COSC349 – Assignment 2

Brooklyn Taylor

4308085

1.1 Introduction

This report outlines my approach to implementing a genetic algorithm (GA) to evolve snake agents in the *Snakes on a Grid* environment. The goal is to design a model where snakes learn behaviours and show evidence of fitness improving over time, evolving to prove best behaviours are propagated in the final population and how they contribute to better outcomes. This report also demonstrates experimentation with different GA parameters and fitness shaping profiles to observe how emergent behaviours and strategies arise.

1.2 Genetic Algorithm Methodology

The agent uses a Genetic Algorithm to evolve better chromosomes across generations. I expanded the variables available for tuning and tested how they influenced training outcomes.

• **Elitism count**	*number of top snakes that are transferred into the next population*
• **Tournament size**	*number of snakes selected for tournament selection*
• **Crossover probability**	*chance that we perform a crossover*
• **Mutation probability**	*chance that each gene will be mutated*
• **Mutation deviation**	*size of the random nudge during mutation*
• **Weight clip limit**	*limits of mutation range*

1.2.1 Chromosome Representation

The chromosome is the snake's **DNA**. It is a flat vector of numbers representing the weights and biases of a single-layer perceptron. For my agent:

- **Weights**: 49 × 3 = 147
- **Biases**: 3
- **Total genes**: 150

This representation was chosen because it keeps the search space small, trains quickly, and produces diverse random behaviours at generation 0. A limitation is that it restricts learning to linear, reactive.

1.2.2 Agent Function / Neural Model

The agent function is implemented as a simple **single-layer perceptron**. The snake perceives its surroundings through a **7x7 local grid** with 49 percepts, each percept showing friend, enemy, food, or empty cells. During each turn of the game, the percepts are flattened into a 49-element vector and multiplied by the weight matrix, then shifted by the biases.

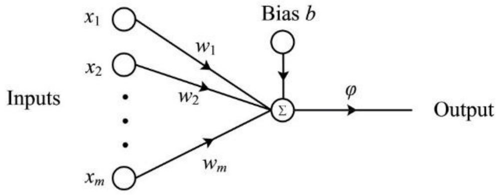


Figure 1 Single-Layer Perceptron Model

The action with the highest output score is then selected for the snakes next movement. This ensured that the behaviour is entirely chromosome-driven rather than hardcoded behaviour. This logic flow is shown using the diagram, where the precepts are the inputs and the resulting action is the output.

1.2.3 Fitness Function

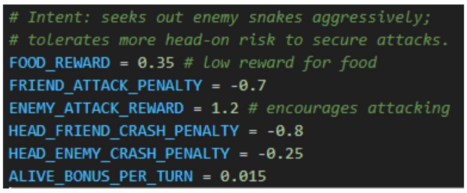
In order to take in as much information as possible from the game environment, I decided to expand the fitness function to also consider the snakes actions and behaviour. By adding a reward/penalty modifier to some of the snakes' previous actions, I had hoped to sway populations away from undesirable behaviours.

This worked by determining the statistics of each snakes behaviour in the previous game, such as the amount of food eaten, enemies attacked, etc., and weigh them against a modifier to encourage behaviours. Defined sets were used to influence certain behaviours and I refer to these as ***Profiles.***

1.2.3.1 Profiles

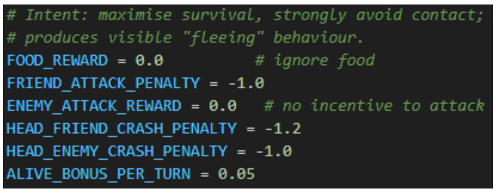
To determine how these modifiers could affect snake behaviour I implemented some **snake profiles**. These are simply predefined sets of values for the fitness evaluations for purposes of having different types of snakes to test with.

Example of the "aggressor" snake.



```
# Intent: seeks out enemy snakes aggressively;
# tolerates more head-on risk to secure attacks.
FOOD_REWARD = 0.35 # low reward for food
FRIEND_ATTACK_PENALTY = -0.7
ENEMY_ATTACK_REWARD = 1.2 # encourages attacking
HEAD_FRIEND_CRASH_PENALTY = -0.8
HEAD_ENEMY_CRASH_PENALTY = -0.25
ALIVE_BONUS_PER_TURN = 0.015
```

Example of the "pacifist" snake.



```
# Intent: maximise survival, strongly avoid contact;
# produces visible "fleeing" behaviour.
FOOD_REWARD = 0.0          # ignore food
FRIEND_ATTACK_PENALTY = -1.0
ENEMY_ATTACK_REWARD = 0.0   # no incentive to attack
HEAD_FRIEND_CRASH_PENALTY = -1.2
HEAD_ENEMY_CRASH_PENALTY = -1.0
ALIVE_BONUS_PER_TURN = 0.05
```

Each behaviour was penalised and rewarded accordingly, this exhibited some emergent behaviour such as running towards or away from enemies depending on the snake profile selected.

The profiles I designed for this project are as follows.

- **Default (baseline):** Moderate food reward, mild penalties.
- **Skirmisher:** Stronger food reward, moderate enemy attack reward, heavy friendly-fire penalties, more balanced priorities.
- **Food Seeker:** High food reward, very strong friendly penalties.
- **Aggressor:** Higher enemy attack reward, weaker survival bias.
- **Pacifist:** Ignores food, heavily penalises combat, maximises survival.

This design allowed testing of how reward shaping drives emergent strategies.

1.3 Genetic Algorithm Operators

1.3.1 Selection Method

I used **tournament selection** (size 3). This balances exploration and exploitation, meaning that stronger snakes are more likely to be chosen, but weaker ones still occasionally contribute. Though I did consider roulette wheel selection, tournament was chosen for its simplicity and diversity.

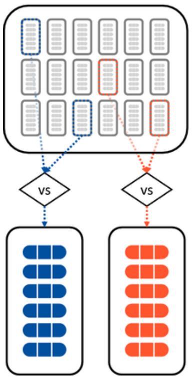


Figure 2 Tournament Selection Example

1.3.2 Crossover

A simple **one-point crossover** was implemented into. Two parent chromosomes are cut at a random point and recombined. This low-complexity method reliably mixes traits without requiring extra complexity.

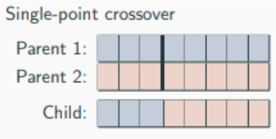


Figure 3 Chromosome Breeding Example

Though I do plan to experiment with *k*-point and Uniform crossover sometime in the future to determine how this might influence scores.

1.3.3 Mutation

Each gene has a probability of being nudged by Gaussian noise. I used:

- **Probability** = 0.05
- **σ** (standard deviation) = 0.1
  This introduces exploration and prevents stagnation. Weights are clipped to ±3.0.

This was to encourage the emergence of potentially stronger candidates in breeding the population, deviation in the chromosome could lead to potentially better results and mimics real-life evolution.

1.3.4 Elitism

The top 2 snakes were copied directly into the next generation. This protected good solutions from being lost when creating the new population. In testing higher elitism counts, this led to premature convergence. That is, the population became too similar too early, leading to an early plateau in the trendline of increasing fitness values over generations.

1.3.5 Training Schedule

The training schedule was set to have each agent play against the random agent over a total of 200 generations. This gave clear baselines and kept the training times reasonable for multiple tests. Alternatively, I had explored staged schedules between other defined agent profiles and against itself. Later explained in the results section, I put two of my agent profiles against each other to determine an overall winner.

The figure on the right demonstrates how the genetic algorithm works to improve the overall population over each generation using the training schedule. Defining an **initial random** population, then incrementally breeding and selecting individual snakes into the next population until it reaches the terminal state (the number of defined generations to train) before the final population is sent to the actual game.

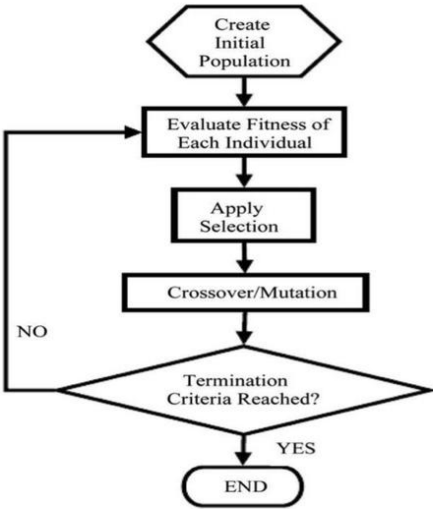

Create Initial Population
Evaluate Fitness of Each Individual
Apply Selection
Crossover/Mutation
NO
Termination Criteria Reached?
YES
END

Figure 4 Genetic Algorithm Flowchart

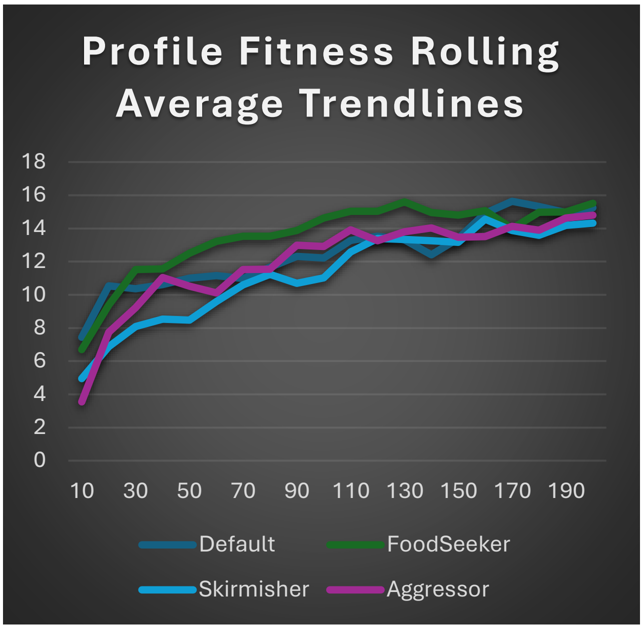
1.4 Snake Performance Analysis

1.4.1 Fitness Over Generations

Across all tested profiles, average fitness increased steadily up to ~150 generations before plateauing. This confirmed that the GA consistently improved chromosomes over time.

- **Baseline:** late-gen average fitness ~13–15.

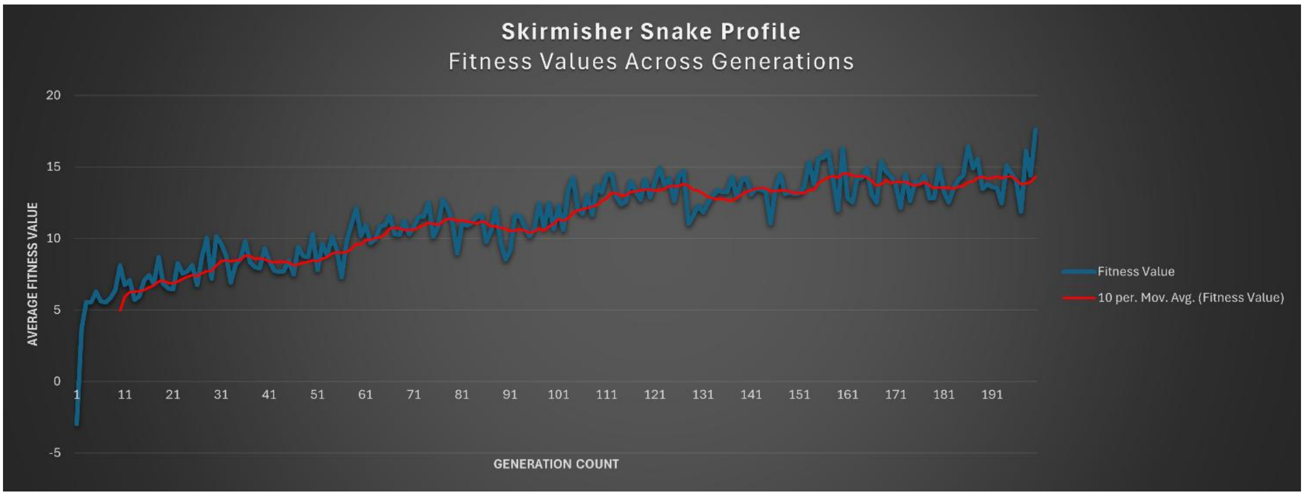
- **Food-First:** fitness ~15–18, but tournament performance suffered.
- **Skirmisher:** late-gen fitness ~14–16.
- **Aggressive:** fitness ~11–13, volatile.

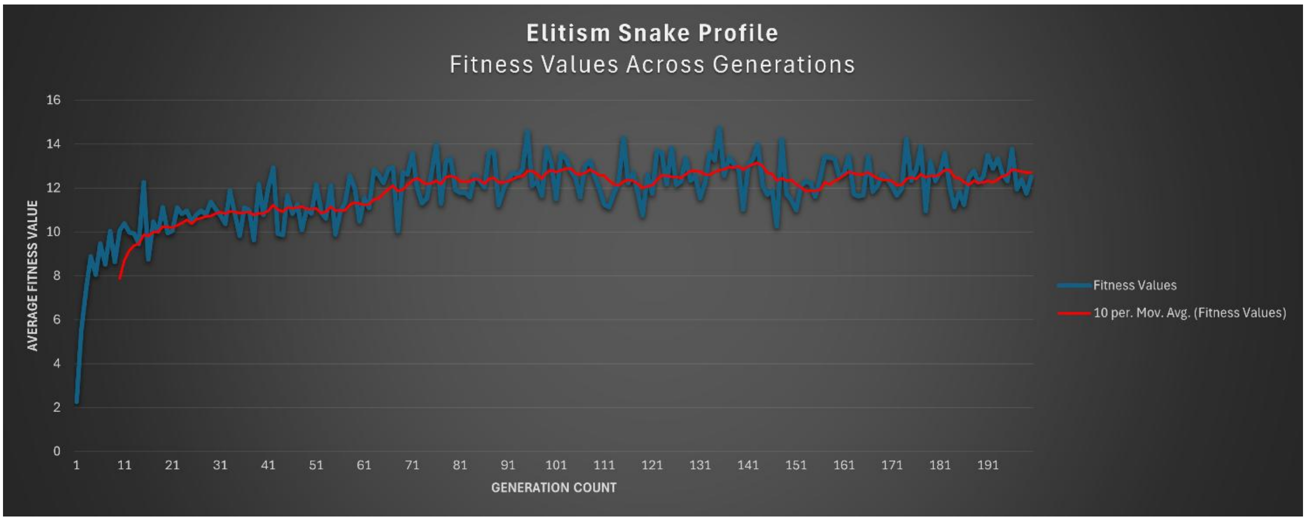


The graph on the right represents a rolling average of the fitness values each agent scored across the 200 generations, this was used to determine which agent showed improvement going forward.

Note that the actual quantitative results for the fitness scores doesn't necessarily determine whether a particular agent is better, more so that the overall trend dictates which agents are improving more over each generation. Since the fitness scores are more subjective and meant to be compared against itself.

In the charts below, it shows how the fitness scores improve over the generations. Though putting all the graphs would bloat the report (more than it already is), so I'm covering the profiles with the worst improving profile, **Elitism**, against the **Skirmisher** profile. Just to demonstrate how my agent improved over time against an agent that showed little improvement (an improvement of **~9.4** > **~4.8**).





1.4.2 Behavioural Observations

Early generations: random, chaotic movement with many snakes spinning or wandering around without any distinct purpose.

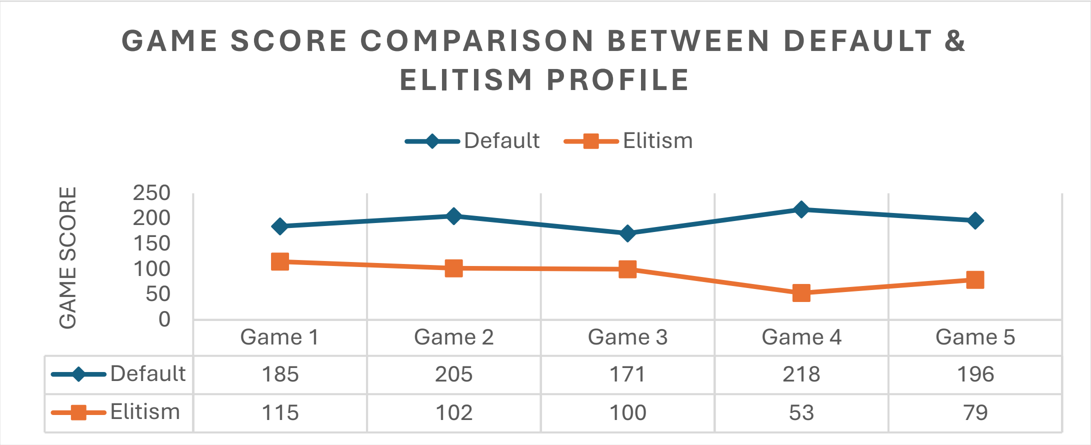
Later generations: more purposeful behaviours emerged. Profiles that were more biased towards food were observed to seek out nearby food, whereas aggressive profiles attacked enemies more often

Unintended behaviours: when snakes had a strong bias towards food with little regard to self-preservation or avoiding "friendly-fire", there was rampant cannibalism and many cases of *ouroboros*. This was a reflection of the mutation noise and mainly due to the limitation of the perceptron model.

1.4.3 Parameter Sensitivity

Due to my personal time constraint (I started this project late due to other priorities) I wasn't able to accurately test each of the parameters of the genetic algorithm thoroughly, and instead used a baseline value set for testing out the fitness evaluation experiments.

In saying this, I did try to improve the algorithm by encouraging a higher "elitism" count to see if more stronger opponents brought into the new generation would improve the overall outcome. This resulted in considerably lower results in terms of final scoring in the game.



	Game 1	Game 2	Game 3	Game 4	Game 5
Default	185	205	171	218	196
Elitism	115	102	100	53	79

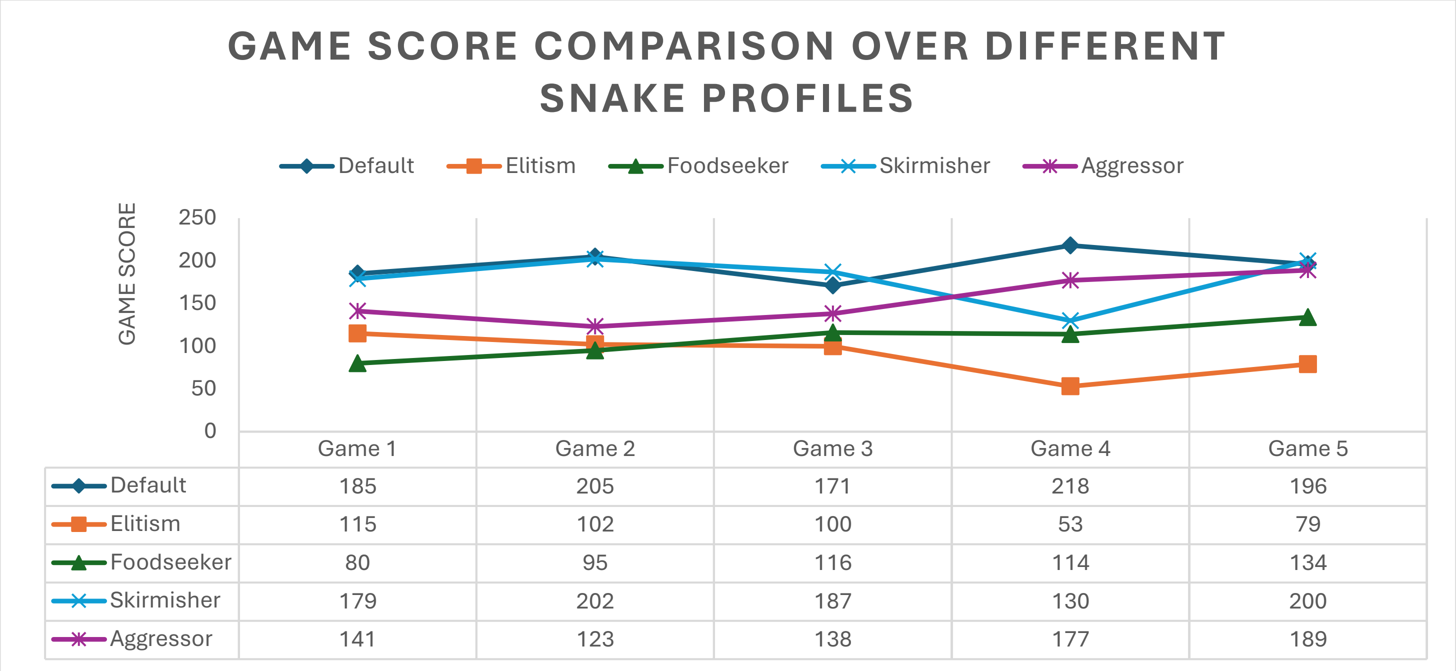
Default Profile
ELITISM_COUNT = 2
TOURNAMENT_SIZE = 3
CROSSOVER_PROBABILITY = 0.7
MUTATION_PROBABILITY = 0.05
MUTATION_STANDARD_DEVIATION = 0.10
WEIGHT_CLIP_LIMIT = 3.0

Elitism Profile
ELITISM_COUNT = 4
TOURNAMENT_SIZE = 4
CROSSOVER_PROBABILITY = 0.8
MUTATION_PROBABILITY = 0.03
MUTATION_STANDARD_DEVIATION = 0.05
WEIGHT_CLIP_LIMIT = 3.0

My idea here was with to reduce the mutation rates and keep the "best" snakes, but this pressure on higher cloning meant that **better strategies were not explored.** Once snakes became stuck on a singular solution, the algorithm wasn't able to push past this. And combined with the lack of mutation rates and higher selection pressure meant less overall diversity.

1.4.4 Training Profile Comparison

In order to determine the best profile to run against the agent I had each of them train against the random agent for 200 generation, then the final score over the five games was then taken into account to determine the best configuration of fitness modifiers to use.



	Game 1	Game 2	Game 3	Game 4	Game 5
Default	185	205	171	218	196
Elitism	115	102	100	53	79
Foodseeker	80	95	116	114	134
Skirmisher	179	202	187	130	200
Aggressor	141	123	138	177	189

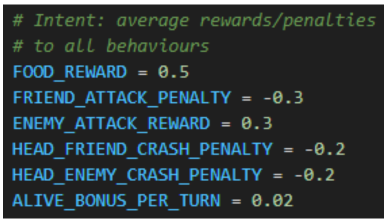
And from the results I then selected the best two profiles and trained them against each other over 500 generations, before they then faced each other in the final game, this final game was run with a random game seed each time and one profile ended up winning each time.

With the balanced skirmisher winning over the default profile with scores ranging from 52 – 140, this profile configuration was selected for the final agent

1.5 Results

To evaluate the impact of different GA configurations and fitness modifiers, I ran five test scenarios with 200 generations of training followed by a 5-game tournament against the random agent. Each test had a fixed random seed of *123*.

- **Test 1 (baseline)** used a modest food reward and standard GA settings. It achieved average tournament scores around 195, showing strong performance overall.

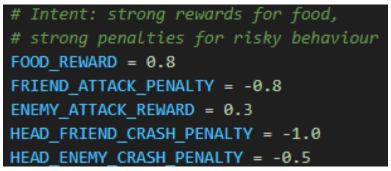


```
# Intent: average rewards/penalties
# to all behaviours
FOOD_REWARD = 0.5
FRIEND_ATTACK_PENALTY = -0.3
ENEMY_ATTACK_REWARD = 0.3
HEAD_FRIEND_CRASH_PENALTY = -0.2
HEAD_ENEMY_CRASH_PENALTY = -0.2
ALIVE_BONUS_PER_TURN = 0.02
```

Figure 5Default Fitness Config Values

- **Test 2 (more elitism and stronger selection pressure)** produced lower fitness improvement trends and tournament scores around ~90, indicating premature convergence and loss of diversity.

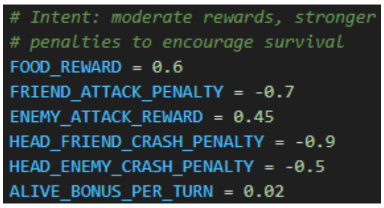
- **Test 3 (food-first profile)** increased initial average fitness trend up to generation 120 before plateauing, and rewarded food collection heavily, but tournament scores averaged only ~108. This suggests that food-seeking alone was not a dominant winning strategy.



```
# Intent: strong rewards for food,
# strong penalties for risky behaviour
FOOD_REWARD = 0.8
FRIEND_ATTACK_PENALTY = -0.8
ENEMY_ATTACK_REWARD = 0.3
HEAD_FRIEND_CRASH_PENALTY = -1.0
HEAD_ENEMY_CRASH_PENALTY = -0.5
```

Figure 6 Food Seeker Fitness Config Values

- **Test 4 (balanced skirmisher)** struck the best balance. With moderate food reward, stronger penalties for friendly crashes, and a reasonable bonus for enemy aggression, it consistently reached a positive upwards trend in fitness values and tournament scores around ~180, the highest competitive success among all tests.

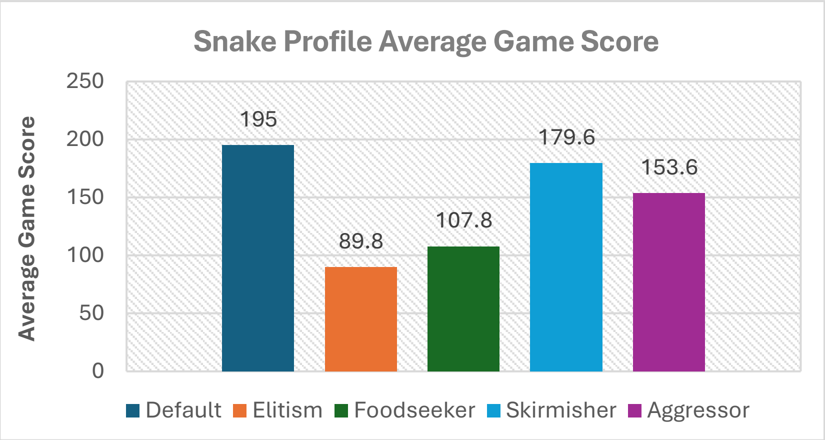


```
# Intent: moderate rewards, stronger
# penalties to encourage survival
FOOD_REWARD = 0.6
FRIEND_ATTACK_PENALTY = -0.7
ENEMY_ATTACK_REWARD = 0.45
HEAD_FRIEND_CRASH_PENALTY = -0.9
HEAD_ENEMY_CRASH_PENALTY = -0.5
ALIVE_BONUS_PER_TURN = 0.02
```

Figure 7 Skirmisher Fitness Config Values

- **Test 5 (aggressor)** encouraged hostile play but reduced stability, leading to average tournament scores of ~154, below the balanced configuration.

From these results, and the outcome defined in ***section 1.4.4***, I selected the **balanced skirmisher configuration** as the final setup. This combination of GA parameters and fitness shaping provided the best mix of exploration and exploitation during training and produced agents that consistently outperformed the random baseline with both higher fitness values and the strongest competitive scores.



The outcome highlighted the importance of balancing the reward/penalty signals. Over-prioritising a single behaviour (e.g., food collection or aggression) led to weaker tournament play, while blending the reward functions produced more robust and effective strategies.

1.6 Discussion & conclusion

1.6.1 Limitations:

The biggest limitation was the brain itself. A single-layer perceptron can only make linear reactive decisions based on the current 7×7 percepts. It doesn't have memory, so it can't plan or carry over "habits" from one turn to the next. Because of this, more complex strategies like running away until safe, or setting up multi-turn plays, were out of reach.

1.6.2 What worked:

Despite the limits, the GA did exactly what it was supposed to: it steadily improved average fitness over 200 generations. Reward shaping clearly influenced behaviour. Food-heavy profiles leaned into food collection, aggressive profiles went after enemies more often, and the balanced skirmisher profile gave the most stable and competitive outcomes. Out of everything I tested, the skirmisher profile turned out to be the best balance between food, survival, and smart aggression.

1.6.3 What didn't:

Two experiments stood out as problems.

- **Strong elitism:** When I raised elitism from 2 → 4 and dropped mutation at the same time, the population converged too fast and got stuck. Fitness values plateaued early, and tournament scores dropped to about half of what the baseline managed. This matches what's described with genetic algorithms: if you keep too many elites, diversity crashes and the population settles into a local optimum too soon.

> *"Premature convergence may occur when an excessive number of elites are preserved in each generation, causing the genetic algorithm to prematurely settle into a local optimum. This scenario restricts the exploration of alternative regions within the search space, potentially hindering the discovery of globally optimal solutions."* source

- **Over-shaped profiles:** The food-first and aggressive profiles showed how leaning too hard into one behaviour doesn't pay off. Food-first snakes bulked up but lost more matches due to ignoring enemies, while aggressive snakes often played recklessly, charging into head-ons that didn't account for the size differences (leading to suicides).

1.6.4 Future Improvements

Several extensions could overcome these issues:

- **Multi-layer Perceptron**: Adding a hidden layer would enable non-linear decision. Adding memory (previous action or orientation as inputs) would allow strategies for snakes to *"plan-ahead"*.
- **Fitness refinements**: Adjusted reward shaping depending on existing conditions (e.g., scaling food reward down once size is large, or penalising repeated spins) could reduce useless behaviours and encourage desirable behaviours.
- **Annealing Mutation Rates**: beginning with a higher mutation rate and reducing this figure down across generations within some bounds would allow for early exploration near the start, but smaller changes in the later stages to ensure that best behaviours aren't altered drastically enough to affect the competing population for the worse.

1.7 SUMMARY

Overall, the experiments show that **GA design is highly sensitive to parameter balance**. The algorithm consistently produced improvements, but also highlighted how elitism, mutation, and fitness shaping must be carefully tuned to avoid stagnation. The chosen balanced skirmisher profile provided the best trade-off between exploration and exploitation, and demonstrated that even with a simple model, careful reward design can produce agents that visibly outperform a random baseline.

1.8 APPENDIX

While the agent was largely implemented by myself, I did make use of external coding assistants during development. ChatGPT 5.0 was occasionally used to help identify and resolve logic errors in my implementation, and GitHub Copilot was used to correct minor syntax issues as they arose. All core design decisions, experimentation, and implementation work was carried out myself.

- *https://www.researchgate.net/figure/Diagram-illustrating-the-tournament-selection-uniform-crossover-and-mutation-operations_fig4_371376186*
- *https://www.mdpi.com/2078-2489/10/12/390*
- *https://algorithmafternoon.com/genetic/elitist_genetic_algorithm*
- *https://www.woodruff.dev/day-6-roulette-tournaments-and-elites-exploring-selection-strategies*