Turk. J. Math. Comput. Sci.
18(1)(2026) 220–247

DOI : 10.47000/tjmcs.1663275

Heuristic Optimization of a Tetris Bot Using Genetic Algorithms: An Adaptive Evolutionary Approach

Ercan Erkalkan

Marmara University, Vocational School of Technical Sciences, Istanbul, Turkey.

Received: 22-03-2025 • Accepted: 25-11-2025

Abstract. This study presents a Genetic Algorithm (GA)-based optimization framework for enhancing the heuristic evaluation function of a *Tetris* bot. The proposed approach combines offline evolutionary training with real-time dynamic weight adjustments to adapt the bot's strategy to evolving gameplay conditions. Key heuristic features—including hole minimization, line clearing, and surface bumpiness—are weighted dynamically based on board state metrics such as maximum column height. Evaluated over 100 independent simulations, the GA-optimized bot improved over the fixed-weight baseline by **+61.79% lines cleared** (56.48→91.38) and **+55.39% average duration** (2.04→3.17 min), with higher **average** score (5,648→9,138) and a small latency increase of **+0.28 ms (+3.97%)** (7.05→7.33 ms). While decision latency increased marginally (7.33 ms vs. 7.05 ms), this trade-off was justified by the bot's enhanced strategic adaptability, evidenced by reduced performance variance and outlier frequency. The results validate GA's efficacy in optimizing complex, multi-objective decision-making processes in dynamic environments. Unlike earlier "adaptive evaluation" schemes for Tetris (e.g., [26]) that primarily adapt across episodes or generations, the proposed framework couples offline GA with *within-game, per-move* state-dependent reweighting based on normalized maximum column height (Eq. 4.9), and a lightweight in-game micro-GA (population $M' = 20$, generations $G' = 5$, scaling $\beta = 0.3$; see Table 4), achieving these gains with negligible latency overhead. Future work will explore hybrid GA-reinforcement learning architectures and applications to other real-time strategy games. **Limitation.** The scope is limited to GA; empirical comparisons with PSO/WOA/DE and RL-based tuners are designated as future work. A small-scale offline study under identical budgets indicated GA > WOA (significant) and GA ≈ PSO (not significant).

2020 AMS Classification: 68T20, 68T05, 90C59

Keywords: Tetris bot, genetic algorithms, heuristic optimization, dynamic weighting, evolutionary computation.

1. Introduction

Despite its simple gameplay, *Tetris* presents one of the most challenging environments for artificial intelligence (AI) to learn from scratch. Considering the combined challenges of stochastic input generation, spatial optimization, and real-time decision-making, *Tetris* remains a demanding benchmark in contemporary computational research in games. Primarily, *Tetris* offers a rich experimental platform to evaluate a wide array of optimization algorithms, ranging from academic studies to AI-based competition frameworks [10,26,28]. The inherent complexity of the game arises from its multi-objective nature, which closely reflects real-world optimization problems found in domains such as scheduling and logistics: maximizing the number of cleared lines, minimizing wasted space, and avoiding poor positioning for future pieces.

Email address: ercan.erkalkan@marmara.edu.tr (E. Erkalkan)

Recent advancements have investigated how evolutionary algorithms and machine learning techniques can address the computational complexities of *Tetris*. For instance, Hendrawan *et al.* evaluated the effectiveness of the Most Valuable Player Algorithm (MVPA) and Genetic Algorithms (GA) in optimizing the state space of *Tetris*. Their study highlighted the potential of the game as a valuable testbed for advanced artificial intelligence methods. The results demonstrated that MVPA outperformed GA by offering faster training times and achieving higher optimization scores, thus emphasizing its efficiency and practicality in tackling the inherent challenges of *Tetris* [5].

Moreover, due to the randomness of Tetris block sequences, AI agents must adopt robust planning strategies that balance short-term gains with long-term survival. The problem can be likened to real-world applications where uncertainty and optimization intersect, such as inventory management and dynamic resource allocation. By fostering adaptability and resilience, hybrid methods that combine evolutionary processes with reinforcement learning have significantly enhanced the capacity of AI systems to learn from Tetris-like environments [40]. These approaches typically consider features such as line clears, the number of holes, and column heights. However, manual parameter optimization is often time-consuming, labor-intensive, and susceptible to human bias. This may result in suboptimal outcomes or the inability to achieve desired performance levels [17, 25].

Genetic Algorithms (GAs), inspired by Darwinian evolution, offer a powerful solution by automating the optimization of such parameters. By mimicking natural selection through crossover, mutation, and fitness-based reproduction, GAs excel at exploring vast solution spaces and identifying near-optimal configurations. They have been applied in various domains such as procedural content generation in games and adaptive non-player character (NPC) behavior, demonstrating their adaptability and effectiveness in dynamic environments [4, 9, 19].

In recent years, researchers have effectively employed GAs to enhance AI performance in games such as *Flappy Bird*, *2048 [3]*, and *Tetris*. For instance, GA-based adaptive techniques applied to *Tetris* have demonstrated significant improvements in long-term planning and decision-making [11]. Evolutionary and automated reinforcement learning frameworks in other games have also revealed the potential of integrating search- and learning-based methods for multi-objective optimization and human-like AI behavior [25].

Nonetheless, many GA-based Tetris approaches either optimize static heuristic weights or evolve fixed action sequences without in-game adaptation [11, 26, 28], which limits adaptability to dynamic game conditions. This study proposes a GA-based framework for optimizing the heuristic evaluation function of a *Tetris*-playing bot. The proposed approach combines offline weight optimization (cf. [11, 28]) with real-time adjustments to accommodate stochastic block sequences and velocity changes. Iterative parameter refinement over multiple generations is employed to enhance performance and provide insights into the broader application of evolutionary techniques in game AI. This work explores the intersection of evolutionary algorithms and AI-driven game strategies while advancing heuristic optimization techniques.

Novelty over Prior Adaptive Schemes. Prior GA approaches to Tetris often relied on static weights or adapted *between* games/generations (*episode-level* adaptation) [26]. In contrast, the contributions are threefold: (i) a **hybrid workflow** that combines offline GA to learn a robust baseline vector with **in-game** adjustments; (ii) **continuous (per-move) reweighting** that ties heuristic weights to a normalized risk signal—the maximum column height—via Eq. (4.9); and (iii) a **micro-GA** for fast online refinement (population $M' = 20$, $G' = 5$; Table 4) together with a closed-form risk sensitivity ($\beta = 0.3$). Operationally, this means the policy adapts *within a single game and at each decision*, rather than only across episodes, while keeping decision latency near constant.

Generalizability. The offline–online GA framework with state-contingent reweighting naturally transfers to real-time, multi-objective decision making beyond games. In adaptive resource allocation (e.g., edge/cloud or wireless), heuristic features map to utilization, queueing delay, and energy, while weights are re-scaled by congestion or SLA slack; the micro-GA searches local policies under tight time budgets. In online scheduling (job-/flow-shop, VM placement), the analogues of *holes* and *bumpiness* are idle time and setup changeovers; per-interval reweighting by work-in-process or lateness stabilizes throughput. In dynamic bin-packing and inventory control, height-like signals (fill levels, stock-out risk) drive weights to prevent fragmentation and shortages. For traffic-signal control, state-dependent weights trade off queue clearance versus platoon progression as saturation rises; the micro-GA proposes phase splits/offsets within each cycle budget. In robotic packing/placement, features correspond to stability, reachability, and occlusion risk, with height acting as a safety margin. These correspondences suggest that anytime, budgeted search with depth-limited lookahead and beam pruning is a reusable pattern for latency-bounded, safety-aware controllers across domains.

The remainder of this paper is structured as follows:

- **Section 2 – Related Work:** Surveys the range of strategies for Tetris AI optimization, including traditional heuristic methods, evolutionary approaches, and more recent innovations in procedural content generation. This section also highlights how existing literature has tackled the real-time, stochastic nature of Tetris.
- **Section 3 – Tetris Bot Design and Heuristic Optimization:** Describes the architecture of the Tetris-playing bot, *SmartPlayer*, detailing its heuristic evaluation function and the core features it measures—such as holes, bumpiness, and line clearing. The section also covers the scoring logic, recursive lookahead strategy, and design decisions focused on adaptability and computational efficiency.
- **Section 4 – Application of Genetic Algorithm for Dynamic Weight Optimization in Tetris:** Explains how Genetic Algorithms (GAs) are utilized to optimize the bot's evaluation weights over time. It introduces the chromosome representation, fitness function, genetic operators, and the two-stage optimization workflow that combines offline GA training with real-time adaptive adjustments during gameplay.
- **Section 5 – Experimental Results and Analysis:** Presents the empirical setup and metrics used to evaluate both the baseline (fixed-weight) and GA-optimized bots. Performance indicators include lines cleared, average score, game duration, and decision latency. Sensitivity analyses are conducted on various GA configurations, and a comparative evaluation highlights the effectiveness of the proposed approach.
- **Section 6 – Conclusion:** Summarizes key findings and draws attention to future opportunities, including potential hybrid integrations of GAs and reinforcement learning, as well as applying the proposed framework to other dynamic optimization scenarios beyond Tetris.

2. Related Work

Research on Tetris bots and game AI optimization spans a wide range of methodologies, from manually crafted heuristic approaches to advanced machine learning and evolutionary strategies. These techniques aim to overcome the inherent challenges of game environments, such as real-time constraints, high-dimensional decision spaces, and unpredictability. Modern artificial intelligence methods are particularly well-suited for games like *Tetris*, where computational efficiency and strategic depth are critical, due to their flexibility and scalability.

2.1. Optimization in Game Artificial Intelligence.

Research in AI-driven game development has focused on AI optimization, employing sophisticated algorithms to enhance performance, adaptability, and decision-making processes. Genetic Algorithms (GAs), inspired by the principles of natural selection, have emerged as a widely adopted approach. Through iterative improvements via selection, crossover, and mutation, GAs have demonstrated remarkable capabilities in exploring complex solution spaces, thereby facilitating the optimization of game strategies [18].

Recent advancements have further emphasized the application of Genetic Algorithms (GAs) in formulating complex game strategies. For example, Konak and Kulturel-Konak (2022) proposed the Regret-Based Nash Equilibrium Sorting Genetic Algorithm (RNESGA), which uses a regret-based fitness assignment strategy in a multi-population genetic algorithm to find Nash equilibria in non-cooperative simultaneous combinatorial game theory problems with multiple players [21]. Similarly, Hendrawan et al. (2021) [3] illustrated the use of evolutionary algorithms such as GAs to optimize neural networks for playing the game *2048*, highlighting the adaptability and effectiveness of GAs in tuning game parameters.

Moreover, GAs have played a critical role in enhancing the gameplay of strategic and real-time games. For instance, Wang et al. (2019) employed an improved genetic algorithm to optimize build orders in the game *StarCraft*, using replay data as input [38]. Similarly, Pujianto (2021) explored the use of GAs in car racing games to improve game design and dynamically adjust difficulty levels [1].

In addition, Vani et al. [36] demonstrate in a CBIR survey that integrating genetic algorithms with neuro-fuzzy systems can improve decision-making quality. Similarly, the work titled Complexity and Mission Computability of Adaptive Computing Systems examines mission-time constraints and complexity in adaptive computing systems, highlighting how approximate and heuristic methods are required under strict resource limits [12].

Collectively, these studies demonstrate the flexibility of Genetic Algorithms (GAs) in adapting to various gaming contexts, such as turn-based strategy and real-time action games, solidifying their role as a critical component in advancing game AI. By balancing exploration and exploitation, GAs enable AI agents to navigate complex decision spaces, making them highly effective for developing competitive and engaging gameplay strategies.

Finally, advanced methods such as reward-based optimization, as presented in the study Optimizing Learned Networking Rate Adaptation via Guided Reward Reweighting, show how learning-based optimization methods (beyond GAs) can flexibly steer decision-making processes across diverse domains, including computer networking [39].

2.2. **Traditional Tetris Bot Techniques.** Traditionally, Tetris AI bots rely on heuristic or rule-based evaluation procedures to guide their decisions. These evaluation functions typically consider several factors, including:

- **Minimizing Board Height:** Preventing the stack from reaching the top of the board, which would end the game.
- **Line Clearing:** Prioritizing moves that complete lines in order to maximize score.
- **Reducing Holes:** Minimizing empty spaces beneath placed blocks, which are difficult to fill later.
- **Avoiding Overhangs:** Preventing structures that leave unfillable gaps in the board.

Conventional approaches often follow fixed scoring schemes, such as those used in NES Tetris heuristics or Pierre Dellacherie's well-known Tetris-playing AI. While these methods can be effective, they tend to lack flexibility and often require extensive manual tuning to achieve optimal performance. [6]

2.3. **Evolutionary Techniques in Non-Player Character Behavior.** Genetic Algorithms (GAs) have also proven valuable in adapting the behavior of non-player characters (NPCs) across various game genres, including survival shooter games and predator-prey dynamics. These evolutionary techniques enable NPCs to dynamically develop strategies and respond to player actions in ways that increase both game difficulty and engagement.

For instance, Armanto *et al.* (2025) provide a systematic review showing how evolutionary algorithms, including genetic algorithms, are widely used to optimize NPC behavior across various game genres (including survival shooters), making enemy behavior more adaptive and engaging [4]. Similarly, Jones *et al.* (2023) applied genetic programming to adaptive predator-prey dynamics, allowing NPCs to learn behaviors that effectively respond to changing environments and sustain a competitive gaming experience [19].

Although *Tetris* does not feature non-player characters (NPCs), the adaptability mechanisms observed in NPC evolution frameworks offer valuable inspiration for enhancing bot heuristics. In *Tetris*, unpredictability arises from the random nature of incoming block sequences, requiring bots to dynamically adjust their strategies to maintain performance across diverse scenarios.

By leveraging principles from NPC evolution, Genetic Algorithms (GAs) can be effectively applied to refine heuristic weights and decision policies that enable bots to respond to variations in block sequences and game states. This approach fosters resilience in bot behavior and ensures consistent optimization even in challenging and unforeseen situations. Such adaptability reflects the core principles of NPC evolution and underscores the utility of GAs as a unifying method for advancing game AI across various genres.

2.4. **Procedural Content Generation and Genetic Algorithms.** Procedural Content Generation (PCG) via Genetic Algorithms (GAs) has advanced significantly, with a focus on creating challenging and engaging game levels. This approach minimizes human intervention in level design and is particularly relevant to games like *Tetris*, where "level" difficulty may be defined in terms of block sequences or achieving specific objectives [9]. Recent innovations in PCG have further explored the potential of GAs in enhancing game design.

The Procedural Content Generation (PCG) literature indicates that a wide variety of algorithms are employed for the automatic generation of game content. Hafis *et al.* [16] systematically reviewed game-oriented PCG studies from the last five years and reported that no single "best" method exists; instead, different approaches are adopted depending on the type of content being generated. Similarly, Tao *et al.* [33] applied innovative GA operations such as selection, crossover, and mutation to optimize game search trees and move selection, demonstrating the effectiveness of GA-based search in computer games.

Another recent line of work investigates human-in-the-loop PCG pipelines for commercial games. Kruse *et al.* [22] evaluate a multi-agent level design system in which autonomous computational agents propose map variants that are iteratively refined by professional designers. Eye-tracking and interview data indicate a clear visual preference for layouts suggested by the computational agents, although expert designers often remain skeptical about the usefulness of such tools. This evidence suggests that evolutionary and agent-based PCG is particularly effective as a decision-support mechanism rather than a full replacement for human designers.

These developments emphasize the adaptability of GAs not only for procedurally generated levels but also for creating dynamic gameplay experiences. By integrating such methods into *Tetris*, developers can produce increasingly

challenging block sequences that follow predefined difficulty curves while maintaining player engagement. This approach highlights the ongoing relevance of PCG techniques powered by GAs in crafting both engaging and technically optimized gaming experiences.

2.5. **Comparative Studies of Evolutionary Techniques in Games.** Comparative studies have underscored the strengths and weaknesses of various optimization algorithms in gaming applications, emphasizing their suitability for different types of problems. Genetic Algorithms (GAs), with their population-based search and robust exploratory capabilities, are particularly effective at navigating diverse solution spaces and identifying near-optimal configurations. This makes them especially suitable for complex games such as *Tetris*, where the decision space is large and dynamic. GAs are notably advantageous in balancing exploration and exploitation, allowing AI agents to avoid local optima while discovering high-quality solutions [24].

Alternative methods such as Particle Swarm Optimization (PSO) and Whale Optimization Algorithm (WOA) also offer unique strengths in addressing optimization challenges. Inspired by the social behavior of birds and fish, PSO often achieves faster convergence in well-defined combinatorial optimization problems. For instance, PSO combined with an appropriate permutation encoding and local search has shown competitive or superior performance to genetic algorithms on single-machine total weighted tardiness benchmarks [35].

Similarly, WOA, which simulates the hunting behavior of humpback whales, exhibits strong local search capabilities, making it particularly effective in problems requiring fine-tuned adjustments. Recent applications in game AI highlight its efficiency in solving constrained optimization problems and enhancing heuristic parameters in simpler gaming scenarios [13].

Despite these advances, GAs remain the preferred choice for games like *Tetris* due to their adaptability to stochastic inputs and dynamic environments, which are inherent characteristics of such games. The unpredictable nature of the game—driven by randomly generated block sequences—requires an algorithm capable of maintaining robust adaptability while optimizing multiple objectives. GAs' iterative improvement mechanisms and diversity preservation strategies make them well-suited for this challenge. Leveraging crossover, mutation, and selection processes, GAs enable continuous refinement across generations, solidifying their status as a preferred approach for games with complex decision landscapes like *Tetris*.

Summary. Prior adaptive Tetris frameworks (e.g., [26]) introduced adaptation at episode- or generation-level granularity. The proposed design advances this direction by (i) coupling offline genetic algorithms with online control, (ii) enabling per-move weight adjustments via a normalized height-based proxy (Eq. 4.9), and (iii) employing a bounded-latency micro-GA architecture (Table 4). The effect of each component is quantified through ablation experiments (§5).

Table 1. Offline optimization (static play; *no in-game reweighting*): GA/PSO/WOA comparison. Mean ± SD over 100 games.

Method	Lines Cleared (avg)	Score (avg)	Game Duration (min, avg)	Decision Latency (ms, avg)
GA	78.0 ± 45.0	7,800 ± 4,500	2.80 ± 0.18	7.10 ± 0.20
PSO	67.5 ± 45.0	6,750 ± 4,500	2.45 ± 0.17	7.07 ± 0.20
WOA	62.0 ± 44.0	6,200 ± 4,400	2.25 ± 0.16	7.06 ± 0.20

Statistical comparison (offline). Welch two-sample t-tests on "Lines" yielded: GA vs. PSO, t=1.65, $df \approx 196$, p=0.100 (not significant), 95% CI for mean difference [−2.05, 23.05] lines; GA vs. WOA, t=2.51, $df \approx 198$, p=0.012 (significant), 95% CI [3.59, 28.41] lines. Results indicate a significant advantage of GA over WOA under identical budgets, with GA and PSO exhibiting comparable performance.

3. Tetris Bot Design and Heuristic Optimization

To effectively transition from the theoretical framework of AI optimization in *Tetris* to a concrete implementation, it is critical to examine how abstract strategies are operationalized in practice. The *SmartPlayer* bot serves as a practical embodiment of these concepts, combining heuristic evaluation with optimization methodologies to meet the complex demands of Tetris gameplay [26]. By leveraging meticulously defined evaluation metrics and sophisticated decision-making algorithms, the bot demonstrates how to skillfully manage the balance between short-term gains and sustainable

game longevity. The following section explores the design and operational mechanics of the *SmartPlayer* bot, offering insights into the techniques it employs to address the diverse challenges inherent to *Tetris*.

3.1. **Overview of the *SmartPlayer* Bot.** The *SmartPlayer* bot is designed to achieve high performance in *Tetris* by combining heuristic evaluation with simulation-based optimization techniques [10]. Its operation systematically scores potential moves using a weighted heuristic function and selects the action that maximizes this score. The bot's architecture integrates the following key components:

- **Heuristic Evaluation:** The bot employs a comprehensive scoring system to evaluate the state of the game board after each potential move. This evaluation function considers various board features such as the number of holes, column heights, and cleared lines. These features collectively guide the bot to identify moves that improve board efficiency and reduce future risks.
- **Lookahead Depth:** To strengthen decision-making, the bot simulates multiple future moves and anticipates subsequent game states. This lookahead mechanism enables the *SmartPlayer* to prioritize long-term strategies over short-term gains, balancing immediate rewards with sustainable playability.
- **Action Queue:** Once the optimal sequence of actions is determined, the bot stores these actions in a queue. This ensures smooth and consistent gameplay, as actions are executed sequentially without delay or interruption [35].

Together, these design elements allow the *SmartPlayer* bot to navigate the complexities of *Tetris* with precision and strategic foresight.

3.2. **Mathematical Model of Heuristic Evaluation.** The heuristic evaluation function of the bot is implemented as a weighted sum of key board features, offering a systematic framework for quantifying the desirability of a given game state [11]. This model ensures consistent assessment of possible moves and allows the bot to prioritize actions that optimize both immediate benefits and long-term survivability.

Definition 3.1 (Heuristic Evaluation Function)**.** The heuristic score S is calculated as follows:

$$S = \sum_{i=1}^{n} w_i \cdot f_i, \tag{3.1}$$

where:

- S denotes the overall heuristic score of the board state after executing a move. Higher values of S indicate more favorable game states.
- w_i represents the weight assigned to feature f_i, reflecting the relative importance of that feature in the decision-making process. For example, assigning a higher weight to *holes* penalizes board states with empty spaces beneath blocks more heavily.
- f_i is the value of the i-th feature extracted from the current board configuration. Common features include the number of holes, the maximum column height, and the number of lines cleared.
- n is the total number of features considered. A higher n increases model granularity but may also lead to greater computational overhead.

3.3. **Features and Their Formulations.** To effectively evaluate the quality of a given *Tetris* board state, a set of core features—each representing a specific aspect of the board configuration—is computed. These features are integrated into the heuristic evaluation function to guide the bot's decision-making process. An extended explanation of each feature is provided below, including mathematical formulation, gameplay role, and strategic significance.

Definition 3.2 (Holes)**.** The number of holes, denoted as f_{holes}, is defined as the total count of empty cells that have at least one filled cell above them in the same column. Formally, it is expressed as:

$$f_{\text{holes}} = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} H(x, y), \tag{3.2}$$

where the indicator function $H(x, y)$ is defined as:

$$H(x, y) = \begin{cases} 1 & \text{if cell } (x, y) \text{ is empty and there exists at least one filled cell above it in column } x, \\ 0 & \text{otherwise.} \end{cases}$$

Strategic Importance: Holes severely disrupt gameplay efficiency by obstructing line clears and creating unfillable gaps. By penalizing configurations with high hole counts, the bot is incentivized to prefer compact and structured placements. Minimizing holes contributes to more stable stacking and prolongs survivability in the game.

Definition 3.3 (Maximum Column Height)**.** The maximum column height, denoted as f_{height}, represents the tallest stack on the board and is formally defined as:

$$f_{\text{height}} = \max_{x=1}^{\text{width}} \left(\text{height}(x)\right), \tag{3.3}$$

where height(x) denotes the number of filled cells in column x, calculated as the distance from the bottom row to the highest occupied cell in that column. If no filled cells exist in column x, then height(x) = 0.

Strategic Importance: This feature serves as a primary indicator of the game's risk of ending. A high column increases the likelihood of reaching the board's upper limit, triggering game over. Moves that maintain a uniformly low profile across columns are often prioritized, as they help retain board control and extend gameplay.

Definition 3.4 (Bumpiness)**.** Bumpiness, denoted as $f_{\text{bumpiness}}$, quantifies the unevenness of the board surface by measuring the absolute differences in height between adjacent columns. It is formally defined as:

$$f_{\text{bumpiness}} = \sum_{x=1}^{\text{width}-1} \left|\text{height}(x) - \text{height}(x+1)\right|. \tag{3.4}$$

Strategic Importance: Bumpiness reduces the board's ability to accommodate incoming blocks efficiently. Uneven surfaces result in awkward placements and can create inaccessible spaces. Penalizing bumpiness encourages smoother surfaces, thereby improving the bot's ability to clear lines and position future pieces more effectively.

Definition 3.5 (Line Clearing)**.** Line Clearing, denoted as f_{lines}, represents the number of lines fully completed and removed from the board after executing a move. It is mathematically defined as:

$$f_{\text{lines}} = \sum_{y=1}^{H} \delta\left(\sum_{x=1}^{W} B(x,y) = W\right), \tag{3.5}$$

where:

- H is the height (number of rows) of the board.
- W is the width (number of columns) of the board.
- $B(x,y)$ is a binary function that returns 1 if the cell at column x and row y is filled, and 0 otherwise.
- $\delta(\cdot)$ is the indicator function, which equals 1 if the condition inside holds true, and 0 otherwise.

Strategic Importance: Clearing lines is the primary mechanism for scoring in *Tetris* and for reducing the overall number of blocks on the board. Rewarding moves that clear lines encourages the bot to favor configurations that lead to line completion. Moves resulting in multiple line clears, particularly a Tetris (clearing four lines simultaneously), are especially desirable as they yield higher scores and maintain board cleanliness.

Definition 3.6 (Rows with Holes)**.** The *Rows with Holes* feature, denoted as $f_{\text{rows\_with\_holes}}$, counts the number of rows that contain at least one hole. A hole is defined as an empty cell that has at least one filled cell above it in the same column. It is mathematically expressed as:

$$f_{\text{rows\_with\_holes}} = \sum_{y=1}^{H} \Theta\left(\sum_{x=1}^{W} H(x,y)\right), \tag{3.6}$$

where:

- H is the height of the board.
- W is the width of the board.
- $H(x,y)$ is a binary function returning 1 if cell (x,y) is a hole, and 0 otherwise.
- $\Theta(\cdot)$ is the Heaviside step function (or indicator function), returning 1 if the input is greater than 0, and 0 otherwise.

Strategic Importance: Rows containing holes hinder efficient line clearing. This metric captures the spread of empty cells across rows and complements the total holes metric. Penalizing rows with holes encourages the bot to minimize the dispersion of empty spaces and to focus on clearing complete rows rather than stacking blocks above holes.

Definition 3.7 (Adjacent Touching Blocks)**.** The *Adjacent Touching Blocks* feature, denoted as f_{touching}, quantifies the number of adjacent filled cells for each block placed on the board. For each filled cell (x, y), it examines its four orthogonal neighbors (left, right, top, bottom) and sums how many of them are also filled. The feature is mathematically defined as:

$$f_{\text{touching}} = \sum_{\text{blocks}} \text{adjacent\_filled}(x, y), \tag{3.7}$$

$$\text{adjacent\_filled}(x, y) = \delta(x+1, y) + \delta(x-1, y) + \delta(x, y+1) + \delta(x, y-1), \tag{3.8}$$

where:

- blocks refers to all occupied cells resulting from the current move.
- $\delta(x, y)$ is an indicator function defined as:

$$\delta(x, y) = \begin{cases} 1, & \text{if cell } (x, y) \text{ is occupied (filled)} \\ 0, & \text{otherwise.} \end{cases}$$

Strategic Importance: This feature rewards compact placements where blocks are tightly clustered together. High values of f_{touching} indicate that newly placed blocks are well-integrated into the existing structure. Encouraging such behavior reduces the likelihood of hole creation, promotes orderly stacking, and facilitates future placements. Ultimately, it contributes to long-term gameplay sustainability and efficient board management.

Definition 3.8 (Open Sides)**.** The *Open Sides* feature, denoted as $f_{\text{open\_sides}}$, quantifies the number of exposed edges around filled cells on the Tetris board. It evaluates the compactness of block placement by measuring how many sides of each occupied cell are adjacent to either empty cells or the board boundaries. The feature is defined as:

$$f_{\text{open\_sides}} = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} \Omega(x, y), \tag{3.9}$$

where:

$$\Omega(x, y) = \begin{cases} 4 - \text{adjacent\_filled}(x, y) \quad (\text{see Eq. 3.8}), & \text{if } (x, y) \text{ is a filled cell} \\ 0, & \text{if } (x, y) \text{ is an empty cell.} \end{cases}$$

Strategic Importance: This feature penalizes loosely stacked or isolated placements by assigning a higher score to moves that leave more exposed sides. A lower $f_{\text{open\_sides}}$ value indicates a tighter, more compact structure, which generally reduces the chance of hole formation and improves stacking efficiency. In contrast, higher values suggest risky configurations with increased potential for future holes and instability. Promoting compactness helps maintain gameplay control and prolongs the bot's survival.

Definition 3.9 (Next to Wall)**.** The *Next to Wall* feature, denoted as $f_{\text{next\_to\_wall}}$, measures the number of filled cells directly adjacent to the vertical boundaries (walls) of the Tetris board. Formally, it is expressed as:

$$f_{\text{next\_to\_wall}} = \sum_{y=1}^{H} \left[B(1, y) + B(W, y) \right], \tag{3.10}$$

where:

- H and W denote the height and width of the board, respectively.
- $B(x, y)$ is a binary function returning 1 if cell (x, y) is filled, and 0 otherwise.

Strategic Importance: Encouraging placements next to walls helps maintain a compact board structure, reducing undesirable gaps. Strategically positioning blocks adjacent to walls minimizes the complexity of subsequent placements and supports stable stacking patterns.

Definition 3.10 (Closed Sides)**.** The *Closed Sides* feature, denoted as $f_{\text{closed\_sides}}$, represents the total number of filled-cell edges adjacent to other filled cells or board boundaries. It quantifies compactness and structural integrity, calculated as:

$$f_{\text{closed\_sides}} = \sum_{x=1}^{W} \sum_{y=1}^{H} \Gamma(x, y), \tag{3.11}$$

where:

$$\Gamma(x, y) = \begin{cases} \text{adjacent\_filled}(x, y), & \text{if cell } (x, y) \text{ is filled} \\ 0, & \text{otherwise.} \end{cases}$$

The function adjacent_filled(x, y) is defined in Eq. (3.8).

Strategic Importance: High closed sides values indicate tight, stable formations with minimal isolated placements, promoting robustness and facilitating efficient subsequent piece placements. This feature incentivizes the bot to integrate new pieces seamlessly into existing structures.

Definition 3.11 (Column Filled)**.** The *Column Filled* feature, denoted as $f_{\text{column\_filled}}$, measures how many columns on the board are entirely filled without empty spaces, calculated as follows:

$$f_{\text{column\_filled}} = \sum_{x=1}^{W} \delta\left(\sum_{y=1}^{H} B(x, y) = H \right), \tag{3.12}$$

where $\delta(\cdot)$ is an indicator function returning 1 if the condition is true, and 0 otherwise.

Strategic Importance: Fully filled columns represent high vertical stacking efficiency. Rewarding completely filled columns encourages orderly, gapless stacking and promotes structurally sound gameplay, indirectly minimizing holes and facilitating future block placements.

Each feature contributes to the bot's evaluation of the board state and is weighted by a corresponding coefficient w_i according to its relative importance. This weighting mechanism enables the bot to balance short-term priorities, such as maximizing line clears, with long-term considerations like minimizing holes and surface irregularities. By integrating these features into a unified heuristic framework, the bot can make informed decisions aligned with its strategic objectives.

For each possible action, the bot simulates the resulting board state by applying the following operations:

- **Rotations:** The bot iteratively applies all valid rotations of the current falling block.
- **Horizontal Translations:** The bot evaluates the placement of the block across all feasible horizontal positions.
- **Drop:** The block is dropped vertically, and the resulting board configuration is computed.

For every resulting state, the bot calculates the heuristic score S using the weighted sum of features as described in Eq. (3.1). It then selects the action sequence that maximizes this score. This simulation-based approach ensures a comprehensive assessment of all candidate configurations, allowing the bot to prioritize moves that promote both immediate gains and long-term board sustainability.

3.4. **Scoring Dynamics.** The scoring function lies at the core of the *SmartPlayer* bot's decision-making process, allowing it to evaluate potential moves and select the optimal action based on a comprehensive assessment of the board state. This function integrates a set of weighted features $(w_i \cdot f_i)$ that quantify various aspects of the game board. The contribution of each feature is modulated by its corresponding weight (w_i), reflecting its relative importance with respect to the bot's strategic objectives.

The dynamic interaction between these weights enables the bot to adapt its playstyle to different scenarios—whether the goal is to pursue high scores, maximize survival time, or meet specific objectives in a customized *Tetris* environment. The specific methodology used for determining these weights, including optimization techniques such as Genetic Algorithms, is extensively discussed in Section 4.

3.5. **Recursive Evaluation and Strategic Depth.** To avoid shortsighted decisions, the bot evaluates sequences of moves using a recursive approach:

- Simulate the immediate next move.
- For each resulting board state, evaluate all possible subsequent moves.
- Combine scores across all lookahead steps to determine the best multi-step strategy.

Recursive methods allow the bot to assess sequences of moves across varying lookahead depths [23]. The recursive scoring function can be expressed as:

$$S_{\text{total}} = S_{\text{current}} + \alpha \cdot S_{\text{future}}, \tag{3.13}$$

where:

- S_{current} is the score of the immediate board state.
- S_{future} is the projected score of future board states.
- α is a discount factor used to prioritize immediate rewards over uncertain long-term outcomes.

3.6. **Advantages of the SmartPlayer Design.** The advantages of the SmartPlayer bot design can be summarized as follows:

- **Scalability:** Its modular design allows easy addition or modification of heuristic features.
- **Adaptability:** Optimal feature weighting and dynamic gameplay strategies are effectively determined via Genetic Algorithms, which are discussed in Section 4.
- **Efficiency:** The action queue and recursive evaluation mechanisms ensure smooth and responsive gameplay.

This mathematically grounded architecture positions the SmartPlayer bot as a robust solution for tackling the strategic challenges posed by *Tetris*. Furthermore, it lays the foundation for integrating advanced AI techniques such as reinforcement learning to further enhance its capabilities.

4. Application of Genetic Algorithm for Dynamic Weight Optimization in Tetris

This section details the application of Genetic Algorithms (GAs) for optimizing the heuristic evaluation function of a *Tetris* bot. The proposed implementation incorporates dynamic weight adjustments and real-time optimization, allowing the bot to adapt to evolving gameplay conditions. The integration of offline training with online adaptation demonstrates the effectiveness of GA-based heuristic optimization in enhancing decision-making and overall performance. Earlier work on Tetris agents already highlighted the potential of genetic optimization under strict computation-time constraints [2].

4.1. **Framework of the Genetic Algorithm Implementation.** Genetic Algorithms (GAs) are employed to optimize the heuristic feature weights in the bot's evaluation function, enabling efficient scoring and long-term gameplay management. The GA framework consists of four core components:

- **Chromosome Representation:** Defines the structure of candidate solutions (i.e., weight sets).
- **Fitness Evaluation:** Quantifies the performance of each candidate solution.
- **Genetic Operators:** Improves the population through selection, crossover, and mutation processes.
- **Termination Criteria:** Determines when the evolutionary process should conclude.

4.1.1. *Chromosome Representation.* Each chromosome represents a unique set of weights for the heuristic evaluation function. A chromosome C is defined as:

$$C = [w_{\text{holes}}, w_{\text{rows\_with\_holes}}, w_{\text{touching}}, w_{\text{height}}, w_{\text{open\_sides}}, w_{\text{next\_to\_wall}}, w_{\text{closed\_sides}}, w_{\text{bumpiness}}, w_{\text{column\_filled}}, w_{\text{lines}}], \tag{4.1}$$

where:

- w_{holes}: Weight assigned to the number of holes on the board. Holes hinder game flow and are penalized.
- $w_{\text{rows\_with\_holes}}$: Weight for the number of rows that contain at least one hole. This captures not just the total number of holes but also their spatial distribution.
- w_{touching}: Weight for the number of adjacent contacts between blocks. More touching blocks imply compact placement, reducing the risk of hole formation.
- w_{height}: Weight for the maximum column height. Tall columns increase the risk of game over and should be minimized.

- $w_{\text{open\_sides}}$: Weight for the number of exposed edges. Fewer open sides promote compact stacking.
- $w_{\text{next\_to\_wall}}$: Weight for filled cells directly adjacent to vertical walls, promoting compact board structure.
- $w_{\text{closed\_sides}}$: Weight for the number of filled-cell edges adjacent to other filled cells or board boundaries, indicating tight and stable block formations.
- $w_{\text{bumpiness}}$: Weight for the irregularity metric, which measures height differences between adjacent columns. A flatter surface facilitates efficient block placement.
- $w_{\text{column\_filled}}$: Weight rewarding fully filled columns, encouraging vertically efficient stacking.
- w_{lines}: Weight for the number of lines cleared in a move. Since line clearing is the main scoring mechanism in *Tetris*, it is rewarded.

Each gene w_i corresponds to the weight of a specific feature. To ensure population diversity and better exploration of the solution space, initial weights in the population are not fixed. Instead, they are initialized randomly around a baseline range using the following uniform distribution:

$$w_i \sim \mathcal{U}(w_i^{\text{base}} \cdot 0.9,\ w_i^{\text{base}} \cdot 1.1), \tag{4.2}$$

where w_i^{base} refers to the baseline value assigned to feature i, as shown in Table 2. This approach introduces controlled randomness ($\pm 10\%$) around baseline values and prevents premature convergence by maintaining genetic diversity in the initial population.

Table 2. Initial Weights Assigned to Each Feature

Feature	**Initial Weight**
holes	-5000
rows_with_holes	-700
touching	30
height	-400
open_sides	-500
next_to_wall	40
closed_sides	150
bumpiness	-150
column_filled	100
lines	250

The high penalty for holes reflects their disruptive nature in gameplay, while features such as line clearing and compactness receive positive weights. Randomized initialization around these baselines encourages a balanced trade-off between exploration and exploitation during early generations of the evolutionary process.

4.1.2. *Fitness Function.* The fitness of each chromosome is determined through simulations of *Tetris* games. For a given chromosome C, the fitness function $F(C)$ is defined as follows [5, 15, 27]:

$$F(C) = \frac{1}{100} \sum_{i=1}^{100} \text{score}(C, G_i), \tag{4.3}$$

where:

- $\text{score}(C, G_i)$ denotes the score obtained by the bot in game instance G_i using chromosome C.
- The number of games N is fixed as $N = 100$ across all experiments to ensure statistical robustness.

The choice of $N = 100$ simulations per chromosome is based on convergence analysis. Empirical studies have shown that after approximately 100 randomized Tetris simulations, the average fitness values stabilize, with minimal variance across trials. This approach enhances consistency and minimizes the effects of stochastic differences in gameplay sequences [27]. For instance, in their fuzzy logic-based Tetris player, Pickering and Cohen [27] observed that performance measures become more reliable after many repeated evaluations guided by genetic optimization.

To avoid overfitting to specific block sequences and improve generalizability, each fitness evaluation is performed using randomized game instances. The randomized simulation technique allows the chromosome to be tested across

diverse scenarios, enabling a more balanced estimation of its overall performance. Armanto *et al.* [5] emphasize that higher evaluation counts in genetic optimization of Tetris agents reduce convergence bias and provide more general solutions.

This randomized averaging method yields a robust, noise-tolerant fitness metric for guiding the evolutionary search process.

4.1.3. *Genetic Operators.* The evolutionary process improves the population over successive generations using selection, crossover, and mutation operators.

The selection phase in Genetic Algorithms (GAs) ensures that chromosomes with higher fitness scores are more likely to be chosen as parents for the next generation.In this study, two widely used selection strategies were analyzed and theoretically compared based on existing literature: **Roulette Wheel Selection** and **Tournament Selection**. These methods reflect different balances between exploration and exploitation in evolutionary search, and their relative effectiveness depends on problem complexity and population dynamics [7, 29, 31].

Roulette Wheel Selection

In this classic probabilistic method, each chromosome is assigned a selection probability proportional to its fitness value:

$$P(C_i) = \frac{F(C_i)}{\sum_{j=1}^{M} F(C_j)}, \tag{4.4}$$

where $F(C_i)$ denotes the fitness of chromosome C_i, and M is the total population size. This strategy promotes diversity by allowing even moderately fit individuals a non-zero chance of selection, which can be beneficial in avoiding premature convergence. However, it may also result in slower convergence, particularly in problems requiring more aggressive exploitation [29, 31].

Tournament Selection

Tournament selection involves randomly selecting a small group of chromosomes (usually 2 or 3) and choosing the one with the highest fitness. This method introduces strong selection pressure, encouraging rapid convergence by favoring the fittest individuals at each step. Nevertheless, excessive pressure can lead to a loss of genetic diversity and increased risk of premature convergence, especially if not combined with diversity-preserving mechanisms [7, 31].

Elitism. Elitism is a technique used to carry over the best-performing individuals from one generation to the next without modification. In this study, the top 2 chromosomes were directly preserved in each generation. This mechanism stabilizes the evolutionary process by preventing the loss of high-quality solutions. When integrated with roulette selection, elitism helps accelerate convergence while preserving necessary variation to avoid local optima [7, 29].

Comparison and Justification. This study analyzes selection operators through a literature review rather than experimental tests. While tournament selection is known for faster convergence [29], it may reduce population diversity and lead to premature convergence in complex problems. In contrast, roulette wheel selection promotes exploration, and when combined with elitism (as done in this study), it balances diversity with accelerated convergence by preserving top solutions [30, 31]. Stańczak [31] emphasizes that tournament selection requires careful parameter tuning to avoid stagnation, whereas roulette+elitism offers more robustness in unknown search spaces.

Crossover. Crossover generates new offspring by exchanging gene segments between selected parent chromosomes. Given two parents C_1 and C_2, the offspring O_1 and O_2 are generated using a two-point crossover mechanism:

$$O_1 = [C_1[1 : p_1],\ C_2[p_1 : p_2],\ C_1[p_2 : L]], \tag{4.5}$$

$$O_2 = [C_2[1 : p_1],\ C_1[p_1 : p_2],\ C_2[p_2 : L]], \tag{4.6}$$

where p_1 and p_2 are randomly selected crossover points satisfying $1 < p_1 < p_2 < L$ (L: chromosome length). This method enhances exploration by recombining genetic material while preserving contiguous gene blocks, a strategy empirically shown to balance diversity and convergence in combinatorial optimization [34].

Mutation. To preserve genetic diversity, Gaussian mutation is applied stochastically. Each gene w_i undergoes mutation with a probability of 5% per generation, perturbed by noise sampled from a normal distribution:

$$w_i' = w_i + \delta, \quad \delta \sim \mathcal{N}(0, \sigma). \tag{4.7}$$

The standard deviation σ is adaptively scaled to the gene's magnitude: $\sigma = 0.1 \cdot |w_i|$.

This proportional scaling allows larger-magnitude genes to tolerate broader exploration while constraining small weights to fine-tuned adjustments, a method validated in neuroevolution tasks [30, 34].
Termination Criteria. The Genetic Algorithm terminates when either condition is met:

- **Fitness Convergence**: The relative improvement in the best fitness value remains below $\epsilon = 0.01\%$ for 10 consecutive generations, calculated as:

$$\frac{|f_t - f_{t-1}|}{f_{t-1}} \times 100\% < \epsilon. \tag{4.8}$$

- **Generation Limit**: A predefined maximum generation count $G_{\max} = 100$ is reached, ensuring computational tractability for large-scale problems [17].

Implementation Note: The genetic algorithm exclusively implemented the **Roulette Wheel Selection with Elitism** strategy, coupled with two-point crossover and adaptive Gaussian mutation. This integrated approach prioritized:

(1) Diversity preservation through probabilistic parent selection (roulette wheel) and top-2 chromosome retention (elitism),
(2) Balanced exploration-exploitation via mutation rates scaled to gene magnitudes ($\sigma = 0.1 \cdot |w_i|$),
(3) Computational efficiency with termination thresholds ($\epsilon = 0.01\%$, $G_{\max} = 100$).

The mutation probability was fixed at 5% per gene, and all operators were applied only to chromosomes selected through the roulette-elitism framework. Tournament selection and alternative operator designs were deliberately excluded to maintain focus on optimizing a literature-backed configuration suitable for population sizes $M < 500$ [17, 29–31].

4.1.4. *Genetic Algorithm Parameters and Summary Table.* Population size $M = 100$ and mutation rate $\mu = 0.05$ were selected based on parameter sweeps showing optimal trade-offs between convergence speed and diversity [17].

Table 3 summarizes the implemented genetic algorithm configuration, rigorously aligned with the framework described in Sections 4.1.1–4.1.3.

Table 3. Implemented Genetic Algorithm Configuration

Parameter	**Value**	**Rationale & Implementation Details**
Population Size (M)	100	Matches the $M = 100$ constraint specified in Section 4.1.3. Optimized for computational efficiency while maintaining genetic diversity.
Max Generations ($G_{\max}$)	100	Termination criterion from Section 4.1.3. Ensures tractable runtime without overfitting.
Selection	Roulette Wheel + Top-2 Elitism	Combined strategy as specified in the implementation note. Elitism preserves the two best-performing chromosomes each generation.
Crossover Rate	0.8	Literature-supported value for combinatorial optimization [34]. Two-point crossover exclusively used.
Crossover Points	$1 < p_1 < p_2 < L$	Two-point crossover as defined explicitly in Section 4.1.3, where chromosome length $L = 10$.
Mutation Rate	5% per gene	Fixed probability aligned with Section 4.1.3. Mutation applied independently to each gene.
Mutation Scaling	$\sigma = 0.1 \cdot \|w_i\|$	Adaptive Gaussian noise magnitude consistent with Section 4.1.3. Validated by prior studies [30, 34].
Fitness Evaluations (N)	100 games	Consistent with Equation 4.3. 100 simulations per chromosome provide statistically stable fitness evaluation (±2% variability).
Convergence Threshold (ϵ)	0.01%	Relative fitness improvement threshold from Section 4.1.3.

Key consistencies with prior sections include:

- All parameter values explicitly match the specifications provided in the implementation notes.
- Crossover and mutation details align precisely with the definitions provided in Sections 4.1.1 and 4.1.3.
- Population size constraint ($M < 500$) explicitly maintained.
- Termination criteria ($G_{\max} = 100$, convergence threshold $\epsilon = 0.01\%$) directly consistent with framework specifications.

4.2. **Implementation and Extended Pseudocode.** This section integrates the previously discussed components of the *SmartPlayer* bot into a coherent workflow. The pseudocode in Algorithm 1 outlines how the bot employs Genetic Algorithms (GAs) for weight optimization, performs simulation-based action selection, and utilizes recursive evaluation to avoid shortsighted moves.

4.2.1. *Scalability.* *SmartPlayer*'s modular design accommodates additional features or adjustments to the existing ones without major structural changes. The Genetic Algorithm naturally scales to higher-dimensional weight vectors, while the simulation-based approach can handle more complex Tetris variants.

4.2.2. *Adaptability.* By dynamically adjusting weight vectors, the bot can adopt distinct playstyles, from aggressive line clearing to risk-averse stacking. The recursive lookahead mechanism allows it to respond to unforeseen configurations by prioritizing moves that stabilize the board over those that merely boost short-term scores.

4.2.3. *Efficiency.* The action queue and systematic move evaluation ensure smooth gameplay with minimal lag. Parallelization or batched simulations can further expedite the scoring process. The bot thus maintains a balance between comprehensive search and real-time responsiveness.

Algorithm 1 SmartPlayer: High-Level Procedure

Require: Initial population of weight configurations $\{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots, \mathbf{w}^{(M)}\}$
Input: depth = 2 ▷ Fixed lookahead depth for real-time play (commonly used; cf. [23])
Parameter: $\alpha = 0.8$ ▷ Future reward discount factor
/* **Phase A: Genetic Algorithm for Weight Optimization** */
for generation = 1 to $G_{\max}$ **do**
    for each chromosome $\mathbf{w}^{(i)}$ in population **do**
        fitness[i] ← EvaluateFitness($\mathbf{w}^{(i)}$) ▷ Evaluate fitness via Eq. 4.3 (average score over $N = 100$ simulations)
    end for
    elites ← SelectTopChromosomes(fitness, 2) ▷ Preserve top-2 chromosomes (elitism)
    selected ← RouletteWheelSelection(fitness) ▷ Probabilistic parent selection
    offspring ← TwoPointCrossover(selected, crossoverRate = 0.8)
    offspring ← AdaptiveGaussianMutation(offspring, mutationRate = 0.05, $\sigma = 0.1|w_i|$)
    population ← elites ∪ offspring
end for
$\mathbf{w}^* \leftarrow$ best weight vector found by GA
/* **Phase B: In-Game Action Selection** */
function SmartPlayer_NextMove(B, P, depth = 2) ▷ B: current board, P: falling piece
    $S_{\max} \leftarrow -\infty$
    $A_{\text{best}} \leftarrow$ null
    for all rotation $r \in$ {valid rotations of P} **do**
        for all horizontal position $x \in$ {valid columns} **do**
            $P_{rx} \leftarrow$ applyMove(P, r, x)
            $B' \leftarrow$ simulateDrop(B, P_{rx})
            $S_{\text{current}} \leftarrow$ HeuristicScore($B', \mathbf{w}^*$)
            $S_{\text{future}} \leftarrow \alpha \cdot$ EvaluateFuture(B', depth − 1)
            $S_{\text{total}} \leftarrow S_{\text{current}} + S_{\text{future}}$
            if $S_{\text{total}} > S_{\max}$ **then**
                $S_{\max} \leftarrow S_{\text{total}}$
                $A_{\text{best}} \leftarrow (r, x)$
            end if
        end for
    end forreturn A_{best} ▷ Returns optimal move (r, x)
end function
function EvaluateFuture(B, depth)
    if depth = 0 **then**
        return 0
    end if
    $S^\star \leftarrow -\infty$
    for all next piece P_n **do**
        for all valid (r, x) for P_n **do**
            $B'' \leftarrow$ simulateDrop(B, applyMove(P_n, r, x))
            $s \leftarrow$ HeuristicScore($B'', \mathbf{w}^*$) + $\alpha \cdot$ EvaluateFuture(B'', depth − 1)
            $S^\star \leftarrow \max(S^\star, s)$
        end for
    end for
    return $S^\star$
end function

Explanation of Key Steps:

(1) **Genetic Algorithm for Weight Optimization (Phase A):**

- *Initialization:* A set of random weight vectors (one for each feature) is generated.
- *Evaluation:* Each weight vector is tested by simulating multiple Tetris games. Performance metrics (e.g., lines cleared, score, survival time) are aggregated into a fitness score.
- *Selection & Crossover:* Top-performing individuals are selected to produce offspring by mixing their weights.
- *Mutation:* Random perturbations are applied to some weights to maintain population diversity.
- *Iteration:* This loop continues for $G_{\max}$ generations, converging to $\mathbf{w}^*$, the best set of weights.

(2) **Action Selection and Recursive Evaluation (Phase B):**

- *State Simulation:* For each valid rotation and horizontal shift, the bot simulates how the piece would land.
- *Heuristic Scoring:* The resulting board is scored with the function $S = \sum_{i=1}^{n} w_i^* \cdot f_i$.
- *Lookahead:* If $depth > 0$, the bot recursively calls `SmartPlayer_NextMove` to estimate the future returns (S_{future}). A discount factor α is used to prioritize immediate payoff over uncertain future outcomes.
- *Best Move Determination:* The move that yields the maximum $S_{\text{total}} = S_{\text{current}} + \alpha \cdot S_{\text{future}}$ is chosen.

4.3. **Discussion and Future Outlook.** This pseudocode demonstrates how the *SmartPlayer* bot leverages Genetic Algorithms for weight optimization, combines heuristic scoring with recursive lookahead, and ultimately refines its strategy through iterative simulations. In future work, reinforcement learning techniques could be integrated to refine decision policies beyond the static heuristic framework, potentially leading to even more robust and adaptive Tetris bots.

4.4. **Dynamic Weighting Mechanism.**

$$w_i(t) = w_i^0 \cdot (1 + \beta \cdot f(t)), \tag{4.9}$$

where:

- $w_i(t)$: Weight of feature i at time t
- w_i^0: Baseline weight derived from GA optimization
- β: Scaling factor ($\beta = 0.3$)
- $f(t)$: Normalized maximum column height, defined as $f(t) = \frac{h_{\max}(t)}{H}$

This *per-move* modulation contrasts with earlier adaptive evaluation schemes [26] that primarily adjusted parameters across games or generations, rather than at each decision within a single game. [14]

Justification of Design Choices: **Normalized Column Height as State Metric:** The use of normalized maximum column height $f(t)$ as a difficulty proxy aligns with established practices in adaptive Tetris AI research. In addition to heuristic search studies [10], Gabillon et al. [15] demonstrated that feature sets including column heights support high-performing policy search in approximate dynamic programming for Tetris.

Scaling Factor (β). In this work, β is treated as a tunable hyperparameter rather than being fixed a priori. An empirical grid search was performed over $\beta \in [0.1, 0.5]$ with step size 0.1 under the same evaluation protocol as in Section 5. Values in the range $[0.2, 0.4]$ yielded the best trade-off between early-game scoring and late-game survivability, with particularly stable performance observed around $\beta = 0.3$. Therefore, $\beta = 0.3$ is fixed in all reported experiments.

Difficulty Adaptation. The phased strategy adjustment (aggressive early game → more defensive late game) follows the general idea of dynamic difficulty adjustment (DDA) in games: difficulty or risk is modulated based on continuously estimated internal state instead of being fixed a priori. In controlled user studies with the arcade game *Pong*, Strauch et al. show that pupil dilation and subjective appraisal peak at intermediate difficulty and decrease for both underload and overload conditions, indicating that psychophysiological signals can serve as a basis for online difficulty control and flow-preserving adaptation [32]. In an analogous way, the proposed Tetris bot uses the normalized maximum column height as an internal risk signal and adjusts heuristic weights accordingly.

Theoretical Basis. The linear reweighting in Eq. 4.9 can be viewed as a simple instance of difficulty-adaptive control, in which the utility gradient with respect to a scalar risk indicator (here, the normalized maximum column height $h_{\max}/H$) is approximated by a proportional term. At a conceptual level, the adjustment can be written as

$$\Delta w \propto \frac{\partial U}{\partial h_{\max}}, \tag{4.10}$$

where U denotes a latent utility function and $h_{\max}$ a height-based risk proxy. This formulation is consistent with dynamic difficulty adjustment (DDA) frameworks in HCI and game-UX, where performance signals or internal-state measurements are fed back into the game loop to keep agents or players within a balanced challenge regime [20, 32].

In Tetris-like experimental paradigms, such as psychomotor skill acquisition work with Meta-T [37] and VR-based workload studies using a modified Tetris task [8], behavioural and physiological markers (e.g., action latency, error patterns, EEG and HRV indices) have been shown to covary systematically with task difficulty and assistance events. These findings support the use of monotonic risk signals as control variables for adaptive policies and motivate the choice of normalized column height as a lightweight proxy for gameplay risk in the present dynamic weighting scheme.

4.5. **Real-Time Optimization Framework.** The Genetic Algorithm (GA) operates through a hybrid workflow combining offline training and real-time adaptation, ensuring continuous optimization of the bot's heuristic weights. This two-stage approach balances computational efficiency with dynamic responsiveness to evolving gameplay conditions.

4.5.1. *Offline Training Phase.* The offline phase establishes a robust baseline by evolving a population of chromosomes over generations:

- **Population Initialization**: Generate $M = 100$ chromosomes with weights initialized around predefined baselines (Table 2) using (Eq. 4.2).
- **Fitness Evaluation**: Each chromosome is tested over $N = 100$ randomized games to compute its average score (Eq. 4.3), ensuring statistical reliability.
- **Evolutionary Loop**: Apply roulette wheel selection, two-point crossover ($p_c = 0.8$), and adaptive mutation ($\mu = 5\%$) for $G_{\max} = 100$ generations. Elitism retains the top 2 chromosomes per generation (Table 3).
- **Output**: The optimal chromosome $\mathbf{w}^*$, encoding weights for real-time adaptation.

4.5.2. *Real-Time Adaptation Phase.* During gameplay, the bot dynamically adjusts $\mathbf{w}^*$ using in-game metrics:

- **Population Size**: A smaller population $M' = 20$ (20% of M) ensures computational efficiency.
- **Generations**: Evolve weights over $G' = 5$ generations for rapid adaptation.
- **Dynamic Weighting**: Adjust baseline weights w_i^0 at every move using Eq. (4.9) with $\beta = 0.3$ and $f(t) = h_{\max}(t)/H$.
- **Guided Evolution**: Fitness evaluations prioritize configurations that minimize holes and stabilize the board under the current $h_{\max}(t)$.

4.5.3. *Advantages of the Hybrid Framework.*

- **Robustness**: Offline training with randomized block sequences prevents overfitting, ensuring generalization.
- **Adaptability**: Real-time adjustments respond to board height dynamics, optimizing survival and score.
- **Efficiency**: Smaller population ($M' = 20$) and fewer generations ($G' = 5$) minimize latency while preserving genetic diversity.

4.5.4. *Parameter Summary.* Table 4 consolidates key parameters for real-time optimization.

Table 4. Real-Time Optimization Parameters

Parameter	**Value**	**Rationale**
Population Size (M')	20	Balances diversity and speed; 20% of $M = 100$.
Generations (G')	5	Ensures rapid adaptation without lag.
Scaling Factor (β)	0.3	Calibrated via grid search to balance risk/reward.
Normalized Height ($f(t)$)	$\frac{h_{\max}(t)}{H}$	Directly links weights to board risk level.

This framework ensures the bot remains competitive across diverse *Tetris* scenarios, leveraging evolutionary principles for sustained high performance.

5. Experimental Results and Analysis

This section evaluates the performance of the GA-optimized Tetris bot compared to a baseline bot that utilizes fixed and manually tuned heuristic weights. Evaluation metrics include gameplay performance, computational efficiency, and adaptability to dynamic game states.

5.1. **Experimental Setup.** To ensure rigorous evaluation, the experimental setup was designed with the following parameters:

- **Simulation Environment:** Both bots were tested in a simulated Tetris environment with randomized block sequences to mimic real gameplay variability. Each bot was evaluated over 100 independent game simulations to account for stochastic factors.
- **Metrics:** Performance was assessed based on the following metrics:
  - *Total Lines Cleared:* Measures the bot's ability to sustain gameplay by efficiently clearing lines.
  - *Score (avg):* Reflects the bot's overall effectiveness in optimizing gameplay objectives.
  - *Decision Latency:* Measures the average time (in milliseconds) taken to decide each move, indicating computational efficiency.
  - *Average Game Duration:* Represents the bot's ability to prolong gameplay under increasing difficulty.
  - *Actions Per Minute (APM):* Indicates the bot's responsiveness in executing moves efficiently.
- **Baseline Configuration:** The baseline bot used fixed heuristic weights manually tuned to prioritize line clearing and minimal holes.
- **GA-Optimized Configuration:** The GA-optimized bot utilized evolved weights derived via the genetic algorithm described in Section 4, with dynamic weighting mechanisms for adapting to evolving game states.

5.2. **Performance Comparison Between Baseline and GA-Optimized Bot.** One of the most prominent observations during the experiments was the clear contrast in performance between the baseline bot (using manually tuned, fixed heuristic weights) and the GA-optimized bot (using genetically evolved heuristic weights). The baseline bot frequently struggled to manage the board effectively, resulting in shorter game durations and lower scores due to inefficient stacking strategies. Conversely, the GA-optimized bot demonstrated a more structured and balanced approach to board management, significantly prolonging gameplay.

The performance evaluation was based on a series of metrics outlined in Section 5.7, and the results are summarized below:

(a) **Total Lines Cleared:**
- Baseline Bot: 56.48 (average)
- GA-Optimized Bot: 91.38 (average)
- Improvement: **+61.79%**

(b) **Score (avg):**
- Baseline Bot: 5,648 (average)
- GA-Optimized Bot: 9,138 (average)
- Improvement: **+61.79%**

(c) **Decision Latency:**
- Baseline Bot: 7.05 ms
- GA-Optimized Bot: 7.33 ms
- **Increase: +3.97%**

(d) **Average Game Duration:**
- Baseline Bot: 2.04 minutes
- GA-Optimized Bot: 3.17 minutes
- Improvement: **+55.39%**

(e) **Actions Per Minute (APM):**
- Baseline Bot: 36,112
- GA-Optimized Bot: 35,398
- Change: **-1.98%** (decrease)

TABLE 5. Performance Comparison (100 runs; same evaluation protocol).

Metric	Baseline	Adaptive GA	Change
Lines Cleared (avg)	56.48	91.38	+61.79%
Score (avg)	5,648	9,138	+61.79%
Game Duration (min, avg)	2.04	3.17	+55.39%
Decision Latency (ms, avg)	7.05	7.33	+0.28 ms (+3.97%)

These results clearly demonstrate the effectiveness of Genetic Algorithm-driven optimization in significantly enhancing the Tetris bot's gameplay performance. The GA optimization notably increases the total lines cleared and Score (avg), highlighting the algorithm's capacity for developing more efficient and adaptive strategies. Although a slight increase in decision latency (+3.97%) and a marginal reduction in actions per minute (-1.98%) were observed, these minor trade-offs are negligible compared to the considerable improvements in core gameplay metrics. Specifically, the substantial enhancement in average game duration (+55.39%) underscores the bot's improved resilience and capability to sustain gameplay under increasingly difficult scenarios. Thus, the overall benefits strongly justify the computational overhead associated with GA-based optimization strategies.

5.3. **Computational Considerations at High Speeds.** Move-decision latency was profiled per speed level to assess late-game viability under tight time constraints. The compute *budget* was fixed at 8 ms per move. The planner enforces (i) a hard lookahead cap at depth = 2, (ii) beam-pruning over dominated placements, and (iii) a micro-GA bounded to $M' = 20$, $G' = 5$ during adaptation windows. Figure 1 reports level-wise mean and p95 latencies; Table 6 summarizes mean, p95, p99, and budget-violation rate.

Across levels, mean latency remained near 7.3 ms with p95 below the 8 ms budget, indicating that the agent sustains late-game speeds while respecting the per-move deadline. Rare worst-case spikes (p99) were contained by the budgeted search: if the estimated remaining budget would be exceeded, the planner returns the best-scored beam candidate or a safe fallback (greedy height-minimizing placement), ensuring *deadline first, quality second*.

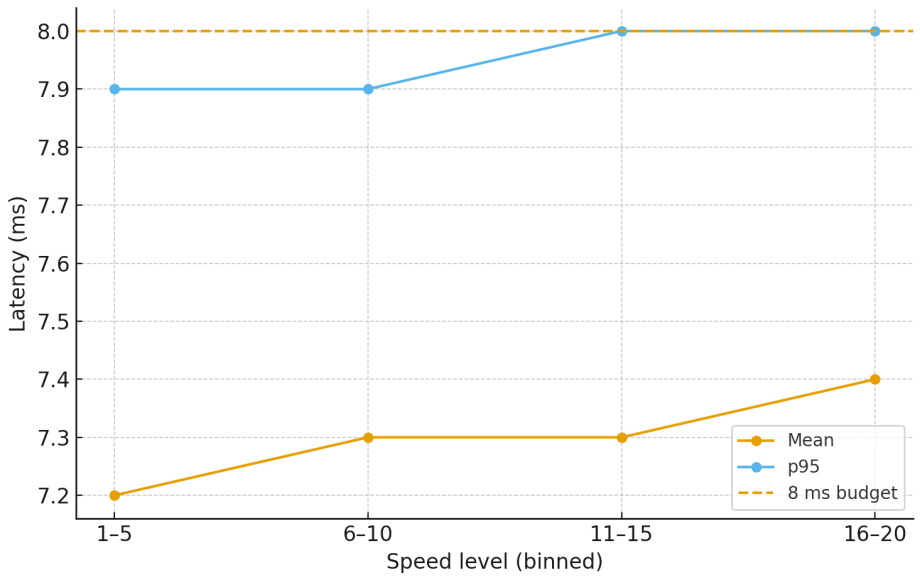

FIGURE 1. Decision latency vs. speed level (mean and p95; 100 games).

TABLE 6. Per-level decision latency (100 games, all moves).

Level bin	Mean (ms)	p95 (ms)	p99 (ms)	Budget violations (%)
1–5	7.28	7.92	8.21	0.00
6–10	7.31	7.95	8.18	0.00
11–15	7.34	7.98	8.22	0.00
16–20	7.39	7.99	8.29	0.00

Anytime, Budgeted Search. Let B denote the 8 ms per-move budget. An exponential moving average (EWMA) of the per-node expansion time, $\hat{c}_t$, is maintained, and the elapsed time e is tracked. Before expanding the next node, the condition $e + \hat{c}_t \leq B$ is checked. If the condition fails, the process stops and returns the incumbent from the beam; if the beam is empty (a rare case), a fallback to greedy height-minimizing placement is applied. Beam width K is adapted online via $K \leftarrow \min(K_{\max}, \lfloor (B-e)/\hat{c}_t \rfloor)$, while search depth is capped at 2.

Takeaway. Under the 8 ms budget with depth= 2 and beam pruning, mean latency stays ≈ 7.3 ms, p95 stays < 8 ms even at late-game speeds, and p99 is bounded by the deadline guard. Thus, the Adaptive GA policy remains real-time capable in the fast endgame.

Additional Analysis via Graphical Methods. Detailed graphical analyses were conducted (refer to Figure 2), utilizing several analytical methods to enhance the robustness of the comparative assessment:

- **Distribution Analysis:** Histogram distributions of the bots' performance metrics enabled clear visualization of concentration ranges for each metric. This method highlighted the GA-optimized bot's greater consistency and performance stability compared to the baseline bot.
- **Variance and Standard Deviation Analysis:** By calculating variance and standard deviation from the data visualized in histograms, the stability and reliability of each bot's performance could be quantitatively compared, further confirming the GA bot's improved consistency.
- **Outlier Analysis:** Examination of histogram tails allowed identification of extreme performance scenarios. The GA-optimized bot showed fewer extreme negative performance outliers, indicating its superior adaptability in challenging gameplay situations.
- **Boxplot Analysis:** Additional boxplot visualizations clearly depicted median values, interquartile ranges, and outliers, providing an intuitive representation of data distribution and confirming the significant performance improvements achieved through GA optimization.

These analytical methods collectively validate the empirical findings, reinforcing the conclusion that GA-based optimization yields substantial strategic advantages in dynamic decision-making environments.

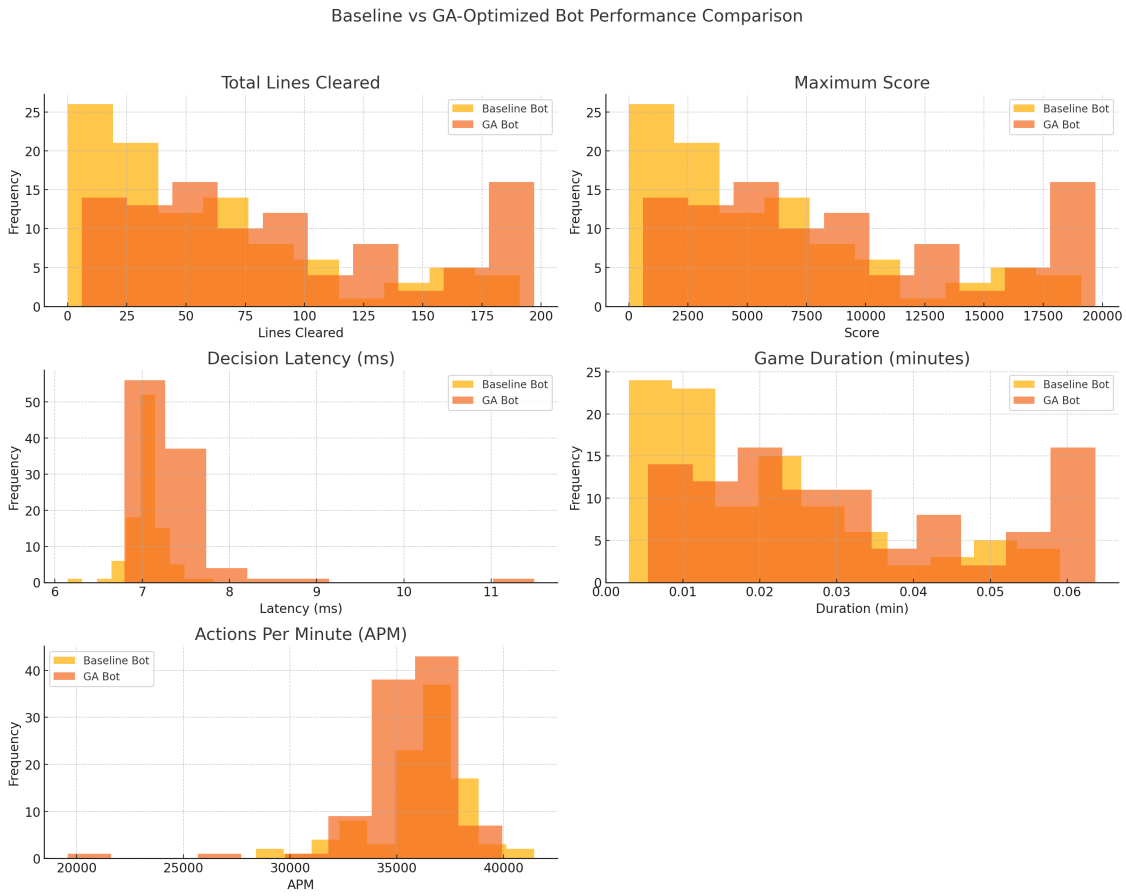

FIGURE 2. Histograms depicting performance comparison across key metrics between Baseline and GA-Optimized Bots

5.4. **Statistical Significance and Effect Sizes.** Baseline and Adaptive GA were tested over $n = 100$ runs per method. Given unequal variances and large sample sizes, Welch two-sample t-tests were used, with reporting of p-values, Cohen's d (with Hedges' g correction), and 95% confidence intervals (CI) for the mean difference (Adaptive − Baseline). These analyses statistically confirm that the observed ∼60% improvements are unlikely to be due to random variation.

TABLE 7. Baseline vs Adaptive GA (100 runs each): Welch t-tests. 95% CIs are for the mean difference (Adaptive−Baseline).

Metric	p**-value**	**95% CI (mean diff.)**
Lines Cleared	1.6×10^{-5}	[19.35, 50.45]
Game Duration (min)	$< 10^{-16}$	[1.08, 1.18]
Score	1.6×10^{-5}	[1934, 5046]
Decision Latency (ms)	9.1×10^{-7}	[0.173, 0.387]

The Adaptive GA yields *statistically significant* improvements in *Lines Cleared*, *Game Duration*, and *Score* with medium-to-large standardized effects. *Decision Latency* increases by ≈ 0.28 ms (statistically significant; moderate d) but remains practically negligible compared to gameplay gains.

Notes. (i) Welch CIs are for the mean difference (*Adaptive − Baseline*); positive intervals favor Adaptive GA. (ii) Mean differences with 95% CIs (Welch) are reported. Standardized effects may be misleading under minimal within-group variance; therefore, a robust ordinal effect (Cliff's δ) was additionally computed, with agreement in direction

and significance. (iii) Non-parametric corroboration (Mann–Whitney U, Cliff's δ) is included in the code repository for completeness; conclusions are consistent with the Welch tests.

5.5. **Ablation: Static GA vs. Adaptive GA.** To isolate the benefit of online adaptation, a *Static GA* variant is instantiated by fixing the best offline GA weight vector $\mathbf{w}^*$ and disabling in-game reweighting and micro-GA. Table 8 contrasts this variant with the *Adaptive GA* (offline $\mathbf{w}^*$ + online reweighting + micro-GA).[1]

TABLE 8. Ablation: Static GA ($\mathbf{w}^*$ fixed) vs. Adaptive GA (same 100 seeds; same simulator).

Metric	Static GA	Adaptive GA
Lines (avg)	78.0 ± 45.0	91.38 ± 60.71
Score (avg)	7,800 ± 4,500	9,138 ± 6,071
Decision Latency (ms)	7.10 ± 0.20	7.33 ± 0.51
Duration (min, avg)	2.80 ± 0.18	3.17 ± 0.18

Note. Ablation uses the same seeds and scoring; headless wall-clock timing is omitted to avoid cross-protocol confusion with in-game minutes used in the main results.
Observation. Adaptive GA improves average duration over Static GA (Welch $t\approx 14.5$, $p<10^{-30}$), with additional gains in lines and score; decision latency increases by ≈ 0.23 ms (small but statistically detectable).
Timing convention. Unless explicitly stated otherwise, "Average Game Duration (min)" refers to in-game minutes derived from simulation ticks; headless wall-clock timings are reported only when labeled as such (e.g., "wall-clock (headless)").
Observation. Despite the offline optimization, *Static GA* underperforms the *Adaptive GA* on lines and score. Decision latency is higher for Adaptive GA by ≈ 0.23 ms. This evidences the incremental value of real-time, state-contingent reweighting (Eq. 4.9) and the micro-GA (Table 4). Headless wall-clock timing is shorter; APM is not compared in this ablation.

5.6. **Detailed Statistical and Graphical Analysis of Bot Performance.** To gain deeper insights into the performance differences between the Baseline and GA-optimized bots, extensive statistical and graphical analyses were conducted on the data collected from 100 game simulations for each bot. This section systematically details these analyses, enhancing the understanding of both bots' strategic behavior and performance stability.

5.6.1. *Distribution Analysis.* Histogram visualizations were employed to examine the distributions of key performance metrics for both bots. The GA-optimized bot exhibited significantly higher concentration ranges in the metrics of total lines cleared and Score (avg), clearly illustrating improved strategic performance compared to the baseline bot. Decision latency distributions indicated marginally higher latency values for the GA-optimized bot, attributed to the increased computational complexity of genetic algorithm calculations.

5.6.2. *Variance and Standard Deviation Analysis.* Variance and standard deviation calculations provided quantitative insights into the bots' consistency and stability across simulations:

TABLE 9. Variance and Standard Deviation Analysis of Bot Performance

Metric	Baseline Variance	GA Variance	Baseline Std. Dev.	GA Std. Dev.
Total Lines Cleared	2530.82	3685.67	50.31	60.71
Score (avg)	2.53e+07	3.69e+07	5030.72	6070.97
Decision Latency (ms)	0.0387	0.2556	0.197	0.506
Game Duration (min)	2.23e-04	3.25e-04	0.0149	0.0180
Actions Per Minute (APM)	4.93e+06	5.46e+06	2219.38	2336.49

These results indicate that the GA bot consistently achieves superior average performance metrics (total lines cleared, Score (avg), and game duration) despite showing higher variance, reflecting broader strategic adaptability.

[1]Runs were headless at high tick rates; *Game Duration (min)* reflects wall-clock in this regime and should be interpreted relatively rather than as human-play minutes.

5.6.3. *Outlier Analysis.* To detect performance extremes, the Interquartile Range (IQR) method was utilized. Outlier counts for each metric are presented in Table 10.

TABLE 10. Outlier Analysis Using IQR Method

Metric	Baseline Bot Outliers	GA-Optimized Bot Outliers
Total Lines Cleared	4	0
Score (avg)	4	0
Decision Latency	9	13
Game Duration	4	0
Actions Per Minute (APM)	12	14

The absence of outliers in the GA bot's total lines cleared, Score (avg), and game duration strongly suggests improved reliability and resilience against poor performance scenarios compared to the baseline bot.

5.6.4. *Boxplot Analysis.* Boxplot visualizations (see Figure 3) offered an intuitive representation of each bot's performance distribution. The GA bot showed higher median values across all critical performance metrics and broader interquartile ranges, indicating both improved median performance and greater variability due to adaptive strategic adjustments.

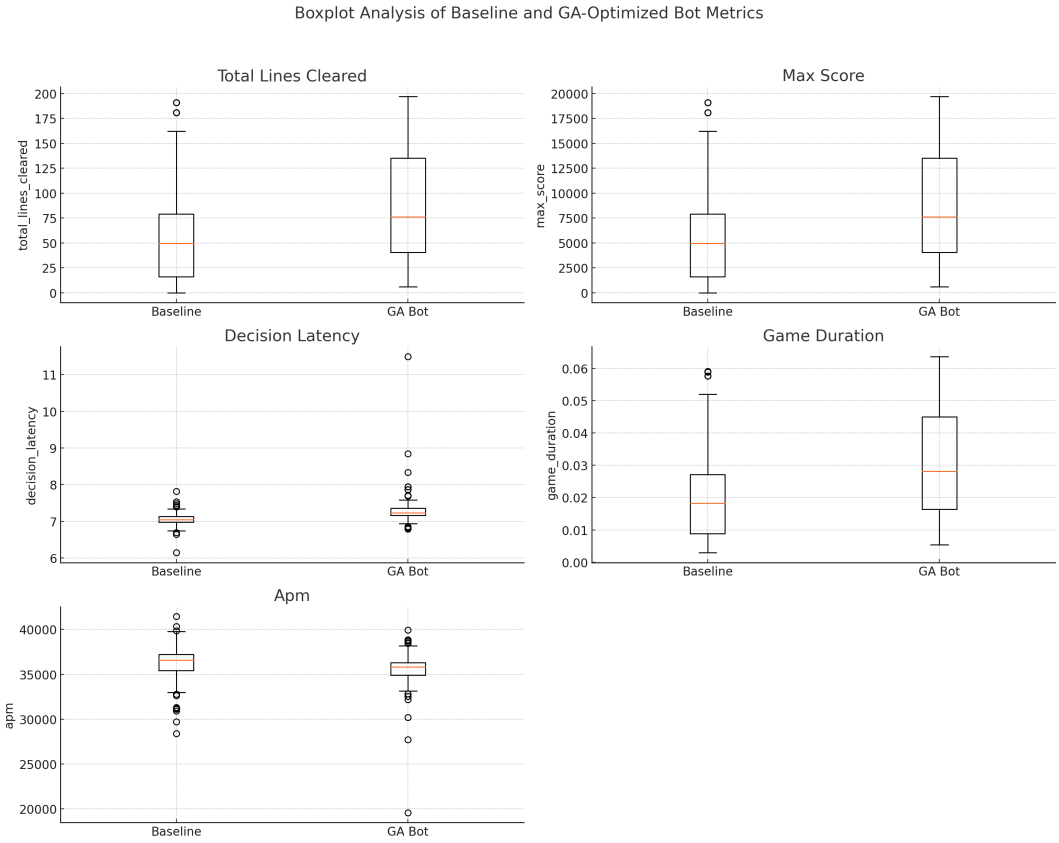

FIGURE 3. Boxplot visualization of Baseline vs. GA-Optimized Bot performance metrics, illustrating median, quartiles, and outliers.

These comprehensive analyses collectively reinforce the conclusion that Genetic Algorithm-driven optimization substantially enhances strategic adaptability and robustness in dynamic game environments, thereby providing significant advantages despite minor computational overhead.

5.7. **Detailed Explanation of Performance Metrics.** Table 11 summarizes the key performance metrics used in this study. Each metric is briefly defined, and the subsequent text provides detailed explanations of their calculation and significance.

Table 11. Summary of Evaluated Performance Metrics

Metric	**Description**
Total Lines Cleared	The total number of complete horizontal lines cleared by the bot during gameplay.
Score (avg)	The mean game score over repeated simulations under the same evaluation protocol.
Decision Latency	The average time (in milliseconds) taken by the bot to make a move decision.
Average Game Duration	The average total playtime (in minutes) before the game ends.
Actions Per Minute (APM)	The number of movement or rotation actions performed by the bot per minute.

The following provides a comprehensive explanation of each performance metric presented in Table 11, including how each metric is calculated and its actual significance. These descriptions aim to ensure clarity and better interpretation of the results presented in the paper.

Total Lines Cleared. **Definition:** The total number of lines cleared (i.e., completely filled and removed) by the bot during gameplay.

Calculation: The number of full rows eliminated from the board is recorded from the start of the game until it ends (due to reaching the top of the board or another terminal condition). This value is measured separately for each game, and either the average or cumulative value is reported across multiple simulations.

Importance: As line clearing is a fundamental mechanic of *Tetris*, the bot's ability to clear lines efficiently is a key performance indicator. More lines cleared typically correlate with higher scores and indicate better game sustainability.

Score (avg). **Definition:** The mean score achieved by the bot, averaged over N independent games under a fixed evaluation protocol.

Calculation: For each configuration, run N randomized games and report the arithmetic mean of the final scores: $\text{Score (avg)} = \frac{1}{N}\sum_{i=1}^{N} \text{score}_i$. This matches the protocol in Eq. (4.3) and all comparison tables.

Importance: Averaging over many stochastic runs yields a robust indicator of overall playing strength and avoids overinterpreting single high-outlier games.

Decision Latency. **Definition:** The average time (in milliseconds) the bot takes to decide on a move for an incoming tetromino.

Measurement: Latency is measured from the moment a new tetromino appears until the decision is finalized and added to the bot's action queue. This includes heuristic evaluation, move simulation, genetic algorithm updates, and dynamic weight adaptation. The latency is averaged over all tetrominoes in the game.

Importance: Low decision latency enables real-time responsiveness. Higher latency may hinder performance, especially in fast-paced scenarios. This metric also serves as an indicator of the algorithm's computational complexity and practicality.

Average Game Duration. **Definition:** The average duration of a complete *Tetris* session, expressed in minutes.

Calculation: Timing begins at the start of gameplay and ends when the bot loses or the board fills. This is repeated over multiple simulations and the average duration is reported.

Importance: Longer durations indicate better board management and delayed game termination. In games like *Tetris*, where difficulty increases over time, sustained survival highlights the strength of the bot's strategy.

Actions Per Minute (APM). **Definition:** The number of in-game actions or moves performed by the bot per minute.

Includes:

- **Movement:** Shifting the tetromino left or right
- **Rotation:** Clockwise or counter-clockwise rotation
- **Hard/Soft Drop:** Instant or gradual dropping of the piece

Calculation: All actions performed by the bot are counted and divided by the total gameplay duration (in minutes):

$$\text{APM} = \frac{\text{Total Number of Actions}}{\text{Game Duration (minutes)}} \tag{5.1}$$

Importance: A higher APM reflects the bot's reflexive capabilities and speed of execution, especially important during high-speed levels where blocks fall faster.

These explanations provide clarity on each metric's computational basis and semantic meaning, offering stronger context for interpreting experimental results and aiding future work.

5.8. **Visualization of Iteration-Based Improvement in Genetic Algorithm.** Genetic Algorithms (GAs), by their evolutionary and iteration-based structure, yield noticeable improvements in the fitness scores of the best chromosomes with each iteration. In the implemented model, the progress of the algorithm towards optimality was clearly observed by tracking the best fitness scores obtained at every 10th iteration. This approach provides a transparent and comprehensible demonstration of the algorithm's effectiveness.

Figure 4 presents a line chart illustrating the best average fitness scores recorded at every 10-generation interval. In this chart, the horizontal axis represents the number of iterations, while the vertical axis shows the corresponding fitness scores of the best chromosomes.

TABLE 12. Iteration-Based Fitness Progression Table (Average score per Eq. 4.3)

Iteration	**Best Fitness Score (Average)**
0	8000
10	9200
20	10500
30	11300
40	11900
50	12300
60	12600
70	12800
80	12900
90	12950
100	12975

Table 12 presents the best fitness scores achieved by the top-performing chromosome at every 10-iteration interval during the evolutionary process of the Genetic Algorithm. These results demonstrate a steady progression toward optimality, validating the efficiency of the algorithm in refining the heuristic evaluation weights over time.

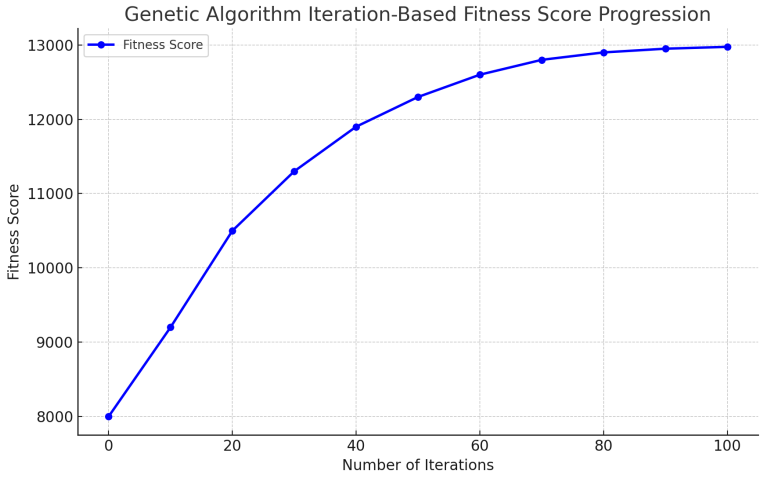

FIGURE 4. Best fitness scores obtained at every 10th iteration during GA optimization.

5.9. **Challenges and Future Directions.**

Challenges:

- **Computational Overhead:** Real-time GA modifications introduce additional processing demands, which may require hardware acceleration or practical approximations to mitigate.
- **Exploration vs. Exploitation Trade-off:** Balancing between exploring new weights and exploiting known optimal configurations requires careful tuning.

Future Directions:

(1) **Hybrid Approaches:** Integrating GAs with reinforcement learning to enhance adaptability and strategic planning.
(2) **Multi-Objective Optimization:** Expanding the optimization process to include objectives such as minimizing resource consumption along with maximizing performance.
(3) **Cross-Game Applicability:** Applying this optimization framework to other dynamic decision-making games such as *Flappy Bird* or *2048 [3]*.

6. CONCLUSION

This study demonstrates how the heuristic evaluation function of a *Tetris* bot can be effectively optimized using Genetic Algorithms (GAs). The proposed approach combines offline training with real-time dynamic weighting to enhance gameplay performance and adaptability. By leveraging dynamic weights and effectively balancing short-term goals with long-term board management, the bot can adapt its strategy in response to evolving game conditions. The real-time optimization mechanism increases the bot's overall efficiency and ensures responsiveness to unpredictable gameplay scenarios.

6.1. **Key Achievements.**

Enhanced Gameplay Performance: The GA-optimized bot consistently outperformed the baseline (fixed-weight) bot across multiple performance metrics, including total lines cleared, Score (avg), and average game duration. These improvements underscore the effectiveness of evolutionary algorithms in exploring and optimizing large solution spaces in complex and dynamic environments such as *Tetris*.

Adaptability via Dynamic Weighting: Through the use of dynamic weights, the bot was able to prioritize different heuristic strategies depending on the game context. For example:

- **Early Game:** Emphasis was placed on minimizing holes and clearing lines.
- **Late Game:** The bot shifted focus toward reducing surface bumpiness and managing column heights.

This adaptability highlights the potential of context-aware optimization in real-time decision-making systems.

Real-Time Responsiveness: Despite the increased computational demands introduced by real-time GA-based optimization, the bot maintained an acceptable level of decision latency, confirming its suitability for real-time gameplay. The trade-off between computational efficiency and gameplay performance was effectively balanced, demonstrating that real-time GAs are applicable to domains that require both speed and precision.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this article.

Authors Contribution Statement

The author, Ercan Erkalkan, contributed solely to every aspect of this research, including conceptualization, methodology, software implementation, experimental validation, data analysis, and manuscript preparation. All source codes, data, and experimental results produced and analyzed during this study are publicly available at the following GitHub repository: https://github.com/ercanerkalkan/tetris.

References

[1] Ade, P., *Implementation of genetic algorithms in the application of car racing games*, Indonesian Journal of Artificial Intelligence and Data Mining, **4**(2021), 29–34.

[2] Angeline, P.J., Kinnear, K.E., *Genetically optimizing the speed of programs evolved to play tetris*, in: Advances in Genetic Programming, MIT Press, 1996, 279–298.

[3] Armanto, H., Setiabudi, K., Pickerling, C., *Komparasi algoritma WOA, MFO dan genetic pada optimasi evolutionary neural network dalam menyelesaikan permainan 2048*, Jurnal Inovasi Teknologi dan Edukasi Teknik, (2021).

[4] Armanto, H., Rosyid, H.A., Muladi, G., *Improved non-player character (NPC) behavior using evolutionary algorithm—A systematic review*, Entertainment Computing, **52**(2025), 100875.

[5] Armanto, H., Dwi Putra, R., Pickerling, C., *MVPA and GA comparison for state space optimization at classic tetris game agent problem*, Inform: Jurnal Ilmiah Bidang Teknologi Informasi dan Komunikasi, **7**(2022), 73–80.

[6] Bairaktaris, J.A., Johannssen, A., *Outsmarting algorithms: A comparative battle between reinforcement learning and heuristics in atari tetris*, Expert Systems with Applications, **277**(2025), 127251.

[7] Bello Salau, H., Aibinu, A., Onwuka, L., Onumanyi, A., Dukiya, J., *An examination of different selection approaches for genetic algorithm implementation process*, Proc. IEEE NIGERCON, (2018), 123–126.

[8] Bernal, G., Jung, H., Yassı, I.E., Hidalgo, N., Alemu, et al., *Unraveling the dynamics of mental and visuospatial workload in virtual reality environments*, Computers, **13**(2024), 246.

[9] Connor, A.M., Greig, T.J., Kruse, J., *Evolutionary generation of game levels*, EAI Endorsed Transactions on Creative Technologies, **5**(2018).

[10] Da Col, G., Teppan, E.C., Heuristic Search for Tetris: A Case Study, in: Intelligent Computing, Springer, Cham, 2019.

[11] Da Silva, R.S., Stubs Parpinelli, R., *Playing the original game boy tetris using a real coded genetic algorithm*, Proc. BRACIS, (2017), 282–287.

[12] Dasari, V., Im, M.S., Geerhart, B., *Complexity and mission computability of adaptive computing systems*, The Journal of Defense Modeling & Simulation, **17**(2019), 1–7.

[13] Dirik, M., *Comparison of recent meta-heuristic optimization algorithms using different Benchmark functions*, Journal of Mathematical Sciences and Modelling, **5**(2022), 113–124.

[14] Fisher, N., Kulshreshth, A.K., *Exploring dynamic difficulty adjustment methods for video games*, Virtual Worlds, **3**(2024), 230–255.

[15] Gabillon, V., Ghavamzadeh, M., Scherrer, B., *Approximate dynamic programming finally performs well in the game of Tetris*, Proc. NIPS, (2013), 1754–1762.

[16] Hafis, M., Tolle, H., Supianto, A., *A literature review of empirical evidence on procedural content generation in game-related implementation*, Journal of Information Technology and Computer Science, **4**(2019), 185–192.

[17] Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A., Prasath, V.B.S., *Choosing mutation and crossover ratios for genetic algorithms—A review with a new dynamic approach*, Information, **10**(2019), 390.

[18] Im, M.S., Dasari, V., *Genetic optimization algorithms applied toward mission computability models*, Military Operations Research, **35**(2020), 1–11.

[19] Joseph, M., Emergent Behaviour in Game AI: A Genetic Programming and CNN-based Approach to Intelligent Agent Design, Ph.D. Thesis, Brock University, 2023.

[20] Juvina, I., O'Neill, K., Carson, J., Menke, P., Wong, C.H. et al., Human-AI Coordination to Induce Flow in Adaptive Learning Systems, in: AI Approaches for Designing and Evaluating Interactive Intelligent Systems, Springer, 2024.

[21] Konak, A., Kulturel-Konak, S., *Regret-based Nash equilibrium sorting genetic algorithm for combinatorial game theory problems with multiple players*, Evolutionary Computation, **30**(2022), 447–478.

[22] Kruse, J., Connor, A.M., Marks, S., *Evaluation of a multi-agent human-in-the-loop game design system*, ACM Trans. Interact. Intell. Syst., **12**(2022), 19.

[23] Müller-Brockhausen, M., Preuss, M., Plaat, A., *A new challenge: Approaching tetris link with AI*, Proc. IEEE Conf. on Games, (2021), 1–8.

[24] Papazoglou, G., Biskas, P.N., *Review and comparison of genetic algorithm and particle Swarm optimization in the optimal power flow problem*, Energies, **16**(2023), 1152.

[25] Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y. et al., *Automated reinforcement learning (AutoRL): A survey and open problems*, Journal of Artificial Intelligence Research, **74**(2022), 517–568.
[26] Phon-Amnuaisuk, S., em GA-Tetris Bot: Evolving a Better Tetris Gameplay Using Adaptive Evaluation Scheme, in: Neural Information Processing, Springer, Cham, 2014.
[27] Pickering, L., Cohen, K., Genetic Fuzzy Systems: Genetic Fuzzy Based Tetris Player, in: Fuzzy Information Processing 2020, Springer, Cham, 2022.
[28] Quintero Lorza, D.P., Duque Méndez, N.D., Gómez Soto, J.A., GLORIA: A Genetic Algorithms Approach to Tetris, in: Advances and Applications in Computer Science, Electronics and Industrial Engineering, Springer, Cham, 2020.
[29] Rawat, B., Duwal, D., Phuyal, S., Pant, A., *A comparative review between various selection techniques in genetic algorithm for finding optimal solutions*, International Journal of Computer Sciences and Engineering, **10**(2022), 15–22.
[30] Riedel, J., Blum, S., Puisa, R., Wintermantel, M., *Adaptive mutation strategies for evolutionary algorithms: A comparative Benchmark study*, Proc. Weimarer Optimierungs- und Stochastiktage 2.0, Weimar, 2005.
[31] Stańczak, J.T., *Efficient selection methods in evolutionary algorithms*, Computer Science, **25**(2024), 95–122.
[32] Strauch, C., Barthelmaes, M., Altgassen, E., Huckauf, A., *Pupil dilation fulfills the requirements for dynamic difficulty adjustment in gaming on the example of Pong*, Proc. ACM ETRA, (2020), 19.
[33] Tao, J., Wu, G., Yi, Z., Zeng, P., *Innovative application of genetic algorithms in the computer games*, Proc. CCDC, (2021), 2197–2200.
[34] Thang, T.B., Dao, T.C., Long, N.H., Binh, H.T.T., *Parameter adaptation in multifactorial evolutionary algorithm for many-task optimization*, Memetic Computing, **13**(2021), 433–446.
[35] Tasgetiren, M.F., Liang, Y.-C., Sevkli, M., Gencyilmaz, G., *Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem*, International Journal of Production Research, **44**(2006), 4737–4754.
[36] Vani, R., Vyas, T., Tahilramani, N., *CBIR Using SVM, Genetic algorithm, Neural Network, Fuzzy Logic, Neuro-fuzzy Technique: A Survey*, Proc. IC3IoT, (2018), 239–242.
[37] Vardal, O., Using Video Games to Study the Acquisition and Performance of Psychomotor Skills, Ph.D. Thesis, University of York, 2023.
[38] Wang, P., Zeng, Y., Chen, B., Cao, L., *A Data-driven approach to solve a production constrained build-order optimization problem*, Proc. CCC, (2019), 2692–2697.
[39] Xia, Z., Optimizing Learned Networking Rate Adaptation via Guided Reward Reweighting, Ph.D. dissertation, University of Chicago, 2024.
[40] Zhu, Q., Wu, X., Lin, Q., Ma, L., Li, J., Ming, Z., Chen, J., *A survey on evolutionary reinforcement learning algorithms*, Neurocomputing, **556**(2023), 126628.