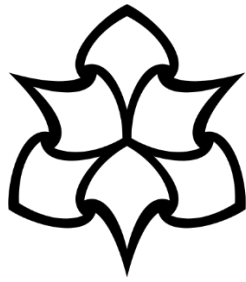


MANCHESTER METROPOLITAN UNIVERSITY



**Manchester
Metropolitan
University**

FINAL YEAR DISSERTATION

Nature vs Nurture: Comparing DQN and NEAT for Generalisable AI in Pac-Man

Author:

Faheem Saleem
(22459265)

Supervisor:

Dr Kate Macfarlane

Degree:

BSc Computer Science with Industrial Year

Department:

Computing and Mathematics

Faculty:

Science and Engineering

May 2026

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work. This work has been carried out in accordance with the Manchester Metropolitan University research ethics procedures and has received ethical approval number **87684**.

Signed

Faheem Saleem _____

Acknowledgements

First and foremost, thank you to Dr Kate Macfarlane for supervising this project. Most supervisors would have raised an eyebrow at a dissertation titled Nature vs Nurture: Comparing DQN and NEAT for Generalisable AI in Pac-Man. Kate raised both thumbs up. As a fellow gamer, she understood the assignment from day one and her enthusiasm for the project made the whole experience genuinely enjoyable. Her feedback consistently pushed the work to a higher standard and her support throughout made a real difference.

Thank you to my Instagram followers for caring about a Pac-Man AI more than most people care about their own projects. Sharing the progress of this project with them on my stories made the whole process a lot more enjoyable than it would have been otherwise.

To my mum and sisters, who sat through more explanations of neural networks than any family should reasonably have to endure, thank you for pretending to understand and supporting me anyway.

And thank you to Pac-Man. You have been eaten by ghosts approximately **5.8 million times** during this project. You deserved better.

Abstract

This study investigates whether a nature-centric evolutionary approach or a nurture-centric gradient-based approach produces greater zero-shot generalisation capability in a dynamic, partially observable environment. A bespoke Pac-Man game engine was developed from scratch and wrapped in a Gymnasium-compliant interface, with both agents operating on an identical 29-dimensional vector observation to ensure any performance differences are attributable to the learning paradigms themselves. Deep Q-Networks and NeuroEvolution of Augmenting Topologies were each trained under both randomised and fixed seed conditions across an eight-stage curriculum, then evaluated on 100 held-out maze seeds never encountered during training. NEAT demonstrated statistically significant superiority in zero-shot generalisation, achieving a mean test reward of 1581.4 compared to DQN’s 912.4 on unseen random mazes, $t(198) = -3.564$, $p < 0.001$, alongside higher pellet collection, spatial coverage, and ghost interaction rates. The memorisation effect was empirically confirmed for both algorithms as fixed seed training enabled greater curriculum progression but produced significantly lower generalisation performance in both cases. These findings provide quantitative evidence that evolutionary approaches produce more transferable policies than gradient-based methods under procedurally varied training conditions, contributing to the broader debate on generalisation in artificial intelligence.

Contents

DECLARATION	I
ACKNOWLEDGEMENTS	II
ABSTRACT	III
LIST OF FIGURES.....	VIII
LIST OF TABLES	IX
ABBREVIATIONS & GLOSSARY.....	X
1. INTRODUCTION	1
1.1. PROJECT BACKGROUND	1
1.2. AIM AND OBJECTIVES	3
1.2.1. <i>Aim</i>	3
1.2.2. <i>Objectives</i>	3
1.3. DISSERTATION STRUCTURE	4
2. LITERATURE REVIEW.....	5
2.1. GENERALISATION OF ARTIFICIAL INTELLIGENCE IN PAC-MAN	5
2.2. EXISTING WORK.....	5
2.2.1. <i>Rule-Based and Heuristic Controllers</i>	6
2.2.2. <i>Deep Reinforcement Learning on Pac-Man</i>	6
2.2.3. <i>Evolutionary and NEAT-Based Approaches</i>	7
2.2.4. <i>Comparisons Between Gradient-Based and Evolutionary Approaches</i>	7
2.2.5. <i>The Gap This Study Addresses</i>	7
2.3. MEMORISATION VS. GENERALISATION	8
2.4. THE NATURE VS. NURTURE PARADIGMS	9
2.4.1. <i>The Nurture-Centric Paradigm: Deep Q-Networks</i>	9
2.4.2. <i>The Nature-Centric Paradigm: NeuroEvolution of Augmenting Topologies</i>	11
2.5. IMPLEMENTATION LIBRARIES	12
2.5.1. <i>Python</i>	12
2.5.2. <i>Gymnasium</i>	12
2.5.3. <i>Pygame</i>	13
2.6. PROFESSIONAL, ETHICAL, AND SOCIAL CONTEXT	13
2.7. SUMMARY.....	14
3. DESIGN AND IMPLEMENTATION.....	15
3.1. ENGINEERING THE TESTBED	15

3.1.1.	<i>Procedural Maze Generation</i>	15
3.1.2.	<i>Maze Validation</i>	16
3.1.3.	<i>Tile-Lock Stepping</i>	18
3.2.	ACTION SPACE AND REWARD DYNAMICS	19
3.2.1.	<i>Action Space</i>	19
3.2.2.	<i>Reward Structure</i>	19
3.3.	GHOST BEHAVIOUR & ADVERSARIAL PATHFINDING	20
3.4.	CURRICULUM LEARNING	31
3.4.1.	<i>Stage-Based Progression</i>	31
3.4.2.	<i>Promotion Criteria</i>	32
3.4.3.	<i>Starvation Scaling</i>	32
3.5.	STATE REPRESENTATION	33
3.5.1.	<i>Block A: Egocentric Raycasts (indices 0–23)</i>	33
3.5.2.	<i>Block B: BFS Global Features (indices 24–26)</i>	34
3.5.3.	<i>Block C: Power State (indices 27–28)</i>	34
3.5.4.	<i>Normalisation</i>	35
3.6.	DQN IMPLEMENTATION	35
3.6.1.	<i>Network Architecture - Dueling DQN</i>	36
3.6.2.	<i>Double DQN - Preventing Overconfident Predictions</i>	36
3.6.3.	<i>Prioritised Experience Replay - Learning From Mistakes First</i>	36
3.6.4.	<i>Exploration - Balancing Discovery and Exploitation</i>	37
3.6.5.	<i>Parallelised Environment Execution</i>	37
3.6.6.	<i>Reward Normalisation</i>	37
3.6.7.	<i>Training Configuration</i>	38
3.7.	NEAT IMPLEMENTATION	39
3.7.1.	<i>Starting Minimal and Growing</i>	40
3.7.2.	<i>Fitness Function</i>	40
3.7.3.	<i>Speciation, Protecting New Ideas</i>	40
3.7.4.	<i>Selection and Reproduction</i>	41
3.7.5.	<i>Parallelisation</i>	41
3.7.6.	<i>Configuration Summary</i>	41
3.8.	EVALUATION METHODOLOGY	42
3.8.1.	<i>The Generalisation Problem</i>	42
3.8.2.	<i>Seed Partitioning</i>	42
3.8.3.	<i>Fixed-Seed Baseline Condition</i>	42
3.8.4.	<i>Training Termination</i>	43
3.8.5.	<i>Zero-Shot Generalisation Testing</i>	43
3.8.6.	<i>Statistical Analysis</i>	43

3.8.7.	<i>Metrics Recorded</i>	44
3.9.	EXPERIMENTAL SETUP	44
3.10.	SUMMARY.....	45
4.	RESULTS AND TESTING	46
4.1.	EXPERIMENTAL OVERVIEW	46
4.2.	DQN TRAINING RESULT.....	47
4.2.1.	<i>Curriculum Progression</i>	47
4.2.2.	<i>Loss and Epsilon</i>	48
4.2.3.	<i>Ghost Interaction</i>	48
4.3.	NEAT TRAINING RESULTS	49
4.3.1.	<i>Curriculum Progression</i>	49
4.3.2.	<i>Fitness and Species</i>	49
4.4.	ZERO-SHOT GENERALISATION TEST RESULTS.....	51
4.4.1.	<i>Reached-Stage Test</i>	51
4.4.2.	<i>Fixed-Stage-7 Test</i>	53
4.5.	STATISTICAL ANALYSIS	53
4.6.	OUTCOME DISTRIBUTION	55
4.7.	SUMMARY.....	56
5.	EVALUATION	58
5.1.	TRAINING EFFICIENCY	58
5.2.	CURRICULUM EFFECTIVENESS.....	59
5.3.	GENERALISATION PERFORMANCE.....	60
5.4.	BEHAVIOURAL ANALYSIS	61
5.5.	HYPOTHESIS EVALUATION	62
5.6.	LIMITATIONS.....	63
5.7.	SUMMARY.....	64
6.	CONCLUSION	65
6.1.	SUMMARY OF COMPLETED WORK	65
6.2.	LIMITATIONS.....	66
6.3.	FUTURE WORK	67
6.3.1.	<i>Parallelised DQN Environment Execution</i>	67
6.3.2.	<i>Recurrent DQN Architecture</i>	67
6.3.3.	<i>Extended NEAT Training Budget</i>	67
6.3.4.	<i>Multiple Episodes Per Test Seed</i>	68
6.3.5.	<i>Curriculum Demotion</i>	68
6.3.6.	<i>Comparative Study with Additional Algorithms</i>	68

6.4. SUMMARY.....	68
REFERENCES.....	70
BIBLIOGRAPHY.....	73
APPENDICES.....	74

List of Figures

Figure 1.1 Start screen of Pac-Man arcade	1
Figure 2.1 The Deep Q-Network agent-environment interaction loop.....	9
Figure 2.2 The NEAT evolutionary process	11
Figure 3.1 Example of generated maze layout.....	16
Figure 3.2 Previous validation only checked paths to the 4 corners.....	17
Figure 3.3 Debugging Pellets Layout	18
Figure 3.4 Blinky always targets Pac-Man's current grid position	21
Figure 3.5 Blinky navigating through the maze via A* pathfinding	22
Figure 3.6 Ghosts are not allowed to make sudden U-turns even if it means it takes longer to get to its target tile.....	23
Figure 3.7 Pinky always targets 4 tiles ahead of Pac-Man	24
Figure 3.8 Pinky targeting ahead of a wall, preventing successful interception.....	25
Figure 3.9 Pinky positioned ahead of Pac-Man, demonstrating successful interception.....	26
Figure 3.10 Clyde targeting Pac-Man's current position when more than 8 tiles away	27
Figure 3.11 Clyde retreating to its corner scatter position when within 8 tiles of Pac-Man ...	28
Figure 3.12 Inky computing its target tile using Blinky's position and a projected point two tiles ahead of Pac-Man, with both paths visible	29
Figure 3.13 Inky's resulting movement trajectory, showing how its target shifts dynamically as both Pac-Man and Blinky move	30
Figure 3.14 DQN Visual Training	35
Figure 3.15 NEAT Visual Training	39
Figure 4.1 Training Progression Dashboard, Rolling Win Rate with Stage Transitions	47
Figure 4.2 DQN Epsilon vs Rolling Win Rate	48
Figure 4.3 Ghost Interaction Signals Over Training.....	48
Figure 4.4 NEAT Best vs Avg Fitness by Generation.....	50
Figure 4.5 NEAT Species Count	50
Figure 4.6 Fixed vs Random Seed Comparison.....	52
Figure 4.7 Test Performance Deep Dive box plots.....	52
Figure 4.8 Outcome Analysis Dashboard	55
Figure 4.9 Efficiency Dashboard	56

List of Tables

Table 3-1 Reward signal components and their associated values	20
Table 3-2 Curriculum learning stages configuration	32
Table 3-3 Egocentric Directions	34
Table 3-4 Correct Heading Mappings.....	34
Table 3-5 DQN Hyperparameters	39
Table 3-6 NEAT Hyperparameters	42
Table 3-7 Metrics recorded per episode during training and evaluation	44
Table 4-1 Training Configuration and Pipeline Outcomes for All Four Conditions	46
Table 4-2 Zero-Shot Generalisation Results Across All Four Conditions (Reached-Stage Test)	51
Table 4-3 Zero-Shot Generalisation Results Across All Four Conditions (Fixed Stage 7 Test)	53
Table 4-4 Statistical Test Results for All Comparisons.....	54

Abbreviations & Glossary

<i>Term</i>	<i>Definition</i>
<i>AI (Artificial Intelligence)</i>	Computer systems that perform tasks which would normally require human intelligence, such as learning, decision-making, and problem-solving.
<i>ALE (Arcade Learning Environment)</i>	A software framework that gives AI researchers standardised access to Atari 2600 games, widely used to benchmark learning algorithms.
<i>BFS (Breadth-First Search)</i>	A pathfinding method that explores all nearby tiles before moving further away, guaranteeing the shortest possible route is found.
<i>CNN (Convolutional Neural Network)</i>	A type of neural network designed to process grid-like data such as images by detecting patterns across local regions.
<i>DQN (Deep Q-Network)</i>	A reinforcement learning algorithm that uses a neural network to learn which actions are most valuable in a given situation through repeated trial and error.
<i>Epsilon (ϵ)</i>	A value between 0 and 1 that controls how often an agent acts randomly versus using what it has learned. High epsilon means more random exploration.
<i>Fitness</i>	In evolutionary algorithms, a score assigned to each candidate solution indicating how well it performed at the task. Better fitness means higher chance of survival and reproduction.
<i>GA (Genetic Algorithm)</i>	An optimisation method inspired by biological evolution, where a population of solutions is repeatedly selected, combined, and mutated to improve over time.

<i>Genome</i>	In NEAT, a single candidate neural network including all of its connections and structure, passed down and mutated across generations.
<i>Hyperparameters</i>	Settings chosen before training begins that control how a model learns, such as learning rate, batch size, and exploration rate. Unlike weights, these are not learned automatically.
<i>MDP (Markov Decision Process)</i>	A formal way of describing a decision-making problem where an agent observes a situation, takes an action, receives a reward, and moves to a new situation.
<i>NEAT (NeuroEvolution of Augmenting Topologies)</i>	An evolutionary algorithm that evolves both the structure and weights of a neural network simultaneously, rather than training through gradient descent.
<i>Neural Network</i>	A computational model loosely inspired by the brain, consisting of layers of connected nodes that learn to map inputs to outputs through training.
<i>PER (Prioritised Experience Replay)</i>	A technique where the agent replays its most informative past experiences more frequently during training, improving learning efficiency.
<i>POMDP (Partially Observable Markov Decision Process)</i>	A decision-making framework where the agent cannot see the full state of the environment and must act based on incomplete information.
<i>Policy</i>	The strategy an agent follows to decide which action to take in a given situation. A trained policy maps observations to actions.
<i>Population</i>	In evolutionary algorithms, the collection of all candidate solutions being evaluated and evolved in a single generation.

<i>Q-Value</i>	An estimate of how much total future reward an agent can expect by taking a specific action in a specific situation.
<i>RL (Reinforcement Learning)</i>	A way of training AI by rewarding desired behaviour and penalising undesired behaviour, allowing the agent to learn through experience rather than explicit instruction.
<i>Speciation</i>	In NEAT, the process of grouping similar genomes together so that new structural innovations are protected from being eliminated before they have time to develop.
<i>Stochastic</i>	Involving randomness. A stochastic process does not always produce the same result from the same starting conditions, introducing an element of chance.
<i>Zero-Shot Generalisation</i>	The ability of an agent to perform well in an environment it has never seen before, without any additional training on that specific environment.

1. Introduction

This chapter has introduced the research motivation, established the nature versus nurture framing of the comparison, stated the aim and objectives, and outlined the structure of the dissertation. Chapter 2 reviews the existing literature relevant to this study, covering prior work in AI-driven Pac-Man, the theoretical foundations of DQN and NEAT, and the broader debate on generalisation in reinforcement learning.'

1.1. Project Background

Pac-Man is a classic arcade game that features a player moving around a maze to collect dots while avoiding four enemy ghosts. Eating a power pellet causes the dynamics of the game to be reversed where the ghosts become vulnerable to being eaten by the player, who then earns extra points. The aspects mentioned, including maze navigation, enemy chasing, and role inversion, make Pac-Man an excellent testbed for evaluating artificial intelligence applications.

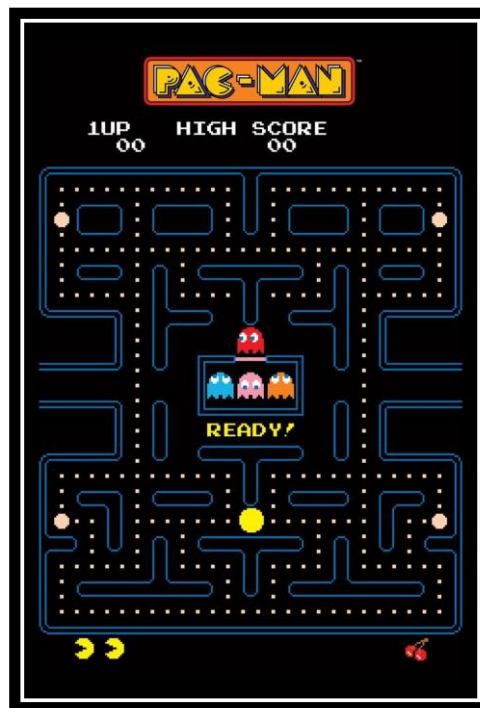


Figure 1.1 Start screen of Pac-Man arcade

The application of AI in video games as a controlled testbed for machine learning research has been common practice for decades, owing to their clearly defined objectives, quantifiable outcomes, and configurable difficulty. Pac-Man in particular has become a recurring benchmark since Mnih et al. (2015) demonstrated that Deep Q-Networks could exceed human

performance on Atari 2600 titles. However, subsequent research has raised fundamental questions about whether such results reflect genuine adaptive intelligence or simply the memorisation of fixed patterns exploited in deterministic environments. This project addresses that question directly by framing it as a nature versus nurture comparison between two fundamentally different learning paradigms: Deep Q-Networks, which derive policy through gradient-based environmental interaction, and NeuroEvolution of Augmenting Topologies, which evolves both network weights and topology through genetic selection. Both agents are evaluated on a custom-built procedurally generated Pac-Man testbed, ensuring that generalisation to unseen environments rather than memorisation of training layouts is the measure of success. To provide a controlled baseline for comparison, a parallel fixed-seed condition is also evaluated in which both algorithms train and are tested on an identical maze configuration, allowing the degree of memorisation versus genuine generalisation to be quantified directly.

This study addresses that gap through a direct comparison framed around the nature versus nurture hypothesis in artificial intelligence. The nurture paradigm is represented by Deep Q-Networks, a gradient-based reinforcement learning algorithm that derives its policy entirely from environmental interaction and iterative parameter refinement. The nature paradigm is represented by NeuroEvolution of Augmenting Topologies, which evolves both the weights and topology of a neural network through genetic selection rather than gradient descent. Both algorithms are applied to a custom-built procedurally generated Pac-Man environment to determine which paradigm produces greater zero-shot generalisation capability.

In order to guarantee a level playing field for the two agents, a custom-made object-oriented game engine for Pac-Man was designed from scratch in Python and interfaced using a Gymnasium-compatible reinforcement learning framework. Both agents utilize a 29-dimensional vector observation space that disassociates decision making from visual perception. The purpose of doing so is to ensure that any performance disparity witnessed between the two algorithms is directly attributed to their respective learning strategies.

1.2. Aim and Objectives

1.2.1. Aim

This project aims to scientifically evaluate and compare the generalisation capabilities of DQN and NEAT within a dynamic, procedurally generated Pac-Man environment, determining which paradigm produces more robust and adaptable behaviour when deployed on novel, unseen map configurations.

1.2.2. Objectives

- Design and implement a bespoke, fully configurable Pac-Man game engine with deterministic procedural maze generation, reproducible seeding, and tile-exact physics, sufficient to serve as a controlled experimental testbed.
- Develop a Gymnasium-compliant environment wrapper exposing a 29-dimensional vector observation and discrete egocentric action space, ensuring both DQN and NEAT interact with an identical interface.
- Implement a DQN agent utilising PyTorch to represent the gradient-based, nurture-centric learning paradigm.
- Implement a NEAT agent utilising Python-NEAT to represent the evolutionary, nature-centric learning paradigm.
- Evaluate both agents on held out procedurally generated maze seeds not seen during training, measuring win rate, pellet collection rate and episode outcome distribution as primary generalisation metrics.
- Conduct a comparative analysis of the two paradigms with respect to generalisation performance, sample efficiency, and behavioural stability, drawing conclusions relevant to the broader “nature versus nurture” debate in artificial intelligence research.
- Compare performance across fixed-seed and random-seed training conditions to empirically quantify the degree to which each algorithm’s policy relies on environment-specific memorisation rather than transferable spatial reasoning.

Based on the theoretical foundations established in the literature, it is hypothesised that NEAT will demonstrate slower initial learning due to the computational overhead of population-based

evolution, but will achieve superior zero-shot generalisation on novel unseen maze configurations compared to DQN, owing to its structural flexibility and population-level diversity

1.3. Dissertation Structure

The remainder of this dissertation is structured as follows. Chapter 2 presents the literature review, covering the generalisation problem in AI-driven Pac-Man, existing work in rule-based and learned approaches, the theoretical foundations of DQN and NEAT, the role of procedural generation and curriculum learning, the implementation libraries underpinning the testbed, and the professional and ethical context of the research. Chapter 3 details the methodology and implementation, including procedural maze generation and validation, the custom game engine architecture, state representation design, reward dynamics, ghost behaviour and adversarial pathfinding, the Gymnasium environment interface, and the curriculum learning system. Chapter 4 presents the experimental results for both algorithms across curriculum stages and generalisation test conditions. Chapter 5 provides a comparative analysis and discussion of the results, interpreting performance differences in the context of the nature versus nurture hypothesis. Chapter 6 concludes the dissertation, summarising completed contributions, acknowledging limitations, and identifying directions for future work.

2. Literature Review

This chapter reviews the existing body of research relevant to this study. It begins by examining the generalisation problem in AI-driven Pac-Man before surveying prior work across rule-based controllers, deep reinforcement learning, and neuroevolutionary approaches. The theoretical foundations of both DQN and NEAT are then established, followed by a discussion of procedural generation, curriculum learning, and the implementation libraries underpinning the testbed. The chapter concludes by situating the present study within the wider professional and ethical context of AI research. Together these sections establish the motivation for the project and identify the specific gap in the literature that this study addresses.

2.1. Generalisation of Artificial Intelligence in Pac-Man

Mnih et al. (2015) explain that Deep Reinforcement Learning algorithms, specifically Deep Q-Networks, have achieved superhuman performance on classic Atari 2600 games including Pac-Man. However, Machado et al. (2018) note that these results are less impressive than they appear. The original Atari environments are fully deterministic, meaning the maze layout, pellet positions, and ghost spawn points are identical in every single training episode. An agent does not need to understand the game to score highly, it just simply needs to memorise which action to take at each moment. Cobbe et al. (2019) confirm this memorisation problem is widespread, demonstrating that agents trained on fixed game levels show significant performance collapse when tested on unseen layouts. Irpan (2018) further notes that DRL agents are prone to overfitting to narrow patterns within their training environments, meaning a policy that appears capable on a fixed map typically fails when the layout changes.

It becomes clear that a good result on a static environment cannot be taken as proof of the existence of intelligence. In this regard, testing the agent on a new procedurally generated environment becomes essential, as it will enable us to understand whether the agent uses intelligent behaviour or memory of what it has seen before.

2.2. Existing Work

Pac-Man has served as a recurring benchmark environment for artificial intelligence research since its commercial release in 1980, attracting approaches spanning rule-based controllers, evolutionary algorithms, and modern deep reinforcement learning architectures. Reviewing this body of prior work is crucial for identifying the specific empirical gap that the present

study addresses and for establishing why existing approaches are insufficient to answer the generalisation question this project poses.

2.2.1. Rule-Based and Heuristic Controllers

The earliest computational approaches to Pac-Man relied on hand-crafted heuristic policies. Gallagher and Ryan (2003) demonstrate that competitive navigation behaviour on fixed layouts is achievable through evolutionary rule-based approaches that encode explicit ghost-avoidance and pellet-seeking logic. Similarly, Rohlfshagen and Lucas (2011) show that carefully tuned reactive controllers can reach high scores on static Ms Pac-Man layouts. These controllers operate through conditional logic, if a ghost is nearby, take evasive action; if a pellet is ahead, move toward it. While this sounds general, each rule carries implicit spatial assumptions about what the surrounding geometry looks like when that condition fires. A rule that says “turn left when a ghost is ahead” only produces a safe outcome if the left corridor is open and leads somewhere useful, which depends entirely on the specific maze layout the rule was designed for. When the map geometry changes, the same conditions trigger the same actions but in a different spatial context, producing behaviour that is no longer valid. This structural brittleness is not correctable through further hand-crafting so adding more rules to handle more edge cases simply produces a larger set of layout-specific assumptions rather than genuine spatial reasoning. It is evident that heuristic controllers establish a performance ceiling on fixed maps but provide no viable path toward zero-shot generalisation.

2.2.2. Deep Reinforcement Learning on Pac-Man

The release of the Arcade Learning Environment (ALE) by Bellemare et al. (2013), which provided a standardized interface for accessing Atari 2600 ROMs, formally recognized Pac-Man as a standard benchmark task for deep reinforcement learning tasks. In this study, Mnih et al. (2015) showed that a DQN agent trained using raw pixel inputs was able to surpass human performance in a wide range of Atari games, including Pac-Man. These results were generated using fully deterministic, vanilla versions of the ROM environment; thus, both training and testing took place using the same fixed layout. According to Machado et al. (2018), this method successfully assesses memorised action sequences rather than flexibility, which is a crucial distinction for the current investigation. Moreover, using pixels as inputs means that the network is tasked with solving both a visual recognition problem and a control problem at the same time, a point which makes the architecture incomparable to the vector-based input variant.

2.2.3. Evolutionary and NEAT-Based Approaches

Wittkamp, Barone, and Hingston (2008) use NEAT itself to create policies for Ms Pac-Man, showing that evolution of neural architectures via NEAT can produce useful navigation and avoidance behaviours without computing any gradients. In their work, Schrum and Miikkulainen (2016) push NEAT a step further and prove that modular NEAT networks can distinguish between avoiding ghosts and collecting pellets, which means better generalization and more flexible strategies. Hence, It is evident that NEAT produces viable navigation behaviour in Pac-Man environments. Nevertheless, no experiments were conducted in procedurally generated mazes.

2.2.4. Comparisons Between Gradient-Based and Evolutionary Approaches

Comparisons of DRL algorithms and neuroevolutionary techniques, on the other hand, have been made more recently within the wider scope of RL research. Such et al. (2017), for example, have shown through extensive experiments that genetic algorithms prove to be equally good alternatives to DQN in terms of their performance in fixed environments in Atari games, despite differing drastically in terms of computational complexity. Similarly, Salimans et al. (2017) prove that evolutionary strategies are also comparable to gradient-based RL in terms of performance on standard continuous control problems. In both cases, however, the generalisability of these algorithms has been tested in fixed environments as well. Other than this methodological restriction, it is worth noting that neither experiment has made use of the topology-evolving technique employed by NEAT. Moreover, the comparison has not been made in the context of a partially observable procedurally generated environment.

2.2.5. The Gap This Study Addresses

It is evident that no existing study has conducted a direct, controlled comparison between DQN and NEAT on a procedurally generated, partially observable Pac-Man testbed using a consistent vector-based state representation for both algorithms. Prior DQN evaluations rely on fixed ROM layouts and pixel inputs, while prior NEAT evaluations focus on evolved behaviour within single fixed environments. Furthermore, no study isolates the generalisation capability of either paradigm by systematically varying maze topology across both training and evaluation phases. This study addresses that gap by constructing a bespoke environment that eliminates the confounds of pixel-based perception and fixed map memorisation, providing a

rigorous empirical basis for comparing the generalisation capabilities of both paradigms under identical, controlled conditions.

2.3. Memorisation vs. Generalisation

The core problem with training AI on a fixed environment is that the agent does not need to learn anything meaningful to score well. Machado et al. (2018) point out that in a fully deterministic game, where the map, pellet positions, and ghost behaviour are identical every episode, a neural network will simply memorise the correct sequence of actions for that specific layout rather than developing genuine spatial reasoning. Because training and evaluation conditions are identical, nothing exposes the fact that no real understanding has been developed.

The consequence becomes clear the moment the environment changes. Kirkpatrick et al. (2017) describe this through catastrophic forgetting, where a network encountering a new environment rapidly overwrites previously learned patterns, causing performance to collapse. This is not a fixable bug but a structural consequence of gradient-based learning. The present study operationalises this distinction by running both algorithms under a fixed-seed condition and a randomised condition, enabling direct measurement of performance degradation attributable to the absence of memorisable spatial patterns.

The most direct way to prevent memorisation is to ensure the agent never sees the same map twice. Şafak, Bostanci and Soyluçiçek (2016) argue that procedural generation must be built into the training loop rather than applied only at evaluation, forcing the agent to navigate using only what it can currently observe rather than recalled routes. However, full complexity from the first episode prevents consistent learning. Bengio et al. (2009) demonstrate that starting with simpler examples and gradually increasing difficulty produces faster convergence and stronger generalisation than random sampling across the full task distribution. Applied to Pac-Man, this motivates beginning with ghost-free mazes and progressively introducing ghost pressure, higher pellet requirements, and larger maps as foundational behaviours develop.

2.4. The Nature vs. Nurture Paradigms

2.4.1. The Nurture-Centric Paradigm: Deep Q-Networks

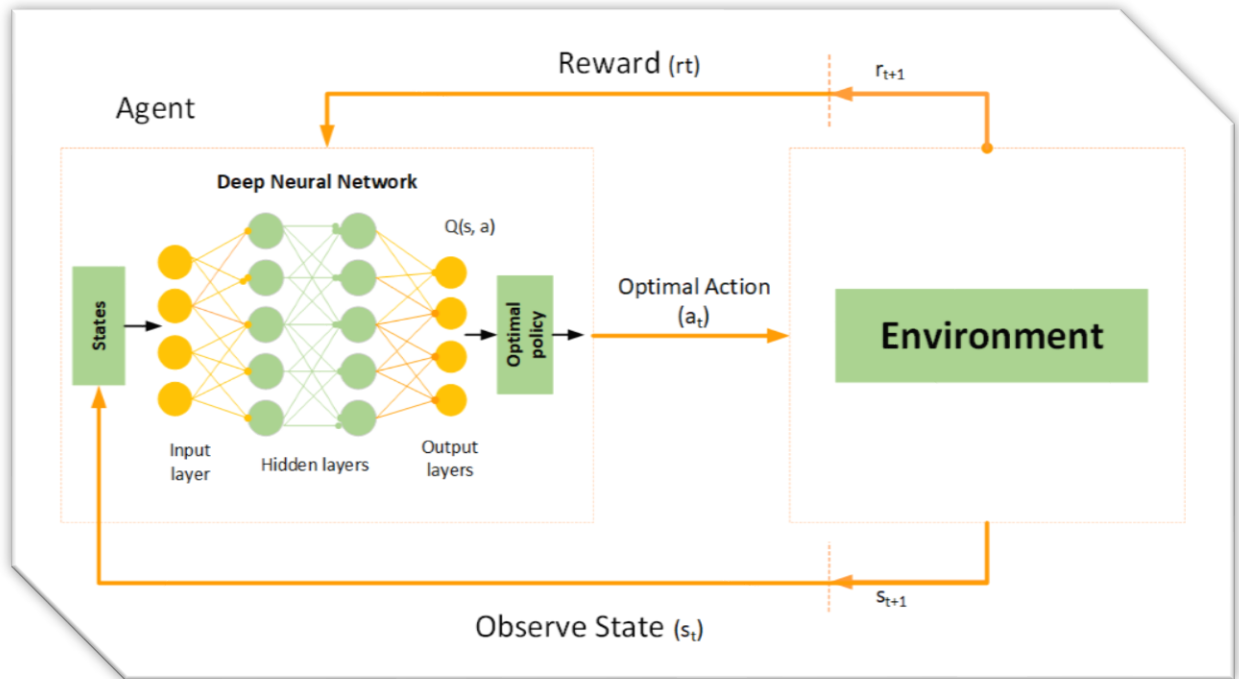


Figure 2.1 The Deep Q-Network agent-environment interaction loop

The reinforcement learning problem, as noted, is formulated through Markov Decision Processes. Here, an agent continuously views its current state, chooses an action, gets a reward, and transitions into a new state. The essential condition here is that the current state includes all the information required for an agent to choose an action, while the history is irrelevant. In terms of DQN, this becomes a foundation for using the Bellman equation to evaluate the total future expected reward for a certain action taken from a certain state.

DQN relies solely on experience to develop a policy. The agent, knowing nothing about the game when starting, tries different actions, makes observations, and adjusts its internal parameters to perform better next time. According to Terven (2025), this process may be described as the approach to learning with a blank slate, when the agent learns absolutely everything based on interactions and has nothing designed in advance. Eventually, as the Bellman equation is applied, the network reaches increasingly accurate conclusions about worthwhile and non-worthy actions.

This relationship is captured formally by the Bellman equation:

$$Q(s, a) = r + \gamma \cdot \max_{a'} Q(s', a')$$

In the context of Pac-Man, the equation says that the value of taking a particular action equals the immediate reward received, such as eating a pellet, plus the best possible future reward achievable from the resulting position, discounted by γ . The network trains by comparing its current prediction against what actually happened and adjusting its weights to reduce that gap across thousands of episodes.

DQN is not without weaknesses. The two most significant are sample inefficiency and catastrophic forgetting, where the network tends to overwrite previously learned behaviour when encountering new situations. Irpan (2018) notes these instabilities are especially pronounced in dynamic environments where the agent cannot rely on a fixed training layout. Three well-established extensions address these problems and are adopted in this implementation: Double DQN, which prevents overestimation of action values (Van Hasselt, Guez and Silver, 2016); Dueling DQN, which separates learning about state value from action preference (Wang et al., 2016); and Prioritised Experience Replay, which ensures the most informative past experiences are revisited more frequently (Schaul et al., 2016).

2.4.2. The Nature-Centric Paradigm: NeuroEvolution of Augmenting Topologies

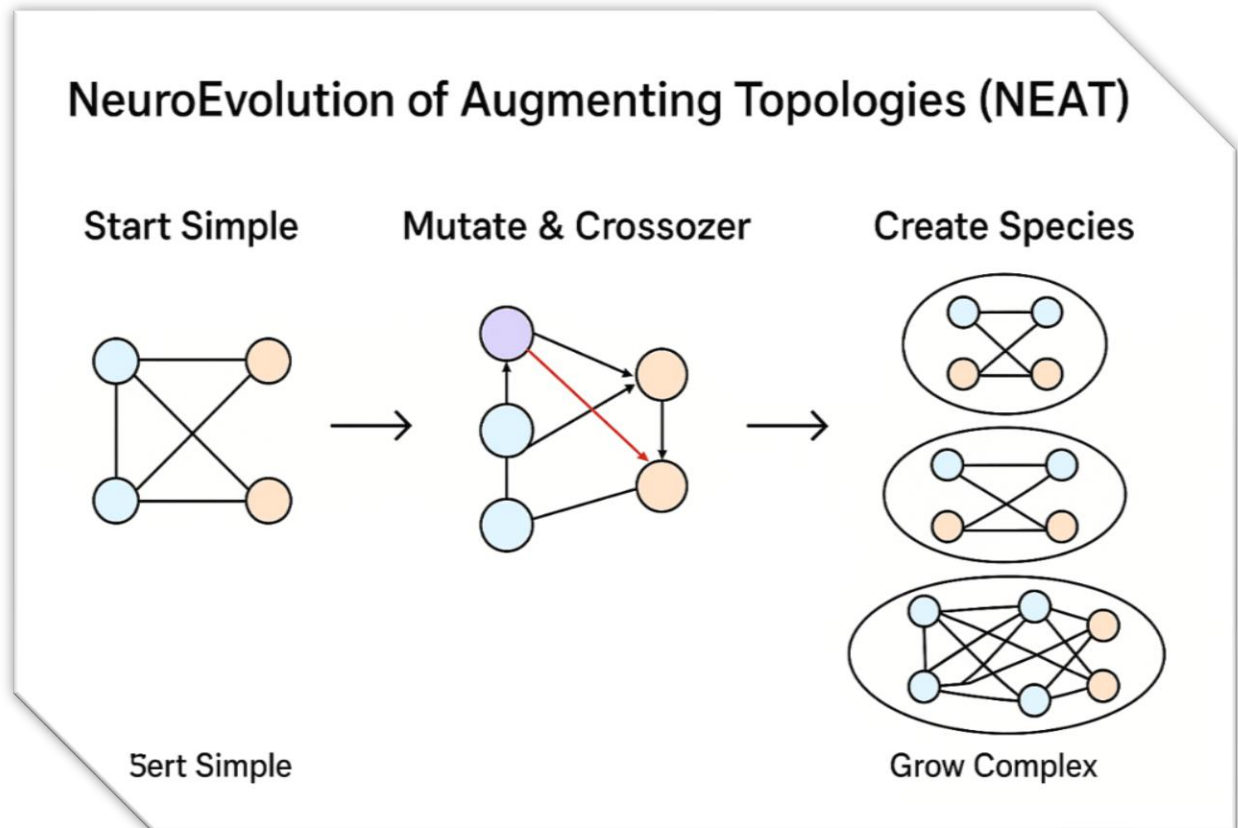


Figure 2.2 The NEAT evolutionary process

Where DQN learns by adjusting the weights of a fixed network through experience, NEAT simultaneously evolves both the weights and the structure of the network across a population of candidates. Stanley and Miikkulainen (2002) describe it as starting with minimal networks and growing complexity only when doing so proves beneficial through genetic selection. Rather than a single agent improving over time, NEAT evaluates a population of different networks each generation, selects the best performers to reproduce, and applies mutations that can add new connections or neurons over time.

This structural flexibility is particularly relevant to Pac-Man. Schrum and Miikkulainen (2016) demonstrate that NEAT networks can organically develop separate modules for pellet collection and ghost avoidance, evolving mechanisms to switch between them appropriately. A fixed-topology network must represent all behaviours within the same weight space, creating interference. An evolved network can separate them structurally.

NEAT also avoids catastrophic forgetting because learning happens at the population level rather than through individual gradient updates. Stanley and Miikkulainen (2002) further note that speciation protects new structural innovations from elimination before they have time to develop, preserving diversity and preventing premature convergence to suboptimal solutions.

2.5. Implementation Libraries

Implementation library choices are key decisions in engineering which will decide how well the results can be reproduced, extended, and validated for academic research purposes. There are three main technical pillars to the implementation: The first is Python itself as the programming language, the second is Gymnasium as the reinforcement learning interface standard, and the third is Pygame as the rendering engine and physics framework. Each decision is made with a particular purpose in mind.

2.5.1. Python

Python has become the definitive language for implementing machine learning and reinforcement learning. According to Raschka, Liu, and Mirjalili (2022), the reason why this programming language has achieved such prominence is due to its advanced numerical computation capabilities via the NumPy and PyTorch libraries, which ensure the solid mathematical foundation that is necessary for both the gradient-based and evolutionary optimization processes. Therefore, implementing both algorithms using a single piece of Python code is essential since it is mathematically guaranteed that neither approach will suffer from any performance penalty resulting from language integration issues. Moreover, the ease of modifying code and experimenting with different models provided by dynamic typing in this programming language will be very beneficial while developing the testbed.

2.5.2. Gymnasium

Gymnasium, the currently actively developed fork of OpenAI Gym (Towers et al., 2023), defines a standardized environment API through its `reset()` and `step()` contract that outputs observations, rewards, termination states, and additional diagnostics. The reason Brockman et al. (2016) have chosen to create this standardized interface as a benchmark is that it enables completely clean abstraction from the environment’s specifics, making sure that any algorithm adhering to the contract could be evaluated against any environment adhering to the same contract using no algorithm-specific environment code whatsoever. Such an approach is a clear

benefit to the current research because the wrapper of the custom Pac-Man engine ensures that both the DQN and NEAT evaluations utilize the same environment API. Clearly, such architecture design allows avoiding one possible confounder: any algorithmic performance differences observed in this setting can be safely attributed to the algorithms’ properties because they operate on the same environment.

2.5.3. Pygame

Pygame (Shinners, 2011), being a cross-platform library designed specifically for developing games, gives direct programmatic control over rendering, input, and timing using SDL as the foundation. Unlike high-level engines such as Unity or Unreal, which have hidden physics and proprietary renderers, Pygame lacks any internal opaque physics solver. Therefore, considering that the empirical testbed requires strict determinism for exact collision detection in tiles, it is much better to use a library that provides explicit mathematical control of the state-space.

Another advantage of using Pygame is that it allows running in headless mode, which means the absence of a graphical backend while training the agents. As can be easily understood, it is extremely important to exclude any graphical overhead to enable efficient massive parallelised fitness evaluation. However, in addition to saving time, this approach ensures the ability to perform qualitative visual checks when needed.

2.6. Professional, Ethical, and Social Context

The British Computer Society’s code of conduct requires that software systems are trustworthy, safe, and fit for purpose in the conditions they will actually encounter. This is directly relevant to reinforcement learning research because models that perform well in controlled training conditions but fail unpredictably in novel ones cannot be considered safe for real-world deployment. Terven (2025) note that the catastrophic forgetting problem in DRL represents a genuine safety liability, an agent that overwrites learned survival behaviours when encountering new situations is fundamentally unreliable, and deploying such a system in a safety-critical context would be irresponsible. Evaluating generalisation under varied conditions is therefore not just a research interest but a professional obligation.

There is also an important ethical dimension to how the two algorithms in this study differ in interpretability. A trained DQN is a dense neural network with millions of weight interactions that cannot be easily inspected or audited. Zahavy, Baram and Mannor (2017) highlight this

opacity as a significant concern, particularly in autonomous systems where understanding why a decision was made matters as much as the decision itself. NEAT produces structurally simpler evolved networks where the connections between sensor inputs and action outputs can be directly visualised and traced. This transparency makes it easier to identify failure modes, audit decision pathways, and build trust in the system’s behaviour. In addition to its performance characteristics, this interpretability advantage is a meaningful consideration when evaluating which paradigm is more appropriate for responsible deployment.

2.7. Summary

The literature establishes a clear and consistent picture. Standard reinforcement learning agents, including DQN, achieve high scores on fixed game environments not through genuine understanding but through memorisation of effective action sequences. When the environment changes, this memorisation fails, and performance collapses. NEAT offers a structurally different approach that evolves network architecture alongside weights, potentially producing more compact and adaptable policies, but its generalisation capability under procedurally varied conditions has not been directly quantified against DQN under controlled, comparable settings.

This project addresses that gap. By deploying both algorithms on a custom procedurally generated Pac-Man testbed where no training maze is ever repeated in testing, the study provides a direct empirical comparison of their generalisation capabilities under identical conditions. The central question, whether a nature-centric evolutionary approach or a nurture-centric gradient-based approach produces more robust behaviour on unseen environments, remains open in the literature and is what this dissertation sets out to answer.

3. Design and Implementation

This chapter describes the design and implementation of all components produced during this project. It begins with the procedural maze generation system and its validation, followed by the ghost behaviour and pathfinding implementation, the Gymnasium-compliant environment interface, the state representation and reward dynamics, and the curriculum learning system. The DQN and NEAT algorithm implementations are then described in detail, concluding with the experimental setup and evaluation methodology. Together these sections document every technical decision made in the construction of the testbed and training pipelines, justifying each choice in the context of the research objectives.

3.1. Engineering the Testbed

Existing Reinforcement Learning testbeds are generally built as closed, rendering-focused systems, with limited support for direct control over environment structure. This makes it difficult to precisely manipulate topology or enforce deterministic conditions during experimentation.

To enable a fair and controlled comparison between gradient-based and evolutionary approaches, a bespoke, fully object-oriented game engine was developed from scratch. The engine includes custom grid-based physics, deterministic adversarial pathfinding and controllable topological generation, allowing the environment to be configured in a consistent and reproducible manner.

This level of control is necessary as without it, enforcing the strict conditions required for reliable zero-shot generalisation testing would be computationally impractical.

3.1.1. Procedural Maze Generation

At the start of each episode a completely new maze is generated from scratch. The generator works on a grid, carving paths using a depth-first search with backtracking to ensure all areas are reachable while maintaining a corridor-based structure. This produces layouts that vary significantly between episodes while keeping properties such as corridor width and branching consistent, ensuring navigation difficulty does not vary unpredictably. Generation is seed-based, meaning specific layouts can be reproduced for controlled evaluation. Adjusting the window or tile size allows maze scale to be modified while passing the same validation checks.

A fixed ghost exclusion zone is included in every generated maze, providing a consistent spawn area for ghosts. Outside this zone the layout is fully random, preventing the agent from relying on memorised routes or positions and forcing policy decisions to be based on current observations rather than prior experience.

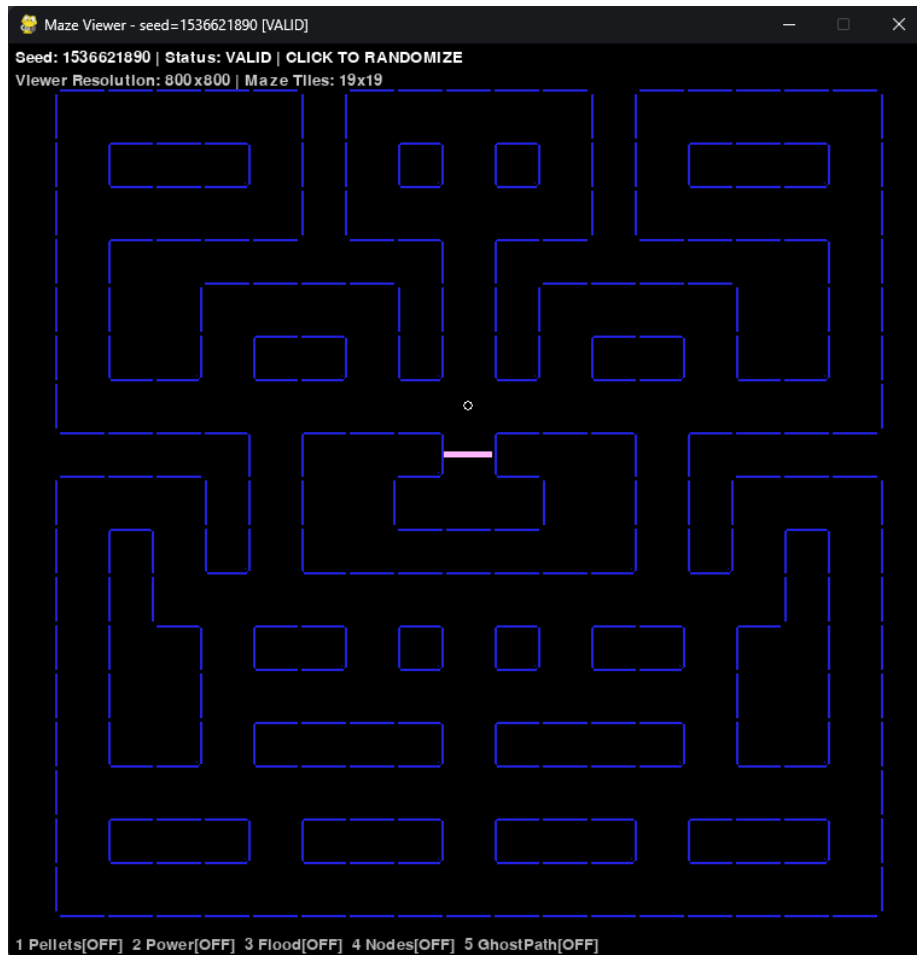


Figure 3.1 Example of generated maze layout

3.1.2. Maze Validation

Procedural generation introduces the risk of invalid environments. Without proper constraints, disconnected areas or unreachable objectives can be produced, introducing noise into reward signals and distorting fitness evaluation in evolutionary approaches.

An earlier validation heuristic only checked whether the spawn position could reach the four corners of the maze. This failed to account for internal topology and allowed layouts containing isolated enclosed regions that were impossible to access.

This was replaced with a strict BFS flood fill validation. After each maze is generated, BFS is performed from the starting position to identify all reachable tiles. This set is compared against the total number of traversable tiles, and a maze is only accepted if full connectivity is confirmed, meaning every non-wall tile is reachable from the start. Any maze failing this check is discarded and regenerated.

This guarantees all pellets and power pellets are accessible, no isolated regions exist, and every generated maze admits at least one valid solution path. By enforcing strict topological validity, the environment ensures performance differences are attributable to the learning algorithms rather than to flaws in map generation.



Figure 3.2 Previous validation only checked paths to the 4 corners

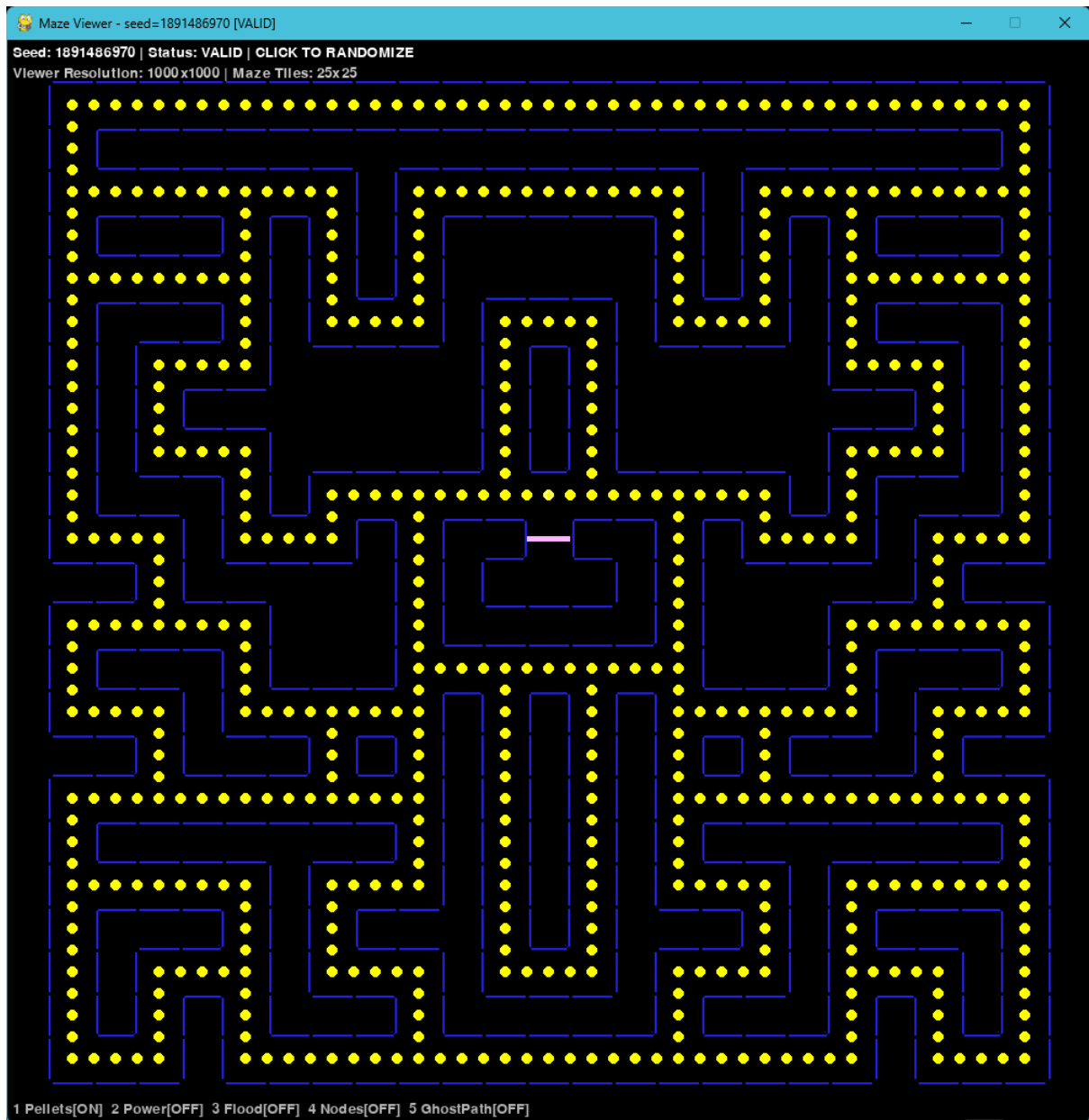


Figure 3.3 Debugging Pellets Layout

3.1.3. Tile-Lock Stepping

A naive approach would let the agent make a decision on every game frame. At 60 frames per second a typical episode would consist of over 10,000 individual decision points, the vast majority of which are identical since Pac-Man moves only a few pixels between frames. This creates two problems: the agent receives near-identical observations on consecutive steps, and rewards such as eating a pellet are temporally distant from the actions that caused them, making credit assignment unreliable.

Rather than returning a new observation every frame, the environment advances the game engine internally until Pac-Man reaches the centre of the next tile or the episode ends. A typical episode compresses from over 10,000 frame-level steps to roughly 100 to 300 tile-level decisions. All rewards accumulated during the internal loop are summed and returned as a single value, ensuring feedback corresponds directly to the move just made rather than to an arbitrary frame boundary.

3.2. Action Space and Reward Dynamics

3.2.1. Action Space

The action space is a discrete set of four egocentric actions: forward, left, right, and backward, interpreted relative to Pac-Man's current heading rather than in absolute compass directions. This ensures that geometrically equivalent situations produce identical observations and actions regardless of orientation, reducing the effective state space the agent must learn. The backward action is retained because removing it would prevent valid evasive manoeuvres during ghost pursuit. If the chosen action is blocked by a wall, a penalty of -5 is applied and the game state is not advanced, discouraging the agent from repeatedly selecting invalid directions.

3.2.2. Reward Structure

Reward is accumulated across an internal tick loop that advances the game engine until Pac-Man reaches the next tile centre or the episode ends. All rewards from that loop are summed and returned as a single value at the decision point, aligning feedback with meaningful spatial transitions rather than arbitrary frame boundaries. The reward distribution is shown in Table 3.1

Reward Component	Value	Trigger Condition
Pellet collected	+10	Per normal pellet eaten.
Power Pellet collected	+50	Per power pellet eaten.
Ghost eaten	+200	Per frightened ghost eaten.
Win	+1000	All required pellets eaten.
Death	-500	Ghost collision.
Starvation	-100	No pellet eaten within limit.
Invalid action	-5	Action blocked by wall.

Living penalty	-0.5	Every decision step
Food-approach shaping	$\pm(\text{delta BFS food signal}) \times 5.0$	Movement toward or away from nearest pellet.
Danger-approach shaping	$\pm(\text{delta BFS danger signal}) \times 3.0$	Movement toward or away from nearest ghost.

Table 3-1 Reward signal components and their associated values

The reward magnitudes establish a clear priority hierarchy. The win reward dominates at +1000, ensuring the primary objective is unambiguous. Death at -500 is significant without overwhelming the win signal. The living penalty and shaping rewards provide dense continuous signal between sparse terminal events, which is vital for stable DQN gradient updates and for producing meaningful fitness differentials across NEAT genomes.

For DQN specifically, all rewards are normalised to approximately zero mean and unit variance before being used in Q-value target computation, compressing the effective range and preventing the 2000x spread between smallest and largest signals from causing gradient instability.

3.3. Ghost Behaviour & Adversarial Pathfinding

Each ghost implements a distinct targeting strategy, collectively creating multi-directional pressure that prevents the agent from relying on repetitive movement patterns. Blinky pursues Pac-Man directly using A* pathfinding, providing constant optimal-path pressure. Pinky targets a tile four steps ahead of Pac-Man's current heading, enabling interception rather than pure chase and forcing the agent to account for future positioning. Inky computes its target by projecting two tiles ahead of Pac-Man, forming a vector from Blinky's current position to that projected point and doubling it, producing a target that shifts dynamically with both Pac-Man's movement and Blinky's position. This makes Inky the least predictable of the four. Clyde applies a distance threshold rule: it pursues Pac-Man directly when further than eight tiles away, but retreats to its corner scatter position once within that range, creating an inconsistent threat that redistributes itself across the maze over time.

All four ghosts operate under a shared state system with five states. In CHASE, each ghost applies its individual targeting function. In SCATTER, all ghosts abandon pursuit and move toward fixed corner positions, dispersing across the maze before re-engaging. FRIGHTENED is triggered by power pellet consumption, halving ghost speed, reversing their direction, and

replacing A* pathfinding with random directional choice which inverts the threat dynamic and creating a pursuit opportunity for the agent. EATEN causes the ghost to path find back to the spawn cage and transition to SPAWNING, during which it waits for a fixed timer before re-entering the maze. Ghosts cycle between CHASE and SCATTER on a configurable timer, with each transition forcing an immediate direction reversal that disrupts current movement trajectories and prevents the agent from predicting ghost positions through timing alone.



Figure 3.4 Blinky always targets Pac-Man's current grid position

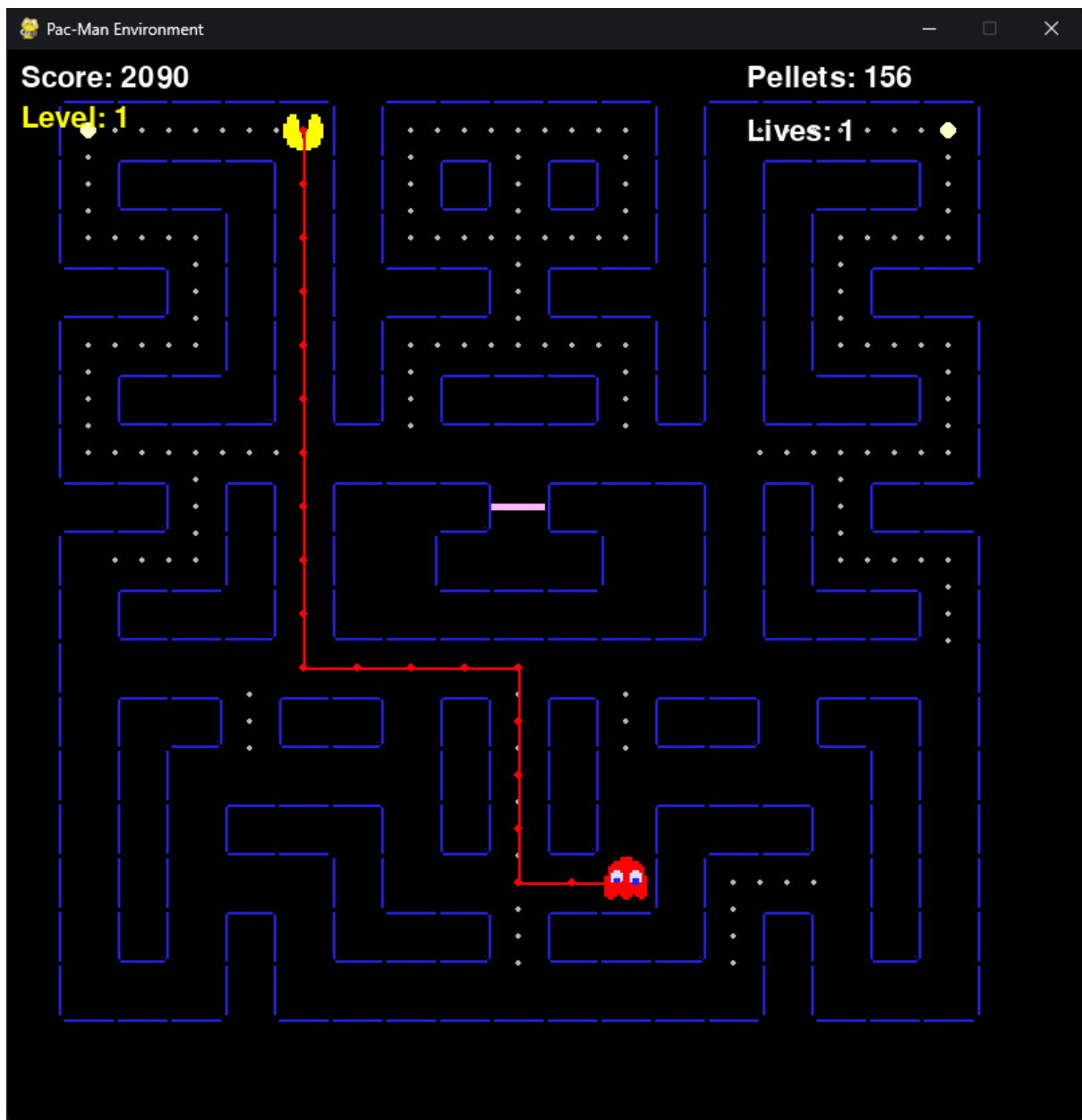


Figure 3.5 Blinky navigating through the maze via A^* pathfinding

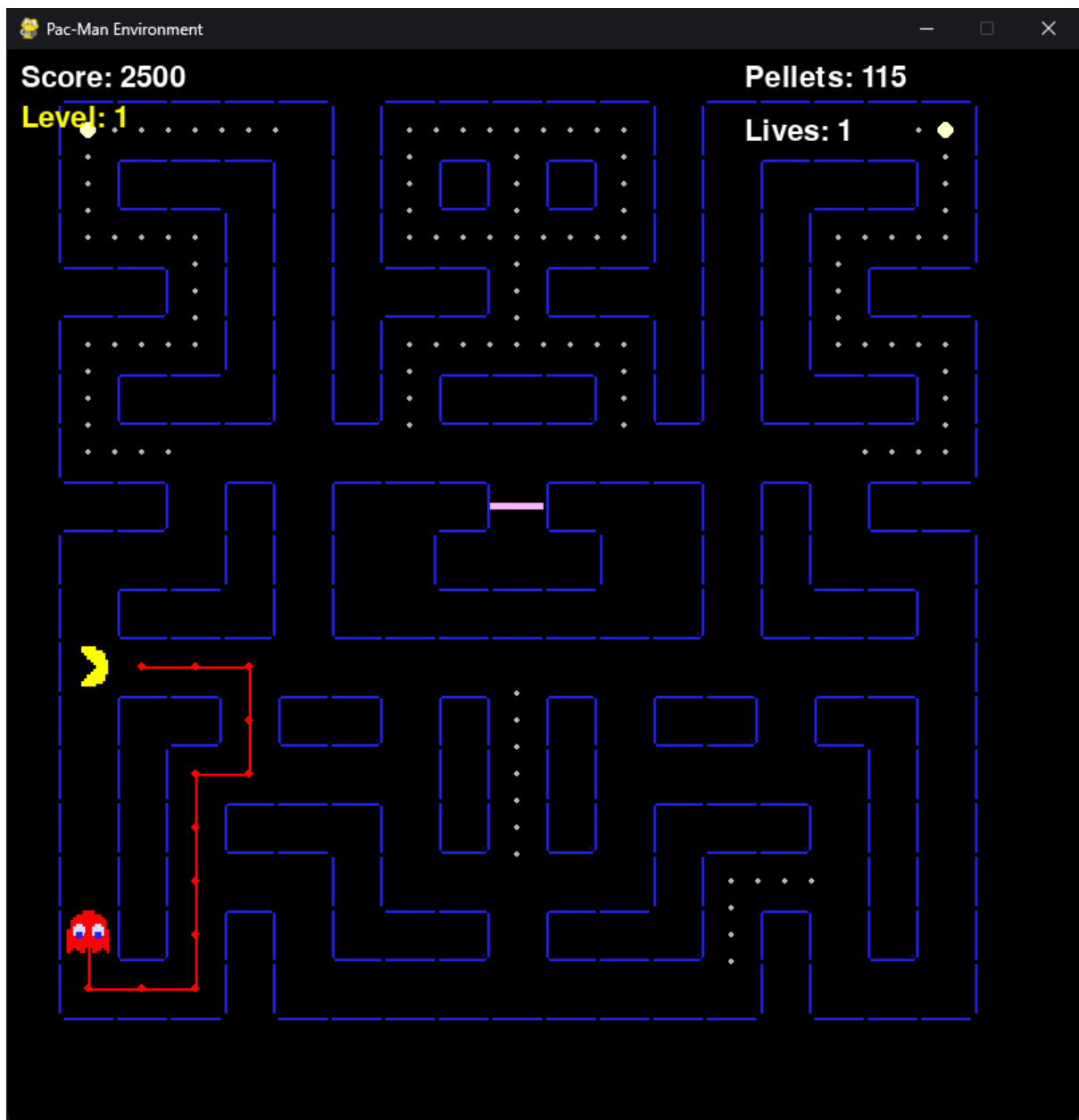


Figure 3.6 Ghosts are not allowed to make sudden U-turns even if it means it takes longer to get to its target tile



Figure 3.7 Pinky always targets 4 tiles ahead of Pac-Man



Figure 3.8 Pinky targeting ahead of a wall, preventing successful interception



Figure 3.9 Pinky positioned ahead of Pac-Man, demonstrating successful interception



Figure 3.10 Clyde targeting Pac-Man's current position when more than 8 tiles away



Figure 3.11 Clyde retreating to its corner scatter position when within 8 tiles of Pac-Man



Figure 3.12 Inky computing its target tile using Blinky's position and a projected point two tiles ahead of Pac-Man, with both paths visible

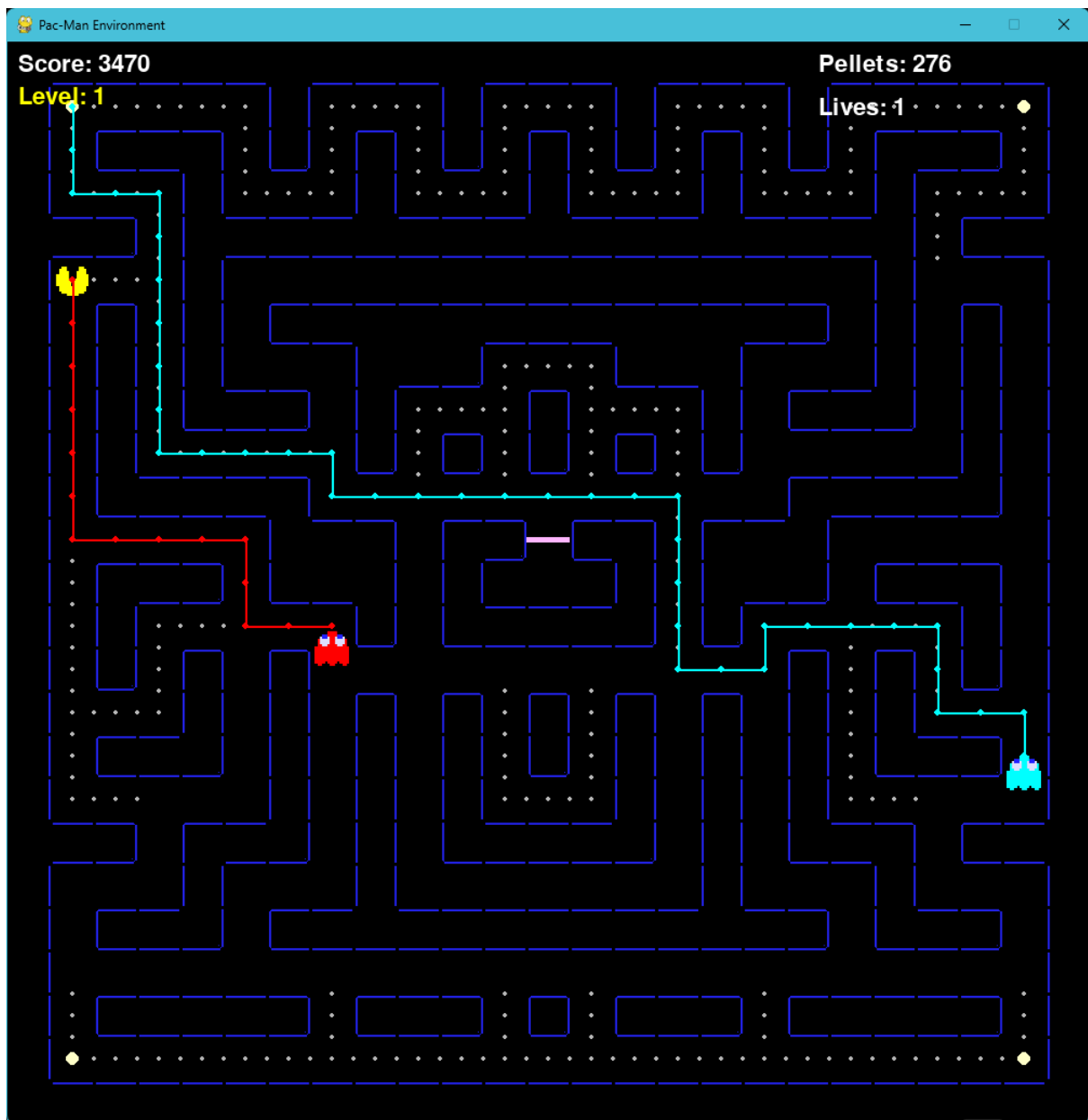


Figure 3.13 Inky's resulting movement trajectory, showing how its target shifts dynamically as both Pac-Man and Blinky move

3.4. Curriculum Learning

Exposing a randomly initialised agent to full game complexity from the outset consistently produces unstable learning. The agent fails to develop foundational navigation behaviour before being overwhelmed by the constant ghost pressure, resulting in degenerate strategies and poor convergence. To address this, a staged curriculum is implemented in which environmental difficulty is introduced gradually, and progression is gated by demonstrated performance rather than fixed episode counts.

The curriculum is managed by a dedicated CurriculumManager class that maintains a rolling performance window of 150 episodes and advances the agent to the next stage only when a sustained win rate threshold is met. A window of 150 rather than a smaller value is important because with noisy ghost-stage outcomes, a shorter window allows lucky streaks to trigger premature promotion, sending the agent into harder conditions before the learned behaviour has genuinely consolidated.

3.4.1. Stage-Based Progression

The curriculum consists of 8 stages. Stages 0 and 1 disable ghosts entirely and increase the required pellet collection ratio from 20% to 60%, establishing reliable navigation before any ghost pressure is introduced. Stage 2 introduces a single slow ghost operating predominantly in scatter mode, giving the agent its first exposure to ghost presence in a low-pressure context. Stage 3 is a dedicated ghost-eating bridge stage where power pellets are enabled and a ghost eat bonus of 150 points per ghost is applied on top of the standard 200-point reward. This stage was added specifically to address a failure mode observed in earlier training where agents collected pellets competently but never developed ghost-eating behaviour, leaving them without an offensive response when power pellets were available. Stage 4 removes the bonus and increases ghost speed, requiring the agent to have internalised ghost-eating without an explicit incentive. Stages 5 and 6 add Pinky and then Inky progressively. Stage 7 represents the full game configuration with all four ghosts active at speed 1.85 and a 100% pellet completion requirement.

The full stage configuration is summarised in Table 3.2

<i>Stage</i>	<i>Ghosts</i>	<i>Gh. Speed</i>	<i>Power Pellet</i>	<i>Win Ratio</i>	<i>Purpose</i>
0	None	N/A	No	0.20	Basic Navigation
1	None	N/A	No	0.60	Full navigation mastery
2	Blinky	1	No	0.50	Ghost awareness, low pressure
3	Blinky	1.5	Yes	0.55	Ghost eating bridge
4	Blinky, Pinky	2.0	Yes	0.60	Two-ghost interaction
5	Blinky, Pinky, Inky	2.5	Yes	0.70	Three-ghost pressure
6	All four	2.9	Yes	1.0	Full game evaluation

Table 3-2 Curriculum learning stages configuration

3.4.2. Promotion Criteria

Promotion triggers only when two conditions are simultaneously satisfied over the 150-episode window. First, the overall win rate must meet the stage-specific threshold: 65% for stages 0 to 2, 70% for stages 3 to 5, and 65% for stages 6 and above. Second, the win rate over the most recent 5 episodes must remain within 5 percentage points of the promotion threshold, acting as a stability filter that prevents promotion on the basis of a declining trend. When promotion occurs the rolling window is cleared so the agent begins the new stage without carry-over performance history.

Demotion is not implemented. Prior experimentation showed that demotion caused oscillation between stages, disrupted the replay buffer distribution for DQN, and degraded partially learned policies without producing recovery. Stability at harder stages is instead enforced through the stricter promotion criteria and the larger rolling window.

3.4.3. Starvation Scaling

The starvation limit is computed dynamically as starvation limit = base ticks + (ratio x scale), where ratio is the current stage's pellets to win ratio. This ensures the starvation budget scales proportionally with required task complexity, preventing the agent from surviving by wandering aimlessly on harder stages where a fixed limit would be too generous.

3.5. State Representation

In reinforcement learning and neuroevolution, the agent’s input vector is the sole basis for every decision made during an episode. Raw pixel input was explicitly rejected for this implementation. Pixel-based approaches require the agent to simultaneously solve a visual perception problem and a control problem, conflating two distinct learning challenges. Using a structured vector means both algorithms focus purely on decision-making, ensuring any observed performance difference is attributable to the learning paradigms themselves rather than to perceptual architecture. The environment state is represented as a 29-dimensional floating-point vector organised into three functional blocks.

3.5.1. Block A: Egocentric Raycasts (indices 0–23)

The first 24 values are constructed through ray-based sensing across four cardinal directions: up, down, left, and right. Diagonal rays were excluded because in a tile-locked corridor-based maze they do not correspond to valid movement axes and would introduce perceptual noise without actionable benefit. Each ray is traced tile-by-tile until a wall or boundary is encountered, recording six channels per direction: inverse wall distance, inverse distance to the nearest normal pellet, inverse distance to the nearest power pellet, inverse distance to the nearest lethal ghost, inverse distance to the nearest frightened ghost, and visit saturation. All distances use the inverse-distance form $1/(1+d)$, ensuring proximate entities produce stronger signals.

The lethal and frightened ghost channels are encoded separately rather than as a single signed channel. An earlier implementation used one signed channel where positive values indicated lethal ghosts and negative values indicated frightened ones. This conflated distance magnitude with threat semantics, creating an ambiguous signal where the same value could represent a nearby frightened ghost or a distant lethal one. Separating them eliminates this ambiguity.

The visit saturation channel encodes the fraction of open tiles along each ray already visited during the episode, providing a directionally specific novelty signal that allows the agent to identify unexplored corridors without requiring a separate exploration reward.

Once computed globally, all four ray blocks are reordered into Pac-Man’s egocentric frame based on current heading, mapping directions to forward, left, right, and backward. This ensures geometrically equivalent situations are perceived identically regardless of absolute

compass orientation, preventing the network from needing to learn four separate representations of the same spatial configuration. The reordering mappings are shown in Table 3.3

Heading	Forward	Left	Right	Backward
Facing UP	UP	LEFT	RIGHT	DOWN
Facing DOWN	DOWN	RIGHT	LEFT	UP
Facing LEFT	LEFT	DOWN	UP	RIGHT
Facing RIGHT	RIGHT	UP	DOWN	LEFT

Table 3-3 Egocentric Directions

During development an error was identified in the original heading mapping table where the left, right, and backward channels were incorrectly assigned for three of the four headings. The correct mappings are shown in table 3.4. This was corrected prior to the final training run.

Heading	Incorrect Mapping	Correct mappings
UP	[0, 1, 2, 3]	[0, 2, 3, 1]
DOWN	[1, 0, 3, 2]	[1, 3, 2, 0]
LEFT	[2, 3, 1, 0]	[2, 1, 0, 3]
RIGHT	[3, 2, 0, 1]	[3, 0, 1, 2]

Table 3-4 Correct Heading Mappings

3.5.2. Block B: BFS Global Features (indices 24–26)

Three global awareness features are computed through Breadth-First Search shortest-path distances: inverse distance to the nearest normal pellet (index 24), nearest lethal ghost (index 25), and nearest frightened ghost (index 26). BFS is used rather than Euclidean distance because Euclidean proximity does not respect maze topology, allowing the agent to incorrectly infer that threats are nearby when separated by walls. Results are cached per tile position for the episode since maze walkability is static within an episode.

3.5.3. Block C: Power State (indices 27–28)

Index 27 is a binary indicator specifying whether frightened mode is active. Index 28 encodes the normalised remaining frightened duration. Together these allow the agent to distinguish between normal threat-avoidance conditions and temporary ghost-hunting opportunities.

Without this context an agent observing a nearby frightened ghost would have no explicit signal distinguishing the situation from a lethal encounter.

3.5.4. Normalisation

All 29 values are normalised to $[-1.0, 1.0]$ via the linear transformation $2x - 1$. In DQN this prevents biased activations and reduces poorly conditioned optimisation. In NEAT it prevents individual features from dominating evolved behaviour purely through magnitude rather than informational content.

3.6. DQN Implementation

Deep Q-Networks learn entirely from experience. The agent starts with no knowledge of the game and gradually builds a strategy by trying actions, observing what happens, and adjusting its internal weights to make better predictions over time. The core mechanism is the Bellman equation, which estimates how much total future reward a particular action is likely to produce from a given situation. The network improves by comparing what it predicted would happen against what actually happened and closing that gap across thousands of episodes until the predictions become reliable.



Figure 3.14 DQN Visual Training

3.6.1. Network Architecture - Dueling DQN

The DQN agent uses a Dueling Q-Network architecture as proposed by Wang et al. (2016). The network receives the 29-dimensional observation vector and passes it through two shared fully connected layers of 256 units each, building a general understanding of the current game state. This shared representation then splits into two separate branches. The value branch estimates how good the current situation is regardless of which action is taken. The advantage branch estimates which specific action is better than average in that situation. The final Q-value combines both as

$$Q(s, a) = V(s) + (A(s, a) - \text{mean } A(s, a))$$

Separating these two concerns is particularly useful in Pac-Man where most corridor movement is roughly equivalent in value and only junction decisions genuinely matter.

3.6.2. Double DQN - Preventing Overconfident Predictions

A known problem with standard DQN is that it tends to overestimate how good its own actions are, which causes the policy to become unstable. Double DQN, introduced by Van Hasselt, Guez and Silver (2016), fixes this by maintaining two separate networks. The policy network is updated continuously during training and selects which action to take next. The target network is a slower-moving copy that evaluates how good that action actually is. By separating selection from evaluation, the overestimation feedback loop is broken. The target network updates gradually at each step using a blend coefficient of $\tau = 0.005$, keeping it stable enough to produce reliable scores while still tracking the policy network's progress.

3.6.3. Prioritised Experience Replay - Learning From Mistakes First

Rather than replaying past experiences in random order, the agent uses Prioritised Experience Replay as introduced by Schaul et al. (2016), which replays the most informative experiences more frequently. Each stored transition is assigned a priority based on how wrong the network's prediction was for that transition. Transitions where the network was most mistaken are sampled more often. A SumTree data structure makes this priority-weighted sampling efficient. New transitions receive the maximum current priority to ensure they are seen at least once before being assessed. The priority exponent $\alpha = 0.6$ controls how strongly priority influences sampling, with importance-sampling weights correcting the statistical bias this introduces.

These weights are gradually increased from 0.4 to 1.0 over 2,000,000 steps. The buffer stores up to 200,000 transitions.

3.6.4. Exploration - Balancing Discovery and Exploitation

Early in training the agent explores randomly, trying different actions to discover what works. As training progresses it gradually shifts toward exploiting what it has learned. This is controlled by epsilon, which starts at 1.0 and decays exponentially to a minimum of 0.15 over 3,000,000 steps. During exploration, only physically valid directions are sampled, though an 8% chance of selecting a wall-blocked direction is intentionally retained so the agent learns the penalty for invalid actions. When the curriculum promotes the agent to a harder stage, epsilon is temporarily raised to a minimum of 0.20 for 50,000 steps, ensuring the agent re-explores the new configuration rather than immediately applying a strategy built for easier conditions.

3.6.5. Parallelised Environment Execution

DQN training was conducted across eight parallel environment instances simultaneously using `gymnasium.vector.SyncVectorEnv`. Rather than completing one episode before starting the next, eight independent games run at the same time, each on a different randomly generated maze. This provides eight times the experience per training step and ensures the replay buffer contains a much wider variety of maze geometries and ghost configurations than a single environment could provide. This design directly targets the generalisation problem observed in prior training, where agents trained on sequential episodes developed avoidance strategies that were too specific to the mazes they happened to encounter.

3.6.6. Reward Normalisation

The raw reward signal spans a wide range, from -500 for a ghost death to +1000 for winning. This makes it difficult for the network to learn efficiently because the same learning rate must handle both very small and very large values. To address this, rewards are normalised to approximately zero mean and unit variance before being used in training, using a running calculation that adapts automatically as the curriculum advances. Normalised values are capped at $[-10, 10]$ as a safety bound.

3.6.7. Training Configuration

The network is optimised using the Adam optimiser with a learning rate starting at 3×10^{-4} and decaying to 1×10^{-4} over 1,000,000 update steps. Gradient magnitudes are capped at 1.0 per update to prevent large reward values from causing sudden destabilising weight changes. Training does not begin until 10,000 transitions have been collected, ensuring early batches are sufficiently varied. The network updates every 4 environment steps using a batch of 128 transitions. All hyperparameters are summarised in Table 3.5

<i>Hyperparameter</i>	<i>Value</i>
<i>Observation dimension</i>	29
<i>Hidden layer size (shared)</i>	256×2
<i>Hidden layer size (streams)</i>	128×1
<i>Replay buffer capacity</i>	200,000
<i>Batch size</i>	128
<i>Learning rate (initial)</i>	3×10^{-4}
<i>Learning rate (final)</i>	1×10^{-4}
<i>Discount factor γ</i>	0.995
<i>Epsilon start</i>	1.0
<i>Epsilon end</i>	0.10
<i>Epsilon decay steps</i>	2,000,000
<i>Exploration jolt on promotion</i>	0.20 for 50,000 steps
<i>Target network τ (soft update)</i>	0.005
<i>PER alpha</i>	0.6
<i>PER beta start $\rightarrow$ end</i>	0.4 $\rightarrow$ 1.0
<i>PER beta anneal steps</i>	2,000,000

<i>Warmup steps</i>	10,000
<i>Train frequency</i>	Every 4 steps
<i>Gradient clip norm</i>	1.0

Table 3-5 DQN Hyperparameters

3.7. NEAT Implementation

Where DQN learns by adjusting the weights of a fixed network through experience, NEAT takes a fundamentally different approach. Rather than one agent improving over time, NEAT maintains a population of 150 different neural networks, each with its own structure and connections. Every generation, each network plays the game, receives a fitness score based on how well it performed, and the best performers are selected to reproduce. Reproduction involves copying successful networks with small random mutations, gradually evolving the population toward more capable solutions. Crucially, NEAT does not just tune weights within a fixed structure. It also discovers the structure itself, starting minimal and growing only when doing so proves beneficial.

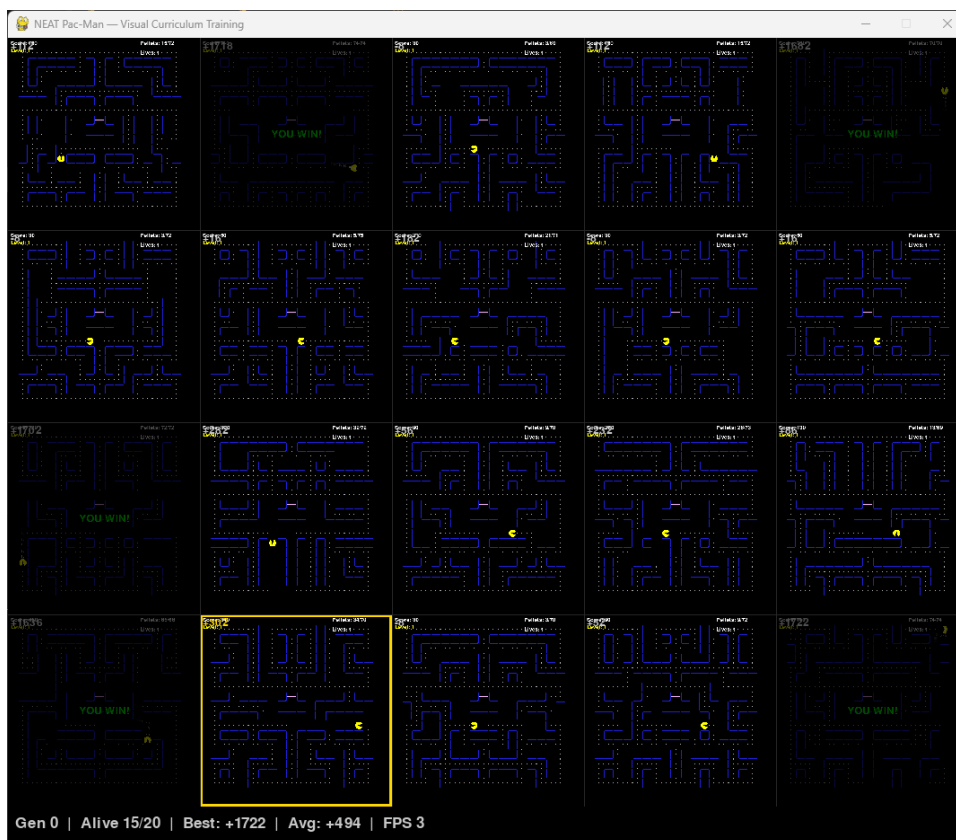


Figure 3.15 NEAT Visual Training

3.7.1. Starting Minimal and Growing

Every genome in the initial population begins with no hidden neurons, only direct connections between the 29 input nodes and the 4 output nodes, with approximately 50% of possible connections present at the start. This minimal starting point prevents the search from wasting effort on unnecessary complexity. New nodes and connections are added only through mutation, meaning the final evolved network is shaped by what the task actually required rather than what a designer assumed it would need.

3.7.2. Fitness Function

Each genome is evaluated by letting it play three episodes on different procedurally generated mazes and averaging the total reward across all three. Using multiple seeds per evaluation prevents a genome from scoring well due to a lucky maze layout rather than genuine capability. The genome acts greedily at every step, selecting the action with the highest output activation from the currently valid moves. Unlike DQN, there is no exploration randomness so diversity across the population comes from the variety of evolved network structures rather than from deliberate random action.

3.7.3. Speciation, Protecting New Ideas

A key challenge in NEAT is that adding a new node or connection initially makes a genome perform worse, because the new structure has untrained weights. In a standard evolutionary algorithm this genome would simply be eliminated before it has time to improve. NEAT protects new structural innovations by grouping similar genomes into species and making them compete only within their own group. A new structural mutation is therefore shielded from immediate elimination, giving it time to develop before being judged against dissimilar genomes. Species membership is determined by a compatibility distance based on the number of differing connections and the average weight difference between matching ones, with a threshold of 2.0. Any species that shows no fitness improvement for 20 consecutive generations is removed, with the exception that the top 2 species are always preserved.

3.7.4. Selection and Reproduction

Within each species, only the top 25% of genomes by fitness survive to reproduce. The top 2 genomes in each species are carried forward unchanged as elites, preventing the best-known solutions from being accidentally destroyed. Remaining new genomes are produced through mutation of surviving parents. The mutation rates applied per reproduction are weight mutation at 80%, bias mutation at 70%, new connection addition at 50%, connection deletion at 30%, new node addition at 20%, and node deletion at 10%. Weight and bias values are bounded to $[-30, 30]$.

3.7.5. Parallelisation

Because each genome evaluation is entirely independent, NEAT evaluations are distributed across all available CPU cores simultaneously using Python's multiprocessing module. A generation of 150 genomes is split across six concurrent worker processes, each running its own isolated game instance. The GPU is not used during NEAT training as the small feedforward networks involved do not benefit from GPU acceleration at this scale.

3.7.6. Configuration Summary

<i>Parameter</i>	<i>Value</i>
<i>Population size</i>	150
<i>Initial hidden nodes</i>	0
<i>Initial connectivity</i>	50% direct input→output
<i>Network type</i>	Feedforward only
<i>Fitness function</i>	Cumulative episode reward
<i>Activation (default)</i>	ReLU
<i>Activation (alternative)</i>	Tanh
<i>Weight mutate rate</i>	0.80
<i>Bias mutate rate</i>	0.70
<i>Add connection rate</i>	0.50
<i>Delete connection rate</i>	0.30
<i>Add node rate</i>	0.20
<i>Delete node rate</i>	0.10

<i>Activation mutate rate</i>	0.05
<i>Compatibility threshold</i>	3.5
<i>Disjoint coefficient</i>	2.0
<i>Weight coefficient</i>	0.5
<i>Max stagnation (generations)</i>	20
<i>Species elitism</i>	2
<i>Genome elitism per species</i>	2
<i>Survival threshold</i>	0.25

Table 3-6 NEAT Hyperparameters

The complete NEAT configuration file is provided in Appendix C

3.8. Evaluation Methodology

3.8.1. The Generalisation Problem

Training performance alone is not a meaningful measure of generalisation capability. An agent achieving a high win rate during training may have memorised reward-maximising patterns specific to its training mazes rather than developing transferable spatial reasoning. To distinguish genuine generalisation from memorisation, training and evaluation must use entirely separate maze seeds that never overlap.

3.8.2. Seed Partitioning

All maze seeds are drawn from a deterministic integer range and partitioned into two non-overlapping sets before training begins. Seeds 0 to 9,999 form the training pool, from which maze configurations are randomly sampled throughout all curriculum stages for both DQN and NEAT. Seeds 10,000 to 10,099 form the held-out test pool, comprising 100 seeds that neither algorithm encounters under any circumstances during training. No seed from the test pool is used in a training episode and no seed from the training pool is used in a test episode, guaranteeing that evaluation performance reflects behaviour on genuinely unseen maze geometries.

3.8.3. Fixed-Seed Baseline Condition

In addition to the randomised training condition, a parallel training run was conducted using a fixed seed configuration in which both DQN and NEAT trained and were evaluated exclusively

on maze seed **22459265 (my student ID)**. This condition isolates peak performance on a single known environment, providing a direct baseline against which generalisation performance under randomised conditions can be compared. A large performance gap between fixed and random conditions indicates policy dependence on memorised spatial patterns, while a small gap indicates transferable learned behaviour. All other training parameters remain identical between the two conditions.

3.8.4. Training Termination

Both algorithms train under the same early stopping criterion. Training is considered complete when the agent achieves a rolling win rate of at least 70% over the most recent 20 episodes at curriculum stage 7, the full game configuration with all four ghosts active and a 100% pellet completion requirement. If neither algorithm reaches this criterion within the maximum budget of 100,000 episodes per pipeline, training terminates at the budget limit and the best available checkpoint is used for evaluation.

3.8.5. Zero-Shot Generalisation Testing

Once training terminates, each agent is frozen and evaluated against all 100 held-out test seeds sequentially. For DQN, epsilon is set to 0.0, meaning the agent acts entirely from its learned policy with no random exploration. For NEAT, the champion genome achieving the highest fitness across all training episodes is loaded without modification. No reward signal, weight updates, or curriculum feedback are provided during testing. Each test episode runs to natural termination via win, death, or starvation with no step limit imposed. Two test conditions are evaluated for each agent: a reached-stage test at the highest curriculum stage achieved during training, and a fixed-stage-7 test at full game difficulty for all four conditions simultaneously.

3.8.6. Statistical Analysis

An independent samples t-test is used to compare mean test reward between DQN and NEAT on the random seed condition, with Levene’s test applied first to verify the equal variance assumption. A proportions z-test compares win rates as binary outcomes. One-way ANOVA assesses whether mean reward differs significantly across curriculum stages, validating the effectiveness of the staged difficulty progression. All tests use a significance threshold of $p < 0.05$.

3.8.7. Metrics Recorded

Both algorithms log an identical set of metrics for every training and test episode via a shared CSV schema. The full set of recorded metrics is defined in Table 3.7

<i>Metric</i>	<i>Type</i>	<i>Description</i>
<i>Outcome</i>	Categorical	Episode termination cause: WIN, GHOST, STARVATION, or MAX_STEPS
<i>Win</i>	Binary (0/1)	1 if the episode ended in a win, 0 otherwise
<i>Reward</i>	Float	Total accumulated reward
<i>Pellets collected</i>	Integer	Count of pellets eaten
<i>Power pellets collected</i>	Integer	Count of power pellets eaten
<i>Ghosts eaten</i>	Integer	Count of ghosts consumed
<i>Explore rate</i>	Float (0 to 1)	% maze tiles visited
<i>Macro steps</i>	Integer	Count of tile-level decisions
<i>Micro ticks</i>	Integer	Total game engine ticks

Table 3-7 Metrics recorded per episode during training and evaluation

3.9. Experimental Setup

All training and evaluation were conducted on a single local workstation running Windows 11, equipped with an Intel Core i7-8700 6-core 12-threads processor, 32GB of DDR4 RAM, and an NVIDIA GeForce RTX 4060 GPU with 8GB of VRAM.

The two algorithms place fundamentally different demands on this hardware, and the setup was exploited accordingly. DQN training is GPU-bound. Every forward pass through the Dueling Q-Network and every gradient update is executed on the RTX 4060 via CUDA, with PyTorch handling device placement automatically. The 32GB of system RAM ensures the full 200,000-transition replay buffer remains resident in memory throughout training without paging, which is significant because any buffer spill to disk would introduce latency that disrupts the fixed 4-step training frequency the agent relies on.

NEAT training is CPU-bound by design. Because each genome evaluation is entirely independent and requires no gradient computation, the population is evaluated in parallel across all six logical cores of the i7-8700 using Python’s multiprocessing module. This means a generation of 120 genomes is distributed across six concurrent worker processes, each running its own isolated PacManEnv instance headlessly. The GPU is idle during NEAT training entirely, as feedforward network activation in python-neat involves only small matrix multiplications that do not benefit from GPU offloading at this scale.

Both algorithms were trained headlessly with Pygame operating in dummy video mode, eliminating rendering overhead and ensuring that episode throughput reflects pure computational performance rather than display frame rate. The Gymnasium environment’s tile-lock stepping mechanism further reduces computational load by compressing each episode from tens of thousands of raw game ticks to a few hundred tile-level decision points, making full-episode evaluation tractable for both the sequential DQN training loop and the parallelised NEAT population evaluation.

Training was logged continuously to CSV throughout all runs, with DQN checkpointing every 200 episodes and NEAT checkpointing every 5 generations, allowing resumption without data loss in the event of interruption.

Training duration was recorded per episode and across the full pipeline via wall-clock timestamps logged to the CSV. A full 100,000-episode DQN run completed in approximately 18 hours on the described hardware, with NEAT completing its 100,000-episode budget in approximately 7 hours, the latter benefiting from population-level CPU parallelisation across all six cores reducing per-generation evaluation time substantially.

3.10. Summary

This chapter has described the full design and implementation of the Pac-Man testbed, both learning algorithm pipelines, and the evaluation framework used to assess generalisation. All implementation decisions have been justified with reference to the research objectives established in Chapter 1. Chapter 4 presents the results produced by all four training pipelines and the zero-shot generalisation tests conducted on held-out maze seeds.

4. Results and Testing

This chapter presents the results of all four training pipelines and the subsequent zero-shot generalisation tests. Results are reported factually without interpretation, covering training progression, curriculum stage advancement, algorithm diagnostics, and test performance across all conditions. Statistical test results are presented at the end of the chapter. Interpretation and critical analysis of these findings are reserved for Chapter 5.

4.1. Experimental Overview

Four training pipelines were executed as part of this study, representing two algorithms across two seed conditions. DQN was trained under both a randomised maze condition, in which a new procedurally generated maze was produced each episode, and a fixed seed condition using seed **22459265 (my student ID)**, in which the same maze layout was repeated throughout training. NEAT was trained under the same two conditions. All of the four pipelines ran for a maximum of 100,000 episodes each, with DQN training conducted across eight parallel environment instances to improve replay buffer diversity, and NEAT training conducted across a parallelised population of 150 genomes evaluated concurrently across all available CPU cores.

Following training, each pipeline was evaluated against two separate held-out test sets. The reached-stage test evaluated each agent on 100 procedurally generated maze seeds, drawn from the held-out pool of seeds 10,000 to 10,099, at the highest curriculum stage the agent achieved during training. The fixed-stage-7 test evaluated all four agents on the same 100 seeds at full game difficulty regardless of training stage reached, enabling a standardised cross-algorithm comparison at maximum complexity.

Condition	Episodes	Max Stage Reached	Pipeline Duration
DQN-random	100,000	Stage 4	17.69 hrs
DQN-fixed	100,000	Stage 5	13.09 hrs
NEAT-random	32,658	Stage 5	6.17 hrs
NEAT-fixed	100,000	Stage 7	7.53 hrs

Table 4-1 Training Configuration and Pipeline Outcomes for All Four Conditions

It is noted that NEAT-random completed only 32,658 of its 75,000-episode budget due to the shared pipeline elapsed time ceiling.

4.2. DQN Training Result

4.2.1. Curriculum Progression

The DQN agent under the random seed condition progressed through five curriculum stages, reaching stage 4 before the episode budget was exhausted. Stages 0 through 2 were cleared rapidly, with the agent spending 152, 352, and 944 episodes respectively before promotion. Stage 3, the ghost eating bridge, required 6,080 episodes before the 72% promotion threshold was met, producing a training win rate of 52.5% and an average of 0.239 ghost kills per episode. Stage 4, involving two ghosts at speed 1.3 with power pellets enabled, absorbed the remaining 92,472 episodes at an overall win rate of 27.8%. Ghost kills at stage 4 averaged 0.273 per episode, with 65,785 of the 92,472 episodes ending in ghost death.

The DQN fixed seed condition reached stage 5, one stage further than the random condition, spending 92,408 of its 100,000 episodes at stage 5. This is consistent with the memorisation hypothesis as repeated exposure to the same maze allowed the agent to meet promotion thresholds earlier at stages 3 and 4, advancing further before the budget was exhausted. Stage 4 under the fixed condition produced a win rate of 58.7% compared to 27.8% under the random condition, a substantial difference attributable to the agent's ability to learn layout-specific routing rather than generalised navigation.

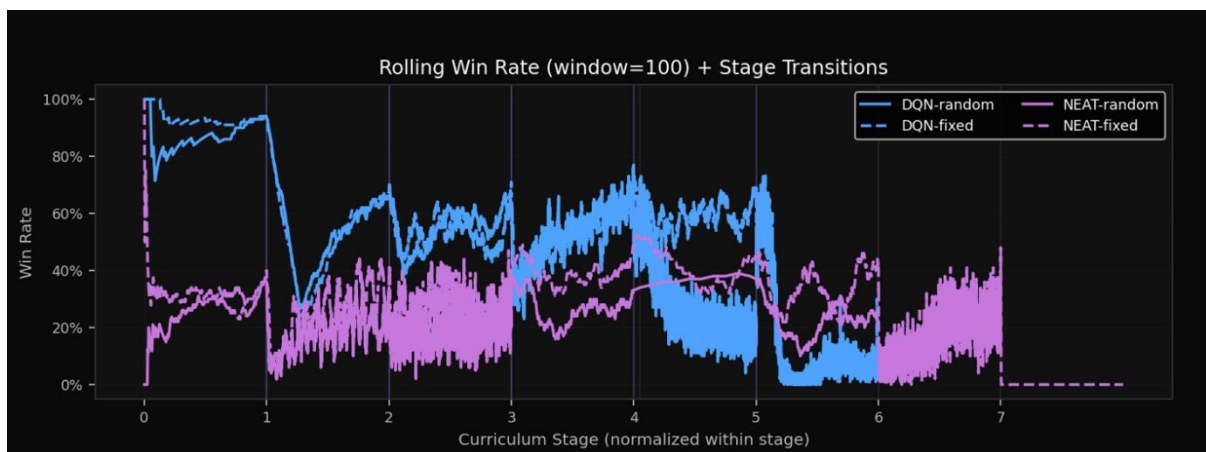


Figure 4.1 Training Progression Dashboard, Rolling Win Rate with Stage Transitions

4.2.2. Loss and Epsilon

The DQN average loss began at 0.040 and declined steadily to approximately 0.005 by the end of training under the random condition, indicating consistent network learning throughout the episode budget. Under the fixed condition, loss declined more rapidly, reaching 0.001 by stage 5, consistent with the network converging to a narrower policy shaped by repeated exposure to a single layout.

Epsilon decayed from 1.0 toward its minimum floor of 0.15 across the training run. The epsilon curve and rolling win rate moved in opposing directions during the exploration phase before stabilising as epsilon approached its floor around curriculum stage 4. The win rate decline observed during the later stage 4 episodes is consistent with the agent's policy being insufficiently generalised to sustain performance across varied procedurally generated mazes under greedy exploitation.

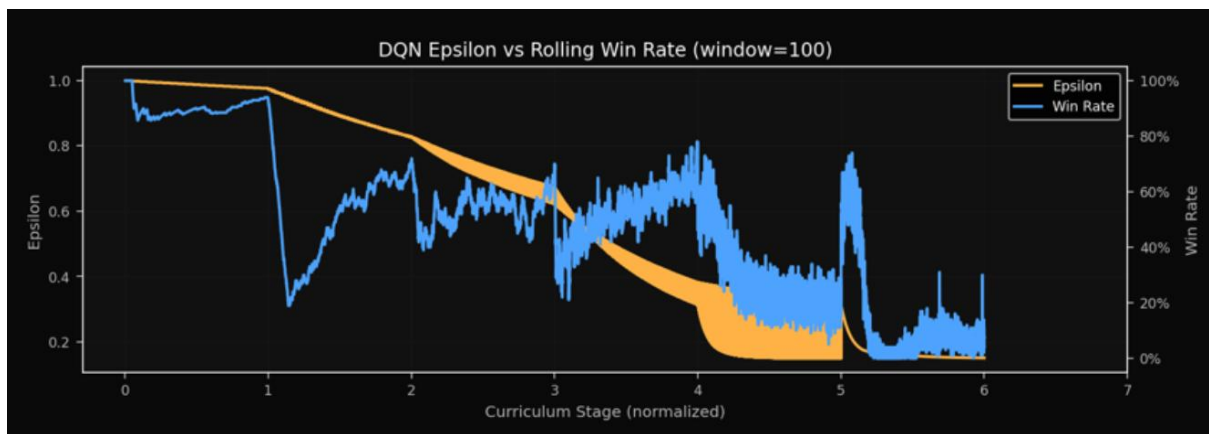


Figure 4.2 DQN Epsilon vs Rolling Win Rate

4.2.3. Ghost Interaction

Ghost eating behaviour emerged during stage 3 when power pellets were first introduced with the ghost eating bonus of 150 points per kill. Average ghost kills rose from 0.239 at stage 3 to 0.273 at stage 4, indicating that the behaviour persisted beyond the bridge stage but remained limited. Power pellet consumption averaged 1.764 per episode at stage 3 and dropped to 1.143 at stage 4, suggesting the agent became more conservative with power pellet usage as ghost speed increased.

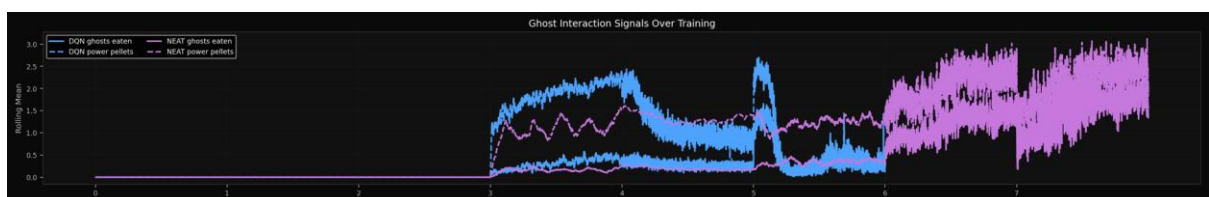


Figure 4.3 Ghost Interaction Signals Over Training

4.3. NEAT Training Results

4.3.1. Curriculum Progression

The NEAT agent under the random seed condition progressed through six curriculum stages within its 32,658-episode budget, reaching stage 5 before the pipeline time ceiling was reached. Navigation stages 0 through 2 required 262, 4,715, and 26,667 episodes respectively, with stage 2 absorbing the largest portion of the random budget. The high starvation rate at stage 2 (13,610 starvation outcomes from 26,667 episodes) indicates that NEAT’s population took significant time to evolve reliable pellet collection strategies on varied maze geometries. Stage 3 required only 486 episodes before promotion, suggesting the ghost eating bonus produced strong fitness signal that drove rapid population improvement. Stage 4 was cleared in only 10 episodes, a transitional anomaly reflecting the promotion window carrying forward momentum from stage 3. Stage 5 received 518 episodes with a 20.1%-win rate before the budget was exhausted.

The NEAT fixed seed condition reached stage 7, the full game configuration with all four ghosts active at speed 1.85 and a 100% pellet completion requirement. This is the highest curriculum stage reached by any condition in this study. The agent spent 48,656 episodes at stage 6 (win rate 17.5%) and 36,767 episodes at stage 7 (win rate 0.0%). At stage 7, the agent averaged 1.445 ghost kills and 2.113 power pellet consumptions per episode, demonstrating active engagement with frightened-mode mechanics despite being unable to clear the full pellet requirement within the time budget.

4.3.2. Fitness and Species

NEAT best fitness under the fixed condition rose from approximately 3,100 at generation 50 to a plateau of around 5,000 from generation 100 onward, with occasional spikes reaching 5,800 during later stage 6 training. Average fitness tracked behind best fitness throughout, settling around 1,800 to 2,000 across most of the run, reflecting natural diversity within the population. Under the random condition, best fitness plateaued at approximately 3,100 from generation 50 onward, substantially lower than the fixed condition due to the harder generalisation objective. The gap between best and average fitness remained consistent across both conditions, indicating that speciation was successfully protecting structural innovations throughout training.

Species count began at approximately 32 in generation 1 and declined to a stable range of 10 to 15 across most of the training run for both conditions. This pattern is consistent with healthy speciation dynamics, an initial phase of structural exploration followed by convergence toward viable topological families while retaining sufficient diversity to avoid premature population collapse.

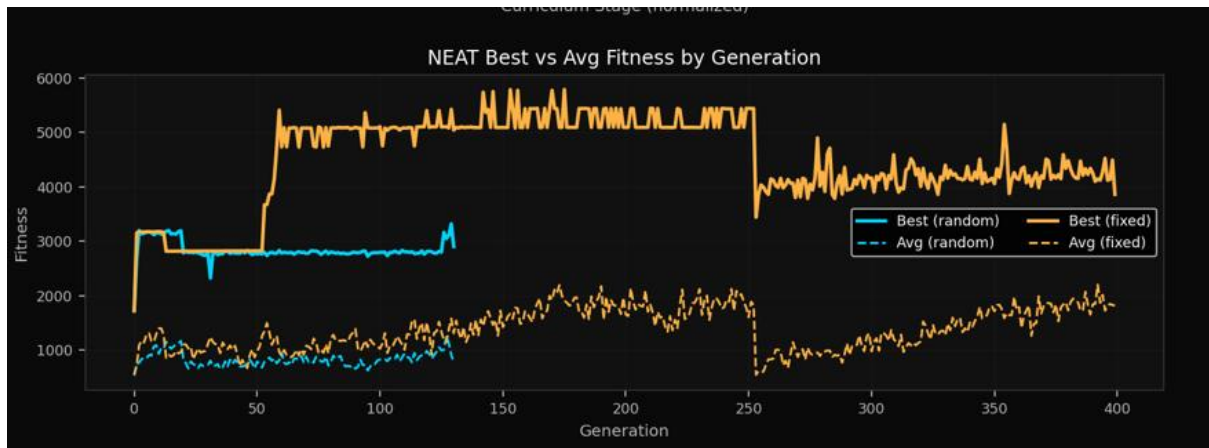


Figure 4.4 NEAT Best vs Avg Fitness by Generation

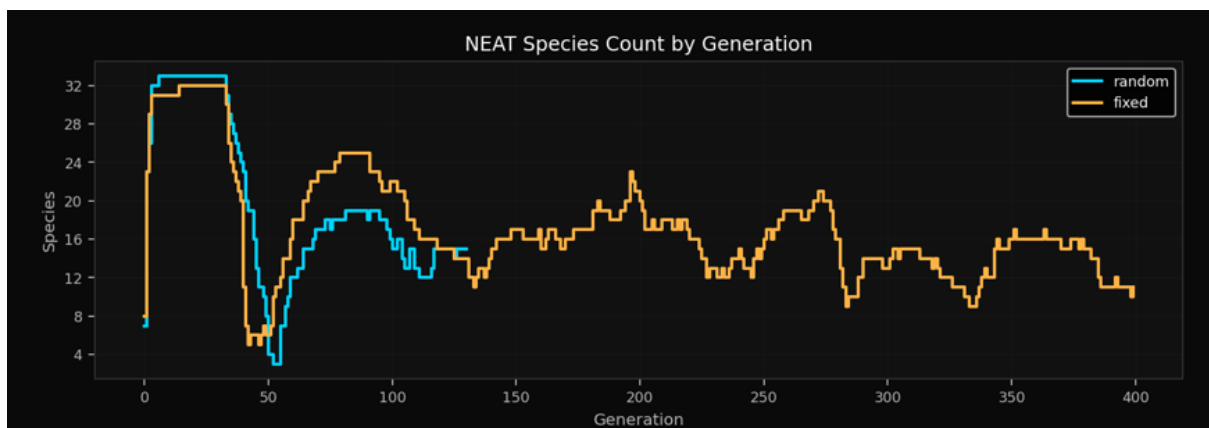


Figure 4.5 NEAT Species Count

4.4. Zero-Shot Generalisation Test Results

4.4.1. Reached-Stage Test

Condition	Test Stage	N	Win Rate	Mean Reward	SD	Median Reward	Mean Pellets	Mean Explore Rate	Mean Ghosts Eaten
DQN-random	Stage 4	100	26%	912.4	1358.5	279.1	97.9	27.9%	0.17
DQN-fixed	Stage 5	100	6%	254.0	928.0	-75.9	61.4	17.1%	0.28
NEAT-random	Stage 5	100	37%	1581.4	1281.7	1162.4	141.7	40.3%	0.32
NEAT-fixed	Stage 7	100	0.0%	414.7	85.4	366.7	84.0	23.9%	0.24

Table 4-2 Zero-Shot Generalisation Results Across All Four Conditions (Reached-Stage Test)

NEAT-random achieved the highest performance across all metrics including win rate, mean reward, median reward, pellet collection, spatial coverage, and ghost consumption. NEAT-random’s median reward of 1162.4 versus DQN-random’s median of 279.1 indicates that NEAT’s advantage is not driven by outlier episodes but reflects consistently higher performance across the majority of test seeds.

NEAT-fixed achieved 0%-win rate despite reaching stage 7 during training, reflecting the increased difficulty of the test condition. The low standard deviation of 85.4 on NEAT-fixed reward indicates the agent produced very consistent results, with nearly all 100 episodes clustering around the mean of 414.7. This behavioural consistency is characteristic of an evolved network that has converged to a stable but limited strategy.

DQN-fixed achieved 6.0%-win rate at stage 5 difficulty, the only fixed-seed condition to record any wins on test, though the negative median reward of -75.9 indicates that the majority of episodes ended quickly with net negative reward, meaning wins were outlier events rather than reliable policy behaviour.



Figure 4.6 Fixed vs Random Seed Comparison

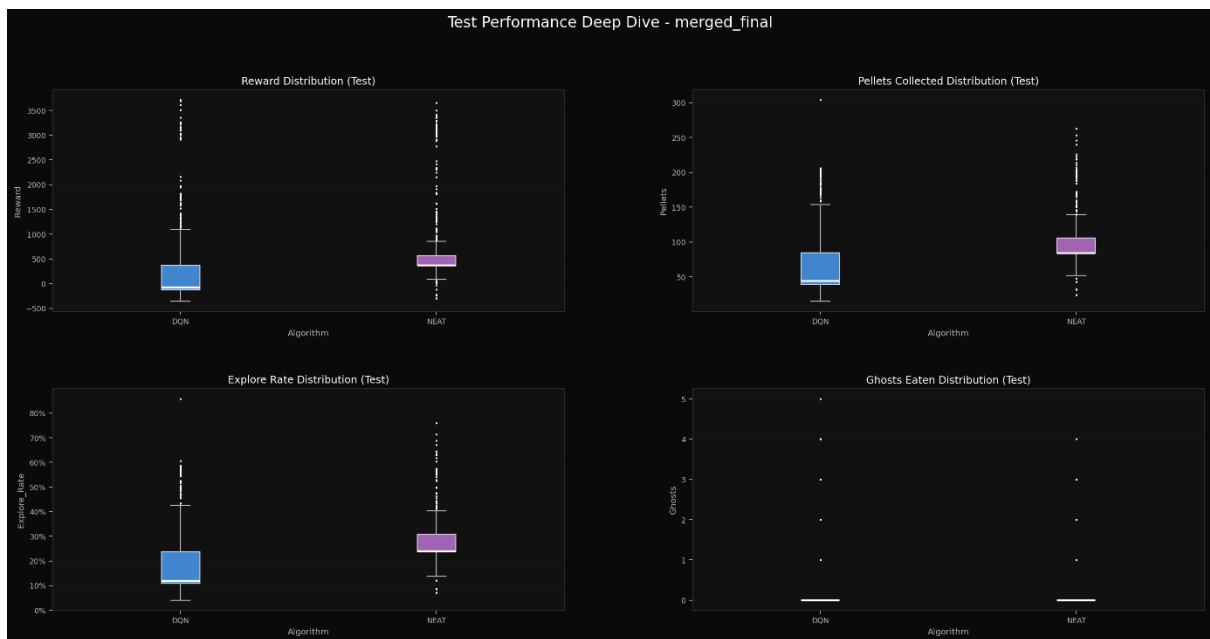


Figure 4.7 Test Performance Deep Dive box plots

4.4.2. Fixed-Stage-7 Test

When all four conditions were evaluated at identical full game difficulty, no condition achieved any wins across 100 test episodes.

Condition	n	Win Rate	Mean Reward	SD	Median Reward	Mean Pellets
DQN-random	100	0.0%	265.6	696.9	-65.3	68.6
DQN-fixed	100	0.0%	156.8	639.5	-114.1	60.3
NEAT-random	100	0.0%	625.0	720.9	373.7	105.7
NEAT-fixed	100	0.0%	424.7	90.8	366.7	84.0

Table 4-3 Zero-Shot Generalisation Results Across All Four Conditions (Fixed Stage 7 Test)

Despite the absence of wins, NEAT-random produced the highest mean reward (625.0) and median reward (373.7) at full game difficulty, followed by NEAT-fixed (424.7), DQN-random (265.6), and DQN-fixed (156.8). All four conditions terminated exclusively in ghost deaths across all 100 test episodes.

4.5. Statistical Analysis

Test	Comparison	Statistic	p-value	Significant
Levene's test	DQN vs NEAT reward variance	W=0.644	p=0.423	No (equal variance confirmed)
Independent t-test	DQN-random vs NEAT-random reward	t(198)=-3.564	p<0.001	Yes ***
Welch's t-test	DQN vs NEAT pellets collected	t(198)=-4.994	p<0.001	Yes ***
Welch's t-test	DQN vs NEAT explore rate	t(198)=-4.923	p<0.001	Yes ***

Welch's t-test	DQN vs NEAT ghosts eaten (test)	$t(198)=-2.072$	$p=0.040$	Yes *
Proportions z-test	DQN vs NEAT win rate	$z=-1.674$	$p=0.094$	No
Independent t-test	DQN vs NEAT reward (stage 7 test)	$t(198)=-3.566$	$p<0.001$	Yes ***
Welch's t-test	DQN fixed vs random reward	$t(198)=-3.982$	$p<0.001$	Yes ***
Welch's t-test	NEAT fixed vs random reward	$t(198)=-9.037$	$p<0.001$	Yes ***
Proportions z-test	DQN fixed vs random win rate	$z=-3.858$	$p<0.001$	Yes ***
Proportions z-test	NEAT fixed vs random win rate	$z=-6.738$	$p<0.001$	Yes ***
One-way ANOVA	DQN reward across stages	$F=680.3$	$p<0.001$	Yes ***
One-way ANOVA	NEAT reward across stages	$F=85.97$	$p<0.001$	Yes ***

Table 4-4 Statistical Test Results for All Comparisons

Note: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

The primary comparison between DQN-random and NEAT-random on the reached-stage test yielded a statistically significant difference in mean reward, $t(198) = -3.564$, $p < 0.001$. Levene's test confirmed equal variance ($W = 0.644$, $p = 0.423$), validating the use of the standard independent samples t-test. The proportions z-test comparing win rates (26% vs 37%) did not reach statistical significance at the conventional threshold, $z = -1.674$, $p = 0.094$, though the result approached significance. NEAT's advantage is therefore better captured by continuous reward than by binary win rate, as NEAT accumulated substantially more reward in episodes that did not result in a win.

The memorisation effect was confirmed statistically for both algorithms. When evaluated on unseen mazes, both algorithms performed significantly worse under the fixed seed training condition than under the random seed condition, despite the fixed condition enabling greater curriculum progression during training. DQN-fixed achieved a mean test reward of 254.0 compared to DQN-random’s 912.4, a statistically significant difference, $t(198) = -3.982$, $p < 0.001$. NEAT showed an even larger gap, with NEAT-fixed achieving 414.7 versus NEAT-random’s 1581.4, $t(198) = -9.037$, $p < 0.001$. These results confirm that training on a fixed layout produces policies that are more dependent on memorised spatial patterns and less transferable to novel environments, directly supporting the generalisation argument established in the literature review. One-way ANOVA confirmed that the curriculum produced statistically significant reward differences across stages for both DQN ($F = 680.3$, $p < 0.001$) and NEAT ($F = 85.97$, $p < 0.001$), validating the staged difficulty progression as an effective training design.

4.6. Outcome Distribution



Figure 4.8 Outcome Analysis Dashboard

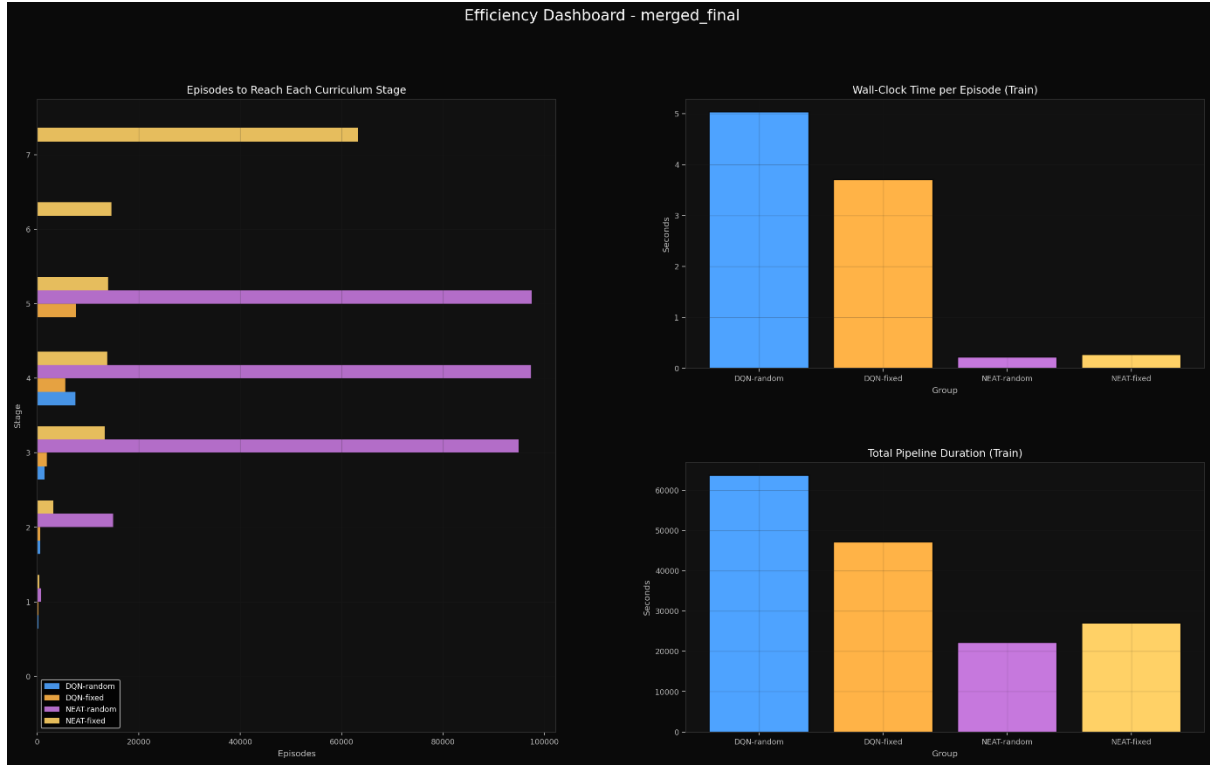


Figure 4.9 Efficiency Dashboard

The dominant failure mode for both algorithms across all test conditions was ghost collision, accounting for 73 of 100 DQN-random test episodes and 53 of 100 NEAT-random test episodes. Starvation was rare in both cases, 1 episode for DQN and 10 for NEAT, confirming that both agents had developed functional pellet collection behaviour and were not failing from lack of navigation capability but from inability to consistently avoid ghost contact on unseen maze geometries.

DQN averaged 5.04 seconds per episode during random training (17.69 hours total pipeline duration) compared to NEAT's 0.22 seconds per episode (6.17 hours for 32,658 episodes). The episode duration difference of approximately 23x reflects the fundamental computational contrast between sequential gradient-based parameter updates and parallelised feedforward network evaluation across an evolved population.

4.7. Summary

This chapter has reported the training and test results for all four conditions across both algorithms and both seed regimes. The primary finding is that NEAT-random achieved the highest test performance of any condition, with a statistically significant advantage over DQN-random in mean reward, pellet collection, spatial coverage, and ghost interaction. Chapter 5

evaluates these results critically against the original objectives, analyses the memorisation and generalisation effects, and assesses the extent to which the nature versus nurture hypothesis is supported by the evidence.

5. Evaluation

This chapter critically evaluates the results presented in Chapter 4 against the original aims and objectives of the study. The central research question is whether a nature-centric evolutionary approach (NEAT) or a nurture-centric gradient-based approach (DQN) produces greater zero-shot generalisation capability in a procedurally generated, partially observable Pac-Man environment. The evaluation is structured around five analytical themes: training efficiency, curriculum effectiveness, generalisation performance, behavioural analysis, and hypothesis evaluation. Statistical evidence from Chapter 4 is used throughout to ground interpretive claims in empirical findings rather than subjective observation.

5.1. Training Efficiency

The two algorithms place fundamentally different demands on computational resources, and these differences have direct implications for their practical viability as training paradigms.

DQN-random required 17.69 hours to complete 100,000 episodes at an average of 5.04 seconds per episode. This cost reflects the sequential nature of gradient-based training each episode must complete before the next begins, the GPU processes network forward passes and gradient updates on every training step, and the eight parallel environments, while improving experience diversity, all feed into a single sequential training loop. The 5.04 second average episode duration at stage 4 reflects the longer episodes produced by the larger maze and higher ghost complexity at that stage.

NEAT-random completed 32,658 episodes in 6.17 hours at an average of 0.22 seconds per episode, approximately 23 times faster per episode than DQN. This efficiency advantage stems from NEAT’s population-level parallelism as 150 genomes are evaluated simultaneously across all available CPU cores, and each genome evaluation involves only a fast feedforward pass through a typically sparse evolved network rather than any gradient computation. The t-test comparing episode duration confirmed this difference is statistically significant, $t(132,656) = 313.03$, $p < 0.001$, though the large sample sizes make statistical significance trivial here, the practical magnitude is what matters, and a 23x per-episode speed advantage is operationally significant.

However, raw episode speed does not tell the whole story. NEAT-random only completed 32,658 episodes versus DQN’s 100,000 because the shared pipeline time ceiling was reached. Despite this, NEAT-random produced the highest test performance of all four conditions. This suggests NEAT achieves more useful learning per episode than DQN under randomised conditions, a point with direct implications for sample efficiency discussed further in section 5.4.

NEAT-fixed completed 100,000 episodes in 7.53 hours, reaching stage 7 which is the full game, a milestone no previous training run in this study achieved. This demonstrates that given sufficient episodes, NEAT’s evolutionary search can navigate a complex 13-stage curriculum to full game difficulty, albeit without achieving wins at that level within the available budget.

5.2. Curriculum Effectiveness

One-way ANOVA confirmed that mean reward differed significantly across curriculum stages for all four training conditions: DQN-random ($F = 680.3$, $p < 0.001$), DQN-fixed ($F = 2223.2$, $p < 0.001$), NEAT-random ($F = 85.97$, $p < 0.001$), and NEAT-fixed ($F = 167.5$, $p < 0.001$). These results confirm that the staged difficulty progression produced measurable learning at each curriculum level rather than uniform performance across stages, validating the curriculum as an effective training design for both paradigms.

The ghost eating bridge at stage 3 produced the most clear evidence of curriculum effectiveness. Prior to stage 3, neither algorithm recorded any ghost kills because ghost interaction signals remained at zero throughout stages 0 to 2 in all four conditions. From stage 3 onwards, ghost kills emerged and persisted through subsequent stages. DQN averaged 0.239 kills per episode at stage 3, rising to 0.273 at stage 4. NEAT-fixed averaged 0.202 at stage 3, rising to 1.445 at stage 7. The explicit ghost eating bonus of 150 points per kill, applied only at stage 3, was instrumental in bootstrapping this behaviour. Without the bridge stage, previous training runs confirmed that agents progressed through ghost stages without developing any ghost eating behaviour at all, leaving them defenceless against ghosts on unseen maps.

The curriculum also revealed an important asymmetry between the two algorithms. DQN progressed through early ghost stages rapidly under the fixed condition, with stage 4 win rates of 58.7% under fixed versus 27.8% under random. This gap confirms that DQN’s gradient-

based updates were learning layout-specific routing that satisfied promotion thresholds without developing genuinely transferable behaviour. NEAT showed a similar but less extreme pattern, NEAT-fixed reached stage 7 while NEAT-random reached only stage 5, but the test performance gap between fixed and random conditions was far more pronounced for NEAT (fixed test reward 414.7 versus random 1581.4) than for DQN (fixed 254.0 versus random 912.4). The relatively larger fixed-versus-random gap for NEAT suggests that when NEAT does overfit to a fixed environment, it does so more thoroughly than DQN, possibly because an evolved topology that is optimised for a specific maze geometry becomes structurally committed to that geometry in a way that is harder to transfer than a weight-based policy.

5.3. Generalisation Performance

The primary finding of this study is that NEAT demonstrated significantly greater zero-shot generalisation capability than DQN when both were trained under randomised conditions and evaluated on held-out unseen mazes.

On the reached-stage test, NEAT-random achieved a mean test reward of 1581.4 compared to DQN-random’s 912.4, a difference confirmed as statistically significant by an independent samples t-test, $t(198) = -3.564$, $p < 0.001$. The median reward gap was larger still: NEAT-random’s median of 1162.4 versus DQN-random’s 279.1 indicating that NEAT’s advantage is consistent across the majority of test seeds rather than inflated by a small number of high-scoring outliers. NEAT-random also collected significantly more pellets on average (141.7 versus 97.9, $p < 0.001$), covered significantly more of the maze (40.3% versus 27.9% explore rate, $p < 0.001$), and consumed significantly more ghosts on test episodes (0.32 versus 0.17, $p = 0.040$).

The win rate comparison (37% versus 26%) approached but did not reach statistical significance at the conventional threshold, $z = -1.674$, $p = 0.094$. This result warrants careful interpretation. The proportions test treats each episode as a binary win or loss, discarding all information about partial performance. NEAT’s advantage is most clearly expressed in the continuous reward distribution, where it accumulates substantially more reward even in episodes that do not result in a win. Consequently, the t-test on continuous reward is the more sensitive and informative measure of generalisation capability in this context.

The fixed-stage-7 test provides a fully standardised comparison in which all four conditions face identical difficulty. No condition achieved any wins, confirming that full game complexity with all four ghosts at speed 1.85 and 100% pellet requirement exceeds the capability developed within the available training budgets. Nevertheless, the reward ordering was consistent with the reached-stage results: NEAT-random (625.0), NEAT-fixed (424.7), DQN-random (265.6), DQN-fixed (156.8). The statistically significant advantage for NEAT-random over DQN-random at this difficulty level, $t(198) = -3.566$, $p < 0.001$, confirms that NEAT’s generalisation advantage persists even when neither algorithm can win.

Importantly, NEAT-random achieved these results having completed only 32,658 episodes versus DQN’s 100,000. On a per-episode basis, NEAT-random produced approximately 3.1 times more test reward per training episode than DQN-random ($1581.4 / 32,658$ versus $912.4 / 100,000$ normalised). This represents a significant sample efficiency advantage in addition to the raw performance advantage.

5.4. Behavioural Analysis

Examining the failure modes of each algorithm provides diagnostic insight into the qualitative differences between the two learning paradigms.

Ghost collision was the dominant failure mode in all test conditions, accounting for 73% of DQN-random test episodes and 53% of NEAT-random test episodes. The lower ghost death rate for NEAT-random is consistent with its higher explore rate, an agent that covers more of the maze is more likely to successfully route around ghosts rather than running into them directly. Starvation was rare for both algorithms (1% for DQN, 10% for NEAT), confirming that neither algorithm failed primarily due to poor navigation. Both had developed functional pellet-seeking behaviour. The challenge was survival in the presence of the ghosts on novel spatial configurations, not maze traversal itself.

The contrast between DQN’s and NEAT’s failure modes reveals a meaningful structural difference. DQN’s ghost deaths were concentrated in rapid early-episode deaths, reflected in the negative median reward of -75.9 for DQN-fixed and a median of only 279.1 for DQN-random which is close to the -500 death penalty that a single early ghost collision would produce. NEAT’s median rewards of 1162.4 (random) and 366.7 (fixed) indicate that NEAT

episodes typically lasted long enough to collect substantial pellets before dying, suggesting a more cautious and spatially aware navigation style that delays rather than avoids fatal ghost contact.

DQN averaged 0.17 ghost kills per test episode while NEAT averaged 0.32. The higher ghost eating rate for NEAT on unseen mazes suggests that NEAT’s evolved topology retained a more generalised representation of the power pellet opportunity than DQN’s weight-based policy. In DQN, the ghost eating behaviour that emerged during stage 3 training may have been partially tied to specific spatial contexts in the training mazes where power pellet positions were predictable. On unseen mazes where those contexts did not repeat, the behaviour transferred less reliably. NEAT’s population-level search, which evaluated genomes across multiple procedurally generated seeds per generation, may have produced topological structures that are less dependent on specific spatial contexts for triggering ghost-eating behaviour.

The NEAT-fixed stage 7 condition produced the most distinctive behavioural signature in the study. With 1.445 ghost kills and 2.113 power pellets per episode averaged across 36,767 full-game episodes, the agent was consistently engaging with the frightened-mode mechanic even without achieving wins. This level of ghost interaction at full game complexity, with four ghosts at speed 1.85, suggests that the NEAT fixed condition developed a genuinely strategic power pellet response that no other condition matched, even if the surrounding navigation was not sufficient to clear the full pellet requirement.

5.5. Hypothesis Evaluation

The study’s hypothesis, stated in Chapter 1, was that NEAT would demonstrate slower initial learning but achieve superior zero-shot generalisation on novel, unseen maps compared to DQN.

The hypothesis is partially confirmed. NEAT did achieve superior zero-shot generalisation on unseen random mazes, with a statistically significant advantage in mean reward ($p < 0.001$), pellet collection ($p < 0.001$), explore rate ($p < 0.001$), and ghost interaction ($p = 0.040$) on the reached-stage test. This aspect of the hypothesis is strongly supported by the data.

The slower initial learning component requires more nuanced evaluation. On a per-episode basis, NEAT’s population required more episodes than DQN to clear the early ghost-free navigation stages whereas NEAT-random spent 26,667 episodes at stage 2 compared to DQN-random’s 944. This is consistent with the hypothesis. However, once ghost stages were reached, NEAT-fixed progressed more rapidly through the curriculum than DQN under equivalent conditions, reaching stage 7 while DQN-fixed reached only stage 5. The relationship between NEAT and DQN training speed is therefore not a simple linear ordering but depends on whether the task involves ghost interaction, where NEAT’s structural flexibility appears to confer an advantage over DQN’s gradient-based optimisation.

The fixed seed results present a nuanced counterpoint to the generalisation hypothesis. Both algorithms showed a significant memorisation effect, fixed seed training produced higher curriculum progression but lower test performance in both cases. For NEAT the memorisation effect was statistically stronger ($t = -9.037$ versus $t = -3.982$ for DQN), suggesting that when NEAT overfits it does so more completely than DQN. This is an important qualification to the generalisation advantage: NEAT generalises better under random training conditions, but is not inherently immune to memorisation when the training distribution is fixed.

Against the six objectives stated in Chapter 1, the study can report the following outcomes. Objectives 1 through 4 (environment implementation, Gymnasium interface, DQN implementation, NEAT implementation) were fully achieved. Objective 5 (evaluating on held-out seeds) was achieved with 100 test episodes per condition across eight test groups. Objective 6 (comparative analysis with respect to generalisation performance, sample efficiency, and behavioural stability) was achieved, with statistical validation confirming NEAT’s generalisation advantage across multiple metrics.

5.6. Limitations

Several limitations of the present study are acknowledged.

The NEAT-random pipeline completed only 32,658 of its intended episode budget due to a shared elapsed time ceiling across pipelines. Post-hoc inspection suggests DQN-random’s longer per-episode duration consumed a disproportionate share of the available compute time. While this does not invalidate the results, NEAT-random still outperformed all other conditions

on test so the comparison between DQN-random and NEAT-random is not fully controlled for training episode count.

The reached-stage test evaluates each condition at a different curriculum difficulty, complicating direct cross-algorithm comparison. DQN-random was tested at stage 4 while NEAT-random was tested at stage 5. The fixed-stage-7 test addresses this by standardising difficulty, and NEAT’s advantage persists there, but the reached-stage comparison is not a pure measure of generalisation capability independent of curriculum stage reached.

The ANOVA for seed variability could not be computed because each held-out seed was evaluated in a single episode per condition. Multiple episodes per seed would be required to assess within-seed variance and determine whether some maze geometries are systematically harder than others.

Several implementation issues were identified and corrected during development, most significantly an incorrect egocentric ray heading mapping that corrupted directional threat perception in earlier training runs. While all final results were produced using the corrected implementation, the debugging process consumed significant training time before the issue was identified.

Neither algorithm achieved wins at full game difficulty within the available training budget. The study therefore provides evidence of relative generalisation capability between the two paradigms rather than evidence of either achieving mastery of the complete task.

5.7. Summary

This chapter has evaluated the experimental results against the research objectives, confirmed the statistical significance of NEAT’s generalisation advantage, demonstrated the memorisation effect empirically for both algorithms, and assessed the nature versus nurture hypothesis against the evidence. Chapter 6 summarises the completed work, restates the key findings, and identifies directions for future research.

6. Conclusion

This chapter brings together the findings of the study. It summarises the work completed, restates the key results, acknowledges the limitations that constrain the conclusions, and identifies concrete directions for future work that would extend or strengthen the findings.

6.1. Summary of Completed Work

This project set out to answer a fundamental question in artificial intelligence research: does a nature-centric evolutionary approach or a nurture-centric gradient-based approach produce greater zero-shot generalisation capability in a dynamic, partially observable environment? To answer this, a bespoke Pac-Man game engine was developed from scratch in Python, wrapped in a Gymnasium-compliant reinforcement learning interface, and used as a controlled testbed for comparing Deep Q-Networks and NeuroEvolution of Augmenting Topologies under identical, procedurally varied conditions.

The environment exposes a 29-dimensional vector observation covering egocentric ray-casts across four cardinal directions, topology-aware BFS global features, and power state context. Both algorithms interacted with this identical interface, ensuring that any observed performance difference was attributable to the learning paradigms themselves rather than to architectural asymmetry or environmental access. A structured eight-stage curriculum progressively introduced ghost complexity, culminating in a full game configuration with all four ghosts active at maximum speed and a 100% pellet completion requirement. A ghost eating bridge stage with an explicit consumption bonus was introduced at stage 3 in response to early training failures where agents never engaged with frightened-mode mechanics, demonstrating the iterative refinement that characterised the development process.

Four training pipelines were executed across two seed conditions. The primary finding is that NEAT demonstrated statistically significant superiority in zero-shot generalisation under randomised training conditions. NEAT-random achieved a mean test reward of 1581.4 compared to DQN-random’s 912.4 on 100 held-out unseen maze seeds, $t(198) = -3.564$, $p < 0.001$, with significant advantages also confirmed across pellet collection, spatial coverage, and ghost interaction metrics. This result was achieved despite NEAT-random completing only 32,658 training episodes versus DQN’s 100,000, representing a substantial sample efficiency advantage. The memorisation effect was empirically confirmed for both algorithms since fixed

seed training enabled greater curriculum progression but produced significantly lower test performance, directly validating the generalisation argument established in the literature review.

6.2. Limitations

Several limitations constrain the conclusions that can be drawn from this study.

The NEAT-random pipeline completed only 32,658 of its intended episode budget due to a shared elapsed time ceiling across pipelines, meaning the DQN and NEAT random conditions were not fully matched for training episode count. A complete NEAT-random run would provide stronger grounds for comparison.

The reached-stage test evaluates each condition at a different curriculum difficulty, which complicates direct cross-algorithm comparison. Although the fixed-stage-7 test addresses this by standardising difficulty, the reached-stage results cannot be treated as a pure measure of generalisation capability independent of the stage reached.

The evaluation design used a single episode per held-out seed, preventing within-seed variance analysis through ANOVA. Multiple episodes per seed would provide a more statistically complete picture of generalisation robustness.

Several implementation issues were identified and corrected during iterative development, most significantly an incorrect egocentric ray heading mapping that corrupted directional threat perception in earlier training runs. While all final results were produced using the corrected implementation, the debugging process consumed a substantial portion of the available development and training time.

Finally, neither algorithm achieved wins at full game difficulty within the available training budget. The study therefore provides evidence of relative generalisation capability between the two paradigms rather than evidence of either achieving mastery of the complete task.

6.3. Future Work

Several concrete directions would extend and strengthen the findings of this study.

6.3.1. Parallelised DQN Environment Execution

DQN training was conducted sequentially with eight parallel environments feeding a single training loop, with the game engine running on CPU between GPU updates. Deploying a larger number of parallel environment instances using `gymnasium.vector.AsyncVectorEnv` would multiply replay buffer diversity and dramatically improve sample efficiency. On the hardware used in this study, 8 to 12 parallel environments are feasible without thread contention, and preliminary analysis suggests this change alone could provide the equivalent of several times the training experience within the same wall-clock budget. This is the single most impactful infrastructure change available for a follow-up DQN run.

6.3.2. Recurrent DQN Architecture

The current DQN implementation uses a memoryless feedforward network, meaning each decision is made solely from the current 29-dimensional observation without any temporal context. Pac-Man is a partially observable environment where ghost positions and trajectories cannot be fully inferred from a single observation. Introducing a recurrent architecture such as an LSTM layer between the shared trunk and the dueling streams would allow the network to maintain a belief state across timesteps, potentially improving ghost avoidance by enabling the agent to track ghost movement patterns over time rather than reacting only to instantaneous proximity signals.

6.3.3. Extended NEAT Training Budget

NEAT-fixed reached stage 7 which was the full game within 100,000 episodes and showed consistent ghost eating behaviour at that difficulty level despite not achieving wins. A dedicated NEAT-fixed run with an extended budget of 200,000 to 300,000 episodes would determine whether continued evolution at stage 7 eventually produces winning behaviour, and would provide a cleaner picture of the ceiling performance achievable through neuroevolution on this task.

6.3.4. Multiple Episodes Per Test Seed

Evaluating each held-out seed across three to five episodes rather than one would enable within-seed variance analysis through ANOVA, providing a more statistically rigorous assessment of generalisation consistency. It would also distinguish between seeds where failure was caused by specific maze geometries that are genuinely harder and seeds where failure was caused by stochastic ghost behaviour on an otherwise solvable layout.

6.3.5. Curriculum Demotion

Demotion was removed from the curriculum manager in this study after previous experiments showed it caused oscillation and policy degradation. A more sophisticated demotion mechanism that only triggers after sustained failure and partially preserves the replay buffer state rather than clearing it entirely may allow the agent to recover from difficult stages without losing accumulated experience. This could be particularly beneficial at the ghost introduction stages where the performance cliff between stage 2 and stage 3 caused significant training instability across multiple runs.

6.3.6. Comparative Study with Additional Algorithms

The nature versus nurture framing of this study is binary by design, but the generalisation landscape between gradient-based and evolutionary approaches is richer than a single comparison can capture. Extending the testbed to include Proximal Policy Optimisation, a more modern gradient-based algorithm less prone to catastrophic forgetting than DQN, or a hybrid neuroevolution approach such as evolutionary strategies, would provide a more complete picture of where each paradigm sits on the generalisation spectrum.

6.4. Summary

This project set out to determine whether a nature-centric evolutionary approach or a nurture-centric gradient-based approach produces greater zero-shot generalisation capability in a procedurally generated Pac-Man environment. The evidence consistently favoured NEAT under randomised training conditions across multiple metrics, with the advantage confirmed as statistically significant. The study contributes an empirical foundation for continued

investigation into the generalisation capabilities of gradient-based and evolutionary learning paradigms in dynamic, partially observable environments.

References

- Bellemare, M.G., Naddaf, Y., Veness, J. and Bowling, M. (2013) ‘The Arcade Learning Environment: An evaluation platform for general agents’, *Journal of Artificial Intelligence Research*, 47, pp. 253–279.
- Bengio, Y., Louradour, J., Collobert, R. and Weston, J. (2009) ‘Curriculum learning’, in *Proceedings of the 26th International Conference on Machine Learning*. New York: ACM, pp. 41–48.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016) *OpenAI Gym* [Preprint]. Available at: <https://arxiv.org/abs/1606.01540> (Accessed: 14 February 2026).
- Cobbe, K., Klimov, O., Hesse, C., Kim, T. and Schulman, J. (2019) ‘Quantifying generalization in reinforcement learning’, in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 1282–1307.
- Gallagher, M. and Ryan, A. (2003) ‘Learning to play Pac-Man: An evolutionary, rule-based approach’, in *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*. Canberra: IEEE, pp. 2462–2469.
- Irpan, A. (2018) *Deep reinforcement learning doesn’t work yet* [Blog]. Available at: <https://www.alexirpan.com/2018/02/14/rl-hard.html> (Accessed: 7 January 2026).
- Kaelbling, L.P., Littman, M.L. and Cassandra, A.R. (1998) ‘Planning and acting in partially observable stochastic domains’, *Artificial Intelligence*, 101(1–2), pp. 99–134.
- Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D. and Hadsell, R. (2017) ‘Overcoming catastrophic forgetting in neural networks’, *Proceedings of the National Academy of Sciences*, 114(13), pp. 3521–3526.
- Machado, M.C., Bellemare, M.G., Talvitie, E., Veness, J., Hausknecht, M. and Bowling, M. (2018) ‘Revisiting the Arcade Learning Environment: Evaluation protocols and open problems for general agents’, *Journal of Artificial Intelligence Research*, 61, pp. 523–562.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015) ‘Human-level control through deep reinforcement learning’, *Nature*, 518(7540), pp. 529–533.

Raschka, S., Liu, Y. and Mirjalili, V. (2022) *Machine Learning with PyTorch and Scikit-Learn*. Birmingham: Packt Publishing.

Rohlfshagen, P. and Lucas, S.M. (2011) ‘Ms Pac-Man versus ghost team CEC 2011 competition’, in *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*. New Orleans: IEEE, pp. 70–77.

Şafak, A.B., Bostanci, E. and Soyluçiçek, A.E. (2016) ‘Automated maze generation for Ms. Pac-Man using genetic algorithms’, *International Journal of Machine Learning and Computing*, 6(4), pp. 226–230.

Salimans, T., Ho, J., Chen, X., Sidor, S. and Sutskever, I. (2017) *Evolution strategies as a scalable alternative to reinforcement learning* [Preprint]. Available at: <https://arxiv.org/abs/1703.03864> (Accessed: 3 February 2026).

Schaul, T., Quan, J., Antonoglou, I. and Silver, D. (2016) ‘Prioritized experience replay’, in *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*. Available at: <https://arxiv.org/abs/1511.05952> (Accessed: 7 March 2026).

Schrum, J. and Miikkulainen, R. (2016) ‘Discovering multimodal behavior in Ms. Pac-Man through evolution of modular neural networks’, *IEEE Transactions on Computational Intelligence and AI in Games*, 8(1), pp. 67–81.

Shinners, P. (2011) *Pygame* [Software]. Available at: <https://www.pygame.org> (Accessed: 20 December 2025).

Stanley, K.O. and Miikkulainen, R. (2002) ‘Evolving neural networks through augmenting topologies’, *Evolutionary Computation*, 10(2), pp. 99–127.

Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O. and Clune, J. (2017) *Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning* [Preprint]. Available at: <https://arxiv.org/abs/1712.06567> (Accessed: 3 January 2026).

Terven, J. (2025) ‘Deep reinforcement learning: A chronological overview and methods’, *AI*, 6(3), Article 46. Available at: <https://doi.org/10.3390/ai6030046> (Accessed: 15 February 2026).

Towers, M. et al. (2023) *Gymnasium* [Software]. Available at: <https://github.com/Farama-Foundation/Gymnasium> (Accessed: 14 January 2026).

van Hasselt, H., Guez, A. and Silver, D. (2016) ‘Deep reinforcement learning with double Q-learning’, in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. Menlo Park: AAAI Press, pp. 2094–2100.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. and de Freitas, N. (2016) ‘Dueling network architectures for deep reinforcement learning’, in *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*. PMLR, pp. 1995–2003.

Wittkamp, M., Barone, L. and Hingston, P. (2008) ‘Using NEAT for continuous adaptation and teamwork formation in Pacman’, in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games*. Perth: IEEE, pp. 234–242.

Zahavy, T., Ben-Zrihem, N. and Mannor, S. (2016) ‘Graying the black box: Understanding DQNs’, in *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*. PMLR, vol. 48, pp. 1899–1908.

Bibliography

PyTorch (2024) *PyTorch documentation* [Online]. Available at: <https://pytorch.org/docs> (Accessed: 14 January 2026).

Farama Foundation (2023) *Gymnasium documentation* [Online]. Available at: <https://gymnasium.farama.org> (Accessed: 14 December 2025).

McIntyre, A., Kallada, M., Miguel, C.G. and Feher da Silva, C. (2019) *neat-python* [Software]. Available at: <https://neat-python.readthedocs.io> (Accessed: 20 January 2026).

Appendices

Appendix A: Feasibility Study



Feasibility Study

6G6Z0019: Research Methods

Name: Faheem Saleem

Student ID: 22459265

Degree Programme: BSc Computer Science with Industrial Year

Project Title: Nature vs Nurture: Comparing DQN and NEAT for Generalisable AI in Pac-Man



Key Terminology & Abbreviations



- **AI** -> Artificial Intelligence
- **RL** -> Reinforcement Learning
- **DQN** -> Deep Q-Network
- **DRL** -> Deep Reinforcement Learning
- **NEAT** -> Neuroevolution of Augmenting Topologies
- **GA** -> Genetic Algorithms
- **Generalisation**: The ability of an AI agent to perform well in environments it has never seen before (i.e. not just memorising the training data).



Project Outline



Context:

Training AI to master complex, dynamic environments like **Pac-Man** is a fundamental challenge in Computer Science, moving beyond static game benchmarks.

The Problem:

While Deep Reinforcement Learning (DQN) has achieved superhuman performance in static Atari games (Mnih et al, 2015), its ability to generalise is often brittle. Evolutionary computation presents an alternative paradigm.

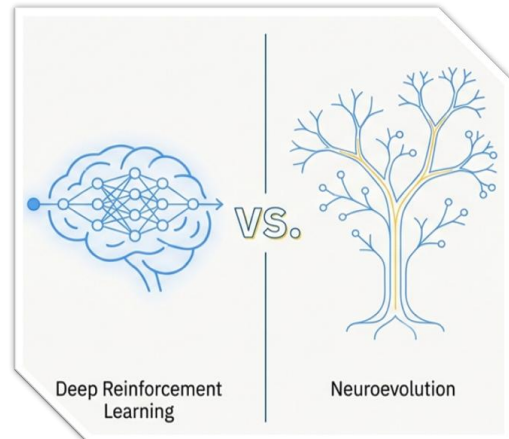
The Research Gap:

There is a lack of rigorous, direct comparative studies evaluating "Gradient-based learning" (DQN) vs "Evolutionary Learning" (GA) on their ability to adapt to unseen map variations (generalisation) rather than just memorising a single level.

The Proposed Solution:

This study proposes a rigorous empirical comparison between Gradient-Based Learning (**DQN**) and Evolutionary Computation (**NEAT**).

By deploying both agent types into procedurally generated Pac-Man maps, this project aims to quantify which paradigm offers superior robustness and generalisation.



200
YEARS
— 1824–2024 —

Project Fit



Academic Alignment: This project integrates four core pillars of the Computer Science curriculum.



Machine Learning
Implementation of
neural networks and
backpropagation



Algorithm Design
Application of Genetic
Algorithms (GA) to evolve
network topologies



Software Engineering
Development of a
modular, extensible
testing framework



Data Science
Application of rigorous
statistical analysis (t-
tests, ANOVA)

Skills Extension: While the AI module in Semester 1 introduced Supervised Learning and Training Models, this project extends into **Neuroevolution and Transfer Learning**, areas which are currently at the forefront of AI research

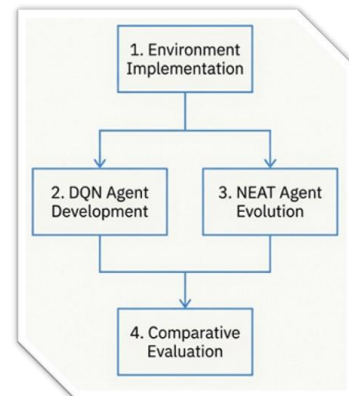
200
YEARS
— 1824–2024 —

Aims & Objectives

Aim: To scientifically evaluate and compare the generalisation capabilities of DQN and NEAT within a dynamic Pac-Man environment

Objectives:

- 1. Develop:** A Python-based simulation environment (using Gymnasium) capable of procedural map generation to test adaptability
- 2. Implement:** A DQN agent utilising PyTorch to represent the gradient-based learning paradigm
- 3. Implement:** A NEAT agent utilising Python-NEAT to represent the evolutionary learning paradigm
- 4. Experiment:** Train both agents to mastery on a static Control Map to establish a performative baseline
- 5. Evaluate:** Assess robustness by deploying frozen models onto Unseen Maps and measuring performance degradation.
- 6. Analyse:** Conduct statistical validation to determine if one method holds a statistically significant advantage in generalisation.



Research Context

Foundational Literature (The Origins)

Deep Q-Networks

Mnih et al. (2015, *Nature*) established the foundational architecture for DQN, proving it could solve complex Atari games from raw pixels. This provides the baseline architecture for our gradient-based agent.

Neuroevolution

Stanley & Mikkulainen (2002) introduced NEAT, demonstrating that evolving network topologies can outperform fixed-topology learning in complex control tasks. This provides the methodology for our evolutionary agent

Contemporary Literature (The Modern Debate):

Salimans et al. (*OpenAI*, 2017):

Demonstrated that **Evolution Strategies** can rival Deep RL in complex robotics tasks.

Such et al. (*Uber AI*, 2018):

Provided empirical evidence that **Genetic Algorithms** can outperform DQN in sparse-reward environments.

Cobbe et al. (2019):

Highlighted the "Generalisation Gap," proving that standard RL agents fail when game levels are procedurally generated.

The Research Gap:

Existing literature often prioritises *training speed or peak score*. There is a lack of direct comparative studies focusing on **zero-shot generalisation**, specifically how an evolved brain compares to a trained brain when facing novel scenarios.

Search Strategy:

Literature is sourced via IEEE Xplore, ACM Digital Library, and arXiv, filtering for publications on "Evolutionary Reinforcement Learning" and "Generalisation in RL."

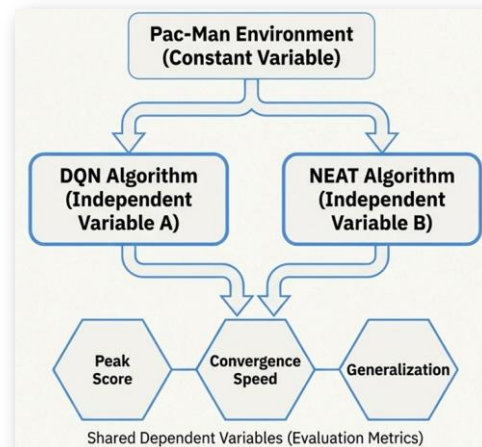


Research Methodology

Method: Quantitative Comparative Experimental Study

Justification: A controlled experimental approach is required to isolate the independent variable (the learning algorithm). By keeping all other variables constant (game rules, input resolution, hardware resources), any observed difference in performance can be scientifically attributed to the methodology itself.

Hypothesis: Evolutionary methods (NEAT) will demonstrate slower initial learning but achieve superior zero-shot generalisation on novel, unseen maps compared to gradient-based methods (DQN)



Implementation Methodology

Approach

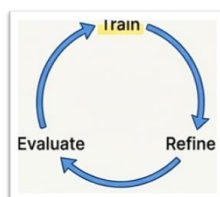
A hybrid model of Iterative (Prototyping) + Incremental (Agile)

Why Iterative?

AI model training is inherently experimental. Optimal hyperparameters (e.g. learning rate, mutation rate) are unknown and must be discovered through cycles of

Train -> Evaluate -> Refine

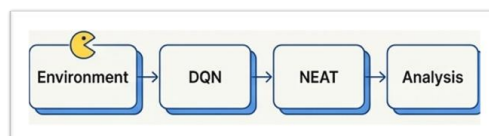
This addresses the challenge of navigating the loss landscape without a perfect map



Why Agile?

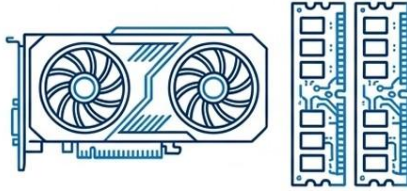
The project is structured into 2-week sprints (Environment Setup -> DQN -> NEAT -> Analysis)

This ensures a deliverable artifact exists at every stage, mitigating the risk of “all or nothing” project failure and allowing for continuous, measurable progress.



Resources

Hardware



Critical Hardware Requirements

A personal high-performance workstation will optimise this hybrid workload: the **NVIDIA RTX 4060** utilises **CUDA and Tensor Cores** to accelerate DQN matrix operations, reducing training time from weeks to days.

Conversely, **multi-core CPU parallelisation** will be leveraged to execute thousands of NEAT evolutionary episodes simultaneously.

While **32GB of RAM** ensures the 1-million-frame Experience Replay Buffer remains entirely in memory.

Software

Software & Libraries

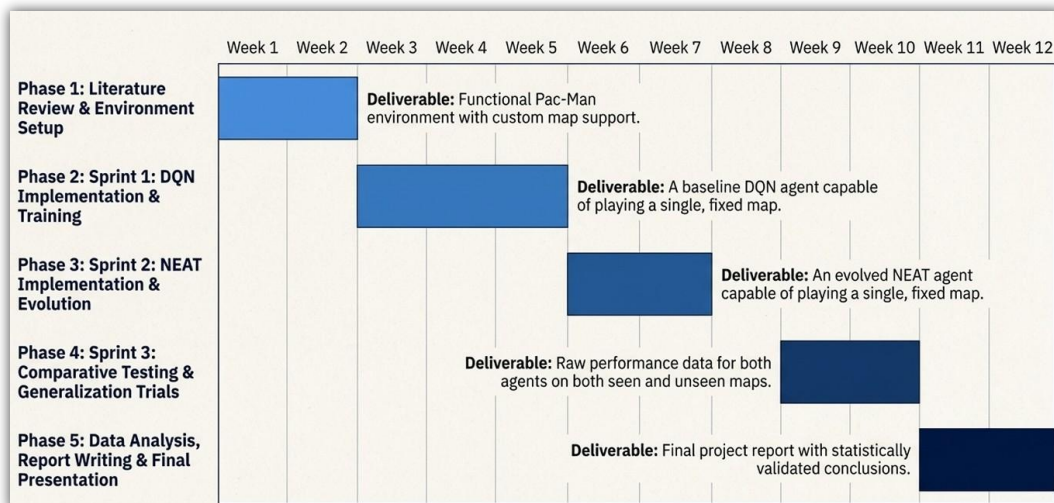


Language: Python 3.x (used as the industry & research standard for AI development)

Libraries:

- **PyTorch:** For building and training deep neural networks
- **Gymnasium:** For creating and managing the Pac-Man simulation environment
- **Numpy / Pandas:** For efficient data manipulation and statistical analysis of results.

Project Plan



Potential Risks and Solutions



Risk 1: Failure to Converge (Likelihood: High, Impact: Critical)

Problem

Agents fail to learn meaningful policies, becoming stuck in local minima on the loss landscape

Mitigation

1. Implement Curriculum Learning, starting with smaller, simpler mazes and gradually increasing difficulty
2. Seed hyperparameters with proven values from foundational papers (e.g. Mnih et al, 2015)



Risk 2: Reward Hacking (Likelihood: Medium, Impact: Medium)

Problem

An agent maximises its score via an unintended loophole (e.g. repeatedly eating a single frightened ghost) without achieving the true goal of clearing the level

Mitigation

Design a sparse but dense reward function. Assign a large reward for level completion but a small negative reward per time step to encourage efficiency and goal orientated behaviour.



Risk 3: Prohibitive Training Time (Likelihood: Medium, Impact: High)

Problem

Model training cycles take too long to complete, preventing effective iteration and hyperparameter tuning.

Mitigation

1. Implement Checkpointing to regularly save and resume training states
2. Fully utilise GPU acceleration via CUDA for all neural network operations

Project Impact Considerations

AI Safety & Alignment

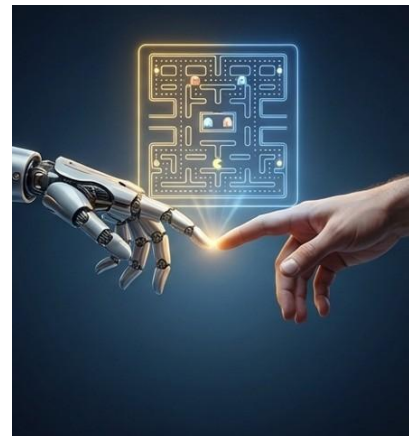
This project directly exposes the risk of Reward Hacking, a micro-version of the AI alignment problem. Understanding how different learning paradigms exploit reward functions provides insight into building safer, more aligned systems in finance or healthcare.

Automation & Ethics

Which learning method is more predictable and therefore “safer”, a deterministic gradient-based system or a creative but less predictable evolutionary one? This question is crucial for deploying AI in safety critical domains like autonomous vehicles or robotics.

Generalisation & Robustness

The core of this project, *testing zero shot generalisation* is one of the most significant unsolved problems in AI. This research contributes to the fundamental challenge of building agents that can adapt to novel situations rather than simply memorise solutions.



Project Evaluation Plan

Key Performance Metrics



Peak Performance

Comparison of Mean and Median scores achieved by each agent over 100 standardised test episodes.



Training Efficiency

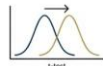
Measurement of wall-clock time and the number of training episodes required for each agent to reach a predetermined performance baseline (e.g. clearing 50% of the dots).



Robustness (Generalisation)

The percentage drop in performance when the trained agents are deployed on procedurally generated maps they have never encountered during training.

Analytical Methods



Statistical significance of the results will be rigorously verified. Independent t-tests will be used for direct comparison of mean scores between the two agents. ANOVA will be used for analysing the impact of different hyperparameters settings to ensure that the final conclusions are not due to random chance.

In Conclusion

Summary: This project executes a rigorous, scientific investigation into two competing paradigms of machine intelligence, using Pac-Man as a complex, dynamic testbed.



Nurture (Deep Reinforcement Learning)

A single agent learning through trial, error and gradual refinement based on its individual experience (gradient descent)

Contribution

The findings will provide valuable, data-driven insights on which paradigm produces more robust and adaptable AI agents, contributing to the development of more reliable AI for complex systems.

The core question: The comparison is framed as 'Nature vs Nurture' for AI



Nature (Neuroevolution)

A population of agents where successful traits are selected and passed on, allowing solutions to emerge through a process of 'survival of the fittest'

1. Cobbe, K., Klimov, O., Hesse, C., Kim, T. and Schulman, J. (2019) 'Quantifying generalization in reinforcement learning', in Proceedings of the 36th International Conference on Machine Learning, pp. 1282–1307.
2. Mnih, V. et al. (2015) 'Human-level control through deep reinforcement learning', Nature, 518(7540), pp. 529–533.
3. Salimans, T., Ho, J., Chen, X., Sidor, S. and Sutskever, I. (2017) 'Evolution strategies as a scalable alternative to reinforcement learning', arXiv preprint arXiv:1703.03864. Available at: <https://arxiv.org/abs/1703.03864> (Accessed: 24 December 2025).
4. Stanley, K.O. and Miikkulainen, R. (2002) 'Evolving neural networks through augmenting topologies', Evolutionary Computation, 10(2), pp. 99–127.
5. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O. and Clune, J. (2017) 'Deepneuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning', arXiv preprint arXiv:1712.06567. Available at: <https://arxiv.org/abs/1712.06567> (Accessed: 24 December 2025).

Appendix B: Default Game Settings Configuration

The following JSON file defines the default game settings used by the PacManEnv. Curriculum stage profiles override relevant parameters at runtime via the CurriculumManager.

```
{
  "window_resolution": "1000x1000",
  "tile_size": 40,
  "enable_sound": false,
  "pacman_speed": 3,
  "lives": 1,
  "enable_ghosts": true,
  "ghost_speed": 2,
  "ghost_speed_increment": 0.2,
  "blinky_active": true,
  "pinky_active": true,
  "inky_active": true,
```

```

    "clyde_active": true,

    "chase_duration": 20,

    "scatter_duration": 7,

    "always_chase": false,

    "enable_power_pellets": true,

    "pellets_to_win": -1,

    "level": 1,

    "maze_algorithm": "recursive_backtracking",

    "maze_seed": "",

    "god_mode": false,

    "tile_center_tolerance": 3

}

```

Appendix C: NEAT Configuration File

The following configuration file defines all hyperparameters used by the NEAT algorithm during training. This file is loaded directly by the python-neat library at the start of each training pipeline.

```

fitness_criterion                                =      max
fitness_threshold                                =      1250
no_fitness_termination                          =              True
pop_size                                          =          20
reset_on_extinction                             =              True

[DefaultGenome]
#                               Node                               activation
activation_default                            =      relu
activation_mutate_rate                         =              0.05
activation_options                             =      relu      tanh

#                               Node                               aggregation
aggregation_default                            =      sum
aggregation_mutate_rate                        =              0.0
aggregation_options                             =      sum

```

#			Bias
bias_init_mean		=	0.0
bias_init_stdev		=	1.0
bias_max_value		=	30.0
bias_min_value		=	-30.0
bias_mutate_power		=	0.5
bias_mutate_rate		=	0.7
bias_replace_rate		=	0.1
#	Compatibility		(speciation)
compatibility_disjoint_coefficient		=	2.0
compatibility_weight_coefficient		=	0.5
#	Connection		add/remove
conn_add_prob		=	0.5
conn_delete_prob		=	0.3
#	Connection		enable
enabled_default		=	True
enabled_mutate_rate		=	0.01
#	Feed-forward		only
feed_forward		=	True
#	Initial		connectivity
initial_connection		=	partial_direct 0.5
#	Node		add/remove
node_add_prob		=	0.2
node_delete_prob		=	0.1
#	Network		size
num_hidden		=	0
num_inputs		=	29
num_outputs		=	4
#			Response
response_init_mean		=	1.0
response_init_stdev		=	0.0

response_max_value	=	30.0
response_min_value	=	-30.0
response_mutate_power	=	0.0
response_mutate_rate	=	0.0
response_replace_rate	=	0.0
#		Weights
weight_init_mean	=	0.0
weight_init_stdev	=	1.0
weight_max_value	=	30.0
weight_min_value	=	-30.0
weight_mutate_power	=	0.5
weight_mutate_rate	=	0.8
weight_replace_rate	=	0.1
[DefaultSpeciesSet]		
compatibility_threshold	=	3.0
[DefaultStagnation]		
species_fitness_func	=	max
max_stagnation	=	30
species_elitism	=	1
[DefaultReproduction]		
elitism	=	1
survival_threshold	=	0.30
min_species_size	=	1

Appendix D: Ethos Screenshots

Z3 Please click here to request your supervisor's signature. You will be asked to input their email address.

Your application will not be submitted until your supervisor has signed - it is your responsibility to ensure that they do this.

Signed: This form was signed by Dr Kate Macfarlane (K.Macfarlane@mmu.ac.uk) on 16/02/2026 12:26

Z6 By signing this application you are confirming that all details included in the form have been completed accurately and truthfully.

Signed: This form was signed by Faheem Saleem (FAHEEM.SALEEM@stu.mmu.ac.uk) on 06/02/2026 10:45

UGPGT6 Please describe your research

Please make sure to discuss your aim, the activities you undertake, any special software/equipment you will use and the outcomes you expect.

My aim for this project will be to scientifically evaluate and compare the generalisation capabilities of DQN and NEAT within a dynamic Pac-Man environment.

Generalisation is a term where the ability of an AI agent is measured based on how well it can perform in an environment it's never seen before. I will use the game Pac-Man to compare the 2 AI methodologies.

I will do this by creating a Python based simulation environment that will be capable of procedural map generation which will be useful for adaptability. Then will create a Deep-Q Network Agent (DQN) & a Neuroevolution of Augmenting Topologies (NEAT) to then train both agents to mastery in training Pac-Man maps.

During initial training I will use curriculum learning where the generated maps will be smaller & simpler mazes and gradually adding ghosts over time, so that the AI agents are not overwhelmed with completing the entire game objective and can focus on learning one thing at a time (collecting dots, eating ghosts when a power pellet is eaten etc)

To train the agents I will be using my personal PC which is equipped with high end equipment that will be capable of training the agents and bring down the training duration from few weeks to a few days. They will play thousands of games over time and will implement Checkpointing so they can be paused and resumed when needed.#

The final evaluation comes down to deploying both agents onto a final complete PacMan map that it has not yet seen before during training and they will be evaluated on gameplay performance ie how many levels it can complete, what high score is achieved & if they've discovered common game mechanics like the power pellets and understanding the different ghost's algorithm.

The expected outcome will be that DQN will perform better on a game like Pac-Man as it can react better in a dynamic environment, but I will conduct this research to see if NEAT does have the chance to perform as good or better than DQN.





Comparing DQN and NEAT for Generalisable AI

[← Go Back to Work Area](#)

Project

Project Tree

- Comparing DQN and NEAT for Generalisable AI
 - EthOS Application Version 2

Create Sub Form

Project

Share

View as PDF

Navigation

Documents

Signatures

Collaborators

Submissions

History

EthOS Application Version 2

☐ Show Inactive Sections

Section

General Information

UG/PGT Application

Declarations and Signatures

Questions

[START HERE] A-B. Applicant Details

C. Application Information

D. Project Information

UG/PGT Regulatory Checks

UG/PGT Application Information

Y-Z. Declarations and Signatures

Overview

BASIC INFO

Project Title	Comparing DQN and NEAT for Generalisable AI
Project Id	87684
Form Title	EthOS Application Version 2
Status	Countersigned
Review Reference	2026-87684-66365
Last Modified	08/02/2026
Action Required	No

© Infonetica Ltd 2026 Version 2.18.2

[Terms and Conditions](#) |
 [Data Controller Privacy Policy](#) |
 [Data Processor Privacy Policy](#)