Superhuman AI for Generals.io Using Self-Play Reinforcement Learning

author names withheld

Under Review for NExT-Game 2026

Abstract

We present a superhuman AI agent for GENERALS.IO, a real-time strategy game that requires both long-horizon planning and short-term tactics under strong imperfect information. Trained for four days on $4\times$ NVIDIA H200 GPUs, our agent reaches #1 on the public 1v1 leaderboard of over 5,000 players. We reconsider the algorithmic machinery used in previous approaches and show that simple, general-purpose methods suffice: we drop behavior cloning, potential-based reward shaping, and population-based self-play in favor of a generic policy-gradient training loop. Alongside the agent we release a JAX-native simulator supporting 1v1, 2v2, and arbitrary-N free-for-all under a shared interface, enabling research on cooperative and general-sum multi-agent play.

1. Introduction

Games have driven much of modern deep reinforcement learning: Chess and Go [15, 17], Poker [4, 11], Stratego [13], StarCraft II [21], and Dota 2 [1]. These settings have motivated several classes of algorithms. Counterfactual regret minimization [23] is the standard for tabular imperfect-information play; neural fictitious self-play [6] pairs a best-response learner with a supervised average-policy network; population-based methods such as PSRO [8] maintain a growing set of policies and an oracle over them. As modern models and training pipelines grow ever more complex, it is worth asking how robust the backbone methods underlying them really are. Recent work shows that a policy gradient with a simple regularizer matches or surpasses these specialized alternatives on small imperfect-information games [14, 18] and reaches superhuman play in Stratego [19].

This work applies the same line to GENERALS.IO, a browser-based RTS with an active community of several thousand players and competitive bots. Unlike StarCraft II or Dota 2, it is light and simple enough to train on a single GPU while retaining a rich set of strategic and tactical challenges—resource allocation, opponent modeling, and deception under fog of war. One rule set covers three modes—1v1, 2v2, and free-for-all—spanning two-player zero-sum, team, and n-player general-sum play. Our case study identifies which practical tweaks carry the weight at this scale.

Contributions.

- **A JAX-native Generals.io environment**[1] that reaches tens of millions of frames per second on a single GPU and supports all three modes through a shared action and observation interface, enabling research in cooperative and general-sum multi-agent reinforcement learning.

1. The environment will be released upon acceptance.

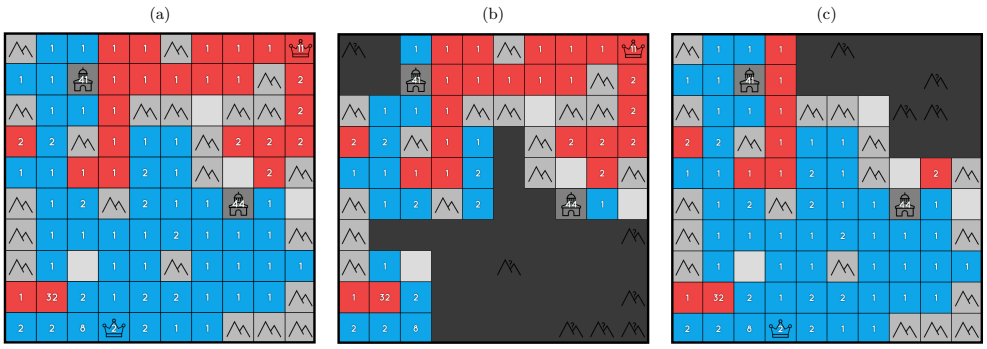

Figure 1: Three views of the same game state: (a) perfect-information view; (b) the red player's view; (c) the blue player's view. Crowns mark bases; numbers are unit counts.

- **A superhuman agent from a simple recipe**[2] (Sections 4, 5). A transformer policy trained end-to-end with sparse win/loss reward, a magnet regularizer [18], and an EMA of policy parameters reaches #1 on the public 1v1 leaderboard by a large margin and beats the top two individual humans head-to-head 13–7 and 15–5 respectively.

The remainder of the paper is organized as follows. Section 2 situates the work in the literature. Section 3 describes the game and the JAX environment. Section 4 describes the agent: network architecture, training objective, spawn-distance curriculum, parameter EMA, and top-advantage filtering. Section 5 reports the human-leaderboard result and head-to-head scores against the top two humans and the strongest non-learning bot. Section 6 isolates the effect of reward shaping and the EMA. Section 7 summarizes the findings and outlines directions for future work.

2. Related Work

Policy-gradient methods in zero-sum games. Rudolph et al. [14] hypothesize that, with proper tuning, generic policy-gradient methods are competitive with or superior to specialized game-theoretic alternatives—CFR [23], fictitious play [3], double-oracle [9], PSRO [8]—in imperfect-information games, and support this with a large exploitability study on small-size benchmarks. Sokota et al. [19] apply the same idea at scale: regularized policy gradients—magnetic mirror descent [18] and R-NaD [13]—reach expert-level play in Stratego. We add a case study in a large real-time strategy game, with results consistent with their hypothesis.

Prior Generals.io work. Bhatia et al. [2] highlighted Generals.io as a promising testbed for AI research and built a data-collection and bot-integration framework on top of Redis, releasing a rule-based agent, `Flobot`. Xu et al. [22] likewise framed Generals.io as a cost-effective analogue of Dota 2 and StarCraft II, and proposed a hierarchical self-play agent (`HASP`) that reaches a reported 77% win rate against `Flobot`. Neither released an environment suited to large-scale RL research. Both agents are also far below the community-developed `Human.exe`, the strongest *non-learning* agent, which wins close to 100% of games against either. `Human.exe` combines a wide range of graph algorithms and dynamic programming, and was until recently the top bot on the leaderboard. Straka and Schmid [20] introduced a gym-like, NumPy-based Generals.io environment and trained

2. Code and trained checkpoints will be released upon acceptance.

a PPO agent that reached a top-25 placement on the 1v1 leaderboard, above `Human.exe`, relying on behavior cloning from expert replays, potential-based reward shaping, and population-based self-play.

3. Game

Generals.io is a real-time multiplayer strategy game on a grid with partial observability; at its core are resource allocation, opponent modeling, and deception under fog of war—properties that together support a wide range of strategies and emergent behaviors. Each player commands an army that grows over time, expands across the grid, and tries to capture the opponent's *general* while defending their own. Army units accumulate on owned cells and are dispatched one cell at a time; contact with enemy units resolves into combat by subtraction. Each player sees only their own cells and the immediate neighborhood around them (Figure 1), making scouting and deception central. The same rule set supports 1v1, 2v2, and free-for-all formats, all exposed through our JAX-native environment via a shared gym-like interface. Full mechanics are given in Appendix A.

4. Agent

This work introduces an AI for Generals.io trained from scratch by self-play reinforcement learning, without any human demonstrations or hand-engineered priors. The remainder of this section describes the key components of the agent.

Network architecture. Figure 2 shows the overall architecture. The policy is parameterized by a Chessformer-style transformer torso [10]. The environment produces a feature tensor $o \in \mathbb{R}^{C\times H\times W}$, where H and W are the grid's height and width and $C = 38$ channels encode the current observation together with a memory augmentation (for example, the last-known location of the opponent's base once it has been spotted). The spatial tensor is split into 3×3 patches, and each patch is embedded into a single input token. Two additional non-spatial tokens carry global temporal statistics rather than per-cell features: a 512-step sliding window of the opponent's total army count and a matching window of their total land count, each projected by an MLP into a single token. These trajectories let the agent infer what the opponent is doing behind the fog of war—whether they are expanding, consolidating, or massing an attack—from the way the scoreboard totals evolve. Value and policy heads sit on top of the torso; the policy head produces an $H \times W \times 9$ distribution that, for each source cell, assigns a probability to each of nine actions: pass, or one of two move types (send all units or send half) in each of the four cardinal directions ($2\times 4+1 = 9$). Appendix C gives the full architecture specification and Appendix B the observation-channel layout.

Training objective. Our training objective combines a standard on-policy policy-gradient surrogate with a KL regularizer toward a fixed *anchor policy* π_{anchor} and a value-function loss:

$$\mathcal{L}(\theta) \;=\; -J_{\text{PPO-Clip}}(\theta) \;+\; \rho\,\mathrm{KL}(\pi_\theta \,\|\, \pi_{\text{anchor}}) \;+\; \beta\,\mathcal{L}_{\text{value}}(\theta).$$

In our setup, π_{anchor} is the uniform random policy, so the KL term is equivalent to an entropy bonus and the overall objective reduces to the classical PPO loss [16]. Adding a further KL regularizer toward the previous iterate $\mathrm{KL}(\pi_\theta \,\|\, \pi_{\text{old}})$ would recover magnetic mirror descent [18]; we omit it, which saves one forward pass through π_{old} per minibatch and in our experiments leaves training stable. For $\mathcal{L}_{\text{value}}$ we use the HL-Gauss categorical value loss of Farebrother et al. [5], which we find noticeably more stable than MSE in our setting.

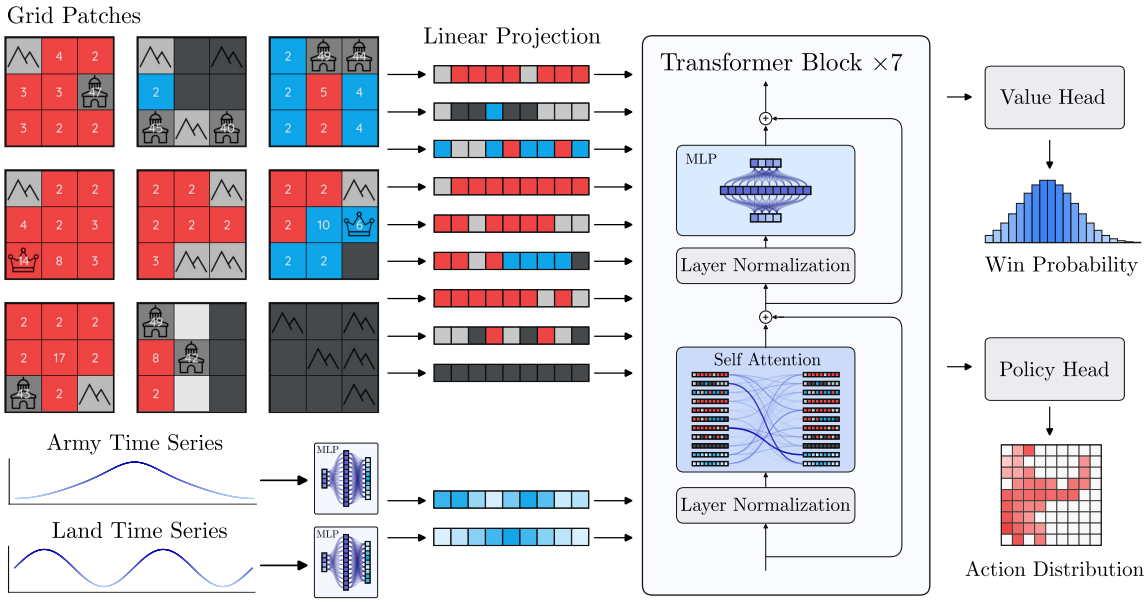

Figure 2: The environment observation is spatially sliced into 3×3 patches and linearly projected into tokens. Alongside these, two time series of land and army counts are each projected by an MLP into the token space. All tokens pass through the transformer and are then consumed by a value head that predicts a win-probability distribution and a policy head that produces a distribution over all possible actions.

Curriculum. In full-size Generals.io, the minimum BFS distance between the two generals is constrained to be at least 17 steps, and training directly in this regime is not effective: a randomly initialized policy rarely stumbles onto capturing the opponent's base, so the win/loss signal is too sparse to bootstrap learning. We therefore run a curriculum over spawn distance: we cap the maximum spawn distance between the two players at a small value (starting at four cells) and gradually raise it over training until the full-game distribution is covered.

Exponential moving average. We accumulate the policy parameters over the course of training into an EMA $\bar{\theta}_t = \tau\bar{\theta}_{t-1} + (1-\tau)\theta_t$. Training gradients act on θ, but at deployment we evaluate $\bar{\theta}$, which consistently outperforms the last iterate at inference.

Top-advantage filtering. Inspired by Sokota et al. [19], we keep only the top 25% of transitions from each rollout batch, ranked by the critic's predicted advantage, and compute the policy-gradient update from those alone.

Computational efficiency. Previous simulators [20] relied on a NumPy-based environment and a PyTorch training stack. We reimplement both the environment and the training loop in JAX, which lets us `jit`-compile entire rollouts end-to-end; this yields a $32\times$ speedup over the prior setup on the same hardware.

5. Evaluation

Our agent played 500 games on the public 1v1 ladder under the nicknames `Average Joe`[3] and `L_7d_gae90_30k_ema`[4], winning 411 of them (82%) to finish as the top-rated player on the ladder. In direct series against the two highest-ranked humans, it won 13 of 20 games against the rank-1 player `shimatetsu` and 15 of 20 against the rank-2 player `Mithraaaa`—a combined 28–12 record across the 40 games (one-sided binomial $p = 0.006$). Against the community state-of-the-art heuristic agent `Human.exe`, our agent won all 10 of 10 games played.

Ratings on the Generals.io 1v1 ladder are computed with OpenSkill [7]. As shown in Figure 3, our agent (114 points) leads the strongest active human (105) by 9 points and the prior AI state of the art, `zero v3` [20] (74), by 40. For context, the top-100 ladder cutoff sits at 68 points; `zero v3` itself already cleared that threshold.

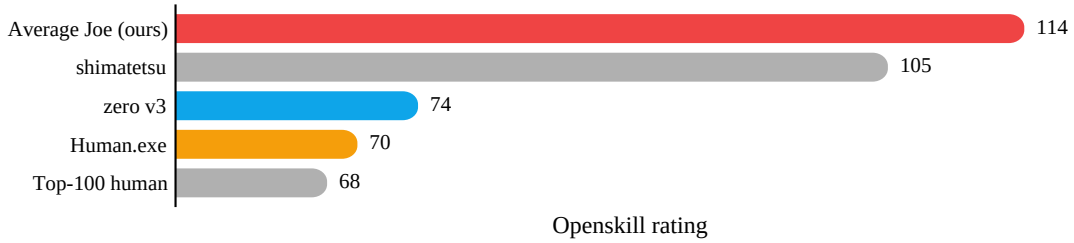

Figure 3: Generals.io 1v1 leaderboard ratings (OpenSkill points). Our agent is the highest-rated player on the ladder, ahead of the strongest human and well clear of the prior AI state of the art (`zero v3` [20]), and of the heuristic agent `Human.exe`.

6. Ablations

6.1. Reward Shaping

The previous state of the art, `zero v3` [20], relied on potential-based reward shaping [12] to bootstrap learning under tight compute constraints: the potential rewarded states with greater material advantage, steering the policy toward stronger strategies faster. The massive speedups from reimplementing the environment in JAX and jitting the full training loop remove that pressure, and we in fact observe that shaped reward starts to hurt as training progresses: it destabilizes late, while sparse win/loss reward converges cleanly (Figure 4, left).

6.2. Exponential Moving Average

Across every experiment we ran during development, the EMA of the policy parameters was stronger at inference than the corresponding last iterate (Figure 4, middle and right). Averaged across three runs, after six days of training the EMA network still led the last iterate by roughly 30 Elo points. We find this consistency striking, especially given that parameter EMA is not discussed as a load-bearing ingredient in prior large-scale self-play work on StarCraft II [21], Dota 2 [1], or Strat-

3. Replays: https://generals.io/profiles/Average%20Joe
4. Replays: https://generals.io/profiles/L_7d_gae90_30k_ema

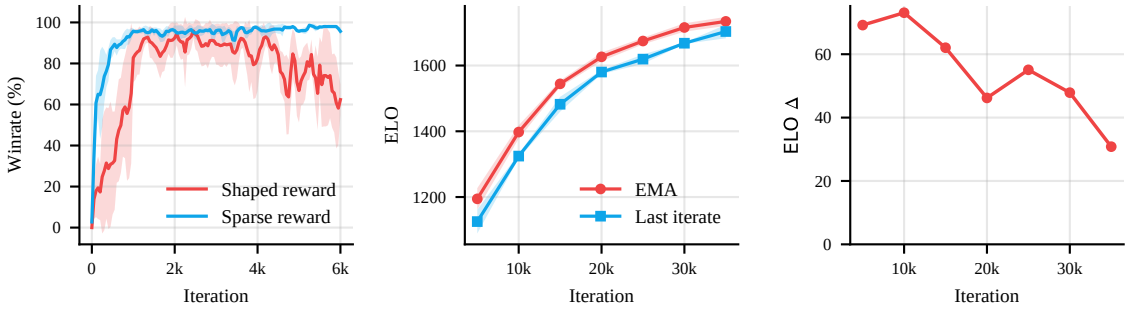

Figure 4: **Left:** winrate against a fixed reference policy for potential-based reward shaping (red) vs. sparse win/loss reward (blue); mean $\pm$ std over $n = 5$ runs. Shaped reward destabilizes late. **Middle:** Elo of EMA parameters (red) vs. last iterate (blue), averaged over three runs. **Right:** Elo gap between EMA and last iterate.

ego [13]; the one exception we are aware of is Sokota et al. [19]. Understanding when and why parameter EMA helps in self-play RL, whether it is a generic stabilizer or specific to certain regimes, is a natural question for future work.

7. Conclusion

We showed that in Generals.io—a large, partially-observed real-time strategy game—a generic policy-gradient loop reaches superhuman play from sparse reward alone. Our ablations isolate which ingredients do the work. We hope this serves as a reference point for future research in games, showing how far policy-gradient methods can scale. Alongside the agent, we release a JAX-native environment that makes these experiments tractable on a single GPU and provides a shared interface for 1v1, 2v2, and free-for-all play—a lightweight yet strategically rich testbed for research on cooperative and general-sum multi-agent RL.

Future work. The natural next step is to test how far simple policy-gradient methods carry beyond two-player zero-sum play. The 2v2 and free-for-all formats in our environment expose general-sum dynamics under the same rules, and they are the obvious place to probe whether these methods remain effective outside the zero-sum regime. The ablations here are only a starting point. One direction concerns top-advantage filtering: we observed noticeably better wall-clock and sample efficiency with it enabled, hinting that many collected samples are redundant and that more principled data-filtering methods could be a fruitful direction. A related open question concerns the $\mathrm{KL}(\pi_\theta \| \pi_{\mathrm{old}})$ trust-region term from the original MMD formulation, which we drop to save a forward pass per minibatch with no visible degradation; the stability/compute tradeoff is worth measuring more carefully. More broadly, while we observe robustness across the schedules we tried (Appendix F), a thorough hyperparameter sensitivity analysis in large games remains to be done.

References

[1] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. URL https://arxiv.org/abs/1912.06680.

[2] Aaditya Bhatia, Austin Davis, Soumik Ghosh, and Gita Sukthankar. Generally genius: A Generals.io agent development and data collection framework. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 400–406, 2023.

[3] George W. Brown. Iterative solution of games by fictitious play. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, 1951.

[4] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[5] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop regressing: Training value functions via classification for scalable deep rl, 2024. URL https://arxiv.org/abs/2403.03950.

[6] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016. URL https://arxiv.org/abs/1603.01121.

[7] Vivek Joshy. OpenSkill: A faster asymmetric multi-team, multiplayer rating system. *Journal of Open Source Software*, 9(93):5901, 2024. doi: 10.21105/joss.05901. URL https://doi.org/10.21105/joss.05901.

[8] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[9] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2003.

[10] Daniel Monroe and Philip A. Chalmers. Mastering chess with a transformer model, 2024. URL https://arxiv.org/abs/2409.12272.

[11] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356:508–513, 2017.

[12] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[13] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*, 378:990–996, 2022.

[14] Max Rudolph, Nathan Lichtlé, Sobhan Mohammadpour, Alexandre Bayen, J. Zico Kolter, Amy Zhang, Gabriele Farina, Eugene Vinitsky, and Samuel Sokota. Reevaluating policy gradient methods for imperfect-information games. *arXiv preprint arXiv:2502.08938*, 2025. URL https://arxiv.org/abs/2502.08938.

[15] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL https://arxiv.org/abs/1712.01815.

[18] Samuel Sokota, Ryan D'Orazio, J. Zico Kolter, Nicolas Loizou, Marc Lanctot, Ioannis Mitliagkas, Noam Brown, and Christian Kroer. A unified approach to reinforcement learning, quantal response equilibria, and two-player zero-sum games. In *International Conference on Learning Representations (ICLR)*, 2023.

[19] Samuel Sokota, Eugene Vinitsky, Hengyuan Hu, J. Zico Kolter, and Gabriele Farina. Superhuman AI for Stratego using self-play reinforcement learning and test-time search. *arXiv preprint arXiv:2511.07312*, 2025. URL https://arxiv.org/abs/2511.07312.

[20] Matej Straka and Martin Schmid. Artificial generals intelligence: Mastering Generals.io with reinforcement learning. *Reinforcement Learning Journal*, 2024.

[21] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, 2019.

[22] Huazhe Xu, Keiran Paster, Qibin Chen, Haoran Tang, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Hierarchical deep reinforcement learning agent with counter self-play on competitive games. 2018.

[23] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

Appendix A. Game Rules

We describe the 1v1 (two-player) format; free-for-all and team variants are covered at the end.

Generals.io is played on an $H \times W$ grid (up to 23×23), with cells that are plain (traversable), mountain (impassable), or hold a *general* or *castle*. Each player owns one general and wins by capturing the opponent's. Each player sees only their owned cells and the Moore (8-cell) neighborhood around them (Figure 1); a scoreboard exposes each player's total cell and army counts.

Generals and owned castles produce one unit every other turn, and every 50 turns each owned plain cell produces one unit as well. On each turn a player selects a source cell and a cardinal direction, dispatching either all but one unit or exactly half of the cell's units to the neighboring cell. Combat is resolved by subtraction: if the destination is empty or friendly the moved units simply accumulate there; otherwise the two stacks cancel one-for-one, and whichever side is larger survives with the remaining difference and takes ownership of the cell. Capturing a neutral castle requires paying down its 40–50 starting units through combat; once captured, it produces for its new owner like any owned castle. In the browser version each turn advances every 0.5 seconds in real time.

In free-for-all with N players, capturing another player's general transfers all of their cells to the captor along with half of their total army, and the captured general is converted into a regular castle. In team play, moving onto an ally-owned cell transfers the cell's ownership to the mover while pooling the ally's units there into the moving stack rather than subtracting them.

Appendix B. Observation Space

The environment exposes a feature tensor with $C = 24 + 2H_{\text{hist}}$ channels, where H_{hist} is the per-cell delta history length (default 7, giving $C = 38$ channels). All channels are (H, W) spatial maps; scalar quantities are broadcast across the grid, and cells outside the playable area are padded with mountain tokens. Channels 0–3 and 9–13 are instantaneous (read from the current step); channels 4–8 and 20–21 are persistent memory accumulated across the episode; channels 16–19 are scalar game-state statistics broadcast to every cell; and channels 24 onward stack per-cell army-count deltas over the last H_{hist} steps. On top of the spatial channels, the transformer also consumes two non-spatial temporal tokens encoding the opponent's army-total and land-count time series over a 512-step sliding window; these are prepended as summary tokens alongside the value token.

Table 1: Observation channels. Indices assume $H_{\text{hist}} = 7$.

Index	Name	Encoding
0	armies	Raw army count on each visible cell (0 if fogged).
1	own_army	Army count on cells owned by the agent; 0 elsewhere.
2	enemy_army	Army count on cells owned by the opponent; 0 elsewhere.
3	neutral_army	Army count on neutral cells (unclaimed cities/generals); 0 elsewhere.
4	seen	Persistent visibility mask: 1 for any cell the agent has ever seen (max-pooled 3×3 owned-cell halo).
5	enemy_seen	Persistent mask: 1 for any cell where an enemy unit has ever been observed.
6	generals	Persistent mask of all general positions ever revealed (own and opponent).
7	cities	Persistent mask of all city positions ever revealed.
8	mountains	Persistent mountain mask (visible mountains $\cup$ known padded border).
9	neutral_cells	Current-frame mask of neutral-owned cells.
10	owned_cells	Current-frame mask of agent-owned cells.
11	opponent_cells	Current-frame mask of opponent-owned cells.
12	fog_cells	Current-frame fog-of-war mask (cells not visible this step).
13	structures_in_fog	Mask of fog cells known to contain a structure (city/mountain/general) or unseen padded border.
14	timestep	Absolute game step, broadcast as a constant map.
15	timestep_mod50	$(\text{timestep} \bmod 50)/50$, broadcast: phase within the army-production cycle.
16	own_land_count	Scalar: number of cells the agent owns, broadcast.
17	own_army_count	Scalar: total agent army, broadcast.
18	opp_land_count	Scalar: number of cells the opponent owns, broadcast.
19	opp_army_count	Scalar: total opponent army (last observed), broadcast.
20	last_enemy_army_seen	Per-cell memory: most recently observed enemy army count at that cell (sticky, not reset when the cell re-fogs).
21	last_enemy_army_age	Per-cell log-decayed age $\log(1+\Delta t)/5$ where Δt is the number of steps since the enemy was last seen there (0 when currently visible).
22	coord_x	Normalized column coordinate in $[0, 1]$, broadcast across rows.
23	coord_y	Normalized row coordinate in $[0, 1]$, broadcast across columns.
24–30	own_army_delta	Stack of H_{hist} per-cell differences $\texttt{own\_army}(t-k) - \texttt{own\_army}(t-k-1)$: recent changes in own army counts per cell.
31–37	enemy_army_delta	Stack of H_{hist} per-cell differences of enemy army counts: recent observed enemy movement and production per cell.

Appendix C. Network

A single learned *value token* is prepended to every observation alongside the spatial patch tokens and the two temporal history tokens. The value head is a linear projection of this value token followed by a softmax over HL-Gauss bins. The policy head takes only the tokens corresponding to cells of the game grid and linearly projects each into a 9-logit vector, yielding the $H \times W \times 9$ action distribution. Transformer hyperparameters are listed in Table 2.

Table 2: Network hyperparameters.

Parameter	**Value**
Depth	7
Embedding dimension	448
Feedforward dimension	1344
Attention heads	8
Total parameters	15.35M

Appendix D. Hyperparameters

Table 3 lists the training hyperparameters used in the reported runs.

Table 3: Training hyperparameters.

Parameter	**Value**
PPO	
Environments per iteration	512
Rollout length	512
Minibatch size	1024
Epochs per iteration	1
Clip range ϵ	0.2
Value-loss coefficient	0.5
Gradient-norm clip	0.267
Top-advantage fraction	0.25
Discount γ	1.0
GAE λ	0.9
Schedules	
Learning rate	$\mathrm{clip}\big(0.5\,(t+1)^{-1.1},\, 5\times10^{-6},\, 1\times10^{-4}\big)$
Entropy	$0.05\,(t+1)^{-0.2}$
Value head (HL-Gauss)	
Bins	128
Range	$[-1, 1]$
σ	0.04
EMA decay τ	0.999

Appendix E. Simulator Performance

We benchmark the JAX environment throughput on a single NVIDIA H200 GPU, sweeping the number of parallel environments in powers of two. Each configuration runs the full step loop end-to-end (observation assembly, action application, reward computation) without a training update. Throughput is reported in millions of environment steps per second (FPS). Results are in Table 4: throughput scales with batch size and saturates in the tens of millions of FPS regime once a few thousand environments are batched.

Table 4: JAX environment throughput on a single NVIDIA H200 GPU as a function of the number of parallel environments.

Environments	Frames Per Second
1,024	13.5M
2,048	22.3M
4,096	31.7M
8,192	40.8M
16,384	45.9M
32,768	50.7M

Appendix F. Schedule Sensitivity

Rudolph et al. [14] report that carefully tuned policy-gradient methods are comparable to or better than CFR, fictitious play, and PSRO across a range of imperfect-information games. Building on that, Sokota et al. [19] claim that the specific annealing schedules of the learning rate and the entropy coefficient were critical for smooth training dynamics. In contrast, we observe robustness across different schedules: combinations of power-law and linear decays for both the learning rate and the entropy coefficient all reach comparable local Elo (Figure 5). Computational constraints did not allow us to run more longer-term experiments, and a broader sensitivity analysis remains for future work.

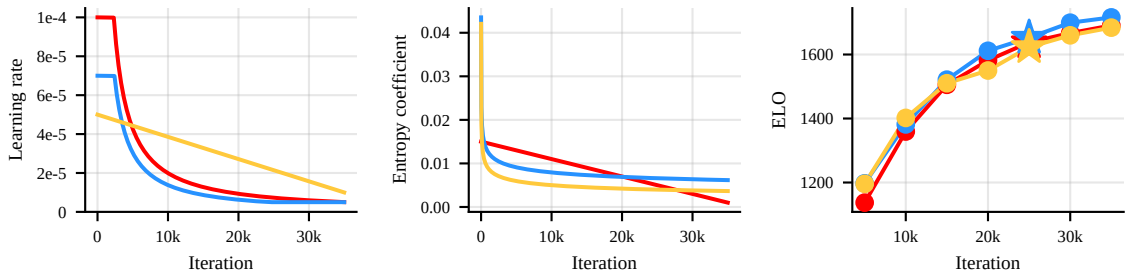

Figure 5: Relationship between different schedules and Elo. **Left:** learning-rate schedules—power-law decays (blue, red) and a linear decay (yellow). **Middle:** entropy-coefficient schedules—power-law decays (blue, yellow) and a linear decay (red). **Right:** Elo from local round-robin evaluation amongst checkpoints.