**Crowd Simulation Application**

MS Project Report

Zhicheng Chen

Understanding complicated crowd behaviors is essential to urban designers and architects. However, grouping a large amount of people to do experiment is dangerous and unrealistic. Among these, design and create an application which could correctly represent crowd behavior is crucial. This project report describes an implementation called *Crowd Simulation* that aim at creating realistic, unique and dynamic crowd by takes agents’ data as input and outputs the result in animation form.

**Introduction**

The increasing proportion of people living in urban areas brings new challenges to urban planning and architecture. Crowd simulation plays an important role in addressing these challenges. With the help of crowd simulation techniques, urban designers and architects could determine the evacuation time of a massive crowd, predict the behavior of a crowd flow inside of a building or prevent overcrowding during certain events.

A crowd forms when a large amount of people gathers in a limited space. Simulating the whole crowd as a single unit could help understand the behavior of the moving crowd. However, if we divide the crowd into groups that contains 2 to 3 people or individuals, the behavior of the crowd can be more realistic. In a group, people know each might walk together. Previous researcher Reynolds [1] proposed a steering approach known as Leader Following (LF). This approach involves pair agents where the “follower” agent follows the leader and stays on its side. This disadvantage of this approach is that in this basic steering approach, the leader agent does not wait for its follower agent if the distance between these two agents is too large, which is not realistic.

More recent simulations of crowds of people use more complicated calculation. For example, previous approach [2] designs agent as ellipses that have a sense of the environment and plan their own path ahead of time to avoid agent collisions. Unfortunately, the output of this kind of simulation lacks realism and flexibility. Since it does not involve dynamic behaviors such as allowing agents to move in and out of different group or queues based on agent’s desire, agents who have planned a path ahead of time might end up in the longest waiting line without being able to switch. In reality, people do not just stay in their waiting line once they choose it, they might need to change waiting lines if there is a better option.

**Related Work**

relate work…

**Report Outline**

This report presents important components of this application which are open source Recast & Detour. Then we discuss intuitions behind the real-life video that are essential in understanding the output produced by QueueBehaviorApp. Also, we present details about the scene initialization and agent initialization. Then we talk about scenario that we want the crowd to behave and strategies we implement to achieve the goal. We present the strategy of simulation results evaluation. At last, we conclude with future work discussion.

**Resources**

Java Port of Recast & Detour navigation mesh toolset

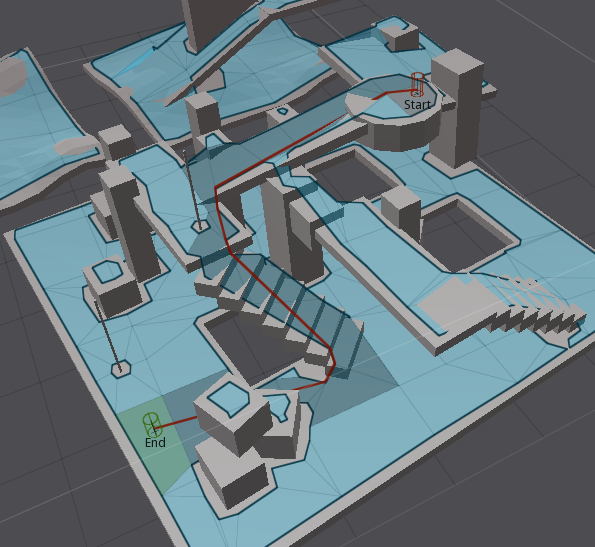
(ref: <http://masagroup.github.io/recastdetour/index.html>)

**Recast**

Recast is a state-of-the-art navigation mesh construction toolset for games. Recast is an open source which could automatically provide you a mesh at any level geometry in instant time; Recast could also be customized to achieve user’s specific purpose.

**Detour**

Detour is a spatial reasoning toolkit which accompanies with Recast to offer a simple static navigation mesh. DetourCrowd is a crowd management module offers features for agents handling and behavior customization. Detour allows user to create lots of agents and move agents in navigation mesh. What’s more, Detour allows user to create customized behaviors that determines agents how to move and react.



**Related Work**

Blabla

**CenturyLink Notes**

We record crowd at CenturyLink on different entrance during events such as concert and Disney Ice Event to study how people gather, walk, form waiting lines and passing security processes. We collected and analyzed common behavior features that crowd might have and implemented those data into every aspect of our application:

* Pair walking, pair line up.
* Queue up behavior.
* Form waiting lines.
* People switch from a long line to shorter one.



**Observation 1:**

Normally, people attend events with their friends or family, thus people in the crowd gather and form small groups. People walk as a small group will constantly stay side by side. If one of the people left behind, people in front will stop somewhere and wait for partner to catch up. Based on the video, this behavior happens a lot of time: people who finished the security check will stand at somewhere in front and wait for his/her partner.

**Observation 2:**

When waiting line is formed, people simply queue up and slowly move in the line. Because people might walk with their companies, they form waiting line that each row could have one or two people. Based on video, waiting lines could have different length, we found out people in the long waiting line will change to the shorter line or people will directly change direction and walk to the new gate if they find out there has a new empty gate. However, for some people, they might be less interested in the shorter line and just stay at their original line.

**Observation 3:**

For people who reach the security gate, they have two security processes to finish before entering the building. For the first process, security faculties will check the ticket and bag (if someone carries bag to the event). People show the ticket to the faculty, then they could quickly pass through the first gate. For people who carries bag, security faculty need to take a few seconds to check the bag, thus they will stay more time on the first gate. For the second process, security faculty will use handheld body scanner to scan people one by one. Because everyone takes almost the same process, the time people finish the second process is evener than the first process.

**Foundation architecture and Scenario**

In this project, I will develop a crowd simulation application which aims at creating realistic, dynamic and accurate crowd. To achieve this goal, I will use an open source state of art navigation mesh construction toolset called Recastnavigation to achieve static avoidance and shortest path calculation. What’s more, I will also utilize a path-finding and spatial reasoning toolkit Detour to achieve dynamic avoidance among agents in the path and to completed calculation of each frame of the simulation [3]. Using these open source platforms, I will build the lower level of my approach – QueueBehaviorApp.

*Scenario*: Single agents or pair agents are randomly generated from the virtual entrance; each agent is initialized with a default start position and end position, and they will walk from start to end position. However, before agents reaching their destination, every agent has to finish security check first. (People do ticket checking and security check during the concert event). Thus, every agent needs to stop near the security to simulate the security process. Since the new agents are generating and security process takes time, the number of agents in the scene increase dramatically and the crowd form. However, instead of generating a massive chaotic crowd, agents in the crowd will queue up orderly and form several waiting lines, and each agent in line will do the security check one by one. After finishing the security check, agents will move to their default end position and depart.

**Implementation, Challenges, and Output**

**Input Data Initialization**

Input file allows us to determine the basic scene information such as agent id, agent start time, start position, end position and behavior mode. Instead of directly applying data from the real-life video, we manually generate agent data based on features we got from the video. We define agents in pair relationship by letting agents have the same enter time, start position and end position.

In order to achieve the crowd features, we design input data base on the following patterns:

Agent id, enter time, start position, end position, behaved mode

* Each agent has its unique id number.
* Agent’s enter time determines when agent will walk into the scene.
* Start position determines where agent will appear.
* End position determines where agent will exit, it also determines which direction agent will go when it appears.
* Enter time, start position and end position determine agent’s relationship.
* Agent’s behavior mode determines how agent behaves – queue up to form line, flee or simply walk from start position to end position. In this report, we mostly focus on queue up behavior.

Below is the data input sample:

30,3656,-60.255486,0.31802097,-5.320471,44.077248,0.318020731,1.1289825,queue

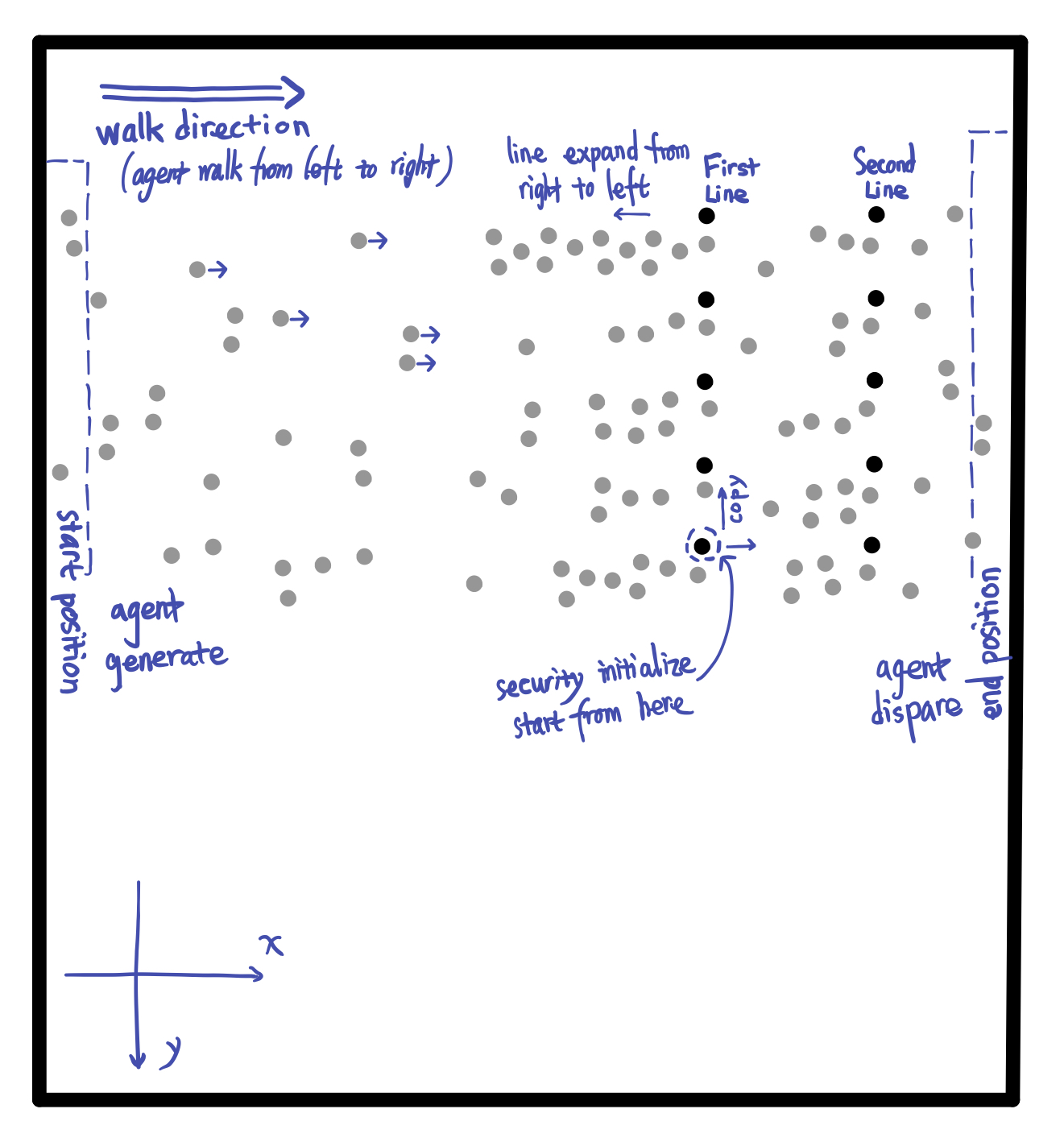
31,3661,-59.755486,0.31802097,-4.820471,44.577248,0.318020731,1.6289825,queue

32,4734,-61.710487,0.31802097,11.044155,44.371113,0.318020731,6.9663258,queue

Blue is agent id; each agent has its unique id; green is agent’s enter time; *3656* means agent enters the scene at 3656 million second; orange is start’s start position (x coordinate, z coordinate, y coordinate); red is start’s end position (x coordinate, z coordinate, y coordinate); purple is agent’s behavior mode.

In the simulation, agents who stand next to their corresponding gate are representing security faculty in the real-life video. Since our video are mostly recorded on a hall that only allows crowd walking from one place to another place, we decide to define agents’ start position and end position within a limited range to simulate the hall environment. What’s more, agent’s walking direction determines the expand direction of waiting line. For example, agent walking from left to right, when it reaches the tail of waiting line, it lines up at the tail, then the waiting line grow from right to left.

Here is a demo to help us better understand how to determine range of start position, end position and security gate lines.

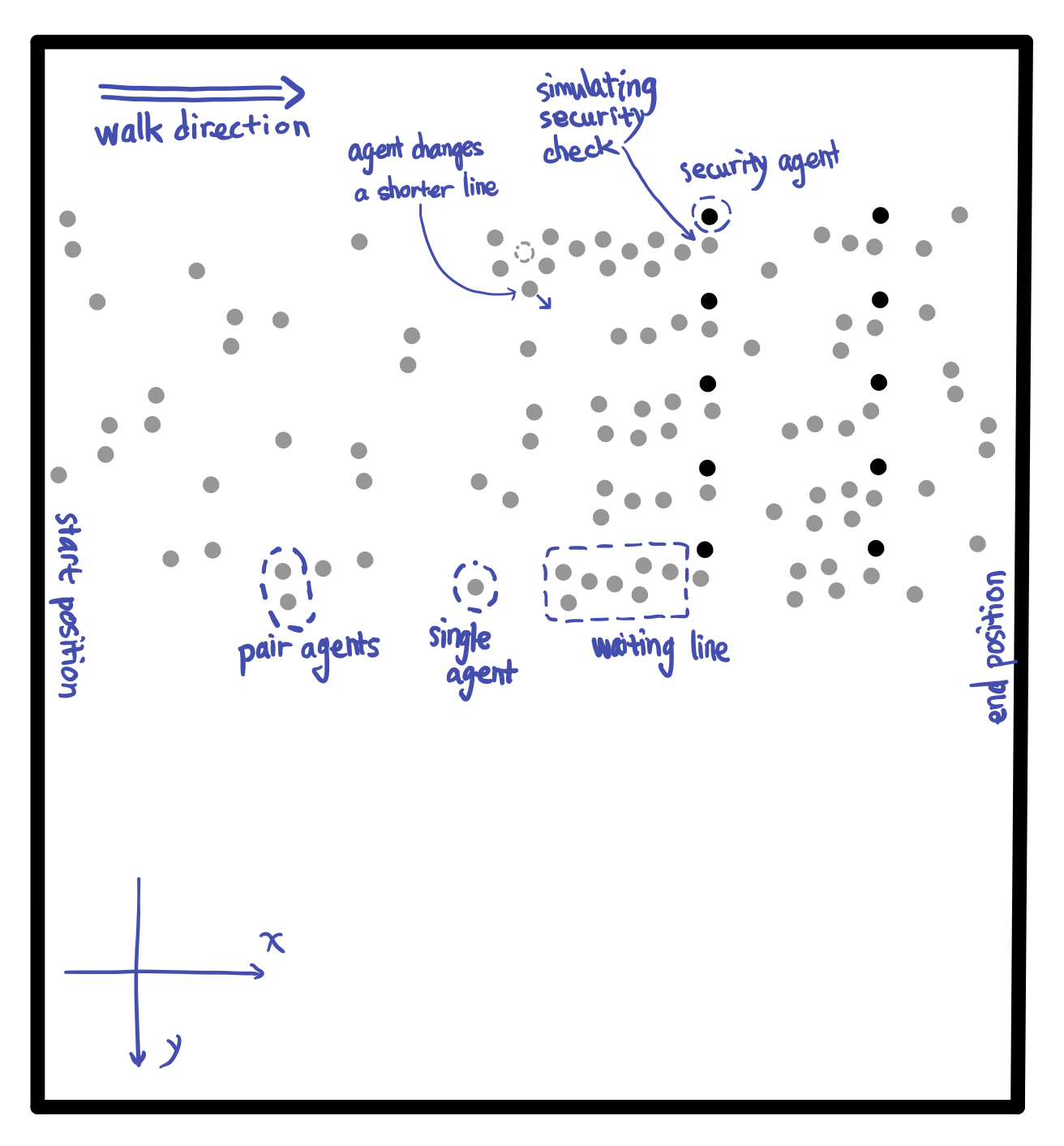


Each cycle represents an agent, agent in black is security faculty, agent in grey is normal agents. Faculty agent are specifically initialized, and they are always the first ten agents in the scene. Agents walk in from the rectangle called start position and walk out of scene called end position. Without security gates, all normal agents simply walk from start position to the end position.

**Environment Initialization**

Before achieving the complicated crowd behavior features such as pair walking, queuing up and form single/pair waiting lines, we need to initialize environment initialization so that agents could perform security check. Before reaching the end position, every agent is required to pass two positions to simulate security checks. Based on the real-life video samples we recorded during events, people have two checking process to finish before they enter the event, one is the ticket and bag checking, another one is body detector scanning. Having the input data is not enough, to make agent move naturally, we need more factors both on environment and agent itself.

At the beginning of the simulation, we will first initialize environment based on the input data that we mentioned above. Image on the below is the mockup demo that demonstrates agents’ environment.



Code below are always run at the beginning of the simulation.

**if** (*count* == 0) {

initGates(10, 0);

initAgentCheckTime();

initAgentGateOption(agents, *GateAll1*);

initFrdRelationship (agents);

initAgentsAnxiety();

*count*++;

}

Based on the code above, descriptions below basically introduce the initialization of the environment and default status of all agents:

* ***initGates*** – In this scene, we create 10 checking gates and split these gates into two lines. Then we initialize ten faculty agents represent security faculties standing next to the gates. All security faculties will stand next to its own checking spot during the simulation.
* ***initAgentCheckTime*** – In real-life video, people have to stop at first checking gate show its ticket and stop at the second gate to do security body scanning before entering the building. For certain people who might carry bag, the security faculty in first gate might need to check that bag and people need to stay more time than people who just bring the ticket. In the simulation, each agent is randomly assigned two values that represent times they need to stop to finish the ticket/bag check and body scanner check. For the value of ticket/bag checking time, because the number of people carry bag is relatively smaller than the number of people who don’t, the initialization of agents’ first gate checking time must achieve this pattern. The value of second gate checking time among each agent will be relatively even.
* ***initAgentGateOption*** – In this simulation, agents are randomly generated on the left side of scene, each agent choose the closest gate to go.
* ***initFrdRelationship*** – Every agent is whether in individual status or pair status. Based on the input data of every agent, we will pair up two agents that have the close enter time, start position and end position. Each agent in pair relationship plays different role, one agent is leader, another one is follower. For individual agent, they are neither leader nor follower.
* ***initAgentsAnxiety*** – In real life, people at tail of a waiting line might not be satisfied with the length of its own line and they will seek opportunity to switch to the other shorter line. To achieve this behavior feature, we allow each agent to have feeling of anxiety. Anxiety degree updates based on the agent’s current position in waiting line. Agents are more likely to change waiting line with higher degree of anxiety.

**Appearance of Upcoming Agents**

In the simulation, in order to create a realistic crowd, we intensively control the flow of crowd and let it follows certain pattern. At the beginning of the simulation, the number of agents is relatively small, but when the simulation continue, the number of agents appear in the scene increase and reaches the peak. After that, the number of upcoming agents slowly decrease.



Charts above illustrate the number of new agents appear every 1320 million seconds. The bar table shows the change of number of new coming agents during the simulation. During time range from 5280 to 6600 million seconds, the number of new coming agents reaches peak.

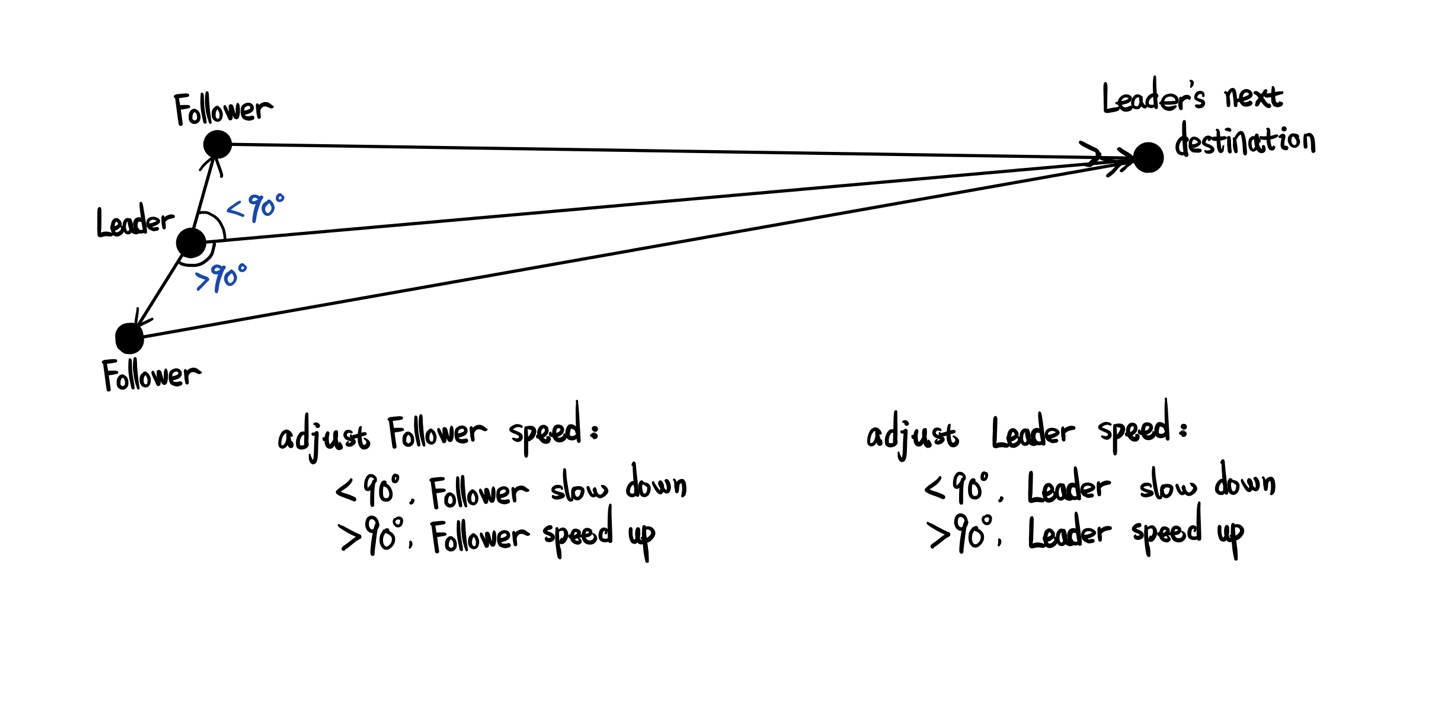
**Agent Behavior Implementation**

**Pair Walk Behavior**

In the real-life video, people in group are talking to each other while they are walking, so they need to stay side by side and sometimes they need to slightly adjust their speed to stay in pair. In order to achieve the natural pair walking pattern, we let agents in pair to be follower or leader. Leader is the one leads them to their common destination; follower always adjust its speed to catch up its leader. However, not only change speed of follower, we also allow leader to adjust its speed to wait or catch its follower. For example, in the simulation, if the leader finished security check, instead of directly moving to the next destination, leader will stop somewhere behind the gate and wait until its follower finish the check.

How to Achieve Pair Walking?

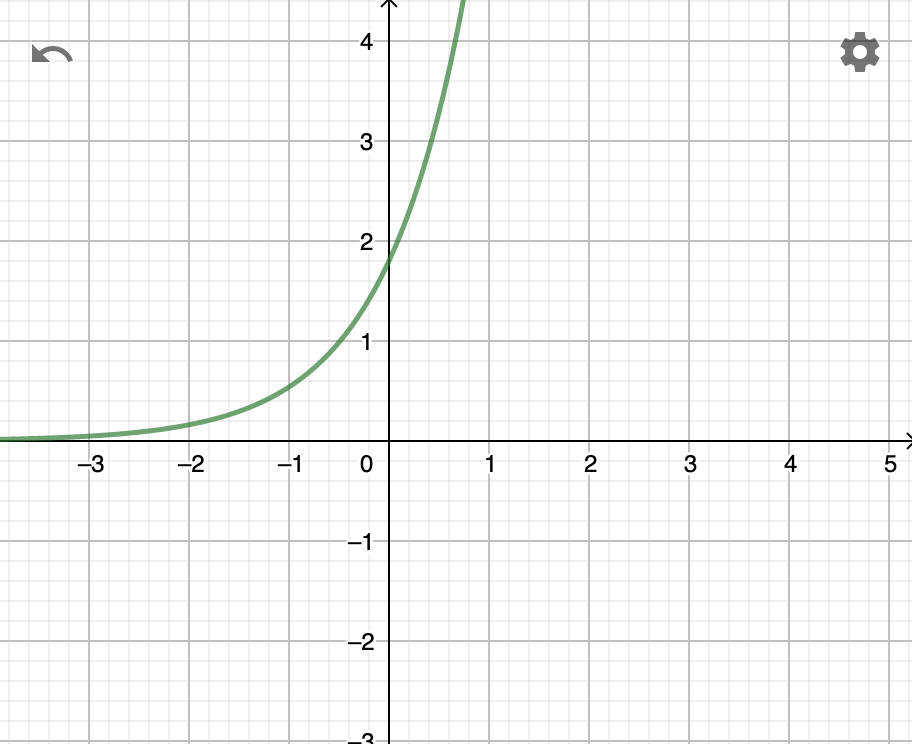
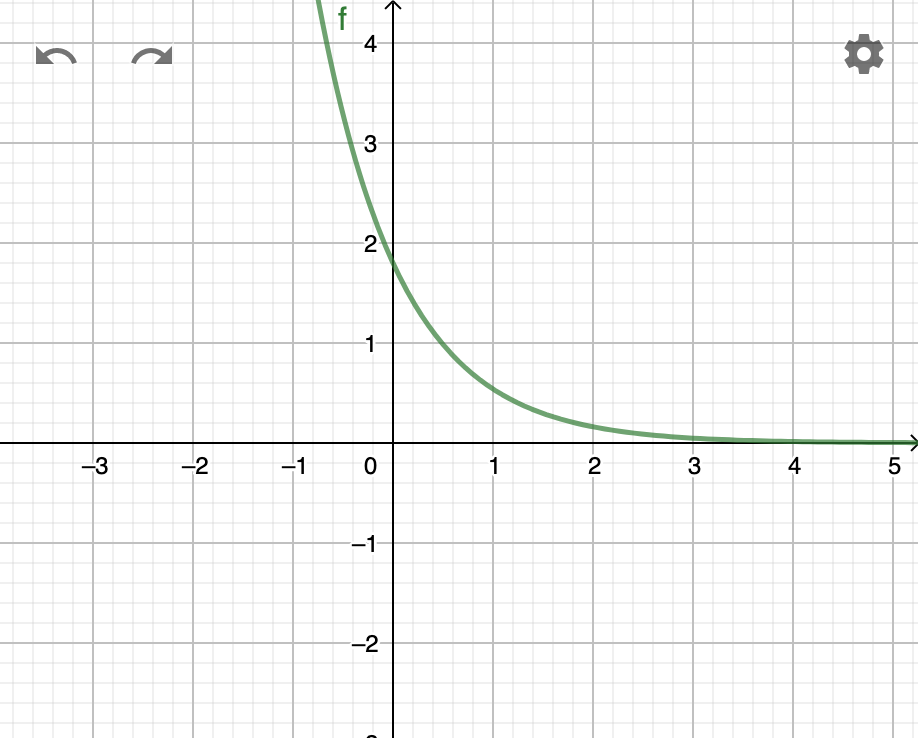
In order to adjust follower’s moving speed, we need to keep track of angle of two vectors: one is from leader’s current position to leader’s current destination. Another one is from leader’s current position to its follower’s current position. After implement low of cosines and these two vectors, we could get angle between these vectors.



Bases on the value of angle, we allow agent’s speed has three states:

1. Angle is 90: the follower is exactly left or right side of the its leader. The follower agent doesn’t need to change it’s moving speed.
2. When the angle is less than 90, it means the follower is in front of its leader, the follower starts to slow down.
3. When the angle is larger than 90, it means the follower is left behind by its leader, the follower starts to speed up.

However, knowing the angle between two vectors is not enough to create the natural speed adjustment. In the simulation, we use a simple curve equation to help us deal with this problem. [**这里需要考证一下**Suppose A is the angle between these two vectors, then we know the range of cos(A) is [-1, 1], and the base is x axis. Thus, from two charts below, we could know from range while x is between -1 and 1, y value is between 0 to.]

f(x)=1.8\*0.3^(-x) f(x)=1.8\*0.3^(x)

**Queue up Behavior**

In order to enter the event, people need to do security check, so they will stop at security gate for a few seconds. However, because the upcoming people are continue moving to security gates, people who wait behind will form waiting lines. To simulate this queue up behavior, we allow each agent to have the following status:

* isWalking – agent is walking.
* isWaiting – agent lined up.
* isChecked – agent finished security check.

These 3 states help determine behavior of the agents. When agent is in *isWalking* state, agent is either walking to the gate/line or walking to the end position. Once agent lines up or for specific agent who could directly reach the empty gate, agent immediately set to *isWaiting* state. If the agent is leader, then the state of its follower is automatically set to *isWaiting*. For agent who is in *isWaiting* state, they are either waiting after certain agent or doing security check. In this case, we design a queue maintenance function to handle each agent in the queue.

Inside the queue maintenance function, in order to let agents maintain the queue, we let each agent in waiting line temporally set the back position of agent in front as its current destination. For agent who is head of the waiting line, it will start count the internal checking time. Once the value of checking time is 0, agent finishes the security checking and continue moving forward to the next destination. Since each agent’s value of time checking is different, we will see different agent take different time to finish checking. Then agent behind it will replace the head position and start the checking process.

When agent finished check, its state will change *isChecked*.

Because agents in pair relationship will walk in pair, they might still stay side by side even after queuing up. Thus, inside the waiting line, conditions between agents could have the following statuses:

* **. .** – individual agent queues up behind another individual agent.
* **. :** – individual agent queues up behind pair agents.
* **: .** – pair agents queue up behind individual agent.
* **: :** – pair agents queue up behind another pair agents.

As we mentioned above, follower agent is set to stay next to its leader agent. Thus, in the waiting line, unlike the other agents that their current destination is set as back position of agent in front, the follower agents’ current destination is set as left/right side of its leader. Then when individual agent queues up behind pair agents, it will set its current destination to middle back position of the pair agents.

**Re-consider Behavior**

**Evaluation**

**Future Work**

Waiting line is form by obstacle such as fence.

Acknowledgements???

**References**

[1] Reynolds,C.:Steeringbehaviorsforautonomouscharacters.In:GDC,pp.763–782(1999)

[2] Baig, Mirza Waqar, et al. "Realistic modeling of agents in crowd simulations." 2014 5th International Conference on Intelligent Systems, Modelling and Simulation. IEEE, 2014.

Test Cases???