A Versatile Platform in Unity for Prototyping Evolutionary-Behavior and AI Research

Ashish Ajin Thomas
University of Toronto
ashish.ajinthomas@mail.utoronto.ca

Tanayjyot Singh Chawla
University of Toronto
tj.singhchawla@mail.utoronto.ca

Anoop Rehman
University of Toronto
anoop.rehman@mail.utoronto.ca

Mohamed Tarek
University of Toronto
mohamedt.mohamed@mail.utoronto.ca

Hector Chen
University of Toronto
hector.chen@mail.utoronto.ca

Matthew Chen
University of Toronto
matthewxk.chen@mail.utoronto.ca

Kushagra Raghuvanshi
University of Toronto
kushagra2125@gmail.com

Sean Ma
University of Toronto
sean.ma@mail.utoronto.ca

Ujjvel Lijo
University of Toronto
ujjvel.lijo@mail.utoronto.ca

Jeslyn Wang
University of Toronto
jeslyn.wang@mail.utoronto.ca

Tejas Raghuvanshi
tejas.raghuvanshi@mail.utoronto.ca

***Abstract*—Researchers exploring evolutionary behavior often face a steep learning curve when integrating neural evolution, environment design, and real-time visualization. To address this, we introduce a versatile platform in Unity that simplifies prototyping for evolutionary behavior and AI research. Our approach combines a 2.5D tile-based environment, configurable resource distributions, and a modular neuro-evolution engine, enabling users to rapidly define fitness functions and agent parameters. We demonstrate the platform's flexibility through standard tasks (XOR and Sine approximation) as well as three custom scenarios highlighting aggression, cooperation, and resource disparity. Results show that agents evolve distinct strategies with minimal reconfiguration, underscoring the platform's utility in producing emergent behaviors. By lowering barriers to scenario setup and data collection, our work aims to accelerate iterative experimentation and expand opportunities for AI-driven evolutionary studies.**

I. Introduction

AI-based evolution simulations have emerged as a powerful tool for studying emergent behaviors and adaptation in artificial agents. By leveraging evolutionary algorithms, researchers can observe how AI-driven entities develop strategies for survival, cooperation, and competition without explicit programming. These simulations provide insights into complex system dynamics, population behavior, and evolutionary decision-making.

One of the most widely used algorithms for evolving artificial intelligence is **NeuroEvolution of Augmenting Topologies (NEAT)**. NEAT dynamically evolves both the structure and weights of neural networks, allowing agents to develop increasingly sophisticated behaviors over generations. Unlike fixed-topology neural network training, NEAT begins with simple architectures and gradually adds complexity through genetic mutations and speciation. This adaptive nature makes NEAT particularly well-suited for open-ended learning, reinforcement learning tasks, and evolving AI in dynamic environments.

Unity serves as an effective platform for visualizing these evolutionary processes due to its high-performance rendering engine, physics simulation capabilities, and flexibility in designing interactive environments. However, despite Unity's strengths, there is currently no publicly available, easy-to-integrate NEAT implementation for Unity using C#. Moreover, even when NEAT is present, researchers still face significant overhead in setting up foundational components (e.g., environment design, resource distribution, and creature perception).

Our work addresses these gaps by developing a **modular and extensible NEAT-based evolutionary simulation platform in Unity**. The platform not only includes a direct NEAT implementation but also provides a whole suite of additional tools: real-time visualization modules, character controllers, customizable 2.5D map configurations, dynamic heatmap spawn for resource distribution, and adjustable fields of view (FOV) or ray-based creature vision. Researchers can seamlessly tailor the simulation to fit specific research questions, whether studying resource disparity, cooperative strategies, or emergent violence. By streamlining these features and delivering a comprehensive toolkit, our project substantially lowers the barrier for evolutionary AI research and accelerates hypothesis testing.

A. Motivation

Evolutionary AI simulations play a crucial role in artificial life, reinforcement learning, and automated agent training. These simulations allow for the study of **adaptation, survival, cooperation, and competition**, providing a valuable

testbed for exploring how intelligent behaviors arise in digital organisms. NEAT has been widely adopted in academic research and industry applications due to its ability to evolve neural networks without predefined architectures, making it a powerful alternative to traditional deep learning techniques.

While NEAT has been successfully implemented in various programming languages, **there is no standard implementation of NEAT for Unity and C# that integrates seamlessly with game environments**. Beyond the lack of a straightforward NEAT port, most existing solutions do not come packaged with robust visualization and scenario-building tools, meaning users must develop or adapt these functionalities themselves. A streamlined, modular NEAT implementation in Unity—bundled with tools like resource placement, vision settings, and an easily customizable environment—would empower a wider audience to prototype and conduct evolutionary experiments.

By providing an accessible, modular platform for NEAT-based evolution in Unity, our work lowers the barrier to entry for AI researchers and game developers. This platform enables quick iteration and evaluation of evolutionary strategies in AI populations, offering insights into **behavioral dynamics that emerge over generations**. The extensive suite of tools we provide—ranging from spawn heatmaps to easily adjustable vision rays—ensures that even nuanced research questions, such as the effect of resource disparity or the advantages of cooperative behavior, can be readily investigated.

B. Related Works

Several prior works have explored neuroevolution for AI behavior modeling and interactive simulations. Stanley et al. [1] demonstrated the potential of **real-time neuroevolution** in interactive environments with the **NERO** project [2], where human players trained AI agents using NEAT to adapt to different combat strategies. Other research efforts have applied NEAT to robotics, multi-agent systems, and game AI, showing its effectiveness in evolving controllers capable of complex decision-making [3] [4] [5] [6].

Despite these successes, **most existing implementations of NEAT are developed for Python, Java, or C++, with few tailored for Unity and C#**. The well-known SharpNEAT library provides a NEAT implementation in C#, but it is designed for console-based applications and lacks direct integration with Unity. Furthermore, Unity's ML-Agents Toolkit [7] supports reinforcement learning but does not include evolutionary strategies like NEAT, creating a gap in accessible tools for neuroevolution in Unity environments [8].

A few independent projects, such as UnitySharpNEAT, have attempted to bridge this gap, but they remain limited in scope and usability. Our work builds upon these efforts by offering a **fully integrated, user-friendly NEAT platform for Unity**, which not only provides a NEAT engine but also includes a host of world-building and visualization tools. This enables real-time evolutionary simulations with minimal setup, making it easier for researchers to focus on experimentation rather than infrastructure.

C. Problem Definition

The primary challenge addressed in this paper is the **lack of an easily accessible NEAT implementation for Unity** that also integrates all necessary components for comprehensive evolutionary experimentation. Researchers and developers aiming to conduct evolutionary AI experiments in Unity currently face three major obstacles:

1) **No standard Unity-compatible NEAT implementation:** Existing NEAT libraries require extensive modification to work within Unity's game engine.
2) **High technical barrier:** Researchers must either implement NEAT from scratch or adapt complex external libraries, diverting effort from core research objectives.
3) **Limited or missing simulation tools:** Most NEAT implementations focus on backend computation without real-time, interactive visualization or scenario-building features, making it difficult to analyze evolving AI behaviors and customize environments.

Our solution is a **modular, extensible Unity-based NEAT simulation framework** that overcomes these challenges by providing:

- A prebuilt NEAT implementation optimized for Unity and C#.
- An intuitive, modular system for defining evolutionary scenarios with adjustable resource distribution, creature vision, and environment structure.
- Real-time visualization tools for tracking agent behaviors and adaptation over generations.

This platform serves as a foundation for AI research, game development, and educational applications, enabling efficient testing of evolutionary hypotheses without the need for extensive custom development. Researchers can quickly configure scenarios—from resource-scarce environments to large-scale population dynamics—and observe how evolved strategies emerge under different conditions. In the following sections, we outline the design, implementation, and evaluation of this platform, demonstrating its utility in studying emergent behaviors in evolving AI populations.

II. Methodology

In this section, we detail the overall workflow of our NEAT-based platform, describe how simulations are configured, and explain how researchers can customize various aspects of the environment and evolution process. We also outline minor modifications made to the original NEAT implementation and introduce our preliminary verification tests (XOR and Sine approximation).

A. Overview of the Simulation Pipeline

A single simulation run typically follows these steps:

1) **Environment Setup and Configuration:** The user defines or imports a 2.5D map design, configures resource distributions through heatmaps, and specifies field-of-view (FOV) parameters (e.g., number of vision rays).

2) **Initial Agent Creation (Generation 0):** The NEAT algorithm initializes N agents with randomized weights and minimal network connections (input and output layers only).
3) **Unity Simulation:** Agents are spawned in the Unity environment. They interact according to the chosen scenario, which may be time-based (limited duration) or event-based (e.g., survival until all agents die).
4) **Fitness Evaluation:** A fitness metric is computed at the end of each simulation round based on scenario objectives (e.g., collecting resources, surviving longer, achieving a particular task).
5) **Evolutionary Update:** Fitness values are passed back to the NEAT algorithm. Crossover and mutation produce the next generation of agents.
6) **Next Generation Deployment:** The updated population is reintroduced into the Unity scene. This cycle repeats until a stopping criterion (e.g., maximum generations) is met.

Throughout this process, the platform captures data for analysis (e.g., population size, species diversity, or user-defined metrics) and provides real-time visualizations.

B. NEAT Algorithm Integration

We integrated *NeuroEvolution of Augmenting Topologies* (NEAT) into Unity with minimal alterations to the core approach. NEAT begins with simple neural networks and incrementally adds complexity through *mutation* (adding nodes and connections) and *crossover* (mixing genomes between selected parents). Our modifications include:

- **Elitism:** Top-performing individuals are carried over unchanged to the next generation, ensuring preservation of highly adapted genomes.
- **Stagnation Tracking:** Species that fail to improve for a specified number of generations are replaced, preventing wasted evaluations.

All other parameters (e.g., mutation probabilities, compatibility thresholds) are configured via a simple text-based config file or within Unity's Inspector, allowing researchers to adjust NEAT's behavior without modifying source code.

C. Scenario and Environment Configuration

a) 2.5D Tile-Based World: We adopt a tile-based approach in an isometric style to reduce computational overhead and simplify resource management. Each tile is low-resolution, making it memory efficient even for large environments. Modifying the appearance or properties of a single tile can automatically propagate changes across all identical tiles, improving both performance and workflow when updating the environment.

b) Heatmap-Driven Resource Spawning: Users can import or generate grayscale heatmap textures that define resource distribution. Each tile spawns resources (or creatures) with a probability derived from the texture's pixel intensity. A maximum spawn limit and adjustable spawn interval further control the rate at which new objects appear, enabling varied population dynamics and experiments (e.g., scarcity vs. abundance).

c) Creature Vision and Field of View: We provide adjustable *FOV* parameters—number of vision rays, angle of spread, maximum distance, and detectable layers—so creatures can perceive their environment more or less effectively. The platform employs raycasts to simulate line-of-sight detection. This is configurable via the Inspector, allowing for quick experimentation with different perception setups (e.g., narrower vision for predator simulations vs. wide vision for social/cooperative tasks).

d) GameManager for Configuration: All AI- and scenario-specific settings are centralized in a `GameManager` object within Unity. Researchers select which AI model to use (e.g., NEAT or Unity ML-Agents), define population size and number of generations, and specify how fitness is computed. This setup mirrors the workflow of established reinforcement learning plugins, making it accessible to those familiar with Unity's editor design.

D. User Workflow

1) **Map and Resource Setup:** Design or import a tile-based map, apply custom tile sprites, and generate heatmaps for resource distribution.
2) **NEAT Configuration:** Using a config file or Unity's Inspector, define key parameters (e.g., population size, compatibility thresholds, mutation rates) for the evolutionary process.
3) **Scenario Definition:** Implement or select a fitness function script. For instance, a survival scenario may reward agents for longevity, while a resource-collection scenario rewards gathering or delivering items.
4) **Simulation Execution:** Run the simulation in Unity. Agents spawn, interact, and gather fitness data in real time.
5) **Analysis and Visualization:** Monitor real-time charts for population size, best fitness, or any custom metric. Swap to an individual creature view to observe specific behaviors, neural network structure, or performance stats.

E. Preliminary Validation: XOR and Sine Approximation

Before tackling large-scale simulations (e.g., population dynamics, cooperative behaviors), we tested the correctness and learning capacity of our NEAT integration using two standard tasks.

a) XOR Gate: We used the four binary input combinations $\{(0,0),(0,1),(1,0),(1,1)\}$ as the training set. Each agent's fitness ranges from 0 to 4, determined by the number of correct outputs (i.e., matching the XOR ground truth).

- **Population Size:** 500
- **Fastest Convergence by Generations:** 11
- **Average Convergence by Generations:** 29

b) Sine Function Approximation: We sampled 125 points between $-\pi$ and π to train agents to approximate the sine function. The fitness function (Equation 1) combines mean squared error, maximum error, and a complexity penalty to encourage simpler networks.

$$fitness = 0.6 \cdot mseFitness + 0.3 \cdot maxErrorFitness \quad (1)$$
$$+ 0.1 \cdot complexityPenalty. \quad (2)$$

Where:

- $mseFitness = e^{-2.0 \cdot (totalError / TestPoints.length)}$
- $maxErrorFitness = e^{-3.0 \cdot (maxError)}$
- $$complexityPenalty = \frac{0.1}{1.0 + 0.1(genome.nodes.count) + 0.05(genome.connections.count)} \quad (3)$$

NEAT configurations:

- **Population Size:** 500
- **Maximum Generations:** 5000 (though it often converged or stagnated earlier, around 100)
- **Least Error:** 38
- **Average Loss Across Trials:** 43

These preliminary results confirm that our NEAT implementation evolves solutions to simple tasks, verifying that the underlying mutation, crossover, and speciation processes function correctly. In the following sections, we demonstrate how the platform scales to more complex scenarios (e.g., exploring resource disparity, cooperative behavior, and aggression) and discuss the resulting emergent behaviors.

III. Experimental Setup

In this section, we describe three experimental scenarios designed to explore different aspects of evolutionary behavior within our NEAT-based simulation platform. Each scenario leverages the modular components and configuration options detailed in Section II, including tile-based world design, heatmap-driven resource spawning, and customizable vision. Each scenario utilizes the same fundamental AI system but encourages different survival strategies based on environmental constraints and agent interactions.

A. Common Agent Inputs

In all scenarios, each agent's neural network receives a standardized set of input features:

- **Agent Position:** The agent's current (x, y) location on the 2.5D tile-based map.
- **Agent Health:** A normalized health or energy value indicating the agent's current vitality.
- **Information on the Three Nearest Objects:** For each of the three closest objects or agents, we pass:
  1) *Relative Position:* The offset in (x, y) from the agent to the object.
  2) *Object Type:* A categorical or encoded input (e.g., resource node, ally, enemy).
  3) *Power Level:* If the object is another agent, how strong it is (e.g., an estimated attack or defense rating).

B. Case 1: Evolutionary Advantage of Violence

The first scenario examines whether violent or aggressive behaviors confer a distinct evolutionary advantage. Here, *violence* is operationalized as an agent's ability to attack or eliminate competitors.

a) Environment.: A moderately resource-scarce tile-based map is generated to increase competitive pressure. Heatmaps restrict resources to scattered patches, forcing agents to interact frequently.

b) Agent Interactions.: Each agent can perform basic actions, including:

- **Movement:** Navigate the 2.5D map to locate resources or opponents.
- **Attack:** Inflict damage on nearby agents within a certain FOV and distance.
- **Consume Resources:** Restore energy or health from resource nodes.

c) Fitness Function.: Agents are rewarded for:

- **Survival Duration:** Staying alive longer yields incremental fitness.
- **Population size:** Higher population of allies are rewarded.
- **Resource Collection (Minimal Reward):** Gathering resources provides a small fitness increment but is secondary to survival in this particular scenario.

A higher weight on staying alive in a resource-deprived world provides an evolutionary incentive to engage in violence, testing whether violent strategies indeed outcompete more passive behaviors.

C. Case 2: Herd Mentality and Cooperation

The second scenario investigates how cooperative or "herding" behaviors might evolve and whether such behaviors provide a survival advantage.

a) Environment.: We use a larger tile-based map with mild resource abundance. Heatmaps place resources in clusters that encourage group gathering (e.g., fruit or water supplies in specific patches).

b) Fitness Function.: Agents are rewarded for:

- **Individual Survival:** Each agent maintains a baseline fitness for staying alive.
- **Survival of Group Members:** Agents gain a partial fitness reward for each allied agent that survives.
- **Shared Resource Benefit:** Resources are spread in pockets with some small regions containing a large amount of resources to motivate grouping.

This design tests whether cooperation emerges as an advantageous strategy, potentially outcompeting lone-wolf or aggressive approaches.

D. Case 3: Resource Disparity and Population Dynamics

The third scenario focuses on how differences in resource availability affect evolution, population size, and density. Unlike the prior two scenarios, which emphasize behavioral traits (violence vs. cooperation), this setup explores environmental pressures.

a) Environment.: Multiple regions of the tile-based map are configured with distinct heatmap parameters:

- **High-Resource Regions:** Dense clusters of resources with frequent respawns, for Species 1.
- **Low-Resource Regions:** Sparse resource nodes that regenerate slowly, for Species 2.

b) Agent Interactions.: While agents can still engage in aggression or cooperation, the primary test is how the species as a whole disperse, migrate, or cluster in response to resource disparity.

c) Fitness Function.:

- **Survival Duration:** Staying alive longer yields incremental fitness, which motivates agents to find resources to survive for longer.
- **Population size:** Higher population of allies are rewarded.
- **Migration/Exploration Reward:** Some fitness for exploring new regions, encouraging dispersion.

E. Stopping Criteria (All Cases).

In each experiment, the simulation may run until:

- A fixed number of generations is reached (N=500).
- A specified performance threshold is achieved (e.g., average fitness plateau).
- A manual stop by the user for observation or analysis.

By examining these scenarios, we aim to reveal whether aggression, cooperation, or adaptive resource-driven strategies confer the highest evolutionary advantage under varying environmental pressures. The results, which we present in subsequent sections, will demonstrate how the platform supports diverse research questions and validates its core features.

IV. Results

In this section, we present evidence that our platform successfully implements and demonstrates evolutionary behaviors in multiple contexts. First, we validate the correctness of our *NeuroEvolution of Augmenting Topologies* (NEAT) integration on two benchmark tasks: XOR and Sine Approximation. We then summarize the outcomes of the three scenarios introduced in Section III, highlighting how minimal changes in configuration yield distinct evolutionary dynamics. While these scenarios are not the primary focus, they exemplify the platform's capability for rapid prototyping and robust data collection.

A. Preliminary Validation

a) XOR Task.: To verify the basic functioning of NEAT within Unity, we trained agents on the four standard XOR input-output pairs, awarding fitness points for correct classifications. On average, 29 generations, the best-performing agent achieved a fitness of 4 (of 4 cases). The fastest convergence happened in 11 Generations, indicating that the system consistently evolved accurate solutions.

TABLE I
XOR Task Results

Metric	Best	Average
Generations to Converge	11	29
Fitness	4	4

b) Sine Function Task.: Next, we tested agents on approximating the sine function over 125 sample points from $-\pi$ to π. The fitness combined mean squared error, maximum error, and a complexity penalty (see Equation 1). After 169 generations, the best agent yielded a fitness of 0.5691, closely matching the ground truth curve with minimal network growth.

TABLE II
Sine Approximation Results

Metric	Best	Average
Fitness	0.5691	0.5570
Generations	169	180.8

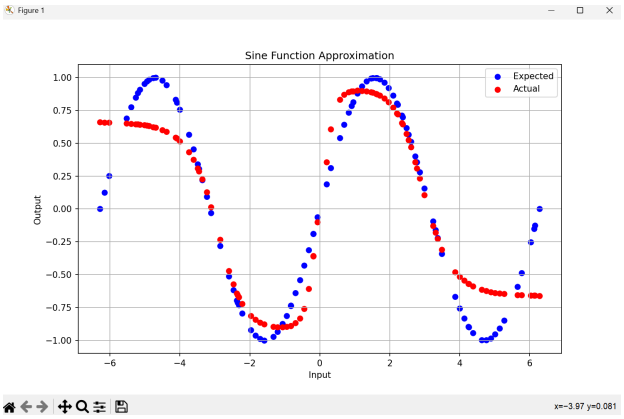

Fig. 1. Agent output vs. actual sine values

Together, these tasks confirm that our NEAT algorithm evolves solutions for classic benchmarks, verifying both the mutation/crossover pipeline and our fitness evaluation process.

B. Scenario Findings

Here, we briefly summarize results from the three scenarios described in Section III. We emphasize how simple parameter changes in our platform facilitate a variety of evolutionary experiments.

1) Case 1: Evolutionary Advantage of Violence: Using a resource-scarce environment with moderate aggression incentives, agents rapidly discovered offensive strategies. The highest aggression levels correlated with slightly higher survival times, suggesting violence can be favored when resources are sparse.

2) Case 2: Herd Mentality and Cooperation: In a more resource-abundant map, cooperative or "herding" behaviors emerged in many trials. Agents that formed clusters increased their collective survival rates and collectively shared resources. Only minor configuration changes were needed (e.g., adjusting the fitness function to reward group survival). Results confirm the platform's flexibility in encouraging cooperative strategies under different reward structures.

3) Case 3: Resource Disparity and Population Dynamics: By varying resource availability across different map regions, we observed distinct migration and clustering patterns. Some populations thrived in high-resource zones, while others specialized in exploration, showcasing adaptive divergence driven solely by parameter tweaks to spawn rates and heatmap distributions.

C. Qualitative Observations

Across all scenarios, we recorded emergent behaviors with minimal manual scripting, indicating the efficacy of our NEAT-driven evolution. In the Violence setup, highly aggressive lineages increased in fitness when resources were scarce, while in the Cooperation scenario, group behaviors led to stable population growth in resource-rich zones. Further, minimal adjustments—such as toggling the Heatmap spawn distribution or adjusting the survival reward—drastically altered population dynamics. This ease of reconfiguration demonstrates how our platform supports rapid hypothesis testing without extensive code modifications.

D. Summary of Results

Overall, these experiments validate that:

- The NEAT algorithm implementation functions correctly (as evidenced by XOR and Sine benchmarks).
- Simple scenario parameter changes enable a broad range of emergent evolutionary behaviors, from aggression to cooperation.
- Statistical information (e.g., fitness scores, population trends) is automatically logged, aiding quick analysis.
- Researchers can set up new experiments with minimal effort, underscoring the platform's **ease of use** for diverse evolutionary simulations.

In the following section, we discuss our conclusions and highlight potential future improvements to further expand the platform's capabilities.

V. Conclusion

In this paper, we presented a modular and extensible platform for evolutionary AI simulations using the NeuroEvolution of Augmenting Topologies (NEAT) algorithm in Unity. By integrating real-time visualization tools, a 2.5D tile-based environment, and highly configurable parameters, the platform lowers barriers for researchers interested in evolving complex agent behaviors. We validated the platform's correctness through standard tasks (XOR and Sine approximation) and demonstrated its versatility with three experimental scenarios: (1) Evolutionary Advantage of Violence, (2) Herd Mentality and Cooperation, and (3) Resource Disparity and Population Dynamics. Minimal changes in configuration led to notably different emergent strategies, underscoring the effectiveness and convenience of this solution.

Looking ahead, we plan to expand the platform to accommodate more nuanced interactions and richer agent behaviors. Possible extensions include integrating hierarchical multi-agent models, introducing advanced resource mechanics (e.g., limited replenishment or specialized item crafting), and further refining neural architectures to incorporate modern neuroevolution techniques. Another promising direction is to combine NEAT with reinforcement learning or other machine learning methods, enabling agents to switch or blend strategies dynamically based on environmental feedback.

Several challenges remain, particularly in optimizing performance for large-scale simulations and ensuring reproducibility across different hardware configurations. Additionally, evaluating long-term evolutionary stability may require more sophisticated statistical tracking. Nonetheless, this work lays the foundation for a flexible, user-friendly platform that streamlines evolutionary AI experimentation in visually rich, interactive domains. By continuing to refine and build upon these capabilities, we hope to foster a powerful tool for researchers, educators, and developers exploring the frontiers of evolutionary and emergent intelligence.

REFERENCES

[1] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–668, 2005.

[3] J. Ericksen, M. E. Moses, and S. Forrest, "Automatically evolving a general controller for robot swarms," in *Proc. of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1–8.

[4] J. E. Auerbach and J. C. Bongard, "Evolving complete robots with cppn-neat: The utility of recurrent connections," in *Genetic and Evolutionary Computation Conference (GECCO 2011)*. ACM, 2011, pp. 1475–1482.

[5] E. J. Kim and R. E. Perez, "Neuroevolutionary control for autonomous soaring," *Aerospace*, vol. 8, no. 9, p. 267, 2021.

[6] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proc. of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009, pp. 241–248.

[7] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[8] K. Kovalský and G. Palamas, "Neuroevolution vs reinforcement learning for training non-player characters in games: The case of a self driving car," in *Intelligent Technologies for Interactive Entertainment (INTETAIN 2021)*. Springer, 2021.