Evolutionary Algorithm for Parameter Optimization of Context Steering Agents

Alexander Dockhorn, Martin Kirst, Sanaz Mostaghim, Martin Wieczorek, and Heiner Zille

***Abstract*—Context Steering is a local approach to control an agent's movement in a dynamically changing scene. Recent works have formalized the context steering approach by Fray and presented a multi-objective view of the context steering problem. Combining a variety of different behaviors, which can be used multiple times in different configurations for different context maps, introduces a large number of parameters that need to be tuned to obtain well-performing agents. This work aims to use evolutionary algorithms to optimize context steering agents for various environments. A special focus lies on the evolution of agents that perform robustly across multiple variations of the same environment. To this end, we develop a real-valued encoding for a context steering agent along with three different fitness functions to represent different goals of the agent. Our experimental evaluation shows that an evolutionary optimization can produce agent configurations that perform well with respect to different tasks and show a high intra-task robustness. The proposed approach based on evolutionary optimization enables the user to optimize context steering agents such that they can explore environments while avoiding dynamic obstacles.**

***Index Terms*—Context Steering, Autonomous Movement, Robustness, Multi-Criteria Optimization, Evolutionary Algorithms**

I. Introduction

Various works in the research community have considered steering to enable agents to move and react in dynamic environments, e.g., computer games. In steering algorithms, agents try to reach a goal by making one decision at a time without storing any information on its environment. Each time-step, the agent selects its next movement direction based on its current surroundings. Steering approaches often make use of multiple simple behaviors, such as following or avoiding close objects. Complex behaviors can emerge from the aggregation of multiple such movement behaviors. The steering approach has become a useful tool for behavior designers, since each strategy on its own is easy to implement and understand.

As a result, steering has become a popular approach in game development [1]. Due to its simplicity, it is often recommended in introductory books for game development [2] and shows a widespread implementation in major game engines [3], [4].

With the rising complexity of game worlds, traditional steering approaches become hard to configure and cannot always ensure the expected result. For this reason, context steering [5]–[7] has been developed as an extension to steering to provide further information about the quality of different directions of movement. While even context steering cannot guarantee the expected result, it generally enhances the capabilities of traditional steering at the cost of further increasing the parameter space. Especially in dynamic worlds that confront the agent with multiple goals to pursue or objects to avoid, it can become a daunting task to balance out a large number of underlying behaviors and parameters. From the perspective of a game designer, it may further be of interest to provide agents that can perform robustly over different variations of the same game world. Failing to do so can result in inconsistent agent behavior, which reduces the player's immersion.

Since the agent's behavior in steering algorithms is dependent on the aggregation of multiple simple behaviors, which in turn can consist of many parameters, the total number of parameters to be tuned can be enormous. Moreover, the parameters will not be independent of each other, which results in a complex optimization task. Evolutionary algorithms have shown to be efficient in solving various optimization problems such as tuning the parameters of AI agents in games [8] or, for instance, tuning the learning rates of a training algorithm for artificial neural networks [9]. This raises the question whether they can be efficiently applied to optimize steering agent behavior.

To answer this question, this work formalizes the search for optimal parameters in a context steering agent as a multi-criteria optimization problem. The goal of the optimization process is to create agents that perform well with respect to various objectives. Special attention will be given to the robustness of the agent's movement across multiple variants of an environment. With our proposed optimization, we aim to ensure that the agent is robust against small changes in its environment, and therefore shows similar performance across variants of the same environment.

The main contributions of this paper are:

- Formalizing the search for optimal parameters in context steering as an optimization problem.
- Defining multiple fitness functions that are able to guide an optimization algorithm with respect to different goals.
- Formalizing a measure for robustness over different game-worlds and incorporate robust behavior into the optimization process.
- Applying an evolutionary algorithm to optimize context steering agents on predefined game worlds in different variations. The results are analyzed for agents with up to 153 parameters.

In the following section, we are presenting background on steering algorithms and review previous works on agent-based movement and robustness. A detailed problem definition

A. Dockhorn, H. Zille, and S. Mostaghim are with the Institute for Intelligent Cooperating Systems at the Department of Computer Science, Otto von Guericke University Magdeburg, Germany, E-mail: {alexander.dockhorn, sanaz.mostaghim, heiner.zille}@ovgu.de

M. Wieczorek and M. Kirst are part of Polarith GmbH, Magdeburg, Germany, E-mail: {martin.kirst, martin.wieczorek}@polarith.com

Manuscript received May 26, 2021

is presented in Section III as well as our specification of fitness functions and robustness. In Section IV, we present our solution for tuning an agent's behavior according to the presented problem using an evolutionary optimization process. Its experimental evaluation and the respective results will be presented in Section V. This paper ends with a short summary of our work, its potential impact, and a brief discussion of opportunities for future work (Section VI).

II. Background and Related Work

In the following, we briefly summarize algorithms for (context) steering (Section II-A) and methods for combining context maps (Section II-B). Furthermore, we present related work on agent-based movement (Section II-C) and robustness (Section II-D).

A. Steering and Context Steering

In steering, agents combine multiple behaviors to determine a movement direction, whereas each behavior returns a favored direction based on the agent's current surroundings [10]. Traditional steering works well for a flock of several agents when the focus lies on the movement of the whole flock as one entity. However, the individual agents may have flaws and inconsistencies in their movement. The final movement direction results from the aggregation of each behavior's preferred direction. A resulting major issue is that this combination of directions can lead to null vectors or oscillating behaviors [6].

Aware of these issues, Fray developed context steering [5]. In contrast to traditional steering, in context steering each behavior returns the rating of each movement direction instead of only the direction with the best rating, therefore returning the context of its decision. Moreover, Fray extended this system from a single goal to multiple different objectives that have to be taken into account for the agent's decision. The agents used in this work apply context steering for their decision-making, therefore the details of the context steering algorithm are explained in the following.

Context steering is a steering system in which an agent can perceive different types of objects that contribute to the different goals of the agent, and each goal can be expressed analytically as a so-called objective function. In terms of video games, these objective functions could be something like gathering objects to maximize your score or avoiding dangerous objects to minimize damage. The context steering system tries to find movement directions that maximize or minimize these objective functions. To perceive its surroundings, an agent has a sensor that consists of receptors. Each receptor has a direction vector and is linked to an element of a scalar array (cf. Figure 1a). This one-dimensional array of scalar values is called a context map. The direction vector of each receptor represents an agent's possible movement direction. Initially, the context map is filled with zero values. To get non-zero values that can be written into the context map, the behavior needs to be sensitive to a particular set of objects. If a receptor is pointing in the direction of an object that the behavior is sensitive to, a scalar value z is written into the corresponding element of the context map. The final value depends on the type of mapping being used for creating the context values and the distance of the object. An example is shown in Figure 1a. Here, a seek behavior is applied to the agent. The distance between the agent and an object is described by the difference in positions. For all receptors, it is checked if an object affects this receptor or not. This is done by measuring the angle ω between the receptors vector $\vec{r}$ and the vector from the agent to the object $\vec{o}$.

$$\omega = cos^{-1} \frac{\langle \vec{r}, \vec{o} \rangle}{\|\vec{r}\| \|\vec{o}\|} \in [0, \pi] \tag{1}$$

If the angle is smaller than a given threshold $\xi \in [0, \pi]$, the receptor is affected by the object, and a context value $z > 0$ is calculated as $z = \psi(\omega) g(\|\vec{o}\|) \in [0, 1]$.

The context value z depends on the exact location of the detected object in relation to the agent. $\psi(\omega)$ maps the angle $\omega \in [0, \xi]$ to a value in the range of $[0, 1]$. Similarly, the function $g(\|\vec{o}\|)$ maps the distance between the agent and the object to a value in $[0, 1]$, where only distance values $\in [r_{min}, r_{max}]$ are considered. The inner radius r_{min} is the distance, after which the detection of objects starts, while the outer radius r_{max} is the maximal distance to which objects are detected. The mapping type used for $\psi(\omega)$ and $g(\|\vec{o}\|)$ depends on the behavior an agent shall have. For a seek behavior that shall find close objects, an inverse linear mapping can be used for $\psi(\omega)$ and $g(\|\vec{o}\|)$. Other possible mapping options are squared mapping, square-root mapping, or their inverse versions.

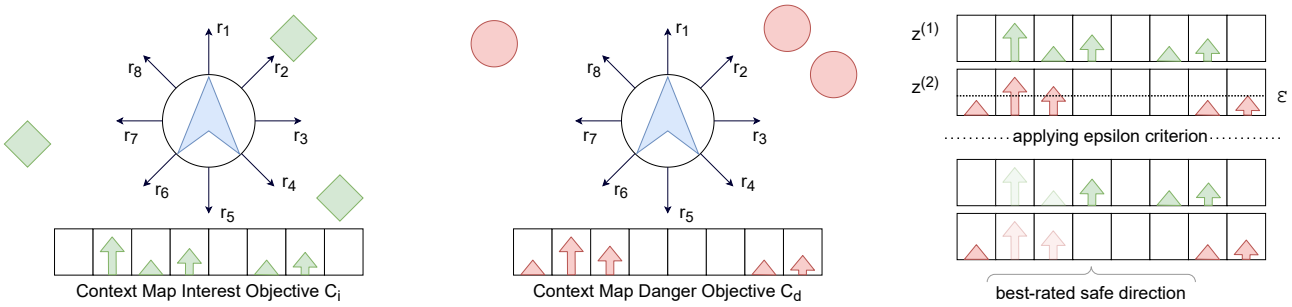
Figure 1a and Figure 1b illustrate this process by showing how different behaviors sense their environment and fill the values of a context map respectively. The length of each arrow indicates the behavior's rating of the corresponding direction. While the figure shows the use of different context maps, indicating interest and danger, it is also possible that multiple behaviors write into the same context map. In this case, we will use the maximum context value returned by all behaviors writing into the same map.

B. Combining Behaviors and Context Maps

While the previous subsection explains how single context maps are written, there exist multiple ways in which context maps can be combined to obtain a decision for the resulting movement direction.

The context steering system Fray developed [5] is able to handle multiple objective functions, which means that multiple context maps can be combined to a final steering direction. While the interest map highlights directions of objects the agent wants to approach, the danger map shows directions of objects to avoid. To combine both context maps to a single steering direction, first the lowest value in the danger map has to be found. All elements with a higher value are eliminated as they are considered to be too dangerous. In the interest map, the same elements are eliminated as they share the same directions. Out of the remaining elements of the interest map, the one with the largest value, is chosen as the steering direction.

Kirst et al. [6], [7] proposed an alternative approach in which the search for the best direction is approached from a multi-objective viewpoint. For a context steering scenario with the two context maps *Interest* and *Danger*, there are respectively



(a) Context map of a *seek* behavior. (b) Context map of a *flee* behavior. (c) Combined context map.

Fig. 1. Example of the context steering aggregation process. (a) and (b) indicate the receptors for perceiving objects and the context maps of the respective behavior. The length of an arrow of the context map indicates the strength of the response, i.e., its scalar value. (c) indicates how multiple context maps are combined according to the multi-objective view by Kirst et al. [5].

the two conflicting context maps, i.e., C_i for the *Interest* map values and C_d for the *Danger* map values. Each map is regarded as one objective in the resulting multi-objective optimization problem, whereas each direction is considered a solution.

In order to implement decision-making in this multi-objective optimization task, Kirst et al. examined different decision-making strategies. The best performing of those has been the ε-constraint method, which is also used in the remainder of this work. In the ε-constraint method, one objective is chosen to be optimized, and all other objectives act as a constraint for the optimization. The interest objective is the one that will be optimized and the danger objective acts as a constraint. The resulting steering direction is chosen as the direction with the largest interest value among all directions whose danger value is less than the constraint ε. If no solution satisfies the constraint, the one with the smallest danger value is chosen (cf. Figure 1c). Extending context steering by multi-criteria optimization methods [7] has been shown to make it more robust to noise but introduces additional parameters to be tuned.

C. Related Work on Agents-based Movement

The development of steering agents involves setting up a set of rules and behaviors that will ultimately be combined to result in an agent's steering behavior. Hand-crafting complex behaviors is a demanding and non-trivial task, due to the interplay of defined subsystems. For this purpose, previous work has focused on either tuning an agent's steering behavior or learning it from scratch.

Gerdelan and O'Sullivan proposed a system where an evolutionary algorithm is used to tune the rules of a fuzzy controller for steering [11]. In their work, they optimized the set of rules for fast and reliable movement in given 3D environments. Their evaluations show that this system is able to produce a better controller configuration than the hand-tuned reference set. Focusing on the efficient interplay of multiple steering agents, Berseth et al. have developed an optimization scheme for tuning the parameters of steering agents to improve the efficiency and visual fidelity of a simulated crowd [12]. Their method resulted in a parameter set that minimized turbulence at bottlenecks, reduced building evacuation times, produced emergent patterns, and overall increased the computational efficiency. Nevertheless, while performing well on a single environment, both of these works do not consider performance on multiple environments. Next to optimization algorithms, data-driven approaches to learn steering behaviors are also being developed. The work by Croitoru explored how trajectory data can be used to learn steering behaviors for approximating group movements [13]. For this purpose, a particle swarm optimization-based method has been proposed to derive agent steering behaviors based on Reynolds' boids model [14]. While the agents were able to traverse simple environments, the steering approach by Reynolds is known for resulting in deadlocks in more complex scenarios [6].

Other than steering, the problem of agent-based movement has been addressed in the field of heuristic search. In contrast to the problem discussed in this work, heuristic search aims to find an acceptable path within reasonable time, often achieved by interweaving planning and execution. In heuristic search, the map is usually represented in a graph data structure, and methods like A^* search or pruning techniques are applied to re-plan the path dynamically with a limited budget (e.g., a time-budget). Examples of such techniques can be found in works by Koenig [15] and Korf [16]. More advanced techniques have been explored, for instance in the works by Hernández et al., where an advanced pruning technique has been explored [17] and modifications of a Time-Bounded A^* were proposed [18].

In recent work, the efficiency of such algorithms has been improved by integrating domain knowledge into the agent's planning process. In the work by Lawrence et al. [19] a database of subgoals has been learned by preprocessing the map data. This higher-level structure reduces the computation effort at the cost of memory complexity. Munoz et al. [20] proposed to store domain knowledge in the form of a neural network used for choosing the agent's actions based on its local neighborhood. A more flexible approach has been proposed by Bulitko [21], who uses evolved heuristic functions to guide the agent-centered heuristic search in a class of similar path-finding problems. These methods considerably speed up the planning process and allow planning longer paths with the same budget restrictions.

The related works on real-time agent-based planning and movement relate to the proposed approach and the problem solved in this work in some aspects. Most importantly, path-

planning is an essential part of the solutions proposed in the mentioned works and is usually done via a graph data structure. In our work, we assume a static path has already been planned from start to destination, and the steering agent is only concerned with the execution of this plan under dynamic environments. Similarly to real-time heuristic search, a central aspect of the current work is the question of how we can create agents that execute such paths in a robust way, where the same agent can perform well in multiple different environments. This notion of robustness is introduced in the following.

D. Robustness

In our work, we focus on the robustness of steering behavior and the associated optimization processes. The robustness of a solution generally refers to a solution that is insensitive to variations or uncertainties that arise in different parts of the optimization problem. Different ways of measuring and optimizing robustness have been used in the literature, depending on the underlying application [22]. In the following, we give a brief overview of existing approaches to optimize robust solutions. The definition of robustness for the specific problem in the current article is given below in Section III-C.

Branke studied the robustness of single individuals and how to estimate their robustness in complex scenarios where the exact calculation is not possible [23]. For such cases, it has been proposed to estimate the individual's effective fitness by strategies such as re-evaluating one or multiple individuals, using perturbations of the decision variables for estimating the variation of fitness around a solution. Several ways of creating robust solutions are compared in regard to their efficiency in a work by Branke [24]. K. Deb and H. Gupta defined robustness for a multi-objective optimization to find not only one robust solution but a whole Pareto-optimal front [25].

In the context of steering, van Hoorn et al. have focused on the agent's performance in the same task but under varying dynamics of the environment. This is important for steering agents that need to frequently adjust their path according to other moving objects in their environment. For this purpose, a recurrent neural network-based controller for imitating human players in a racing scenario was developed [26]. Three fitness measures were defined, of which two evaluate the imitation of the player's commands, and the third measured how well the car performed on a particular track. While the robustness in the work above is mostly concerned about the uncertainty of the evaluation, cross-task robustness describes another type of challenge in which different environments need to be solved by the same individual. This problem has been studied by Pérez et al. who investigated the robustness of search-based controllers in a general video game playing task [27]. Therefore, they compared the agents' performance across multiple games of the general video game artificial intelligence framework [28]. Their goal was to evaluate how the different controllers react to the different reward systems and types of noise to test if there is a controller that is relatively robust towards these changes.

Similar to the works of van Hoorn et al., the present article focuses on the robust performance on variations of the same environment. To our knowledge, no work has yet combined context steering of agents in dynamic environments with robustness across different variations. In the next sections, we outline the design of our agent and the optimization process to achieve such robust solutions in the steering context.

III. Problem Definition

A. Problem Overview

The problem we address in this work is how we can build agents that behave well / optimally in a given environment according to multiple, conflicting goals. As introduced above, we aim to create agents that, in each time step, make a decision in which direction to move. Their goals include the collection of interest objects, the avoidance of dangerous objects, and staying close to a predefined path. Planning the path itself is not done by the agent. Instead, the focus lies on the execution of a path that has already been obtained by other means such as path-planning algorithms [29]. Therefore, the agent should react to dynamic changes in the environment which have not been known at planning time.

Let an environment $\mathbb{E}$ be described by a 2D space to be traversed. In each time-step, the agent can decide about its next movement direction. An agent A is defined by the settings of its parameters $\vec{x}$ and the corresponding decision-making algorithm (in this work represented by context steering). In addition, an agent contains a set of sensors $R = \{r_1, .., r_n\}$ to obtain information about its local environment. In context steering, sensors are used to detect different types of objects and their distance in a set of directions around the agent. As a result, an agent can be seen as a function that maps the current sensor input and its parameters $\vec{x}$ (here, consisting of steering behaviors and their parameters) to a movement direction.

The goal of our optimization problem is to find the agent's parameters so that the series of decisions made by the agent throughout the course of the simulation fulfills certain goals. Research in this area has commonly used two conflicting goals for context steering agents: collection of interest objects and avoidance of dangerous objects [5]. To this end, certain objects exist in the environment that are associated with either interest or danger (throughout this paper visualized as green diamonds and red circles respectively). An agent is typically expected to collect all the objects of interest while at the same time not colliding with any dangerous object. This process becomes even more challenging in case those objects move around, which would require frequent re-planning for path-planning agents and makes steering the preferable solution in case of short decision intervals. In addition, scenarios like racing games may require the agent to follow a certain recommended path. This path may be obtained by a user or planned by a path-planning algorithm. Deviating from this path is possible, but undesirable. These objectives (cf. specification of fitness functions in Section III-B) may be in conflict with each other. If dangerous objects are located on the defined path, the avoiding-danger objective can only be satisfied by sacrificing on the path-following objective. Similar trade-offs can be constructed for the relationships between the other objectives. An example of such an environment with a path and different objects is shown in Fig. 2. Another layer of complexity is introduced when

TABLE I
PARAMETERS OF CONTEXT STEERING BEHAVIORS AND THEIR BOUNDARIES

Parameter	Boundaries	Description
Magnitude Multiplier	[0, 10]	Impact of the behavior
Sensitivity Offset	[-90, 90]	Modifier of threshold. Affects the threshold ξ in the range $[0, \pi]$
Inner Radius	[0, 5]	The agent's minimum perception distance.
Outer Radius	[5, 50]	The agent's maximum perception distance.
Prediction Magnitude	[0, 10]	The agent's point of perception is projected along its current movement direction by a fixed offset.
Value Mapping $\psi(\omega)$	[0.33, 3]	Maps the angle ω to the range $[0, 1]$, uses an inverse URQ mapping.
Radius Mapping $g(\|\vec{o}\|)$	[0.33, 3]	Maps the distance $\|\vec{o}\|$ to the range $[0, 1]$, uses an inverse URQ mapping.
Plane Bend	[0, 90]	The parameter rotates the plane towards the detected object. (Only used in Avoid)
Max Prediction Time	[0, 5]	Together with its current position and velocity, the maximal prediction time is used to determine the position of the target object. (Only used in Pursue and Evade)
Objective Constraint	[0, 1]	The ε-constraint threshold of the Danger objective.

the robustness of the agent's behavior is considered. Different definitions of robustness exist in the literature (cf. Section II-D), which concern how well the agents can perform in a dynamic environment with external influences (e.g. wind or moving objects), or in environments that have similar characteristics (e.g., a set of labyrinths). Examples for such variations of environments can be seen in Fig. 6 in the appendix. Our specification of robustness will be highlighted in Section III-C. As a result, the problem solved in this work is the following: Find a set of parameters for an agent that enables it to satisfy all three objectives as best as possible while at the same time showing a robust behavior on different variations of the same environment. In this multi-objective optimization we aim to minimize each objective. The final outcome will be a Pareto-optimal set of solution candidates, of which the user can select a preferred individual.

In contrast to existing works on agent-centered movement (e.g., [15], [16]), the optimization's goal is not to find an optimal path to reach a specified goal. Our environment does not allow simulating future time steps and is by itself unknown to the agent. Under these conditions, planning-based approaches are not applicable without building a model of our environment, that would enable the agent to predict the outcome of its actions. Nevertheless, reinforcement learning would allow learning to iteratively navigate in such an environment. Similarly, our agent is meant to provide local decisions on its next movement direction based on its current sensor input, with the aim of achieving an optimal performance over the course of the simulation. In addition, we aim for a solution that does not store information in between time steps, thus keeping the memory and computation profiles as low as possible. Such a memory-free approach constrains the agent's capabilities in cases where the time to act is limited. Therefore, such steering methods may fail to recognize and escape deadlock situations. On the other hand, memory-free approaches may be applied in scenarios in which the environment is unknown. Although a predefined path is used for training, our agent does not need such information during runtime, and the path can be obtained from a different source and be unknown to the agent itself.

B. Specification of Fitness Functions

In this work, the task is to find a parameter set that enables a context steering agent to navigate through a game scene while collecting objects of interest, avoiding danger objects, and sticking as close as possible to a user-defined path that the agent cannot perceive. The fitness function uses a simulation of the scene in which different measurements are made to evaluate how well the agent solved these three requirements. The fitness functions are designed in a way that they provide continuous fitness landscapes, which is essential for a successful search with an EA [30]. The details of the measurements and resulting fitness functions are covered in the following.

Fitness by Target: The goal of this fitness function is to tell the agent that it has to collect as many targets as possible. $\mathcal{T}$ specifies the set of all targets objects, and U is a subset of $\mathcal{T}$, namely all uncollected targets. When all targets are collected ($U = \varnothing$), the agent obtains a perfect fitness score of zero for this part. For each target that is not collected by the end of the simulation, there is a distance-based penalty. How large the penalty is depends on the starting distance towards that target and how close the agents got towards the target while moving during the simulation. Formally, this is defined as:

$$f_T(\vec{x}) = \frac{1}{2} \sum_{u \in U} (1 + p_T(A(\vec{x}), u)) \tag{2}$$

with a distance based penalty p_T

$$p_T(A(\vec{x}), u) = \frac{\min_t\{d_t(A(\vec{x}), u)\}}{d_{t_{start}}(A(\vec{x}), u)} \tag{3}$$

where t_{start} is the starting time step of the simulation and $d_t(A(\vec{x}), u)$ is the distance between agent A (whose behavior is defined by the vector $\vec{x}$) and an uncollected target u at timestep t of the simulation. The distance-based penalty can reach a maximum value of 1.0, since the agent achieved at least the starting distance. In addition, we add a fixed penalty in Eq. (2) for not reaching a target, since there should be a difference between getting close to a target or reaching it.

Fitness by Danger: The danger fitness function is designed so that the agent avoids dangerous objects. To this end, we define a set of danger-events E_D, which indicate how often an agent A moves within a certain radius Θ of a danger object $d^{(e)}$. The set of all danger objects is called $\mathcal{D}$. When an agents enters a danger object's vicinity, an event e starts at timestep $t^{(e)}_{start}$, when the agents leaves the vicinity, the event ends at

Algorithm 1 Pseudo-code of the Fitness Evaluation

Input: Scene variations V, Solutions to evaluate S, Set of Behaviors B
Output: Set of evaluated solutions S
1: **for** $i = 1$ **to** $|S|$ **do**
2: **for** $j = 1$ **to** $|V|$ **do**
3: Run three independent simulations of solution $S[i]$ on scene variation $V[j]$ using the set of behaviors B
4: $f^{(1)}, f^{(2)}, f^{(3)} \leftarrow$ Calculate the fitness according to Eq. (6) for each of the three simulations
5: $f_j \leftarrow$ median($f^{(1)}, f^{(2)}, f^{(3)}$)
6: **end for**
7: $S[i].F \leftarrow$ set the fitness value from $\{f_j\}_{j\in\{1,\dots,|V|\}}$ according to the used robustness function h (see Eq. (7))
8: **end for**
9: **return** S

timestep $t_{end}^{(e)}$. We define danger-related fitness as:

$$f_D(\vec{x}) = \frac{1}{|\mathcal{D}|\Theta} \sum_{e\in E_D} \left(\Theta - \min_{t\in[t_{start}^{(e)}, t_{end}^{(e)}]} \{d_t(A(\vec{x}), d^{(e)})\} \right) \tag{4}$$

The penalty for one event is normalized by the threshold value so it can be at maximum 1.0. Additionally, all events are normalized by the number of danger objects so that a penalty of 1.0 is achieved if the agent crashes into each danger object with the maximum penalty. Theoretically, a value greater than one could occur since another event is created if the agent re-enters a threshold radius of a danger object it entered before. However, penalizing an agent even more when it relentlessly crashes into the same danger object multiple times benefits the goal of teaching the agent to avoid danger objects.

Fitness by Path: The last fitness function influences the agent to follow a certain predefined path. By placing a path close to danger objects, the agent acts "braver" if it finds a way to follow the path and simultaneously avoid the danger objects. A path far away from dangerous objects should result in a more cowardly behavior under the assumption that the agent sticks to the path, and the training is not dominated by one of the other fitness parts. Formally, we define the path-following fitness function f_P as follows:

$$f_P(\vec{x}) = \frac{\sum_{t=t_{start}}^{t_{end}} d_t(A(\vec{x}), P)}{(t_{end} - t_{start}) \cdot d_{max}} \tag{5}$$

where t_{start} is the starting time of the simulation, t_{end} is the end time of the simulation, $d_t(A(\vec{x}), P)$ is the smallest distance from the agent's position at time t to the path P, and d_{max} the maximum distance an agent shall have from the path.

The distance to the path is accumulated over the entire evaluation and normalized by a distance value that is considered bad behavior for a specific scene. However, the agent can move further away, which increases the penalty even more. The closer an agent sticks to the path, the better is the fitness value. Ideally, the agent follows the path directly and obtains a value of $f_P(\vec{x}) = 0$. Nevertheless, this scenario is not very likely since the paths used in this work are composed of linear segments, and the movement of the agent has some limitations that will not allow performing fast turning operations.

Relations between Fitness Functions: The described fitness functions are all designed in a way that the evolutionary algorithm tries to minimize each of them. For most environments, this will not be possible since the individual fitness functions can work contrarily. For example, if an interest object and a danger object share the same position, the agent can either collect the interest object and also touch the danger object or avoid the danger object and also not collect the target object, but minimizing both objectives is not possible. Similarly, if danger objects are placed on the path or target objects are not on the path, the agent can either follow the path, while touching danger objects and missing target objects, or leave the path to avoid danger objects and collect target objects.

Aggregation into a Single-Objective Problem: To select a final solution candidate from the Pareto-optimal front of candidate solutions, we chose not to minimize all the fitness parts individually, but to find a trade-off where the weighted sum of all fitness functions is minimal. Therefore, the overall fitness for a single run of a simulation is a weighted sum of the described three tasks as follows:

$$f(\vec{x}) = w_T \cdot f_T(\vec{x}) + w_D \cdot f_D(\vec{x}) + w_P \cdot f_P(\vec{x}) \tag{6}$$

In which $f_T(\vec{x})$, $f_D(\vec{x})$, and $f_P(\vec{x})$ follow the definitions above. w_T, w_D, w_P are the weights of each fitness part. Each fitness part is designed in a way that the optimal value is zero, so the EA has to solve a minimization problem.

C. Robustness of Context Steering

In order to produce agents that are robust in their behavior, we need to define the robustness measurement used in the optimization of the agents. Since the behavior of an agent is precisely defined by the set of parameters, the robustness is not measured in terms of perturbation of the steering parameters. Instead, what we are interested in is a behavior that performs well in different scene variations, i.e., the environment in which the agent needs to fulfill its tasks is changed. Thus, robustness for context steering is defined as follows.

The robustness of a solution $\vec{x}$ will be measured by a function h over the fitness of all variations of an environment:

$$\begin{aligned} &\text{optimize} && h(f_{v_1}(\vec{x}),\ f_{v_2}(\vec{x}),\ \dots,\ f_{v_{|V|}}(\vec{x})) \\ &\text{subject to} && v_j \in V, \vec{x} \in \Omega. \end{aligned} \tag{7}$$

Here, V is a set of environment variations on which the solution $\vec{x}$ is tested. f_{v_i} is the fitness function for context steering as in Eq. (6), evaluated using the problem variation $v_i \in V$. To evolve robust agents in our experiments, the employed evolutionary algorithm optimizes the problem shown in Eq. (7). Since the simulations for each agent's performance are non-deterministic, every evaluation is done three times, and the median of the three runs is used as the fitness of that respective scene variation (see Line 5 of Algorithm 1).

The individual fitness values on the scene variations are combined by a function h, which reflects the type of robustness the user prefers. In our work, we employ two versions. First, we use the median performance on all variations, which reflects a robustness that is concerned with overall performance, even if some variations perform worse or better. The second type of robustness concerns the worst-case outcome. In this case, the function h is the maximum of the fitness values. This type of robustness is useful for applications with critical infrastructure, e.g., self-driving cars. In both version, lower values of h indicate a higher robustness. The pseudo-code of our fitness evaluation is summarized in Algorithm 1.

IV. Algorithm Design

In this section, we describe the design of the evolutionary algorithm that is used to solve the defined problem from the previous section. Together with the algorithm design, we define the design of our encoding for the evolutionary algorithm, which stems from the selection and combination of multiple context steering behaviors. The goal of our evolutionary algorithm (EA) is to find a set of parameters for the steering behaviors so that the resulting agent behaves optimally with respect to different criteria (Section III-B) as well as robust in terms of variations of the same environment (Section III-C).

To reach this goal, the EA uses a population of individuals according to the specified encoding (see below). Given a random initial population, the EA applies evolutionary operators and selection mechanisms to this population to gradually improve the quality of solutions. For the environmental selection, a $(\mu + \lambda)$-selection scheme is used where the individuals are selected randomly. Additionally, elitism is used to retain the best found solutions. The elite individuals are stored in an external set with a size of 10% of the population size and used in the parental selection process of the following generation. In addition to the individuals generated through crossover and mutation, the current population undergoes a separate mutation process to enable the algorithm to make small adjustments to the agents. For further details, please refer to the appendix, where the pseudo-code of the algorithm is presented.

The basis of our algorithm are the underlying steering behaviors and their parameters, which together define the search space of our optimization. In this article, two context maps (interest and danger) are used, which are associated with two types of objects in the agent's environment. The five behaviors Seek, Flee, Avoid, Pursue and Evade were chosen (see the appendix for a detailed description and visual comparison), which, depending on the configuration of the experiments below, can be applied to interest or danger objects and map to one of the two used context maps.

The encoding of an agent's behavior is defined by the collection of all its parameters. Table I shows a summary of the parameters and their boundaries that are used in the configuration of an agent's behavior. The upper and lower boundaries are chosen based on preliminary experiments and the steering framework's[1] standard values. As a result, an individual of the EA consists of a real-valued vector in which each of the behaviors contains 7 (Seek, Flee) or 8 (Avoid, Pursue, Evade) parameters. In addition, the independent objective constraint parameter for the ε-constraint decision making is added to the encoding. In total, the length of the encoding depends on the number of used behaviors and differs between the experiments (see Section V).

Special attention was given to the parameter design of the value and radius mapping. The used software provides predefined settings for the different mapping types. These are linear, squared, square root mappings as well as their inverse forms. Those have two issues, i.e., (1) they need to be encoded as categorical genes requiring separate evolutionary operators, and (2) we expect that the change in behavior when switching between different mapping types is too big to ensure a continuous fitness landscape. The latter would lead to a sub-optimal optimization behavior since the evolutionary algorithm can no longer accumulate small changes to increase the fitness of an individual [30]. For these reasons, the predefined mapping types are replaced by a modified version of the Uniform Rational Quantization (URQ) mapping [31], which is able to produce different mapping types depending on a parameter u. A mathematical description of the modified URQ mapping and a comparison with the predefined mapping types is shown in the appendix. In addition, as specified above, the encoding consists only of real values, making standard genetic operators applicable in our experiments.

[1]The simulations in this work are based on the Polarith AI framework for Unity, available at https://assetstore.unity.com/packages/tools/ai/polarith-ai-free-movement-with-2d-sensors-92029

V. Experimental Evaluation

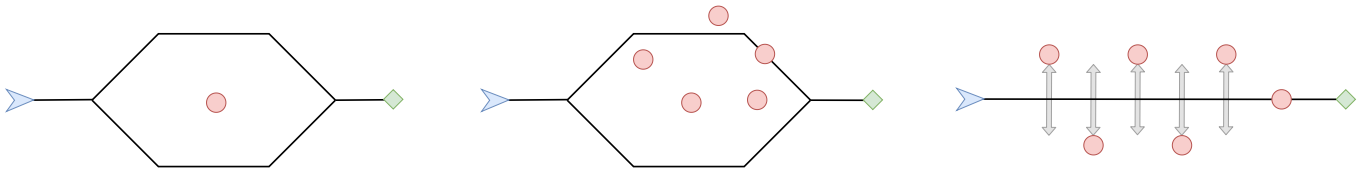
The experimental evaluation's goal is to test whether the proposed evolutionary algorithm can solve the configuration task of creating a complex context steering agent. We design three different experiments to examine several aspects of the performance. The first experiment is designed to evaluate the effect of different weightings of the fitness function (see Section V-B). The second experiment evaluates how much design effort has to be made to create successful context steering agents (see Section V-C). The third experiment evaluates the agent's robustness in different training variations (see Section V-D). The results of each experiment are consequently used in the following experiments. In particular, the most promising weightings from the first experiment are used in the second and third experiment and the better agent configuration from the second experiment is also used in the third experiment.

To obtain statistically meaningful results, for each experiment we compute 31 independent runs and report the median values of all performances in the following sections. The corresponding IQR values can be found in the supplemental material. For comparison of the results, a two-sided Mann-Whitney U test is used to tell if the results are significantly different from each other. A run ends when a fixed number of update steps has been performed or the target is reached.

A. Scenes and Evaluation Settings

Our evaluation consists of three scenes. Each features an agent, a target, one or more danger objects, and a set of paths that serve as a user guideline for the agent's movement. The agent's task is to collect the targets while avoiding all dangers and sticking as close as possible to the provided path. For Scenes 2 and 3, five different variations were created to test the agent's robustness in changing environments in experiment three, while the first two experiments only use the first variation of each scene.

The first scene consists of one target and one danger object. The danger object is placed between the agent and the target object so that the agent cannot just move forward. Two symmetrical paths shall lead the agent around the danger object at a defined distance (cf. Fig. 2 (a)). The second Scene has the same setup regarding the agent's position, target object,



(a) Scene 1, single obstacle (b) Scene 2, multiple obstacles (c) Scene 3, moving obstacles

Fig. 2. Visual representations of the three used scenes. Danger objects are shown as red circles, interest objects as green diamonds. The blue arrow marks the starting location of the agent and the lines show the desired paths. For Scenes 2 and 3, the respective variant 1 is shown.

TABLE II
COMPARISON OF AGENTS WITH HAND-SELECTED VS. ALL BEHAVIORS. EACH CELL INDICATES THE FINAL MEDIAN FITNESS VALUE $f(\vec{x})$ AND ITS THREE COMPONENTS $(f_T(\vec{x}), f_P(\vec{x}), f_D(\vec{x}))$.

	hand-selected		all behaviors		p-value
S_1	92.62	(0, 92.62, 0)	65.00	(0, 65.00, 0)	4.0e-07
S_2	98.25	(0, 98.25, 0)	191.58	(0, 191.58, 0)	0.0007
S_3	412.64	(0, 412.64, 0)	236.18	(0, 236.18, 0)	0.0555

and user paths. This time, multiple danger objects are placed randomly in the scene, except for variation 3, which contains only one danger object. The first variation of Scene 2 is shown in Fig. 2 (b) and the remaining variations are shown in the appendix Fig. 6. The third Scene involves dynamically moving dangers (cf. Fig. 2 (c)). The recommended path leads straight to the target. Moving danger objects continuously intersect with the user's path. This forces the agent to leave the path to avoid danger objects. The last danger object before the target is stationary and blocks the predefined path. The different variations affect the speed, moving distance, and spacing between the danger objects. The different parameters for each variation can be found in Fig. 7 in the appendix.

Since the scenes differ in appearance, they also have different fitness settings. In all three scenes, the danger threshold Θ from Eq. (4) is set to 2.0. For Scene 1 and 2, d_{max} from Eq. (5) is set to 5.0. In Scene 3, d_{max} is set to 10.0 as the scene is designed to force the agent to leave the path. The following settings were used for the evolutionary algorithm. The population size is 100, and the algorithm runs for 100 generations. A Gauss-mutation was chosen with a mutation rate of 0.05 and a standard deviation of 0.2 of the gene boundaries. For crossover the simulated binary crossover was used with a crossover probability of 1.0, and an η value of 20. The selection mechanism for reproduction is the tournament selection with a tournament size of 2. A step-based and time-independent controller was chosen for the agent's movement to provide a better and more reproducible evaluation. Between two frames, the agent travels a fixed step-size of 0.1 *Unity units*. In addition, the maximal turning rate has been set to a 5 degree angle to mimic a physics-based controller to a certain degree.

B. Experiment 1: Finding Suitable Weights

The first experiment is designed to evaluate the effect of different weightings of the fitness function parts. Due to limited space, we show numerical results and detailed discussion of this experiment in the supplementary material (Section D). We evaluate four different configurations for the three weights w_T, w_D and w_P as seen in Eq. (6). The first weighting $\vec{w}_{equal} := (1.0, 1.0, 1.0)$ assigns equal values to all three fitness parts, while the other three $\vec{w}_{target}$, $\vec{w}_{path}$ and $\vec{w}_{danger}$ emphasize one of the three compared to the other two respectively. In the first two scenes, the results show that there are no statistically significant differences between the performances of the four configurations. All weightings achieve perfect scores for collecting targets and avoiding dangers. In contrast, the only configuration that can avoid danger objects effectively in Scene 3 is $\vec{w}_{danger} = (1.0, 1.0, 3.0)$, most likely due to the increased complexity of moving danger objects along the agent's path. Since Scene 3 contains moving danger objects, an increased focus on avoiding these may be of importance to evolve meaningful agents. Therefore, for the experiments in the following we apply equal weights ($\vec{w}_{equal}$) to all three fitness parts for Scenes 1 and 2, while for Scene 3 the danger weighting $\vec{w}_{danger}$ is used.

C. Experiment 2: Search Space Dimensionality

The second experiment evaluates how much design effort has to be made to create successful context steering agents. This is done by comparing the hand-selected agents from Experiment 1 with agents that contain a combination of all possible behaviors. Compared with the hand-selected agents, there will be no difference between the agents with all behaviors across the three scenes. They contain the five behaviors: seek, flee, avoid, pursue, and evade. Of each behavior, four different versions exist in which the two object types are mapped to the two objectives so that all combinations are covered. In total, the agent contains 20 behaviors.

In Table II the fitness value of the hand-selected agents and the agents with all behaviors is shown. The table shows the median fitness values of 31 independent evaluations, whereas Fig. 3 shows the value distribution. The statistical analysis and IQR values can be found in the supplements. The agent with all behaviors is able to collect the target object while avoiding all danger objects. Furthermore, the agent with all behaviors performs better in Scenes 1 and 3, whereas the hand-selected agent is better in Scene 2. It is quite surprising that the agent with all behaviors performs better in Scene 1 and not in Scene 2, although both scenes have a similar structure and task. For Scene 1 and 3, the agent with all behaviors has overall better performance. Since both agents achieved to collect all targets and avoid all dangers, the path fitness function is the only one left to optimize. It is expected that the agent with all behaviors performs better since, with more behaviors, a better

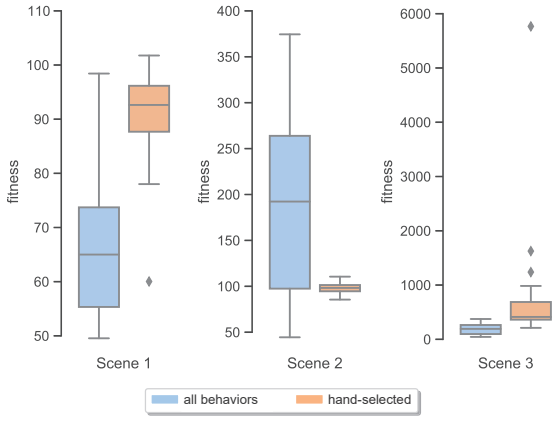


Fig. 3. Fitness distributions of 31 independent runs.

approximation of the path can be achieved. For Scene 1, the learning task is easy enough to quickly find better solutions for the agent with all behaviors than for the hand-selected agent. In Scene 3, the task is so difficult that the hand-selected agent has not as many good adjustment options as the agent with all behaviors to find a better solution in the given evaluation time. Scene 2 seems to be in the middle of that. It is simple enough that the hand-selected agent is able to find good solutions quickly, but it is too complex for the agent with all behaviors to find solutions of the same quality, as it has to tune far more parameters. However, some solutions outperform the best-found solutions of the hand-selected agent. Since the agents with all behaviors show more promising results this agent configuration is also used for the next experiment.

The results show that with more behaviors, a better configuration can be found. In all scenes, the target and danger part of the fitness is minimized to zero, and only the path part is left for optimization. Because of the controller, an agent can only make turns that result in a curve to follow the path, but the user-defined path is linear with hard edges. So it is impossible to follow the path perfectly. However, with more behaviors, the agent is able to do it more precisely.

To better understand the performance level of the used EA, we compare it with the well-known parameter optimization framework Hyperopt [32]. Results can be found in the supplementary material. Concerning the optimality of the EA solutions, the EA performs worse than the TPE-based Hyperopt in Scenes 1 and 2, while showing no significant difference in the more complex Scene 3. This indicates that EAs may unfold their potential for complex environments with moving targets compared to simpler Scenes. More specialized EAs, for instance, from the large-scale area [33], [34] may be used to improve the performance on such instances in future research.

D. Experiment 3: Evolving Robust Agents

The third experiment evaluates how robust agents can be created through a modified training process. Therefore, five different variations of Scenes 2 and 3 are made (cf. Figs. 6 and 7 in the supplements). For each variation, we train one agent. Additionally, two agents are trained on all variations of a scene according to the robustness definitions in Eq. (7). Resulting in seven differently trained agents per scene. Afterward, the agents are evaluated and compared on each variation individually.

The results of the robustness analysis are shown in Table III. The IQR values are shown in Table VII and the statistical analysis can be found in Table VIII in the supplemental material. For both scenes, the data shows the same results. The agents that are trained on one scene variation (Agent 1 - Agent 5) also have the best performance that was achieved in this variation $(V1 - V5)$, whereas the robust agents (Agent_{median} and Agent_{max}) have a mediocre performance in most of the variations. According to Table VIII in the supplements, the differences in performances can be regarded as significant compared to the respective best performing agent in all variations. Averaged over all scene variations, the robust agent Agent_{median} achieves the best performance.

This experiment shows that when agents are trained for a scene, they can become quite good at solving this particular task. However, when some changes were made in the scenes, this performance can drop dramatically. In contrast, an agent that is trained on several scene variations can robustly yield better results in all of the environments that were used in the training. This shows that with the presented type of robustness, a generalization effect can be achieved, at the cost of a slight performance decrease across all scenes. Note that further research is needed to evaluate how well this generalization transfers to additional, unknown variations, that were not available during the training phase.

VI. Conclusion

In this work, we proposed an evolutionary algorithm to optimize parameter configurations for context steering agents. For this purpose, a real-valued vector-encoding that fits the needs of context steering has been developed along with three fitness functions that enable the agent to perform according to different criteria. Our experiments have shown that (1) a different weighting of the individual fitness parts can influence the agent's final behavior, (2) the optimization is able to optimize parameter spaces with up to 153 variables resulting from up to 20 behaviors, and (3) resulting agents show high robustness across multiple variants of an environment. In terms of the latter, using the median performance over all scenes as the primary optimization goal has shown to result in the best average performance. In the case of performance-critical applications, the maximum performance of all scenes can be used instead, which ensures that the agent performs well across all scene variants but has shown to be slightly worse than using the median. This work shows that the proposed EA is able to optimize agents for static scenes with a single target object and simple dynamic scenes with fixed movement patterns. While this constrained experiment setup allowed us to better understand the capabilities of the optimization process, we would like to extend the approach to more complex scenarios. The existence of multiple target points and more complex movement dynamics would be of interest to enable the application of context steering agents in real-world environments. Furthermore, future work may involve

TABLE III
MEDIAN FITNESS VALUES OF DIFFERENTLY TRAINED AGENTS TESTED ON ALL VARIATIONS OF SCENE 2 AND SCENE 3. AGENT$_{median}$ AND AGENT$_{max}$ WERE TRAINED ON ALL VARIATIONS WITH THE RESPECTIVE ROBUSTNESS MEASURE. BEST PERFORMANCE IS MARKED WITH A GRAY BACKGROUND.

Agent	Scene 2						Scene 3					
	V1	V2	V3	V4	V5	average	V1	V2	V3	V4	V5	average
Agent$_{median}$	419.24	213.55	227.02	355.16	215.98	286.19	626.39	314.92	266.91	393.74	310.58	382.50
Agent$_{max}$	362.08	293.56	293.44	331.74	313.93	318.95	392.39	622.29	602.92	375.68	615.09	521.67
Agent 1	191.58	378.55	384.89	473.74	395.76	364.90	236.18	362.94	920.61	335.88	489.44	469.01
Agent 2	333.33	138.04	287.48	287.25	536.45	316.51	978.92	227.49	1359.89	890.53	1603.54	1012.07
Agent 3	546.03	698.79	117.19	1494.47	618.01	694.89	592.04	1015.18	74.11	715.71	598.50	599.10
Agent 4	343.66	238.16	1580.83	241.37	207.62	522.32	2540.85	1424.84	305.08	214.78	806.06	1058.32
Agent 5	337.02	333.73	617.22	269.01	154.93	382.50	1231.19	618.35	560.91	553.90	121.58	617.18

optimizing not just the performance of our agents, but also their diversity and authenticity. The former might be achievable by replacing the weighted sum in the fitness function with a multi-criteria optimization in which elements of the resulting Pareto-front would represent agents with differing personas.

REFERENCES

[1] S. L. Tomlinson, "The long and short of steering in computer games," *International Journal of Simulation Systems, Science and Technology*, 2004.

[2] S. Rabin, Ed., *Game AI Pro 360: Guide to Movement and Pathfinding*. Taylor and Francis, 2020.

[3] M. Modrzejewski and P. Rokita, "Implementation of generic steering algorithms for AI agents in computer games," in *Studies in Big Data*. Springer International Publishing, May 2018, pp. 15–27.

[4] J. Belón, "Flockai," https://github.com/juaxix/FlockAI, 2013.

[5] A. Fray, "Context steering: behavior driven steering at the macro scale," in *Game AI Pro 2: Collected Wisdom of Game AI Professionals.* CRC Press, 2015, pp. 183–193.

[6] M. Kirst, "Multicriteria-optimized context steering for autonomous movement in games," *Master's thesis, Otto-von-Guericke University Magdeburg*, 2015.

[7] A. Dockhorn, S. Mostaghim, M. Kirst, and M. Zettwitz, "Multi-objective optimization and decision-making in context steering," in *IEEE Conference on Games, COG*. IEEE, 2021.

[8] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–9.

[9] H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park, "Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates," *Neurocomputing*, vol. 11, no. 1, pp. 101–106, 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0925231296000094

[10] C. W. Reynolds, "Steering behaviors for autonomous characters," in *Game developers conference*, vol. 1999. Citeseer, 1999, pp. 763–782.

[11] A. Gerdelan and C. O'Sullivan, "A genetic-fuzzy system for optimising agent steering," *Computer Animation and Virtual Worlds*, vol. 21, no. 3-4, pp. 453–461, 2010.

[12] G. Berseth, M. Kapadia, B. Haworth, and P. Faloutsos, "SteerFit: Automated Parameter Fitting for Steering Algorithms," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, V. Koltun and E. Sifakis, Eds. The Eurographics Association, 2014.

[13] A. Croitoru, "Deriving low-level steering behaviors from trajectory data," in *2009 IEEE International Conference on Data Mining Workshops*. IEEE, 2009, pp. 583–590.

[14] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH*. ACM Press, 1987.

[15] S. Koenig, "Agent-centered search," *Artificial Intelligence Magazine*, vol. 22, no. 4, pp. 109–132, 2001.

[16] R. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2–3, pp. 189–211, 1990.

[17] C. Hernández, A. Botea, J. A. Baier, and V. Bulitko, "Online bridged pruning for real-time search with arbitrary lookaheads," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI, Melbourne, Australia*, 2017, pp. 510–516.

[18] C. Hernández, R. Asín, and J. A. Baier, "Time-bounded best-first search," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[19] R. Lawrence and V. Bulitko, "Database-driven real-time heuristic search in video-game pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 3, pp. 227–241, 2013.

[20] F. Munoz, M. Fadic, C. Hernandez, and J. Baier, "A neural network for decision making in real-time heuristic search," in *Proceedings of the 11th International Symposium on Combinatorial Search, SoCS 2018*, V. Bulitko and S. Storandt, Eds. AAAI press, 2018, pp. 173–177.

[21] V. Bulitko, "Evolving initial heuristic functions for agent-centered heuristic search," in *Proceedings of the IEEE Conference on Games (COG)*, 2020, pp. 534–541.

[22] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[23] J. Branke, "Creating robust solutions by means of evolutionary algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1998, pp. 119–128.

[24] ——, "Efficient evolutionary algorithms for searching robust solutions," in *Evolutionary Design and Manufacture*. Springer, 2000, pp. 275–285.

[25] K. Deb and H. Gupta, "Searching for robust pareto-optimal solutions in multi-objective optimization," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2005, pp. 150–164.

[26] N. Van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, "Robust player imitation using multiobjective evolution," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 652–659.

[27] D. Pérez-Liébana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the robustness of general video game playing agents," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[28] D. Perez-Liebana, S. M. Lucas, R. D. Gaina, J. Togelius, A. Khalifa, and J. Liu, *General Video Game Artificial Intelligence*. Morgan & Claypool Publishers, 2019, vol. 3, no. 2, https://gaigresearch.github.io/gvgaibook/.

[29] A. Botea, B. Bouzy, M. Buro, C. Bauckhage, and D. Nau, "Pathfinding in Games," in *Artificial and Computational Intelligence in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 21–31. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4333

[30] R. Kruse, C. Borgelt, C. Braune, S. Mostaghim, and M. Steinbrecher, *Computational Intelligence*. Springer London, 2016. [Online]. Available: https://doi.org/10.1007/978-1-4471-7296-3

[31] C. Schlick, "Quantization techniques for visualization of high dynamic range pictures," in *Photorealistic rendering techniques*. Springer, 1995, pp. 7–20.

[32] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, p. I–115–I–123.

[33] H. Zille, "Large-scale multi-objective optimisation: new approaches and a classification of the state-of-the-art," Ph.D. dissertation, Otto-von-Guericke-Universitaet Magdeburg, Fakultaet für Informatik, 2019.

[34] H. Zille and S. Mostaghim, "Linear search mechanism for multi- and many-objective optimisation," in *Evolutionary Multi-Criterion Optimization*, K. Deb, E. Goodman, C. A. Coello Coello, K. Klamroth, K. Miettinen, S. Mostaghim, and P. Reed, Eds. Cham: Springer International Publishing, 2019, pp. 399–410.

APPENDIX A
STEERING BEHAVIORS

The following behaviors are used in our experiments:

- **Seek, Flee:** Whereas seek generates context values towards the detected object, flee generates these values in the opposite direction.
- **Avoid:** A plane is created with its normal vector towards the detected object. The better a receptor is aligned with the plane, and thus being perpendicular to the detected object, the greater the context value it generates.
- **Pursue, Evade:** Pursue and evade are similar to seek and flee with the only difference being that not the actual position of the target is used but a predicted future position. Pursue creates objective values towards the target object and evades away from it.

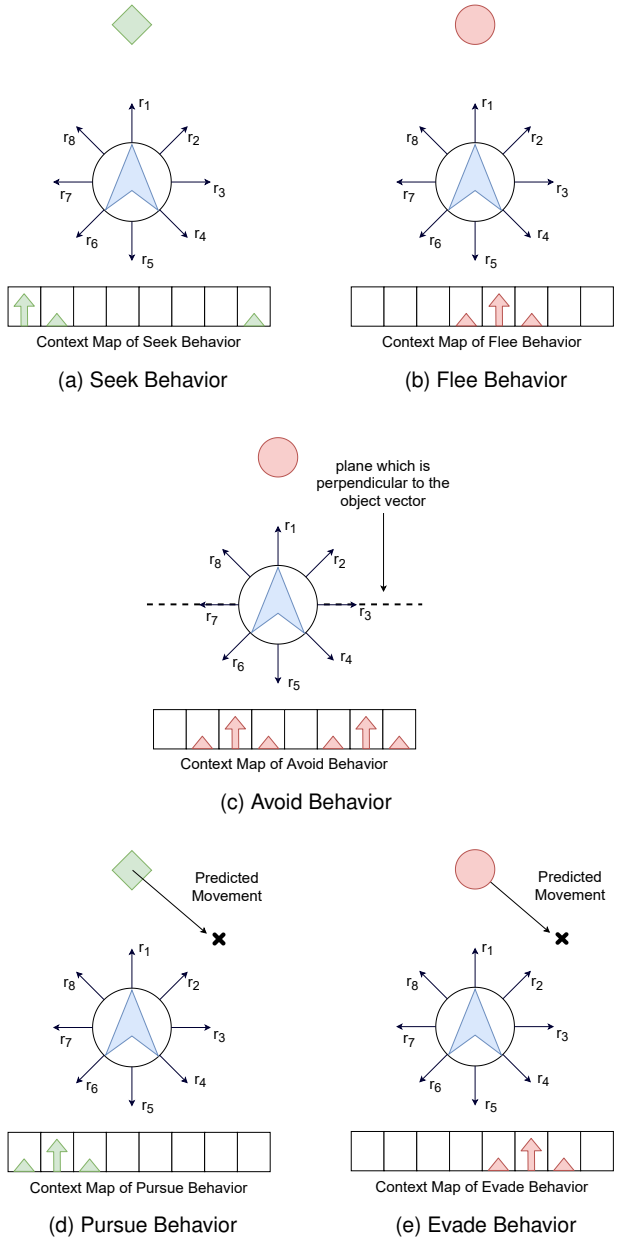


Fig. 4. Comparison of steering behaviors used in our algorithm.

APPENDIX B
URQ MAPPING

To match the boundaries of our parameters, we adjusted the original equation to be able to map values of x from arbitrary ranges $[x_{min}, x_{max}]$ to the interval $[0, 1]$ as follows.

$$\mathrm{URQ}(x, u) = \frac{u \cdot (x - x_{min})}{u \cdot (x - x_{min}) - x + x_{max}} \tag{8}$$

In our work, we chose an inverse URQ-mapping since it is more appropriate for our context steering tasks (i.e., objects that are far away should have a lower impact on the context map than close objects). In the following, we compare the modified URQ mapping with predefined mappings that have previously been used in context steering [7].

How the values are mapped is defined by the u value. The case of $u = 1.0$ is equivalent to an inverse linear mapping. Because the predefined mapping types serve as a baseline, the URQ-mapping shall try to cover the same range of mapping possibilities. By setting the range of the mapping parameters between 0.33 and 3.0, we obtain an adequate replacement for the predefined mapping types without causing big jumps in fitness when changing this value slightly. Fig. 5 shows the resulting inverse URQ-mappings for values of $u = 1.0$, $u = 0.33$ and $u = 3.0$.

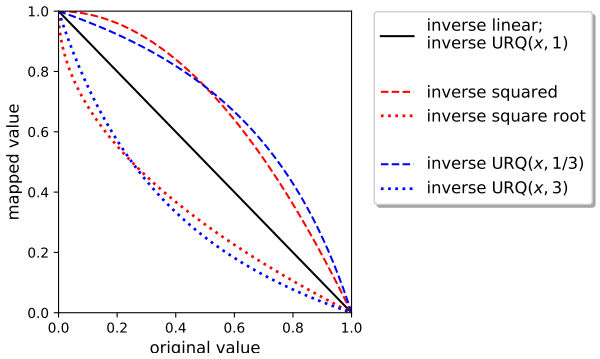


Fig. 5. Comparison of mapping functions.

APPENDIX C
EVOLUTIONARY ALGORITHM

Algorithm 2 Pseudocode of the used Evolutionary Algorithm

Input: Set of Scene Variations V, Set of Behaviors B, maxEvaluations
Output: Set of Solutions S with behavior parameters

$S \leftarrow$ Random initial population of Solutions, containing parameters for every used behavior
$S \leftarrow$ fitnessEvaluation(V, S, B) //see Algorithm 1
currentEvaluations $\leftarrow |S|$
$Q \leftarrow \emptyset$ //archive of elitist solutions
while currentEvaluations $<$ maxEvaluations **do**
    $P \leftarrow$ Selection of solutions from $S \cup Q$ for reproduction
    $O \leftarrow$ Apply crossover to P
    $C \leftarrow$ Apply mutation to S and O
    $C \leftarrow$ fitnessEvaluation(V, C, B) //see Algorithm 1
    currentEvaluations $\leftarrow$ currentEvaluations $+ |C|$
    $Q \leftarrow \lfloor 0.1 \cdot |S| \rfloor$ best solutions from $C \cup Q$
    $S \leftarrow |S|$ random solutions from C
end while
return S

TABLE IV
RESULTS FOR DIFFERENT WEIGHTINGS PER SCENE

Objectives	Weightings Scene 1				Weightings Scene 2				Weightings Scene 3			
	$\vec{w}_{equal}$	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$	$\vec{w}_{equal}$	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$	$\vec{w}_{equal}$	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$
$f_T(\vec{x})$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$f_P(\vec{x})$	92.62	93.42	93.59	91.89	98.25	97.62	96.75	95.95	174.40	285.82	31.99	412.64
$f_D(\vec{x})$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	140.06	69.51	346.73	0.0
total	92.62	93.42	93.59	91.89	98.25	97.62	96.75	95.95	314.46	355.33	378.72	412.64
weighted	92.62	93.42	280.78	91.89	98.25	97.62	290.25	95.95	314.46	355.33	442.71	412.64

TABLE V
MANN-WHITNEY U TEST FOR DIFFERENT WEIGHTINGS PER SCENE

p-value	Scene 1			Scene 2			Scene 3		
	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$
$\vec{w}_{equal}$	0.42	0.55	0.72	0.93	0.70	0.33	0.22	0.02	0.005
$\vec{w}_{target}$		0.89	0.75		0.86	0.48		0.48	0.07
$\vec{w}_{path}$			0.89			0.83			0.15

TABLE VI
EXPERIMENT1: IQR VALUES OF THE TOTAL FITNESS FOR DIFFERENT WEIGHTINGS PER SCENE

IQR-value	$\vec{w}_{equal}$	$\vec{w}_{target}$	$\vec{w}_{path}$	$\vec{w}_{danger}$	all behaviors
S1	8.48	8.05	6.08	7.94	18.39
S2	6.79	8.23	12.04	9.58	166.41
S3	128.03	127.54	12.42	325.57	225.82

APPENDIX D
DETAILED RESULTS OF EXPERIMENT 1

The first experiment is designed to evaluate if an evolutionary algorithm can find context steering solutions and which effect the different weightings of the fitness function have. Therefore, four different weightings were chosen. The first weighting is an equally weighted fitness function with $w_T = w_D = w_P = 1$. The other three weighting configurations are obtained by weighting one of the three objectives higher than the other two respectively, i.e., the second setting uses $w_T = 3$, $w_D = w_P = 1$, and so forth. These weightings are applied to hand-selected agents of which it is known that feasible solutions exist. In the following, we list which behaviors are used for the hand-selected agents in each of the scenes.

$$\begin{aligned}
&\text{Scene1:} \quad Seek(I \to I),\ Seek(D \to D)\\
&\text{Scene2:} \quad Seek(I \to I),\ Seek(D \to D)\\
&\text{Scene3:} \quad Seek(I \to I),\ Seek(D \to D),\\
&\qquad\qquad Avoid(D \to I),\ Evade(D \to I)
\end{aligned}$$

where Seek$(I \to I)$ indicates, that a seek behavior is used that maps interest objects to the interest context map, with I indicating interest and D indicating danger.

A. Results

The resulting fitness values for all three scenes can be found in Table IV and an analysis of the different weightings is shown in Table V. For Scene 1 and 2, the tested weightings did not significantly impact the resulting agent's behavior in terms of their final fitness value. However, in Scene 3 different results were achieved. While the equal weighting scored best in terms of overall fitness, the danger-emphasizing weighting is the sole solution that managed to avoid all danger objects. The weighting that puts higher emphasis on the path has found solutions that had minimal path fitness. The results of the Mann-Whitney U test which compares the results of all 31 independent evaluations per weighting support the claim that the resulting behaviors of tested weightings are statistically different.

For the other two experiments, we decided that the equal weighting will be used for Scenes 1 and 2, as we did not observe significant differences between tested weightings. For Scene 3, we chose the danger-focused weighting as this is the only weighting that achieves to avoid all danger objects completely.

B. Interpretation

In Scenes 1 and 2, the given task was not complex enough to enforce different movement strategies for different weightings. These scenes were designed in a way that by avoiding the danger object, following the path, and collecting the target object happened automatically. While in Scene 3, the task was designed so that the individual fitness parts work contradictory to each other. Here either the path could be followed, or danger objects could be avoided, but both at the same time is not possible. So if the scene is designed with contradictory fitness parts in mind, the different weightings will have an effect on the training and the found solutions.

TABLE VII
IQR VALUES OF DIFFERENTLY TRAINED AGENTS TESTED ON ALL VARIATIONS OF SCENE 2 AND SCENE 3.

Agent	Scene 2					Scene 3				
	V1	V2	V3	V4	V5	V1	V2	V3	V4	V5
Agent_{median}	185.21	64.81	62.29	91.70	43.82	494.59	731.16	567.07	147.48	242.78
Agent_{max}	108.71	113.76	53.98	96.17	91.18	139.66	139.24	251.04	101.53	166.61
Agent 1	166.48	599.88	321.44	671.46	665.27	203.60	673.57	694.46	701.99	1021.12
Agent 2	137.41	41.44	112.10	166.23	236.25	533.76	169.30	798.22	613.49	538.23
Agent 3	1044.31	1256.00	78.76	1142.54	1079.35	438.39	612.11	27.79	610.73	1132.58
Agent 4	134.29	270.17	708.06	45.54	359.80	525.54	579.47	993.47	121.44	881.13
Agent 5	232.29	289.27	318.41	193.27	55.17	339.57	934.79	519.84	628.19	23.80

TABLE VIII
STATISTICAL COMPARISON (P-VALUE) OF THE BEST FOUND AGENT FOR A VARIATION WITH ALL OTHER AGENTS OF SCENE 2 AND SCENE 3.

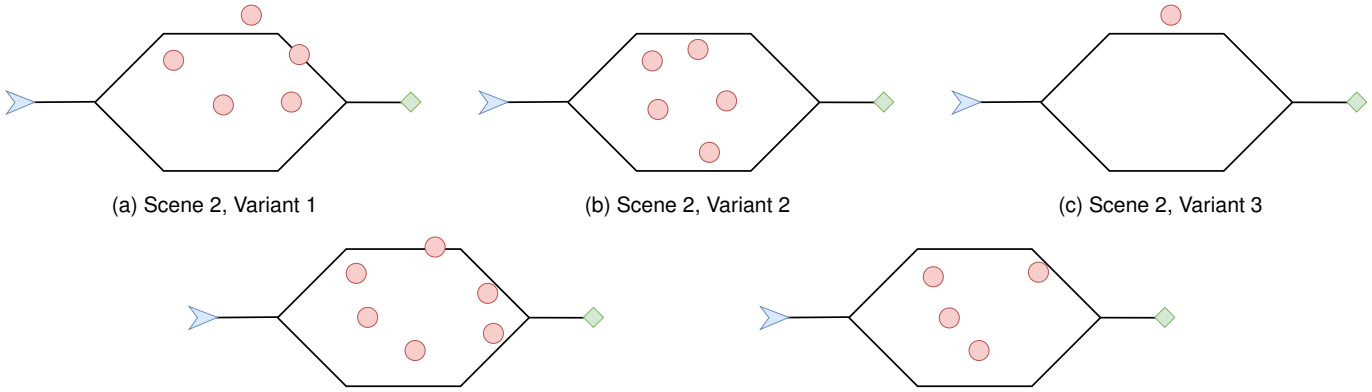
Agent	Scene 2					Scene 3				
	V1	V2	V3	V4	V5	V1	V2	V3	V4	V5
	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5
Agent_{median}	1.65e-05	1.00e-08	8.57e-11	1.65e-08	1.47e-06	6.23e-07	1.19e-06	5.84e-10	2.12e-05	1.80e-10
Agent_{max}	3.46e-07	5.36e-11	1.40e-11	3.46e-07	3.67e-11	8.50e-09	1.11e-06	2.50e-11	9.12e-10	1.87e-11
Agent 1		1.40e-11	1.40e-11	4.43e-11	1.40e-11		3.36e-09	1.40e-11	1.24e-10	1.40e-11
Agent 2	1.80e-10		1.54e-11	1.84e-09	3.67e-11	5.89e-11		1.70e-11	1.80e-10	1.40e-11
Agent 3	1.87e-11	1.40e-11		1.40e-11	1.40e-11	4.03e-11	7.81e-11		1.97e-10	1.54e-11
Agent 4	3.98e-09	2.75e-11	1.70e-11		4.03e-11	3.33e-11	4.47e-10	4.03e-11		4.88e-11
Agent 5	2.37e-10	3.03e-11	1.40e-11	2.16e-10		3.03e-11	1.64e-10	2.50e-11	6.62e-09	

TABLE IX
COMPARISON OF AGENTS TRAINED WITH HYPEROPT VS. EVOLUTIONARY ALGORITHM. EACH CELL INDICATES THE MEDIAN, IQR, BEST AND WORST VALUES OF THE FINAL FITNESS $f(\vec{x})$.

	Hyperopt				Evolutionary Algorithm				p-value
	Median	IQR	Best	Worst	Median	IQR	Best	Worst	
S_1	53.47	7.83	37.51	62.80	65.00	18.93	49.54	98.42	3.7e-05
S_2	69.70	35.23	51.59	315.62	191.58	185.85	44.48	374.44	7.6e-06
S_3	206.46	119.82	112.87	910.24	236.18	227.34	114.79	955.29	0.2206

APPENDIX E
DETAILED RESULTS OF EXPERIMENT 2

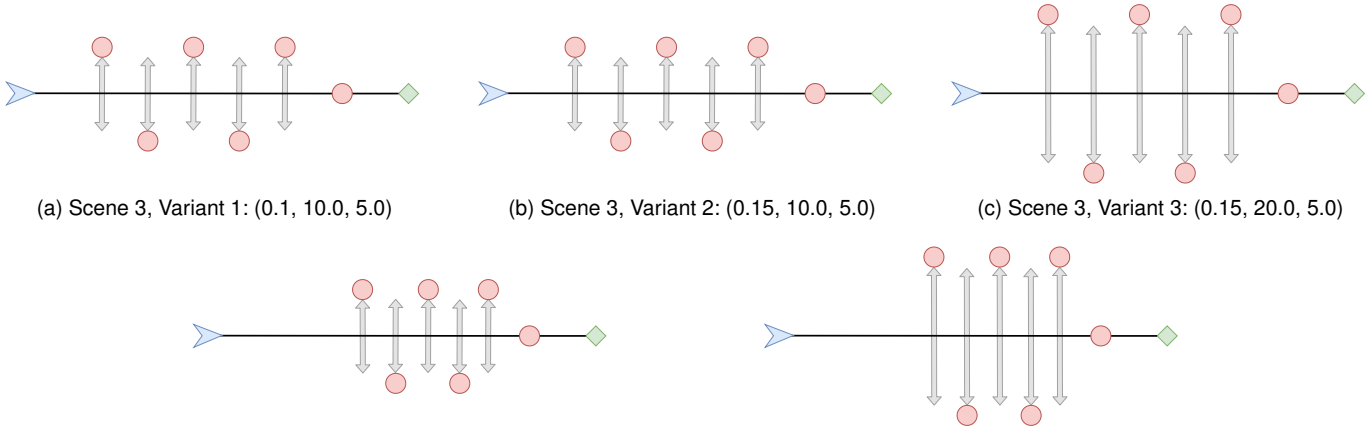
For comparison with other state-of-the-art parameter optimization methods, we repeat the optimization of an agent with all behaviors (similar to Experiment 2) using Hyperopt. Table IX shows the fitness values of the agents tuned by Hyperopt with the agents tuned by our proposed EA. Similarly to Table II, the table shows the median fitness values of 31 independent evaluations. In addition, we show the IQR, best and worst obtained fitness values for each method. According to the value distribution and a two-sided Mann-Whitney U test, the agent optimized by Hyperopt performs significantly better in Scenes 1 and 2, whereas there seems to be no significant difference in agent performance in the more complex Scene 3 that contains moving danger objects. Comparing the best and worst values of both algorithms reveals that both algorithms yield similar best and worst results. However, the IQR of the EA shows that the values around the mean show a wider spread.



(a) Scene 2, Variant 1 (b) Scene 2, Variant 2 (c) Scene 2, Variant 3

(d) Scene 2, Variant 4 (e) Scene 2, Variant 5

Fig. 6. Variants of Scene 2.



(a) Scene 3, Variant 1: (0.1, 10.0, 5.0) (b) Scene 3, Variant 2: (0.15, 10.0, 5.0) (c) Scene 3, Variant 3: (0.15, 20.0, 5.0)

(d) Scene 3, Variant 4: (0.1, 10.0, 3.0) (e) Scene 3, Variant 5: (0.2, 20.0, 3.0)

Fig. 7. Variants of Scene 3 listing the parameters (speed, distance, spacing) of dynamic danger objects.