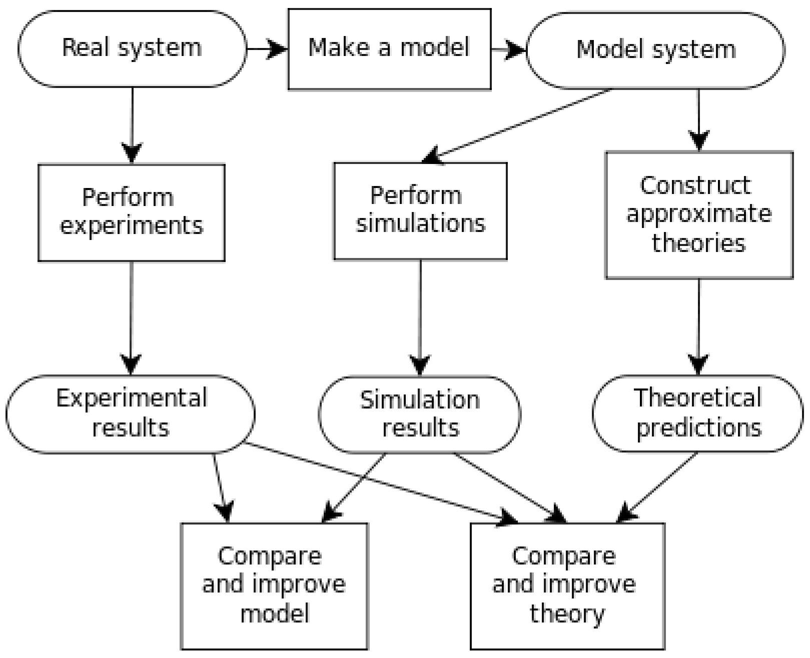
As we know very well that it is very important to detect stampedes for better crowd management , so here we are going to implement the simulation of this real life scenario for better understanding of the problem and how to overcome it . We will be using the method of discrete event simulation for the detection of stampedes .

2.0 Background

2.1 Simulation in Computer Science

Computer simulation is the process of mathematical modeling, performed on a computer, which is designed to predict the behaviour of, or the outcome of, a real-world or physical system. The reliability of some mathematical models can be determined by comparing their results to the real-world outcomes they aim to predict. Computer simulations have become a useful tool for the mathematical modeling of many natural systems in computational physics, astrophysics, climatology, chemistry, biology and manufacturing, as well as human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

Process of building a computer model, and the interplay between experiment, simulation, and theory(Fig 2.1).



2.1.1 Computer simulation in Practical Context

Computer simulations are used in a wide variety of practical contexts, such as:

- analysis of air pollutant dispersion using atmospheric dispersion modeling
- design of complex systems such as aircraft and also logistics systems.
- design of noise barriers to effect roadway noise mitigation
- modeling of application performance
- Flight simulators to train pilots
- weather forecasting
- simulation of electrical circuits

- Power system simulation
- Simulation of other computers is emulation.

2.1.2 Computer simulation: Types

Computer models can be classified according to several independent pairs of attributes, including:

- Stochastic or deterministic
- Steady-state or dynamic
- Continuous or discrete
- Dynamic system simulation,
- Local or distributed.

2.2 Discrete Event Simulation

A **discrete-event simulation** (**DES**) models the operation of a system as a (discrete) sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation time can directly jump to the occurrence time of the next event, which is called **next-event time progression**.

A common exercise in learning how to build discrete-event simulations is to model a queue, such as customers arriving at a bank to be served by a teller. In this example, the system entities are **Customer-queue** and **Tellers**. The system events are **Customer-Arrival** and **Customer-Departure**. (The Teller-Begins-Service event can be part of the logic of the arrival and departure events.) The system states, which are changed by these events, are **Number-of-Customers-in-the-Queue** (an integer from 0 to n) and **Teller-Status** (busy or idle). The random variables that need to be characterized to model this system stochastically are **Customer-Interarrival-Time** and **Teller-Service-Time**. An agent-based framework for performance modeling of an optimistic parallel discrete event simulator is another example for a discrete event simulation

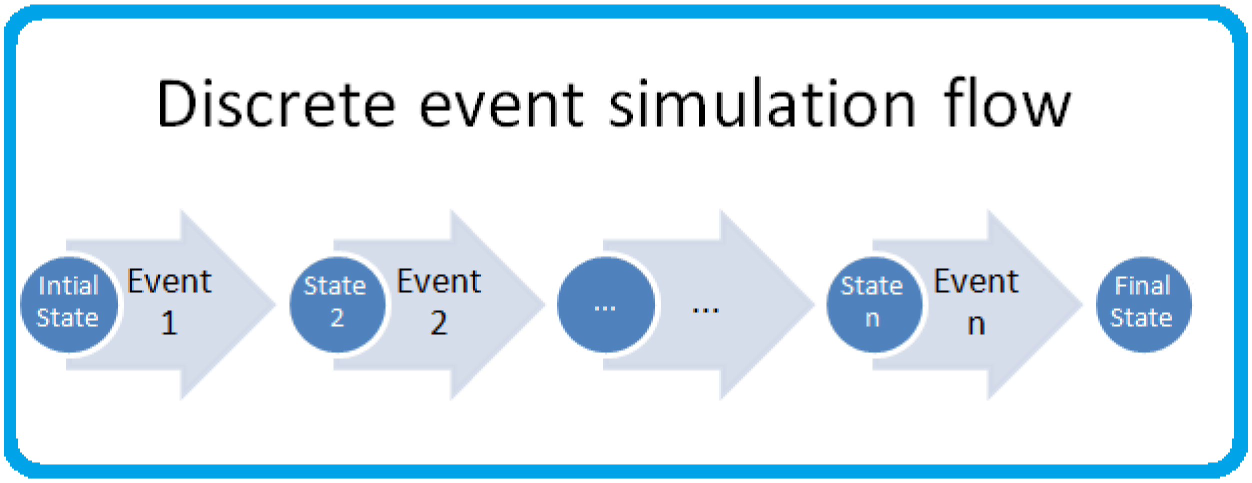


Fig 2.2 (Flow of DES)

2.2 Components of Discrete Event Simulation

In addition to the logic of what happens when system events occur, discrete event simulations include the following:

- Priority queue,
- Animation event handler, and
- Time re-normalization handler (as simulation runs, time variables lose precision. After a while all time variables should be re-normalized by subtracting the last processed event time).

2.3 OpenGL: As a tool for Simulation

Here we will be using OpenGL for performing the simulation of crowds. Using this tool we will be generating the entities and make their movement possible at certain time of interval .

OpenGL is mainly considered an API (an Application Programming Interface) that provides us with a large set of functions that we can use to manipulate graphics and images. However, OpenGL by itself is not an API, but merely a specification, developed and maintained by the Khronos Group.

<u>3.0 System Requirements and Design</u>

3.1 Basic Structure and Components

Here we will be using OpenGL for performing the simulation of crowds. Using this tool we will be generating the entities and make their movement possible at certain time of interval .

- ❖ Entity (People categorized in old , young and adult)
- ❖ Graphical representation of Roads
- ❖ There will be three roads as specified in figure.
- ❖ Movement of entities
- ❖ Threshold pressure
- ❖ Stampede detection.

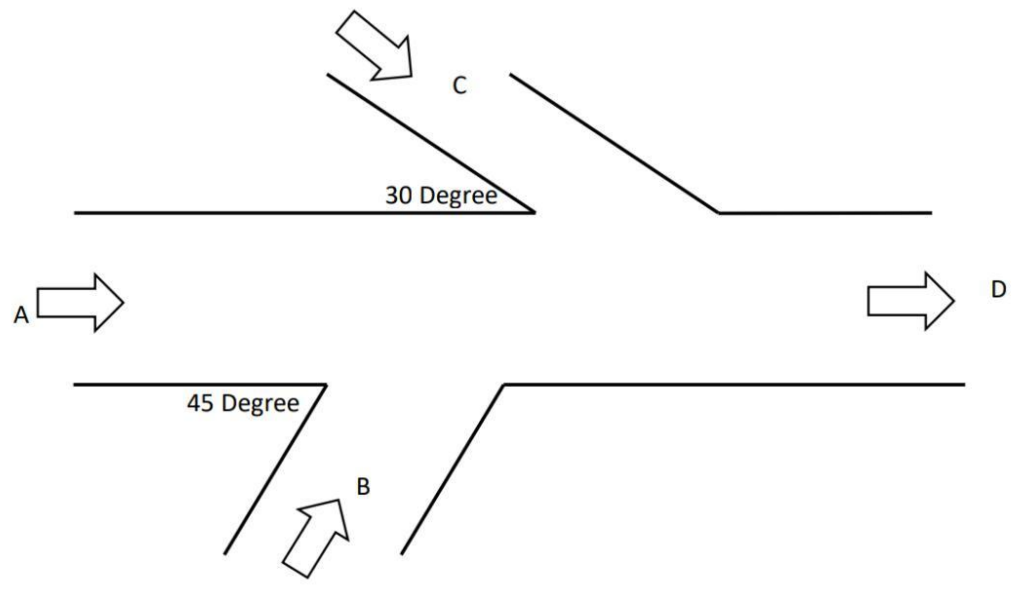


Fig 3.1 (Road Structure)

3.2 Functional Requirements

3.2.1 Brief Description of working

- We will generate entities at random time
- The generated entity may belong to any of the one category (Old , Children and Adult)
- Depending on the category they belong there will be fixed time taken by them to complete one unit of step.
- Generated entity will be sent into a priority queue.
- The priority queue will be created on the basis of the fact that who will complete it’s step first.
- Parallely on the second thread we have a queue in which those entity will be pushed who have completed their step and on the basis of this they will be displayed.
- Our road is divided into multiple segments , if number of people at any segment is greater than the threshold , then it is declared as stampede.

3.2.2 OpenGL and Implementation related

- ❖ OpenGL (Version >=3.3)
- ❖ Libraries of OpenGL
    - ➢ Glew(glew.h)
    - ➢ Thread
    - ➢ Chrono
    - ➢ Glfw3 (glfw3.h)
    - ➢ ctime
- ❖ Priority Queue Implementation
- ❖ Threading Concept

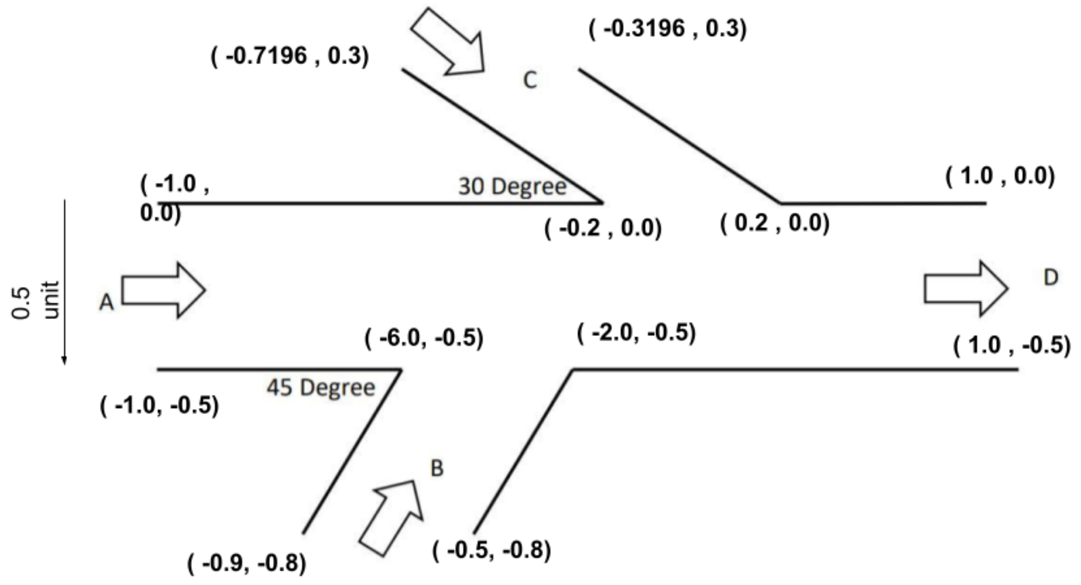
<u>4.0 Algorithm and Implementation</u>

4.1 Road Creation

4.1.1 Road Coordinates and rendering of road

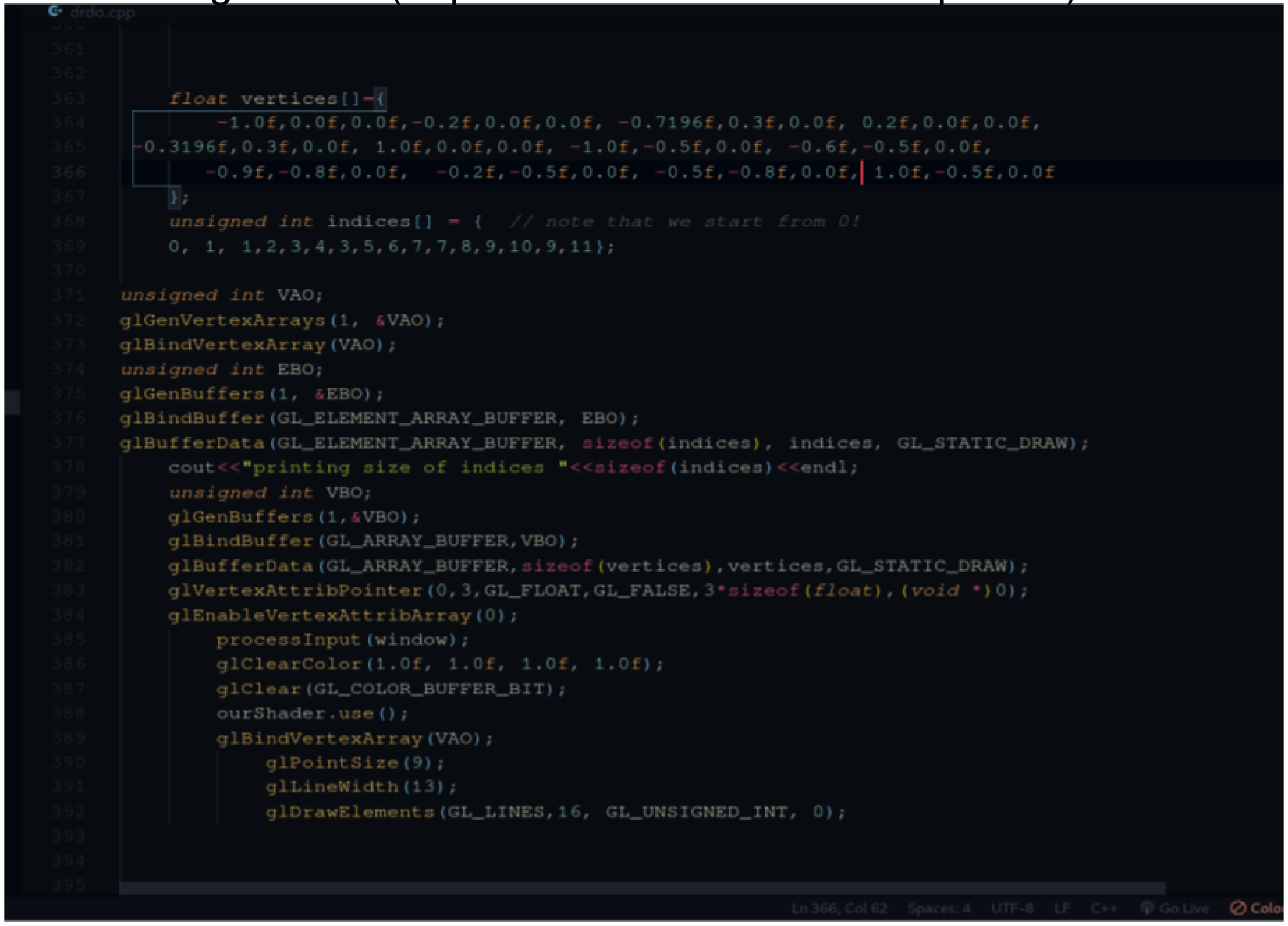
In OpenGL we use Normalized Device Coordinates. That is everything ranges between -1.0 to 1.0. So here also we will convert the road into these coordinates .

<u>FIG 4.1.1 (Road coordinates)</u>



- ❖ Now we will store all this point in a vertex buffer object and we will bind this object to GL_ARRAY_BUFFER.
- ❖ Using glDrawElements we will draw lines between these vertices.

Fig 4.1.1-a (Implementation of road via openGL)



drdo.cpp

```cpp
        float vertices[]={
            -1.0f,0.0f,0.0f,-0.2f,0.0f,0.0f, -0.7196f,0.3f,0.0f, 0.2f,0.0f,0.0f,
-0.3196f,0.3f,0.0f, 1.0f,0.0f,0.0f, -1.0f,-0.5f,0.0f, -0.6f,-0.5f,0.0f,
           -0.9f,-0.8f,0.0f,  -0.2f,-0.5f,0.0f, -0.5f,-0.8f,0.0f, 1.0f,-0.5f,0.0f
        };
        unsigned int indices[] = {  // note that we start from 0!
        0, 1, 1,2,3,4,3,5,6,7,7,8,9,10,9,11};

    unsigned int VAO;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    unsigned int EBO;
    glGenBuffers(1, &EBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
        cout<<"printing size of indices "<<sizeof(indices)<<endl;
        unsigned int VBO;
        glGenBuffers(1,&VBO);
        glBindBuffer(GL_ARRAY_BUFFER,VBO);
        glBufferData(GL_ARRAY_BUFFER,sizeof(vertices),vertices,GL_STATIC_DRAW);
        glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE,3*sizeof(float),(void *)0);
        glEnableVertexAttribArray(0);
            processInput(window);
            glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
            glClear(GL_COLOR_BUFFER_BIT);
            ourShader.use();
            glBindVertexArray(VAO);
                glPointSize(9);
                glLineWidth(13);
                glDrawElements(GL_LINES,16, GL_UNSIGNED_INT, 0);
```

Ln 366, Col 62   Spaces: 4   UTF-8   LF   C++   Go Live

Note: The road creation(rendering) is done in thread t2. That means I've used two threads(t1, t2) one for entity generation and another for rendering purposes.

4.2 Entity Generation

- I have classified the entity on the basis of the number of seconds it will take to complete one step.
- On the basis of this each entity will have an ID , time_to_complete_one_step(I’ve used step as a variable

name in the program for this) and the time it enters on the road for the very first time.

- ❖ Now I’ve three different step time (1.0 , 2.0 , 1.5)
- ❖ All those entities whose step time is 1.0 belong to the adult age group and will be assigned an id 3.
- ❖ All those entities whose step time is 1.5 belong to the children's age group and will be assigned an id 2.
- ❖ All those entities whose step time is 2.0 belong to the Old age group and will be assigned an id 1.
- ❖ Now the value of id can also be 4 or 5, these two new values denote that the entity belongs to the side road and in this case we will generate three random values 1,2 or 3 on the basis of which we will assign a value to the variable step .
- ❖ So overall an entity will mainly contain the following attributes:
    - ➢ ID
    - ➢ step(no of seconds to complete one step)
    - ➢ times (the time when the entity enters the road)
    - ➢ Coordinates
    - ➢ In the code implementation there are few more attributes which have their own significance in the logic building.
- ❖ The entities will be generated until the stampede gets detected , the entity generation is entirely random on all the three roads , even their coordinates will be generated randomly , this feature leads the implementation to a realistic implementation.

❖ Once a particular entity get generated it will be inserted in a priority queue (min heap based priority queue)

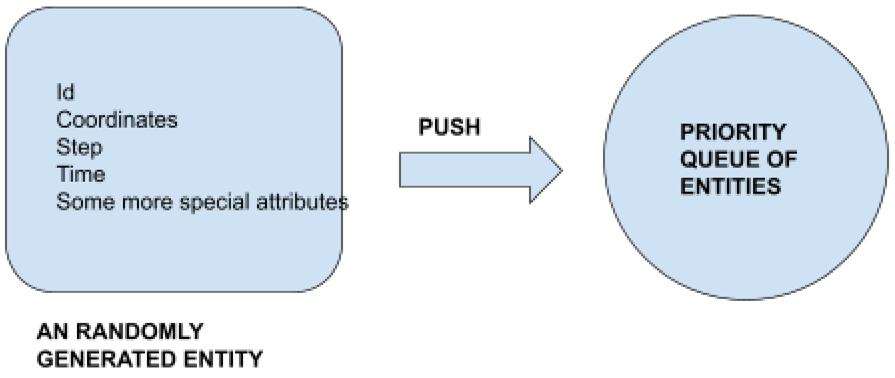
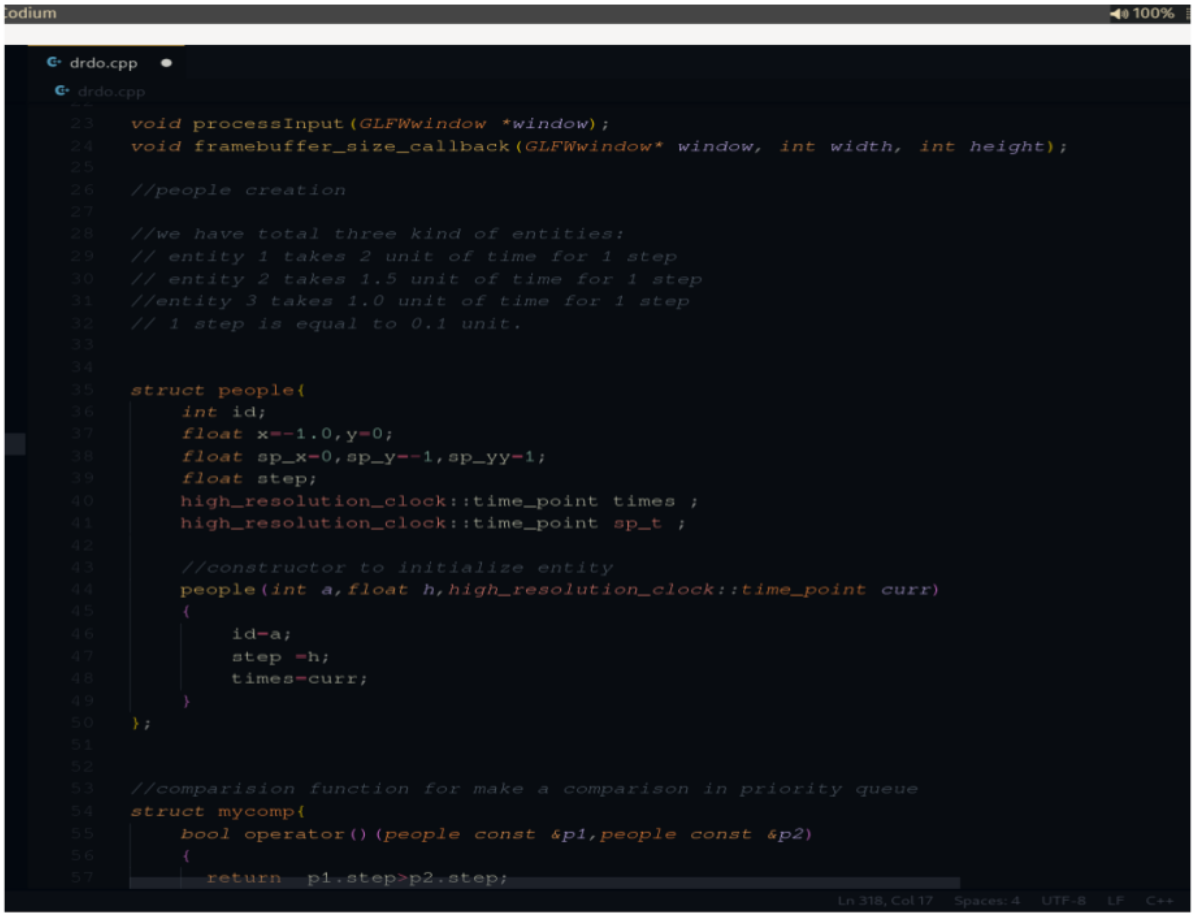
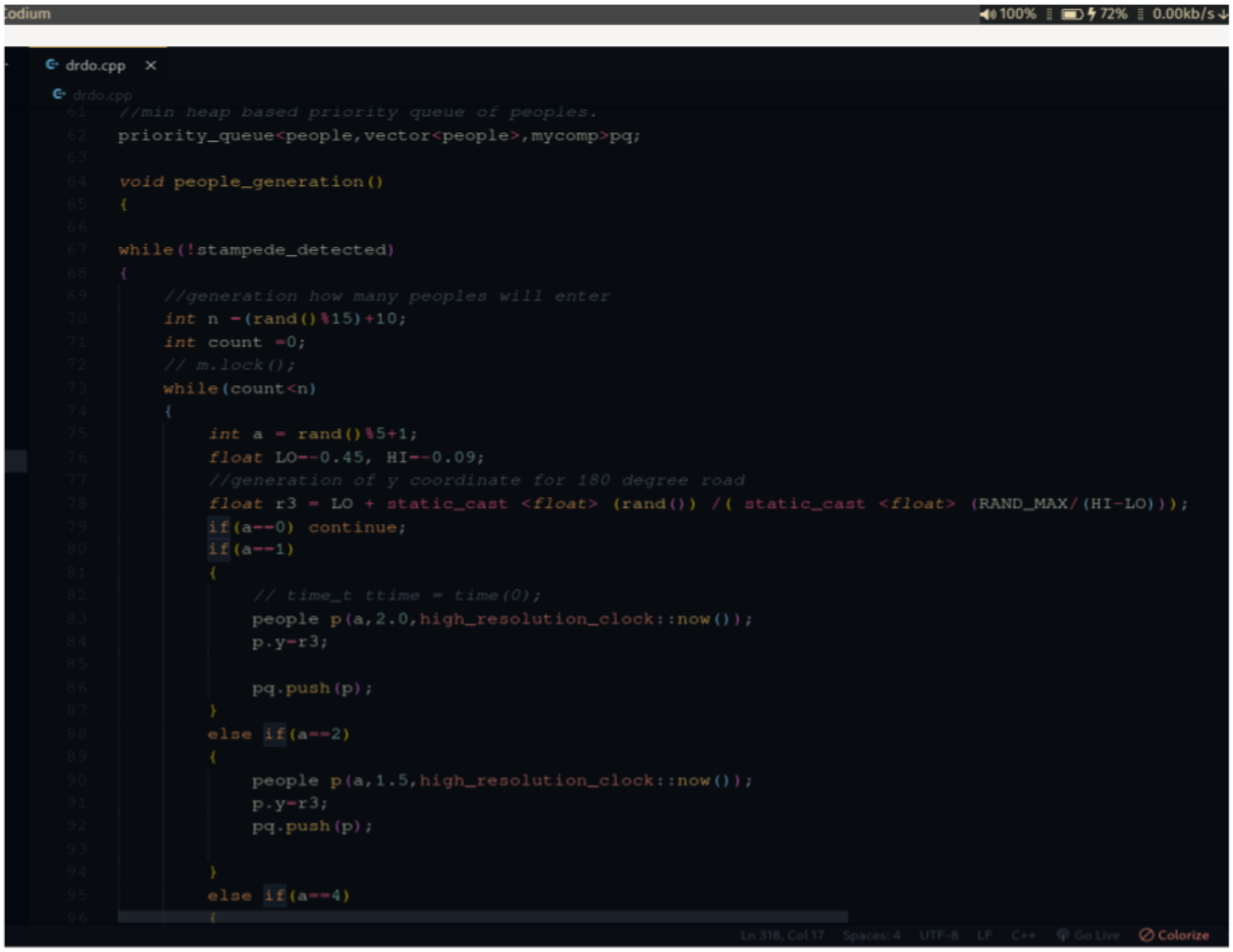

Id
Coordinates
Step
Time
Some more special attributes
PUSH
PRIORITY QUEUE OF ENTITIES
AN RANDOMLY GENERATED ENTITY

FIG 4.2 Entity generation and pushing into priority queue



```cpp
void processInput(GLFWwindow *window);
void framebuffer_size_callback(GLFWwindow* window, int width, int height);

//people creation

//we have total three kind of entities:
// entity 1 takes 2 unit of time for 1 step
// entity 2 takes 1.5 unit of time for 1 step
//entity 3 takes 1.0 unit of time for 1 step
// 1 step is equal to 0.1 unit.


struct people{
    int id;
    float x=-1.0,y=0;
    float sp_x=0,sp_y=-1,sp_yy=1;
    float step;
    high_resolution_clock::time_point times ;
    high_resolution_clock::time_point sp_t ;

    //constructor to initialize entity
    people(int a,float h,high_resolution_clock::time_point curr)
    {
        id=a;
        step =h;
        times=curr;
    }
};


//comparision function for make a comparison in priority queue
struct mycomp{
    bool operator()(people const &p1,people const &p2)
    {
      return  p1.step>p2.step;
```

FIG 4.2.1 Structure of people with constructor (Coding part).



```
//min heap based priority queue of peoples.
priority_queue<people,vector<people>,mycomp>pq;

void people_generation()
{

while(!stampede_detected)
{
    //generation how many peoples will enter
    int n =(rand()%15)+10;
    int count =0;
    // m.lock();
    while(count<n)
    {
        int a = rand()%5+1;
        float LO=-0.45, HI=-0.09;
        //generation of y coordinate for 180 degree road
        float r3 = LO + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(HI-LO)));
        if(a==0) continue;
        if(a==1)
        {
            // time_t ttime = time(0);
            people p(a,2.0,high_resolution_clock::now());
            p.y=r3;

            pq.push(p);
        }
        else if(a==2)
        {
            people p(a,1.5,high_resolution_clock::now());
            p.y=r3;
            pq.push(p);

        }
        else if(a==4)
        {
```

FIG 4.2.2 Code of people Generation.

4.3 Simulation per clock cycle

In this section I have described what happens in each clock cycle.

- ❖ I have considered the simulation period as 100ms.

- ❖ As the clock cycle begins , all those elements pop out from the priority queue on the basis of whether they have completed their step period or not.
- ❖ These popped out elements are pushed into a vector one by one, once the insertion of these elements gets done, the rendering part starts .

4.3.1 The position of the entity of straight road at time t

Let us say the entry time of a particular entity is T and the time it takes to complete one step is s. If the step length is d then at a time t the length it should travel from its initial position can be given by:

Let X =int(t-T)/s

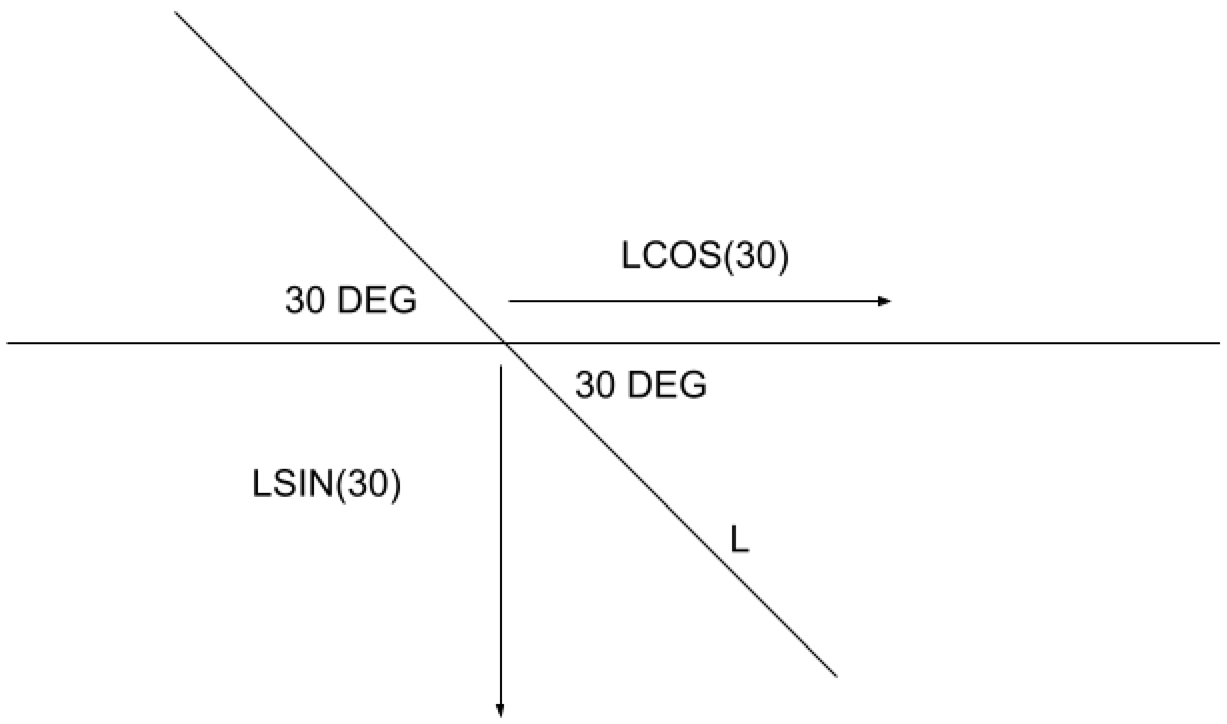
Final distance would be X*d

- ❖ Now as the entity belongs to the straight road only, it's y coordinate is not going to change.
- ❖ The x coordinate will become X*d.

4.3.2 The position of the entity of 30 deg angled road at time t

Let us say the entry time of a particular entity is T and the time it takes to complete one step is s. If the step length is d then at a time t the length it should travel from its initial position can be given by:

Fig 4.3.2 components of the road



Let X =int(t-T)/s

Final distance would be X*d

Let Z= X*d

- ❖ Now as the entity belongs to the straight road only, it's y coordinate is not going to change.
- ❖ The x coordinate will become ZCOS30.
- ❖ The y coordinate will become (-ZSIN30)

Note: We will be doing so only till the entity is on the side road,

Once the entity comes on the straight road we will start following the equation described in sec 4.3.1

4.3.3 The position of the entity of 45 deg angled road at time t

I am not describing it in detail , the concept is the same as described in the section 4.3.2 , only the angle will be changed here from 30 deg to 45 deg.

4.4 Stampede Detection

Here I have described the concept used for detection of stampedes.

4.4.1 Road division

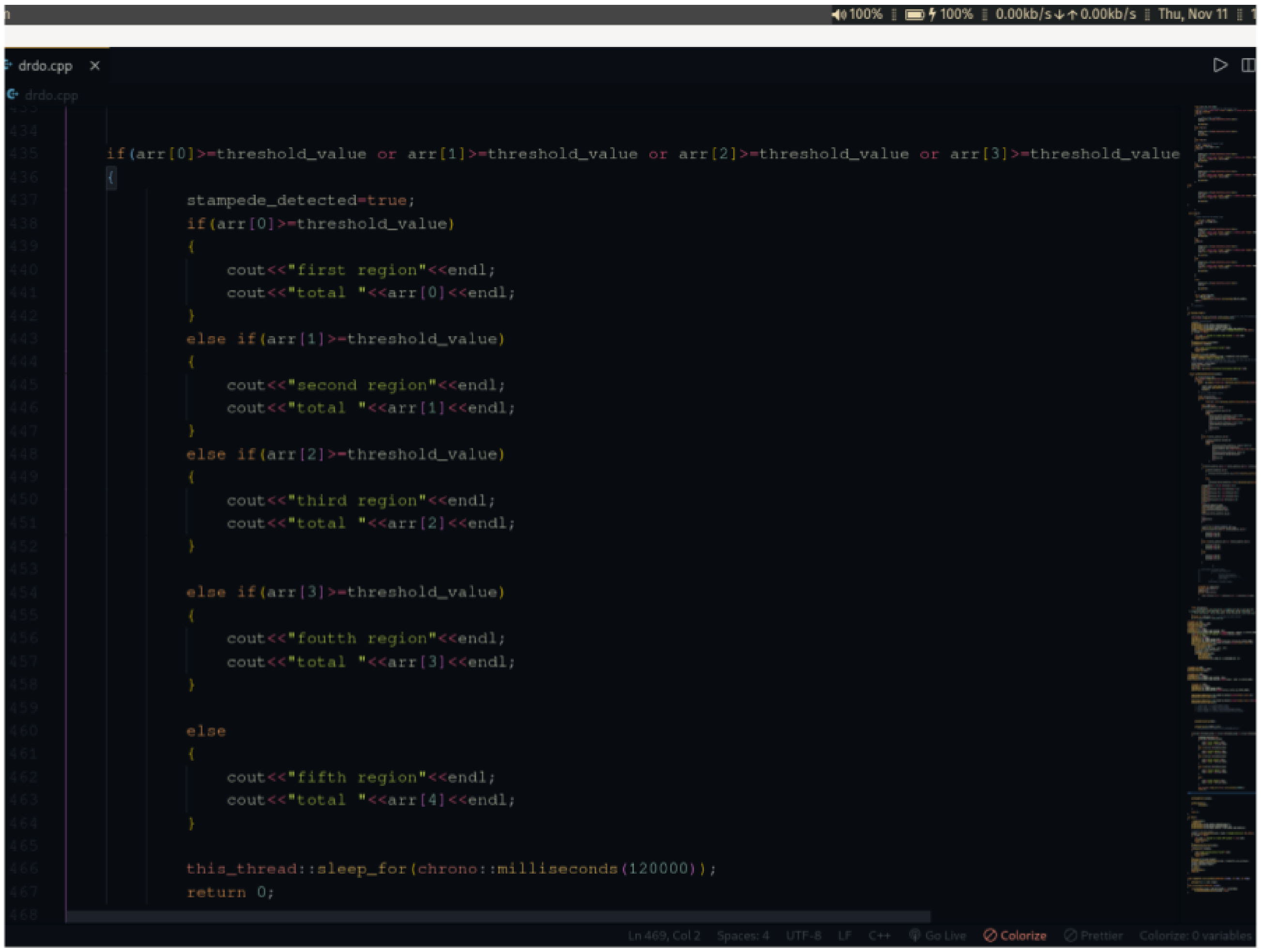
- ❖ The entire straight road is divided into 5 sections.
- ❖ As the x-coordinate of the road begins at -1.0 and ends at 1.0 , so the total length is 2.0.
- ❖ Now I have divided the into five equal section of 0.4 each , which are as following:
    - ➢ First Region [-1.0 ,-0.6)
    - ➢ Second Region [-0.6 , -0.2)
    - ➢ Third Region [-0.2, 0.2)
    - ➢ Fourth Region [0.2, 0.6)
    - ➢ Fifth Region [0.6 , 1.0]

4.4.2 Threshold Pressure

The maximum number of entities that can be there in any section of the road such that there is no stampede. I have decided its value 28.

4.4.2 Stampede declaration

If in any region the number of people becomes greater than the threshold value , we will declare that region to be stampede.



```cpp
    if(arr[0]>=threshold_value or arr[1]>=threshold_value or arr[2]>=threshold_value or arr[3]>=threshold_value
    {
            stampede_detected=true;
            if(arr[0]>=threshold_value)
            {
                cout<<"first region"<<endl;
                cout<<"total "<<arr[0]<<endl;
            }
            else if(arr[1]>=threshold_value)
            {
                cout<<"second region"<<endl;
                cout<<"total "<<arr[1]<<endl;
            }
            else if(arr[2]>=threshold_value)
            {
                cout<<"third region"<<endl;
                cout<<"total "<<arr[2]<<endl;
            }

            else if(arr[3]>=threshold_value)
            {
                cout<<"fouth region"<<endl;
                cout<<"total "<<arr[3]<<endl;
            }

            else
            {
                cout<<"fifth region"<<endl;
                cout<<"total "<<arr[4]<<endl;
            }

            this_thread::sleep_for(chrono::milliseconds(120000));
            return 0;
```

Fig 4.4.2 Code for detection of stampede

4.5 Important points to Note

4.5.1 Libraries used

In case , if anyone forgets to include these libraries , the program may fail to execute , even some of the header files are not there by default and you have to download it from the respective websites.

```cpp
#include<bits/stdc++.h>
#include<GL/glew.h>
#include <GLFW/glfw3.h>
```

```
#include<thread>
#include<chrono>
#include <ctime>
#include <ratio>
#include "Headers/Shader.h"
```

4.5.1 Execution command on linux

Compilation--> g++ drdo.cpp -lpthread -lGL -lGLEW -lglfw -o drd.out

To execute→ ./drd

5.0 Results and Conclusion

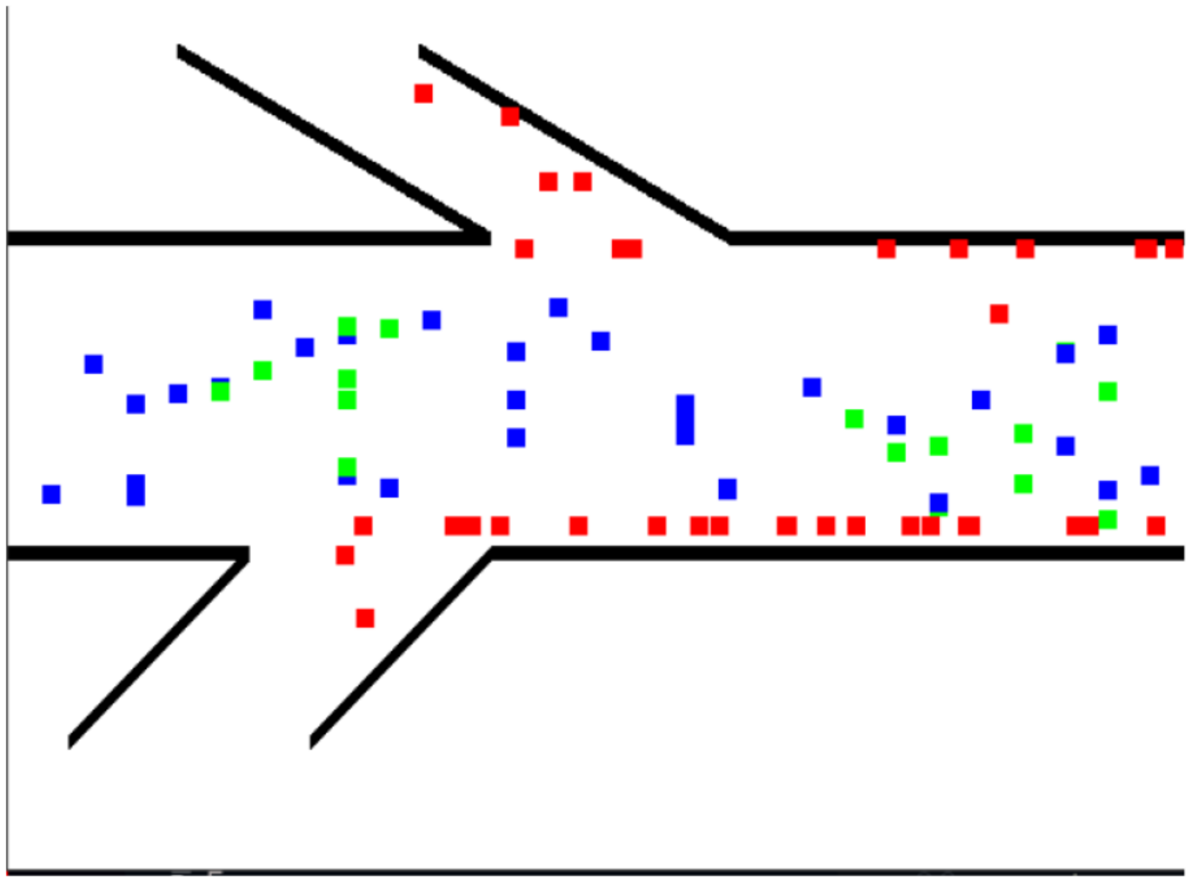
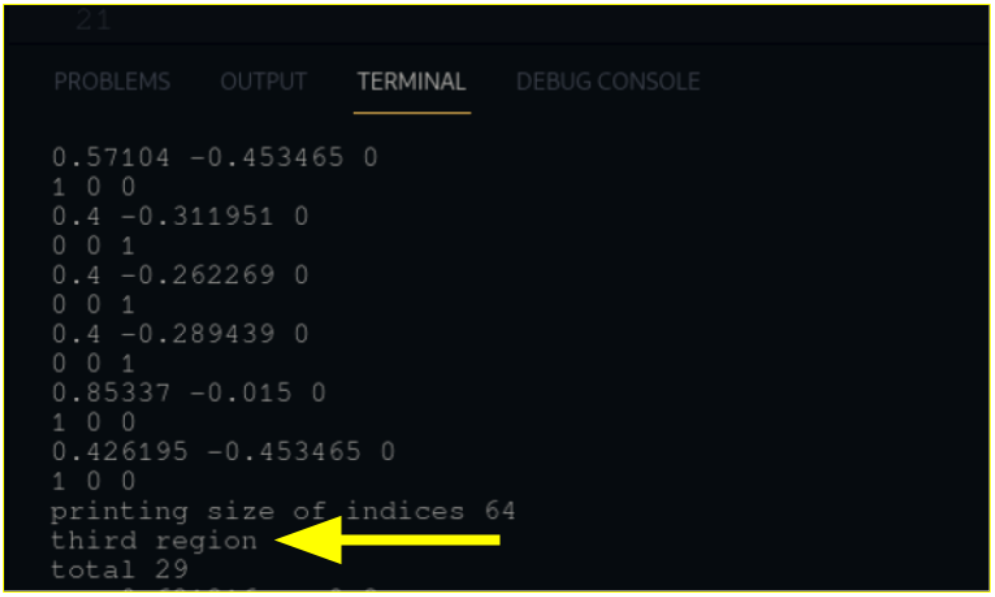


Fig 4.0 Simulation image

We have discussed a few cases where stampedes caused life loss to people.This application works fine for which it was made and can be improved to perform well in other scenarios also.

Fig 4.0-a Conclusion from the simulation



```
0.57104 -0.453465 0
1 0 0
0.4 -0.311951 0
0 0 1
0.4 -0.262269 0
0 0 1
0.4 -0.289439 0
0 0 1
0.85337 -0.015 0
1 0 0
0.426195 -0.453465 0
1 0 0
printing size of indices 64
third region
total 29
```

The fig 4.0-a depicts that the stampede was in the third region where the number of peoples were greater than the threshold value which was set to be 28.

6.0 References

- ❖ Wikipedia – Discrete event Simulation
    - ➢ https://en.wikipedia.org/wiki/Discrete-event_simulation
- ❖ OpenGL
    - ➢ https://learnopengl.com/
- ❖ Priority Queue
    - ➢ https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/
- ❖ Multiprocessing/Multithreading
    - ➢ https://www.geeksforgeeks.org/multithreading-in-cpp/?ref=rp