A Comparative Analysis of Particle Swarm Optimization Trained Neural Networks and Genetic Programming for Training Snake Playing Game Agents

Rayden Logan Burger
Computer Science
University of Pretoria
Pretoria, South Africa
rayden.logan.viljoen@gmail.com

***Abstract*—This paper presents a comparative analysis of two different techniques for creating game-playing agents for the classic game Snake. The first technique uses a particle swarm optimization (PSO) algorithm to train a neural network (NN). The second technique employs a genetic program (GP) to create three separate yet similar agents, differing in feature input and genetic capabilities. Additionally, the performances of these two techniques will be compared with three other hand-made game-playing agents. The evaluation is based on several metrics including game wins, average score and amount of moves made. The study contributes to the development of effective and efficient game-playing agents using different optimization algorithms.**

***Index Terms*—Artificial Intelligence, Particle Swarm Optimisation, Neural Network, Genetic Program, Game Playing Agent, Snake.**

I. Introduction

In recent years, the realm of Artificial Intelligence (AI) has witnessed remarkable strides, progressing from simplistic rule-based systems to powerful machine-learning techniques capable of tackling multifaceted problems with exceptional finesse [1]. The proliferation of AI has extended its reach into various domains, including robotics, healthcare, finance, and gaming [2] [3] [4]. In particular, AI-driven game-playing agents have gained prominence, serving as testbeds for exploring the capabilities of intelligent systems. These agents are designed to exhibit human-like or superhuman-level gameplay in a variety of video games, thereby challenging and pushing the boundaries of AI research.

A game-playing agent, at its core, is an AI system engineered to interact with a digital environment, analyze the game's state, and make decisions to maximize its chances of success [5]. These agents can be constructed using various methodologies, ranging from handcrafted heuristics to more sophisticated approaches involving machine learning techniques [6] [7] [8]. Game-playing agents have not only revolutionized the gaming industry but have also become vital tools for testing AI algorithms, fine-tuning them, and extracting insights into their performance in dynamic, uncertain environments [7].

This research paper delves into the fascinating world of game playing agents, focusing on the classic game of Snake. Snake is a simple yet intriguing game where a snake navigates a grid, consuming food to grow in length while avoiding self-collision and collisions with the walls. Its simplicity conceals a remarkable level of complexity, making it an excellent playground for testing AI capabilities.

The primary objective of this research paper is to implement and compare different approaches for training snake-playing game agents. Specifically, two distinct methodologies will be explored: a Particle Swarm Optimization trained Neural Networks approach designed by Van Rooyen et al [6] and a Genetic Programming approach designed by Ehlis [8]. Additionally, three handcrafted agents, each representing a different heuristic-based strategy as done by Van Rooyen et al [6] will be compared against the agents. By conducting a comparative analysis of these approaches, valuable insights into the performance, strengths, and weaknesses of each agent type will be extracted. This analysis can contribute to the understanding of the applicability of PSO-trained neural networks and genetic programming in training game-playing agents and the potential for harnessing their power in various other problem domains.

The remainder of this paper is organized as follows: In Section II, an in-depth description of the game of Snake, outlining its rules, challenges, and dynamics is explained. Section III will present the various agents and techniques used in this study, providing insights into their design and operation. Section IV will detail the experimental work, including the evaluation metrics, parameters and setup. Finally, in Section V, a summarization of the findings and conclusions, shedding light on the implications and future directions of using PSO-trained neural networks and genetic programming for training game-playing agents in Snake and beyond will be discussed.

II. The Game of Snake

In this section, the foundational context of the game of Snake, upon which this research paper is based, is established.

Feature	Value	Meaning
1 to 8	1	Cell in (b) corresponding to feature number is a food item
	0.5	Cell in (b) corresponding to feature number is unoccupied
	−0.5	Cell in (b) corresponding to feature number is a body segment
	−1	Cell in (b) corresponding to feature number is a wall
9	1	Food is positioned in a column right of the snake head
	0	Food is positioned in the same column as the snake head
	−1	Food is positioned in a column left of the snake head
10	1	Food is positioned in a row above the snake head
	0	Food is positioned in the same row as the snake head
	−1	Food is positioned in a row below the snake head

Fig. 1: Sensory input from Van Rooyen et al [6]

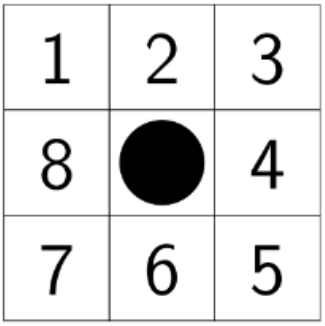

Fig. 2: Positional states around the head of the snake

Subsection A expounds on the core functionalities and applications of the game, along with an exploration of extant versions and modifications. Subsection B delves into prior research endeavors related to the game, offering a comprehensive overview of existing scholarship.

A. Design and Gameplay

Snake is a classic video game, popularized from early Nokia devices [9] that typically involves a single player controlling a snake-like creature on a two-dimensional grid. The objective of the game is to navigate the snake around the grid, consuming food items that appear at random locations. As the snake consumes food, it grows in length, making it increasingly challenging to maneuver without colliding with its own body or the boundaries of the grid. Any form of collision instantly ends the game.

The game mechanics of Snake are relatively simple. The player controls the direction of the snake's movement using arrow keys or similar controls. The snake continuously moves in the direction it is facing until directed otherwise by the player. The grid is typically rectangular and wraps around, allowing the snake to pass from one edge to the opposite edge seamlessly. Van Rooyen et al use a version of the game whereby a back move is possible [6], which instantly ends the game, however, most applications do not include this option.

The standard time interval between successive moves in the game is one second, presenting a challenge for human players who must react swiftly to avoid collisions. In contrast, an autonomous bot encounters no such challenge, as it can execute a move within a fraction of that time.

The player's score typically corresponds to the number of food items eaten. However, the game may also introduce additional objectives, such as achieving a high score within a limited timeframe or avoiding specific obstacles. Van Rooyen et al [6] prioritizes completing a game, whereas other objectives focus on a maximum score or potentially the quickest possible completion of the game.

Over the years, Snake has been subject to numerous variations and enhancements, introducing new features and challenges. These modifications may include obstacles or barriers that hinder the snake's movement, power-ups that grant temporary advantages, different grid layouts, dimensions or shapes, multiplayer modes, and visually enhanced graphics. This research paper focuses on the standard game of Snake where these variations do not come into play.

B. Previous Research

Previous research has explored the use of AI agents to play the game Snake. One study investigated the application of deep reinforcement learning (DRL) techniques to train AI agents for playing Snake [10]. The study utilized the Arcade Learning Environment (ALE) as an evaluation platform and benchmark for training and testing the AI agents. The results showed that the DRL agent achieved high scores and demonstrated the ability to learn effective strategies for playing the game.

Another study focused on developing autonomous agents for playing Snake using DRL too using a model that gets stricter as time goes on [11]. The experimental results showed that the DRL agent outperformed the baseline model and even surpassed human-level performance in playing the game. This research highlights the potential of DRL techniques in training AI agents to excel in complex tasks like playing Snake.

Van Rooyen et al. and Ehlis have independently devised autonomous agents [8] [6]. These agents harness a PSO-trained NN and Genetic programming respectively to construct decision-making models following a training phase. These models rely on a set of features to gather pertinent information about the current state of the game.

Van Rooyen et al. opted for a feature set with considerably fewer sensory inputs compared to Ehlis' approach. Specifically, Van Rooyen's model employs ten distinct feature values, as depicted in Figure 1. The first eight values are designed to perceive immediate surroundings around the agent's head, while the ninth and tenth values are geared towards assessing the global positioning of food items across the board.

In contrast, Ehlis employs a GP approach to formulate the agent model [8]. In this context, the feature set is composed of functions that approximate specific feature values. For example, in Van Rooyen et al. implementation, a value of 1 in feature input 1 can correspond to a function such as "IsFoodTopLeft," which would evaluate to true if a food token is situated in position 1, as illustrated in Figure 2. By extrapolation, a direct mapping of functions from the feature

input described in Figure 1 can be derived to suit the GP function set.

However, it is noteworthy that Ehlis adopts a significantly broader feature space, incorporating functions that extend their influence up to two blocks away from the snake's head, in addition to considering the direction in which the snake is moving. These extended features are distinct from the feature set detailed in Van Rooyen et al.'s implementation. Ehlis' approach is also directional based, meaning a left move is dependent on the direction the snake is moving, i.e. if the snake is moving left, a left move moves the snake down. This dichotomy in feature spaces and movement logic serves as a focal point for the present study, aiming to harmonize the feature sets employed in both implementations and subsequently compare the outcomes.

III. Agents and Techniques

This section aims to Describe the methods and logic of the agents used in this paper. Subsection A explains the handwritten agents that will serve as a benchmark test to compare the autonomous agents designed by Van Rooyen et al and Ehlis. Subsection B gives insight into the method used by Ehlis. Subsection C explains the method used by Van Rooyen et al.

A. Hand Designed Agents

Three novel hand-designed agents were implemented.

1) Randomiser: This agent makes random moves, that do not end the game immediately until no move is available. The idea behind this agent is to create a comparison against chance.

Algorithm 1: Cell Movement Algorithm

Data: Initial state, obstacles, food position
Result: Move to the cell with the lowest non-obstacle density

```
Initialize counters C2, C4, C6, C8 with zero values;
while Game not Over do
    Loop through each counter if If moving to that cell will bring the snake closer to food then
        Counter–;
    else
        Do Nothing;
    Compute density values for cells 2, 4, 6, 8 based on obstacle presence;
    Determine the minimum density among C2, C4, C6, C8;
    if multiple counters have the minimum density then
        Choose a random winner among the tied counters;
    else
        Move to the cell corresponding to the winning counter;
        Update counters based on obstacle presence;
```

2) TowardsLessDense: This agent calculates densities which are defined by Algorithm 1. The obstacle presence is the count of obstacles in that cell position as well as adjacent cells of the head and the cell, i.e. Counter 2 is the count of obstacles in Cells 1, 2 and 3. This agent simulates moving toward food and away from near danger.

3) WallHugger: This agent was defined by Nillson [12]. The agent aims to skirt around obstacles while moving towards food, to avoid getting stuck in definite traps early. The algorithm is defined below:

```
Set a zero-valued counter for cells 2, 4, 6, and 8
if cell 1 is an obstacle or cell 8 is an obstacle then
    Increment counter for cell 2
end
if cell 2 is an obstacle or cell 3 is an obstacle then
    Increment counter for cell 4
end
if cell 4 is an obstacle or cell 5 is an obstacle then
    Increment counter for cell 6
end
if cell 6 is an obstacle or cell 7 is an obstacle then
    Increment counter for cell 8
end
Increment counters of cells that will move the agent closer to food
Move to non-obstacle cell with highest counter, breaking ties randomly
```

Fig. 3: Algorithm for the Wall Hugger agent

B. Genetic Programming

Genetic Programming is a computational methodology rooted in the field of evolutionary algorithms. It encompasses a technique whereby computer programs or mathematical expressions are evolved through the application of principles inspired by natural selection and genetic operations, enabling the automatic solution of complex problems [13]. GP employs a representation of computer programs as a collection of genes, which can be manipulated and combined using genetic operators like mutation and crossover, mirroring the genetic mechanisms observed in biological evolution.

In the realm of GP, a population of computer programs is initially generated randomly, typically using a tree-based structure or other suitable representations. Each program represents a potential solution to the problem at hand. Through iterative generations, the programs undergo processes of evaluation, selection, and modification, thus simulating the concepts of survival of the fittest and reproduction. During the evaluation phase, programs are assessed based on their performance, often measured by a fitness function that quantifies their suitability to solve the problem. In the case of Ehlis' implementation, programs were evaluated on their overall score, aiming to develop programs that achieve a highest score [8].

As the GP algorithm progresses through multiple generations, favorable traits become more prevalent in the population, driving the convergence towards solutions of increasing quality. Through the continuous interplay between evaluation, selection, and genetic operations, GP explores the program space and adapts the programs to fit the problem's requirements. This iterative process continues until a termination criterion, such

as reaching a maximum number of generations or achieving a satisfactory solution, is met.

The application of GP spans various domains, including symbolic regression, classification, optimization, and control systems, among others [13] [14]. It offers several advantages, such as the ability to handle complex and non-linear problems, adaptability to different problem domains, and the potential for discovering novel and creative solutions.

C. Particle Swarm Optimisation

Particle Swarm Optimization (PSO), designed by Kennedy and Eberhart [15] was based upon a simplified social model that represents swarm theory. Like many other probabilistic search algorithms, PSO was designed to mimic a natural phenomenon. A physical analogy can be represented by that of a swarm of birds flocking. In this representation, each bird (which represents a particle in PSO), makes use of its memory as well as the group's findings to deduce its next move. This allows for a semi-random search, using not only a single instance but multiple instances working in parallel to find an optimal solution.

As described by Venter and Sobiesszczanski-Sobieski [16], a typical PSO implementation will look as follows:

Algorithm 2: Particle Swarm Optimization

```
Data: Initialization of particles' positions and
      velocities randomly
Data: Initialization of global best position (gBest)
(1) while termination condition is not met do
(2)     for each particle i do
(3)         Update particle's velocity;
(4)         Update particle's position;
(5)         if fitness(position_i) < fitness(pBest_i) then
(6)             Update particle's best position:
                pBest_i ← position_i;
(7)         if fitness(position_i) < fitness(gBest) then
(8)             Update global best position:
                gBest ← position_i;
```

The update of the velocity of a particle works as follows:

$$\begin{aligned} v_i(t+1) = w \cdot v_i(t) + c1 \cdot rand1 \cdot (pbest_i - x_i(t)) \\ + c2 \cdot rand2 \cdot (gbest - x_i(t)) \end{aligned} \tag{1}$$

where t denotes the specific time instance. i denotes the particle under consideration. The coefficients c1 and c2 correspond to the cognitive and social factors, respectively. These coefficients exert influence on the velocity change of the particle's individual best and the swarm's global best. Typically, the values of c1 and c2 impact the rate of change. Random values, ranging from 0 to 1, are introduced to incorporate an element of exploration and exploitation. By leveraging these values, the equation compels the particle to explore uncharted regions of the search space or potentially exploit well-explored areas in pursuit of more optimal solutions. The parameter w signifies the inertia weight, a hyperparameter that undergoes tuning specific to the problem at hand. This weight governs the extent to which the previous velocity influences the new velocity.

The particle's position gets updated as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2}$$

Guo-Wei Xu proposes a mathematical way to tune the parameters of the PSO implementation [17], allowing for one to further increase the proficiency of PSO. While many parameter tuning methods do exist, the process defined by Van Rooyen et al [6] will likely be used, and if time permits, a more in-depth process can be added.

PSO has emerged as a versatile and powerful optimization algorithm with diverse applications in various scientific and engineering domains. The inherent nature of PSO, inspired by the collective behavior of bird flocking and fish schooling, lends itself to solving complex optimization problems efficiently. PSO has found utility in fields such as engineering design, telecommunications, finance, image processing, and machine learning, to name a few [18] [19] [20] [21]. Its ability to explore and exploit solution spaces effectively, coupled with its simplicity and ease of implementation, has positioned PSO as a go-to tool for tackling problems characterized by high-dimensional parameter spaces, hence it serves as a valid means of training a neural network as a Snake-playing agent.

D. Neural Networks

C. Bishop defines an Artificial Neural Network as a non-linear mathematical function that transforms a set of input variables into a set of output variables [22]. This model originates from the middle 1900s as an artificial replica of the way the human brain processes information. The theory differs from traditional computing programs whereby rules are defined and used to make decisions. Instead, the NN uses a training process where the model processes examples of the solved problem in order to reform itself into an optimal model for the problem. This process started the AI movement and has served as the foundation of most advancements in the AI realm.

Specifically, a neural network is defined in terms of three different components, namely neurons, weights and layers. Neurons act as the information storage unit, whereby any input will eventually get passed into every respective neuron for processing. Layers refer to the structural components that make up the network architecture. A neural network is typically organized into multiple layers, each playing a specific role in the overall computation and information flow.

The most fundamental layer in a neural network is the input layer, which receives the initial data or features as input. This layer is responsible for passing the input values forward to the subsequent layers. Following the input layer, there are one or more hidden layers, which perform the majority of the computations in the network. The hidden layers consist of interconnected neurons.

The final layer in a neural network is the output layer, which generates the network's predictions or outputs based on the computations performed in the preceding layers. The number of neurons in the output layer is typically determined by the nature of the problem being addressed. For instance, in a binary classification task, a single neuron with a sigmoid activation function may be used to produce a probability between 0 and 1, representing the likelihood of belonging to one class. In multi-class classification, the output layer may consist of multiple neurons, often employing a softmax activation function to generate a probability distribution across different classes.

In summary, layers in neural networks represent the different levels or stages of computation, from the input layer that receives the initial data to the hidden layers that perform transformations and computations, culminating in the output layer that produces the final predictions or outputs of the network.

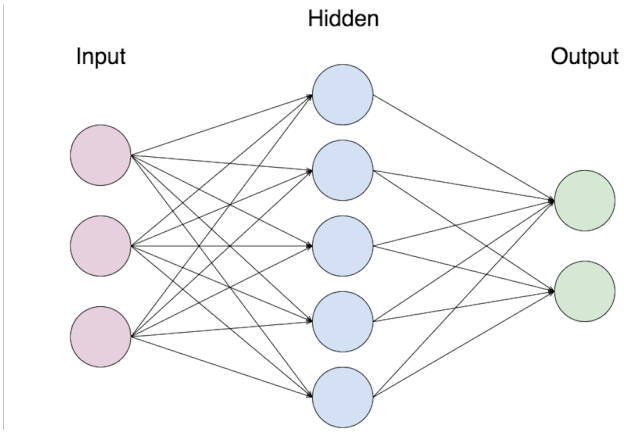

Fig. 4: A Simple Neural Network

E. PSO Trained NN

In the context of training a Neural Network using Particle Swarm Optimization, each dimension of the particle's position represents a weight associated with the NN. The weights in an NN determine the strength of connections between neurons and play a crucial role in the network's overall performance. PSO treats each particle as a potential solution or vector of weights for the NN. The position of a particle represents a candidate solution, where each dimension corresponds to a specific weight value. During the optimization process, particles explore the search space as described above. In the context of weight optimization, PSO iteratively updates the particle's position to find the combination of weight values that minimizes the network's objective function. By adjusting the weights associated with the NN's connections, PSO effectively guides the network towards improved performance by seeking optimal or near-optimal weight configurations. This approach allows PSO to leverage its global search capabilities to efficiently explore the weight space and discover promising solutions for training the NN. Anna Rakitianskaia et al [23] researched multiple different PSO implementations and found fruitful results showing that PSO is generally a good form of training in real-life applications.

IV. Experimental Work

This section aims to explain the implementation details, as well as the paramter values and statistical testing approach that was used to infer conlcusions about the agents. Subsection A focuses on the details of the specific agents implemented, whereas subsection B and C focus on the results, namely how to fairly compare the results and the specific findings respectively.

A. Experimental Procedure

The paper authored by Van Rooyen et al. presented a comprehensive comparative analysis that highlighted the significant superiority of their novel approach over three hand-crafted agents [6]. However, it was noted that the paper lacked a comparison against other AI implementations. To ascertain whether their model effectively addressed intricate challenges beyond alternative implementations, particularly in contrast to Ehlis' approach, a critical evaluation was warranted. This evaluation aimed to determine whether the PSO-trained neural network (NN) exhibited superior performance in playing the game of Snake compared to a genetic program (GP). Notably, the divergence in feature input between the two approaches rendered a direct comparison infeasible.

In response to this challenge, two potential solutions emerged: one involved adapting the Van Rooyen et al. implementation to accommodate the broader feature input characteristic of Ehlis' approach, while the other entailed modifying Ehlis' approach to align with the feature input of the former. The latter approach was ultimately chosen, as it offered the advantage of leveraging the existing hand-crafted agents and the possibility of streamlining the GP's complexity through a more constrained feature input. Consequently, Van Rooyen et al.'s implementation remained unchanged, and a new implementation was developed, preserving the core principles and structure of Ehlis' approach. The specific modifications made to Ehlis' approach are detailed below:

1) Changes of the Approach: Ehlis' approach adhered to a conventional Genetic Programming (GP) framework, employing a function set predominantly composed of Boolean functions. A prog2n function was used to make successive moves, however, this entire function was removed to accommodate a comparison between Van Rooyen et al. approach. The terminal set, on the other hand, consisted of move operators. The genetic programs were structured as binary trees, and the standard GP genetic operators of crossover and mutation were applied [8]. Tournament selection was utilized to select candidate trees within the population for fitness comparison. The individual with the highest score attained through playing games of Snake, was identified as the fittest and selected for reproduction.

Notably, Ehlis pursued a diversified approach by developing three distinct GP configurations. The first configuration featured a limited function set, aimed at investigating whether a

simple solution could effectively address the Snake game. The second configuration extended the function set, introducing more complex operations to explore the game's dynamics. The third configuration retained the same expanded function set as the second configuration but introduced a unique twist. In this case, the initial population generation was primed using individuals from the final population of the second GP. This strategic adaptation sought to leverage the advantages of a previously evolved population to potentially exploit undiscovered optimal solutions that the second GP may have overlooked.

Ehlis' multifaceted approach thus explored a spectrum of GP strategies, ranging from simplicity to complexity. It introduced a priming mechanism to harness the accumulated knowledge from prior populations, all with the overarching goal of enhancing the GP's problem-solving capabilities in the context of the Snake game. The primary distinction between the first GP configuration and Ehlis' original approach pertained to the feature set employed. In Ehlis' original method, feature functions were designed to identify the presence of food in the lane aligned with the snake's head.

In the first GP configuration, these specific functions were replaced with a new set of four functions: ifFoodInColumnUp, ifFoodInColumnDown, ifFoodInRowRight, and ifFoodInRowLeft. Each of these functions would evaluate to true if a food piece was detected within the respective column or row corresponding to the direction indicated. These evaluations were exclusively derived from feature inputs 9 and 10, which were inherited from the Van Rooyen et al. approach. For instance, ifFoodInColumnUp would utilize feature inputs where feature 9 had a value of 0, and feature 10 had a value of 1. Additionally, the functions ifDangerUp, ifDangerLeft, ifDangerRight, and ifDangerDown were retained, operating by returning true if an obstacle was observed in the immediate cell adjacent to the snake's head.

In contrast, the second GP configuration exhibited a broader and more intricate function set. This configuration preserved functions that mapped to the feature set employed in the PSO-trained approach, while eliminating those functions that could not be mapped. The final repertoire of features encompassed all the functionalities employed in the first GP, in addition to new functions that assessed the relative positioning of food in relation to the snake's head, including checks for food in the left, right, up, or down directions. Furthermore, a new function called ifNextMoveLosesGame was introduced to enable the agent to avoid making a move that would lead to an immediate game loss if a safer alternative move was available. The third GP functioned exactly like Ehlis' implementation.

2) Parameter Tuning: The PSO-NN solution used identical parameters as used by Van Rooyen et al. These can be seen in figure 5 below:

Parameter	Symbol	Value
Input neurons	m	10
Hidden neurons	n	10
Output neurons	o	4
Particle dimensionality	dim	154
Swarm size	$size$	100
Inertia weight	ξ	0.72
Cognitive and social terms	c_1 and c_2	1.42

Fig. 5: Parameters for the PSO-NN [6]

The parameters for the GP were decided by using the parameter mentioned in Ehlis' paper [8] or through the use of a trial-and-error manual search. The full parameter list can be seen in table 1 below:

TABLE I: Parameter Values for GPs

Parameter	GP1 value	GP2 value	GP3 value
Population Size	1000	2000	2000
Generations	800	800	200
Tournament Size	10	20	20
Max Depth	5	8	8
Crossover	0.5	0.5	0.5
Mutation	0.5	0.5	0.5

B. Experimental tests

As in Van Rooyen et al., all agents underwent their respective training phases. After that, each agent underwent 50 independent runs playing 100 games each and recording the results. These included the mean points scored ($\bar{\text{p}}$), the mean number of moves ($\bar{\text{r}}$), the mean number of games won ($\bar{\text{q}}$) as well as the standard deviations, respectively, σp. σr and σq. The means and standard deviations are recorded in Table II.

The statistical significance of performance differences between algorithms was assessed using the Mann-Whitney U test at a 95% confidence level with a Bonferroni correction. The results between the PSO approach and the GP agents are shown in Table II-IV. The results comparing the GP approaches or PSO-NN approach to the handwritten approaches were left out as the GP approaches, like the PSO approach had means three standard deviations above the respective handwritten agents mean for scores, implying there was no overlap in data and the GP models would outperform the handwritten models consistently. There were no opposing findings against Van Rooyen et al. paper regarding the handwritten agents [6].

C. Analysis of Results

This subsection aims to explain the insights in the empirical data and make inferences and assumptions based on the results. the p-values are shown, whereby red implies the PSO-NN outperformed the respective GP agent. Green implies no significant difference and blue implies the GP agent outperforms the PSO-NN.

1) Score: The evaluation reveals that the PSO-NN exhibited significantly superior performance to GP1 in terms of achieving higher scores. Regrettably, the results for GP1 proved to be underwhelming and failed to surpass the efficacy of a random agent. Conversely, GP2 and GP3 outperformed the PSO-NN, particularly on larger board dimensions. Notably, GP3's performance closely resembled that of GP2, indicating that, within the scope of these specific experimental assessments, the utilization of a primed initial population did not confer any discernible advantages to the model. Although the disparities in performance were not exceptionally pronounced, it appears that GP2 would be better suited for real-world snake game applications, which often involve a 40 x 40 dimension board, in achieving higher scores compared to the PSO-NN. Furthermore, it was observed that both the PSO-NN and GP agents faced challenges when confronted with odd-numbered board sizes, exhibiting notably less increment in score compared to even-numbered sizes. Further research endeavors could investigate the underlying causes of this phenomenon and extend the experimentation to encompass both larger odd and even board sizes.

2) Number of Moves: In the context of the total number of moves made during gameplay, the GP approach consistently outperformed the PSO-NN. This observation suggests that the GP-based strategies converged towards the development of tactics that extended the lifespan of the snake, involving avoidance of rudimentary traps and the pursuit of non-obvious paths to food sources. Such a tendency is particularly desirable in scenarios with larger board dimensions, where the most expedient route to a food item might lead the snake to an inevitable demise by self-imposed blockages. Further research endeavors could explore the refinement of this strategy by incorporating the number of moves into the fitness function or by introducing incentives for the snake's longevity.

3) Game Wins: In the context of board sizes ranging from 3 to 5, the PSO-NN exhibited superior performance in terms of securing game victories when compared to the GP agents. However, a reversal in outcomes was observed for board size 6, where the GP agents outperformed the PSO-NN. Notably, for board sizes 7 and beyond, neither agent achieved victories, primarily due to the heightened complexity of the game at these dimensions.

V. Conclusion

This study introduced two distinct computational intelligent agent implementations for playing the game of Snake. However, their comparative evaluation was initially challenging due to the utilization of dissimilar feature inputs. To facilitate a meaningful comparison, one of the agents underwent modification to align its feature set with that of the other agent. Subsequently, both modified agents were implemented and subjected to rigorous testing against each other. The empirical results obtained were systematically juxtaposed, enabling the identification of their respective strengths and weaknesses.

Future research endeavors are envisioned to expand this comparative analysis by incorporating a reinforcement learning framework, a domain that has undergone extensive exploration in previous studies. This extension will not only enhance the comprehensiveness of the evaluation but also provide valuable insights into the efficacy of reinforcement learning in the context of the Snake game. Furthermore, both agent models remain amenable to further refinements, offering opportunities for the investigation of improved strategies, fine-tuned parameter adjustments, refined stopping conditions, and the exploration of more advanced and adaptive techniques.

References

[1] R. Cioffi, M. Travaglioni, G. Piscitelli, A. Petrillo, and F. De Felice, "Artificial intelligence and machine learning applications in smart production: Progress, trends, and directions," *Sustainability*, vol. 12, no. 2, p. 492, 2020.

[2] C. W. Connor, "Artificial intelligence and machine learning in anesthesiology," *Anesthesiology*, vol. 131, no. 6, pp. 1346–1359, 2019.

[3] H. Go, M. Kang, and S. C. Suh, "Machine learning of robots in tourism and hospitality: interactive technology acceptance model (itam)–cutting edge," *Tourism review*, vol. 75, no. 4, pp. 625–636, 2020.

[4] S. Kumar, W. M. Lim, U. Sivarajah, and J. Kaur, "Artificial intelligence and blockchain integration in business: trends from a bibliometric-content analysis," *Information Systems Frontiers*, vol. 25, no. 2, pp. 871–896, 2023.

[5] M. Green, A. Khalifa, G. Barros, and J. Togellius, "" press space to fire": Automatic video game tutorial generation," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, no. 2, 2017, pp. 75–80.

[6] C. Van Rooyen, W. Van Heerden, and C. Cleghorn, "Playing the game of snake with limited knowledge: Unsupervised neuro-controllers trained using particle swarm optimization," in *2017 IEEE 4th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, November 2017, pp. 80–84.

[7] 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, September 2016, pp. 1–8.

[8] T. Ehlis, *Application of genetic programming to the snake game*. Cengage, 2009, pp. 113–136.

[9] T. T. K. A. S. Goel, "Snake game for android," 2019.

[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[11] Z. Wei, D. Wang, M. Zhang, A.-H. Tan, C. Miao, and Y. Zhou, "Autonomous agents in snake game via deep reinforcement learning," in *2018 IEEE International conference on Agents (ICA)*. IEEE, 2018, pp. 20–25.

[12] N. J. Nilsson, *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[13] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, pp. 87–112, 1994.

[14] D. R. Lewin, "Evolutionary algorithms in control system engineering," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 45–50, 2005.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, November 1995, pp. 1942–1948.

[16] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization," *AIAA Journal*, vol. 41, no. 8, pp. 1583–1589, 2003.

[17] G.-W. Xu, "An adaptive parameter tuning of particle swarm optimization algorithm," *Applied Mathematics and Computation*, vol. 219, no. 9, pp. 4560–4569, 2013.

[18] A. Idris, A. Iftikhar, and Z. Rehman, "Intelligent churn prediction for telecom using gp-adaboost learning and pso undersampling," *Cluster Computing*, vol. 22, no. Suppl 3, pp. 7241–7255, 2019.

[19] M. Omran, A. Engelbrecht, and A. Salman, "Particle swarm optimization for pattern recognition and image processing," in *Swarm Intelligence in Data Mining*. Springer, 2006, pp. 125–151.

TABLE II: Means and Standard Deviations

Board Size	Model	Score Mean	Score Standard Deviation	numMoves mean	numMoves Standard Deviation	Wins Mean	Wins Standard Deviation
3	randomizer	6.723	0.136	46.887	1.857	46.1	4.965
4	randomizer	7.361	0.28	109.459	4.374	2.84	1.848
5	randomizer	6.959	0.274	197.098	8.364	0.0	0.0
6	randomizer	6.744	0.22	310.753	14.489	0.0	0.0
7	randomizer	6.582	0.247	445.965	18.672	0.0	0.0
8	randomizer	6.218	0.2	590.37	22.138	0.0	0.0
9	randomizer	5.63	0.222	722.043	22.251	0.0	0.0
40	randomizer	0.195	0.045	1000.0	0.0	0.0	0.0
3	towardslessdense	7.022	0.121	16.15	0.448	55.3	4.957
4	towardslessdense	8.514	0.272	31.568	1.283	1.88	1.465
5	towardslessdense	9.528	0.357	46.535	1.937	0.02	0.14
6	towardslessdense	10.625	0.32	60.917	2.54	0.0	0.0
7	towardslessdense	11.732	0.42	76.788	3.281	0.0	0.0
8	towardslessdense	12.891	0.516	93.917	4.397	0.0	0.0
9	towardslessdense	14.142	0.546	112.231	5.042	0.0	0.0
40	towardslessdense	29.575	0.818	823.266	22.824	0.0	0.0
3	wallhugger	7.144	0.146	17.811	0.52	63.02	5.708
4	wallhugger	9.718	0.316	40.936	1.49	9.26	3.56
5	wallhugger	10.814	0.426	68.759	2.769	0.28	0.449
6	wallhugger	11.362	0.446	99.985	3.877	0.0	0.0
7	wallhugger	11.51	0.49	130.768	6.601	0.0	0.0
8	wallhugger	11.577	0.428	160.848	7.177	0.0	0.0
9	wallhugger	11.486	0.419	189.362	8.247	0.0	0.0
40	wallhugger	9.092	0.248	711.677	23.887	0.0	0.0
3	PSOTrained	7.97	0.042	23.022	0.677	99.3	0.854
4	PSOTrained	13.44	0.237	58.664	1.386	54.04	5.227
5	PSOTrained	18.657	0.356	161.797	14.441	4.4	1.908
6	PSOTrained	17.872	0.442	158.236	3.98	0.2	0.529
7	PSOTrained	26.591	0.457	306.039	6.554	0.0	0.0
8	PSOTrained	24.361	0.386	344.936	6.196	0.0	0.0
9	PSOTrained	28.271	0.728	464.051	13.675	0.0	0.0
40	PSOTrained	26.416	0.536	928.402	16.825	0.0	0.0
3	GPFirst	3.38	0.205	174.407	30.237	6.6	2.349
4	GPFirst	6.522	0.391	109.074	25.684	7.58	2.426
5	GPFirst	9.777	0.369	123.026	24.699	0.36	0.794
6	GPFirst	9.775	0.88	598.781	40.753	0.08	0.271
7	GPFirst	6.798	0.444	119.245	23.767	0.0	0.0
8	GPFirst	14.249	1.353	685.345	35.804	0.0	0.0
9	GPFirst	7.558	0.429	134.283	19.269	0.0	0.0
40	GPFirst	17.628	0.271	974.082	9.057	0.0	0.0
3	GPSecond	7.592	0.091	25.449	6.201	79.02	4.032
4	GPSecond	13.346	0.24	59.723	2.425	53.56	5.162
5	GPSecond	16.614	0.334	108.562	2.231	1.68	1.348
6	GPSecond	22.594	0.478	217.702	5.364	2.74	1.764
7	GPSecond	23.281	0.579	254.794	9.174	0.0	0.0
8	GPSecond	27.15	0.569	368.119	9.012	0.0	0.0
9	GPSecond	31.848	0.708	470.403	12.879	0.0	0.0
40	GPSecond	35.459	0.442	977.721	9.461	0.0	0.0
3	GPThird	7.579	0.103	25.286	6.188	78.34	4.278
4	GPThird	13.352	0.233	61.397	4.084	53.16	5.069
5	GPThird	16.618	0.31	108.76	2.338	1.7	1.153
6	GPThird	22.473	0.528	216.835	5.06	2.78	1.285
7	GPThird	23.221	0.648	253.89	10.078	0.02	0.14
8	GPThird	27.25	0.612	369.993	9.181	0.0	0.0
9	GPThird	31.827	0.643	540.682	12.264	0.0	0.0
40	GPThird	35.534	0.339	979.424	8.633	0.0	0.0

[20] Y. Marinakis, M. Marinaki, M. Doumpos, and C. Zopounidis, "Ant colony and particle swarm optimization for financial classification problems," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 604–10 611, 2009.

[21] Y. Rahmat-Samii, "Genetic algorithm (ga) and particle swarm optimization (pso) in engineering electromagnetics," in *17th International Conference on Applied Electromagnetics and Communications, 2003. ICECom 2003*. IEEE, October 2003, pp. 1–5.

[22] C. M. Bishop, "Neural networks and their applications," *Review of Scientific Instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.

[23] A. Rakitianskaia and A. P. Engelbrecht, "Training neural networks with pso in dynamic environments," in *2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 667–673.

TABLE III: Comparison of mean points scored for PSO-NN against the GP agents

Board Size	PSO-NN		Other Agents			p-value
	$\bar{p}$	σp	Agent	$\bar{p}$	σp	
3	7.97	0.042	GP1	3.38	0.205	3.382×10^{-18}
			GP2	7.592	0.091	3.361×10^{-18}
			GP3	7.579	0.103	3.582×10^{-18}
4	13.44	0.237	GP1	6.522	0.391	7.008×10^{-18}
			GP2	13.346	0.24	0.049×10^{-18}
			GP3	13.352	0.233	0.080×10^{-18}
5	18.657	0.356	GP1	9.777	0.369	7.024×10^{-18}
			GP2	16.614	0.334	7.026×10^{-18}
			GP3	16.618	0.31	7.024×10^{-18}
6	17.872	0.442	GP1	9.775	0.88	7.048×10^{-18}
			GP2	22.594	0.478	7.029×10^{-18}
			GP3	22.473	0.528	7.046×10^{-18}
7	26.591	0.457	GP1	6.798	0.444	7.046×10^{-18}
			GP2	23.281	0.579	7.021×10^{-18}
			GP3	23.221	0.648	7.042×10^{-18}
8	24.361	0.386	GP1	14.249	1.353	7.046×10^{-18}
			GP2	27.15	0.569	7.473×10^{-18}
			GP3	27.25	0.612	7.058×10^{-18}
9	28.271	0.728	GP1	7.558	0.429	7.050×10^{-18}
			GP2	31.848	0.708	7.048×10^{-18}
			GP3	31.827	0.643	7.046×10^{-18}
40	26.416	0.536	GP1	17.628	0.271	7.019×10^{-18}
			GP2	35.459	0.442	7.018×10^{-18}
			GP3	35.534	0.339	7.021×10^{-18}

TABLE IV: Comparison of mean moves made for PSO-NN against the GP agents

Board Size	PSO-NN		Other Agents			p-value
	$\bar{r}$	σr	Agent	$\bar{r}$	σr	
3	23.021	0.677	GP1	174.407	30.237	7.051×10^{-18}
			GP2	25.449	6.200	0.006
			GP3	25.285	6.187	0.006
4	58.663	1.385	GP1	109.073	25.683	8.981×10^{-18}
			GP2	59.723	2.424	0.022
			GP3	61.397	4.084	0.0001
5	161.797	14.440	GP1	123.026	24.698	1.989×10^{-12}
			GP2	108.561	2.230	7.061×10^{-18}
			GP3	108.7602	2.337	7.062×10^{-18}
6	158.236	3.979	GP1	598.781	40.752	7.064×10^{-18}
			GP2	217.702	5.363	7.064×10^{-18}
			GP3	216.835	5.059	7.064×10^{-18}
7	306.039	6.554	GP1	119.244	23.767	7.066×10^{-18}
			GP2	254.794	9.174	7.066×10^{-18}
			GP3	253.889	10.078	7.066×10^{-18}
8	344.936	6.196	GP1	685.344	35.804	7.06×10^{-18}
			GP2	368.118	9.012	4.495×10^{-16}
			GP3	369.993	9.181	5.973×10^{-17}
9	464.051	13.674	GP1	134.2832	19.268	7.064×10^{-18}
			GP2	470.403	12.878	0.023
			GP3	540.682	12.264	7.064×10^{-18}
40	928.401	16.824	GP1	974.082	9.056	2.473×10^{-17}
			GP2	977.721	9.461	2.070×10^{-17}
			GP3	979.4244	8.633	1.075×10^{-17}

TABLE V: Comparison of mean wins achieved for PSO-NN against the GP agents

Board Size	PSO-NN		Other Agents			p-value
	$\bar{q}$	σq	Agent	$\bar{q}$	σq	
3	99.3	0.85	GP1	6.6	2.349	2.810×10^{-18}
			GP2	79.02	4.032	3.005×10^{-18}
			GP3	78.34	4.2783	3.014×10^{-18}
4	54.04	5.226	GP1	7.58	2.425	6.417×10^{-18}
			GP2	53.56	5.162	0.7504
			GP3	53.16	5.068	0.4149
5	4.4	1.907	GP1	0.36	0.793	5.119×10^{-17}
			GP2	1.68	1.348	1.203×10^{-10}
			GP3	1.7	1.153	3.835×10^{-11}
6	0.2	0.529	GP1	0.08	0.271	0.306
			GP2	2.74	1.76	2.113×10^{-15}
			GP3	2.78	1.285	6.682×10^{-17}
7	0	0	GP1	0	0	n/a
			GP2	0	0	n/a
			GP3	0.02	0.139	0.327
8	0	0	GP1	0	0	n/a
			GP2	0	0	n/a
			GP3	0	0	n/a
9	0	0	GP1	0	0	n/a
			GP2	0	0	n/a
			GP3	0	0	n/a
40	0	0	GP1	0	0	n/a
			GP2	0	0	n/a
			GP3	0	0	n/a