1. **[done] when space between gate lines full, checking process of 1st gate line stop and wait.**
2. **[done] distribution of check-bag-time among all agents should be generated follow certain pattern.**
3. **[done] implement time counter in application, then improve algorithm to reduce calculating time.**
4. **[done] remove hard code and test it (stay next to leader agent feature)**
5. **[done] resume the *change gate feature*** 
   1. **[current] agents have anxiety degree(different level of time-oriented) to decide to re-consider waiting line.**
6. **[current] complete the anxiety degree feature(details.)**
7. **[current] arrange document**
8. **[next] anxiety degree of agent could has larger range, be more subtle**
9. **[next] agent currently could only change line one by one, next is to let agent directly pick the optimized option.**

**[mark:**

agent now wait until space between gate lines is free to move forward.

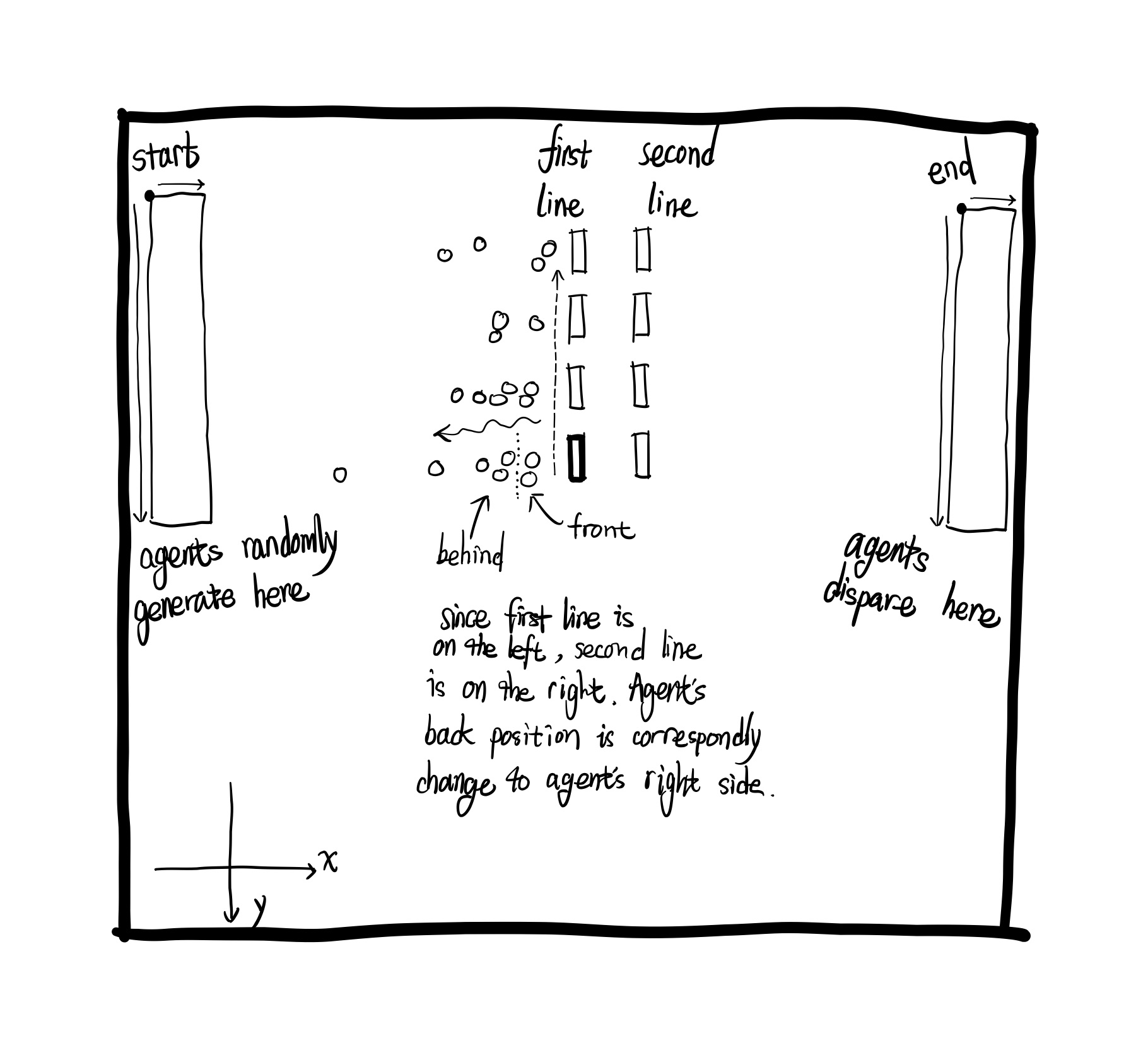
how:

* when the number of agent of the second waiting line which wait in front of the second line of gate is greater than 15, agent’s bag-checking internal timer who still in the corresponding first waiting line is paused and wait. When the number is less than 15, agent who finished the first check will move forward.

**]**

**INPUT GENERATION AND INITIALIZATION**

**Input Generator**



(notice the x and y axis is different from the regular one. y axis is pointing from top to bottom.)

The initialization of scene needs to handle multiple coordinates with different directions, in order to make user better understand the initialization of the scene, the graph above could help explain the process of initialization.

**Start and End rectangle**

For the image above, there are two small rectangles which are denoted as *start* and *end*. In this simulation, supposed all agents need to walk through an entrance to enter the scene, in this case, the start rectangle will be the entrance that agents walk into this scene, which means agent will be randomly generated within this rectangle to simulate when agent walk into the scene. What’s more, this simulation also assume agents need to walk through an exit to walk out of the scene, which work similar to the start rectangle, agents will disappear in random position within the end rectangle to simulate agent walk out of the scene.

Because the start and end rectangle only need to consider its size the position in the scene, and it doesn’t need to worry about its direction, factors that determine the initialization of those two rectangles could reduce to 2 coordinates and two pairs of length and width for each rectangle, instead of 8 coordinates in total. First, it creates two coordinates to be the most left-top corner of those two rectangles. Second, it initializes length and width for each rectangle to complete the ranges that representing entrance and exit.

**Gate line initialization**

[description about gate generation and gate line’s direction are out of date. NEED TO UPDATE.]

gateLineDirection - determine how security agent line up.

\* 1 - gates line from left to right, first line is at the top,

\* 2 - gates line from right to left, first line is at the bottom,

\* 3 - gates line from top to bottom, first line is on the left,

\* 4 - gates line from bottom to top, first line is on the right.

[the 10 security agents in input file determine gate in the scene.]

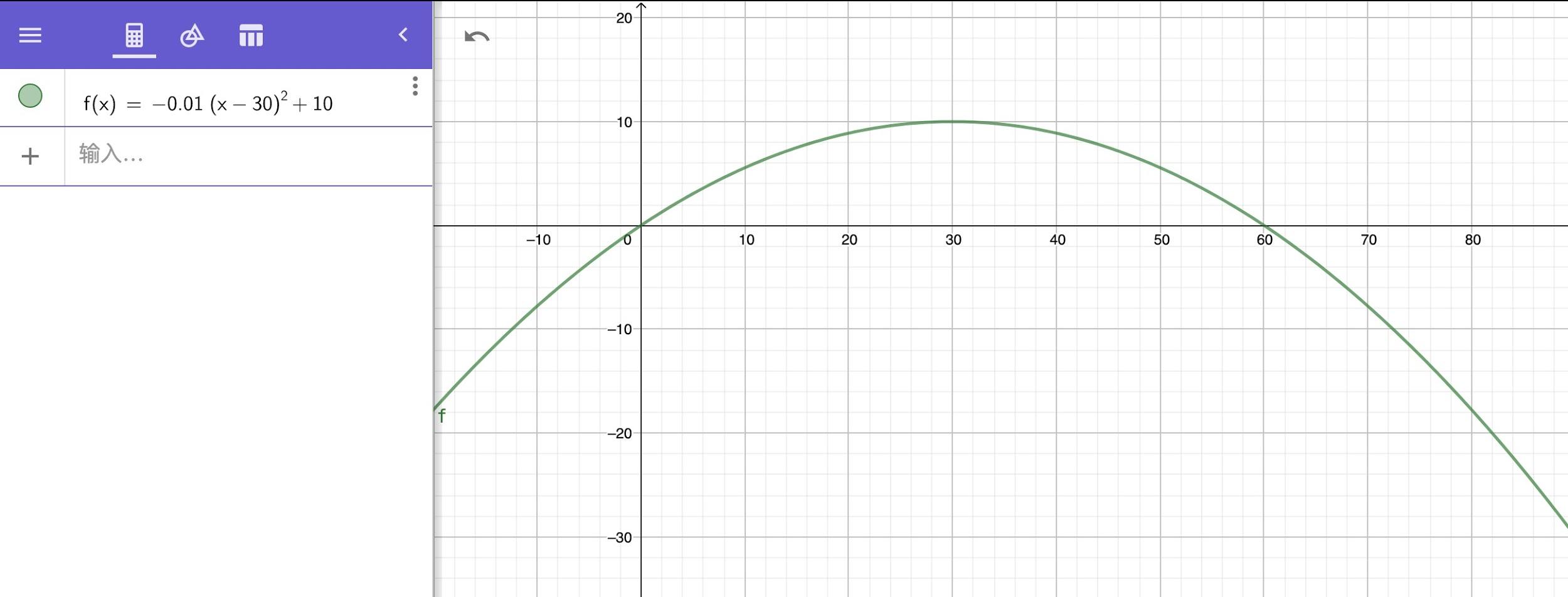
In this simulation, agents need to walk through two lines of gates. These two lines of gates could affect the direction of agent’s back position and then affect the direction the length of waiting line increase. Thus, in this case, assume there is a direction pointing from the second gate line to the first gate line.

The initialization of these two gate line is similar to initialization of start and end position. First, create a coordinate to locate first gate of the first gate line, then based on certain direction of the first gate line we create the rest of gates (the direction of gate line is pointing from the first gate to the last gate of the current gate line). For each gate in the first gate line, there is a parameter which could determine the distance between each two gates. Second, based on the direction that pointing from the second gate line to the first gate line, we could locate and create each corresponding second gate by a given distance between these two lines of gate.

**Agents Enter Distributing Pattern.**

**Version 1:**

The changing pattern of agent number that enter the scene is currently similar to the following graph.

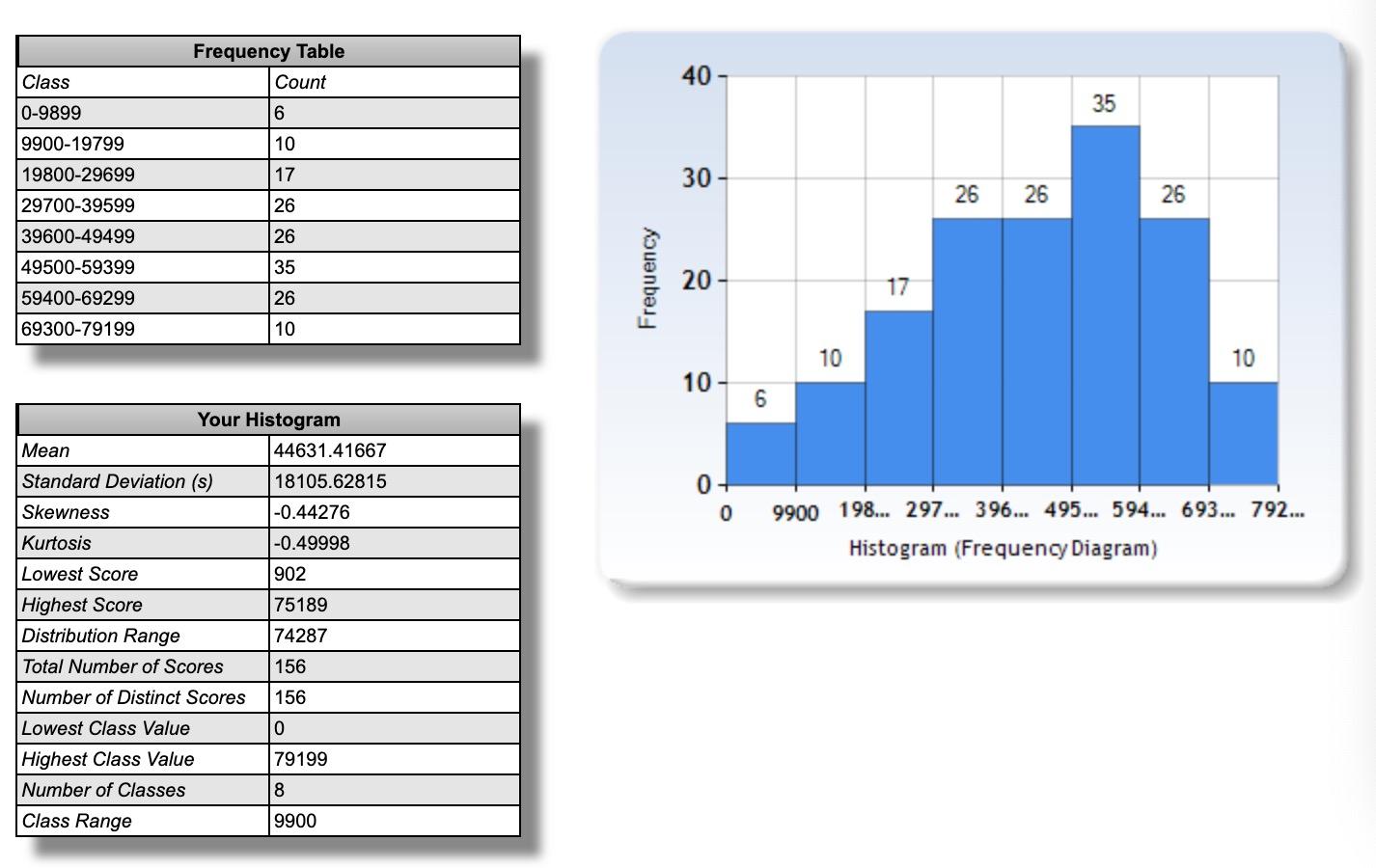


Formula:

f(x) = -0.01 \* (x - 30) ^ 2 + 10

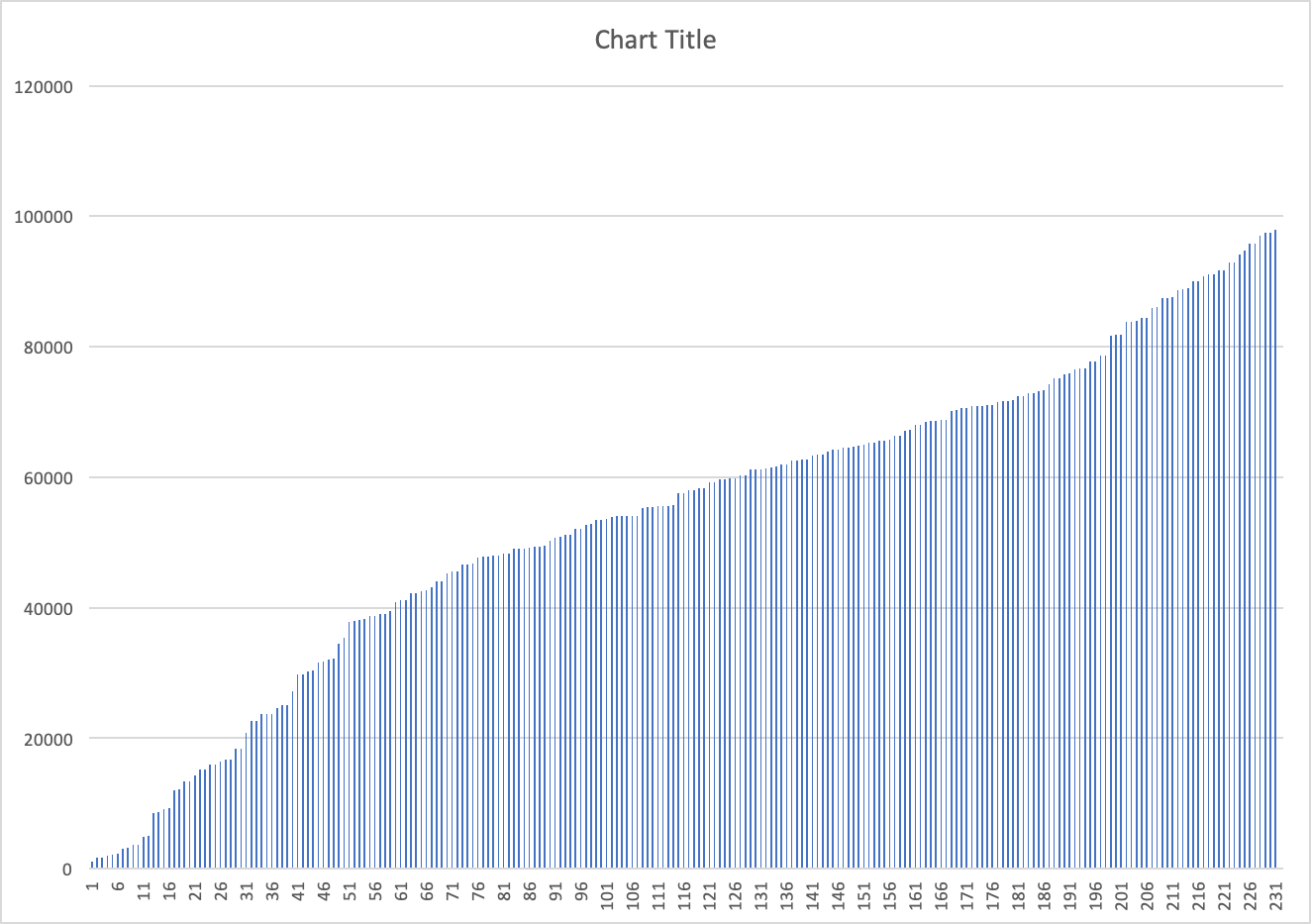
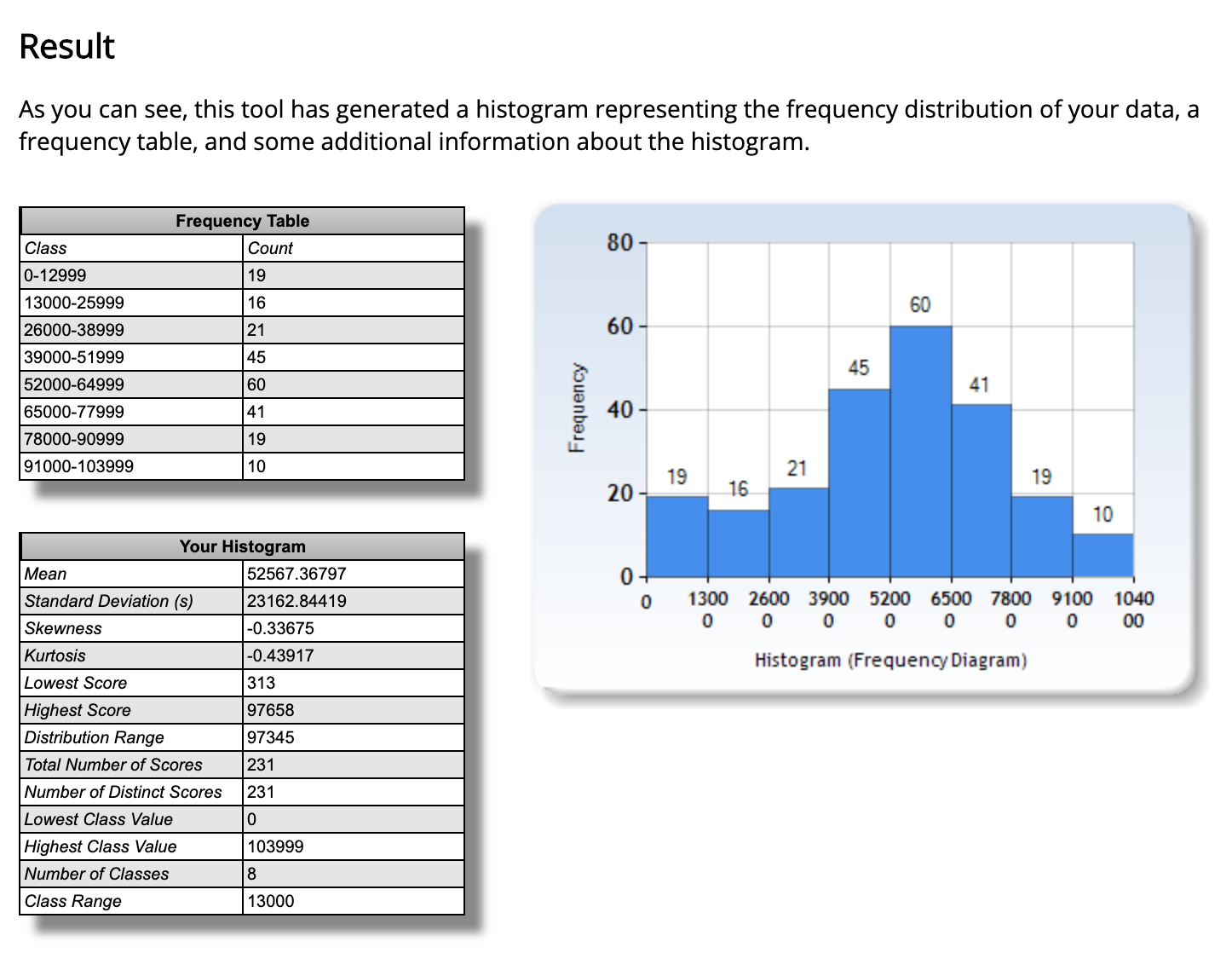
Based on the graph, the range of x is [0, 60], and the range of f(x) is [0, 10]. Supposed there has 60 seconds, for each second, there has a corresponding integer, which could be the following pattern: 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0.

In order to make the number of agent enter the scene more randomly, this approach transfer the integer mentioned above into percentage. For example, if it is 10%, then it means there is only 10% to have 4 times to be generated with a certain range of million seconds, which could represent that there have 4 agents appear from their start positions. Because there are still have certain percentage that might not generate any time, the total number of time might not equal to the required agent number in the scene. Thus user could increase the number of time it generates from 4 to 5 when it hits that 10% 's change, which means the total number of time it generated might be larger than the required one, after that, this approach evenly reduce the times it generated to make its total number equal to the required agent number.



The distribution shows the number of agents enter the scene in every 9900 million seconds.

**Version 2:**



This version of time generated approach is more simple and straightforward.

First evenly generate all agents’ scene entered time. Then sort the data. So far we assume in every certain period of time, the number of agents that enter the scene is the same. (i.e., every 2 seconds there are 4 agents entered the scene.) In order to achieve pattern that number of agents enter the scene have slightly exploded at the middle of the simulation. The input generator needs to strictly control number of agents enter the scene in a given period of time.

Given a sorted times list that represent each agent’s enter time. First, break the sorted time list into three parts. In this case, assume there are 90 times in total. Second, generate 30 times within the middle time range, and then every time the random number has been generated and inserted into the list, we delete time that in the first range or the last range. After the modification of the sorted list have been done, we sort the data again. Thus, the time list now could have the distribution pattern shown in the graphs above.

**BEHAVIOR**

**Queue-up in Pair Behavior**



Agents have follower and leader roles. In this simulation, follower always following its leader agent, and based on different situation, it could have different queue up behavior.

Pair walking period: while leader agent is walking to the gate or rear of line, follower needs to stay next to the leader and both are facing the same direction.

Queue up period: not matter leader or follower reaches the empty gate first, both follower agent and leader are set to waiting status, then both area added into the waiting line, leader agent is added into waiting line first, then is the follower. Thus, in the waiting line, the position of leader is always ahead of its follower agent.

Agents in waiting status which means already added into waiting line are maintained by a specific function. Since only one agent could do security check at a time, if the first two agents in head of waiting line are in pairs status, if both agents are required to do bag checking, leader agent always check first, then the follower. these two agents need to break into two individual agents to pass the security gate by lining up one after another, instead of staying side by side. Thus, in the simulation, agents in head of waiting line are in pair status will stay side by side. the first three agents of the waiting line (here the length of the waiting line is larger than or equal to three) are always line up one after the other (no matter agents are in pairs relationship or not.). After the third agent, agents are in pairs will have pair queue up pattern. If agent is single agent, the way it queues up at the end of line could have two different behaved pattern. For the upcoming agents (not in waiting line yet), the status of the rear of the line determines what queue up pattern should be. (details shown below)

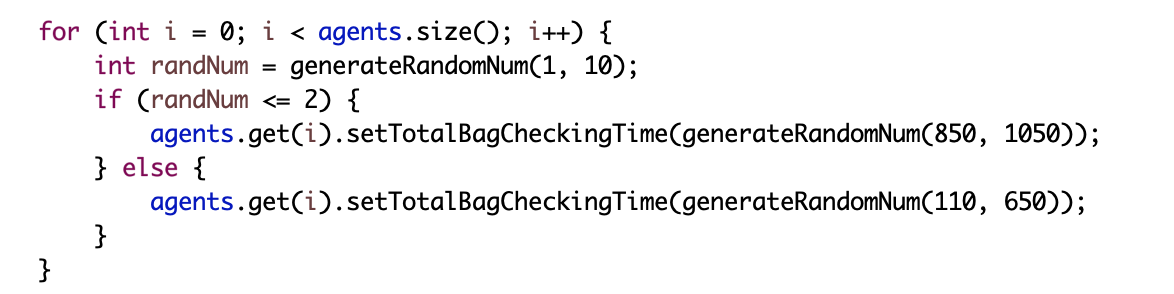
Since agent in the waiting line could only belong to one of two types: single agent or pair agent. For pair agents, in each pair agent, there are two types of agent, leader and follower. Follower always follow its leader and both are walking side by side. Thus there is an internal behavior system between the follower and the leader. The leader always leading to their destination, and what the follower does is staying next to leader agent, even they already queue up in the waiting line. For single agent, the internal behavior system is itself.

Since agent that belongs to different type has different internal behavior system pattern, by combining them agents the waiting line could have the following waiting pattern: (1) pair agents followed by pair agents, (2) pair agents followed by single agent, (3) single agent followed by pair agents and (4) single agent followed by single agent.

1. **: :** (pair agents followed by pair agents)  
   Both pair agents have follower and leader. Leader behind will line up at back position of the other leader agent. The followers agents will stay next to their own leader agents. (Here follower agent only stay right side of its leader agent with random distance [each pair’s distance between leader and follower could slightly different].)
2. **: ·** (pair agents followed by single agent)  
   If the single agent lines behind the pair agents, the upcoming single agent lines up at back position of the middle place between leader agent follower.
3. **· :** (single agent followed by pair agents)  
   If pair agents queue up behind a single agent, leader agent will go to back position of the single agent, and the follower will stay next to the leader agent.
4. **. .** (single agent followed by single agent)  
   While single agent line up behind the other single agent, it directly go to the other single agent’s back position.

**Agent checking process**:

Agent have internal timer to simulate how much time they need to finish bag-checking and body scanning. During agent initialization, each agent was assigned a pair of random values that denoted as the amount of time it needs to spend to finish bag checking and body scanning. (the value that represent bag checking time could present the size or the amount of stuff agent carried that need to check). Since each agent has to walk through two gates, each gate costs different time because the checking purpose are different, one is to check what agents carry in their bag, one is to use metal detector scan agent’s body. Therefore, the time spending patterns between two gates are different. What’s more, since not everyone carry bag, agent who has no bag could directly walks through the first gate, thus the time each agent spends in the first gate also follow certain pattern. (current there has 20% of agents need to take 850 to 1050 (*how many ms it needs still need to be calculated*) to do bag checking.)



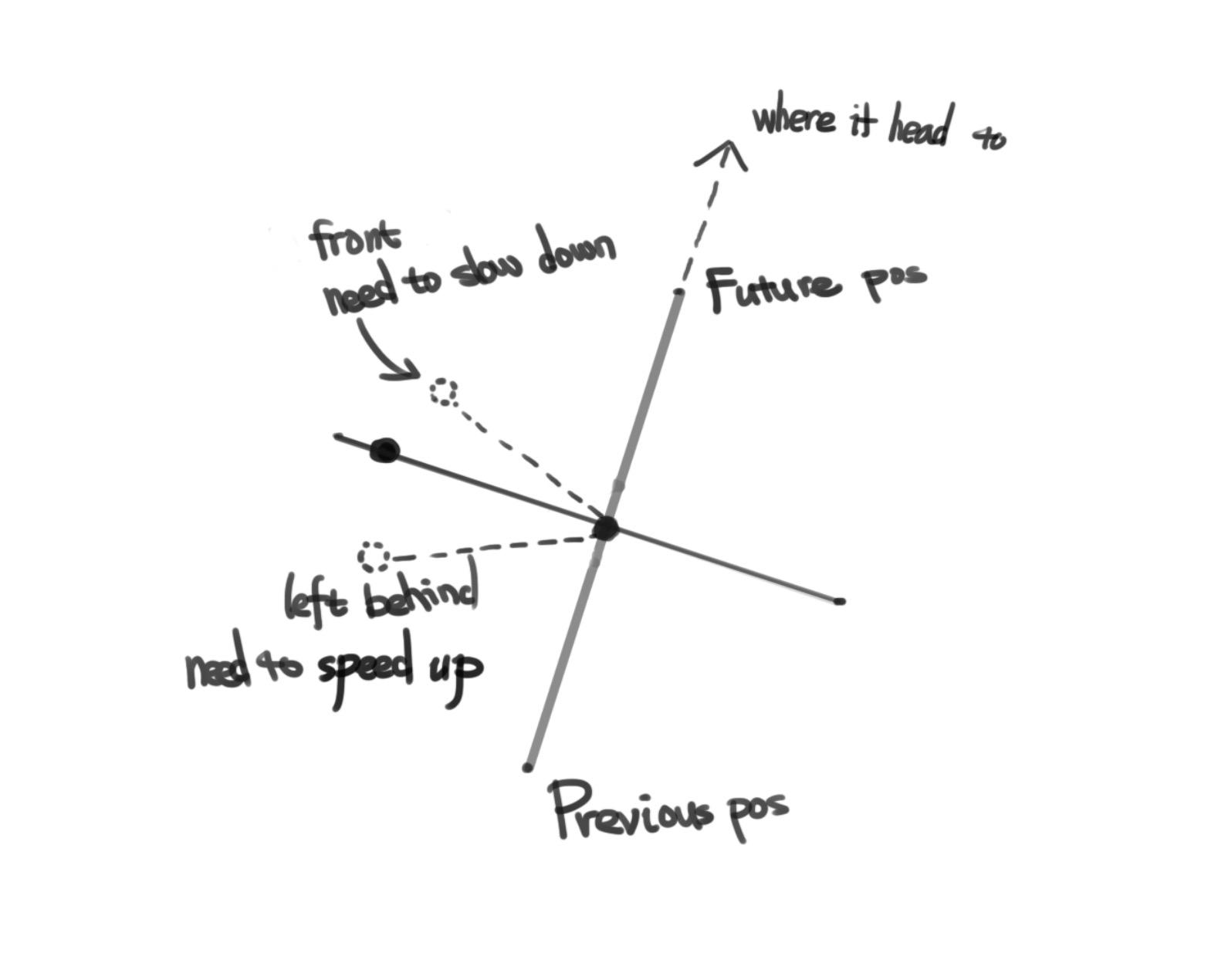
Not every agent has to do check in first gate, but every agent needs to do check in second gate, thus the space between first line of gates and second line of gates is easy to get full. When space between gate lines is full, agent who finishes the first check might need to wait until it has empty space to let it move forward.

**How agents walk in pairs**

Since pair agents start at the same position at the same time and have the same destination, agents in pair will basically have similar path.To achieve pair walking pattern, during walking, follower agent will adjust its speed to make follower agent stay left/right side of leader agent so that agents could walk next to each other (follower agent: fol, leader agent: le).

Version 1:

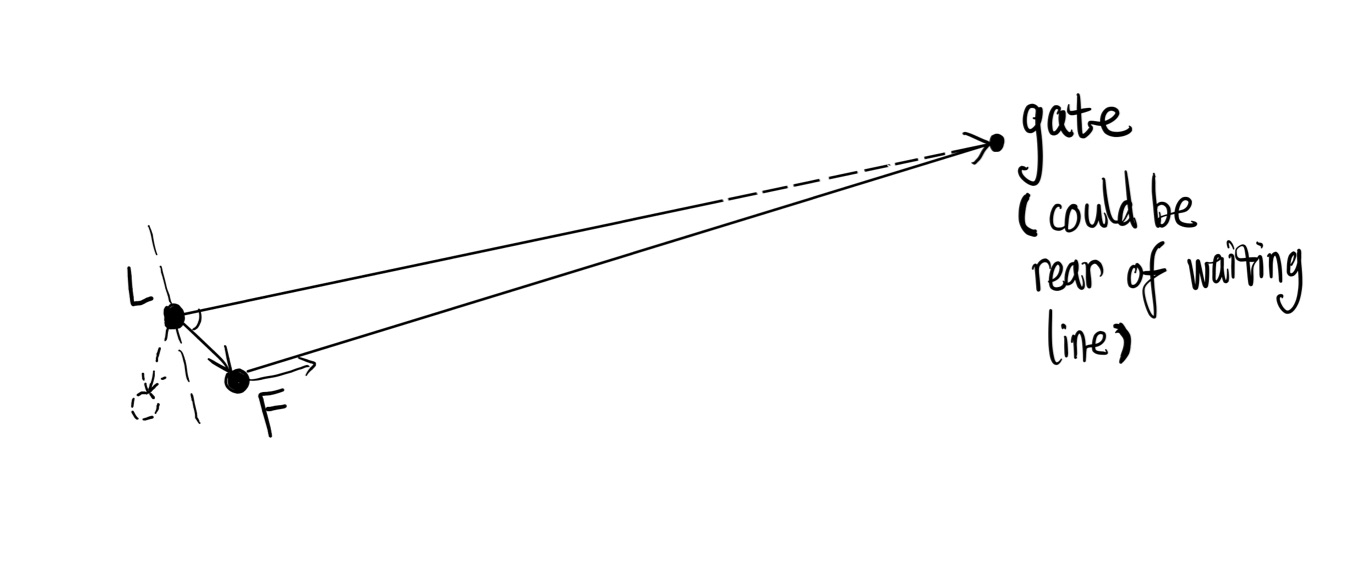
Every agent stores information such as coordinate(x, y) of previous position in every certain period of time once it enters the scene. By implementing formulas for vectors, it could predict where the agent could be in the future by rotating agent’s previous position 180 degrees up to the front. Then there have an angle between two vectors: one is the le agent’s current position to the future position, the other one is the le agent to it’s fol agent. Then the program could adjust the fol agent’s speed base on the angle between these two vectors. There have three cases need to be concerned:



Version 2:

Instead of keeping track of leader agent’s previous position and its future position, we keep track of leader agent’s current target position, which could be rear of waiting line or checking gate. In order to make follower agent successfully adjust its speed, we use a triangle formula: using two vectors to calculate the angle between these two vectors to determine how much follower.

Distance method



Agent’s speed needs to be modified, thus front position of the leader agent changed.

Follower agent based on angle between two vectors to update moving speed.

*vector 1*: **leader\_agent** to **leader\_agent’s\_current\_target\_des**

*vector 2*: **leader\_agent** to **follower agent**

Follower agent still use the same strategy to update speed. However, instead of rotating leader’s previous position 180 degrees to get the leader's future position to be head-part of *vector 1*, currently we use leader’s next\_destination as head-part of *vector 1* (here “head-part of vector” means the arrow part of the vector). After knowing these two vectors, we could use formula below to calculate the cos(Angle) between these two vectors to adjust follower’s speed.

cos formula changed

cos formula now is

\* Formula 1

\* distance of vectors are v1, v2 and v3.

\* (v3 ^ 2) = (v1 ^ 2) + (v2 ^ 2) - 2 \* v1 \* v2 \* cos(Angle)

\* cos(Angle) = ((v1 ^ 2) + (v2 ^ 2) - (v3 ^ 2)) / 2 \* v1 \* v2

After calculating the cos(Angle), we use the formula below:

speedChange = 1.8 \* 03 ^ x (*x is cos(Angle)*)

to calculate the follower agent’s speed, also we set maximum limit of speed to 2.1 (the max value) could be changed based on what user needs).

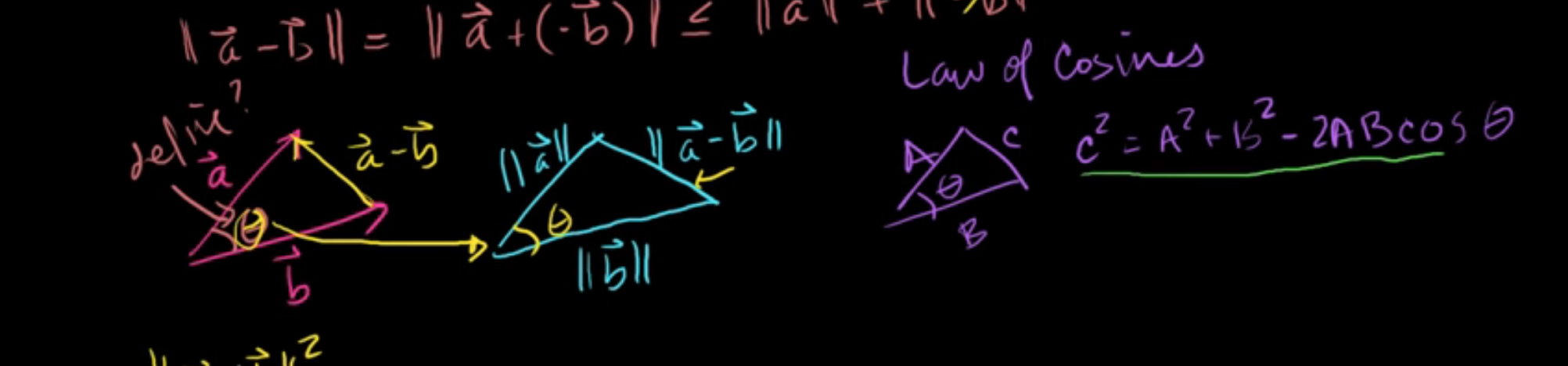
Since leader agent is moving, if it wants to pass through all the checking process, it has several state, including line up state, walking to the gate state, walking to the rear of line state, etc. Because when leader agent is in line, follower always stay next to leader agent, which is handle by waiting line maintenance method, this case will not be considered. We only consider case when agent is walking, then follower adjust its moving speed. Choosing the next\_destination as the head\_part of vector can be more stable than predicting the future position by rotating the previous position. However, agent will always reaches the current target place, once agent current position and next\_destination is within a certain limit, the next\_destination is updated. (why need to update next\_destination: if leader reach the next\_destination, it returns the wrong result if use the distance between cureent\_position and next\_destination to calculate the cos result.)

How it updates (not include case in the waiting condition):

* current agent could have the following next\_destination:
  + gate1:
    - when it close to gate1, update to gate2
  + gate2
    - when it close to gate2, update to end destination
  + [“close”: within certain distance]

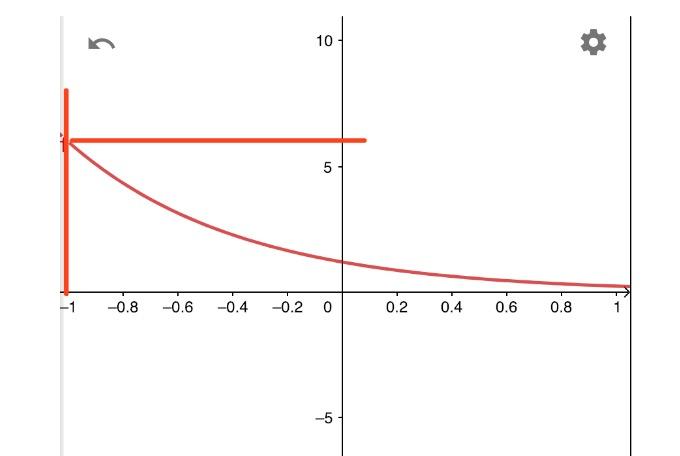
1. Angle is 90: since the angle between two vector is 90, the fol agent is relatively left or right side of the le agent. The fol agent doesn’t need to change it’s moving speed.
2. Agle is less than 90: when the angle is less than 90, it means the fol agent is relatively in front of le agent. Then the fol agent starts to slow down.
3. Agle is less than 90: when the angle is larger than 90, it means the fol agent is left behind by le agent. Then the fol agent starts to speed up.

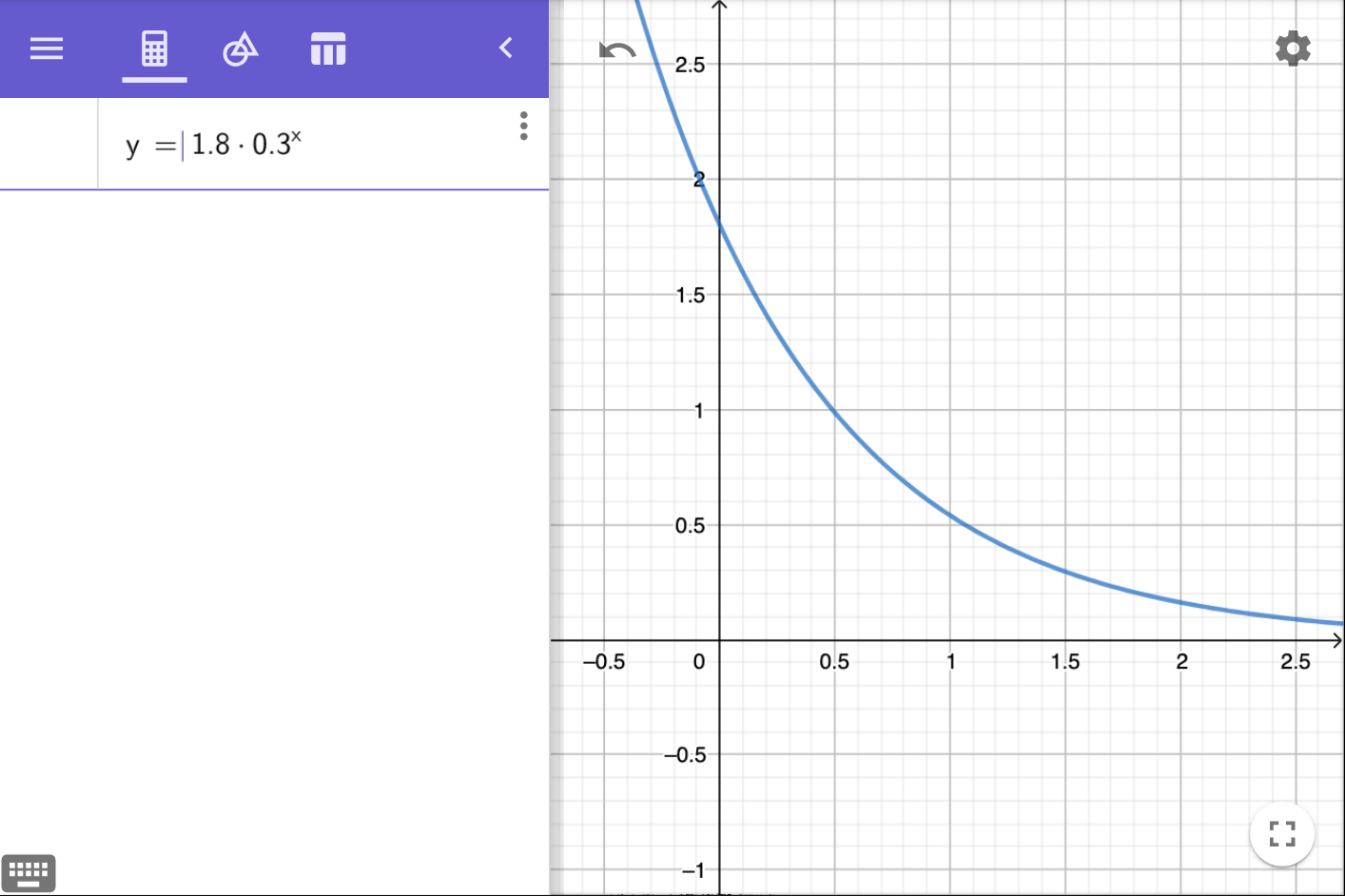
How to calculate the angle:



Since the degree of angle is the only one factor that determines the fol agent’s moving speed, this program simply use a curve equation to adjust fol agent’s moving speed. Suppose *A* is the angle between two vectors that just mentioned above. Then range of cos(A) is [-1, 1], which is the x axis. When cos(90 degree), *A* is 90 degree, cos*A* is 0, speed of fol agent doesn’t change, otherwise, fol agent change speed to wait for le agent or speed up to catch up with the le agent.

Below is a graph of the curve equation:





**Re-consider behavior**

Agent makes change waiting line decision based on *anxiety degree*.

Each agent has internal anxiety monitor to keep monitoring and updating the anxiety, agent based on the anxiety degree to make change line decision. Agent in higher anxiety is more likely to change line than agent in low anxiety.

Anxiety update interval:

* Every 5600 million seconds, agent in waiting line updates its anxiety.   
  For every waiting line, anxiety update is always start from the tail. It will check the left and right side of the current line (or if agent currently at the most left/right line it will check right/left side).
* Next, agent retrieve its index value in the line (list), then compare the index value with the size of the other lines. If agent’s index value is larger than size value of other lines, agent might increase its anxiety (it has certain percentage to increase the anxiety, in this approach, the percentage could be: 100/135 to 95/130 → 73% to 74% (*percentage might need to change for further update.*)).

So far, each agent has 4 degrees of anxiety: 1, 2, 3 and 4.

Agent with high anxiety will do:

* Look for shorter line and then leave the current line and queue up the new line.
* Single agent leaves on it own, pair agent leave together.
* After successfully change line, agent anxiety go downs

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

1. **Reasons to decrease:**
   1. Agent at line that shorter than lines nearby
   2. Agent’s current waiting line proceed relatively faster than other (***waiting line continue moving in short 2000ms seconds***)
   3. Once agent have successfully changed to its wanted line (line up a new desired line), its anxiety degree drop to **1**.
   4. Agent in a relatively short line.
   5. **When the number of upcoming agent to target line reaches certain number, anxiety degree drop.**
2. **How it decreases:**
   1. Since anxiety of every normal agent itself has 50/50 chance to be modified. So:
      1. agent with high anxiety degree(>= 3) is less likely to decrease its anxiety degree (15% less), so it has 35% to increase anxiety.
      2. agent with low anxiety degree(<= 2) is more likely to decrease its anxiety degree (15% more), so it has 65% to decrease anxiety.

From above **1.b 5000ms** to **3.b 2000ms** we got waiting-line-progress-intervals-formulas:

* *x* < 2000ms → line is moving fast → satisfied, anxiety degree down
* 2000ms < *x* < 5000ms is normal → nothing happen!!
* 5000 < *x* → line is moving slow → unsatisfied, anxiety degree up up up.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

POTENTIAL PROBLEM:

1. when shorter line exist, all agent with high anxiety might leave.

**DATA OF ANIMATION**

* 40 frames per million seconds
* 4500 million seconds in total
* each frame one iteration of each agents

**CODE EHANCE**

**after *MaintainQueue* method improve (ms)**

* before: 27693, 32781, 40193, 36202, 35832, 33540, 51994, 46427 = 38082.75
* after: 25113, 28334, 29407, 25489, 24440, 29070, 30610, 26156 = 27327.37

**BUGs**

1. Pair agents follower and leader initialized with different gates(it could happen)

**VIDEO NOTE**

**Observation from video**

Agents not only walk in pairs, but also line in pairs. When they are lining, follower could be on the leader agent’s left or right side.

Crowd number didn’t explode too much. People line up casually: people will line up to shorter queue once they notice there has one, but people talking to each other, some of them might not notice there has a shorter queue. Thus, the crowd appearance explode can be slightly increase.

**Current Problem (limitation):**

1. Crowd appear pattern:
2. Width/length of agents’ start position generated range need to similar to width/length of 1st gate line, otherwise, the majority of agents will go to one or two gates that near to their start position, which means make crowd difficult to form a waiting line, because too many agents rush to those one or two gates to make “re-consider” didn’t work well and make it difficult to form waiting line.

**TO DO**:

1. Empty/not-empty state of gate is changing quickly. Agent keep detecting whether the empty gate exist or not, because the empty/not-empty state of gate shift so quickly, other agents will act weird because the state of it’s target gate changes.  
   Sol: Agent who obtains a new empty gate and wants to move to it need to not just look at the state of gate is empty or not, agents who want to change gate also need to check how many agent almost or near the gate that the state have just changed to empty.
2. Agent pay more attention to objects in front at smaller distances than side object. Pay more attention to people in the field of vision is 90.
3. Consider
4. Imply *“****Personal Reaction Bubble*** *- Realistic Modeling of Agents in Crowd Simulations”* to agents in waiting line (realistically adjust distance between each other).