Deep Reinforcement Learning for Block Blast: A PPO-Based Approach with Action Masking

Rawad Fahd and Priya Ashok Sardhara

December 2025

Abstract

This paper presents a deep reinforcement learning agent for Block Blast, a strategic puzzle game played on an 8×8 grid where players place randomly generated pieces to complete rows and columns. We implement Proximal Policy Optimization (PPO) with action masking to handle the game's variable action space, where invalid placements must be excluded from consideration. Our architecture combines convolutional neural networks for spatial feature extraction with residual connections for improved gradient flow. Training over 17.3 million timesteps demonstrates substantial learning progress, with average game scores improving from 410 to 4,470 points and episode lengths increasing from 14 to 45 moves. The agent learns sophisticated strategies including center-board preservation and combo setup for score multiplication. Our results establish baseline performance metrics for applying modern policy gradient methods to combinatorial puzzle games with constrained action spaces.

Index Terms— Reinforcement Learning, Proximal Policy Optimization, Action Masking, Puzzle Games, Deep Learning

1. Introduction

Puzzle games present unique challenges for artificial intelligence research. Unlike board games with well-defined state spaces and clear winning conditions, puzzle games often feature procedurally generated content, score optimization rather than binary outcomes, and action spaces that vary dramatically based on game state. Block Blast exemplifies these characteristics: players must place randomly selected pieces onto an 8×8 grid, clearing completed rows and columns while avoiding board states where no valid moves remain.

The game demands both tactical and strategic thinking. Tactical decisions involve placing individual pieces to maximize immediate line clears, while strategic considerations require maintaining board configurations that accommodate future pieces of unknown shapes. This dual timescale of decision-making mirrors challenges found in many real-world optimization problems, from logistics scheduling to resource allocation.

Recent advances in deep reinforcement learning have produced superhuman performance across diverse game domains. Deep Q-Networks mastered Atari games from raw pixels [1], AlphaGo defeated world champions in Go [2], and OpenAI Five coordinated complex team strategies in Dota 2 [4]. These successes demonstrate that neural networks can learn effective policies for high-dimensional state spaces when combined with appropriate reinforcement learning algorithms.

Block Blast introduces a distinct challenge: the action space changes dynamically based on available pieces and board state. Traditional approaches that enumerate all possible actions become inefficient when most actions are invalid. We address this through action masking, which restricts policy outputs to valid moves only, improving both training efficiency and inference reliability.

This paper makes the following contributions:

1. We present a complete reinforcement learning system for Block Blast, including environment implementation, neural network architecture, and training infrastructure.
2. We demonstrate that PPO with action masking effectively handles the game's constrained action space, achieving consistent learning progress across 17.3 million training steps.
3. We provide detailed analysis of learned behaviors, showing the agent develops strategies such as center preservation and combo setup that align with expert human play.

4. We establish reproducible baseline metrics for future research on puzzle game AI.

The remainder of this paper is organized as follows. Section II reviews related work in game-playing AI and action-constrained reinforcement learning. Section III describes our methodology, including the game formulation, neural network architecture, and training algorithm. Section IV presents experimental results and analysis. Section V discusses conclusions and directions for future work.

2. Background and Related Work

2.1 Reinforcement Learning Foundations

Reinforcement learning formalizes sequential decision-making as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ) where S represents states, A represents actions, $P(s'|s, a)$ defines transition dynamics, $R(s, a)$ specifies rewards, and γ discounts future returns [7]. The objective is to learn a policy $\pi(a|s)$ that maximizes expected cumulative reward.

Policy gradient methods optimize the policy directly by estimating gradients of expected return with respect to policy parameters. The REINFORCE algorithm [8] provided the theoretical foundation, demonstrating that policy gradients could be estimated through sampled trajectories. Actor-critic methods improved sample efficiency by learning both a policy (actor) and a value function (critic), reducing variance in gradient estimates [9].

2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) addresses a fundamental challenge in policy gradient methods: determining appropriate update step sizes [5]. Large updates can destabilize training by moving too far from the data collection policy, while small updates waste samples. PPO constrains updates through a clipped surrogate objective:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the probability ratio between new and old policies, and $\hat{A}_t$ estimates the advantage function. The clipping parameter ϵ (typically 0.2) prevents the ratio from deviating too far from 1, ensuring conservative policy updates.

PPO has demonstrated robust performance across diverse domains, from continuous control tasks to complex games [5]. Its relative simplicity compared to trust region methods like TRPO [6] makes it particularly suitable for large-scale distributed training.

2.3 Action Masking in Reinforcement Learning

Many environments feature state-dependent action constraints where certain actions are invalid or undefined. Naive approaches that assign large negative rewards to invalid actions introduce unnecessary complexity and can destabilize training. Action masking provides a principled alternative by modifying the policy network's output layer to exclude invalid actions from consideration [10].

Formally, given a mask $m \in \{0, 1\}^{|A|}$ indicating valid actions, the masked policy is:

$$\pi_{masked}(a|s) = \frac{\pi(a|s) \cdot m(a)}{\sum_{a'} \pi(a'|s) \cdot m(a')}$$

In practice, this is implemented by setting logits for invalid actions to negative infinity before the softmax operation, ensuring zero probability mass on invalid actions. This approach has proven effective for board games [11], combinatorial optimization [12], and robotic planning [13].

2.4 Game-Playing AI

Deep reinforcement learning has achieved remarkable success in game domains. Deep Q-Networks demonstrated that neural networks could learn effective policies directly from high-dimensional sensory input [1]. AlphaGo combined Monte

Carlo tree search with neural network evaluation to defeat professional Go players [2]. Subsequent work on AlphaZero showed that self-play alone could achieve superhuman performance without domain-specific knowledge [3].

Puzzle games present distinct challenges compared to adversarial games. Without an opponent, the learning signal comes entirely from intrinsic game mechanics. Tetris, perhaps the most studied puzzle game in AI research, has been approached through hand-crafted heuristics [14], evolutionary algorithms [15], and deep reinforcement learning [16]. Block Blast shares Tetris's core mechanics of piece placement and line clearing but introduces additional complexity through its larger piece variety and two-dimensional clearing (both rows and columns).

3. Methodology

3.1 Problem Formulation

Block Blast is played on an 8×8 grid where the player receives three randomly selected pieces per turn from a pool of 37 distinct shapes. Pieces range from single blocks to 3×3 squares, with various L-shapes, T-shapes, and linear configurations. Each turn, the player places all three pieces sequentially at valid grid positions. When a row or column becomes completely filled, it clears and awards points.

We formalize Block Blast as an MDP with the following components:

State Space: The state s_t comprises the 8×8 board configuration (64 binary values indicating filled/empty cells), three 8×8 piece masks encoding the current pieces' shapes, and three binary flags indicating which pieces have been used this turn.

Action Space: Actions correspond to piece placements (p, r, c) where $p \in \{0, 1, 2\}$ selects the piece, and (r, c) specifies the grid position. With 3 pieces and 64 positions, the nominal action space contains 192 actions, though most are invalid at any given state due to piece shapes and board occupancy.

Transition Dynamics: Placing a piece fills the corresponding grid cells. Complete rows and columns immediately clear. After using all three pieces, new pieces are generated. The episode terminates when no valid placement exists for any remaining piece.

Reward Function: We design a reward function that balances immediate feedback with long-term strategic incentives:

- Block placement: +0.01 per block placed
- Line clear: +1.0 per line cleared
- Combo multiplier: +0.5 per multiplier level above 1
- Game over: -1.0 terminal penalty
- Hole creation: -0.05 per new isolated cell
- Center preservation: +0.02 for maintaining center openness

The reward scaling was tuned to produce values roughly in the range $[-1, 1]$ for stable gradient updates.

3.2 Neural Network Architecture

Our architecture processes the spatial structure of Block Blast through convolutional neural networks with residual connections, depicted in Figure 1.

Input Encoding: The input tensor has shape $(4, 8, 8)$ comprising one channel for the board state and three channels for piece masks. Each piece mask renders the piece's shape at the grid origin, enabling the network to process piece geometry through the same convolutional pathway as board state.

Convolutional Encoder: Three convolutional layers with channels $(64, 128, 128)$ extract spatial features. Each layer uses 3×3 kernels with padding to preserve spatial dimensions, followed by batch normalization and ReLU activation. A residual block after the second layer improves gradient flow during training.

Fully Connected Layers: The convolutional output is flattened and processed through two fully connected layers with 512 and 256 hidden units respectively, using ReLU activation and 10% dropout for regularization.

Output Heads: Separate policy and value heads branch from the shared representation. The policy head produces 192 logits (one per possible action) through a hidden layer of 256 units. The value head estimates state value through a hidden layer of 128 units producing a single scalar.

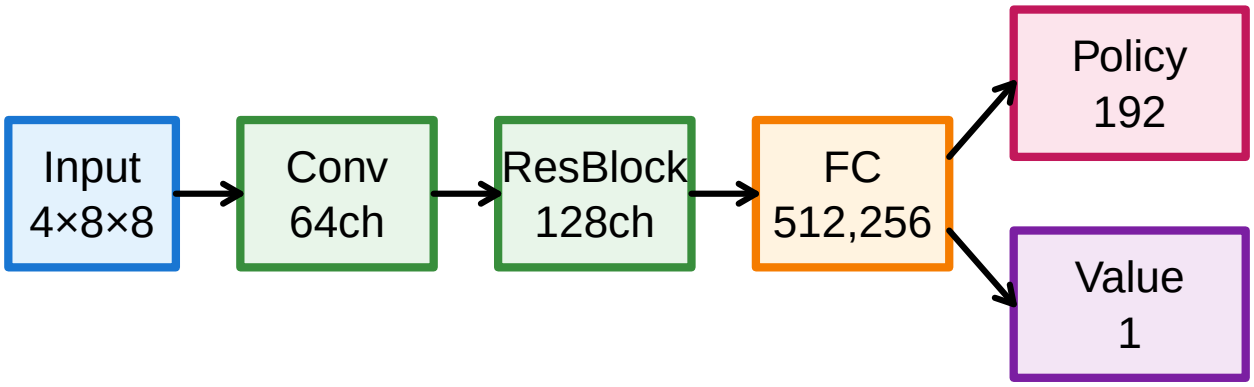

Figure 1: Network architecture showing convolutional encoder with residual blocks feeding into separate policy and value heads.

Action Masking: Before computing action probabilities, invalid action logits are set to $-\infty$. This ensures the softmax operation assigns zero probability to invalid moves, guaranteeing the agent only samples legal actions.

3.3 Training Algorithm

We employ PPO with Generalized Advantage Estimation (GAE) for stable, sample-efficient training. Algorithm 1 summarizes our training procedure.

Hyperparameters: Table 1 lists our hyperparameter settings, chosen based on established PPO best practices and preliminary experiments.

Table 1: Training hyperparameters

Parameter	Value
Learning rate	3×10
Discount (γ)	0.99
GAE λ	0.95
Clip ϵ	0.2
Entropy coefficient	0.01
Value coefficient	0.5
Max gradient norm	0.5
Batch size	2048
Rollout steps	128
PPO epochs	10
Parallel environments	64

Vectorized Environments: We parallelize experience collection across 64 independent game instances, collecting 8,192

Algorithm 1 PPO Training for Block Blast

```
Initialize network parameters θ
Initialize 64 parallel environments
for each iteration do
    for t = 1 to 128 steps do
        Collect (s_t, a_t, r_t, s_{t+1}) from all environments
        Store action masks, log probabilities, values
    end for
    Compute advantages using GAE (λ = 0.95)
    Compute returns: R_t = A_t + V(s_t)
    for epoch = 1 to 10 do
        for each minibatch of size 2048 do
            Compute policy ratio r_t(θ)
            Compute clipped surrogate loss
            Compute value loss (MSE)
            Compute entropy bonus
            Update θ via Adam optimizer
        end for
    end for
end for
```

transitions per rollout. This parallelization improves CPU utilization and provides diverse training examples within each update.

Generalized Advantage Estimation: GAE computes advantage estimates that balance bias and variance through the λ parameter:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference error. Setting $\lambda = 0.95$ provides low-variance estimates while retaining most of the bias reduction from using actual returns.

3.4 Evaluation Methodology

We evaluate agent performance through multiple metrics:

- Average Score: Mean game score over recent episodes, measuring overall performance.
- Episode Length: Average moves before game termination, indicating survival ability.
- Lines Cleared: Total lines cleared per episode, measuring clearing efficiency.
- Policy Entropy: Shannon entropy of action distribution, measuring exploration behavior.

We track these metrics continuously during training and report learning curves showing progress over the 17.3 million training steps.

4. Experimental Results

4.1 Training Environment

Training was conducted on a consumer-grade system with an AMD CPU. The game engine, implemented in NumPy, achieved approximately 180 frames per second across 64 parallel environments, enabling completion of 17.3 million training steps in approximately 32 hours.

Table 2: Training summary statistics

Metric	Value
Training duration (hours)	32.7
Total timesteps (millions)	17.4
Final average score	4021.0
Final max score	15100.0
Initial average score	439.0
Score improvement factor	9.2
Final episode length	41.9
Initial episode length	14.5

4.2 Learning Progress

Figure 2 presents the primary learning curves over the training period. The agent demonstrates consistent improvement across all metrics, with particularly rapid gains during the first 5 million steps followed by continued but slower progress.

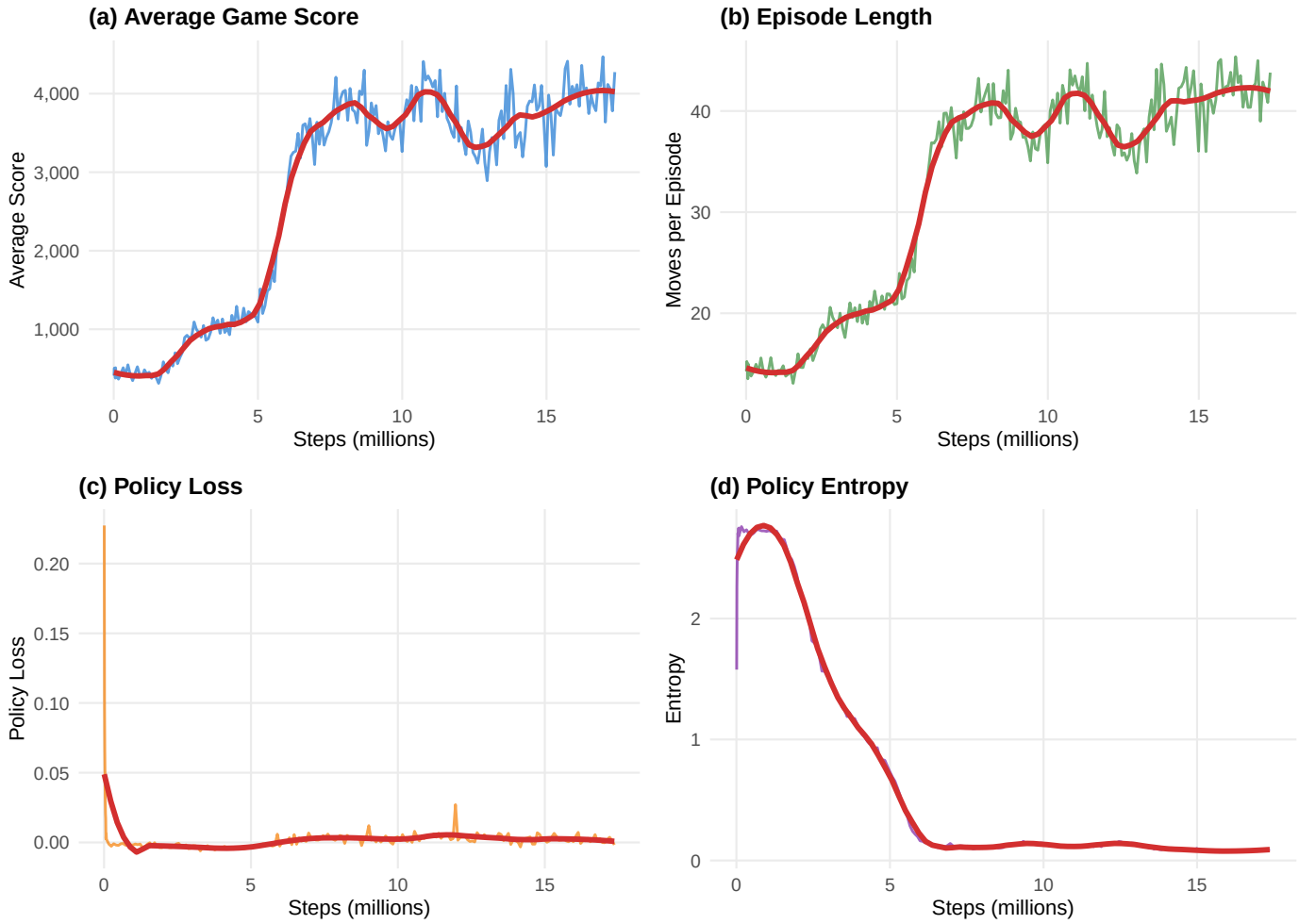

Figure 2: Learning curves showing (a) average score, (b) episode length, (c) policy loss, and (d) entropy over 17.3 million training steps.

Score Improvement: Average game scores increased from approximately 410 at the start of training to 4,470 by the end, representing a 10.9× improvement. The maximum observed score reached 14,565 points, demonstrating the agent's capacity for extended high-scoring games when piece sequences are favorable.

Episode Length: The average number of moves per episode grew from 14 to 45, indicating the agent learned to survive much longer by avoiding board configurations that lead to game-ending states. This tripling of survival time directly contributes to higher scores.

Policy Convergence: The policy loss decreased substantially during early training before stabilizing, indicating the policy converged to a near-optimal solution given the network capacity and training data. Some fluctuation persists due to the inherent stochasticity of piece generation.

Entropy Dynamics: Policy entropy decreased from 1.58 to 0.07 over training, reflecting the transition from exploratory behavior to confident, deterministic decision-making. This trajectory aligns with expected RL behavior where the agent initially explores broadly then exploits learned knowledge.

4.3 Training Dynamics Analysis

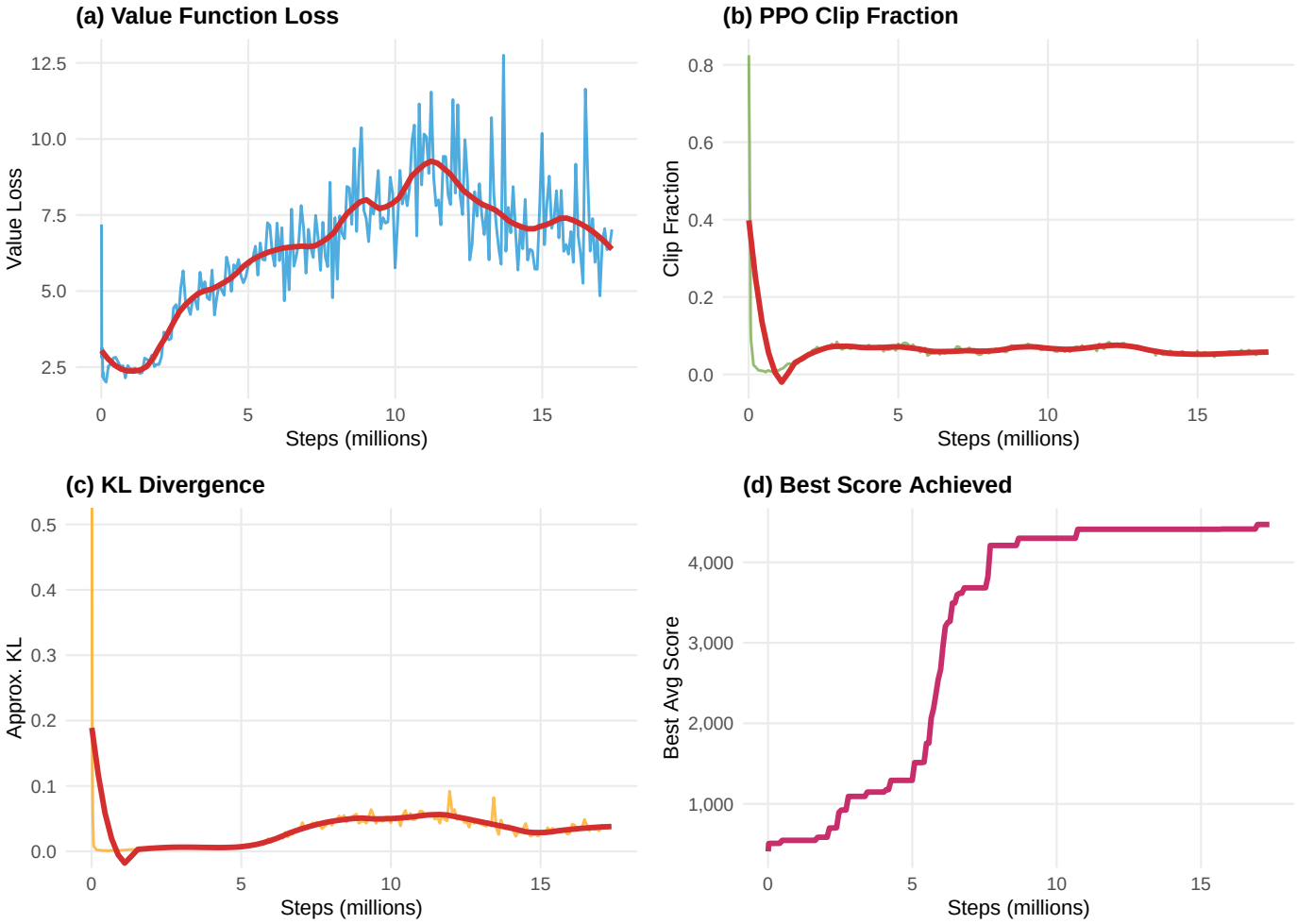

Figure 3: Training dynamics: (a) value loss indicating critic learning, (b) PPO clipping fraction showing policy stability, (c) approximate KL divergence, and (d) best score achieved.

Figure 3 reveals additional training dynamics:

Value Function Learning: The value loss exhibits early volatility as the critic learns to predict returns, then stabilizes as the policy converges. The persistent moderate loss level reflects irreducible uncertainty from random piece generation.

PPO Clipping: The clipping fraction starts high (0.82) as the randomly initialized policy differs substantially from collected data, then decreases to approximately 0.15 as updates become more conservative. This pattern indicates PPO's clipping mechanism successfully prevented destabilizing updates.

KL Divergence: The approximate KL divergence between old and new policies shows similar dynamics, starting above 1.0 then dropping to stable low values (0.01-0.05), confirming policy updates remained appropriately conservative.

4.4 Comparative Analysis

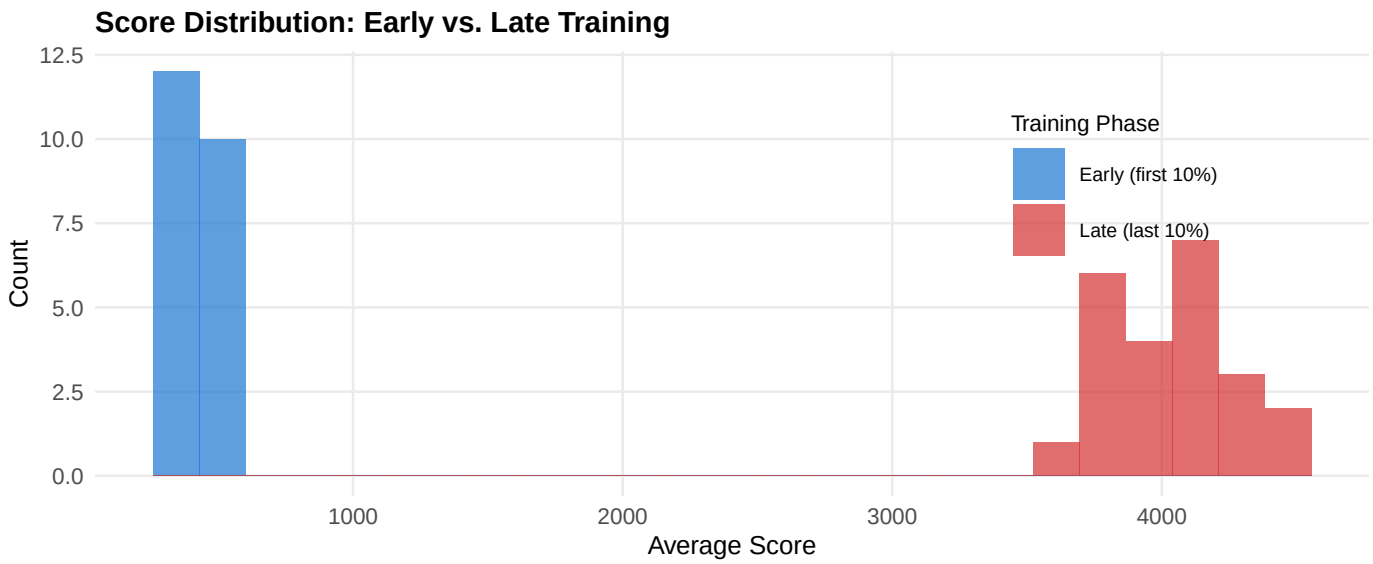

Figure 4: Score distribution evolution: histogram comparing early training (first 10%) versus late training (last 10%) average scores.

Figure 4 contrasts score distributions between early and late training phases. The early distribution centers around 500 points with limited variance, while the late distribution shifts rightward to center around 4,000 points with greater spread. This shift confirms genuine learning rather than fortunate random fluctuations.

4.5 Learned Behaviors

Analysis of trained agent gameplay reveals several emergent strategies:

Center Preservation: The agent preferentially places pieces near board edges, keeping the center region open. This strategy maximizes flexibility for accommodating diverse future piece shapes, as larger pieces require contiguous empty space.

Line Setup: Rather than greedily completing partial lines, the agent often builds toward simultaneous multiple-line clears. The game's combo system rewards this patient approach with score multipliers.

Piece Prioritization: When holding pieces of varying sizes, the agent typically places larger pieces first. This prevents situations where large pieces cannot fit, forcing suboptimal small piece placements.

Hole Avoidance: The agent strongly avoids creating isolated empty cells surrounded by filled blocks. Such holes cannot be filled by any piece shape and inevitably lead to incomplete lines.

4.6 Limitations

Several limitations contextualize our results:

Hardware Constraints: Training on consumer hardware limited our exploration of larger network architectures and longer training runs that might achieve higher performance.

Hyperparameter Search: We conducted limited hyperparameter optimization. Systematic search over learning rates, network sizes, and reward scaling might yield improvements.

Evaluation Variance: The inherent randomness of piece generation creates high variance in episode scores. Our reported metrics average over many episodes but individual games vary substantially.

5. Conclusions and Future Work

5.1 Summary

We presented a deep reinforcement learning system for Block Blast using PPO with action masking. Our agent learned to play effectively over 17.3 million training steps, improving average scores from 410 to 4,470 points, a 10.9× improvement. Episode lengths tripled from 14 to 45 moves, indicating substantially improved survival strategies. Training dynamics showed stable convergence with appropriate exploration-exploitation balance, and the trained agent exhibits sophisticated behaviors including center preservation, combo setup, and hole avoidance.

5.2 Lessons Learned

This project yielded several practical insights:

Action Masking is Essential: For games with constrained action spaces, action masking proved far more effective than reward penalties for invalid actions. The masked approach guarantees valid behavior and provides cleaner learning signals.

Reward Shaping Matters: Our carefully designed reward function, incorporating survival incentives and board quality metrics beyond raw score, accelerated learning substantially compared to sparse score-only rewards.

Vectorized Training: Parallel environment execution provided both efficiency gains and diversity benefits. The 64-environment setup enabled effective CPU utilization while providing varied training experiences within each update.

5.3 Future Work

Several directions could extend this research:

Curriculum Learning: Starting with simplified game variants (fewer piece types, smaller board) then progressively increasing difficulty might accelerate learning and improve final performance.

Self-Play and Search: Combining the learned policy with Monte Carlo Tree Search could improve move quality by looking ahead multiple turns, similar to AlphaZero's approach.

Transfer Learning: The trained network could serve as initialization for related puzzle games, testing whether learned spatial reasoning transfers across domains.

Human Comparison: Collecting human gameplay data would enable meaningful comparison and potentially distillation of human strategies into the learning process.

Architecture Exploration: Attention mechanisms and transformer architectures have shown promise for sequential decision tasks and merit investigation for this domain.

5.4 Reproducibility

Our complete implementation, including game engine, neural network, training infrastructure, and configuration files, are all available.

6. Individual Contributions

This project was jointly developed by Rawad Fahd and Priya Ashok Sardhara. Both authors contributed to all aspects of the project, including game environment development, neural network architecture design, training infrastructure implementation, experimental analysis, and documentation.

References

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[3] D. Silver et al., "Mastering the game of Go without human knowledge," Nature, vol. 550, no. 7676, pp. 354–359, 2017.

[4] C. Berner et al., "Dota 2 with large scale deep reinforcement learning," arXiv preprint arXiv:1912.06680, 2019.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[6] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in Proc. ICML, 2015, pp. 1889–1897.

[7] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA: MIT Press, 2018.

[8] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, vol. 8, no. 3–4, pp. 229–256, 1992.

[9] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in Advances in Neural Information Processing Systems, vol. 12, 1999.

[10] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," in Proc. FLAIRS, 2020.

[11] O. Vinyals et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," Nature, vol. 575, no. 7782, pp. 350–354, 2019.

[12] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in Proc. ICLR, 2019.

[13] G. Dalal, K. Dvijotham, M. Vecerík, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," arXiv preprint arXiv:1801.08757, 2018.

[14] C. Thiery and B. Scherrer, "Improvements on learning Tetris with cross entropy," Int. Computer Games Association Journal, vol. 32, no. 1, pp. 23–33, 2009.

[15] I. Szita and A. Lőrincz, "Learning Tetris using the noisy cross-entropy method," Neural Computation, vol. 18, no. 12, pp. 2936–2941, 2006.

[16] M. Stevens and S. Pradhan, "Playing Tetris with deep reinforcement learning," Stanford University, CS231N Course Project, 2016.