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

Matej Straka, Martin Schmid
{straka,schmidm}@kam.mff.cuni.cz

Department of Applied Mathematics, Charles University, Prague, Czech Republic

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 (Silver et al., 2017; Schrittwieser et al., 2020), Poker (Moravčík et al., 2017; Brown & Sandholm, 2018), Stratego (Perolat et al., 2022), StarCraft II (Vinyals et al., 2019), and Dota 2 (Berner et al., 2019). These settings have motivated several classes of algorithms. Counterfactual regret minimization (CFR) (Zinkevich et al., 2007) is the standard for tabular imperfect-information play; neural fictitious self-play (Heinrich & Silver, 2016) pairs a best-response learner with a supervised average-policy network; population-based methods such as PSRO (Lanctot et al., 2017) 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 (Sokota et al., 2023; Rudolph et al., 2025) and reaches superhuman play in Stratego (Sokota et al., 2025).

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] Available at https://github.com/strakam/generals-bots.

- **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 (Sokota et al., 2023), and an exponential moving average (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 27–12 and 172–58 respectively.

2 Related Work

2.1 Policy-Gradient Methods in Zero-Sum Games

Rudolph et al. (2025) hypothesize that, with proper tuning, generic policy-gradient methods are competitive with or superior to specialized game-theoretic alternatives such as CFR (Zinkevich et al., 2007), fictitious play (Brown, 1951), double-oracle (McMahan et al., 2003), and PSRO (Lanctot et al., 2017) in imperfect-information games, and support this with a large exploitability study on small-size benchmarks. Sokota et al. (2025) and Perolat et al. (2022) apply the same idea at scale, using two regularized policy gradients, magnetic mirror descent (MMD) (Sokota et al., 2023) and Regularized Nash Dynamics (R-NaD) respectively, to reach top-level play in Stratego. We add a case study in a large real-time strategy game, with results consistent with their hypothesis.

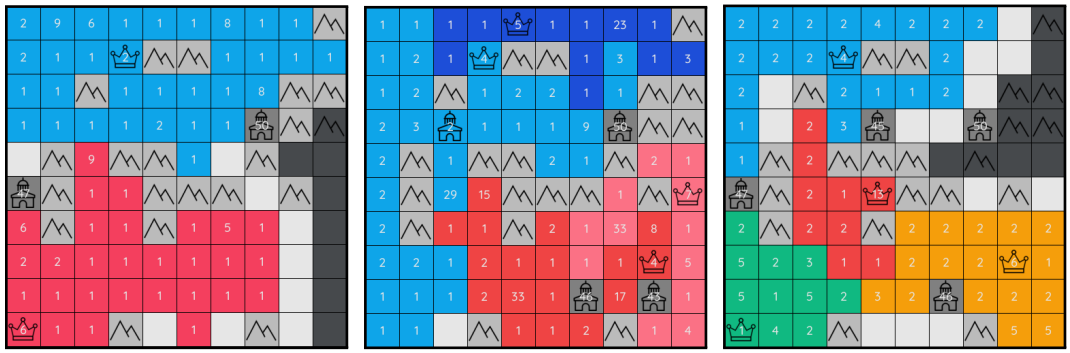
2.2 Prior GENERALS.IO Work

Bhatia et al. (2023) 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. (2018) 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 & Schmid (2025) introduced a gym-like, NumPy-based GENERALS.IO environment and trained 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 (Fig. 8 in the appendix), making scouting and deception central. The same rule set supports 1v1, 2v2, and free-for-all formats (Fig. 1), all exposed through our JAX-native environment via a shared gym-like interface. Full mechanics are given in Appendix A; throughput benchmarks of the simulator are reported in Appendix B.

[2] Code and trained checkpoints at `https://github.com/strakam/AverageJoe`.



(a) 1v1 (b) 2v2 (c) Free-for-all

Figure 1: The three game formats supported by the environment.

4 Agent

This work introduces an AI for GENERALS.IO trained from random initialization 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.

4.1 Network Architecture

Fig. 2 shows the overall architecture. The policy is parameterized by a Chessformer-style transformer torso (Monroe & Chalmers, 2024). 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 location of the opponent's general 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 or consolidating, 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 but one unit or send half) in each of the four cardinal directions ($1 + 2 \times 4 = 9$). Appendix C gives the observation-channel layout.

4.2 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:

$$\mathcal{L}(\theta) = -J_{\text{PPO-Clip}}(\theta) \; + \; \rho \, \text{KL}(\pi_\theta \, \| \, \pi_{\text{anchor}}) \; + \; \beta \, \mathcal{L}_{\text{value}}(\theta). \tag{1}$$

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 (Schulman et al., 2017). We also experimented with several non-parametric heuristic anchors but did not observe any improvement over the uniform choice, so we kept the uniform anchor for simplicity. Adding a further KL regularizer toward the previous iterate $\text{KL}(\pi_\theta \, \| \, \pi_{\text{old}})$ would recover magnetic mirror descent (Sokota et al., 2023); 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 categorical HL-Gauss value loss of Farebrother et al. (2024).

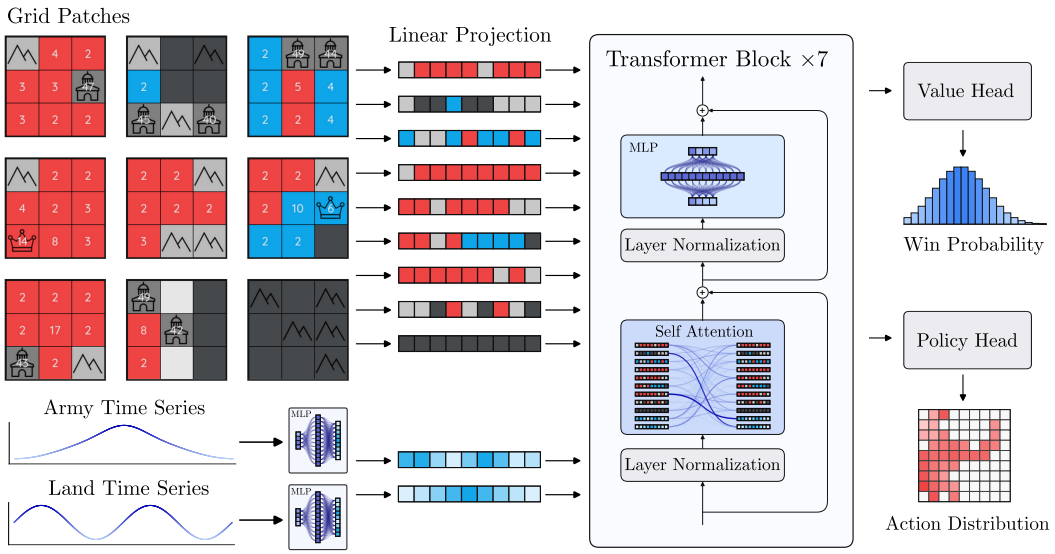


Figure 2: The environment observation is spatially sliced into 3×3 patches and linearly projected into tokens. Alongside these, two time series of land and army counts are each projected by an MLP into the token space. All tokens pass through the transformer and are then consumed by a value head that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

4.3 Training Recipe

Three additions to the basic PPO loop turn out to matter in our setup. The first is a **spawn-distance curriculum**: in full-size GENERALS.IO the minimum BFS distance between the two generals is at least 17 steps, and training directly in this regime is not effective because a randomly initialized policy rarely stumbles onto capturing the opponent's general, so we instead cap the maximum spawn distance at a small value (starting at four cells) and raise it gradually over training until the full-game distribution is covered. The second is a **parameter exponential moving average**: we accumulate the policy parameters into $\bar{\theta}_t = \tau\bar{\theta}_{t-1} + (1-\tau)\theta_t$, train on θ, and evaluate $\bar{\theta}$ at deployment, which consistently outperforms the last iterate (Section 6). The third is **top-advantage filtering**, following Sokota et al. (2025): each rollout batch is sorted by the critic's predicted advantage and only the top 25% of transitions contribute to the policy-gradient update, which gave us noticeably better wall-clock and sample efficiency.

5 Evaluation

We evaluate the agent in three settings: ladder play against the GENERALS.IO community, head-to-head series against the two highest-ranked human players, and head-to-head play against the strongest non-learning bot, `Human.exe`. All ratings reported on the public ladder are OpenSkill points (Joshy, 2024).

5.1 Baselines

We compare against the two strongest prior bots on the public ladder. `Human.exe` is a rule-based agent built around an explicitly maintained fog-of-war belief state, with per-turn decisions chosen by a priority cascade over roughly thirty precomputed plans backed by classical combinatorial optimizers (Appendix F); it was the top-ranked non-learning bot prior to our agent. `zero v3` (Straka & Schmid, 2025) is a convolutional U-Net trained in three stages—behavior cloning on expert replays, PPO self-play under potential-based reward shaping, and population-based self-play over a

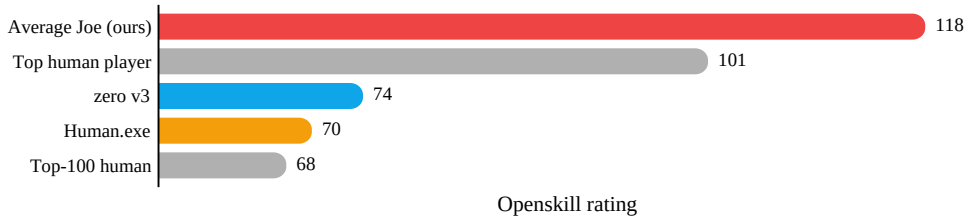


Figure 3: GENERALS.IO 1v1 leaderboard ratings (OpenSkill points). Our agent is the highest-rated player on the ladder, ahead of the strongest human and well clear of the prior AI state of the art `zero v3` (Straka & Schmid, 2025) and the heuristic agent `Human.exe`.

three-model pool—reaching the top 25 of the 1v1 leaderboard and a 54.8% win rate over 529 games against `Human.exe`.

5.2 Ladder Results

Our agent played its first 1,000 games on the public 1v1 ladder under the nicknames `Average Joe`[3] and `L_7d_gae90_30k_ema`[4], winning 815 (81.5%) to finish as the top-rated player on the ladder: 118 points, ahead of the strongest active human (101) by 17 and of `zero v3` by 44 (Fig. 3). Table 1 breaks down the head-to-head record against the two highest-ranked humans and the two strongest prior bots; the 199–70 aggregate against the humans corresponds to $p < 0.001$ on a one-sided binomial test against a fair coin. These were ordinary rated 1v1 matches assigned by the ladder matchmaker, with both players aware the opponent was a bot; the agent had no input or interface advantage—it saw the same client view and issued one action per 0.5-second tick, while human players can additionally queue moves in advance, so the two sides take a nearly identical number of actions per game. We encourage readers to browse the linked profiles and watch a few replays to get a direct feel for how the agent plays.

Table 1: Head-to-head record of `Average Joe` against the two highest-ranked human players and the two strongest prior bots.

Opponent	**W**	**L**	**Win rate**
`shimatetsu` (rank-1 human)	27	12	69.2%
`Mithraaaa` (rank-2 human)	172	58	74.8%
`Human.exe`	20	0	100.0%
`zero v3`	20	0	100.0%
Total	239	70	77.3%

6 Ablations

We isolate several design choices from Section 4: the reward signal (Section 6.1), the deployment policy (Section 6.2), and the top-advantage filtering fraction (Section 6.3). We additionally report robustness to learning-rate and entropy-coefficient schedules (Section 6.4) and an interpretability sketch of one attention head (Section 6.5).

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

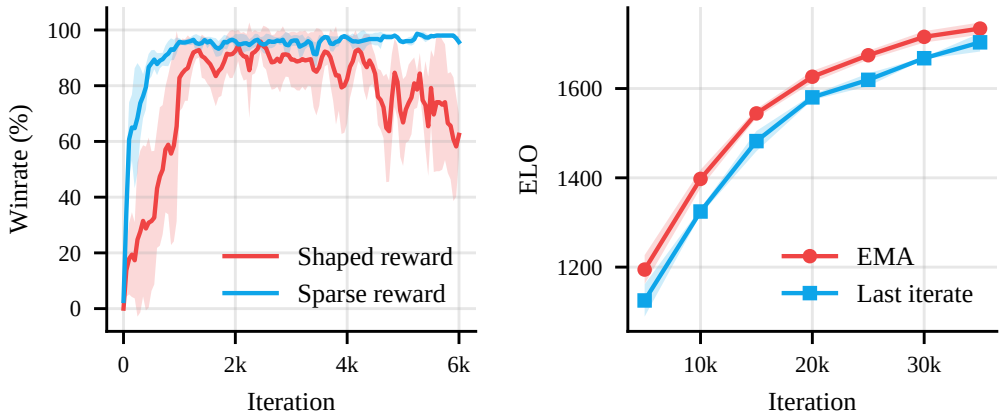


Figure 4: Elo of EMA parameters (red) vs. last iterate (blue), averaged over three runs.

6.1 Reward Shaping

The previous state of the art, `zero v3` (Straka & Schmid, 2025), relied on potential-based reward shaping (Ng et al., 1999) 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 `jit`-compiling 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 (Fig. 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 (Fig. 4, right). Averaged across three runs, after six days of training the EMA network still led the last iterate by roughly 30 Elo points—a non-trivial gap at this level of play. 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 (Vinyals et al., 2019), Dota 2 (Berner et al., 2019), or Stratego (Perolat et al., 2022); the one exception we are aware of is Sokota et al. (2025). 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.

6.3 Top-Advantage Filtering

We ablate the top-advantage filter described in Section 4.3 by sweeping the fraction of transitions retained per rollout batch, with three seeds per setting. Fig. 5 plots Elo against wall-clock training time. The aggressive 25%-only setting has the highest variance across seeds, but its best seed dominates every other setting by a large margin: after 24 hours of training it leads the second-best run by 112 Elo.

The seed-level variance at this scale is, in our experience, an artifact of single-GPU training. When we scaled the same recipe from one GPU to four, the 25% setting became substantially more stable across seeds while maintaining its relative advantage over the other choices, and this is what the final agent in Section 5 uses. Compute constraints prevented us from running the full ablation sweep again at the four-GPU scale. The same filter is also more sample-efficient, not only faster in wall-clock terms (Appendix G).

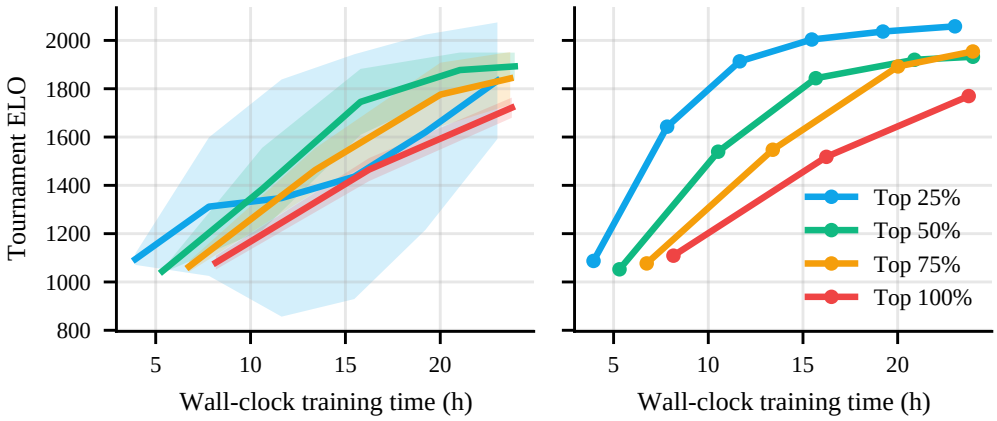


Figure 5: Top-advantage filtering ablation: Elo vs. wall-clock training time, three seeds per setting. **Left:** mean and per-seed range. **Right:** best seed per setting. The aggressive 25% filter is the most variable across seeds but its best seed dominates the next-best setting by 112 Elo after 24 hours.

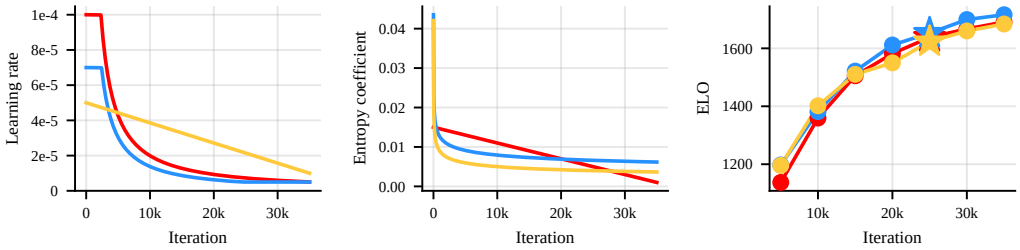


Figure 6: Relationship between different schedules and Elo. **Left:** learning-rate schedules—power-law decays (blue, red) and a linear decay (yellow). **Middle:** entropy-coefficient schedules—power-law decays (blue, yellow) and a linear decay (red). **Right:** Elo from local round-robin evaluation amongst checkpoints. Stars mark the iteration at which the agent became superhuman.

6.4 Schedule Sensitivity

Rudolph et al. (2025) 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. (2025) 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 (Fig. 6). Within our sweep, the *shape* of the learning-rate schedule matters less than the *range* of values it covers.

6.5 Attention Head Analysis

Most attention heads in the trained policy resist a clean per-head reading, but a few play a consistent role across games. One is the value-token attention to spatial patches in head 4 of layer 6, which closely resembles the agent's posterior over the opponent's general's location. Fig. 7 walks four snapshots of one self-play game. At $t = 1$, before any contact, the mass is a banana-shaped band at BFS distance ≥ 17 from the agent—the map-generation prior. At $t = 44$, first contact narrows it to roughly a third of the map with three peaked candidates. By $t = 55$, without further observation, it sharpens to a single tile by elimination: had the opponent been at either closer candidate, they

would already have attacked. At $t = 233$ the agent finally sees the general (marked with a star) and the distribution collapses onto that tile for the rest of the game. The same pattern holds across many self-play and ladder games, with this head's accuracy competitive with the hand-engineered fog-tracker of `Human.exe` (Section 5.1).

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, with parameter EMA as a stabilizer at inference time and without behavior cloning, potential-based reward shaping, or population-based self-play. Together with prior results on Stratego (Sokota et al., 2025; Perolat et al., 2022) and small-game benchmarks (Rudolph et al., 2025), this adds to a growing body of evidence that well-tuned PG methods reach very strong play in two-player zero-sum games. The natural next question is whether the same holds in general-sum settings: our 2v2 and free-for-all formats expose those dynamics under the same rules and through the same environment, and are the obvious place to test it.

Acknowledgments

All authors were supported by the grant no. 25-18031S of the Czech Science Foundation (GAČR). The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

References

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.

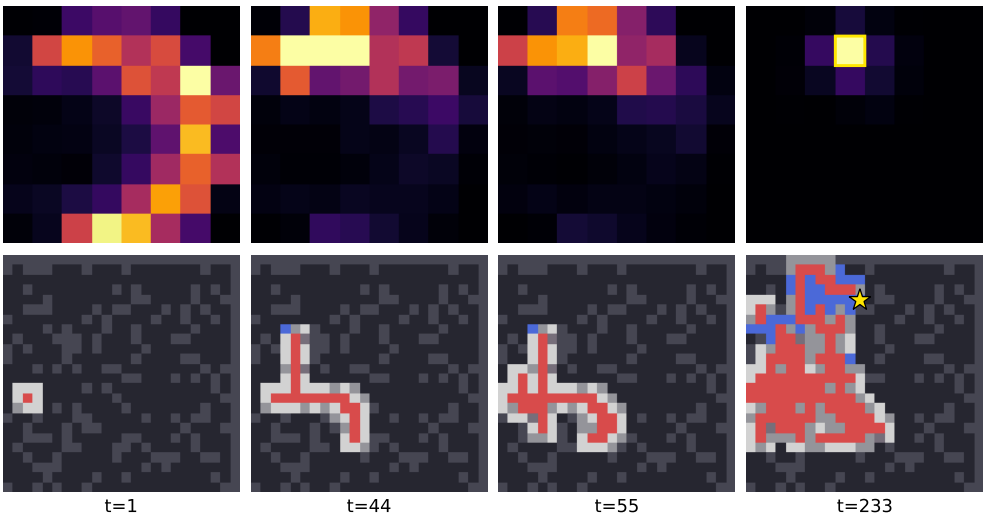


Figure 7: Value-token attention from head 4 of layer 6 at four snapshots of a single self-play game, overlaid on the player-0 view. The distribution tracks the agent's evolving posterior over the opponent's general's location: it starts as a map-generation prior, sharpens as evidence accumulates, and collapses onto the true tile (marked with a star) once observed.

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, pp. 400–406, 2023.

George W. Brown. Iterative solution of games by fictitious play. In Tjalling C. Koopmans (ed.), *Activity Analysis of Production and Allocation*. Wiley, 1951.

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

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. In *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13049–13071, 2024.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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

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.

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.

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.

Matej Straka and Martin Schmid. Artificial generals intelligence: Mastering Generals.io with reinforcement learning. *arXiv preprint arXiv:2507.06825*, 2025. URL https://arxiv.org/abs/2507.06825.

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.

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. Withdrawn ICLR 2019 submission.

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.

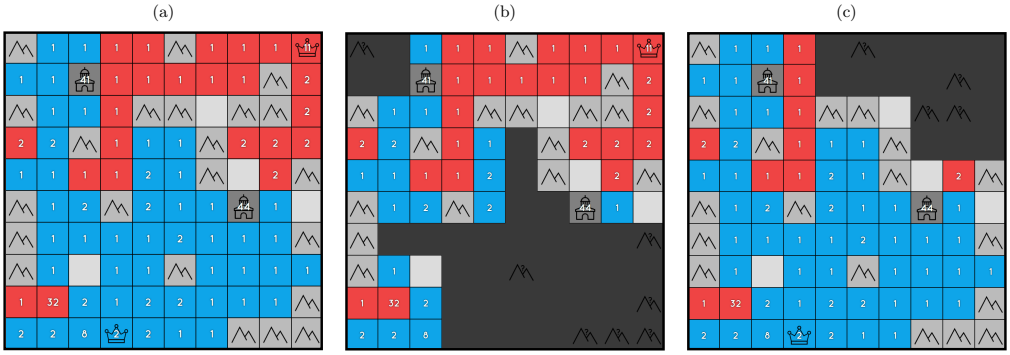


Figure 8: Three views of the same game state: (a) perfect-information view; (b) the red player's view; (c) the blue player's view. Crowns mark bases; numbers are unit counts.

Player	Army	Land
Sky	41	23
Emerald	34	12
Amber	46	21
Red	22	9

Figure 9: The in-game scoreboard, exposing each player's total land (owned-cell count) and total army for the current turn.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Game Rules

A.1 Grid Generation

In the official GENERALS.IO game, matches are played on an $H \times W$ grid (up to 23×23) populated with four cell types: *plain* (traversable), *mountain* (impassable), *general* (a player's base), and *castle* (army-generating cell). The map generator places roughly 20% mountain cells and 9–11 neutral castles, each pre-loaded with 40–50 neutral units. The two generals are placed at least 17 BFS steps apart, and each player is guaranteed at least one neutral castle within BFS radius 6 of their general. Our simulator reproduces these settings by default, but exposes each as a tunable parameter for research.

A.2 Ownership and Partial Observability

Each cell is either neutral or owned by a single player. A fog-of-war mechanic restricts each player's view to the cells they own together with the Moore (8-cell) neighborhood around them; all other cells are hidden (Fig. 8). In parallel, a public global scoreboard (Fig. 9) exposes each player's total owned-cell count and aggregate army count. By watching how these totals evolve, both players can infer what the opponent is doing despite the fog of war, for instance whether they are expanding or consolidating, and bound the maximum army the opponent could have stockpiled at their base.

A.3 Army Growth

Players grow their armies through two production mechanisms acting on different timescales. Generals and owned castles produce one unit every other turn. In addition, every 50 turns the entire territory ticks: each plain cell the player owns gains one unit simultaneously. Capturing a neutral castle requires paying down its 40–50 starting units through combat; once captured, it produces like any owned castle.

A.4 Movement and Combat

Gameplay advances in turns; in the browser version each turn lasts 0.5 seconds in real time, and all players' moves are resolved simultaneously. On each turn, a player selects one of their owned cells and a cardinal direction, dispatching either all-but-one unit or exactly half of the cell's units to the neighboring cell. Players may also pass, leaving the board unchanged.

Combat is resolved by subtraction. If the destination is empty, the dispatched units simply move there. If it is held by an enemy army, the two stacks cancel one-for-one and whichever side is larger survives with the remaining difference, taking ownership of the cell (ties leave the cell with its previous owner). The win condition is to capture the opponent's general: move an army onto the general's cell and win the resulting combat.

A.5 Team and Free-for-All Variants

The same mechanics scale to multi-player formats with two natural modifications. 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. The last general standing wins. In team play (2v2), 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 as enemy units would. Allies share the win/loss condition: the team loses when both of its generals have been captured.

A.6 Strategic Complexity

Several interacting mechanics give GENERALS.IO its strategic depth. *Fog of war* injects persistent uncertainty: a player must infer the opponent's general location, army composition, and intent from local observations and the global scoreboard. *Deception* emerges naturally: feints toward one flank can mask a real attack elsewhere, and a player can route an army around the opponent through the fog and strike their general from behind once it is left undefended. *Tempo* is central: capturing territory just before the 50-turn reinforcement tick yields a disproportionate payoff, while over-expanding leaves the army too thin to repel a focused attack. *Snowballing* compounds early advantages, since each captured cell contributes to future production, so small early mistakes can cascade into game-deciding asymmetries. Capturing a castle is the clearest instance of a longer-horizon investment: it yields a permanent boost to production, but the army spent on the capture leaves the player materially weaker for a stretch of turns, so the agent must time the investment to a window when the opponent cannot exploit the gap.

B Simulator Performance

Prior research on GENERALS.IO has relied either on a non-vectorizable bot-integration framework (Bhatia et al., 2023) or on a NumPy-based vectorized environment coupled to a PyTorch training stack (Straka & Schmid, 2025), the latter reporting around $3{,}500$ steps per second on a 12-core CPU. We reimplement the environment, the rollouts, and the training loop in JAX, so that the entire pipeline can be `jit`-compiled end-to-end and dispatched to a GPU as a single graph. Observations, rewards, fog-of-war masking, army production, combat resolution, and the spawn-distance curriculum are all expressed as pure functions over a flat `PyTree` of game state. This

yields a peak throughput of 50.7M environment steps per second on a single H200 GPU (Table 2), more than four orders of magnitude above the prior CPU baseline, and a 32× speedup of the full training loop on the same hardware as that prior work.

Table 2: Throughput of the JAX environment on a single NVIDIA H200 GPU, by 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

The environment exposes the same observation and action interface for the 1v1, 2v2, and free-for-all formats. The action space is fixed at $H \times W \times 9$ for every mode (a per-cell choice of pass, or one of two move types, send all but one unit or send half, in each of the four cardinal directions), and the observation tensor encodes the agent's view of the board (Appendix C).

C 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 14–15 encode the absolute and cyclic game timestep and channels 22–23 the normalized cell coordinates; 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.

D Network Heads

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.

E Hyperparameters

Table 4 lists the training hyperparameters used in the reported runs.

Table 3: 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 castles/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	castles	Persistent mask of all castle 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 (castle/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 $\texttt{own_army}(t-k) - \texttt{own_army}(t-k-1)$.
31–37	enemy_army_delta	Stack of H_{hist} per-cell differences of enemy army counts.

Table 4: 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

F Detailed Description of `Human.exe`

`Human.exe` is a rule-based agent built around an explicitly maintained belief state. Its per-turn decision is structured as a strict priority cascade of roughly thirty precomputed plans (defense first, then opportunistic plays, then the productive default of expansion or gather), in which the first applicable plan is selected. Planning is amortized into a perception pass at the start of each turn, so the cascade only selects which plan to execute. The underlying optimizers are classical: army consolidation is solved as a rooted prize-collecting Steiner tree with a binary search over the cost basis to enforce a turn budget; territorial expansion is solved as min-cost flow over a graph contraction in which contiguous tile regions become nodes; and candidate paths from both subsystems are composed via a multiple-choice knapsack dynamic program. Short-range engagements are resolved by a simultaneous-move minimax whose root payoff matrix is collapsed to a mixed-strategy Nash equilibrium, with a Decoupled-UCT MCTS variant as an anytime alternative. A belief model over the fog of war projects disappeared enemy armies along expected routes; at branch points they are held as an entangled superposition over candidate tiles, each copy contributing independently to threat assessment so that defense and intercept plans hedge across all live possibilities until observation collapses the superposition. When an enemy tile reappears carrying unexplained army, a backward fog search reconstructs the most plausible delivery route and commits to it as ground truth: the traversed fog tiles are declared empty (and thus weak attack targets), and any undiscovered obstacles along the route are inferred to be hidden enemy castles, since only castles could have produced the unaccounted army. A parallel conservation-law accounting layer attributes every unexplained score, tile, or emergence delta to specific fog activity, refining estimates of hidden army totals and possible enemy general locations.

G Top-Advantage Filtering: Sample Efficiency

Fig. 10 shows the same top-advantage filtering ablation as Section 6.3, plotted against environment samples instead of wall-clock time. The 25% setting is more sample-efficient than the alternatives, so the wall-clock gains observed in Section 6.3 are not just a matter of cheaper-per-step updates.

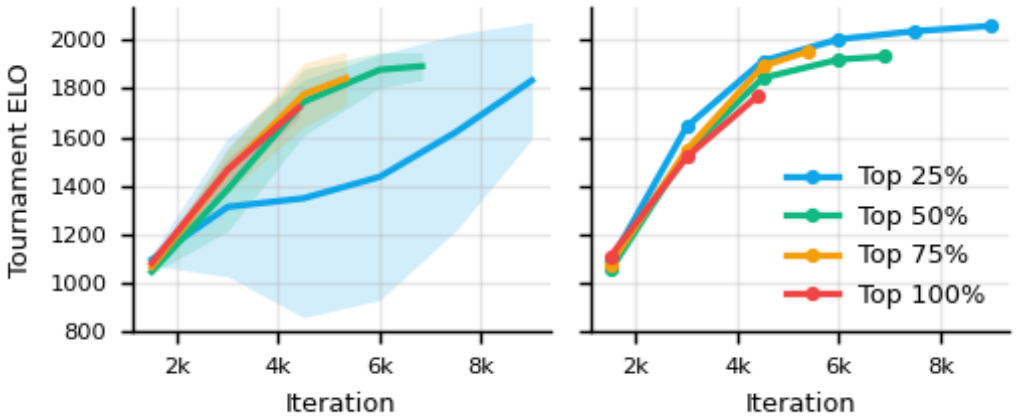


Figure 10: Top-advantage filtering ablation, plotted against environment samples. The 25% setting is more sample-efficient than the alternatives.