**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. In order to create realistic and trustworthy crowd simulation result, I will design a sophisticated crowd simulation algorithm that allows agents to achieve complex features. The features I will implement will be obtained from data collections and direct observations from video records in real-life events.

A crowd forms when a large amount of people gathers in a limited space. Simulating the whole crowd as a single unit might help understand the flow movement of the moving crowd. However, each people in the crowd are independent. In order to achieve this feature, we will divide the crowd into groups that contains 2 people or individuals, the behavior of the crowd can be more realistic. In a group, people know each other might walk together. Previous researcher Reynolds [1] proposed a steering approach known as Leader Following (LF). This approach will also involve 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 complex 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 a situation such as an unnatural long waiting line in the scene where several shorter lines exist. However, in reality, people do not just stay in waiting line once they choose it, they might need to change waiting lines if there is a better option.

**Related Work**

Many simulations have achieved the complicated behaviors. Julio Godoy [Julio Godoy] provided dynamic agent base approach that agents in the scene could have distinct goals to plan their own movements and collision avoidance ability. Interactions among each agent are being “polite” and natural after agents learned optimal strategy in the given simulated environment. Carmine [Carmine] extended state-of-art predictive approach with the social awareness, prediction and social collision avoidance to achieve the prediction in social path following behavior. Social awareness is signaling agents that are approaching each other, then agents adjust its behavior and direction for the future social interaction could improve the realism. In the simulation, simply allow the agents to have interaction, repulsive forces and given priority are not enough. In order to simulate social interactions among agents in the crowd, Walk Along Steering [Walk Along Steering] developed the Walk Along Steering allows agents in small group have six social steering behaviors that makes agents’ movement and reaction smoothly and naturally. Based on three-layer architecture controlling motion of IVAs designed by [Reynolds], agents could achieve patterns like following, avoidance, waiting and approaching. To allow the crowd has more dynamic social behaviors, sai-keung’s [sai-keung] approach presents a crowd simulation that involves behaviors that agents are more interactive. In the simulation, two or more agents are required to perform actions simultaneously to finish certain tasks, agents are divided into two categories: workers and pedestrians. Tasks can be assigned to worker agents, workers will cooperate with each other and create complicated behaviors by decomposing a complex task into numbers of simple task.

**Report Outline**

Next section, this report presents important components of this application which are open source Recast & Detour. Then we discuss observations and collections of real-life video records that are essential to understand 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 goals. We present the strategy of simulation results evaluation. At last, we conclude with future work discussion.

**Resources**

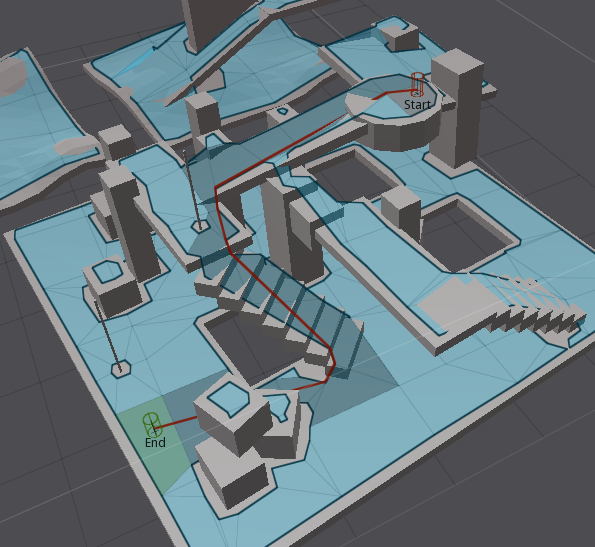
Java Port of Recast & Detour navigation mesh toolset [recast&retour]

**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.



**Observations from Video Records:**

Previously, in order to understand how crowd move, we recorded crowd videos at CenturyLink on different entrance during events such as concert and Disney Ice Event to study and analyze how people gather, walk, form waiting lines and passing security processes. We collected and gather common behavior features that crowd might have and implemented those features 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 – Pair Walking**

Normally, people attend events with their friends or family, thus people in the crowd are divided into numbers of small groups. Because people know each might have conversation while they are walking, the ongoing people in a small group are more likely to stay side by side. If one of the people in the group is left behind, people in front will stop somewhere and wait until his partner catch up, or if people realize themselves are left behind their partner, they will speed up to catch up is partner. Based on the videos, this behavior happens a lot of time. For example, people who finished the security check will stand at somewhere in front and wait for his/her partner.

**Observation 2 – Queue up Behavior**

When waiting line is formed, people simply queue up and slowly move forward in the line. Because people might walk with their companies, they might form waiting line that each row could have one or two people. Based on videos, 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 moving to the shorter line and just stay at their original line.

**Observation 3 – Security Checking Behavior**

For people who reach the security gates, 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). For people just bring the ticket, 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 people’s bag. Thus, people who carry bag 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 difference among people need to finish the second process is more consistent 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 two security checks first. (People do ticket checking and security check during the concert event). Thus, every agent needs to stop near the security gate 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 orderly queue up 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 need to manually generate agent data based on features we observed from the video. For example, we will define agents in pair relationship by letting agents have the same enter time, start position and end position.

In order to achieve more crowd features, we will 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 agent’s walking direction.
* Enter time, start position and end position determine agent’s relationship.
* Agent’s behavior mode demonstrates agent’s behavior
  + Queue - means agent will queue up to form line.
  + Flee - means agent will choose the shortest path to walk out of the scene.
  + None - means agent will simply walk from start position to end position.

Below is the data input sample:

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

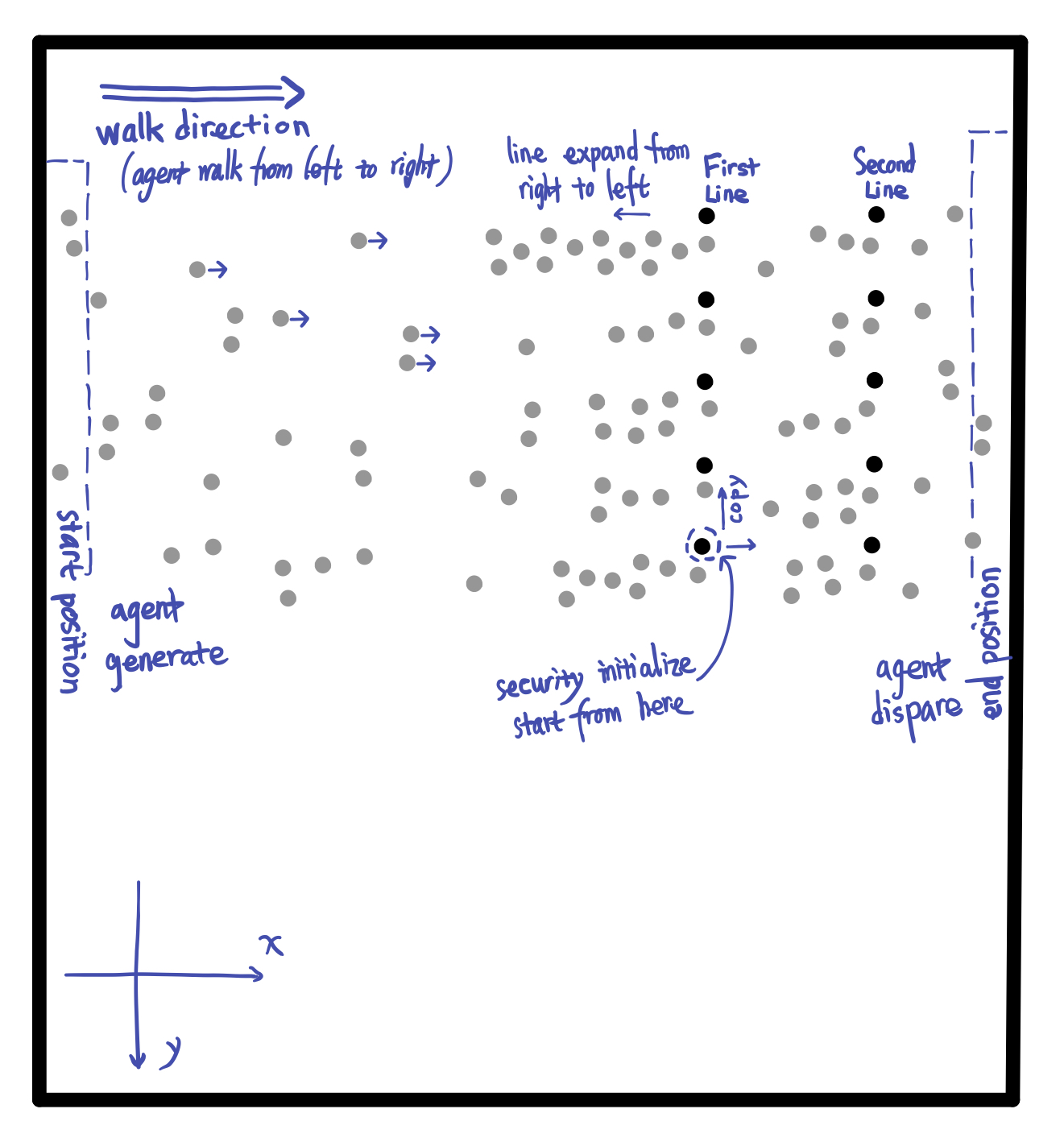
31,3661,-59.755486,0.31802097,-4.820471,44.577248,0.318020731,1.6289825,queue

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 the start position (x coordinate, z coordinate, y coordinate); red is the 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 this 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.

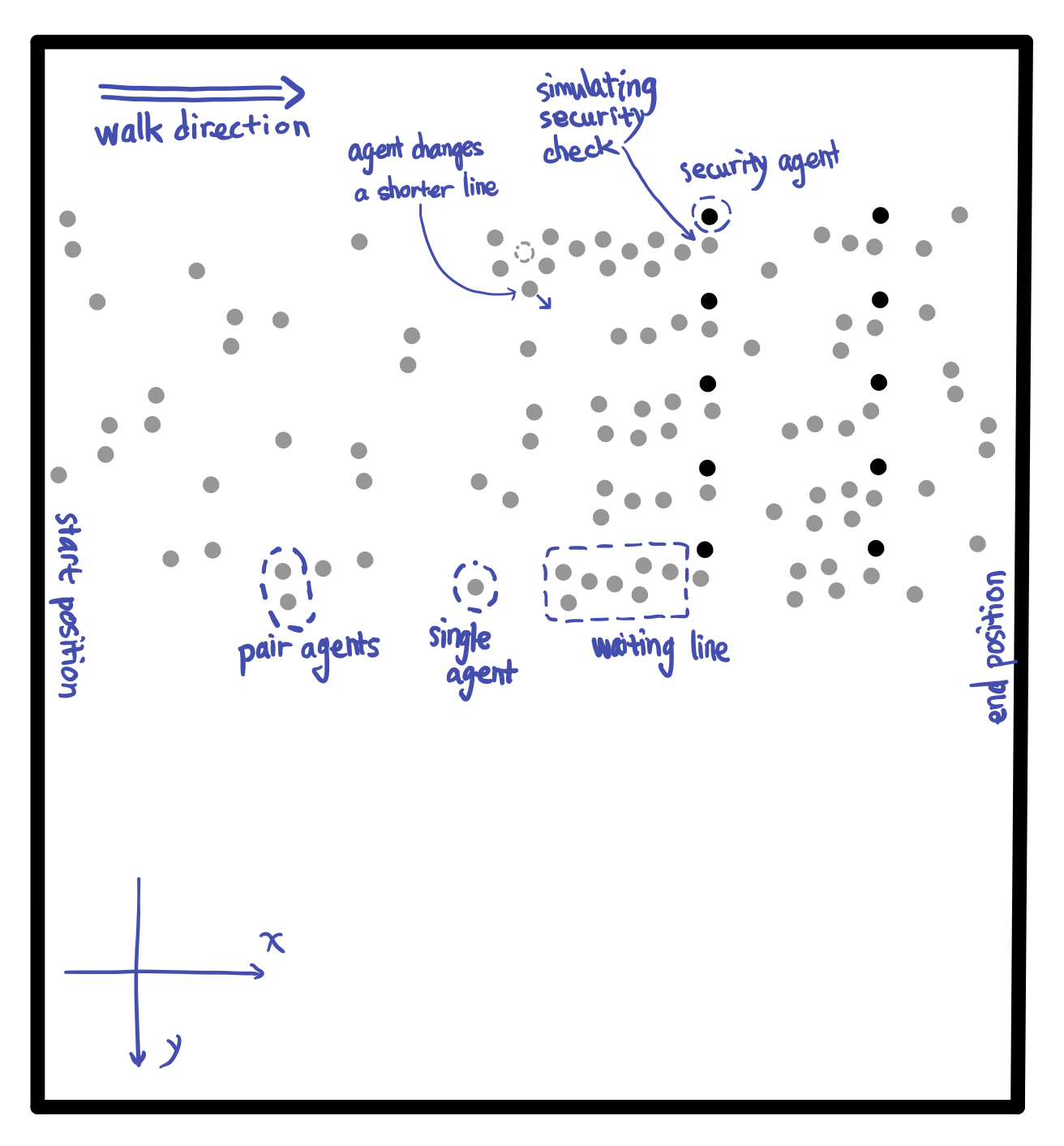


Each cycle represents an agent, agent in black is security faculty, agent in grey is normal agents. Faculty agent are specifically initialized, and they will stay at the same position during the simulation. Agents walk in from the rectangle on the left called start position and walk out of scene on rectangle on the right called end position. Without security gates, all normal agents simply walk from start position to the end position. However, in this case, agents will do security check and queue up.

**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 to demonstrate simulation result.



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

initGates();

initAgentCheckTime();

initAgentGateOption();

initFriendRelationship();

initAgentsAnxiety();

Methods above cover all the initializations required in the simulation. Descriptions below will talk more about details of the initialization:

* ***initGates*** – In this scene, we will create 10 checking gates and divided these gates into two lines to represent two checking processes. Then we initialize 10 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*** – As we mentioned above, each people have two processes to finish before entering the building, one is ticket and bag checking, another one is body checking. For people who might carry bag, their checking time at first gate is longer than people who just bring the ticket. Thus, in the simulation, each agent is randomly assigned two values that represent times they need to stop to finish the ticket and bag check and body scanner check. For the value that represents ticket/bag checking time, because the people carry bag are randomly appear and the number of them is relatively smaller than the number of people who don’t, the initialization of agents’ first gate checking time will follow the corresponding pattern. However, for the value that represents the second gate checking time. Since everyone has the same checking process, the values among each agent will be more consistent.
* ***initAgentGateOption*** – In this simulation, agents will be randomly generated on the one side of the scene. Then, based on each agent’s current position, they will choose the closest gate to go and move forward.
* ***initFrienddRelationship*** – Every agent is whether in individual status or pair status. Based on the input data, 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 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 their own line and then they will seek opportunity to switch to the other shorter line. To achieve this behavior feature, we allow each agent to have anxiety. The agent’s anxiety degree updates based on the its current position in waiting line. Agents are more likely to change waiting line when they have higher degree of anxiety.

**Appearance Feature 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 continues, the number of agents appear in the scene will start to increase and then reaches the maximum. After that, the number of upcoming agents will slowly decrease.



Charts above roughly 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 the maximum.

**Agent Behavior Implementation**

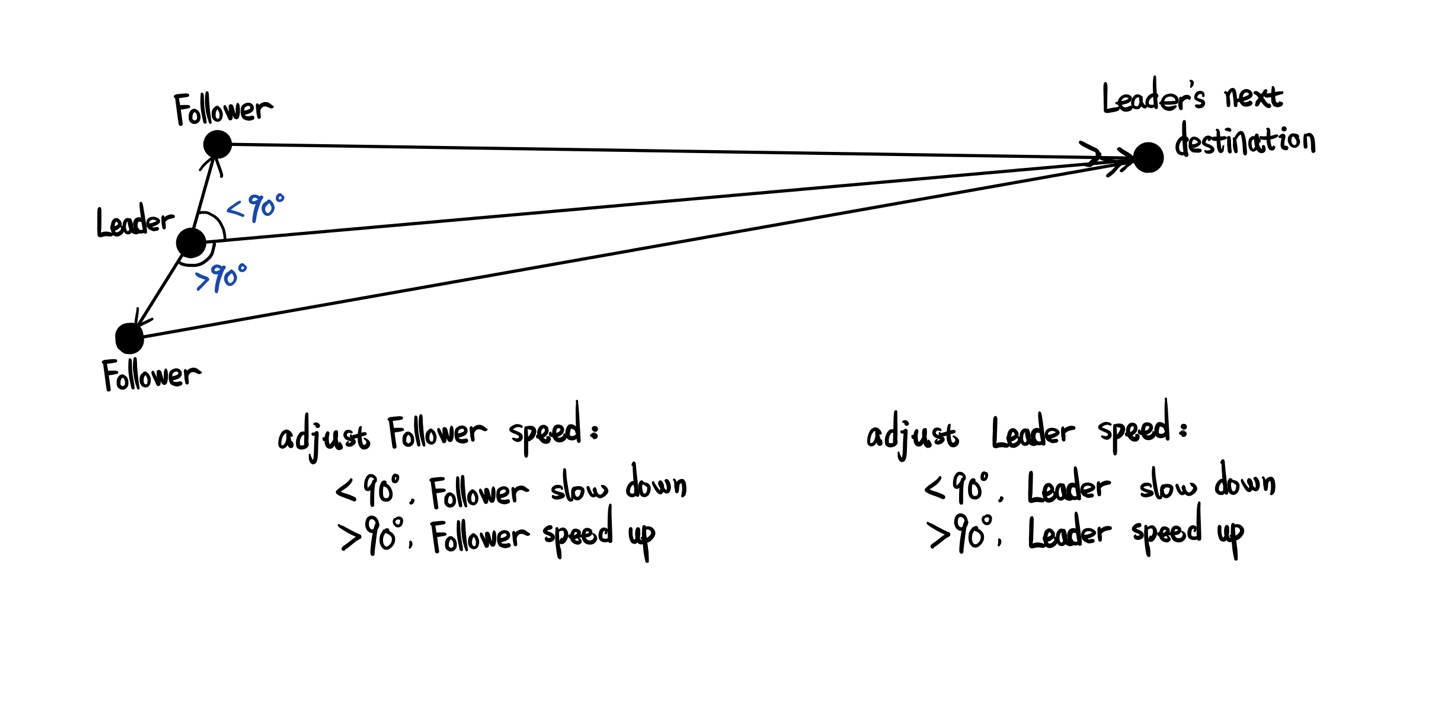
**Pair Walk Behavior**

In the real-life video, people in group are talking to each other while they are walking. To maintain this status, 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 will be the one leading them to their shared destination; follower will 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 first, instead of directly moving to the next destination, leader will stop somewhere in front and wait until its follower finish the check.

【这里：要如此这么细节的讲How？】

How to Achieve Pair Walking?

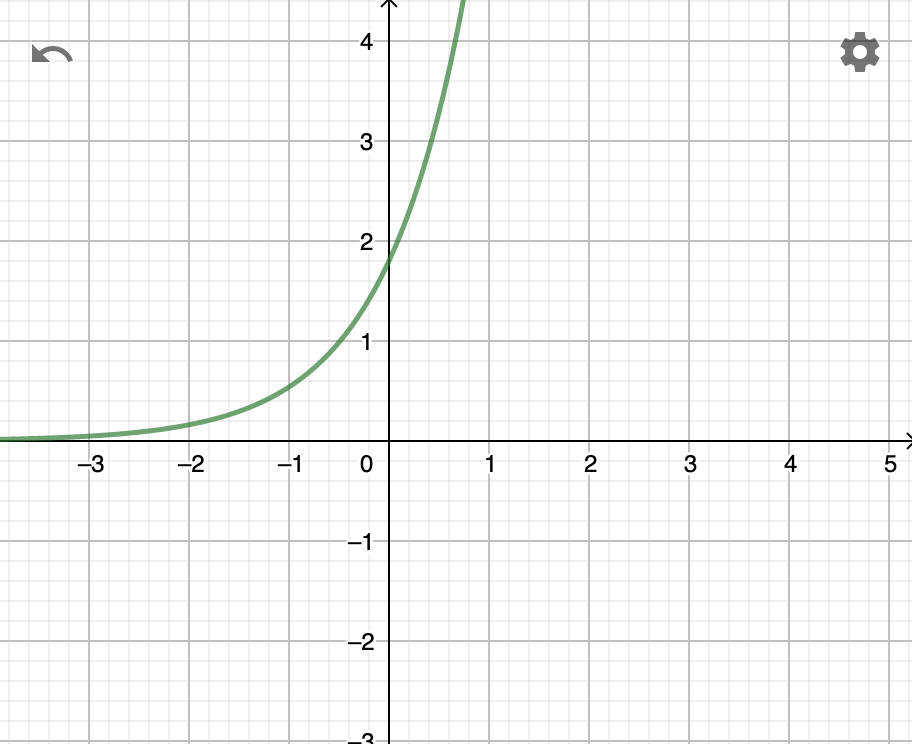
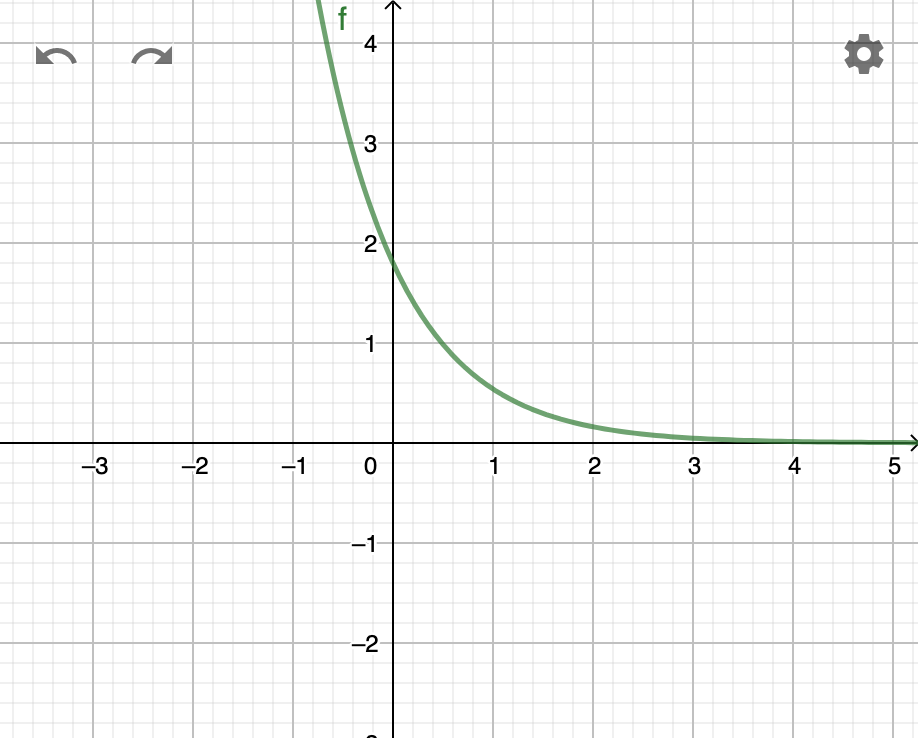
In order to adjust follower’s moving speed, we need to keep track of angle of two vectors: one is from leader’s current position to leader’s current destination. Another one is from leader’s current position to its follower’s current position. After implement low of cosines and these two vectors, we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

Once agent enters the scene, each agent is assigned a value represents its anxiety level. From real-life video we record, we find out people will switch to other waiting line if they have a better option. However, instead of switching line, we also find out there are certain numbers of agents prefer to stay in their own line. Thus, we assign anxiety to each agent to simulate how agents make the switch line decision.

Assigning anxiety level to agents are not enough to simulate the reconsider behavior, we also have anxiety monitor to adjust the anxiety degree based on the waiting lines conditions. Agents based on the anxiety degree to change line decision, agent in higher anxiety is more likely to change line than agent in low anxiety. Each agent has 4 level of anxiety degree. Anxiety degree update follows 4 steps: (1) agent queue up in waiting line. (2) agent check the length of left and right side of current line. (3) based on the difference between the current line and other line, anxiety degree is correspondingly updated. (4) if agent is satisfied with its length of waiting line, agent’s anxiety degree decreases or remains. If agent is not satisfied with the length of its waiting line, agent increase the anxiety degree.

Once agent’s anxiety degree reaches the maximum value and have a shorter line on left or right side, agent will leave its current waiting line and move to the new line. For individual agent, it will move by itself. For pair agents, both agents will leave and move together. After queuing up in the new line, all agents’ anxiety will set to 1, which is the lowest degree.

**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.

[Carmine]< Prediction in Social Path Following >

[Walk Along Steering] Popelová, M., Bída, M., Brom, C., Gemrot, J., & Tomek, J. (2011, November). When a couple goes together: walk along steering. In International Conference on Motion in Games (pp. 278-289). Springer, Berlin, Heidelberg.

[Reynolds] Reynolds,C.: Steering behaviors for autonomous characters. In: GDC, pp. 763–782(1999)

[sai-keung] <https://dl.acm.org/citation.cfm?id=3190839> (记得下载)

[Julio Godoy] Online Learning for Multi-Agent Local Navigation

[recast&retour] (ref: <http://masagroup.github.io/recastdetour/index.html>)

[]