**Crowd Simulation Background** (Why do crowd simulation):

The increasing urbanization of the world population presents new challenges for decision makers. Real-time crowd simulation is crucial in addressing these challenges, including determining evacuation times in complex buildings, avoiding overcrowded areas during mass events, and improving the crowd flow in cities. We are developing a simulation framework with unique features that aim at realism, speed and accuracy. Our software is available for research and commercial use.

(Reference: <https://www.uu.nl/en/research/algorithms/computational-geometry/research-themes/crowd-simulation>)

Crowd and multi-agent simulation is the process of simulating large numbers of people, creatures, or other characters, each interacting in one environment. These actors are expected to move to their goals, interact with their environment, and respond to each other. Crowd simulations have many uses, including improving architectural planning, enhancing training environments and virtual realties, and driving artificially intelligent (AI) characters in games and movies. Our group has worked on many problems in crowd simulation, including fast, guaranteed, collision avoidance, real-time path and motion planning, crowd flows, and directed behaviors. See also our related work in ([motion and path planning](http://gamma.cs.unc.edu/research/robotics/)) for single and multiple robots or agents.

(Reference: <http://gamma.cs.unc.edu/research/crowds/>)

==============================================================================

**Existing approaches (others’ works)**:

[May or may not have features in common]

Walk Along steering for navigating a couple of agents to reach a certain place together. The results of a believability study with 26 human subjects who compared the new steering to the known Leader Following steering in eight different scenarios suggest the superiority of the Walk Along steering in social situations. (from Abstract Malostranské nám)

< ***When a Couple Goes Together: Walk along Steering*** >

The algorithm is composed of three phases:

1) perception of agents and obstacles through the peripheral vision of the agent setting up a list of neighbors and time contacts;

2) building ranges of available speeds and orientations;

3) discretization of ranges, finding the best combination considering a heuristic function about energy consumption, deviation angle and risk of collision.

< ***Prediction in Social Path Following*** >

Problem Descriptions:

The main objective of this work is to introduce online learning techniques in multi-agent navigation so that the agents can exhibit a more intelligent collision avoidance behavior. In a typical multi-agent navigation problem, we are given a virtual environment containing static obstacles and n heterogeneous agents Ai (1 ≤ i ≤ n) with specified start and goal positions. The task is then to steer each of these agents to its goal without colliding with the other agents and the obstacles present in the environment. We also require that the agents navigate independently without explicitly communicating with each other.

< ***Online Learning of Mutil-Agent Local Navigation*** >

What this paper do:

This paper presents a novel technique to design the agents to with following modules 1) Path planning behavior for collision avoidance 2) Situation awareness during herding behavior and turbulent flow in high density crowds. 3) Personal Reaction bubble (PRB) based response and perceptions. The evaluation with real life situations is performed to validate the RBAS model.

< ***Realistic Modeling of Agents in Crowd Simulations*** >

**Question**:

What unique feature(s) our project have?

How I verify my result?

Speed up 🡪 acceleration

**PAPER**

**Abstract**

Current crowd simulation research could achieve path planning for collision avoidance, researchers are more focus on develop walking behavior of single or pair agents. Unfortunately, their crowd simulation approach is lack of realism and flexibility because it does not involve complex behavior such as allowing agents to move in and out of different lines based on agent’s desire. In order to create realistic and trustworthy crowd simulation result, we proposed new, sophisticated crowd simulation algorithm that allow agents to achieve complex features. In the simulation, agents not only queue up to form waiting lines in the scene to pass through gates, but also switch between lines if agents think there has a shorter line to queue up. These crowd features are collected from various observations of real-life crowd videos recorded among different events.

**Introduction**

[**Previous work**] The increasing proportion of people living in urban areas brings new challenges to urban planning and architecture. Bringing a large amount of people to perform the crowd experiment is unmanageable and dangerous. Thus, 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, detect the behavior of crowd flow inside of a building or prevent overcrowding during certain events.

[**Our contribution**] A large crowd of people is a complicated system. Technique such as agent based modeling which model the situation comprised of individual agent into simulations and making decision based on the situation agent dealing with could successfully allow agents behave as pedestrians in the simulation.For example, agent based simulated approach [Baig, Mirza Waqar] 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 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 in the longest waiting line without. However, 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.

[Realistic Modeling of Agents in Crowd Simulation: …*Path planning for collision avoidance…* do I need to mention about the lower level approach?]

**Related Work**

**Approach**

Before we dive into our approach, we need to finish our simulation setup: various agents in the scene could achieve features such as interaction, navigation, perception and optimization; agents walk in the virtual two-dimensional plane which is represented as the environment; time is counting, for each time a new velocity vector is calculated, each agent’s next position is updated for each frame. Our approach is based upon the work of an Open Source Application [##*open source app link*##]; summarizing the Open source application’s algorithm, which consists of the following main steps in which are agents could avoid static obstacle collision, retrieves other agents’ collisions and navigate to the desired destination. We expand these three stages by increasing social behavior and complex queue up behavior patterns.

**Video Observation**

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: 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.

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 chooses 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.

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 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.

**Implementation: 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.

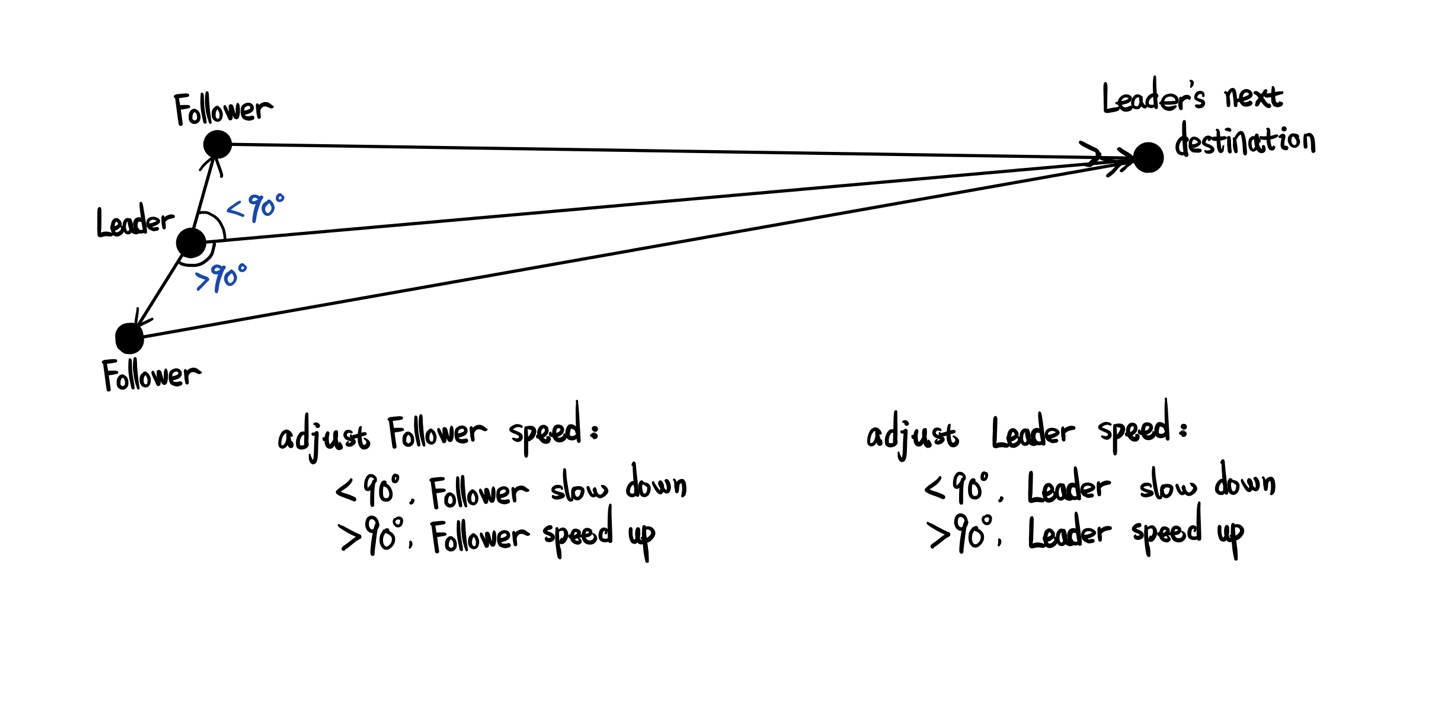
As we mentioned above, follower agent is set to stay next to its leader agent. Thus, in the waiting line, unlike the other agents that their current destination is set as back position of agent in front, the follower agents’ current destination is set as left/right side of its leader. Then when individual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

**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.



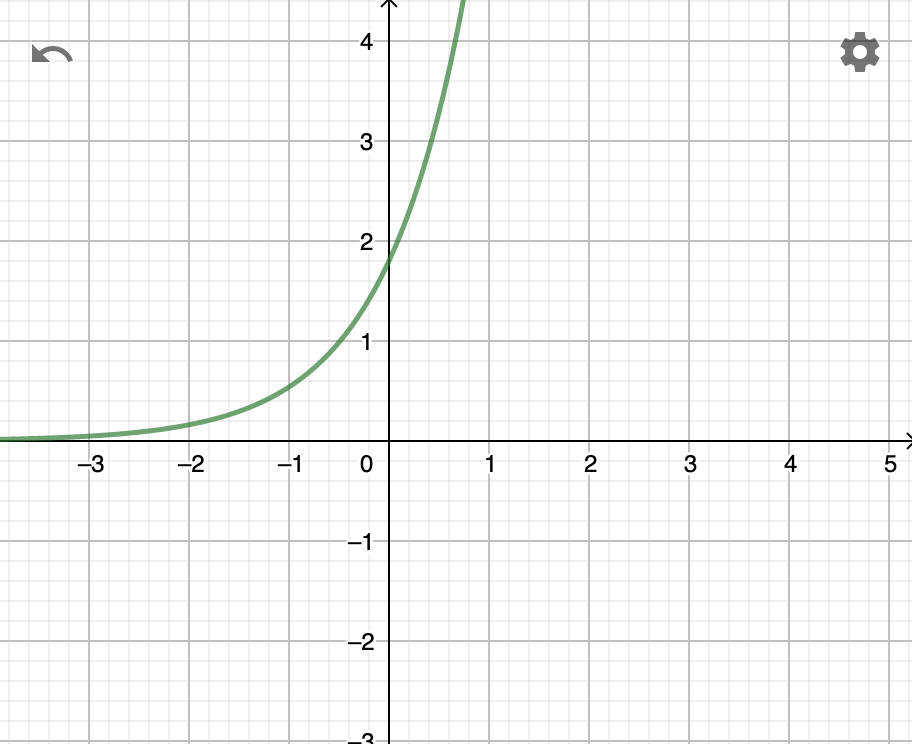
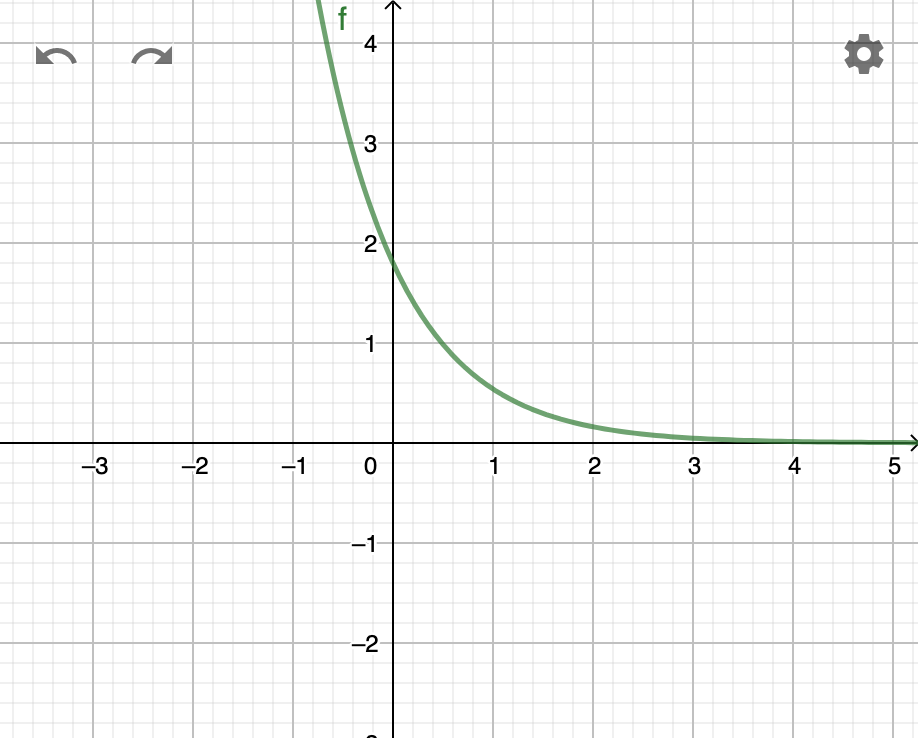
Bases on the value of angle, we allow agent’s speed has three states:

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.

When the angle is less than 90, it means the follower is in front of its leader, the follower starts to slow down.

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)

**Reference**:

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