Simulating Crowd Dynamics of a Moshpit via Multi-Agent System

Michael Wahba
CPSC 565 (Emergent Computing)

ABSTRACT

Moshpits are when a crowd of concert-goers engage in collisions with neighbors at a concert. Crowd dynamics in emergency and possibly dangerous situations are of great interest to researchers. This project implements a moshpit simulation in the Unreal Engine 4 game engine using a multi-agent system implemented via the Entity-Component-System.

INTRODUCTION

Modelling the dynamics and interactions of individuals in a crowd is of great interest to the fields of epidemiology [1], architecture, escape planning, people transportation and many others [2]. As urban populations increase, the danger of density-dependent hazards such as crowd stampedes and viral infections will only increase [3]. Having a robust crowd simulation environment can help mitigate the impact of these events by letting researchers perform *in silico* crowd experiments.

The problem being addressed is: How can we create a real-time simulation of hundreds of individuals in a crowd, each with their own behavior?

To address this problem, I used Unreal Engine [4] to create an agent-based model to simulate the complex emergent behaviors present in a crowd of humans. Unreal Engine is a game engine, making it an ideal tool to render many individuals on screen. Unreal Engine's Entity-Component-System also allows us to assign behaviors to each of these individuals, thus creating a multi-agent system where each agent responds to their environment and their immediate neighbours.

Crowd behavior models can be categorized as being agent-based, flow-based and particle-based. The distinction between these models is dependent on whether the emergent behavior emerges from simulating the individual agents or the general behavior of the whole crowd [5]. For this project, an agent-based model was employed.

The evaluation of the efficacy of this simulation will be done by constructing a model system: That of a moshpit.

Moshpits as a Model System

Silverberg et al. defines a moshpit as a large crowd (consisting of 10^2-10^5 attendees) where the collective mood is influenced by the "combination of loud, fast music, synchronized with bright, flashing lights, and frequent intoxication" [6]. The unique environment under which a moshpit emerges is fascinating as a model system because the emergent results in atypical social behavior where the likelihood of injury is high.

Moshpits consist of participants (hereafter referred to as *moshers)* randomly colliding into each other in close proximity. The qualitative result of a moshpit has been related to that of a disordered gaslike state [6]. In order to alleviate the potential dangers of a moshpit, it is necessary for emergency personnel to effectively plan how to deliver medical care in these extreme environments [7].

Thus, an agent-based approach to modelling the emergent behavior of a moshpit is proposed to help understand how a moshpit emerges and what conditions are necessary for it to stop safely.

RELATED WORK

Agent-Based Crowd Dynamic Simulations

Agent-based crowd simulations have been of great interest to researchers, as such there is already plenty of work to build on.

Zhou et al. provides an overiew of the model types and simulation technologies pertaining to crowd modeling [8]. Of interest to this project, is the categorization of agent-based, flow-based and entity-based models, however there can be overlap between the different model types. Zhou et al. also provides a set of factors that are often considered by each agent when choosing an appropriate action: Behavioral, physical, social and psychological. The physical and social factors are most obviously relevant to this project as the agents are influenced by the behaviors of those surrounding them. The psychological and behavioral factors are less obvious but still relevant as each agent will have certain personality quotients that dictate their aggressiveness or passivity within the crowd.

The psychological factors of individuals and the crowd as a whole will be influenced by the work done by Durupinar et al. which provides a framework for the emotional, irrational and seemingly homogenous behavior" of a crowd or a mob [9].

Zhou et al. suggests an evaluation criteria consisting of two main categories: 1) evaluation from the modeler's point of view and 2) evaluation from the runtime point of view [8]. Each category has two sub-categories as explained in the table below.

Crowd Simulation Criteria	
Modeler's POV	**Runtime POV**
Flexibility	Execution Efficiency
Extensibility	Scalability

Table 1 Evaluation criteria for crowd simulations that will be used to evaluate this project. From Zhou et al. [8]

Lubas et al. highlighted the use of cellular automata as the basis for realistic agent-based models of crowd behavior [10]. By allowing each agent to only access the states of their immediate neighbors we can reduce computational costs while still allowing for emergent behaviors. This approach was used within this project to allow emergent behaviors to propagate through the crowd.

Moshpit Specific Simulations

There is a plethora of research delving into the social-psychological phenomena present in a heavy metal moshpit. And while there are many model systems similar to a moshpit, the study of moshpit-specific dynamics still remains a niche field.

The most prominent study by Silverberg et al. analyzed videos of moshpits and noted two types of emergent behavior: 1) diffuse gas-like moshing and 2) vortex-like circling behavior where individuals in the crowd vortex around a central axis [6]. Silverberg et al. (2013) looked at the moshpit under the framework of flocking simulations and simplified the "complex behavioral dynamics of each human mosher to that of a simple soft-bodied particle" resembling a Vicsek model [6].

The Vicsek model is a seminal agent-based model for flocking behavior in which at each time step, the agents in the model assume the average direction of motion of the agents in their neighborhood [11]. The net result is the emergent behavior of collective motion in which the crowd as a whole appears to be moving towards the same point.

IMPLEMENTATION OF MODEL

The implementation of the crowd dynamics model follows an agent-based paradigm. Each agent is a mosher in the crowd and is affected by a set of forces. The magnitude and weighting of that force is dependent on the current state of that agent.

Every tick, each agent first updates their local neighborhood creating a list of nearby agents and storing a reference to the closest neighbor and the location of the middle of the neighborhood. These values will be used later in the force calculations. After the neighborhood is updated, the forces are then applied to each agent. Note that gravity is excluded from this simulation and agents are constrained to the XY-plane within the simulation environment. A future direction would be to include true 3D motion and giving agents the ability to jump up and down in the space.

I will first outline and show the forces present on each agent before explaining the underlying state machine and conditions for state transitions.

Forces Acting On Agents

In choosing which forces to include in this project, consideration was given to representing Vicsek-like forces detailed in the Silverberg et al. paper as well as moshpit forces that are more concerted and less random (i.e. vortexing around a local focal point).

Force	Description of Force
Gas-Like	Random, gas-like motion.
Vortex	Circling around the global origin.
Thrash	Propelled towards nearest neighbor.
To Global Middle	Towards the global origin.
From Global Middle	Away from the global origin.
To Local Middle	Towards the neighborhood origin.
From Local Middle	Away from the neighborhood origin.

Table 2 Description of all forces acting on agents in crowd simulation.

Gas-Like Force

The Gas-like force is calculated by generating a random unit vector and applying that vector as a force on the agent. In the simulation's debug settings, the gas-like force is visualized by red lines as seen in the figure below.

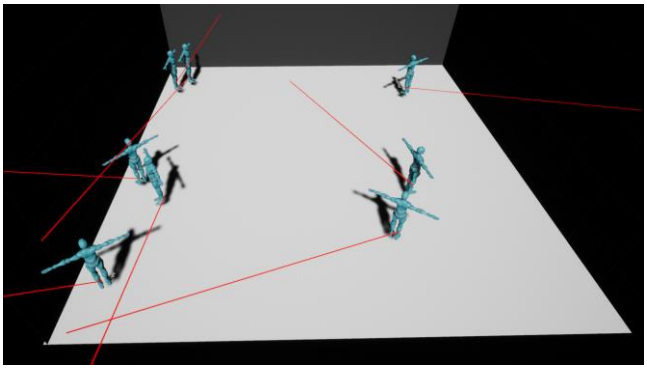

Figure 1 Demonstration of Gas-like force using debug functionality (red lines).

The random vector is not calculated every tick as that would cause unrealistic behavior but is instead controlled by a timer. The time interval is updated globally from the simulation manager object as a variable titled "moshRedirectTimer". A value of 5.0 would indicate that the agents' random direction changes every 5.0 seconds.

Vortex Force

The vortex force is meant to represent the circular motion commonly seen in moshpits where moshers travel around the center of the pit. Vortex force was calculated by taking the cross product of the agent's up vector, and the vector going from the middle to the agent. The resulting vector points the agent in a direction around the middle. The vortex force can be viewed in the debug view as blue lines.

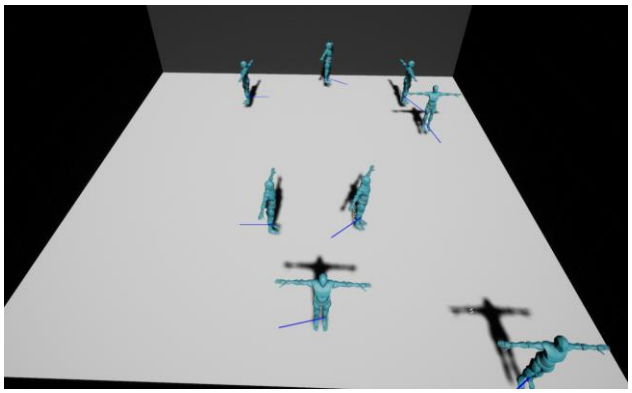

Figure 2 Demonstration of Vortex force using debug functionality (blue lines).

As of now, the moshers will only vortex around the global origin of the simulation. A future direction would be to have the vortex force localized to neighborhoods to allow smaller moshpits to arise from the larger whole.

To/From Global Middle Force

There are two forces related to pushing and pulling the agents to and from the global origin point. These two forces, in conjunction with the aforementioned vortex force add to the circular motion of the moshpit.

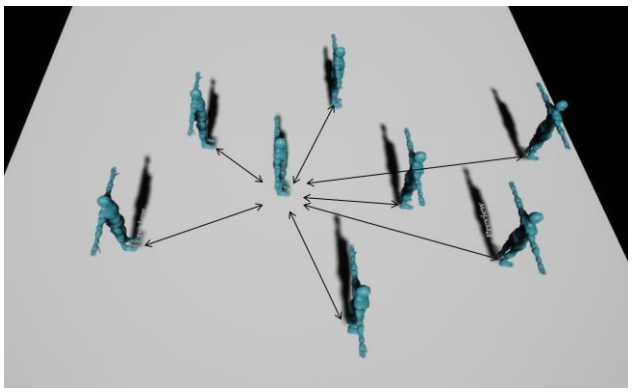

Figure 3 Demonstration of To/From Global Middle forces.

Figure 3 shows how these forces are perceived in the simulation. Note that both of these forces act independently of each other.

To/From Local Middle Force

These two forces act very similar to the "To/From Global Middle Forces" except that the force is pushing/pulling the agent to/from the middle of each agent's local neighborhood. The local neighborhood is defined by a sphere collider surrounding each agent, and its size can be adjusted as a parameter within the simulation manager. The middle point of an agent's local neighborhood can be viewed in the debug mode as a red sphere.

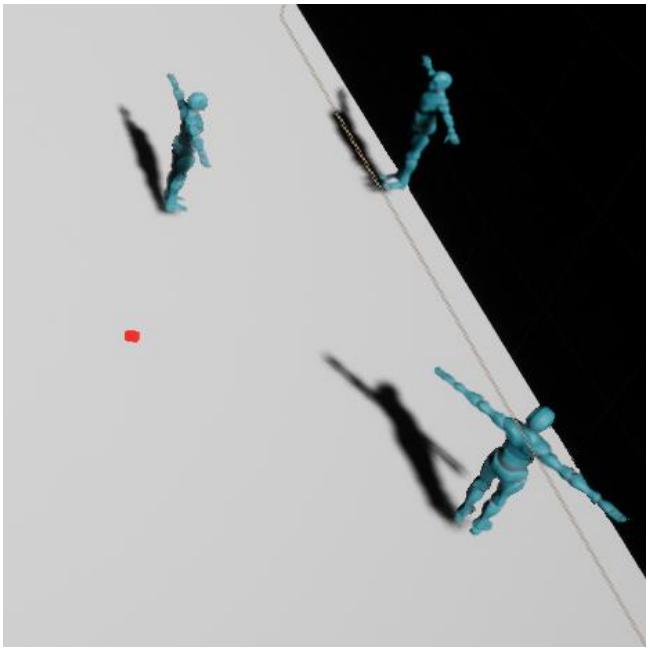

Figure 4 Demonstration of Local neighborhood. Middle point is represented by red sphere

State-Based Modeling

Each force has a corresponding weight and force value associated with it. The weight and force values are different for each state and are what dictate the different behaviors of the different states. (i.e. the 'Vortexing' state has a high Vortex Force and Weight, while the 'Moshing' state has a relatively low value). The force and weight values can be adjusted from the Simulation Manager object as shown in the figure below.

The forces and weights stored in the simulation manager act as a baseline for each agent. Those forces are then modified based on the values of the agent's traits as detailed in the section below.

Figure 5 Demonstration of Local neighborhood. Middle point is represented by red sphere

When an agent changes states it changes its reference to the struct that has the values for that given state.

State transitions are dependent on a two main factors: the intensity of the music currently playing and the agent's traits.

Agent Traits

Each agent has a set of three traits that affect their state transitions. Aggression, Music Feel and Group Think. These traits are used to calculate a set of quotients which will dictate the agent's next state. Once an agent's state has been chosen, that will update the values of weights and forces acting on the agent and then those weights and forces will be modified based on the traits. This will lead to variation within each state so that each agent does not have identical weights and forces acting on them, hopefully resulting in more emergent behavior.

The aggression trait acts as a modifier for the "Thrash" force. The higher an agent's aggression, the more weight will be added to the Thrashing force. The aggression trait also increases the weight of the "From Global Middle Force" and "From Local Middle Force" sending the agent outwards from the middle and away from other local groups. The aggression trait makes the agents act in a more anti-social manner.

Music feel is meant to represent how an individual, at a concert, becomes entranced by the beat of the music. This trait acts as a modifier for the "To/From Global Middle" forces, as well as the "Vortex" force.

Group think represents an agent's propensity to be influenced by its neighbors. An agent with a higher group think is more likely to take on the same state as its neighbors and more likely to follow the movements of its neighbors. The group think trait increases the "To/From Local Middle" force as well as decreases the "Thrashing" force.

These three traits along with music intensity and the agent's speed are used to calculate a set of quotients. These quotients then dictate what the agent's next state will be.

Music Intensity

To factor in the intensity of the music currently playing in the simulation, I used the "Sound Visualizations" plugin included in Unreal Engine 4. This plugin provides a library of blueprints that let users visualize sound waves. For the purposes of this simulation, I only fetch sound waves at 100 Hz to only look at the heavy bass sounds. Bass is usually an indicator of how intense a song is and is typically the aspect of a sound that resonates most with a crowd. The magnitude is made public to every agent, so they can use that data as a modifier for their current state values.

Music intensity is added to aggression to calculate the weight of thrashing.

Main Menu

When starting the simulation, users will be met with the main menu (Fig. 6). The main menu is where users can initialize the agents starting traits, choose how many agents will be in the simulation, set the size of the agents' neighborhoods as well as change the song that will be playing. Since music intensity is a factor in agents' state transitions, choosing different songs will have different effects on agents and hopefully lead to differing emergent properties.

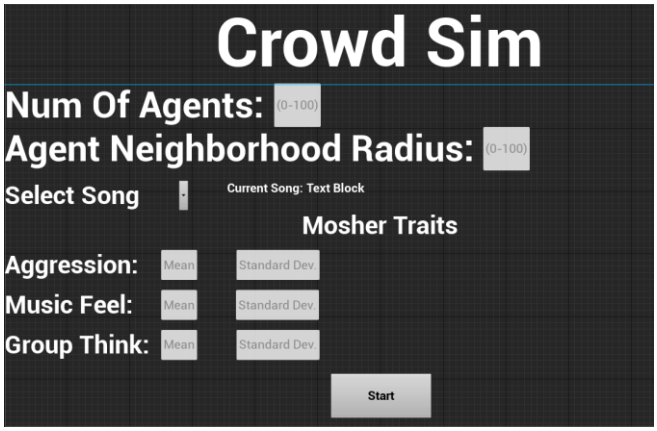

Figure 6 Demonstration of Local neighborhood. Middle point is represented by red sphere

EVALUATION OF SIMULATION

As mentioned in the Introduction and detailed in Table 1, the simulation will be judged on how it is perceived by the modeler, and how it is perceived by the user. Since I was not

able to send this simulation out to other users/modelers to elicit feedback, the evaluation of the efficacy of this simulation will be done through self-reflection and self-criticism.

Modeler's Point of View

Flexibility

From the Modeler's point of view, I tried to include as many ways of influencing the agents as possible. This was mostly done through the values of forces and weights (Fig. 5) that can be customized by the modeler. The expansiveness of these forces tries to capture the behaviors exhibited by moshers. In the current iteration of this simulation, the forces cannot be modified during runtime unless the modeler is running the simulation from the Unreal Editor. A future direction would be to include debug functionality where a modeler can edit these values in real-time making it easier to test out different values.

I did include an in-game debug menu that will let the modeler (or user) to visualize the forces acting on the agents.

Figure 7 In-Game debug menu allows visualization of forces and local neighborhoods.

Fig. 6 shows the in game debug menu (toggled by pressing the 'J' key) which allows the user to visualize the gas-like force, vortex force and final calculated agent vector. It also can show the middle point of agents' local neighborhoods.

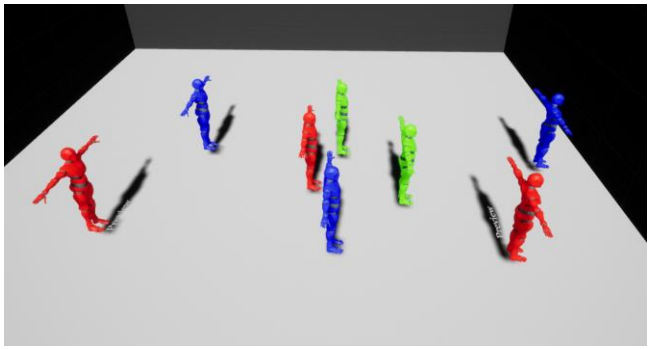

Figure 8 In-Game agents change color according to state. Blue represents "Vortex" state, Red is "Thrashing", and green is "Moshing".

During the simulation, agents will change color depending on what state they are in. This was done to make it very easy for the modeler to see the status of their agents. I originally had plans to give the user more access to the internal data of each mosher by enabling them to click on a mosher and see states such as their velocity and traits, however I ran out of time before I could implement such a feature. One could still see this data in the Unreal Engine editor as these are all public variables, however they are inaccessible during run time.

A desired feature was to let the user choose any song they wanted to play during the simulation. It would have added novelty to the emergent by letting users test out different genres of music to see what behaviors emerge. However, this feature was eventually abandoned in favor of providing the user with a pre-defined set of songs.

To compensate, I did make it easy to add new songs to the simulation. All the user needs to do is import their media into the Unreal Engine editor's content browser and add songs to the list using this array on the Game Manager blueprint.

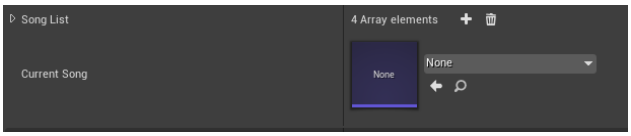

Figure 7 Users can add songs to the simulation through the Game Manager object.

Extensibility

Given the short time frame I had to make this project, extensibility was not a priority. Certain design considerations could have been made to make it easier for future developers to extend on the code base that I created. However, these will all have to be left for future directions.

For example, in its current iteration, the forces present in the simulation are hard-coded. There is not an easy way to add in more forces without going through the code and adding the references in the appropriate places. A future direction would be to set up a framework where a user can easily define new forces and have them automatically added to the mosher's struct and the simulation manager at start up. Given Unreal Engine's excellent customizability, accomplishing a feat is certainly possible however for the purposes of this project I was more concerned about getting the base simulation up and running.

In terms of the agents, each agent inherits from a base mosher class. To create more specialized mosher types, it would be easy enough to inherit from the base class and customize them. The simulation manager and game manager would still be able to register those customized classes and use them in the simulation.

User's Point of View

Execution Efficiency and Scalability

To account for as many agents on screen as possible effort was made to make the agents as lightweight as possible. Agents consist only of a static mesh and a collision sphere to calculate their local neighborhoods. I have been able to spawn about 150 agents on screen without too much lag. 200+ agents are possible, however with considerable slow down.

A future direction would be to re-implement the agents using instanced static meshes. Static mesh instancing reduces the memory footprint of multiple entities on screen, so long as they are using the same mesh. Since every agent uses the same mesh, instancing would have been a very effective optimization tool. However, implementing instancing would require a re-write of how the simulation manager is organized, thus will be left as a future direction.

In its current iteration, this simulation does not generate any data that would be of use to the user. I previously mentioned a possible future direction to calculate risk of injury for concert goers, such a statistic could be of great use for event organizers.

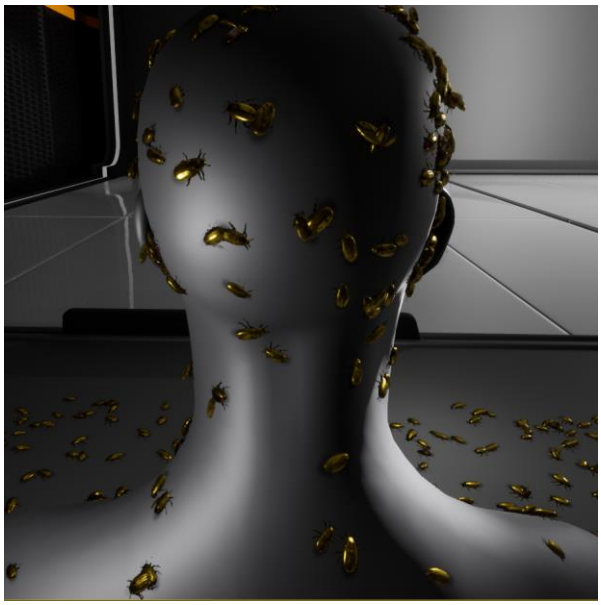

Figure 8 Particle swarms created using Unreal Engine's Niagara particle system. Particles use Avoid Distance Fields and Spatial Hashing to detect other meshes and particles.

DISCUSSION

Original Approach Using Particle Systems

The original proposed implementation for this project was to implement the moshpit simulation using Niagara, Unreal Engine's particle system. Niagara is an extremely powerful tool as it allows the rendering of thousands of particles on screen. The particles are programmable using a combination of Blueprint visual scripting and HLSL. Since the particle computations are offloaded to the GPU, the particles are very performant even when there are thousands of particles on screen querying their neighbors.

I spent the first half of the proposed term learning how to use the full power of Niagara to create responsive mosher particles that react to their neighbors. Unfortunately, progress was slow, too slow in fact that I had to pivot or risk not completing this project by the end of the term.

Thus, the decision was made to instead implement the multi-agent system using the traditional Entity-Component-System. The tradeoff is that the moshpit simulation would not be able to render as many agents as it would if it used the particle system.

Another by-product of this pivot is that I did not have the full amount of time to implement my simulation in the Entity-Component-System. The time crunch led to many shortcomings in this simulation which are outlined in the following section.

Shortcomings of Simulation

A shortcoming of this simulation is that the moshers do not experience fatigue and are not physically affected by forces exerted on them. Moshers do not get injured as they would in a real moshpit environment. A future direction for this simulation would be to include injuries and moshers becoming unconscious on the ground where they pose further tripping risk to other moshers. Including such a feature would also introduce the potential to model cooperation among moshers where a downed mosher could be helped up by a cooperative neighbor.

State Changes

The biggest hurdle in completing this project was figuring out the best way to handle state changes.

My initial idea was to use the mosher's traits and the music's intensity to calculate the likelihood a mosher would transition to a given state. However, most moshers would just stay in the same state that they started in which did not make for an interesting simulation.

Eventually I decided that introducing more randomness could make the simulation more dynamic and increase the chance of interesting emergent behaviors to appear. This is a bit of a cheat, but I believe it does lead to more interesting behavior seen in the moshers.

I would have liked to involve more complex social dynamics such as increasing the likelihood of a mosher transitioning into the thrasher state based on how much they have been bumped into during the simulation.

The most important future direction for this project to be a worthwhile tool will be to make the state transitions more consistent and reflect the social dynamics of each mosher's immediate surroundings.

Physicality of Agents

The most glaring issue with this simulation is that agents do not act with realistic physics. Agents lack momentum and bounce off walls and each other in perfectly elastic collisions. Agents also lack friction.

A future direction would be to hone the physics of the agents so their physics are more realistic and indicative of a real situation.

Emergent Properties

The process for evaluating the emergent properties of this simulation will be to reflect on each of the criterion laid out in Timothy O'Connor's seminal paper on the topic [12].

Supervenience

Each agent is acted on by forces. Those forces are supervened by the state that the agent is currently in. The agent's current state is influenced by music intensity, agents in their neighborhood, and the agents traits. The chain of supervenience that dictates an agent's actions can be seen in the figure below.

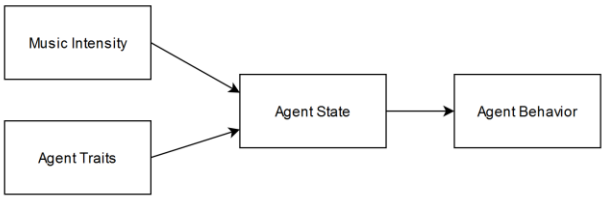

Figure 8 Chain of supervenience seen in the simulation. Music intensity and agent traits supervenes agent state which supervenes agent behavior.

Functional Realization

Agents are functionally realized through the forces acting on them at every step of the simulation. The only action agents have is to move through the world space and collide with other agents and the environment. The only "decision" given to each agent is which direction they will move in and with what magnitude of force.

Nomological or Causal

The motions of the agents are caused by calculations based on their traits and the music playing, thus making them causal. The constants relating to forces and their weights are nomological as they are set by the modeler prior to the start of the simulation. These constants act as the universal constants of sort of the simulation.

Non-Aggrativity

The only place in this simulation where non-aggrativity could be argued is in the agents interactions with their neighbors. As a whole, the crowd does not exhibit behaviors that cannot be observed in an individual agent. Behaviors such as vortexing, thrashing and random gas-like movement are seen both at the individual level and at the crowd level.

The fact that an agent's neighbors can influence its state change means that decomposition of the system's parts can lead to variance. Thus, non-aggrativity is achieved in the sense that an agent's neighborhood affects its next state.

Multiple Realizability

Multiple realizability is achieved in this simulation because there are different internal weights and forces can lead to similar behaviors. For example, two moshers may be in the Thrash state but have different weights and forces as a result of their traits. However, due to their state they will still be thrashing and attacking other agents.

Distinctive Efficacy

I would argue that distinctive efficacy was not achieved in this simulation. Any of the behaviors that can be seen in a crowd of moshers can also be seen in an individual mosher. The efficacy of the whole is not distinct from the components that make it up.

CONCLUSION

A multi-agent moshpit simulation was successfully created using the Entity-Component-System within Unreal Engine 4. The simulation lets users tweak values to produce varying results and see how changes to individuals can change the dynamics of a crowd of moshers.

There is much work left to be done to improve the state of this simulation. More realistic physical interactions between agents and taking social interactions into account when calculating state changes are two of the biggest priorities for expanding the efficacy of this simulation.

REFERENCES

[1] Y. Xiao, M. Yang, Z. Zhu, H. Yang, L. Zhang, and S. Ghader, "Modeling indoor-level non-pharmaceutical interventions during the COVID-19 pandemic: a pedestrian dynamics-based microscopic simulation approach," *arXiv*, Jun. 2020, Accessed: Jan. 21, 2021. [Online]. Available: http://arxiv.org/abs/2006.10666.

[2] J. A. Kirkland and A. A. Maciejewski, "A simulation of attempts to influence crowd dynamics," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2003, vol. 5, pp. 4328–4333, doi: 10.1109/icsmc.2003.1245665.

[3] M. Xu, C. Li, P. Lv, N. Lin, R. Hou, and B. Zhou, "An Efficient Method of Crowd Aggregation Computation in Public Areas," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 2814–2825, Oct. 2018, doi: 10.1109/TCSVT.2017.2731866.

[4] "The most powerful real-time 3D creation platform - Unreal Engine." https://www.unrealengine.com/en-US/ (accessed Dec. 13, 2020).

[5] V. Kountouriotis, S. C. A. Thomopoulos, and Y. Papelis, "An agent-based crowd behaviour model for real time crowd behaviour simulation," *Pattern Recognition Letters*, vol. 44, pp. 30–38, Jul. 2014, doi: 10.1016/j.patrec.2013.10.024.

[6] J. L. Silverberg, M. Bierbaum, J. P. Sethna, and I. Cohen, "Collective Motion of Humans in Mosh and Circle Pits at Heavy Metal Concerts," 2013, doi: 10.1103/PhysRevLett.110.228701.

[7] T. Janchar, C. Samaddar, and D. Milzman, "The mosh pit experience: Emergency medical care for concert injuries," *American Journal of Emergency Medicine*, vol. 18, no. 1, pp. 62–63, Jan. 2000, doi: 10.1016/S0735-6757(00)90051-2.

[8] S. Zhou *et al.*, "Crowd modeling and simulation technologies," *ACM Transactions on Modeling and Computer Simulation*, vol. 20, no. 4, pp. 1–35, Oct. 2010, doi: 10.1145/1842722.1842725.

[9] F. Durupinar, U. Gudukbay, A. Aman, and N. I. Badler, "Psychological Parameters for Crowd Simulation: From Audiences to Mobs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 9, pp. 2145–2159, Sep. 2016, doi: 10.1109/TVCG.2015.2501801.

[10] J. Wąs and R. L. Lubaś, "Towards realistic and effective Agent-based models of crowd dynamics," 2014, doi: 10.1016/j.neucom.2014.04.057.

[11] T. Vicsek, A. Czirk, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles," *Physical Review Letters*, vol. 75, no. 6, pp. 1226–1229, Aug. 1995, doi: 10.1103/PhysRevLett.75.1226.

[12] T. O'CONNOR, "Emergent properties," *American Philosophical Quarterly*, vol. 31, no. 2, pp. 91–104, 1994, doi: 10.2307/20014490.