EvoDraughts

Evolving Game Strategies using Grammatical Evolution

Cormac Greaney

22352228

Department of Computer Science and Information Systems

Faculty of Science and Engineering

University of Limerick

Submitted to the University of Limerick for the degree of

BSc. in Computer Systems academic year 2025/26

1. Supervisor: Dr. Conor Ryan

University of Limerick
Ireland

2. Supervisor: Dr. Tabea De Wille

University of Limerick
Ireland

Abstract

Designing effective strategies for complex board games traditionally relies on domain expertise and manually engineered evaluation functions. This project, EvoDraughts, investigates an alternative approach: evolving draughts strategies using Grammatical Evolution (GE). Candidate strategies are represented as grammar-constrained symbolic expressions and evaluated through repeated gameplay rather than manual design.

The system integrates a custom draughts engine supporting both 6×6 and 8×8 boards, a grammar-based representation ensuring syntactic validity and interpretability, evolutionary search implemented using the GRAPE framework, and a web application for human–agent gameplay. Finally, a separate minimax-based agent was used for a short classical comparison where a set of representative 8×8 evolved strategies were played against it.

Across many experimental runs, evolved strategies consistently outperform a random baseline, with standardised test win rates on the 8×8 board reaching up to 94.7% under a fixed multi-seed evaluation protocol. However, this performance is specific to the evaluation setting and reflects the ability to exploit weak baseline opponents rather than demonstrating strong strategic play. Several strategies achieve win rates above 80%, demonstrating that the evolutionary framework is capable of producing effective decision-making behaviour within the defined search space. However, performance varies across configurations, and some strategies exhibit evidence of overfitting, where behaviour is highly effective against training opponents but fails to generalise to different gameplay conditions.

Human evaluation shows that while evolved strategies are capable of winning games, they remain consistently beatable by human players and are generally perceived as moderately challenging rather than difficult. This highlights a gap between strong performance against simple baseline opponents and the ability to produce robust, high-level gameplay.

Overall, the results demonstrate that Grammatical Evolution can generate functional and interpretable draughts strategies, while also revealing key characteristics of the approach, including a tendency toward simple solutions, sensitivity to evaluation design, and limitations in producing robust, high-level gameplay in adversarial domains. The project provides a complete experimental framework for evolving, evaluating, and analysing game-playing strategies.

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Irish or foreign examination board.

The thesis work was produced under the supervision of Dr. Conor Ryan and Dr. Tabea De Wille at University of Limerick.

Limerick, 2026

AI Declaration

I herewith declare that I have used artificial intelligence to produce my project and report in the following ways:

- To assist with development, debugging and code commenting using integrated IDE IntelliSense tools.
- To assist with report drafting using AI powered grammar checking and formatting tools.

I further declare that I have discussed this use of artificial intelligence with my supervisor and received permission to use it.

Limerick, 2026

Ethics Declaration

I herewith declare that my project involves human participants and that I have received approval from the Science and Engineering Ethics Committee prior to undertaking this part of the project. The application number for this project is: 2025_11_28_S&E

Limerick, 2026

Acknowledgments

I would like to thank my supervisor, Dr. Conor Ryan, for his guidance, feedback, and support throughout this project. His expertise in Grammatical Evolution and evolutionary computation was invaluable in shaping the direction of EvoDraughts.

I am also grateful to the staff of the Department of Computer Science and Information Systems at the University of Limerick for providing the foundational knowledge and skills that made this project possible.

Finally, I would like to thank the friends and classmates who provided feedback on the system and volunteered to participate in the playtesting as part of the human evaluation.

Table of Contents

1 Introduction

Games have long been used as controlled environments for research in artificial intelligence because they combine clear rules with non-trivial decision-making. Draughts is a particularly appropriate domain for this type of work due to its history in machine learning and game-playing research, while remaining sufficiently simple to support detailed implementation and analysis.

The central idea behind EvoDraughts is to evolve draughts strategies automatically rather than hand-crafting them. Instead of manually designing a fixed evaluation function, the project uses Grammatical Evolution to generate strategy expressions from a predefined grammar. This enables exploration of a large space of possible decision rules while maintaining interpretability of the resulting strategies.

The primary focus of the project is the standard 8×8 game. A 6×6 version was introduced as a practical development step, allowing the engine, feature representation, grammars, and evaluation process to be tested efficiently before scaling to the full game.

1.1 Aim & Objectives

The overall aim of this project is to design, implement, and evaluate a system capable of evolving draughts strategies using Grammatical Evolution, and to assess how effective this approach is for producing interpretable and functional gameplay.

To address this aim, the project pursues the following objectives:

- Implement a draughts engine that supports reliable simulation and interactive play.
- Design a grammar-based representation for strategies so that evolved expressions remain syntactically valid and human-readable.
- Integrate the draughts engine, feature extraction, grammars, and GRAPE into an evolutionary pipeline.
- Use the 6×6 board as a lower-cost validation environment during development before moving to the standard 8×8 board.
- Run and organise experiments that evaluate evolved strategies against baseline opponents and under standardised re-evaluation conditions.

- Develop a local web application that allows human players to play against selected evolved strategies.

The project is guided by the following research questions:

- Can Grammatical Evolution produce draughts strategies that outperform simple baseline opponents in the target 8×8 game?
- To what extent are the evolved strategies interpretable when expressed through grammar-derived rules?
- How do human players perceive the strength and coherence of the evolved agents?

1.2 Methodology

The project follows an iterative and experiment-driven methodology. Development began with the construction of the core draughts engine, feature extraction, grammar representation, and evolutionary loop. Early testing was carried out on a 6×6 board to establish correctness and enable faster experimentation.

Once the core system was functioning, the work shifted to the standard 8×8 board, which is the main target environment of the project. Additional tooling was developed for logging runs, re-evaluating stored strategies, and comparing agents under consistent conditions. A shallow minimax opponent was also added for a brief post-hoc comparison with selected evolved strategies.

Alongside the simulation framework, a local Flask web application was developed to support human play against evolved agents. This allows evaluation not only of quantitative performance but also of how coherent and challenging the strategies appear in practice.

1.3 Report Outline

Chapter 2 reviews the background literature relevant to game-playing AI, evolutionary computation, Grammatical Evolution, draughts research, fitness evaluation, and interpretability.

Chapter 3 describes the methodology used in the project, including the draughts environment, strategy representation, evolutionary process, evaluation design, and project management decisions.

Chapter 4 presents the system design and implementation, covering the repository structure, core engine, evolutionary scripts, logging tools, web application, and minimax comparison module.

Chapter 5 outlines the experimental setup and explains how strategies are evaluated, re-evaluated, and used in human testing.

Chapter 6 presents the experimental results, including training performance, standardised evaluation, and human testing outcomes.

Chapter 7 discusses the results, including generalisation, variability across configurations, and limitations of the approach.

Chapter 8 concludes the report and summarises the main findings, along with brief directions for future work.

2 Background and Literature Review

2.1 Artificial Intelligence and Game-Playing

Games have historically played a central role in artificial intelligence research, providing controlled environments in which decision-making algorithms can be developed and evaluated. Board games in particular offer clearly defined rules, deterministic state transitions, and measurable outcomes, making them well suited to studying computational approaches to strategy, planning, and optimisation. As a result, many foundational ideas in artificial intelligence have been explored and validated within the domain of game-playing.

Early AI research in games focused primarily on search-based approaches, where a program explores possible future game states to determine the best move. In this paradigm, gameplay can be represented as a tree in which nodes correspond to board states and edges correspond to legal moves. One of the most influential algorithms for analysing such game trees is the minimax algorithm, which models the interaction between two rational players by assuming that one player attempts to maximise their score while the opponent attempts to minimise it. Because the number of possible game states grows exponentially with search depth, practical implementations often rely on optimisation techniques such as alpha–beta pruning to eliminate branches of the search tree that cannot affect the final decision.

These methods have been applied successfully in many classical board games, including chess, draughts, and Othello. However, search-based systems rely heavily on evaluation functions that estimate the quality of a board position when the search horizon is reached. Designing effective evaluation functions traditionally requires significant domain knowledge and careful manual tuning.

In recent decades, machine learning techniques have been increasingly applied to game-playing problems. Reinforcement learning and deep neural networks have enabled systems such as AlphaGo and AlphaZero to achieve superhuman performance in complex games through large-scale self-play and neural policy/value networks. Monte Carlo tree search remains a widely used building block in modern game-playing systems (Browne et al., 2012). While these approaches demonstrate remarkable performance, they often produce models that

are difficult to interpret and require substantial computational resources. For research contexts where interpretability and analysis of decision logic are important, symbolic and rule-based approaches remain valuable alternatives.

The EvoDraughts project explores one such alternative by applying evolutionary computation techniques to automatically generate strategies for the game of draughts. Rather than relying on manually designed heuristics or opaque neural models, the system evolves symbolic strategies that can be inspected and analysed.

2.2 Evolutionary Computation and Game Strategy

Evolutionary computation (EC) refers to a class of optimisation techniques inspired by the principles of biological evolution. These algorithms operate by maintaining a population of candidate solutions that evolve over successive generations through mechanisms analogous to natural selection, mutation, and recombination. Individuals that perform better according to a defined fitness function are more likely to contribute genetic material to future generations, leading to gradual improvement of the population.

Several forms of evolutionary computation have been developed, including genetic algorithms, genetic programming, evolutionary programming, and Grammatical Evolution. Among these, genetic programming (GP) is particularly relevant to the evolution of strategies because it evolves symbolic structures capable of representing programs or decision rules.

A foundational contribution to this area is Koza's work on Genetic Programming. In Genetic Programming: On the Programming of Computers by Means of Natural Selection, Koza (1992) demonstrated that evolutionary processes could be used to evolve executable programs capable of solving complex problems. In GP, candidate solutions are typically represented as tree-structured programs composed of functions and terminals. Koza showed that evolutionary algorithms could produce non-trivial behaviours in domains such as symbolic regression, automated control systems, and game strategy.

When applied to game-playing environments, evolutionary methods offer several advantages. Strategies can be evaluated directly through gameplay outcomes, allowing the evolutionary process to optimise behaviour without requiring explicitly designed heuristics. This allows strategies to emerge from iterative competition rather than from human intuition.

However, evolutionary game-playing systems also present several challenges. The performance of a strategy is inherently relative to its opponents, meaning that fitness evaluation may depend heavily on the set of strategies used for testing. Additionally, the representation used to encode strategies has a strong influence on the search space. If the representation is poorly designed, the evolutionary process may produce strategies that exploit weaknesses in the evaluation procedure rather than demonstrating robust gameplay.

These issues have motivated research into structured representations that constrain the space of possible strategies while still allowing evolutionary exploration. One such representation is Grammatical Evolution, which allows domain knowledge to be incorporated directly into the representation of candidate programs.

2.3 Grammatical Evolution and Strategy Representation

Grammatical Evolution (GE) was introduced by Ryan, Collins and O'Neill (1998) as a grammar-based evolutionary computation technique that separates the representation of candidate solutions from the evolutionary search process. In GE, individuals are represented as linear genomes consisting of integer values. These genomes are mapped to executable programs through the use of a context-free grammar, typically expressed in Backus–Naur Form (BNF).

This mapping process creates a genotype–phenotype separation, where the genotype corresponds to the integer genome manipulated by the evolutionary algorithm and the phenotype corresponds to the program generated from the grammar. The grammar defines the syntactic structure of valid programs, ensuring that all evolved individuals correspond to syntactically correct expressions.

One of the key advantages of GE is that domain knowledge can be embedded directly into the grammar. By controlling which operators, variables, and constructs are available, researchers can restrict the evolutionary search to meaningful regions of the solution space while maintaining flexibility. This is particularly valuable in complex domains such as game-playing, where unconstrained program evolution may produce invalid or meaningless strategies.

O'Neill and Ryan (2003) later formalised GE as a general-purpose evolutionary technique for automatic programming. Their work demonstrated that grammar-based evolution could be applied to a wide range of problem domains, including classification, regression, and decision-making tasks.

In the context of game strategy development, grammar-based representations provide an additional advantage: interpretability. Strategies generated through GE are typically expressed as symbolic expressions referencing identifiable features of the problem domain. This allows researchers to inspect and analyse the logic behind evolved strategies, making it possible to understand how particular behaviours arise during the evolutionary process.

In EvoDraughts, GE is used to evolve symbolic expressions that operate on features extracted from the game state. The grammar defines the allowable operations and structure of these expressions, enabling the system to explore a range of possible decision rules while ensuring syntactic correctness.

2.4 Evolutionary Approaches to Draughts and Checkers

The game of draughts (checkers) has been an important domain in the development of artificial intelligence techniques. One of the earliest and most influential contributions in this field was Arthur Samuel's checkers program, developed in the late 1950s. Samuel's system combined minimax search with a learned evaluation function that improved through self-play. By adjusting the weights of the evaluation function based on gameplay outcomes, the system was able to gradually improve its performance over time. Samuel's work is widely regarded as one of the first practical demonstrations of machine learning in artificial intelligence (Samuel, 1959).

Later research explored evolutionary approaches to strategy development in checkers. Chellapilla and Fogel (1999) applied evolutionary computation to evolve evaluation functions that guided gameplay decisions. Instead of manually designing heuristics, the system evolved behavioural tendencies that influenced move selection. The results demonstrated that competitive behaviour could emerge through evolutionary processes without explicit domain-specific knowledge.

The same authors also introduced competitive co-evolution, where strategies evolve by competing against other evolving individuals rather than against fixed benchmark opponents (Chellapilla and Fogel, 1999). This approach encourages the development of more robust strategies by continuously adapting to increasingly capable opponents.

Kendall and Whitwell (2001) later examined alternative representations and evaluation strategies for evolutionary chess systems. Their research highlighted the importance of representation design and fitness evaluation methods in determining the effectiveness of evolutionary game-playing algorithms. In particular, they demonstrated that carefully structured representations allow evolutionary search to discover more sophisticated strategies while avoiding premature convergence.

These studies collectively demonstrate that evolutionary methods can successfully produce competitive behaviour in draughts-like games. They also emphasise the importance of representation, evaluation methodology, and experimental design when applying evolutionary computation to game strategy development.

2.5 Fitness Evaluation, Self-Play, and Co-Evolution

Fitness evaluation is one of the most challenging aspects of evolutionary game-playing systems. Unlike traditional optimisation problems, the performance of a strategy in a competitive game depends on the behaviour of its opponents. As a result, strategies cannot be evaluated independently, and the fitness landscape may change dynamically as new strategies emerge.

Many evolutionary game-playing systems therefore rely on self-play or co-evolutionary evaluation. In self-play systems, strategies are evaluated by competing against copies of themselves or against other individuals in the current population. In co-evolutionary systems, strategies evolve through competition against other evolving opponents, creating an arms race dynamic in which strategies continually adapt to each other.

While co-evolution can lead to the emergence of complex behaviours, it can also introduce instability. For example, strategies may evolve to exploit specific weaknesses in particular opponents rather than demonstrating general competence. In some cases, evolutionary progress may cycle between different strategies rather than improving consistently over time (Chellapilla and Fogel, 1999).

Researchers have proposed several mechanisms to mitigate these issues. These include evaluating individuals using round-robin tournaments, maintaining archives of strong historical strategies, or including fixed baseline opponents in the evaluation process. Such methods help stabilise the evaluation process and provide more reliable indicators of performance.

In the EvoDraughts system, strategies are evaluated through repeated simulated games using a combination of random baseline and co-evolved opponents. The random baseline provides a stable and interpretable reference point, while the inclusion of co-evolved opponents introduces a more dynamic and challenging evaluation environment. The balance between these opponent types is controlled through experimental configurations, allowing the impact of different training conditions to be analysed.

2.6 Classical Search Approaches and Minimax Evaluation

Alongside evolutionary approaches, classical search algorithms remain a fundamental component of AI game-playing systems. The minimax algorithm is one of the most widely used methods for decision-making in deterministic two-player games with perfect information. In minimax search, the algorithm constructs a game tree representing possible sequences of moves and evaluates the outcome assuming optimal play by both players.

Because exploring the entire game tree is typically infeasible, practical implementations use depth-limited search combined with heuristic evaluation functions. Alpha–beta pruning further improves efficiency by eliminating branches of the search tree that cannot influence the final decision. These techniques allow search-based systems to evaluate significantly deeper game trees than would otherwise be possible.

Historically, minimax-based systems have achieved strong performance in board games when combined with carefully designed evaluation functions. In draughts and checkers programs, these evaluation functions typically consider factors such as material advantage, piece mobility, positional control, and king formation.

Within the EvoDraughts project, a minimax-based engine is implemented as an exploratory comparison system. The purpose of this component is not to replicate the performance of highly optimised competitive engines, but rather to provide a classical AI reference point for evaluating the behaviour of evolved strategies. By comparing the gameplay of evolved agents with a search-based system, it becomes possible to examine differences in strategy formation, decision logic, and play style.

This comparison highlights the conceptual distinction between the two approaches. While minimax relies on explicit search through the game tree to determine optimal moves, evolutionary approaches attempt to discover heuristic decision rules through optimisation. Studying how these different methods perform within the same game environment provides valuable insight into their respective strengths and limitations.

2.7 Positioning of the EvoDraughts Project

The EvoDraughts project builds upon prior research in evolutionary game-playing while focusing specifically on the evolution of interpretable strategies using Grammatical Evolution. Unlike earlier approaches that primarily evolved numerical parameters or evaluation weights, EvoDraughts evolves symbolic strategy expressions derived from a grammar.

By representing strategies as explicit functions of game-state features, the system enables both automated optimisation and human-readable analysis of evolved behaviour. This supports investigation not only of strategy performance but also of the decision logic underlying evolved gameplay.

The project also integrates simulation-based evaluation with a web-based interface that allows human players to interact directly with evolved strategies. This provides an opportunity to complement quantitative performance metrics with qualitative observations about gameplay behaviour.

Finally, the inclusion of a brief minimax-based comparison situates the project within the broader landscape of AI game-playing research, allowing selected evolutionary strategies to be evaluated in relation to classical search-based approaches.

Together, these elements position EvoDraughts at the intersection of evolutionary computation, interpretable AI, and game-playing research, providing a framework for investigating how grammar-based evolutionary techniques can be used to generate meaningful strategies in a turn-based board game environment.

3 Methodology

This chapter describes the methodological approach used to design, implement and evaluate EvoDraughts. As discussed in the previous chapter, existing research demonstrates the effectiveness of both evolutionary computation and classical search techniques in board-game AI. Building on this background, the EvoDraughts project investigates whether Grammatical Evolution (GE) can be used to evolve effective and interpretable strategies for the game of draughts, and how these strategies compare with simple baseline opponents and a classical search-based approach.

The project is therefore framed as an experimental software and AI study. The methodology combines the development of a complete experimental framework with systematic evolutionary experiments, standardised evaluation procedures, and a human-centred assessment through a locally hosted web application.

The overall approach was iterative and experiment-driven. Work began with the construction of a robust draughts environment and an initial evolutionary pipeline, followed by rapid experimentation on a simplified 6×6 board to validate design choices. Once correctness and basic effectiveness had been established, experiments were extended to the standard 8×8 board, and additional tooling was introduced to support standardised testing, logging, and analysis. In parallel, a minimax-based draughts engine and a web interface for human-agent gameplay were implemented, enabling comparative and qualitative evaluation in the later stages of the project.

3.1 Research Design and Overall Approach

The research design centres on building and exercising a complete framework for grammar-based evolutionary game strategy. Rather than focusing solely on algorithmic novelty, the project emphasises the careful engineering of a reproducible experimental platform and its systematic empirical evaluation.

Development proceeded in several phases. First, a core infrastructure was implemented, consisting of a draughts game engine, a feature extraction mechanism, an initial grammar for strategy representation, and an evolutionary loop using the GRAPE framework. This provided an end-to-end pipeline capable of evolving strategies and playing complete games.

Second, the system was exercised extensively on a simplified 6×6 board. The reduced board size yields shorter games and smaller search spaces, which in turn lowers computational cost and allows faster iteration over grammars, fitness definitions, and configuration parameters. This phase was used to validate rule enforcement, explore alternative representations, and tune evolutionary settings.

Third, once the approach had been validated on the smaller board, experiments were scaled up to the standard 8×8 draughts board. This required adjustments to population sizes, generation counts, and games per evaluation, as the larger board introduces greater complexity and variability in game length. The same architectural components were reused, but configurations were adapted to manage the increased computational load.

Throughout these phases, runs were carefully logged and summarised. Scripts were added to generate configuration summaries, update experiment logs, and standardise test conditions for the re-evaluation of evolved strategies. In the final phase, a minimax-based draughts engine and a Flask-based web application were integrated into the framework, enabling matches between evolved and search-based strategies and facilitating structured human evaluation of evolved agents.

3.2 Draughts Game Environment

A custom draughts engine was developed in Python to provide a controlled and reproducible environment for both automated and interactive play. Implementing the engine in-house ensured full control over the ruleset, board sizes, and integration details.
The engine supports configurable board sizes, with 6×6 and 8×8 configurations used in this project. The board is represented as a two-dimensional grid whose cells contain integer codes indicating the presence and type of piece: empty square, man, or king for each player. The rules correspond to English draughts. Men move diagonally forward on dark squares and are promoted to kings upon reaching the far rank. Kings move diagonally in both directions. Captures are mandatory: whenever at least one capturing move is available to the player to move, all non-capturing moves are treated as illegal for that turn. Multi-jump capture sequences are supported where legal.

Move generation is handled by scanning the board for all pieces belonging to the current player and constructing the set of legal moves according to these rules. The engine

distinguishes between capturing and non-capturing moves and enforces the mandatory capture rule by discarding simple moves whenever a capture is possible. Game termination occurs when a player has no remaining pieces, when a player has no legal moves, or when a maximum move limit is reached. In the latter case, the engine compares remaining material or declares a draw, depending on the configuration used for the experiment.

The same engine is used in both the evolutionary scripts and the web application, ensuring that strategies are evaluated under a consistent ruleset regardless of whether games are simulated automatically or played interactively by humans.

3.3 Feature Representation of Game State

Strategies evolved by Grammatical Evolution operate on a numerical feature representation of the game state. The design of this representation must balance expressiveness, computational tractability, and interpretability.

At each decision point, the current state is encoded as a feature vector. This vector includes a flattened representation of the board, where each cell is mapped to an integer corresponding to its contents: empty, own man, own king, opponent man, or opponent king. This flattening preserves the full spatial configuration of pieces while presenting it in a form suitable for numerical computation.

In addition to the raw board encoding, several derived features are computed. These include counts of own pieces and kings, counts of opponent pieces and kings, and differences between these counts, which together capture the current material balance. Simple positional features are also included, such as the number of own and opponent pieces occupying central squares, indicators of back-rank coverage, and measures of mobility based on the number of legal moves available to each side. Finally, the feature vector includes an indicator of which player is to move, so that expressions can be written relative to a consistent own-side-versus-opponent perspective.

This combination of raw and derived features gives strategies access to both detailed board-state information and higher-level game statistics. It also supports interpretability, as the evolved expressions can be read directly in terms of intuitive game concepts such as material advantage, central control, king safety, and mobility.

3.4 Strategy Representation Using Grammatical Evolution

Strategies in EvoDraughts are represented as grammar-derived symbolic expressions evolved using Grammatical Evolution. The representation separates genotype from phenotype: each individual in the population is a linear genome, represented as a sequence of integers, which is mapped to a syntactically valid expression by applying a context-free grammar written in Backus–Naur Form (BNF).

The grammars define a small domain-specific language for draughts strategies. Non-terminals correspond to expression categories such as full expressions, terms, factors, and feature references. Terminals include constants and references to individual features in the state vector. The production rules combine these into expressions using arithmetic operators such as addition, subtraction, multiplication, and protected division, together with comparisons and conditional constructs. Protected division is used to avoid numerical exceptions during evaluation.

Two principal styles of strategy representation are supported, depending on which evolutionary script produced the strategy. In the first approach, used in evolve_draughts.py, the evolved expression is evaluated on the current feature vector and the resulting numeric value is mapped to an index over the available legal moves. In the second approach, used in evolve_draughts2.py and evolve_draughts_8x8.py, the expression is treated as a position-evaluation function: each legal move is applied to a copy of the board, the resulting state is evaluated, and the move with the best score is selected. In both cases, the grammar constrains the search space to syntactically valid and semantically meaningful expressions.

Different grammars are maintained for different experimental settings, for example to adjust the operator set or the allowed level of complexity between 6×6 and 8×8 boards. Throughout the project, the grammars were designed to remain explicit and readable so that evolved expressions could be inspected and analysed afterwards.

3.5 Evolutionary Search Process

The evolutionary search is implemented using the GRAPE framework. For each evolutionary run, an initial population of genomes is generated, subject to bounds on genome length and derivation depth in order to control the initial complexity of the strategies. Random seeds are recorded to improve reproducibility.

Each generation consists of evaluation followed by reproduction. During evaluation, every individual strategy is tested by playing multiple games of draughts against an opponent. In the simplest configuration this opponent is a random-move player, providing a stable and easily interpretable baseline, while later configurations use a mixture of random and co-evolved opponents where specified. To reduce bias due to first-move advantage, strategies are evaluated in both roles, alternating between playing first and second across their evaluation games.

Fitness is defined as a scalar value to be minimised and is based primarily on the proportion of games won by a strategy across its evaluation set. Under this formulation, stronger strategies receive lower fitness values, with the best possible performance corresponding to consistently winning all evaluation games. In some configurations, a small additional penalty term is applied to discourage overly simple or degenerate expressions, for example expressions that ignore key features of the game state.

After evaluation, tournament selection is used to choose parents for reproduction. This selection method favours individuals with lower fitness while maintaining diversity. Offspring are generated using crossover and mutation operators acting on genomes. Crossover exchanges segments between parent genomes, while mutation perturbs individual genes. Elitism is employed to ensure that the best-performing individuals are carried forward unchanged to the next generation. In addition, a hall-of-fame mechanism records historically strong individuals.

Each run proceeds for a fixed number of generations, chosen based on preliminary experiments to allow meaningful improvement without exceeding computational budgets. Population sizes, numbers of generations, and games per evaluation were tuned empirically, with shorter, lower-budget configurations used on 6×6 and larger, more conservative settings adopted for 8×8 once the system was stable.

3.6 Fitness Evaluation and Experimental Design

Evaluating strategies in an adversarial domain introduces inherent stochasticity, as performance depends on the opponent and on the specific sequence of games played. The experimental design therefore placed particular emphasis on obtaining fitness estimates that were as robust and comparable as possible within the available computational budget.

For each evolutionary run, strategies were evaluated through repeated simulated games against baseline opponents. In the initial 6×6 configuration this baseline was a random-move player only, while later 6×6 and all 8×8 configurations incorporated a mixture of random and co-evolved opponents, with the proportion of co-evolved opponents controlled through configuration parameters.

To assess generalisation beyond the evolutionary environment, a standardised re-evaluation protocol was applied to all evolved strategies. This was implemented using the script reevaluate_strategies_match_evolution.py, which evaluates stored strategies using the same strategy interpretation as the evolutionary runs. This ensures that post-evolution testing is consistent with the evaluation logic used during training.

Each strategy was evaluated over multiple seeds, with a fixed number of games per seed, producing an aggregated win rate together with additional statistics such as confidence intervals and variance across seeds. These metrics provide a more reliable estimate of performance than single-run evaluations and allow differences between strategies to be assessed consistently.

This standardised re-evaluation framework forms the basis for all reported results in later chapters, ensuring that comparisons between strategies are based on a common and reproducible evaluation procedure.

3.7 Baseline and Comparative Methods (Minimax)

In addition to the random baseline used in evolution and standardised testing, a brief post-hoc comparison was carried out against a depth-limited minimax agent implemented in a separate engine_eval module. The minimax player uses a simple hand-crafted evaluation (material and king weighting).

For this comparison, a fixed set of representative 8×8 strategies were selected from the main evolutionary runs. Each strategy was executed with the same 8×8 position-evaluation move selection as in evolve_draughts_8x8 (consistent with training and standardised re-evaluation). Games were played alternating colours against minimax at a fixed depth (depth 2), with 50 games per strategy for reporting and verification.

Because the opening position and both players' policies are deterministic under these settings, repeated games do not provide independent random samples; the repeated trials are included for verification and to support clear reporting of outcomes.

3.8 Human Evaluation via Web Application

Quantitative evaluation through simulation is necessary but not sufficient for understanding the behaviour and perceived quality of evolved strategies. To complement simulation-based metrics, a locally hosted web application was developed to support human-agent gameplay.

The backend of the application is implemented using the Flask framework. It exposes endpoints to start new games with specified board sizes and strategies, submit human moves, validate those moves using the draughts engine, and compute and return the agent's response moves. The backend reuses the same draughts engine and core gameplay logic as the evolutionary system, ensuring consistency in rules and game behaviour, although strategy execution may differ slightly from the evolutionary evaluation pipeline.

The frontend consists of HTML templates, CSS, and JavaScript. It renders the draughts board in the browser, highlights legal moves, handles user interaction, and communicates with the backend to advance the game state. Users can select from a set of evolved strategies and play complete games against them on the 6x6 board or the standard 8×8 board.

Human evaluation was conducted through structured testing sessions. A total of ten participants each played five games against one of two selected evolved strategies, resulting in fifty human–agent games overall. After completing their games, participants were asked to rate the perceived difficulty of the strategy on a scale from 1 to 10. Game outcomes and difficulty ratings were recorded for later analysis.

The evaluation was designed to be lightweight and privacy-preserving. No personal or identifying information was collected, and participation was limited to local testing. The aim of this component was not to produce large-scale user data, but to provide an additional perspective on how evolved strategies perform when interacting with human players.

3.9 Project Management, Iteration and Risk Handling

The project followed an iterative, experiment-driven management approach, with explicit tracking of runs, configurations, and risks. A run_tracker document was maintained to record the purpose, configuration, and headline results of the main evolutionary runs. Supporting scripts were used to collate configuration parameters and outcomes into central logs, including CSV experiment logs and human-readable summary reports. This ensured that experiments could be repeated or extended with full knowledge of their original settings.

Several risks were identified and addressed through methodological choices. Premature convergence and overfitting to particular opponents were mitigated by running multiple independent runs, maintaining a hall-of-fame archive of strong strategies, and performing independent test evaluations and standardised re-evaluation. Computational cost, particularly for 8×8 experiments, was managed by beginning on a 6×6 board, tuning parameters iteratively, and increasing budgets only when necessary. Risks associated with human evaluation, including data sensitivity and participant welfare, were addressed through an ethics application, local-only deployment of the web application, and strict limitations on the type and amount of data collected.

Taken together, these methodological practices produced a framework in which strategies could be evolved, evaluated, and analysed in a systematic and reproducible manner, while maintaining a focus on both quantitative performance and qualitative, interpretable behaviour.

4 System Design and Implementation

This chapter presents the design and implementation of the EvoDraughts system. Building on the methodology described in Chapter 3, the project was implemented as a modular Python codebase comprising a custom draughts engine, a set of grammar-based evolutionary scripts, supporting tools for logging and re-evaluation, a web application for human play, and a separate minimax-based comparison component. The system was designed to support both automated experimentation and interactive use, while maintaining a clear separation between core gameplay logic, evolutionary search, evaluation tooling, and user-facing interfaces.

4.1 Repository Structure and Overall Architecture

The EvoDraughts repository is organised around a small number of core modules together with a wider set of support scripts and directories. At the top level, the main implementation files include draughts_game.py, which contains the core draughts engine; draughts_functions.py, which defines the safe arithmetic and logical operators used by grammar-derived strategies; config.py, which stores the main configuration values for different iterations and experimental settings; and the three principal evolutionary scripts: evolve_draughts.py, evolve_draughts2.py, and evolve_draughts_8x8.py.

These core files are supplemented by several important subdirectories. The grammars/ directory contains the BNF grammars used by Grammatical Evolution. The grape/ directory contains the GRAPE framework itself, which provides the mapping, mutation, crossover, and evolutionary algorithm support used by the project. The results/ directory stores the outputs of evolutionary runs, including textual summaries, statistics files, standardised re-evaluation outputs, and experiment logs. The webapp/ directory contains the Flask-based application used for human-versus-agent gameplay, while engine_eval/ contains the minimax-based comparison engine and associated scripts. Additional support is provided by scripts such as play_against_evolved.py, reevaluate_strategies_match_evolution.py, and generate_config_summary.py, each of which contributes to replay, evaluation, analysis, or experiment management.

From a design perspective, the repository separates core gameplay and strategy evolution from supporting evaluation and user interaction tools. The draughts engine, grammar files, configuration, and evolutionary scripts form the central implementation. The web application, minimax component, result-analysis scripts, and experiment-management tools extend this core in order to support comparison, inspection, and human-centred interaction. The result is a

system that is not limited to running isolated evolutionary experiments, but instead provides an end-to-end framework for evolving, replaying, testing, comparing, and interacting with draughts strategies.

4.2 Core Draughts Engine

The core draughts engine is implemented in draughts_game.py and centres on the DraughtsBoard class. This class encapsulates board state, move generation, move execution, promotion, capture handling, and game termination. The board is represented as a NumPy array of integers, with 0 denoting an empty square, 1 and 2 denoting player one's man and king respectively, and -1 and -2 denoting player two's man and king. This compact numeric encoding supports efficient copying, feature extraction, and rule enforcement.

The engine supports both 6×6 and 8×8 boards. The same class is used in both cases, with board size passed as a parameter when the board is constructed. Initial setup is handled internally, with two rows of pieces per side on the 6×6 board and three rows per side on the 8×8 board. This made it possible to use the same engine for both the developmental 6×6 experiments and the later 8×8 experiments without duplicating rule logic.

Moves are represented as tuples of source and destination coordinates. Legal move generation is handled by scanning the board for all pieces belonging to the current player and then calling a piece-level move-generation method for each. Ordinary men move diagonally forward, while kings may move diagonally in both directions. The engine distinguishes between simple moves and captures, and enforces the English draughts rule that captures are mandatory. If at least one capture is available for the current player, all non-capturing moves are excluded from the returned move set.

Multi-jump capture sequences are supported through recursive capture search. The engine traces successive jumps from a landing square and continues the sequence until no further legal captures exist. When a move is executed, continuation captures are applied automatically until the full legal sequence has been completed, and promotion to king is checked throughout this process. This ensures that move execution remains consistent with the legal-move generation logic.

Game termination is handled in two ways. First, the engine declares the game over if a player has no remaining pieces or no legal moves. Second, the play_game function enforces a maximum move limit to prevent excessively long or pathological games during simulation. If this limit is reached, the outcome is resolved by comparing the remaining material, or recorded as a draw if material is equal. This rule is particularly useful in automated experiments, where very long games would otherwise inflate computational cost.

The same draughts engine is reused across the evolutionary scripts and the web application. This is an important design decision, as it ensures that evolved strategies are trained, tested, and presented to human users under a consistent ruleset.

4.3 Feature Representation of Game State

Strategies in EvoDraughts operate on a numerical representation of the current board state. Feature extraction is implemented directly within the draughts engine through the get_board_features() method of DraughtsBoard, rather than through a separate feature-extraction module. This method returns a single feature vector that combines raw board information with a set of derived summary features.

The first part of the vector is a flattened encoding of the board. Since the board itself is stored as a numeric matrix, flattening it produces a linear representation of all squares while preserving piece identity and ownership. This provides the strategy with direct access to the full board configuration.

The flattened board is followed by eleven derived features. These include the number of pieces and kings for each player, measures of centre control, counts of pieces occupying the back ranks, mobility estimates based on the number of legal moves for each side, and an indicator of which player is currently to move. The central region used for the centre-control feature differs slightly by board size: a 2×2 central area is used on the 6×6 board, while a larger central region is used on the 8×8 board. Aside from the change in flattened-board length and centre-region size, the same feature structure is maintained across both board sizes. This design gives the evolved strategies access to both low-level and high-level game information. The flattened board preserves detailed spatial information, while the derived features provide concise indicators of concepts such as material balance, mobility, and positional structure. This is important not only for strategic expressiveness but also for

interpretability, as evolved expressions can later be read in terms of meaningful game features rather than opaque hidden representations.

4.4 Strategy Representation and Grammar Integration

The strategy representation in EvoDraughts is based on Grammatical Evolution. The grammars used by the project are stored in the grammars/ directory. These grammars define the space of legal strategy expressions and act as domain-specific languages for evolved draughts decision-making.

At the implementation level, genomes are represented as integer sequences and mapped to phenotype strings by the GRAPE framework. This mapping begins with the grammar's start rule and repeatedly expands non-terminals according to codon values in the genome. The result is a syntactically valid Python-style expression that can be evaluated at runtime. Because the grammar controls which operators, constants, and feature references are available, the search space is constrained to valid symbolic programs rather than arbitrary code.

The grammars expose feature references such as x[i], along with arithmetic and conditional operators including addition, subtraction, multiplication, protected division, maximum, minimum, negation, absolute value, and conditional branching. The safe helper functions used by these grammars are implemented in draughts_functions.py. Of particular importance is pdiv, a protected division operator that avoids division-by-zero errors by returning a safe fallback value. Expressions are evaluated in a restricted namespace using Python's eval, with built-ins disabled and only the intended strategy functions and feature variables made available. Exceptions are caught and handled so that invalid evaluations do not crash the evolutionary process.

A notable aspect of the implementation is that the project supports two different runtime interpretations of evolved expressions. In the first approach, used in evolve_draughts.py, the phenotype is evaluated on the current feature vector and the resulting numeric value is converted into an index over the available legal moves. In the second approach, used in evolve_draughts2.py and evolve_draughts_8x8.py, the phenotype is treated as a position-evaluation function: each legal move is applied to a copy of the board, the resulting board state is evaluated, and the best move is chosen based on these successor-state scores. The

same broad grammar style is therefore reused across different experimental settings, but with different move-selection semantics.

This dual interpretation is an important implementation detail because it reflects the iterative development of the project. The original direct move-index approach provided a simple end-to-end starting point, while later scripts adopted a more conventional board-evaluation interpretation for improved decision-making.

4.5 Evolutionary Pipeline Implementation

The evolutionary component of EvoDraughts is implemented through three main scripts: evolve_draughts.py, evolve_draughts2.py, and evolve_draughts_8x8.py. These scripts correspond to different stages of the project and different strategy interpretations. The first script is the original 6×6 implementation using direct move selection. The second is an improved 6×6 implementation using position evaluation and co-evolutionary elements. The third extends the position-evaluation approach to the 8×8 board and forms the main evolutionary script for full-scale draughts experiments.

All three scripts use the GRAPE framework together with DEAP-based evolutionary tooling. A typical run begins by loading the relevant grammar and configuration settings, seeding the random number generators, and creating the DEAP toolbox. Individuals are represented as grape.Individual objects and initial populations are generated using bounded random initialisation. Configuration values such as population size, number of generations, mutation probability, crossover probability, elite size, and hall-of-fame size are stored in config.py, with separate parameter sets for different stages of the project.

Fitness evaluation is performed by converting each phenotype into an executable strategy and then playing a fixed number of games against an opponent. In the simplest case this opponent is a random-move strategy. In later versions, co-evolutionary elements are incorporated, with opponent pools drawn partly from random play and partly from previously evolved or hall-of-fame strategies. Fitness is based on game outcomes and is minimised, so stronger strategies receive lower fitness values. Some configurations also apply a complexity penalty to overly simple or degenerate expressions.

Selection is performed using tournament selection. Crossover is implemented using one-point genome crossover, while mutation is implemented through per-codon integer mutation. Elitism is built into the evolutionary loop, and a hall-of-fame mechanism is maintained across generations to preserve historically strong individuals. Statistics on best and average fitness are recorded generation by generation.

The separation into three evolutionary scripts is significant. The project should not be described as using a single monolithic evolutionary implementation. Instead, the scripts reflect the staged development of the system and the shift from initial direct move-selection experiments to later position-evaluation and 8×8 experimentation.

4.6 Result Logging and Experiment Management

A substantial part of the implementation is devoted not to gameplay itself, but to structured output, logging, and experiment management. Each evolutionary run produces both a textual result summary and a statistics CSV file. The textual result file records metadata such as board size, population, generations, and evaluation settings, together with the best evolved individual and its phenotype. It also includes later test results and per-generation summaries. The statistics CSV file records the best and average fitness values for each generation and is intended to support later plotting and analysis.

These outputs are stored in the results/ directory. Over time, this directory becomes the main repository of experiment artefacts, including evolutionary summaries, standardised test results, experiment logs, configuration summaries, and analysis files. The naming convention of the result files distinguishes between 6x6 and 8×8 runs, which is useful when handling multiple stages of the project.

Several support scripts build on these stored outputs. generate_config_summary.py parses result files and produces summaries of experimental settings and best-strategy metadata. The human-maintained run_tracker.md file complements these automated logs by recording the purpose, status, and headline outcomes of the main runs for the 8×8 experiments.

This logging infrastructure is important because it turns the system into a manageable experimental platform rather than a collection of ad hoc scripts. It allows runs to be compared, revisited, summarised, and reported systematically.

4.7 Re-evaluation and Standardised Testing

Post-training comparison is implemented in reevaluate_strategies_match_evolution.py. The script walks through results/, reads each saved run's phenotype text and metadata, rebuilds executable strategies via the same reconstruction path used elsewhere, and runs the fixed multi-seed test protocol described in Chapter 5. Summary statistics are written to timestamped CSV and text files under results/.

For 8×8 strategies, move choice uses the same successor-state evaluation semantics as evolve_draughts_8x8.py, so standardised tests exercise the same decision rule as training. Storing phenotypes as strings in the result files makes this replay straightforward without preserving original genomes.

4.8 Web Application for Human Play

To support human interaction with evolved strategies, EvoDraughts includes a web application implemented in the webapp/ directory. The backend is written in Flask and is centred on app.py, while strategy loading is handled by strategy_loader.py and application-specific configuration is stored in webapp/config.py. The frontend consists of HTML templates together with CSS and JavaScript assets located in the templates/ and static/ directories.

The application supports both 6×6 and 8×8 boards, though in the final evaluation phase only the 8×8 board was used. Strategies are organised into difficulty categories through a manual mapping defined in the application configuration. When a user starts a game, the selected strategy is loaded from a result file and stored in the session together with the board state and other game metadata.

Game state is maintained entirely in the Flask session. This includes the current board configuration, current player, move count, game settings, and strategy metadata. When the player submits a move, the backend validates it using the same draughts engine used in the evolutionary code, applies the human move, computes the AI's response using the loaded strategy, and then stores the updated state back into the session. No separate database is used, and no persistent user profiles or long-term gameplay records are maintained.

The application was used to conduct structured human testing sessions in which participants played multiple games against selected evolved strategies. This ensured that the same underlying draughts engine and rules were used for both simulation and human evaluation, maintaining consistency in gameplay behaviour.

From an architectural perspective, the web application is tightly coupled to the core engine and replay functionality. It reuses the same draughts rules and strategy reconstruction logic rather than reimplementing them separately. This design helps ensure consistency between automated experiments and human-facing play, while remaining lightweight and suitable for local research use.

4.9 Minimax and Classical Search Integration

The engine_eval/ directory contains a separate 8×8 minimax-style engine (adapted from an educational checkers implementation; Tech With Tim, 2020) used only for post-hoc comparison, not for the main Grammatical Evolution fitness evaluations. It maintains its own board representation (Piece objects), while minimax_eval.py bridges positions to the project's DraughtsBoard representation so that evolved strategies receive the same feature vector and legal-move lists as in training, using evolve_draughts_8x8's position-evaluation move selection for 8×8 result files.

The minimax player performs depth-limited minimax with a simple static evaluation (material balance with a king bonus). minimax_debug.py supports traced games for inspection. Running minimax_eval.py evaluates selected strategies from results/ and writes timestamped .txt and .csv summaries under results/ (e.g. win/draw/loss totals for a fixed depth and game count).

This component is deliberately small in scope: it provides a classical search reference alongside the main EvoDraughts pipeline, without changing how strategies are evolved in evolve_draughts_8x8.py.

4.10 Supporting Scripts and Utilities

Besides the core engine, evolution scripts, web app, and minimax module, a few scripts or utilities tie results/ to replay and reporting. play_against_evolved.py is particularly important, it parses saved run files (load_strategy_from_file), builds a playable strategy from the

phenotype (strategy_from_phenotype), and offers a small CLI to list strategies and play against a random opponent. The web app and the minimax module reuse the same loading helpers so phenotype parsing stays in one place.

generate_config_summary.py scans evolution .txt files and writes consolidated configuration summaries (e.g. CSV) for comparing runs and supporting documentation. analyze_results.py and plot_standardized_test_evolution_match.py parse runs or the latest standardised-test CSV and produce plots under results/analysis/ for figures such as Figure 6.1 – 6.4. They do not define the evaluation protocols in Chapter 5; they support inspection and presentation after the fact.

4.11 Summary

In summary, the EvoDraughts implementation is structured as a modular experimental platform for evolving and evaluating draughts strategies. The core of the system is a custom draughts engine that supports both 6×6 and 8×8 boards, combined with grammar-based strategy evolution through GRAPE and DEAP. Around this core, the project includes structured logging and re-evaluation tools, a local web application for human play, and a separate minimax-based comparison engine. The result is a system that supports not only evolutionary search, but also replay, testing, analysis, comparative evaluation, and human interaction.

5 Experimental Setup

This chapter describes how the EvoDraughts system was used to conduct experiments and evaluate evolved strategies. While Chapter 4 detailed the system implementation, this chapter focuses on the experimental configurations, the training and testing protocols, and the procedures used to ensure fair and reproducible comparison between strategies.

5.1 Overview of Experimental Approach

Experiments were conducted in two stages. An initial stage used a 6×6 draughts board to validate the evolutionary pipeline and explore parameter settings under reduced computational cost. The main stage of the project then used the standard 8×8 board, which forms the primary basis of the final analysis.

Evaluation took place at several levels. During evolution, strategies were assessed through repeated simulated gameplay against random or mixed opponent pools. After training, all selected strategies were subjected to a standardised re-evaluation procedure under fixed test conditions. Additional post-hoc experiments were conducted against a minimax-based opponent, and selected 8×8 strategies were also tested through human play using the local web application.

5.2 Evolutionary Configuration

Different evolutionary configurations were used for the 6×6 and 8×8 stages.

For the initial 6×6 experiments, two main configurations were used. The earlier configuration used a population of 200 individuals over 100 generations, with a mutation rate of 0.05 and 20 games per evaluation. A later 6×6 configuration used a population of 300 individuals over 50 generations, with a mutation rate of 0.10 and 25 games per evaluation. This second configuration also introduced co-evolutionary elements.

For the 8×8 experiments, the main configuration used a population of 400 individuals over 75 generations, with a crossover rate of 0.8 and 20 games per evaluation. Mutation rates varied across conditions, taking values of 0.10, 0.15, and 0.20. Co-evolution ratios were also varied, with different runs using 30%, 40%, 50%, or 60% co-evolved opponents. A complexity penalty was included in some configurations to discourage trivial strategies.

These parameter changes were not random adjustments, but part of a structured sequence of runs recorded in the run tracker. In this way, the 8×8 experiments were used not only to evolve stronger strategies, but also to compare how different training conditions affected generalisation, variance, and consistency.

The 8×8 experiments followed a structured experimental plan rather than ad hoc parameter tuning. After establishing a baseline configuration (Config_6), this configuration was replicated multiple times with different random seeds to assess consistency. Following this, four experimental conditions were defined and tested systematically:

- Condition A: reduced co-evolution (30%)
- Condition B: increased co-evolution (50%)
- Condition C: reduced mutation rate (0.10)
- Condition D: increased mutation rate (0.20)

Each condition was executed with three independent replicates using different random seeds, resulting in a total of 17 main 8×8 runs. This design allows performance to be analysed not only in terms of absolute outcomes, but also in terms of consistency across repeated runs under the same configuration.

5.3 Training Evaluation Protocol

During evolution, each strategy was evaluated by playing multiple games of draughts against an opponent. To reduce first-move bias, strategies were always evaluated in both roles, playing an equal number of games as the first and second player.

In the baseline 6×6 configuration, strategies were evaluated exclusively against a random-move opponent. In later 6×6 and all 8×8 configurations, training used a mixture of random and co-evolved opponents. The exact proportion of co-evolved opponents depended on the condition being tested. This allowed the project to compare random-only training with mixed-opponent training under otherwise similar settings.

A maximum move limit was imposed to prevent excessively long games. This was set to 100 moves for 6×6 experiments and 200 moves for 8×8 experiments. If this limit was reached, the result was determined from the remaining board state according to the engine rules.

Randomness during evolution was controlled through fixed run seeds. These seeds were recorded alongside the result files, allowing individual runs and conditions to be tracked and reproduced.

5.4 Standardised Re-evaluation

Following training, all selected strategies were re-evaluated under a common testing protocol. This procedure was used to produce the final comparative results reported in Chapter 6.

Each strategy was tested using five fixed random seeds. For each seed, the strategy played 30 games against the same random baseline opponent, giving a total of 150 games per strategy. The use of fixed seeds ensured that all strategies were exposed to the same opponent behaviour, allowing like-for-like comparison across runs.

The final re-evaluation procedure was aligned with the same strategy interpretation used during evolutionary runs. This is important because it ensures that stored strategies are tested under conditions consistent with those used during training. The outputs recorded for each strategy include total wins, test win rate, confidence interval, variance across seeds, and train–test gap.

This standardised re-evaluation forms the basis of the final reported comparisons between strategies and between experimental conditions.

5.5 Strategy Selection for Analysis

For each evolutionary run, the best-performing strategy was selected for later analysis. In practice, this meant taking the final best individual recorded in the corresponding result file and subjecting it to standardised re-evaluation.

Only one strategy per run was used in the main comparative analysis. This allowed the report to compare runs as experimental units rather than comparing large numbers of individuals from within the same run. Run metadata, configurations, and outcomes were tracked through the saved result files, the run tracker, and supporting experiment logs.

This approach supports the analysis of questions such as which configurations generalised best, whether lower training fitness predicted stronger test performance, and which conditions produced lower variance across repeated testing.

5.6 Minimax Evaluation Setup

A supplementary minimax comparison was conducted after the main evolutionary experiments, using the harness described in Section 4.9. The minimax agent used fixed depth 2 and the same 50 games per strategy reported in the results log.

A fixed shortlist of representative 8×8 strategies was chosen from the main runs (filenames listed in Section 6.5 / the archived minimax_eval_*.txt output). Strategies were executed with evolve_draughts_8x8 position-evaluation move selection, matching training and standardised re-evaluation. Games alternated colours (25 games as White, 25 as Red when 50 games are used).

Under deterministic play, outcomes are not independent samples; the 50 games serve verification and reporting (see Section 3.7). No evolutionary runs included in this project used minimax as a training opponent; this comparison is not part of the fitness function used in Chapter 5's evolutionary configuration.

5.7 Human Evaluation Setup

Human evaluation was conducted using the local web application on the full 8×8 board. Two evolved strategies were selected for testing: evolution_8x8_20260320_104123.txt and evolution_8x8_20260316_010459.txt.

A total of ten participants took part, with five participants assigned to each strategy. Each participant played five games against their assigned strategy, producing a total of 25 games per strategy and 50 human-versus-agent games overall.

After completing their games, each participant was asked to rate how challenging they found the strategy on a scale from 1 to 10, where 10 indicated an opponent that felt almost impossible to beat and 1 indicated an opponent the participant felt confident beating consistently. Alongside these difficulty ratings, game outcomes were recorded for later analysis.

The human evaluation was intended to complement simulation-based testing by providing a small but structured indication of how the evolved agents performed against real players and how challenging they were perceived to be in practice.

5.8 Reproducibility and Experiment Control

Reproducibility was a central consideration throughout the experimental design. Each evolutionary run used a defined random seed, which was recorded in the corresponding result file and tracked across configurations. The standardised re-evaluation also used a fixed set of test seeds, ensuring that strategies could be compared under identical conditions.

A structured logging system was used to capture outputs at multiple stages. Evolutionary runs produced result files and per-generation statistics, while the re-evaluation process generated standardised test results and summary reports. Supporting scripts were used to maintain configuration summaries and experiment logs, allowing runs to be traced back to their parameter settings and outcomes. The minimax comparison outputs also generated txt and csv result files.

Together, these practices ensured that the experiments were organised systematically, that comparisons were reproducible, and that later analysis could be based on a clear and consistent record of how each strategy had been produced and tested.

6 Results

This chapter presents the results obtained from the final set of evolutionary experiments and the subsequent standardised re-evaluation. Reported results are based on the final evaluation procedure described in Chapter 5, using five fixed seeds and 150 games per strategy against a random baseline opponent. Minimax comparison and human evaluation results are also included.

6.1 Overview of Results

A total of 8 strategies from 6×6 experiments and 17 strategies from 8×8 main evolutionary runs were evaluated using the standardised re-evaluation procedure. A separate post-hoc minimax comparison used additional games and is not part of the multi-seed random-baseline protocol.

The results are reported in terms of:

- Test win rate (wins out of 150 games)
- 95% confidence interval
- Variance across seeds
- Train–test gap

The 8×8 experiments form the primary focus of the project, while the 6×6 results are included mainly for comparison and validation.

6.2 6×6 Results

The 6×6 strategies showed a wide range of performance under standardised testing.

Best strategy:

- evolution_20251215_032724.txt → 127/150 wins (84.7%), CI: 78.9%–90.4%

Second best:

- evolution_20251216_034713.txt → 122/150 wins (81.3%), CI: 75.1%–87.6%

Lowest performance:

- evolution_20251214_220441.txt → 48/150 wins (32.0%)

Summary statistics (6×6):

- Mean win rate: 52.2%
- Standard deviation: 18.9%
- Range: 32.0% – 84.7%

The results show substantial variation between runs, with some strategies performing strongly against the random baseline and others performing poorly.

6.3 8×8 Results (Main Experiments)

The 8×8 experiments produced consistently stronger performance overall, though variation between runs was still present.

Best-performing strategy:

- evolution_8x8_20260219_203646.txt → 142/150 wins (94.7%), CI: 91.1%–98.3%

Other high-performing strategies:

- evolution_8x8_20260218_003143.txt → 82.7%
- evolution_8x8_20260309_155326.txt → 81.3%
- evolution_8x8_20260320_104123.txt → 81.3%
- evolution_8x8_20260216_052949 and 20260311_043951.txt→ 78.0%

Lowest-performing strategy:

- evolution_8x8_20251216_222007.txt → 67/150 wins (44.7%)

Summary statistics (8×8 main runs):

- Mean win rate: 73.6%
- Standard deviation: 10.2%
- Range: 44.7% – 94.7%
- Number of runs: 17

Compared to 6×6, the 8×8 results show:

- higher average performance
- lower relative spread
- more consistent behaviour across runs

Figure 6.1 shows the standardised test performance of all evaluated strategies across both 6×6 and 8×8 experiments. Each bar represents the test win rate of a single strategy under the fixed multi-seed evaluation protocol, with the dashed line indicating the 50% win rate baseline corresponding to random play.

The figure highlights the clear difference between 6×6 and 8×8 performance distributions. The majority of 8×8 strategies achieve win rates well above 70%, with several exceeding 80% and one reaching approximately 95%. In contrast, the 6×6 strategies show substantially greater variability, with results spanning from below 40% to above 80%.

This visualisation reinforces the summary statistics presented earlier, showing both the improved average performance and the reduced relative spread in the 8×8 experiments compared to the 6×6 runs.

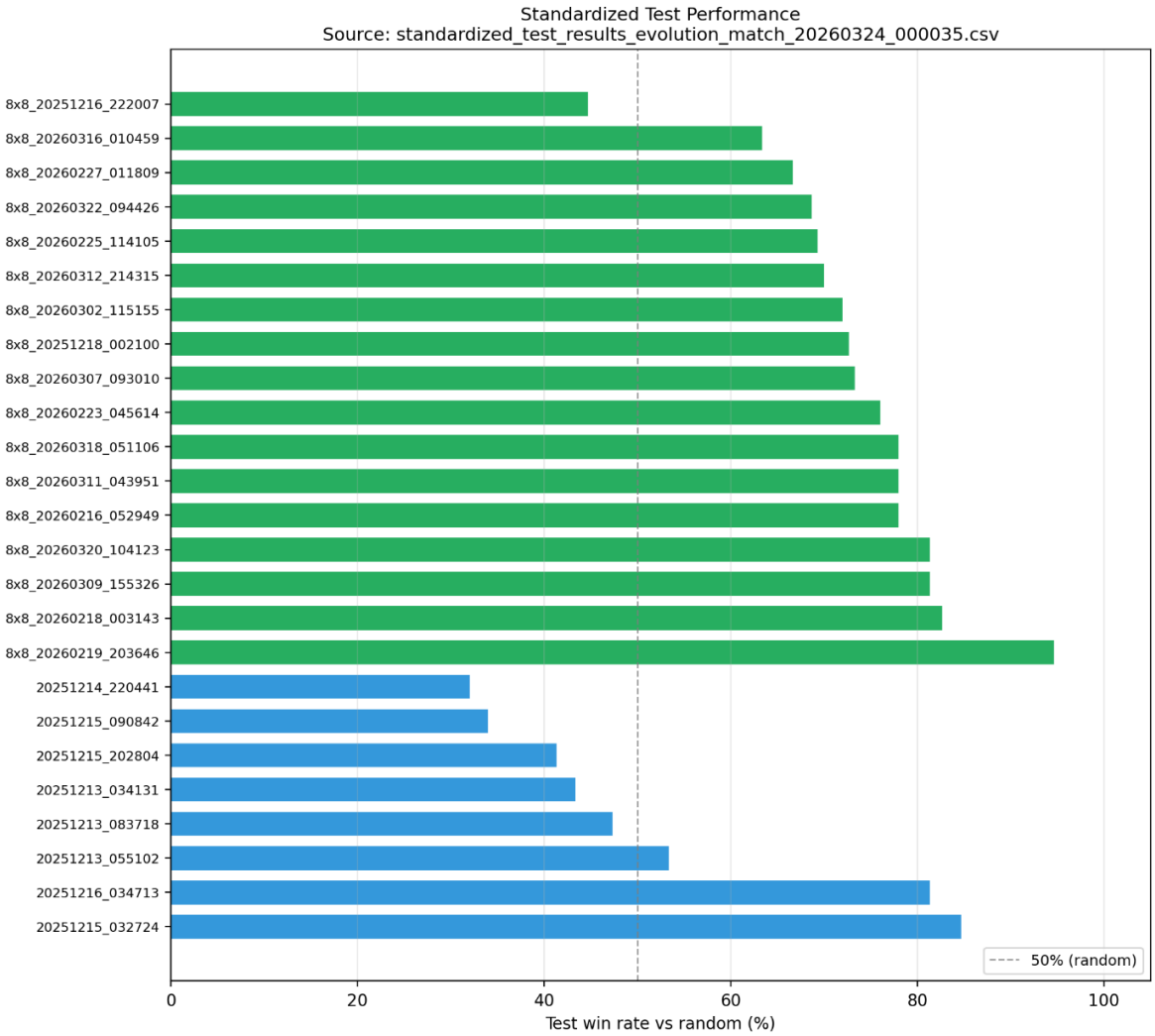

Figure 6.1: Standardised test performance of all evolved strategies under the fixed multi-seed evaluation protocol.

Figure 6.2 compares training win rate with standardised test win rate. Points below the diagonal indicate strategies whose test performance fell short of training performance; points near the diagonal indicate better alignment between the two measures. The spread is wider for 6×6 than for most 8×8 strategies, consistent with the train–test gaps reported in Section 6.4 and discussed in Chapter 7.

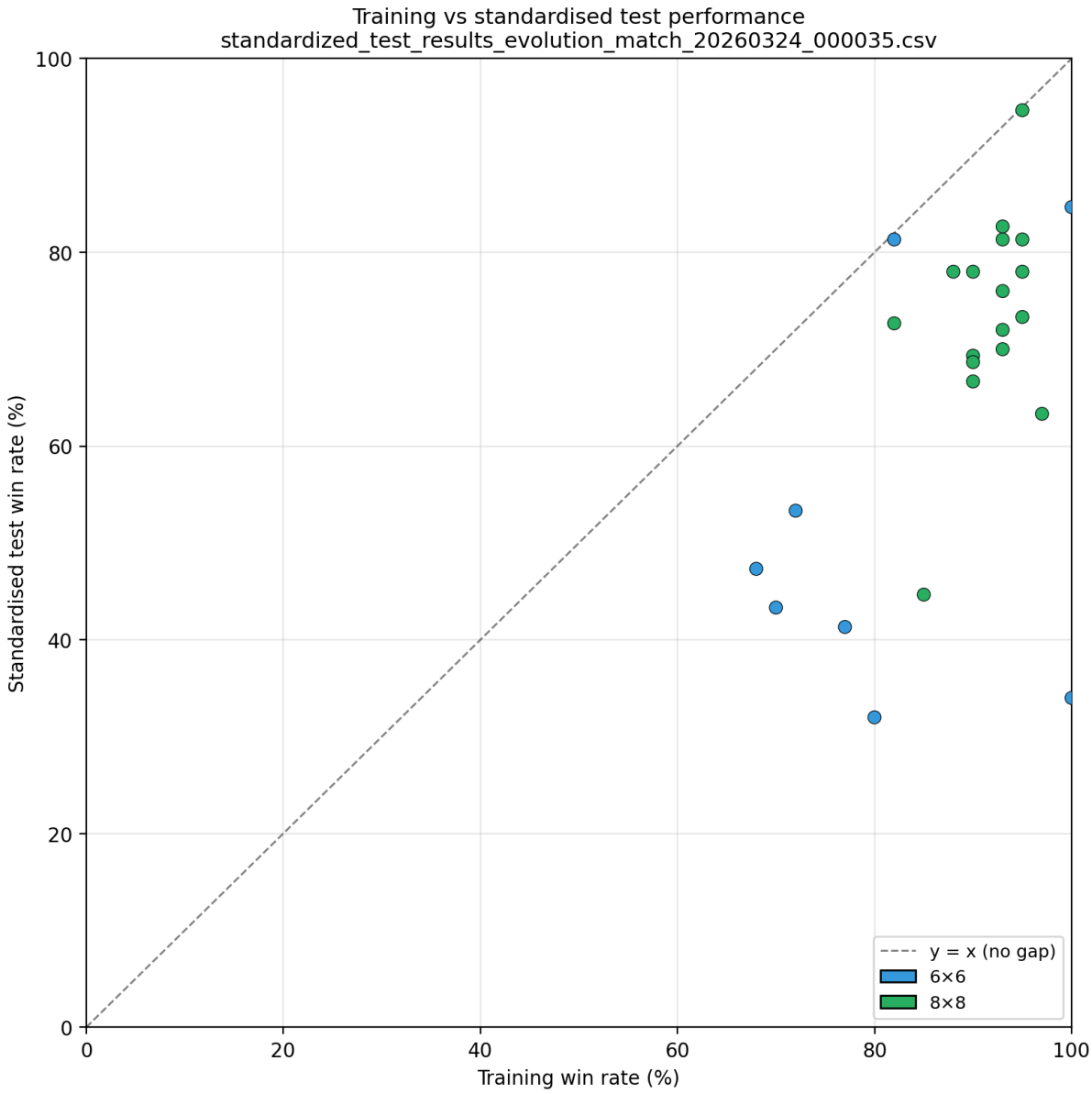

Figure 6.2: Training win rate (evolutionary fitness) versus standardised test win rate for each strategy.

Figure 6.3 shows how the best fitness in the population improved over generations during training for each 8×8 run. Trajectories refer to the fitness used in evolution (games against random and co-evolved opponents), not the post-hoc standardised test.

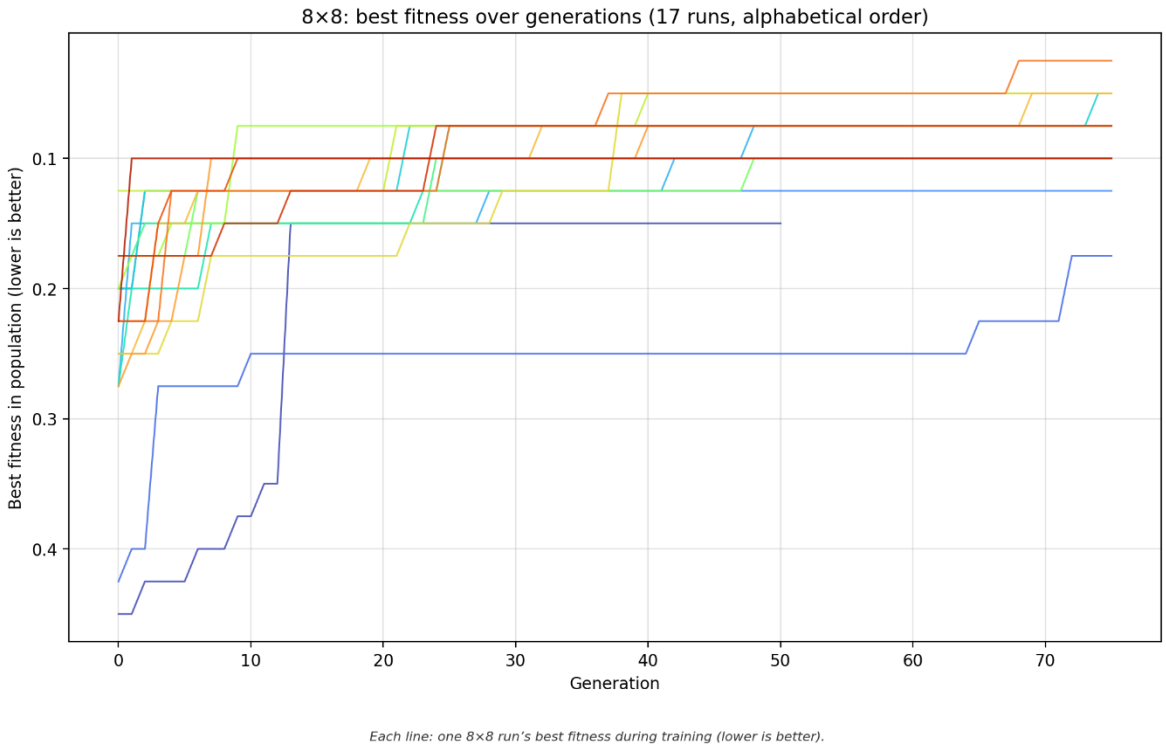

Figure 6.3: Best fitness in the population by generation for every logged 8×8 evolutionary run.

Figure 6.4 shows the learning behaviour within a single representative 8×8 evolutionary run. The best of generation fitness decreases rapidly in the early generations before plateauing, indicating that strong individuals are discovered relatively early in the search process. In contrast, the average population fitness improves more gradually, reflecting the slower adaptation of the overall population.

This gap between best and average fitness suggests that while higher-performing strategies emerge early, the population as a whole does not fully converge to these solutions. This indicates that the evolutionary process maintains diversity rather than collapsing to a single dominant strategy, but also highlights that improvements are not uniformly distributed across individuals.

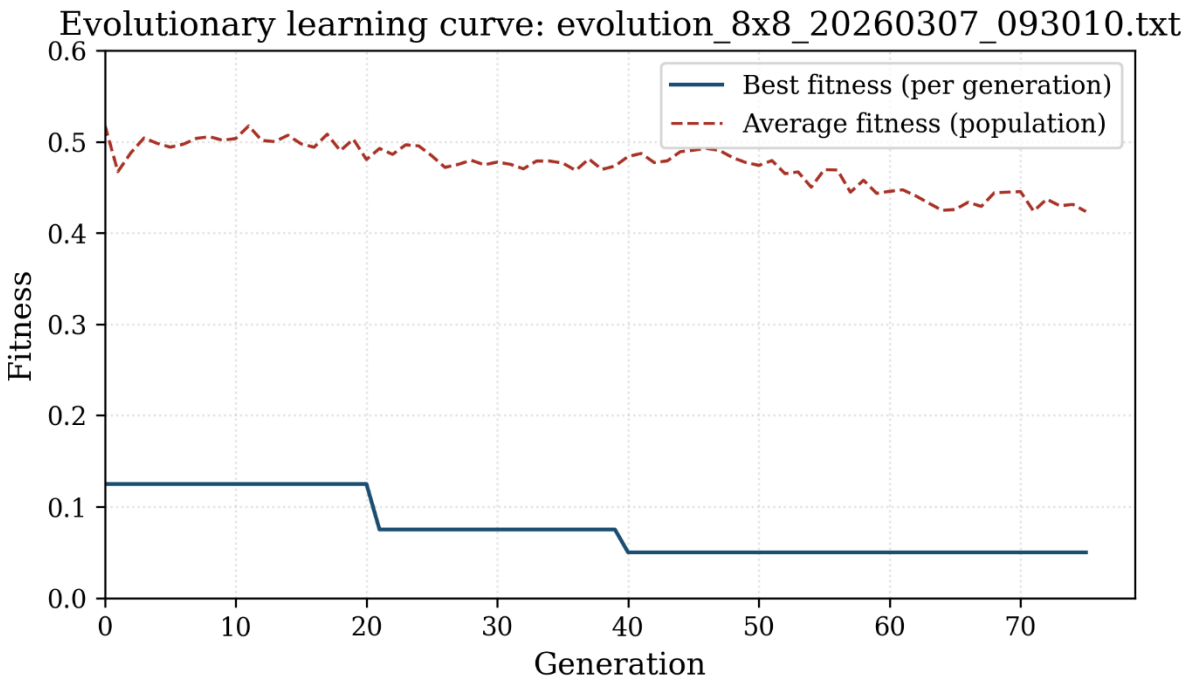

Figure 6.4: Best of generation fitness and average fitness over generations for a representative 8×8 evolutionary run (evolution_8x8_20260307_093010.txt).

6.4 Effect of Experimental Conditions

The 8×8 experiments were organised into a structured set of conditions, each executed with three independent replicates using different random seeds. This allows comparison not only between conditions, but also within-condition consistency.

The conditions evaluated were:

- Baseline (Config_6, 40% co-evolution, mutation 0.15)
- Condition A (30% co-evolution)
- Condition B (50% co-evolution)
- Condition C (mutation 0.10)
- Condition D (mutation 0.20)

Figure 6.5 summarises standardised test performance by condition. Baseline (Config_6) includes multiple independent replicates; Conditions A–D each pool three replicates (Chapter 5). The plot makes within-group spread and differences between groups visible alongside the numerical summaries in the text.

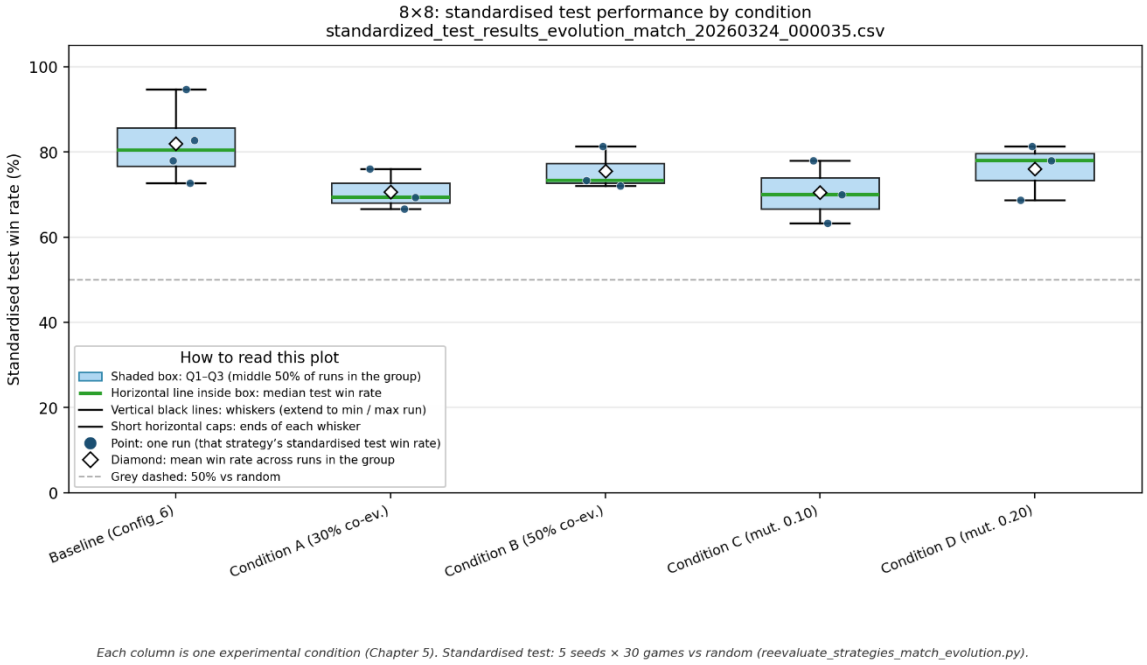

Figure 6.5: Distribution of standardised test win rates by experimental conditions.

Different experimental conditions produced different performance outcomes.

- The baseline configuration (Config_6) produced strong performance across its replicates, including one of the highest-performing strategies at 94.7% win rate.
- Condition A (30% co-evolution) produced strategies with win rates generally between 66.7% and 76.0%, showing moderate performance.
- Condition B (50% co-evolution) showed variation across runs, with results ranging from 72.0% to 81.3%.
- Condition C (mutation 0.10) produced mixed results, with win rates between 63.3% and 78.0%. One replicate performed notably lower (63.3%), while the others achieved higher performance.
- Condition D (mutation 0.20) produced results between 68.7% and 81.3%, indicating competitive performance but with variation across replicates.

Across all runs, variance values ranged from 0.013 to 0.092, indicating that some strategies performed consistently across seeds while others showed greater variability.
Train–test gaps ranged from near zero (e.g. 0.3%) to over 40%, indicating that some strategies generalised well while others performed significantly worse under standardised testing.

6.5 Minimax Results

A post-hoc head-to-head evaluation was run using minimax_eval.py with depth 2 and 50 games per strategy, alternating colours. Outputs were saved as results/minimax_eval_20260405_192623.txt (and matching .csv). Strategies were executed with evolve_draughts_8x8 position-evaluation semantics, consistent with the main 8×8 experiments.

Aggregate win/draw/loss totals were:

- evolution_8x8_20260316_010459.txt → 25 Wins, 25 Draws, 0 Losses
- evolution_8x8_20260219_203646.txt → 0 Wins, 25 Draws, 25 Losses
- evolution_8x8_20251218_002100.txt → 0 Wins, 0 Draws, 50 Losses
- evolution_8x8_20260312_214315.txt → 0 Wins, 0 Draws, 50 Losses
- evolution_8x8_20260227_011809.txt → 0 Wins, 25 Draws, 25 Losses

The evolution_8x8_20260316_010459.txt strategy recorded wins against this agent (25 wins; the remaining 25 games were draws), while the other listed strategies recorded no wins under these settings.

Because play is deterministic, the split into 25 vs 25 outcomes by colour is expected; the repeated games are for verification/reporting, not independent sampling (see Section 3.7).

6.6 Human Evaluation Results

Human evaluation was conducted using two selected 8×8 strategies:

- evolution_8x8_20260320_104123.txt
- evolution_8x8_20260316_010459.txt

A total of 10 participants took part, each playing 5 games, resulting in:

- 25 games per strategy
- 50 games total

Strategy 1: evolution_8x8_20260320_104123.txt

- GE wins: 7 / 25
- Player wins: 18 / 25
- Win rate (GE): 28%

Difficulty ratings (1–10):

- Values: 5, 4, 4, 7, 2
- Mean rating: 4.4 / 10

Strategy 2: evolution_8x8_20260316_010459.txt

- GE wins: 4 / 25
- Player wins: 21 / 25
- Win rate (GE): 16%

Difficulty ratings:

- Values: 4, 3, 5, 2, 5
- Mean rating: 3.8 / 10

Across both strategies:

- Human players won the majority of games
- Difficulty ratings were generally low to moderate

Figure 6.6 summarises the outcomes of the human evaluation for both selected strategies, including win rates and perceived difficulty ratings. The results highlight that human players won the majority of games against both strategies, and that perceived difficulty remained relatively low.

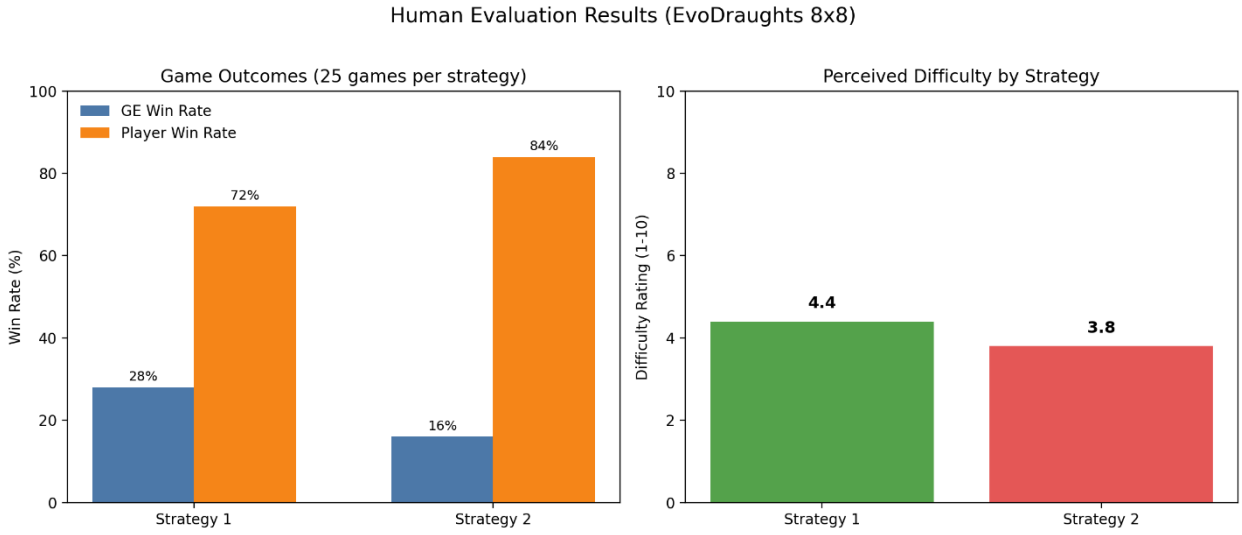

Figure 6.6: Human evaluation results for two selected 8×8 evolved strategies, showing win rates and average perceived difficulty ratings across participants.

6.7 Summary of Key Results

- Best 8×8 strategy achieved 94.7% win rate against the random baseline
- Mean 8×8 performance was 73.6%, compared to 52.2% for 6×6
- Performance across runs varied, with some strategies performing poorly (as low as 44.7%)
- Train–test gaps varied widely, indicating differences in generalisation
- In the post-hoc minimax comparison one selected strategy recorded a win against the minimax agent, other selected strategies drew or lost under the same settings
- Human players were able to defeat both tested strategies consistently
- Perceived difficulty ratings ranged from 2/10 to 7/10, with averages below 5

7 Discussion

7.1 Overview of Findings

The results show a clear difference between the 6×6 and 8×8 experiments. While the 6×6 runs produced highly variable outcomes, the 8×8 runs consistently achieved higher performance against the random baseline, with a mean win rate of 73.6% and a best result of 94.7%.

This reflects both the intended role of the two environments and an important characteristic of the evolutionary process. The 6×6 board was primarily used for validation and rapid experimentation, while the 8×8 board represents the main experimental setting of the project. The improved performance on the larger board suggests that Grammatical Evolution benefits from a richer search space, where simple heuristics can exploit more structure in the game. At the same time, the continued variability across runs indicates that the evolutionary process remains sensitive to initial conditions and stochastic effects.

However, strong performance against a random baseline opponent should be interpreted as demonstrating that the system is functioning correctly, rather than as evidence of strategically strong play.

7.2 Strategy Representation and Evolution Approach

The transition from direct move selection to position-evaluation-based strategies corresponds with improved and more consistent performance in later experiments.

The position-evaluation approach allows strategies to evaluate resulting board states rather than selecting moves directly, which aligns more closely with classical game-playing approaches such as minimax. This change has a significant impact on the evolutionary process: instead of learning arbitrary mappings from state to move index, the system evolves general-purpose evaluation functions that can be applied consistently across different positions. This leads to more stable decision-making and improved performance, as strategies are no longer tied to specific move orderings or local artefacts of the game representation.

Grammatical Evolution provides a structured way to generate these strategies while maintaining syntactic validity and interpretability. However, the results show that this structure also constrains the search space in important ways. Many high-performing strategies are relatively simple expressions, and in some cases the evolutionary process converges to

trivial solutions such as constant-valued functions. This suggests that GE, under the given fitness formulation, has a tendency to favour low-complexity solutions that are sufficient to exploit the evaluation environment. While this supports interpretability, it also highlights a limitation: the representation does not inherently encourage the development of deeper or more strategic behaviour.

7.3 Effect of Experimental Conditions

The structured experimental design allows comparison across conditions and between replicates.

The baseline configuration (Config_6) produced strong and consistent performance, including the highest-performing strategy overall (94.7%). This suggests that the chosen baseline configuration was already effective for the 8×8 problem.

Reducing co-evolution to 30% (Condition A) resulted in moderate performance, while increasing it to 50% (Condition B) introduced greater variability across runs. This suggests that while co-evolution can introduce a more challenging training environment, it also destabilises the fitness landscape. As strategies adapt to each other, the objective being optimised becomes less consistent, which can lead to oscillating or inconsistent performance rather than steady improvement.

Reducing the mutation rate to 0.10 (Condition C) produced mixed results across the three replicates. While two runs achieved strong performance, one replicate was notably weaker at 63.3%, indicating that this configuration was not consistently strong. Increasing mutation to 0.20 (Condition D) also produced competitive strategies but with greater variation between runs.

Overall, the results suggest that moderate co-evolution and controlled mutation rates provide a balance between exploration and stability. From a Grammatical Evolution perspective, this reflects a broader trade-off: sufficient variation is required to explore the search space, but excessive variation or overly dynamic opponents can prevent convergence to stable strategies. The most effective configurations therefore appear to be those that introduce challenge gradually without disrupting the evolutionary signal.

7.4 Generalisation and Train–Test Gap

A clear difference between training performance and standardised test performance is observed across all runs.

In the 6×6 experiments, the train–test gap is large and highly variable, indicating that many strategies overfit to their training conditions. In contrast, the 8×8 experiments show smaller and more consistent gaps, suggesting improved generalisation.

Some strategies show near-zero train–test gap, indicating stable performance between training and evaluation, while others show gaps exceeding 40%, indicating overfitting or instability. In practical terms, overfitting in this system typically corresponds to strategies that perform well against the specific opponent behaviours encountered during training but fail to generalise to slightly different conditions. For example, strategies may learn to exploit predictable patterns in random or co-evolved opponents, such as consistently selecting moves that are only effective against weak or non-adaptive play, but which are easily countered under different gameplay dynamics.

The use of a fixed multi-seed re-evaluation protocol provides a more reliable measure of performance and highlights these differences in generalisation. It also exposes cases where strategies that appear strong during training are in fact exploiting artefacts of the evaluation setup rather than demonstrating robust gameplay. This is further supported by the presence of trivial or low-complexity strategies achieving competitive training performance, indicating that the fitness function does not always distinguish between genuinely strong strategies and those that are merely well-adapted to a narrow evaluation context.

7.5 Which Runs Generalised Best

The strongest-performing strategy (evolution_8x8_20260219_203646.txt) achieved both a high test win rate (94.7%) and a very small train–test gap (0.3%), indicating strong generalisation.

Other high-performing strategies also show relatively small gaps, suggesting that strong performance is often associated with stable generalisation rather than overfitting. This indicates that, within this system, the most effective strategies are those that capture broadly useful patterns in the game rather than exploiting specific opponent behaviours. In the context

of Grammatical Evolution, this suggests that successful individuals are not necessarily the most complex or highly optimised for training fitness, but those that maintain consistent behaviour across varied game conditions.

In contrast, lower-performing strategies tend to exhibit larger train–test gaps, indicating that good training performance alone is not a reliable indicator of final performance.

7.6 Variance and Consistency Across Runs

Variance across seeds provides an indication of how consistent a strategy is under different game sequences.

Some strategies achieved very low variance (e.g. 0.013), indicating highly consistent behaviour across different seeds. Others showed higher variance (up to 0.092), suggesting sensitivity to specific game sequences or opponent behaviour. In practice, higher variance strategies tend to rely on behaviours that are effective in some situations but fail in others, indicating a lack of robustness. Lower variance strategies, by contrast, appear to apply more consistent decision rules that generalise across different game trajectories.

Across conditions, no single configuration eliminated variance entirely, but some settings, particularly those with moderate mutation and co-evolution, produced more stable outcomes. This reinforces the view that stability in Grammatical Evolution depends not only on the quality of individual strategies, but also on the consistency of the evolutionary process itself. Configurations that introduce excessive randomness or instability in evaluation tend to produce strategies whose performance is less predictable.

The use of multiple replicates per condition makes it possible to distinguish between consistent performance and results that occur due to random variation.

7.7 Comparison with Minimax

The minimax comparison is intentionally brief. It uses a shallow search depth, a simple evaluation function, and a separate engine bridged to the project's feature representation. It does not establish overall strength versus classical AI in general but shows how a few evolved strategies behave against one fixed search-based opponent after training.

The observed outcomes are consistent with deterministic play (Section 3.7): each colour can produce a single repeated trajectory, so aggregate statistics are not analogous to the multi-seed random-baseline testing. The notable empirical point for this project is that at least one representative evolved strategy achieved decisive wins in that evaluation (evolution_8x8_20260316_010459.txt), while others did not under identical settings. This highlights that performance is strongly opponent-dependent, even when random-baseline test performance is high. In particular, it suggests that strategies evolved primarily against random or co-evolved opponents may not consistently perform well against more structured or deterministic play. This reinforces the limitation that strong performance in the evolutionary setting does not necessarily translate to robustness against fundamentally different types of opponents.

7.8 Human Evaluation

Human evaluation results show that both tested strategies were consistently beatable by human players.

Across both strategies:

- Human players won the majority of games
- Difficulty ratings ranged from 2/10 to 7/10
- Average difficulty ratings were below 5/10

The stronger of the two tested strategies achieved a higher win rate against human players (28%) and a higher average difficulty rating (4.4/10) but was still perceived as only moderately challenging. Even in this case, players were able to adapt after a small number of games, suggesting that the strategy lacked variability and could be learned and countered over time.

These results indicate that while evolved strategies perform well against a random baseline, they do not yet produce behaviour that is consistently challenging for human players. In practice, human players were able to exploit weaknesses such as predictable move selection, limited long-term planning, and poor defensive positioning. Strategies often responded effectively to immediate board features but failed to maintain coherent plans over multiple moves, making them vulnerable to simple tactical play.

7.9 Interpretability of Evolved Strategies

A key advantage of the approach is that evolved strategies are expressed as symbolic expressions over defined features.

This makes it possible to inspect individual strategies and identify patterns in their behaviour. For example, many high-performing strategies rely on relatively simple combinations of features, suggesting that effective behaviour can emerge without highly complex expressions. This indicates that, within this system, Grammatical Evolution tends to favour compact and easily interpretable solutions, which are sufficient to achieve strong performance under the given evaluation conditions.

At the same time, some strategies reduce to trivial expressions (e.g. constant values or single-feature references) and still achieve moderate performance, highlighting the importance of evaluation design and opponent choice. This demonstrates a key limitation of the approach: while interpretability is achieved through constrained symbolic representations, the evolutionary process does not inherently promote meaningful or strategically deep expressions, and may instead converge on simplistic behaviours that exploit weaknesses in the evaluation setup.

7.10 Limitations

Several limitations affect the interpretation of the results.

- Performance is measured primarily against a random baseline, which does not represent strong strategic play. As a result, high win rates reflect the ability to exploit weak or non-adaptive opponents rather than demonstrating robust or strategically deep behaviour
- The minimax comparison is limited in depth and engine choice so it cannot support broad claims about classical search versus evolution
- Human evaluation is based on a small sample size (10 participants and 2 strategies)
- Some strategies achieve strong performance despite simple or unintuitive expressions, indicating weaknesses in the evaluation setup. In particular, the fitness function does not reliably distinguish between genuinely strong strategies and those that exploit artefacts of the opponent or move selection process, contributing to overfitting and the emergence of trivial solutions

These limitations mean that while the system demonstrates effective behaviour under the chosen evaluation conditions, it does not yet demonstrate strong strategic competence in a broader sense.

7.11 Summary

The discussion shows that:

- The 8×8 experiments produce stronger and more consistent results than 6×6
- Position-evaluation strategies improve stability and performance
- Moderate co-evolution and controlled mutation produce the most reliable outcomes
- Strong test performance is associated with good generalisation, not just training fitness, indicating that Grammatical Evolution can produce stable strategies when the evaluation environment is sufficiently consistent
- Minimax comparison performance is opponent dependent with one strategy winning games and others drawing or losing
- Human players are still able to defeat the evolved strategies consistently

Overall, the results demonstrate that the system is capable of evolving functional and interpretable strategies, but also highlight the gap between performance against simple baselines and genuinely strong gameplay.

8 Conclusion

8.1 Conclusion

This project presented EvoDraughts, a system for evolving draughts strategies using Grammatical Evolution. The system integrates a custom draughts engine, a grammar-based strategy representation, and an evolutionary pipeline to generate interpretable decision-making strategies.

The results demonstrate that the approach is capable of producing functional strategies, particularly in the 8×8 environment, although this performance is primarily observed against simple baseline opponents rather than strong or adaptive gameplay. Across multiple experimental runs, evolved strategies achieved consistently strong performance against a random baseline, with a best result of 94.7% and an average performance of 73.6%. The structured experimental design also showed that performance depends on configuration choices, with moderate co-evolution and controlled mutation rates producing the most consistent outcomes.

The project also highlights the importance of evaluation methodology. While strong results were achieved against a random opponent, human testing showed that these strategies remain beatable and only moderately challenging in practice. This indicates that performance against simple baselines should be interpreted as demonstrating that the system is functioning correctly, rather than as evidence of strong strategic competence.

In addition, the use of Grammatical Evolution enables strategies to be represented as explicit symbolic expressions. This provides a level of interpretability that is not typically available in more opaque approaches such as neural networks, allowing strategies to be inspected and analysed in terms of game-state features.

Overall, the project demonstrates that grammar-based evolutionary methods can be used to generate functional and interpretable strategies in a board-game setting, while also revealing important characteristics of the approach. In particular, Grammatical Evolution tends to favour simple, interpretable solutions and is highly sensitive to the design of the evaluation environment, which can limit its ability to produce robust, high-level gameplay. These findings highlight both the strengths and limitations of the approach in adversarial domains.

8.2 Future Work

Future work could extend the project in several directions.

A more comprehensive evaluation against stronger opponents would provide a clearer assessment of strategic strength. This could include larger-scale testing against minimax agents with higher search depth or against more sophisticated baseline strategies.

The human evaluation could also be expanded, both in terms of the number of participants and the range of strategies tested, to provide a broader assessment of perceived difficulty and gameplay quality.

Finally, further analysis of evolved strategies could be carried out to better understand how effective behaviour emerges. This could include identifying common structural patterns in successful strategies and examining how these relate to known concepts in draughts gameplay.

9 Appendices

9.1 Appendix A: Complete List of Evolutionary Experiments

This appendix lists every main evolutionary run used in the results chapters. Training win rate is derived from the best individual's fitness at the end of evolution (proportion of games won in the training evaluation). Standardised test win rate, 95% confidence interval, train–test gap, and variance across seeds come from the fixed multi-seed protocol in Chapter 5 (five seeds, 30 games per seed, 150 games total per strategy), produced by reevaluate_strategies_match_evolution.py. All 8×8 runs in Table A.2 use grammar file draughts_8x8.bnf, population 400, 75 generations, 20 games per evaluation, five co-evolution opponents, tournament selection, and complexity penalty 0.2 unless noted in the Condition column. Co-evolution ratio is stated as co-evolved proportion / random opponent proportion.

Table A.1 - 6x6 Evolutionary Runs:

ID	Result file	Configuration	Pop.	Gens	Games / eval	Mutation	Co-ev. ratio	Train WR	Test WR	95% CI (test)	Train–test gap	Var (seeds)
6C-1	evolution_20251213_034131.txt	6×6 classic	100	10	10	0.05	Random only	70.0%	43.3%	35.4%–51.3%	26.7%	0.112
6C-2	evolution_20251213_055102.txt	6×6 classic	200	100	20	0.05	Random only	72.0%	53.3%	45.4%–61.3%	19.2%	0.070
6C-3	evolution_20251213_083718.txt	6×6 classic	200	100	20	0.05	Random only	68.0%	47.3%	39.3%–55.3%	20.2%	0.093
6I-1	evolution_20251214_220441.txt	6×6 improved	150	30	15	0.10	60% / 40%	80.0%	32.0%	24.5%–39.5%	48.0%	0.098
6I-2	evolution_20251215_032724.txt	6×6 improved	300	50	25	0.10	60% / 40%	100.0%	84.7%	78.9%–90.4%	15.3%	0.075
6I-3	evolution_20251215_090842.txt	6×6 improved	300	50	25	0.10	60% / 40%	100.0%	34.0%	26.4%–41.6%	66.0%	0.049
6I-4	evolution_20251215_202804.txt	6×6 improved	300	50	25	0.10	60% / 40%	77.0%	41.3%	33.5%–49.2%	35.7%	0.091
6I-5	evolution_20251216_034713.txt	6×6 improved	300	50	25	0.10	60% / 40%	82.0%	81.3%	75.1%–87.6%	0.7%	0.034

Standardised test seeds for 6x6: 999-1003.

Table A.2 - 8x8 Main Evolutionary Runs:

ID	Result file	Condition	Evo. seed	Mut.	Co-ev. / rnd	Train WR	Test WR	95% CI (test)	Train–test gap	Var (seeds)
8-01	evolution_8x8_20251216_222007.txt	Config_5 (pilot)	—	0.10	60% / 40%	85.0%	44.7%	36.7%–52.6%	40.3%	0.050
8-02	evolution_8x8_20251218_002100.txt	Baseline Config_6	123	0.15	40% / 60%	82.0%	72.7%	65.5%–79.8%	9.8%	0.068
8-03	evolution_8x8_20260216_052949.txt	Baseline rep. 1	456	0.15	40% / 60%	88.0%	78.0%	71.4%–84.6%	9.5%	0.034
8-04	evolution_8x8_20260218_003143.txt	Baseline rep. 2	789	0.15	40% / 60%	93.0%	82.7%	76.6%–88.7%	9.8%	0.025
8-05	evolution_8x8_20260219_203646.txt	Baseline rep. 3	1011	0.15	40% / 60%	95.0%	94.7%	91.1%–98.3%	0.3%	0.034
8-06	evolution_8x8_20260223_045614.txt	Cond. A (30% co-ev.)	2001	0.15	30% / 70%	93.0%	76.0%	69.2%–82.8%	16.5%	0.061
8-07	evolution_8x8_20260225_114105.txt	Cond. A rep. 2	2002	0.15	30% / 70%	90.0%	69.3%	62.0%–76.7%	20.7%	0.013
8-08	evolution_8x8_20260227_011809.txt	Cond. A rep. 3	2003	0.15	30% / 70%	90.0%	66.7%	59.1%–74.2%	23.3%	0.063
8-09	evolution_8x8_20260302_115155.txt	Cond. B (50% co-ev.)	2011	0.15	50% / 50%	93.0%	72.0%	64.8%–79.2%	20.5%	0.027
8-10	evolution_8x8_20260307_093010.txt	Cond. B rep. 2	2012	0.15	50% / 50%	95.0%	73.3%	66.3%–80.4%	21.7%	0.092
8-11	evolution_8x8_20260309_155326.txt	Cond. B rep. 3	2013	0.15	50% / 50%	95.0%	81.3%	75.1%–87.6%	13.7%	0.062
8-12	evolution_8x8_20260311_043951.txt	Cond. C (mut. 0.10)	3011	0.10	40% / 60%	95.0%	78.0%	71.4%–84.6%	17.0%	0.050
8-13	evolution_8x8_20260312_214315.txt	Cond. C rep. 2	3012	0.10	40% / 60%	93.0%	70.0%	62.7%–77.3%	22.5%	0.087
8-14	evolution_8x8_20260316_010459.txt	Cond. C rep. 3	3013	0.10	40% / 60%	97.0%	63.3%	55.6%–71.1%	34.2%	0.092
8-15	evolution_8x8_20260318_051106.txt	Cond. D (mut. 0.20)	3021	0.20	40% / 60%	90.0%	78.0%	71.4%–84.6%	12.0%	0.034
8-16	evolution_8x8_20260320_104123.txt	Cond. D rep. 2	3022	0.20	40% / 60%	93.0%	81.3%	75.1%–87.6%	11.2%	0.050
8-17	evolution_8x8_20260322_094426.txt	Cond. D rep. 3	3023	0.20	40% / 60%	90.0%	68.7%	61.2%–76.1%	21.3%	0.072

Standardised test seeds for 8x8: 888-892. Maximum moves per game: 200.

9.2 Appendix B: 8x8 BNF Grammar

Strategies are evolved as symbolic expressions mapped from integer genomes using Grammatical Evolution. The grammar below defines the 8×8 position-evaluation representation: the phenotype is a numeric expression over the feature vector x, evaluated on successor board states to choose a move (Chapter 3-4). Operators are implemented in draughts_functions.py (add, sub, mul, pdiv, max_, min_, neg, abs_, if_, comparisons). Features x[0]-x[74] index the flattened board (64 cells) plus eleven derived features for the 8×8 board. The 6×6 experiments used the same operator and constant sets with features x[0]–x[46] in grammar file grammars/draughts2.bnf

BNF Grammar - draughts_8x8.bnf:

```
<position_eval> ::= <eval>
<eval> ::= <op> | <feature> | <constant> | if_(<condition>, <eval>, <eval>)
<condition> ::= greater_than_or_equal(<eval>, <eval>) |
less_than_or_equal(<eval>, <eval>)
<op> ::= add(<eval>, <eval>) | sub(<eval>, <eval>) | mul(<eval>, <eval>) |
pdiv(<eval>, <eval>) | max_(<eval>, <eval>) | min_(<eval>, <eval>) |
neg(<eval>) | abs_(<eval>)
<feature> ::= x[0] | x[1] | x[2] | x[3] | x[4] | x[5] | x[6] | x[7] | x[8]
| x[9] | x[10] | x[11] | x[12] | x[13] | x[14] | x[15] | x[16] | x[17] |
x[18] | x[19] | x[20] | x[21] | x[22] | x[23] | x[24] | x[25] | x[26] |
x[27] | x[28] | x[29] | x[30] | x[31] | x[32] | x[33] | x[34] | x[35] |
x[36] | x[37] | x[38] | x[39] | x[40] | x[41] | x[42] | x[43] | x[44] |
x[45] | x[46] | x[47] | x[48] | x[49] | x[50] | x[51] | x[52] | x[53] |
x[54] | x[55] | x[56] | x[57] | x[58] | x[59] | x[60] | x[61] | x[62] |
x[63] | x[64] | x[65] | x[66] | x[67] | x[68] | x[69] | x[70] | x[71] |
x[72] | x[73] | x[74]
<constant> ::= -10.0 | -5.0 | -2.0 | -1.0 | -0.5 | -0.1 | 0.0 | 0.1 | 0.5 |
1.0 | 2.0 | 5.0 | 10.0
```

9.3 Appendix C: Example Evolved Phenotypes

The following expressions are example evolved strategies included to illustrate the form of GE outputs: one high-performing 8×8 strategy and one 8×8 strategy that collapsed to a trivial constant despite competitive training fitness. They are not an exhaustive catalogue; full phenotypes for all runs are stored in the project's result files.

Longer or more nested expressions appear in other result files; the examples below span simple constants, unary/binary operators, and feature references typical of the search space.

Example C.1 - Strong 8×8 strategy (evolution_8x8_20260219_203646.txt):

Context: Baseline Config_6, replicate 3; evolution seed 1011. Standardised test: 142/150 wins (94.7%). Phenotype depth 6.

Phenotype: mul(x[73], x[24])

Like all evolved strategies here, this string is the mapped phenotype from Grammatical Evolution: a valid expression in the grammar, evaluated as Python after x and the operator functions are supplied. It is not hand-designed; it is a product of two sub-expressions, x[73] and x[24], combined with mul.

The vector x comes from DraughtsBoard.get_board_features() in draughts_game.py: x[0]–x[63] are the flattened board and x[64]–x[74] are eleven derived features in fixed order - piece and king counts, centre and back-row counts, mobility for each side, and side to move. In that layout, x[73] is mobility for player -1 (the second player). x[24] is the board cell at flat index 24 (one square's encoding), i.e. a local piece/empty value, not a global count.

When in play, for each legal move the engine builds the successor board, recomputes x, and evaluates the phenotype so the score is x[73] * x[24] on that successor. Because both inputs change with the position, the strategy does distinguish positions: it prefers moves whose

resulting states yield a higher product. The exact strategic “meaning” is not prescribed; evolution has coupled opponent mobility with one board square’s occupancy into a single scalar score. Move choice is still argmax over successor evaluations.

In plain language this is a non-trivial, position-dependent evaluator: unlike a constant, it uses real features of the successor state so different moves can receive different scores.

Example C.2 - Trivial 8×8 phenotype (evolution_8x8_20260225_114105.txt):

Context: Condition A (30% co-evolution), replicate 2; evolution seed 2002. Standardised test: 104/150 wins (69.3%). Phenotype depth 4.

Phenotype: 1.0

Under Best Individual in that result file, the phenotype is the literal expression 1.0 (a floating-point constant). That string is what Grammatical Evolution mapped from the genome.

In evolve_draughts_8x8.py, the phenotype is a position evaluation: for each legal move, the code applies the move, builds the successor board, and evaluates the phenotype, supplying the feature vector x. For the phenotype 1.0, eval("1.0", ...) always returns 1.0, regardless of board state or features, so every successor position receives the same score.

The move-selection logic keeps the move with the highest score seen so far. After the first legal move scores 1.0, every later move also scores 1.0; since 1.0 > 1.0 is false, the best index is never updated. The strategy therefore always plays valid_moves[0] which is the first legal move in whatever order the engine returns them.

In plain language 1.0 is a trivial constant evaluator: it does not distinguish positions; in effect it implements always play the first legal move. That is consistent with the log: very low codon use (2/70), depth 4, and a fitness objective that trades complexity against play. The run can still reach roughly 69% wins against the random opponent on the held-out test because a fixed move ordering can remain awkward for a random player to exploit. This shows us that competitive training fitness does not guarantee a non-degenerate expression under the grammar; this run illustrates a collapse to a constant.

10 References

Brabazon, A. and O'Neill, M., 2006. *Biologically inspired algorithms for financial modelling*. Berlin: Springer.

Browne, C.B. *et al*., 2012. 'A survey of Monte Carlo tree search methods'. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), pp.1–43.

Chellapilla, K. and Fogel, D.B., 1999. 'Evolving neural networks to play checkers without relying on human expertise'. *IEEE Transactions on Neural Networks*, 10(6), pp.1382–1391.

DEAP Development Team, 2023. *DEAP: distributed evolutionary algorithms in Python.* Available at: https://deap.readthedocs.io (Accessed: 20 October 2025).

Grammatical Evolution in Python (GRAPE), n.d. *GRAPE repository*. Available at: https://github.com/bdsul/grape (Accessed: 14 October 2025).

Kendall, G. and Whitwell, G., 2001. 'An evolutionary approach for the tuning of a chess evaluation function using population dynamics'. In: *Proceedings of the 2001 Congress on Evolutionary Computation (CEC'01)*. Piscataway, NJ: IEEE, pp.995–1002.

Koza, J.R., 1992. *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

O'Neill, M. and Ryan, C., 2003. *Grammatical evolution: evolutionary automatic programming in an arbitrary language*. Boston: Kluwer Academic Publishers.

O'Neill, M., Brabazon, A., Nicolau, M., McGarraghy, S. and Keenan, P., 2010. *Grammar-based genetic programming*. Berlin: Springer.

Russell, S. and Norvig, P., 2021. *Artificial intelligence: a modern approach*. 4th edn. Harlow: Pearson.

Ryan, C., Collins, J.J. and O'Neill, M., 1998. 'Grammatical evolution: evolving programs for an arbitrary language'. In: Banzhaf, W., Poli, R., Schoenauer, M. and Schwefel, H.-P., eds.

Genetic Programming: First European Workshop, EuroGP'98. Lecture Notes in Computer Science, 1391. Berlin: Springer, pp.83–96.

Samuel, A.L., 1959. 'Some studies in machine learning using the game of checkers'. *IBM Journal of Research and Development*, 3(3), pp.211–229.

Schaeffer, J. *et al*., 2007. 'Checkers is solved'. *Science*, 317(5844), pp.1518–1522.

Tech With Tim, 2020. *Python checkers tutorial series*. Available at: https://www.youtube.com/c/TechWithTim (Accessed: 3 February 2026).