Ty Roberts
04/28/2021

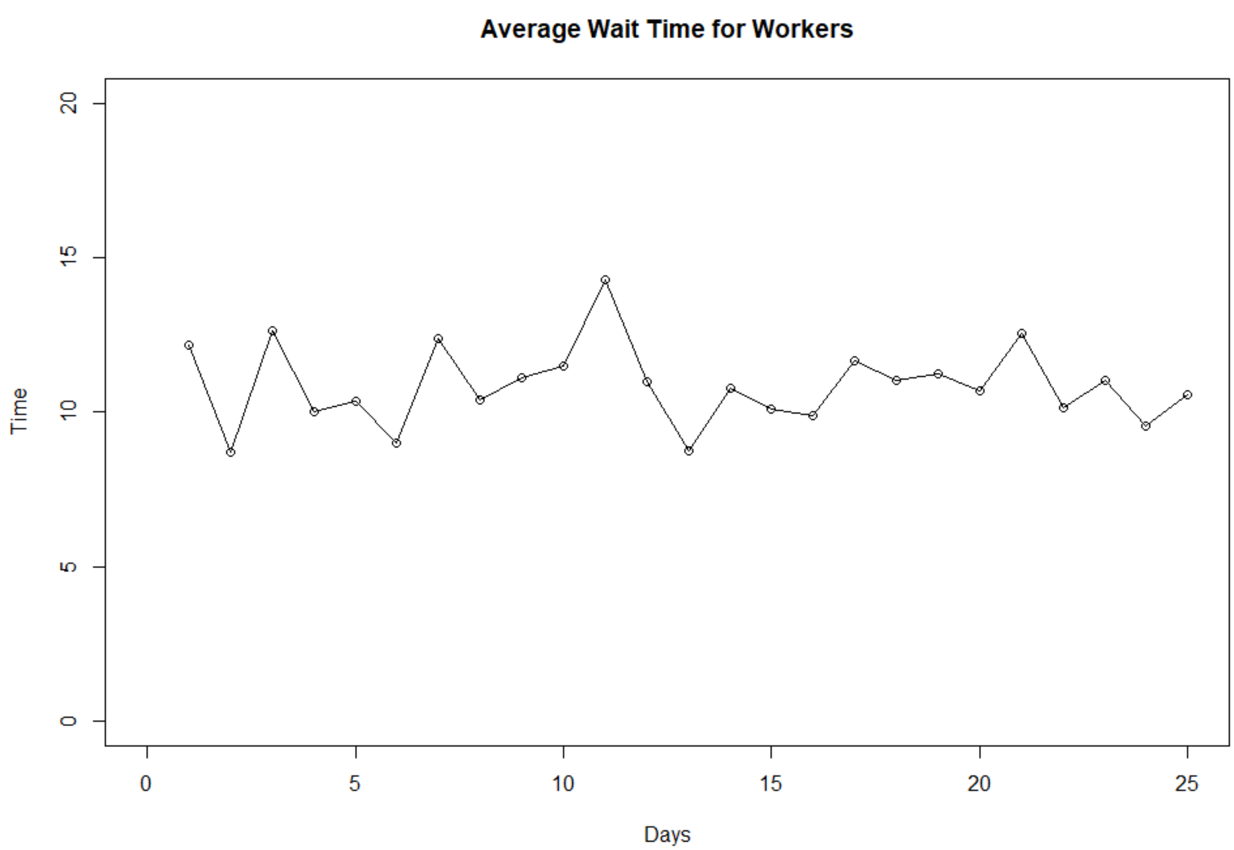
Final Project

Results:

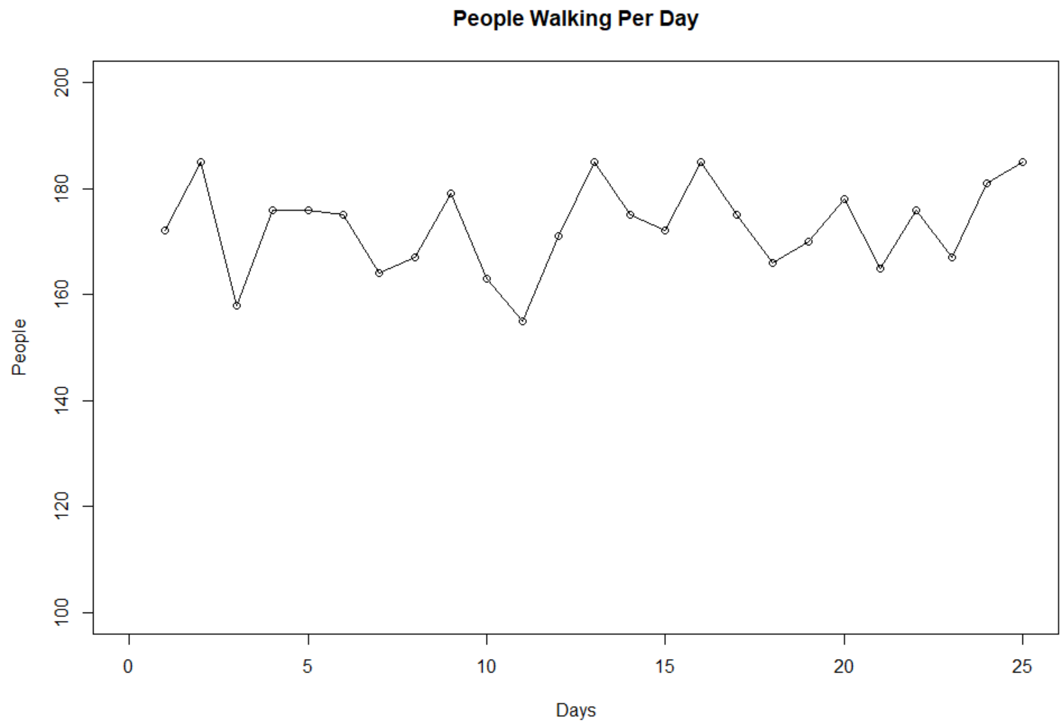
a) The following is the average for all the metrics over a 25 day period:

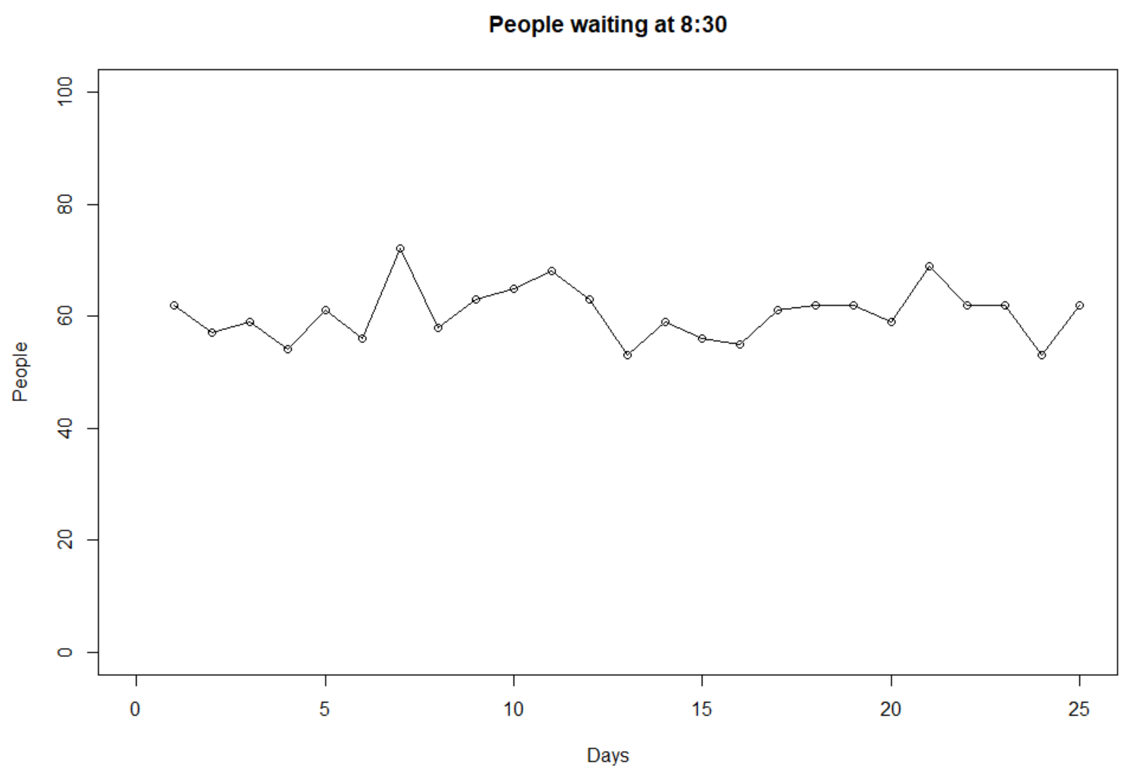
```
Total wait time average: 10.5564
Average people who walk per day: 174
Average people waiting at 8:30: 59
Average people waiting at 8:45: 70
Average people waiting at 9:00: 74
Average time last person leaves: 9:16
--------------------------------
Process exited after 0.3893 seconds with return value 0
Press any key to continue . . .
```

The following are graphical representations for each day:

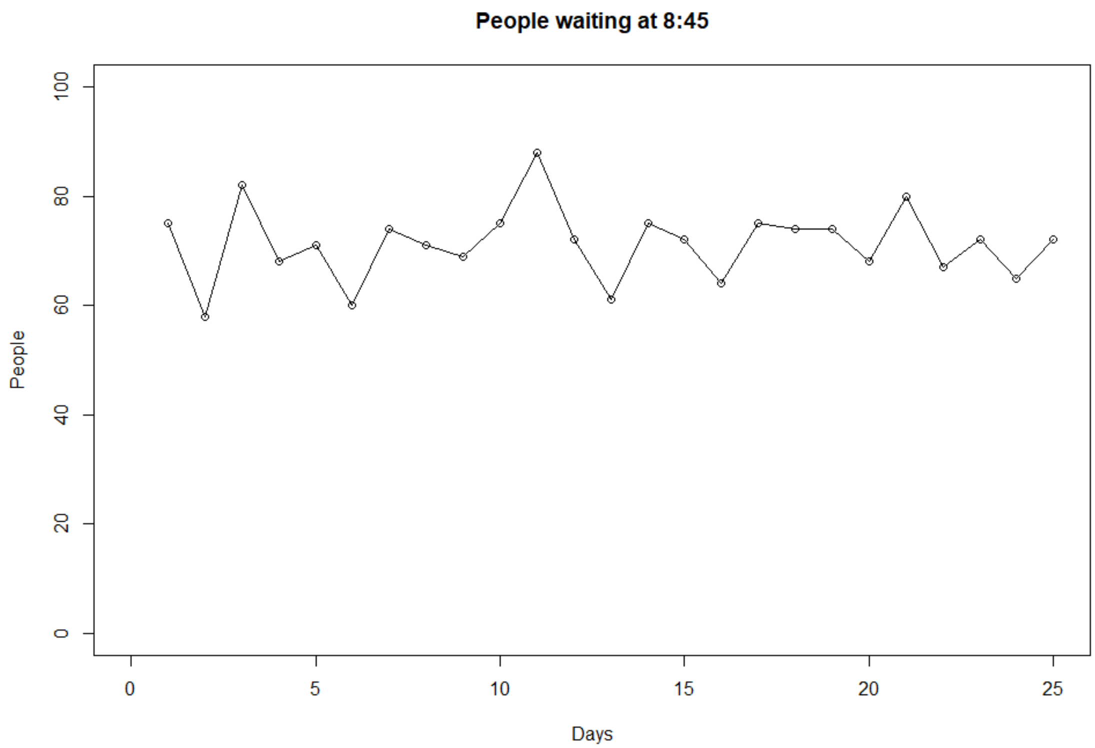


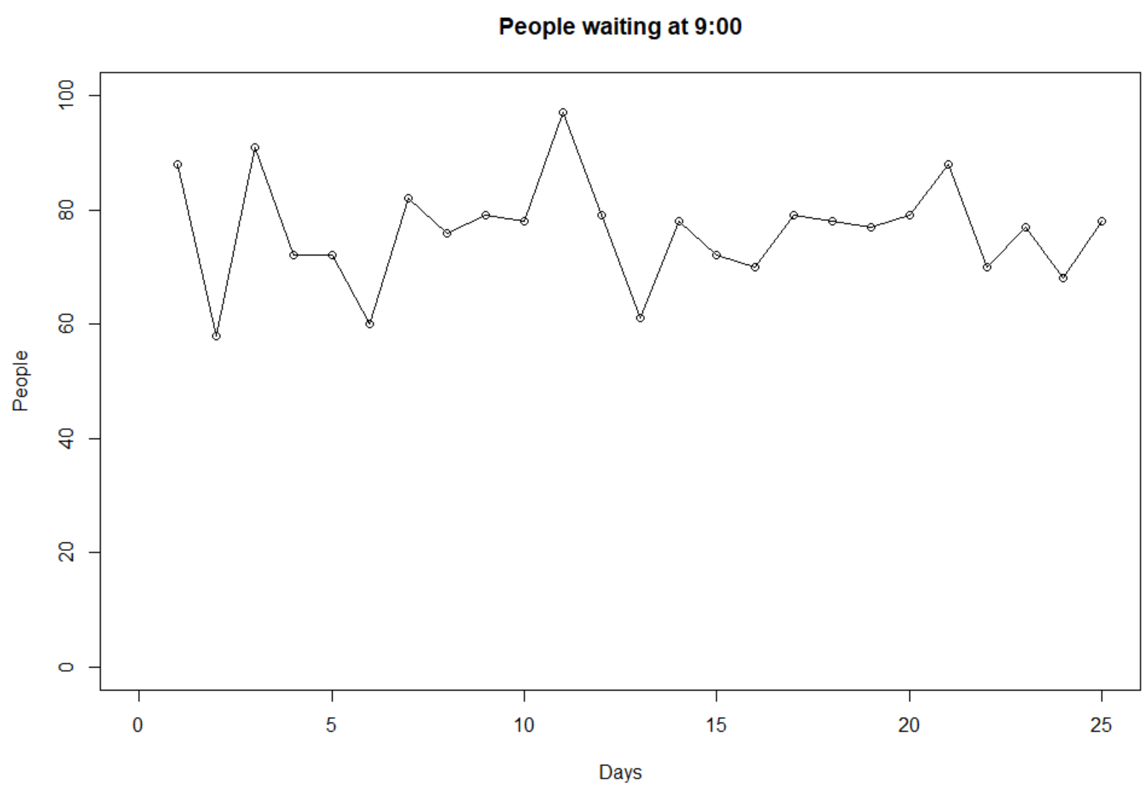
Ty Roberts
04/28/2021



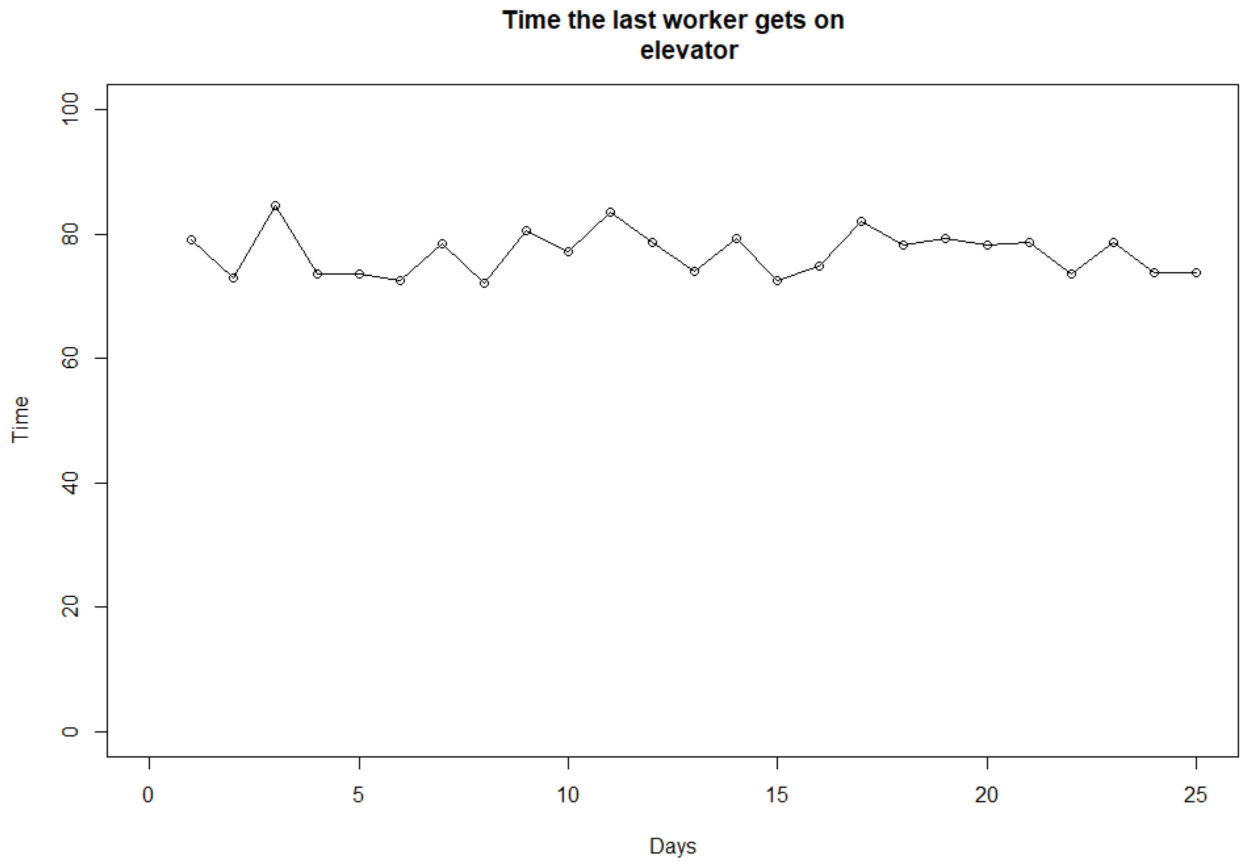


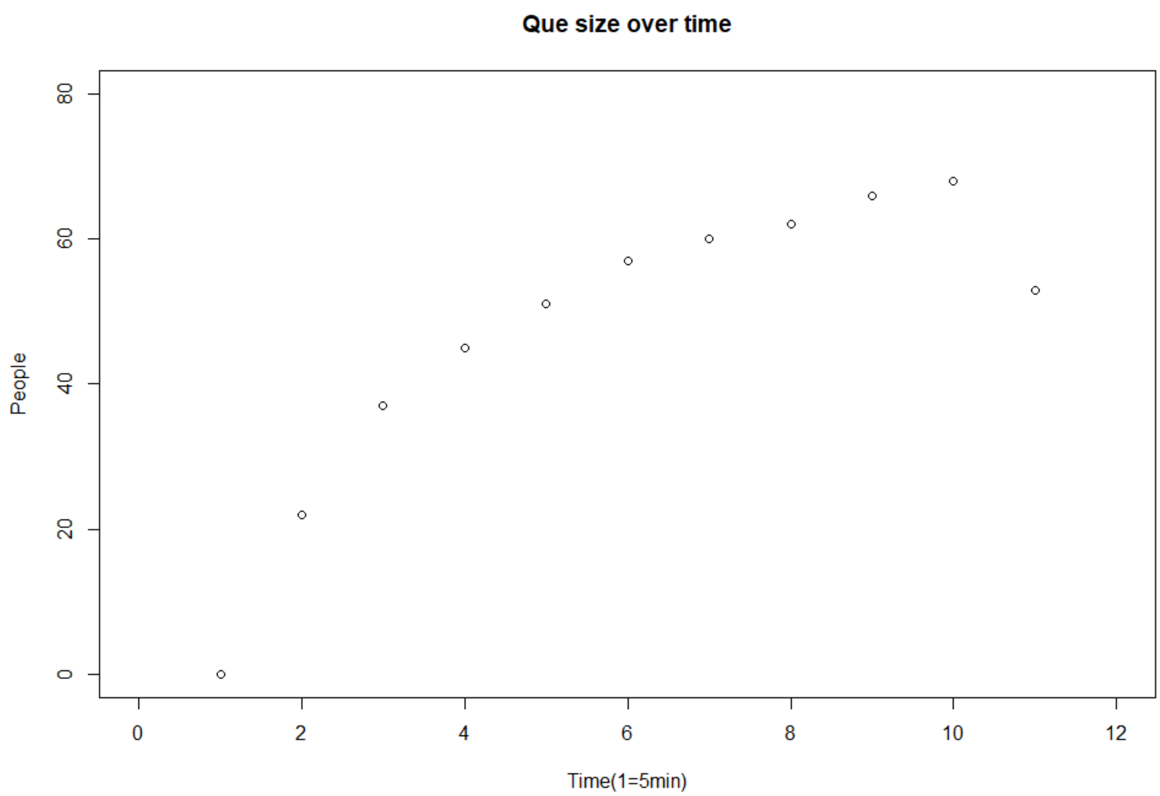
Ty Roberts
04/28/2021





Ty Roberts
04/28/2021





Ty Roberts
04/28/2021

b) From the perspective of a worker, they should expect to wait around 10 minutes if they want to take the elevator. However, the best way to avoid the wait is to get there early as can be seen from the average que size at 8:30 compared to 8:45 and 9:00. The growth between 8:30 and 8:45 is higher than between 8:45 and 9:00. I created the que over time graph to show this logarithmic que size growth over the span of a day. I would say that a worker should be ready to walk if they arrive after 8:30. Also, if the worker has to be in his office by nine, they definitely should walk if they are on the second floor and all workers that are on the 4th floor should get there early.

c) From the perspective of the building owner, with only one elevator, the first thing to notice is how quickly the wait line increases at the start of the hour. If I were the building owner, I would put a health initiative to try to get as many people to walk to walk up the stairs as possible because when we compare the graph of wait times in a day and the number of people who walk in a day. On day 11 of the wait time per day, there is a big spike and that same day coincides with a big dip in the number of people who walk on day 11. One other thing the owner could do is make it mandatory that people who work on the second floor have to walk and this would help in two ways. First the que size would be cut by about a 3rd but also it would make the elevator operate faster because it doesn't have to stop on the second floor.

Implementation:

a) For this project, I used two different programs. I used C++ for the actually coding aspect and I transferred the information to an excel file so it could be ran on RStudio. The reason I used C++ for the programming is because it is the language I am most comfortable with and I can create classes, however, it can be more difficult to create graphs. That is why I transferred the results to an excel file and used RStudio for the graphs since it is program made to easily create visual representations of data.

b) I created a class for workers so I could store the wait time and the floor that worker is going to and I stored the workers into a vector because vectors are dynamic memory structures that make it easy to add new workers and take old ones off.

c) I ran the simulation 25 times and I came up this as a rough estimation. When I was building the program, I noticed I was getting around 10 minute wait time averages so I estimated by output to be 10 minutes and I gave myself an error of + or – 2 minutes. I then used the equation: (5*1.96)^2/2^2 which represents a 95% confidence interval and got 24.01 which I rounded up to 25.

d) Some notes on the code:

- Every worker is assigned a uniform random number that represents the floor that they are going to and this effects who has the higher chance of walking and how long elevator takes
- My starting assumption was that the elevator at ground floor stays open for 0.5 minutes because that's how long it stays open at every floor; therefore, I started my simulation assuming that the first 3 people enter the elevator and that the fourth person to arrive gets there at exactly 0.1667 minutes after the elevator leaves. This sets up the basis for how I add wait time to new arrivals into the queue.
- To find the floors the elevator is going to, I set up 3 booleans to check which floor the elevator goes to. The elevator time is calculated through getting the time from the 2D array created from the table given so I add the time for each floor as well as the 0.5 minutes for each floor it stops at as well as another 0.5 minutes for the opening on the ground floor so people can get on.
- After the elevator time is calculated, I check to see which people chose to walk, so no new arrivals choose to walk, only workers who have waited in the que for at least one cycle. For any worker that walks, the wait time is taken and it is removed from the vector using the erase function and the correct index number
- Next I calculate the number of people who arrive as the elevator is moving and I do that that by taking the total elevator time and dividing by 0.1667 and taking the floor, however, I don't discard the extra time. I hold on to that extra time and add it to the following cycle because add waiting time to new arrivals at exactly 0.1667 minutes from the time the elevator leaves which is obviously not possible because that would mean every new cycle someone would have to arrive at exactly 0.1667 minutes from when the elevator left so the time from when the elevator actually leaves to when it is 0.1667 minutes is added to each wait time.
- Finally, I go through the first 12 elements of the vector and push it on to the elevator vector and remove the first 12 elements from the queue.

.

Ty Roberts
04/28/2021

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<fstream>
#include<queue>
#include<vector>

using namespace std;

class worker{
    private:
        float floor;
        float wt;

    public:
        worker(){
            floor = 0;
            wt = 0;
        }
        float setFloor(float x)
        {   floor = x;  }
        float getFloor()
        {   return floor;   }
        float setWt(float k)
        {   wt = k; }
        float getWt()
        {   return wt;  }
};

int main(){

    float elevArr[4][4] = {0, 1.0, 1.5, 1.75, // 2d array of elevator time
                           1.0, 0, 0.5, 0.75,
                           1.5, 0.5, 0, 0.5,
                           1.75, 1.5, 1.0, 0};

    float elevOpen = 0.5;
    float queTime;
    float interArrival = 0.1667;
    int people;
    worker suit;
    vector<worker> que;
    vector<worker> elevator;
    vector<float> waitTime;
    vector<float> totAvgWait;
    vector<int> pplWalk;
    vector<int> walkers;
    vector<int> halfWork;
    vector<int> threeWork;
    vector<int> fullWork;
```

```
    vector<float> lastWork;

    for (int i = 0; i < 25; i++){
        waitTime.clear();
        int ppw = 0;
        bool half = true;
        bool three = true;
        bool full = true;
        float totTime = 0.0;
        float exTime = 0;

        srand(i+5); // random seed for floor number
        float stFloor;
        for (int i = 0; i < 3; i++){ // start simulation with three people in elevator
            stFloor = (float) rand()/RAND_MAX;
            suit.setFloor(stFloor);
            elevator.push_back(suit);
        }

        while(totTime <= 60 || !que.empty()){
            bool second = false;
            bool third = false;
            bool fourth = false;
            float elevTime = 0.0;
            for (int i = 0; i < elevator.size(); i++){
                if(elevator[i].getFloor() <= 0.33)
                    second = true;
                else if (elevator[i].getFloor() > 0.33 && elevator[i].getFloor() <= 0.66)
                    third = true;
                else
                    fourth = true;
            }
            if (second && third && fourth) // go through each possible combination and get elevator time
                elevTime = elevArr[0][1] + elevArr[1][2] + elevArr[2][3] + (3 * elevOpen) + elevArr[3][0] + elevOpen;
            if (second && !third && !fourth)
                elevTime = elevArr[0][1] + elevOpen + elevArr[1][0] + elevOpen;
            if (third && !second && !fourth)
                elevTime = elevArr[0][2] + elevOpen + elevArr[2][0] + elevOpen;
            if (fourth && !second && !third)
                elevTime = elevArr[0][3] + elevOpen + elevArr[3][0] + elevOpen;
            if (second && fourth && !third)
                elevTime = elevArr[0][1] + elevArr[1][3] + (2 * elevOpen) + elevArr[3][0] + elevOpen;
            if (second && third && !fourth)
                elevTime = elevArr[0][1] + elevArr[1][2] + (2 * elevOpen) + elevArr[2][0] + elevOpen;
            if (third && fourth && !second)
                elevTime = elevArr[0][2] + elevArr[2][3] + (2 * elevOpen) + elevArr[3][0] + elevOpen;
            totTime = totTime + elevTime;
            if (totTime >= 30 && half){
                halfWork.push_back(que.size());
```

Ty Roberts
04/28/2021

```
            half = false;
        }
        if (totTime >= 45 && three){ // chech que size at specific times
            threeWork.push_back(que.size());
            three = false;
        }
        if (totTime >= 60 && full){
            fullWork.push_back(que.size());
            full = false;
        }
        elevator.clear();

        float walk;
        int walkcount = 0;
        if (que.size() > 12){ // check to see who walks while elevator moves
            srand(i+7);
            for (int i = 0; i < que.size(); i++){
                walk = (float) rand()/RAND_MAX;
                if (que[i].getFloor() <= 0.33 && walk <= 0.5){
                    waitTime.push_back(que[i].getWt());
                    que.erase(que.begin()+i);
                    walkcount++;
                }
                if (que[i].getFloor() > 0.33 && que[i].getFloor() <= 0.66 && walk <= 0.33){
                    waitTime.push_back(que[i].getWt());
                    que.erase(que.begin()+i);
                    walkcount++;
                }
                if (que[i].getFloor() > 0.66 && walk <= 0.1){
                    waitTime.push_back(que[i].getWt());
                    que.erase(que.begin()+i); // erase walker from que
                    walkcount++;
                }
            }
        }
        ppw = ppw + walkcount; // add to total people who walk

        people = elevTime/interArrival; // calculate new people that join que and waiting times
        worker wait;
        if (que.empty()){
            srand((unsigned)time(NULL));
            for (int i = 1; i <= people; i++){
                stFloor = (float) rand()/RAND_MAX;
                wait.setFloor(stFloor);
                wait.setWt(elevTime - (interArrival * i));
                que.push_back(wait);
            }
        }
        else{
            for (int i = 0; i < que.size(); i++){
                que[i].setWt(que[i].getWt() + elevTime);
```

```
                }
                if (totTime <= 60){
                    srand((unsigned)time(NULL));
                    for (int i = 1; i <= people; i++){
                        stFloor = (float) rand()/RAND_MAX;
                        wait.setFloor(stFloor);
                        wait.setWt(elevTime - (interArrival * i) + exTime); // time is added b/c worker
                        que.push_back(wait);                                // wait time interval
                    }
                }
            }
            exTime = elevTime - (interArrival * people);

            int index = 0;
            if (que.size() <= 12){
                while (index < que.size()){
                    waitTime.push_back(que[index].getWt());
                    index++;
                }
            }
            else{
                while (index < 12){
                    waitTime.push_back(que[index].getWt());
                    elevator.push_back(que[index]);
                    index++;
                }
            }

            que.erase(que.begin(), que.begin()+index);
        }
        float sum = 0;
        for (int i = 0; i < waitTime.size(); i++)
            sum = sum + waitTime[i];

        totAvgWait.push_back((float) sum/waitTime.size());
        pplWalk.push_back(ppw);
        lastWork.push_back(totTime);
    }

    ofstream myFile; // write output to csv file for Rstudio
    myFile.open("elevator_sim_total2.csv");
    myFile << "Wait,Walk,8:30,8:45,9:00,Last";
    myFile << endl;
    for (int j = 0; j < 25; j++){
        myFile << totAvgWait[j] << "," << pplWalk[j] << "," << halfWork[j] << ","
               << threeWork[j] << "," << fullWork[j] << "," << lastWork[j];
        myFile << endl;
    }
    myFile.close();

    float waitSum = 0;
    for (int i = 0; i < totAvgWait.size(); i++){
```

```
        waitSum = waitSum + totAvgWait[i];
    }
    int pplsum = 0;
    for (int i = 0; i < pplWalk.size(); i++)
        pplsum = pplsum + pplWalk[i];

    int halfsum = 0;
    for (int i = 0; i < halfWork.size(); i++)
        halfsum = halfsum + halfWork[i];

    int threesum = 0;
    for (int i = 0; i < threeWork.size(); i++)
        threesum = threesum + threeWork[i];

    int fullsum = 0;
    for (int i = 0; i < fullWork.size(); i++)
        fullsum = fullsum + fullWork[i];

    float lastsum = 0;
    for (int i = 0; i < lastWork.size(); i++)
        lastsum = lastsum + lastWork[i];
    float avgLastSum = lastsum/lastWork.size();
    float minTime = avgLastSum - 60;
    int realtime = (int) (minTime + 0.5);
    cout << "Total wait time average: " << (float) waitSum/totAvgWait.size() << endl;
    cout << "Average people who walk per day: " << (int) pplsum/pplWalk.size() << endl;
    cout << "Average people waiting at 8:30: " << (int) halfsum/halfWork.size() << endl;
    cout << "Average people waiting at 8:45: " << (int) threesum/threeWork.size() << endl;
    cout << "Average people waiting at 9:00: " << (int) fullsum/fullWork.size() << endl;
    cout << "Average time last person leaves: 9:" << realtime;

    return 0;
}
```

Critical Thinking:

a) The most difficult part was the conceptual idea of how to add the proper waiting time to new arrivals into the queue. I had originally started measuring for wait time as soon as the elevator left which would have started the index at 0 but it stop the at one less from the total number of people who would join the que and I wasn't getting the right extra time amount. I switch the index to starting at 1 and went to the exact number of people who were added in the queue.

b) One thing I would change is fixing my que graph. I had created the others and decided to create this one last minute to show the logarithmic nature of how the queue fills up. Also one thing I could have done was write the program in a way to where new arrivals could choose to walk immediately instead of just having workers who were already in the queue choose the walk.

c) One thing I learned from this project was how to take different elements and sort them in a way that I draw the correct information. For example, how to look at the elevator as essentially a server in queuing theory, it just takes a certain amount of time for it do its job and during that time, more people are adding to the queue and people wait for that much time. Also the added complexity of people getting to choose to walk instead adds a more realistic aspect to the simulation. The ability to take real world situation and break them down into a way it they could be displayed in a computer is essentially what simulation and modeling is all about. The skill of taking complex scenarios and sifting through the details you don’t need and extracting just the bare essentials is very valuable in all of computer science.