Pacman-NEAT: Exploring the Power of Evolutionary Algorithms in Game Playing

A Case Study with NEAT and Pacman

Andrea Spinelli

08/07/2025

University of Trieste

Introduction

Can an Evolutionary Algorithm Play Pac-Man?

The goal of this project is to train a neural network to master the classic arcade game, Pac-Man. This non-trivial task requires the agent to develop complex strategies for:

- Efficient maze navigation
- Dynamic evasion of multiple, intelligent opponents (ghosts)
- Strategic use of resources (power-ups)
- Long-term planning to clear the entire level

Figure 1: Pac-Man environment

Instead of traditional programming, It's possible to use an evolutionary approach to **discover** these strategies automatically.

NEAT - NeuroEvolution of Augmenting Topologies

We use **NEAT** [3], a powerful algorithm that evolves both the **weights** and the **structure (topology)** of neural networks.

Key features of NEAT:

- **Complexification:** Starts with simple networks and gradually adds nodes and connections.
- **Innovation Numbers:** Tracks the historical origin of genes to solve the "competing conventions problem" during crossover.
- **Speciation:** Groups similar networks to protect new innovations until they're ready to compete.

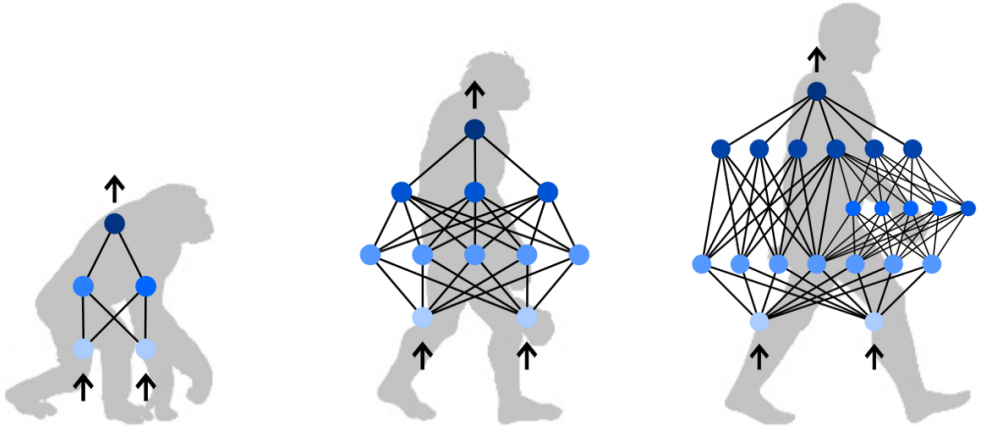


Figure 2: NEAT increases network complexity over time [2].

Choosing a Game Environment

For training NEAT agents, I chose the *PyPacman* environment [1].

Advantages:

- Lightweight and easy to understand
- Fully open-source and modifiable
- Minimal dependencies

This enables ***rapid simulation of thousands of games per generation***.

Limitations:

- Contained several bugs and inconsistencies
- Not optimized for large-scale, automated runs

The repository provided a solid foundation, but required several changes to meet the needs of this project:

- Fixed bugs, optimized performances
- Added a ***Gym-like interface*** for agent-environment interaction

Project Architecture

The project integrates the NEAT algorithm with a Pac-Man environment.

Pac-Man Game Engine

- Adapted from *PyPacman* [1].
- Refactored game state management.
- Core logic encapsulated in a Gym-like environment.

NEAT Framework

- `trainer.py`: Manages the evolutionary loop, population, and checkpoints.
- **Parallel Evaluation:** Evaluates genomes simultaneously to speed up training.

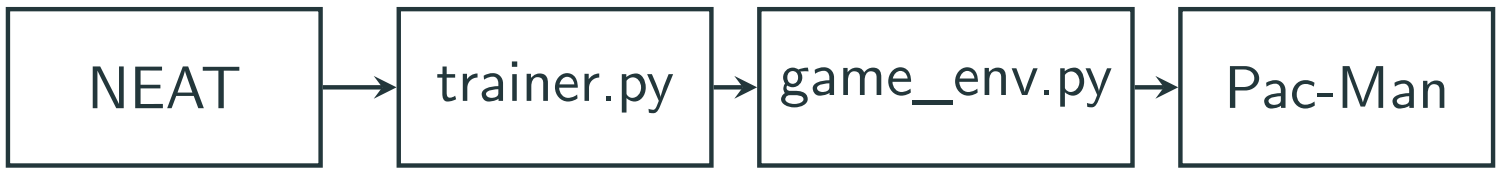


Figure 3: High-level interaction between NEAT and the game environment.

Problem Formulation

How Does Pac-Man "See"? The Observation Model

For the neural network to make a decision, it needs a numerical representation of the game state. This is the **observation vector**.

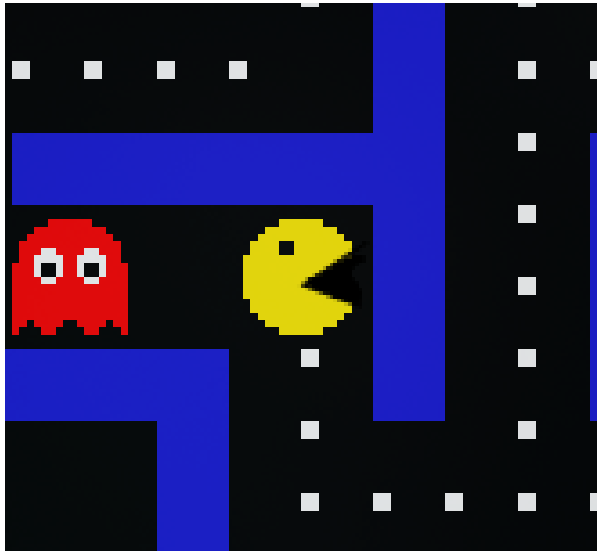
Vectorial Observation Vector (26 inputs):

Feature	Description	# items
Ghost Relative Positions	(x, y) for each ghost	8
Ghost Scared Status	1 bit per ghost	4
Closest Dot Vector	(x, y) to nearest dot	2
Closest Power-up Vector	(x, y) to nearest power-up	2
Remaining Dots	Number of dots left	1
Wall Distances	Distance in 4 directions	4
Power-up Active	1 if power-up is active	1
Last Action	One-hot, previous move	4

Table 1: Total: 26 elements. *Limitation: lacks spatial context.*

Minimap Observation Vector (64 inputs):

The observation shown above was soon found to be ineffective. To give more awareness of the environment, I implemented an 8x8 minimap centered on Pac-Man.



$$\begin{bmatrix} 0.5 & 0.5 & 0.5 & 0 & 0 & -1 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0.5 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0.5 & 0 \\ -0.75 & 0 & 0 & 0 & 0 & -1 & 0.5 & 0 \\ -1 & -1 & -1 & 0.5 & 0 & -1 & 0.5 & 0 \\ 0 & 0 & -1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0 & 0 & -1 & 0.5 & 0 & 0 & 0.5 & 0 \end{bmatrix}$$

It encodes the positions of walls (-1), dots (0.5), and ghosts (-0.75). The four central cells represent the current Pac-Man position.

The final observation vector is the concatenation of the vectorial observation and the minimap observation, for a total of **90 elements**.

How Does Pac-Man "Learn"? The Reward Function

The agent's goal is to maximize its cumulative **reward**. The design of the reward function (*reward shaping*) is critical to guide its behavior.

A naive approach is insufficient:

- *Reward only for points?* The agent might learn to get stuck in a safe corner, doing nothing.
- *Reward only for survival?* The agent might learn to run in circles and never eat dots.

The Challenge: How do we design a task that is simple enough to be learned, yet complex enough to lead to intelligent behavior?

Early Issues: Unintended Behaviors

Initial experiments with reward functions led to classic AI failures:

Reward Hacking

The agent found loopholes in the reward function. For instance:

- If the reward function penalizes being stuck, the agent learns to oscillate between two positions.

Premature Complexity

- Starting with the full game was overwhelming.
- The agent couldn't learn basic evasion while also needing to manage power-ups and complex ghost behaviors.

These challenges motivated a structured approach:

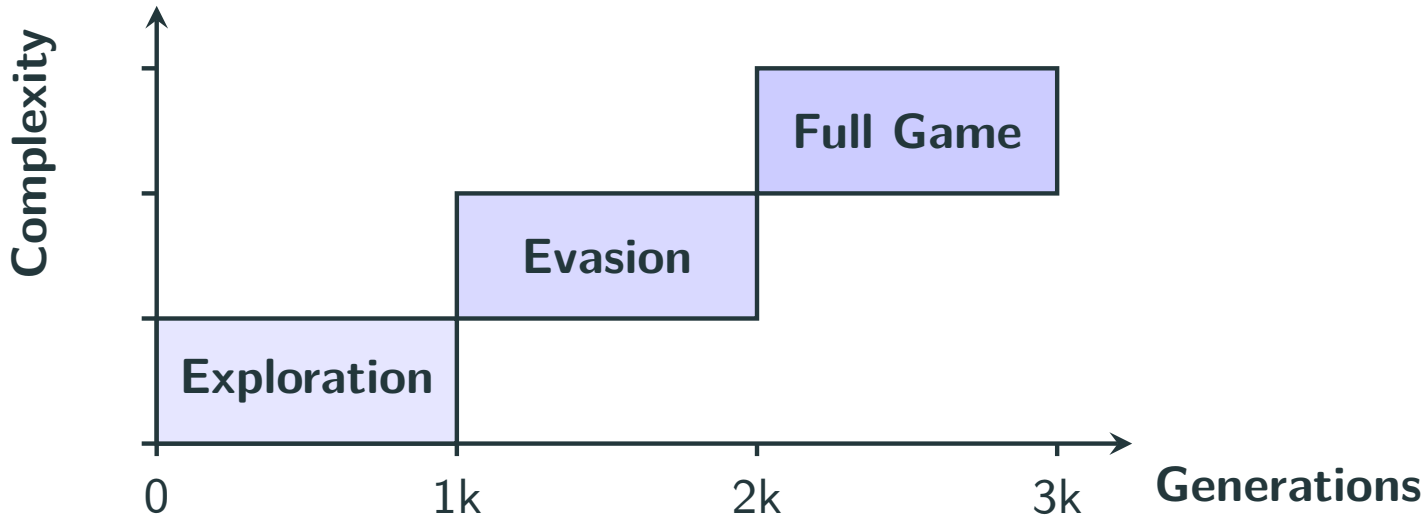
Curriculum Learning

Curriculum Learning

The Concept of Curriculum Learning

Curriculum Learning is a training strategy where the agent is exposed to tasks of increasing difficulty, much like a human student.

Rather than facing the full complexity of the final task immediately, we decompose it into a progression of easier, more manageable sub-tasks.



The hypothesis is that by mastering fundamental skills first (like exploration), the agent can build upon that knowledge to solve more complex problems (like evasion, power-up usage, etc.).

Our Pac-Man Curriculum

The training was divided into automated phases, controlled by the generation number.

Our Pac-Man Curriculum

The training was divided into automated phases, controlled by the generation number.

- **Phase 1: Exploration Training (Gen 0-999)**
  - **Goal:** Master map exploration.
  - **Setup:** No power-ups, no ghosts, only dots.
  - **Reward:** Simple function rewarding dot collection and penalizing time.
  - **Key Tweak (Gen 500):** Switched from Feed-Forward to **Recurrent Networks** to enable memory.

Our Pac-Man Curriculum

The training was divided into automated phases, controlled by the generation number.

- **Phase 1: Exploration Training (Gen 0-999)**
  - **Goal:** Master map exploration.
  - **Setup:** No power-ups, no ghosts, only dots.
  - **Reward:** Simple function rewarding dot collection and penalizing time.
  - **Key Tweak (Gen 500):** Switched from Feed-Forward to **Recurrent Networks** to enable memory.
- **Phase 2: Evasion Training (Gen 1000-1999)**
  - **Goal:** Ghosts are introduced.
  - **Setup:** Only dots and ghosts. Still no power-ups.
  - **Reward:** The same simple reward function.
  - **Key Tweak (Gen 1500):** Introduced scatter mode for ghosts.

Our Pac-Man Curriculum

The training was divided into automated phases, controlled by the generation number.

- **Phase 1: Exploration Training (Gen 0-999)**
  - **Goal:** Master map exploration.
  - **Setup:** No power-ups, no ghosts, only dots.
  - **Reward:** Simple function rewarding dot collection and penalizing time.
  - **Key Tweak (Gen 500):** Switched from Feed-Forward to **Recurrent Networks** to enable memory.
- **Phase 2: Evasion Training (Gen 1000-1999)**
  - **Goal:** Ghosts are introduced.
  - **Setup:** Only dots and ghosts. Still no power-ups.
  - **Reward:** The same simple reward function.
  - **Key Tweak (Gen 1500):** Introduced scatter mode for ghosts.
- **Phase 3: Full Game Mastery (Gen 2000+)**
  - **Goal:** Master the complete game.
  - **Setup:** The full game, including power-ups.
  - **Reward:** A highly-shaped, complex reward function is activated.

The Advanced Reward Function (Phase 3)

To encourage mastery, the final reward function included several components:

- **Dynamic Point Multiplier:** `SCORE_MULTIPLIER` increases as more dots are eaten, incentivizing level completion.
- **Event Bonuses:** Large, distinct rewards for eating dots, power-ups, and scared ghosts.
- **Ghost Proximity Shaping:**
  - Small **penalty** for being near a dangerous ghost.
  - Small **reward** for chasing a scared ghost.
- **Exploration Bonus:**
  - A small **reward** for visiting a new tile for the first time.
  - A large **penalty** for revisiting the same tile.
- **Penalties:**
  - Getting stuck against a wall.
  - Taking too long to eat the next dot (anti-stalling).
  - A large penalty for dying.

Results

Fitness Evolution Across the Curriculum

The fitness graph clearly shows the impact of our curriculum learning.

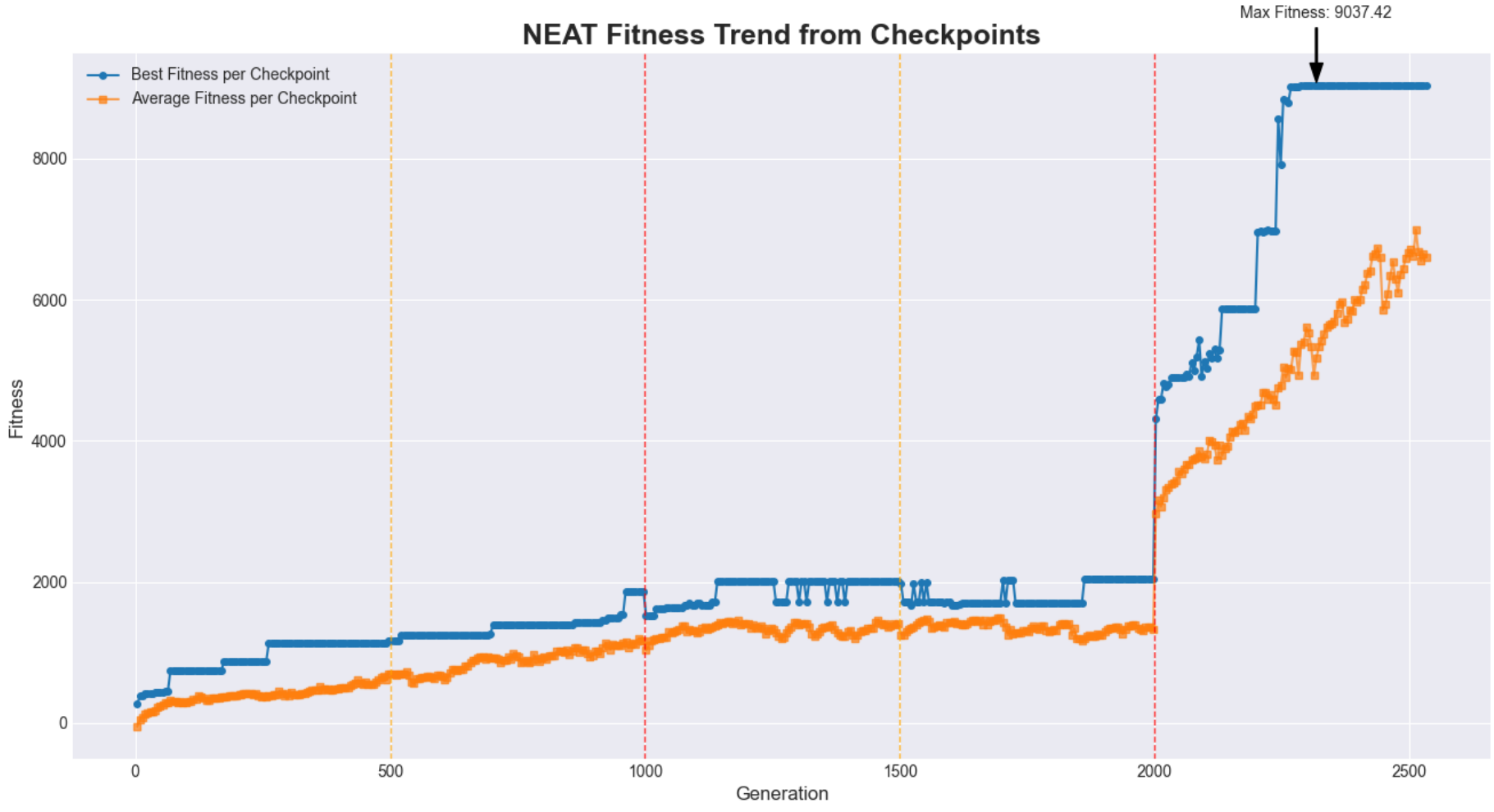


Figure 4: Best fitness per generation. Vertical lines mark curriculum phases.

Analysis of Emergent Behaviors

Analysis of the top-performing agents across curriculum phases reveals a progression in strategy:

Early Generations (Exploration)

- Agents acquire basic ***navigation skills***.
- Tend to become ***stuck against walls*** or exhibit ***oscillatory behavior***.

Mid Generations (Evasion)

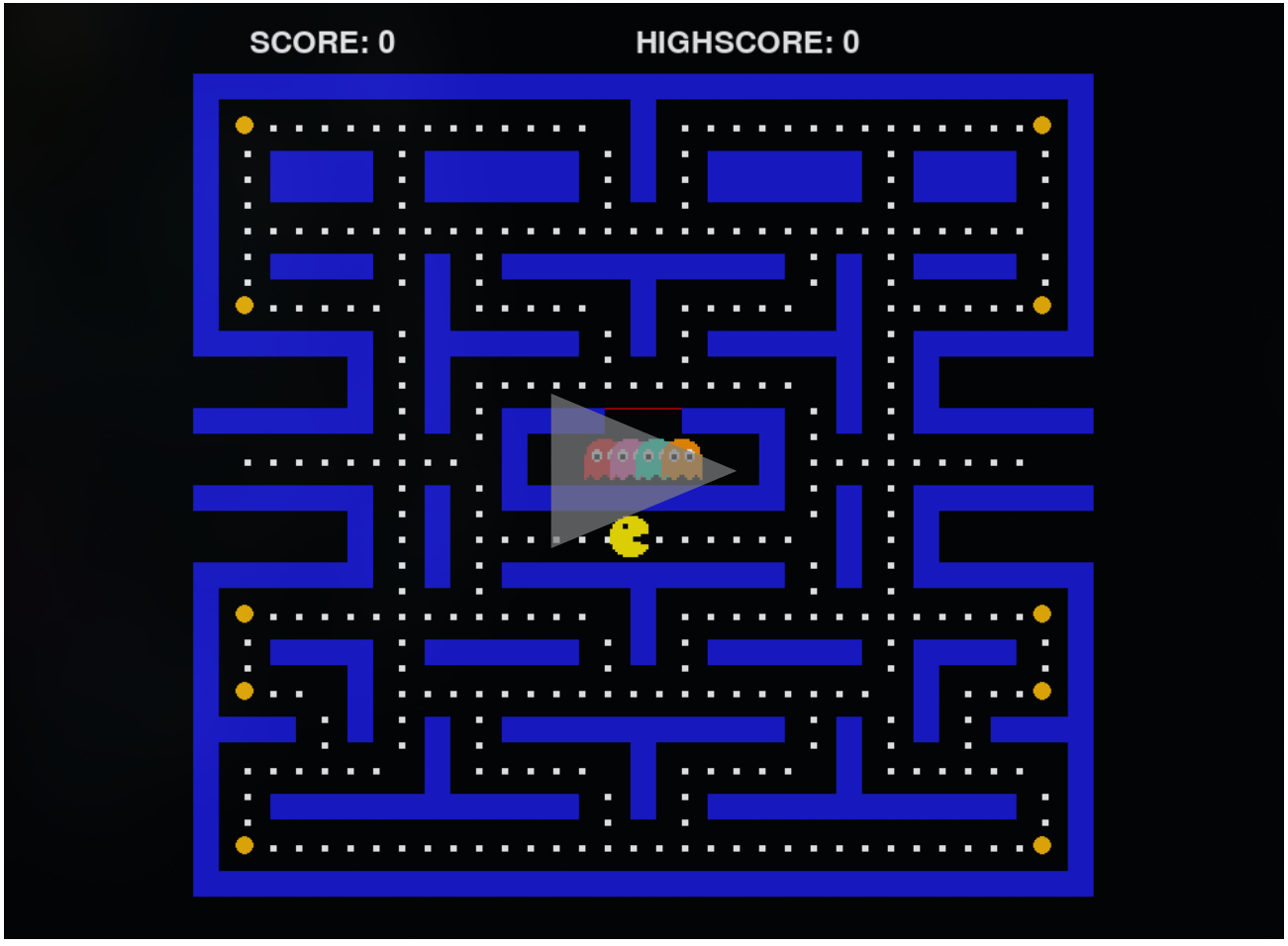
- Agents begin to ***evade ghosts***, despite little improvement in fitness.
- The transition to scatter mode represents the most challenging phase due to ***stochasticity***.

Late Generations (Full Game)

- Agents gradually adapt to the revised reward structure.
- Demonstrate ***improved ghost avoidance*** compared to earlier phases.
- Continue to ***struggle with effective use of power-ups***.

The Best Genome in Action

A demonstration of the final agent's performance, showcasing its ability to navigate, evade, and clear the majority of the level.



Performance Plateau and Model Limitations

Agents achieve high scores but never ends the level, often getting stuck in "safe" corners, especially the top-left, rather than exploring the maze. Power-ups further reinforce this risk-averse behavior.

Key limitations:

- ***Perception bottleneck:*** The 8x8 minimap restricts the agent's view, limiting planning and awareness.
- ***No global planning:*** The agent cannot avoid traps or optimize routes without a broader perspective.
- ***Reward shaping:*** The reward function may favor safe, suboptimal strategies like corner camping.
- ***Potential improvements:*** Richer observations and refined rewards could promote more effective behaviors.

Conclusions

Conclusions

- **NEAT is highly effective:**
  The NEAT algorithm allowed us to evolve neural networks that performed well at playing Pac-Man.

- **Gradual training leads to better results:**
  Training the agent in stages encouraged it to learn more advanced behaviors, such as avoiding ghosts and collecting pellets efficiently. These skills did not emerge when trying to train everything at once.

- **The agent's view limits its performance:**
  Since the agent could only see a small part of the maze at a time, it often made the same mistakes in certain locations.

Future Work

To achieve a perfect score and consistently clear the level, several enhancements could be explored:

1. **Enhanced Observation Space:**
   - Increase the minimap size (e.g. 16x16) for a wider field of view.
   - Provide the agent the entire game grid (increase complexity).
2. **Headless, Pygame-Independent Environment:**
   Rewrite the game loop in pure Python/NumPy to remove the Pygame overhead; this would drastically speed up training.
3. **More Granular Curriculum:**
   Introduce stages where ghost speed or intelligence gradually increases over generations.
4. **Stochasticity:**
   Introduce stochastic elements by randomizing the agent's starting positions. This will encourage the model to develop strategies that are effective throughout the entire maze, rather than favoring specific areas.

References i

- GitHub - AnandSrikumar/PyPacman: Pacman clone written in python — github.com.
  `https://github.com/AnandSrikumar/PyPacman.git`.
- D. Shrestha and D. Valles.
  Reinforced neat algorithms for autonomous rover navigation in multi-room dynamic scenario.
  Fire, 8:41, 01 2025.
- K. O. Stanley and R. Miikkulainen.
  Evolving neural networks through augmenting topologies.
  Evolutionary Computation, 10(2):99–127, 2002.

Thank You!