SYNTIA – Systems and Technologies in Artificial Intelligence Applications.
Volume 1 Issue 1 (2025). Page: 17-24.

Implementation of Neural Networks for Car Navigation System Development in a 2D Car Game Simulation Using Genetic Algorithms

Endi Wicaksono[1*], Yuri Yudhaswana Joefrie[2]

[1] *Informatics Engineering Study Program, Faculty of Engineering, Universitas Tadulako, Palu, Indonesia.*
[2] *Department of Information Technology, Faculty of Engineering, Universitas Tadulako, Palu, Indonesia.*

**Corresponding author: endiwicaksono@gmail.com*

Abstract: Cars in a racing game can be driven by a small neural network whose weights are evolved by a genetic algorithm instead of being programmed or trained on labeled data. This study documents such a system built in Unity: each car reads five distance sensors, passes them through a 5–4–3–2 network (47 evolved values), and is evolved with a genetic algorithm that keeps the two best of 20 cars as parents. We re-examine the evaluation reported for it. Ten of eleven functional tests passed; the network and sensor display failed. On three tracks the share of cars finishing rose from 30%, 20%, and 10% in generations 1–10 to 70%, 50%, and 30% in generations 41–50, a growth of 1.00, 0.75, and 0.50 percentage points per generation that is almost perfectly linear ($R^2 \geq 0.98$), and the open-ended window "51+" reaches 100%, 60%, and 40%. A generation took about 15 s and showed no trend over five generations. Mean FPS fell by 0.20 (high-end PC) and 0.32 (low-end PC) per added car, and only the low-end PC at 100 cars reached the 30 FPS limit. Several reported values are mutually inconsistent, including identical memory columns for the two PCs that the accompanying chart contradicts. The system shows that the pipeline works on the tracks used for training; evidence of learning that generalizes, and of its cost, needs repeated runs, raw logs, and a held-out track.

Keywords : neural network; genetic algorithm; neuroevolution; autonomous car; 2D game simulation; Unity.

Received: November 03, 2025 | Revised: November 27, 2025 | Accepted: December 17, 2025 | Published: December 31, 2025

1. Introduction

Computer-controlled opponents decide much of how a racing game feels: they must follow the track, avoid walls, and react to what lies ahead. Neuroevolution, the evolution of neural-network controllers by an evolutionary algorithm, is a long-standing way to obtain such behavior without hand-written rules and without a training set, and it has been applied across games [1]. A feed-forward network maps sensor readings to actions [2]; instead of backpropagation, the genetic algorithm (GA) searches the space of weights using a score earned by playing. This differs from learning a value function from pixels by reinforcement learning, which has reached human-level play on many video games [3].

Combining GAs with neural networks has been studied for decades, from optimizing the connections of a network [4] to evolving weights, topology, and learning rules [5], [6]. Closer to this study, related reports cover the combination of neural networks and genetic algorithms in general [7], a neural-network lane recognizer for a self-driving car prototype [8], a GA for a Ms. Pacman agent [9], and the NEAT algorithm applied to a board game [10]. For a student or small team building a game, however, the practical questions are different: does the pipeline run, does it improve with generations, and what does it cost on ordinary hardware? A published evaluation of such a system is only useful if its numbers can be traced to runs, and that is often not shown.

SYNTIA – Systems and Technologies in Artificial Intelligence Applications is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

This article documents a 2D car-navigation simulation in Unity that uses a neural network and a GA, and re-examines the evaluation reported for it. We ask three questions. (RQ1) Do the functional components of the simulation work as specified? (RQ2) How does the share of cars that finish a track change with generations on tracks of different difficulty, and how strongly do the reported numbers support a claim of learning? (RQ3) How does the computational cost scale with the number of cars on two machines, and which reported values are consistent? The contributions are a complete description of the controller and GA with the number of evolved values, a quantified reading of the accuracy and scalability results with linear trends and sampling uncertainty, and a list of internal inconsistencies in the source evaluation together with the data needed to resolve them. Section 2 describes the system and the evaluation, Section 3 reports and discusses the results, and Section 4 concludes.

2. Method

2.1. Platform and materials

The simulation is written in C# in the Unity game engine with Visual Studio; the neural network and the GA are implemented by hand, without a machine-learning library. Development and tests used a laptop with an Intel Core i5-11400H (2.70 GHz, 12 logical CPUs), 16 GB of RAM, and an NVIDIA GeForce RTX 3050 laptop GPU. Most of the project material, including the track pieces, car sprites, and the structure of the evolution scripts, was taken from a public Unity project by Samuel Arzt. The source does not give the project's licence, the parts taken beyond those listed, or the Unity version.

2.2. Development process

The work followed an iterative cycle of requirements analysis, design, prototype, and testing, in the manner of rapid game prototyping [11]: if the prototype did not work, the cycle returned to requirements. Fig. 1 shows the controller and the evolutionary loop that came out of it.

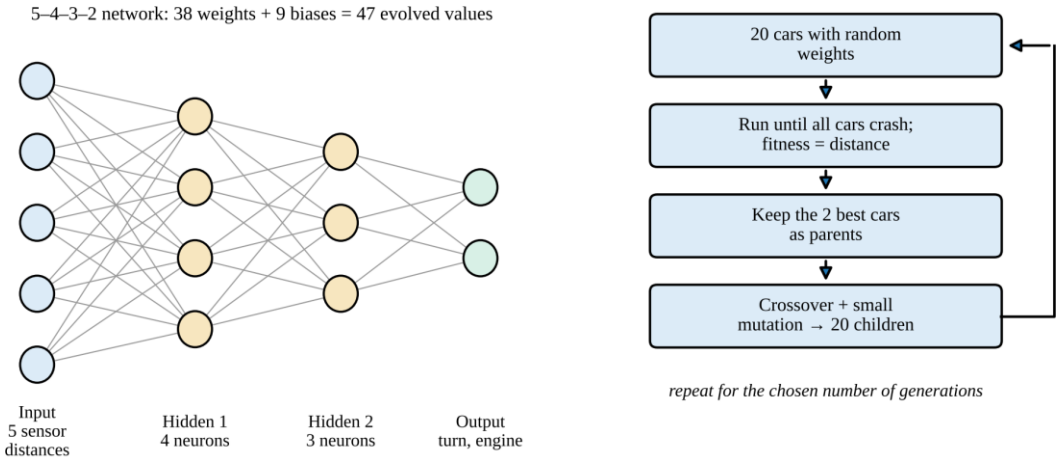


Figure 1. Controller (left) and evolutionary loop (right) of the simulation.

2.3. Controller

Each car carries five forward-facing virtual sensors, implemented by ray casting, whose distance readings $x \in R^5$ are normalized and fed to a network with two hidden layers of four and three neurons and two outputs, the steering (turn) and the engine command:

$$h^{(1)} = \varphi\left(W^{(1)}x + b^{(1)}\right), \quad h^{(2)} = \varphi\left(W^{(2)}h^{(1)} + b^{(2)}\right), \quad y = \varphi\left(W^{(3)}h^{(2)} + b^{(3)}\right) \tag{1}$$

where $W^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l and φ is the activation function. The source names only an "activation function" and gives neither its form nor the scaling of the outputs. A hyperbolic tangent in the output layer is the usual choice for steering and engine commands, since it keeps both outputs

in [−1, 1], from full left to full right and from full reverse to full throttle. With layer sizes n = (5, 4, 3, 2) the number of evolved values is

$$P = \sum_{l=1}^{3}(n_{l-1} + 1)\, n_l = (5 + 1) \cdot 4 + (4 + 1) \cdot 3 + (3 + 1) \cdot 2 = 47 \quad (2)$$

that is 38 weights and 9 biases. A search space of 47 real values is small, which matters for the discussion of selection pressure in Section 3.3.

2.4. Genetic algorithm

A generation starts with 20 cars with random weights [12]. All cars run until they hit a wall; the fitness of a car is the distance it covered along the track, measured through checkpoints. The two cars with the highest fitness become the parents. Their values are combined by crossover into 20 children, small random perturbations are added as mutation, and the children replace the population. The loop repeats for the chosen number of generations. Taking only the two best individuals is a form of truncation selection with the strongest possible pressure, unlike fitness-proportionate selection, which keeps weaker individuals in play [13], [14]. The source gives single-point crossover and "small noise" only as examples ("e.g."). It does not state the crossover operator actually used, the mutation probability and magnitude, whether the parents survive into the next generation, the stopping rule, or the random seeds.

2.5. Evaluation design

Functional test. Eleven cases were run once each, from the setup of tracks and cars to evolution and display [15]. *Performance.* Time per generation and CPU use were recorded over five generations. *Accuracy.* The accuracy of a generation window is the share of cars that reach the end of the track without hitting a wall; it was reported for ten-generation windows and an open-ended window "51+" on three tracks of increasing difficulty, starting from 50 random cars. *Scalability.* The number of cars was raised from 10 to 100 on a "high-end" and a "low-end" PC, recording mean and minimum FPS, CPU, GPU, and memory; 30 FPS was the lowest acceptable mean.

We reanalyzed the reported tables as follows. Accuracy trends were summarized by least-squares slope in percentage points per generation against the window midpoints (5.5 to 45.5) and the coefficient of determination R^2, excluding the open-ended window. Scalability was summarized by the slope per added car. Because the number of cars behind each accuracy value is not stated, we illustrate the sampling uncertainty with the Wilson interval for a proportion [16] for an assumed single generation of n = 50 cars:

$$\frac{\hat{p}+\frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}+\frac{z^2}{4n^2}}}{1+\frac{z^2}{n}}, \qquad z = 1.96 \quad (3)$$

where $\hat{p}$ is the observed share. Values in Tables 3 and 4 are those printed in the source; the generation timing and CPU values are read from the source charts (± one unit) because no table is given. The source provides no raw logs (per generation: fitness of each car, number of finishers, time, CPU) and does not say how many independent runs lie behind each reported value.

3. Results and Discussion

3.1. Implemented simulation

Fig. 2 shows the simulation. The track is a winding corridor with checkpoints (red dots); four tracks of different difficulty were prepared. Cars are drawn in red, the best car of the generation in green, and the second best in beige; the sensors appear as crosses ahead of the car, and the network of the best car is drawn in the corner with line thickness and color showing the size and sign of each weight. The interface also shows the turn and engine outputs, the evaluation value, and the generation counter.

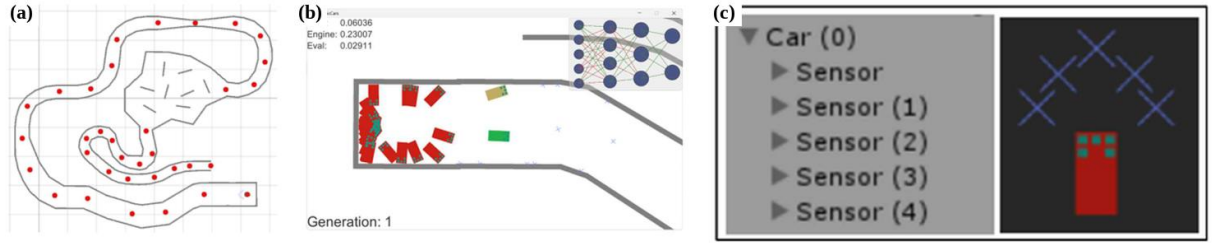


Figure 2. The simulation: (a) one of the tracks with checkpoints; (b) generation 1 with the network panel; (c) the five front sensors of a car.

3.2. Functional test (RQ1)

Ten of the eleven cases passed (Table 1). The failed case is the display of the network and sensors in the interface; the source does not describe the failure, although Fig. 2 shows both. Together with the conclusion of the source that the system "works well", this means the failure is open and should be described and retested. The cases are single runs with valid input and check that components exist and respond, not that the evolved behavior is good, so a pass of the evolution cases means that selection, crossover, and mutation ran, not that they improved the cars.

Table 1. Functional test of the simulation (one run per case).

No	Case	Input or parameter	Observed	Result
1	Track and car setup	4 tracks, 20 cars	Initialized correctly	Pass
2	Initial generation	Network and sensors on every car	All cars initialized	Pass
3	Sensor operation	Distance to obstacles	Measured with high accuracy	Pass
4	Network decision	Decision from sensor input	Relevant decisions	Pass
5	Car movement	Movement follows the network	As expected	Pass
6	Collision detection	Car and wall	All collisions detected; car stopped	Pass
7	Fitness evaluation	Progress along the track	Computed from distance	Pass
8	Selection and crossover	Two best cars chosen	Best cars selected	Pass
9	Mutation	Applied to the new generation	Applied	Pass
10	New generation	Cars placed on the track	Initialized	Pass
11	Network and sensor display	Shown correctly in the interface	Not shown correctly	Fail

3.3. Learning across generations (RQ2)

Table 2 and Fig. 3 give the share of finishing cars. It rises monotonically on all three tracks, and the ordering follows difficulty: in generations 41–50 it is 70% on the easy track, 50% on the medium one, and 30% on the hard one, against 30%, 20%, and 10% in generations 1–10. The first five windows grow by 1.00, 0.75, and 0.50 percentage points per generation on tracks 1, 2, and 3 ($R^2 = 0.98, 0.99, 1.00$).

Table 2. Share of cars finishing the track (%) by generation window, as reported, and the least-squares slope of the first five windows.

Generations	Track 1 (easy)	Track 2 (medium)	Track 3 (hard)
1–10	30	20	10
11–20	40	25	15
21–30	55	35	20
31–40	60	40	25
41–50	70	50	30
51+	100	60	40
Slope, windows 1–5 (points per generation)	1.00	0.75	0.50

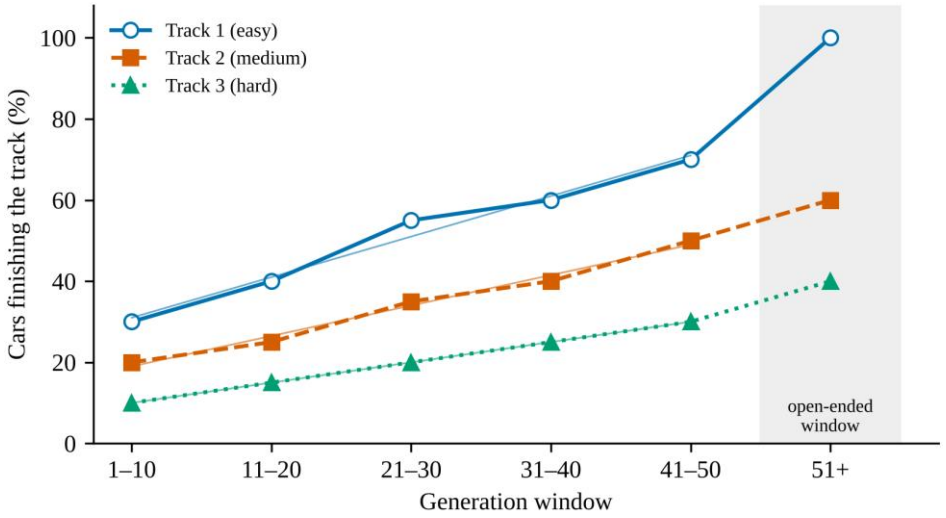


Figure 3. Share of cars finishing the track by generation window. Thin lines: least-squares trend of the first five windows.

Five points limit what these numbers can show. First, the growth is too regular: the slopes stand in the exact ratio 4 : 3 : 2 and the fits are almost perfect, whereas learning curves of a stochastic search are normally noisy and concave. This does not show that the values are wrong, since they may be averages over many runs, but it is what rounded or smoothed values would also look like, and without the raw logs it cannot be told apart. Second, the window "51+" has no upper limit and cannot be compared with ten-generation windows; track 1 reaches 100% there, a jump of 30 points against steps of 5 to 15 before, which suggests a different aggregation or a different regime. Third, the number of cars behind each value is unknown. If one value stood for one generation of 50 cars, a single car would be 2 points and the 95% Wilson intervals would be wide: 30% (15 cars) has an interval of 19–44%, 20% of 11–33%, and 10% of 4–21%, so the intervals of the three tracks in the first window would largely overlap. Fourth, window averages hide generation 1, the one point where the cars still have random weights, and a mean finishing share of 30% in the first ten generations means that evolution already acts strongly inside the first window.

Fifth and most important, the cars are scored on the track they are evolved on. A rising share of finishers then shows that the population converges on a solution for that track, not that the system "adapts to different tracks and obstacles", as the source concludes; no test puts an evolved population on an unseen track. Convergence is also what this algorithm is built for. With 47 evolved values, only two parents, and 20 near-copies as children, diversity falls quickly, which fits a quick rise on the easy track and a plateau on the hard one, but this is a hypothesis: it would be tested by comparing with a larger parent pool or fitness-proportionate selection [13], [14] and by logging population diversity. This answers RQ2: the pipeline improves the finishing share on the training tracks, but the reported figures do not allow an estimate of how fast, how reliably, or how well it transfers.

3.4. Computational cost (RQ3)

Over five generations a generation took about 15 s (14–16 s) with no trend (slope −0.2 s per generation, $p = 0.45$), and mean CPU use was about 33% with peaks of 65–75% (Table 3). The old version of this article showed a different timing chart, with times of 0.4–1.0 s that grew linearly over 50 generations; it is not in the source and is not used here. Table 3 uses only the series printed in the source. A flat time per generation is plausible when a generation ends as soon as all cars have crashed, but then time depends on how long the best car survives and should rise as cars improve, so the flat series needs an explanation.

Table 3. Execution time and CPU use for five generations (values read from the source charts, ±1).

Generation	Time per generation (s)	Mean CPU (%)	Peak CPU (%)
1	16	30	65
2	15	32	70
3	14	34	75
4	15	35	70
5	15	33	72

Table 4 gives the scalability results for the two machines. Mean FPS falls linearly with the number of cars, by 0.20 FPS per car on the high-end PC and 0.32 on the low-end PC; CPU use grows by 0.26 percentage points per car on both, and memory by about 3.9 MB per car. The high-end PC stays at 95 FPS or more up to 100 cars. The low-end PC reaches exactly 30 FPS in the mean and 25 FPS at the minimum at 100 cars, which is not above the stated limit; the source marks 100 cars as not scalable in its yes/no table, which agrees with the limit if that table refers to the low-end PC (the source does not say). The source lists the time of one network decision and of the GA step per generation as measured quantities but reports no values.

Table 4. Scalability as reported: mean / minimum FPS, CPU (%), GPU (%), and memory (MB) for the high-end and the low-end PC.

Cars	FPS HE	CPU HE	GPU HE	Mem HE	FPS LE	CPU LE	GPU LE	Mem LE
10	115 / 105	5	5	150	60 / 55	20	25	150
20	110 / 100	10	10	180	55 / 50	25	30	180
30	105 / 95	15	20	250	50 / 45	30	35	250
40	100 / 90	20	25	350	45 / 40	35	40	350
100	95 / 85	30	35	500	30 / 25	45	50	500

HE = high-end PC, LE = low-end PC. The hardware of the two PCs is not specified in the source.

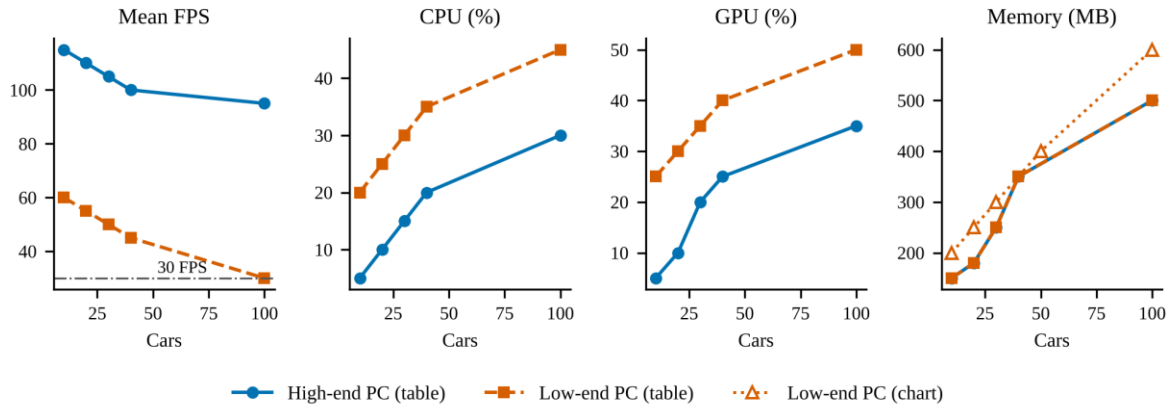


Figure 4. Scalability by number of cars on both PCs. In the memory panel the low-end series is plotted twice: from Table 4 (identical to the high-end series) and read from the source chart.

Several of these values are inconsistent. (i) The memory column of the low-end PC is identical to that of the high-end PC (150–500 MB), whereas the source chart shows about 200, 250, 300, 400, and 600 MB for the low-end PC (Fig. 4). (ii) The table lists 10, 20, 30, 40, and 100 cars, but the chart plots 10, 20, 30, 50, and 100; the text mentions 20, 50, 100, and 200 cars, and the yes/no table 10, 20, 30, 50, and 100. (iii) The minimum FPS is exactly the mean minus 10 at every car count on the high-end PC and minus 5 on the low-end PC, and the low-end CPU use equals the high-end value plus exactly 15 points at every count. Regularities of this kind arise when values are rounded, smoothed, or filled in by hand, and a profiler seldom produces them. (iv) The configurations of the "high-end" and "low-end" PCs are not given. We therefore treat the scalability results as indicative only. Table 4 and Fig. 4 reproduce the values as printed in the source; profiler logs, the hardware of both PCs, and one common set of car counts would be needed to replace them.

3.5. Limitations

We did not run the simulation, and the code was not available to us, so everything rests on the source tables, charts, and screenshots. The evaluation has no repeated runs or random seeds, no variance, no comparison with a baseline, such as random search, a hand-written steering rule, or a different GA setting, and no held-out track. Fitness, accuracy, and the finishing criterion are described in words only. The study does not compare against neuroevolution of topologies [10] or reinforcement learning [3], and the tracks are few. These limits restrict the claims to what the pipeline demonstrably does: a GA with two-parent truncation selection evolves a 47-value controller that completes the training tracks to a growing degree.

4. Conclusion

The simulation shows that a 5–4–3–2 neural network evolved by a genetic algorithm can drive cars around 2D tracks in Unity, and ten of eleven functional tests passed. The reported finishing shares rise with generations on three tracks of increasing difficulty, but they grow almost exactly linearly, the number of cars and runs behind them is unknown, the cars are scored on the tracks they evolve on, and the scalability tables contain regularities and contradictions that raw logs would have to resolve. The claims of adaptation to different conditions and of efficient scaling are therefore not yet supported.

The next study should publish the code and the logs, report mean and variance over repeated runs with fixed seeds, add a held-out track, compare two-parent truncation with a larger parent pool and with fitness-proportionate selection, and measure time per generation and per network decision with a profiler on documented hardware.

AI Disclosure Statement

The authors used Generative AI and AI-Assisted Technologies throughout the preparation of this manuscript: to draft and revise portions of the text, to write and run the scripts behind the reanalysis of the accuracy and scalability results, to generate the figures, and to format the document from the authors' own research data. The authors verified the scripts and their outputs, reviewed the resulting text, cross-checked every reference against its original source, and accept full responsibility for the content of this publication.

Author Contributions Statement

Endi Wicaksono: Conceptualization, Methodology, Software, Investigation, Data Curation, Writing.
Yuri Yudhaswana Joefrie: Supervision, Methodology, Formal Analysis, Writing – Review and Editing.

References

[1] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 1, pp. 25–41, 2017, doi: 10.1109/TCIAIG.2015.2494596.

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.

[3] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[4] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990, doi: 10.1016/0167-8191(90)90086-O.

[5] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999, doi: 10.1109/5.784219.

[6] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, 2008, doi: 10.1007/s12065-007-0002-4.

[7] R. Mahajan and G. Kaur, "Neural networks using genetic algorithms," *Int. J. Comput. Appl.*, vol. 77, no. 14, pp. 6–11, 2013, doi: 10.5120/13549-1153.

[8] L. R. Manangka, H. Suprijono, and D. Nurcipto, "Pengenalan pola lintasan berbasis neural network pada prototype self-driving car" (in Indonesian), *Elektrika*, vol. 12, no. 2, p. 67, 2020, doi: 10.26623/elektrika.v12i2.2732.

[9] Y. B. Setiawanda, M. K. Rasyid, M. J. Ramadhan, and A. D. Hartanto, "Penerapan algoritma genetika dan algoritma ghost framework pada game Ms. Pacman" (in Indonesian), *Creat. Inf. Technol. J.*, vol. 5, no. 3, p. 174, 2019, doi: 10.24076/citec.2018v5i3.206.

[10] B. A. Qader, K. H. Jihad, and M. R. Baker, "Evolving and training of neural network to play DAMA board game using NEAT algorithm," *Informatica*, vol. 46, no. 5, pp. 29–37, 2022, doi: 10.31449/inf.v46i5.3897.

[11] M. Tjahyadi, A. Sinsuw, V. Tulenan, and S. Sentinuwo, "Prototipe game musik bambu menggunakan engine Unity 3D" (in Indonesian), *J. Tek. Inform.*, vol. 4, no. 2, 2015, doi: 10.35793/jti.4.2.2014.6990.

[12] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimed. Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, 2021, doi: 10.1007/s11042-020-10139-6.

[13] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evol. Comput.*, vol. 4, no. 4, pp. 361–394, 1996, doi: 10.1162/evco.1996.4.4.361.

[14] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Phys. A Stat. Mech. Appl.*, vol. 391, no. 6, pp. 2193–2196, 2012, doi: 10.1016/j.physa.2011.12.004.

[15] S. Nidhra, "Black box and white box testing techniques – A literature review," *Int. J. Embedded Syst. Appl.*, vol. 2, no. 2, pp. 29–50, 2012, doi: 10.5121/ijesa.2012.2204.

[16] L. D. Brown, T. T. Cai, and A. DasGupta, "Interval estimation for a binomial proportion," *Stat. Sci.*, vol. 16, no. 2, pp. 101–133, 2001, doi: 10.1214/ss/1009213286.