**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

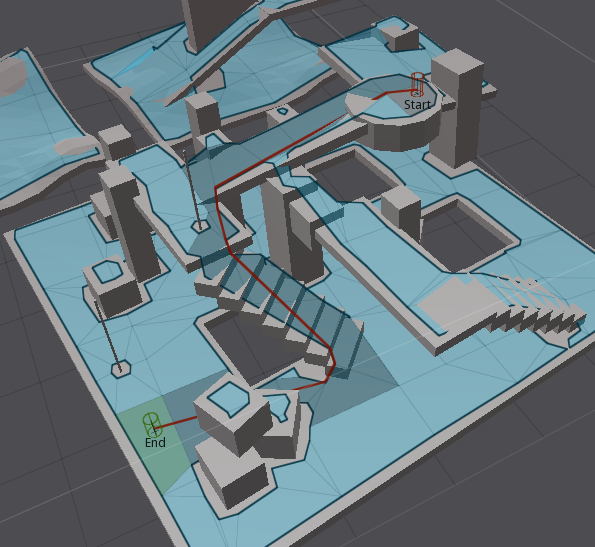
(ref: <http://masagroup.github.io/recastdetour/index.html>)

**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

**Implementation, Challenges, and Output**

**Scenario and Design**

In this project, I will develop a crowd simulation application which aim at creating realistic, unique and accurate crowd. To achieve this goal, I will use an open source state of art navigation mesh construction toolset called Recast 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.

**Input Data Initialization**

Input data contains basic scene information, which contains agent id, agent start time, start position, end position and behavior mode. Thus, scene initialization determines number of agents, size of scene. In the simulation, there has two type of relationships among agents, pair and individual. To determine agent is in pair mode or individual mode, input data of agent have the following specific patterns.

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).

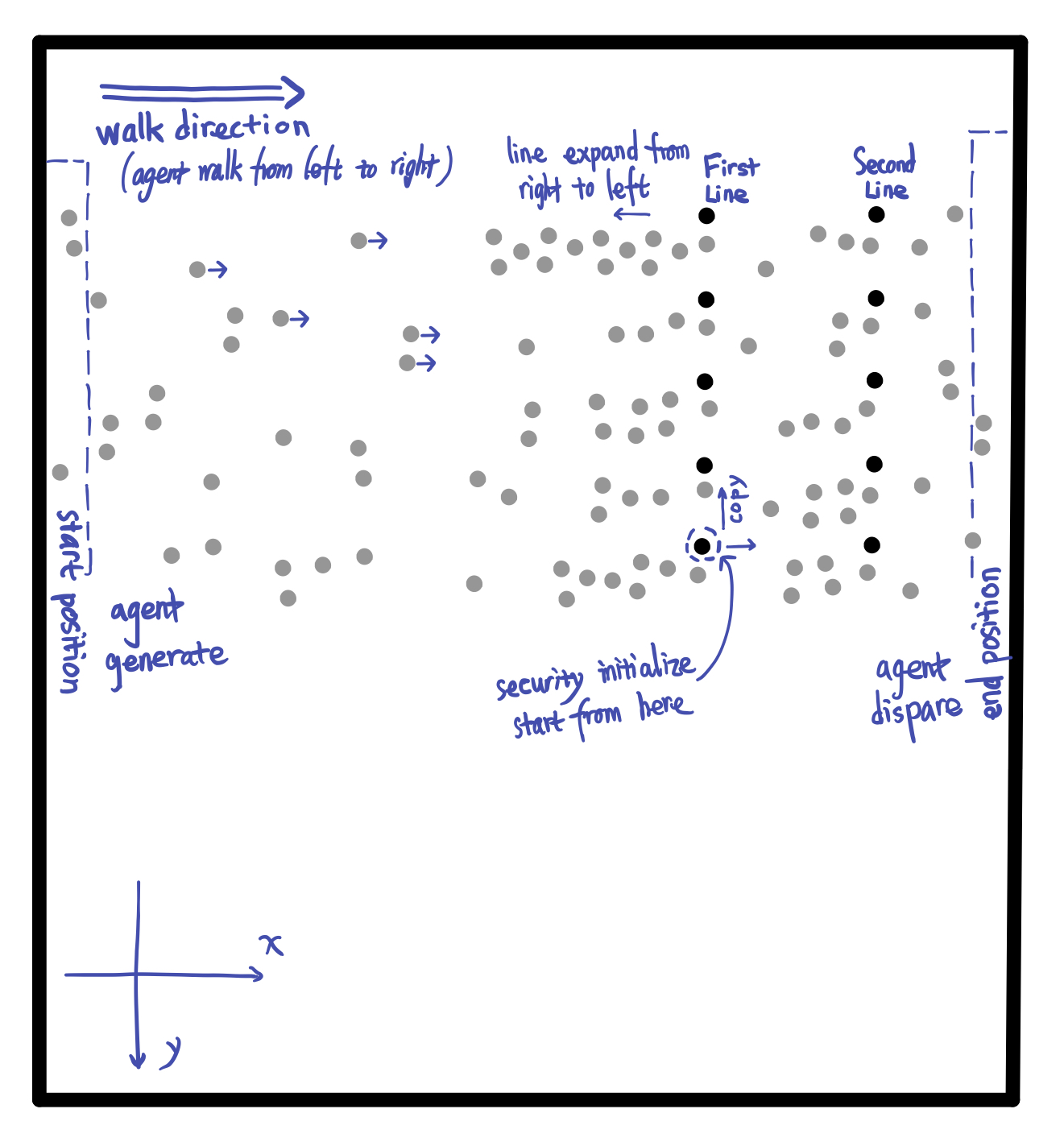
Red is start’s end position (x coordinate, z coordinate, y coordinate).

Purple is agent’s behavior mode.

Agents in pair relationship requires both agents enter the scene at the same time, both agents share the same start and end position. Thus, in the simulation, differences between enter times, start/end positions of agents in pair relationship have the following requirements:

* Start position, end position and enter time determine agents’ friend relationship.
* Agent enter time determines when agent will walk into the scene.
* Start position and end position determine where agent start to walk and where agent walk out of scene
* Agent’s behavior mode determines how agent behave – queue up to form line, flee or simply walk from start position to end position.

Here describe how to determine range of start position, end position and security gate lines.

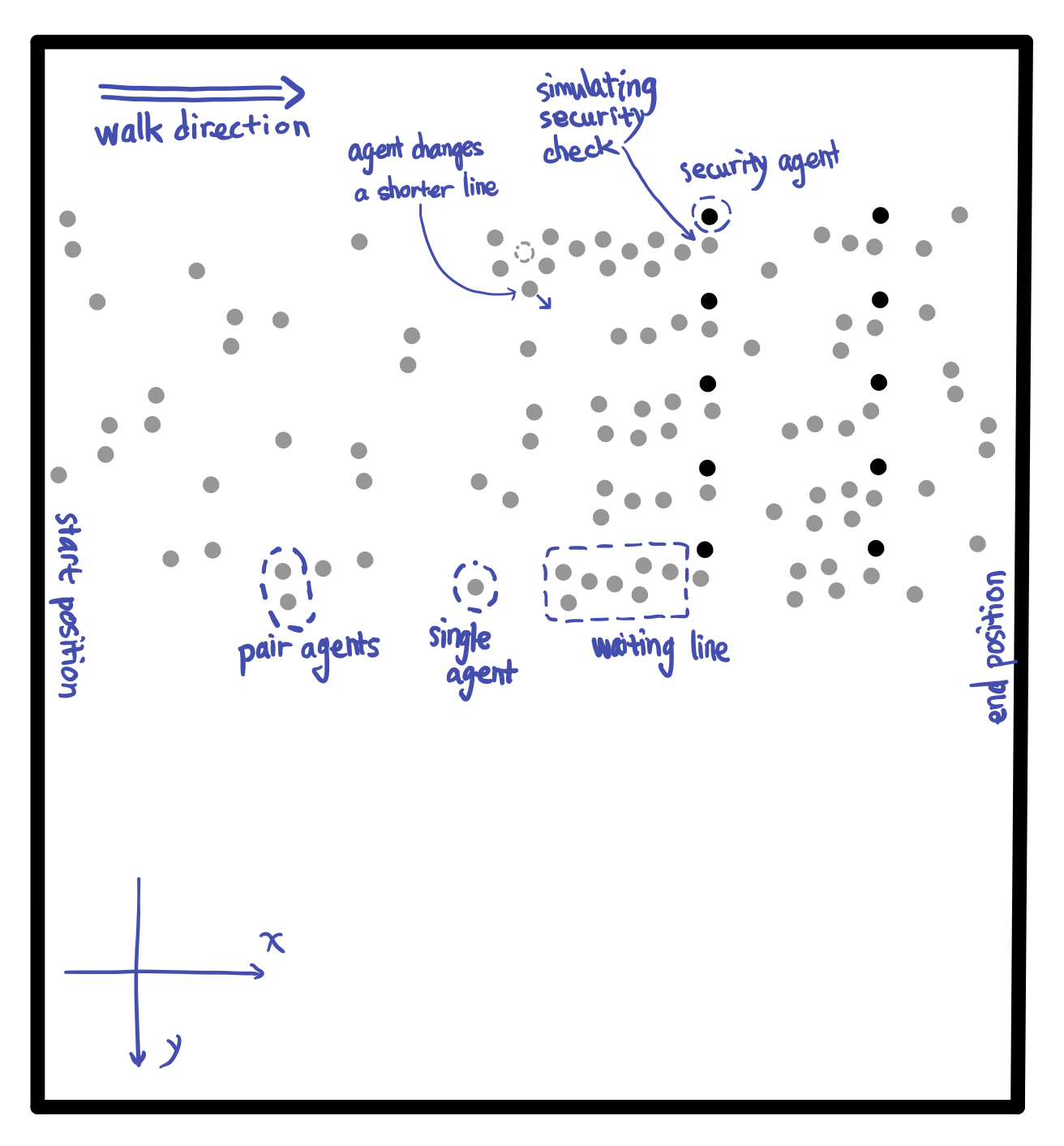


[to be continue…]

**Environment Initialization**

Before achieving the complicated crowd behavior features such as pair walking, queuing up and form single/pair waiting lines, we initialize environment initialization so that agents could perform security check. In this scene, agents are generated and walk from left to right. However, 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 several concert 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. Thus, in order to achieve those two checking patterns, once agents reach the security gates, agents will stop for a few seconds to simulate the security checking process.

At the beginning of the simulation, we will first initialize environment based on the input data that we mentioned above. Image on the left is the mockup demo that demonstrates the scene which represents agents’ environment, image on the right is snapshot of real-word video of a certain event.

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*++;

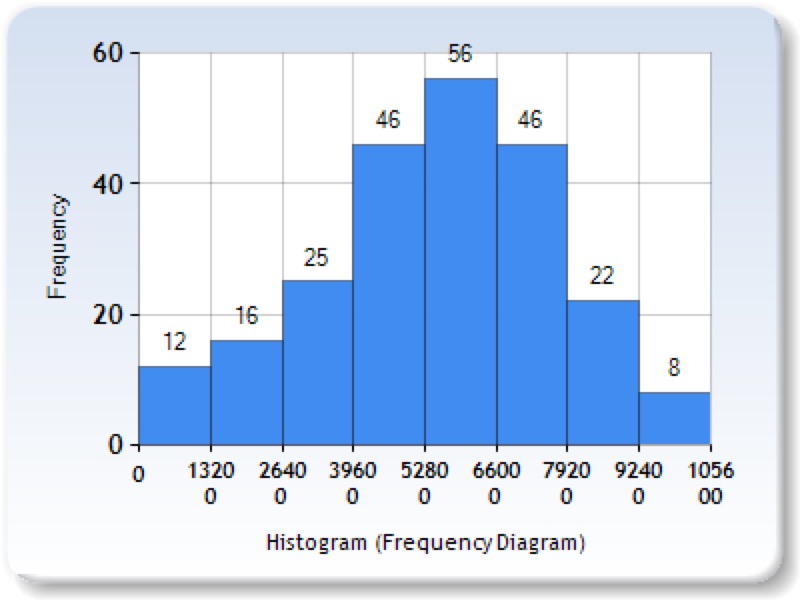
}

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, people have to stop at checking gate show its ticket and do security body scanning before entering the building. For certain people who might carry bag, the security faculty in first line might need to check that bag and they need to stop for a few more seconds. 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.
* ***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 each two agents that have the close enter time, start position and end position. For agents in pair agent relationship, one of the agents in pair plays leader role, and the other plays follower role. 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 that agents in the crowd could have natural queue behavior, we intensively control the flow of crowd which follows certain pattern. At the beginning of the simulation, number 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**

**Queue-up Behavior**

In the real life, since people need to do security check, people need to 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 several 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.

不同状态的不同agent在不同的位置

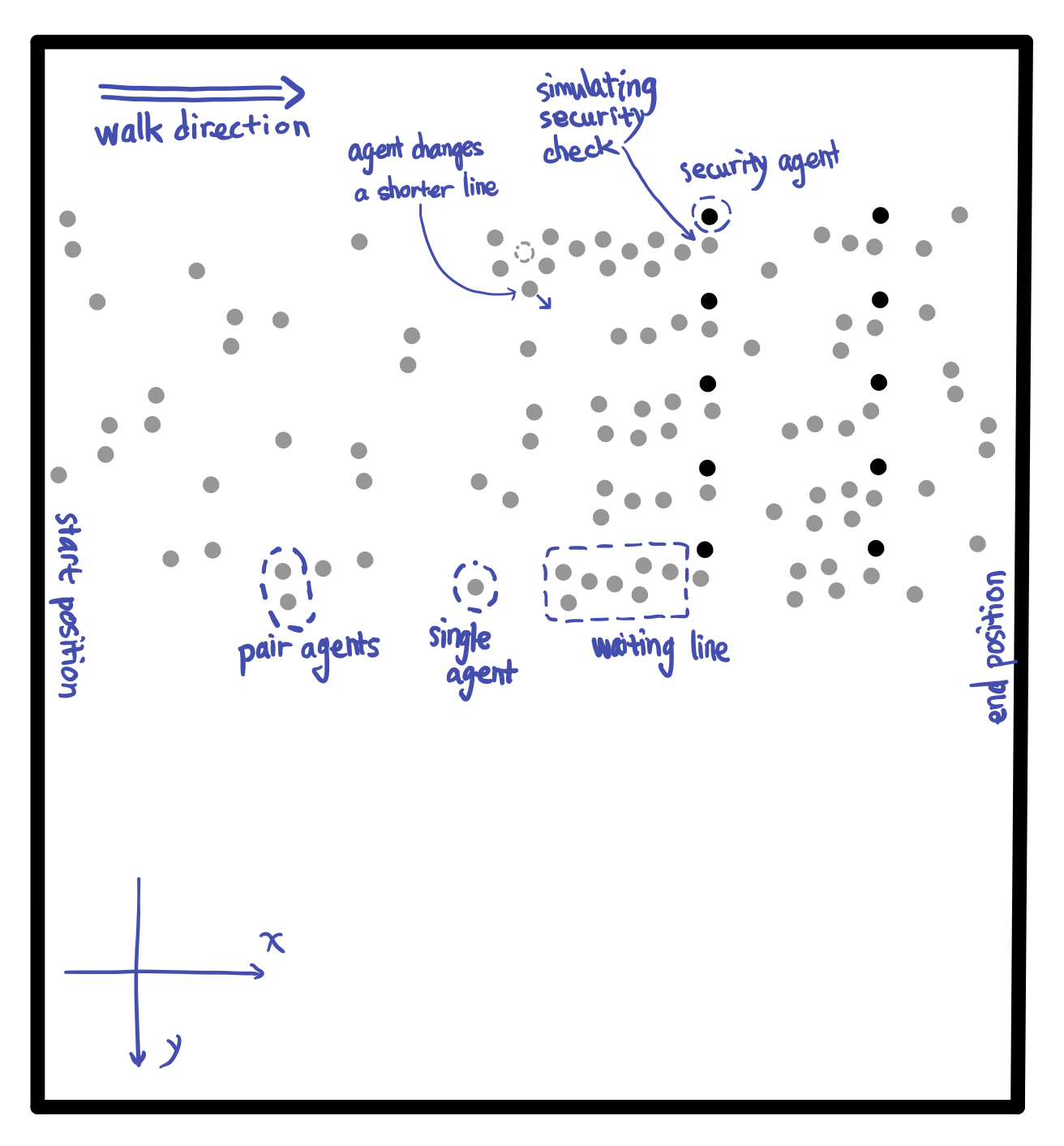
**Pair Walk Behavior**

**Do Security Check Behavior**

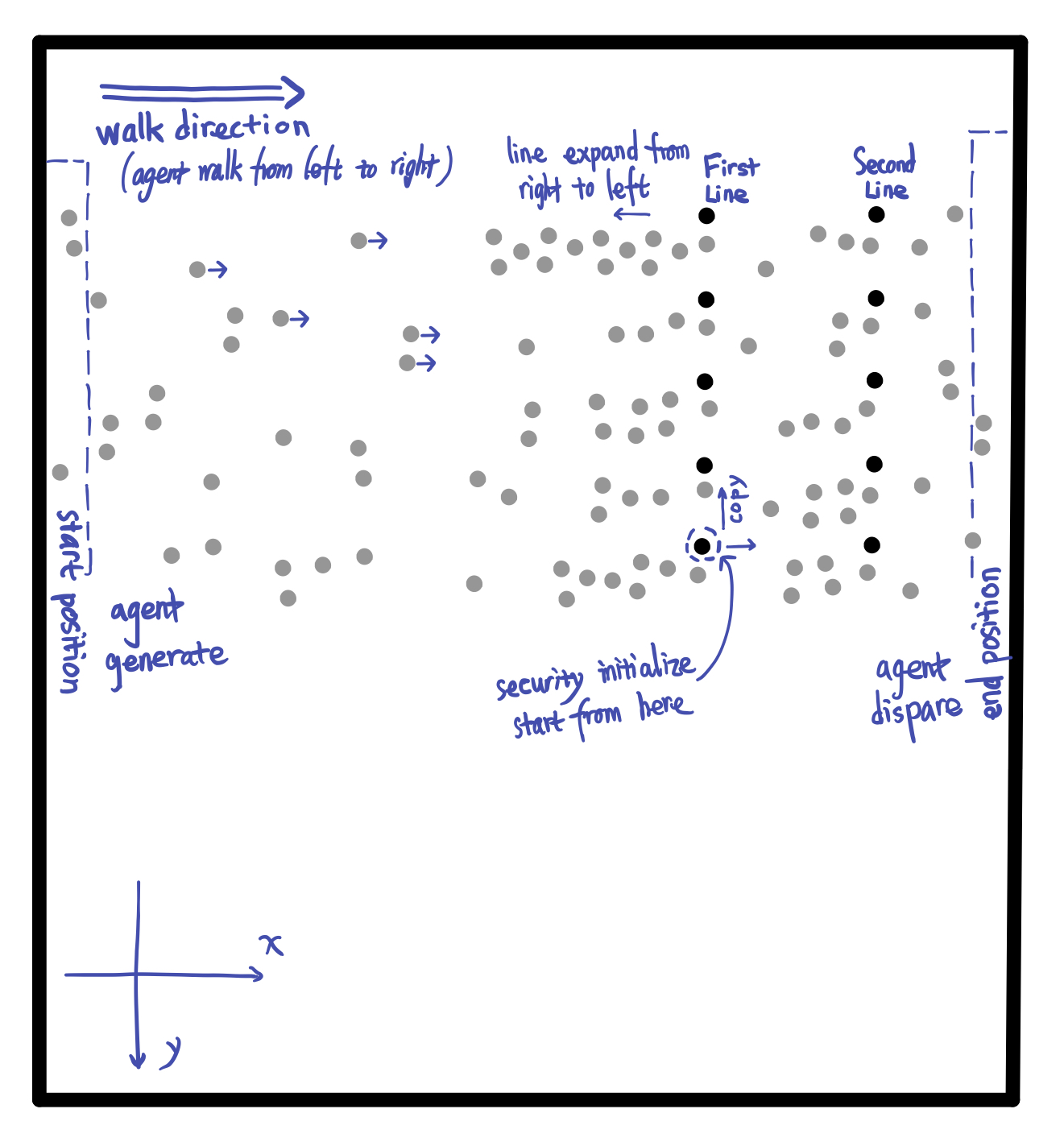
**Re-consider Behavior**

**////////// 素材 //////////**

QueueBehaviorApp will simulate crowd behaviors such as pair walking, queue up and form single/pair waiting lines, agent switch from one waiting line to the others and then doing security checking. The image below is the mockup demo that demonstrates the scene I will create. In this scene, agents are generated and walk from left to right. Among each agent, some agents might know each other, so they walk with each other while others walk alone. Once agent reach the security gate, agents stop for a few seconds to simulate the security checking process. If the security gate is occupied, the upcoming agents wait behind. Thus, the lines are created. If agents in the line are not satisfied with the length of the line, they will look to the left of right side of line to find out if there are any shorter line to go to. If there is a shorter line, the agent will leave its original line and queue up to a better line.



However, before doing that, I will initialize scene to generate input for the crowd simulation. Below is the mockup demo of the application interface with detailed requirement that could help understand the input and default requirements of the crowd simulation:



In this simulation, agents are initialized at the left side of the scene (*start position*) and disappear at the right side of the scene (*end position*). Agents in scene will move from left to right, this moving direction determines the direction of the waiting line, with the waiting line grow from right to left. In the simulation, there are two lines of agents always standing at the same position in the whole simulation. These two lines of agents play roles as security faculties of the event in the simulation to simulate security check. The upcoming agents will stop in front of the security faculty for a few seconds to simulate a security check (or a ticket check) as we do in our real life. Once agents finish check, they continue moving to their final destination. Implementation

**Evaluation**

**Future Work**

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???