Human-Inspired Crowd Dynamics Learning for Agent-Based Crowd Control: Designing Intelligent Behaviors Utilizing Unity ML-Agents

Except where reference is made to the work of others, the work described in this project is my own or was done in collaboration with my advisory committee. Further, the content of this project is truthful in regards to my own work and the portrayal of others' work. This project does not include proprietary or classified information.

Dylan Santiago

Certificate of Approval:

Michael J. Reale
Associate Professor
Department of Computer & Information Science

Amos Confer
Associate Professor
Department of Computer & Information Science

Christopher Urban
Lecturer
Department of Computer & Information Science

Human-Inspired Crowd Dynamics Learning for Agent-Based Crowd Control: Designing Intelligent Behaviors Utilizing Unity ML-Agents

Dylan Santiago

A Project

Submitted to the
Graduate Faculty of the
State University of New York Polytechnic Institute
in Partial Fulfillment of the
Requirements for the
Degree of

Master of Science

Utica, New York
April 12, 2024

Human-Inspired Crowd Dynamics Learning for Agent-Based Crowd Control: Designing Intelligent Behaviors Utilizing Unity ML-Agents

Dylan Santiago

_____
Signature of Author

_____
Date of Graduation

Vita

Dylan Santiago started his technology career in 2016, when he attended the Gerard R. Claps Career and Technical Center to learn computer animation. In his final year of the technical program, he gained exposure to game development and participated in a Skills USA video game development competition. After graduation, he started his undergraduate degree at SUNY Canton for Game Design and Development. In late 2020, he transferred to finish his undergraduate degree at SUNY Polytechnic in Interactive Media and Game Design. Dylan always enjoyed tutoring and teaching computer science to students, so he decided to pursue a Master's degree at SUNY Polytechnic Institute to further increase his knowledge. Shortly before graduation, Dylan decided he wanted to specialize in artificial intelligence after developing a 3D survival game.

Project Abstract

Human-Inspired Crowd Dynamics Learning for Agent-Based Crowd Control: Designing Intelligent Behaviors Utilizing Unity ML-Agents

Dylan Santiago

Master of Science, April 12, 2024
(B.S., SUNY Polytechnic Institute, 2022)

45 Typed Pages

Directed by Michael J. Reale

This work presents applications of Unity's ML-Agents framework in the development and design of models for simulating seemingly realistic and optimized crowd behavior. By leveraging machine learning algorithms such as proximal policy optimization, soft actor-critic, and proximal policy optimization with continuous action spaces within the Unity environment, this research aims to create seemingly realistic crowd simulations capable of imitating diverse behaviors and interactions observed in real-life scenarios. By leveraging the functionality of the ML-Agents toolkit, we can seamlessly integrate machine learning algorithms into the Unity game engine. This approach seeks to enhance the flexibility and intelligence of agents within a crowd. Additionally, optimization strategies will be investigated to ensure scalability and computational costs in large and complex environments. This research contributes to applying reinforcement learning frameworks to crowd simulation techniques in video game and real-time environments.

Acknowledgments

I would like to thank each of my committee members. I am particularly grateful to Bruno Andriamanalimanana for his invaluable instruction in the fundamentals and complexities of machine learning and artificial intelligence. His classes made this project possible.

I'd also like to acknowledge the contributions of the following students for their past assistance: Thomas Cunningham and Julian Elmasry.

An additional round of thanks goes to Michael J. Reale for his help with the headache of installing all the necessary packages to get the project up and running.

Style manual or journal used Journal of Approximation Theory (together with the style known as "sunypolyms"). Bibliography follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX2e) together with the style-file `sunypolyms.sty`.

Table of Contents

Chapter 1

Introduction

Crowd simulation is an essential element in several fields, including urban planning, safety analysis, and entertainment. It involves comprehending and reproducing realistic crowd behavior, which is of utmost importance. Precise crowd behavior modeling not only provides a valuable understanding of crowd dynamics, but it also enables informed decision-making and scenario analysis. Due to recent advancements in machine learning methods, there has been a significant increase in the desire to use these algorithms in a wide range of disciplines and domains.

This study utilizes Unity's ML-Agents framework as an effective tool for smoothly incorporating machine learning algorithms into the Unity game engine. This integration facilitates the quick creation of complex intelligent behavior in crowd simulation models, beyond the limits of realism and optimization. The Unity ML-Agents platform enables us to investigate a wide variety of machine learning techniques, such as proximal policy optimization (PPO), soft actor-critic (SAC), and proximal online contextual actor-critic (POCA).

The objective of this study is to develop intricate crowd simulations that can accurately attempt to replicate a wide range of behaviors and interactions found in real-life scenarios. By using the capabilities offered by the ML-Agents toolbox, our goal is to improve the adaptability and cognitive abilities of numerous agents in a group, resulting in more engaging and lifelike simulations. By incorporating these methods into the construction of crowd simulation models, we may attain several advantages, such as facilitating the generation of very adaptable agents capable of seeing and analyzing about 100 activities. These agents have the ability to acquire knowledge and modify their actions in response to external stimuli and feedback, such as dynamically changing barriers.

Furthermore, machine learning techniques make it easier to create simulations of crowds that can handle more complicated settings. This lets us study crowd behavior across a wider range of contexts. Furthermore, these techniques ensure efficient utilization of computational resources, even in scenarios involving a large number of agents, and yield satisfactory simulation results. Through the examination of optimization algorithms specifically designed for these jobs, our objective is to reduce computational expenses while maintaining a high level of simulation accuracy.

Through experimentation, this study contributes to the growing topic of integrating machine learning frameworks into crowd simulation approaches in a variety of settings. This study seeks to expand the capabilities of crowd simulation by utilizing Unity's ML-Agents framework and machine learning techniques. The goal is to create more realistic, immersive, and optimal simulations for a variety of applications.

In the subsequent sections of this report, I present a comprehensive examination of the methodology and theory employed, the precise implementation of crowd simulation models, the configuration of the environment (e.g., including the simulation of complex scenes with crowds of up to 200 agents), the configuration parameters used for model training, and the results derived from the investigations. Below is a list of all the accomplishments of this research:

- Agent RL basic navigation
- Obstacle avoidance with movement adjustments
- Dynamic agent avoidance
- Crowd "flocking" behavior
- Realistic human movement adaptation
- Scalability optimization for large real-time scene changes
- Modularity for creation of diverse scenes and agent modification

- Edge case testing for unique environments
- Training on large scale scenes

Furthermore, this report examines the repercussions of these findings and prospective avenues for additional exploration in this rapidly advancing field.

Chapter 2

Related Work

2.1 Integration of Physics-based and Deep Learning Models

Multi-agent crowd simulation methods that combine deep learning with a traditional special force model were proposed in this 2024 study[3]. This method makes crowd simulation paths more accurate and easy to understand, which lets you make simulations with lots of people and different kinds of crowd behavior. It clarifies that we can categorize existing crowd simulation methods into rule-based and data-driven methods. It leverages deep learning to capture intricate crowd behavior. Their model is trained on modified real-world data, allowing it to more closely resemble real-world features when simulating behavior. Their model can effectively simulate complex iterations and reduce collisions between crowds, but it lacks computational efficiency.

Current physics-based models provide strong reliability and applicability, but they lack accuracy due to the intricate and diverse nature of human behaviors. In addition, deep learning techniques excel at capturing intricate micro-dynamics but face challenges when it comes to generalizing beyond the distributions they were trained on. The authors suggest integrating the advantages of physics-based and deep learning models into a machine learning simulation that incorporates principles from physics [5]. The main concept is to allow these two categories of models to mutually learn from each other in a step-by-step manner, using a combination of physics-informed machine learning and machine-learning-assisted physics discovery. PIML, or Physics-Informed Machine Learning, enables neural network models to enhance their robustness and generalizability by incorporating observational, inductive, or learning biases. These biases are introduced to constrain the neural

network's learning processes and ensure the solutions align with physical principles. Machine learning-assisted physics discovery allows physical models to enhance their ability to characterize complex and heterogeneous crowd dynamics by learning from well-trained deep learning models. They employ symbolic regression to analyze the components of the neural network model. The proposed framework utilizes a neural network model called the Physics-Informed Crowd Simulator (PCS). This model incorporates robust inductive biases by leveraging the social force model and employing a graph to represent pedestrian interactions. The models surpass state-of-the-art simulation methods in terms of accuracy, fidelity, and generalizability. Despite the superior performance of this model in evaluation metrics, its algorithms are complex and necessitate meticulous design and implementation. Additionally, the model heavily depends on the quality and quantity of training data.

2.2 Synthetic Data Generation for Training

Acquiring genuine data from sensors that rely on infrastructure may be very time-consuming. This methodology offers an efficient means of producing synthetic datasets that can be utilized for training models. They present CrowdSim2 [11], a simulation system developed using the Unity 3D engine. The primary objective of CrowdSim2 is to achieve a high level of realism in several domains, including surroundings, weather conditions, traffic, and individual agent behavior. Motion-matching algorithms are used to animate agents, allowing for diverse activities like dancing or fighting. The simulator produces artificial datasets in the style of Multiple Object Tracking (MOT), which includes accurate annotations for assessment purposes. It replicates realistic situations with multiple agent interactions. Although the approach may be used for many tasks beyond people's monitoring, the results are not very encouraging because of a significant deficit in dataset realism. Like any synthetic dataset, it may not accurately represent the intricate and varied behaviors of crowds in the actual world. Creating these datasets necessitates substantial computing resources, which restricts scalability and accessibility for some users.

2.3 Data-driven Models for Movement Prediction

Conventional crowd simulation models that rely on traditional knowledge have difficulties in effectively representing the movement paths of pedestrians, especially at a very detailed level. Data-driven models show potential but often lack versatility when applied to various geometric contexts. A crowd simulation paradigm called Visual-Information-Driven (VID) is suggested as a solution to address these limitations. The algorithm utilizes past social-visual information and mobility data of persons to forecast the velocity of pedestrians at the following time step. The model utilizes data to identify crucial factors that influence pedestrian movement. Subsequently, the aforementioned characteristics are used to train a temporal convolutional neural network (TCN) with the purpose of forecasting pedestrian velocities[9]. These anticipated velocities are subsequently utilized to execute a crowd simulation, which generates prospective pedestrian trajectories. The similarity between this technique and the observed trajectories in controlled trials suggests that the model is very successful in precisely reproducing pedestrian behavior. The model's evasion and comparison rely on precise situations and geometries that have been thoroughly examined in controlled testing. Although it excels in these situations, its ability to adapt to unfamiliar or more complex real-world settings is restricted.

2.4 Deep Reinforcement Learning for Crowd Evacuation

Existing crowd simulation models have many limitations, but by introducing a deep reinforcement learning-based model specifically designed for crowd evacuation in 3D environments, The proposed framework is called HDRLM3D, which consists of three main components: Agent, Environment, and Interactions[4]. It incorporates a vision-like ray perception (VLRP) and a redesigned global or local perception (GOLP) to mimic human perception. This allows the model to capture more realistic environmental information and improve the accuracy of crowd behavior prediction. By utilizing a double-branch feature extraction and decision network (DBFED-Net), the model can process diverse types of

environmental information effectively. This framework is able to effectively replicate the bottleneck effect observed in real-life, crowded situations. It also achieves evacuation times that are the closest to real crowd experiments. This model does not account for certain environmental factors, like emergency situations.

2.5 Summary

In general, these methods tackle certain issues in crowd simulation modeling but lack the capacity to handle large-scale scenarios, adjust to different situations, and take into account environmental factors. Several of these solutions encounter difficulties when attempting to expand for bigger simulations or real-world implementation. Environmental modeling is a significant limitation of these models, since many of them are inadequate or unable to accurately represent emergency conditions and achieve a high level of realism. To conclude, these techniques do not possess the capacity to adjust to a wide range of intricate real-world situations that exist outside of controlled testing conditions. My goal is to tackle the issue of adaptability by integrating reinforcement learning with modular Unity scenes. More precisely, using an agent's capacity to learn from experiences and apply it to new environments over time.

Chapter 3

Method

3.1 Background

Neural networks are the main technique used in machine learning. Here is a breakdown of the main aspects of neural networks and what they are used for in this project: Consider a neural network as the cognitive core of an entity, composed of a multitude of matrices and vectors of numbers. A neural network assigns weights and biases to each connection between its neurons, forming its parameters. During the training phase, the network acquires knowledge of these parameters and adjusts its weights at each epoch to reduce the disparity between expected and actual results. Activation functions heavily depend on the calculation of each neuron, which its input determines. These functions add non-linear elements to the network. Without this, the network would just acquire rudimentary linear associations between inputs and outputs. Thus, we are not learning complex relationships between the data. Loss functions measure the disparity between the network's projected outputs and the desired target outputs. Reducing the loss is the goal of training a network, often achieved through gradient descent. In gradient descent, the chain rule in calculus calculates the derivative of the loss function with respect to each weight or bias. In the weight/bias space, the gradient represents both the direction and size of the most rapid increase in the loss function. We adjust these parameters in the opposite direction from the computed gradient. The neural network's training method consists of sending inputs to the network, computing the loss based on prediction, and using back-propagation to adjust the weights and biases in order to enhance output prediction.

3.1.1 Unity Machine Learning Agents

Unity's ML-Agents toolkit integrates machine learning techniques into Unity. [1] The primary component of this framework is reinforcement learning (RL), which is essential for the development of intelligent behaviors. In essence, this methodology encompasses the use of agents, environments, states, actions, and rewards. Agents are those who engage with an environment and have the ability to make decisions. The agent observes the current situation, makes decisions based on it, and then receives feedback in the form of either favorable or unfavorable rewards. The environment refers to the exterior surroundings, specifically a three-dimensional setting that includes walls, floors, goals, and other elements. States are manifestations of existing circumstances or surroundings. The current reward condition determines an agent's behavior. Ultimately, rewards function as a basic scalar feedback system for the model. When the agent performs desired activities, it is granted positive rewards, whereas less desirable acts result in negative rewards. Using these, the agent will acquire the ability to make choices that result in favorable outcomes. Thus enabling intelligence and learning through reinforcement.See Figure 3.1 for an example of a reinforcement learning model architecture.

3.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines deep learning techniques with reinforcement learning principles to enable agents to learn optimal behavior by interacting within an environment. Fundamentally, DRL involves training artificial agents to make sequential decisions in environments in order to maximize cumulative rewards.

The concept of reinforcement learning revolves around an agent interacting with an environment over a series of continuous or discrete time steps. The agent observes the current state of its environment at each step, chooses an action, receives a reward based on the action, and creates a new observation state. The agent's objective is to learn a policy, from mapping states to actions, that maximizes a cumulative reward over time.

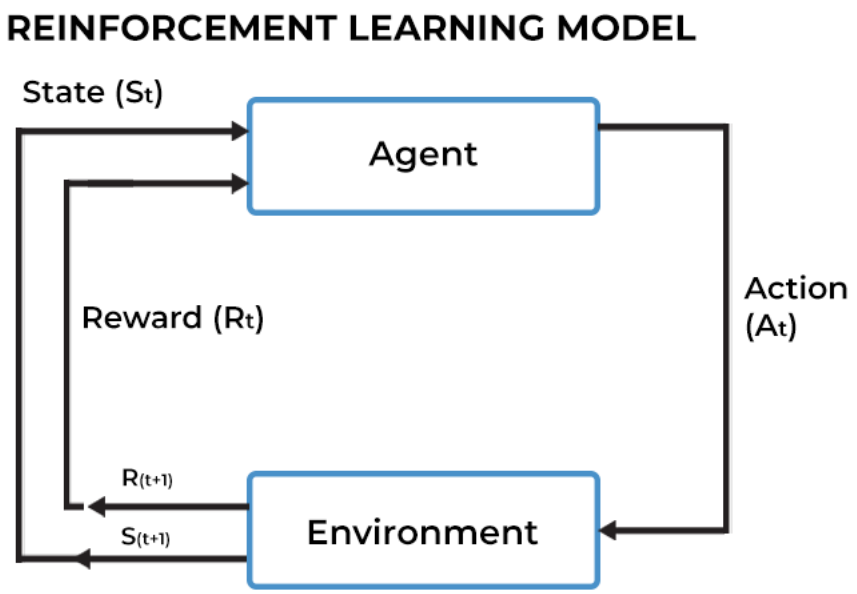


Figure 3.1: Example Reinforcement Learning Framework

Deep learning, on the other hand, involves the use of deep NNs to approximate complex functions, such as the mapping of inputs to outputs. Deep NNs are well-suited for handling high-dimensional data and learning hierarchical representation, making them a natural fit for RL problems where stat and action spaces can be large and continuous. These deep NNs are used to approximate the agent's policy. Gradient-based optimization techniques train them to minimize a loss function, which quantifies the discrepancy between predicted and actual rewards. The ML-Agents toolkit implements these DRL algorithms within Unity, and the combination of its rich graphics and physical simulation capabilities allows us to focus on teaching agents how to simulate certain tasks.

3.3 Architecture

The majority of crowd simulation models significantly depend on real-world data; while this can be beneficial, it lacks flexibility when it comes to adapting to new scenarios quickly. By instructing agents on the fundamental principles of human decision-making in crowd

navigation, we can strive to develop simulations that are more flexible and responsive. I employ a methodology that I have labeled "Human-Inspired Crowd Dynamics Learning."

Perception of the surroundings is crucial for agents to make effective decisions in a crowd simulation. My agent utilizes an efficient observation method that centers around two vital pieces of information: the location of its objective and its present position. By restricting the range of variables that we observe to only these important ones, we greatly decrease the dimensionality of the model. Nevertheless, this method has inherent constraints as it limits the agent's capacity to observe other important conditions inside the environment.

In order to overcome this constraint, I utilize a method within ML-Agents called vector stacking. Through the process of stacking observation vectors, we supply the agent with a historical record of its recent states rather than solely the present state. Say we have a vector of size 3 recording the x, y, and z positions of the agent. If we want to double stack those vectors, we instead send a vector of size 6 to the neural network. The first three elements would be the x, y, and z positions of the previous state, and the next three would be the current state position. This enhancement provides the agent with additional context and temporal data, enhancing its ability to make informed decisions. By incorporating historical observations with present ones, the agent acquires a more rich comprehension of the dynamic progression of the environment. Consequently, this improves the agent's capacity to adjust to dynamic circumstances and make well-informed choices.

Every agent has an action buffer that acts as a source for choosing and performing actions in the environment. The action buffer, with a size of 2 in this instance, is designed to handle the precise movements that the agent is capable of performing. Within this given situation, the agent's movement is restricted to the x-axis and z-axis, resulting in a minimal buffer size. During each time step, the agent fills this buffer with continuous actions that specify the direction of movement to take.

Employing continuous action provides numerous benefits. By granting the agent the ability to choose from a spectrum of decimal values to determine the direction of travel, we facilitate a more seamless motion. Discrete action spaces are most utilized in racing models,

where some actions can be brake, turn left, or turn right. This adaptability improves the agent's ability to navigate intricate situations with greater efficiency. In addition, continuous actions offer more precise control, enabling the agent to make precise adjustments to its movement in response to cues from the environment or other agents.

3.3.1 Environments

Unity environments are crucial in the development, training, and evaluation of intelligent behaviors. They provide a flexible platform that allows for the construction of complex and varied scenarios with astonishing simplicity. The importance of Unity environments lies in their capacity to provide a lifelike and dynamic simulation area where agents may acquire and refine their behaviors. The versatility of Unity settings lies in their ability to accurately depict a wide range of circumstances and obstacles that agents may face in the actual world. We can mimic real-world settings, such as city streets, indoor spaces, and more, allowing agents to learn and adapt to complex and dynamic surroundings. Using Unity's robust tools and assets, I develop a range of testing settings to facilitate agent learning. The significance of Unity settings in agent testing becomes apparent when evaluating the influence of environmental elements on agent behavior. Agents that have undergone training in several contexts have a wide range of behaviors and methods, which are a result of their ability to adapt to the unique difficulties and limitations of each environment. For instance, an agent that has been taught in a straightforward corridor setting could display linear movement, but an agent that has been trained in a winding, crowded pathway might demonstrate movement that seems more organic and realistic. Unity environments provide the capability to modify variables and characteristics, enabling controlled experimentation and precise adjustment of agent performance. Through the manipulation of variables such as the intricacy of the terrain, the density of obstacles, and the presence of environmental factors, I am able to assess the impact of environmental modifications on the behavior and performance of the agent. The iterative process of testing and refining allows for the optimization of agent designs and methods to suit particular tasks and goals.

3.3.2 Basic Navigation

Basic navigation serves as the foundation for effective crowd simulation agents by providing them with the fundamental ability to traverse and interact within their environment in a meaningful way. This foundational skill lays the groundwork for the development of more sophisticated behaviors and capabilities, paving the way for a realistic and immersive simulation experience. This training process is significant in many ways:

1. Incremental complexity: Once agents have mastered basic navigation, we can gradually introduce more complex scenarios and objectives required for more advanced crowd simulation. For example, we can add obstacles or dynamic environments.

2. Transferability: Navigation skills learned during training can be transferred to other tasks within the crowd simulation. For example, agents that can efficiently navigate to a goal are better equipped to respond to emergent situations or interact with other agents in the environment.

Specifically, my agent has received training in fundamental navigation skills by utilizing the physics system included inside Unity. Through this use, the agent is capable of responding more accurately to applied forces. I add motion to each agent from the action buffer, limited to the x-z plane. This allows us to simplify the model for a 2D space dramatically. Thus improving the scalability of the model. By using goal-oriented behavior, it may acquire knowledge of the specific x-z action it must take in each frame. By adjusting the x-z coordinates that are closer to the goal's coordinates, the agent is rewarded, thus instructing it to approach its objective. Given just these criteria, the agent may successfully transition from its present location to a desired target position. While this is not enough to simulate realistic navigation, it provides a foundation for adding the desired behavior we want to achieve.

3.3.3 More Complex Movement

By adhering to the design principle of human-inspired learning, we need to effectively train the agent to avoid obstacles and adjust its speed when approaching. Humans, when approaching an obstacle, naturally slow down slightly to potentially avoid hitting it. We apply this behavior to the agent with dynamic speed adjustments. After each action step, we calculate the adjusted speed based on the closest obstacle distance, the closest agent distance, a specified safe distance, the minimum speed, and the maximum speed. By $Mathf.Clamp(maxSpeed - (closestObstacleDistance + closestAgentDistance - safeDistance), minSpeed, maxSpeed)$

We modify the agent's velocity in real-time using this equation, taking into account its distance from obstacles and other agents. Although this technique is crucial for achieving realism, it is not enough on its own to correctly replicate human behavior. In order to enhance the realism of my navigation approach, we use obstacle avoidance strategies that are influenced by the perceptual processes of humans. Instead of just modifying its speed, our agent must actively avoid obstacles in its path. We replicate human visual perception using ray casting methods, which mimic the way individuals visually perceive their environment, analyze the data, and respond appropriately. Nevertheless, in my approach, we directly alter the action buffer's values to effectively steer the agent away from obstacles. The method starts by projecting a beam at a certain distance in front of the agent. Should the ray connect with an object, indicating the presence of a possible obstacle, we need to calculate a course of action to avoid a collision. In order to make this decision, we use additional rays to the agent's right and left, replicating a human's process of scanning for alternate options. Examining the ray that doesn't intersect with an "Obstacle" or "Agent" item allows us to determine a secure path for the agent to follow. See figure 3.2 for a visual of agent ray casting directions. Once we find the safe direction, we modify the *moveX* and *moveZ* values in the action buffer to reroute the agent's path and avoid the obstacle. The raycast's termination point provides significant positional data, directing the agent towards a course that avoids collisions even if it diverges from the initial intended path. Building

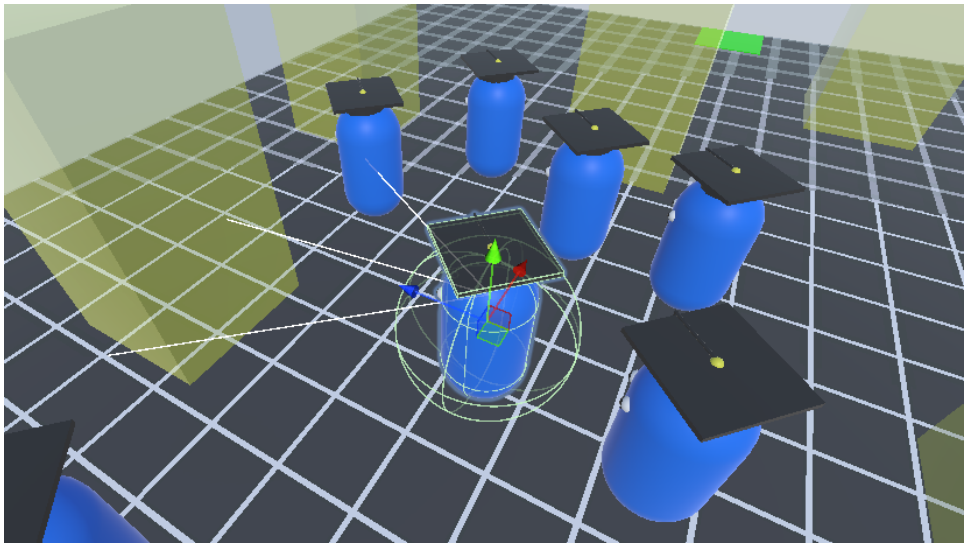


Figure 3.2: Example of Perception ray casts for agent

upon this methodology, we can effortlessly broaden its application to effectively manage the prevention of collisions with other agents present in the surrounding environment. The agent can navigate congested environments with greater realism and agility by using ray casting principles and adjusting the action buffers depending on dynamically observed barriers.

3.3.4 Flocking

We can apply the phenomenon of flocking, often observed in birds, to human crowd behavior. In 1986, mathematician Craig Reynolds devised the Boids algorithm, which accurately replicates the movement patterns of birds. He successfully replicated this behavior using a blend of separation, alignment, and cohesion[12]. First, we need to compute the distance between each individual agent. To prevent overcrowding, we store the separation as a vector that indicates the distance between close agents. We go over all the agents within a specified radius of the agent and compute a separation vector by adding the normalized vectors pointing away from each nearby agent. We modify the agent's action buffer values to travel in the opposite direction from another agent if it is located within the separation radius of that agent. We normalize the generated vector to ensure consistent behavior

across different agent setups. Next, we need to compute the alignment of agents. In order to preserve group unity, the alignment vector represents the mean direction of movement of neighboring individuals in order to preserve unity with the group. Like the separation process, we systematically go over each neighboring agent and calculate the sum of their velocities. If an agent is located inside the alignment radius, its velocity is equivalent to the alignment vector. We normalize the resulting alignment vector to ensure consistent behavior. For a visual representation of alignment, refer to figure 4.4. Finally, we calculate the cohesion vector for the agent. The cohesion vector points towards the center of mass of neighboring agents in order to maintain the unity of the crowd. We iterate over each of the agents and summate their locations. If an agent is located inside the cohesion radius, its location is taken into account while calculating the cohesion vector. Upon summing the locations, I compute the mean position and then remove the agent's location to get the cohesion vector directed towards the center of mass. We once again normalize the resulting vector to ensure consistent behavior. By integrating these elements, we are able to replicate the phenomenon of "flocking," or crowding, as it occurs in the natural world. Although some modifications are necessary, since this algorithm was specifically designed for birds, it is necessary for us to identify the key distinctions in the ways birds and people gather in groups. To accomplish this, we assign specific weight values to each of the components involved in flocking. Generally, people lack coherence while moving in a crowd, particularly if the crowd is small. We compute the number of nearby agents, and if their count falls below a certain threshold, we dynamically modify the cohesion weight accordingly. We can replicate the nuanced crowding behaviors observed in real-world situations by incorporating this modified flocking algorithm.

3.3.5 Rewards

Rewards are essential for agent behavior. They assist the agent in determining which actions are favorable and which are unfavorable, depending on its observations and movements. The agent receives rewards for performing a variety of tasks. Distance is one of

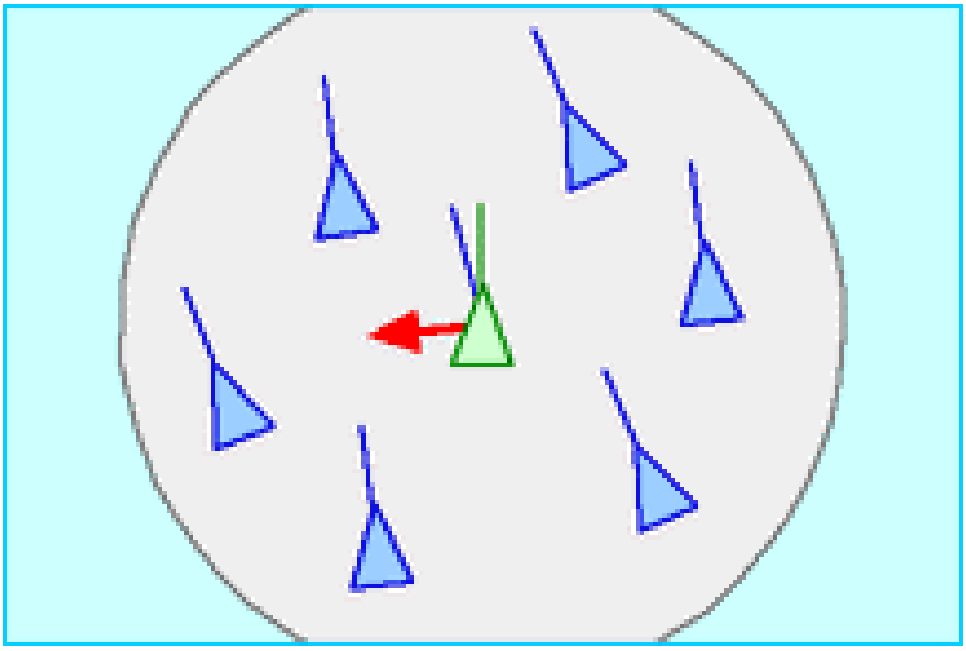


Figure 3.3: Visual example of alignment of agents

the fundamental incentives that directs the agent towards its objective. Every action we take, whether it moves the agent closer to the objective or further away from it, yields a minuscule reward. We intentionally set the minimal incentive to prevent the agent from solely focusing on moving towards the objective without exploring its surroundings. Agents had unpredictable movement patterns throughout their first navigation training. In order to address this issue, I introduced an incentive for fluid motion. This is computed for every given number of frames using their prior velocities. We can ascertain if the agent exhibited irregular movement and impose penalties accordingly. As a result, navigation became much more seamless. The primary purpose of each agent is to achieve the goal. Therefore, we want to provide a very high reward value to reinforce the agent's understanding that accomplishing the goal is a highly favorable action. All other reward values are related to this specific target reward value. In reinforcement learning, reward values are not determined by particular numerical numbers but rather by their relative values to each other. For example, the agent is rewarded for successfully avoiding collisions, striking walls, avoiding

other agents, achieving alignment, and demonstrating cohesiveness. In RL, rewards are maximized and may achieve a maximum cumulative reward. In one of the model training scenarios, we can visually see the agent learning to maximize this policy. Refer to figure 3.4 for a table demonstrating the maximum cumulative reward achieved. The rewards assigned to each frame are sometimes computed as a proportion of the ultimate goal reward amount. Colliding with an obstruction results in a deduction of 40 percent from the goal reward. It acquires the knowledge that colliding with an obstruction is more detrimental than colliding with another agent, as observed in real crowd scenarios. The significance of these incentives lies in their ability to influence the behavior of agents. Modifying any one reward value among these may **significantly** alter the behavior of agents. For example, removing the incentive for achieving a goal would result in the agent remaining stationary and evading other agents instead of reaching its intended destination. The agents' immediate feedback plays a crucial role in determining the optimal course of action.

3.4 Training

I utilized three main training algorithms: PPO, POCA, and SAC. When it comes to training this specific crowd simulation agent, each has its own pros and cons.

Proximal Policy Optimization aims to improve an agent's policy by iteratively adjusting its policy parameters based on the observed reward and state. The utilization of truncated surrogate objective functions guarantees consistent and efficient updates to policies. By imposing limitations on the policy update process, this prohibits significant modifications that could result in policy alteration.

Proximal Policy Optimization with Continuous Action Spaces is an expansion of PPO that integrates continuous action spaces, allowing agents, such as this one, to select actions from a wide range of values. The technique employed is Gaussian distributions, which are utilized to simulate the probability distribution of continuous activities. I observed superior outcomes compared to the standard PPO as I implemented continuous action steps for the agent.

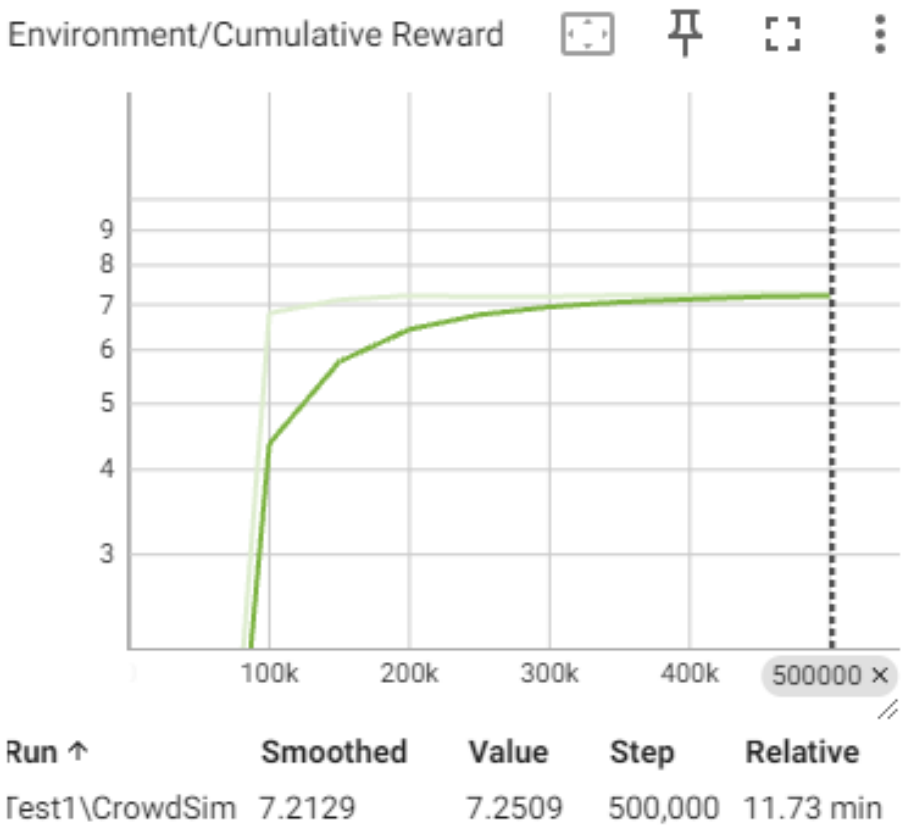


Figure 3.4: Tensorboard example of cumulative reward

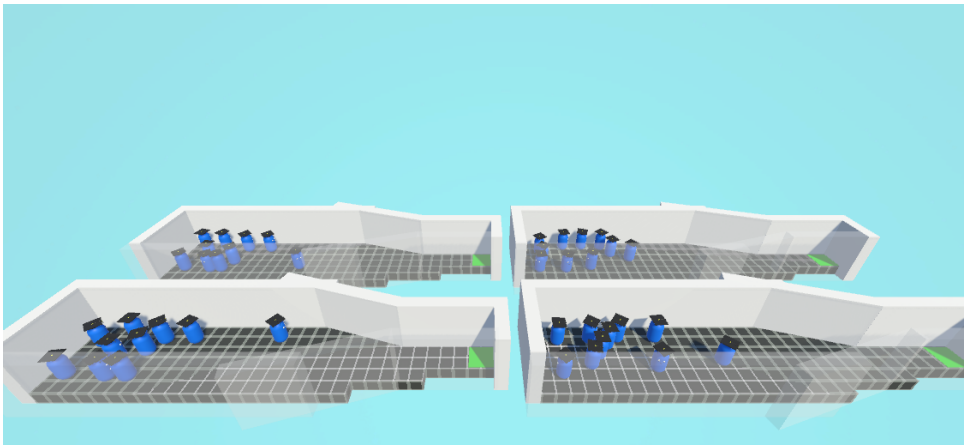


Figure 3.5: Example of Concurrent Environment Training

Soft Actor-Critic leverages the benefits of policy gradient techniques while also capitalizing on the effectiveness of Q-learning. PPO focuses primarily on policy direction revision, while SAC simultaneously learns both a policy function and a value function. The policy has been revised to optimize the anticipated outcomes, while the value function estimate quantifies the projected return from a certain state-action combination. By implementing entropy regularization, which discourages too-deterministic rules through penalties, a wider range of policies can be achieved, especially in contexts with intricate or scarce rewards.

By evaluating these three algorithms, each with its own distinct benefits and drawbacks, we may enhance our comprehension of their impact in certain contexts. Specific algorithms may produce extremely different results when compared to others.

Chapter 4

Experiments and Results

4.1 Models

Below is a list of all the major models trained in my research, along with the most pivotal training parameters utilized. These models encompass various architectures tailored for crowd simulation tasks. Each model was fine-tuned to optimize performance across these different simulation scenarios. In their respective sections, I evaluate and analyze their results based on quantitative and qualitative factors for the environments they were trained in.

4.2 Narrow

A narrow hallway is a common setting used for testing crowd simulation. As previously described, I trained my agent using this scene and obtained the expected results. In terms of visual appearance, there was minimal distinction between PPO and POCA. However, POCA exhibited an average episode duration that was approximately 30 seconds shorter.

Model	Trainer Type	AVG Episode Length
Obstacles1-1	ppo	763
Narrow1-1	ppo	487
LargeOpen1-1	ppo	206
Obstacles2-1	poca	723
Narrow2-1	poca	451
LargeOpen2-1	poca	183
Obstacles3-1	sac	656
Narrow3-1	sac	511
LargeOpen3-1	sac	56

Table 4.1: Table outlining the model names with their main corresponding parameters

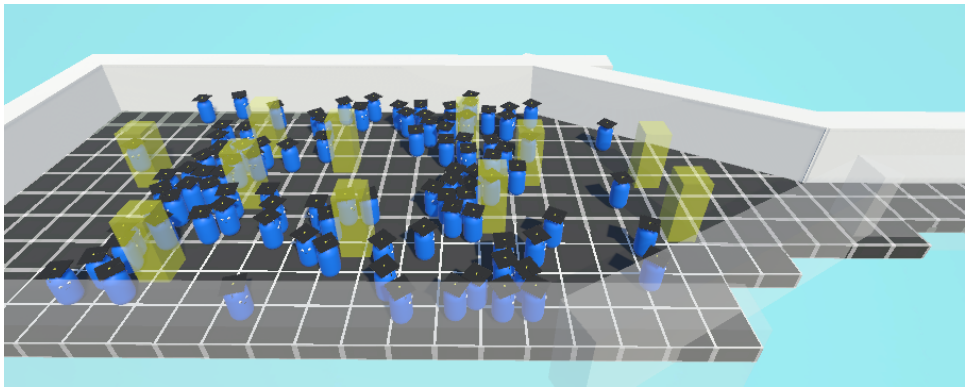


Figure 4.1: Obstacle Environment Testing

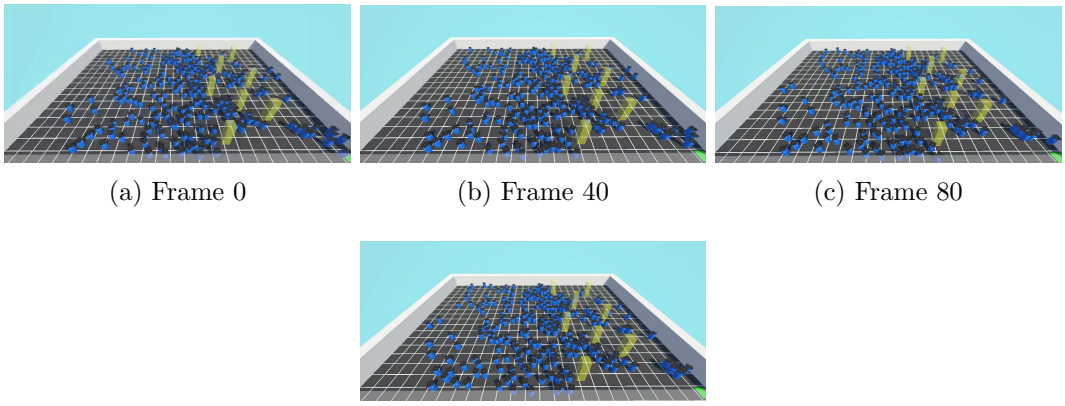
SOCA is an advanced learning algorithm that incorporates elements of PPO and POCA, resulting in diverse outcomes for each scene. To access details about each of the trained models, please refer to the table labeled as 4.1.

4.3 Obstacles

Similar to the narrow hallway but instead consists of several pillars blocking paths to the goal. The results were relatively similar. Visually, the agents have trouble navigating around some obstacles. In specific cases, certain agents may become permanently trapped on barriers; my model's architecture places a higher emphasis on optimizing crowd movement than navigational proficiency. While agents are able to adapt to some specific obstacle placement configurations, other variants do not yield the same results. Refer to figure 4.1 for agent behavior with obstacle placement and 4.2 for frame-by-frame testing examples.

4.4 Edge Cases

As stated before, the model does not adapt well initially to diverse environmental changes. But when we add even more complex navigation elements to the scene, we get an unintended result. These results are expected because of the way the reward functions are calculated and implemented. Since I am using simple machine learning pathfinding methods



(a) Frame 0 (b) Frame 40 (c) Frame 80

(d) Frame 120

Figure 4.2: Example From Large Scale Testing Scene With Obstacles

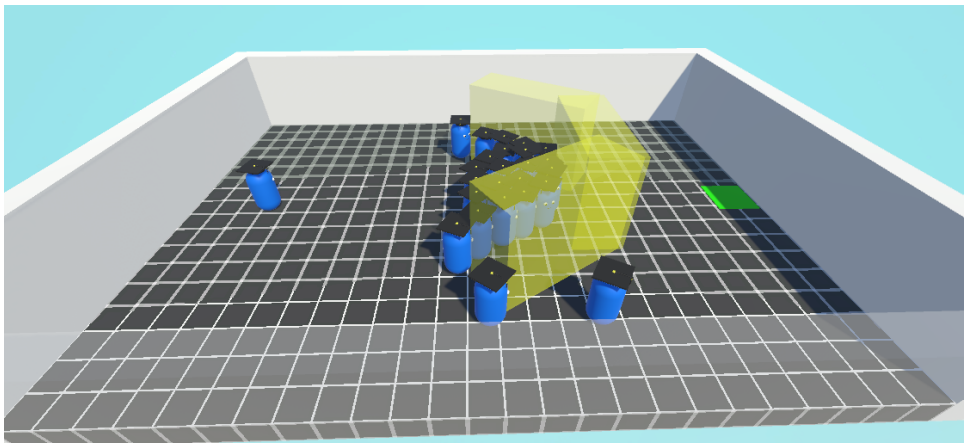
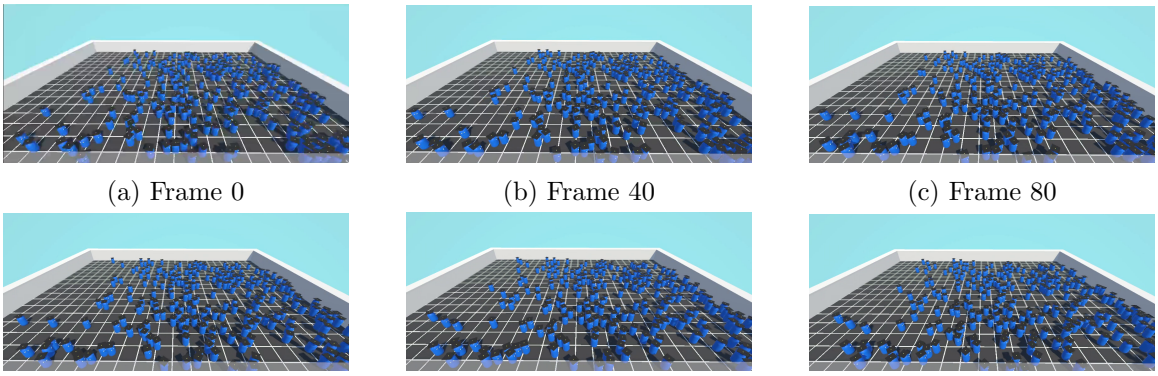


Figure 4.3: Edge Case Testing Scenario

instead of popular A-star navigation, we get varied results. In this specific case of testing with a large U obstacle blocking the center of the exit agents, just get stuck infinitely. Most other state-of-the-art models are also not tested on extremely varying environmental edge cases and instead prompt users to use the common basic doorway funnel scene. To address this type of edge case, we must significantly restructure the way we assign and calculate navigation rewards, or incorporate navigation meshes. For screenshots of edge case testing and training, refer to figure 4.3.



(a) Frame 0 (b) Frame 40 (c) Frame 80

(d) Frame 120 (e) Frame 160 (f) Frame 200

Figure 4.4: Example From Large Scale Testing Scene

4.5 Scalability Testing

Although doing tests with numerous agents usually leads to significantly improved performance outcomes, training more than 20 agents in a single setting can be exceedingly time-consuming. By conducting a larger simulation, we can gain a more distinct understanding of the outcomes of flocking and movement reward parameters. See figure 4.4 for a frame-by-frame time lapse of this testing scene. We are able to easily see the groups and paths that certain agents are forming. Although this simulation has a slight decline in performance after it reaches approximately 250 agents, there are several parameter training adjustments that we can make to potentially address this issue. Machine learning is more of a science that involves extensive empirical testing and modifying small hyperparameters, which can have a big impact on the findings. To view a sample screenshot of my extensive simulation testing, please refer to figure 4.5.

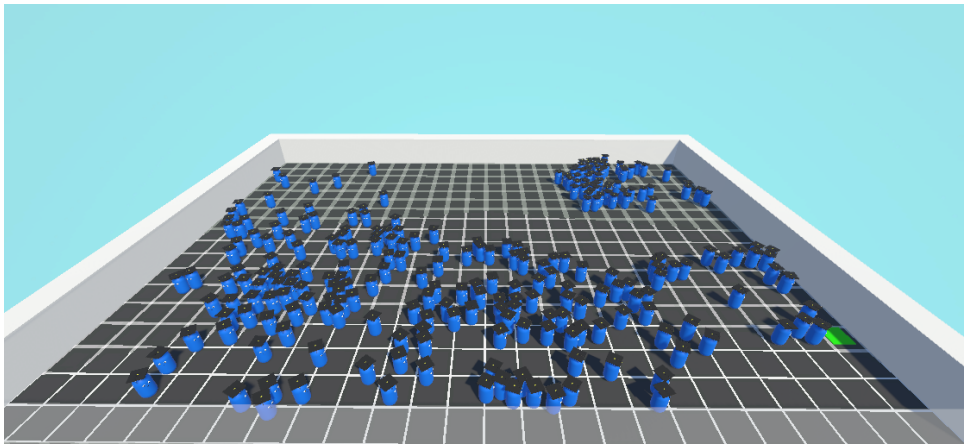


Figure 4.5: Large Simulation Testing

Chapter 5

Conclusion

The crowd dynamics learning architecture I have developed takes inspiration from real human behavior and offers a unique solution to simulate crowd movement. I attempt to tackle the main obstacles faced by existing models, including flexibility, scalability, and real-time adjustments. Despite the need for extensive training for each individual scenario, I have developed an architecture for an agent that can quickly adapt to various situations. This agent provides the necessary framework for significant advancement and understanding of realistic crowd simulation, particularly in its ability to rapidly adjust to various environments, thereby improving the realism and applicability of simulations. This model aims to resolve the inherent challenges in existing models by simulating human basic decision-making processes, which frequently fail to adequately capture the dynamic nature of real-world events. The agent's scalability, adaptability, and modularity provide the potential for advancements by facilitating simulations in many situations with minimal manual intervention. In summary, this work offers a practical approach to developing a model based on real-world human decision-making, potentially addressing the challenges of adaptability and modularity.

Chapter 6

Future Work

As machine learning continues to evolve and push boundaries every day, there is constant experimentation with enhancing the realism, complexity, and applicability of crowd behavior models. There is much to add to my work, and I will go over a few of the key aspects that will improve these models. These areas of focus include exploring training in complex environments, implementing procedural animation to enhance realism, modeling group dynamics and social interactions, and refining movement patterns to more closely match real crowds. These are only a few of the many avenues we can explore to further enhance crowd simulation modeling.

6.1 Complex Environments

It would be advantageous to train and test these models in contexts that are both more intricate and varied. This encompasses various types of land, mobile barriers, and external elements, including atmospheric conditions. Conducting tests and training in environments of diverse complexity allows us to better assess the model's capacity to adapt to various circumstances found in real-world situations. As surroundings get more complicated, there are many issues that arise during training and testing, such as parameter tuning, scene setup, and computing efficiency. Furthermore, achieving authentic interactions between agents and their surroundings may require a significant amount of retraining.

6.2 Procedural Animations

The most simple way to enhance realism within these simulation models is to add procedural animation. This type of animation has become increasingly popular as a way

to quickly create character animations. Mimicking more natural movements of limbs can provide us with insights about crowd behavior in certain settings. Procedural animation enables agents to dynamically adjust motion based on environment factors, such as unusual terrain and crowd density. Unlike pre-defined animation sequences, which we have more control over, procedural ones allow us to easily develop new animations for new actions more quickly. These new animation techniques can be challenging because they require extensive experience in animation programming and math. Furthermore, with an increase in agents, the procedural animations can be a heavy computational load, with hundreds of agents trying to generate new actions for the diverse actions they are taking.

6.3 Group Dynamics

Exploring and closely modeling group dynamics with crowd simulations is vital for capturing collective behaviors observed in real-world situations. For example, individuals form groups, coordinate movements (e.g., leg placement syncing), and influence each other's actions within social contexts. Understanding how group dynamics function in real-life scenarios and applying them to models can help us simulate crowd simulation believability. Modeling these group dynamics involves addressing complex social dynamics that may be difficult to simulate with current technology.

6.4 Social Interactions

By integrating social interactions into crowd simulation models, agents are able to demonstrate authentic social behaviors, including talks, gestures, and adherence to crowd etiquette. These interpersonal exchanges have the capacity to impact the choices and movements of agents. By modeling these specific behaviors, we may enhance simulations with more depth and complexity, resulting in a more immersive experience. To design these social interactions, one must possess knowledge of social psychology, communication dynamics, and cultural conventions. Difficulties may arise when trying to replicate complex

activities, understanding social signals, and ensuring a variety of encounters to prevent repeated patterns.

6.5 Movement Patterns

Examining and perfecting the motion patterns of individuals is essential for creating accurate crowd simulation and navigation. Movement patterns may involve several facets of real-world dynamics, such as the flow of pedestrians, barriers, the act of avoiding them, and the reaction to environmental cues. Gaining insight into various movement patterns is crucial for optimizing crowd navigation efficiency and minimizing congestion, hence boosting overall crowd dynamics. Acquiring and duplicating these varied movement patterns may require substantial gathering and analysis of real-world data. When striving for cohesive movement among a crowd, several problems may occur.

Bibliography

[1] Unity machine learning agents. https://unity.com/products/machine-learning-agents, 2020. Accessed: 2024-4-04.

[2] Unity ml-agents installation documentation. https://github.com/Unity-Technologies/ml-agents/blob/release_21_docs/docs/Installation.md, 2022.

[3] Dapeng Yan, Gangyi Ding, Kexiang Huang, Chongzhi Bai, Lian He, and Longfei Zhang. Enhanced crowd dynamics simulation with deep learning and improved social force model. 2024.

[4] Dong Zhang, Wenhang Li, Jianhua Gong, Lin Huan, Guoyong Zhang, Shen Shen, Juantao Liu, and Haonan Ma. A deep reinforcement learning-based model with human-like perceptron and policy for crowd evacuation in 3d environments. 2022.

[5] Guozhen Zhang, Zihan Yu, Depeng Jin, and Yong Li. Physics-infused machine learning for crowd simulation. 2022.

[6] Hamidrexa Ghahremani and Alex McCarthy and Ibrahim Alhas and Miguel Alonso Jr. *ML-Agents Official Documentation*. Unity Technologies, 2022. Accessed: 2024-03-10.

[7] Vijay Kanade. What is reinforcement learning? working, algorithms, and uses. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-reinforcement-learning/, 2022. Accessed: 2024-4-04.

[8] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison Wesley, 2nd edition, 1994.

[9] Xuanwen Liang and Eric Wai Ming Lee. Visual-information-driven model for crowd simulation using temporal convolutional network. 2024.

[10] Overleaf. Overleaf, online latex editor. https://www.overleaf.com/, 2021. Accessed: 2021-05-19.

[11] Pawel Foszner, Agnieszka Szczena, Luca Ciampu, Nicola Messina, Adam Cygan, Bartosz Bizon, Michal Cogiel, Dominik Golba, Elzbieta Macioszek, and Michal Staniszewski. Development of a realistic crowd simulation environment for fine-grained validation of people tracking methods. 2023.

[12] Craig Reynolds. Boids. https://www.red3d.com/cwr/boids/, 2008.

Appendices

Appendix A

Notes from the Original Style Files

These style-files for use with LaTeX are maintained by Darrel Hankerson[1] and Ed Slaminka[2].

In 1990, department heads and other representatives met with Dean Doorenbos and Judy Bush-Crofton (then responsible for manuscript approval). This meeting was prompted by a memorandum[3] from members of the mathematics departments concerning the *Thesis and Dissertation Guide* and the approval process. There was wide agreement among the participants (including Dean Doorenbos) to support the basic recommendations outlined in the memorandum. The revised *Guide* reflected some (but not all) of the agreements of the meeting.

Ms Bush-Crofton was supportive of the plan to obtain "official approval" of these style files.[4] Unfortunately, Ms Bush-Crofton left the Graduate School before the process was completed. In 1994, we were revisiting some of the same problems which were resolved at the 1990 meeting.

In Summer 1994, I sent several memoranda to Ms Ilga Trend of the Graduate School, reminding her of the agreements made at the 1990 meeting. Professors A. Scottedward Hodel and Stan Reeves provided additional support. In short, it is essential that the Graduate School honor its commitments of the 1990 meeting. It should be emphasized that Dean Doorenbos is to thank for the success of that meeting.

Maintaining these LaTeX files has been more work than expected, in part due to continuing changes in requirement by the graduate school. The Graduate School occasionally has

[1] Mathematics and Statistics, 221 Parker Hall, 844-3641, `hankedr@auburn.edu`

[2] Mathematics and Statistics, 218 Parker, `slamiee@auburn.edu`

[3] Originally, the memorandum was presented to Professor Larry Wit. A copy is available on request.

[4] Followup memoranda gave a definition of "official approval." Copies will be sent on request.

complete memory loss about the agreements of the 1990 meeting. If the Graduate School rejects your manuscript based on items controlled by the style-files, ask your advisor to contact the Graduate school (and copy to the chair) to urge cooperation.

Finally, there have been several requests for additions to the package (mostly formatting changes for figures, etc.). While such changes are not really part of the thesis-style package, it could be beneficial to collect these options and distribute with the package (making it easier on the next student). I'm especially interested in changes needed by various departments.

Appendix B

Code README.md

The Github respository contains all the Unity project files I used for creating and testing the models. Github repo here: https://github.com/notfennecks/CrowdAgentSimulation

B.1 Unity Environments

Unity environments are what I build and test in all my agent models. Below, I outline the three main environments I created to test my models. All of the objects are stored as prefabs in the "Assets" folder of the Unity project. Unity prefabs are self-contained GameObjects that enable the storage of a mix of components, attributes, and settings. In this specific case, the floor, wall, goal, agent, and any other essential items required for executing a simulation are stored.

B.1.1 Corridor

The Corridor is the most basic testing environment. The layout comprises a straightforward hallway where agents must adeptly navigate between several starting points. I have created this environment with the intention of making it readily modifiable and capable of accommodating more agents for future experimentation.

B.1.2 Obstacle

The Obstacle is a more complex testing environment. It consists of an open space with several columns blocking direct paths to the goal. This environment was also developed to be easily modifiable and salable.

B.1.3 Narrow Hallway

The Narrow Hallway tests agents ability to move efficiently when less space is provided. The layout is a simple hallway, but it gets increasingly narrow towards the goal.

B.1.4 U Blocker

The U Blocker is designed as an edge case to push the limits of the model. The layout is a simple open area, but in the center there is a big U facing the agent's starting position.

B.1.5 Large Open

The Large Open environment is designed as a scalability test. During training, I found out that it is easier to pinpoint flocking behavior within a larger open area with many agents. Think of this scene as a subsection of a larger corridor, where agents go from left to right, and when they reach the rightmost wall, they are reset to a random position on the left side.

B.2 Behavior Agent Code

`CrowdSimAgent.cs`

This script file comprises the majority of the agent model implementation. The system includes the distribution of incentives, the application of movement, the creation of a flocking algorithm, and the monitoring and implementation of perception sensors. The method "Initialize()" is responsible for executing once the agent is first imported into the scene. *FixedUpdate()*: executes during each frame of the environment, *FlockingBehavior()*: implements the BOIDS algorithm on the agents, The *adjustSpeed()* function is used to dynamically modify the speed depending on particular criteria. The function *OnActionReceived(ActionBuffers actions)* is called when an agent performs an action. The actions performed are computed based on the values stored in the buffers. The *Heuristic(int ActionBuffers actionsOut)* function is used for manual testing of the environment by directly

modifying the action buffer based on user input. The method *OnEpisodeBegin()* is executed at the beginning of each episode, whereas the method *CollectionObservations(VectorSensor sensor)* is used to transmit observations to the agent.

`CrowdSimManager.cs`

The primary objective of this manager script is to facilitate the management of different groups of agents. This code is irrelevant to understanding the model's architecture and implementation. It counts agents in the scene and adds them to a list of GameObjects.

B.3 Installation and Testing

To evaluate and potentially retrain these models, it is necessary to install the ML-Agents package. The official documentation for installing and configuring the models for testing may be found in the following reference[2]. I am using the ML-Agents package version 3.0.0 and ML-Agents Extensions version 0.6.1. To ensure smooth operation, it is essential to install Unity version 2023.2.8f1, since moving between versions may result in work-related issues.