# Modeling Crowd Dynamics

# Final Report

Math 485 - Mathematical Modeling

Dr. Karl Glasner

Mitch Wilson, Brent Morgan, Andy Platta, Krista Parry

# Table of Contents

**Acknowledgments**

This project was mentored by Jorge Ramirez, whose help is acknowledged with great appreciation. Support from a University of Arizona TRIF (Technology Research Initiative Fund) grant to J. Lega is also gratefully acknowledged.

**Abstract**

In this paper we investigate the simulation of crowds in the event of an evacuation. By examining papers from Henin et al., Kirchner et al., and others, we look at ways to replicate realistic behaviors. We first implement our own model based on electric field particles to verify that we have a working program structure. We next work on implementing the Kirchner and Swarm Force models in aims of producing similar results. Results from our simulations show how the number of people injured changes while varying parameters, as well as the change in how many people successfully exit the room while varying parameters.

**Introduction**

The topic of crowd dynamics encompasses a broad variety of situations, ranging from traffic control, emergency escape routes, or simple pedestrian flow. In particular, we will study the case of a finite crowd in a confined room, and the movement of the crowd through a single exit. While this situation can have elements of random movement, we will assume that each individual's main goal is to leave the room through the single exit. Some parameters we will consider include pushing and shoving of individuals, the proximity of individuals to one another with relation to the exit, and the proximity of individuals to walls or other obstacles in the room.

The study of crowd dynamics must take into consideration the goals of individuals as a whole, and the movement of the individual with respect to the crowd as well. We would like to design a model that closely simulates the movement of a crowd out of a single exit in a real life scenario such as students exiting a classroom after lecture is dismissed. This model can also be altered to simulate the movement out of a crowd in an emergency situation, although some parameters such as crowding and shoving will play a more significant role in this situation.

In the case of the students exiting the classroom, we will assume that each individual would prefer to maintain a degree of personal space, although this may vary as the individual becomes closer to exiting the room. In cases of emergency evacuations, the crowd is more likely to dismiss the need for personal space, and move more urgently towards the exit.

**Background Information**

To simulate the situation of crowd dynamics in an emergency situation, we researched methods of mathematical modeling that would efficiently and realistically simulate each individual's behavior as a part of the crowd. We chose to implement the idea of cellular automation for modeling purposes. For these types of computations, the system consists of a regular grid of cells, and every cell has the same rule for updating. Each time these rules are applied a new generation of the grid is created. In the 1980's, Stephen Wolfram made the proposal that, rather than understanding nature by equations, one should instead build models that are based directly on simple computer programs. An area where such approaches may work well is in the study of evolving natural patterns, in particular, coarsening phenomena where the pattern size grows.

In this work, the impacts of various rules, which can be used to simulate crowd dynamics, are investigated. The key component is the movement of particles, or "people," over a lattice controlled by rules that govern the movement of the people and their interactions.

**Previously Designed Models**

There are two different models whose rules and equations we hope to implement into our simulation in order to make it more realistic. These two models are the Kirchner Field Model and the Swarm Force model. Both of these models contain parameters and equations that simulate real life movement of people in a crowd as they attempt to evacuate a room.

In the Kirchner Field Model, a grid represents a location where the agents can stand. Each cell can only be occupied by at most one agent at a time. Cell adjacency is defined in the model by the four Cartesian neighbors of a cell, although the model can also be adjusted to include all eight of the neighboring cells in cell adjacency.

In this model, information is available to agents by means of elementary particles called bosons. These bosons provided the agents with information about their surroundings and the agents use this information to help them determine where to move during each individual time step. Bosons are placed onto cells whenever an agent moves from a cell, and other agents are attracted to cells that have a high concentration of bosons.

There are two fields that are used in the Kirchner field model that help to direct agents' movement throughout the grid. The first field present in the model is a static field. The static field contains a gradient that has high values near desirable areas and low values in less desirable areas. This field does not change during the course of the simulation. The value of the static field at a position (i,j) relative to a door at position (a,b) is described by the following two equations: $S_{ij} = [(a-i)^2 + (b-j)^2]^{1/2}$ and $S_{ij} = \max_{ij}(S_{ij}) - S_{ij}$. If multiple doors are present in the grid, the static field equation is calculated only for the door that is closest to the agent (Henin & White pgs. 697-698).

The other field present in this model, the dynamic field, changes throughout the course of the simulation. This field counts the bosons particles that are dropped by agents onto cells. Whenever an agent moves from one cell to another, the value on the dynamic field that it left is updated by: $D_{ij} \rightarrow D_{ij} + 1$. The boson particles decay with probability δ in each time step and those that do not decay diffuse to a random neighboring cell with probability α (Henin & White pg. 698).

Agents navigate through the grid by determining a score associated with its current cell and all neighboring cells. Based on these scores, the agents select a cell to occupy during the next time step. Movement is simultaneous in the model and if an agent finds that the cell that it wishes to occupy is occupied, the agent must remain stationary for the current time step. The score for each neighboring cell, which is zero for occupied and wall cells, is a mathematical combination of the field values present on that cell and is determined by the following equation:

$$p_{ij} = N \exp\left(k_S S_{ij}\right) \exp\left(k_D D_{ij}\right) \left(1 - n_{ij}\right) \zeta_{ij}$$ (Henin & White pg. 698).

The parameters $k_D$ and $k_S$ represent the agents' consideration of the dynamic and static fields respectively. When one of these numbers is high the agent is strongly influenced by that particular field and when one of these numbers is low the agent is influenced little by that particular field. $p_{ij}$ represents the probability that an agent will select a neighboring cell, or its own, with coordinates (i,j) while $D_{ij}$ and $S_{ij}$ represent the value of the dynamic and static fields respectively in this cell. $n_{ij}$ is an occupation number which is zero if a cell is occupied and one otherwise. $\zeta_{ij}$ is another occupation number that is zero for walls and occupied cells and one otherwise. N is a normalization number that is equal to $(\sum p_{ij})^{-1}$ (Henin & White pg. 698). In this

model when an agent exists by stepping onto a door cell it disappears from the model in the next time step and is said to have exited the room successfully.

The second model, The Swarm Force Model, is a model that was inspired by the Kirchner field model. It uses many of the same methods that the Kirchner model uses, but it improves upon it by making a few changes. These changes include an addition of a dynamic force field, cell selection rules, and the addition of injuries. These changes allow the model to become more realistic in simulating evacuation scenarios by modeling the effects of people pushing each other, people crowing densely, and people being disabled by injuries.

The biggest change that the Swarm Force model makes to Kirchner's is adding a dynamic force field. The forces in this model are represented by force bosons. Force bosons are unit vectors that are analogous to directional elementary particles. The force field has a discrete value on each cell that is the vector sum of the force bosons on that cell, and the field represents the magnitude and direction that is experienced by an agent on a cell. The forces on one cell are not perceptible to agents on other cells. Force bosons are deposited within the model when agents desire to move to a cell but cannot because the cell is occupied. This is done to simulate pushing done by agents, and each agent can drop a variable amount of force bosons each time they wish to move to an occupied cell that is determined by the input parameters of the model (Henin & White pg. 699).

During every time step, the force bosons on a cell are propagated to neighboring cells. This is because it has been observed that crowd forces are retransmitted from person to person through interpersonal contacts within crowds. This retransmission causes the sum of the force

bosons to propagate rather than the individual bosons themselves, and force dissipates if it moves onto an empty cell, a wall, or an injured person (Henin & White pg. 699).

The force field vector at a point (i,j), $v_{ij}$ indicates the magnitude and direction of force transmission. When $v_{ij}$ points between two cells, a force boson is placed on one of the two cells it is determined which cell gets the boson by probabilistic means. The probability of a boson being placed on one of the cells is determined by: $1-p_a = \theta_{ij}/90 = p_b$, where $p_a$ is the probability of selecting the neighbor with the lower angle, $p_b$ is the probability of selecting the neighbor with the higher angle, and $\theta_{ij}$ is the angle taken modulo 90 of $v_{ij}$ (Henin & White pg. 699).

The cell selection methods that were present in Kirchner's model have been changed in a couple of ways. The first change is that normal cell selection methods are bypassed if the force experienced by an agent exceeds a threshold level. If this happens an agent moves in the direction of the force that it experiences (Henin & White pg. 700).

The other major change to the model is to replace the occupancy number $n_{ij}$ with a vacancy number $\Phi_{ij}$. This number is zero if the cell is unoccupied and 0.5 otherwise. This halves the cell score for occupied cells relative to unoccupied cells and changes the cell selection formula to: $p_{ij} = N \exp(k_D D_{ij}) \exp(k_S S_{ij}) (1-\Phi_{ij}) \zeta_{ij}$. This change is made to the Kirchner model in order to make this model more realistic. In the Kirchner model an agent would never select a cell that is occupied during one time step even if it becomes unoccupied by the next time step. This artificially spaces out the crowd and is unrealistic since people will usually follow each other closely when leaving a room. This change allows an occupied cell to be selected if it become unoccupied by the next time step. This change also makes it less likely that agents will remain in the same cell for multiple time steps (Henin & White pg. 700).

The final change that the Swarm Force model makes to Kirchner's is the addition of injuries. It is important to include the simulation of injuries within the model to simulate exit behavior because it can have a drastic effect on the progress of other agents in the room. Whenever an agent experiences a force that is greater than a threshold parameter $\phi$, it becomes injured. When an agent becomes injured in this model, it loses the ability to move throughout the cell, and other agents throughout the model treat the agent like it is a wall cell when determining where they would like to move (Henin & White pg. 700).

**Our Strategy**

To implement these ideas, we used MATLAB to create code and display our results. We will introduce our program structure, including specific functions we have created, as well as indicate some results we have obtained thus far.

We start off by establishing a number of parameters, including the dimensions of the room (m x n), the number of people, N, the occupation matrix, S, and the location of the exit (exitx,exity).

We next establish the initial conditions for each person. Using a uniform probability distribution, each person is placed randomly throughout the room at point (p,q). We then associate a Boolean with each person, I, indicating whether or not the person is still in the room. This value is set to 1 for all people, and the value of the $(p,q)_{th}$ entry in matrix S to 1. We then determine which people in the room will act first by establishing a force for each person. This force is then ranked, from which the people with the highest perceived force will be the first to move in that time step, as seen in Table 1. We will update these forces with every time step.

| # | p | q | F | # | p | q | F |
|---|---|---|---|---|---|---|---|
| 1 | 45 | 23 | 12 | 2 | 28 | 14 | 40 |
| 2 | 28 | 14 | 40 | 4 | 99 | 49 | 34 |
| 3 | 76 | 66 | 6 | 1 | 45 | 23 | 12 |
| 4 | 99 | 49 | 34 | 3 | 76 | 66 | 6 |

**Table 1: An example of sorting people by their perceived force.**

We then begin a loop tracking the movements of people per time step until every person has left the room. At this time, new forces are introduced to determine which people

will get to act first. We first developed formulas based on an electrical field model, in which people are “attracted” to the door while experiencing “repulsion” from other individuals. We found this to be somewhat realistic as people will want to head towards the exit but are also aware of, and fearful of, getting crushed by everyone else. People experience forces in both the x and y direction, depending on their proximity to the exit and their location with respect to the rest of the crowd.

For an attractive force towards the door, we used the following formula for the x-direction:

$$F_x \propto \frac{sign\left(exitx - p\right)}{\left(\left|q - exity\right|^3 + 1\right)\left(\left(p - exitx\right)^2 + 1\right)},$$

where *p* and *q* are the coordinates of the person, and *exitx* and *exity* are the location of the exit. We had a *y*-term in this *x*-directional force for a couple of reasons. We found this formula to be effective in attracting people towards the exit, and we needed to introduce a *y* component so that the field was not uniform for any particular value of *x*. Without it, we found that people were much too likely to head towards the walls, contrary to our notions of group dynamics.

Similarly, we have a repulsion force between individual persons relating to their location. For the y-direction, this force between persons *i* and *j* is

$$F_{ij,y} \propto -\frac{1}{\left(q_j - q_i\right)R_{ij}^{\ 2}},$$

where *R* is the distance between *i* and *j*. If both people have the same value of *p* or *q*, then the directional force is zero. This is summed up over the entire field of people still within the room. This value greatly diminishes with distance, so individuals with closest proximity to a particular

individual will produce the greatest repulsion. We currently have the magnitude of these forces, which determines our movement order, derived by summing the absolute values of the directional components. We also tried using the L2 norm, but noticed little if any difference.

We keep track of the directional forces to determine which direction each person wants to move to. While they are all trying to reach the exit, this can mean moving in any number of directions. To give preference to certain directions based on the force vector, we developed the function *moveDirection*. This function establishes a probability distribution from which we will choose a random number and determine which direction to move each person. Since people can only move one cell at a time, we found the angle formed by the force vector and grouped it into one of eight directions (right, up-right, up, up-left, left, down-left, down, down-right). Each preferred direction then has its own probability distribution for determining the likelihood of that person to go to a specific cell. Under our current construction, it is possible, though unlikely, for people to go in the opposite direction they intend to move to. See Figure 1 for one such example. We do this because we feel that people will always try and move, given the urgency of the situation. This enables them to find other vacant cells which may be closer to the exit in a future time step.

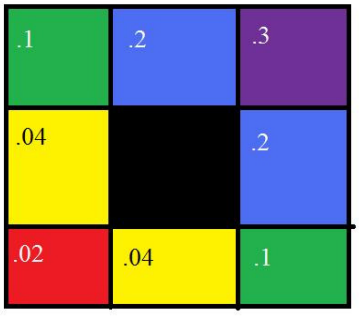


**Figure 1: A probability distribution for a person in the center cell wanting to move up-right.**

We repeat this *moveDirection* function for every person. However, since there are multiple people involved, we must address the issue of people interacting. What if a cell they try to move to is occupied? In that case, we use our pre-move function *canIMoveTo*. This function returns a Boolean indicating whether or not the person can move to the selected location. If it is open, they are free to move there, and do such immediately before the next person moved. However, in the event that the desired cell is a wall or occupied by another person, we restart the probability process using the same distribution to try and see if the person will move to another cell. Again, if that second choice is vacant, they are free to move there. We give each person a set number of times they can try moving again. If after all these times the person has not been able to move, then the person forgoes moving this time step and stays put in the same cell.

As each person moves, we accordingly update the S matrix, placing a 1 in each new spot and deleting the value in the previous cell. This enables another person to move to that cell during the same time step. We also check to see if the person moving has reached the exit. If they have, we change their value of one to zero, indicating that they are no longer in the room. This takes them out of the pictures in terms of computing repulsive forces, as well as enables others to exit.

For those who are still in the room after the time step, we generate a plot illustrating their locations. We use the Boolean I, and store the coordinates of remaining people into vectors which we then plot. We refresh the plot after every time step, so we can observe the behavior of the people throughout the entire time period. Such an evolution over time can be

seen in Figure 2. Using MATLAB, we are also able to store each graph as a frame in a video sequence. This allows us to display our results outside of MATLAB.

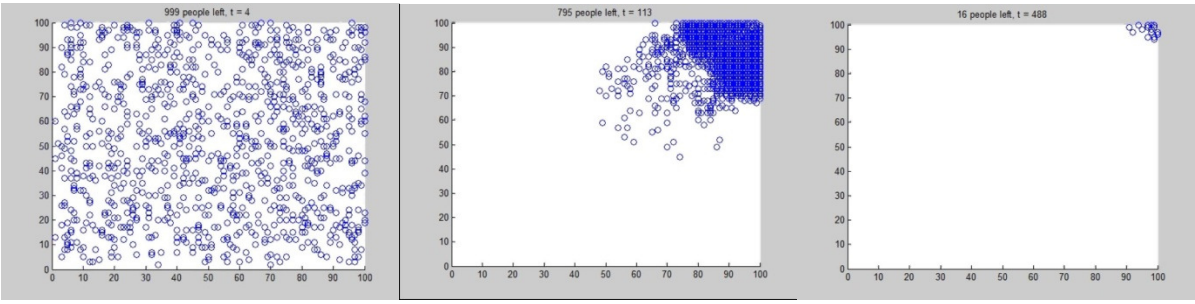


**Figure 2: Three phases in a typical run. In this trial, 1000 people are escaping through a door in the upright corner of the room.**

We repeat the process until everybody has left the room, at which point we automatically reach our exit time and stop the program.

**Implementation of Kirchner and Swarm Force Models**

In our transition from our electric field model to recreating the Kirchner and Swarm Force models, we realized that our base construction enabled us to take a few shortcuts. We felt that by treating walls and occupied spots as equal, this would enable us to reduce the number of variables in the equation by one. Similarly, we also investigated methods to make the values of $D_{ij}$ and $S_{ij}$ more comparable, such that they both have a common range of values.

As mentioned, the Kircher formula can be written as

$$p_{ij} = N \exp\left(k_s S_{ij}\right) \exp\left(k_d D_{ij}\right) \left(1 - n_{ij}\right) \zeta_{ij},$$

where $n_{ij}$ and $\zeta_{ij}$ are variables meant to track the location of walls and other occupants separately. This way, if either a wall or another individual already occupied a particular spot, there would be zero probability of our agent trying to move to that spot. By increasing the size of our occupancy matrix by 2 in each direction, we were able to track the locations of walls as well as moving people. We were able to adjust our movement rules accordingly, and thus we felt we would simplify the probability as

$$p_{ij} = N \exp\left(k_s S_{ij}\right) \exp\left(k_d D_{ij}\right) \left(1 - n_{ij}\right).$$

One limitation we did face with our implementation was the movement of bosons from one cell to another. We decided to forgo the transient motion of bosons but did implement a method for them to decay over a period of time. We decided to use a geometric progression to quantify the rate of decay, which would decrease after each time step. Thus, this also maxed out the number of bosons possible in a cell. By tracking the number of bosons per cell in a separate matrix, it was easy to add bosons as people moved, as well as represent decay over

time. Cells that had not been visited in a large number of time steps would have a lower probability of being visited in subsequent time steps.

Unfortunately, this simplification, which worked well for the Kirchner model, did not translate as smoothly with the Swarm Force model. Henin, et al. implemented a number of variables to account for the chaotic movement and pushing of individuals, which we felt we would be unable to maintain properly. It was noted that there is a reduced probability that a person will want to move to a spot that is already occupied, so in implementing the Swarm Force probability distribution, we had to allow for people to want to move to occupied locations. Thus, we came up with the probability distribution

$$p_{ij} = N \exp\left(k_s S_{ij}\right) \exp\left(k_d D_{ij}\right) \left(1 - \frac{n_{ij}}{2}\right).$$

Our function *canIMoveTo* also enabled us to implement the notion of shoving with relative ease. Since we already check to see if a person is in the desired location, we implement that the person in that spot gets pushed. Afterwards, the person trying to move attempts to find another open cell, and may end up pushing more people before moving or running out of turns. With this implementation, we also had to keep track of how many times a person would get pushed and when would they be considered injured. As in the Swarm Force paper, they test a number of values to see how the “toughness" of people, their ability to be pushed multiple times, affected the number of individuals that eventually were able to escape. We will observe this behavior later. In general, we used the value *paintol* to max out the number of times a person could be shoved around without getting injured. We reset the number of times a person was pushed to zero when they move to a new cell, but if the person was pushed a

certain number of times without being able to move, we marked them as being injured. For a majority of our cases, we set our value of *paintol* to 100, given that each person would have 10 chances to move to a new cell when it was their turn.

In our implementation of the Kirchner and Swarm Force models, one large problem we had was determining proper values of $k_S$ and $k_D$ to test. Since our simulation programs were somewhat different from the original models, our results would vary. In general, we felt that values of $k_S$ and $k_D$ greater than 1 would have the greatest effect on the probability distribution. As these values approach zero, the distribution becomes more uniform; as a check, we did run simulations with $k_D$ and $k_S$ equal to zero; individuals would just wander around aimlessly.

Per the advice of our mentor, we looked into methods to normalize our values of $S_{ij}$ and $D_{ij}$ such that it would be easier to focus on comparing our results by changing only $k_S$ and $k_D$. In the literature, these values do not necessarily hold equal weight. Since $S_{ij}$ is determined relative to the distance to the nearest exit, and $D_{ij}$ is the number of bosons in each particular cell, there was no guarantee that the static and dynamic fields had equal influence. We created a mapping for each individual cell that translated these values of $S_{ij}$ and $D_{ij}$ to [0,1]. We renamed these values as $R_{ij}$ and $B_{ij}$, respectively. For both scalings, we based the new values with their respect to the range of the values of other cells. Thus, for example with the bosons, the cell with the fewest bosons would scale to zero and the cell with the most bosons scaled to one. For simplicity, we did the same thing for the values of $R_{ij}$; the cell closest to the exit was scaled to zero and the cell furthest away scaled to one. For values in between the minimum and maximum, we scaled linearly. Thus,

$$B_{ij} = \frac{B - B_{\min}}{B_{\max} - B_{\min}},$$

and likewise for $R_{ij}$. For the case of the bosons, if all cells had the exact same number of bosons (an unlikely but still plausible event) the $\exp(k_D B_{ij})$ had no influence on the probability distribution as it was the same factor for all cells.

Of course, given the original probability distribution, our new derived values of $R_{ij}$ meant that people would prefer to go to the cell furthest from the exit. To remedy this, we changed the sign in our probability distribution. Thus, our new $p_{ij}$ can be written as

$$p_{ij} = N \exp\left(-k_s R_{ij}\right) \exp\left(k_d B_{ij}\right) \left(1 - \frac{Q n_{ij}}{2}\right),$$

where Q = 1 for the Swarm Force model, and 2 for the Kirchner model.

**Results**

After we were able to implement our modified equations for the Kirchner Field Model and the Swarm Force Model into our program, we were able to run some simulations with our model to analyze the effect of changing different parameters.

The first series of simulations that we ran were designed to determine how the ratio of different values of $k_S$ and $k_D$ would change the rate at which people exited the room. We fixed the value of $k_S$ to be equal to three and then we ran simulations for $k_D$ = 0, 1, 1.5, 2, 3, 4, 5, 6, and 7. We used a room of 100 x 100 cells, starting each simulation with 1000 people in the room, and used a *paintol* of 100. For each of these different values we kept track of the number of people left in the room as time progressed and graphed the number of people left in the room as time elapsed. This gave us the following graph.

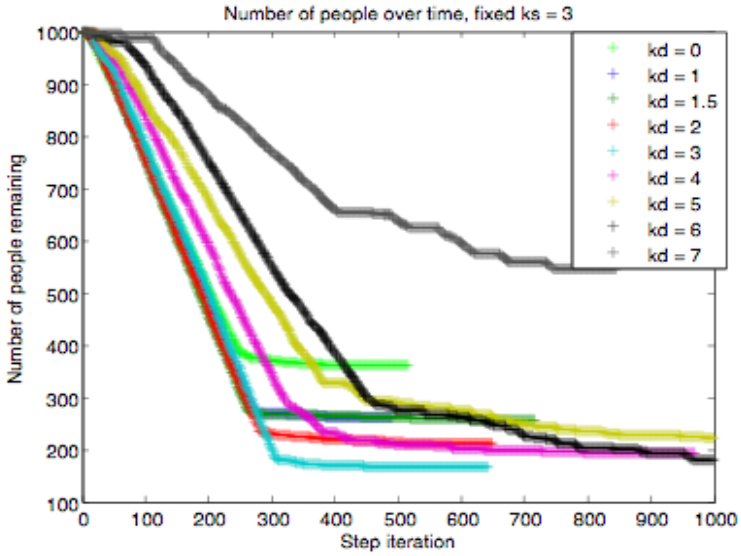


**Figure 3: Graphs of people remaining in room as a function of time. $K_D$ see key, $K_s$=3.**

We also analyzed how changing the injury threshold in our model would affect the number of people that were injured during the course of the simulation. For this analysis we used $k_S = 3$ $k_D = 2$, a 63 x 63 cell room, and started with 1116 people in the room. We ran simulations for a paintol of 1, 50, 100, 150, 200, 250, and 300 and graphed how many people were injured during each run. This gave us the following graph.

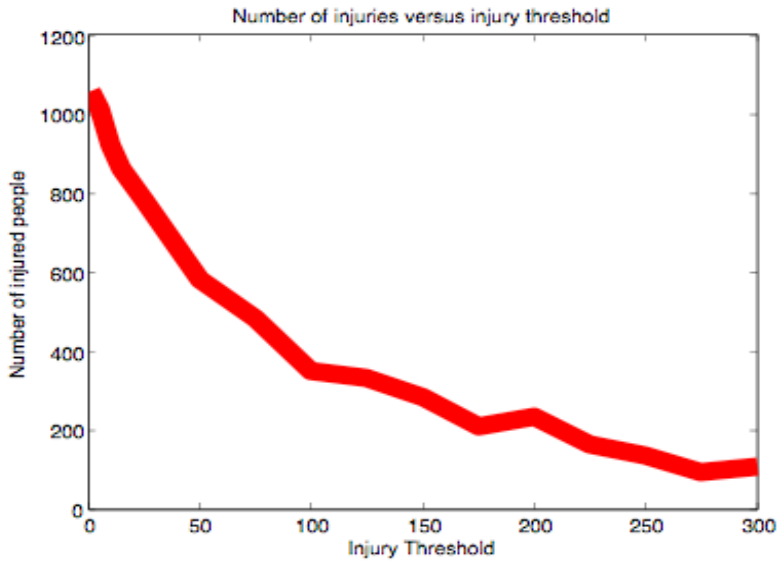


**Figure 4: Plot of number injured as a function of the injury threshold. ($K_S$=3, $K_D$=3)**

Another thing that we wanted to analyze was how the density of people in the room would affect the percentage of people that were able to exit. We started each simulation with a 63 x 63 grid cell, $k_S = 3$, $k_D = 2$. We then ran a simulation to completion by starting out with enough people in the room so that 5, 10, 15, 20, 25, 30, 35, and 40 percent of the total cells in the room were occupied and recorded how many people escaped the room during the simulation. This data gave us the following graph:

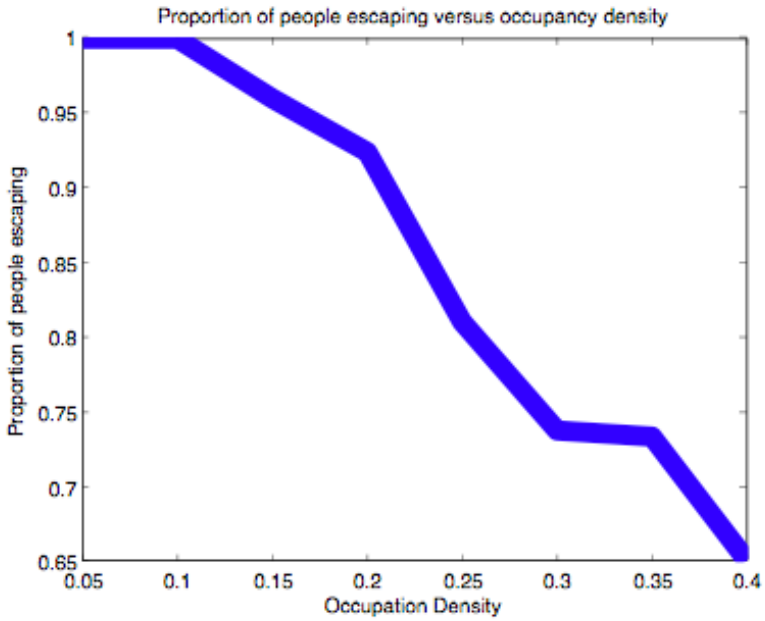


**Figure 5: Plot of percentage of individuals leaving the room as a function of density of individuals in the room. ($K_S$=3, $K_D$=2)**

**Conclusions & Discussion**

A number of things stand out from the results. First, when comparing how the different ratios of $k_S$ and $k_D$ affected the rate at which people left the room, we saw that for most ratios people leave the room at a similar rate. The only differences observed for different ratios of these parameters is the amount of time that it took for people to start exiting the room, and how many people were able to exit. We found optimal results when $k_S$ and $k_D$ were equal since more people were able to exit and they did so rather quickly. We saw the worst results when $k_D$ was large in relation to $k_S$, since relatively few people were able to exit the room and it took them much longer to begin leaving. This shows that the most optimal conditions for people to exit a room is to compromise between their own belief on the best course of action, which would be to head for the door, while still following and reacting to the actions of others.

When we analyzed the effect of the injury threshold on the number of injured people, we were able to closely replicate the results that are expected and the results obtained by Henin and White (pg. 708). As expected the more pain that people can tolerate the less likely they are to be injured by other people shoving them.

When we looked at how the density of people in a room effected how many people were able to exit, we saw that increasing the density of people increases the percentage of people that got injured. This makes sense since a more tightly packed room results in more collisions and more people pushing one another, thus causing injuries. It is important to look at how the crowd density affects the number of injuries so that a maximum occupancy for a room can be determined that will minimize the risk of injury in the event of a required evacuation.

**Future Work**

There are a number of things that could be done in the future to further our work and make our model even more realistic. One thing would be to implement a more exact version of the Kirchner Field/Swarm Force Model with all the same parameters and values. Doing this would allow us to attempt to reproduce the results that were obtained in Henin and White's paper.

Another thing that could be done is to institute a penalty into our code so that people do not get stuck going back and forth when $k_D$ is high. This change to the motion rules would make the model more realistic since it would prevent people from acting in an illogical way. Finally, one other thing that could be done would be to implement a model that allows for multiple exits to the room. This could then help to determine the best locations for exits in a room to allow for efficient escapes with minimal injuries.

**References**

Henin, C. M. & White, T. Macroscopic Effects of Microscopic Forces Between Agents in Crowd Models. *Physica.* A 373, 694-712 (2007).

Kirchner, Ansgar & Schadschneider, Andreas. Simulation of Evacuation Processes Using a Bionics-Inspired Cellular Automaton Model for Pedestrian Dynamics. Physica. A 312, 260-276 (2002).

Seyfried, A., Steffen, B., Lippert, T. Basics of Modelling the Pedestrian Flow. *Physica* A 368, 232-238 (2006).

Weng, W.G., Pan, L.L., Shen, S.F., Yuan, H.Y. Small-grid Analysis of Discrete Model for Evacuation from a Hall. *Physica.* A 374, 821-826 (2007).

**Appendix**

Below is the code we developed for this project. We used the following three programs in our simulations for the Swarm Force model; the main program, and two auxiliary functions for the main program.

==========

```
%*This is the Main Program*

clear all

 %This is the code for MATH 485 Project code. This is the main function.
%Make sure to have this and the other functions in the same directory. The
%code template was provided by Brent.

%This versions implements the Swarm Force model by Henein, et al. as
%introduced in their paper. One main difference between the Swarm Force
%model and the Kirchner model is that people can become injured and are no
%longer able to reach the exit. We implemented this by establshing a
%modified probability distribution as discussed in the m-file
%moveDirectionSF. We introduce other important factors for the sawrm force/
%boson model as they are developed.

% %These lines initiate the video capture so I can make the movies. It ifs
often preferred to have
% %these commented out to save time and computing power.
%
% aviobj = avifile('485video42309','COMPRESSION','Cinepak','FPS',10);
% fig = figure;

%Initiate the number of people. More people will require more computing
%time.

N = 1588;

%Initiate the dimensions of the room

m = 63;%length
n = 63;%width

%Initiate a number of parameters involved for the simulation

%The static factor, how fast people wanna reach the exit
ks = 3;

%The dynamic factor, how likely people are to follow other people,
%as if they were unable to see the exit directly.
kd = 2;
```

```
%The rate of which bosons dissappear. In the Kirchner model, bosons have a
%probability to decay and also to move to another cell. In our model, we
%only have the rate drop geometrically.
bosdecay = .5;

%How many "tries" a person gets to try and move to an adjacent cell before
%giving up.
trialmoves = 10;

%paintol is the number of times a person is able to be "pushed" before they
%become injured. We tried experimenting with paintol as a function of other
%parameters, but for the bulk of our siimulations we have set it as a
%constant.
%paintol = trialmoves*(ks+1)*(kd*bosdecay+1)*2;
paintol = 250;

%Our maximum number of iterations.
tmax = 1000;

%Other stuff that tracks the number of people at any given time.
tlast = 0;
Nplot37 = [];
tplot37 = [];

%Initiate the matrix monitoring each person's location. The main ones are
%S and Sold. We also initiate the Boson matrices to keep track of new ones
%and adding them per step to the remaining decaying ones. One difference
%between our original "electric field" model and the Kirchner models is the
%size of our matrices. In our original model we only had them as 100x100,
%but now we must treat the walls as potential spots as well, hence the +2.
Sold = zeros(m+2,n+2);
S=zeros(m+2,n+2);
Bos = zeros(m+2,n+2);
BosNew = zeros(m+2,n+2);

%Initiate the locations of exits and doorways. These values can be
%distinct. Right now our versions of the Kirchner/SF models are only able
%to support one exit, which is a clear advantage our initial model had in
%that it was relatively easy to implement multiple exits.

%The door location is used in the attraction "forces" to see who gets to
%move first. This is separate than the forces we actually measured in our
%first model.
door1atx = 1;
door1aty = 1;

%door2atx = 80;
%door2aty = 101;

exit1x = 1;%The exit location is used to determine when people leave.
exit1y = 1;

%exit2x = 80;
%exit2y = 100;
```

```
%Initialize the matrix with people all over

hit=1;
while hit<N+1

    %Set up coordinates with a uniform random distribution.
    i=round(m*rand+0.5);
    j=round(n*rand+0.5);

    %Fill the matrices with 1's where people are. We introduce a number of
    %variables including coordinates p and q, the number of the person, an
    %arbitrary initial force and directional forces to determine rank, and
    %a boolean indicating the statuts of the person (still inside the room
    %= 1, outside or injured = 0). These vectors grow within the loop.
    if Sold(i+1,j+1)<0.5
           Sold(i+1,j+1)=1;
           person(hit)=hit;
           p(hit)=i;%x coordinate
           q(hit)=j;%y coordinate
           F(hit)=1/((door1atx-i)^2+(door1aty-j)^2);
           FX(hit) = .1;
           FY(hit) = .1;
           I(hit) = 1;%boolean for are they mobile
           shoves(hit) = 0;%tracks # of shoves per person
           disabled(hit) = 0;%special case if disabled
           hit=hit+1;
    end
end

%We organize our data into a large matrix which lists all the important
%factors per person. We then organize the order of which people move by the
%magnitude of force experienced, so those with the greatest force (and
% closest to the door) will go first.
PQ=[person; p; q; F; I; FX; FY; shoves; disabled];
PQ2=flipud(sortrows(PQ', 4));

%PQ2 contains all the necessary information for each person, including
%their number of generation, their current location, the total force
%directed upon them, whether or not they are still in the room, how many
%times they've been pushed, and their current status.

%Initialize timestep
t=0;
figure;
tTrack = 0;
while t<tmax
    clf;%erases old graph
    Ntrack = (mean(PQ2(:,5))+mean(PQ2(:,9)))*N;
    hit = 1;
    Bos = Bos*bosdecay;
    while hit < N+1


        %These next few lines pertain to our old model. PLease ignore.
```

```
% % %     %Establish attraction rules towards exits. Right now the force is
% % %     %proportional to |dy|^-3*dx^-2 for x direction. The cubic part is
to
% % %     %diminish forces far away from the sources. We will aim to
implement the
% % %     %other forces as indicated in the literature in a short time.
% % %
% % %     %for door1
% % %     %PQ2(hit,6)= 10*sign((door1atx - PQ2(hit,2)))/((abs(door1aty -
PQ2(hit,3))^3 + 1)*((door1atx - PQ2(hit,2))^2 + 1));
% % %     %PQ2(hit,7)= 10*sign((door1aty - PQ2(hit,3)))/((abs(door1atx -
PQ2(hit,2))^3 + 1)*((door1aty - PQ2(hit,3))^2 + 1));
% % %
% % %     %for door2
% % %     %PQ2(hit,6)= PQ2(hit,6) + 1000*sign((door2atx -
PQ2(hit,2)))/((abs(door2aty - PQ2(hit,3))^4 + 1)*((door2atx - PQ2(hit,2))^2 +
1));
% % %     %PQ2(hit,7)= PQ2(hit,7) + 1000*sign((door2aty -
PQ2(hit,3)))/((abs(door2atx - PQ2(hit,2))^4 + 1)*((door2aty - PQ2(hit,3))^2 +
1));
% % %
% % %     %establish repulsive forces between people. Right now the force is
% % %     %proportional to the distance between people.
% % % %     for k = 1:N
% % % %         if k ~= hit
% % % %             R = (PQ2(hit,2) - PQ2(k,2))^2 + (PQ2(hit,3) -
PQ2(k,3))^2;
% % % %             if ((PQ2(k,2) ~= PQ2(hit,2)) && PQ2(k,5) == 1)
% % % %             PQ2(hit,6) = PQ2(hit,6) - 0.0000001/((PQ2(k,2) -
PQ2(hit,2))*R);
% % % %             end
% % % %             if ((PQ2(k,3) ~= PQ2(hit,3)) && PQ2(k,5) == 1)
% % % %             PQ2(hit,7) = PQ2(hit,7) - 0.0000001/((PQ2(k,3) -
PQ2(hit,3))*R);
% % % %             end
% % % %         end
% % % %     end

    %force is currently ranked by the inverse of the distance to the door. We
rerank our PQ2 matrix
    %to see who will get to move first.
    PQ2(hit,4) = 1/((PQ2(hit,2) - door1atx)^2 + (PQ2(hit,3) - door1aty)^2);
    hit=hit+1;
    end
    PQ2=flipud(sortrows(PQ2, 4));

    %Now we use number generation to determine where the person will move,
one at a time.
    %We use two functions, canIMoveTo and moveDirection, to
    %probabilistically determine which direction a person wants to move to,
    %and based on the probability, determine if a person can move to the
    %selected spot.
    for i=1:N
      if PQ2(i,5) == 1%only tracking those still able inside the room
          trials = 1;
```

```
            while (trials <= trialmoves)%This value can change. With more
trials,
                %the person is more likely to have moved even if backwards.

                %Establish our random probability
                locator_prob = rand;

                %This calls the function moveDirection. This function
                %determines the probability distribution depending of the
                %locations and bosons experienced by each individual.

                [p1,p2,p3,p4,p5,p6,p7] =
moveDirectionSF(ks,kd,S,Bos,PQ2(i,2),PQ2(i,3),door1atx,door1aty);

                %Based on these distributions, these lines move the people
                %in that particular direction. We start with moving
                %up-right, and the order moves counterclockwise from there.
                %The probability distributions were made with the
                %moveDirecttion function.

                %The function canIMoveTo is a boolean indicating whether or
                %not a specific move is valid. We check to make sure that
                %the person can move there first. If they can, they are
                %allowed to do so. If not, they push whom/whatever is there
                %and retry moving somewhere else via another trial.

             if (locator_prob>0.0 && locator_prob<=p1)%move up-right

                 canmove = canIMoveToK(PQ2(i,2) + 1, PQ2(i,3) + 1, m,n,S);
                 if (canmove)
                 %If they can move to the desired spot, we update their
                 %coordinates and the matrices to keep track.
                 BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
                 PQ2(i,2) = PQ2(i,2)+1;
                 PQ2(i,3) = PQ2(i,3)+1;
                 Sold(PQ2(i,2),PQ2(i,3))=0;
                 S(PQ2(i,2),PQ2(i,3))=0;
                 S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
                 %We also reset the number of times they have been pushed
                 %around to 0. We assume that these people are kinda
                 %resilient and can only become injured if they get pushed
                 %too much in the same place.
                 PQ2(i,8) = 0;
                 trials = 1000;%to kick out of trial loop for that person
                 else
                     %If they can't move there, we have to find the person
                     %who is there and then get them pushed by the current
                     %mover. If the person reaches the pushing threshold,
                     %they become injured and can no longer move.
                     A1 = find(PQ2(:,2) == PQ2(i,2) + 1);
                     A2 = find(PQ2(:,3) == PQ2(i,3) + 1);
                     AP = intersect(A1,A2);
                     PQ2(AP,8) = PQ2(AP,8) + 1;
                     if PQ2(AP,8) > paintol
```

```
                    PQ2(AP,9) = 1;
                    PQ2(AP,5) = 0;
                end
                trials = trials + 1;%have the person trying to move try
again
            end
        elseif (locator_prob>p1 && locator_prob<=p2)%move up

            canmove = canIMoveToK(PQ2(i,2), PQ2(i,3) + 1, m,n,S);
            if (canmove)
            BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
            PQ2(i,2) = PQ2(i,2); PQ2(i,3)=PQ2(i,3)+1;
            Sold(PQ2(i,2)+1,PQ2(i,3))=0;
            S(PQ2(i,2)+1,PQ2(i,3))=0;
            S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
            PQ2(i,8) = 0;
            trials = 1000;
            else
                A1 = find(PQ2(:,2) == PQ2(i,2));
                A2 = find(PQ2(:,3) == PQ2(i,3) + 1);
                AP = intersect(A1,A2);
                PQ2(AP,8) = PQ2(AP,8) + 1;
                if PQ2(AP,8) > paintol
                    PQ2(AP,9) = 1;
                    PQ2(AP,5) = 0;
                end
                trials = trials + 1;
            end

        elseif (locator_prob>p2 && locator_prob<=p3)% move up-left
            canmove = canIMoveToK(PQ2(i,2) - 1, PQ2(i,3) + 1, m,n,S);
            if (canmove)
            BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
            PQ2(i,2) = PQ2(i,2) - 1; PQ2(i,3)=PQ2(i, 3)+1;
            Sold(PQ2(i,2)+2,PQ2(i,3))=0;
            S(PQ2(i,2)+2,PQ2(i,3))=0;
            S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
            PQ2(i,8) = 0;
            tirals = 1000;
            else
                A1 = find(PQ2(:,2) == PQ2(i,2) - 1);
                A2 = find(PQ2(:,3) == PQ2(i,3) + 1);
                AP = intersect(A1,A2);
                PQ2(AP,8) = PQ2(AP,8) + 1;
                if PQ2(AP,8) > paintol
                    PQ2(AP,9) = 1;
                    PQ2(AP,5) = 0;
                end
                trials = trials + 1;
            end


        elseif (locator_prob>p3 && locator_prob<=p4)%move left, etc.
            canmove = canIMoveToK(PQ2(i,2) - 1, PQ2(i,3), m,n,S);
```

```
                if (canmove)
                BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
                PQ2(i,2) = PQ2(i,2)-1; PQ2(i,3)=PQ2(i, 3);
                Sold(PQ2(i,2)+2,PQ2(i,3)+1)=0;
                S(PQ2(i,2)+2,PQ2(i,3)+1)=0;
                S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
                PQ2(i,8) = 0;
                trials = 1000;
                else
                    A1 = find(PQ2(:,2) == PQ2(i,2) - 1);
                    A2 = find(PQ2(:,3) == PQ2(i,3));
                    AP = intersect(A1,A2);
                    PQ2(AP,8) = PQ2(AP,8) + 1;
                    if PQ2(AP,8) > paintol
                        PQ2(AP,9) = 1;
                        PQ2(AP,5) = 0;
                    end
                    trials = trials + 1;
                end

            elseif (locator_prob>p4 && locator_prob<=p5)
                canmove = canIMoveToK(PQ2(i,2) - 1, PQ2(i,3) - 1, m,n,S);
                if (canmove)
                BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
                PQ2(i,2) = PQ2(i,2)-1; PQ2(i,3)=PQ2(i, 3)-1;
                Sold(PQ2(i,2)+2,PQ2(i,3)+2)=0;
                S(PQ2(i,2)+2,PQ2(i,3)+2)=0;
                S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
                PQ2(i,8) = 0;
                trials = 1000;
                else
                    A1 = find(PQ2(:,2) == PQ2(i,2) - 1);
                    A2 = find(PQ2(:,3) == PQ2(i,3) - 1);
                    AP = intersect(A1,A2);
                    PQ2(AP,8) = PQ2(AP,8) + 1;
                    if PQ2(AP,8) > paintol
                        PQ2(AP,9) = 1;
                        PQ2(AP,5) = 0;
                    end
                    trials = trials + 1;
                end

            elseif (locator_prob>p5 && locator_prob<=p6)
                canmove = canIMoveToK(PQ2(i,2), PQ2(i,3) - 1, m,n,S);
                if (canmove)
                BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
                PQ2(i,2) = PQ2(i,2); PQ2(i,3)=PQ2(i, 3)-1;
                Sold(PQ2(i,2)+1,PQ2(i,3)+2)=0;
                S(PQ2(i,2)+1,PQ2(i,3)+2)=0;
                S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
                PQ2(i,8) = 0;
                trials = 1000;
                else
```

```
                    A1 = find(PQ2(:,2) == PQ2(i,2));
                    A2 = find(PQ2(:,3) == PQ2(i,3) - 1);
                    AP = intersect(A1,A2);
                    PQ2(AP,8) = PQ2(AP,8) + 1;
                    if PQ2(AP,8) > paintol
                        PQ2(AP,9) = 1;
                        PQ2(AP,5) = 0;
                    end
                    trials = trials + 1;
                end
            elseif (locator_prob>p6 && locator_prob<=p7)
                canmove = canIMoveToK(PQ2(i,2) + 1, PQ2(i,3) - 1, m,n,S);
                if (canmove)
                BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
                PQ2(i,2) = PQ2(i,2)+1; PQ2(i,3)=PQ2(i, 3)-1;
                Sold(PQ2(i,2),PQ2(i,3)+2)=0;
                S(PQ2(i,2),PQ2(i,3)+2)=0;
                S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
                PQ2(i,8) = 0;
                trials = 1000;
                else
                    A1 = find(PQ2(:,2) == PQ2(i,2) + 1);
                    A2 = find(PQ2(:,3) == PQ2(i,3) - 1);
                    AP = intersect(A1,A2);
                    PQ2(AP,8) = PQ2(AP,8) + 1;
                    if PQ2(AP,8) > paintol
                        PQ2(AP,9) = 1;
                        PQ2(AP,5) = 0;
                    end
                    trials = trials + 1;
                end
            elseif (locator_prob>p7 && locator_prob<=1)
                canmove = canIMoveToK(PQ2(i,2) + 1, PQ2(i,3), m,n,S);
                if (canmove)
                BosNew(PQ2(i,2)+1,PQ2(i,3)+1) = BosNew(PQ2(i,2)+1,PQ2(i,3)+1)
+ 1;
                PQ2(i,2) = PQ2(i,2)+1; PQ2(i,3)=PQ2(i, 3);
                Sold(PQ2(i,2),PQ2(i,3)+1)=0;
                S(PQ2(i,2),PQ2(i,3)+1)=0;
                S(PQ2(i,2)+1,PQ2(i,3)+1)= 1;
                PQ2(i,8) = 0;
                trials = 1000;
                else
                    A1 = find(PQ2(:,2) == PQ2(i,2) + 1);
                    A2 = find(PQ2(:,3) == PQ2(i,3));
                    AP = intersect(A1,A2);
                    PQ2(AP,8) = PQ2(AP,8) + 1;
                    if PQ2(AP,8) > paintol
                        PQ2(AP,9) = 1;
                        PQ2(AP,5) = 0;
                    end
                    trials = trials + 1;
                end

            end
            end
```

```
            %If a person lands at the exit, they leave.
            if ((PQ2(i,2) ==  exit1x) && (PQ2(i,3) == exit1y))
                PQ2(i,5) = 0;%change their status
                BosNew(exit1x + 1, exit1y + 1) = Bos(exit1x + 1, exit1y + 1)
+ 1;
                S(exit1x +1,exit1y +1) = 0;%reopen that cell
            end
      end

    %This updates the old matrix to the most recent movements.
    new=0;
    for w=1:m
        for j=1:n
            if S(w+1,j+1)>0.5
                Sold(w+1,j+1)=S(w+1,j+1);
                new = new +1;
            end
        end
    end

    end

    %This updates that other stuff from earlier
    Nleft = (mean(PQ2(:,5)+mean(PQ2(:,9))))*N;
    Nplot37 = [Nplot37 Nleft];
    tplot37 = [tplot37 t];


    %update the time step
    t = t +  1;
    tlast = t;

    %Update the Boson matrices
    Bos = Bos + BosNew;
    BosNew = zeros(m+2,n+2);

    %We have 3 cases that exit our program. This is the first of them. If
    %nobody new has left the room within 100 time steps, we assume that
    %nobody else will make it our of the room so we just quit.
    Ntrack2 = (mean(PQ2(:,5)+mean(PQ2(:,9))))*N;
    if (round(Ntrack2) == round(Ntrack))
        tTrack = tTrack + 1;
        if tTrack >= 100
           t = tmax;
           tlast = tlast - 100;
        end
    else
        tTrack = 0;
    end

    %These establish our graph with all the locations of the people.
    NC=0; x = []; y = [];

    for i = 1:N
```

```
        if PQ2(i,5) > 0.5 %for people still in the room
            NC = NC + 1;
            x(NC) = PQ2(i,2);%We track their current coordinates in vectors
            y(NC) = PQ2(i,3);
        end
    end

    NC = 0; dx = []; dy = [];

    for i = 1:N
        if (PQ2(i,9) > 0.5)
            NC = NC + 1;
            dx(NC) = PQ2(i,2);
            dy(NC) = PQ2(i,3);
        end
    end

%graphical results, plotting the lot of people who are still in the room.
%We change the title to account for time and how many people are still in
%the room.

hold on;
numleft = (mean(PQ2(:,5))+ mean(PQ2(:,9)))*N;
H = [num2str(numleft) ' people left, t = ' num2str(t) ', ks = ' num2str(ks)
', kd = ' num2str(kd)];
title(H)

%This line plots the people. The last option in quotes is used to change the
color and style of
%points. b is blue, r is red, k is black, and there are others. The '.' tells
%MATLAB to just plot the points and to not connect consecutive points as is
%the default. 'o' makes a circle and you can use '-', '+', and probably
%others.

plot(x,y,'b.');
hold on;
plot(dx,dy,'r.');
axis square
axis([0 m 0 n])

    %These are the other two criteria for quitting the program. Either
    %everyone who is still able to has left the room before the time limit,
    %or if we reach the time limit, we stop. Upon quitting the program, we
    %display our end results.
    if ((mean(PQ2(:,5)) == 0) || t == tmax)

        t = tlast;
        %This sentence gets displayed on the main terminal
        fprintf('\n%d people escaped in %d seconds\n',round(N -
N*(mean(PQ2(:,5))+mean(PQ2(:,9)))),tlast);
        fprintf('\n%d people are stuck
inside\n',round(N*(mean(PQ2(:,5))+mean(PQ2(:,9)))));
        fprintf('\n%d people are injured\n',round(N*mean(PQ2(:,9))));
        t = tmax;
        figure (3)
```

```
        hold on;
        plot(tplot37,Nplot37,'r');
        axis normal
    end
%This simply slows down the program slightly to display the graph for just
%a tad longer.
pause(.0002)

% %These lines are used with the video capture. Have these uncommented if you
% %wish to make a video.
%
% F = getframe(fig);
% aviobj = addframe(aviobj, F);

end
================================================================================

%*This is one of the two auxiliary functions*

This function takes the neighboring cells into account and determines
%where they want to generally go. The outputs are a cumulative distribution
%function for moving in specific directions.

%THIS MODEL IS MEANT TO NORMALIZE THE PREFERENCE BETWEEN THE BOSONS AND THE
%STATIC FIELD. moveDirection0415K IS FOR THE ONE BASED ON THE KIRCHNER MODEL.

function [p1,p2,p3,p4,p5,p6,p7] = moveDirectionSF(ks,kd,S,Bos,x,y,exx,exy)

%We first track how many bosons are in each neighboring cell. We then scale
%them from 0 to 1 based on the range of the bosons. Cells with a scale of 1
%will be more likely to be visited as the dynamic factor kd increases.
Bur = Bos(x+2,y+2);
Bu = Bos(x+1,y+2);
Bul = Bos(x,y+2);
Bl = Bos(x,y+1);
Bdl = Bos(x,y);
Bd = Bos(x+1,y);
Bdr = Bos(x+2,y);
Br = Bos(x+2,y+1);
Bsum = Bur + Bu + Bul + Bl + Bdl + Bd + Bdr + Br;
Bees = [Bur Bu Bul Bl Bdl Bd Bdr Br];
Bmin = min(Bees);
Bmax = max(Bees);
if (Bmax ~= Bmin)
    Bur = (Bur - Bmin)/(Bmax - Bmin);
    Bu = (Bu - Bmin)/(Bmax - Bmin);
    Bul = (Bul - Bmin)/(Bmax - Bmin);
    Bl = (Bl - Bmin)/(Bmax - Bmin);
    Bdl = (Bdl - Bmin)/(Bmax - Bmin);
    Bd = (Bd - Bmin)/(Bmax - Bmin);
    Bdr = (Bdr - Bmin)/(Bmax - Bmin);
    Br = (Br - Bmin)/(Bmax - Bmin);
end

%Likewise we do a similar thing with the distance of each cell to the exit,
```

```
%scaled from 0 (closest) to 1 (farthest)
Dur = sqrt((exx-(x+1))^2+(exy-(y+1))^2);
Du = sqrt((exx-(x))^2+(exy-(y+1))^2);
Dul = sqrt((exx-(x-1))^2+(exy-(y+1))^2);
Dl = sqrt((exx-(x-1))^2+(exy-(y))^2);
Ddl = sqrt((exx-(x-1))^2+(exy-(y-1))^2);
Dd = sqrt((exx-(x))^2+(exy-(y-1))^2);
Ddr = sqrt((exx-(x+1))^2+(exy-(y-1))^2);
Dr = sqrt((exx-(x+1))^2+(exy-(y))^2);
Dees = [Dur Du Dul Dl Ddl Dd Ddr Dr];
Dmin = min(Dees);
Dmax = max(Dees);
Dur = (Dur - Dmin)/(Dmax - Dmin);
Du = (Du - Dmin)/(Dmax - Dmin);
Dul = (Dul - Dmin)/(Dmax - Dmin);
Dl = (Dl - Dmin)/(Dmax - Dmin);
Ddl = (Ddl - Dmin)/(Dmax - Dmin);
Dd = (Dd - Dmin)/(Dmax - Dmin);
Ddr = (Ddr - Dmin)/(Dmax - Dmin);
Dr = (Dr - Dmin)/(Dmax - Dmin);

%We then initiate the probability distribution for moving in a particular
%direction by incorporating our scaled values for distance and boson
%attraction. THE BIG DIFFERENCE IN THE SWARM FORCE MODEL IS THAT WE DIVIDE
%THE LAST OCCUPANCY TERM BY 2 TO GENERATE A PROBABILITY THAT SOMEONE MIGHT
%STILL TRY AND GO TO THAT SPOT AND PUSH A PERSON. THE KIRCHNER MODEL DOES
%NOT ALLOW FOR PUSHING AT ALL.
pur = exp(kd*Bur)*exp(-ks*Dur)*(1-S(x+2,y+2)/2);
pu = exp(kd*Bu)*exp(-ks*Du)*(1-S(x+1,y+2)/2);
pul = exp(kd*Bul)*exp(-ks*Dul)*(1-S(x,y+2)/2);
pl = exp(kd*Bl)*exp(-ks*Dl)*(1-S(x,y+1)/2);
pdl = exp(kd*Bdl)*exp(-ks*Ddl)*(1-S(x,y)/2);
pd = exp(kd*Bd)*exp(-ks*Dd)*(1-S(x+1,y)/2);
pdr = exp(kd*Bdr)*exp(-ks*Ddr)*(1-S(x+2,y)/2);
pr = exp(kd*Br)*exp(-ks*Dr)*(1-S(x+2,y+1)/2);

%We then rescale the probabilities such that the sum of all probabilities
%adds up to 1.
sum = pur + pu + pul + pl + pdl + pd + pdr + pr;
if (sum ~= 0)
pur = pur/sum;
pu = pu/sum;
pul = pul/sum;
pl = pl/sum;
pdl = pdl/sum;
pd = pd/sum;
pdr = pdr/sum;
%pr is not necessary as its just 1 - sum(other probs)
p1 = pur;
p2 = p1 + pu;
p3 = p2 + pul;
p4 = p3 + pl;
p5 = p4 + pdl;
p6 = p5 + pd;
p7 = p6 + pdr;
else
```

```
    p1 = .125;
    p2 = .250;
    p3 = .375;
    p4 = .5;
    p5 = .625;
    p6 = .75;
    p7 = .875;
end

    %THIS IS OLD STUFF FROM OUR ELECTRIC FIELD MODEL. PLEASE IGNORE.
   % % % %Based on the angle, we determine the p values for the
probabilitistic
% % % %movement in the main function. The numbers we made up ourselves,
saying
% % % %that there is a 70% chance of going in the desired direction, but only
a
% % % %30% chance in going in the exact direction. These can be modified,
given
% % % %a more thorough literature review and more criteria.
% % % % Suppose you want to go to the right, then the distribution is
% % % %
% % % % p(move right) = .3
% % % % p(move up-right) = p(move down-right) = .2
% % % % p(move up) = p(move down) = .1
% % % % p(move up-left) = p(move down-left) = .04
% % % % p(move left) = .02
% % %
% % % % if (th > -pi/8 && th <= pi/8)%move to the right
% % % %     p1 = .2;
% % % %     p2 = .3;
% % % %     p3 = .34;
% % % %     p4 = .36;
% % % %     p5 = .4;
% % % %     p6 = .5;
% % % %     p7 = .7;
% % % % elseif (th > pi/8 && th <= 3*pi/8)%move up and right
% % % %     p1 = .3;
% % % %     p2 = .5;
% % % %     p3 = .6;
% % % %     p4 = .64;
% % % %     p5 = .66;
% % % %     p6 = .7;
% % % %     p7 = .8;
% % % % elseif (th > 3*pi/8 && th <= 5*pi/8)%move up
% % % %     p1 = .2;
% % % %     p2 = .5;
% % % %     p3 = .7;
% % % %     p4 = .8;
% % % %     p5 = .84;
% % % %     p6 = .86;
% % % %     p7 = .9;
% % % % elseif (th > 5*pi/8 && th <= 7*pi/8)%move up and left
% % % %     p1 = .1;
% % % %     p2 = .3;
% % % %     p3 = .6;
% % % %     p4 = .8;
% % % %     p5 = .9;
```

```
% % % %     p6 = .94;
% % % %     p7 = .96;
% % % % elseif (th > 7*pi/8 && th <= 9*pi/8)%move left
% % % %     p1 = .04;
% % % %     p2 = .14;
% % % %     p3 = .34;
% % % %     p4 = .64;
% % % %     p5 = .84;
% % % %     p6 = .94;
% % % %     p7 = .98;
% % % % elseif (th > 9*pi/8 && th <= 11*pi/8)%move down and left
% % % %     p1 = .02;
% % % %     p2 = .06;
% % % %     p3 = .16;
% % % %     p4 = .36;
% % % %     p5 = .66;
% % % %     p6 = .86;
% % % %     p7 = .96;
% % % % elseif (th > 11*pi/8 || th <= -3*pi/8)%move down
% % % %     p1 = .04;
% % % %     p2 = .06;
% % % %     p3 = .1;
% % % %     p4 = .2;
% % % %     p5 = .4;
% % % %     p6 = .7;
% % % %     p7 = .9;
% % % % elseif (th > -3*pi/8 && th <= -1*pi/8)%move down and right
% % % %     p1 = .1;
% % % %     p2 = .14;
% % % %     p3 = .16;
% % % %     p4 = .2;
% % % %     p5 = .3;
% % % %     p6 = .5;
% % % %     p7 = .8;
% % % % end
end

================================================================================

%*This is one of the two auxiliary functions*

%This function determines if a person can move to point (i,j) in an m x n
%room with occupation matrix S.
function canmove = canIMoveToK(i,j,m,n,S)
canmove = 1;%Default is that they can move there
if (i <= 0 || i > m) %if they wanna go a wall
    canmove = 0; %tell them, no, they can't
end
if (j <= 0 || j > n) %likewise for the other wall
    canmove = 0;
end
if (i <= m && j <= n && i > 0 && j > 0)
    if (S(i+1,j+1) == 1)%if spot occupied by another person
    canmove = 0;
    end
end
end
```