CROWD SIMULATION BY DEEP REINFORCEMENT LEARNING

Andreas Panayiotou

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

June, 2022

APPROVAL PAGE

Master of Science Thesis

CROWD SIMULATION BY DEEP REINFORCEMENT LEARNING

Presented by

Andreas Panayiotou

Research Supervisor _____
Yiorgos Chrysanthou

Committee Member _____
Panayiotis Charalambous

Committee Member _____
Andreas Aristidou

University of Cyprus

June, 2022

ABSTRACT

The simulation of realistic virtual crowds has been an active research area in several domains, including film industry, video games, engineering, and psychology. The main aspect of visual crowd is the ability of having agents capable of navigating to their goal by avoiding collisions with obstacles and other agents. The authoring part, which is the ways of building and controlling the simulations, is also a key point. During the years, multiple approaches and methods have been utilized for crowd simulation, with remarkable results. With the rise of deep learning methods, reinforcement learning has shown great results in sequential decision-making problems, and it would not be long until it was also applied to virtual crowds. Reinforcement learning is able to produce simulations with complex scenarios by using simple reward functions that control the navigation and behavior of the agents. Thus, this work explores various methods which focusing on how reinforcement learning can be applied to the area of crowd simulation. Specifically, two RL-based approaches are presented, a microscopic which focuses on bringing diversity among agents' behaviors and enables the real-time modification of them, and a mesoscopic approach which concentrates on creating large-scale simulations by combining a list of available components.

Andreas Panayiotou – University of Cyprus, 2022

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Yiorgos Chrysanthou, for the trust he had shown in me and the opportunity he gave me to work on this interesting area of computer graphics.

Second, I am thankful to Dr. Panayiotis Charalambous for introducing me to the field of virtual crowds, for his invaluable support and direction, and the inspiration and assistance he had provided me to submit part of this work to SIGGRAPH, an annual conference on computer graphics.

In addition, I would like to thank my co-authors on the submitted paper, Theodoros Kyriakou and Marilena Lemonari, for their hard work, dedication, responsibility and support.

Lastly, I would like to thank my family and friends for their support and patience during the last months.

CREDITS

This thesis has been contributed to the following publication.

Andreas Panayiotou, Theodoros Kyriakou, Marilena Lemonari, Yiorgos Chrysanthou and Panayiotis Charalambous. 2022. CCP: Configurable Crowd Profiles. In Proceedings of the International Conference on Computer Graphics and Interactive Techniques, Siggraph 2022. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3528233.3530712

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Virtual environments is an area of computer graphics that concerns the research community for years. Realistic environments with high fidelity graphics and human-like simulations is the ultimate goal. Specifically, crowd navigation and simulation is essential in the creation of such environments while they can be found in many other applications including games, movies and training simulators. In recent years, with the raise of the available computational power, artists manage to create impressively realistic 3D models which populate visual environments. However, the ability of animating these models realistically has not evolved at the same rate. This is due to that the movement of a live person affected by various factors including, psychology, environment and health. Thus, these factors adds to the complexity of the problem and many challenges that need to be overcame arise.

Human crowd simulation and animation have been extensively researched thought the years and various works and approaches have been proposed. The majority of works focused on crowd navigation, where agents have to arrive efficiently at a goal position by avoiding collisions with obstacles and other agents. Although, there are many successful works regarding crowd navigation,

the diversity of behaviors is missing, a thing that is important when we have live-like simulation expectations. Likewise, in everyday life people is not just moves with the aim of arriving at their goal; some people may stop to talk to someone else, while others may stand around certain kiosks to buy goods. Thus, the ability to have heterogeneity and behavioral diversity is crucial in crowd simulations if we want to reach the levels of real human movement.

Furthermore, another critical part of virtual crowds is the capability of controlling agents' behaviors and movement from a higher level. We do not only need realistic dynamic movement, but we also have the need of easy-to-use tools that enable users to generate crowds based on their desires and even variate them in real-time.

Artificial Intelligence seems to be a useful tool applied in many areas, while Machine Learning (ML) found a solution to various problems which may were unsolvable using traditional methods. In other words, ML has opened the horizons in various applications and it was expected that many people would be motivated to apply ML in computer graphics, and specifically in crowd simulation. Recently, numerous data-driven approaches for learning behaviour models in the context of crowd simulation have been investigated, the bulk of which are based on Supervised Learning (SL). Reinforcement Learning (RL) is another method that has been introduced by a few researchers in crowd related applications and seems that it has a massive potential. The difference from the traditional data-driven machine learning methods is that, the data are not passed directly as input to the network, but they are simulated during the training phase while agents try to learn a behaviour by maximizing a simple reward function. Thus, would be interesting to further investigate RL related methods with the aim of evolving crowd simulation capabilities.

1.2 Scope of the Work

The scope of this work is the use of Neural Networks and specifically Deep Reinforcement Learning to train models that can be applied to simulate virtual crowds. The main focus of the work is the heterogeneity of behaviors in virtual crowds while other methods will be explored which allows the creation of medium-to-large scale populated environments in a higher level. Likewise, some emphasis will be given on ways that the user can alter and control these behaviors with ease, even in real-time.

1.3 Thesis Structure

This thesis is divided into chapters and sections. First, the Chapter 2 is focused on the presentation of a list of related works divided in four sections, heterogeneity in crowds, global navigation, crowds authoring and reinforcement learning. The Chapter 3 contains a description the main tools that have been used in the implementation of this work. Next, in Chapter 4 the first approach, which is focused in the diversity of behaviors is presented. Every detail about the implementation including methodology, rewards, observations, actions and strategy is described, followed by an evaluation phase which contains analysis, comparisons, discussions and future work. With similar structure as the previous chapter, the focal point of Chapter 5 is the presentation of another RL method which aims in a mesoscopic crowd model. Finally, Chapter 6 summarises this thesis.

Chapter 2

Related Work

Crowd simulation is a topic of computer graphics that has widely researched during the last years, while various techniques have been proposed to evolve the area. Crowd simulation models can be categorised in three groups, microscopic, macroscopic and mesoscopic. First, microscopic models concentrate on the behavior and individual features of each agent, as well as the creation of collective behavior at a global level from local regulations. In this category, agents are distinct objects whose movement is impacted by their surroundings in the environment including other agents and obstacles. They are ineffective for large-scale simulations as usually need high computational power. Second, macroscopic models are utilised in large-scale simulations and are concerned with the larger picture. The movement of agents is guided by fields or flows and is shown as a cohesive and continuous entity. They are incapable of accurately replicating atomic motion. Finally, mesoscopic models are a newly developed division that has garnered the interest of researchers. They are models whose primary purpose is to bridge the gap between microscopic and macroscopic models by combining their respective benefits. Thus, microscopic models are the most appropriate when behavioral diversity is the main goal. In the following sections of this

chapter a list of related works are presented through four categories, heterogeneity in crowds, global navigation, crowd authoring and reinforcement learning.

2.1 Heterogeneity in Crowds

Virtual Crowds that consisting of agents able to act with diverse behaviors have been addressed in literature with numerous techniques. The first works achieve heterogeneity by using predetermined weights [49, 50]. More recent works take advantage of personality traits to simulate diverse behaviors through a list of parameters and constraints that the user can alter to achieve the desired behaviors [9, 27, 13]. Furthermore, a list of works utilise data-driven methods, where data are used to define the simulation model, have been proposed which are using graph-based techniques [30, 29]. Although they produce good results, they lack of variety of behaviors which are found in real human crowds.

Researchers perceived that behavior diversity is a challenging task, thus they had to search for different methods to encounter the problem. A common approach take use of databases that contains a list of paradigms that the agents try to imitate [33, 36]. While this method allows realistic behaviors, is strongly based on the amount and quality of recorded data; larger amount of data improves the diversity and realism of the movements, on the other hand they increases the computational cost. Furthermore, some other works extend the approach mentioned above by adding a list of auxiliary actions to increase realism [37, 66]. Likewise, a more sophisticate work by Ju et al. [21] presents a method that blends existing crowd data, passed to the system as input, to generate a new crowd animation that can be applied to an arbitrary number of agents. Temporal Perception Patterns as state representation is also a method that applied by Charalambous and Chrysanthou. They presented PAG (Perception-Action-Graph) as a way to improve the quality and

efficiency of data-driven methods [3]. Database-based methods undeniably bring noteworthy improvements, however the availability, quality, diversity and amount of input data affect drastically the end results, while the generalisability is limited.

Moreover, some works have been proposed that analyse referenced real world crowd data and compare them over the desired simulated, as a way to evaluate the performance of the simulated models [14, 4, 23, 61, 16, 25]. Wolinski et al. [62] introduce a framework to enable fair comparisons by automatically estimating the parameters that enable the simulation algorithms to best fit the given data. He et al. [16] propose a method that enables the application of real world noisy input data to pilot crowd simulations.

2.2 Global Navigation

As mentioned at the beginning of this chapter, macroscopic and some mesoscopic models guide agents' movement by fields or flows. Early works are based on the computation of velocity fields, extracted by analyzing the geometry of the environment, that govern the global behavior of crowds [64, 48]. Likewise, other works control the simulation by manually designing those fields [5, 65], while others utilise the continuum theory for the flow of crowds [57]. Furthermore, data-driven methods have been applied where researchers implement models that control the group behavior of crowds by using captured videos from real human crowds [36, 44, 6]. Hu et al. [20] propose a method for learning typical flow patterns in dense crowds from videos by formulating these patters as a clustering problem of motion flow fields. Moreover, a list of other works present methods that forces direct agents' trajectories by using a responsive path following scheme [42, 45]. Finally, another approach gives the ability to the user to alter the behavior of groups of agents by interactively editing their formation patterns [29, 56].

2.3 Crowd Authoring

Except from realistic and effective crowd simulations, the ability of supervising these simulations from a higher level and efficiently creating multiple scenarios is also vital. Thus, intuitive tools are required which have to be adjusted depending on the type of simulation that the users want to take control of [35]. First, regarding the navigation of agents in crowds, the majority of works utilise sketching interfaces that enables the user to draw the flows that guide the agents [21, 43]. On the other hand, controlling and authoring the animation and visualisation part of a simulation mostly relies in the manipulation of templates, assets and components [39, 59]. Likewise, another method that extensively found in literature, is based on editing where different variations of a simulation are expressed as handle manipulation. An example of editing is the work proposed by Kwon et al. [29] where individuals can be pinned or dragged to distort a collective motion while multiple group movements may be joined or combined to create a longer or bigger group motion. Moreover, Kraayenbrink et al. [28] presents an interactive crowd editor that provides high-level editing parameters for defining crowd templates; these templates can be applied to various environments. Finally, authoring local motion can be a result of regulating high-level behavior and mostly concerned with coding low-level constraints, like time horizon for the avoidance strategy, in order to manage local movements [26].

2.4 Reinforcement Learning

The growth of Deep Learning, as well as the large variety of applications for which it is being used, led researchers to investigate and apply RL for crowd simulation, as it has been shown that is an effective technique for learning optimum solutions in sequential decision-making problems [53, 55].

Reinforcement Learning is based on Markov Decision Process (MDP) [2, 54]. A RL problem can be described by an environment and an agent that uses a set of observations in order to make decisions and execute suitable actions. The main objective of an agent is to maximise a reward function, during an academic episode, that describes the desirable behaviour of the agent. An academic episode runs until the agent reaches a termination state, for example accomplishes its goal, or a threshold of steps has been passed. Specifically, the agent observes the environment through a set of observations that can be vector, visual or raycast and then, based on that, chooses one of the available actions. Then, the current state of the environment changes and the agent receives a positive or negative reward that evaluates if the action selected was the appropriate for the current state.

One of the first works that presented the potential of RL for crowd related task is by Treuille et al. [58] who train nearly optimal controllers that enables real-time characters animation while avoiding collisions. Then, Peng et al. [47] added to that method by applying it in physically-based characters. Reinforcement learning approaches have been used in several researches to train policies for the domain of crowd simulations [40, 11, 18, 32]. Despite the remarkable results, the majority of works try to describe basic crowd behaviors including goal seeking and collision avoidance, by simple reward functions which manually tune to achieve the desired results [32, 38, 41, 51]. This task consumes too much time and effort, as it based on a trial-and-error procedure until the most suitable values are found; expanding the behavior set of the agents requires readjustments of that values and increase the complexity of the problem. Thus, usually researchers define a final set of values that are well suited for a specific scenario and apply them. Except from the difficulty in determining the appropriate values, this approach often lacks of diversity in agents' behaviors as they are fixed over the whole crowd, while also the high-level control of that behaviors is limited.

Lately, researchers focused on exploring methods to improve the generalizability and include more complex tasks by using policy parameterization approaches. Lee et al. [34] present a new algorithm that learns a parameterized family of motor skills from a single motion clip, while Won et al. [63] propose a method of learning parametric controllers for body shape variation. A more recent work by Hu et al. [19] propose a multi-agent RL approach that learns a parametric predictive collision avoidance and steering policy, while they also present that training over a parameter space produces a flexible model across various crowds scenarios. The works that have been described above introduce a list of control signal as input to the model policy while they use predetermined reward functions with constant weights that describe various behaviors observed in human crowds.

Chapter 3

Technical Review

This chapter lists and describes the different tools and technologies that have been used in this work. Correct selection of tools is an essential process as every tool has its own capabilities, positives and negatives, which will affect the end result. Thus, in the following sections there is a description for every tool used in this work and the reasons that they have chosen over other solutions.

3.1 Unity3D

Unity 3D is simple modeling software which is regarded as one of the most popular game engines in the gaming industry. Although is mostly used for game development, is suited very well to host scientific simulations for various works. Unity is a relatively easy to learn tool, especially if you have a computer science background and some basic understanding of computer graphics fundamentals. It provides support of scripts written either in C# or JavaScript, both have same capabilities, excluding some libraries which may be language-specific. Furthermore, libraries and frameworks can be included to Unity that enable for example, machine learning integration, something that provides many possibilities. Finally, Unity game engine was selected over other

engines as it has a massive community of users, where you can search for support, is very well documented and especially provides integration with ML-Agents Framework which is described in Section 3.2.

3.2 ML-Agents Toolkit

ML-Agents [22] is an open-source library for Unity that enables games and simulations to serve as environments for training intelligent agents. By using an easy to use Python-API, the framework gives the ability to the users to train the agents thought reinforcement learning, imitation learning, neuroevolution, or other machine learning methods. Furthermore, ML-Agents Toolkit benefits both game developers and AI researchers by providing a centralised platform for evaluating developments in AI on Unity's rich settings and then making them available to the broader academic and game developer community.

ML-Agents Toolkit contains four main high-level components, a learning environment, a Python low-level API, an external communicator and python trainers. First, a learning environment consists of a Unity scene which includes the simulation area where the agents use to observe, act and learn. Second, the Python low-level API contains a low-level Python interface which is used for interaction and manipulation with a learning environment. Third, the external communicator is utilised by connecting the learning environment with the Python low-level API. Python trainers are where the machine learning algorithms live, which they are implemented in Python, and they are used for the training of the agents. Finally, this toolkit was selected as it based on Unity and is very well documented. Moreover has build-in reinforcement learning algorithms and provides a number of ready-made sensor components that can be used for the observation set of the agents including, Raycast Sensors, Camera Sensors and Grid Sensors.

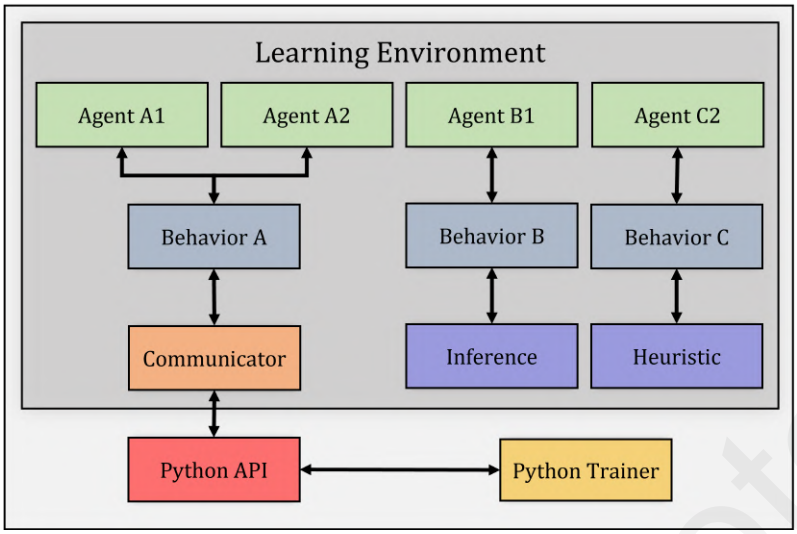

Figure 1: A visual representation of ML-Agents high-level components. Multiple agents can have different behaviors simultaneously including default, inference and heuristic [22].

3.3 TensorBoard

Training a deep learning model is a time-consuming and iterative process, thus it is essential to have a tool to visualise the model's development and monitor the learning process. Tensorboard is provided as an open source toolkit, created by TensorFlow, that gives the ability to the user to track the evolution of the model during training. The toolkit's dashboard shows information including gradients, losses, metrics, and intermediate outputs thought various forms like graphs, histograms, pictures and text [1].

Chapter 4

Microscopic Approach

This approach focuses on bringing diverse behaviors in crowd simulation by learning a configurable crowd model that enables the mixing and learning of several behaviors simultaneously; the presented method uses control signals as inputs to a non-linear learned policy. Likewise, this work gives the ability to the user to adjust, at run-time, the behavior of agents to the desired needs by modifying a list of model's values via simple sliders; change in the model values leads to an immediate, logical and intuitive switch of the behavior of the affected agents.

4.1 Methodology

The proposed method is inspired from the work by de Woillemont et al. [31] who presented CARI: a Configurable Agent with Reward as Input; multiple player styles are being trained simultaneously, using a single training loop, for the needs of a simple two player game. They also show that this method can achieve large variety of behaviors while keeping the performance of the classic reward shaping approach or even outperform it in some cases. The RL approach above could applied for the needs of heterogeneity in crowd simulations, although there are various challenges that have to be addressed including balancing the reward signals for each behavior. Specifically,

Proximal Policy Optimization (PPO) implementation provided by ML-Agents Toolkit [22] has been used for the training of the model. Likewise, the model have been trained using a personal computer equipped with an Intel i5-9600K CPU, a Nvidia RTX2070 GPU and 16GB RAM; the training process lasted around 4 days. In the following sections of this chapter they are presented all the details of the work, how RL has been utilised for the needs of the problem, followed by an extensive evaluation and discussion.

4.2 Rewards

Reward signals are a critical part in Reinforcement Learning as is the mechanism that describes the desired behavior of the agents; models are trained by maximizing the cumulative reward, obtained by a reward function, during the training phase [52]. A reward function is constructed by combining a list of individual rewards; selecting and balancing those rewards is usually a challenging process. Specifically, regarding the crowd simulation scenario there is the need of rewards that describe the real human behavior as close as possible; rewards, that describe each sub-task, have to be assigned for common behaviors including goal seeking and collision avoidance, while in case of more complex behaviors the solution is not trivial.

As mentioned in previous chapters, heterogeneity in crowds is important when we want to achieve realistic results; every person has a different personality and behavior that the majority of the times may be affected by its mood and also environmental factors. This diversity of behaviors is difficult and time-consuming to be described with constant hard-coded rewards. Thus, in this work a different approach is presented while a list of weights are used to describe the importance of each behavior at any given time. Specifically, they have been selected four behaviors a) goal seeking, b) collision avoidance, c) grouping and d) interaction with point of interests (POIs), where each behavior has its own weight. The combination of those weights $\{w_g, w_{ca}, w_{gr}, w_i\}$

describes the behavior of an agent and defines a profile. Each one of those weights has its own range of values, which was selected after a trial-and-error process, as shown in Table 1. The higher a value of a weight, the greater is the effect of the behavior that it describes; for example by setting $w_g = 1.3$ has an effect of a more goal-oriented agent than when is set to $w_g = .2$. Finally, is noted that w_g and w_{ca} takes only positive values as it has been assumed that the agents always have to be governed by a goal seeking and collision avoidance behavior.

Table 1: Value range for every behavior's weight.

Weight	**Range**	**Behavior**
w_g	$[.1, 1.8]$	Goal Seeking
w_{ca}	$[.5, 3.5]$	Collision Avoidance
w_{gr}	$[-3, 5]$	Grouping
w_i	$[-5, 5]$	Interaction

Furthermore, each weight shown in Table 1 is multiplied by the corresponding fixed value, as shown in Table 2 to form the reward signal for each sub-task. Both sparse and dense rewards are used which will be described extensively in the following paragraphs.

Sparse rewards are large positive and negative rewards that are given to the agent when reaching its goal position R_g and colliding with other agents or obstacles R_{ca}, R_{co}. First, regarding R_g, is multiplied with goal seeking behavior weight w_g to describe the importance of arriving at goal position at any given time. Second, following the same logic, collision related rewards R_{ca}, R_{co} are multiplied with w_{ca}, where colliding with obstacles has a larger penalty than colliding with other agents.

Dense rewards are small positive and negative rewards that indirectly guide the agent to achieve the desire behavior. First, $R_g t$ and $R_g a$ are multiplied with goal seeking behavior weight w_g, to reward the agent when moving towards or away from the goal position. In order for an

agent to move towards the goal position, the current distance from its position and the goal point has to be smaller than the one of the previous step, while also the angle $|\theta|$ between its look vector and the vector starting from its position and ends at the goal point, has to be smaller and equal than 45^o. Likewise, a negative reward R_l is combined with w_g to form a living reward that encourage the agent to arrive at goal position as fast as possible when goal seeking behavior is superior to the others. Second, two more complex behaviors are incorporated in the profile of an agent. When grouping behavior is set, agents are enforced to form small groups in the environment; an agent receives positive reward for successfully executing grouping behavior when a) is part of a neighbourhood consisting of less than N_T nearest neighbors, and b) agent's look vector is pointing towards the centre of mass of all agents in the neighborhood. During the experiments $N_T \in [3, 5]$ to enforce small groups, however the user can define the desirable range. Furthermore, regarding interaction behavior, the reward R_i is used in combination with w_i; an agent receives a positive interaction reward when a) its look vector is pointing towards the center of POI object and b) has $N \leq N_T$ agents around it to avoid overcrowding situations.

Table 2: Reward per event with its corresponding weight.

Event	**Symbol**	**Base Reward**	**Weight**	**Type**
Reached Goal	R_g	+1.0	w_g	Sparse
Agent Collision	R_{ca}	-.01	w_{ca}	Sparse
Obstacle Collision	R_{co}	-.5	w_{ca}	Sparse
Towards Goal	R_{gt}	+.00075	w_g	Dense
Away from Goal	R_{ga}	-.00025	w_g	Dense
In Group	R_{gr}	+.001	w_{gr}	Dense
Interacting	R_i	+.001	w_i	Dense
Living Penalty	R_l	-.00015	w_g	Dense

Finally, is mentioned that the ranges of the weights shown in Table 1 are utilized during training to calculate total reward R^t at any given step t as shown in Equation 1; for convenience the

following sections refers to normalized weights, for instance actual $w_i = 0$ corresponds to .5 normalized.

$$R^t = w_g(R_g + R_{gt} + R_{ga} + R_l) + w_{ca}(R_{ca} + R_{co}) + w_{gr}R_{gr} + w_iR_i \quad (1)$$

4.3 Observations

The observation set is the tool that enables the agent to collect information about the current state of the environment. The process of defining suitable observations is crucial for a successful training; they have to provide to the agent all the necessary information in order to be able to make a correct decision, while on the other hand, they must not overhead the agent with ineffective information that add up in the complexity of the problem.

Table 3: Observations of the agent

Observation	**Type**
Local Agent's Velocity	Vector Observation
Current Goal Distance ($\rho \in [0, 1]$)	Vector Observation
Current Goal Angle ($\theta \in [0, 2\pi]$)	Vector Observation
Raycast Sensor $3 * 30$ rays around agent, $7m$ search distance Detecting: Agents, Obstacles, POIs and Walls	Raycast Observation
Normalized Goal Weight (w_g)	Vector Observation
Normalized Collision Weight (w_{ca})	Vector Observation
Normalized Grouping Weight (w_{gr})	Vector Observation
Normalized Interaction Weight (w_i)	Vector Observation

Specifically, in this work have been used two different types of observations including both vector observations and raycast observations as shown in Table 3; also a virtual representation of an agent and its observations are presented at Fig. 2. First, the agent collects information about goal-oriented features including its local velocity, current distance to goal and current goal angle.

Second, a raycast sensor has been used that through the rays that shoots, measures distances to objects (agents, obstacles, POIs and walls) in the environments. Finally, the profile is passed to the agent as observation which contains the four normalized weights $\{w_g, w_{ca}, w_{gr}, w_i\}$.

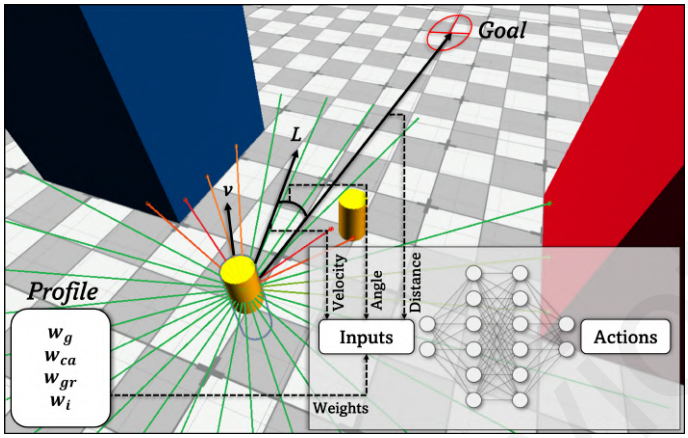

Figure 2: A character from a crowd and its observations [46].

4.4 Actions

In this section the action set which has been used in this work is described. An agent can move in every direction while also is able to rotate. Specifically, seven actions have been assigned including Stand Still (SS), Move Forward (MF), Move Backwards (MB), Move Left (ML), Move Right (MR), Rotate Left (RL) and Rotate Right (RR). The agent is able to move forward with maximum speed of $1.3m/s$, while with speed of $.13m/s$ in all other directions since it is more natural for people to move in a forward direction. Moreover, a Stand Still (SS) action is utilized to give the ability to the agent to stay still in its current position with the aim of executing other behaviors, except from goal seeking. Finally, as the policy defines a living penalty reward (R_l), enforces the agents to select Move Forward (MF) action, when seeking a goal position, as they are moving faster and minimize the punishment that they receive from that reward signal.

4.5 Training Strategy

This section presents the training strategy of this work by describing how observations, actions and rewards have been used, while also shows the process of designing an environment suitable for a successful training. First, the implementation of PPO provided by ML-Agents Toolkit has been used in combination with a Fully Connected Neural Network consisting of two hidden layers each having 128 nodes. The actual configuration parameters that have been used are shown in Table 4.

Table 4: Default values for configuration parameters used in training

Parameter	**Value**	**Description**
Learning Rate	3e-4	For Gradient Descent Updates
γ	.99	Discount factor
H	15000	Maximum steps per episode
Epochs	3	Training Epochs
Batch Size	1024	Batch Size
Buffer Size	10240	Buffer Size
β	5e-3	Entropy Regularization Strength
ϵ	.2	Divergence Threshold

In order to execute a successful training, a suitable simulation environment is mandatory; Fig. 3 shows the training environment, blue objects are obstacles and walls, red objects define the POIs and yellow cylinders represent the agents. During training agents spawn randomly in one of the four corners of the environment while they randomly set as destination position one of the rest; the places of POIs change randomly over a period of time. Parameters related to the agent and simulation are shown in Table 5.

Table 5: Default simulation parameters used in training

Parameter	**Value**	**Description**
r	$.5m$	Agent Radius
R_s	$7m$	Maximum search distance
T	$.04s$	Simulation step

Furthermore, the training phase is divided into episodes; each agent is trained during its own episode which starts by the time the agent spawns in the environment and ends when the agent collides with an obstacle (not other agents), when it arrives at the goal position or a predefined number of steps has passed. After an episode ends, the object of the agent is destroyed and a new agent is initialized again in a random position, while the rest of the agents already in the environment continue the training.

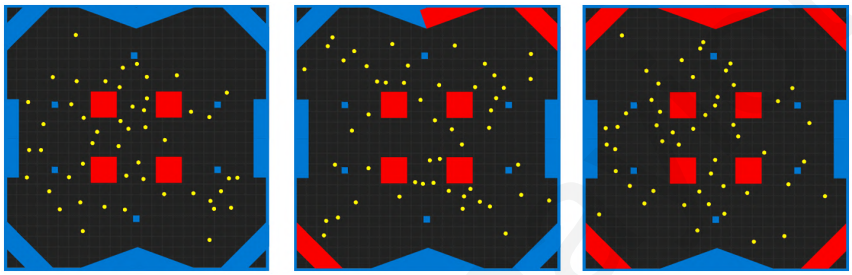

Figure 3: The training environment with alternations between obstacles and POIs [46].

Moreover, a curriculum-based approach has been used to train the agents, where the difficulty of the environment is increased gradually. First, the training process starts with a small number of agents while the number increases to reach the maximum of 75 agents. Second, regarding the various behaviors, it would be very difficult for the agent to distinguish them if were mixing up from the beginning. Thus, the weight of every behaviour is randomized while keeping the weights of the others constant in order to enable the agents to understand how different values of a single weight affects the specific behavior. Note that each change on the weights is kept constant for several episodes instead of changing them in every episode. Likewise, in the beginning of randomization of individual weights, only values near to minimum and maximum values of the range are used before the whole spectrum is covered; this also helps for the agent to understand faster the behavior. Finally, when the agent successfully discovers the various behaviors and

manages to adjust to the fluctuations of the weights, the training continue to the final phase where all the weights are randomized to train the agent with every combination of them.

4.6 Experiments and Evaluation

This section is focused on the evaluation of the trained model through a list of various methods described in the following subsections. Evaluation and results are based on multiple statistics including agents' speed, density(number of agents in an area), distance to the nearest neighbor (DNN) and distance to the closest point of interest (DPOI). The different experiments are took place in four different simulation environments, as shown in Fig. 4; statistics are collected only inside blue regions where agents inherit the different profiles. First, Hall environment simulates a bidirectional scenario where agents spawn in one of the two spawn areas and have as destination a random position in the opposite side of the hall. Similarly, the 4-Way Crossing environment consists of four spawn areas where again, agents have to move to the opposite side. Second, the Circular environment is utilised to spawn agents in the perimeter of a circle while setting as their goal position the exact opposite point on circle's perimeter. Finally, the Museum environment is used as a demo scene and simulates a large scene consisting of 400 agents with different distribution of behaviors.

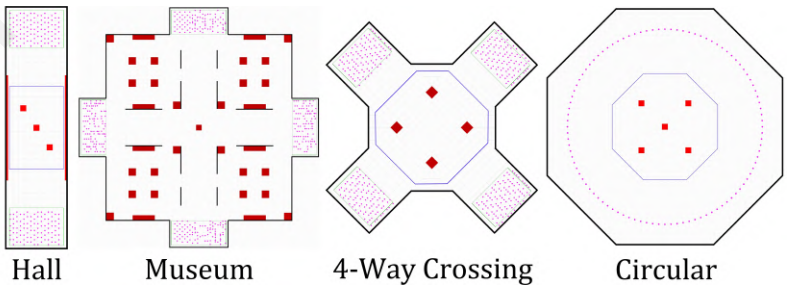

Figure 4: The various simulation environments [46].

4.6.1 Sensitivity Analysis

The following subsections 4.6.1.1 and 4.6.1.2 describe the results of experiments regarding variations of values for weights and density respectively.

4.6.1.1 Weight Sensitivity

This type of experiment is utilised to extract information about how different values of weights affect each one of the behaviors goal seeking, collision avoidance, grouping and interaction. Specifically, the weight of every behaviour is gradually increased in four different normalized weight values 0, .33, .67 and 1, while the weights of the other three behaviors are kept constant to .5; the profile of every run is assigned to all agents in the scene. Moreover, the experiment described above is executed on all three environments, Hall, Crossing and Circular, considering both small (6) and large (75) number of agents. Results are based on three metrics speed, distance to the nearest neighbor (DNN) and distance to the closest point of interest (DPOI); Fig. 5 demonstrates the effect of different values of each weight with 75 agents in the Hall environment, while also shaded areas indicate the standard deviation from the mean.

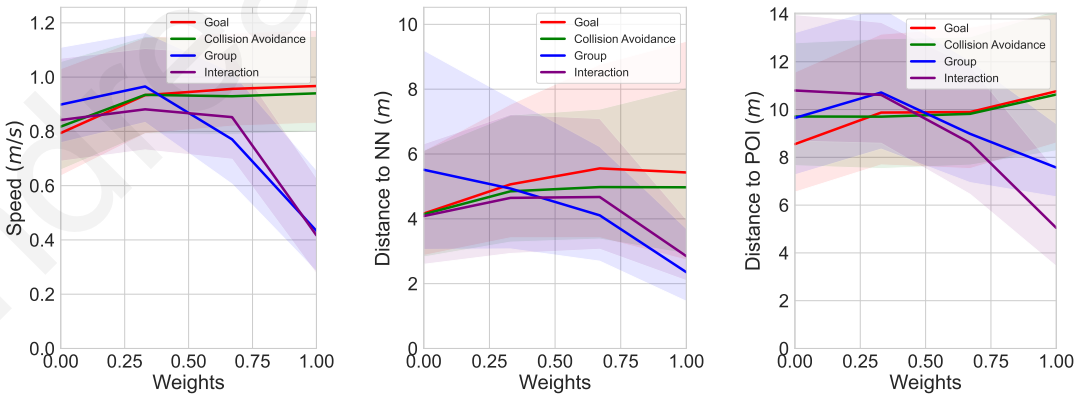

Figure 5: Weight Sensitivity in the Hall environment. Metrics include Speed, Distance to NN and DPOI [46].

Goal seeking behaviour is one of the more crucial tasks when simulating human crowds. As shown in the left chart of Fig. 5 when w_g increases, the average speed of the agents is also increased, as they become more goal oriented and prefer to move towards the goal position, as fast as possible, avoiding grouping with other agents or interacting with the environment. Notice that there is some deviation from the mean, which happens because the weights of the rest of the behaviors are set to .5, a situation that makes the agents phase a mixture of behaviors; in some cases, agents may slow down to avoid a collision or execute some minor grouping or interaction with the environment. Furthermore, the value of DNN and DPOI are also increased as w_g is rising; that indicates again that agents start to ignore the other behaviors and set as main task the arrival at the destination.

A similar situation happens with collision avoidance where as w_ca increases agents start to move faster, as they keep larger distance to other agents. Likewise, they stay further from POIs both to avoid collision and ignore the interaction point. Deviation from mean, again, is due to the value of the rest of the weights and the mixture of behaviors.

Moreover, the increase of grouping behavior's weight w_gr has a significant effect on the average speed of the agents as they slow down to form groups with other agents. In addition, DNN is also decreasing because agents stay in groups near to their neighbors, for a longer period of time. The value of DPOI is slightly decreases too, as because of $w_i = .5$ several agents are forming groups near to POIs executing a mixture of both grouping and interaction with them. The effects of different values of w_gr in Hall environment are shown in the bottom side of Fig. 6.

Finally, regarding the Interaction with POIs, similar results to those of the Grouping behavior are extracted. As the value of w_i is rising, agents move slower to interact with POIs in the environment. The distance to both agents and POIs is decreased as an effect of agents stay around

POIs, near to each other, for long period of time. How different values of w_i affect the movement of agents in Circular environment is shown in the top part of Fig. 6.

As demonstrated by the evaluation results of that experiment, the profile of the agents is strongly affected by the different values of $\{w_g, w_{ca}, w_{gr}, w_i\}$. This method brings diversity of behaviors in virtual crowds with a single training, while it enables real-time modification and control of agents' profile by adjusting the weights of each behavior to achieve the desired results.

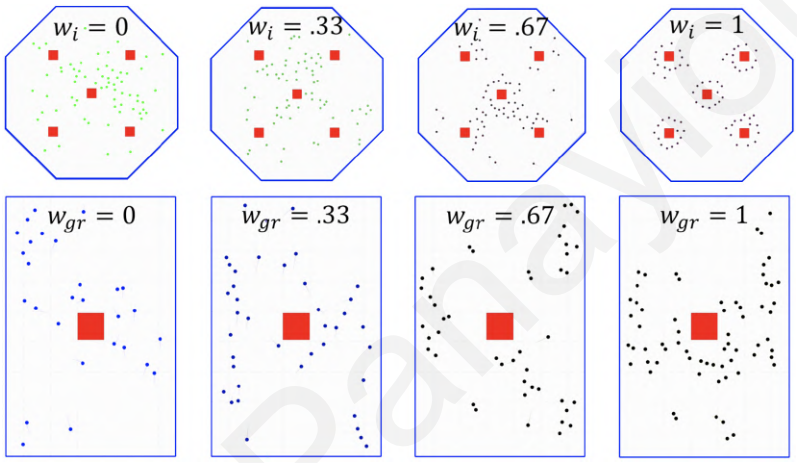

Figure 6: Effect of weight changes for the Interaction (Top) and Grouping (Bottom) Behavior [46].

4.6.1.2 Density Sensitivity

This section contains the experiment that examines how the model behaves with different number of agents. As mention in section 4.5 the model has been trained with maximum of 75 agents. This experiment, stress test the model by examining its generalizability using a range of 50-300 agents; specifically, a set of constant ideal weights is set for every behavior. Generally, as shown in Fig. 7 the system is able to reflect the effects of each behaviour even with a large number of agents, outside the training limits.

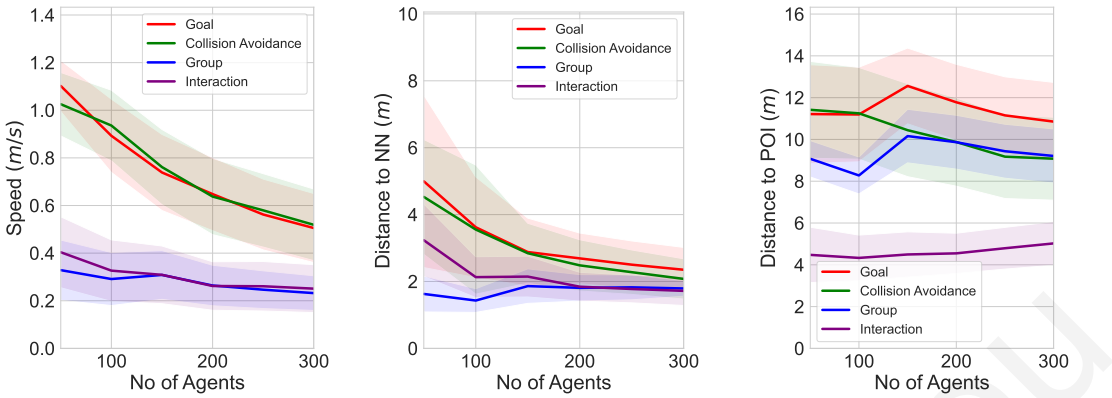

Figure 7: Density Sensitivity in the Crossing environment. Metrics include Speed, Distance to NN and Distance to POI [46].

First, the speed of the agents is significantly decreased in goal seeking and collision avoidance, something that seems logical and expected, as more agents are moving in the same space, however they are still able to avoid collisions while they are moving towards their goal destinations. The speed of grouping and interaction behaviors, on average, seem to remain constant as agents continue to form groups or interacting with POIs, regardless the density. Second, distance to Nearest Neighbors (DNN) is decreasing for all behaviors, except grouping. This is also an effect of more agents moving in the same space, something that enforcing them to come closer to each other, while regarding the grouping behavior stays almost constant as agents were already near to each other, regardless the density. Finally, the results of the experiment shows that different densities are not strongly affecting DPOI in the case of interaction behavior, something that make sense as agents just need to stay around POIs. Moreover, DPOI face some decrease as density rises in collision avoidance behavior, while in grouping and interaction generally remains constant with some fluctuations. The effects of different densities in Hall environment are demonstrated in Fig. 8 for every behavior.

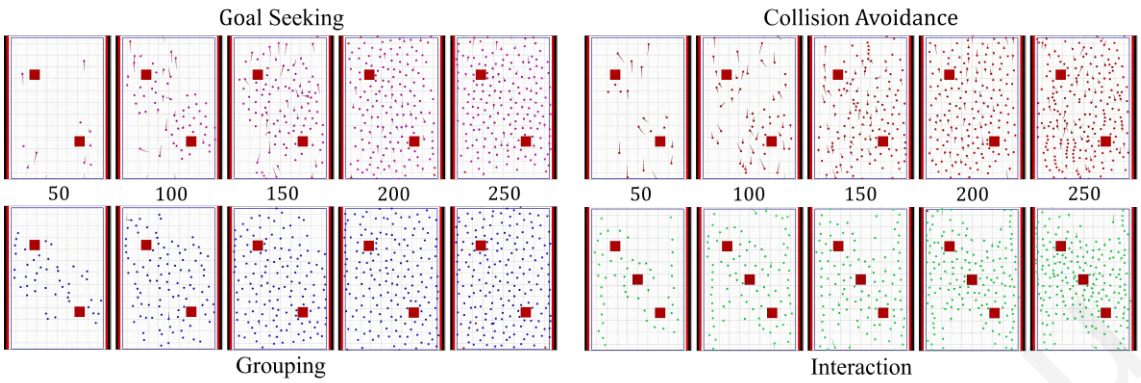

Figure 8: Effect of different densities in Hall environment [46].

4.6.2 Comparison with Power Law and Real-World Data

This section presents the comparison of the trained model, named CCP, against simulated crowd data from Power Law (PL) [24] and real-world captured data (RW) [36]. Real world data are a result of two different datasets, Zara which represents a medium density bidirectional flow on a sidewalk, and a more dense dataset University, which consists of students walking in a university environment; the two dataset combined contains 25 minutes of captured data with 1317 tracked trajectories. The simulated data from CCP (34 minutes) and PL (32 minutes) consist of 2800 trajectories in a bidirectional scenario without any obstacles, while the number of agents varies between 50 and 350. Moreover, both simulators had the same parameters, agent radius was set to ($r = .5m$) and neighborhood radius to ($R_s = 7m$); for CCP the following weights have been used $w_g = 1, w_{ca} = 1, w_{gr} = 0, w_i = 0$ to represent the ideal goal seeking and collision avoidance behavior. The details of each dataset used in the comparison are shown in Table 6.

Table 6: Parameters of dataset used in Fundamental Diagram.

Dataset	**Description**	**Trajectories**	**Duration(min)**
Zara University	Medium Density, Bidirectional Flow High Density, Multidirectional Flow	1317	25
CCP	50-350 Agents, Bidirectional Flow	1400	34
Power Law (PL)	50-350 Agents, Bidirectional Flow	1400	32

The Fundamental diagram presented in Fig. 9 demonstrates the speed(m/s) of agents related to the density($agents/m^2$). The equation by Helbing et al. [17] has been used for the calculation of total density; d_a^t is the local density of an agent a at time t, where $\mathbf{p}_a^t$ and $N(a^t)$ are the position and all neighbors of agent a at time t, as shown in the following equation:

$$d_a^t = \sum_{i \in N(a^t)} \frac{1}{\pi R_s^2} exp(-||\mathbf{p}_i^t - \mathbf{p}_a^t||^2 / R_s^2). \tag{2}$$

As the results show in Fig. 9, generally the speed of agents decreases as density rises in both real word and simulated data. Notice that both CCP and PL have similar trend, while on the other hand the RW data produces richer results because of the natural movements of humans; this open the doors for further exploration and research.

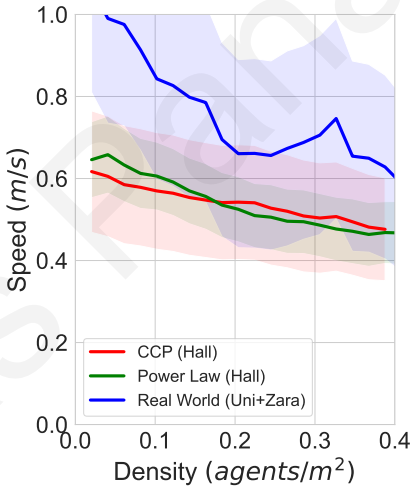

Figure 9: Fundamental model for current model, Power-Law and Real-World Data [46].

4.6.3 Comparison with Baseline Collision Avoidance RL Model

This experiment compares the CCP model to a simple model that has been trained only for the basic crowd behaviors, goal seeking and collision avoidance. In order to compare the two models,

the normalized weights for CCP have been set to ($w_g = 1, w_{ca} = .5, w_{gr} = .375, w_{ca} = .5$). Visually, the two models produce similar results while the Baseline model seems to be slightly more efficient, as the agents does not get affected by environmental elements, such as POIs. However, the comparison results shows that with a simple manipulation of weights, end results can be comparable, while optimizing the goal seeking behavior, to be more intelligent, would be beneficial and also indicates directions for future work.

4.6.4 Weights Outside Training Limits

Section 4.6.1.2 presented the generalizability of the system in the aspect of density; this section evaluates how the system behaves when weights that have not been seen during training phase are used. Similar to the experiment in Section 4.6.1.1, the weight of every behavior takes various values while keeping the weights of the other behaviors constant. In the Fig. 10 are presented statistics, taken from Crossing environment, for Speed, DNN and DPOI for weights values inside the range [-4,4]; note that first, the weights used during training have been normalized to the range [-1,1] and then, the experiment is based on runs with weights two and four times outside the training bounds. The general observation for all three statistics is that, as the weights are assigned to values further outside the training range, they strengthen the effect of the behaviors in some cases, where in other cases they remain almost constant; generally agents are still able to execute the various behaviors successfully.

First, regarding the average speed of the agents, the results show that all the behaviors are strengthen in the range [-2,2], while in larger values there are minor effects; grouping behavior seems to benefits speed for smaller values, this may be due to the fact that agents try to strongly avoid grouping with other agents which may enables them to not slow down as much.

Second, the range [-2,2] seems to positively affect grouping and interaction for DNN statistic, while further values leave them unaffected. The trend of goal seeking shows a decrease in DNN for values larger than 1; this may be due to agents have a too goal-oriented profile that enforces them to find their goal destination, as fast as possible, leaving collision avoidance secondary.

Regarding DPOI metric, the average goal behavior distances stay unchanged, as such distances have no direct influence on the goal-seeking abilities of agents. Furthermore, lower extreme weights seems to positively affecting collision avoidance behavior, as a reduction in the corresponding weight correlates to a decrease in DPOI, showing that agents cannot reliably avoid obstacles. Finally, the distance to POI is decreased for lower weights, in grouping and interaction behaviors, while the distance remains fairly constant as the weights increase.

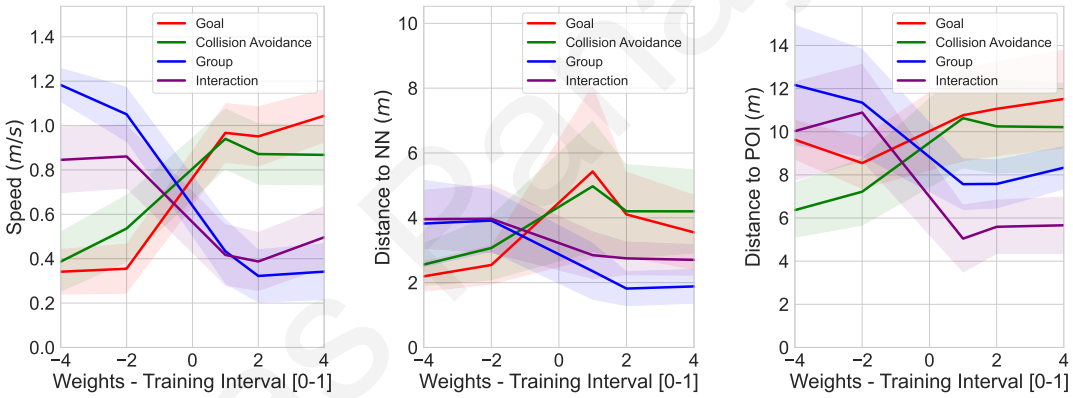

Figure 10: Statistics for extreme weights outside training bounds. [46]

4.7 Demonstration in Museum Environment

As mentioned in Section 4.6 and shows in Fig. 4, a museum environment has been developed to test the performance of the model in a larger scale. Likewise, the museum scene has been utilized not only to demonstrate the various behaviors of the agents, but also to show the authoring aspect; in the provided video of the submitted work various simulations, with different profile

distributions are presented. The model is capable of reflecting its abilities in a large, complex environment while giving the ability to the user to customize the profile of individual agents or group of agents, at run-time. A demonstration of the museum scene is shown in Fig. 11.

Figure 11: Museum Exhibition.

4.8 Discussions and Future Work

This section summarises the work that has been presented in Chapter 4, a framework that utilize a reinforcement learning method that enables the simulation of virtual crowds with diverse behaviors. The system is able to simulate heterogeneous crowds thought a single training process that learns multiple behaviors simultaneously. Likewise, the framework can simulate both simple crowd behaviors, like goal seeking and collision avoidance, while also enables the mixture of more complex behaviors like, grouping and interaction with the environment; additional behaviors can easily be incorporated to the policy.

This work is contributing to the area of virtual crowds by developing a framework that provides configurability to the profiles of learned agents, even during run-time. Moreover, this method provides a high degree of behavior diversity, while not only is able to simulate agents with multiple fixed behaviors, but also, enables the mixture of them through different combinations of the representative weights. As shown in the evaluation process, the system manages to reflect the various behaviors, for a large number of simulated agents, in a complex environment. Finally, the user can fully control the simulation by adjusting the profile distribution to achieve the desired results.

Despite the successful generalizability of the system in term of scenario variations, the lack of variation in such scenarios during training reduces its effectiveness. Furthermore, this work only incorporates a limited amount of behaviors, while the addition of more behaviors is straightforward. Lastly, the definition and balance of every weight's range is a challenging and manual process, however it has to be done once.

The proposed work opens various directions for future research study. First, the training process could produce more effective results by investigating more the reward function, observations and action space. First, more sophisticated reward function could be used that is not just a linear combination of weights. Second, would be every interesting and effective to search ways to automate the process of defining and balancing the ranges for the weights of the behaviors. Third, the simulation results could be improved by increasing the degrees of freedom that the agents have for action selection. Finally, the agents could explore the environment more effectively by using a more complex observation set, something that would improve the simulation results.

Chapter 5

Mesoscopic Approach

This approach mainly focuses on creating crowd simulations with a large number of agents by combining various components. Specifically, in this work two components have been trained, a field component, which navigates the agents efficiently, and a crossing component which simulates a real zebra crossing. These components can be placed and combined strategically to develop various populated environments.

5.1 Methodology

This section describes all the details about the proposed method and all the tools and techniques that have been utilized. The models for the two components have been trained using Proximal Policy Optimization (PPO) implementation provided by ML-Agents Toolkit [22] on a personal computer equipped with an Intel i5-9600K CPU, a Nvidia RTX2070 GPU and 16GB RAM.

5.1.1 Unity Navigation System

Unity provides an AI package which contains components that enable agents to intelligently navigating in a game. The package provides solution to two different problems regarding agents' navigation, first how to find a path to the destination position and second, how to move to that location. The first issue can be characterized as global and static, as the agent take into account a large part of the scene to calculate an appropriate path, while the second one is local and dynamic, as the agent has to worry about local collision avoidance. Unity's navigation system consists of three key components, which are shown in Fig. 12, including NavMesh Component, NavMesh Agent Component and NavMesh Obstacle Component, which the use of each one is described below; note that Off-mesh link is also one of the main components which allow agents to cross non-walkable paths, e.g by jumping, however they have not been used in this work.

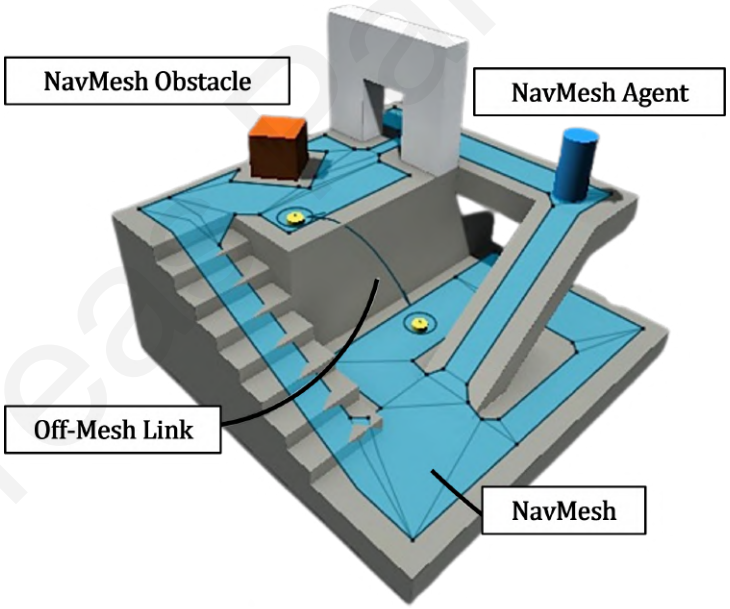

Figure 12: Visual overview of Unity Navigation System components.

First, NavMesh Component is utilized to define the walkable surfaces that agent can use to navigate through the environment. A NavMesh can be baked from multiple surface areas which

each one is able to have different area cost. The area cost of the surface defines how difficult it is to walk across a specific area; lower values means the area is more easy to walk. Specifically, the agents, in order to calculate the shortest path to the destination, they use the A* Algorithm [15]. In order to apply A* in an environment, a graph of connected nodes has to be built; this happens by placing a point on each polygon of the mesh. Thus, area costs mentioned above are used as link cost for the A* algorithm during the calculation of the path; the cost to move between two nodes is calculate as $distance * linkCost$. This tool gives the ability to the user to manually tune the navigation surface to achieve the desired results.

Furthermore, NavMesh Agent Component is attached to the agents that need to move over NavMesh surfaces. This component describes all the parameters of the agent, while is also used to define an object as moving agent to take advantage of RVO algorithm (Section 5.1.2) for collision avoidance. This component gives the ability to the user to define a list of parameters for the agent including, radius, height, speed, angular speed and acceleration.

Finally, NavMesh Obstacle component is used to define the an object as obstacle; this obstacle can be either static or moving. Agents try to avoid these objects during the simulation while also, they can be used as a way to completely block a specific path.

5.1.2 Reciprocal Velocity Obstacles (RVO)

The Reciprocal Velocity Obstacles (RVO) [60] algorithm is used for multi-agent real time navigation and is based on Velocity Obstacle (VO) [10] concept, which was originally introduced for the navigation of robots in the real world. RVO is based on the idea that each agent assumes that all the other agents have the same decision process on how to avoid collisions. When an agent predicts that is about to collide, moves only halfway out of the way of the collision direction, and assumes that the other agent will follow the same process by moving halfway to the opposite

direction; a visual representation of the process is shown in Fig. 13. The paper by van den Berg et al. [60] demonstrated that this approach can provide optimal collision avoidance by eliminating incorrect predictions, that were an issue of simple VO methods.

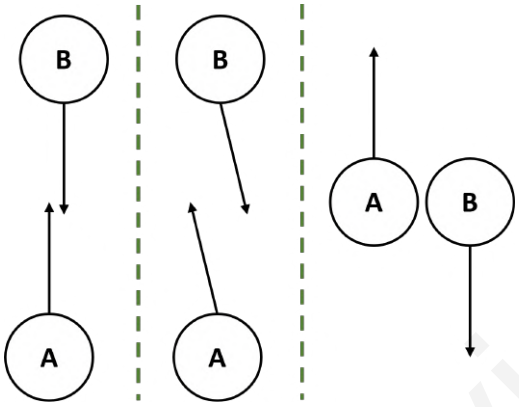

Figure 13: RVO upcoming collision avoidance between two agents.

5.1.3 Delaunay Triangulation and Interpolation via Barycentric Coordinates

In computer graphics the process of building a mesh from a list of points is very common; a mesh with uniform triangles is very important as affects both quality and performance. Delaunay triangulation [8, 7] is a method that enables the construction of a set of uniform triangles; this method tends to avoid narrow triangles by maximizing the minimum angle of all the angles of the constructed triangles.

In order to define a set of connected triangles as a Delaunay triangulation, in two-dimensional space, it has to meet a specific condition. As shown in Fig. 14 which demonstrate a Delaunay triangulation, circumcircles of all triangles must not contain any of the other points. The main mechanism of the algorithm is to determine whether one or more points lies inside any circumcircles, a situation that set the triangulation as invalid; this evaluation can happen in two-dimensional spaces using the determinant. Finally, they have been proposed multiple algorithms with various

methods including flipping of edges, incrementally adding edges and divide and conquer methods; for this work, a unity package named ”delaunator-sharp” has been used [12].

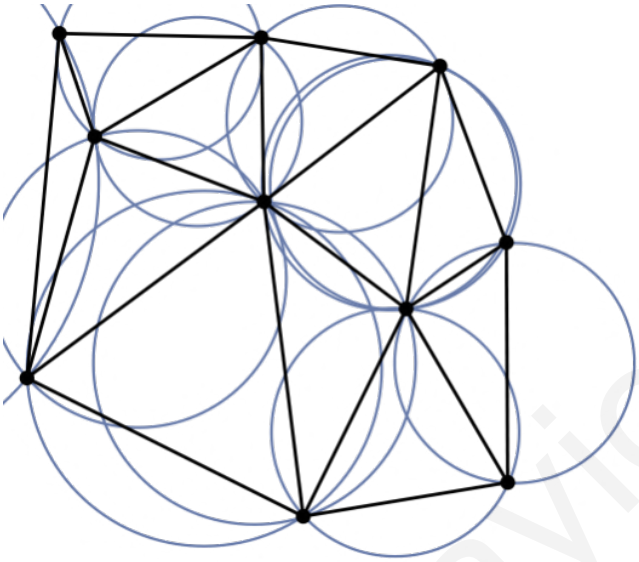

Figure 14: A Delaunay triangulation with circumcircles of triangles.

In this work the triangulation is used to divide a grid into triangles and then interpolate the values inside those triangles based on the main values on triangles' vertices. There are multiple ways to interpolate values inside a triangle, however they are not yield the same quality results. First, there is the naive solution which is based on distance; a point is affected more by the closer vertices and less by them which are farther away. The interpolation of the value of a point p inside a triangle $\triangle v_1 v_2 v_3$ using distance is given by the equation:

$$\begin{aligned} Value_p &= \frac{W_{v1} * Value_{v1} + W_{v2} * Value_{v2} + W_{v3} * Value_{v3}}{W_{v1} + W_{v2} + W_{v3}} \\ where,\ \ W_{v1} &= \frac{1}{Distance_{v1}},\ W_{v2} = \frac{1}{Distance_{v2}},\ W_{v3} = \frac{1}{Distance_{v3}} \end{aligned} \tag{3}$$

This method provides quite good results and is very easy to implement, however is problematic when the triangles are not balanced. As is shown in Fig. 15, which tries to interpolate colors inside a triangle, the results of a balanced triangle on the left side are appropriate, however on the right

triangle an issue arises. The point p that lies on the edge v_1v_3 should be a combination of v_1 and v_3 only, and should not be affected by the color of v_2. This may not seem a big issue in color interpolation, however can cause serious problem when is used to interpolate data.

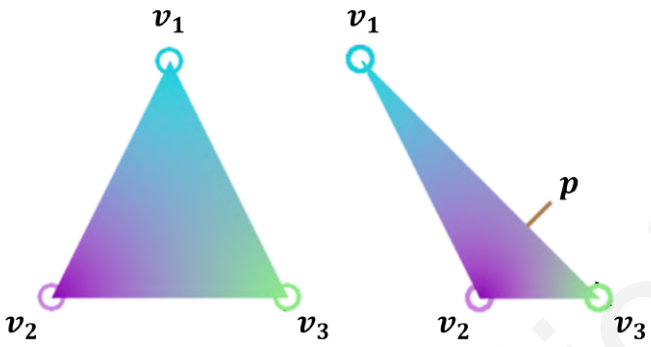

Figure 15: Naive triangle interpolation by distance and its issues.

In order to successfully interpolate values inside a triangle a method based on Barycentric Coordinates has to be utilised. Barycentric coordinates can be used to express the position of every point located on the triangle with three scalars w_1, w_2 and w_3. This method is the most suitable for interpolation of data inside a triangle and the interpolated values are not affected by triangle form. The interpolated value of a point p and the calculation of the three scalars w_1, w_2 and w_3 on a 2d plane are given by the following equations:

$$Value_p = W_{v1} * Value_{v1} + W_{v2} * Value_{v2} + W_{v3} * Value_{v3}$$

$$where,$$

$$W_{v1} = \frac{(Y_{v2} - Y_{v3})(P_x - X_{v3}) + (X_{v3} - X_{v2})(P_y - Y_{v3})}{(Y_{v2} - Y_{v3})(X_{v1} - X_{v3}) + (X_{v3} - X_{v2})(Y_{v1} - Y_{v3})} \tag{4}$$

$$W_{v2} = \frac{(Y_{v3} - Y_{v1})(P_x - X_{v3}) + (X_{v1} - X_{v3})(P_y - Y_{v3})}{(Y_{v2} - Y_{v3})(X_{v1} - X_{v3}) + (X_{v3} - X_{v2})(Y_{v1} - Y_{v3})}$$

$$W_{v3} = 1 - W_{v1} - W_{v2}$$

5.2 Rewards

As mentioned in previous sections, a model in RL is trained by maximizing a reward function. The following paragraphs describe all the rewards signals for both field and crossing components. Details about reward signals for both field and crossing components are shown in Table 7.

Field Component focuses on guiding the agents to their goal positions fast and efficiently. They have been used both sparse and dense rewards. First, regarding the sparse rewards, a field receives a positive reward R_g for every agent that arrives at its destination, while it be punished by a negative reward R_d for every agent that did not manage to find its destination during the defined training steps. Furthermore, a negative dense reward R_{dc} is used to force the field to avoid producing multiple values for same coordinates. Likewise, two rewards R_{dn} and R_{dp}, are used to encourage the field to avoid overcrowding as much as possible; inter-agent distance is calculated by all the agents that currently appear in the field and is a normalized value in the range $[0, 1]$. Finally, a living reward R_l is utilized to enable fast navigation of the agents; the living penalty is multiplied by the inverse of current agent's normalized speed, thus slower agents contributes to the field with larger negative reward.

Table 7: Reward per event for both components.

Event	**Symbol**	**Field Reward**	**Crossing Reward**	**Type**
Agent Goal Arrival	R_g	+1.0 / N	+1.0 / N	Sparse
Agent Destroyed	R_d	-.5 / N	-.5 / N	Sparse
Duplicate Coordinates	R_{dc}	-.025	-.025	Dense
Inter-agent Distance $<= .65$	R_{dn}	-.00025	-.00075	Dense
Inter-agent Distance $>= .65$	R_{dp}	+.00015	+.0001	Dense
Living Penalty	R_l	(-.001 / N)*(1-S_a))	(-.0008 / N)*(1-S_a)	Dense
Correct Door Open	R_{lc}	-	+.15 * L_t	Dense
Incorrect Door Open	R_{lnc}	-	-.15 * L_t	Dense
Agents crossing while Red	R_{ar}	-	-.002	Dense
Agents crossing while Green	R_{ag}	-	+.00025	Dense

Crossing Component is used to guide the agents to pass a road using a zebra crossing. The majority of rewards are similar to those in the field component while a list of dense rewards has been introduced to control the crossing. First the rewards R_{lc} and R_{lnc} are used to reward or punish the component every time that correctly or incorrectly opens the door to allow agents to enter the crossing; these rewards are multiplied by the normalized timestep of lights L_t, thus if the component incorrectly opens or closes the door near the alternation time of light state, penalty is smaller. Finally, a set of dense rewards R_{ar} and R_{ag} are used to punish the component when agents crossing the road while should not, and positively reward it in an opposite situation where the crossing is empty.

5.3 Observations

In this section the observation sets for both field and crossing components are described. Observations are mainly obtained by visual observations and specifically using grid sensors; the observation set for every component is described below. The Table 8 contains all the details about observations for both components; grid sensors have been used for both components while crossing utilize another two vector observations.

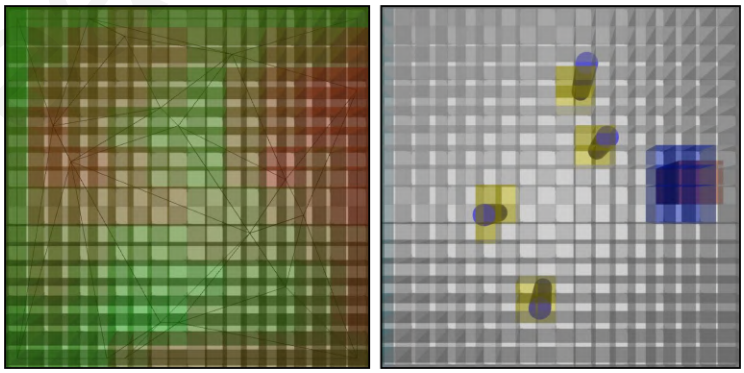

Figure 16: A visual representation of Grid Sensors.

Field Component collects observations using two grid sensors. The first grid sensor has a size of 20*20 and collects information about agents, obstacles and walls; for this method the observation stack has been set to 2, something that helps the component to estimate the velocity of moving objects including, agents and dynamic obstacles. The second grid sensor is used to enable the component to indirectly read the interpolated values of surface grid. This sensor gets the average value of every 2x2 patch in the surface grid which is in the range [-1, 1]. The two sensors are presented in Fig. 16, first sensor on the right and second on the left.

Crossing Component utilize the two grid sensors as used in the field component while another two vector observations have been added. First, L_s is a boolean value that indicates whether the light for the crossing is green(open) or red(closed). Furthermore, L_t is normalized float value in the range [0, 1] which indicates the time remaining for the lights to change state; for example if lights duration is 45 seconds and 30 seconds remain until state change, then $L_t = 30/45 \approx .67$.

Table 8: Observations for the field and crossing components

Observation	**Type**	**Field**	**Crossing**
Grid Sensor Grid Size: $20 * 20$, Observation Stack: 2 Detecting: Agents, Obstacles, Walls	Visual Observation	Y	Y
Grid Sensor Grid Size: $20 * 20$, Observation Stack: 1 Detecting: average value of each 2x2 patch	Visual Observation	Y	Y
Light Timestep (L_t)	Vector Observation	N	Y
Light State (L_s)	Vector Observation	N	Y

5.4 Actions

This section presents the actions which have been used for both components. First, both components use 52 continuous values, in range [-1.0.1.0], as shown in Table 9. The first 4 values are

assigned as main values to the points of the four corners of the surface grid; these points remains fixed and just take a value so the triangulation process can fill the entire surface. Then, the remaining 48 values are used to construct 16 sets of the form $\{X_i, Y_i, Value_i\}$, where X_i and Y_i are the coordinates of point i, while $Value_i$ is the value at point i.

Table 9: Actions for field and crossing components

Action	**Description**
52 Continuous Values [-1.0,1.0]	First 4 used as values for the four corners. The 48 rest values remaining are used as a set $\{X_i, Y_i, Value_i\}$
1 Discrete Value [0,1]	A boolean value representing whether the door is open or closed; used only in crossing component.

Moreover, for the calculation of the coordinates, the continuous values that have been produced by the network are normalized in the range [0.0, 1.0] and then multiplied by grid size; for instance, if we have a 20x20 grid and the values 0.2 and 0.5 for X and Y respectively, the coordinates of the point i would be ($X_i = 4$, $Y_i = 10$). A visual representation of a grid surface with both main and interpolated values is shown in Fig. 17; the color of a cell represents its value, positive and negative values are represented by green and red respectively, while the intensity shows how positive or negative the value is.

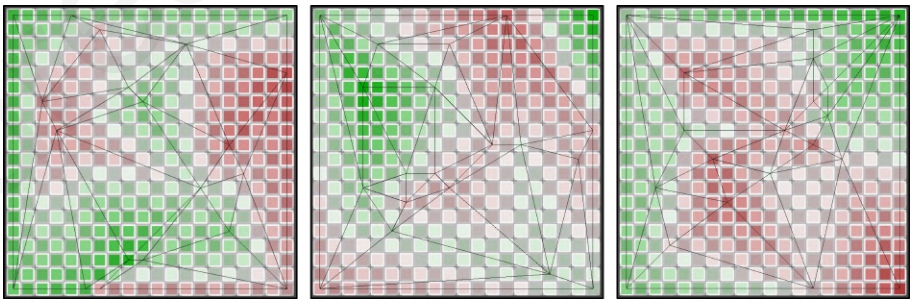

Figure 17: A visual representation of main and interpolated values on grid surface.

5.5 Training Strategy

This section explains the training strategy of this work by discussing how observations, actions, and rewards have been employed, as well as demonstrating the process of building environments conducive to successful training for both components. An implementation of the PPO provided by ML-Agents Toolkit has been used in combination with a Fully Connected Neural Network consisting of two hidden layers each having 128 nodes. The actual configuration parameters for the training of both components are shown in Table 10.

Table 10: Default values for configuration parameters used in training for both components

Parameter	**Value**	**Description**
Learning Rate	6e-4	For Gradient Descent Updates
γ	.99	Discount factor
H	5000	Maximum steps per episode
Epochs	3	Training Epochs
Batch Size	512	Batch Size
Buffer Size	10240	Buffer Size
β	5e-3	Entropy Regularization Strength
ϵ	.2	Divergence Threshold

First, regarding the training of the **Field Component**, the environment shown on the top of Fig. 18 has been designed. In every episode, several NavMesh Agents (black capsules) spawn in the blue area and set as destination a randomly generated position inside the red area; orange objects are obstacles that change position on every episode and alter the size of the openings. Furthermore, the white/gray area is a single field component with 20x20 surface size. An episode ends when all the agents arrive at their destination or a number of predefined steps have been passed; multiple instances of the environment have been used to decrease training time. During training, the rewards, observations and actions, that have been described in the previous sections, are used. Specifically, the field collects observations and every 100 fixed time-steps (2 seconds)

produces 20 main values, for 20 distinct points in the surface grid, using the process presented in Section 5.4. Then, these points are used in the triangulation process in order to cover the surface area with a mesh of triangles. The value of each cell on the surface grid is calculated using interpolation between the three main values on the vertices of the triangle; details about triangulation and interpolation methods are described in Section 5.1.3. Finally, the NavMesh surface of a field component consists of patches of 4 cells each and the various costs that have been used are shown in Table 11; the cost of a NavMesh patch is calculated by the average value of all four cells in the current patch.

Table 11: Costs used for every NavMesh area.

Area	**NavMesh Cost**	**Average Patch Value Range**
A_0	8	$[-1, -.8]$
A_1	6	$(-.8, -.6]$
A_2	5	$(-.6, -.4]$
A_3	4	$(-.4, -.2]$
A_4	3.5	$(-.2, 0]$
A_5	3	$(0, .2]$
A_6	2.5	$(.2, .4]$
A_7	2	$(.4, .6]$
A_8	1.5	$(.6, .8]$
A_9	1	$(.8, 1]$

Crossing Component has been trained using a similar approach as the field component. However, in this case there is the need of controlling the navigation of the agents based on the state of the crossing lights. Thus, first the NavMesh surface of the component is constructed using the same process as above. During the training of the crossing component a setup like the one shown in the bottom of Fig. 18 has been used. A number of agents spawn equally in the two blue areas and have as destination a random point on the other side. The box below indicates whether the crossing is open (green light) or closed (red light), and the orange object represents a door that

opens and closes to allow or prevent agents to enter the crossing. The training environment consists of two connected crossing components, while multiple instances have been used to decrease training time. Each instance has its own controller that handles the lights; specifically in the training phase the state of the lights change every 60 seconds. Finally, the observations described in Section 5.3 are used and an action is taken every 100 fixed time steps (2 seconds).

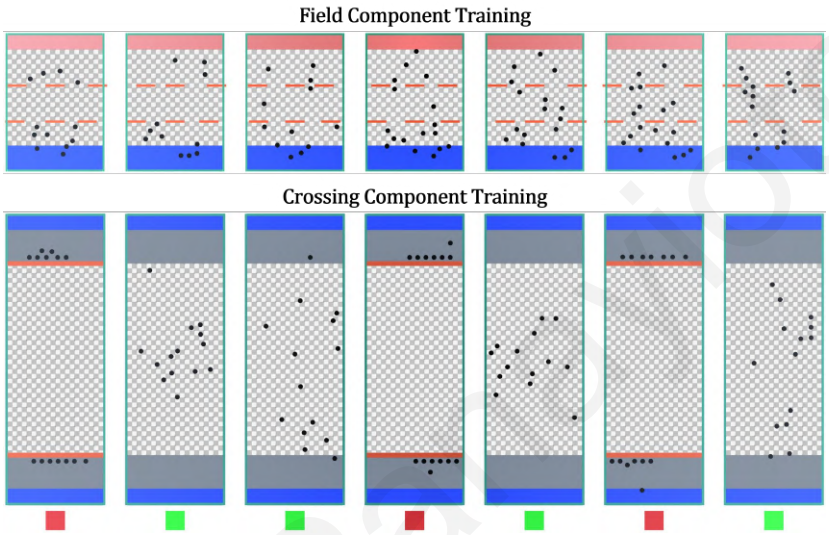

Figure 18: A visual representation of the training process for both field and crossing components.

5.6 Experiments and Evaluation

This part focuses on evaluating the trained model of a field component using a variety of approaches outlined in the subsections that follow. Multiple statistics, including agents' speed, distance to the nearest neighbor (DNN), and time to arrive at goal, have been used for evaluation and establishing results. The evaluation focuses on two aspects, the first is about density sensitivity and comparison with pure RVO implementation and the second is related to different update times where new values are assigned to the surface grid. Finally, for evaluating the model, four different setups have been developed, a) a default setup, b) a setup containing columns, c) a maze setup and d) an empty setup, which they are presented and described in the following subsections.

5.6.1 Density Sensitivity and Comparison with pure RVO Implementation

This experiment evaluates field component's model, over multiple setups, with different densities, while also compares it with the pure RVO implementation. The model has been trained with 10-20 agents, thus model's generalizability is evaluated too, with 25-125 agents moving over a single field component. Agents spawn in blue area and set as destination a point in red area.

First, the default environment that has been used during training is used. As shown in Fig. 19 the speed of the agents is decreased as the number of agents increased, while the average agents' speed of the model is higher than that of pure RVO. Moreover, the model's goal arrival time is significantly smaller compared to that of pure RVO, while also the average distance to NN of the model seems to be larger in the more dense scenarios.

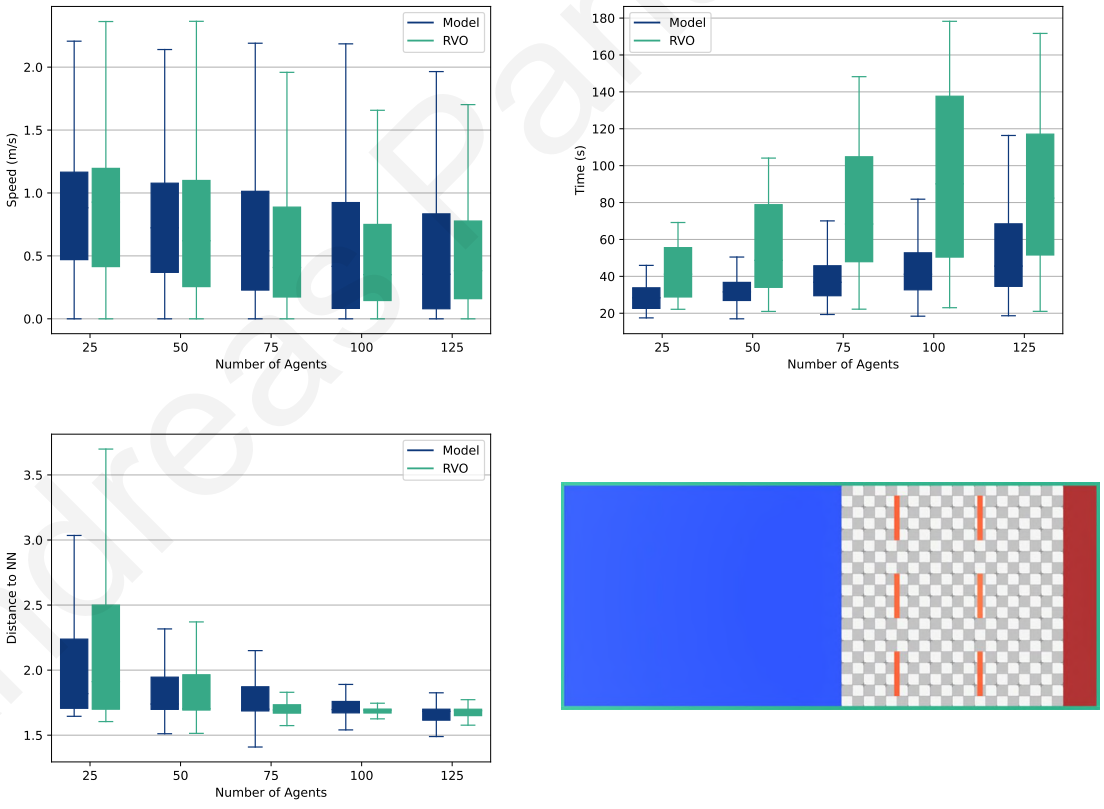

Figure 19: Comparison of the model with plain RVO in default training environment.

The next setup differs from that used in the training phase and contains several columns as obstacles; the environment setup and the charts of the three statistics are shown in Fig. 20. Similarly, we notice that the average agents' speed for the trained model is higher. Likewise, the average time is smaller which indicates that the model is able to navigate the agents more efficiently; the above observation is also supported by the larger DNN which agents have, something that shows that the model tries to avoid overcrowding when is possible. The results using this environment setup shows that the model is able to perform well event with setups not seen during the training. Finally, the charts for another environment setup, see Fig. 21, which represents a maze, show similar trends to those of columns setup; despite model's efficiency is still remarkable, is not as good as before due to the increased complexity of the environment.

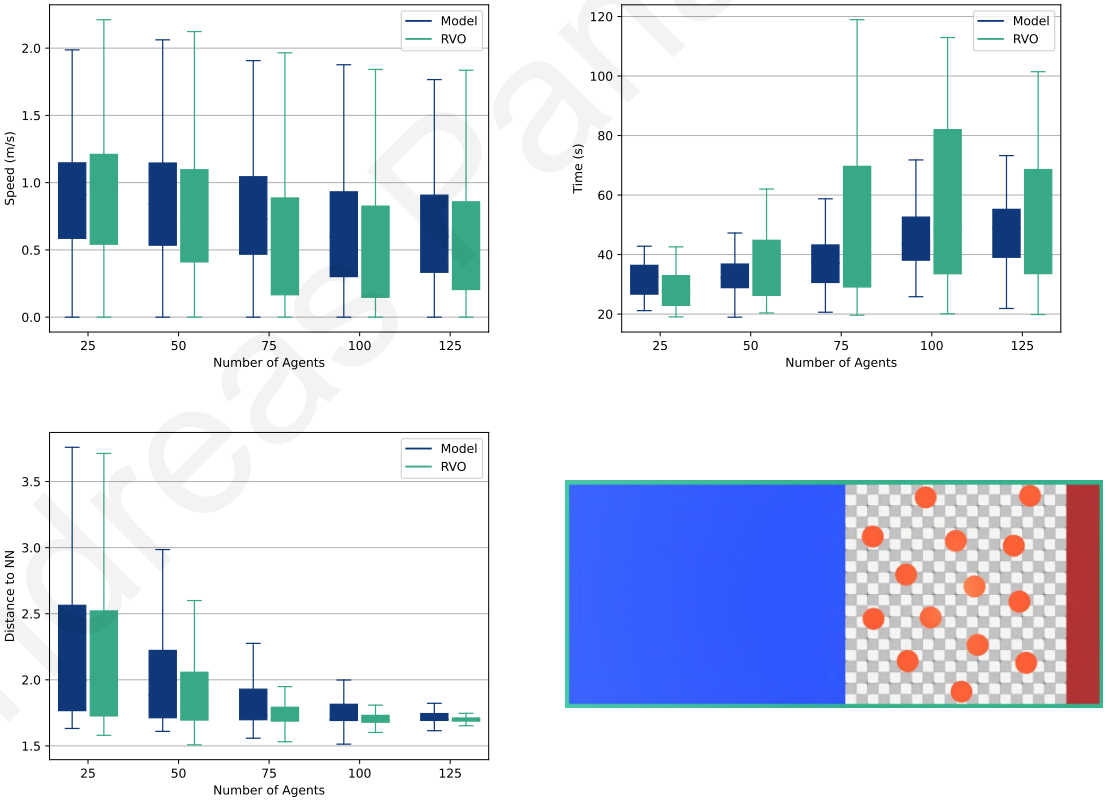

Figure 20: Comparison of the model with plain RVO in columns environment.

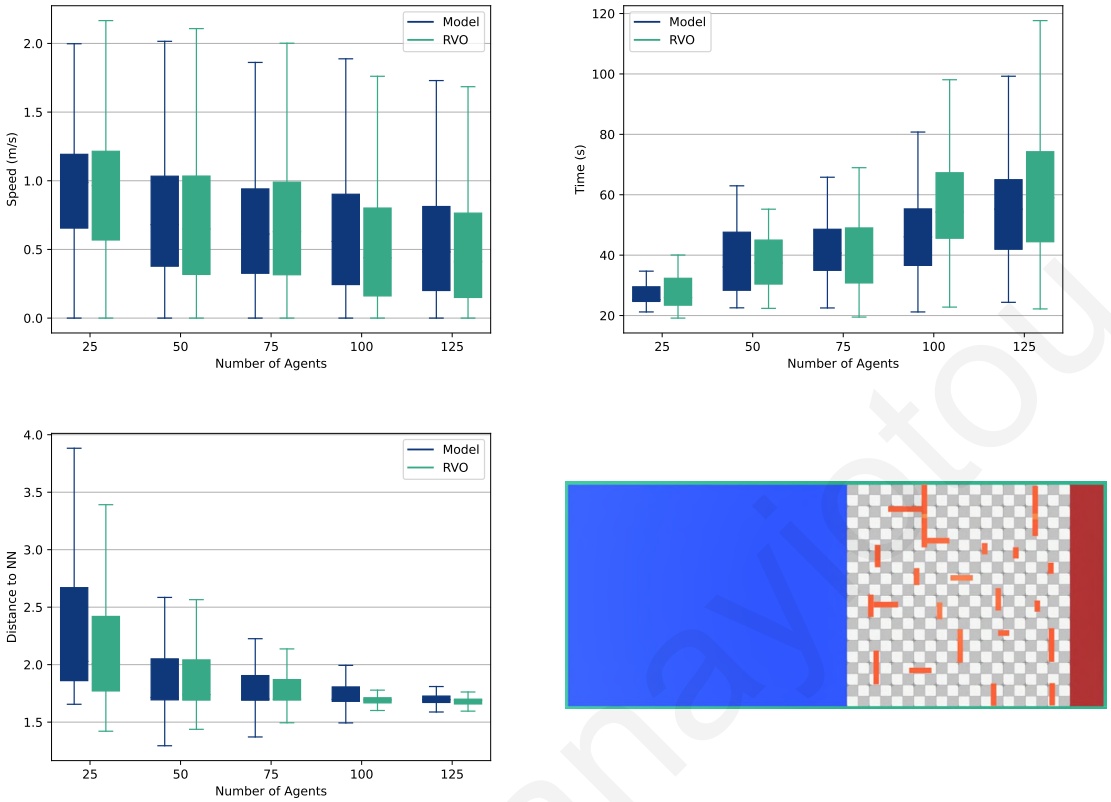

Figure 21: Comparison of the model with plain RVO in maze environment.

Furthermore, except from the previous environment setups that contain obstacles with different placements, another setup, which is an empty field, has been also used to evaluate the model. The setup is presented at Fig. 22 combined with the charts for the three statistics. The main purpose of the field component is to navigate agents as efficiently as possible through a number of obstacles. However, this environment setup has been developed to test the behavior of the model when no obstacle are presented in the field. As shown in the results of Fig. 22, the average goal arrival time of the agents is noticeably smaller compared to pure RVO implementation, while also the average distance to NN is slightly larger in denser scenarios. Finally, the speed of the agents is similar in both situations as there are no obstacle to significantly decrease the speed of the agents.

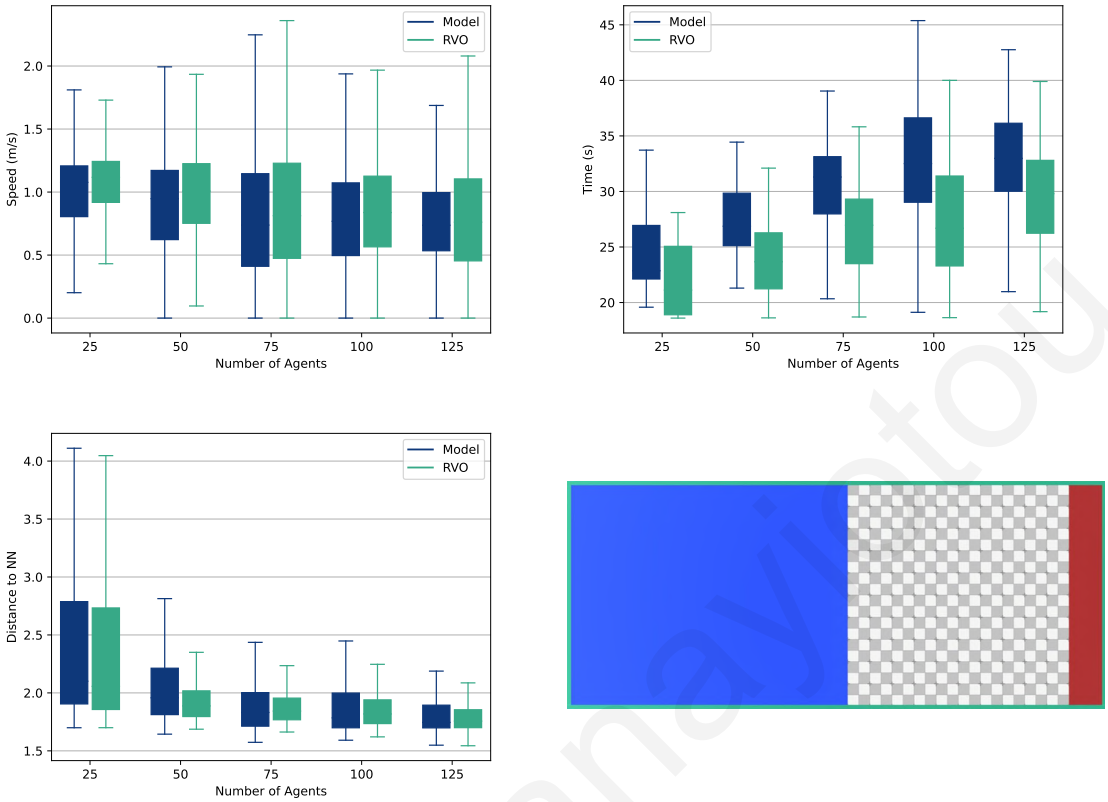

Figure 22: Comparison of the model with plain RVO in empty environment.

5.6.2 Timestep Sensitivity

This section focuses on evaluating the model with different update times. As mentioned in Section 5.4, during the training process, an update timestep of 100 steps (2 seconds) has been used. Thus, in order to evaluate systems generalizability in terms of update times, three different timesteps have been applied, 100 steps (2 seconds), 250 steps (5 seconds) and 500 steps (10 seconds). For the purpose of this experiment, the default training environment setup has been used. As shown in the results, the model produces fairly similar results by using the three different timesteps. This indicates that a larger update timestep could be used to reduce the performance cost of the model, as every time the model produces new values, triangulation and interpolation have to take place, while also the agents have to re-select a new path.

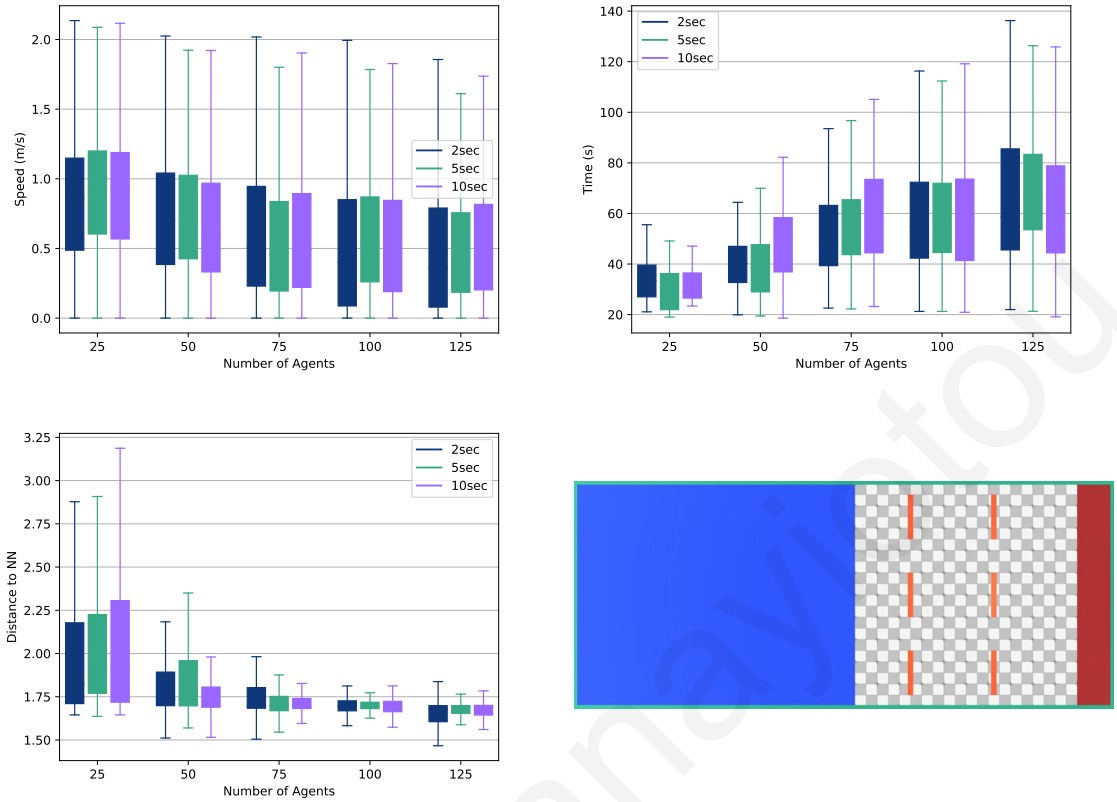

Figure 23: Comparison of different update times.

5.7 Discussions and Future Work

This section provides a summary of the work given in Chapter 5, a framework that aims to give the ability of creating medium-to-large scale crowd simulation by combining various components. In this work they have been presented two components, trained using Reinforcement Learning, a field component that navigates the agents efficiently, and a crossing component that simulates a real-world zebra crossing with road lights. The RVO implementation provided by Unity's AI package has been utilize in order to give the agents a basic ability of collision avoidance. Furthermore, a list of evaluation experiments have been employed to test the efficiency and generalizability of the model, both in terms of density and update time, in different environment setups.

The evaluation process shows that, regarding the navigation of the agents, the system is able to perform better compared to pure RVO implementation, even in setups that have not been seen during training. However, there are some limitations. First, there is an added performance cost, as every time the field produces new values, triangulation and interpolation have to be applied in order to assign a new value to every cell in the grid surface, while also agents have to repath as the the costs of the NavMesh areas change. A second limitation is that the field only receives observations for objects inside its area, something that may affect the generalizability when multiple components are combined. A way of sharing information between neighbor components would be beneficial when developing very large scale environments. The efficiency of the model is strongly based on the costs of every area patch; thus, the exploration of how to balance these costs and how many different area types should be used could be an interesting topic to investigate. Finally, the current work proposed only two components, while the development and training of more components is definitely a future plan.

Chapter 6

Conclusion

This chapter concludes this thesis. First, Chapter 2 presented a list of related works, regarding crowd simulation and crowd authoring, divided in four categories, heterogeneity in crowds, global navigation, crowd authoring and reinforcement learning. Second, Chapter 3 showed all the necessary tools that have been used for the implementation of this work. Furthermore, Chapter 4 has described every detail about the microscopic approach including methodology, reward signals, observation set, action set and training strategy. Then, the results of an extensive evaluation have been presented followed by a demonstration of the model in a demo museum scene. Finally, the chapter ends with a discussion about the contribution of the work to the field of crowd simulation, limitations and future plans. Moreover, Chapter 5 presented a mesoscopic approach and described all the tools and techniques used including Unity's Navigation System, Reciprocal Velocity Obstacles (RVO), Delaunay triangulation, and triangle interpolation using Barycentric coordinates. Likewise, has showed all the details about how reinforcement learning has been applied to train the model and how rewards, observations and actions has been utilized by the training strategy. Finally, a list of experiments are executed to evaluate the efficiency of the model followed by a discussion and future work section.

During the months that I was working on this thesis, I had the opportunity to get involved in an area of computer science that I am interested in and gain significant knowledge about various things, something that will definitely be helpful for my future career. First, the exploration of various techniques for crowd simulation was a chance to learn about existing works. Second, the hands-on usage of deep reinforcement learning made me more familiar with machine learning and how can be applied to solve various problems. Moreover, I had the chance to further explore methods that are used in computer graphics including triangulation and interpolation. Finally, the submission and acceptance of part of this thesis to an academic conference will definitely be a starting force for my future academic career.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[3] Panayiotis Charalambous and Yiorgos Chrysanthou. The PAG Crowd: A Graph Based Approach for Efficient Data-Driven Crowd Simulation. *Computer Graphics Forum*, 33(8):95–108, 2014.

[4] Panayiotis Charalambous, Ioannis Karamouzas, Stephen J. Guy, and Yiorgos Chrysanthou. A Data-Driven Framework for Visual Crowd Analysis. *Computer Graphics Forum*, 33(7):41–50, 2014.

[5] Stephen Chenney. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, page 233–242, Goslar, DEU, 2004. Eurographics Association.

[6] Nicolas Courty and Thomas Corpetti. Crowd Motion Capture. *Computer Animation and Virtual Worlds (selected best papers from CASA 2007)*, 18(4–5):361–370, 2007.

[7] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Delaunay Triangulations*, pages 183–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[8] B. Delaunay. Sur la sphère vide. *Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et na*, 1934(6):793–800, 1934.

[9] Funda Durupinar, Nuria Pelechano, Jan Allbeck, Uğur Güdükbay, and Norman I. Badler. How the Ocean Personality Model Affects the Perception of Crowds. *IEEE Computer Graphics and Applications*, 31(3):22–31, 2011.

[10] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[11] Julio E. Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1577–1585, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[12] Patryk Grech. Delaunator c#, 2020.

[13] Stephen J. Guy, Sujeong Kim, Ming C. Lin, and Dinesh Manocha. Simulating Heterogeneous Crowd Behaviors Using Personality Trait Theory. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 43–52, New York, NY, USA, 2011. ACM.

[14] Stephen J. Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C. Lin, and Dinesh Manocha. A Statistical Similarity Measure for Aggregate Crowd Dynamics. *ACM Trans. Graph.*, 31(6), November 2012.

[15] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[16] Feixiang He, Yuanhang Xiang, Xi Zhao, and He Wang. Informative Scene Decomposition for Crowd Analysis, Comparison and Simulation Guidance. *ACM Trans. Graph.*, 39(4), July 2020.

[17] Dirk Helbing, Anders Johansson, and Habib Zein Al-Abideen. Dynamics of crowd disasters: An empirical study. *Physical review E*, 75(4):046109, 2007.

[18] Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. Learning to navigate through crowded environments. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 981–986, Anchorage, AK, USA, 2010. IEEE.

[19] Kaidong Hu, Michael Brandon Haworth, Glen Berseth, Vladimir Pavlovic, Petros Faloutsos, and Mubbasir Kapadia. Heterogeneous Crowd Simulation using Parametric Reinforcement Learning. *IEEE Transactions on Visualization and Computer Graphics*, PP:1–1, 2021.

[20] Min Hu, Saad Ali, and Mubarak Shah. Learning motion patterns in crowded scenes using motion flow field. In *2008 19th International Conference on Pattern Recognition*, pages 1–5, 2008.

[21] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable Crowds. *ACM Trans. Graph.*, 29(6), December 2010.

[22] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, abs/1809.02627, 2018.

[23] Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinman, and Petros Faloutsos. Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 53–62, New York, NY, USA, 2011. ACM.

[24] Ioannis Karamouzas, Brian Skinner, and Stephen J. Guy. Universal power law governing pedestrian interactions. *Physical review letters*, 113(23):238701, 2014.

[25] Ioannis Karamouzas, Nick Sohre, Ran Hu, and Stephen J. Guy. Crowd space: a predictive crowd analysis technique. *ACM Transactions on Graphics (TOG)*, 37(6):1–14, 2018.

[26] Ioannis Karamouzas, Nick Sohre, Rahul Narain, and Stephen J. Guy. Implicit Crowds: Optimization Integrator for Robust Crowd Simulation. *ACM Trans. Graph.*, 36(4), July 2017.

[27] Sujeong Kim, Stephen J. Guy, Dinesh Manocha, and Ming C. Lin. Interactive Simulation of Dynamic Crowd Behaviors Using General Adaptation Syndrome Theory. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 55–62, New York, NY, USA, 2012. Association for Computing Machinery.

[28] Nick Kraayenbrink, Jassin Kessing, Tim Tutenel, Gerwin Haan, Fernando Marson, Soraia Musse, and Rafael Bidarra. Semantic Crowds: Reusable Population for Virtual Worlds. *Procedia Computer Science*, 15:122–139, December 2012.

[29] T. Kwon, K. H. Lee, J. Lee, and S. Takahashi. Group motion editing. In *ACM Transactions on Graphics (TOG)*, page 80, New York, NY, United States, 2008. ACM.

[30] Yu-Chi Lai, Stephen Chenney, and ShaoHua Fan. Group motion graphs. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 281–290, Los Angeles, California, 2005. ACM.

[31] Pierre Le Pelletier de Woillemont, Rémi Labory, and Vincent Corruble. Configurable Agent with Reward as Input: A Play-Style Continuum Generation. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, Copenhagen, Denmark, 2021. IEEE.

[32] Jaedong Lee, Jungdam Won, and Jehee Lee. Crowd simulation by deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, pages 1–7, Limassol, Cyprus, 2018. ACM.

[33] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. A Data-driven Approach to Crowd Simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 109–118, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[34] Seyoung Lee, Sunmin Lee, Yongwoo Lee, and Jehee Lee. Learning a Family of Motor Skills from a Single Motion Clip. *ACM Trans. Graph.*, 40(4), July 2021.

[35] Marilena Lemonari, Rafael Blanco, Panayiotis Charalambous, Nuria Pelechano, Marios Avraamides, Julien Pettré, and Yiorgos Chrysanthou. Authoring Virtual Crowds: A Survey. *Computer Graphics Forum*, 2022.

[36] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by Example. *Computer Graphics Forum*, 26(3):655–664, 2007.

[37] Alon Lerner, Yiorgos Chrysanthou, Ariel Shamir, and Daniel Cohen-Or. Context-Dependent Crowd Evaluation. *Computer Graphics Forum*, 29(7):2197–2206, 2010.

[38] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning, 2017.

[39] Jonathan Maïm, Barbara Yersin, and Daniel Thalmann. Unique Character Instances for Crowds. *IEEE Computer Graphics and Applications*, 29(6):82–90, 2009.

[40] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Multi-Agent Reinforcement Learning for Simulating Pedestrian Navigation. In *International Workshop on Adaptive and Learning Agents*, page 53, Berlin, Heidelberg, 2011. Springer, Berlin, Heidelberg.

[41] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Emergent behaviors and scalability for multi-agent reinforcement learning-based pedestrian models. *Simulation Modelling Practice and Theory*, 74:117–133, 2017.

[42] Ronald Metoyer and Jessica Hodgins. Reactive pedestrian path following from examples. *The Visual Computer*, 20, 04 2003.

[43] Ronald A. Metoyer and Jessica K. Hodgins. Reactive Pedestrian Path Following from Examples. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, page 149, Washington, DC, USA, 2003. IEEE Computer Society.

[44] Soraia R. Musse, Cláudio R. Jung, Julio C. S. Jacques Jr, and Adriana Braun. Using computer vision to simulate the motion of virtual agents. *Computer Animation and Virtual Worlds*, 18(2):83–93, 2007.

[45] Masaki Oshita and Yusuke Ogiwara. Sketch-based interface for crowd animation. In *Smart Graphics*, 2009.

[46] Andreas Panayiotou, Theodoros Kyriakou, Marilena Lemonari, Yiorgos Chrysanthou, and Panayiotis Charalambous. CCP: Configurable Crowd Profiles. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques, Siggraph 2022*, ACM, New York, NY, USA, 10 pages, 2022.

[47] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[48] Julien Pettre, Helena Grillon, and Daniel Thalmann. Crowds of moving objects: Navigation planning and simulation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3062–3067, 2007.

[49] Craig W. Reynolds. Steering behaviors for autonomous characters, 1999.

[50] Wei Shao and Demetri Terzopoulos. Autonomous Pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 19–28, New York, NY, USA, 2005. Association for Computing Machinery.

[51] Libo Sun, Jinfeng Zhai, and Wenhu Qin. Crowd Navigation in an Unknown and Dynamic Environment Based on Deep Reinforcement Learning. *IEEE Access*, 7:109544–109554, 2019.

[52] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[53] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[54] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[55] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[56] Shigeo Takahashi, Kenichi Yoshida, Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Sung Yong Shin. Spectral-based group formation control. *Computer Graphics Forum*, 28(2):639–648, 2009.

[57] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, jul 2006.

[58] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal Character Animation with Continuous Control. *ACM Trans. Graph.*, 26(3), July 2007.

[59] Branislav Ulicny, Pablo de Heras Ciechomski, and Daniel Thalmann. Crowdbrush: Interactive Authoring of Real-Time Crowd Scenes. In *Symposium on Computer Animation*, SCA '04, pages 243–252, Goslar, DEU, 2004. Eurographics Association.

[60] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[61] He Wang, Jan Ondrej, and Carol O'Sullivan. Trending Paths: A New Semantic-Level Metric for Comparing Simulated and Real Crowd Data. *IEEE Transactions on Visualization and Computer Graphics*, 23(5):1454–1464, May 2017.

[62] D. Wolinski, S. J. Guy, A.-H. Olivier, M. Lin, D. Manocha, and J. Pettré. Parameter estimation and comparative evaluation of crowd simulations. *Computer Graphics Forum*, 33(2):303–312, 2014.

[63] Jungdam Won and Jehee Lee. Learning Body Shape Variation in Physics-Based Characters. *ACM Trans. Graph.*, 38(6), November 2019.

[64] Barbara Yersin, Jonathan Maïm, Pablo De Heras Ciechomski, Sébastien Schertenleib, and Daniel Thalmann. Steering a virtual crowd based on a semantically augmented navigation graph. In *In Proceedings of the First International Workshop on Crowd Simulation*, pages 169–178, 2005.

[65] J. Zhang, X. Jin, S. Huang, J. Xu, and C. L. Wang. Interactive control of large-crowd navigation in virtual environments using vector fields. *IEEE Computer Graphics and Applications*, 28(06):37–46, nov 2008.

[66] M. Zhao, W. Cai, and S. J. Turner. CLUST: Simulating Realistic Crowd Behaviour by Mining Pattern from Crowd Videos. *Computer Graphics Forum*, 37:184–201, 2017.