# Harmonomino: Tetris Agent Optimization using Stochastic Local Search Heuristics written in Rust

E. Cerpac [a] and A. Tomatis [a]

[a] *Technische Universität Berlin, Berlin, Germany*

## 1. Introduction

Tetris, the iconic puzzle video game created by Alexey Pajitnov in 1984, has attracted substantial interest from the artificial intelligence and optimization research communities. A Tetromino is a geometric shape composed of four squares connected orthogonally (i.e. at the edges and not the corners) [Gol94], and in Tetris, a sequence of these pieces falls from the top of a $10 \times 20$ game board. The player must rotate and translate each piece to form complete horizontal rows, which are then cleared. The game terminates when the accumulated pieces prevent new pieces from entering the board, making the objective to clear as many rows as possible.

Demaine et al. [DHL03] proved that in the offline version of Tetris, maximizing the number of cleared rows, maximizing the number of simultaneous four-row clears ("Tetrises"), and minimizing the maximum height of occupied cells are all NP-complete problems. Furthermore, these objectives are inapproximable to within a factor of $p^{1-\varepsilon}$ for any $\varepsilon > 0$ , where $p$ is the number of pieces in the sequence. The immense complexity of Tetris is rooted in its state space, which encompasses approximately $7 \times 2^{20}$ possible configurations for a standard 20×10 board, as stated by Algorta et al. [AŞ19]. This vast state space, combined with the stochastic nature of piece generation, makes Tetris a challenging domain for both exact algorithms and heuristic approaches. As a result, researchers have turned to metaheuristic optimization techniques to develop agents capable of playing Tetris effectively, often by optimizing a set of heuristic weights that guide the agent's decision-making process.

### *1.1. Research Questions*

This work is guided by three research questions, which align with the broader goals of understanding and improving metaheuristic optimization for Tetris:

- **RQ1:** Can metaheuristic optimization converge to high-quality Tetris agents using only board-state features?
- **RQ2:** What structure exists in the learned weight space; are certain features consistently emphasized?
- **RQ3:** How does Harmony Search compare to Cross-Entropy Search under identical feature sets and simulation conditions?

### *1.2. Related Work*

The dominant approach to building Tetris-playing agents relies on a *state-evaluation function*: a linear combination of weighted board features that scores each possible placement. Given $n$ feature functions $f_i(s)$ mapping a board state $s$ to a real value, and corresponding weights $w_i$ , the agent selects the move that maximizes

$$V(s) = \sum_{i=1}^{16} w_i \cdot f_i(s). \tag{1}$$

The optimization problem then reduces to finding the weight vector $\boldsymbol{w}$ that yields the highest number of cleared rows [RTY11].

A variety of metaheuristic and machine learning approaches have been applied to this weight optimization problem. Böhm et al. [BKM05]

used evolutionary algorithms, evolving a population of weight vectors via selection, crossover, and mutation, and demonstrated that relatively simple feature sets can produce competent agents. Chen et al. [Che+09] applied ant colony optimization (ACO) to Tetris using a set of feature functions, reporting results competitive with other methods.

### *1.3. Cross-Entropy Search*

The application of Cross-Entropy Search to Tetris has been a cornerstone of heuristic optimization research, most notably advanced by Szita et al. [SL06]. They identified that the standard CES update rule often suffers from "variance collapse" due to the highly stochastic and "noisy" nature of the Tetris fitness landscape. Because the score for a single weight vector can vary significantly depending on the piece sequence, the sampling distribution may prematurely concentrate on "lucky" outliers—weights that performed well in a specific scenario but lack general robustness. To mitigate this, Szita et al. introduced the Noisy Cross-Entropy method, which injects additive noise into the covariance update. This technique ensures that the standard deviation never drops below a critical threshold, maintaining the exploratory pressure required to find truly global optima.

This line of research was further refined by Thiery et al. [TS09], who demonstrated that the performance of CES is highly sensitive to the evaluation budget and the choice of the elite set. They proposed improvements in how the "noise" is scaled relative to the current variance, allowing for more stable convergence in long-horizon simulations. Complementary to these distribution-based approaches, Langenhoven et al. [LHB10] explored the use of Particle Swarm Optimization (PSO) for the same weight-tuning problem, providing a benchmark that highlights CES's superior ability to navigate high-dimensional, non-convex reward surfaces.

Furthermore, Gabillon et al. [GGS13] positioned these optimization techniques within the broader framework of Approximate Dynamic Programming (ADP). They noted that while CES is effective at finding high-performing weights, it essentially performs a policy search in a space where the "true" value function is unknown. Their work emphasizes that the success of CES in Tetris—often achieving millions of lines cleared—stems from its ability to effectively approximate these complex decision boundaries through iterative sampling of the policy space.

### *1.4. The Harmony Search Algorithm*

The Harmony Search (HS) algorithm, introduced by Geem et al. [GKL01] in 2001, is a metaheuristic optimization algorithm inspired by the improvisation process of musicians. When musicians seek to create pleasing harmony, they may (1) play a known piece from memory, (2) play something similar to a known piece with slight variations, or (3) compose freely from random notes. These three strategies correspond to the three core mechanisms of HS: harmony memory consideration, pitch adjustment, and randomization.

The Harmony Search (HS) algorithm maintains a harmony memory (HM), a population of solution vectors. Each iteration constructs a new solution by either copying a value from HM (probability $r_{\text{accept}}$) or sampling randomly, with optional perturbation (probability $r_{\text{pa}}$). If the candidate outperforms the worst solution in HM, it replaces it. HS considers all solutions in HM, unlike genetic algorithms, which recombine only two parents [GKL01, Yan09].

Romero et al. [RTY11] were the first to apply Harmony Search to the Tetris weight optimization problem. Using 19 board feature functions and a harmony memory of size 5, their system demonstrated that HS can efficiently discover high-quality weight configurations, achieving a spawned-pieces-to-cleared-rows ratio approaching the theoretical optimum of 2.5. Our specific parameterization and implementation details are described in Section 2.4.

### 1.5. Contributions

This work presents *Harmonomino*, a Tetris agent optimization system implemented in Rust Our contributions are: that builds upon and extends the approach of Romero et al. [RTY11].

1. **Reimplementation and refinement.** We reimplement the Harmony Search-based Tetris optimizer in Rust for improved performance. Of the original 19 feature functions, we retain 16 that depend solely on the board state, removing three (removed rows, landing height, eroded pieces) that require game-context information beyond the current board configuration.

2. **Cross-Entropy Search as a comparative optimizer.** In addition to the Harmony Search algorithm, we implement a Cross-Entropy Search (CES) optimizer Szita et al. [SL06], Thiery et al. [TS09], enabling direct comparison between the two metaheuristic approaches under identical feature sets and simulation conditions.

3. **Benchmarking and parameter sweep framework.** We provide a benchmarking binary with parameter sweep support, enabling systematic exploration of hyperparameter sensitivity for both optimizers.

## 2. Methodology

### 2.1. Game Model

We implement a Tetris simulator on a $10 \times 20$ board with precomputed rotation tables and uniformly random piece generation (no 7-bag).

### 2.2. Agent Environment

The simulation employs a parallelized heuristic search to determine the optimal placement for each incoming tetromino by exhaustively evaluating the state space of possible moves. The system operates under a model where only the current piece is known, and the evaluation logic relies on a specific set of weights provided as a direct input to the simulation. For every potential coordinate on the board, along with the four possible rotations, the simulation validates that the piece can legally fit within the boundaries and ensures the gravity requirement is satisfied by confirming the piece can be locked into that specific configuration.

These valid placements are explored across all possible combinations of rotation and horizontal position in parallel, utilizing thread-safe iterators to maximize computational throughput. Each resulting board state, after accounting for row clearances, is appraised by a scoring framework that calculates a scalar fitness value as in Eq. 1

$$V(s) = \sum_{i=1}^{16} w_i \cdot f_i(s),$$

where $f_i(s)$ are board heuristics and $w_i$ are learned weights. The move yielding the highest score is executed, updating the global game state before the next piece is generated. This cycle continues until the board reaches a terminal "game over" condition or a predefined maximum move limit is reached, providing a deterministic and high-performance methodology for assessing the efficacy of different heuristic weight configurations.

The complete optimization pipeline (simulator + optimizer) is shown in Fig. 1. The agent's heuristic weights are optimized by an outer loop (HSA or CES) that iteratively generates candidate weight sets, which are evaluated by the inner Tetris simulator. The simulator computes the best move for each piece based on the current board state and the provided weights, returning a fitness score that guides the optimization process.

### 2.3. Heuristic Feature Set

We use 16 board features, all computable from the current board state alone. They fall into six categories:

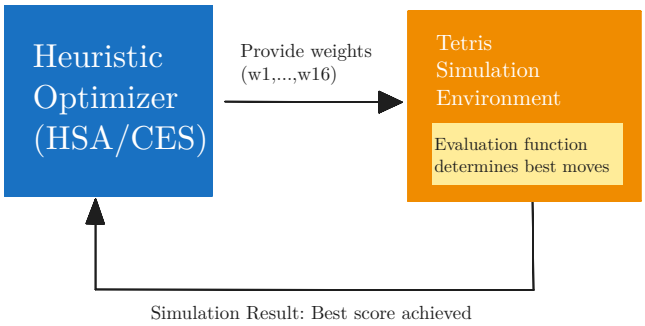


*Figure 1: Optimization pipeline.*

| | |
|---|---|
| **Height/Surface** | pile height<br>altitude difference<br>smoothness |
| **Holes** | holes<br>connected holes<br>highest hole<br>row holes<br>hole depth |
| **Wells** | max well depth<br>sum of well depths |
| **Transitions** | row transitions<br>column transitions |
| **Blocks** | total blocks<br>weighted blocks<br>blocks above highest hole |
| **Rows** | potential rows |

This feature set is a subset used by Romero et al. [RTY11] that does not require additional in-game context (we exclude removed rows, landing height, and eroded pieces).

### *2.4. Harmony Search Algorithm (HSA)*

The optimization framework employs a Harmony Search Algorithm (HSA) as described in Section 1.4. As illustrated in Fig. 2, the algorithm maintains a Harmony Memory (HM) of size 5. Each iteration generates a new candidate weight vector via three mechanisms:

- **Memory Consideration**: Inheriting values from the HM with probability $r_{\text{accept}} = 0.95$ .
- **Pitch Adjustment**: Applying localized perturbations to inherited values (governed by bandwidth 0.1) with probability $r_{\text{pa}} = 0.99$ .
- **Random Selection**: Sampling new values globally to maintain diversity.

Candidates are evaluated by averaging performance over multiple stochastic simulation runs to ensure robustness. If a candidate outperforms the weakest individual in the HM, it is replaced. The implementation always exhausts the budget of 100 iterations, as it does not employ early stopping.

### *2.5. Cross-Entropy Search (CES) Algorithm*

The framework utilizes Cross-Entropy Search (CES) presented in Section 1.3 to optimize weights by treating the search as a rare-event estimation problem. Unlike the population-based HSA, CES maintains a parameterized multivariate Gaussian distribution over the weight space.

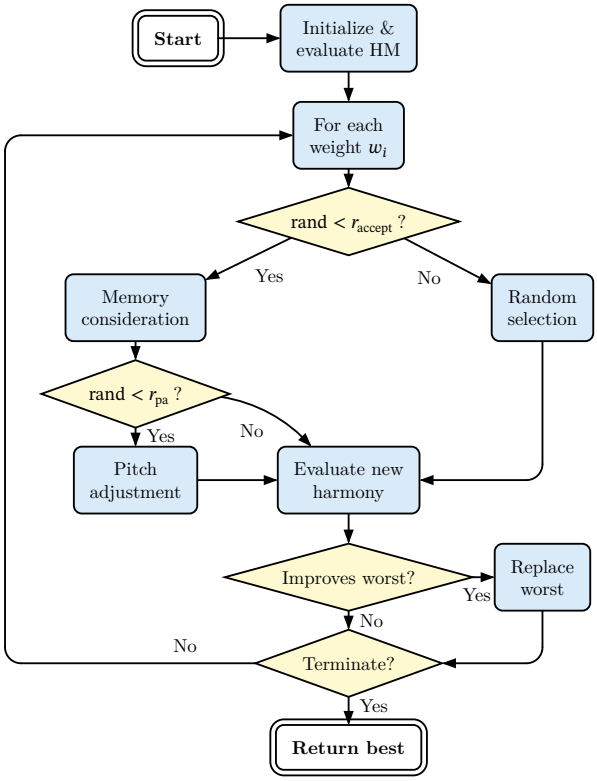


*Figure 2: Flowchart of the Harmony Search Algorithm (HSA). Each iteration generates a candidate by balancing memory exploitation and random exploration, replacing the worst member if fitness improves.*

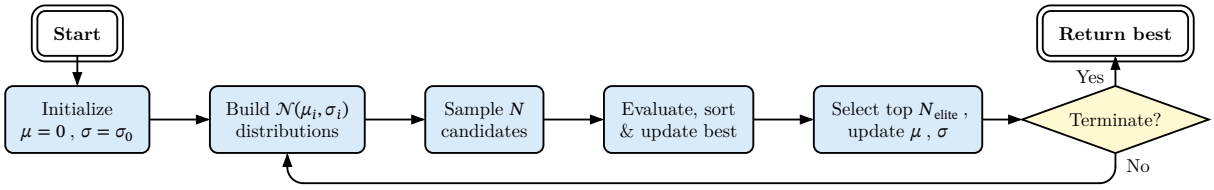


*Figure 3: Flowchart of the Cross-Entropy Search (CES) algorithm. The structure emphasizes the cyclic update of distribution parameters based on elite samples.*

The iterative process follows a linear sequence of sampling, evaluation, and refinement:

1. **Sampling**: $N = 50$ candidate weight sets are sampled from the current distribution.
2. **Evaluation**: Candidates are scored via simulation, with results averaged over multiple runs to mitigate game stochasticity.
3. **Refinement**: The top $N_{\text{elite}} = 10$ performers are selected to recalculate the distribution's mean and variance, shifting probability mass toward high-fitness regions.

The search begins with a standard deviation of 10 and enforces a floor of 0.01 to prevent premature convergence. CES employs early stopping if the best fitness reaches 399 otherwise, it exhausts the budget of 100 iterations. The procedure is summarized in Fig. 3.

### *2.6. Experimental Protocol*

We run 10 training seeds for each optimizer with a simulation length of 1000 pieces and a maximum of 100 iterations per seed. CES may terminate early when its fitness target is reached; HSA always exhausts its full iteration budget. For evaluation we use 30 fixed seeds with a simulation length of 2000 pieces. As a baseline we evaluate 30 random weight vectors sampled uniformly from $[-1, 1]$ . We report mean, median, standard deviation, and 95% confidence intervals of cleared rows, plus convergence, early-stopping, and weight-distribution plots.

## 3. Results

### *3.1. Agent Performance*

Tab. 1 summarizes the evaluation results across all methods. Each method is evaluated over $n$ independent games using fixed seeds. $\sigma$ denotes the sample standard deviation and *CI 95%* is the 95% confidence interval for the mean, computed as $1.96 \cdot \sigma/\sqrt{n}$ .

Both optimized methods dramatically outperform the baseline: CES achieves a mean of 417 cleared rows (median 351 ) and HSA a mean of 276 (median 210 ), compared to 4.55 for random weights.

Fig. 4 shows the full distribution of cleared rows. The box plots confirm that both optimized methods exhibit substantial variance (standard deviation 232 for CES and 208 for HSA), yet their lower quartiles still comfortably exceed the best baseline outcomes. CES edges out HSA in both mean and median, though the distributions overlap considerably.

### *3.2. Convergence Properties*

Fig. 5 traces the best and mean fitness of each optimizer over up to 100 iterations. CES converges rapidly, quickly narrowing its sam-

*Table 1: Evaluation statistics for each method (rows cleared over 30 fixed seeds).*

| Method | n | Mean | Median | σ | CI 95% |
|---|---|---|---|---|---|
| CES | 90 | 417 | 351 | 232 | 47.9 |
| HSA | 300 | 276 | 210 | 208 | 23.6 |
| Random | 900 | 4.55 | 1.00 | 10.7 | 0.702 |

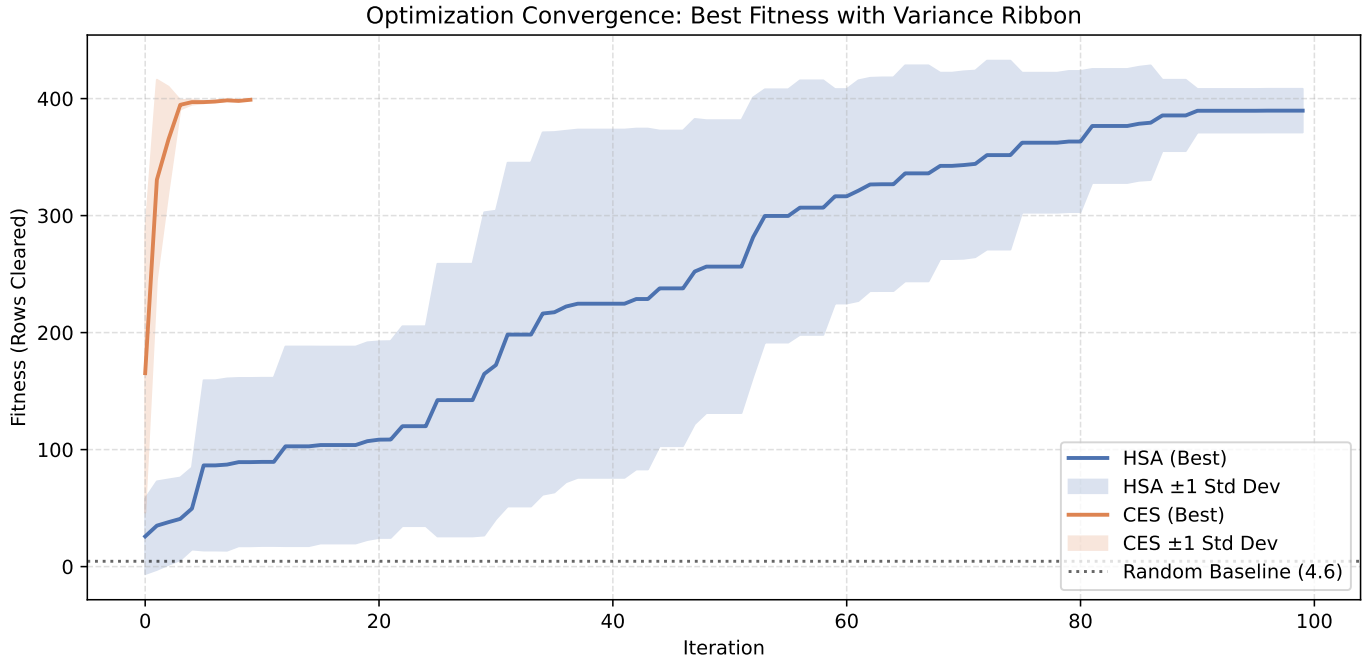


*Figure 5: Mean and best fitness across iterations for HSA and CES.*

pling distribution around high-fitness regions. HSA follows a more gradual trajectory, with steady improvements throughout the run as the harmony memory slowly replaces weaker candidates.

### *3.3. Early Stopping*

Fig. 6 shows the actual number of iterations used by each seed before optimization terminated. CES employs early stopping with a fitness target of 399 and consistently converges well before exhausting its 100-iteration budget, typically finishing within 5 iterations. HSA does not use early stopping and always runs for the full 100 iterations. This confirms that CES is

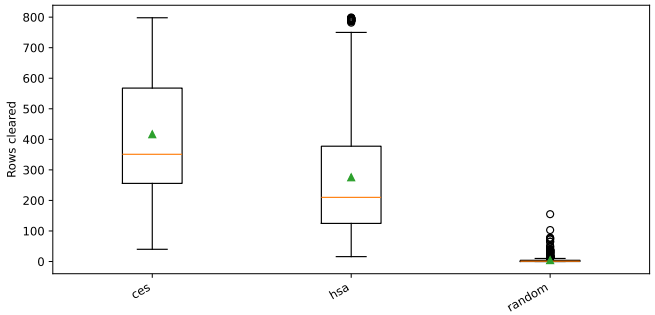


*Figure 4: Distribution of cleared rows for HSA, CES, and baselines.*

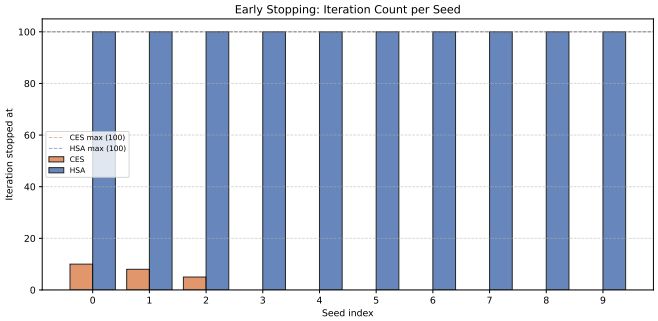


*Figure 6: Iteration at which each seed's optimization terminated.*

dramatically more efficient for this problem: it finds near-optimal weights in an order of magnitude fewer iterations than the allotted budget, while HSA requires the entire budget and still shows gradual improvements through the final iterations.

This difference in performance also shows up in the speed of execution, as shown in Fig. 7, where CES runs much slower than HSA. A clear performance gap is observed between the two configurations, reflecting the different computational demands of each approach. The accurate HS method exhibits relatively stable execution times, generally ranging between 12 and 19

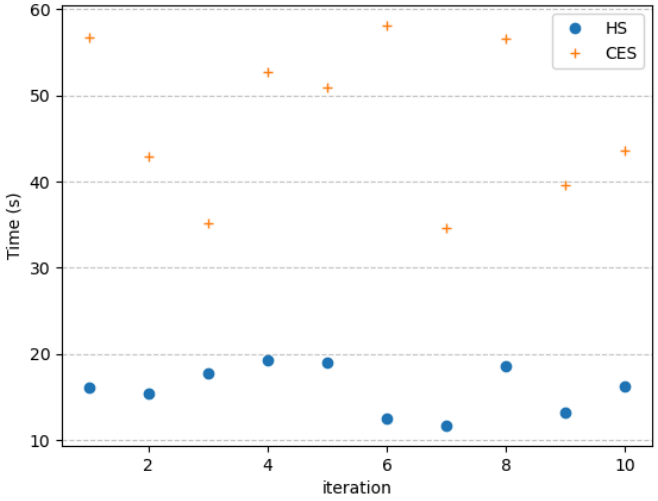


*Figure 7: Execution time comparison of the two algorithms (seconds).*

seconds per iteration. In contrast, the accurate CES method is significantly more computationally intensive, with processing times consistently exceeding those of HS and fluctuating between approximately 35 and 58 seconds.

### *3.4. Weight Distribution and Analysis*

Fig. 8 and 9 reveal the structure of the learned weight space. The most consistent weights, those with low standard deviation across seeds, include $w_8$ (Weighted Blocks, mean $\approx -0.561$ ), $w_9$ (Row Transitions, mean $\approx -0.621$ ), and $w_{12}$ (Blocks Above Highest, mean $\approx 0.558$ ). These features are assigned decisive, stable values regardless of

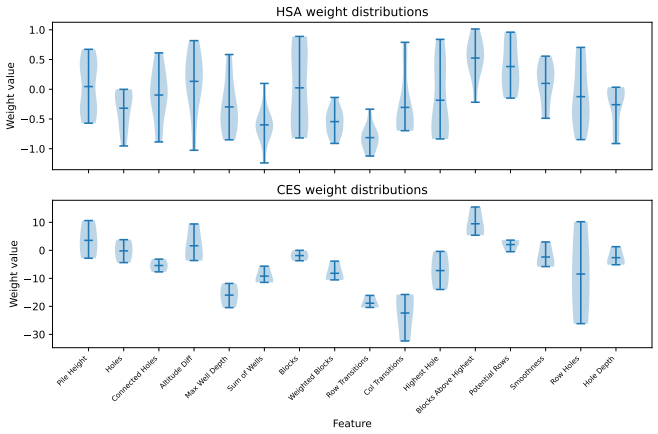


*Figure 8: Violin plots of learned weight distributions for HSA and CES.*

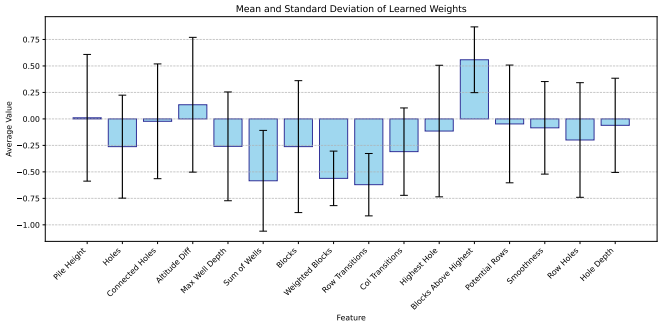


*Figure 9: Mean and standard deviation of each weight across all optimized runs.*

the optimization seed, suggesting they capture the most important aspects of board quality.

In contrast, several weights show high variance: $w_4$ (Altitude Diff), $w_7$ (Blocks), and $w_{11}$ (Highest Hole) all have standard deviations exceeding 0.6 . This indicates that multiple weight configurations achieve similar performance, and that the solution landscape admits a family of good solutions rather than a single optimum.

Fig. 10 shows the pairwise Pearson correlation between learned weights across all optimized runs. Most off-diagonal correlations are weak, indicating that the features capture largely independent aspects of board quality. However, notable positive correlations exist between height-related features, specifically Blocks Above Highest and Pile Height, as well as between Holes and

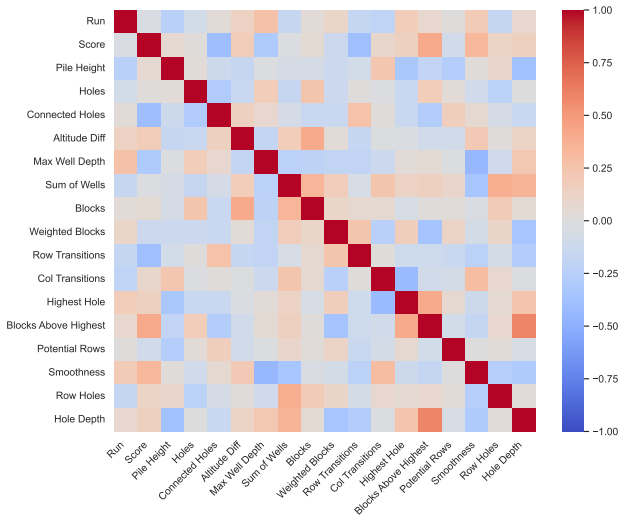


*Figure 10: Pairwise Pearson correlation of learned weights across all optimized runs.*

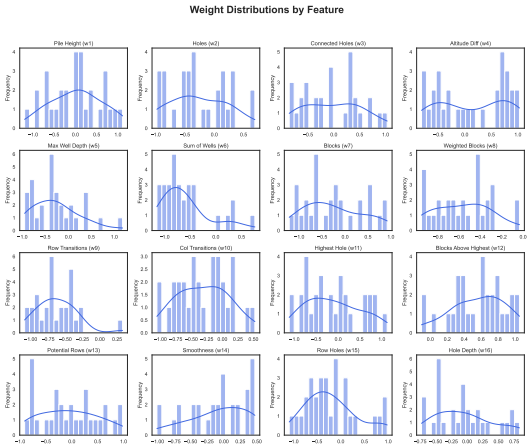


*Figure 11: Frequency distribution of learned weights across all optimization runs.*

Connected Holes. Conversely, several features show near-zero or slightly negative correlations, such as Max Well Depth relative to Row Transitions, suggesting the optimization process successfully distinguishes between internal board structures and surface-level instability.

The learned weight distributions demonstrate clear directional trends for specific game-state features. As shown in the weight histograms Fig. 11, features like Row Transitions and Col Transitions consistently gravitate toward negative values, while others are aggregated by category to show their mean impact.

The consistency of these results is validated through clustering. The k-distance plot in Fig. 13 identifies an elbow at approximately 1.35, providing a principled epsilon value for DBSCAN. Using this parameter, the stability heatmap reveals that the majority of optimization runs converge into a single primary cluster.

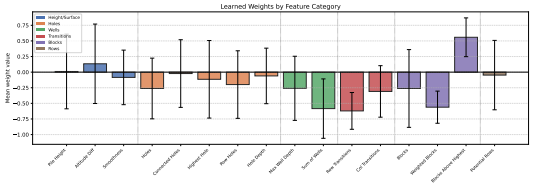


*Figure 12: Mean weight values grouped by feature category showing relative importance.*

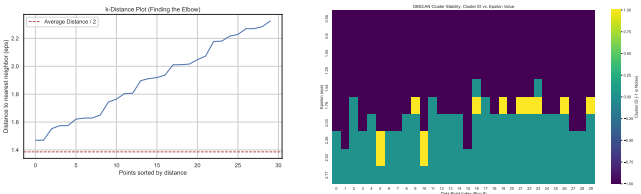


*Figure 13: K-distance elbow plot and DBSCAN stability analysis.*

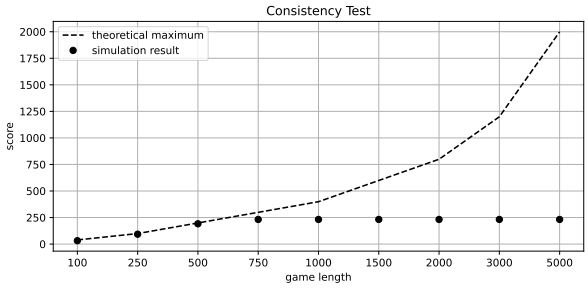


*Figure 14: Comparison between simulation results and the theoretical maximum score across increasing game lengths.*

### *3.5. Consistency Analysis*

The consistency of the simulation environment was evaluated by comparing empirical results against a theoretical performance model across varying game lengths. As the game length increases, the simulation results initially follow the theoretical maximum closely but begin to plateau after a length of approximately 500.

The divergence between the theoretical expectation and the actual agent performance is further detailed in the absolute error analysis shown in Fig. 15. While the error remains near zero for shorter durations (up to a game length of 500), it grows significantly as game length exceeds 750, eventually reaching an absolute error of over 1750 at a game length of 5000.

As illustrated in Fig. 14, the agent's performance becomes decoupled from the theoretical maximum as the simulation progresses. This trend indicates that while the agent is highly consistent in short-term scenarios, long-term performance is capped by constraints that the theoretical model does not account for.

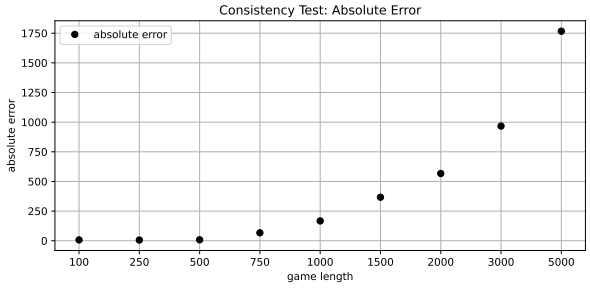


*Figure 15: Absolute error between theoretical and simulation results as a function of game length.*

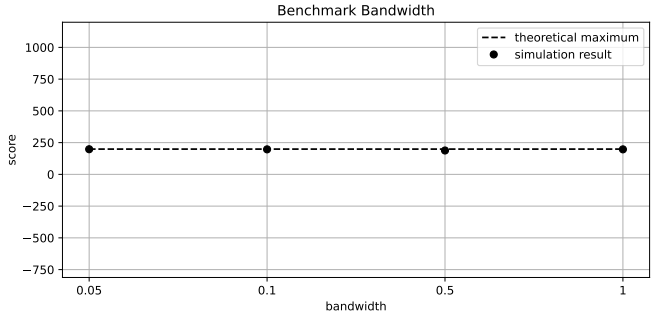


*Figure 16: Effect of pitch-adjustment bandwidth on agent score.*

### *3.6. Parameter Sensitivity*

To assess hyperparameter sensitivity for Harmony Search, we sweep three key parameters while holding the others fixed.

Fig. 16 shows that bandwidth has a moderate effect: too-small values restrict exploration, while excessively large values introduce disruptive perturbations. Fig. 17 confirms diminishing returns beyond roughly 170 iterations, consistent with the convergence analysis above. Fig. 18

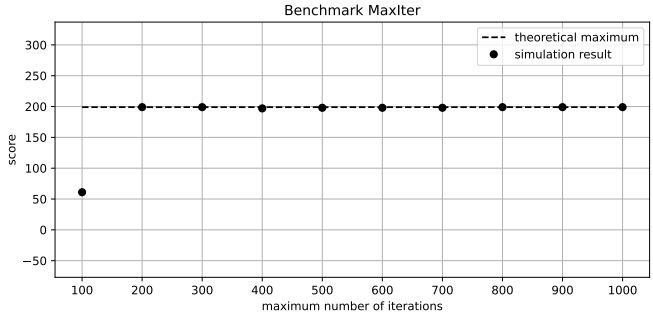


*Figure 17: Effect of maximum iterations on agent score.*

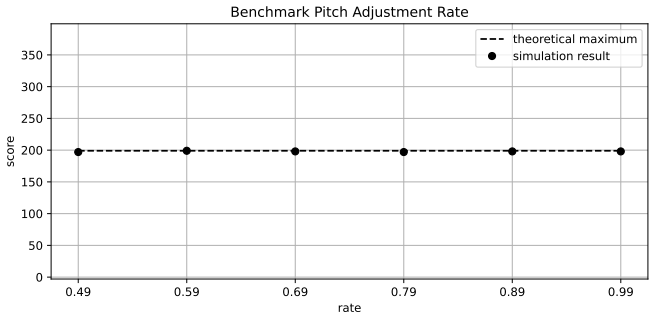


*Figure 18: Effect of pitch-adjustment rate on agent score.*

indicates that the pitch-adjustment rate has little effect as well on final performance.

## 4. Conclusion and Discussion

We presented *Harmonomino*, a Rust-based system that optimizes Tetris-playing agents via Harmony Search and Cross-Entropy Search over 16 board-state features. Both optimizers converge to weight vectors that dramatically outperform baselines: CES achieves a mean of 417 cleared rows and HSA 276 , compared to 4.55 for random weights. CES reaches higher mean and median scores than HSA while converging faster in early iterations. These results affirm that metaheuristic optimization over a linear heuristic suffices to produce competent Tetris agents from board-state features alone (RQ1), and that CES holds a moderate advantage over HSA under identical conditions (RQ2).

Analysis of the learned weight distributions reveals some structure in the solution space (RQ3). Certain features, such as $w_8$ (Weighted Blocks), $w_9$ (Row Transitions), and $w_{12}$ (Blocks Above Highest), receive consistent values across seeds, with standard deviations below 0.3 . Conversely, weights for $w_4$ (Altitude Diff), $w_7$ (Blocks), and $w_{11}$ (Highest Hole) vary widely, suggesting that the loss landscape admits a family of similarly-performing solutions rather than a single optimum.

### *4.1. Limitations*

Several design choices constrain the generality of these findings. The simulator uses a

uniform random piece generator rather than the 7-bag system used in modern Tetris implementations, which might change statistical properties of piece sequences. The agent operates without lookahead, evaluating only the current piece, which limits its ability to plan for future placements. The high variance across evaluation seeds (standard deviation 232 for CES and 208 for HSA) indicates substantial sensitivity to the random piece sequence, and different seeds can yield performance ranging from near-baseline to several hundred cleared rows. Finally, the reduced feature set, 16 of the original 19 features from Romero et al. [RTY11] excludes game-context heuristics such as landing height and eroded pieces, which may limit the agent's performance ceiling.

### *4.2. Future Work*

Promising extensions include adopting the 7-bag piece generator for closer fidelity to standard Tetris, introducing limited lookahead (one or two pieces) to enable planning, and reintroducing the three excluded game-context features. Hybrid optimizers that combine Cross-Entropy sampling with local search refinement may further improve convergence speed. Larger-scale parameter sweeps across both optimizers, together with longer simulation lengths, would provide more robust sensitivity analyses and tighter confidence intervals.

## References


[AŞ19] S. Algorta and Ö. Şimşek, "The Game of Tetris in Machine Learning," *arXiv preprint arXiv:1905.01652*, 2019. doi: 10.48550/arXiv.1905.01652.

[BKM05] N. Böhm, G. Kókai, and S. Mandl, "An Evolutionary Approach to Tetris," in *The Sixth Metaheuristic International Conference (MIC)*, 2005. Available: https://api.semanticscholar.org/CorpusID:123172319.

[Che+09] X. Chen, H. Wang, W. Wang, Y. Shi, and Y. Gao, "Apply ant colony optimization to Tetris," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, , pp. 1741–1742, Association for Computing Machinery, 2009. doi: 10.1145/1569901.1570136.

[DHL03] E. D. Demaine, S. Hohenberger, and D. Liben-Nowell, "Tetris is Hard, Even to Approximate," in *International Computing and Combinatorics Conference*, , pp. 351–363, 2003. doi: 10.48550/arXiv.cs/0210020.

[GGS13] V. Gabillon, M. Ghavamzadeh, and B. Scherrer, "Approximate Dynamic Programming Finally Performs Well in the Game of Tetris," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[GKL01] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search," *Simulation*, , vol. 76, , pp. 60–68, 2001. doi: 10.1177/003754970107600201.

[Gol94] S. W. Golomb, "Polyominoes: Puzzles, Patterns, Problems, and Packings," Princeton University Press, 1994.

[LHB10] L. Langenhoven, W. S. van Heerden, and A. P. Barnard, "Swarm Tetris: Applying Particle Swarm Optimization to Tetris," in *IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2010. doi: 10.1109/CEC.2010.5586050.

[RTY11] V. I. M. Romero, L. L. Tomes, and J. P. T. Yusiong, "Tetris Agent Optimization Using Harmony Search Algorithm," *International Journal of Computer Science Issues*, , vol. 8, 2011. Available: https://www.researchgate.net/publication/265566527_Tetris_Agent_Optimization_Using_Harmony_Search_Algorithm.

[SL06] I. Szita and A. Lörincz, "Learning Tetris Using the Noisy Cross-Entropy Method," *Neural Computation*, , vol. 18, , no. 12, , pp. 2936–2941, 2006. doi: 10.1162/neco.2006.18.12.2936.

[TS09] C. Thiery and B. Scherrer, "Improvements on Learning Tetris with Cross Entropy,"

*ICGA Journal*, , vol. 32, , no. 1, , pp. 23–33, 2009. doi: 10.3233/ICG-2009-32104.

[Yan09] X.-S. Yang, "Harmony Search as a Metaheuristic Algorithm," *Music-Inspired Harmony Search Algorithm: Theory and Applications*, Springer, 2009. doi: 10.1007/978-3-642-00185-7_1.

---

### *Declaration of Generative AI in Scientific Writing*

*During the preparation of this work, the authors used generative AI tools in order to improve readability and ensure a correct use of the English language. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the report.*