5-1-2023

Automated Discovery of Candidate Simulation Models for Steering Behavior Simulation

Hai Le

Automated Discovery of Candidate Simulation Models for Steering Behavior Simulation

by

Hai Le

Under the Direction of Xiaolin Hu, PhD

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2023

ABSTRACT

Steering behavior of autonomous agents plays important roles in many simulation applications, such as simulation of pedestrian crowds, simulation of evacuation scenarios, simulation of ecosystems, simulation of autonomous robots, and simulation of artificial life in virtual environments used in computer games. It is desirable to have an approach that can automatically discover multiple candidate models for steering behavior simulation besides manual approach (trial-and-error fashion) and data-driven approach. Towards this goal, this work presents an approach that searches for candidate models of steering behavior in an automated way. The proposed framework includes two components. A model space specification provides a formal specification for a general structure from which various models can be constructed, and a search method to search for a set of candidate models based on requirements. To support more complex scenarios, we further add three major extensions including: (1) Activation component assign dynamic priorities for behaviors depending on surround environments. (2) Multiple search stages are provided to assist the evolutionary search algorithm to distribute computational resources better. (3) A special type of entity called space entity to assist agents receive information not only from other entities (agents, obstacles), but also from surrounding empty space. The approach is able to discover multiple candidate models for three basic steering behaviors including the leader-following ($B_{leader\_following}$), personal space maintenance ($B_{personal\_space}$), and mobile obstacle avoidance ($B_{obstacle\_avoidance}$). The results show that different possibilities of steering behavior support modelers to have a better understanding of the problem under study, hence assist modelers to develop more advanced models by testing different combinations of the basic steering behaviors. We evaluate all combinations between three basic steering behaviors including: (1) $B_{leader\_following} + B_{obstacle\_avoidance}$, (2) $B_{obstacle\_avoidance} + B_{personal\_space}$, (3) $B_{leader\_following} + B_{personal\_space}$,

and (4) $B_{leader\_following} + B_{obstacle\_avoidance} + B_{personal\_space}$. We further test the approach with two variations of scenario 4: (5) The leader surrounding $+ B_{personal\_space}$, (6) Hall-way evacuation with an obstacle in the middle. The results show that the framework is also able to discover multiple models for each of these composite steering behaviors, and several of them have good scalability and robustness.

INDEX WORDS: Agent-based, Automatic discovery, Steering behaviors, Space entity, Genetic algorithm, Complex models

Automated Discovery of Candidate Simulation Models for Steering Behavior Simulation

by

Hai Le

Committee Chair: Xiaolin Hu

Committee: Anu Bourgeois

Raj Sunderraman

Jing Zhang

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

May 2023

DEDICATION

I would like to dedicate this work to my parents Hanh Le and Hanh Hoang who love me unconditionally and always want the best for their children. To my brother Long Le who encouraged me to study Computer Science and shaped my career. Finally, I would like to thank my wife Diem Phung and my daughter Khue Le for supporting me unselfishly and sacrificing a lot for me to archive my dream

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Xiaolin Hu for assisting me conduct the research to this point and supporting me unconditionally. I also would like to thank committee members. Dr. Bourgeois, Dr. Sunderraman, and Dr. Zhang for spending time and effort to evaluate my work.

TABLES OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Steering Behaviors Definition

Steering behaviors are behaviors that make autonomous agents move around their world in a realistic and improvisational manner. Steering behavior of autonomous agents plays important roles in many simulation applications, such as simulation of pedestrian crowd, simulation of evacuation scenarios, simulation of ecosystems (e.g., predator-prey systems) and animal groups (e.g., schools of fish), simulation of autonomous robots (e.g., UAVs), and simulation of artificial life in virtual environments used in computer games. A common theme of these applications is to reflect movements in real life more truthfully, hence making the simulations more accurate and convincing. To achieve this goal, it is essential to model the steering behaviors that make autonomous agents move around their world in a realistic and improvisational manner. For example, to accurately study people's evacuation from a busy shopping mall, it is important to model individuals' steering behavior in a crowded environment. Similarly, to create an engaging virtual world for game players it is important to generate realistic steering behaviors for the game characters. Steering behavior simulation can also support education and learning – educators can use these simulations to excite students and to teach them about complex systems. For example, simulations of ecosystems and animal groups, such as flocks of birds, can show the emergence of complex patterns from simple rules. [1]

1.2. Current Approaches to Develop Steering Behaviors Simulations.

Developing steering behavior models in a multi-agent environment is a complex task because the developed behaviors need to be natural, consistent, and able to fulfill required tasks.

1.2.1. Manually Crafted Approach.

Traditionally, these models are crafted manually in a trial-and-error fashion. Modelers use their knowledge or consult experts to manually create an initial model that captures the behavior and structure of a system under study. Then, the initial model is gradually improved until an end result meets the simulation requirements [1]. This approach is useful when testing theories of how a system works. However, the handcrafted models often have biases from their creators.

1.2.2. *Data-Driven Approach*

In recent years, due to the rapid development of video processing techniques, researchers began to use real world data to help evaluate or develop models of steering behavior. For example, supervised learning has been used to train a machine learning model for predicting the next-step steering behavior of agents based on specific conditions surrounding the agents [2] [3] [4]. This type of machine learning-based approach relies on the existence of large and high-quality data in order to train a model. It does not work when data are lacking, incomplete, or inconsistent.

1.3. The Automated Discovery Model Approach – The Framework

The complexity of steering behavior simulation asks for more ways to developing steering behavior models. In particular, it is desirable to have an approach that can automatically discover multiple candidate models for steering behavior simulation. A key component of the developed this modeling approach is the model space for searching candidate models. To define an effective model search space, a formal model specification is important so that automated model discovery is possible. Our work provided a model specification for mobile agent-based systems where a world includes a 2D space and a set of entities. Each entity has several pre-defined properties such as position, (moving) direction, speed. One entity category called agents has one or more behaviors that manipulate properties' values. This agent category present objects that modelers want to study their movement such as: vehicles, animals, humans, etc. At each time step, agents execute their

behaviors in behavior groups to sense the surrounding environment and move accordingly in the 2D space. The search space is defined by all possible behaviors of agents based on their properties and how they interact with other entities by sensing, filtering, and acting. Once the requirements and evaluations are set, a search algorithm is used to search for the best models. More details about the framework as well as some discovered models can be found in [5].

Because our previous work focused on developing the framework to support automatically discovering models for emerging behaviors such as: snake formation or circle formation, it cannot be applied directly for steering behaviors simulation. Towards this goal, this work presents an approach that searches for candidate models of steering behavior in an automated way. The multiple discovered candidate models can assist modelers see the different possibilities of steering behavior for a problem under study. They can be evaluated and further extended for developing more advanced models. This compares to the existing approaches that typically result in only one model for steering behavior simulation. The model space specification must be flexible enough to represent a wide variety of possible behavior behaviors. Furthermore, it must capture steering behavior in a way that is comprehensible and human readable. This latter requirement is important so that the discovered candidate models can potentially be modified and extended by modelers. To fulfill these new requirements for the framework, three major extensions are added including: Activation component, a Genetic Algorithm-based multiple search stage procedure, and Space entity component.

1.4. Activation Component

For complex scenarios, agents usually have multiple behaviors where their heading direction decision are not unanimous. The approach of averaging moving actions from multiple behaviors is not uncommon in the literature. For example, in the Boids model [1], an agent's

steering direction in each step is averaged from the separation, alignment, and cohesion behaviors. However, there are many other situations where prioritizing behaviors would work better when computing an agent's final movement. Consider the behavior of obstacle avoidance, which moves the agent away from the obstacle, as an example. When the agent is relatively far away from the obstacle, it is less important to incorporate the moving action of this behavior into the overall movement. As the agent moves closer to the obstacle, this behavior becomes more important: the closer the agent is from the obstacle, the more important the behavior is. In other words, the moving action of this behavior becomes more dominant as the agent gets closer to the obstacle. To support this capability, The Activation component is used to prioritize the behaviors based on how "important" the behaviors are. This allows the priority of a behavior to be modeled because different activation levels represent different priorities. The Action component specifies the action of the behavior, i.e., how the behavior changes the value of a property. This component is similar to what we had in previous work. Both the Activation component and Action component dynamically compute their outputs based on agents' conditions and surrounding environment. Differentiating these two components makes it possible to define behaviors that are dynamically activated based on conditions of the environment and then act accordingly to respond to the environment. Although this extension is not related directly to discover steering behaviors, it plays a very important role to assist multiple steering behavior working together; thus, supports discovery of more complex steering simulation models and increases the applicability of the modeling approach to even more applications.

1.5. **Genetic Algorithm-based Multiple Search Stage Procedure**

Genetic Algorithm (GA) is used to find models that satisfy the desired behavior patterns specified by a modeler. The model specification's components are designed to easily interchange

and manipulate between models; hence it is reasonable to present them as chromosomes and use GA to encode a potential solution to a specific problem and apply recombination on these structures so as to preserve critical information [49]. We notice that GA search efficiency reduces significantly for complex scenarios because of two main reasons. First, complex scenarios with multiple requirements typically need multiple steering behaviors to fulfill all the tasks. With the automated discovery approach, the more behaviors needed to be discovered, the larger model space requires. Hence, GA takes much longer time to search for the desired models. Second, GA ranks candidate models based on their fitness scores, and in many cases, conflictions between evaluation criteria make it difficult to rank these models. The leader-following + mobile obstacle avoidance is one example. The leader-following behavior makes agents form a cluster around a leader while mobile obstacle avoidance behavior makes agents scatter. In case of two candidate models with the same score, one has higher score for the leader following fitness and the other one has higher score for mobile obstacle avoidance fitness; GA does not know which one should be rank higher, hence it randomly switches the rank between these candidates and most of the time is trapped in its local optimal. To resolve these limitations, we divide the search process into two different stages, and each stage focuses on searching for specific level components of the framework to assist GA use the computational resource better.

1.6. Space Entity Component

Our previous work has mainly focused on agents' interactions with other physical entities (including other agents); hence an agent can only act on properties (e.g., position, direction, speed) of some physical entities. This limitation of considering only the physical entities, hence makes it difficult for agents to move in directions that are not directly associated with any physical entity. In particular, it is difficult for agents to navigate through open spaces among physical entities,

which is crucial for steering behavior simulation. It has been observed that when steering in complex environments, autonomous agents use information not only from nearby physical entities, but also from open space directions surrounding them. Even though an open space appears to have nothing, it actually contains useful information such as how far agents can move along the open space direction, and how much the angle difference is between the open space direction and the agents' current moving directions. Having the capability of processing these types of information would give agents more options for choosing their movement in steering behavior simulation. This would allow agents to move more effectively and naturally in experiment scenarios where utilizing the open space around agents are crucial such as multiple obstacles avoidance, or leader-following.

Motivated by the above discussion, this extension develops a new behavior specification for steering behavior that gives agents the option to interact with the open space surrounding them. Hence, a new category of entity called "space entity" is introduced to add to the physical entities that have been supported in the previous framework. Each agent has a set of space entities corresponding to a set of landmarks that an agent can head to from its current position. At each timestep, an agent's space entities are dynamically generated based on an adaptive space entity generation method, which checks the agent's nearby physical entities and generates a set of possible steering directions adaptively. To work with the existing automated model discovery framework, each space entity has a set of pre-defined properties. For example, the travel distances from the agent to the space entities, or the angle differences between the directions to space entities and the agent' current direction. Agents use the values of these properties to filter which space entity they want to change direction toward. This extension gives agents more flexibility when using steering behaviors. For example, in previous work, when an agent wants to avoid a physical entity, it first needs to get a reference direction from the physical entity. Then a static angle offset

is added so that the agent can move away from the entity. This approach is cumbersome and not adaptable (for example, the static angle offset makes the agent always turns left or right with a fixed angle away from the entity). The space entities provide multiple reference directions for agents to evaluate, hence make the steering behaviors flexible and adaptive to many situations.

1.7. Organization

The remainder of this work is organized as follows. Section 2 describes the related work. Section 3 presents the automated model discovery framework, including the model space specification, activation component and the search method. Section 4 describes the space entity component. Section 5 presents experiment results of applying the developed approach to discover candidate models for a set of steering behavior simulation scenarios. Section 6 shows the discovered models' evaluations and analyses from different aspects are carried out. Section 7 concludes this work and Section 7 shows more potential research that can be carried out from this work.

2. RELATED WORK

2.1. Steering Behavior Simulation Applications

2.1.1. Simulation of Pedestrian Crowds

Steering behaviors are basic components in many crowd movement experiments. In the simulation of the pedestrian crowd, human's characteristics such as psychology or society are usually contributed to the final steering decisions. The main goal of these simulations is to study the crowd movements in different facility layouts under normal circumstances [6] [7] [8] [9]. For example, Figure 1.a shows movement pattern of individual passengers (microscopic movements) in a section of an airport [7]. Figure 1.b shows a similar application with shopping center layout. However, crowd movements here are capture as a group (macroscopic movements) instead of individuals [9].
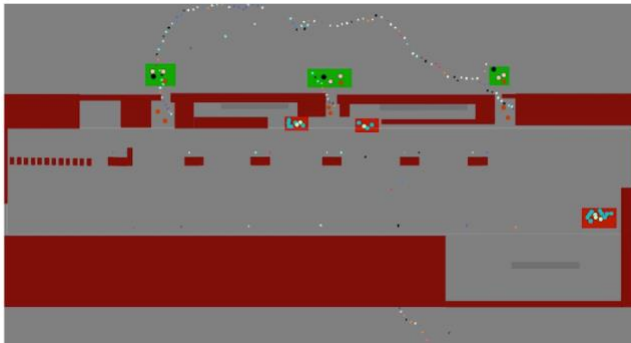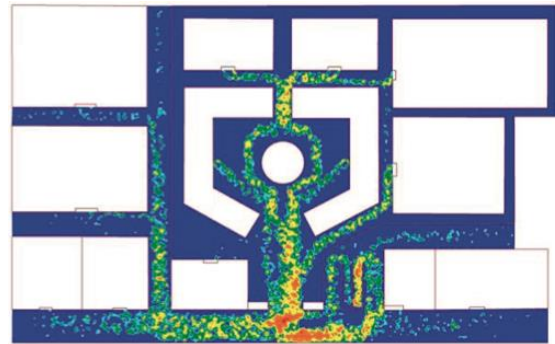
(a) An airport layout   (b) A shopping center layout

Figure 1. Application examples for pedestrian crowd simulations.

2.1.2. Simulation of Evacuation

In simulation of evacuation, the main task is to achieve the max throughput in the shortest times, hence the set-up scenarios usually in closed space, and have a great number of agents. Normal moralities can be broken in panic situations; thus, behavior rules work differently

compared to regular crowds. Body mass, risk sensitivity, physical limitations are usually extra factors that contribute to the steering decision in fire [10] [11], flood [12] [13], or other evacuation scenarios [14]. The applications for this type of simulation are typically related directly to real life situations. For example, Figure 2.a shows an evacuation example of a classroom [11]. These small-scale simulations focus on factors that can affect the evacuation speed such as: furniture layout, locations and the size of exit doors, or number of people, etc. Figure 2.b shows a flood evacuation simulation where a layout of a stadium is used as a real study case [13]. Large scale simulations similar to this example are used to design evacuation plans for big buildings rather than concentrate on the small details.
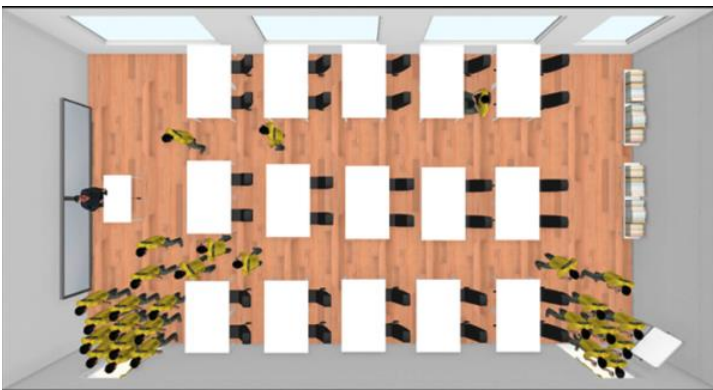
| (a) A classroom evacuation. | (b) A disaster (flooding) evacuation. |
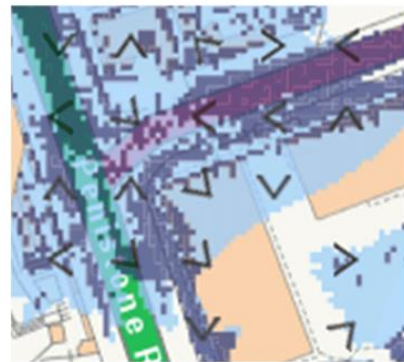
Figure 2. Application examples for simulation of evacuation.

2.1.3. *Simulation of Ecosystem*

Besides humans, many works focus on animal crowds. For example, simulations of ecosystems research are mainly inspired by nature and concentrate on simulating the interaction between animals. The study of steering behaviors of animal groups has been intensively discussed with Reynold's work [1] as a prime example. In his work, flock of bird movements are combined between three steering behaviors: separation, cohesion, and alignment. For more complex systems where two or more different animal pieces are involved such as predator-prey [15] [16] or

shepherding [17] [18], the related works focus on designing steering behaviors to manage high-level path planning and low-level single agent dynamics. For example, green area (Figure 3.a) [15] and gray travel path (Figure 3.b) [18] present a group of animals (small birds and sheep respectively). Because they move in group, the behaviors only need to capture movement dynamic. In other hand, the path planning for the dog shown in Figure 3.b is complex, and the developed behavior is able to capture the side-to-side steering movements.

(a) Predator-prey example   (b) Shepherding example
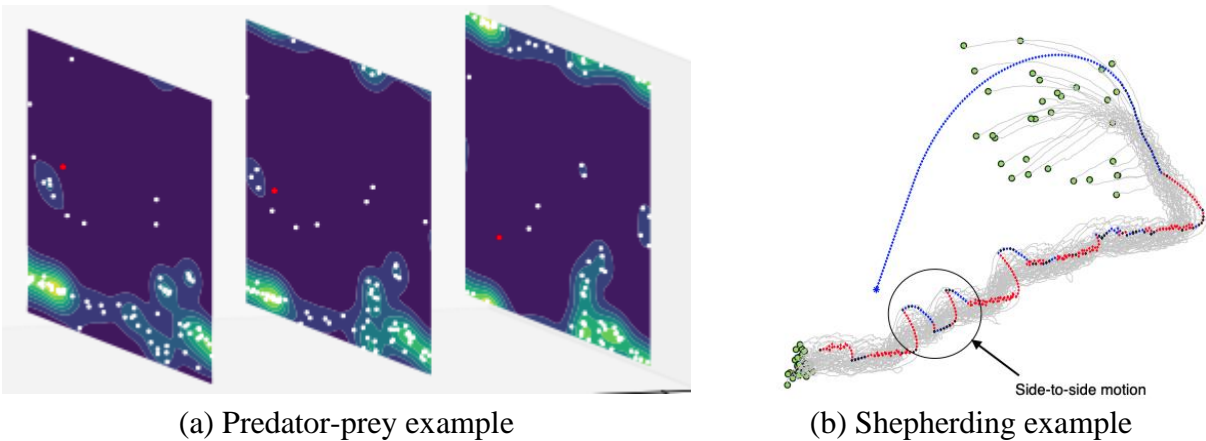
Figure 3. Application examples for simulation of ecosystem.

2.1.4. *Simulation of Autonomous Robotics*

In simulation of autonomous robotics, to reduce operation costs and risks of failure, simulations are commonly the first step to evaluate the outcome before deploying on real hardware devices. Research in this field develops simulation models mainly for two scenarios. In the first scenario, modelers know the travel path of each individual in advance and pre-set all parameters to satisfy the need. The related works in this scenario focus on optimizing the efficiency of steering behaviors and are mostly used in drone performance (Figure 4.a) [19] [20]. The second scenario uses a decentralized approach where each robot is fully automated and can make their own decisions. These robots rely on a set of equipped sensors to sense the surrounding environments

and decide the next steering direction. Recent works focus on movements of Unmanned Aerial Vehicles (UAVs) in many applications such as: fire observation [21], bio-inspired swarm [22] [23], or farming [24] [25]. For example, Figure 4.b illustrate a fire observation simulation [21] where a group of UAVs automatically determine their travel paths (distinguished by colors) based on the activeness of fire areas.
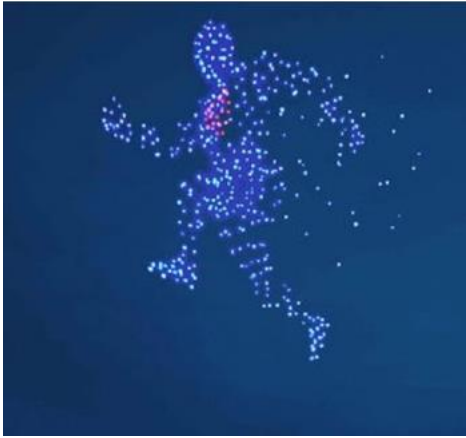
(a) A drone performance   (b) UAV path-planning for fire observation
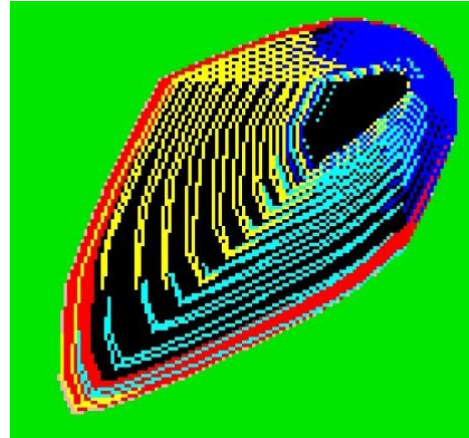
Figure 4. Application examples of simulation of robotics.

2.1.5. Simulation of Artificial Life

Besides nature, artificial life concepts also play an important role in the construction of simulation models of non-player characters in computer games. In many applications, to add more realistic to the steering models, evolutionary computations including genetic algorithm, evolutionary strategies, genetic programing, evolutionary programing are used [26]. For example, in [27], [3] the authors develop a real-valued encoding for context steering to have a comprehensive evaluation for each agent's goals. In gaming, this goal can be anything such as reaching a destination or avoiding dangers ahead. Based on the assessment, a set of fitness functions are used to find the best steering directions.

2.2.   Manually Crafted Approach.

Regardless of the applications and the requirements, developing natural steering behaviors for crowd simulation and modeling is a challenging task because they are influenced by many factors such as physics, psychology, and society [28]. Traditionally, these models are crafted fully based on knowledge from modelers. Behaviors-based [1] [29] [30] [31], force-based [32] [33], velocity-based [34] [35], and vision-based [36] [37] techniques are among the most common techniques to manually develop autonomous crowd simulation models. [38]

2.2.1.   Behavior-based Model

The most famous behavior-based model is Boids model [1] that simulates flocks' behaviors. Each member of the group interacts with a nearby neighbor by following a set of rules (Separation, Alignment, and Cohesion (Figure 5.a, b, c respectively).
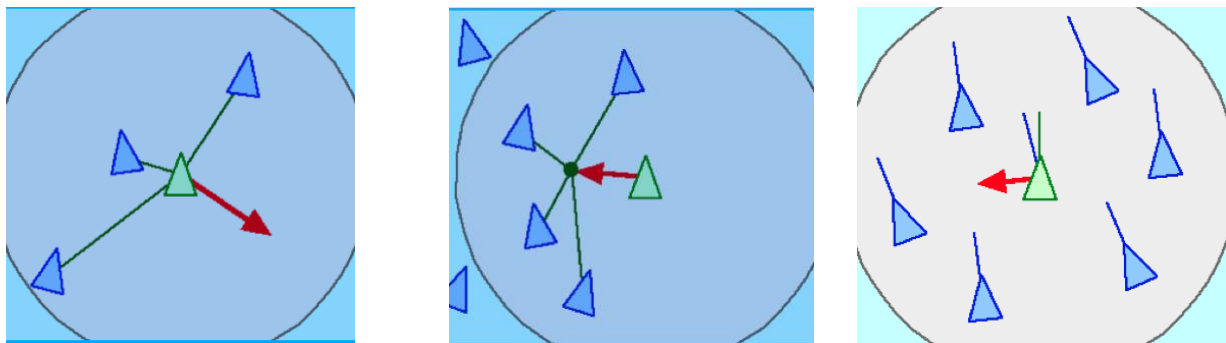
(a) separation   (b) cohesion   (c) alignment

Figure 5. Flock of bird movements are presented as three steering behaviors.

In recent years, researchers have developed many advanced behavior-based models. For example, the work in [39] used a behavior called Centroidal particle dynamics to explicitly ask pedestrians in dense crowds to step in the direction that would best to maintain and attempt to regain personal space. Another example is [22] where modelers proposed a flocking model for real

UAVs incorporating an evolutionary optimization framework with carefully chosen order parameters and fitness functions.

2.2.2. Forced-based Model.

In 2000, the famous Social Force Model (SFM) was presented by [32] to study the behaviors of crowd panics. Each agent is affected by attractive and repulsive forces of every object in the environment including other agents, physical objects, and other factors. Figure 6.a shows main timesteps of an evacuation scenario where a group of agents need to escape a closed room through a small door using SFM. This concept later on is extended and became Universal Power Law that governs pedestrian interactions in [33]. The key point of the law is pedestrians attempt to minimize the energy they spend on interactions, knowing that a predicted collision has a lower probability of actually occurring if it is farther away. As shown in Figure 6.b, agents at initiated state moves in random direction. By applying Universal Power Law, eventually, all of them have the same moving flow.

(a) Social Force Model   (b) Universal Power Law Model
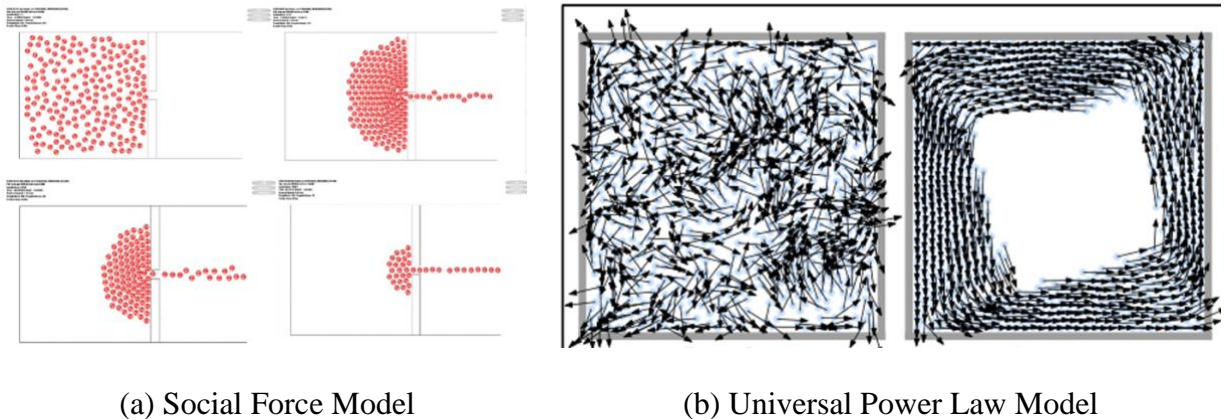
Figure 6. Examples of Forced-based model.

2.2.3. Velocity-based Model

Velocity-based models consider not only neighbors' positions but also their velocities to make decisions. The main advantage of this type of model-based is it can predict the collisions

between objects in the next multiple timestep and is mainly used in obstacle avoidance scenarios. The work in [40] creates a model that combines the advantages of force-based and velocity-based models to create better evacuation models. Its social force equation is expanded to include the collision prediction force; hence their model has better quality in terms of evacuation time and prediction times, and the steering behaviors are more natural compared to the of [32].

2.2.4. Vision-based Model

The vision-based models are variations of the velocity-based models. Instead of focusing on the neighbors' raw information, vision-based models process and convert this information to pixels and gradient, and base on the color combinations to make the decisions. For example, in [36], a target person processes surrounding neighbor information, then converts it to three different color pixel maps. These maps are further processed to analyze the risk of collision based on the combined color (red is high risk, and blue is low risk) (Figure 7.a), and the person chooses the path with the lowest risk. This work in [37] goes one step further by combining a vision-based model with a velocity-based model to steer agents through dynamic environments. The optical flow of human eyes is used as a vision in this work. As a result, agents have the depth perception of far and near objects (Figure 7.b) and use it to extract visual features such as extract visual features such as the focus of expansion and time-to-collision.
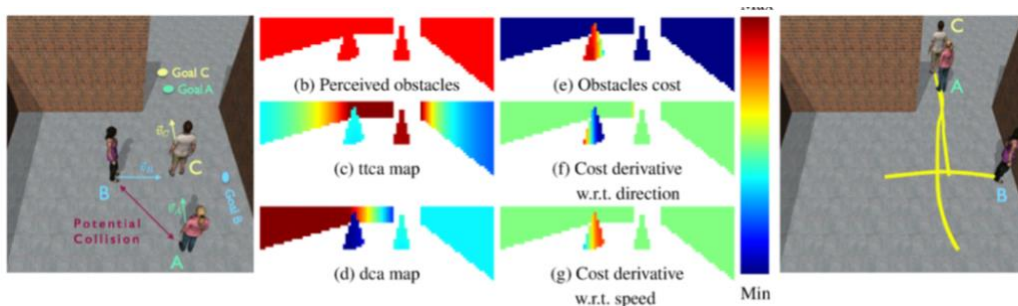
(a) Risk of collision based on the gradient.
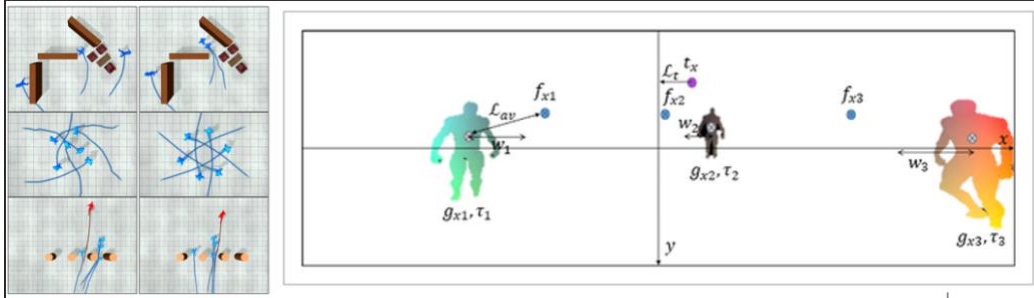
(b) Depth perception that similar to humans.

Figure 7. Examples of vision-based model.

2.3. Data-Driven Approach

In recent years, with the rapid development of video processing techniques, the data-driven approach is also used to simulate crowd steering behaviors to take advantage of the collected data. Calibration-based, evolutionary algorithm-based, and machine learning-based techniques are commonly used.

2.3.1. Calibration-based Techniques

In the calibration-based method, modelers develop a calibration function to adjust the parameters until it matches the ground truth data. As a result, datasets in many proposals [41] [42] [43] mainly are used to validate the tested model and play a small role in model construction. The calibration processes are still depended heavily on the expert's domain. For example, the work's formulation is based on incrementally learning pedestrian behaviors from crowd video (Figure 8) and calibrating the parameter to extract correct pedestrian trajectory [44].

Figure 8. An example of extracting travel path of pedestrians from crowd video clips.

2.3.2. *Evolutionary Algorithm-based Techniques*

In evolutionary algorithm-based techniques, the model construction steps are automated by using an evolutionary algorithm (GA for example) to capture generic behavior rules from video data. In [45], a symbolic regression problem is used to formulate crowd behaviors and a self-learning gene expression programming is utilized to solve the problem and automatically obtain behaviors that match data. The work in [46] goes one step further by using genetic algorithms to discover individual agent behavioral rules from videos. Pedestrian movements in these clips have prescribed objectives, and by designing the rules that can reproduce, GA can discover the purposes of crowd behaviors solely from the collected data.

2.3.3. *Machine Learning-based Techniques*

In machine learning-technique, a form of artificial intelligence (AI) that makes predictions from data. One or more supervised, unsupervised, or reinforcement learning AI models are chosen. Supervise methods rely on prediction labels from training data to build appropriate models [4]. In the work [2], each agent has a 360-degree Field of View (FOV) and is split to 4 equal direction segments (North, South, East, West) (Figure 9.a). 24 policies are generated to cover all possible combinations between the statuses of each segment, and Decision Tree AI model is used to predict

the next steering direction. Unsupervised method does not build a prediction model and uses unlabeled training data directly to predict the outcome [47] [48]. Reinforce learning method trains AI models using reward and penalty mechanisms. In [15], the authors show that by using a multi-agent reinforcement learning model (SELFish) (Figure 9.b), emergent escape-based flocking behavior can be discovered. Even though the goal of training agents is to survive as long as possible, their crowd movements lead to flocking behavior similar to Boids. Regardless of the choice, AI models use the training data to learn and predict the next steering action of each agent based on the surrounding environments. The more correct predictions, the better the AI models are. These data driven techniques help to reduce the manual workload and bias from modelers significantly.

(a)   (b)
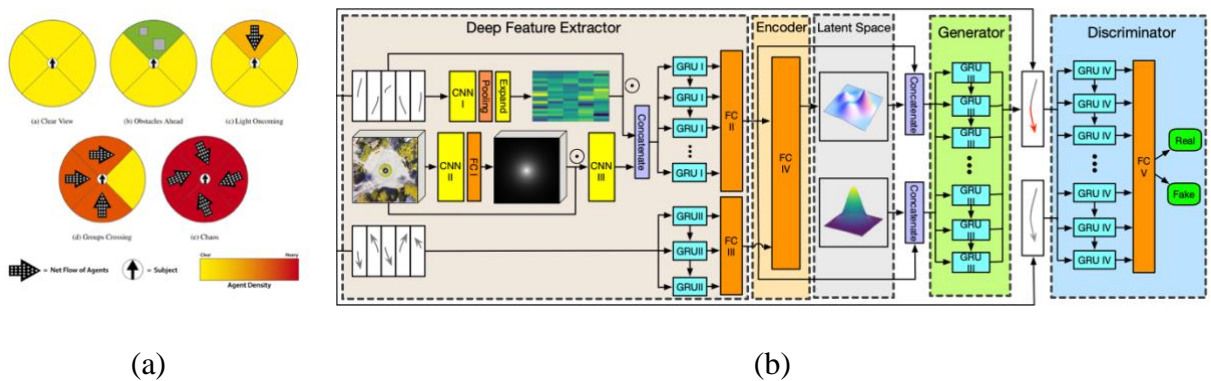
Figure 9. Two different machine learning techniques to predict the next steering direction.

(a) Multiple policies for Decision Tree model

(b) A reinforcement learning approach using a deep learning model named SELFish

3. METHODOLOGY

3.1. The Framework

3.1.1. The Overview

The framework includes two major components: a model space specification and a search algorithm. The model space specification provides a formal specification for the general model structure from which various models can be generated. It defines a "meta-model" for behaviors-based mobile agent systems [5]. GA is chosen as the search algorithm to discover the best fit models base on a set of pre-defined fitness metrics. Figure 10 shows an overview workflow of how these two components works in the framework.
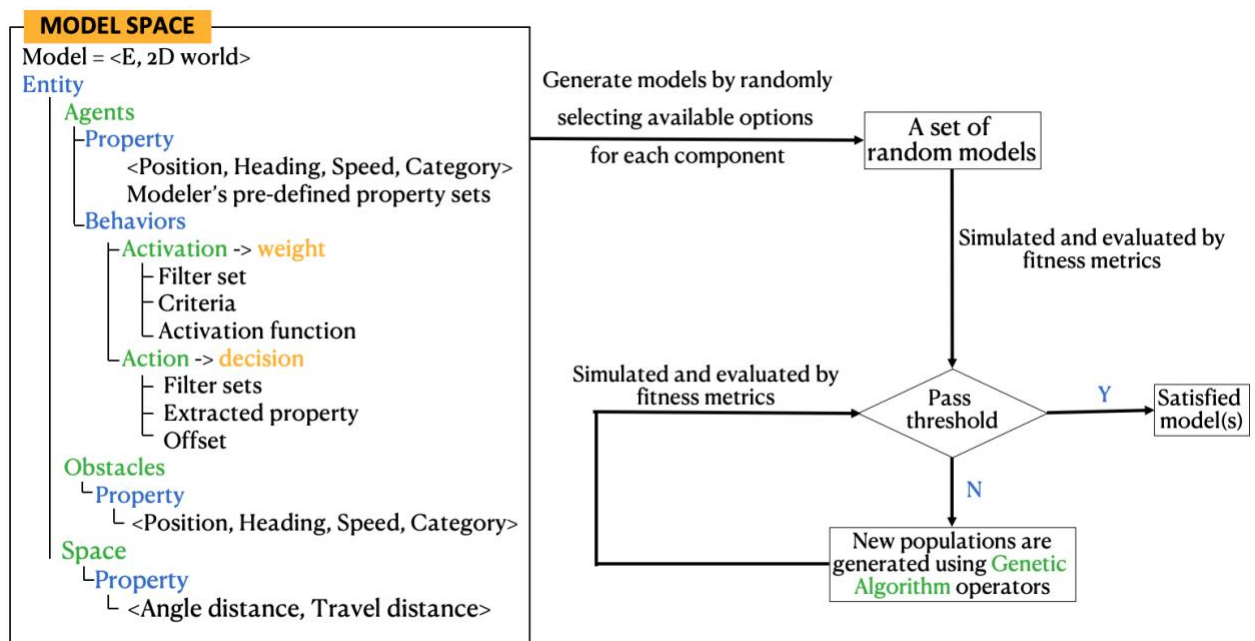
Figure 10. Overview each step of the automated model discovery approach.

The model space specification is designed to take advantage of mobile agent-based system where abilities of perception, decision-making and action are provided [38]. In addition, it can be combined with other type of models to create complex steering behavior. The model space is composed of a world and a set of entities (agents, obstacles, and space), and each entity category

has a set of pre-defined properties **P** such as: position, heading direction, or speed. Because we use agent-based model, each agent **a = < P, B>** contains a set of behavior **b**. A world is a 2-D space that wraps vertically and horizontally. Besides agent entity, others basic type of entities can be added easily to the framework depends on the scenarios such as: obstacles, or space entities. All agents share the same set of behaviors. Each behavior manipulates a pre-defined property such as: position, heading direction, or speed. Properties play critical roles because each behavior acts on a specific property by changing its value based on observations of the environment and nearby agents' properties. In complex scenarios where the discovered models need to fulfill several requirements, multiple steering behaviors usually are needed, and conflictions can happen between them such as force cancelation. Therefore, it is essential for the framework to have a mechanism to dynamically prioritize each behavior based on the surrounding environment. As a result, we separated the behavior specification into two main components: Activation and Action or **b = < p, Activation, Action>** in **B.** The **Activation** component = **< Criteria, Activation function>** is used to assign a weight for the behavior depends on agents' self or neighbors' criteria and activation function (binary or linear). It is useful when agents need different priorities for their behaviors under different circumstances. The **Action** component = **<F, offset>** has a set of filters **F** to filter entities and extract property's values from them. After the offset is added, the final value replaces current property value **p** of the agent. More detail of basic model space specification can be found in [5].

Based on the model space, Genetic Algorithm (GA) is used to find models that satisfy the desired behavior patterns specified by a modeler. GA encodes a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination on these structures so as to preserve critical information [49]. Based on the specification, a random model can have

one or more behaviors. Each behavior has several choices to choose for range filters (distance, angle, speed, etc.), method filters (nearest, furthest, average, etc.), and extract property functions (position, direction, speed). These options are considered as chromosomes for models, and all possible combinations between them create the search space. First, random models are generated within the model space, and all of them create an initial population. Next, simulations of each model are performed by using a set of agents that make decisions based on model's behavior specifications. A set of fitness functions is used to evaluate how close the outcomes of models are to the simulation goals. After the evaluation, if the fitness scores of one or more models pass the threshold, GA finds the best models and stops. If they do not pass, a new population is created with 25% best from old generation, 25% mutation from 25% best of old generation, and 25% crossover from old population. The last 25% is randomly generated to increase the diversity of the populations [5], and a new circle begins until GA finds the best model(s) or reaches the computation limitations.

3.1.2. *Agent Property Specification*

To support more complex scenarios, we would allow a modeler to add new properties into the model space for automated model discovery. The goal of property specification is to provide a well-defined structure for agent properties so that new user-defined properties can be added into the model space and be searched by the search method in a unified way. To achieve this goal, a formal and general structure for user-defined properties is needed. We define a user-defined property **p** has a general structure as below:

p = <type, range, v_{init}> where

Type: numerical, or categorical

Range: if type is *numerical*, range $=[v_{low}, v_{upper}]$, where v_{low} is the lower bound of the numerical value, and v_{upper} is the upper bound of the numerical value.

if type is *categorical*, range = the set of all possible categorical values.

v_{init}**: initial value** of the property:

if type is *numerical*: v_{init} is a real number between v_{low} and v_{upper}

if type is *categorical***:** v_{init} is an element of the **Range** set.

To add a new property to the model space, a modeler needs to specify the type, range, and v_{init} for that property so that it can be searched by the search method. Typically, the modeler has some knowledge about the system and the property, and thus can define the type and range of a new property to be searched. For example, if agents' energy is important for a specific application, the modeler can define a new property called energy, and specify its type to be numerical and define its range to be between 0 and 100. Similarly, if agents can change color between green, yellow, and red for a specific application. The modeler can define a property called color that has **type** = categorical and **range** = {green, yellow, red}. The modeler may also specify the initial value v_{init} for the property if starting from that initial value is important for the application. Otherwise, by default v_{init} is a random value within the Range.

We note that specifying the type, property, and v_{init} of a property is different from defining how the property is used by specific behaviors. The modeler specifies the property, but then it is up to the search method during the model discovery process to decide if and how the property will be used by a specific behavior. In general, when a modeler provides a property that is meaningful and has relevant range, it would make it easier for the search method to find behaviors using this property.

3.1.3. *A Demonstration Example of How a Behavior Manipulates an Agent's Properties.*

Based on the model space specification, in each iteration of the discrete time simulation, a behavior goes through multiple steps to compute a moving action of the behavior. Figure 11 illustrates these steps where each step is represented by a box. The arrows before and after the boxes represent the input and output of these steps. In the first step, agents sense a set of nearby neighbors within their field of view (FOV). The second step applies filters to get a specific set of agents. The next step extracts property from the set of agents and combines them into one reference value. Then, the last step is to add an offset to the reference value to compute a desired value for the property the behavior is acting on. This desired value is referred to as the desired action for this behavior. If there are multiple behaviors that belong to the same behavior group, the described actions are averaged to compute an overall action of the agent. Note that a special property is the position property, which is treated differently based on which step a position value is checked. Specifically, in the extract action property step, a position value is always converted to the direction towards that position; the direction then becomes the reference value for the next step (the add action offset step). In all other situations, a position value is treated as a value for calculating a distance to that position.
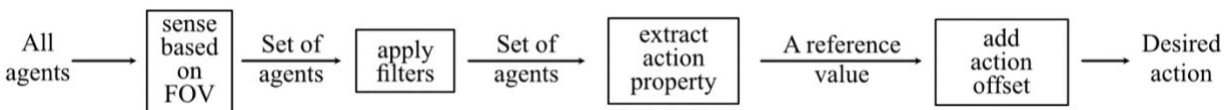
Figure 11. Steps of a behavior.

Figure 12 illustrates the behavior steps for a sample behavior that makes an agent steer left from its nearest neighbor. In this example, the 2D space includes ten regular agents (hollow white circle) and one obstacle (black solid circle), where agent A is the agent that owns this behavior.

Through step 1, there are five entities: four regular agents (1-4) and one obstacle within agent A's FOV. In step 2, agent A applies couple of filters to eliminate unsatisfied neighbors. The first filter chooses regular agent only (step 2.a). Hence, the obstacle is removed. The second filter chooses the nearest neighbor (step 2.b), and as a result agent 1 is selected. In step 3, agent 1's position property is extracted and then converted to the direction towards that position. This direction becomes the reference value for the moving action of this behavior. Then step 4 adds an offset value to the reference direction to compute the desired action of this behavior. In the last step, agent A executes the desired action and moves according to the new direction.
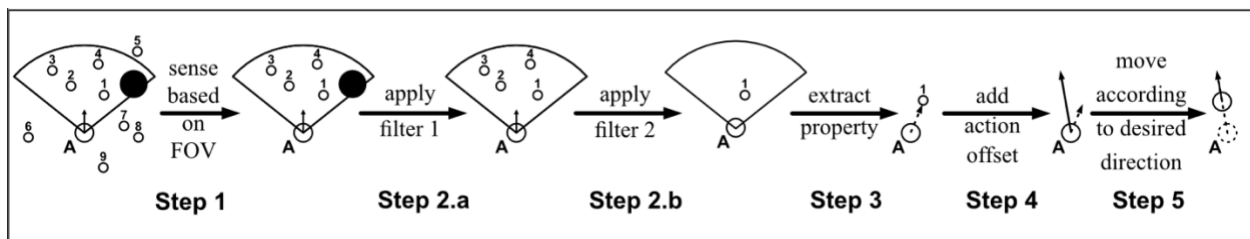
Figure 12. Illustration of the behavior steps.

3.2. Activation Component

3.2.1. *Motivations*

In previous work, the described actions from the multiple behaviors of a same behavior group are averaged to compute the final action. In many cases, this method does not work well because the desire action of each behavior would cancel each other after being averaged. For example, Figure 3.a shows a scenario where agents have an obstacle avoidance behavior but cannot successfully perform it. Figure 3.b explains why this happens. In this example, agent A has two behaviors that manipulate its direction: B1 and B2. Agent A uses behavior **B1** to avoid the obstacle by steering to the left, and **B2** to follow its nearest agent. Because the final decision $\overline{B}$ is averaged between **B1** and **B2**, agent A moves toward to the obstacle directly.

(a)   (b)
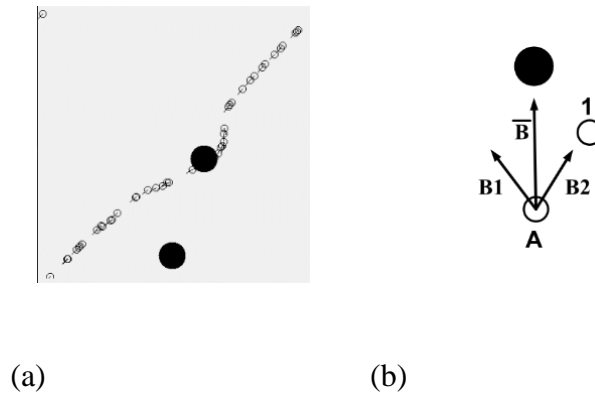
Figure 13. An example where average method does not well.

To address this problem, we need a mechanism to specify the importance of each behavior at each iteration. Therefore, we extend the previous work so that each behavior has two components: an Activation component and an Action component. The Activation component specifies the level of activation of the behavior. This allows the priority of a behavior to be modeled because different activation levels represent different priorities. The Action component specifies the action of the behavior, i.e., how the behavior changes the value of a property. This component is the same as a behavior in previous work. In other words, previous work considered only the Action component, and our extension adds a new Activation component.

3.2.2. *Activation Component Specification*

With the differentiation of the Activation component and the Action component, Figure 4 shows how the overall action of a behavior group is computed in previous work (4.a) and in current work (4.b). As can be seen, in previous work, each behavior B_i returns a reference value $\mathbf{v_i}$ (i = 1, 2, 3, …, n), then the final result is averaged among all reference values.

In the current work, along with reference value v_i, each behavior B_i also returns a weight value w_i.

The overall action is the summation of product between normalized weight $\frac{w_i}{sum_w}$ and reference

value v_i of each behavior. Because this weight distribution method sums up all $\frac{w_i}{sum_w}$ to 1 after

(a): Average overall action.   (b): Weight overall action.

Figure 14. Combining desired actions from multiple behaviors.
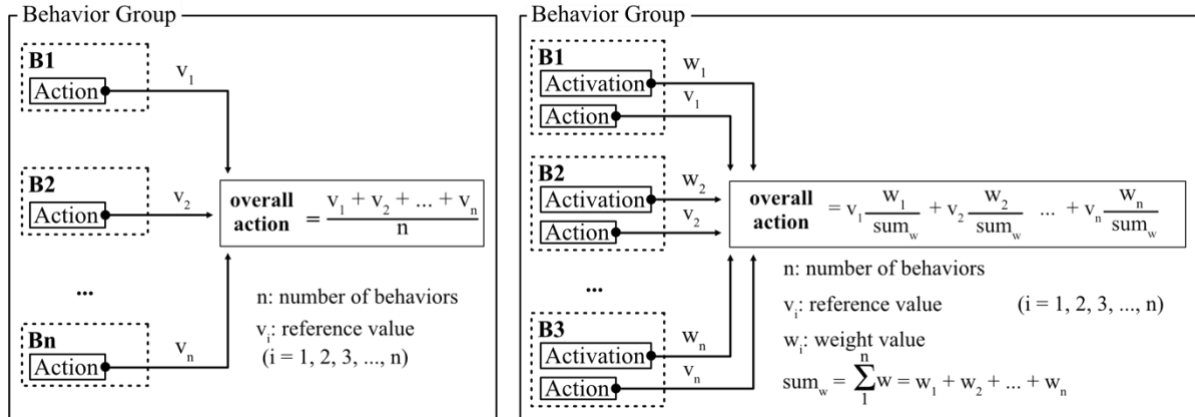
normalization, the more weight a behavior has, the more portion it takes. Hence, this is an approach

to support the priority mechanism.

The steps of the Action component are similar to the steps of the behavior in the previous

work (illustrated in Figure 1). Thus, in this paper we focus on the Activation component. In

general, the Activation component checks the condition of itself or its environment and returns a

weight represents the importance of the behavior. This involves sensing the environment based on

FOV, applying filters, and extracting property (referred to as activation property) in the same way

as in the Action component. Afterwards, it uses an activation function to compute a weight based

on the activation property value. Figure 5 illustrates the steps of the Activation component, which

returns an activation weight in the end. Note that a special case is to check an agent's own property

to computer the activation weight. In this case, a self-filter is used to return the agent itself before the extra property step.

More formally, we define the Activation component to have three elements:

Activation = <F$_a$, p$_a$, activation_function>

F$_a$ is a set of filters where **f$_a$ ∈ F$_a$**

f$_a$ is a filter where:
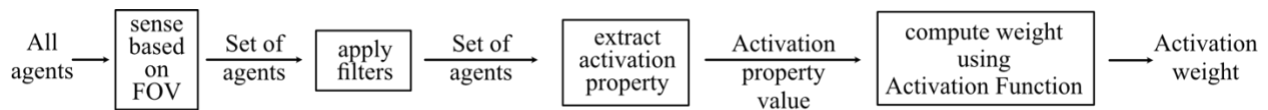
Figure 15. Steps of the activation component

$\qquad$ **f$_a$ = <p$_f$, c$_f$>**

$\qquad\qquad$ **p$_f$** is the filtered property

$\qquad\qquad$ **c$_f$** is the filter criteria:

$\qquad\qquad\qquad$ • if **p$_f$** is *numerical* type, **c$_f$** = [c$_{f\_low}$, c$_{f\_upper}$], where v$_{low}$ is the lower bound, and v$_{upper}$ is the upper bound of the criteria.

$\qquad\qquad\qquad$ • if **p$_f$** *categorical* type, criteria = a subset of the set of all possible categorical values.

p$_a$ is the activation property.

activation_function = <**type**, **r$_a$**> where:

$\qquad$ **type:** activation type: binary or linear (see explanation below).

$\qquad$ **r$_a$** is the activation function range, where **r$_a$** = [r$_{a\_low}$, r$_{a\_upper}$], follows the same regulation as **c$_f$.**

$\qquad$ The activation function computes a weight based on the value of the activation property. This function needs to have a well-defined structure so that it can use the same structure to cover different situations of changing activation weight based on dynamic values of the activation

property. In this work, we allow a behavior to have one of the following two activation function

types: **binary** function and **linear** function.

Binary function returns weight value of 1 or 0 based on if p_a is inside or outside a specified range

$[r_{a\_low}, r_{a\_upper}]$. By giving weight of 0, a behavior is considered not activated because its product is

also equal 0 and not contributed to the overall action. In some cases, modelers want the passing

conditions are not within the range. To capture all these situations, specification of the binary

function is:

binary = <inside> where**:**

      inside: a Boolean value to decide the satisfied conditions are inside or outside the range of

r$_a$

      Table 1 shows the details of the binary function. When **inside** is True, the function returns

weight of 1 if **p$_a$** is within the range of **r$_a$**, and 0 otherwise. When **inside** is False, the function

returns 0 if **p$_a$** is within the range of **r$_a$**, and 1 otherwise.

Table 1. Weight value using binary function.

Inside	Weight value	Function graph
True	$weight = \begin{cases} 1 \ if \ p_a \in [r_{a\_low}, r_{a\_upper}] \\ 0 \ if \ p_a \notin [r_{a\_low}, r_{a\_upper}] \end{cases}$	
False	$weight = \begin{cases} 0 \ if \ p_a \in [r_{a\_low}, r_{a\_upper}] \\ 1 \ if \ p_a \notin [r_{a\_low}, r_{a\_upper}] \end{cases}$	

      Different from the binary function that returns only two values: 0 or 1, the linear function

returns different values based on the inputs of the activation property. The specification of linear

function is:
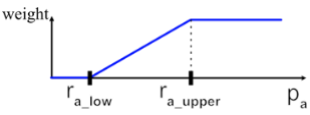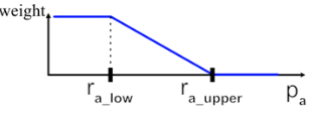
linear = <slope, increase> where:

slope: measures the rate of weight change for different property values**.** It is an integer and has range [0, s_{upper}]. s_{upper} is the upper bound of **slope** value.

increase: a Boolean value to indicate the increment/decrement direction of the weight value. To make the search process more efficiency, modelers need to define upper bound of **slope**. Thus, the search space includes all possible values within range of [0, s_{upper}] (note: in our implementation we define a granularity to make the search space finite). Table 2 illustrates the linear function for various **slope** and **increase** variables. If **increase** is True, the weight increases when $\mathbf{p_a}$ moves toward to the upper bound $\mathbf{r_{a\_upper}}$ of $\mathbf{r_a}$. If **increase** is False, the weight increases when $\mathbf{p_a}$ moves toward to the lower bound $\mathbf{r_{a\_low}}$ of $\mathbf{r_a}$. When $\mathbf{p_a}$ is no longer within range of $\mathbf{r_a}$, the weigh is equal to either **0** or ($\mathbf{r_{a\_upper}} \times$ slope) depends on the $\mathbf{p_a}$ and **increase** value.

Table 2. Weight value using linear function.

Increase	Weight value	Function graph
True	$weigh = \begin{cases} 0 & if\ \boldsymbol{p_a} < r_{a\_low} \\ \boldsymbol{slope} \times (\boldsymbol{p_a} - r_{a\_low}) & if\ r_{a\_low} \leq \boldsymbol{p_a} \leq r_{a\_upper} \\ \boldsymbol{slope} \times (r_{a\_upper} - r_{a\_low}) & if\ \boldsymbol{p_a} > r_{a\_upper} \end{cases}$	
False	$weight = \begin{cases} \boldsymbol{slope} \times (r_{a\_upper} - r_{a\_low}) & if\ \boldsymbol{p_a} < r_{a\_low} \\ \boldsymbol{slope} \times (r_{a\_upper} - \boldsymbol{p_a}) & if\ r_{a\_low} \leq \boldsymbol{p_a} \leq r_{a\_upper} \\ 0 & if\ \boldsymbol{p_a} > r_{a\_upper} \end{cases}$	

3.2.3. Illustrative Example

Below is an illustrative example shows how the activation weight of a behavior is computed at different time steps. Figure 6 shows this example, where an obstacle is moving toward to agent A and the distances between them at time t = 0, 10 and 20 are 130, 75, and 10 respectively.

(a) t = 5   (b) t = 10   (c) t = 15   (d) weight assignments
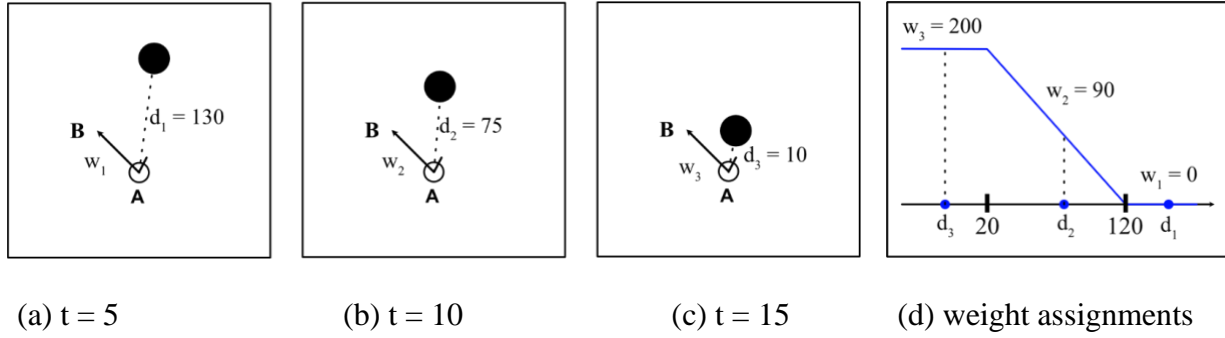
Figure 16. A scenario where obstacle is moving toward to an agent at three different time steps and their weight values.

Behvaior B changes the agent's direction if it is too close to an obstacle. The specification of behavior B is in Table 3

Table 3. Model specification of behavior B.

Activation:	Action:
Filter:	**Filter:**
f_1 : $\mathbf{p_f}$ = type	f_1 : $\mathbf{p_f}$ = type
$\mathbf{c_f}$ = obstacle	$\mathbf{c_f}$ = obstacle
$\mathbf{p_{activation}}$ = position	$\mathbf{p_{action}}$ = position (converted to the direction
$\mathbf{r_{activation}}$ = [20-120]	towards that position and use it as the
(note: distance range between 20 and 120).	reference value).
Activation function:	**offset**: 40 (add 40 degree to the reference
type: linear, **increase** = False, **slope** = 2.	direction)

Behavior B's activation has one filter with filter property = *type* and criteria = *obstacle*. In other word, the filter chooses only obstacles and eliminates regular agents. Because $\mathbf{p_{activation}}$ chooses position as activation property, the linear activation function calculates the activation weight based on range $\mathbf{c_{activation}}$ = [20,120] and the relative distance between position of agent A and position of the obstacle. Figure 6.d shows how the weight is calculated. Because **increase** value is False, weight is decreasing when distance is increasing. In this example, Figure 6.b shows

the distance d2 = 75 and it within activation criteria range. As a result, weight of behavior B at t = 10 is $2 \times (120-75) = 90$. At t = 5 and t = 15, because the distance d_1 (Figure 6.a) and d_3 (Figure 6.c) both exceed the range of activation criteria c_a, weight of B at t = 5 equals to 0 ($d_1 = 130 >$ c_{a_upper}), and weight of B at t = 15 is $2 \times (120 - 20) = 200$ ($d_3 = 10 < c_{a\_low}$). For the Action component of this behavior, it has one filter that is similar to the filter of activation. It then extracts the obstacle's position to get the direction to the obstacle as the reference value. A 40 degree is added to this reference direction to make the agent turn away from the obstacle.

3.2.4.   A Demonstration Experiment.

To demonstrate adding user-defined properties for automated model discovery, we consider an example that we studied in our previous work: snake shape (Table 4 - ID: 4) with personal space (Table 4 – ID: 5,6) [5]. In that example, agents form a snake shape and also maintain personal space, so they do not collide with each other. The discovered model works well if number of agents and world size are set correctly. However, in the situation where there are too many agents in a limited space world, there is not enough space for agents to execute the speeding up behavior. As a result, their speeds decrease to zero and stand still until the rest of the simulation (Figure 7.a). To prevent that from happening, beside snake formation and maintain personal space, we also add speed goal. Function $speedzero(a_i, t)$ (1) adds penalty every time speed of agent a_i is zero at a specific time step *t* by returning value of 1. Equation (2) sums all the penalty during the whole simulation. Hence, the lower the sum value is, the better the speed goal achieves.

$$speedzero(a_i, t) = if\ (speed(a_i, t) = 0)\ return\ 1, else\ return\ 0. \tag{1}$$

$$Number\ of\ speed\ is\ zero\ = \ \sum_{t=0}^{T}\sum_{i=0}^{|A|} speedzero(a_i, t) \tag{2}$$

T is total number of simulation iterations.

|A| is a set of agents, (a_i,t) is the ith agent at time step t.

With this speed fitness function added, a better model is discovered and agents able to maintain acceptable speeds. The behavior to keep the speed is when speed of an agent is in range [0,0.1], it turns left 60 degrees. Hence, open more spaces for agents to speed up. However, agents cannot maintain a stable snake formation because most of the agents change their directions too quickly as Figure 7.b shows.

To find models that can lead to stable snake formation, it is necessary to add a new property that can control how long agents need to wait before changing the direction. Thus, we add a user-defined property called **patience** to the search space. Patience is numerical type with initial value v_{init} = 100 and range = [0,100]. Table 3 shows the final discovered model specification, and the first three (ID: 1-3) are used by agents to wait patiently before changing their directions.

Table 4. A set of behaviors to maintain more stable snake formation.

Behavior Group: Patience		
ID	Behavior specification	Purpose
1	**Activation:** if self-speed is within [0, 0.1], return 1. **Action:** reduce the value of self-patience by 1.	If speed of agent itself is slow, its patience decreases.
2	**Activation:** if self-speed is within [0.2, 2], return 1. **Action:** increase the value of self-patience by 2.	If speed of agent itself is fast, its patience increases.
Behavior Group: Angle		
3	**Activation:** if self-patience is within [5,10], return 1. **Action:** direction increases by 50	If patience of agent itself is low, it steers left 50 degrees.
4	**Activation:** if having one or more regular agent within FOV, return 1. **Action:** change direction to nearest agent	Follow the nearest agent.
Behavior Group: Speed		
5	**Activation:** if the distance to nearest agent is within [0,15], return 1 **Action:** speed decrease by 0.3	Slow down if too close to an agent.

6	**Activation:** if the distance to nearest agent is within [20,150], return 1 **Action:** speed increases by 0.5	Speed up when there is enough space ahead.

Whenever speed of an agent decreases and within range [0 – 0.1], it begins to lose patience by 1 at each iteration. During that time, if agent is able to gain speed above 0.2, its patience increases by 2. If not, the agent will wait until its patience level is below 10, then turn its direction 50 degree to the left. In other word, instead of breaking the snake shape immediately, agents wait until their patience is low. As a result, these behaviors maintain snake formation (Figure 7.c) better than previous one. Without a user-defined property, it is challenging to express a behavior that makes agent wait patiently before making another decision.

(a)   (b)   (c)
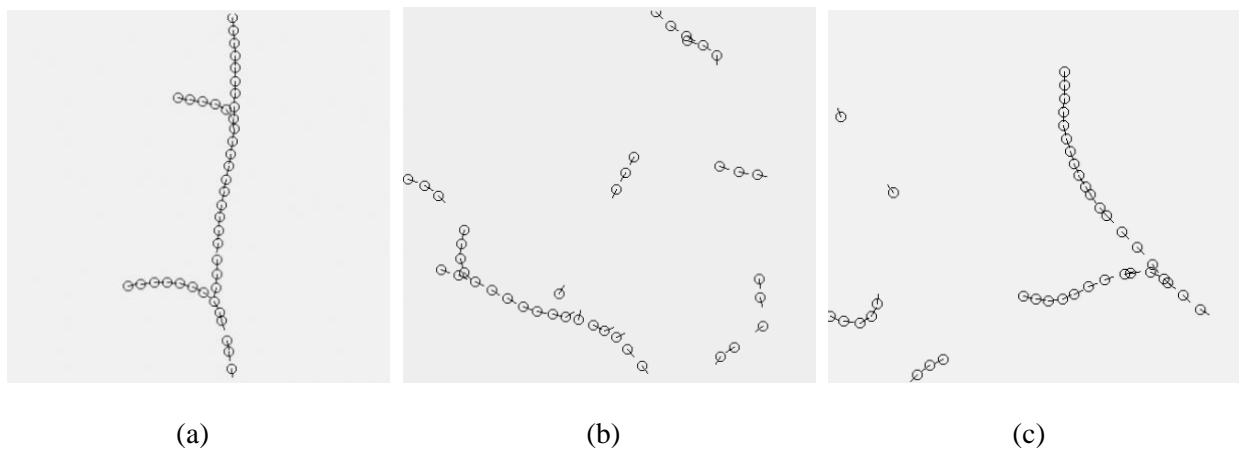
Figure 17. Three experiments of snake formation with personal space.

To compare quantitative results between previous and current experiment, three fitness functions are used include: snake shape, personal space, and speed fitness function. First, snake shape fitness function measures the differences between an agent's direction and the direction toward to its nearest neighbor. Second, personal space fitness function adds penalty whenever agent violate other agents' personal space (Keller and Hu 2019). Next, speed fitness function is calculated using equation (1) and (2). Final fitness score of them is averaged between 100

simulations. Lastly, the scores are normalized to have common scale of 0 being the best, and 1 being the worst. Compare the results between experiment 1 (without **patience** property) and 2 (with **patience** property) of Table 4, we see that experiment 2 gives better score in snake shape and personal space behaviors because the snake shape maintains stably for a longer time. Hence, agents do not change direction as frequency as experiment 1 and have less chance to collide with each other. Speed in other hand, experiment 1 receives the better score because agents in experiment 2 will wait an amount of time before changing the direction when their speed is zero. Thus, more speed penalties are added to it.

Table 5. Fitness scores of experiments with and without patience.

Snake Shape + Personal Space + Maintain speed models	
Experiment 1: without Patience property	**Experiment 2:** with Patience property
Snake Shape: 0.77	Snake Shape: 0.53
Personal Space: 0.513	Personal Space: 0.308
Speed Fitness: 0.12	Speed Fitness: 0.22

3.3.   Genetic Algorithm and Two Stage Searches.

3.3.1.   *Motivations*

As mentioned in section 3.1.1, Genetic Algorithm (GA) is used to find models that satisfy the desired behavior patterns specified by a modeler. The model specification's components are designed to easily interchange and manipulate between models; hence it is reasonable to present them as chromosomes and use GA to encode a potential solution to a specific problem and apply recombination on these structures so as to preserve critical information [49]. We notice that GA search efficiency reduces significantly for complex scenarios because of two main reasons. First, complex scenarios with multiple requirements typically need multiple steering behaviors to fulfill all the tasks. With the automated discovery approach, the more behaviors needed to be discovered,

the larger model space requires. Hence, GA takes much longer time to search for the desired models. Second, GA ranks candidate models based on their fitness scores, and in many cases, conflictions between evaluation criteria make it difficult to rank these models. The leader-following + mobile obstacle avoidance is one example. The leader-following behavior makes agents form a cluster around a leader while mobile obstacle avoidance behavior makes agents scatter. In case of two candidate models with the same score, one has higher score for the leader following fitness and the other one has higher score for mobile obstacle avoidance fitness; GA does not know which one should be rank higher, hence it randomly switches the rank between these candidates and most of the time is trapped in its local optimal.

3.3.2. Properties of Fitness Functions

To rank the generated model correctly, each scenario needs a set of fitness metrics to evaluate how close the discovered models are to the requirements. Depending on the scenarios, each fitness metric can either counts each time agents violate the requirements' rules or check how quick agents fulfill a goal. Because requirements are different, their equivalent fitness metrics should be measure differently as well. For example, in an obstacle avoidance in pedestrian crowd scenario, the obstacle avoidance fitness metric can be set so strict that the model is considered bad with only one violation between the obstacle and a pedestrian. In other hand, many personal space violations between pedestrians can be acceptable. As a result, all metrics are treated with min-max normalization before multiplying together to give the overall fitness score. The normalized score ranges from $[10^{-9} - 1]$, the better a model completes a requirement , the higher score of the equivalent fitness metric is. The minimum score is kept as 10^{-9} instead of 0 because the overall fitness score is the product between all metrics. If the minimum score is 0, it will negate all other metric scores, and does not give accurate evaluations. For example, Table 6 shows overall scores

of three models for a scenario that has two fitness metric f1 and f2. By checking the score for each model, we can easily rank model 3, model 2, then model 1 in descending order. However, with the normalize range = [0-1], all three models' overall score = 0 and are rank the same. With normalized score ranges from $[10^{-9} - 1]$, not only three models now are rank correctly, but the good behaviors also are preserved for the next generations. It is shown in $f_2 = 0.5$ of model 2 and $f_1 = 0.8$ of model 3 where they have a good metric score.

Table 6. Overall scores between different normalize ranges.

Name	Normalize range = [0-1]	Normalize range = $[10^{-9} - 1]$
Model 1	$f_1 = 0$, $f_2 = 0$ Overall score = $f_1*f_2 = \mathbf{0}$	$f_1 = 10^{-9}$, $f_2 = 10^{-9}$ Overall score = $f_1*f_2 = \mathbf{10^{-18}}$
Model 2	$f_1 = 0$, $f_2 = 0.5$ Overall score = $f_1*f_2 = \mathbf{0}$	$f_1 = 10^{-9}$, $f_2 = 0.5$ Overall score = $f_1*f_2 = \mathbf{5^{-10}}$
Model 3	$f_1 = 0.8$, $f_2 = 0$ Overall score = $f_1*f_2 = \mathbf{0}$	$f_1 = 0.8$, $f_2 = 10^{-9}$ Overall score = $f_1*f_2 = \mathbf{8^{-10}}$

3.3.3. Two Stages Search Process

To resolve these limitations in section 3.3.1, we divide the search process into two different stages, and each stage focuses on searching for specific level components of the framework to assist GA use the computational resource better. Figure 18 shows the overview of multiple search stage processes for automated discovery model approach.
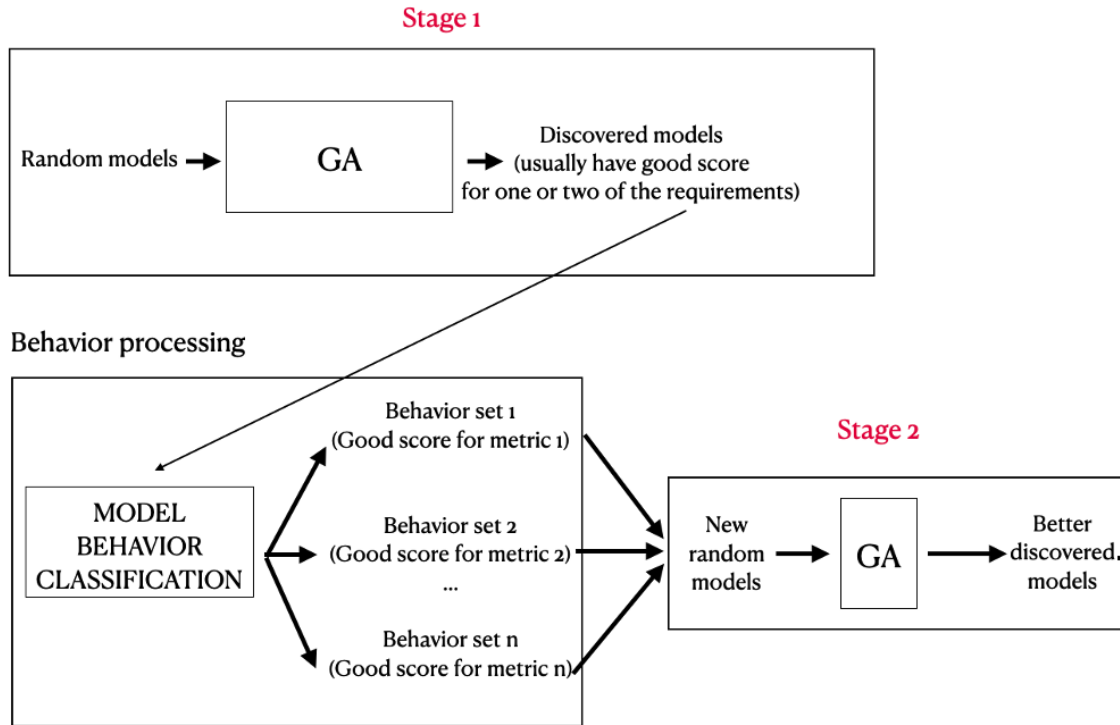
Figure 18. Overview of two search stages for automated discovery model approach.

Stage 1 is described in section 3.1. We notice that at this stage, discovered models usually have high fitness metric scores for one of the requirements. Thus, it is acceptable for discovered models at stage 1 to be low because the goal is to search at model specification level and find the purpose of each behavior. Next, discovered behaviors are classified using a semi-automated process. First, a two-way validation method classifies the purpose of each behavior automatically. Next, modelers further observe how agents move in these selected behaviors and classify them manually based on their movement's patterns.

Because sometimes it needs more than one behavior to fulfill a requirement, using a proposed validation approach, each behavior in one model is tested with two validations to be classified correctly. First validation tests each behavior with a set of fitness metrics to classify the type: one behavior fulfills one requirement. If selected behavior has a high score in one fitness metric, it is considered to satisfy the equivalent requirements. Second validation first removes one

behavior, then tests the remains in a model to classify the type: multiple behaviors fulfill one requirement. If the model gives a bad score for a selected metric, the removed behavior is considered important for the equivalent requirement. The examples in Figure 19 show the reasons why two validations are needed. The model in *Figure* 19.a has both behaviors B1 and B2 that gives a good score for obstacle avoidance metric. If only second validation is used, the classifier will miss both behaviors because if one is removed, the other still gives a high score for the obstacle avoidance requirement. *Figure* 19.b shows a reverse case where the model needs both behavior B1 and B2 to have a good score for the leader following requirement. Hence, if only first validation is used, both B1 and B2 will give a bad score for the requirement and are not selected as candidate behaviors.
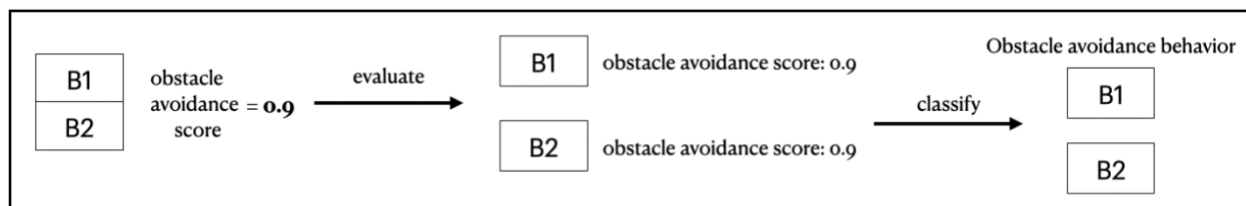
(a)
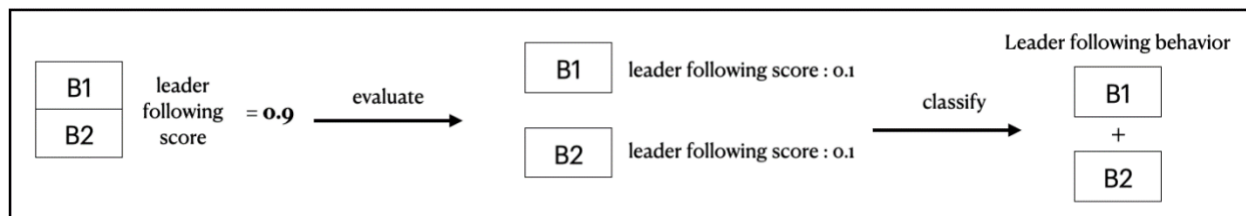
(b)

Figure 19. Classify each behavior fulfills one requirement (a) & multiple behaviors fulfill one requirement (b) examples.

After the classifications, the selected behaviors become the building blocks for stage 2. The next populations for GA are created by combining between one candidate behavior for each

requirement. Because the purposes of these behaviors are already defined, stage 2 mainly focuses on Activation component where GA needs to search for the weights between these behaviors. In other words, stage 2 focuses on component level with the goal to make the candidate behaviors work together to archive the higher overall fitness scores. Because each stage focuses on a different level of model specification, GA is able to put computational resources more efficiently and discover better quality models for complex scenarios. As a result, the two-stage search process shows significant improvements in both searching time and quality of discovered models, especially in complex scenarios where there are more than three requirements. Even though the searching time to discover one set of models takes longer, the overall quality of discovered models is much better. More importantly, GA is able to escape its local optimal, and satisfied models are discovered with fewer number of search.

4. SPACE ENTITY

4.1.   Motivations

Our previous work has mainly focused on agents' interactions with other physical entities (including other agents); hence an agent can only act on properties (e.g., position, direction, speed) of some physical entities. This limitation of considering only the physical entities, hence makes it difficult for agents to move in directions that are not directly associated with any physical entity. For example, action component relies on a reference value from a nearby entity and a static offset value to steer either left or right. Figure 20.a illustrate an example where agent A get an angle reference from agent 1, and steer to the left to avoid it. Figure 20.b shows a different position of agent 1 and agent 2 where the better option is to steer to the right (blue dash arrow). However, agent A final decision is still turn to the left (red solid arrow) because of the static offset.

(a)   *(b)*
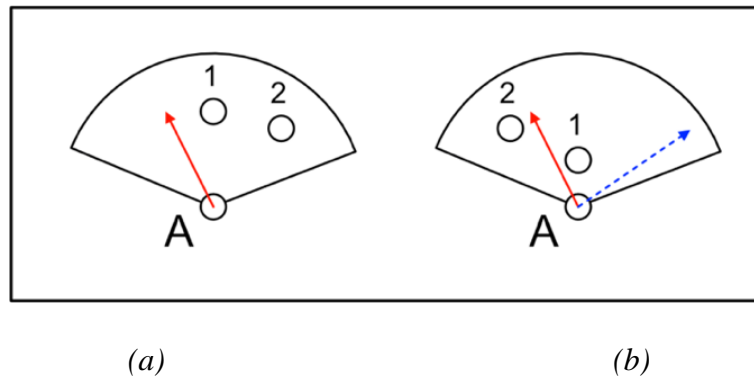
Figure 20. An example shows static offset is limited agent's steering option.

It has been observed that when steering in complex environments, autonomous agents use information not only from nearby physical entities, but also from open space directions surrounding them. In many situations where agents form clusters, they have limited area to move, hence it is crucial for agents to have the capacity to maneuver around empty space efficiently. Even though an open space appears to have nothing, it actually contains useful information such

as how far agents can move along the open space direction, and how much the angle difference is between the open space direction and the agents' current moving directions. Having the capability of processing these types of information would give agents more options for choosing their movement in steering behavior simulation. This would allow agents to move more effectively and naturally in experiment scenarios where utilizing the open space around agents are crucial such as multiple obstacles avoidance, or leader-following.

4.2.   Space Entity Specification

Motivated by the above discussion, we provide a new behavior specification for steering behavior that gives agents the option to interact with the open space surrounding them. Hence, a new category of entity called "space entity" is introduced to add to the physical entities that have been supported in the previous framework. Each agent has a set of space entities corresponding to a set of landmarks that an agent can head to from its current position. At each timestep, an agent's space entities are dynamically generated based on an adaptive space entity generation method, which checks the agent's nearby physical entities and generates a set of possible steering directions adaptively. To work with the existing automated model discovery framework, each space entity has a set of pre-defined properties. For example, the travel distances from the agent to the space entities, or the angle differences between the directions to space entities and the agent' current direction. Agents use the values of these properties to filter which space entity they want to change direction toward. This extension gives agents more flexibility when using steering behaviors. For example, in previous work, when an agent wants to avoid a physical entity, it first needs to get a reference direction from the physical entity. Then a static angle offset is added so that the agent can move away from the entity. This approach is cumbersome and not adaptable (for example, the static angle offset makes the agent always turns left or right with a fixed angle away from the entity

like the example in Figure 20). This extension assists agents move around narrow space more efficiently and adaptively in many situations because the space entities provide multiple reference directions for agents to evaluate.

To fully support an automated discovery model approach, a formal specification of space entity set called **S** is included in entity specification from previous work beside agents, obstacles, and zone entities. **S** is defined as a set of space entities where**:**

s = <position, P> $\in$ **S** where

position is value x and y of s in 2d space.

P is a set of pre-defined properties of a space entity.

4.3.   Space Entity Generation Method

Figure 21 shows an example of how an agent uses the adaptive space entity generation method to generate a set of space entities **S**. Figure 21.a illustrates a current processing red agent A with a 360-degree FOV. Agent **A** has some nearby entities including four agents (**A$_1$, A$_2$, A$_3$, A$_4$**), one black circle obstacle (**O$_1$**), and two rectangle walls (**W$_1$, W$_2$**). To generate set **S**, at each timestep, agent A first groups entities of the same categories (agents, obstacles, or walls) into clusters. Next, a pair of tangent lines called key heading options are created based on the most outward entities' positions of each cluster. Figure 21 (b, c, d) show how agent A generates blue, red, and green key heading options for agent, obstacle, and wall categories respectively. These key headings are important because they present the boundary between occupied segments (colored by blue, red, and green) and empty space segments (white) within an agents' FOV.
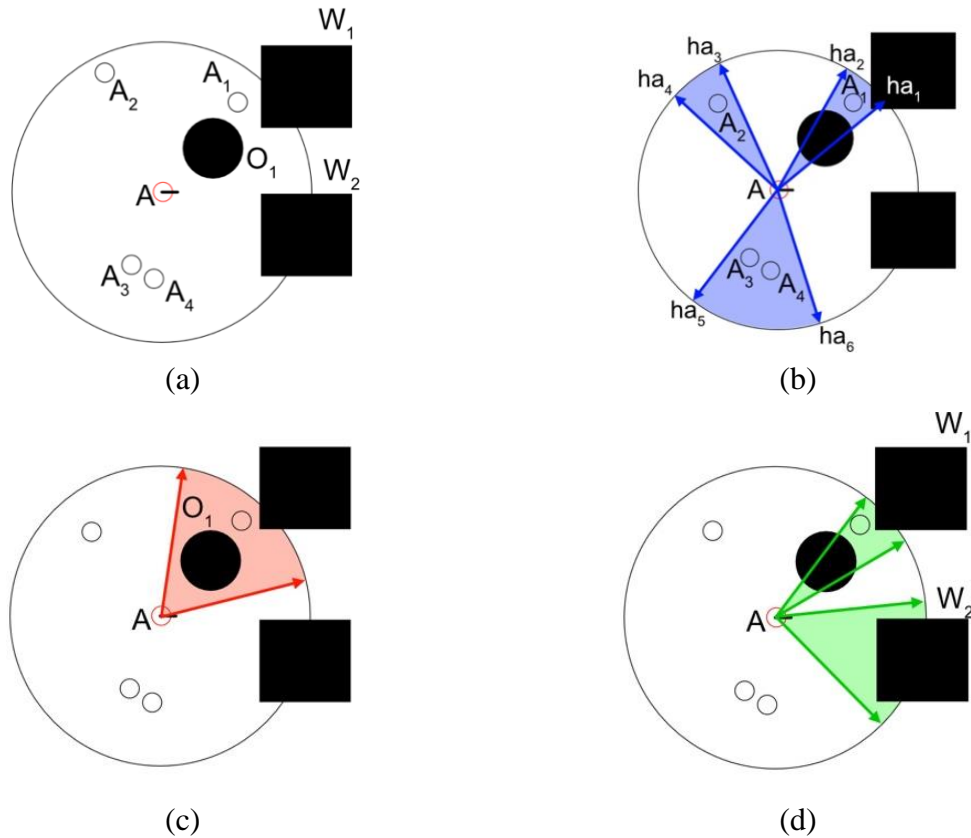
Figure 21. Key heading options of different entity's categories.

Next, all key heading options are combined to create an initial heading set for agent A as Figure 22.a shows. In many cases, the space between two adjacent key headings is so large that it may leave many good potential headings between the boundaries, such as the gap between h_2 and h_3 in Figure 22.a. As a result, staring from FOV's right bound of the agent, more trajectory options are generated incrementally by a pre-set interval, and follow a counterclockwise direction until it reaches FOV's left bound. While increasing, if an option hits the same value as the key heading value before the next interval, the counting starts again at the matched key value (illustrated by hs_1 – hs_7 in Figure 22.b). These heading options are dynamically changed at each time step depending on the positions and heading directions of agents, as well as the positions of nearby neighbors, hence agents have a better spatial set to interact. After all key heading options are generated, their

intersections with the processing agent's FOV become the positions of space entities **s** (Figure 22.c).
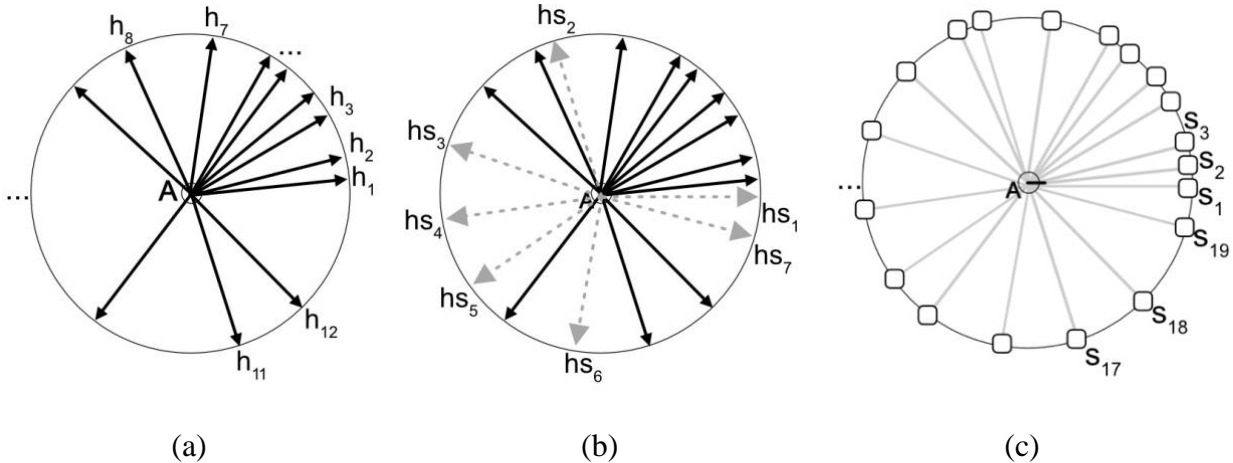
Figure 22. Sub heading options and space entities generated.

4.4. Space Entity Properties

As mentioned above, besides position property, each space entity **s** also has a set of pre-defined properties from the modelers. Because we focus on obstacle avoidance and leader-following steering behaviors, there are two important properties for each space entity: travel distance and angle distance.

4.4.1. Travel Distance

The travel distance property measures various distances on the paths between agents and space entities. In many situations, one heading direction path can be intersected by more than one entity category. Therefore, nearest travel distance values to each and all categories are generated for space entities. Figure 23.a illustrates that a path is drawn from agent A to s_1 and intersected with obstacle O_1, hence the travel distance to the nearest obstacle is d_0. Since there is no agent or wall along the path, the travel distance to the nearest agent and the nearest wall properties are equal to agent A's FOV distance. The travel distance to all entities is assigned by the distance to the

nearest entity (in this example it is O_1 with distance = d_o). Similarly, Figure 23.b shows a case where the path from A to s_2 is intersected with all entity categories. Hence, d_a, d_o, d_w, and d_o are assigned to the travel distances to the nearest agent, obstacle, wall, and all entities respectively. Figure 23.c shows one simple case where the path from A to s_3 is not blocked by any entity. Therefore, all travel distances have values equal to agent A's FOV distance. These property values give agents the flexibility to focus on the category they are interested in. For example, if an agent wants to avoid a nearby obstacle, it only needs to check which path gives the shortest travel distance to the obstacle and avoids using it.

| (a) | (b) | (c) |
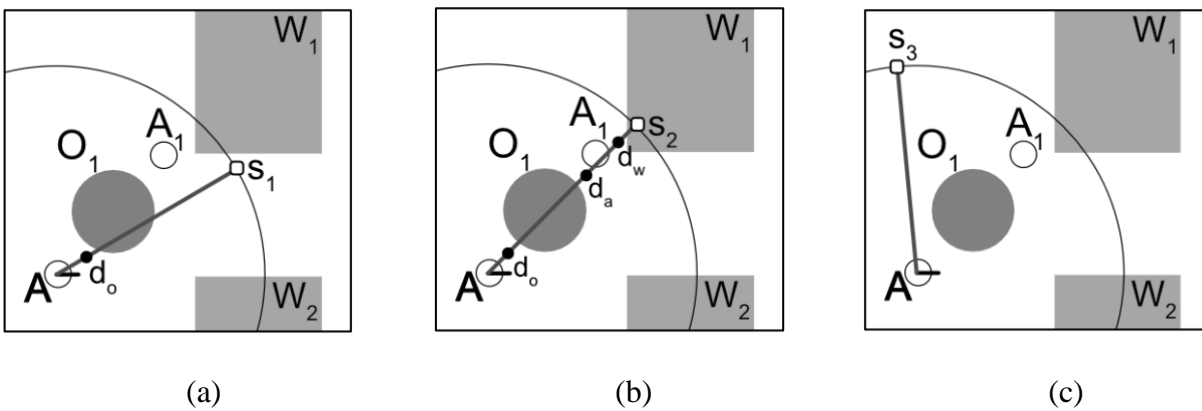
Figure 23. The travel distance property's values of different entity's categories.

4.4.2. *Angel Distance*

Two angle distances are used to measure the relative angle differences between a heading option and agents' heading values. First, in the situation where an agent needs to choose between two or more trajectory options that have the same travel distance, the angle distances between the headings and the agent's current heading direction (ad_c) are used to analyze and compare each heading option further. Figure 24.a illustrates α_1, α_2, α_3 are the angle distances ad_c from s_1, s_2, s_3 to the current heading direction (green arrow) of agent A respectively. Second, we assume that agents have knowledge of directions heading toward a pre-set area in some scenarios such as: the

leader-following or evacuation, hence the angle distance from the desired area to a heading option is provided to guide agents to the correct area. Depending on the layout of the testing experiment, this desired landmark is pre-defined by modelers. For example, Figure 24.b shows points B and C are the bounds of an escape hallway. If an agent is not in the hallway, its position, point B and point C form an area where all heading options inside are considered within desired range direction. In this example, heading options to s_1 and s_2 have angle distances to the desired area ad_d equal to 0. For the options that are outside of the range such as the trajectory between agent A and s3 in Figure 24.b, β_3 is the angle distance between it and the nearest desired area bound (AC). If an agent wants to escape, it will select the space entity where the angle distance to the desired area ad_d is smallest (either s_1 or s_2).

(a) current direction ad_c (b) desired area ad_d.
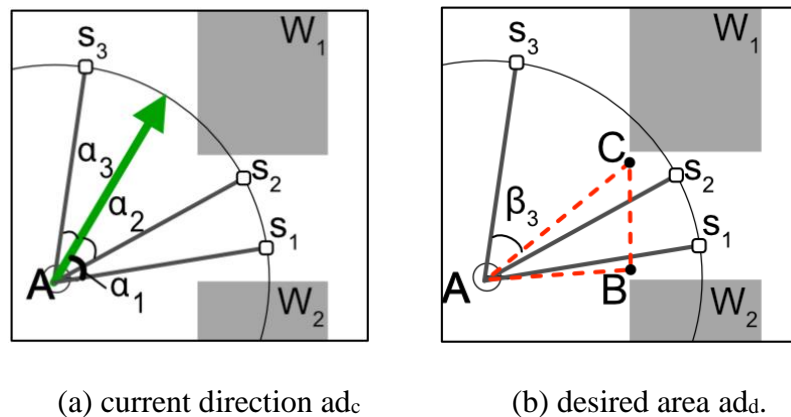
Figure 24. The angle distance property's values to

These distance properties, when combined with filter components from previous work, create a new set of behaviors for agents to interact with the space surrounding them. Not only are agents able to navigate through physical entities more efficiently, but they also have more heading options on both the left and right sides to evaluate. Most importantly, this extension increases the search space with better options and the behavior sets are diverse enough for GA to discover multiple good strategies where agents' movements are more natural and effective.

5. EXPERIMENTS

We evaluate the framework by applying it to discovering simulation models in various steering behavior scenarios. First, we consider three basic steering behavior scenarios including leader-following, mobile obstacle avoidance, and person space maintenance. The discovered behavior models in these scenarios are referred to as $B_{leader\_following}$, $B_{obstalce\_avoidance}$, and $B_{personal\_space}$, respectively. Then, we further evaluate the framework in more complex steering behavior scenarios that combine the basic steering behavior scenarios described above. All the combinations are tested including: (1) $B_{leader\_following}$ + $B_{obstalce\_avoidance}$, (2) $B_{leader\_following}$ + $B_{obstalce\_avoidance}$, (3) $B_{leader\_following}$ + $B_{personal\_space}$, and (4) $B_{leader\_following}$ + $B_{obstalce\_avoidance}$ + $B_{personal\_space}$ scenarios. We also evaluate the framework with two extra scenarios called: (5) the leader-surrounding scenario, and (6) hall-way evacuation with an obstacle in the middle scenario. More discovered models for different set up can be accessed through the webpage (www.). Each agent has a FOV angle = 360 and a FOV distance = 60, its speed has range = [0-2]. Agents' position and heading direction are set randomly (without overlap each other) at the initial steps, and their speed are all = 2.

5.1. Basic Steering Behaviors

5.1.1. *The Leader Following Scenario*

• **Experiment setup:** the leader represents a light from a high altitude; hence all agents know the desired direction toward the leader regardless of the distance between them. The leader does not have physical constraints with other entities, and it changes direction/speed randomly after a period.

• **The leader following fitness score:** Because the only requirement of this scenario is agents are as close to the leader as possible, *Average distance to a leader* measure the average

distance of all agents in |A| set to the leader at the end of each time step (Equation 1). Because all entities are placed randomly at the start of each simulation, agents need a burn-in time (in this case is 1/3 of the simulation time T) to gather around the leader first. As a result, the average distance to the leader is counted for the last 2/3 simulation time.

Equation 1. The leader-following metric equation.

$$Average\ distance\ to\ a\ leader = \sum_{t=\frac{T}{3}}^{T} \left[\frac{\sum_{i=0}^{|A|} distance(a_{i,t}, leader)}{A.size} \right] / (T - T/3)$$

T: simulation time.

- **Specification of discovered models:** Three candidate models are discovered for $B_{leader\_following}$. Figure 25 a, b, c shows heat maps where agents in all three models are fulfilled by following the leaders, however, each of the models presents distinct movements. Model 1 (Table 7), model 2 (Table 8), and model 3 (Table 9) shows model's specifications of the movements presented in Figure 25 a, b, c respectively. For model 1, agents use a set of space entities to decide where to steer next. Because the desired direction in this scenario is toward the leader, agents select the space entity in front of them (within [0-60] of angle distance to current direction ($\mathbf{ad_c}$)) and has the nearest angle distance ($\mathbf{ad_d}$) to the leader. Figure 25.a.1 shows an example where the current direction of agent A is indicated by a green arrow, the light blue segment illustrates the 1st filter of the behavior Table 7:ID-1; thus, s_1 eliminated because it is not within [0-60] of $\mathbf{ad_c}$ of agent A. Between s_2 and s_3, since ad_d = alpha of s2 is smaller, agent A head toward to it. For model 2, it has two filters that make agents select the nearest leaders and extract the leader's position. Then, agents simply steer to the leader position (Figure 25.b.1) and eventually, they gather in a very small area around the leader. In model 1, since each pair of adjacent space entities is set with an interval, most of the time, the selected space entity does not head directly toward the leader. Hence, model 1's

heat map shows agents gather around the leader in a larger area compared to model 2's heat map (Figure 25.a.2 vs Figure 25.b.2). Model 3 although has similar specification as model 2, with the adding offset of 75 to the left of the agent (Figure 25.c.1), it shows a very different movement where agents create a spiral pattern before gathering around the leader. (Figure 25.c.2).
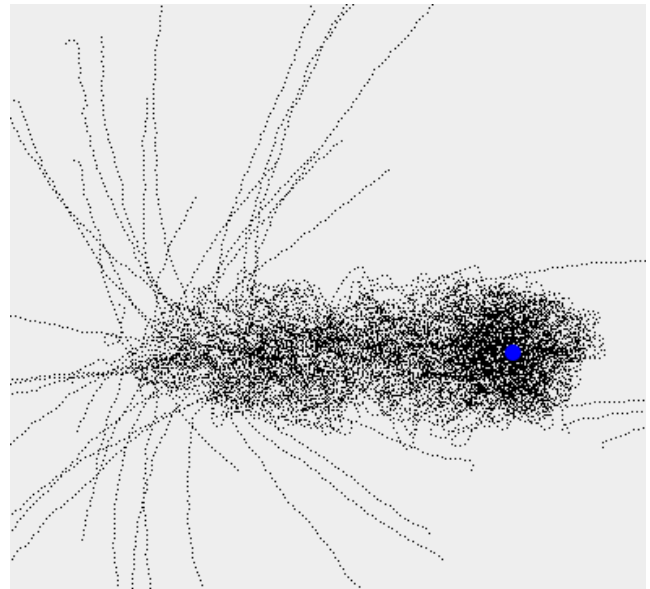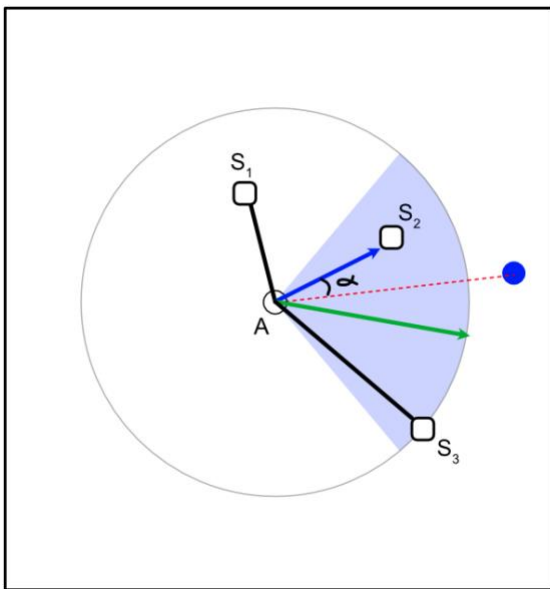
(a.1) Steering decision of a singular agent and (a.2) the heat map of $B_{Leader\_following}$ Model 1
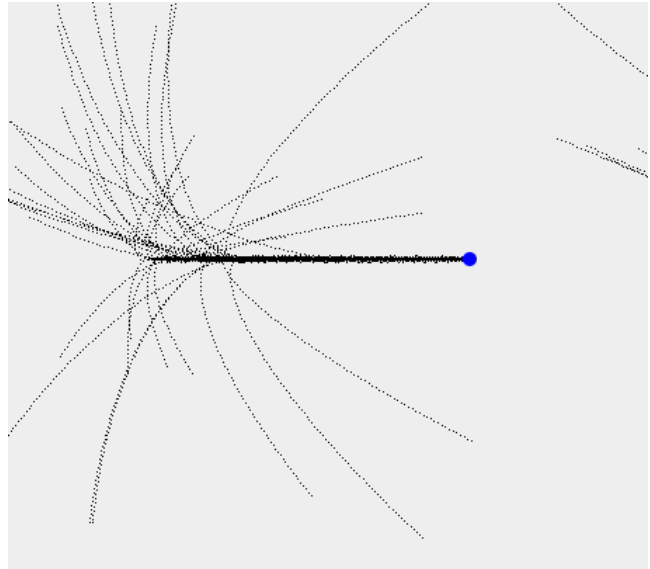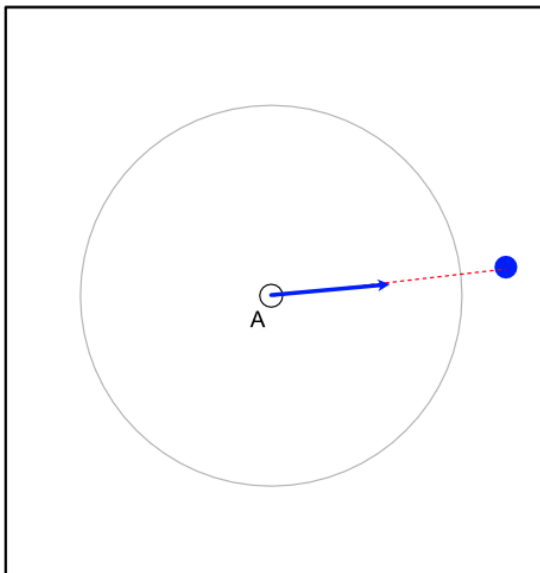
(b.1) Steering decision of a singular agent and (b.2) the heat map of $B_{Leader\_following}$ Model 2
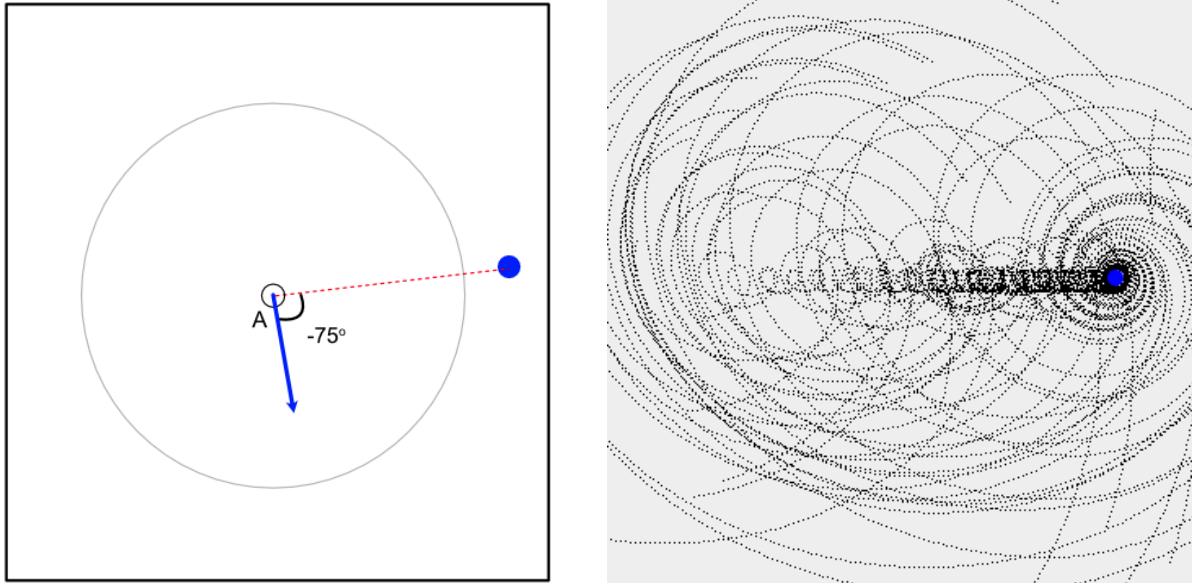
(**c.1**) Steering decision of a singular agent and (**c.2**) the heat map of $B_{\text{Leader\_following}}$ Model 3

Figure 25. Steering decisions and heat maps of three different movement patterns of the

leader following scenario.

Table 7. Model 1 specification of $B_{leader\_following}$.

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		
1	Activation: ▪ Always activate with weight = 1.0 Action: ▪ 1st filter: select the neighbor with the angle distance to the current direction within [0-60] ▪ 2nd filter: select the neighbor with the nearest distance to the desired direction. ▪ Property to extract: position of the chosen space entity.	Steer to the leader using its nearest space location.

Table 8. Model 2 specification of B$_{leader\_following}$.

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		

1	Activation:
	  ■ Always activate with weight = 1.0
	Action:
	  ■ 1st filter: select leader category.
	  ■ 2nd filter: select the nearest neighbor.
	  ■ Property to extract: position of the chosen entity.
	  ■ Offset = 0
		Steer to the nearest leader's position.

Table 9. Model 3 specification of B$_{leader\_following}$.

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		

1	Activation:
	  ■ Always activate with weight = 1.0
	Action:
	  ■ 1st filter: select leader category
	  ■ 2nd filter: select the nearest neighbor
	  ■ Property to extract: position of the chosen entity.
	  ■ Offset = -75
		Point to the leader's position, then add -75 offset to create the spinning movement.

5.1.2. *Mobile Obstacle Avoidance Scenario*

- **Experiment setup:** 6 mobile obstacles are placed randomly in a warped world. Their directions and speeds are randomly changed after a period.

- **Obstacle avoidance fitness score**: This fitness counts every time an agent is overlap with an obstacle. The total number of violations at the end is averaged.

Equation 2. Obstacle avoidance metric equation.

$$\text{Number of violations} = \sum_{t=5}^{T} \sum_{i=0}^{|A|} \sum_{k=0}^{|O|} violation(a_{i,t}, o_{k,t})$$

$$violation(a_{i,t}, o_{k,t}) = \text{if} \left(\text{distance}(a_{i,t}, o_{k,t}) < (agent\ radius + obstacle\ radius) \right) return\ 1\ else\ return\ 0$$

T: simulation time.

(Normalize range [0,1] – A model is considered not good if agents overlap with obstacles more than averagely 1 time at each timestep)

- **Specification of discovered models:** The approach is able to discover three models for $\mathbf{B_{obstalce\_avoidance}}$. The first discovered model (

- Table 10) makes agents avoid obstacles by extracting the position of the nearest obstacle, then adding an offset = 60 to steer agents away from the obstacles (the black solid circle) (Figure 26.a.1). Even though the model has one only one behavior (

- Table 10- ID: 1) and its activation component does not affect the overall decision, this activation specification is important for later experiments. The priority to avoid obstacles is increased when agents move toward the obstacles. With the slope = 100, this obstacle avoidance behavior can easily dominate other behavior. Model 2 (Table 11) simply make agents select a space entity with furthest travel distance to avoid obstacles. Figure 26.b.1 illustrate agent A has three space entities. S_1 and s_2 intersect with obstacle, hence their travel distances are shorter than s_3. By steering to s3's location, agent A is able to avoid the obstacle. $\mathbf{B_{obstalce\_avoidance}}$ of Model 3 is similar to (Table 11- ID:1). However, this model also has speed behaviors where behavior Table 12 – ID:2 slows down agents if they are too near to a nearby neighbor, and speeds up when agents' self-speed is slow (Table 12 – ID:3). This is the first model that is presented in this work that shows the usefulness of Activation component. The speed up behavior has priority pretty low compared to the slow down behavior (10 vs 600). In other words, even if agents want to speed up but there are close by neighbors in front of them, slow down behavior will dominate and stop agents from speeding up. Comparing heat maps of model 1, model 2, and model 3 (Figure 26.a.2, Figure 26.b.2, and Figure 26.c respectively) shows that movement of model 2 utilizes the space surrounding the obstacle better than model 1. Agents in model 2 leave a little empty space around the obstacle

while model 1 leave much larger gap. Model 3 pushes the movement efficiency one step further by limiting the movement of agents. Because agents only move when necessary, the risk of collision is reduced to minimum.
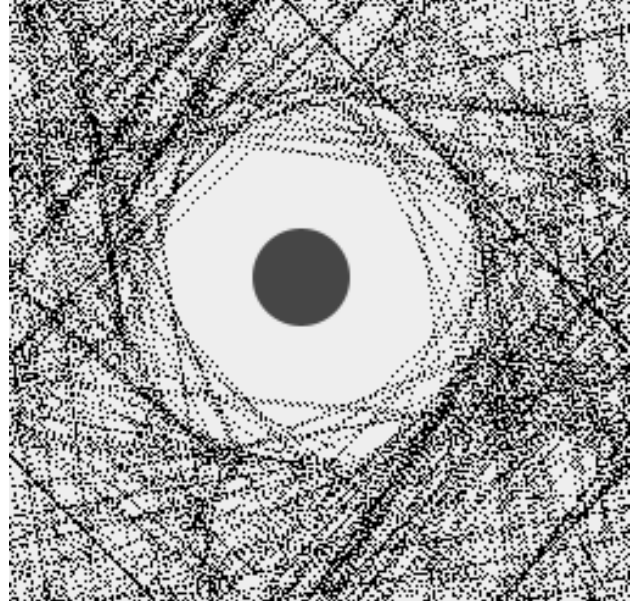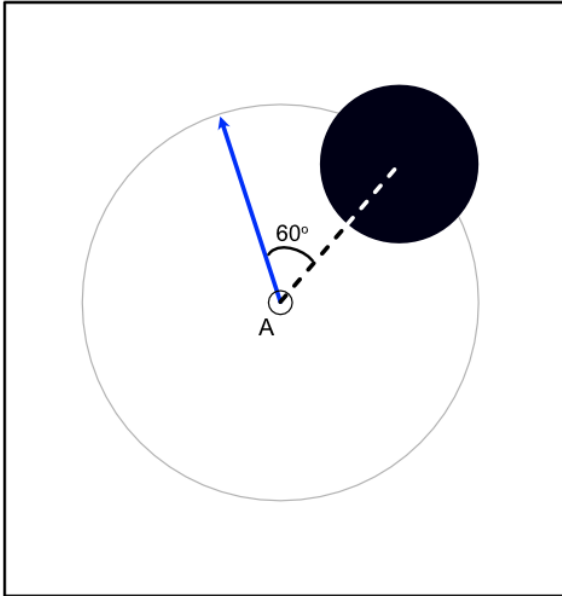
(**a.1**) Steering decision of a singular agent and (**a.2**) the heat map of $B_{Obstacle\_avoidance}$ Model 1
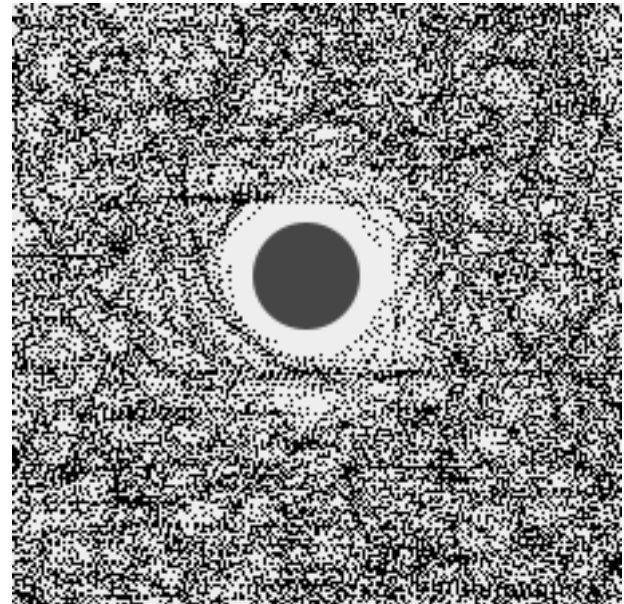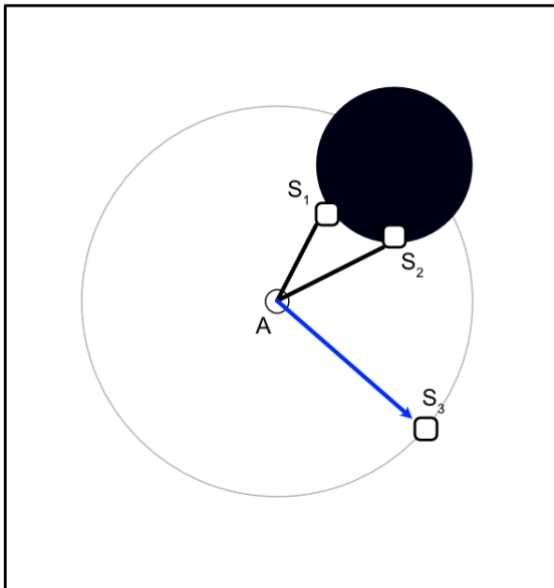
(**b.1**) Steering decision of a singular agent and (**b.2**) the heat map of $B_{Obstacle\_avoidance}$ Model 2
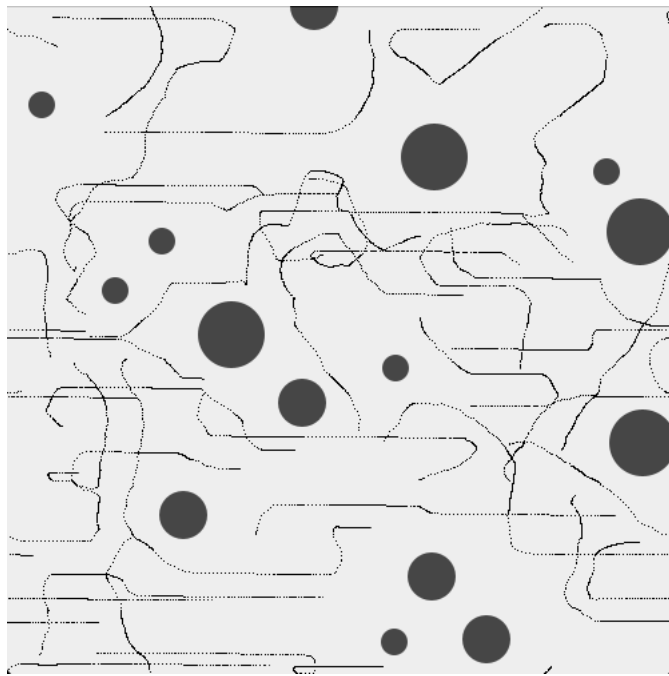
(c) The heat map of B_{Obstacle_avoidance} Model 3

Figure 26. Steering decision and heat maps of three different movement patterns of the mobile obstacle avoidance behavior.

Table 10. Model 1 specification of B_{obstacle_avoidance}.

ID	Behavior Specifications	Interpreted Purpose
	Angel Behavior	

1	Activation:
	   • 1st filter: select obstacle category.
	   • 2nd filter: select the neighbor within [0-40] of the current direction.
	   • Criteria: distance to the select neighbor [0-60]
	   • Linear activation function:
	       o Slope: 100
	       o Increase: False
	Action:
	   • 1st filter: select obstacle category.
	   • Property to extract: position of the chosen entity.
	   • Offset = 60
		Head to the select's obstacle, then add an offset = 60 to steer away from the obstacle. (W = [0-6000]

Table 11. Model 2 specification of $B_{obstacle\_avoidance}$.

ID	Behavior Specifications	Interpreted Purpose
	Angel Behavior	

1	Activation:
	   ▪ Always activate with weight = 1.0
	Action:
	   ▪ 1st filter: Select the space entity with the furthest travel distance.
	   ▪ Property to extract: position of the chosen entity.
		Steer to the furthest space.

Table 12. Model 3 specification of $B_{obstacle\_avoidance}$.

ID	Behavior Specifications	Interpreted Purpose
	Angel Behavior	

1	Activation:
	   ▪ Always activate with weight = 1.0
	Action:
	   ▪ 1st filter: Select the space entity with the furthest travel distance.
	   Property to extract: position of the chosen entity.
		Steer to the furthest space.

	Speed Behavior	

2	Activation:
	   ▪ 1st filter: choose a set of space entities with the angle distance to current direction within [0-45].
	   ▪ Criteria: travel distance to a neighbor [0-5]
	   ▪ Linear activation function:
	      ○ Slope: 100
	      ○ Increase: false
	Action.
	   ▪ Property to extract: speed of the chosen entity.
	   ▪ Offset: -3.0
		Slow down when too close to the nearest neighbor in front.
		(W= [0 – 500])

3	Activation:
	   ▪ Criteria: self-speed is within [0-1.5]
	   ▪ Binary activation function:
	      ○ Inside: true
	      ○ Weight: 10
	Action:
	   ▪ Property to extract: self-speed.
	   ▪ Offset: 2.0
		Speed up when self-speed is slow.
		(W = 10)

5.1.3. *Personal Space Maintenance Scenario*

- **Experiment setup:** 30 agents are placed randomly in a warp world and not overlap with each other.

- **Personal space fitness score:** similar to obstacle avoidance score, this fitness function keeps track every time agents overlap each other.

Equation 3. Personal space metric equation.

$$\text{Number of violations} = \sum_{t=5}^{T} \sum_{i=0}^{|A|} \sum_{k=0}^{|A|} \boldsymbol{violation}(a_{i,t}, a_{k,t})$$

$$\boldsymbol{violation}(a_{i,t}, a_{k,t}) = \text{if } \big(\text{distance}(a_{i,t}, a_{k,t}) < \text{personalSpace} \wedge i \neq k\big) \, return \, 1 \, else \, return \, 0$$
$$personalSpace = agent \, radius \times 2$$

(T: simulation time)

(Normalize range: [0,4] - A model is considered not good if agents overlap with other agents more than averagely 4 time at each timestep)

- **Specification of discovered models:** For personal space maintenance scenarios, there are three discovered models for **B**personal_space. First model selects the space entity with the furthest travel distance. This model is similar to the obstacle avoidance behavior Table 10 -ID:1, hence agents will avoid colliding with nearby neighbors, in this case, other agents. Figure 27.a.1 illustrates agent A has two nearby neighbors A_1, A_2, and three space entities S_1, S_2, S_3. S_1 and S_2 are intersected with A_1 and A_2 respectively, hence their travel distances are shorter than S_3's. Hence, agent A selects S_3 to head toward. To get an informative heat map (Figure 27.a.2) for this model, the initial set up is modified so that there are several agents that have speed = 0, hence serve the same purpose as static obstacles. Therefore, the heat map is able to show how moving agents avoid the static ones. Second model uses a similar behavior to alignment behavior in Reynold's work [1]. Reflect the model's specification in Table 14, Figure 27.b.1 shows agent A

first selects the nearest agent neighbor, then matches its heading direction with the select neighbor. Heat map in Figure 27.b.2 shows that all agents eventually move in the same direction, in this case is from left to right and the direction angle is 4 degree. The third model uses a speed behavior (Table 15- ID:1) to make agents slow down as soon as the simulation starts. The second speed behavior (Table 15- ID:2) tries to speed up agents but its weight is only equal to 1, hence does not contribute much to the final action. However, this speed behavior combination plays a critical role when searching for more complex scenarios later. The heat map (Figure 27.c) shows a minimal movement of all agents and make this strategy minimize the risk of collisions between agents as well. Second and third models of this experiment show that by using the automated approach, the bias from humans is reduced because the approach focuses only on fulfilling the tasks. In this case, these solutions only work because agents are set up to not overlap each other at the initial placement step, and there are no other requirements.
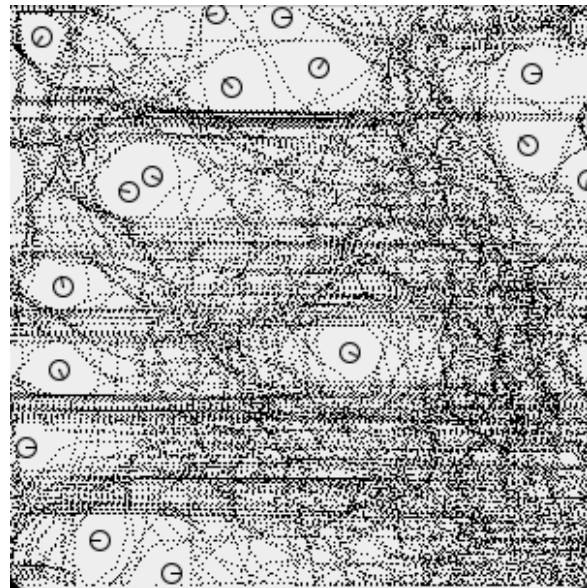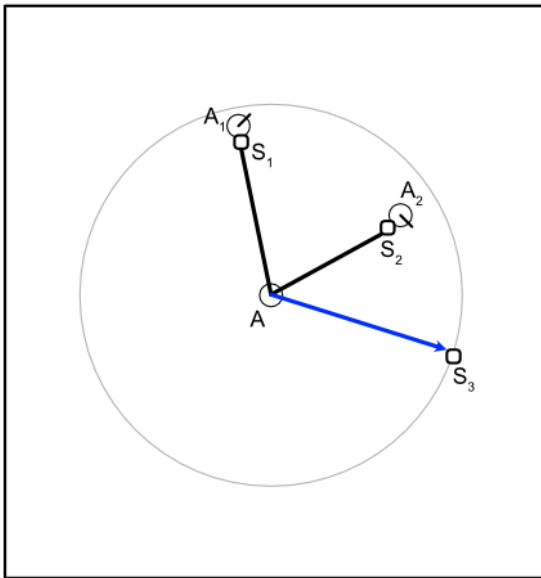
(a.1) Steering decision of a singular agent and (a.2) the heat map of $B_{Personal\_space}$ Model 1
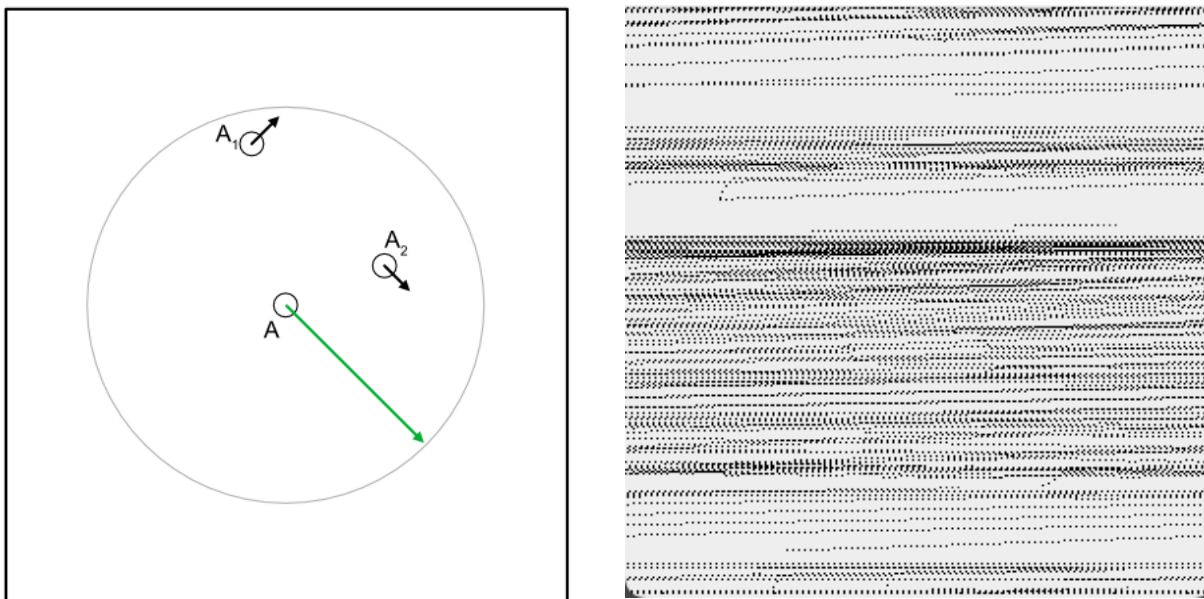
(b.1) Steering decision of a singular agent and **(b.2)** the heat map of $B_{Personal\_space}$ Model 2

(c) The heat map of $B_{Personal\_space}$ Model 3

Figure 27. Steering decision and heat maps of two different movement patterns of personal space behavior.

Table 13. Model 1 specification of personal space maintenance *scenario.*

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		
1	Activation: 　■ Always activate with weight = 1.0 Action: 　■ 1st filter: Select the space entity with the furthest travel distance. 　■ Property to extract: position of the chosen entity.	Steer to the furthest space.

Table 14. Model 2 specification of personal space maintenance *scenario.*

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		
1	Activation: 　■ Always activate with weight = 1.0 Action: 　■ 1st filter: Select the nearest neighbor. 　■ Property to extract: current heading direction	Align with the nearest neighbor.

Table 15. Model 3 specification of personal space maintenance *scenario.*

ID	Behavior Specifications	Interpreted Purpose
Speed Behavior		
1	Activation: 　● Always active, W = 100 Action. 　■ Property to extract: speed of the chosen entity. 　■ Offset: -2.5	Always try to slow down. W = 100
2	Activation: 　● Always active, W = 10 Action: 　■ 1st filter: choose a set of agents with the angle to current direction within [0-50]. 　■ 2nd filter: choose a set of agents within [10-60] 　■ 3rd filter: select the nearest agent. 　■ Property to extract: selected neighbor speed. Offset: 2.0	Speed up when there is no nearby entity.

5.2. Composite Steering Behaviors.

To test the approach for more advanced scenarios, we increase the complexity of the experiments by combining the requirements from the above basic cases. As mentioned in section 3.3.1, we noticed GA search efficiency reduces significantly when searching for candidates in these scenarios because of two reasons. First, the model space becomes very large, hence making it more difficult to search for the good candidates. Second, these scenarios need to fulfill more than one requirement, hence it is challenging for GA to rank the candidates, and the overall scores are likely to be trapped in the local optimum. As a result, we use the above basic steering behaviors as a set of building blocks and combine them based on the scenario requirements to generate the initial population. With this approach, the behaviors of models already have the correct purpose, and the potential candidates are generally well-built from the very beginning. The composite steering behaviors including 6 scenarios: For the combination of 2 basic steering behaviors, we have: (1) $B_{leader\_following} + B_{obstacle\_avoidance}$, (2) $B_{obstacle\_avoidance} + B_{personal\_space}$, (3) $B_{leader\_following} + B_{personal\_space}$, Scenario (4) is **The leader surrounding** $+ B_{personal\_space}$, which is a variation of scenario (3) where the requirement for agents is surrounding the leader with a pre-set distance, instead of come as near the leader as possible. For the combination of 3 basic steering behavior, we have (5) $B_{leader\_following} + B_{obstacle\_avoidance} + B_{personal\_space}$. Scenario (6) is **Hall-way evacuation with an obstacle in the middle**, which is a variation of scenario (5) where we test how agents maneuver to escape in a closed simulation space. The experiment set up and the fitness metric scores, if not mentioned, are the same from the basic steering experiments above (Section 5.1.1, 5.1.2, and 5.1.3).
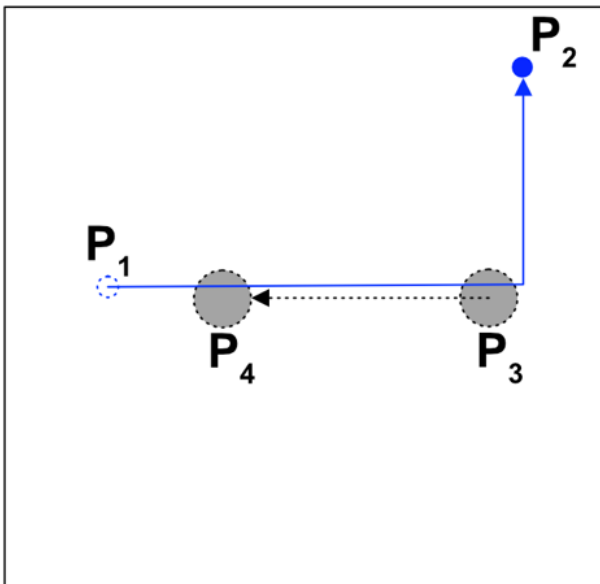
5.2.1. The Leader-Following + Mobile Obstacle Avoidance Scenario

• **Specification of discovered models**: Table 16 shows the $B_{leader+following}$ of discovered models for this scenario are similar to the behaviors in basic scenarios in section 5.1.1. For $B_{obstacle\_avoidance}$, its activation component first selects only obstacles in front of it. It is necessary because agents have FOVs = 360, and they do not need to avoid obstacles behind their travel paths. Based on the criteria and activation function, the priority for this $B_{obstacle\_avoidance}$ increases when the distance of agents to obstacles decreases with a range from [0-6000] for the distance from 60 -> 0. In other words, $B_{obstacle\_avoidance}$ affects the overall steering behavior at the moment agents sense an obstacle. Because the weight of $B_{leader+following}$ of all three models = 1, $B_{obstacle\_avoidance}$ dominates them easily. *Figure 28*.a shows a simple set up to capture the movement patterns of both $B_{leader+following}$ and $B_{obstacle\_avoidance}$. A leader moves from P_1 to P_2 while an obstacle moves from P_3 to P_4 with the intention to make their travel paths intersect each other. *Figure 28*.b, c and d illustrate the heat map of model 1, 2, and 3 in *Table 16* respectively. As expected, $B_{leader+following}$ are similar to the heat map in section 5.1.1. The $B_{obstacle\_avoidance}$ is shown as the interruption in travel movements of agents when following the leader. Beside informing about the gathering area around the leader, the heat maps also show how fast the discovered model gathers agents around the leader as well. The more density along the leader's travel path, the quicker agents surround the leader. Thus, model 2 is the best, followed by model 1, and last is model 3.

Table 16. Model specifications of the leader-following + mobile obstacle avoidance scenario.

Name	$B_{leader\_following}$	$B_{obstacle\_avoidance}$
Model 1	Table 7*: ID – 1. Weight = 1*	Activation: • 1st filter: select obstacle category. • 2nd filter: select the neighbor within [0-40] of the current direction. • Criteria: distance to the select neighbor [0-60] • Linear activation function: ○ Slope: 100
Model 2	Table 8*: ID – 1. Weight = 1*	

Model 3	Table 9: *ID – 1. Weight = 1*	○ Increase: False Action: Similar to action component of Table 11: ID – 1.

(a) Initial set up to generate the heat map.
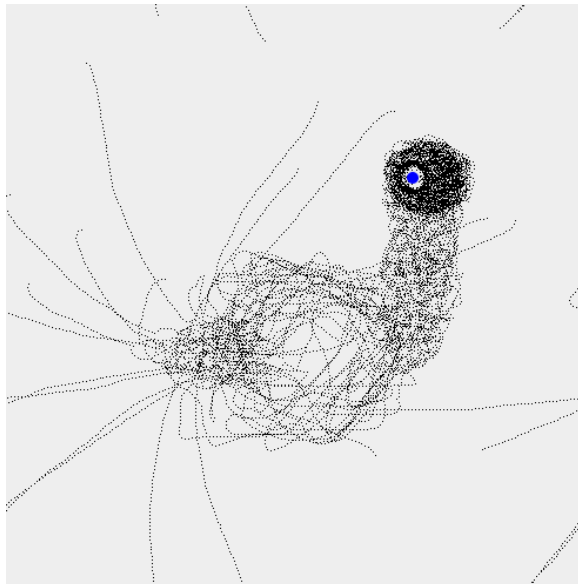
(b) Model 1
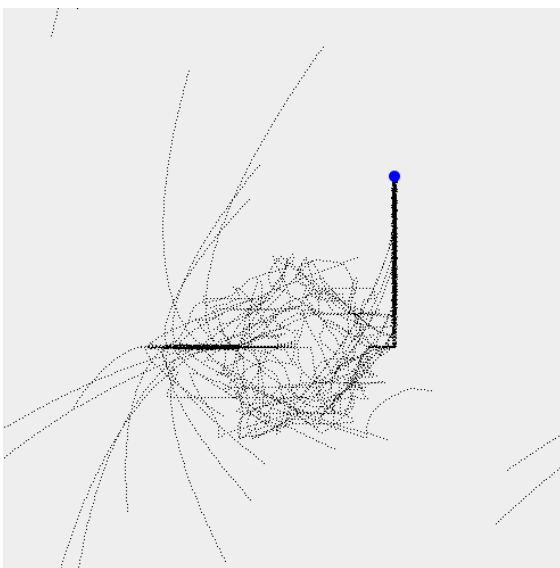
(c) Model 2
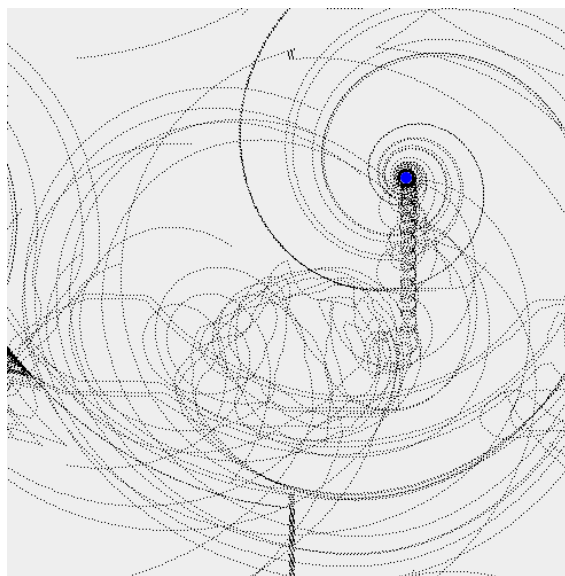
(d) Model 3

Figure 28. Heat maps of three different movement patterns of $B_{leader\_following} + B_{obstacle\_avoidance}$ scenario.

5.2.2. *Mobile Obstacle Avoidance + Personal Space Maintenance Scenario*

Discovered models for this scenario are similar to models in Table 11 and 12 even though these basic steering models should only fulfill one requirement. In this scenario, it works because one of the travel distance properties of a space entity measures the distance to all entities. As a result, beside avoiding obstacles, the $B_{obstacle\_avoidance}$ in Table 11 and 12 also avoid other agents as well. Heats map in Figure 29.a & b shows similar pattern as Figure 26.b.2 and c as well. However, with personal space maintenance requirements, agent density that Model 2 can handle will be higher than Model 1 because agents in Model 1 will constantly move; hence personal space violation is more likely to occur.

| (a) Model 1 | (b) Model 2 |
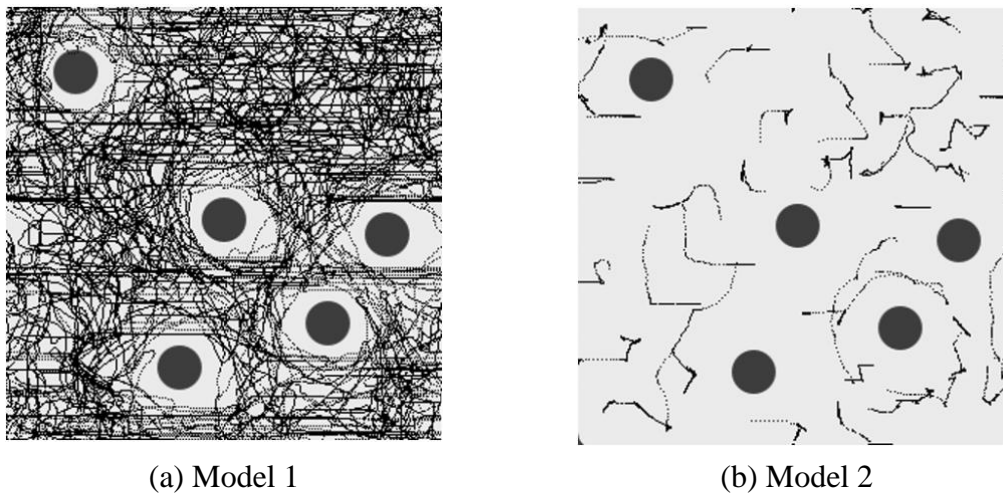
Figure 29. Heat maps of three different movement patterns of $B_{obstacle\_avoidance}$ + $B_{personal\_space}$ scenario.

5.2.3. **The Leader-Following + Personal Space Maintenance Scenario**

For this scenario, GA is able to discover three different strategies to satisfy these requirements. $B_{leader\_following}$ behaviors of these three models (Table 17) are similar to model 1, model 2, and model 3 in Table 7, Table 8, and Table 9 respectively. For $B_{personal\_space}$, Model 1 of Table 17 selects the furthest space to make agents avoid their neighbor and move faster the leader's speed by 0.5. After forming a cluster around the leader where agents are very near each other, they

do not collide with each other even though they constantly move. By moving slightly faster than the leader, agents move very slowly when the leader stops, hence giving them enough time to respond to the surrounding environment. Table 18 shows $B_{personal\_space}$ behaviors used for Model 2 and Model 3 in (Table 17). It is a speed behavior to help agents slow down when too close to another neighbor, and speed up when there is enough space in front of it. It is a better variation of Table 15's model because the weight of slowing behavior is dynamically changed depending on the surrounding environment. Hence, this $B_{personal\_space}$ is robust and can handle the larger scale experiments later on. Figure 30 shows that at the end of the simulation, there is no overlap between agents, and distinguishes movement patterns across all models. Compared to Figure 25, heat maps of three models in Figure 31 shows that the gathered areas of agents around the leaders are significantly larger due to $B_{personal\_space}$.

Table 17. Model specifications of the leader-following +Table 17 personal space maintenance scenario.

Name	$B_{leader\_following}$	$B_{personal\_space}$
Model 1	Table 7: ID – 1. Weight = 1	Table 13: ID – 1. Weight = 1
Model 2	Table 8: ID – 1. Weight = 1	Table 18: ID – 1 Weight = [0-500]
Model 3	Table 9: ID – 1. Weight = 1	ID -2 Weight = [10]

Table 18. Speed behavior used for Model 2 and Model 3 of the leader-following + personal space maintenance scenario.

ID	Behavior Specifications	Interpreted Purpose
	Speed Behavior	

1	Activation:
	 ▪ 1st filter: choose a set of space entities with the angle distance to current direction within [0-50].
	 ▪ Criteria: travel distance to a neighbor [0-5]
	 ▪ Linear activation function:
	     o Slope: 100
	     o Increase: false
	 Action.
	 ▪ Property to extract: speed of the chosen entity.
	 ▪ Offset: -3.0
		Slow down when too close to the nearest neighbor in front.
		 (W= [0 – 500])

2	Activation: • Always active, W = 10 Action: ▪ 1st filter: choose a set of agents with the angle to current direction within [0-50]. ▪ 2nd filter: choose a set of agents within [10-60] ▪ 3rd filter: select the nearest agent. ▪ Property to extract: selected neighbor speed. ▪ Offset: 2.0	Speed up when self-speed is slow. **(W = 10)**
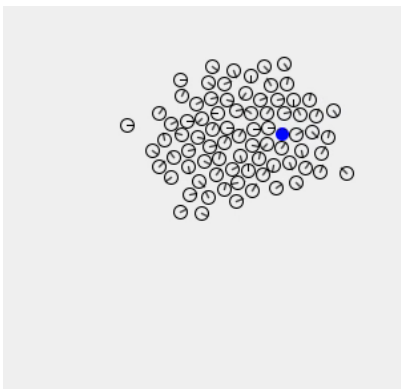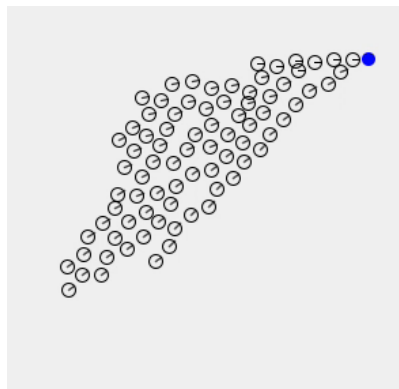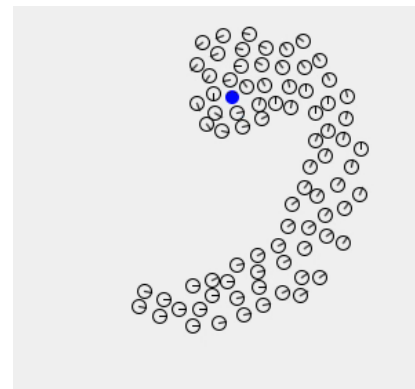
(a) Model 1   (b) Model 2   (c) Model 3

Figure 30. Simulation snapshots of three different movement patterns of leader_following + personal space scenario.

(a) Model 1   (b) Model 2   (c) Model 3
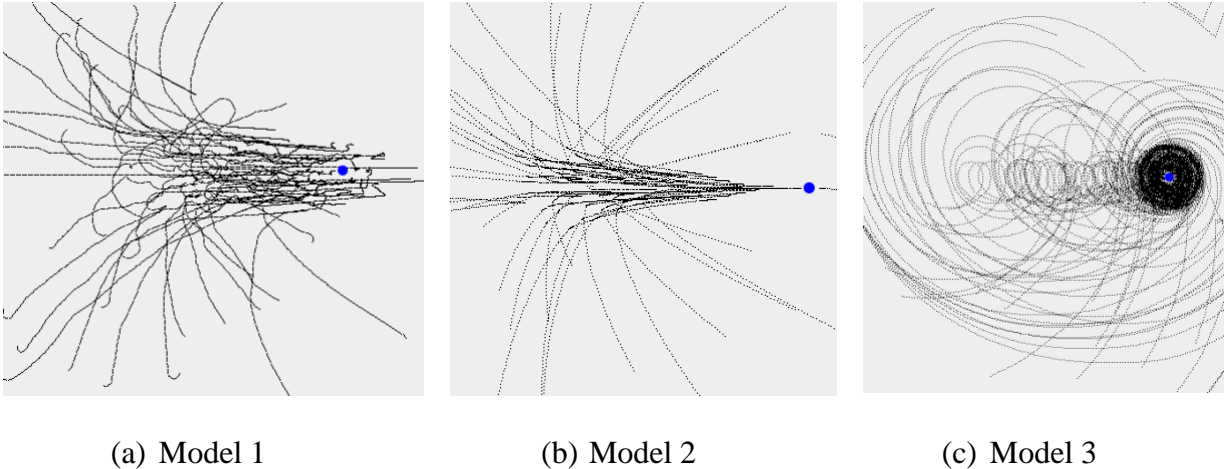
Figure 31. Heat maps of three different movement patterns of $B_{leader\_following} + B_{personal\_space}$

scenario.

5.2.4. The Leader Surrounding + Personal Space Maintenance (2 models)

• **Experience set up:** One leader is placed randomly on the map and, changes direction/speed randomly after a period.

• **Fitness metrics:**

o   **Personal space score:** as same as Equation 3.

o **The leader-surrounding score:** this fitness metric measures two conditions. First one is that agents need to maintain a certain distance to the leader. Thus, the leader-following metric's Equation 1 is modified so that if the average distance to the leader is within [20-60. With only the first condition, the surrounding phenomenon is not captured because agents can maintain the distance to the leader with one big cluster. As a result, a second condition is needed to measure the number of agents around the leader. The area around the leader is divided into 8 segments, and 80% of the agents need to be inside all 8 segments for the score to be considered good enough.

• **Specification of discovered model:** Unlike the leader-following score where agents move toward the leader as close as possible, the distance to the leader in this case cannot be too

close or too far, hence makes the scenario more difficult to have good discovered models because

the model space speciation can only select the nearest or furthest nearby entity.

Table 19. Model 1 of the leader-surrounding + personal space scenario.

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		

1	Activation:
	 ■ 1st filter: select only leader category.
	 ■ 2nd filter: select the nearest neighbor.
	 ■ Criteria: distance to a neighbor [0-60]
	 ■ Linear activation function:
	   ○ Slope: 10
	   ○ Increase: true
	Action:
	 ■ 1st filter: select the neighbor with the nearest distance to the desired direction.
	 ■ Property to extract: position of the chosen space entity.
	 ■ Offset: -45.0
		Circle around the leader when not too close.
		(W = [0-600])

2	Table 7 – *ID: 1*	Follow the leader **(W= 10)**
Speed Behavior		
3	Table 18 – ID: 1	Slow down when too close to the nearest neighbor in front. **(W= [0 – 500])**
4	Table 18 – ID: 2	Speed up when self-speed is slow. **(W = 10)**

Table 20. Model 2 of the leader-surrounding + personal space scenario.

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		

1	Activation:
	■ 1st filter: choose the nearest entity.
	■ Criteria: angle distance to current direction [0-60].
	■ Linear activation function:
	   o Slope: 5
	   o Increase: false
	Action:
	■ 1st filter: choose the nearest entity.
	■ Property to extract: position of the chosen space entity.
	■ Offset: -60.0
		Steer away from a nearby entity if it is in front of.
		(W = [0-300])

2	Activation:
	■ 1st filter: select only leader category.
	■ 2nd filter: select the nearest neighbor.
	■ Criteria: distance to a neighbor [0-40]
	■ Linear activation function:
	   o Slope: 20
	   o Increase: true
	Action:
	Similar to Action component of (Table 9 - ID: 1)
		Point to the leader's position, then add -75 offset to create the spinning movement.
		(W = [0-800])

Speed Behavior		
3	Table 18 – ID:1	Slow down when too close to the nearest neighbor in front. **(W= [0 – 500])**
4	Table 18 – ID:2	Speed up when self-speed is slow. **(W = 10)**

To make the agents surround the leader, **Table 19***: ID-1 is a variation of* **Table 9** *and it makes agents first steer to the space entity with nearest angle distance to desired direction (**ad**_d) when they are near the leader, then turn 45 degrees to the left. The activation makes agents want to surround the leader most when they are far away (the weight is highest when the distance to the leader ≥ 60), and the surrounding desire reduces when the distances to the leader reduce; hence agents are able to keep their space around the leader. These scenarios once again show the important role of the Activation component because it helps two steering behaviors with two different purposes work together. When far away from the leader, agents have higher priority to*

surround it first. However, their desires decrease until they are close enough. Because the leader is moving, the second steering behavior (Table 19: ID-2) assist agents to follow the leader, while still keep the circle formation around it. Speed behaviors (Table 19:ID-3+4) are similar to the speed behavior set of Table 18. To keep the circle formation while the leader is moving, Table 19: ID-2 makes agents steer to the leader. Model 2 shows a different strategy where there are three behaviors to fulfill personal space requirements. Two speed behaviors that are similar to Table 18 – ID: 1 and 2, and an angle behavior (

Table 20 – ID: 1) that makes agents steer away from a nearby neighbor in front of it when too close. The spinning movement behavior (

Table 20 – ID: 2) is similar to model 1 with the same priority assignments. With these model specifications, agents that are far away from the leader steer around the leader as soon as the simulation begins. They continue to reduce their distances to the leader until a set of agents is close enough and starts to reserve their distances to the leader as

Figure 32.a and b illustrate. For model 1, because these agents cannot both form a single circle while maintaining the distances to the leader, the outer group tries to form the second rings as the heat map in

Figure 32.a shows, to make the leader-surrounding score as high as possible. For model 2, it does not work as well as model 2 because the behavior

Table 20 – ID: 1 not only makes agents steer away from the neighbor, but also from other agents as well. As a result, agents surround the leader in a much larger area, and not enough agents to form a circle shape surrounding the leader like Model 1. (The travel path of the leader that is

illustrated in the heat maps is similar to set up in section 5.2.1 *Figure 28*.a but without the involvement of the obstacle).

(a)   (b)

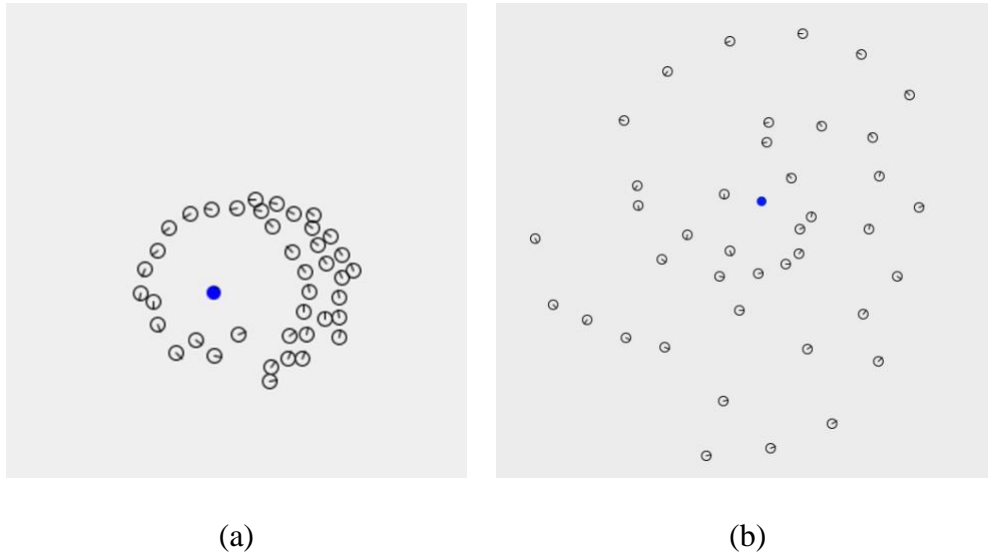*Figure 32. Snapshots of two different movement patterns of the leader-surrounding + personal space scenario.*

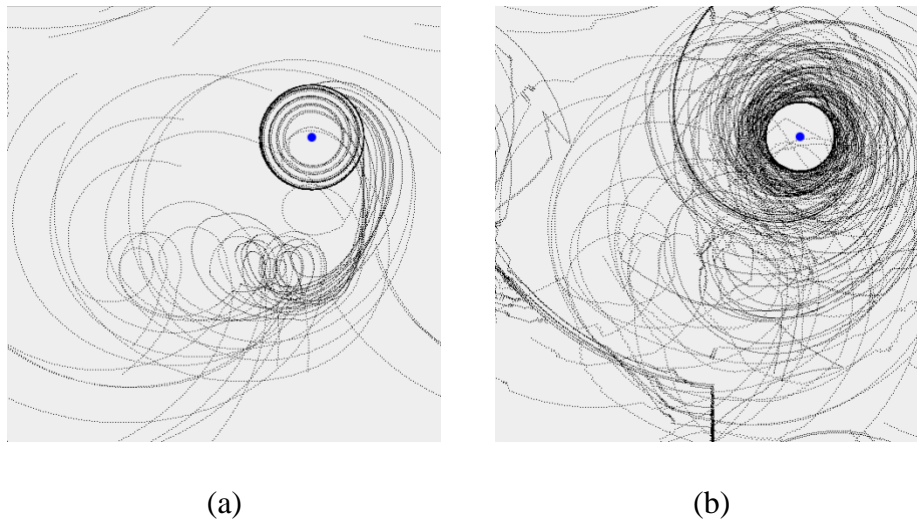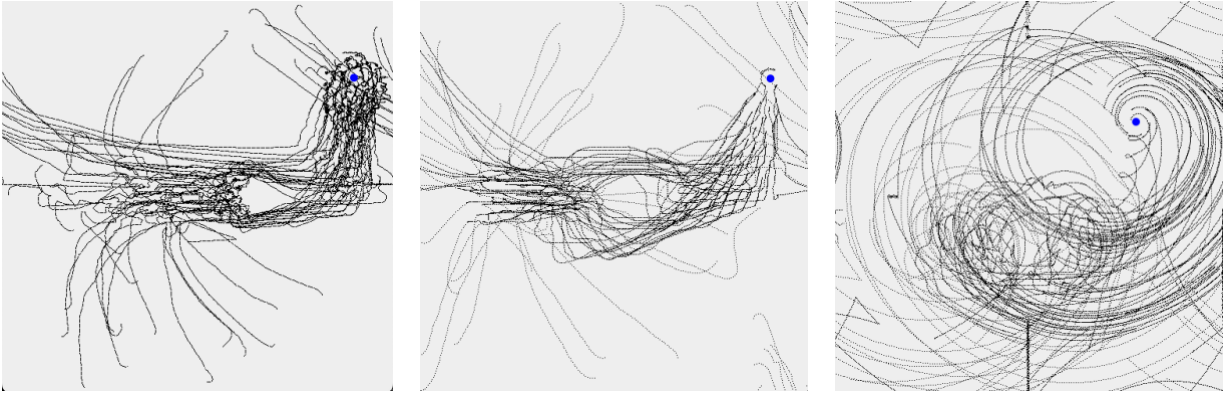(a)   (b)

Figure 33. Heat maps of two different movement patterns of the leader-surrounding + personal space scenario.

5.2.5. *The Leader Following + Personal Space Maintenance + Mobile Obstacle Avoidance Scenario*

We increase the complexity of the experiment further by adding $\mathbf{B_{obstalce\_avoidance}}$ requirement beside $\mathbf{B_{leader\_following}}$ + $\mathbf{B_{personal\_space}}$. *Figure* 34.a shows the initial set up for a simple demonstration to show the difference in movement patterns between candidates. A leader (blue circle) is moving from position P1 to position P2 while an obstacle (gray circle) is moving in the opposite direction from P3 to P4. These two objects' travel paths intentionally overlap each other to show the gather and scatter movement pattern of agents. Figure 34.b and *Figure* 34.c shows the heat map movements of two discovered models. In both models, one steering behavior makes agents gather around the leader while another steering behavior makes agents scatter when nearby an obstacle. These two behaviors are conflicting with the purpose of each other. If agents focus too much on forming a large cluster around the leader, there is a high chance they will collide with obstacles. On the other hand, if agents avoid obstacles intensively, they will not be able to gather near the leaders. As a result, the role of activation component in this scenario is critical because it assists to adjust the sensitivity of both behaviors. Based on the discovered models' specifications, agents always head to the leader location. However, this behavior has a small weight. The $\mathbf{B_{obstalce\_avoidance}}$ has its priority gradually increasing when the distance between agents and the nearest obstacle is reduced. Hence, eventually, agents' priority to avoid the obstacle is so large that it dominates $\mathbf{B_{leader\_following}}$. As a result, for these models, if there are obstacles nearby, agents always avoid the obstacle first before considering moving toward the leader. For the $\mathbf{B_{personal\_space}}$ behavior, since it is a speed behavior, it does not create the conflict like the two steering behaviors above. *Figure* 34.a & b show the travel paths of discover models where a group of agents are able to avoid a moving toward obstacle and form the cluster around the leader afterward. The main

difference is **B**leader_following takes advantage of space entities in Model 1 (*Figure* 34.b), agents form a cluster faster, and are able to wait until the obstacle is very near before scattering; compared to Model 2 (*Figure* 34.c) where **B**leader_following is simply steer to the leader position.

(a) Model 1   (b) Model 2   (c) Model 3

Figure 34. Heat maps of three different movement patterns of the leader-following +

personal space + mobile obstacle avoidance scenario.

5.2.6. Hall-Way Evacuation with an Obstacle in the Middle

We choose this experiment because of two reasons. First, it is an evacuation scenario and agents now need to move a close tight space instead of open space like the experiment above. Second, this example presents a practical example where our approaches can be used.

• **Experiment setup:** Figure 35.a illustrates the setting with a hallway that is constructed by 2 rectangle obstacles above and below the simulation space, and a large circle obstacle presented in the middle. All agents have an initial heading direction = 0, are spawned within the dash line rectangle area on the left and are removed when they reach the blue rectangle on the right. Points P1 and P2 are pre-defined locations to form a desired heading area with an agent's position.

• Because this scenario is an evacuation application, it needs two fitness functions: the "**time fitness function**" measures how fast all agents can escape, and the "**remain agent fitness function**" counts the number of remaining agents after the simulation time is up. It is used to compare between models that cannot evacuate all agents within a limited time.
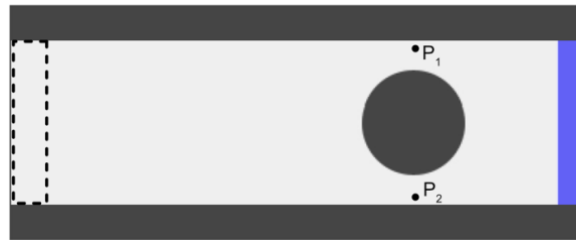
(a) *Initial set up to generate the heat map.*

(b.1) Model 1 simulation snapshot
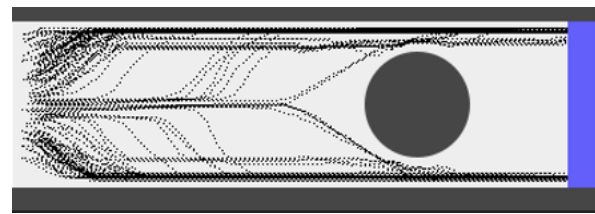
(b.2) Model 1 heat map
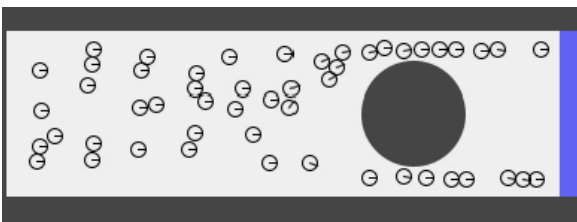
(c.1) Model 2 simulation snapshot
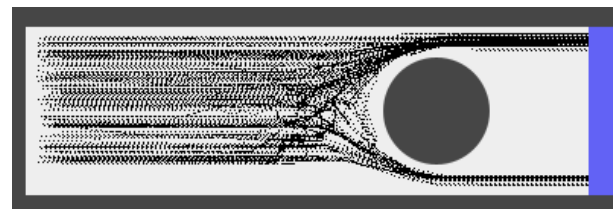
(c.2) Model 2 simulation snapshot

Figure 35. Scenario setup, simulation snap shots and the heat maps of two discovered models for hallway with obstacle scenario.

Table 21. Model 1 specification for $B_{goal\_reaching} + B_{personal\_space}$

ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		

1	Activation:
	     ▪ Always activate with weight = 1.0
	 Action:
	     ▪ 1st filter: select the neighbor with travel distance to obstacle within [41-60]
	     ▪ 2nd filter: select the neighbor with the angle distance to the current direction within [0 - 60]
	     ▪ 3rd filter: select the neighbor with the nearest distance to the desired direction.
	     ▪ Property to extract: position of the chosen space entity.
		Steer to the desired direction and avoid obstacles if needed.

Speed Behavior		
2	Table 18 – ID: 1	Slow down when too close to the nearest neighbor in front. **(W= [0 – 500])**
3	Table 18 – ID: 2	Speed up when self-speed is slow. **(W = 10)**

Table 22. Model 2 specification for $B_{goal\_reaching} + B_{personal\_space}$

| ID	Behavior Specifications	Interpreted Purpose
Angel Behavior		

1	Activation:
	     ▪ 1st filter: choose a set of space entities with the travel distance to obstacles within [0-50].
	     ▪ Criteria: travel distance to a neighbor [0-60]
	     ▪ Linear activation function:
	        o Slope: 1
	        o Increase: false
	 Action:
	     ▪ 1st filter: choose a set of space entities with the travel distance to obstacles within [0-50].
	     ▪ 2nd filter: select the neighbor with the nearest distance to the desired direction.
	     ▪ Property to extract: position of the chosen space entity.
		Move toward the wall (rectangle obstacle), then steer to desired direction. [W = 0 -50]

2	Activation:
	    ▪ Always activate with weight = 1.0
	Action:
	    ▪ 1st filter: select the neighbor with the furthest travel distance.
	Property to extract: position of the chosen space entity.
		Move along the wall by head toward to farthest travel distance to obstacles.

Speed Behavior		
3	Table 18 – ID: 1	Slow down when too close to the nearest neighbor in front. (W= [0 – 500])
4	Table 18 – ID: 2	Speed up when self-speed is slow. (W = 10)

Although model 1 has only one steering behavior (Table 22 – ID: 1), it is able to handle both requirements. When approaching the obstacle, agents steer at an angle within 60 degrees from left or right depending on which side gives them the farthest travel distance. By selecting one of the space entities in front of agents, they can move forward to the goal while avoiding the obstacle. Figure 36.a shows an example of this behavior. Agent A eliminated s_2 because it is not in front, and s_3 because the travel distance to the obstacle is not far enough. Hence, agent A chooses s_1 even though there is a nearby agent in front of it (green agent). This behavior is similar to $B_{obstacle\_avoidance}$ in Table 11, however, agents in this model do not steer away from surrounding neighbor agents.

Desired direction
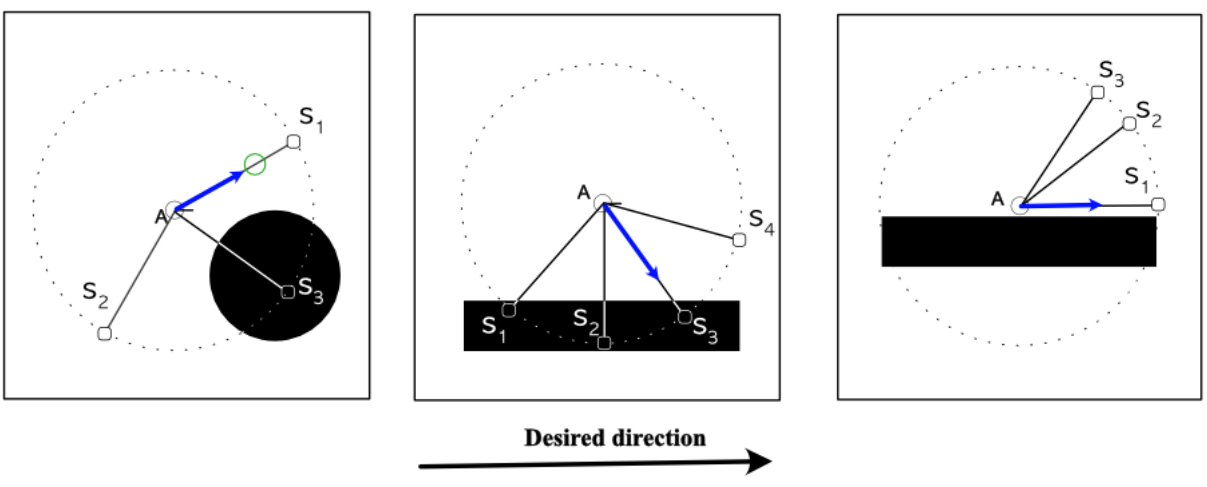
(a) Model 1 - Table 22 – ID:1   (b) Model 2 - Table 22 – ID: 1   (c) Model 2 - Table 22 – ID: 2

Figure 36. Steering decision of a singular agent of Bgoal_reaching + Bpersonal_space

Model 2 (Table 21) on other hand, shows a different strategy where agents first move toward the wall, then follow the wall side regardless of their distances to the obstacle by using two angle behaviors. First, Table 22 – ID:1 behavior removes all space entities on the opposite side of the wall and forces agents to head toward the wall. Figure 36.b shows an example of how the behavior decides where to steer next. Agent A has 4 space entities including: s_1, s_2, and s_3 that are intersected with the wall, and s_4 is not. By using the 1st filter, this behavior removes s1 and s4 because their travel distances to the obstacle (in this case is the rectangle wall). The 2nd filter further eliminates s_2 because its angle distance to the desired direction is not minimum. Hence, agent A steers to s_3 location. This behavior remains dominated (assign weight = [0-50]) the other steering behavior until agents adjacent to the wall. At that moment, the filters of this behavior eliminated all agents' space entities, hence will not contribute anything to the overall decision. The second behavior of this model (Model 2 - Table 22 – ID: 2) even though the weight is always 1, it is good enough for it to take over the first behavior by simply steer to the nearest desired direction, in this case is a horizontal line from left to right (Figure 36.c). As a result, agents move along the wall and escape from the hallway. Both models share the same speed behaviors as Table 18. Figure 35.b.1 and Figure 35.c.1 shows movement different of Model 1 and Model 2 respectively. Based on the heat map of Model 2 (Figure 35.c.2), some agents do not move toward the wall at the beginning because they cannot sense the wall. As a result, the quality of Model 2 depends on the hallway width or agent's FOV distance. Model 1 (Figure 35.b.2), on other hand, has a better robustness capacity because the initial experiment set up do not affect the behavior.

Figure 37 shows an overview of all discovered models from basic steering behaviors, and how these models shape various composite steering behavior by creating different combinations between them. As the legend in the Figure 37 indicates, dash boxes present basic steering behavior scenarios, and solid boxes present composite steering behavior scenarios. The arrow $\longrightarrow$ indicates that model specifications in the composite scenarios are very similar to the one in basic scenarios. On the other hand, the arrow $\longrightarrow\!\!\!\!\!\!\!\longrightarrow$ shows that there are major changes (most of the cases are in Activation component) between the transitions. If the arrows point to a whole scenario box instead of each model, it means all models in the box use the same basic model. For example, all models of the leader-following + mobile obstacle avoidance scenario use the similar $B_{obstacle\_avoidance}$ in Table 11.
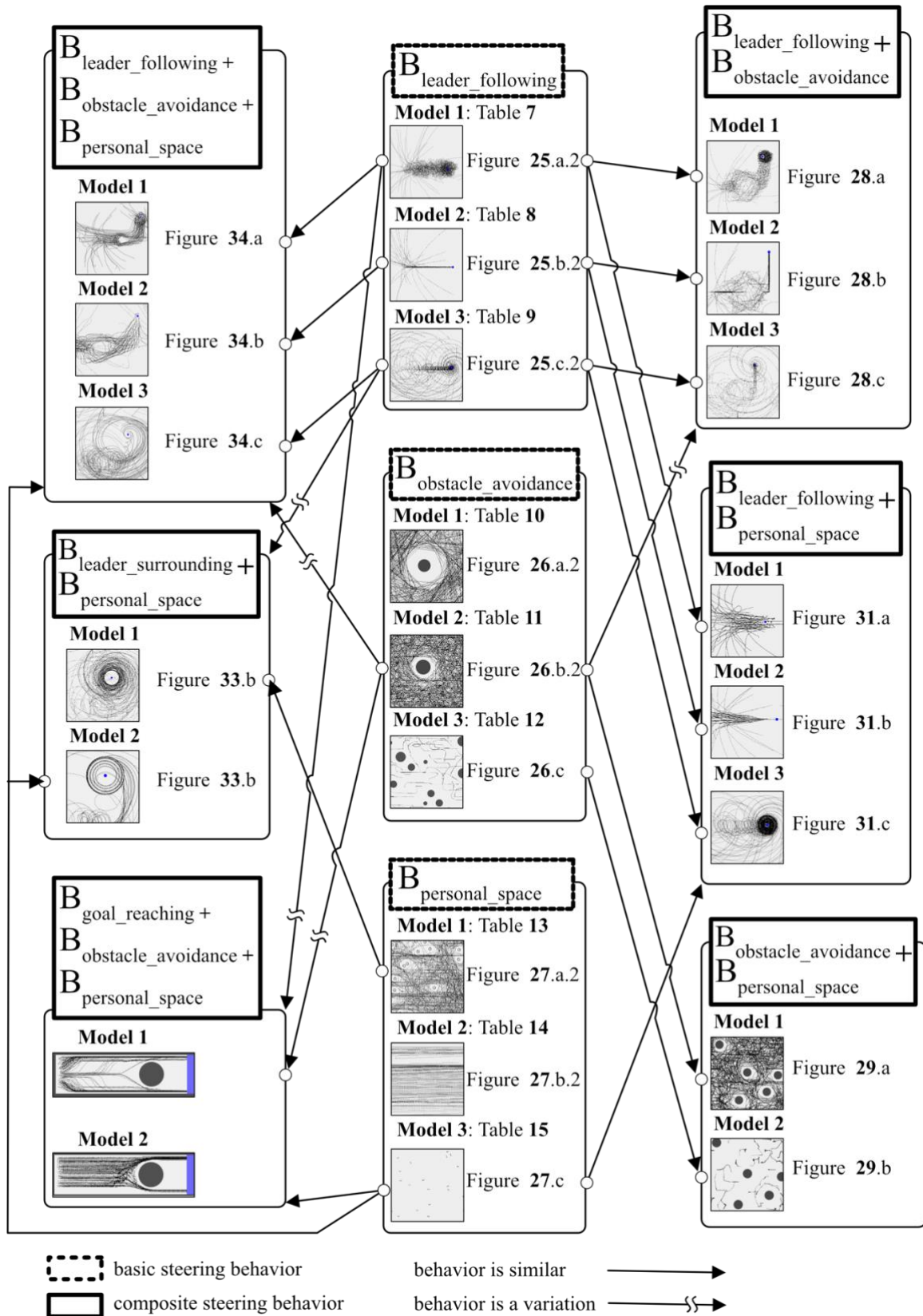
Figure 37. Combinations of composite steering behaviors from basic steering behaviors.

6. MODEL EVALUATION

We have two different evaluations to assess the framework. One evaluation measures the effectiveness of the GA multiple stage search. Another one evaluates the quality of the discovered models. We put two models: (1) the leader surrounding + personal space maintenance scenario and (2) the leader-following + mobile obstacle avoidance + personal space maintenance scenario to the test because these two scenarios are the most complex models in our experiments, hence very challenging for GA to discover good models.

6.1. Evaluation of Model Fitness Score Between One Stage and Two Stage Search.

For each scenario, we choose the discovered models with the best scores to evaluate and compare the effectiveness between one stage and two stages GA search in two aspects. First, the best models in stage 1 and stage 2 fitness scores across 40 generations are compared. Second, the best 50 models in both stages are counted and the fitness score distributions of them are compared.

6.1.1. The Leader Surrounding + Personal Space Maintenance Scenario

Figure 38.a, b, and c show the best discovered model's fitness scores of 40 generations for stage 1 and stage 2 searching respectively. The fitness scores for this scenario shows the battles between personal space and the leader-surrounding metrics score. In Figure 38.a, personal space behaviors dominate the first half of the search until 22rd generation. After that, the leader-surrounding behavior begins to increase while the personal space decreases, hence the overall fitness remains low and reaches the max at 35th generation (overall score = 0.5263). In stage 2, FFigure 38.b shows that even though the leader-following still remains lower than the personal space score, it has a higher score than stage 1 even at the 1st generation. As a result, the overall fitness score is able to escape the local optimum and reach much higher overall fitness score = 0.8041.
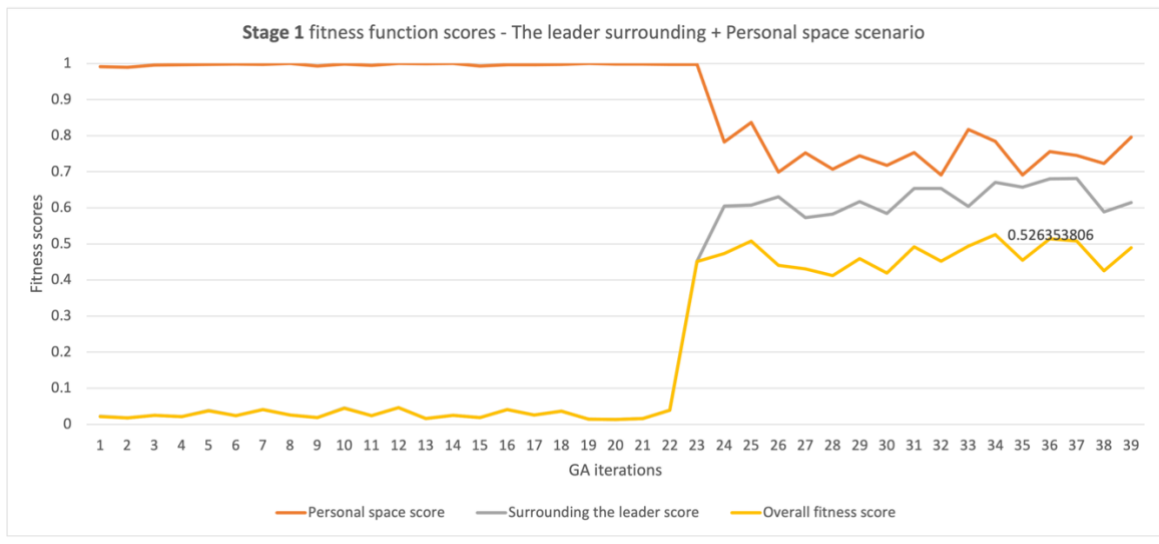
(a) Stage 1
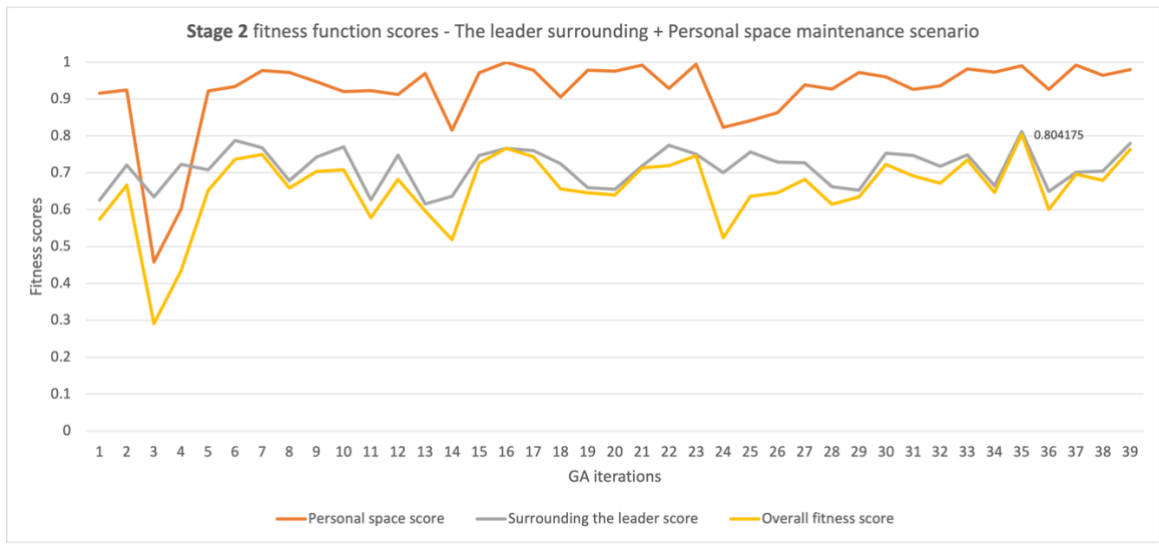
(b) Stage 2

Figure 38. Fitness scores of one and two stage search for the Leader-Surrounding + Personal space maintenance scenario.

Figure 39.a and b show 50 best model distributions for stage 1 and stage 2 respectively. In stage 1, among 50 models, 24 models have overall fitness scores that are less than or equal 0.2, 22 are between [0.2-0.4], and 5 models are above 0.4. For one stage search, 90% of the discovered models have the score below 0.4, compared to two stage search where only 34% of the models are below 0.4. Similar to the fitness score evaluation, this assessment confirms the significant improvement between one two stage search.

(a) Stage 1
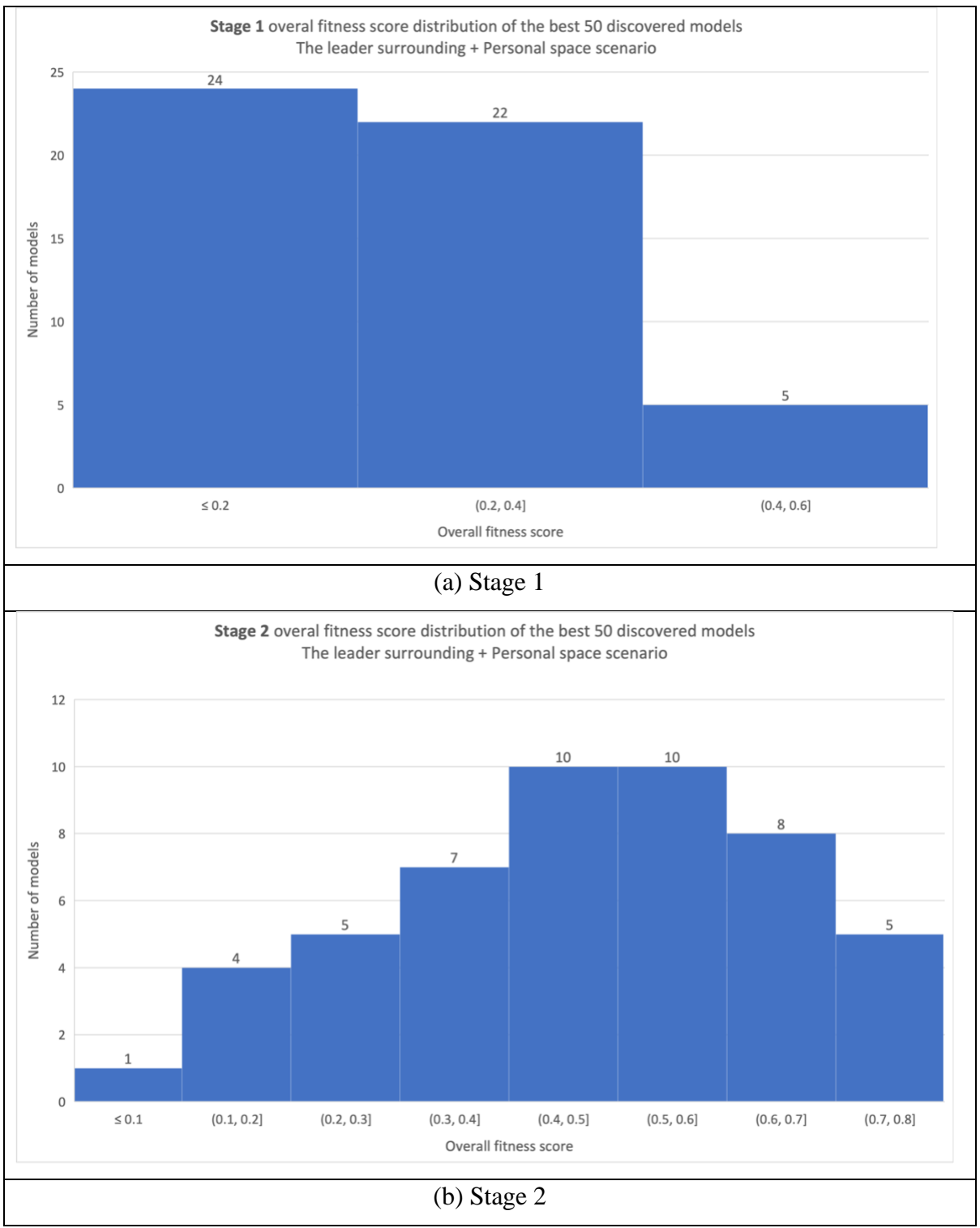
(b) Stage 2

Figure 39. Overall fitness score distribution of one and two stages search for the Leader-Surrounding + Personal space maintenance scenario.

6.1.2. The Leader Following + Mobile Obstacle Avoidance + Personal Space Maintenance Scenario

Figure 40 shows the fitness scores of the best discovered model at each stage of this scenario. The challenging factor of this scenario is that its requirements conflict with each other. If agents have higher priority to follow a leader, they will be more likely to collide with obstacles and other neighboring agents, and vice versa. As a result, activation component in this scenario plays an important role to assist all the steering behavior working together. One stage search shows that GA is struggling to find the best weight combinations. Two stage search once again is able to escape the local optimum, and achieve a better score of 0.507 vs 0.477 of one stage search.
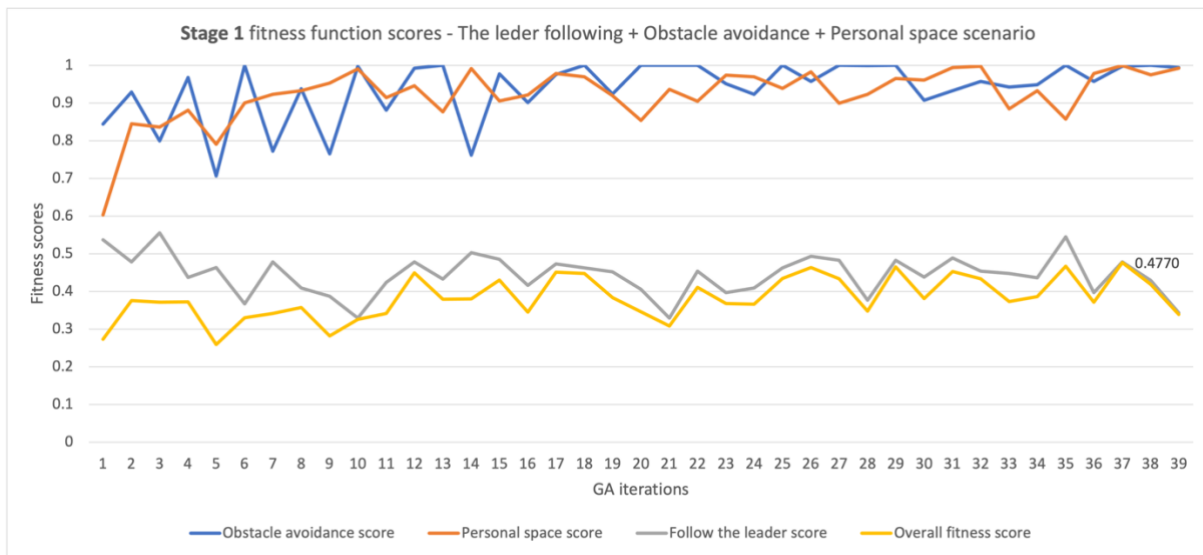
(a) Stage 1
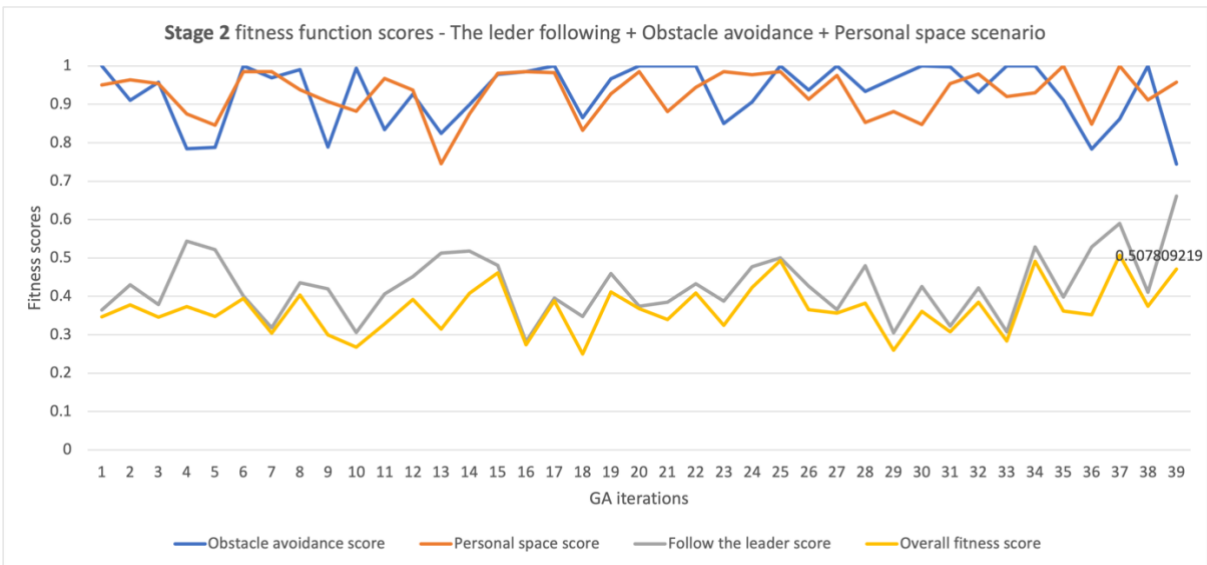
(b) Stage 2

Figure 40. Fitness scores of one and two stage search for the Leader-Following + Mobile obstacle avoidance + Personal space maintenance scenario.

Figure 41 shows the top 50 models' distribution of stage 1 and stage 2. There are significant improvements from stage 1 to stage 2 where 95% of discovered models in stage 1 have the score below 0.3. At stage 2, the score distribution shifts toward higher grade where more than 40% of the models have scored higher than stage 1.
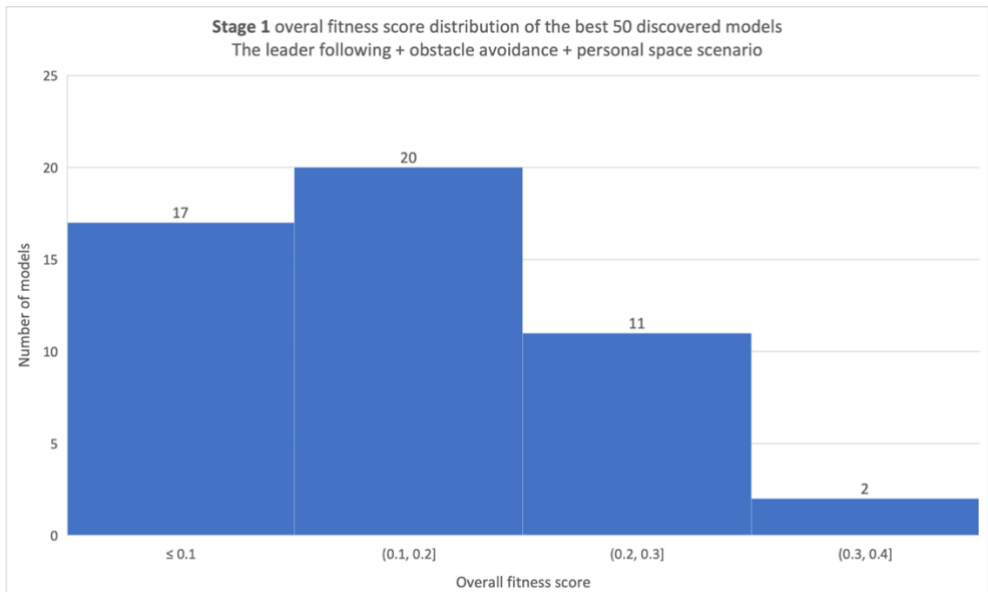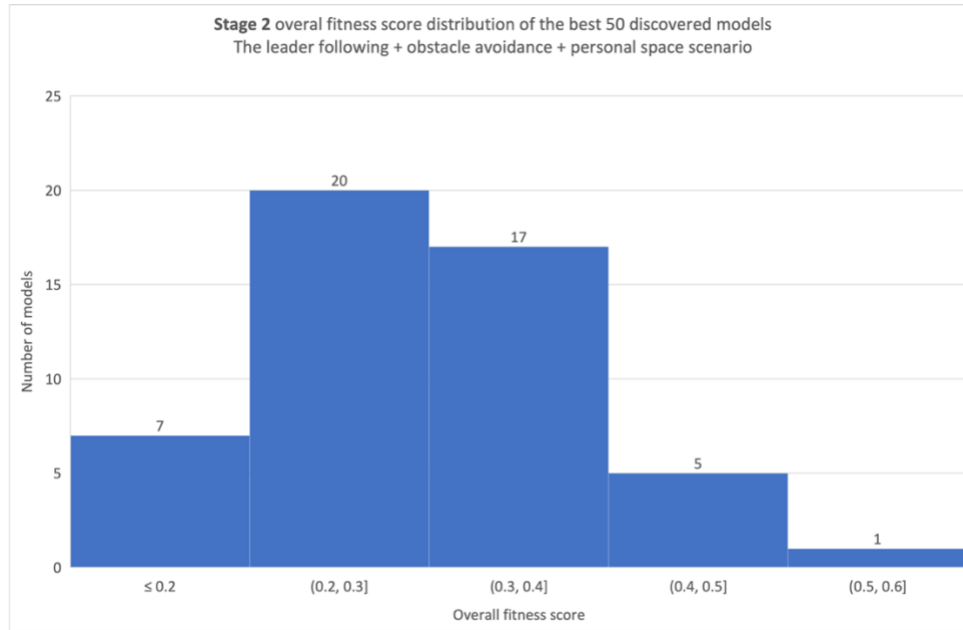

(a) Stage 1

(b) Stage 2

Figure 41. Overall fitness score distribution of one and two stages search for the Leader-Following + Mobile obstacle avoidance + Personal space maintenance scenario.

6.2. Comprehensive Model Evaluation

Our previous work provides a set of categories to formally evaluate quality of discovered models including Flexibility, Comprehensibility, Composability, Controllability and Robustness [6]. Among them, Flexibility, Comprehensibility and Composability are covered in above section:

• **Flexibility**: ability to discovered different candidate models (demonstrated in section 5)

• **Comprehensibility**: the results can be understood and explained (demonstrated in tables that shows model specification in section 5)

• **Composability**: each component of the discovered behavior can be combined, exchanged, and reused (demonstrated by using GA's operators and multiple stage search)

As a result, this section mainly focuses on evaluating Controllability and Robustness of discovered models.

• **Controllability**: ability to modify models' parameters and predict the new behavior patterns.

• **Robustness**: testing the limits of the model by modifying various initial conditions such as: increasing the number of agents or leaders, changing max speed of the leader, etc..

6.2.2. *Evaluation of Two Discovered Models of Leader Surrounding + Personal Space Maintenance Scenario*

• **Controllability**: We choose model 1 (Table 19) for this evaluation because it has the best overall fitness scores. To evaluate Controllability of this scenario, we manually change the Activation criteria of the circle around behavior (Table 19 – ID: 1) to three different ranges: [0-40], [0-60], and [0-100], and Figure 42. a, b. and c show how surrounding radius is changed for these three ranges respectively. For the criteria range = [0-40], agents form a smaller circle because they keep their priority to surround the leader until they are within the distance to 40 compared to 60 and 100 of the other two ranges.
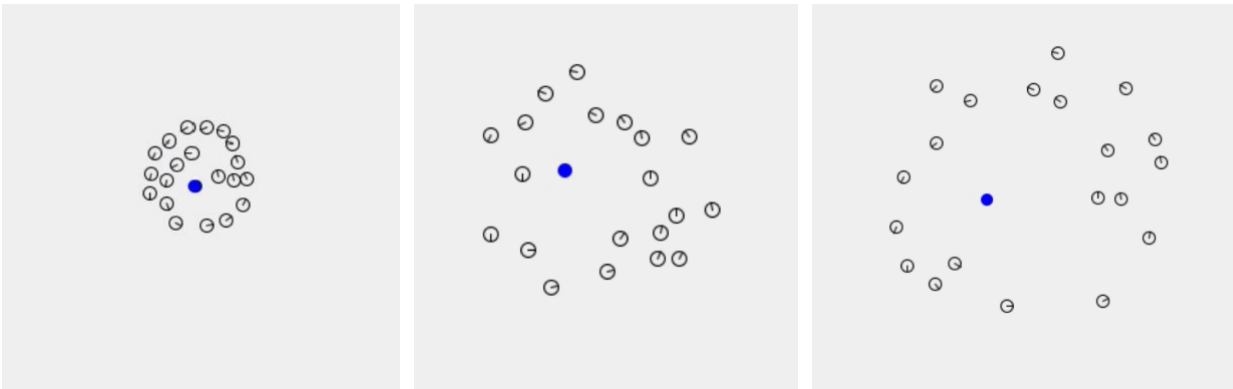
| a. Surround leader with | b. Surround leader with | c. Surround leader with |
| [0-40] range | [0-60] range | [0-100] range |

• *Figure 42. Agents' behaviors change predictively after manually modifying the model specification of Table 19 – ID: 1 $B_{leader\_surrounding} + B_{personal\_space}$ model*

• **Robustness**:

• **Test 1 (Increase number of agents from 20 to 200) :** Figure 43 illustrates how 180 agents surround the leader using model 1 (Table 19) by showing snapshots of three different timesteps. Figure 43.c shows that agents are heavily distributed on the right side of the leader because the leader is moving from left to right.
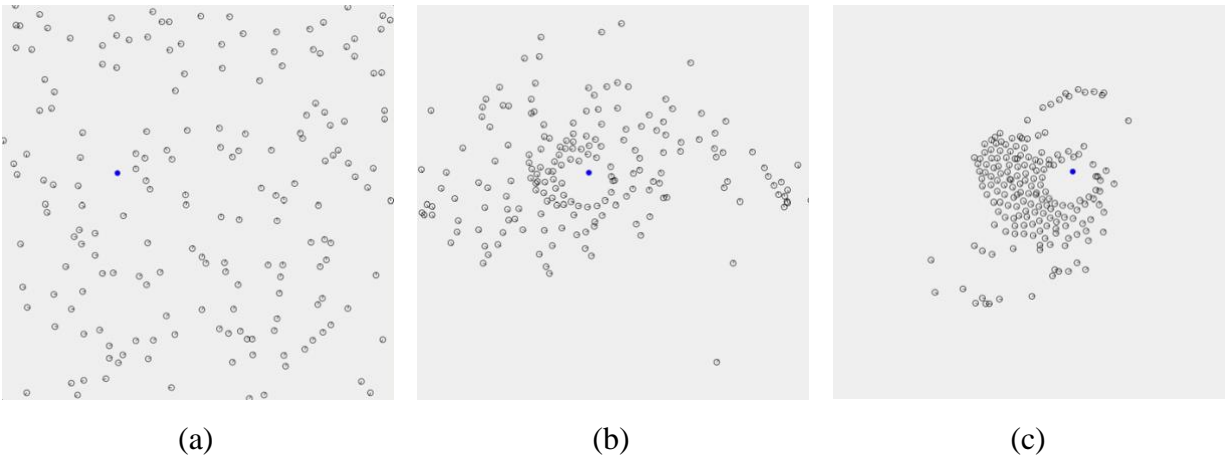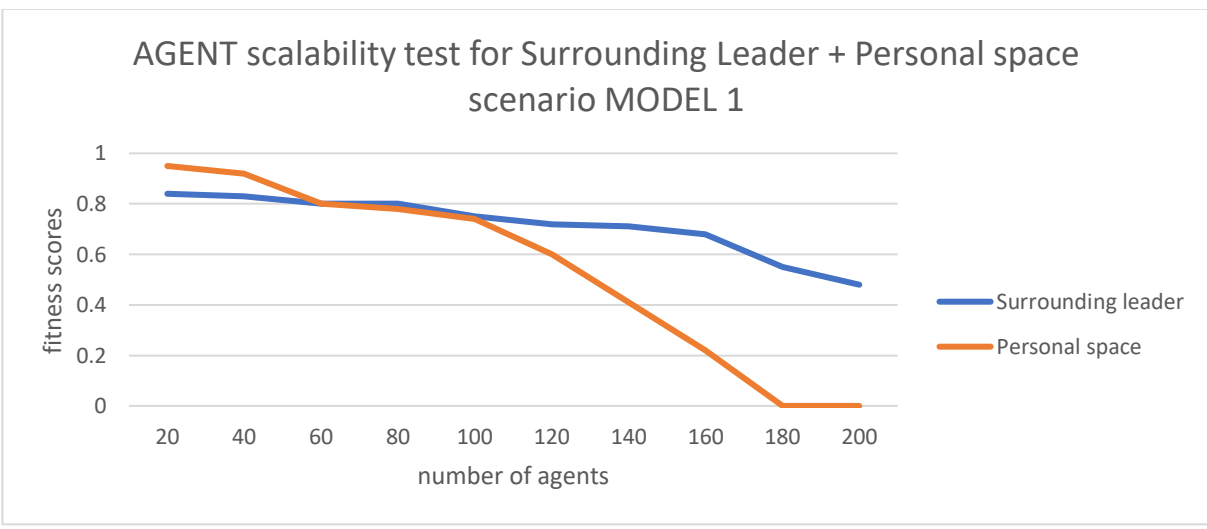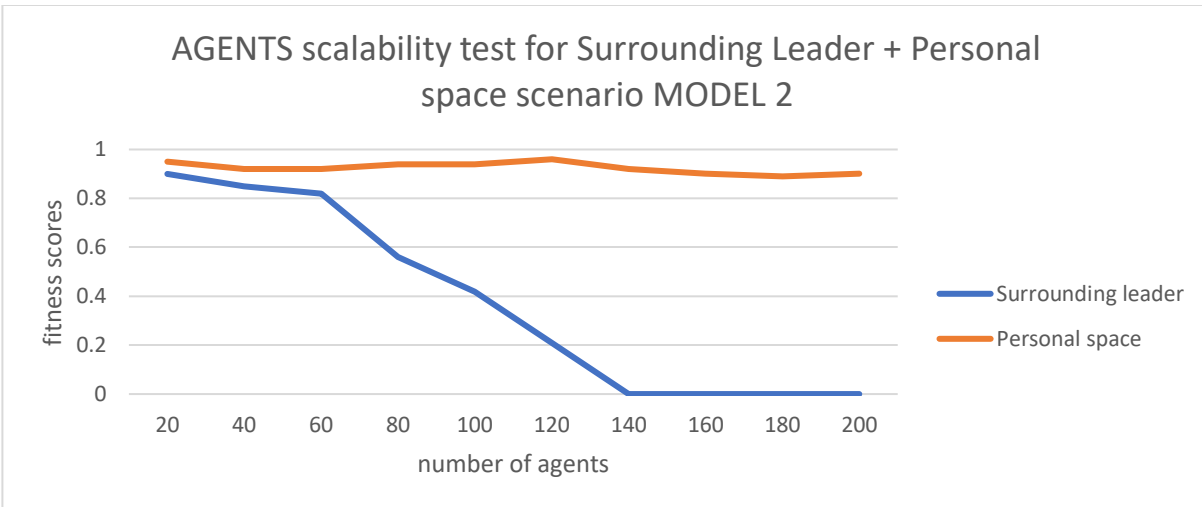

(a)   (b)   (c)

Figure 43. Snapshots at three different timesteps of the Leader-surrounding + Personal space maintenance scenario with 180 agents.

(a)

AGENTS scalability test for Surrounding Leader + Personal space scenario MODEL 2

(b)

Figure 44. Fitness scores for the agent's population from [20-200] for 2 discovered models of the Leader-Surrounding + Personal space scenario.

Figure 44 shows how fitness metrics change for **the** number of agents in simulation from [20-200]. The fitness scores for model 1 and model 2 are opposite from each other. Model 1 has higher score**s** for **the s**urrounding leader fitness metric, while model 2 has higher score**s** for personal space fitness metric. As snapshots and heat map in

Figure 32 and Figure 33 show, model 1 surrounding the leader in **a** much smaller radius, hence the probability to violate other agents' personal space is higher. **Consequently**, model 1's personal space score is lower. Model 2 **on the** other hand, surround**s** the leader in **a** more scatter pattern, hence its fitness score is opposite compared to model 1. Overall, model 1 has much better scalability capacity than model 2. Model 2 surrounding leader score starts declining when there are only around 60 agents, while model 1 can still handle 200 agents fairly well. It is predictable because of two reasons: First, model 2 does not surround the leader in a concentrate area. Second, it needs much more space compared to model 1.

• **Test 2 (Increase the leader's speed from 0.25 to 1.5) :** Figure 45 shows fitness score of both models when testing the scenario with 40 agents and different leader's speed from [0.25 – 1.50].

Model 1 breaks when the leader's speed is above 1.5. Because there are always two steering forces to keep agents surrounding and following the leader, hence when the leader moves too fast, agents do not have enough time to adjust, and the leader just crosses through the circle formation. Model 2 shows an interesting pattern for the leader surrounding's score. Even though it has lower scores than model 1's in most of the tests, it does not break suddenly like model 1 shows.
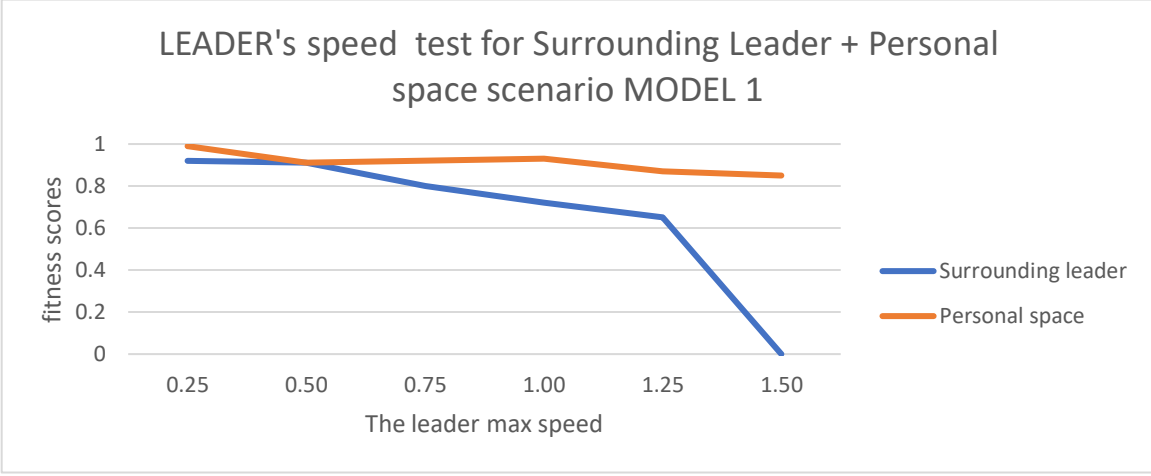

(a)
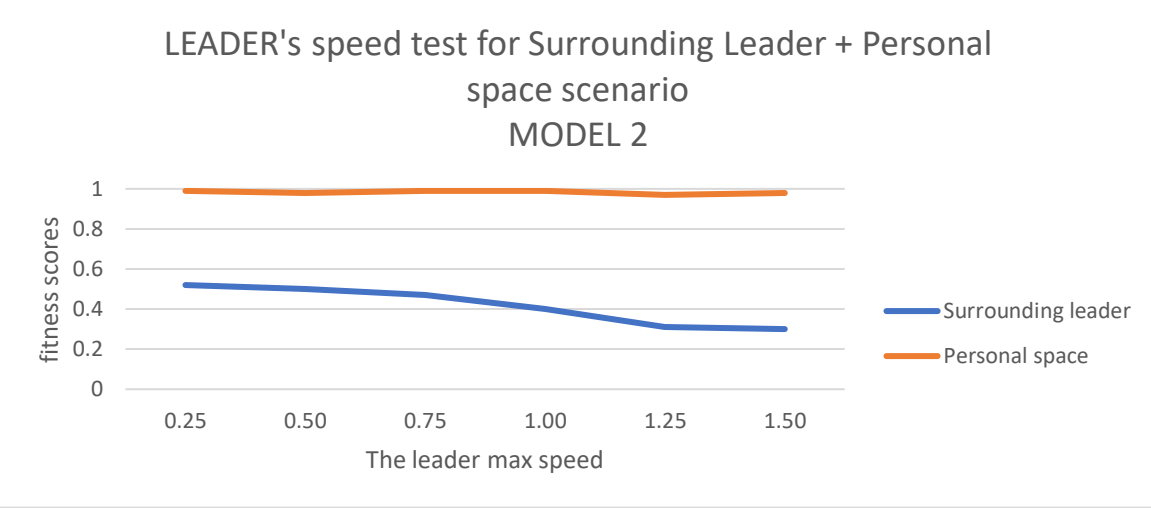
(b)

Figure 45. Fitness scores for the leader's speed from [0.25-1.50] for 2 discovered models of the Leader-Surrounding + Personal space scenario.

There are two reasons for it. First, this evaluation is tested with only 40 agents, hence model 2 has space to form a large area around the leader. Second, because the forming cluster is large and scatter, when the leader's speed increases, it is not easier for the leader to escape the center point of the cluster completely, hence the surrounding leader score only slightly decreases.

6.2.3. *Evaluation of Three Discovered Models of the Leader Following + Mobile Obstacle Avoidance + Personal Space Maintenance Scenario*

- **Controllability:** We choose Model 1 of the leader following + mobile obstacle avoidance + personal space maintenance scenario for this evaluation because it has the best overall fitness score. Figure 46.a shows the original agents' movement pattern of Model 1 (*Figure* 34.b) at the moment the leader intersects the obstacle. By manually changing different parameters, we can control the sensitivity of the behavior as we want; thus, changing movement patterns. For example, in Figure 46.b, we increase the distance where agents begin to slow down when they detect neighbors in front of them, hence making personal space become wider. Figure 46.c shows an opposite effect with obstacle avoidance reaction distance where agents wait until the obstacle is very close to steer away from it. Different from Figure 46.b & c, in Figure 46.d, we reduce the desire to follow the leader significantly by changing the priority of the behavior. As a result, when there are obstacles nearby, agents do not want to steer back to the leader right away and wait until the obstacle is far enough.
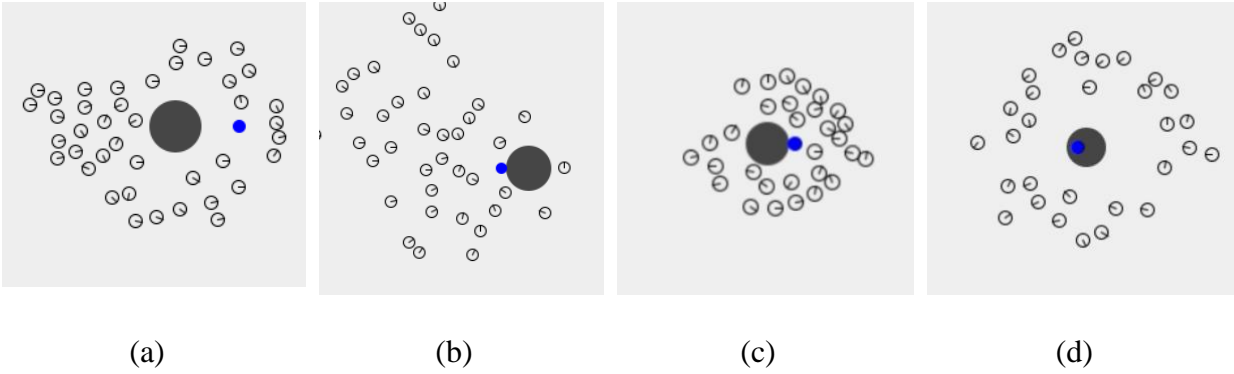
<center>(a)   (b)   (c)   (d)</center>

Figure 46. Controllability tests by changing model parameters.

(a) Original discovered model, (b) Adjust $B_{personal\_space}$ distance, (c) Adjust $B_{obstalce\_avoidance}$ distance, (d) Adjust the priority between $B_{obstalce\_avoidance}$ and $B_{leader\_following}$

- **Robustness**:

We test the limitation of the candidates by setting up different initial conditions for the scenarios. Three scalability tests are used including: (1) increase the number of agents and the space size is preserved (agent's density is increased). (2) increase the number of agents and the space size is increased (agent's density is preserved). (3) increase the number of leaders. The result shows that by taking advantage of information gathered from space entities in the framework, Model 1 (Figure 34.a) performs better than model 2 (Figure 34.b) and model 3 (Figure 34.c) in scalability and robustness tests.

(1) Increase agent's density: We test the model with the number of agents increasing from 100 to 600 and keep the world size as 700x700 pixels to test how agents move in limited space. Figure 48.a shows a timestep with 400 agents and 15 mobile obstacles (Model 1 *Figure* 34.b). Figure 48 illustrates that all candidates have similar performance for $B_{personal\_space}$ and $B_{obstalce\_avoidance}$. Because model 1 uses the space entity component to follow the leaders, it gives a better fitness score compared to the other two models. Model 3 gives the worst score for $B_{leader\_following}$ as

predicted because it takes the longest time to move toward the leader. Since the larger the agent's clusters are, the more challenging for them to maneuver around limited space and fulfill all requirements, the overall scores for all models gradually decrease after the number of agents reaches 400.

(a)　　　　　　　　　　　(b)　　　　　　　　　　　(c)

Figure 47. Snapshots at three different timesteps of the Leader following + Obstacle avoidance + Personal space maintenance scenario with 400 agents (world size: 500 × 500)
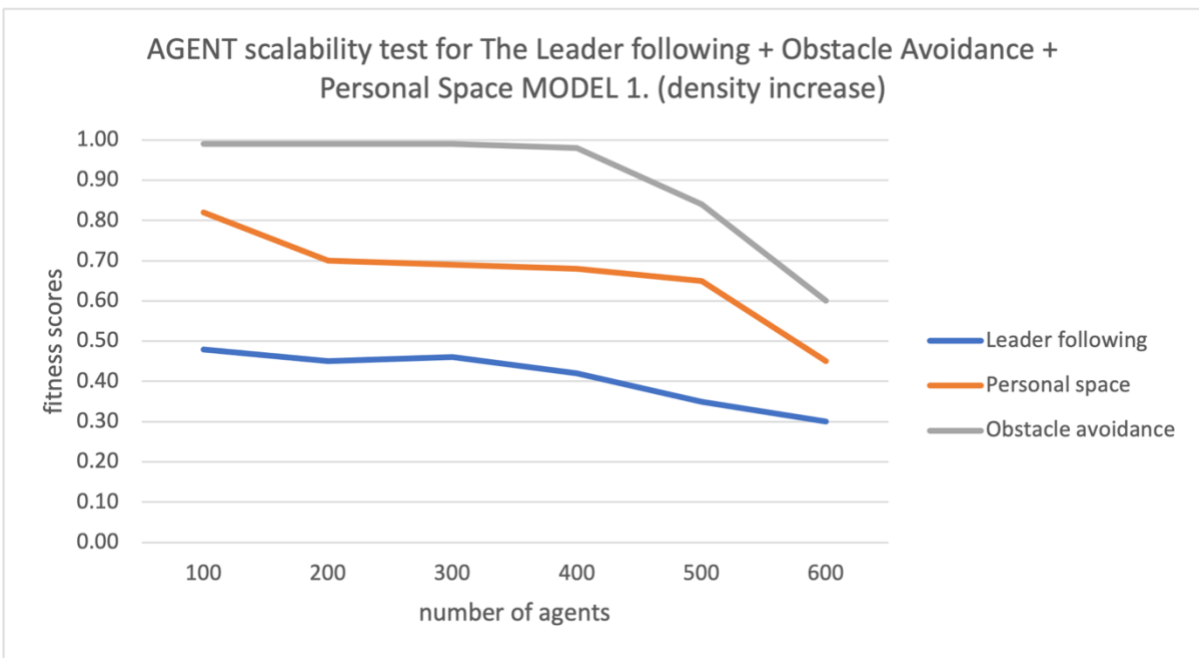
(a) Model 1
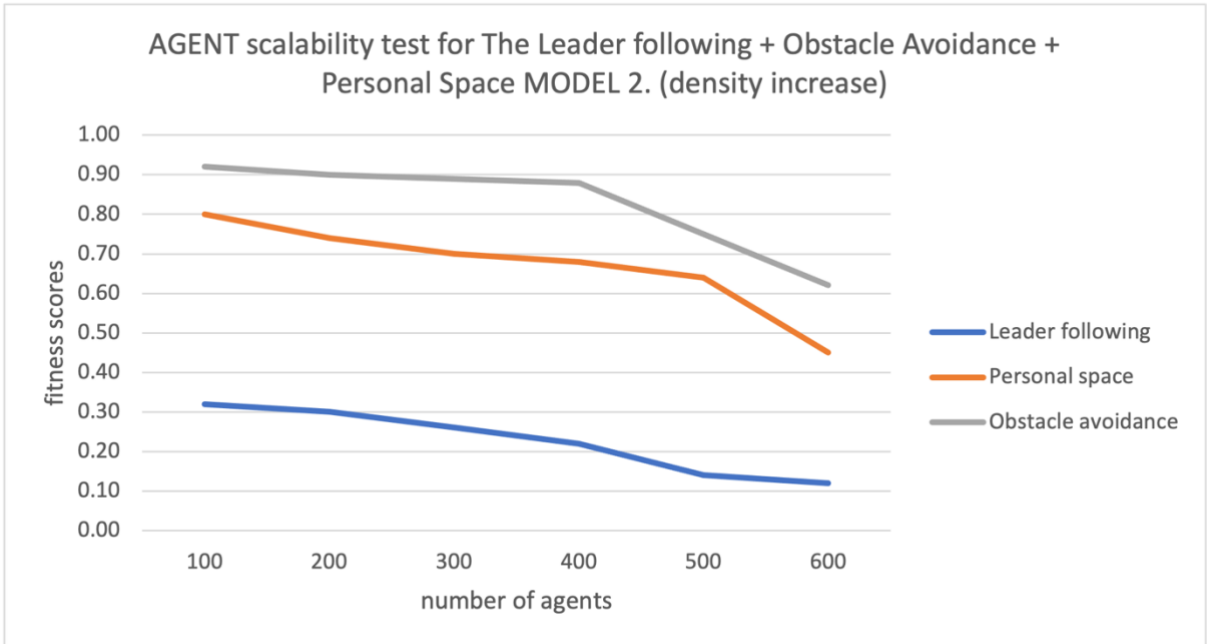
(b) Model 2
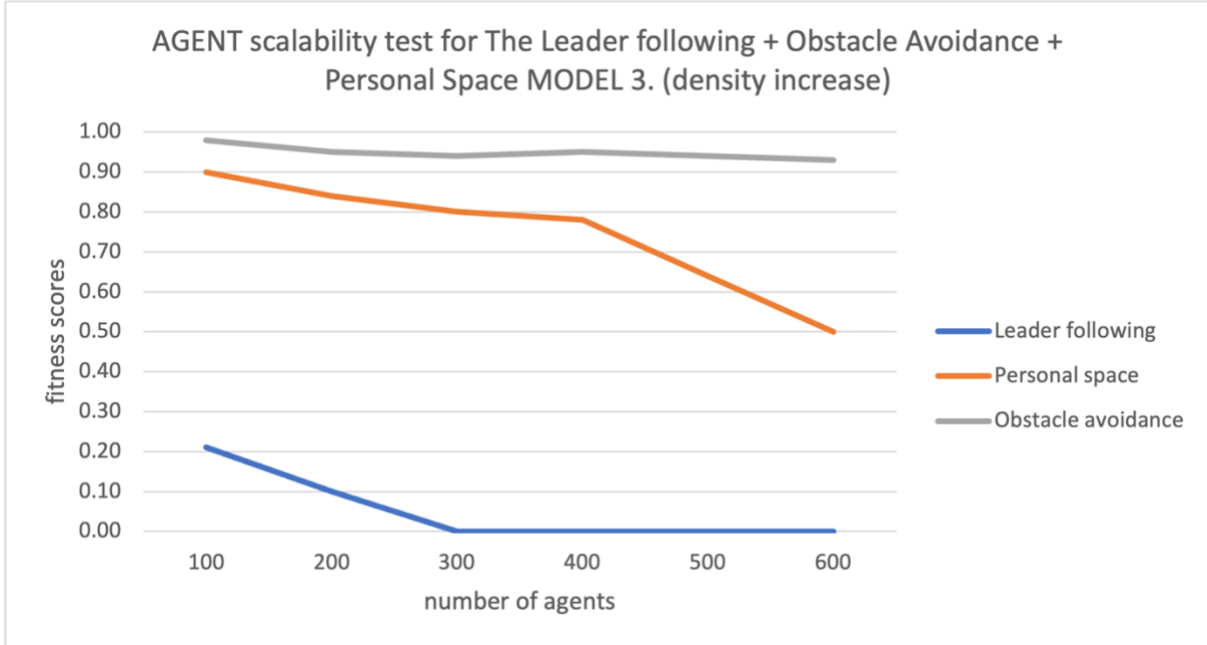
(c) Model 3

Figure 48. A test for $B_{leader\_following} + B_{personal\_space} + B_{obstalce\_avoidance}$ scenario – Agent density is

increased.

(2) Agent's density is preserved: In this scalability test, we increase the number of agents from 100 to 400, and also increase the size of the world from 500x500 to 1000x1000 respectively with the goal to keep the density the same across different tests. Figure 50.a illustrates the screenshot with 400 agents and 15 mobile obstacles of Model 1 in 1000 x1000 pixels simulation space. As the chart in Figure 50, all models maintain the fitness score very well even with 400 agents, and Model 1 (*Figure* 34.a) still has a best $B_{leader\_following}$ score. The reason for the $B_{leader\_following}$ fitness score gradually decreasing is because the larger the world size is, the more time it takes for agents to form the cluster around the leader. Since all the tests share the same simulation time (600 timesteps) with the first 1/3 as burn-in time for agents to move closer to the leader. For a larger world, this 200 burn-in time step is not long enough for all agents to gather around the leader, hence the score is decreased across all models.

(a)   (b)   (c)
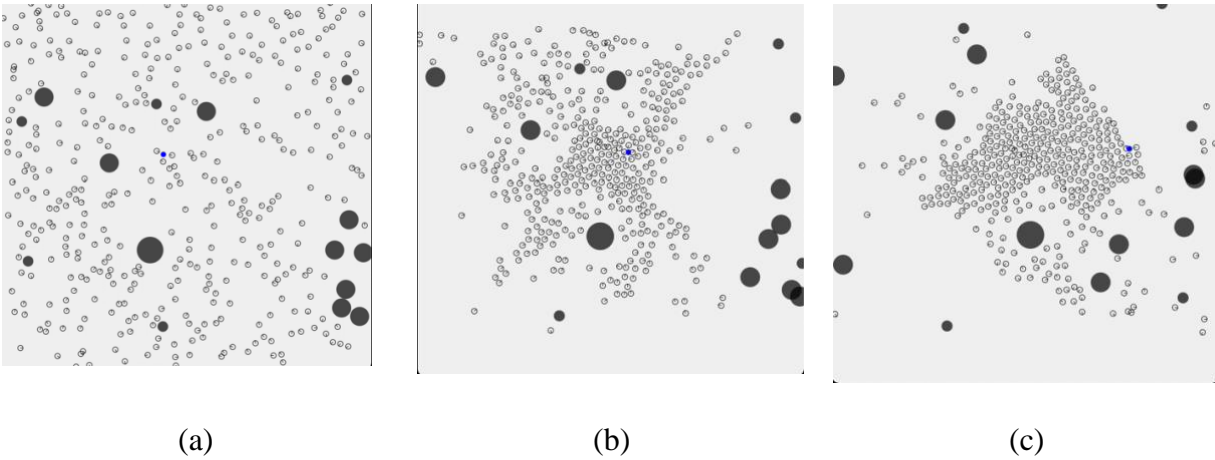
Figure 49. Snapshots at three different timesteps of the Leader following + Obstacle avoidance + Personal space maintenance scenario with 400 agents (world size 1000 ×1000)
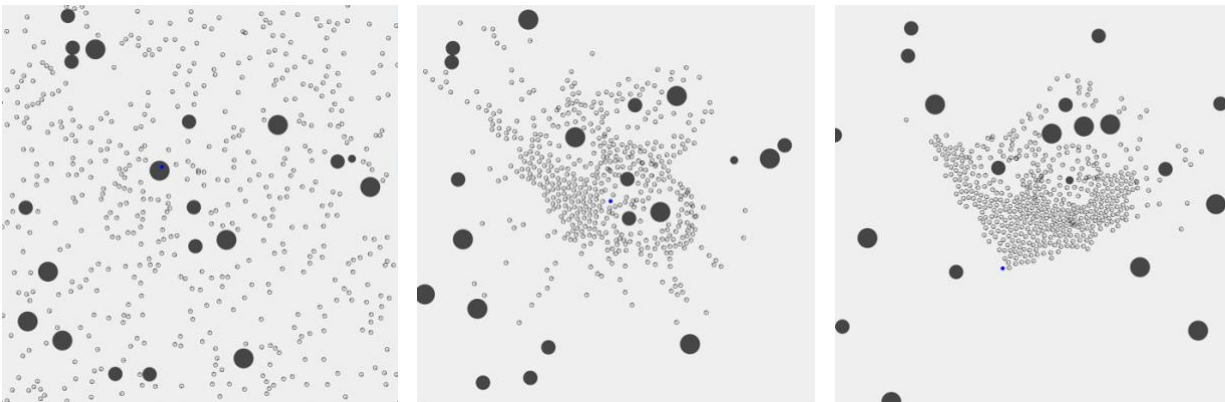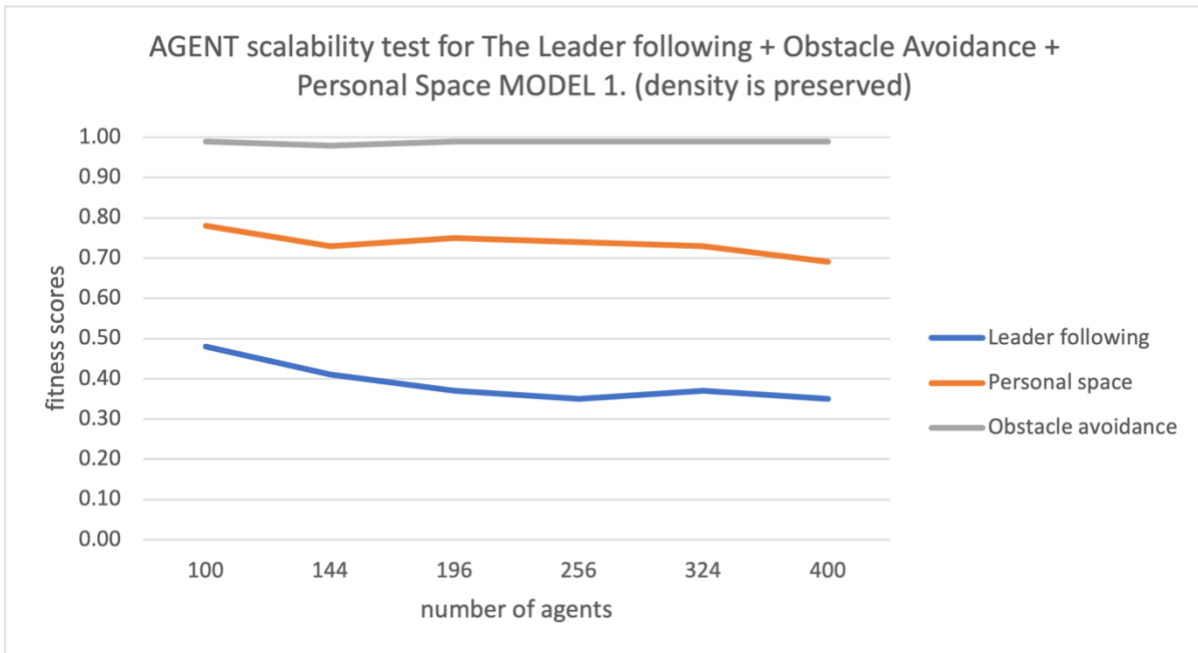
(a) Model 1
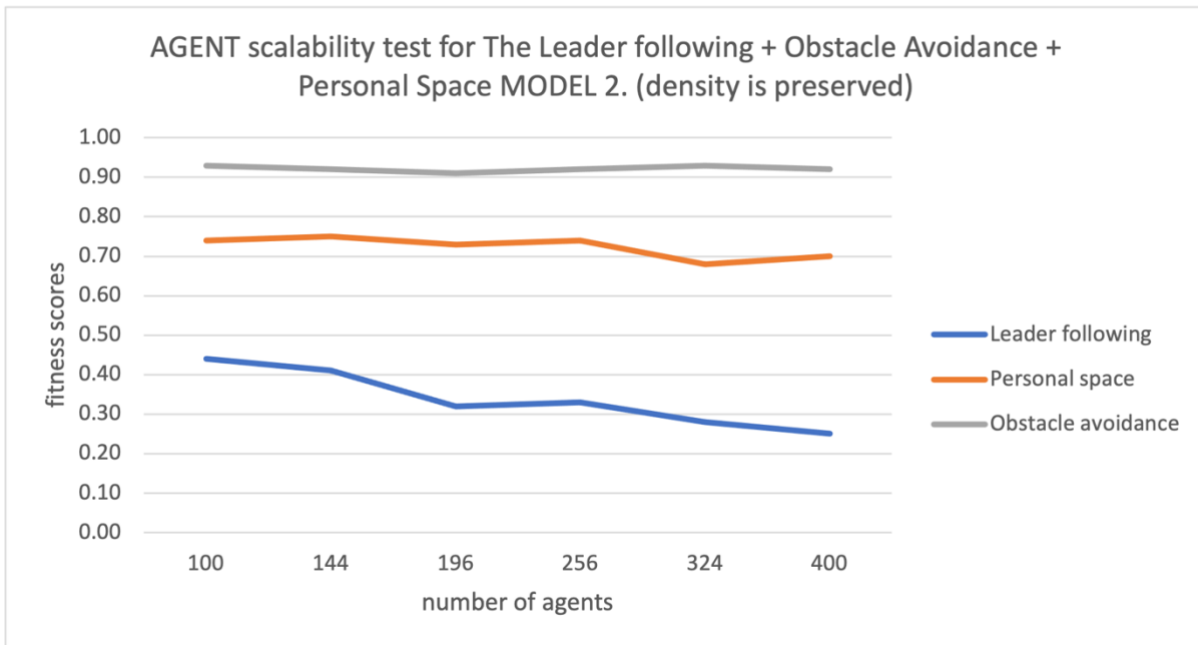
(b) Model 2
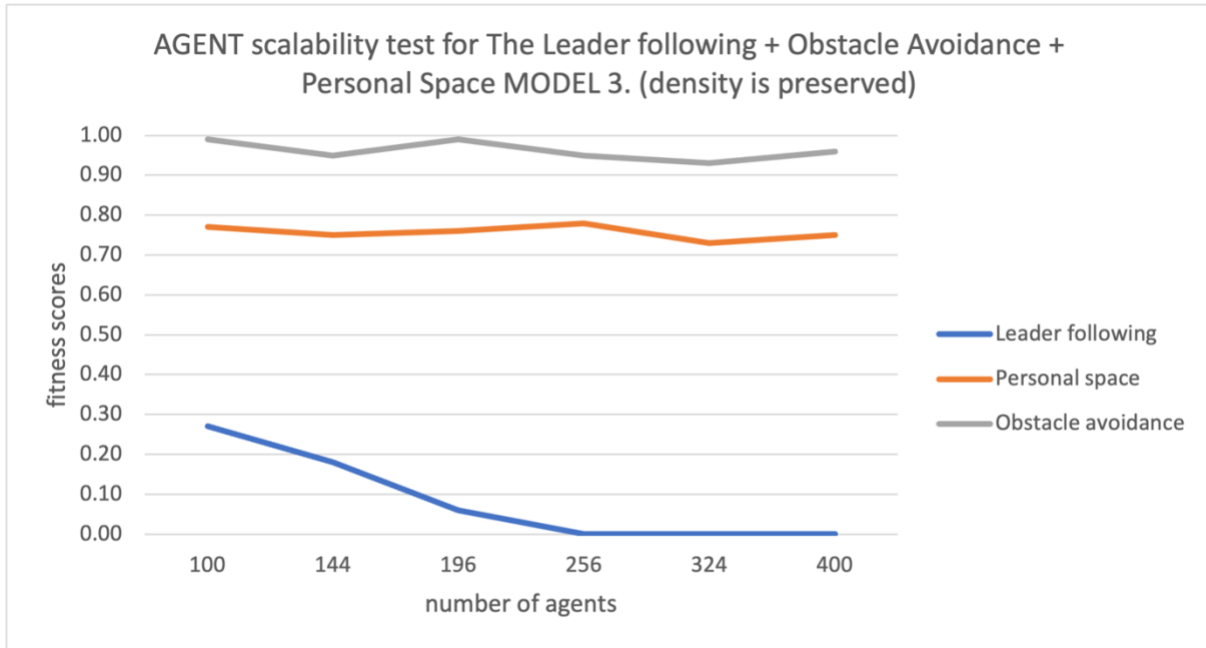
(c) Model 3

Figure 50. A test for $B_{leader\_following}$ + $B_{personal\_space}$ + $B_{obstalce\_avoidance}$ scenario - agent density is

preserved.

(3) Increase number of leaders: In this test, we increase the number of leaders from 1 to 10.

Figure 51 illustrates three different timestep of Model 1 (*Figure* 34.a) with 800 agents, 15 mobile

obstacles and world size of 800x800. With 8 leaders, the $B_{leader\_following}$ makes agents follow their

nearest leader. The chart in Figure 52 shows that the leader following score increases along with

the number of leaders. Because agents no longer need to form one big cluster, many small clusters

are formed around leaders, hence making the crowd distribution much better, and the models are

able to improve the fitness scores. This test once again shows the advantage of using space entity

component because the $B_{leader\_following}$ fitness score of Model 1 (Figure 52.a) is still the best

compared to the other two models (Figure 52.a & b)

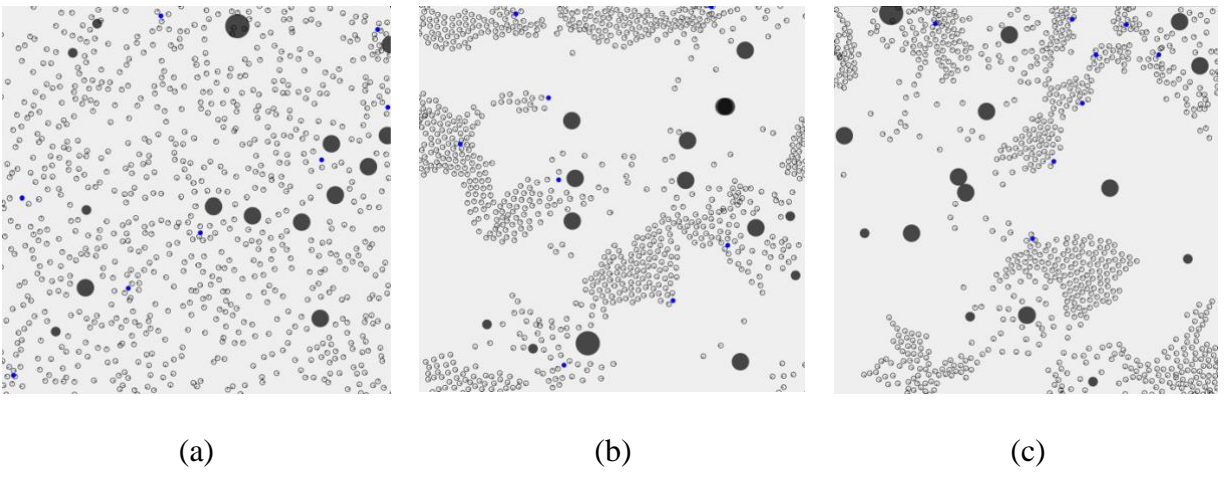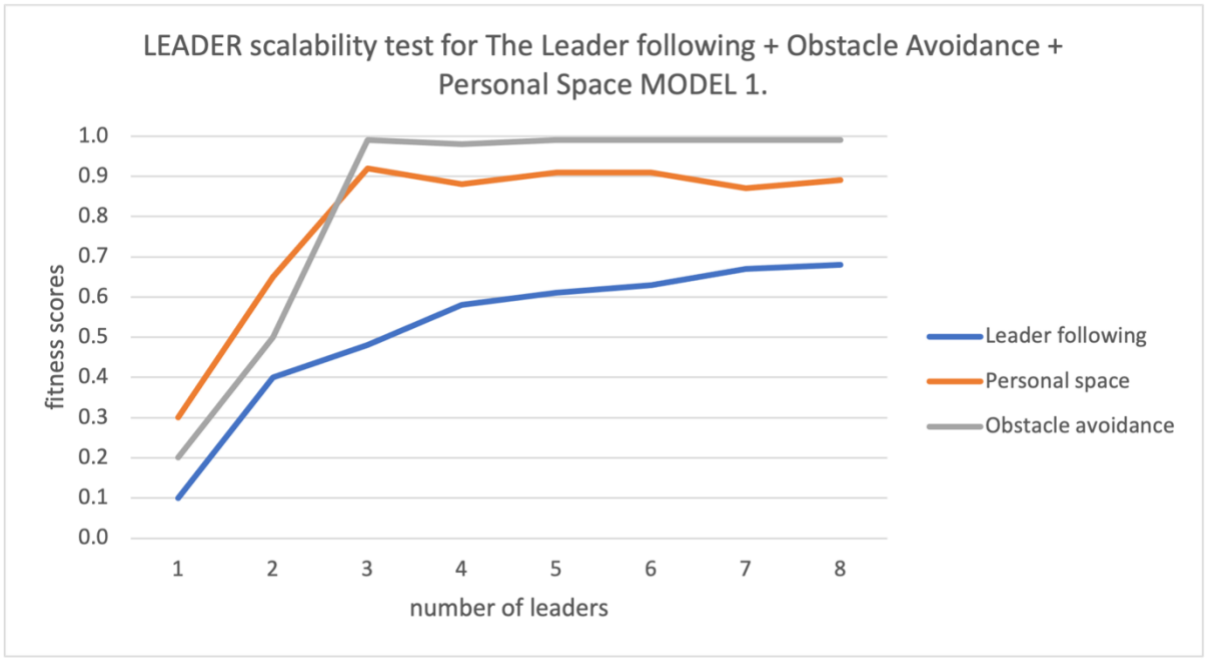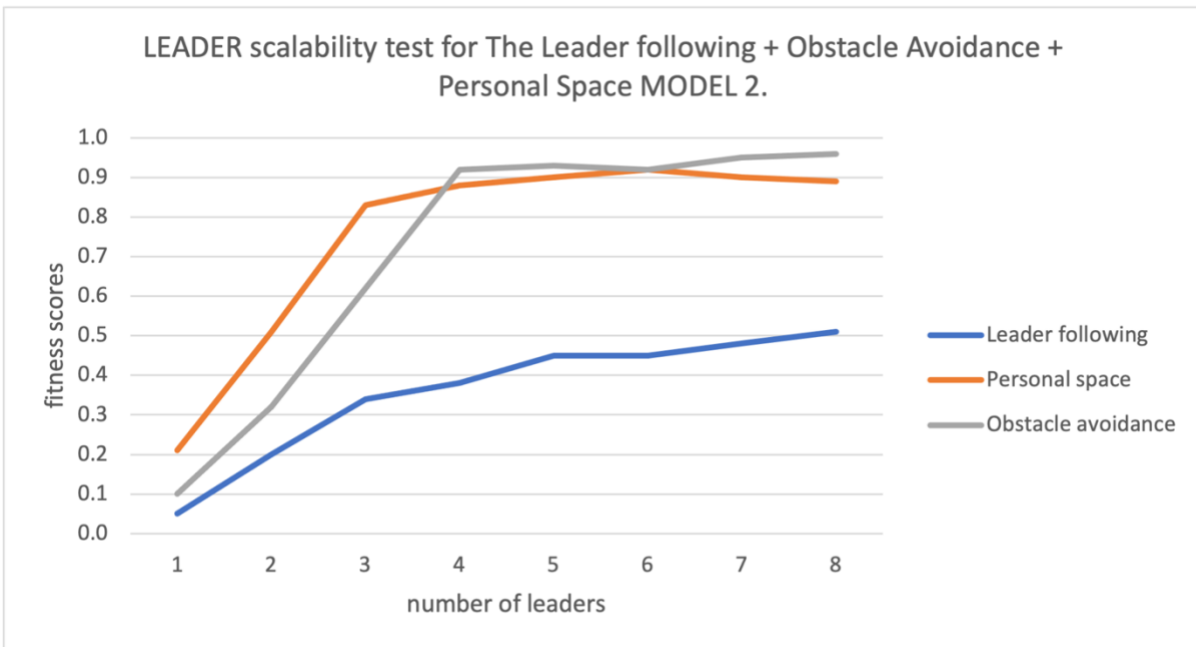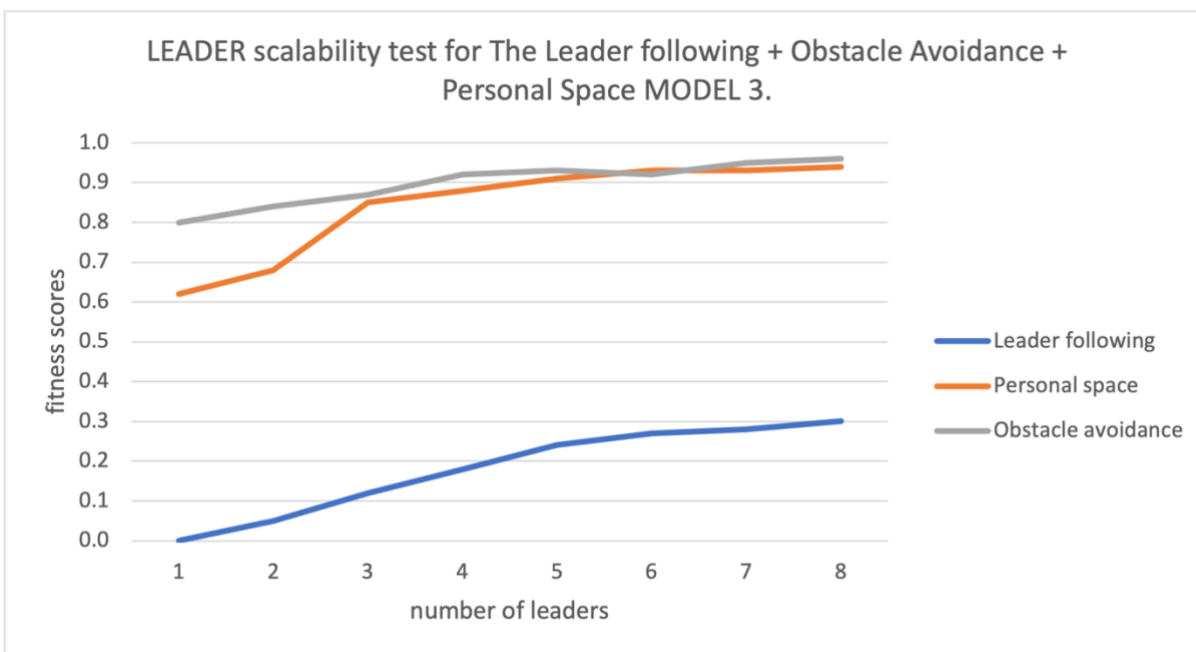*Figure 51. Snapshots at three different timesteps of the Leader following + Obstacle avoidance + Personal space maintenance scenario with 8 leaders (world size 800 ×800)*

(a) Model 1

(b) Model 2

(c) Model 3

Figure 52. A test for $B_{leader\_following} + B_{personal\_space} + B_{obstalce\_avoidance}$ scenario - number of leaders

is increased.

7. CONCLUSION

This work presents an approach that searches for candidate models of steering behavior in an automated way. Several major extensions are added to assist developing more complex scenarios that include multiple steering behavior. First, Activation component decides which agents' behaviors have the highest priority depending on the nearby neighbors and environment. Thus, the framework assists better for complex models where agents need to compute overall movement action from multiple behaviors. Second, a multiple search stage method shows to assist GA distribute computational resources better. Finally, by applying the adaptive space entity generation method, the evaluations for each of the space entities are meaningful, and the computational cost is kept low. Different distance property information provides a wide range of options for agents to filter, select, and act. With these contributions, multiple candidates for $B_{leader\_following}$, $B_{personal\_space}$, and $B_{obstalce\_avoidance}$ basic steering behaviors are discovered. Not only that, but we also further test the framework for more complex scenarios by combining these basic steering behaviors to create various composite steering behavior scenario including: $B_{leader\_following}$ + $B_{obstalce\_avoidance}$, $B_{obstalce\_avoidance}$ + $B_{personal\_space}$, $B_{leader\_following}$ + $B_{personal\_space}$, $B_{leader\_surround}$ + $B_{personal\_space}$, $B_{leader\_following}$ + $B_{obstalce\_avoidance}$ + $B_{personal\_space}$, Hall-Way evacuation with an obstacle in the middle. The search efficient evaluation shows that two stage search is able to escape the local optimal, and discovered candidates have better overall scores compared to one stage search. The controllability and robustness tests show the better performance of models that use activation component and space entities effectively, hence prove that these components are crucial when using the framework to automatically discover multiple candidate models for steering behaviors simulation.

8. FUTURE WORK

8.1.   Macroscopic Movements.

The framework so far has been conducted to discover steering behavior at microscopic scale, meaning the framework focuses on behaviors of each agent. At macroscopic level, a group of many agents are treated as one big individual (similar to fluid simulation). To make it possible, the 2D world is transformed to a 2D Marker-and-Cell grid and it brings two advantages. First, each cell can store more information that a space entity cannot, such as relative distances to a certain landmark or density of an area. Second, treating agents as a group reduces the computation cost, hence the scale of the simulation can greatly increase.

8.2.   Evaluate the Approach Further with More Scenarios.

8.2.1.   More Evacuation Scenario

The set up for a more complex evacuation scenario is shown in Figure 53.a where a group of agents are placed in a close room, and their goal is to escape it by evacuating through the small hallway on the right side. Evacuation scenario brings two new challenges for the research. First, the framework will be tested with a closed (un-warped) space scenario where agents only have limited space to move around. Space entity component relies on empty space to steer agents, and with the space being limited, it is more challenging to find good space behaviors. Second, agents need to evacuate fast to replicate a real-life situation, hence a time metric will be used, and simulation time will be restricted. If a searching model cannot evacuate all agents within a specific time, the model is not qualified as a candidate for next the stage.
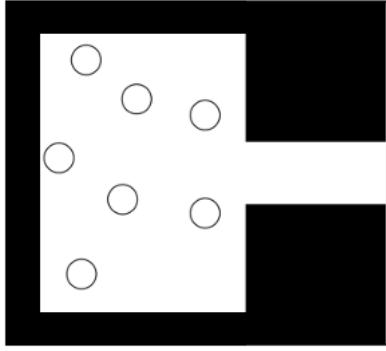
(a) An evacuation scenario lay out   (b) A Shepherding scenario layout
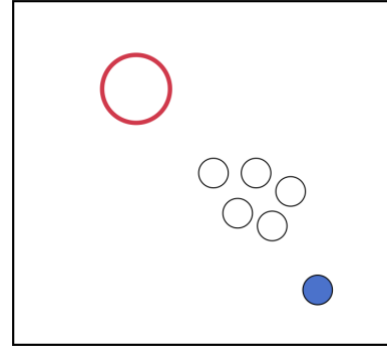
Figure 53. Example of two scenario initial set ups

8.2.2. Shepherding Scenario

Shepherding as shown in Figure 53.b is an interesting problem to test our automated approach. White agents present a flock of sheep, and the blue agent presents a dog. Red circle is the area where sheep need to be guided into. This scenario has different challenges compared to evacuation scenario. First, sheep agents and the dog agent will have two different sets of behaviors. Sheep's behavior might need to be manually crafted using macroscopic or mesoscopic models in (Yang, et al. 2020) because we focus on discovering dog's behavior automatically. Second, as observed in real life, a shepherd dog moves back and forth to guide the whole group of sheep to the destination. The model specification might need to be expanded so that the target agent (the dog) can extract property of a whole group of the same entity category (the sheep) to make the model work. The complexity of this scenario later can be increased by including multiple dogs where the communications between them are necessary, and bring a whole new behavior set to the model space.

8.3. Combined with Existed Methodology to Create Hybrid Approach.

Combine our automated approach with existing approaches to take advantage of both.

8.3.1. Combined with Manually Crafted Approach.

There are two potential research directions for this approach. First, new properties can be added based on the usefulness of them in a manually crafted approach. For example, velocity-based models are well known for predicting collision with obstacles in advance. Agents can retrieve information about "distance to collision point" or "time until collision". Adding this knowledge to the search space might assist to discover better candidates for mobile obstacle avoidance steering behavior. Second, the whole manual crafted behavior can be added as initial knowledge that agent knows at the beginning. Overtake behavior is a complex behavior where an agent, then it needs to speed up and avoids slow moving entities in front of it before steering back to the original path. Modelers can hand-crafted this behavior to the search space and give GA and agents opportunities to use it at the beginning.

8.3.2. *Combined with Data-Driven Approach.*

Data-driven approach can assist the framework in three aspects. First, it can be used to validate the discovered models further. Data of real humans' movements are extracted and compared to the agents' movements to test how close the simulations are to real situations. Second, it can be used to predict the next steering action and create a whole new set of behaviors for the framework. For example, agents have an option that is solely based on the positions of neighbors, then use an AI model to predict what it the next steering action. Third, GA can also be beneficial from machine learning models where beside fitness scores, a potential candidate can be predicted to be good or bad based on its model specification.

8.4. Develop an Agent-based Simulation and Modeling Application using a Front-end Development:

There is not much focus on cloud-based systems for agent-based simulation and modeling, even though there are many simulation software nowadays such as: NetLogo or StarLogo. The goal of this work is developing a frontend system where users can config and simulate steering behaviors for a group of agents. This application should be user friendly so that users with no computer science background can use it and it has the potential to attract more students to the simulation field.

REFERENCES

[1] C. Raynold, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987.

[2] B. D. Cory, K. Mubbasir, S. M. Jennie and B. I. Norman, "Generating a multiplicity of policies for agent steering in crowd simulation," *Computer Animation and Virtual Worlds,* pp. 483-494, 2015.

[3] M. Modrzejewski and P. Rokita, "Implementation of generic steering algorithms for AI agents in computer games," in *Studies in Big Data*, vol. 40, Springer, Cham, 2018.

[4] S. Mohamad, M. Oshita, T. Noma, F. M. N. Mohd Shahrizal Sunar, K. Yamamoto and Y. Honda, "Making decision for the next step in dense crowd simulation using support vector machines," in *VRCAI '16: Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry*, Zhuhai, China, 2016.

[5] N. Keller and X. Hu, "Towards Data-Driven Simulation Modeling for Mobile Agent-Based Systems," *ACM Transactions on Modeling and Computer Simulation,* pp. 1-26, 2019.

[6] M. J. Alam, M. A. Habib and D. Holmes, "Pedestrian movement simulation for an airport considering social distancing strategy," *Transportation Research Interdisciplinary Perspectives,* vol. 13, p. 100527, 2022.

[7] W. Wang, S. Lo, S. Liu and J. Ma, "On the use of a pedestrian simulation model with natural behavior representation in metro stations," *Procedia Computer Science,* vol. 52, pp. 137-144, 2015.

[8] J. Charlton, L. R. M. Gonzalez, S. Maddock and P. Richmond, "Fast Simulation of Crowd Collision Avoidance Evacuation simulation," in *Advances in Computer Graphics*, Calgary, 2019.

[9] R. A. Saeed, D. R. Recupero and P. Remagnino, "Modelling group dynamics for crowd simulations," *Personal and Ubiquitous Computing,* vol. 26, pp. 1299-1319, 2022.

[10] N. Wang, Y. Gao, C.-y. Li and W.-m. Gai, "Integrated agent-based simulation and evacuation risk-assessment model for underground building fire: A case study," *Journal of Building Engineering ,* vol. 40, p. 102609, 2021.

[11] C. Delcea and L.-A. Cotfas, "Increasing awareness in classroom evacuation situations using agent-based modeling," *Physica A: Statistical Mechanics and its Application,* vol. 523, pp. 1400-1418, 2019.

[12] M. Shirvani, G. Kesserwani and P. Richmond, "Agent-based modelling of pedestrian responses during flood emergency: mobility behavioural rules and implications for flood risk analysis," *Journal of Hydroinformatics,* vol. 22, no. 5, pp. 1078-1092, 2020.

[13] M. Shirvani and G. Kesserwani, "Flood–pedestrian simulator for modelling human response dynamics during flood-induced evacuation: Hillsborough stadium case study," *Natural Hazards and Earth System Sciences,* vol. 21, no. 10, pp. 3175-3198, 2021.

[14] S. Hassanpour and A. A. Rassafi, "Agent-Based Simulation for Pedestrian Evacuation Behaviour Using the Affordance Concept," *KSCE journal of Civil Engineering,* vol. 25, pp. 1433-1445, 2021.

[15] C. Hahn, T. Phan, T. Gabor, L. Belzner and C. Linnhoff-Popien, "Emergent Escape-based Flocking Behavior using Multi-Agent Reinforcement Learning," arXiv, 2019.

[16] C. Netz, H. Hildenbrandt and F. J. Weissing, "Complex eco-evolutionary dynamics induced by the coevolution of predator–prey movement strategies," *Evolutionary Ecology,* no. 36, pp. 1-17, 2021.

[17] N. K. Long, K. Sammut, D. Sgarioto, M. Garratt and H. A. Abbass, "A Comprehensive Review of Shepherding as a Bio-Inspired Swarm-Robotics Guidance Approach," *Transactions on Emerging Topics in Computational Intelligence,* vol. 4, no. 4, pp. 523-537, 2020.

[18] I. I. S. N. Y. M. K. I. Hiroyuki Hoshi, "Ro- bustness of herding algorithm with a single shepherd regarding agents' moving speeds," *Journal of Signal Processing,* vol. 22, no. 6, pp. 327-335, 2018.

[19] S. A. P. Quintero, F. Papi, D. J. Klein, L. Chisci and J. P. Hespanha, "Optimal UAV coordination for target tracking using dynamic programming," in *49th IEEE Conference on Decision and Control (CDC)*, Atlanta, 2010.

[20] A. I. Hentati, L. Krichen, M. Fourati and L. C. Fourati, "Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis.," in *14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Limassol, Cyprus, 2018.

[21] S. M. T. Islam and X. Hu, "eal-time On-board Path Planning for UAS-based Wildfire Monitoring," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Athens, Greece, 2021.

[22] G. VÁSÁRHELYI, C. VIRÁGH, T. N. GERGŐ SOMORJAI, A. E. EIBEN and T. VICSEK, "Optimized flocking of autonomous drones in confined environments," *Science Robotics,* vol. 3, no. 20, 2018.

[23] A. A. Paranjape, S.-J. Chung, K. Kim and D. H. Shim, "Robotic herding of a flock of birds using an unmanned aerial vehicle," *IEEE Transactions on Robotics ,* vol. 34, no. 4, pp. 901-915, 2018.

[24] Z. W. (伍志军), M. L. (李蒙良), X. L. (雷小龙), Z. W. (吴振元), L. Z. (. Chengkun Jiang (蒋承锟) a, R. M. (马荣朝) and Y. C. (陈勇), "Simulation and parameter optimisation of a centrifugal rice seeding spreader for a UAV," *Biosystems Engineering,* vol. 192, pp. 275-293, 2020.

[25] N. Hashimoto, Y. Saito, M. Maki and K. Homma, "Simulation of Reflectance and Vegetation Indices for Unmanned Aerial Vehicle (UAV) Monitoring of Paddy Fields," *Remote Sensing,* vol. 11, no. 18, 2019.

[26] D. Terzopoulos, "Artificial life for computer graphics," *Communications of the ACM,* vol. 42, no. 8, pp. 32-42, 1999.

[27] A. Dockhorn, M. Kirst, S. Mostaghim, M. Wieczorek and H. Zille, "Evolutionary Algorithm for Parameter Optimization of Context-Steering Agents," *IEEE Transactions on Games,* vol. 15, no. 1, pp. 26-35, 2023.

[28] S. Wang and G. Wainer, "A simulation as a service methodology with application for crowd modeling, simulation and visualization," *Simulation: Transactions of the Society for Modeling and Simulation International,* vol. 91, no. 1, pp. 71-95, 2015.

[29] S.-K. Wong, Y.-S. Wang, P.-K. Tang and T.-Y. Tsai, "Optimized evacuation route based on crowd simulation," *Computational Visual Media,* vol. 3, no. 3, pp. 243-261, 2017.

[30] O. Hesham and G. Wainer, "Advanced models for centroidal particle dynamics: short-range collision avoidance in dense crowds," *Simulation: Transactions of the Society for Modeling and Simulation International,* vol. 97, no. 8, pp. 529-543, 2021.

[31] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics,* vol. 3, no. 20, 2018.

[32] D. Helbing, I. Farkas and T. Vicsek., "Simulating dynamical features of escape panic," *Nature,* pp. 487-490, 2000.

[33] I. Karamouzas, B. Skinner and S. J. Guy, "Universal Power Law Governing Pedestrian Interactions," *Physical review letters ,* vol. 113, no. 23, p. 238701, 2014.

[34] S. Kim, S. J. Guy, K. Hillesland, B. Zafar, A. A.-A. Gutub and D. Manocha, "Velocity-Based Modeling of Physical Interactions in Dense Crowds," *The Visual Computer,* vol. 31, pp. 541-555, 2015.

[35] C. Totzeck, "An anisotropic interaction model with collision avoidance," arXiv, 2019.

[36] T. B. Dutra, R. Marques, J. B. Cavalcante-Neto, C. A. Vidal and J. Pettré, "Gradient-based steering for vision-based crowd simulation algorithms," *Computer graphics forum,* pp. 337-348, 2017.

[37] A. López, F. Chaumette, E. Marchand and J. Pettré, "Character navigation in dynamic environments based on optical flow," *Computer Graphics Forum,* vol. 38, no. 2, 2019.

[38] S. Yang, L. Tianrui, G. Xun, P. Bo and H. Jie, "A review on crowd simulation and modeling," *Graphical Models,* 2020.

[39] O. Hesham and G. Wainer, "Advanced models for centroidal particle dynamics: short-range collision avoidance in dense crowds," *SAGE journals,* vol. 97, no. 8, 2021.

[40] J. Qingge, W. Fuchuan and T. Zhu, "VPBS: A Velocity-Perception-Based SFM Approach for Crowd Simulation," in *International Conference on Virtual Reality and Visualization (ICVRV)*, Hangzhou, 2016.

[41] P. Scovanner and M. F. Tappen, "Learning Pedestrian Dynamics from the Real World," in *IEEE 12th International Conference on Computer Vision*, Kyoto, Japan, 2009.

[42] S. Lemercier, A. Jelic, R. Kulpa, J. Hua, J. Fehrenbach, P. Degond, C. Appert-Rolland, S. Donikian and J. Pettré, "Realistic following behaviors for crowd simulation," *Computer Graphics forum,* vol. 31, no. 2pt2, pp. 489-498, 2012.

[43] A. Vemula, K. Muelling and J. Oh, "Modeling Cooperative Navigation in Dense Human Crowds," in *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, 2017.

[44] A. Bera, S. Kim and D. Manocha, "Efficient Trajectory Extraction and Parameter Learning for Data-Driven Crowd," *Graphics Interface.,* vol. 70, 2015.

[45] J. Zhong, W. Cai, M. Lees and L. Luo, "Automatic model construction for the behavior of human crowds," *Applied Soft Computing,* vol. 56, pp. 368-378, 2017.

[46] J. Zhong, L. Luo, W. Cai and M. Lees, "Automatic Rule Identification for Agent-Based Crowd Models Through Gene Expression Programming," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems.*, Paris. France, 2014.

[47] J. Yue, D. Manocha and H. Wang, "Human trajectory prediction via neural social physics," in *European Conference on Computer Vision*, Tel Aviv, Israel, 376-394.

[48] J. Li, H. Ma and M. Tomizuka, "Conditional Generative Neural System for Probabilistic Trajectory," in *Conditional Generative Neural System for Probabilistic Trajectory Prediction*, Macau, China, 2019.

[49] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing,* pp. 65-85, 1994.

[50] Q. Steven, P. Francesco, K. Daniel, C. Luigi and H. João, "Optimal UAV coordination for target tracking using dynamic programming," in *49th IEEE Conference on Decision and Control (CDC)*, Atlanta, 2010.

[51] A. Idriss Hentati, L. Krichen, M. Fourati and L. F. Chaari, "Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Limassol, 2018.

[52] Z. Wang, Z. Rencheng, K. Tsutomu and N. Kimihiko, "Driver-automation shared control: Modeling driver behavior by taking account of reliance on haptic guidance steering," *IEEE Intelligent Vehicles Symposium,* pp. 144-149, 2018.

[53] H. Jin-Hyuk and S.-B. Cho, "Evolving Reactive NPCs for the Real-Time Simulation Game," in *IEEE Symposium on Computational Intelligence and Games*, United Kingdom, 2005.

[54] U. A. S. Iskandar, N. M. Diah, M. Ismail and A. Abdullah, "Comparing the efficiency of Pathfinding Algorithms for NPCs in platform games," *Journal of Positive School Psychology,* pp. 8434-8441, 2022.

[55] H. Le and X. Hu, "Automated Model Discovery For Steering Behavior Simulation," in *Annual Modeling and Simulation Conference (ANNSIM*, San Diego, CA, USA, 2022.