**Crowd Simulation: Queue-up Behavior**

MS Project Report

Zhicheng Chen

University of Nebraska at Omaha

Advisor: Brian Ricks  
January 25, 2019

**Abstract**

Understanding complicated crowd behaviors is essential to urban designers and architects. However, grouping a large number of people to do an experiment is dangerous and unrealistic. In order to do experiments researchers are required to design and create an application that could correctly represent crowd behavior. This project report describes an implementation called Queue-up Crowd Simulation that aims at creating realistic, unique and dynamic crowds by taking agents’ data as input and outputting the results in animation form. Various validation methods are implemented to evaluate the simulation results from various perspectives. In a user study of over 100 people, people found our algorithm make simulated crowds look more realistic.

**1 Introduction**

The increasing proportion of people living in urban areas brings new challenges to urban planning and architecture. Crowd simulation can play 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 flowing inside of a building or prevent overcrowding during events. In order to create realistic and trustworthy crowd simulation results, I designed a sophisticated crowd simulation algorithm that allows agents to reproduce realistic features. The features I implemented are observed from data collections and direct observations from video records of real-life events.

In this report I first discuss related previous approaches. Then I discuss observations from real-life video records that were used to guide the creation of the QueueBehaviorApp. Then I present important components of this application that use the open-source library Recast & Detour. I then present details about the scene initialization and agent initialization. Then I discuss the goal scenarios observed and abstracted from source videos and the implementations details. After that, I present various evaluation methods based on the simulation result from different perspectives. Finally, I conclude with a future work discussion.

The main contributions of this report are:

* A realistic and dynamic crowd algorithm with agents that can form waiting lines with each agent orderly passing through simulated security checking spots one by one.
* Agents in the scene that dynamically move in and out of different waiting line to move to the most desired waiting line.
* A flexible pair walking algorithm that allows agents in pair relationships to dynamically stay side by side while walking in the scene.
* Statistically significant results from a user study that show these improvements make crowds appear more realistic.

**2 Previous Work**

There are several existing crowd simulation approaches. Some approaches aim at simulating natural human behaviors when agents encounter specific condition, while other approaches allow agents to implemented certain strategies to make decisions while they are walking.

Mirza Waqar[2] designed agents as ellipses that have a sense of the environment and plan their own path ahead of time to avoid agent collisions. However, simply implementing the collision avoidance strategy is not sufficient to generate the realistic crowd simulation result. Since it does not involve complex behaviors such as allowing agents to move in and out of different group or queues based on agents’ desires, 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, as we observed in real crowds, people do not just stay in the waiting line once they choose it; they might need to change lines if there is a better option.

Other simulation approaches have achieved other complicated behaviors. Julio Godoy [6] provided a dynamic agent base approach that allowed agents in a scene to have distinct goals and plan their own movements and collision avoidance ability. Interactions among each agent are “polite” and natural after agents learned an optimal strategy in the given simulated environment. Carmine [3] extended state-of-art predictive approaches with social awareness, prediction, and social collision avoidance to achieve the prediction in social path following behavior. Social awareness signals agents that are approaching each other, then agents adjust their behavior and direction for future social interaction to improve realism.

Previous approaches mentioned above are mostly focused on external interactions between agents that walking independently. In order to simulate social interactions between agents within groups, Walk Along Steering [4] developed the Walk Along Steering algorithm that allows agents in small groups to have six social steering behaviors that make agents’ movements and reactions smoothly and naturally. Based on a three-layer architecture controlling the motion of Intelligent virtual agents mentioned by Reynolds [1], agents could achieve patterns like following, avoidance, waiting and approaching. To allow the crowd to have more dynamic social behaviors, Sai-Keung’s [5] approach presents a crowd simulation that involves behavior agents that are more interactive. In the simulation, two or more agents are required to perform actions simultaneously to finish certain tasks and agents are divided into two categories: workers and pedestrians. Tasks can be assigned to worker agents. In this case, workers will cooperate with each other and create complicated behaviors by decomposing a complex task into numbers of simple tasks. Similarly, I implemented a dynamic crowd simulation algorithm that allows agents in the crowd to achieve certain behaviors based on its current status. When walking, agents consider not only maintaining the pair status, but also having natural interactions among other agents. The various dynamic interactions could occur

**3 Observations from Video Records**

To understand how real crowds moved I study and analyze recorded crowd videos at CenturyLink at different entrances during events such as concerts and a Disney on Ice Event. Figure 1 shows the screenshot of those videos. I abstracted typical behaviors of the crowd among those videos:

* Pair walking and lining up in pairs.
* Queue up behaviors.
* People forming waiting lines.
* People switching from a long line to shorter one.



Figure 1. We gathered our data from videos of crowds at the CenturyLink Center. Shown are two screenshots from two different events.

**3.1 Pair Walking**

Based on our observation, people attend events with their friends or family, thus people in the crowd are divided into a number of small groups. Because people know each other and have conversations while they are walking, people in a small group are more likely to stay side by side. In the videos, we observed when one of the people in the group is left behind, people in front will stop somewhere and wait until his/her partner catches up, or if people realize they have left behind their partner, they will speed up to catch up with their partner. For example, people who finished the security check would stand somewhere in front and wait for his/her partner.

**3.2 Queue up Behavior**

When a waiting line is formed, people queue up and slowly move forward in the line. Because people might walk with their friends, they might form a waiting line so that each row could have one or two people. Based on the videos, waiting lines could have different lengths, and I found that people in the longest lines would change to the shorter lines or people will directly change direction and walk to the new gate if they find out there in a new empty gate. However, for some people, they might be less interested in moving to the shorter line and they would just stay at their original line.

**3.3 Security Checking Behavior**

For people who reach the security gates, they have two security processes to finish before entering the building. This two-step process is typical for many entertainment venues. For the first step, security officials check the ticket and bag (if someone carries a bag to the event). For people just bringing the ticket, people show the ticket to the checker, then they can quickly pass through the first gate. For people who carry bags, security officials need to take a few seconds to check people’s bags. Thus, people who carry bags spend more time on the first gate. For the second process, the security officials use a handheld body scanner to scan people one by one. Because everyone takes almost the same time during this step, the time difference among people needing to finish the second process is more consistent than the first process. Based on our videos, the scenes contain two lines of security gates. If the number of people passing the first gate line is large enough, the room between the security and ticketing areas can fill up. When this space is full, people finished with the first checking process still need to wait until there is space to let them move forward.

**4 Foundation architecture and Scenario**

Based on these observations I mentioned above, I developed a crowd simulation application which aims at creating a realistic, dynamic and accurate crowd. To achieve this goal, I used an open-source state of art navigation mesh construction toolset called Recastnavigation to achieve static avoidance and shortest path calculations. I also utilized a path-finding and spatial reasoning toolkit called Detour to achieve dynamic avoidance among agents in the path and to complete the calculation of each frame of the simulation [3]. Using these open source platforms, I built the observed higher-level behaviors of my approach – QueueBehaviorApp.

**4.1 Recast and Detour**

This application uses an open source library Java Port of Recast & Detour navigation mesh toolset [7]. Recast is a state-of-the-art navigation mesh construction toolset for games. Recast is an open source library that can automatically provide you a mesh at any level geometry quickly; Recast could also be customized to achieve user’s specific purpose.

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 a user to create lots of agents and move agents in a navigation mesh. Detour also allows user to create customized behaviors that determines agent’s how to move and react at a low level.

**4.2 Scenario**

A crowd forms when a large amount of people gathers in a limited space. Instead of strictly simulate the appearance of the crowds in videos, the overall appearance of arriving people in our crowd simulation approach follows specific patterns: the number of arriving people increases at the beginning of the event approaches, then it starts to decrease after the event starts.

Simulating the crowd as a single unit might help understand its flow movement. However, each person in a crowd is independent. In order to achieve this feature, I divided the crowd into groups that contain both pairs of 2 people and single individuals so the behavior of the crowd would better reflect what we observed in our film. In a group, people that know each other might walk together. Previous researcher Reynolds [1] proposed a steering approach known as Leader Following (LF). This approach also involves pair agents where the “follower” agent follows the leader and stays to its side. The disadvantage of this approach is that the leader agent does not wait for its follower agent if the distance between these two agents is too large, which is not realistic.

For our main scenario, single agents or pair agents are randomly generated at 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 reach their destination, every agent has to finish two security checks. (People did 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 generated while others are stopped at security, the total number of agents in the scene increases dramatically and a 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.

**5 Environment Implementation and Setup**

Initialization of environmental factors such as agents’ entering position and time are essential to the simulation’s appearance.

**5.1 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, I need to manually generate agent data based on features I observed from the video. For example, I defined agents in pair relationship by letting agents have the same enter time, start position and end position.

In order to achieve more crowd features, I designed input data that based on the following patterns: Agent id, enter time, start position, end position, behavior mode. Each agent has its unique id number. The agent’s enter time determines when agent will walk into the scene.

The start position determines where an agent will appear. The end position determines where agent will exit, which also determines the agent’s walking direction. The enter time, start position and end position determine agent’s pair relationship. Agent’s behavior mode demonstrates the agent’s behavior and can be one of two options:

* Queue – means the agent will queue up to form a line.
* None – means the agent will simply walk from the 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 model.

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

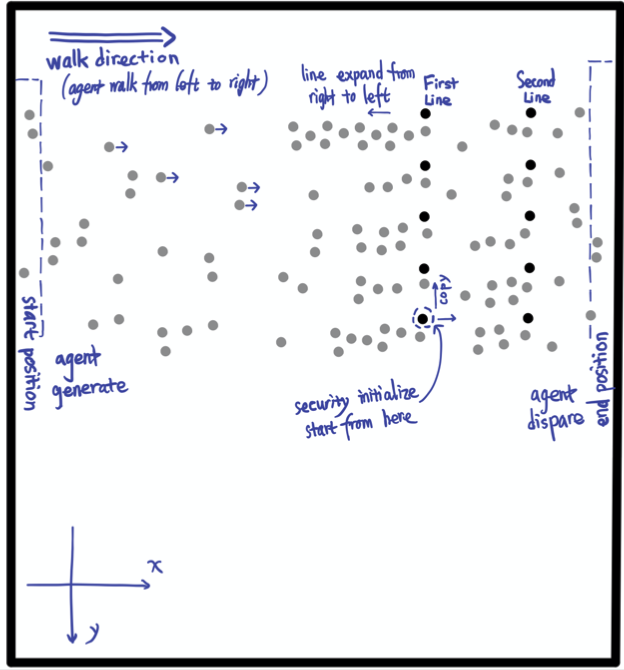


Figure 2. Agents enter the scene from the start position and walk out of the scene through the end position after they finish the security checks. Shown is illustration of the simulation result.

In figure 2 each circle represents an agent: agents in black are security faculty and agents in grey are normal agents. Faculty agents are specifically initialized, and they will stay in the same position during the simulation. Agents walk in from the rectangle on the left called the start position and walk out of the scene on the rectangle on the right called the end position. Without security gates, all normal agents simply walk from start position to the end position. However, since there are security gates in this scenario, agents will do security checks and queue up.

**5.2 Environment Initialization**

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

At the beginning of the simulation, the environment is initialized based on the input data. Figure 3 illustrates how multiple initialization functions setup the environment.

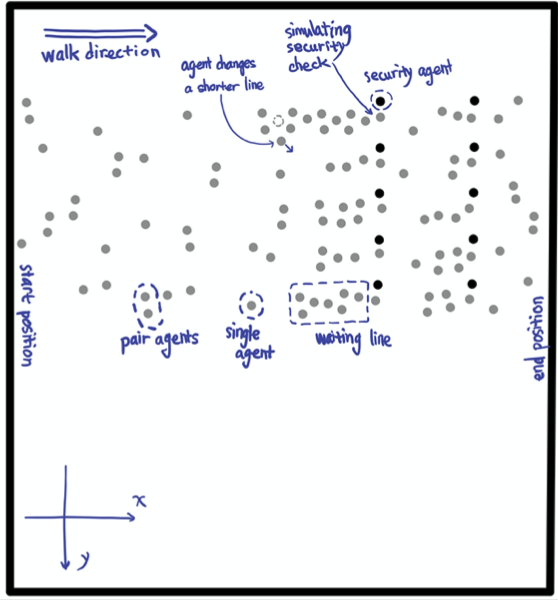


Figure 3. The initialization of the environment is divided into several parts. Shown is multiple parts of the environmental setup process achieved by initialized functions.

The code below always runs at the beginning of the simulation.

initGates();

initAgentCheckTime();

initAgentGateOption();

initFriendRelationship();

initAgentsAnxiety();

The methods above cover all the initializations required in the simulation. The description below shows more about details of the initialization:

* **initGates** – In this scene, 10 checking gates are created and divided into two gate lines to represent two checking processes. Then 10 faculty agents are initialized and represent security faculties standing next to the gates. All security faculties will stand next to their own checking spot during the simulation.
* **initAgentCheckTime** – Each person will have two processes to finish before entering the building, one is ticketing and bag checking, another one is body checking. Based on the observation, at the first gate, people carrying bag spend more checking time than people who just bring the ticket. However, the amount of time people spend in the second gate is more similar. Thus, in the simulation, each agent is randomly assigned two values to represent the amount of time they need to spend to finish both checks. Based on the observation of the video records, the people who carry bag is random and the total amount of them is smaller than the amount of people who don’t. Thus, a certain proportion of agents are initialized with a larger value for the first checking process to represent agents who are carrying bags. However, for the value that represents the second gate checking time, since everyone has the same checking process, the values among each agent are more consistent.
* **initAgentGateOption** – In this simulation, agents are randomly generated on one side of the scene. Then, based on each agent’s current position, they choose the closest gate to go and move forward.
* **initFriendRelationship** – Every agent is either in individual status or pair status. Based on the input data, agents that have a close enter time, start position and end position are set to paired status and stay walking side by side. Each agent in a pair has a different role: one is leader and the other one is follower. Otherwise, agents are in individual status, which means they are neither a leader nor follower.
* **initAgentsAnxiety** – In real life, people at the 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, each agent could make a line change decision based on their anxiety degree. The agent’s anxiety degree updates based on its current position in the waiting line. However, as not every person in the line is seeking the opportunity to switch to a shorter line, the variety of anxiety degrees among each agent follows certain patterns. The agents with high anxiety are only a small proportion of all agents in the scene.

**5.3 Appearance Feature of Upcoming Agents**

In the simulation, in order to create a realistic crowd, I intensively control the flow of the crowd and let it follows a 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 start to increase and then reaches the maximum. After that, the number of upcoming agents slowly decreases.

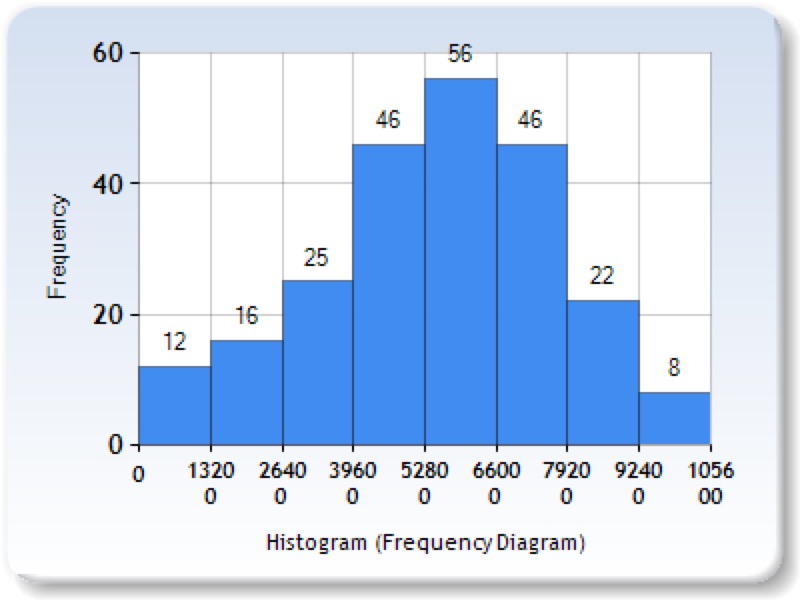


Figure 4. We allowed the distribution of the appearance of the agents roughly follows the bell-shape pattern. This figure presents the number of new agents appear every certain amount of time.

In the figure 4, the bar graph shows the change in the 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.

**6 Agent Behavior Implementation**

In order to simulate a dynamic flexible crowd, agents in the scene could have various behaviors such as pair walking, lining up, coming in and out of waiting line and etc. Also, agents behave based on their current state such as the anxiety degree to decide switching to a shorter waiting line.

**6.1 Pair Walk Behavior**

In the real-life video, people in the groups are talking to each other while they are walking. To reproduce this behavior, they need to stay side by side and sometimes they need to slightly adjust their speed to stay in pairs. In order to simulate this natural pair walking pattern, I let agents in pairs to be followers or leaders. In the pair, the leader is the one leads them to their shared destination and the followers always adjust its speed to catch up its leader. However, only changing the speed of the follower is not enough, the leader also needs to adjust its speed to wait or catch its follower. For example, in the simulation, if the leader finishes the security the check first, instead of directly moving to the next destination, the leader will stop somewhere in front and wait until its follower finishes the check.

Agents in pair adjust their moving speed based on the 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. By utilizing the law of cosines, the angle between these two vectors is calculated.

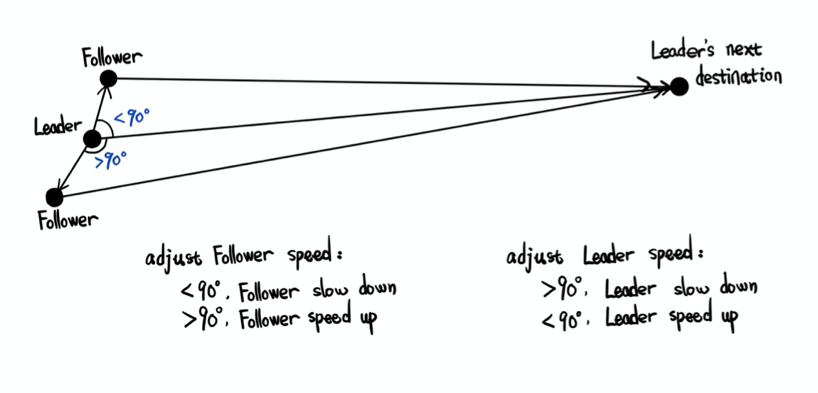


Figure 5. Each agent in pair adjust its speed based on their position. Shown is how agents in pair determine their speed.

Figure 5 illustrates how the agents’ position effect their speed within a group. Bases on the value of the angle, the agent’s speed has three states:

* When the degree is 90: the follower is exactly left or right side of its leader. The follower agent doesn’t need to change it’s moving speed.
* When the degree is less than 90: the follower is in front of its leader. The follower starts to slow down and the leader starts to speed up.
* When the degree is larger than 90: the follower is left behind by its leader. The follower starts to speed up and the leader starts to slow down.

However, knowing the angle between two vectors is not enough to create natural speed adjustment. Because agents have to queue up or pass through the security checking process, the current destination of the leader’s agent might change. The modification of the current destination might affect the value of the angle I mentioned above. Thus, in order to maintain the correct current destination, leader agent could update and set the next destination after agent has reached the current destination.

**6.2 Queue up Behavior**

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

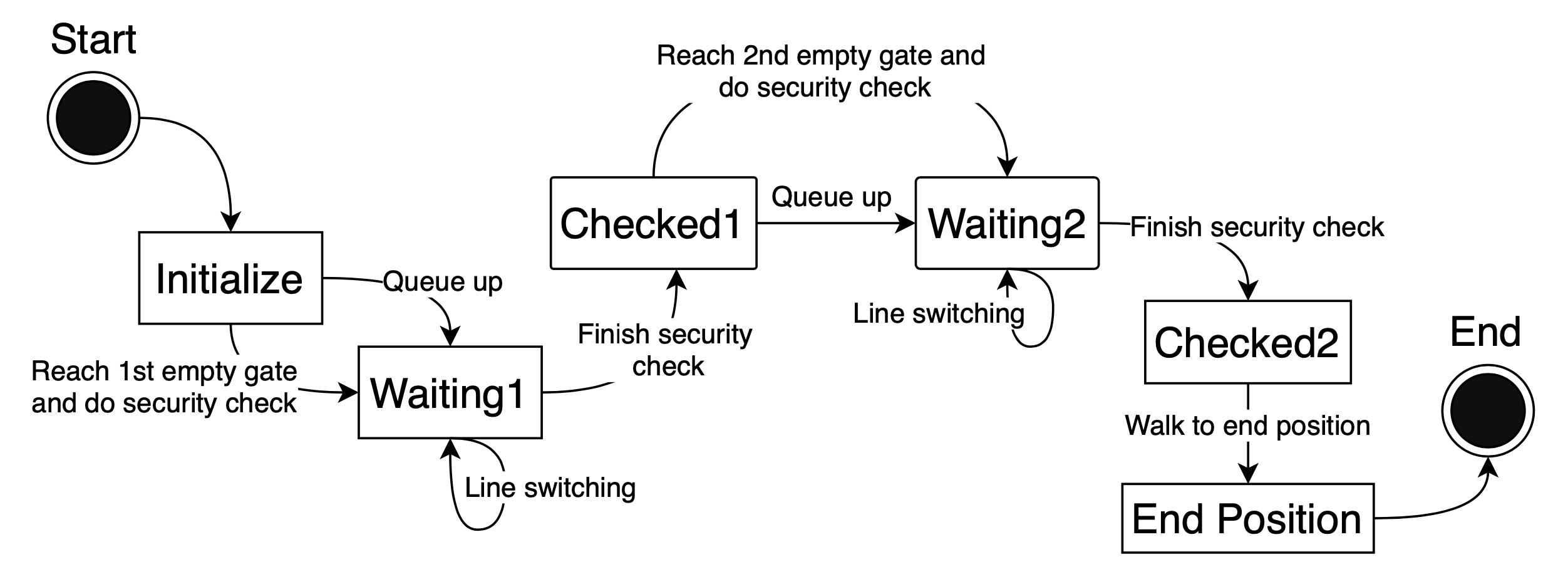


Figure 6. Agents behave based on their state. Shown is indicating how the state determines the behavior of the agent.

Figure 6 shows sequence of behaviors each agent in the process of security check. It illustrates the executions of queue up behavior and security checking processes that change the agent’s states.

To simulate this queue up behavior, I allow each agent to have the following status:

* Waiting1 – agents lined up before the first gate line.
* Checked1 – agents finished the first security check.
* Waiting2 – agents lined up before the second gate line.
* Checked2 – agents finished the second security check.

In the simulation, I utilized 4 states to determine the current state of agents. When all 4 states are negative, the agent is currently walking from their start position to the first gate line. When the agent arrives, two situations could arise. The first is agent could directly reach the security gate. The other is that the agent queues up at the tail of a certain waiting line. The agent’s state is set to Waiting1 state for both cases. Thus, I have a queue maintenance function to handle the status of each agent is in Waiting1 state.

For agents that could directly reach the security gate, the agent starts to count down its bag/ticket checking time. The agent will change its state to Checked1 after it finishes the security check at the first waiting line. However, for agents queue up a waiting line, they need to take more time in the Waiting1 state. The agent first changes to Waiting1 state once it is in the line, then the agent needs to set its current destination to back position of another agent in front so that it can stay in the queue. Once the head agent finishes the security checking, it will change the state from Waiting1 to Checked1 and move forward to the next destination. The agent behind the leaving head agent will replace the head position and start its checking process. Every agent orderly passes through the checking processes the same way.

Because agents in pair relationships will walk in pairs, they 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 agents.
* : : – pair agents queue up behind another pair agents.

The follower agent always tries to stay next to its leader agent. Thus, 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 the middle back position of the pair agents.

In this simulation, each agent needs to finish two security processes. Even though two processes might cause different times, the process contents are basically the same and waiting lines also form in front of the second gate line. Therefore, after the agent finished the first checking process, the agent moves to the second checking gate to do similar actions. The agent changes its Waiting2 state to Checked2 state after finishing the second checking process and moves to its final destination.

However, space between gate lines is limited. Each agent takes a different time to finish the first security check. In the simulation, there are more agents who do not carry bag, which could lead to certain situation: because the majority of agents don’t take too much time on the first checking gate; the checking time of the second checking gates is longer; the space between first gate line and second gate line will be quickly fill up. Once the room between gate lines is full, agents will have to wait until there is space to move forward. In the simulation, head agents who are doing the security check at the first gate line always start to count the bag checking time only when there is enough space between itself and tail agent of the second gate’s waiting line in front.

**6.3 Re-consider Behavior**

Once agents enter the scene, each agent is assigned a value to represents its anxiety level. From the real-life video, I found out some people will switch to another waiting line if they have a better option. However, instead of switching to the other shorter line, they are more willing to stay in their own line. Thus, in order to simulate this interesting condition, I allow agents to have anxiety. The anxiety degree determines an agent’s line switch decision.

However, simply assigning anxiety level to agents is not enough to simulate the reconsidered behavior—I also have an anxiety monitor to adjust the anxiety degree based on the waiting lines conditions. Agents with higher anxiety are more likely to change line, agents with lower anxiety are more willing to stay at its original line. In the simulation, each agent will have certain levels of anxiety degree.

Anxiety degree updates in certain internal conditions: (1) the agent queues up in the waiting line. (2) the agent checks the length of the waiting line of the left and right side of the current line. (3) based on the difference between the current line and other lines, an anxiety degree is correspondingly updated. (4) if the agent is satisfied with its length of the waiting line, the agent’s anxiety degree will decrease or remain still. If the agent is not satisfied with the length of its waiting line, agent will increase the anxiety degree.

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

**7 Evaluation and Results**

In order to evaluate the simulation result thoroughly, we used three validation methods: input data evaluation, behavior evaluation and a Mechanical Turk survey evaluation.

**7.1 Input Data Evaluation**

The evaluation of the input data generated by QueueBehaviorApp follows the requirement of the simulation. For agents in pair relationships, agents’ information in the input data set have should achieve the requirements mentioned in the Input Data Initialization section. However, based on the limitation of the pair relationship in the simulation, I only allow at most two agents in pairs. Also, agent cannot have pair relationship with more than one agent.

**7.2 Behavior Evaluation**

Based on observations of the simulation results, agents could achieve the following behaviors:

* Agents in pair relationships do pair walking.
* Agents adjust their speed to stay side by side.
* Agents do two processes of security check before walking to their final destinations.
* Agents queue up and form the waiting line.
* Waiting lines have one or two agents in each row.
* Agents stop if no room between gate lines.
* Agents leave and move to a shorter waiting line.
* Agents increase speed while changing the line.
* Agents could change their target line before lining up.

In the simulation, 260 agents are randomly generated which are distributed at one side of the scene. The initialization of various status of agents is reset every time the simulation starts. The amounts of agents in pair relationship, agents could orderly and naturally form waiting lines and the moment agent leaves its current waiting line, etc, can be clearly observed from the input sets and simulation results.

During the waiting process, the anxiety degree of each agent in the waiting line could represent the desire of line changing. Based on the observation, once the anxiety degree is maximum and there are other shorter waiting lines nearby, agents leave their current line and line up at the shorter one. The line switching behavior happens to both individual agents and pair agents.

**7.3 Mechanical Turk Survey**

In this project, I also evaluated the simulation result by implementing Amazon Mechanical Turk to do surveys about the comparison of different crowd simulation videos. The whole demographic information of the participants is restricted by Amazon [8]. However, we still could guarantee that all participants are adult who live in the United State and they all have legal right to work in the US. In addition, each participant is only allowed to do the test once.

In the survey, one simple instruction and two versions of crowd simulation results were presented on the survey interface. Two simulation results were played at the same time: one contains all the behavior features I implemented; another one contains a plain simulation in which agent only do the security checking process. Participants are required to watch the whole simulation result, after watching the whole video, they have to pick the one they think is more realistic. In order to reduce the probability of the graphical location of videos that affects the survey result, I flipped the position of the comparing videos and also do the similar number of tests.



Figure 7. We validated the simulation result by implementing an Amazon Mechanical Turk to do survey. Shown is the interface of the survey.

|  |  |  |
| --- | --- | --- |
| Survey | Improvements on the left | Improvements on the right |
| Left votes | 33 | 20 |
| Right votes | 18 | 37 |
| P-value | 0.0244 | 0.0166 |

Table 1. Two rounds of survey and 108 participants who do the Mechanical Turk Survey. Shown is the table of the survey result.

There are 2 rounds of survey and 108 participants who do the Mechanical Turk Survey. Table 1 shows result of surveys, the full-featured simulation is on the left and it had 33 votes, the one that has no improvements is on the right and had 18 votes. For the second survey, after flipping those two videos, the full-featured simulation is on the right and had 37 votes while the other one had 20 votes. Using a one-tailed, one-sample proportion calculator, we calculated the P-value of survey 1 as 0.0244, and the p-value of survey two is 0.0166. This leads us to reject the null-hypothesis that the difference in votes was random. Note that in both tests there were at least 10 votes for each side, a required assumption to be fulfilled to use this statistical method.

**7.4 Result**

Based on the observation, some agents walk in pairs. Agents do pair walking from the start position to the end position. The pair walking behavior also includes waiting behavior. For instance, once leader agent finished its check, instead of thoughtlessly moving forward, it will wait in front until its partner finished its check.

Once agents are generated from their start positions. They move forward the closest checking spots and do security checks. The upcoming agents in the scene waiting for check start to form waiting lines. In the simulation, 4 waiting lines have been formed, each row in the waiting line contains at most 2 agents. Agents in pair relationship also keep the pair status in the line.

**8 Conclusion and Future Work**

In this paper, we present a flexible dynamic crowd simulation algorithm which could achieve behaviors such as walking in pair, forming waiting line, passing security checks, etc. Future work could expand this work by allowing groups to have more than 2 people. This would allow the group agents in the simulation to have more dynamic behaviors. Further, the future work could also expand on this work by allowing for queues to form that are not perfectly straight. This would allow the agents in simulation to adjust their line up position based on the environment.

**9 Acknowledgements**

This project was funded in part by NSF Grant #1718139. Also, I would like to thank my project advisor Dr. Brian Ricks of the Computer Science Department at the University of Nebraska Omaha. I can always find help from him when I have questions about my research or writing. I would also like to thank Dr. Robin Gandhi and Dr. Hassan Farhat for serving on my project committees. It is a great honor to present my project to them and gather feedback from them. Finally, I would like to thank the support of my family and friends. The help from them is immeasurable through my studies and life.

**10 References**

[1] Reynolds, C.: Steering behaviors for autonomous characters. In: GDC, pp.763–782(1999) 10

[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.

[3] Oliva, Carmine, and Hannes Högni Vilhjálmsson. "Prediction in social path following." Proceedings of the seventh international conference on motion in games. ACM, 2014.

[4] 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.

[5] Wong, Sai-Keung, Yi-Hung Chou, and Hsiang-Yu Yang. "A framework for simulating agent-based cooperative tasks in crowd simulation." Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. ACM, 2018.

[6] Godoy, Julio, et al. "Online learning for multi-agent local navigation." CAVE Workshop at AAMAS. sn, 2013.

[7] Ref: http://masagroup.github.io/recastdetour/index.html

[8] Difallah, Djellel, Elena Filatova, and Panos Ipeirotis. "Demographics and dynamics of mechanical Turk workers." Proceedings of the eleventh ACM international conference on web search and data mining. 2018.