Extended Abstract

Motivation Recent advances in reinforcement learning allow for agents to be capable of playing through many different games on levels far surpassing human players. While these agents can perform strongly in many different single-player games, not many agents have been adapted to multi-player environments.

This highlights an important problem in reinforcement learning. Due to dynamic environments and rewards factoring in other agents, adapting the same reward functions in single-player games directly to multi-player environments will not be as effective. With sparse environments yielding prolonged reward signals, we want to determine a way to effectively learn in multi-agent environments without explicitly defined rewards, which we plan on doing through multiplayer Tetris.

Method Previous successful attempts at model-free RL, notably Deepmind's AlphaZero with chess, Shogi, and Go Silver et al. (2017) have performed exceedingly well. However, these frameworks depend on well-shaped rewards for zero sum games. In contrast, our work focuses on competitive multi-agent environments where our reward function isn't explicitly known, but rather learned starting with baseline expert trajectories. Also, by looking at multi-agent training in the context of multiplayer Tetris, we hope to see how our agent adapts to competitive environments with individual game state spaces, influenced by other agents.

In our paper, we propose the implementation of game-playing agents for multiplayer Tetris, capable of competing against other agents or human players, where each player has an individual board. Our implementation consists of converting the board state into discernible features, through which we utilized a pre-trained single player model. We then sampled trajectories from the single player agent, which we would use as expert trajectories for Multi-Agent Adversarial Inverse Reinforcement Learning (MA-AIRL) Yu et al. (2019a), in order to distinguish dense rewards through interactions with other agents. We had several different agents play many games of Multiplayer Tetris, through which we updated a shared reward function with individual policies over the course of 5 million steps, starting with an expert replay buffer of 50 trajectories, each with 150 step long episodes (150 pieces placed).

We evaluate our MA-AIRL framework against behavioral cloning from the single player agent, measuring average win rate, We also will look at discriminator accuracy and loss, in order to evaluate the effectiveness of training our MA-AIRL model.

Results and Discussion Evaluating our MA-AIRL agent, we observed higher win rate with the MA-AIRL trained agent relative to the pre-trained single-player agent, with the MA-AIRL agent winning $\approx 65\%$ of the time. Our discriminator network, correctly classified about $50 - 55\%$ of the time, and discriminator loss decreased as our training runs went on. However, as our agent trained further, our reward function peaked at around 1 million steps, and began linearly decreasing toward the end of training.

From our results, we emphasize several main observations. The $50 - 55\%$ classification rate of the discriminator network indicates that despite improvement over training, our discriminator struggles to distinguish between expert trajectories and MA-AIRL trained policy rollouts. Also, the MA-AIRL agent winning $\approx 65\%$ of the time relative to an expected 50% baseline shows that our MA-AIRL agent performs better relative to the baseline pre-trained single-player model. On the other hand, the decreasing reward on our MA-AIRL agent following 1 million steps might be due to discriminator not being updated frequently enough or due to our reward signal gradually drifting.

Conclusion In our work, we adapted an existing multiplayer Tetris game into a multi-agent environment, and adapted MA-AIRL to multiplayer Tetris. For other experiments, we also explored meta-RL applications in single-player Tetris, along with different variations of ϵ-greedy exploration in the DQN network, like top-k action sampling. Our results for MA-AIRL look promising, as we observe noticeably better performance of MA-AIRL agents against pre-trained agents, and although our discriminator can still be improved upon, is still unable to distinguish between expert trajectories and generated MA-AIRL trajectories. Future directions of our project include generalization to other multiplayer games, or adapting our framework to real life applications, such as the stock market.

Reinforcement Learning in Tetris with Multi-Agent Systems

Eric He
Department of Computer Science
Stanford University
eriche26@stanford.edu

Deren Ji
Department of Computer Science
Stanford University
derenji@stanford.edu

Karl Songcuan
Department of Computer Science
Stanford University
ksong15@stanford.edu

Abstract

Recent advances in reinforcement learning allow for agents to be capable of playing through many different games on levels far surpassing human players. While these agents can perform strongly in many different single-player games, not many agents have been adapted to multi-player environments. We adapted a multi-agent adversarial inverse reinforcement learning (MA-AIRL)Yu et al. (2019b) framework to multiplayer Tetris, and achieved $\approx 65\%$ win rates with MA-AIRL trained agents against baseline pre-trained DQN models, proving our agent's performance to improve relative. Future work can include generalization of this framework to other real life tasks, such as stock market investing or algorithmic trading, although these applications lie past the current scope of our project. Our project ultimately integrates MA-AIRL for training Multiplayer Tetris agents.

1 Introduction

Multiplayer Tetris is a variation of the classic arcade game Tetris, adapted for competition between two players. Several adaptations of multiplayer Tetris (jstris, tetr.io, Tetris 99) have been popularized online, where players compete to clear lines, subsequently sending lines of "garbage" to the other player.

This introduces a unique challenge not seen in regular Tetris, as the ability to send opponents lines of "garbage" shifts the game's prime focus from sole survival (clearing lines faster as levels increase), to balancing attack and defense. This problem is important, as this problem can also be reframed toward the context of many other real world tasks, such as trading stocks or algorithmic trading.

Current methods such as DQN have shown strong performance in classic Tetris, but struggle with respect to adversarial multi-agent environments, due to sparse rewards and confounding variables from other agents within the multi-player environment.

Prior successful implementations of multi-agent RL in games, like Google Deepmind's AlphaZero have primarily focused on shared multi-agent environments, but none have focused on individual state spaces influenced by other agents, like in multiplayer Tetris. In the context of multiplayer Tetris, effective and representative reward functions cant be trivially determined, and rewards are sparse, making learning difficult.

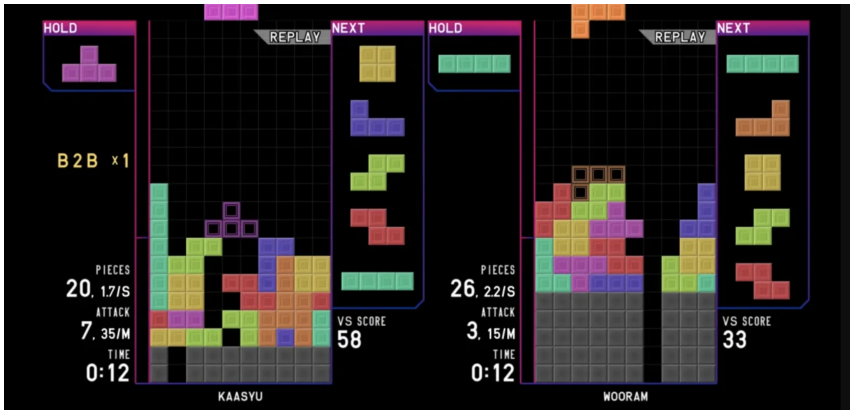

Figure 1: TETR.IO gameplay, garbage lines in gray

To approach this problem, we adapted the previously explored MA-AIRL framework Yu et al. (2019b) to an existing multiplayer Tetris codebase dicksontan2618 (2025). Our approach to MA-AIRL begins with utilizing a pre-trained single-player DQN model. Stevens and Pradhan (2016). From this model, we then sample expert trajectories which we use as a baseline for MA-AIRL. We placed several agents against each other,

After training, we observed our MA-AIRL discriminator correctly labels MA-AIRL generated trajectories $50 - 55\%$ of the time. Comparing the performance of our MA-AIRL model against the pre-trained single-player DQN model, we observed our model wins 65% of the time, relative to the baseline 52%.

In this project, our main contributions are converting an existing multiplayer Tetris game into a multi-agent environment. We also performed different experiments with singleplayer Tetris models, ranging from top-k sampling in place of $\epsilon-$greedy, to applications in meta-RL.

2 Related Work

Although RL in multiplayer Tetris remains underexplored, insights from single-player agents and recent MA-AIRL methods guide our approach to implementing multiplayer Tetris game-playing agents.

Single-Player Heuristics & RL The *Clear Board* heuristic (1990s) uses four weighted features—lines cleared, column heights, holes, bumpiness—to clear tens of thousands of lines without learning Lundgaard and McKee (1998). Lundgaard & McKee (1998) added six parameters (current/next piece, valley metrics, buried-hole bins) to train Q-learning and neural agents: a greedy Q-learner matched Clear Board, a high-level network pre-trained on heuristic labels briefly outperformed on score but only achieved two-thirds the lines long term, while a low-level network underperformed dramatically Lundgaard and McKee (1998). Stevens & Pradhan (2016) compared rewards based on raw score versus heuristic metrics and grouped versus atomic actions: heuristic-reward models with grouped actions reached just 10% of Clear Board performance, highlighting local-optima traps in naive shaping Stevens and Pradhan (2016).

Deep RL Baselines End-to-end DQN, A3C/PPO, and Dreamer flatten the 20×10 grid into MLPs but plateau at a few hundred lines due to sparse, delayed feedback and exploration deadlocks Mnih and et al. (2016); Schulman and et al. (2017); Hafner and et al. (2020).

Multi-Agent Strategies

- **MADDPG** centralizes critics over all boards while actors remain local, stabilizing discrete competitive updates in Tetris-like exchanges of garbage lines Lowe and et al. (2017).
- **On-Policy MARL (MAPPO)** rapid policy updates track evolving opponent strategies in real time, suited to asynchronous Tetris match dynamics Iqbal and Sha (2020).

- **QMIX** factorizes joint Q-values into per-player utilities, learning coordinated attack/defense chains via a monotonic mixing network Rashid and et al. (2020).
- **Comm-Aware Architectures** enable agents to broadcast planned garbage sends or vulnerability alerts, improving responsiveness in partially observed play Iqbal and Sha (2020).

Adversarial IRL MA-AIRL automatically infers dense, context-sensitive reward functions from expert play, eliminating the need to hand-craft complex multi-agent reward signals that balance offense, defense, and survival Yu et al. (2019a). By training a discriminator to distinguish expert trajectories from policy rollouts, it captures subtle strategic incentives such as timely garbage sends and opponent baiting that static reward designs often miss Yu et al. (2019a); Gruver et al. (2020). Its adversarial training loop naturally adapts to non-stationary opponent behavior by continuously refining the reward model in response to emerging strategies, offering greater robustness to evolving game dynamics Yu et al. (2019a). Unlike purely policy-search methods or value-decomposition frameworks like QMIX Rashid and et al. (2020), MA-AIRL embeds reward inference within the learning process, providing richer feedback signals that accelerate exploration and convergence. Latent-variable extensions of MA-AIRL handle noisy or partial demonstrations by incorporating unobserved strategic factors into the discriminator, making the approach resilient to imperfect or incomplete expert data Gruver et al. (2020).

3 Methods

Game Environment Design

We extend the single-player `TetrisEnv` to a headless, vectorized multi-agent setting via `DummyVecEnv` and `VecNormalize`. Each agent controls an individual 20×10 board, receiving observations $s_t = (\text{grid}_t, \text{next piece}_t, \text{hold piece}_t)$. Competitive dynamics are introduced through garbage-line attacks: when an agent clears L lines, it appends $L-1$ bottom rows of garbage to opponents, creating a non-stationary and sparse reward landscape. The headless, vectorized framework enables parallel sampling across multiple environments, enhancing sample efficiency and decorrelating gradient estimates. Reward normalization via `VecNormalize` mitigates non-stationarity induced by adversarial garbage-line attacks, thus stabilizing policy updates under both sparse and dynamic reward signals.

State Representation

Raw observations are preprocessed by `preprocess_state` into a vector of dimension 207:

$$s = [\underbrace{g_{1,1}, \ldots, g_{20,10}}_{200},\ n,\ h,\ c,\ r,\ x,\ y,\ \text{canHold}],$$

where $g_{i,j} \in \{0,1,2\}$ indicates board occupancy, n, h are IDs of next and hold pieces, and $(c, r, x, y, \text{canHold})$ encode the current piece's shape, rotation, position, and hold availability. Explicitly separating spatial grid data and piece metadata allows convolutional layers to exploit local spatial correlations (e.g., line formations), while the MLP embedding captures discrete, high-level piece information. This decomposition yields a structured representation aligned with theoretical principles of feature factorization and hierarchical abstraction.

Multi-Agent Adversarial Inverse Reinforcement Learning (MA-AIRL)

Building on AIRL Yu et al. (2019b), we extend to multi-agent by sharing a discriminator and reward function across agents but training individual policies. A shaped reward network (`BasicShapedRewardNet`) computes:

$$D_\psi(s,a) = \sigma\big(r_\psi(s,a) + \gamma\, h_\psi(s') - h_\psi(s)\big),$$

where h_ψ is a potential function. The adversarial objective is

$$\min_\theta \max_\psi\ \mathbb{E}_{\tau\sim D_E}[\log D_\psi(s,a)] + \mathbb{E}_{\tau\sim\pi_\theta}[\log(1 - D_\psi(s,a))].$$

Our trajectory generators are PPO agents (`MlpPolicy`), trained under the learned reward. We perform a 100k-step warmup under initial rewards, then alternate discriminator and generator updates

for 5 million timesteps, with discriminator update rate $n_{disc} = 1$ per round and learning rates $\ell_{disc} = 10^{-6}, \ell_{PPO} = 10^{-4}$.

Adversarial IRL enables recovery of latent reward structures in settings with sparse or unmodeled dynamics. Potential-based shaping through h_ψ aims to guarantee policy invariance under reward transformations Ng et al. (1999). Sharing a central discriminator enforces consistent reward inference across competitive agents, while independent generators adapt to non-stationary opponents within a dense reward landscape.

Key AIRL Network Components

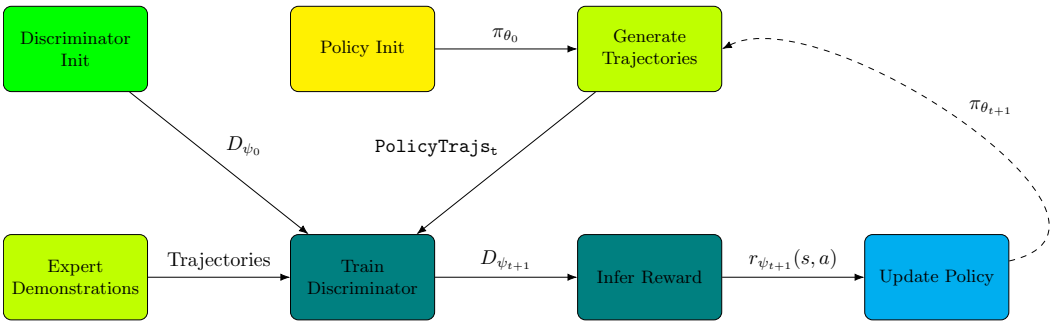

Figure 2: AIRL training cycle.

1. State Encoder

$$z_s = \mathrm{Flatten}\big(\sigma(\mathrm{Conv}_2(\sigma(\mathrm{Conv}_1(x))))\big) \in \mathbb{R}^{d_s},$$

The state encoder extracts spatial features from the 20×10 board. In AIRL, a rich z_s helps the discriminator and policy distinguish subtle board patterns (e.g. imminent game-over).

2. Action Embedder

$$z_a = W^a\, e_a + b^a \in \mathbb{R}^{d_a},$$

The action embedder maps each action to a dense vector. As a result, this allows the reward network to effectively learn which moves motivate expert behavior.

3. Reward Network

$$f_\theta(s,a) = (w_2^f)^\top \sigma\big(W_1^f\,[z_s; z_a] + b_1^f\big) + b_2^f,$$

The reward network takes in the state-action pair, returning a the reward value taking a specific action at a given state. As MA-AIRL is trained, f_θ adapts to adversarial dynamics between agents, associating higher rewards with strategic offensive and defensive actions (i.e. prioritizing moves like Tetrises more frequently).

4. Potential Function (Shaping)

$$\hat{r}_\theta(s,a,s') \;=\; f_\theta(s,a) + \gamma\, h_\theta(s') - h_\theta(s),$$

with

$$h_\theta(s) = (w_2^h)^\top \sigma(W_1^h\, z_s + b_1^h) + b_2^h.$$

This reward shaping guarantees that the learned reward is shaped for stable learning; the potential term h_θ functions to smooth sparse signals in the environment, accelerating training convergence.

5. Policy Network & Update

$$\pi_\phi(a \mid s) = \mathrm{softmax}\big(W_2^\pi\, \sigma(W_1^\pi\, \tilde{z}_s + b_1^\pi) + b_2^\pi\big),$$

$$\nabla_\phi J =_{\pi_\phi} \big[\nabla_\phi \log \pi_\phi(a \mid s)\, \hat{A}_{\hat{r}}(s,a)\big].$$

Our policy network learns decentralized policies using the shaped reward $\hat{r}_\theta$. We expect faster policy improvement compared to raw sparse rewards utilized in previous DQN implementations in classic Tetris.

6. Discriminator Update

$$\nabla_\theta \mathcal{L}_D = -\mathbb{E}_E L[\nabla_\theta \log D_\theta] - \mathbb{E}_\pi L[\nabla_\theta \log(1 - D_\theta)],$$

The discriminator trains the objective $D_\theta(s, a) \approx \frac{\exp(f_\theta)}{\exp(f_\theta)+\pi(a|s)}$, in order to distinguish between expert trajectories and MA-AIRL agent trajectories. We intend for our discriminator to show accurate reward signals to guide the policy toward optimal behavior..

Expected Performance in AIRL: These components work together to produce dense, context-aware rewards that (1) resolve exploration deadlocks by shaping rare line-clear events, (2) adapt in adversarial multi-agent matches via the discriminator, and (3) converge faster due to potential-based shaping. The result is strategic, robust multiplayer Tetris agents capable of offensive and defensive play.

4 Experimental Setup

4.1 Task

Our agent plays a simple version of multiplayer Tetris against other agents. This means that agents will send lines to each other during training, and aggressive play scoring many high line clears is preferred as agents place pieces at the same rate.

4.2 Baselines and Comparisons

As a baseline for evaluating MA-AIRL work, we compare the performance of our MA-AIRL agent to our initial pre-trained DQN model, by comparing each agent's performance against the same baseline DQN model.

4.3 Metrics

We evaluate the relative performance of our MA-AIRL agent through several different metrics in training and in evaluation. During training, we emphasize

1. Discriminator accuracy, which measures the ability of the classifier to correctly distinguish between sampled MA-AIRL agent trajectories and expert trajectories
2. Discriminator loss, by which lower values can imply more the formation of a more idealistic policy
3. Discriminator reward, which returns a value, taking an action given a state is.

During evaluation through games, we measure the relative performance of the MA-AIRL agent through

1. Win Rate, where we measure the win rate of each agent against a baseline pre-trained DQL agent.
2. Average lines cleared per game, which is a measure of how effectively the model performs against the baseline DQL agent on average.
3. Standard deviation of lines cleared per game, which is a measure of how consistent an agent's performance is, with lower standard deviation indicating more consistent performance.

With these metrics, we can determine the performance of our MA-AIRL network, both through the discriminator and through comparative performance to other baseline agents.

5 Results

5.1 Quantitative Evaluation

Table 1: Performance Comparison Against Baseline Pre-Trained DQN Model

Method	Win Rate	Avg Lines / Game	Std Lines / Game
Baseline DQN	0.52	35.5	4.3
MA-AIRL	0.65	52.5	5.6

In our evaluation, we compared the relative performance of the pre-trained DQN model against the MA-AIRL model, taken from a checkpoint at $1,024,000$ steps (where reward plateaus), by pitting each model against an instance of the pre-trained DQN model. As we can see in Figure 1, we observe that the MA-AIRL model wins 65% of the time, compared to the baseline's 52% winrate. We also observe that in each game, the baseline DQN model clears 35.5 lines per game with a standard deviation of 4.3 lines, while the MA-AIRL model clears an average of 52.5 lines per game with a standard deviation of 5.6 lines. From this, we see that our trained MA-AIRL model on average performs noticeably better compared to the baseline DQN.

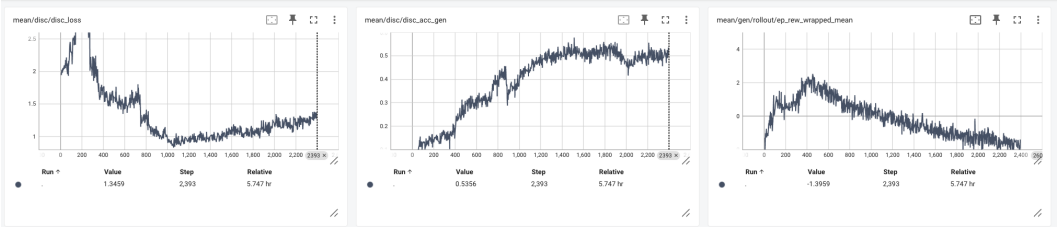

Figure 3: Discriminator loss generated classification accuracy, and reward
(each discriminator step = 2048 training steps

During our training, we observed that our discriminator loss gradually decreased over the course of the 5 million steps of training. From Figure 3, We also observed that the discriminator accuracy for trajectories sampled from our MA-AIRL agent plateaued at about $50-55\%$, indicating that our discriminator's ability to distinguish between expert trajectories and our agent trajectories decreased as training went on. However, we observed that our reward function plateaus at around $1,024,000$ steps, and begins to linearly decrease afterward. This trend might indicate that the learned reward function is becoming less informative as training continues. Since MA-AIRL's reward function is dependent on the discriminator, this trend can possibly be attributed to our discriminator not being updated frequently enough, or that small inefficiencies in the MA-AIRL agent policy compound over time.

5.2 Qualitative Analysis

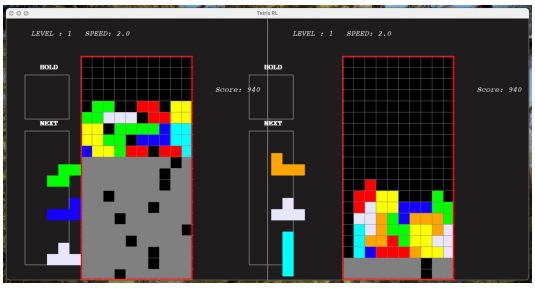

Figure 4: DQN Agent (left) vs. MA-AIRL trained agent (right)

Figure 4 depicts an instance of a game between the pre-trained DQN agent and the MA-AIRL trained agent, where we observe noticeably more "garbage" lines being sent to the DQN agent. This disparity in "garbage" lines indicates that the MA-AIRL agent clears more lines in a given number of timesteps compared to the DQN agent, which we see has only sent 1 to 2 lines of garbage. From this, we can see that our agent's performance improves over MA-AIRL training. The MA-AIRL trained agent is also optimized to clear Tetrises, which is much more valuable in sending garbage than the single and double line clears the greedy DQN agent scores.

However, towards the end of some episodes where the MA-AIRL agent loses, we sometimes see our agent making erratic moves, placing blocks in non-optimal places, subsequently losing the game only a few moves later.

6 Discussion

Every Tetris game is different. Our environment was set up such that each action is a final choice of piece's final position and rotation. This significantly simplifies our experimentation, as seen with our actor critic method. Our agent performs very poorly when the rewards are made even more sparse when compared when each action does not translate directly to signals from the board. Our simplified model was able to be trained with relatively smaller network, but is unable to account for more complex movements such as a tuck - an insertion beneath a hanging block in a column, or T-spin - spinning a T piece into a space with an overhang above. Because T-spins are as viable as, if not more viable than Tetrises at current top-level human play, it would be necessary to incorporate these important moves to a truly top-level agent that relies on more than piece placement speed.

7 Conclusion

By adapting MA-AIRL to multiplayer Tetris, we achieved a 13% higher winrate compared to behavioral cloning on baseline DQN models.

Through our experiments in both single-player and multi-player Tetris, we demonstrated that our methods generalize more with respect to vastly changing game states, performing better compared to baseline DQN performance.

Looking forward, our adapted multiplayer Tetris and MA-AIRL framework could be applied to many different areas, including outside video games. Applying this to other simulated forms of tasks could significantly improve performance on different real-life tasks, like stock market investing, or algorithmic trading. Additionally, Meta-RL applications in multiplayer video games could also potentially be applied to real life analogues, like social situations. These potential ideas building off of our current work would not only benefit many, but could potentially allow for more generalized agents, learning to perform in unseen environments, given experience in similar tasks an an ability to distinguish dense rewards.

8 Team Contributions

- **Eric He:** Implementing compatibility of multiplayer Tetris game with both single-player agent training on own, and with two agents competing)
  Implementing and refining MA-AIRL, processing of board and pieces into discernible features,
- **Karl Songcuan:** Implementing compatibility of multiplayer Tetris game with both single-player agent training on own, and with two agents competing)
  Implementing MA-AIRL, top-k sampling in singleplayer DQN, MA-AIRL logging metrics, implementing compatibility of singleplayer expert trajectories and MA-AIRL
- **Deren Ji:** Implemented game environment, gym wrapper, and RL utilities. Implementing compatibility of multiplayer Tetris game with both single-player agent training on own, and with two agents competing)
  Implementing DREAM variant application to multiplayer Tetris, refining singleplayer DQN.

Changes from Proposal These adjustments were necessary, as 1: adapting the multiplayer Tetris game to be compatible with both singleplayer and multiple agents took far more debugging due to the

codebase and model implementation being so different. After our compatibility issues were fixed, we decided start by looking mostly into different ways to improve singleplayer models, initially starting with top K learning and DREAM variant implementation. We also experimented with AC networks and timestep based reward, but that ended up not working as well. Afterward, we shifted toward MA-AIRL implementation, which we'd encountered many bugs, specifically this one instance where episodes would keep prematurely ending.

References

F. Agostinelli, A. Wang, et al. 2019. Learning to Play Tetris with Depth-First Search and Policy Network. *arXiv preprint arXiv:1905.00134* (2019). https://arxiv.org/abs/1905.00134

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47 (June 2013), 253–279. https://doi.org/10.1613/jair.3912

Rémi Coulom. 2007. *Computing Monte Carlo Tree Search*. Technical Report. HAL Archives Ouvertes. https://hal.archives-ouvertes.fr/hal-00197380

dicksontan2618. 2025. local-multiplayer-tetris: A simple local-machine multiplayer Tetris game. https://github.com/dicksontan2618/local-multiplayer-tetris. GitHub repository, MIT License.

G. Dulac-Arnold, T. Hester, S. Schmitt, et al. 2021. Challenges of Real-World Reinforcement Learning. *arXiv preprint arXiv:2107.04743* (2021). https://arxiv.org/abs/2107.04743

N. Gruver, J. Song, M. Kochenderfer, and S. Ermon. 2020. Multi-Agent Adversarial IRL with Latent Variables. *AAMAS* (2020). https://arxiv.org/abs/2006.04495

D. Hafner and et al. 2020. Dream to Control: Learning Behaviors by Latent Imagination. *arXiv preprint arXiv:1912.01603* (2020). arXiv:1912.01603 https://arxiv.org/abs/1912.01603

D. Hafner, T. Lillicrap, J. Fischer, et al. 2020. Mastering Atari with Discrete World Models. *arXiv preprint arXiv:2010.02193* (2020). arXiv:2010.02193 https://arxiv.org/abs/2010.02193

S. Iqbal and F. Sha. 2020. Actor-Critic with Communication: Multi-Agent PPO in Cooperative Settings. *arXiv preprint arXiv:2003.07959* (2020). arXiv:2003.07959 https://arxiv.org/abs/2003.07959

R. Lowe and et al. 2017. Multi-Agent Deep Deterministic Policy Gradient. *NeurIPS* (2017). https://arxiv.org/abs/1706.02275

N. Lundgaard and B. McKee. 1998. Reinforcement learning and neural networks for Tetris. *Proc. ICML '98* (1998). https://mcgovern-fagg.org/amy_html/courses/cs5033_fall2007/Lundgaard_McKee.pdf

V. Mnih and et al. 2016. Asynchronous Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1602.01783* (2016). arXiv:1602.01783 https://arxiv.org/abs/1602.01783

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. , 278–287 pages. https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/NgHaradaRussell-shaping-ICML1999.pdf

T. Rashid and et al. 2020. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent RL. *ICML* (2020). https://arxiv.org/abs/1906.06344

J. Schulman and et al. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017). arXiv:1707.06347 https://arxiv.org/abs/1707.06347

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359. arXiv:1712.01815 [cs.AI] https://arxiv.org/abs/1712.01815

K. O. Stanley and R. Miikkulainen. 2019. Tetris as a Testbed for Deep Reinforcement Learning. In *GECCO*. https://gecco-2019.sigevo.org

Matt Stevens and Sabeek Pradhan. 2016. Playing Tetris with Deep Reinforcement Learning. Stanford CS231n project report. https://cs231n.stanford.edu/reports/2016/pdfs/121_Report.pdf.

L. Yu, J. Song, and S. Ermon. 2019a. Multi-Agent Adversarial Inverse Reinforcement Learning. *ICML* (2019). https://arxiv.org/abs/1909.12844

Lantao Yu, Jiaming Song, and Stefano Ermon. 2019b. Multi-Agent Adversarial Inverse Reinforcement Learning. arXiv:1907.13220 [cs.LG] https://arxiv.org/abs/1907.13220

A Additional Experiments

A.1 Actor Critic (AC)

Unlike line–only reward schemes (where reward = number of lines cleared), we adopt the environment's standard reward signal: the default scoring mechanism of Tetris, where clearing a single line yields a reward of 2, two lines yields 5, three lines yields 8, and a Tetris (four lines) yields 12. This standard reward closely aligns with historical benchmarks and fosters a balanced objective between clearing lines and prolonging gameplay.

Formally, at time step t, if the agent clears $k_t \in \{0, 1, 2, 3, 4\}$ lines, the reward is:

$$w(l) = \begin{cases} 0, & l = 0, \\ 3, & l = 1, \\ 5, & l = 2, \\ 8, & l = 3, \\ 12, & l = 4, \end{cases} \qquad R_{\text{base},t} = w\big(l_t\big)\,(\ell + 1)$$

$$R_t = \begin{cases} R_{\text{base},t} \;-\; 100, & \text{if game over,} \\ R_{\text{base},t} \;+\; 10\,l_{t-1} \;-\; \frac{1}{2}\big(h_t - h_{t-1}\big) \;-\; \frac{1}{2}\big(H_t - H_{t-1}\big) \;-\; \frac{1}{2}\big(B_t - B_{t-1}\big), & \text{otherwise,} \end{cases}$$

where

$$h_t = \sum_{c=1}^{10} \text{height}_{c,t}, \qquad H_t = \text{number of holes at time } t,$$

$$B_t = \sum_{i=1}^{9} \big|\text{height}_{i,t} - \text{height}_{i+1,t}\big|, \qquad \ell = \text{current game level},$$

$$l_t = \text{lines cleared at step } t.$$

This choice replicates the canonical reward conventions frequently used in RL tests on Tetris Bellemare et al. (2013).

A.1.1 Actor–Critic Network Architecture

Our baseline network implements an actor critic algorithm in Tetris. The detailed architecture is as follows:

Table 2: Shared Backbone and Heads

Component	**Layer Dimensions**	
Input	$\mathbb{R}^{207}$	Flattened board + piece encoding
Shared Hidden Layer 1	$207 \to 512$	Fully connected, ReLU
Shared Hidden Layer 2	$512 \to 256$	Fully connected, ReLU + Dropout (p=0.2)
Shared Hidden Layer 3	$256 \to 128$	Fully connected, ReLU
Actor Head	$128 \to 8$	Policy logits, followed by softmax
Critic Head	$128 \to 1$	State-value estimate $V(s)$

Key statistics:

- Total parameters: 407,209.
- Input dimension: 207.
- Output dimension: 8 discrete actions.
- Dropout rate: 0.2 on second layer of shared backbone.

- Entropy regularization coefficient: 0.01.

Action selection follows an ε–greedy schedule during early training (initial $\varepsilon = 1.0$ decaying to 0.1 over 100k steps). The critic is trained with mean–squared error on the one–step Bellman target, and the actor is optimized by the advantage estimate:

$$A(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{1}$$

with discount factor $\gamma = 0.99$.

Despite the theoretical appeal of actor–critic methods, detailed studies applying them to full–sized Tetris with discrete action sets are scarce. The primary gaps we aim to address are:

1. *Representation learning for high–dimensional board states*: Most prior actor–critic studies operate on low–dimensional features or small board variants. Our baseline evaluates a deep shared backbone on the standard 20×10 grid.
2. *Standard RL pipeline evaluation*: By deploying actor–critic with standard hyperparameters and reward signals, we quantify the empirical performance floor on which to gauge more advanced methods.
3. *Exploration vs. exploitation trade–off*: Tetris exhibits sparse high rewards (Tetris clears) and many suboptimal local actions. Incorporating entropy bonuses and ε–greedy schedules allows us to examine how simple exploration heuristics fare in this domain.

This baseline thus serves as an ablation test to isolate the value of network depth, dropout, and standard reward design, without introducing auxiliary curiosity or model–based planning. By isolating components (dropout, shared backbone, advantage estimation), we identify which modifications yield marginal gains. The negligible progress on the primary game objective (lines cleared) suggests that the vanilla actor–critic is insufficient for full–sized Tetris, likely due to:

- *Sparse, delayed rewards*: Clearing lines is infrequent compared to piece placements, leading to high variance in advantage estimates.
- *Large action–state space*: The combinatorial board configurations demand extensive exploration beyond random exploration schedules.
- *Lack of model–based planning*: Reactive policies cannot anticipate multi–step line–clearing opportunities without explicit lookahead.

A.2 DQN

Model Setup

We treat each Tetris board state s_t as a vector

$$s_t = \big[b_t,\, c_t,\, n_t\big] \in \mathbb{R}^{212},$$

where $b_t \in \{0,1\}^{200}$ is the flattened 20×10 board, $c_t \in \mathbb{R}^6$ encodes current piece type/rotation/position, and $n_t \in \mathbb{R}^6$ the next piece features. The action set is

$$\mathcal{A} = \{(x, y, r) \mid x \in [0, 9],\, y \in [0, 19],\, r \in \{0, 1, 2, 3\}\},$$

of size $|\mathcal{A}| = 800$, with index

$$a = y \cdot 40 + x \cdot 4 + r.$$

Network SetUp

Q-Network. We approximate the state–action value by a three-layer MLP:

$$Q(s_t, a; \theta) = \mathrm{MLP}_\theta(s_t)[a].$$

Bellman Update. Given transition (s_t, a_t, r_t, s_{t+1}), minimize

$$\mathcal{L}(\theta) = \mathbb{E}\big[(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta))^2\big],$$

where θ^- are the frozen target-network parameters.

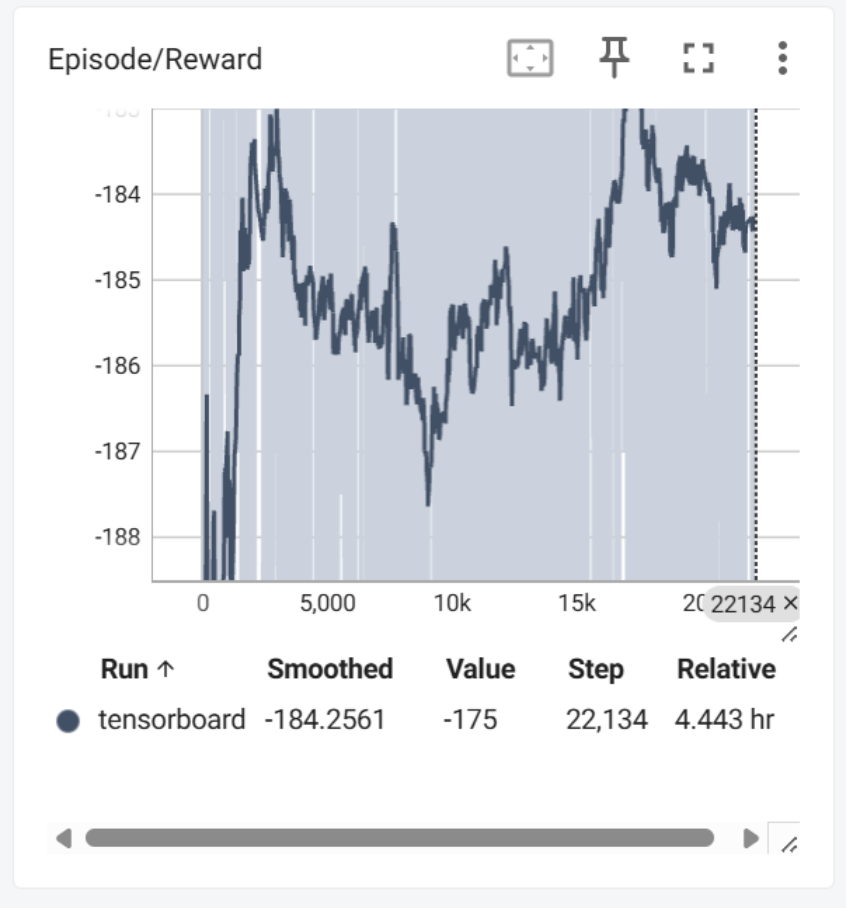

Figure 5: Actor critic smoothed episode reward over training.

Reward Model. We define the per-step reward as

$$r_t = \begin{cases} 2, & \text{if 1 line cleared at step } t, \\ 5, & \text{if 2 lines cleared,} \\ 8, & \text{if 3 lines cleared,} \\ 12, & \text{if 4 lines cleared (Tetris),} \\ 0, & \text{otherwise.} \end{cases}$$

ϵ-Greedy Policy. At step t, choose

$$a_t = \begin{cases} \text{random} \in \mathcal{A}, & \text{w.p. } \epsilon_t, \\ \arg\max_a Q(s_t, a; \theta), & \text{w.p. } 1 - \epsilon_t, \end{cases}$$

with ϵ_t annealed from 0.9 to 0.05 over 80,000 steps.

Discussion

This locked-placement DQN simplifies placement by treating each of the 800 possible drop locations and orientations as atomic actions, reducing credit-assignment complexity and enabling single-step policy decisions with next-piece lookahead Agostinelli et al. (2019); Stanley and Miikkulainen (2019). However, the vast action space drastically impedes exploration: random sampling under an ϵ-greedy schedule rarely visits high-reward placements, leading to sparse and noisy Q-value updates Dulac-Arnold et al. (2021). Combined with rapid ϵ decay, dropout noise, and absence of prioritized or multi-step replay, learning quickly stalls on suboptimal policies. In contrast, primitive-action DQNs

and model-based planners (e.g., MCTS, Dreamer) balance granularity and lookahead more effectively Coulom (2007); Hafner et al. (2020). Our team was restricted by our computational resources to effectively train 4M+ parameters the network may require to capture full learning. Though we were able to observe peak at performance at $\epsilon = 0.8$ around 80000 episode and a gradual increase after the plateau at 0.5. This is evidence that our model would have performed better if give more episodes for high randomness exploration. The first peak indicates that the overly rapid epsilon decline to capture better placements for the model to fully learn. The later climb indicates our model's eventual readiness to learn at its given randomness.

Dreamer Architecture

World Model

- RepresentationModel: A VAE-style encoder mapping observation o_t to latent $z_t \sim \mathcal{N}(\mu_t, \sigma_t^2)$.
- DynamicsModel: Predicts next latent z_{t+1} from (z_t, a_t) via parameters $(\mu_{t+1}, \sigma_{t+1}^2)$.
- RewardModel: Estimates scalar reward r_t from (z_t, a_t).
- ContinueModel: Predicts continuation probability γ_t from z_t.
- Decoder: Reconstructs observation $\hat{o}_t$ from z_t for an auxiliary reconstruction loss.

Policy & Value

- Actor: MLP in latent space producing logits for the policy $\pi(a_t \mid z_t)$.
- **Critic:** MLP in latent space predicting state value $V(z_t)$.

Training Procedure

1. Phase 1 (Pretraining): Collect random rollouts; train the world model components (representation, dynamics, reward, continue, decoder).
2. Phase 2 (Imagination + RL): Alternate
    - real-data world-model updates, and
    - policy/value updates using imagined rollouts in latent space (with λ-returns and an entropy bonus).

DREAM suffers similar limitation as actor critic, where exploration for optimal states is critical. It has shown characteristic slowly expanding block placement x coordinates,, which indicate expanding potential for block placements. It usually takes our actor critics 100k+ episodes to explore line clearing as the actor isn't initially motivated to take on edge actions such as left or right movement.