Mastering the Game of Stratego with Model-Free Multiagent Reinforcement Learning

Julien Perolat[*,1,‡], Bart de Vylder[*,1,‡], Daniel Hennes[1], Eugene Tarassov[1], Florian Strub[1], Vincent de Boer[†1], Paul Muller[1], Jerome T. Connor[1], Neil Burch[1], Thomas Anthony[1], Stephen McAleer[1], Romuald Elie[1], Sarah H. Cen[1], Zhe Wang[1], Audrunas Gruslys[1], Aleksandra Malysheva[1], Mina Khan[1], Sherjil Ozair[1], Finbarr Timbers[1], Toby Pohlen[1], Tom Eccles[1], Mark Rowland[1], Marc Lanctot[1], Jean-Baptiste Lespiau[1], Bilal Piot[1], Shayegan Omidshafiei[1], Edward Lockhart[1], Laurent Sifre[1], Nathalie Beauguerlange[1], Remi Munos[1], David Silver[1], Satinder Singh[1], Demis Hassabis[1], and Karl Tuyls[*,1,‡]

[1]DeepMind

[‡]Shared lead authors

We introduce *DeepNash*, an autonomous agent capable of learning to play the imperfect information game Stratego[1] from scratch, up to a human expert level. Stratego is one of the few iconic board games that Artificial Intelligence (AI) has not yet mastered. This popular game has an enormous game tree on the order of 10^{535} nodes, i.e., 10^{175} times larger than that of Go. It has the additional complexity of requiring decision-making under imperfect information, similar to Texas hold'em poker, which has a significantly smaller game

[*]corresponding authors : perolat@deepmind.com, bartdv@deepmind.com and karltuyls@deepmind.com

[†]Independent consultant to DeepMind

[1]Stratego is a trademark of Jumbo Diset Group, and is used in this publication for information purposes only.

tree (on the order of 10^{164} nodes). Decisions in Stratego are made over a large number of discrete actions with no obvious link between action and outcome. Episodes are long, with often hundreds of moves before a player wins, and situations in Stratego can not easily be broken down into manageably-sized sub-problems as in poker. For these reasons, Stratego has been a grand challenge for the field of AI for decades, and existing AI methods barely reach an amateur level of play. *DeepNash* uses a game-theoretic, model-free deep reinforcement learning method, without search, that learns to master Stratego via self-play. The Regularised Nash Dynamics (R-NaD) algorithm, a key component of *DeepNash*, converges to an approximate Nash equilibrium, instead of 'cycling' around it, by directly modifying the underlying multi-agent learning dynamics. *DeepNash* beats existing state-of-the-art AI methods in Stratego and achieved a yearly (2022) and all-time top-3 rank on the Gravon games platform, competing with human expert players.

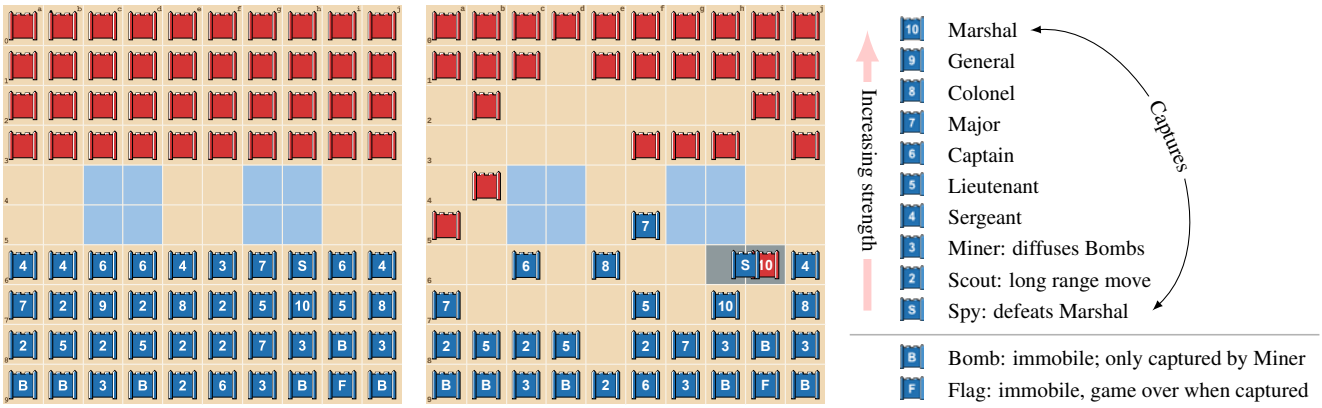
1 Introduction

Progress in Artificial Intelligence (AI) has been measured via mastery of board games since the inception of the field (*1*). Board games allow us to gauge and evaluate how humans and machines develop and execute strategies in a controlled environment. The ability to plan ahead has been at the heart of successes in AI for decades in perfect information games such as chess, checkers, shogi and Go, as well as in imperfect information games such as poker and Scotland Yard (*2*–*8*). For many years the Stratego board game has constituted one of the next frontiers of AI research. For a visualization of the game phases and game mechanics see Figure 1a. The game poses two key challenges. First, the game tree of Stratego has 10^{535} possible states, which is larger than both no-limit Texas hold'em poker, a well-researched imperfect information

game with 10^{164} states (*9*), and the game of Go, which has 10^{360} states (*9*). Second, acting in a given situation in Stratego requires reasoning over 10^{66} possible deployments at the start of the game for each player, whereas poker has only 10^3 possible pairs of cards (*10*). Perfect information games like Go and chess do not have a private deployment phase, therefore avoiding the complexity this challenge poses in Stratego. Currently it is not possible to use state-of-the-art model-based perfect information planning techniques, nor state-of-the-art imperfect information search techniques that break down the game into independent situations (*7, 8*).

For these reasons, Stratego provides a challenging benchmark for studying strategic interactions at an unparalleled scale. As in most board games, Stratego tests our ability to make relatively slow, deliberative, and logical decisions sequentially. Most recent successes in large imperfect information games have been achieved in real-time strategy games such as Starcraft, Dota and Capture the flag (*11–13*) in which most decisions must be made quickly and instinctively, and are of a continuous-time nature. Stratego is a game in which little progress has been achieved by the AI research community due to many complex aspects of the game's structure. Successes in the game have been limited, with artificial agents only able to play at a level comparable to a human amateur, see e.g. (*14–20*). Developing intelligent agents that learn end-to-end to make optimal decisions under imperfect information in Stratego, from scratch, without human demonstration data, remained one of the grand challenges of AI research.

In this work we introduce *DeepNash*, an agent that learns to play Stratego in self-play in a model-free manner without human demonstration, beating previous state-of-the-art AI agents and achieving expert human-level performance in the most complex variant of the game, Stratego Classic. At the core of *DeepNash* is a principled, model-free reinforcement learning algorithm called *Regularized Nash Dynamics* (R-NaD). *DeepNash* combines R-NaD with a deep neural network architecture and converges to an ϵ-Nash equilibrium, which means it learns to play at a highly competitive level, and is robust against opponents that try to exploit it. All games of



Phase 1: Private deployment **Phase 2:** Game play Piece types

(a) Stratego is a two-player board game where each player aims to capture the opponent's flag. To do so, they each have 40 pieces of diverse strengths. The game starts with the deployment phase, where both players secretly position their pieces on the board. In a second game-play phase, the players take turns moving pieces. When two pieces are in the same location, they are revealed, and the weaker piece is removed, or both if they have the same strength. When the weakest movable piece, the Spy, attacks the 10, however, it wins and the 10 is captured. The players have only a partial view on the opponent's pieces: seeing their position but not their type. The complete rules (*21*) are defined by the International Stratego Federation.

Replicator dynamics: $\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a^i)\left[Q^i_{\pi_\tau}(a^i) - \sum_{b^i}\pi^i_\tau(b^i)Q^i_{\pi_\tau}(b^i)\right]$

Reward transformation: $r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) - \eta\log\left(\frac{\pi^i(a^i)}{\pi^i_{\text{reg}}(a^i)}\right) + \eta\log\left(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{\text{reg}}(a^{-i})}\right)$

(b) An overview of the DeepNash approach. DeepNash is an autonomous agent that learns to play the imperfect information game Stratego (A). It learns a policy represented by a deep neural network (B) through self-play from scratch (C) in order to converge to a Nash equilibrium (D).

Figure 1: The Stratego game (a) and an overview of the *DeepNash* approach (b).

imperfect information possess a Nash equilibrium in mixed strategy (*22*), assigning a mixed (or stochastic) strategy for all the players in which no player benefits from deviating from their strategy as long as no other player deviates. While it is sufficient to take deterministic decisions that maximize the value of the equilibrium strategy in turn-taking two-player zero-sum games of

full information, this approach is theoretically unsound when dealing with imperfect information games. In such games, other tactics need to be deployed, which better reflect decision-making processes in the real world. As von Neumann described it "real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do." (*23*). Figure 1b illustrates a high-level overview of the *DeepNash* approach.

We lay out our new model-free reinforcement learning method *DeepNash*, and systematically evaluate its performance against various state-of-the-art Stratego bots and human expert players on the Gravon games platform (*24*). *DeepNash* convincingly beats all current state-of-the-art bots that have been developed to play Stratego with a win rate of over 97% and achieves a highly competitive level of play with human expert Stratego players on Gravon, where it ranks among the top 3 players, both on the annual (2022) and all-times leaderboards, with a win rate of 84%. As such, it is the first time an AI algorithm is able to learn to play at a human-expert level in a complex board game without deploying any search method in the learning algorithm, and the first time an AI achieves human-expert level in the game of Stratego.

2 Methods

DeepNash takes an end-to-end learning approach to solve Stratego, by incorporating the learning of the deployment phase, i.e., putting the pieces tactically on the board at the start of a game (see Figure 1a), in the learning of the game-play phase, using an integrated deep RL and game-theoretic approach. The agent's purpose is to learn an approximate Nash equilibrium through self-play. A Nash equilibrium guarantees that the agent will perform well, even against a worst case opponent. Designing a strategy to be robust in the worst case is typically a good choice to play well against humans in two-player zero-sum games (see e.g. (*6, 7, 25*)), as a Nash equilibrium guarantees an unexploitable agent, and thus the best possible worst-case performance. In perfect information games, search techniques aided by reinforcement learning, i.e. model-based learning

techniques, have provided state-of-the-art superhuman bots in Go and chess (*3, 26*). However, searching for a Nash equilibrium in imperfect information games requires estimating private information of the opponent from public states (*5, 8, 27*). Given the vast number of such possible private configurations in a public state, Stratego computationally challenges all existing search techniques as the search space becomes intractable. We therefore chose an orthogonal route in this work, without search, and propose a new method that combines model-free reinforcement learning in self-play with a game-theoretic algorithmic idea, *Regularized Nash Dynamics* (R-NaD). The model-free part implies that we don't build an explicit opponent model tracking belief space (calculating a likelihood of the opponent's state), and the game-theoretic part is based on the idea that by modifying the dynamical system underpinning our reinforcement-learning approach we can steer the learning behavior of the agent in the direction of the Nash equilibrium. The main advantage of this combined approach is that we do not need to explicitly model private states from public ones. A complex challenge, on the other hand, is to scale up this model-free reinforcement learning approach with R-NaD to make self-play competitive against human expert players in Stratego, which has not been achieved to date. This combined *DeepNash* approach is illustrated in Figure 1b.

In the following subsections, we will use elementary concepts from game theory, and refer the unfamiliar reader for more details to the background section in the supplementary material.

2.1 Learning approach

We learn a Nash equilibrium in Stratego through self-play and model-free reinforcement learning. The idea of combining model-free RL and self-play has been tried before, but it has been empirically challenging to stabilize such learning algorithms when scaling up to complex games, as for example Capture the flag, Dota and StarCraft (*12, 28, 29*). Some empirical work manages to stabilize the learning either by training against past versions of the agent (*12, 28, 29*), or by

adding reward-shaping (*12, 29*) or expert data (*28*) in the training algorithm. While these are helpful tricks, such approaches lack theoretical foundations, remain hard to tune and do not easily generalize to new games. Furthermore, in a game like Stratego, it is difficult to define a loss whose minimization would converge to a Nash equilibrium without introducing prohibitive computational obstacles at large scale. For instance, minimizing the exploitability (*30*), a well-known quantity that measures the distance to a Nash equilibrium, requires estimating an agent's best response during training, which is computationally intractable in Stratego. However, it is possible to define a learning update rule that induces a dynamical system for which there exists a so-called Lyapunov function. This function can be shown to decrease during learning and as such guarantees convergence to a fixed point. This is the central idea behind the R-NaD algorithm, and the successful recipe for *DeepNash*, which scales this approach using a deep neural network.

2.2 Regularized Nash Dynamics algorithm

The R-NaD learning algorithm used in *DeepNash* is based on the idea of regularization (*31–37*) for convergence purposes, which we briefly first explain in the context of zero-sum two-player normal form games (illustrated on the matching pennies game).[2] R-NaD relies on three key steps (see also Figure 2b):

First a *reward transformation* step is performed based on a regularization policy π_{reg} which induces a modified game with rewards: $r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) - \eta \log(\frac{\pi^i(a^i)}{\pi^i_{\text{reg}}(a^i)}) + \eta \log(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{\text{reg}}(a^{-i})})$, with $\eta > 0$ a regularization parameter and i the player index ($i \in [1, 2]$). Note that this transformed reward is policy-dependent.

Second, in the *dynamics* step we let the system evolve according to the replicator dynamics system (*38–41*) on this modified game. Replicator dynamics are a descriptive learning process

[2] An NFG is an abstraction of a decision-making situation involving more than one agent. Each agent needs to simultaneously take an action, after which they receive a game reward, and the game starts a new iteration of the same situation.

		Player 2	
		Head: H	Tail: T
Player 1	Head: H	1	-1
	Tail: T	-1	1

(a) Matching pennies

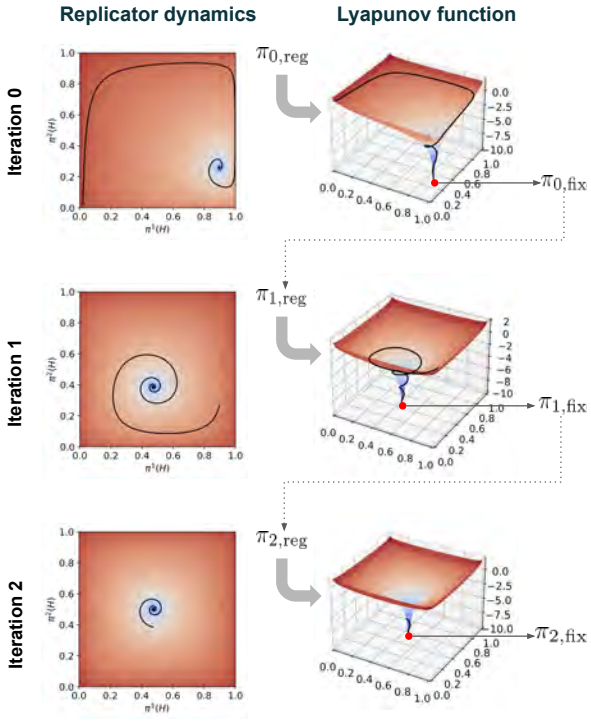
R-NaD Iteration

Start with an arbitrary regularization policy: $\pi_{0,\text{reg}}$

1. Reward transformation: Construct the transformed game with: $\pi_{n,\text{reg}}$
2. Dynamics: Run the replicator dynamics until convergence to: $\pi_{n,\text{fix}}$
3. Update: Set the regularization policy: $\pi_{n+1,\text{reg}} = \pi_{n,\text{fix}}$

Repeat steps until convergence

(b) Algorithmic steps



(c) Dynamics and Lyapunov function

Figure 2: The R-NaD learning algorithm illustrated with the matching pennies game

from evolutionary game theory, equivalent to RL algorithms (*31, 41*), that are also known as Follow the Regularized Leader (*37*), and defined as follows:

$$\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a^i)[Q^i_{\pi_\tau}(a^i) - \sum_{b^i} \pi^i_\tau(b^i)Q^i_{\pi_\tau}(b^i)]$$

with $Q^i_{\pi_\tau}(a^i)$ the quality or fitness of an action. These dynamics reinforce the probability of taking actions with high fitness (relative to other actions). Thanks to the reward transformation this system has a unique fixed point π_{fix} and convergence to it is guaranteed, which can be proven by the Lyapunov function $H_{\pi_{\text{fix}}}(\pi) = \sum_{i=1}^{2} \sum_{a^i \in A^i} \pi^i_{\text{fix}}(a^i) \log\left(\frac{\pi^i_{\text{fix}}(a^i)}{\pi^i(a^i)}\right)$ (*34*). however, this fixed point is not yet a Nash equilibrium of the original game.

In the final *update* step, the fixed point obtained is used as the regularization policy for the next iteration. These three steps are applied repeatedly, generating a sequence of fixed points which

can be proven to converge to a Nash equilibrium of the original (unmodified) game (*34*). Figure 2c illustrates the R-NaD algorithm on the two-player matching pennies game (with the payoff table in Figure 2a). The first iteration starts from $\pi^i_{0,\text{reg}}[H,T] = [0.999, 0.001]$, ($\eta = 0.2$) and the replicator dynamics converge to $\pi^0_{0,\text{fix}}[H,T] = [0.896, 0.104]$ and $\pi^1_{0,\text{fix}}[H,T] = [0.263, 0.737]$. The right figure shows the evolution of the logarithm of the Lyapunov function and illustrates it decreases while learning. Three iterations of R-NaD are shown.

2.3 DeepNash: R-NaD at scale

DeepNash consists of three components: (1) a core training component R-NaD, the model-free RL algorithm presented above, implemented using a deep convolutional network, (2) fine-tuning of the learnt policy to reduce the residual probabilities of taking highly improbable actions and, (3) test-time post-processing to filter out low probability actions and clear mistakes.

We start by concisely laying out some essential background information on imperfect information games necessary to understand how R-NaD is scaled to a deep learning model. Then we continue to unpack the three algorithmic steps of R-NaD and summarize how they are implemented in the neural architecture. For a detailed description we refer to the supplemental material.

2.3.1 Imperfect information games

In a two-player zero-sum imperfect information game, two players (player $i = 1$ or $i = 2$) sequentially interact in turns. At turn t the players receive a reward signal (r^1_t, r^2_t) and the current player $i = \psi_t$ observes the game state through an observation o_t and selects an action a_t according to a parameterized policy function $\pi(.|o_t)$. In model-free reinforcement learning the trajectories $\mathrm{T} = [(o_t, a_t, (r^1_t, r^2_t), \pi(.|o_t)), \psi_t]_{0 \le t < t_{\text{max}}}$ are the only data the agent will leverage to learn the parameterized policy.

2.3.2 Model-free Reinforcement Learning with Regularized Nash Dynamics

DeepNash scales the R-NaD algorithm by using deep learning architectures. It carries out the same three algorithmic steps as before in NFGs: (1) the *reward transformation step*, which modifies the reward, (2) the *dynamics step* which allows for convergence to a fixed point, and (3) the *update step* in which the algorithm updates the policy that defines the regularization function.

Neural architecture and observation representation: *DeepNash*'s network consists of the following components: a U-Net torso with residual blocks and skip-connections (*42, 43*), and four heads which are smaller replicas of the torso augmented with final layers to generate an output of the appropriate shape. The first *DeepNash* head outputs the value function as a scalar, while the three remaining heads encode the agent's policy by outputting a probability distribution over its actions at deployment and during gameplay. The agent architecture is described in detail in the supplementary material.

The observation is encoded as a spatial tensor consisting of the following components: *DeepNash*'s own pieces, the publicly available information about both the opponent's and *DeepNash*'s pieces and an encoding of the 40 last moves. This public information represents the types each piece can still have given the history of the game. In total, the observation contains 82 stacked frames encoded in a single tensor. The structure of this observation tensor is illustrated in Figure 3 and details are provided in the supplementary material.

The R-NaD loop: Given a trajectory, the reward transform used at turn t is $r^i_{t,\pi_{m,\text{reg}}}(a,\pi) = r^i_t - \eta \log(\frac{\pi(a|o_t)}{\pi_{m,\text{reg}}(a|o_t)})$, if $i = \psi_t$ and $r^i_t + \eta \log(\frac{\pi(a|o_t)}{\pi_{m,\text{reg}}(a|o_t)})$ if $i \neq \psi_t$, starting at the initial policy $\pi_{m=0,\text{reg}}$.

The *dynamics* step of *DeepNash* is composed of two parts, the first part estimates the value function which is done through an adaptation of the v-trace estimator (*44*) to the two-player

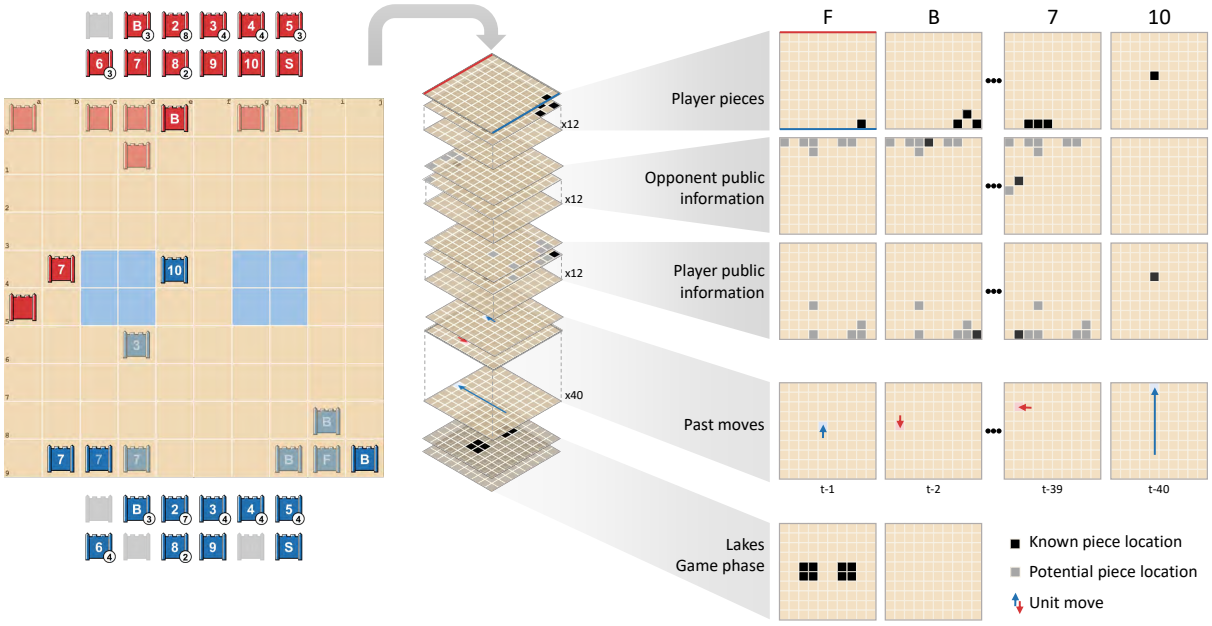


Figure 3: The input of the neural network is a single tensor encoding the position of pieces, the currently known information of both opponent and own pieces (whether a piece moved or was revealed), a limited move history and the position of the lakes.

imperfect information case, and the second part learns the policy through the Neural Replicator Dynamics (NeuRD) update (*45*) using an estimate of the state action value based on the v-trace estimator. These parts are detailed in the supplementary material.

After a fixed number of learning steps, an approximate fixed point policy $\pi_{m,\text{fix}}$ is obtained, which is then used as the next regularisation policy: $\pi_{m+1,\text{reg}} = \pi_{m,\text{fix}}$. The three steps are repeated using a smooth transition from the reward transformation of step m to the one of step $m + 1$.

Fine-tuning : Directly learning with the above-described method leads to convergence to an empirically satisfying solution, which however is slightly distorted by low-probability mistakes. Those mistakes appear because the softmax projection used to compute the policy from the logits assigns a non-zero probability to every action. In order to alleviate this issue we fine-tune during

training by performing additional thresholding and discretization to the action probabilities. The supplementary material provides more details on this and also describes a few additional techniques applied at test-time to remove any remaining obvious mistakes from the policy.

3 Results

In this section we present an overview of the evaluation results of *DeepNash* against both human expert players and current state-of-the-art Stratego bots. For the former we have worked with the Gravon platform, a well-known online games server popular among Stratego players. For the latter we have tested *DeepNash* against eight known AI bots that play Stratego. A detailed analysis is also presented with regard to some of the capabilities of the agent's game-play including deployment, bluffing, and trading off of material vs information.

3.1 Evaluation on Gravon

Gravon is an internet platform for human players, offering several online games, including Stratego. It is by far the largest online platform for Stratego, where some of the strongest players compete (*24*). The Gravon platform uses the same rating system as the International Stratego Federation for the world championship (i.e. the Kleier rating (*46*)).[3] Gravon offers two rankings: one all-time *Classic Stratego ranking* and one *Classic Stratego challenge ranking 2022*. To be included in these rankings, Gravon imposes some limitations to makes sure players are regularly confronted with opponents of comparable strength.

DeepNash was evaluated against top human players over the course of two weeks in the beginning of April 2022, resulting in 50 ranked matches. Of these matches, 42 (i.e. 84%) were won by *DeepNash*. In the *Classic Stratego challenge ranking 2022* this corresponds to a rating of 1799, which resulted in a 3rd place for *DeepNash* of all ranked Gravon Stratego players (the

[3] Similar to the Elo rating system, the Kleier rating system models the win probability between two players from the difference in their rating.

top two ratings are 1868 and 1831). In the all-time *Classic Stratego ranking* this resulted in a rating of 1778 which also puts *DeepNash* in the 3rd place of all ranked Gravon Stratego players (the top two ratings are 1876 and 1823). The rating for this leaderboard considers all ranked games going back to the year 2002.

These results confirm that *DeepNash* reaches a human expert level in Stratego, only through self-play, without bootstrapping from existing human data.

3.2 Evaluation against state-of-the-art Stratego bots

DeepNash was also evaluated against several existing Stratego computer programs: *Probe* was a three-fold winner of the Computer Stratego World Championship (2007, 2008, 2010) (*47*); *Master of the Flag* won that championship in 2009 (*47*); *Demon of Ignorance* is an opensource implementation of Stratego with an accompanying AI bot (*48*); *Asmodeus*, *Celsius*, *Celsius1.1*, *PeternLewis*, and *Vixen* are programs that were submitted in an Australian university programming competition in 2012 (*49*), won by *PeternLewis*.

As shown in Table 1, DeepNash wins the overwhelming majority of games against all of these bots, despite not having been trained against any of them and only being trained using self-play. As such it is not necessarily expected that the residual losses against some of these bots would vanish, even if the exact Nash-equilibrium were reached. For example, in most of the few matches that *DeepNash* has lost against *Celsius1.1*, the latter played a high-risk strategy of capturing pieces early on with a high-ranking piece, and as such was trying to get a significant material advantage. Most often this strategy does not work, but occasionally it can lead to a win.

3.3 Illustration of *DeepNash*'s abilities

DeepNash's only goal during training is to learn a Nash equilibrium policy and by doing so it learns qualitative behavior one could expect a top player to master. Indeed, the agent is able to

Opponent	Number of Games	Wins	Draws	Losses
Probe	30	100.0%	0.0%	0.0%
Master of the Flag	30	100.0%	0.0%	0.0%
Demon of Ignorance	800	97.1%	1.8%	1.1%
Asmodeus	800	99.7%	0.0%	0.3%
Celsius	800	98.2%	0.0%	1.8%
Celsius1.1	800	97.9%	0.0%	2.1%
PeternLewis	800	99.9%	0.0%	0.1%
Vixen	800	100.0%	0.0%	0.0%

Table 1: Evaluation of *DeepNash* against existing Stratego bots. The numbers are reported from *DeepNash*'s point of view. More games (800) were played against bots which we could run automatically. The same number of matches were played as Red and Blue, except against *Master of the Flag* which only plays as Blue.

generate a wide range of deployments which makes it difficult for a human player to find patterns to exploit by adapting their own deployment. We also show situations where *DeepNash* is able to make non-trivial trade-offs between information and material, to execute bluffs and to take gambles when needed. The rest of this section illustrates these behaviors through matches that were played on Gravon.

For convenience, the behavior is described in a way a human observer might naturally interpret it, including terms like "deception" and "bluffing".

3.3.1 Piece deployment

The imperfect information in Stratego arises during the initial phase of the game where both players place their 40 pieces on the board in a secret configuration. As described above, *DeepNash* learns this deployment strategy simultaneously with the regular game-play, and so both strategies co-evolve during learning. Having an unpredictable deployment is important for being unexploitable and indeed *DeepNash* is capable of generating billions of unique deployments. At the same time, not all possible deployments are equally strong (e.g. putting a Flag in the open

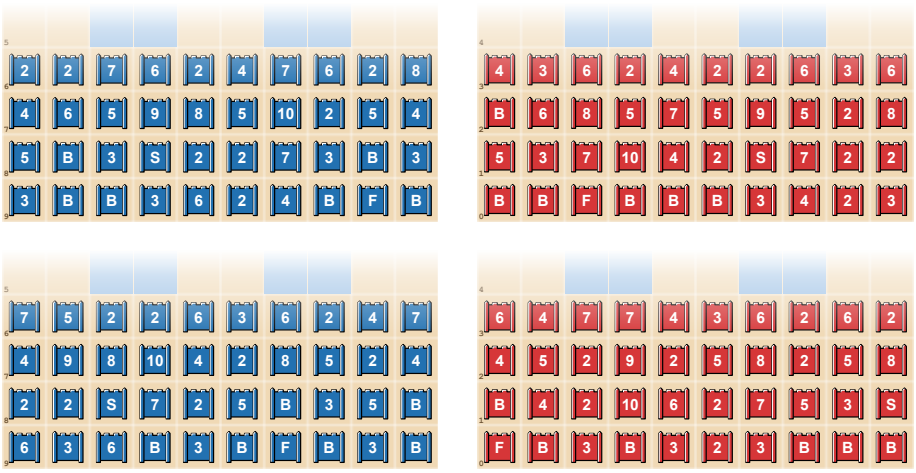
on the front row is bad for obvious reasons) and some often recurring deployment patterns by *DeepNash* are exemplified in Figure 4a.

The Flag is almost always put on the back row, and often protected by Bombs. Occasionally, however, *DeepNash* will not surround the Flag with Bombs. Experts (e.g. Vincent de Boer, 3-fold World Champion) believe that it is indeed good to occasionally not protect the Flag because this unpredictability makes it harder for the opponent in the end-game. Another pattern observed is that the highest pieces, the 10 and 9, are often deployed on different sides of the board. Additionally, the Spy is quite often located not too far away from the 9 (or 8), which protects it against the opponent's 10. *DeepNash* does not often deploy Bombs on the front row, which complies with the behavior seen from strong human players. The 3's (Miner), which can defuse Bombs, are often placed on the back row, which makes sense because their importance typically increases throughout a game as more opponent Bombs and potential Flag positions get revealed. The eight 2's (Scout) are typically deployed both in the front and more in the back, allowing to scout opponent pieces initially but also in later phases of the game.

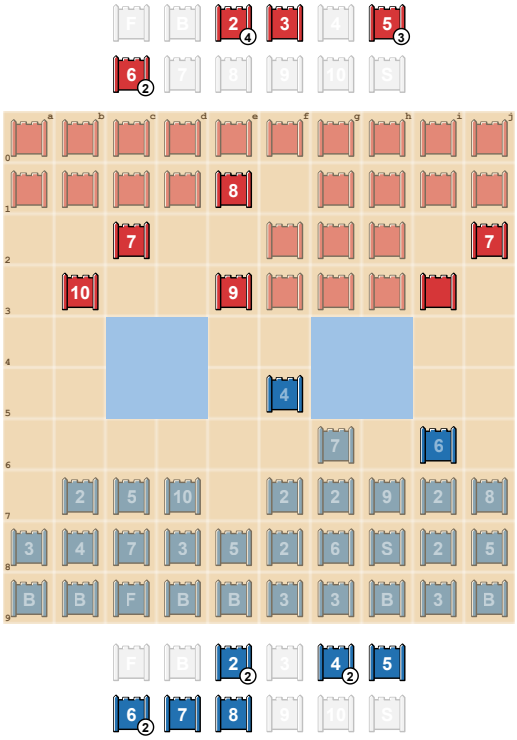
3.3.2 Trade-off between information and material

An important tactic in Stratego is to keep as much information as possible hidden from an opponent in order to gain an advantage. During certain game situations there will be trade-offs to be considered where a player needs to balance the value of capturing an opponent's piece (or even moving a piece), and as such revealing information on their own piece, versus not capturing a piece (or not moving), but keeping the identity of a piece hidden. *DeepNash* is able to make such trade-offs in remarkable ways.

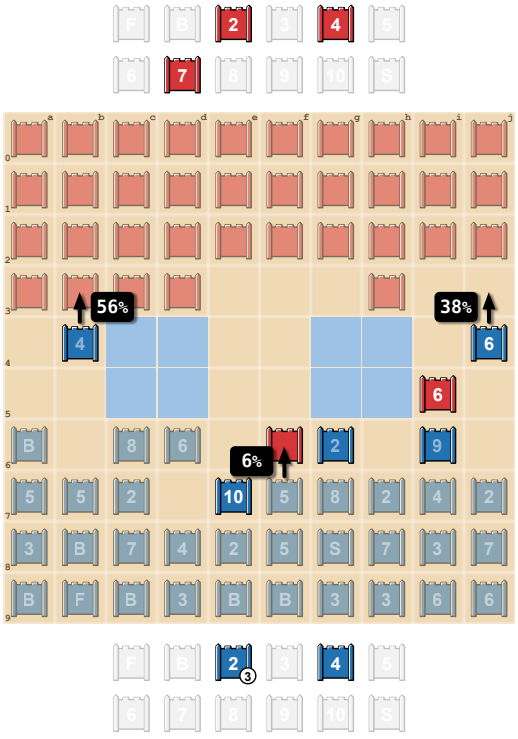
Figure 4b shows a situation where *DeepNash* (in blue) is behind in pieces (it lost a 7 and an 8) but is ahead in information as the opponent in red has its 10, 9, an 8 and two of its 7's revealed. Valuing information and material in Stratego is non-trivial a-priori, but the agent has learned a



(a) Four example deployments *DeepNash* played on Gravon.



(b) While Blue is behind a 7 and 8, none of its pieces are revealed and only two pieces moved. As a result *DeepNash* assesses its chance of winning to be still around 70% (Blue indeed won this match).



(c) Blue to move. *DeepNash*'s policy supports three moves at this state, with the indicated probabilities (the move on the right was played in the actual match). While Blue has the opportunity to capture the opponent's 6 with its 9, this move is not considered by *DeepNash*, likely because the protection of 9's identity is assessed to be more important than the material gain.

Figure 4: Illustration of *DeepNash*'s assessment of the relative value of material versus information in two human (red) - *DeepNash* (blue) matches.

policy through self-play that seems to naturally make this trade-off between information and material. In the above example, *DeepNash* is behind in material but knows the identity of many of the opponents' high-ranked pieces. On the contrary, almost all of *DeepNash*'s remaining pieces have not yet moved and its opponent is left in the blind. The value function ($v = 0.403$) credits this information asymmetry as an advantage for *DeepNash* (with an expected win rate of around 70%) despite having lesser material on the board. This game was won by *DeepNash*.

The second example in Figure 4c shows a situation where *DeepNash* has the opportunity of capturing the opponent's 6 with its 9, but this move is not considered, probably because protecting the identity of the 9 is deemed more important than the material gain. The situation also illustrates the stochasticity of *DeepNash*'s policy during game-play.

3.3.3 Deceptive behavior and bluffing

In addition to being able to value an asymmetry of information, one can also expect the agent to occasionally bluff in order to deceive its opponent and potentially gain an advantage. The situations shown in Figures 5a, 5b and 5c illustrate this ability. In Figure 5a we illustrate *positive bluffing*, in which a player pretends a piece to be of a higher value than it actually is. *DeepNash* chases the opponent's 8 with an unknown piece, a Scout (2), pretending it to be the 10. The opponent believes this piece has a high chance of being the 10 and guides it next to its Spy (which can capture the 10). In an attempt to capture this piece, however, the opponent loses its Spy to *DeepNash*'s Scout.

A second type of bluff, called *negative bluffing*, is shown In Figure 5b, which means that one pretends to be a lower piece as opposed to a positive bluff. Here the movement of the unknown 10 of *DeepNash* is interpreted by the opponent as a positive bluff as they try to capture it with a known 8 assuming *DeepNash* is moving a lower-ranked piece, potentially the Spy, bringing it closer to the opponent's 10. The opponent instead encounters *DeepNash*'s 10 and loses a 8.

A more complex bluff is shown in Figure 5c, where *DeepNash* brings its unrevealed Scout (2) close to the opponent's 10, which can be easily interpreted as a Spy. This tactic actually allows Blue to capture Red's 5 with its 7 a few steps later, thereby gaining material but also preventing the 5 from capturing the Scout (2) and revealing it is actually not the Spy.

4 Conclusion

In this work, we have introduced a novel method called *DeepNash* that learns to play the imperfect information game Stratego from scratch in self-play, up to human expert-level. This model-free learning method combines a deep residual neural network with the game-theoretical *Regularized Nash Dynamics* (R-NaD) multi-agent learning algorithm, without doing any form of search or explicit opponent modelling. As such, *DeepNash* takes an orthogonal approach to state-of-the-art model-based learning methods that have been successfully applied to other complex games such as Go, chess, and imperfect information games such as poker and Scotland Yard, but which, due to their computational toll and the inherent complexity of the Stratego game itself are not applicable to such an elaborate game. *DeepNash* learns both the deployment phase of pieces at the start of the game and the actual game-play itself, end-to-end, in one approach.

The core component behind *DeepNash* is the at-scale implementation of the R-NaD algorithm. *DeepNash* carries out three essential steps in an iteration of the algorithm: *reward transformation* starting from a random regularised policy to define a modified game, subsequently applying the *replicator dynamics* on this modified game to converge to a fixed point policy, and finally *update* the regularization policy to this new fixed point. Repeatedly applying this three-fold process empirically demonstrates convergence of the learning algorithm to an ϵ-Nash equilibrium in Stratego.

We thoroughly evaluated *DeepNash* against eight state-of-the-art Stratego bots and against human-expert Stratego players on the Gravon platform. Against other AI bots we achieve a

minimum win-rate of 97%, and in the evaluation against human-expert players we achieve an overall win-rate of 84% on the Gravon platform, which places us in the top-3 rank of both the 2022 and all-times leaderboards. This is a remarkable result that the Stratego community did not believe would have been possible with current techniques, viz. quotes by Thorsten Jungblut (owner of the Gravon platform) and Vincent de Boer, which can be found in the supplemental material. Since June 2021, we have worked on *DeepNash* with Vincent de Boer (co-author on this paper), a former three times Stratego world champion, currently ranked 4^{th} on the official world ranking. Vincent has helped evaluating *DeepNash* and detecting weaknesses in its consecutive versions, which fed back into agent improvements.

In conclusion, we believe that *DeepNash* can unlock further applications of RL methods in real-world multi-agent problems with astronomical state spaces, characterized by imperfect information, that are currently out of reach for state-of-the-art AI methods to be applied in an end-to-end fashion.

Acknowledgments

We would like to thank several colleagues for their feedback on this manuscript and advice during the development of *DeepNash*: Laurel Prince, Aja Huang, Martin Schmidt, Michael Bowling, Georg Ostrovski, Martin Riedmiller, Koray Kavukcuoglu, Wojtek Czarnecki, Sukhdeep Singh. We would also like to thank Thorsten Jungblut, the Gravon platform owner, for giving us the opportunity to have access to Gravon, evaluate the *DeepNash* agent against human expert players, and for his support, allowing us to use matches played online for illustration in this manuscript.

Related Work

Regret Minimization and Regularization in Games

One general class of methods for (approximately) solving games uses self-play of regret-minimizing algorithms. An algorithm is said to minimize regret if the difference between the average value of the sequence of actions it generates, and that of the best alternate in hindsight, approaches zero over time. There are different regret measures based off of different sets of alternatives, with the simplest being external regret which considers static alternatives (*50*). That is, in a repeated environment with some set of actions A with value $v_t(a)$ for action a at time $t \in \mathbb{N}$ and a sequence of policies π_t, the external regret is $R_T := \max_{a \in A} \sum_{t=1}^{T} (v_t(a) - \pi_t \cdot v_t)$. An external-regret minimizing algorithm, like the regret-matching algorithm (*51*), often has a guarantee of the form $R_T \leq k\sqrt{T}$ for some constant k, which implies $\lim_{T \to \infty} \frac{R_T}{T} = 0$. In the context of zero-sum two-player games, minimizing regret is useful because it is well known that if two regret-minimizing algorithms play against each other, their average policies $1/T \sum_t \pi_t$ approach the set of Nash equilibrium strategy profiles (see for example, (*52*)). However, the action set A grows exponentially in the number of information sets, so game-specific regret-minimizing algorithms are needed for larger extensive form games, with sequential decisions.

The Counterfactual Regret Minimization algorithm (*53*) applies regret-matching independently at each information set, with a proof that minimizing regret at all information sets also minimizes external regret. The simplicity and computational efficiency of CFR has led to many variants, like the sampling variant MCCFR (*54*), Pure CFR (*55*), CFR+ (*56*), and Discounted CFR (*57*). However, CFR is a tabular method that stores regret information and action probabilities for all information sets in the game, so is limited to games which can fit in storage. Using compression, tabular CFR variants have been applied to games with up to 10^{14} information sets (*56*), but are incapable of scaling up over a hundred orders of magnitude to deal with games

like Stratego.

Follow the Regularized Leader (FoReL) is another approach to online learning. The motivation for FoReL comes from examining the behaviour of the simple rule of Follow the Leader, which looks back to find the historically best decision and follows that choice at the current time. While this is a natural and perhaps intuitive approach to sequential interactions, Follow the Leader generally lacks any interesting guarantee on performance. To get such a guarantee, there are a number of different intuitions which lead to the idea of adding a bonus (or penalty depending on the point of view) to the historically-observed values, like dampening the response or adding a strongly-convex term to the optimisation. The resulting approach, of following the historically-best decision when including an additional regularization term, describes the class of follow the regularized leader (FoReL) algorithms. Without game-specific approaches or function approximation, FoReL has the same limitations as regret-matching: the direct application would be on an exponentially large space of actions, and it is a tabular method that tracks historical values for all actions.

Reinforcement Learning in Two-Player Zero-Sum Games

Reinforcement learning methods have been applied to two-player zero-sum games going back to TD-Gammon (*58*). However until recently, most successful methods were limited to the perfect information case (*58–62*). New deep RL methods based on regret minimization, best responses, and policy gradients have shown success in imperfect-information games such as poker.

The first category of deep RL algorithms are based on regret minimization techniques such as counterfactual regret minimization (CFR) (*53*). Deep CFR (*63*) approximates CFR by training a regret network on a buffer of counterfactual values. However, Deep CFR uses external sampling, which may be impractical for games with a large branching factor such as Stratego and Barrage Stratego. DREAM (*64*) and ARMAC (*65*) are model-free regret-based deep learning approaches

that can take advantage of outcome sampling and can therefore scale to large games. However, they rely on importance sampling terms to remain unbiased, and this importance weight might become very large in games with a long horizon such as Stratego. Furthermore, all these techniques when generalized to using neural networks require generating an average strategy which is either memory heavy (as one needs to store all strategies all iterations to get an exact average) or error prone when using approximation (DeepCFR for instance use a supervised learning step to approximate the average).

The second category of deep RL algorithms for two-player zero-sum games include best-response techniques. Generally, best response techniques iteratively train a best response via reinforcement learning every iteration. Neural Fictitious Self Play (NFSP) (*66*) approximates extensive-form fictitious play by progressively training a best response against an average of all past policies using off-policy reinforcement learning. The average policy is a neural network that is trained to imitate the average of the past best responses. Policy Space Response Oracles (PSRO) (*67*) iteratively adds a reinforcement learning best response to a population of policies, one for each player. The best response is computed by training against a meta-distribution over the current population policies. This meta-distribution is computed by finding a Nash equilibrium of the empirical game matrix formed by considering each policy as a pure strategy in a normal form game. AlphaStar (*28*) beat top humans at Starcraft using a method inspired by PSRO, where several agents were training at the same time against a dynamically-updated meta-distribution over all policies. Similarly, OpenAI Five (*68*) beat top humans at Dota using a mixture of self-play and a dynamically-updated meta-distribution over past policies. Finally, a similar population-based method combined with population-based hyperparameter optimization has achieved human-level performance on Capture the Flag (*69*). Despite these successes, best response techniques remain memory-intensive because they potentially require an exponential (in the number of distinct states) number of different policies to represent an optimal policy,

which, combined with the time required to compute a best-response, also makes them very slow.

The third category of deep RL algorithms, and where this work falls under, is policy gradient methods. Regret Policy Gradient (RPG) (*70*) approximates CFR via a weighted policy gradient, but is not proven to converge to a Nash equilibrium. Neural Replicator Dynamics (NeuRD) (*71*) approximates Replicator Dynamics with a policy gradient and is proven to converge to a Nash equilibrium in the time average. Prior to this work, neither of these algorithms have been applied to large-scale domains, or have demonstrated human-level performance; this work uses NeuRD combined with the regularization idea laid out in (*34*) to converge in the last iterate.

Work on Barrage Stratego

Recently, a smaller variant of Stratego (Barrage Stratego Barrage) has seen progress from a reinforcement learning perspective. Current Barrage Stratego bots are based on imperfect information tree search and are unable to beat intermediate-level human players (*72, 73*). Pipeline PSRO (*20*) was able to beat these handcrafted bots in Barrage Stratego (by at most 81% win-rate), but didn't show results against top human players.

Methods: additional information

Details on R-NaD in Two-Player Zero-Sum Normal Form Games

We now concisely describe the above process more formally in the context of NFGs. In a two-player zero-sum normal-form game, player 1 and 2 simultaneously play actions $a^1 \in A^1$ and $a^2 \in A^2$ with policies $\pi^1(a^1)$ and $\pi^2(a^2)$. As a result, player 1 receives a reward $r^1(a^1, a^2)$ and player 2 will receive the opposite reward $r^2(a^1, a^2) = -r^1(a^1, a^2)$ (due to the zero-sum nature of the game). The *reward transformation* step of R-NaD is defined based on a regularization policy $\pi_{\text{reg}} = (\pi^1_{\text{reg}}, \pi^2_{\text{reg}})$ which modifies the reward as follows: $r^i(\pi^i, \pi^{-i}, a^i, a^{-i}) = r^i(a^i, a^{-i}) -$

$\eta \log(\frac{\pi^i(a^i)}{\pi^i_{\text{reg}}(a^i)}) + \eta \log(\frac{\pi^{-i}(a^{-i})}{\pi^{-i}_{\text{reg}}(a^{-i})})$, with $\eta > 0$ a regularization parameter.[4] The initial regularization policy can be chosen arbitrary (as long as all actions have a non-zero probability).

The *dynamics* step of R-NaD determines a new policy $\pi_{\text{fix}} = (\pi^1_{\text{fix}}, \pi^2_{\text{fix}})$ derived from convergence to a fixed point of the modified game by applying the replicator dynamics (*38–40, 74*). Replicator dynamics are a learning process that are also known as an instance of Follow the Regularized Leader (*37*), which are defined as follows:

$$\frac{d}{d\tau}\pi^i_\tau(a^i) = \pi^i_\tau(a^i)[Q^i_{\pi_\tau}(a^i) - \sum_{b^i} \pi^i_\tau(b^i) Q^i_{\pi_\tau}(b^i)], \text{with} \quad Q^i_{\pi_\tau}(a^i) = \mathbb{E}_{a^{-i}\sim\pi^{-i}_\tau}\left[r^i(\pi^i_\tau, \pi^{-i}_\tau, a^i, a^{-i})\right]$$

The replicator dynamics are a descriptive learning process from evolutionary game theory that aims to reinforce the probability of the actions with a high fitness $Q^i_{\pi_\tau}(a^i)$ (Q standing for *Q*uality of an action), and decreasing the probability of the actions with low fitness (*38–41, 74*). As such it measures the expected payoff for an action vs. that of the average of all actions. If an action performs better than average its probability will increase, otherwise decreasing. This dynamical system has a fixed point π_{fix} and convergence to it is guaranteed through the Lyapunov function $H_{\pi_{\text{fix}}}(\pi) = \sum_{i=1}^{2} \sum_{a^i \in A^i} \pi^i_{\text{fix}}(a^i) \log\left(\frac{\pi^i_{\text{fix}}(a^i)}{\pi^i(a^i)}\right)$. In this case it is a strong Lyapunov function of the system (in fact $\frac{d}{d\tau} H_{\pi_{\text{fix}}}(\pi_\tau) \leq -\eta H_{\pi_{\text{fix}}}(\pi_\tau)$) which means that the distance to the fixed point will decrease exponentially to 0.

Finally, in the *update* step of R-NaD we use the previously-obtained fixed point as the regularization policy of the next iteration. So the whole process can be described as follows using an extra iteration index: one starts with an arbitrary regularisation policy $\pi_{0,\text{reg}}$. Then given any such regularization policy $\pi_{n,\text{reg}}$, we compute the fixed point $\pi_{n,\text{fix}}$ under the replicator dynamics of the game with transformed reward. Finally we choose $\pi_{n+1,\text{reg}} = \pi_{n,\text{fix}}$ and start the next iteration. This process generates a sequence of fixed points $n \to \pi_{n,\text{fix}}$, which is known to converge to the Nash equilibrium[5], π_{nash}, of the original game as the sequence of distances to

[4] We follow the convention of denoting the opponent of player i as $-i$.

[5] For simplicity we assume the Nash equilibrium is unique.

this Nash equilibrium $n \to \sum_{i=1}^{2} \mathrm{KL}(\pi^i_{\text{nash}}, \pi^i_{n,\text{fix}})$ can be proven (*34*) to decrease to 0 where KL is the Kullback–Leibler divergence.

The regularization parameter η is fixed throughout this process. Its value has two effects on the dynamics step of R-NaD: on the one hand a higher value gives more stable and faster convergence to the fixed point. On the other hand a higher η results in a fixed point that is more biased towards the regularization policy, which means one might need more overall iterations to approach the Nash equilibrium sufficiently close.

DeepNash: R-NaD at Scale

As a reminder, *DeepNash* consist of three components: (1) the core component is R-NaD, the model-free RL algorithm implemented using a deep Neural Network, (2) fine-tuning of the learnt policy to reduce the residual probabilities of taking highly improbable actions and, (3) test-time improvements to avoid remaining obvious mistakes.

We first lay out some essential background information on imperfect information games necessary to understand how R-NaD is scaled with a deep neural network. Then we continue to unpack the three algorithmic steps of R-NaD and detail how they are implemented in the neural architecture.

Imperfect Information Games

In a two-player Imperfect Information Game, each player (player 1 and player 2) play in turn starting from an initial history h_{init}. The set $\mathcal{H}$ is the set of all histories and $\mathcal{A}$ is the set of all possible actions. In each history $h \in \mathcal{H}$, the current player takes an action $a \in \mathcal{A}$. As a result of this action a, the player i receives a reward $r^i(h, a) \in \mathbb{R}$ and the history is updated to $h' = ha$ (the concatenation of both history h and action a). For any given history h the player's turn is noted $\psi(h) \in \{1, 2\}$. The information state $x(h)$ is the set of all histories $h' \in x(h)$, which are

indistinguishable from h from player $\psi(h)$'s point of view. We consider the information set to be of perfect recall (there is as much information in the information set $x(h)$ as in the sequence of information sets that were seen by the player until history h). Each player's goal is to produce a policy $\pi^i(a|o)$ where o is the observation given to the player at history h (we overload the notation and also write $o(h)$ the observation function). As described above we will also consider policy dependent rewards $r^i(\pi, h, a)$ used by R-NaD.

For a given joint policy $\pi = (\pi^1, \pi^2)$ the value of an history h for player i is :

$$v^i_\pi(h) = \sum_{a \in \mathcal{A}} \pi^{\psi(h)}(o(h)) \left[r^i(\pi, h, a) + v^i_\pi(ha) \right]$$

. The Q-function of a policy π is defined as $Q^i_\pi(h, a) = r^i(\pi, h, a) + v^i_\pi(ha)$. This function expresses how good it is to take action a from history h. In addition, for a given policy the reach probability of a history h, $\rho_\pi(h)$, is defined recursively $\rho_\pi(ha) = \pi^{\psi(h)}(a|o(h))\, \rho_\pi(h)$, expressing how probably it is for history h to occur. The value given an observation is defined as $v^i_\pi(o) = \frac{\sum_{h \in o} \rho_\pi(h) v^i_\pi(h)}{\sum_{h \in o} \rho_\pi(h)}$.

In a self-play, model free reinforcement learning setting, the agent plays against itself using the policy π starting from $h_0 = h_{init}$. at each time t the player samples an action a_t according to $\pi(.|o_t)$ (with $o_t = o(h_t)$ and $r^i_t = r^i(h_t, a_t)$) and the state becomes $h_{t+1} = h_t a_t$ at time $t+1$. The following trajectories are collected $\mathrm{T} = [(o_t, a_t, (r^1_t, r^2_t), \mu_t(.) = \pi(.|o_t)), \psi_t = \psi(h_t)]_{0 \le t < t_{\text{max}}}$ and are the only information we will use during training. We also write $t_{\text{effective}}$ the effective length of the trajectory if it ended up finishing before t_{max}.

Model-free Reinforcement Learning with Regularized Nash Dynamics

Again, as a reminder, *DeepNash* is essentially the R-NaD algorithm at scale using a deep neural network. As in NFGs, it is done in 3 steps: (1) the *reward transformation step*, (2) the *dynamics step* which is used to empirically converge to a fixed point, and (3) the *update step* in which the algorithm updates the policy used to define the regularization function.

R-NaD's learning update generates a sequence of policy and value parameter θ_n indexed by n, and of target parameter $\theta_{n,\text{target}}$ (the policies are written π_{θ_n} and $\pi_{\theta_{n,\text{target}}}$ and the values are written v_{θ_n} and $v_{\theta_{n,\text{target}}}$). The policy π_θ is defined as $\pi_\theta(.|o) = \frac{\exp(l_\theta(.|o))}{\sum_b \exp(l_\theta(b|o))}$ ($l_\theta(.|o)$ is called the logit of the policy).

***Transformation* of the reward:** The reward transformation[6] of R-NaD is done over an interval of size Δ_m. It is based on the policy dependent reward $r^i_{\pi_{m,\text{reg}}}(\pi, h, a) = r^i(h, a) + (1 - 2 \times \mathbf{1}_{i=\psi(h)})\eta \log(\frac{\pi(a|o(h))}{\pi_{m,\text{reg}}(a|o(h))})$ starting with $\pi_{-1,\text{reg}} = \pi_{0,\text{reg}}$. At each step $n \in [0, \Delta_m]$ the reward transformation is an interpolation between $r^i_{\pi_{m,\text{reg}}}$ and $r^i_{\pi_{m-1,\text{reg}}}$. The reward at step n is defined as : $r^i_{\text{reg},n}(\pi, h, a) = \alpha_n r^i_{\pi_{m,\text{reg}}}(\pi, h, a) + (1 - \alpha_n) r^i_{\pi_{m-1,\text{reg}}}(\pi, h, a)$ (with $\alpha_n = \min(1, 2 \times \frac{n}{\Delta_m})$) in order to smooth the transition between a regularization policy to another (also $\pi_{m,n,\text{reg}} = \alpha_n \pi_{m,\text{reg}} + (1 - \alpha_n)\pi_{m-1,\text{reg}}$). We write T_n for the trajectory with the transformed reward at step n.

The *Dynamics* step at scale of the R-NaD algorithm is composed of two parts, one concerns the estimation of the value function, and the second part concerns learning update of the Q-function and of the policy.

***Dynamics* : estimators for the value function.** The fixed point $\pi_{m,\text{fix}}$ associated to the regularization policy $\pi_{m,\text{reg}}$ is learnt over Δ_m steps using two learning updates: (1) an update to learn a value function and generate an estimate of the Q-function and (2) an update to learn a policy from the estimated Q-function.

It is challenging in Stratego to generate a good estimate of a Q-function as the action space is vast with 3600 possible actions at every step, even if not all are legal actions (e.g. a trapped

[6]The reward transformation used to train *DeepNash* is based on the theory developed in (*34*) for imperfect information zero-sum games. It describes both the convergence properties and the Lyapunov functions used in sequential imperfect information games. Here we demonstrate how to practically scale these principles to learn in zero-sum games at scale.

piece that cannot move). The estimators at time t and at learning step n of the value function $\hat{v}^i_{t,n}$ and of the Q-function $\hat{Q}^i_{t,n}$ for player i adapt the v-trace estimator (*44*) to the two player case. It uses information on the future steps in order to create a low variance and low bias estimator of the value and the Q-function of the policy π_{θ_n}, even if the trajectory was generated with a different policy.

This recursive backward process takes as an input a joint policy π, a joint value v, and a trajectory T and output $\hat{v}^1_t$, $\hat{v}^2_t$, $\hat{Q}^1_t$ and $\hat{Q}^2_t$ (we write $\hat{v}^1_t, \hat{v}^2_t, \hat{Q}^1_t, \hat{Q}^2_t = \Upsilon(\mathrm{T}, \pi, v)$). Given a trajectory $\mathrm{T} = [(o_t, a_t, (r^1_t, r^2_t), \mu_t(.), \psi_t = \psi(h_t)]_{0 \leq t < t_{\max}}$ (with transformed rewards) the backward update is defined as follow for all player in $[1, 2]$ starting from $t = t_{\text{effective}}$ (and $\hat{v}^i_{t_{\text{effective}}+1} = 0$, $V^i_{\text{next},t_{\text{effective}}+1} = 0$, $\hat{r}^i_{t_{\text{effective}}+1} = 0$ and $\xi_{t_{\text{effective}}+1} = 1$) :

if $i \neq \psi_t$:

$$\hat{v}^i_t = \hat{v}^i_{t+1},\ V^i_{\text{next},t} = V^i_{\text{next},t+1},\ \hat{r}^i_t = r^i_t + \frac{\pi(a_t|o_t)}{\mu_t(a_t)}\hat{r}^i_{t+1},\ \xi_t = \frac{\pi(a_t|o_t)}{\mu_t(a_t)}\xi_{t+1} \tag{1}$$

if $i = \psi_t$:

$$\hat{v}^i_t = v(o_t) + \delta_t V^i + c_t(v^i_{t+1} - V^i_{\text{next},t+1}),\ V^i_{\text{next},t} = v(o_t) \tag{2}$$

$$\delta_t V^i = \rho_t(r_t + \frac{\pi(a_t|o_t)}{\mu_t(a_t)}\hat{r}^i_{t+1} + V^i_{\text{next},t+1} - v(o_t)),\ \hat{r}^i_t = 0,\ \xi_t = 1 \tag{3}$$

$$\rho_t = \min(\bar{\rho}, \frac{\pi(a_t|o_t)}{\mu_t(a_t)}\xi_{t+1}),\ c_t = \min(\bar{c}, \frac{\pi(a_t|o_t)}{\mu_t(a_t)}\xi_{t+1}) \tag{4}$$

$$\begin{aligned}\hat{Q}^i_t(a) = &-\eta \log(\frac{\pi_{\theta_n}(a|o_t)}{\pi_{m,n,\text{reg}}(a|o_t)}) \\ &+ \frac{\mathbf{1}_{a=a_t}}{\mu_t(a_t)}\left(r^i_t + \eta \log(\frac{\pi_{\theta_n}(a|o_t)}{\pi_{m,n,\text{reg}}(a|o_t)}) + \frac{\pi(a_t|o_t)}{\mu_t(a_t)}(\hat{r}^i_{t+1} + \hat{v}^i_{t+1}) - v(o_t)\right) + v(o_t)\end{aligned} \tag{5}$$

These estimator are computed over the full trajectory from the end to the beginning without any bootstrapping. This ensures that these estimates have a minimum bias.

***Dynamics* : learning update of the Q-function and of the policy.** First we use $\hat{v}^1_{t,n}, \hat{v}^2_{t,n}, \hat{Q}^1_{t,n}, \hat{Q}^2_{t,n} = \Upsilon(\mathrm{T}_n, \pi_{\theta_{n,\text{target}}}, v_{\theta_{n,\text{target}}})$ as estimate to compute the value and policy update.

The value is learned through a regression loss written $l_{\text{critic}}(\theta) = \sum_{i=1}^{2} \frac{1}{t_{\text{effective}}} \sum_{t=0}^{t_{\text{effective}}} \mathbf{1}_{i=\Psi_t} \| v_\theta(o_t) - \hat{v}_t^i \|$ and the policy is updated through the Neural Replicator Dynamics (NeuRD) loss (*45*) and the update direction is defined as:

$$\Lambda_n = -\left[\text{lr}_n \nabla l_{\text{critic}}(\theta_n) + \sum_{i=1}^{2} \frac{1}{t_{\text{effective}}} \sum_{t=0}^{t_{\text{effective}}} \sum_{a} \hat{\nabla}\theta(l_{\theta_n}(a, o_t) \text{Clip}\left(Q_{t,n}^{\psi_t}(a, o_t), c_{\text{clip NeuRD}} \right), \text{lr}_n, \beta) \right]$$

with $\hat{\nabla}\theta(z(\theta), \eta, \beta) = \eta \nabla\theta z(\theta) \mathbf{1}_{z(\theta + \eta \nabla\theta z(\theta)) \in [-\beta, \beta]}$, $\text{Clip}(., c) = \min(\max(., -c), c)$ and finally the parameters are updated through an adam optimizer :

$$\theta_{n+1} = \text{Adam}(\theta_n, \text{Clip}(\Lambda_n, c_{\text{clip gradient}}), b_{1,\text{adam}}, b_{2,\text{adam}}, \epsilon_{\text{adam}})$$

and,

$$\theta_{n+1,\text{target}} = \gamma_{\text{averaging}} \theta_{n+1} + (1 - \gamma_{\text{averaging}}) \theta_{n,\text{target}} \text{ with } \gamma_{\text{averaging}} \in [0, 1[$$

Updating **the transformed reward and the learning parameters.** After the dynamics steps of the algorithm is completed we define the new fixed point policy as $\pi_{m,\text{fix}} = \pi_{\theta_{n=\Delta_m},\text{target}}$. The next regularisation policy is defined as : $\pi_{m+1,\text{reg}} = \pi_{m,\text{fix}}$ and we go on to the next step $(m + 1)$ starting from the parameters θ_0, $\theta_{0,\text{target}}$ and the state of the optimizer being the ones we finished the iteration m with. the new reward transform at step $m + 1$ interpolates between $r^i_{\pi_{m+1,\text{reg}}}$ and $r^i_{\pi_{m,\text{reg}}}$.

Fine-tuning : Learning only with the previous method is enough to converge to an empirically satisfying solution but limited by low probability mistakes. Those mistakes appear because the softmax projection used to compute the policy from the logits assigns non-zero probability to every action. Although individually rare, an opponent who prolongs the game by avoiding to get his piece captured and making a long series of neutral waiting moves will eventually benefit from one of these low-probability errors. In order to alleviate this issue we fine-tune the training with a different projection that thresholds and discretizes the action probabilities. The policy is written

$\pi_{\theta,\epsilon_{\text{tres}},n_{\text{disc}}}$ where ϵ_{tres} is the threshold level and n_{disc} is the number of probability quanta used. This new projection of the policy is used to define new value function estimate and Q-functions estimate as $\hat{v}^1_{t,n}, \hat{v}^2_{t,n}, \hat{Q}^1_{t,n}, \hat{Q}^2_{t,n} = \Upsilon(\mathrm{T}_n, \pi_{\theta_{n,\text{target}}\epsilon_{\text{tres}},n_{\text{disc}}}, v_{\theta_{n,\text{target}}})$ that will be used instead of the previous estimate without any change to the rest of the 3 steps of the algorithm.

The parameters used to train *DeepNash* are summarized in Table 2.

Infrastructure for learning

The IMPALA architecture (*44*) is adapted to the needs of the R-NaD algorithm, namely storing full episodes in the replay buffer. The learner reads a batch of full episodes, splits them into the sequence of ordered chunks of fixed length (mini batches), and computes the parameter updates on that sequence in a reverse order. This allows computing the exact full returns and to carry over the necessary information between mini-batches (see R-NaD algorithm above).

Game Rules and Neural Network Input Representation

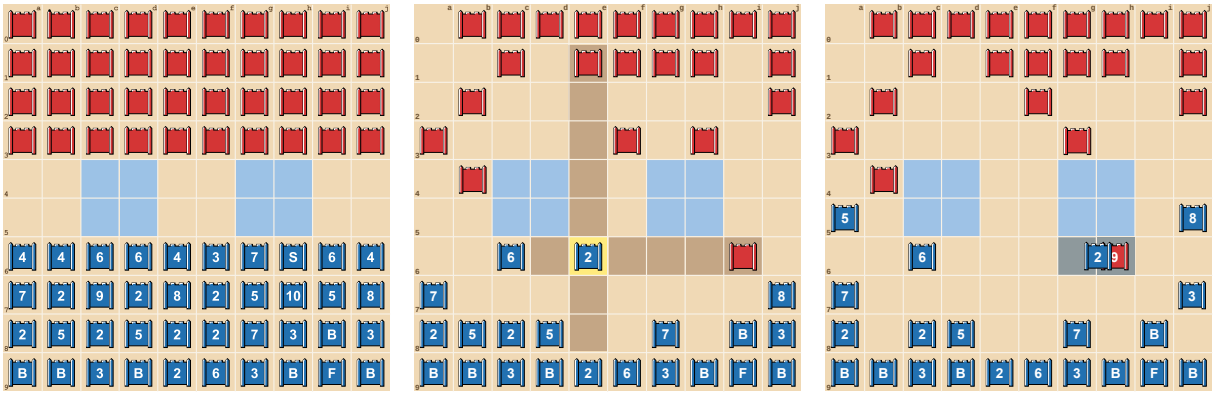
Basic Rules

Stratego is a two-player board game, played between red and blue, corresponding to the colors of their pieces. The game board is a 10×10 grid of squares, with two 2×2 'lakes' which pieces may not move on or through - shown in light blue in Figure 6. Each player starts with 40 pieces of 12 different types shown in the table below:

parameter	value
Reward transform:	
η	0.2
Δ_m	10k for $m \leq 100$, 100k for $100 < m \leq 165$, 35k for $m > 165$
max number of steps	7.21M steps
lr_n	0.00005
$c_{\text{clip gradient}}$	10000
NeuRD parameters:	
β	2.0
$c_{\text{clip NeuRD}}$	10000
adam parameters:	
$b_{1,\text{adam}}$	0.0
$b_{2,\text{adam}}$	0.999
ϵ_{adam}	10^{-8}
target network parameters:	
$\gamma_{\text{averaging}}$	0.001
v-trace parameters:	
$\bar{\rho}$	1.0
$\bar{c}$	1.0
Trajectory parameters:	
t_{max}	3600
Batch size (number of trajectories per step)	768
Fine tuning parameters	
ϵ_{tres}	0.03
n_{disc}	32

Table 2: Parameters used during the training of *DeepNash*.

type	symbol	name	count
0	F	Flag	1
1	S	Spy	1
2	2	Scout	8
3	3	Miner	5
4	4	Sergeant	4
5	5	Lieutenant	4
6	6	Captain	4
7	7	Major	3
8	8	Colonel	2
9	9	General	1
10	10	Marshal	1
11	B	Bomb	6



(a) (b) (c)

Figure 6: The phases of a Stratego game: (a) the deployment phase during which players put their pieces on the board in a private configuration. (b) the game phase, where players alternate turns in moving pieces. Here a player chose to attack an opponent piece with the Scout (2), taking advantage of the long-range Scout action (c). As a result, both pieces are revealed; the Scout (2) is defeated by the General (9) and removed from the board.

The game consists of two phases: the deployment phase, and a play phase. During the deployment phase — as shown in Figure 6a – both players independently and privately position their 40 pieces on their side of the board in a 4×10 rectangular area, in any configuration they choose. In the play phase, the players take turns to move one of their pieces, starting with the red player. The goal of the game is to either capture the opponent's Flag or capture all of their movable pieces.

On each player's turn, the current player moves one of their pieces either to an empty square or to a square occupied by one of the opponent's pieces, attacking it. It is not possible to move onto a lake or onto a square already occupied by one of the player's own pieces. Pieces can only move up, right, down, or left and cannot move diagonally. All pieces besides the Scout, Flag, and Bomb can move only one square at a time to one of their adjacent squares. The Scout can move through any number of empty spaces in one of the four directions. The Flag and Bombs remain static and as such cannot be moved.

When an attack is initiated, both players reveal their piece's type. The higher-valued piece remains on the board while the lower-valued piece is captured and removed from the board. If the pieces are of equal value, both are removed. There are two exceptions to this rule: Miners are able to capture Bombs, and Spies are able to capture Marshals if the Spy attacks the Marshal. A more detailed description of game rules, including provisions against repetition and endless chases, can be found in the official rule book (*21*).

Drawing rules

Under normal game rules, players can declare a game a draw by mutual agreement. We do not offer such negotiation actions to our agents, but instead trained and evaluated with the following (non-standard) rules: a game automatically ends in a draw either after 2000 total moves, or 200 consecutive moves have been made without any piece being attacked.

Agent action space

The action space for the agent is discrete and of size 100 throughout the game, each action corresponding to a square on the board. The interpretation of an action and the definition of which actions are legal in a certain state depends on the phase of the game.

During the deployment phase, the player needs to 'play' exactly 40 actions, one for each piece that is being deployed on the board as shown in Figure 6a. The piece deployment order

is fixed and sorted by piece-type: Flag (F), Bomb (B), Marshal (10), General (9), Colonel (8), Major (7), Captain (6), Lieutenant (5), Sergeant (4), Miner (3), Scout (2) and Spy (S). Thus each player first deploys their Flag, then their six Bombs, etc.

Deployment ends when both players have deployed all their pieces, entirely filling their four deployment rows. The play phase then commences with red making the opening move.

During the play phase, a full move is decomposed into two actions. First, the agent selects the current location of one of its own pieces. Second, the agent selects a legal destination location for the selected piece as shown in Figure 6b. If the new location is empty, the selected piece is moved there. If the new location contains an opponent piece, it initiates an attack, which is resolved as described in the section on the game rules, and illustrated in Figure 6c. Note that this two-step action decomposition affects both the observation and the agent design as detailed in the following subsections.

Neural Network Input Representation

When played as a physical board game, human players observe the game as a sequence of board positions where they see both the location and type of their own pieces, but only the location of the opponent's pieces. The type of a piece is disclosed when it attacks or is attacked, but this needs to be memorized by the player. Similarly, when a piece moves, the information that it cannot be a Bomb or a Flag must be deduced by the human players and memorized.

In common with many online versions of the game, we chose to present *DeepNash* with richer observations which include the most important pieces of information available from the prior play. For example, if an opponent piece has moved by more than square, the network inputs will reflect that it could only be a Scout. Note that these deductions are policy independent, and exclude heuristics which depend on the opponent's policy and other higher level strategic choices.

Let a **piecetype assignment** for player i be an assignment of a specific piecetype to every piece of that player still on the board. It can be represented by a $10 \times 10 \times 12$ tensor, where the two leading dimensions correspond to a row (r) and column (c) of the Stratego board and such that $T_i[r, c, t] = 1$ if the square (r, c) holds a piece of color i of type t, else the value is 0. For instance, $T_1[5, 6, 1]$ indicates that player red ($i = 1$) has a Spy ($t = 1$) at location $(5; 6)$

The **private information tensor** of player i, $\mathbf{Prv}_i$ is the piecetype assignment tensor of that player corresponding to the actual state of the board for that player.

The **public information tensor** of player i, $\mathbf{Pub}_i$, is a $10 \times 10 \times 12$ tensor such that $\mathbf{Pub}_i[r, c, t]$ is the probability that square (r, c) holds a piece of player i of type t, under uniform sampling of piecetype assignments consistent with the move sequence of the game so far. The requirement here is just that the move sequence would be legal, not that it would be rational or in accordance with some policy.

This public information tensor has the property that $\sum_t \mathbf{Pub}_i[r, c, t] = 1$ for each square (r, c) occupied by i, and that $\sum_{r,c} \mathbf{Pub}_i[r, c, t]$ is the number of pieces of player i of type t still on the board. Given there are only two categories of unknown pieces (moved and non-moved), this public information tensor can be computed efficiently as follows:

$$\mathbf{Pub}_i[r, c, t] = \begin{cases} 0 & \text{if there is no piece belonging to player } i \text{ at position } (r, c) \\ 1 & \text{if the piece at } (r, c \text{ is known to have type } t \\ \frac{\#\text{unrevealed}(t)}{\sum_{k \neq \text{B,F}} \#\text{unrevealed}(k)} & \text{if the piece at } (r, c) \text{ has ever moved and } t \neq \text{B, F} \\ \frac{\#\text{unrevealed}(t)}{\sum_k \#\text{unrevealed}(k)} & \text{if the piece at } (r, c) \text{ has never moved} \end{cases} \tag{6}$$

Where $unrevealed(t)$ is the number of pieces of player i of type t which are still on the board and where the type of the piece hasn't been revealed by an attack or by making a Scout move.

A move m during a game observed by player i is encoded as a 10×10 tensor $\mathbf{Mov}_i^m$ such

observation component	shape
The lakes on the board, where a square that is a lake has value 1, otherwise 0.	10×10
The player's own private information $\mathbf{Prv}_i$	$10 \times 10 \times 12$
The opponent's public information $\mathbf{Pub}_{-i}$. Contains all 0's during the deployment phase.	$10 \times 10 \times 12$
The player's own public information $\mathbf{Pub}_i$: this informs i on the information $-i$ has on i's pieces. Contains all 0's during the deployment phase.	$10 \times 10 \times 12$
An encoding of the last 40 moves: $\mathbf{Mov}_i^m$ for each move made up to 40 steps ago.	$10 \times 10 \times 40$
The ratio of the game length to the maximum length before the game is considered a draw.	scalar
The ratio of the number of moves since the last attack to the maximum number of moves without attack before the game is considered a draw.	scalar
The phase of the game: either deployment (1) or play (0).	scalar
An indication of whether the agent needs to select a piece (0) or target square (1) for an already selected piece. 0 during deployment phase.	scalar
The piece selected in the previous step (1 for the selected piece, 0 elsewhere), if applicable, otherwise all 0's.	10×10

Table 3: The components of the agent observation. These are stacked into a $10 \times 10 \times 82$ tensor by expanding the scalar components to a 10×10 tensor with constant value.

that:

$$\mathbf{Mov}_i^m[r,c] = \begin{cases} -1 & \text{if the piece made a regular move from square } (r,c) \\ -(2+t/12) & \text{if a piece of type } t \text{ attacked from square } (r,c) \\ 1 & \text{if the piece moved to or attacked square } (r,c) \\ 0 & \text{elsewhere} \end{cases} \tag{7}$$

In the remainder, when referring to a player $i \in \{\text{red}, \text{blue}\}$, we use $-i$ to denote the opponent. The observation for player i consists of the components shown in table 3. These are stacked into a $10 \times 10 \times 82$ tensor by expanding the scalar components to a 10×10 tensor with constant value.

Player-centric observations and actions

To facilitate training a single agent that can play both sides of the board, the observation is presented player-centric: the tensor for player blue is rotated 180 degrees. Likewise, the interpretation of the actions (between 0 and 99) as squares on the board are also done relative to the side of the board of the player.

Network Architecture

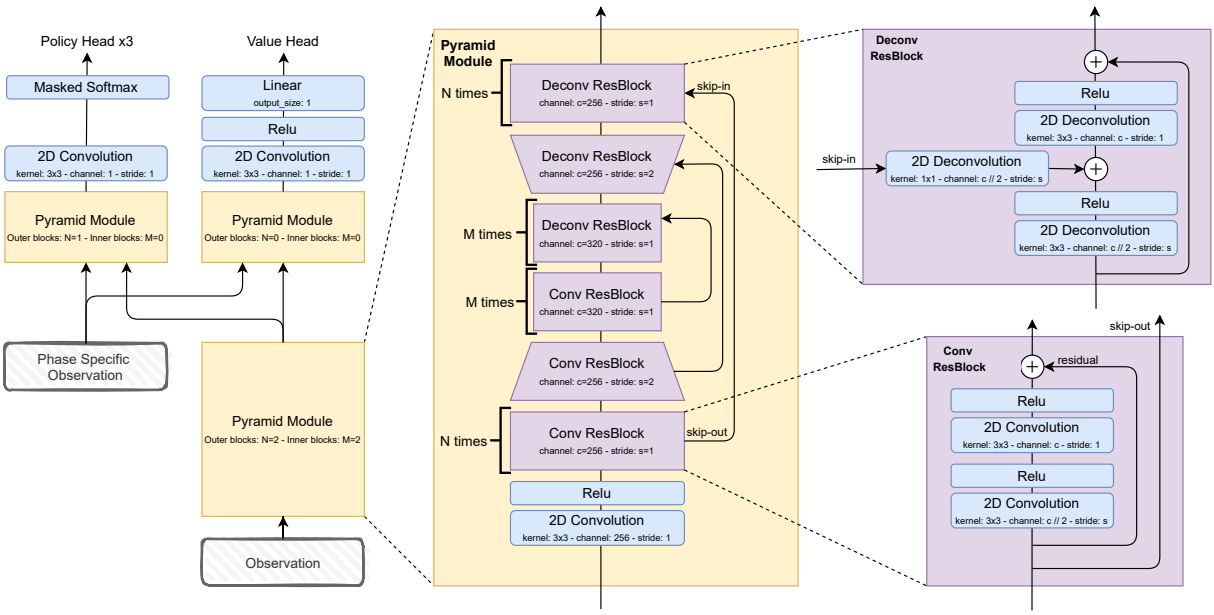


Figure 7: Network implementation details. When applying striding, residual connections are also processed by a convolution layer with 1×1 kernel (hidden for clarity).

Network Modules The agent policy is parameterized through a deep neural network. A torso module first processes the observation to produce a board game embedding, represented as a tensor of spatial dimension 10×10 and channel dimension 256. This representation is then provided to three policy head modules specialized by game phase and a value-head module.

First, the deployment head is used during the deployment phase, i.e. the 40 first steps of the game. It takes as input the board game embedding. It then outputs a probability distribution of dimension 10×10 over the board (restricted to legal positions) to indicate the emplacement of the next piece to deploy. The piece deployment always follows the same order, so we did not provide complimentary observation to the board game embedding.

Second, the piece-selection head is used during the first stage of the game phase, i.e., selecting the unit to play. It takes as input the concatenation of the board game embedding and the tiled no-attack ratio. It then outputs the probability distribution of dimension 10×10 (restricted to playable units) to pick the unit to play. This distribution is then sampled to proceed to the second stage of the game phase.

Third, the piece-displacement head is used during the second stage of the game phase, i.e., moving the selected piece and potentially attacking an opponent unit. It takes as input the concatenation of the board game embedding, the no-attack ratio, and the one-hot representation of the selected piece. This one hot representation is a sparse tensor of spatial dimension 10×10 and channel dimension 10 that respectively encode the spatial location of the selected piece and the unit-type across movable units (all units but the Bomb (B) and the Flag (F)).

Fourth, the value head is used during the training to compute the value function of the agent. It takes as input the concatenation of the board game embedding, the no-attack ratio, and the one-hot representation of the selected piece. Note that the no-attack ratio is zeroed-out during deployment, and the one-hot representation of the selected piece is zeroed-out during deployment and the first stage of the game phase. While we could use a value head for each policy head, we used a single value head for the three policies to reduce the agent's memory footprint. This zeroing is then necessary to have a fixed input shape across all phases.

Neural Implementation All neural modules, i.e. the torso and heads, are based on a pyramid-like architecture (*75*), which are themselves composed of diverse resblock modules (*43*). Note we refer as inner blocks, the blocks that perform spatial dimension reduction through convolution layers (*76*), and outer blocks that spatially upscale the intermediate feature map through deconvolution layers (*77*).

The pyramid module is constructed as follow from input to output:

- 1 convolution layer with $C = 256$ channel, no striding, 3×3 kernel and relu.
- N outer-convolution resblocks with $C = 256$ channels and no striding.
- 1 strided-convolution resblock with $C = 256$ channels and $S = 2$ striding.
- M inner-convolution resblocks with a channel size $C = 320$ and no striding.
- M inner-deconvolution resblocks with a channel size $C = 320$ and no striding.
- 1 strided-deconvolution resblock with $C = 256$ channels and $S = 2$ striding.
- N outer-deconvolution resblocks with $C = 256$ channels and no striding.

Pyramid modules also include skip-connections that go from the convolution resblock to its symmetric deconvolution resblock. Policy heads use $N = 1$ outer resblocks, strided-convolution resblocks and no inner resblocks $M = 0$. The final deconvolution is followed by a 2D-convolution layer with a single channel, no striding, 3×3 kernel, and relu activation to output the action logits. Forbidden actions are then masked by turning their logits to $-\infty$, and the remaining logits are turned into a probability distribution through a softmax activation layer. The value head uses no outer resblocks $N = 0$, the strided-convolution resblocks and no inner resblocks $M = 0$. The final deconvolution is followed by a 2D-convolution layer with a single channel, no striding and 3×3 kernel, and relu activation. The logits are then flattened and processed by a linear layer to output the final value-function scalar.

The convolution resblocks are constructed as follow from input to output:

- A skip-out connection that is fed back to the symmetric deconvolution resblock.

- A convolution layer with $C//2$ channel, optional S striding, 3×3 kernel and relu.
- A convolution layer with C channel, no striding, 3×3 kernel and relu.
- A residual connection that sum the initial resblock input and the current logits.

The deconvolution resblocks are constructed as follow from input to output:

- A deconvolution layer with $C//2$ channel, optional S striding, 3×3 kernel and relu.
- A skip-in connection that sum the symmetric skip-out connection and the current logits. The skip-connection is first processed by a 2D-deconvolution layer with $C//2$ channel, optional S striding, 1×1 kernel and no activation.
- A deconvolution layer with C channel, no striding, 3×3 kernel and relu.
- A residual connection that sum the initial resblock input and the current logits.

When performing striding in resblocks, the residual connection is completed by a 2D-convolution layer with 2 stride, 1×1 kernel and no activation to fit the dimension change.

Infrastructure and Setup

The *DeepNash* training pipeline follows Sebulba Podracer architecture from (*78*). It consists of Actors, Replay Buffer, Learner and Evaluators (Figure 8).

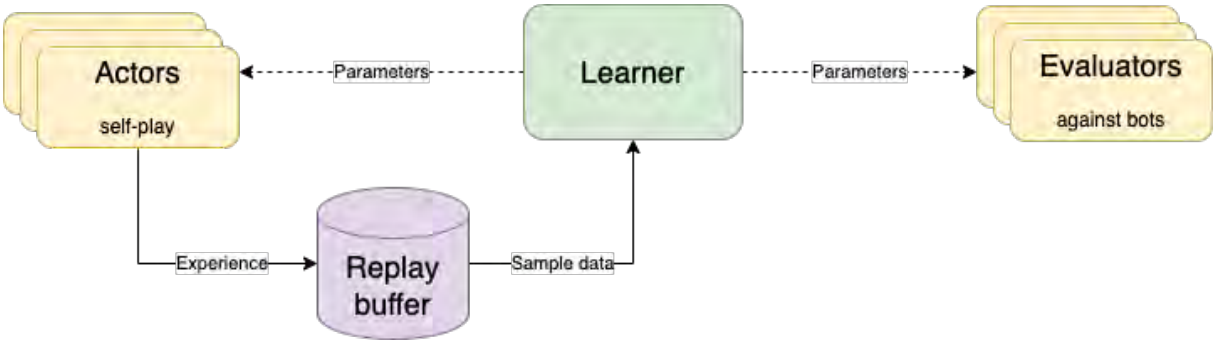


Figure 8: Agent decomposition. Learner samples games/trajectories from Replay Buffer, improves weights, and periodically distributes these to actors and evaluators. Actors self-play and write experience into Replay Buffer. Evaluators play against other bots to assess performance.

* The Actors self-play and write full games into Replay Buffer.

* The Replay Buffer, as a queue, stores games from Actors until the Learner reads it.
* The Learner, extracts a batch of games from Replay Buffer, improves the network weights, and periodically sends those to Actors/Evaluators.
* The Evaluators play against fixed opponent bots, e.g. Uniform or Demon of Ignorance; then plot the aggregated statistics.

Deployment We deploy Actors, Learner and Buffer together on the same machine, since they all use/need different kinds of resources:

- Actors run environment and inference and use CPU and a small amount of accelerator FLOPs.
- Replay Buffer predominantly uses RAM.
- Learner heavily uses accelerator FLOPS.

Evaluators run separately as a flock of low-priority machines.

Scalability To scale the training pipeline we rely on SIMD model (Figure 9) implemented in JAX (*79*) as ‘pmap‘ transformation.

Briefly on SIMD, several learner machines participate in training; each does a learning step in sync with other machines; each reads a batch of different games, computing network parameter gradients; computes the average of the gradients across all machines; and finally each applies those average gradients, as updates, to network weights.

To generate the batches of games, each learner machine runs, as separate threads, its own Actors and a single Replay Buffer.

To train the final agent we used 768 TPU nodes used for Learners and 256 TPU nodes for Actors.

Differences from Sebulba Podracer

Figure 9: Several machines participate in training, each embedding its own Learner, Actors and Replay Buffer. At each parameter updates, the gradients are averaged across all machines. Everything else is as if machines were independent.

- The Actors environment-agent loop is implemented in C++ using fibers and the native open spiel (*80*) interfaces. This provides better overall throughput for playing games.
- Full games are stored in the Replay Buffer, and each Learner step consumes a batch of full games. Note that the game length is variable. See below for the details.
- The evaluators are run a separate flock of low-priority machines, that evaluate the agent against opponent bots. It decouples opponent bot' hardware requirements from the main training pipeline.

Full games learning To compute the exact returns on a variable-length trajectory, we store full games in the Replay Buffer (instead of a fixed length sequences of steps as in most RL agents). When learner samples a batch of games, shorter games are padded to the length of the

longest game in the batch. The batch is chopped into a variable number of fixed length chunks along the time-axis. The learner then does two passes over this list of chunks: a) a forward pass to recompute the exact network (LSTM) state for each timestep and trajectory statistics; b) a backward pass, where each chunk is reprocessed again, but is given the exact network state at the beginning of the chunk and full trajectory statistics, such as the exact returns, etc.

Note that (a) happens outside of gradients computation, and serves to avoid bootstrapping or any kind of approximation for both the returns and for the network state. While (b) happens within the loss and serves to compute gradients for network weights.

The gradients computed on each chunk during (b) are accumulated, and applied using optimizer onto the network weights at the end of the (b) pass.

Figure 10: Variable length games are padded and split into fixed size chunks. Chunks are forward processed to compute global statistics and up-to-date beliefs/memory. Then chunks are processed in reverse order to compute parameter gradients. Finally accumulated gradients are used to update network weights.

Test Time Improvements

When evaluating our Stratego agent, we make some adjustments to the policy given by the trained neural network to improve the strength of the agent further and remove some non-performance-improving characteristics of its play which are annoying to human opponents.

Post-processing of policy

Given Stratego is a complex imperfect information game, we expect the Nash equilibrium policy to be non-deterministic in many states of the game. We indeed observe that the policy produced by the R-NaD algorithm is non-deterministic in most states. However, the policy often gives a small but non-zero probability to moves which are clear blunders, and which would result in the agent losing a game which it should have won. Since Stratego games can take up to 2,000 moves, playing according to this stochastic policy will result in taking several very-low-probability actions over the course of the game. This is especially true if the opponent is aware of the issue and prolongs the game by avoiding attacks and making neutral waiting moves. Presumably an exact equilibrium strategy would give a probability of precisely zero to these actions – we therefore apply a heuristic post-processing of the policy to eliminate very-low-probability moves, without making the policy of the agent too deterministic, which would make it exploitable:

- Thresholding: all actions with probability lower than a fixed threshold ϵ_{tres} are dropped and the policy is renormalized. If no actions with positive probability would remain, the policy is left unchanged.
- Discretization: sorted from high to low, probabilities are rounded up to the nearest multiple of $1/n_{\text{disc}}$ and remaining weights are discarded once a sum of 1 is reached.

Table 4 lists the values of these parameters we used.

Avoid repetitive play

We observed that our agent often repetitively moved a piece back-and-forth on two squares, thereby threatening a known lower-valued opponent piece. While this is allowed by the rules up to a certain limit (see the two-square and more-square rules in (*21*)), the repetition is annoying for human opponents.[7] Moreover we observed in some cases in human evaluation that this style of play resulted in a draw where the agent actually could have won. We apply two techniques to our agent that avoid this style of play:

- Avoid pointless threats: initially the agent is allowed to threaten an opponent piece by moving move back and forth three times between two adjacent squares, in accordance with the two-square rule. After that, for the same two pieces on the same 2x2 region of the board, only a single threatening move is allowed: if the opposing piece retreats as before, the agent cannot immediately move to re-threaten it unless the policy has no alternative moves.
- Eagerness: decreases the agent's perception of how much time remains before the game will be considered a draw due to no no piece having been attacked. As explained in the section on Neural Network Input Representation, the observation presented to the agent contains a ratio r indicating how close the game is to being declared a draw, which is reset to zero every time an attack takes place, and reaches the value 1 when a draw is declared. In order to induce the agent to be more eager in its play, at test time we modify this to $r' = 1 - (1 - r)^{\alpha_{\text{eag}}}$ with α_{eag} a parameter defined in table 4.

Note that the pointless threat restriction applies only to our agent, not to its opponent.

[7] In some cases it could even be considered as "unsporting behavior" according to the ISF rules (*21*).

Memory heuristic

In our human evaluation it was observed that in some cases the observation presented to the neural network, as described in the section on the Neural Network Input Representation, does not contain sufficient history to allow a detailed assessment of a Stratego state. This is an area we think has considerable potential for further performance improvement, by using agent architectures that can better cope with long memory.

That being said, many of these memory problems were very similar and could be solved by tracking of a small number of move patterns. Specifically, our *memory heuristic* assumes that:

- A Spy would attack a known Marshal;
- A Marshal would attack a known General or Colonel if no recapture is possible;
- A General would attack a known Colonel if no recapture is possible;

And therefore that if a player instead makes a non-attacking move, then any piece that had the chance to attack according to the rules described is not of the corresponding type. These eliminated possibilities are tracked per piece throughout the game and influence the policy by modifying the network input: the corresponding entries in both $\mathbf{Pub}_i$ and $\mathbf{Pub}_{-i}$ are set to 0.

The assumptions of this heuristic are very often, but not always, met, even in self-play. Despite the occasional inaccuracies, overall using this heuristic seems to improves the performance in matches against humans. A qualitative observation is that it mostly acts through $\mathbf{Pub}_i$, i.e. by more accurately tracking the information the human has on the agent's pieces, for example by avoiding some types of inconsistent bluffing.

Value bounds heuristic

While the policy post-processing described above removes most errors without making use of game-specific logic, the evaluation of successive agent checkpoints with Vincent de Boer revealed that occasionally an obvious mistake still had support in the policy and if played, could

decide a game. We therefore added a *value bounds heuristic* which aims to eliminate obvious mistakes by leveraging the value function which is trained alongside the policy, and which can be used to score any state where the agent needs to act. Two-step lookahead is performed to estimate an *upper bound* of the value of each action in the policy. These values are not informative about the relative merits of actions considered, but if an estimated upper bound for an action is *lower* than the value of the current state, then the action is considered to be an error, and removed from the policy.

In performing this evaluation, no probabilistic assessment is made of the opponent's private information; instead the best-case private information of the opponent is assumed for the purpose of evaluating attacks, and the worst-case response by the opponent from a set of safe actions. More precisely (using terminology and notations introduced in the section on Imperfect Information Games) let $v^{\text{nn}}(x)$ be the value function provided by the neural network applied to information state x. We define an action (consisting of a piece moving or attacking) to be *safe* if all of the following conditions are met:

- the piece is already known to be movable,
- in case of a long move/attack, the piece already known to be a Scout,
- in case of an attack, it is a guaranteed win, for any private info of both players consistent with the public info,
- the piece cannot be subsequently captured, for any private info of both players consistent with the public info

So safety is defined based only on public information of both players. We write $\mathcal{A}_{\text{safe}}(h)$ for the set of all safe actions at history h. Given an information state x, we define the value upper bound for action a as

$$\widehat{v^{\text{nn}}}(x, a) = \max_{h \in x} \min_{a' \in \mathcal{A}_{\text{safe}}(ha)} v^{\text{nn}}(x(haa')) \tag{8}$$

If $\widehat{v^{\text{nn}}}(x, a) + \epsilon_{\text{vb}} < v^{\text{nn}}(x)$ then a is removed from the policy, if it supports alternative actions,

$\epsilon_{\text{tres}}^{(\text{deploy})}$	0.03	threshold used in deployment phase
$\epsilon_{\text{tres}}^{(\text{play})}$	0.03	threshold used in play phase
$n_{\text{disc}}^{(\text{deploy})}$	32	discretization used in deployment phase
$n_{\text{disc}}^{(\text{play})}$	16	discretization used in play phase
α_{eag}	2.0	eagerness parameter
ϵ_{vb}	0.05	margin used by the value bounds heuristic

Table 4: Test time improvement settings.

with ϵ_{vb} a margin defined in table 4. Equation 8 does not deal with terminal states for simplicity but can be generalized to these cases straightforwardly. In practice $\widehat{v^{\text{nn}}}(x, a)$ can be determined efficiently without enumerating all $h \in x$ because at most two unknown opponent pieces can be involved: a piece attacked by the agent and subsequently a piece attacking the agent.

This heuristic only occasionally intervenes; for example in the matches played on Gravon, it affected less than 1.5% of *DeepNash*'s turns. Also, it is interesting to note that both this heuristic and the memory heuristic do not significantly improve the winrate when *DeepNash* is evaluated against a version of itself without these heuristics. They only empirically seem to avoid some mistakes observed in matches against humans.

Additional results

Details on human evaluation on Gravon

DeepNash was evaluated on Gravon, beginning of April 2022, with the consent of Thorsten Jungblut, the Gravon platform owner. This resulted in roughly 50 ranked matches. The ratings of *DeepNash* in the 2022 and all-time ranking were computed on April 22$^{\text{nd}}$ 2022 and three matches were ignored for this computation: two were played with an earlier training snapshot, one timed out due to a human error.

Details on external Stratego program evaluation

Probe is a program which was three-fold winner of the Computer Stratego World Championship (2007, 2008, 2010) (*47*) and is currently available as the "Heroic Battle" app on Android. We evaluated version 2.0.37 using the Expert skill level and played the same number of games for the different playing styles available: Conservative, Moderate and Aggressive.

Master of the Flag is a program which won the Computer Stratego World Championship in 2009 (*47*). It is currently available on a website (*81*) and we tested against algorithm version 5.2.0.40. In the interface, Master of the Flag only plays as Blue so we evaluated in this setting only.

Demon of Ignorance is an open-source implementation of Stratego with an accompanying AI bot, written in Java (*48*). We evaluated version 0.13.4. The bot's AI-level can be configured, which controls its thinking time. We observed that the performance of this bot against our agent saturates at an AI-level of 8, so this is the setting we use for evaluation. Occasionally this bot does not produce a valid action, if this happens 3 times in a row, the ongoing game is discarded.

Asmodeus, **Celsius**, **Celsius1.1**, **PeternLewis**, **Vixen** are agents that were submitted in an Australian university programming competition in 2012 (*49*), won by *PeternLewis*. These agents sometimes pick moves that violate the two- and more-square rule of Stratego. We therefore evaluated using a game variant that allows such repetitive moves, but we discarded matches that resulted in a draw due to such endless repetition. We evaluated at commit 54f0978 of the repository.

Quotes from Stratego Experts

Thorsten Jungblut, owner of the Gravon platform:

> Many players in the past thought that there will never be an AI for Stratego that

could be a real competition for human players, or even play in the top ten. Obviously, they were wrong.

Vincent de Boer, former Stratego world champion, evaluated *DeepNash* as follows:

> The level of play of DeepNash surprised me. I had never seen or heard of an artificial Stratego player that came close to the level needed to win a match against an experienced human player, but after playing against DeepNash myself I was not surprised by the top-3 ranking it later on achieved on the Gravon internet platform. I would expect this agent to also do very well if it participated in the World Championship.

References

1. C. E. Shannon, *Philosophical Magazine* **41**, 256 (1950).

2. M. Campbell, A. J. Hoane Jr., F.-h. Hse, *Artificial Intelligence* **134**, 57 (2002).

3. D. Silver, *et al.*, *Science* **362**, 1140 (2018).

4. J. Schrittwieser, *et al.*, *Nature* **588**, 604 (2020).

5. M. Moravčík, *et al.*, *Science* **356**, 508 (2017).

6. N. Brown, T. Sandholm, *Science* **359**, 418 (2018).

7. N. Brown, T. Sandholm, *Science* **365**, 885 (2019).

8. M. Schmid, *et al.*, *CoRR* **abs/2112.03178** (2021).

9. A. Arts, Competitive play in stratego, Ph.D. thesis, Maastricht University (2010).

10. M. Johanson, Measuring the size of large no-limit poker games (2013).

11. O. Vinyals, *et al.*, *Nature* **575**, 350 (2019).

12. M. Jaderberg, *et al.*, *Science* **364**, 859 (2019).

13. C. Berner, *et al.*, *arXiv preprint arXiv:1912.06680* (2019).

14. C. Treijtel, L. J. M. Rothkrantz, *2nd International Conference on Intelligent Games and Simulation (GAME-ON 2001), November 30 - December 1, 2001, London, UK*, Q. H. Mehdi, N. E. Gough, eds. (2001), p. 17.

15. V. de Boer, *Invincible: a stratego bot. MsC thesis* (TU Delft, the Netherlands, 2007).

16. V. de Boer, L. J. M. Rothkrantz, P. Wiggers, *Int. J. Intell. Games Simul.* **5**, 25 (2008).

17. M. Schadd, M. Winands, *Proceedings of the Twenty-First Benelux Conference on Artificial Intelligence (BNAIC 2009)* (2009), pp. 225–232.

18. I. Satz, *J. Int. Comput. Games Assoc.* **34**, 27 (2011).

19. S. Redeca, A. Groza, *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)* (2018), pp. 97–104.

20. S. Mcaleer, J. Lanier, R. Fox, P. Baldi, *Advances in Neural Information Processing Systems* **33**, 20238 (2020).

21. Official stratego rules, `https://isfstratego.kleier.net/docs/rulreg/isfgamerules.pdf`. Accessed: 2021-11-22.

22. J. Nash, *Annals of Mathematics* **54**, 286 (1951).

23. J. von Neumann, O. Morgenstern, *Theory of games and economic behavior* (Princeton University Press, 1947).

24. Gravon, `https://www.gravon.de/gravon/`. Accessed: 2022-05-16.

25. M. Moravcík, *et al.*, *CoRR* **abs/1701.01724** (2017).

26. D. Silver, *et al.*, *Nat.* **529**, 484 (2016).

27. N. Brown, A. Bakhtin, A. Lerer, Q. Gong, *Thirty-fourth Annual Conference on Neural Information Processing Systems (NeurIPS)* (2020). `https://arxiv.org/abs/2007.13544`.

28. O. Vinyals, *et al.*, *Nature* **575**, 350 (2019).

29. OpenAI, OpenAI Five, https://blog.openai.com/openai-five/ (2018).

30. E. Lockhart, *et al.*, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19* (International Joint Conferences on Artificial Intelligence Organization, 2019), pp. 464–470.

31. K. Tuyls, K. Verbeeck, T. Lenaerts, *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings* (2003), pp. 693–700.

32. K. Tuyls, P. J. Hoen, B. Vanschoenwinkel, *Autonomous Agents and Multi-Agent Systems* **12**, 115 (2006).

33. K. Tuyls, D. Heytens, A. Nowé, B. Manderick, *Machine Learning: ECML 2003, 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, N. Lavrac, D. Gamberger, L. Todorovski, H. Blockeel, eds. (Springer, 2003), vol. 2837 of *Lecture Notes in Computer Science*, pp. 421–431.

34. J. Perolat, *et al.*, *International Conference on Machine Learning* (2021).

35. G. Piliouras, L. J. Schulman, *arXiv preprint arXiv:1711.06879* (2017).

36. P. Mertikopoulos, C. Papadimitriou, G. Piliouras, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (SIAM, 2018), pp. 2703–2717.

37. H. B. McMahan, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, G. J. Gordon, D. B. Dunson, M. Dudík, eds. (JMLR.org, 2011), vol. 15 of *JMLR Proceedings*, pp. 525–533.

38. E. Zeeman, *Lecture Notes in Mathematics, Global theory of dynamical systems* **819** (1980).

39. E. Zeeman, *Theoretical Biology* **89**, 249 (1981).

40. J. Maynard Smith, *On Evolution* (1972).

41. D. Bloembergen, K. Tuyls, D. Hennes, M. Kaisers, *J. Artif. Intell. Res.* **53**, 659 (2015).

42. O. Ronneberger, P. Fischer, T. Brox, *International Conference on Medical image computing and computer-assisted intervention* (Springer, 2015), pp. 234–241.

43. K. He, X. Zhang, S. Ren, J. Sun, *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

44. L. Espeholt, *et al.*, *International Conference on Machine Learning* (2018), pp. 1406–1415.

45. D. Hennes, *et al.*, *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2020).

46. Kleier rating system, `https://www.kleier.net`. Accessed: 2022-05-24.

47. Stratego on wikipedia, `https://en.wikipedia.org/wiki/Stratego`. Accessed: 2021-12-01.

48. Demon of Ignorance Stratego AI, `https://github.com/braathwaate/stratego`. Accessed: 2021-12-01.

49. University Computer Club in Australia 2012 programming competition, `https://github.com/braathwaate/strategoevaluator`. Accessed: 2021-12-13.

50. J. Hannan, *Contributions to the Theory of Games* **3**, 97 (1957).

51. S. Hart, A. Mas-Colell, *Econometrica* **68**, 1127 (2000).

52. Y. Freund, R. E. Schapire, *Proceedings of the ninth annual conference on Computational learning theory* (1996), pp. 325–332.

53. M. Zinkevich, M. Johanson, M. Bowling, C. Piccione, *Advances in neural information processing systems* (2008), pp. 1729–1736.

54. M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling, *Advances in neural information processing systems* **22** (2009).

55. R. Gibson, *Regret Minimization in Games and the Development of Champion Multiplayer Computer Poker-Playing Agents* (University of Alberta, Canada, 2014).

56. O. Tammelin, N. Burch, M. Johanson, M. Bowling, *Twenty-fourth international joint conference on artificial intelligence* (2015).

57. N. Brown, T. Sandholm, *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), pp. 1829–1836.

58. G. Tesauro, *et al.*, *Communications of the ACM* **38**, 58 (1995).

59. M. Buro, *International Conference on Computers and Games* (Springer, 1998), pp. 126–145.

60. J. Schaeffer, M. Hlynka, V. Jussila, *International Joint Conference on Artificial intelligence* (2001).

61. D. Silver, *et al.*, *Nature* **550** (2017).

62. D. Silver, *et al.*, *Science* **362** (2018).

63. N. Brown, A. Lerer, S. Gross, T. Sandholm, *International Conference on Machine Learning* (2019), pp. 793–802.

64. E. Steinberger, A. Lerer, N. Brown, *arXiv preprint arXiv:2006.10410* (2020).

65. A. Gruslys, *et al.*, *arXiv preprint arXiv:2008.12234* (2020).

66. J. Heinrich, D. Silver, *arXiv preprint arXiv:1603.01121* (2016).

67. M. Lanctot, *et al.*, *Advances in Neural Information Processing Systems* (2017), pp. 4190–4203.

68. C. Berner, *et al.*, *arXiv preprint arXiv:1912.06680* (2019).

69. M. Jaderberg, *et al.*, *Science* **364**, 859 (2019).

70. S. Srinivasan, *et al.*, *Advances in neural information processing systems* (2018), pp. 3422–3435.

71. S. Omidshafiei, *et al.*, *arXiv preprint arXiv:1906.00190* (2019).

72. M. Schadd, M. Winands, *Proceedings of the 21st Benelux Conference on Artificial Intelligence. Eindhoven, the Netherlands* (2009).

73. S. Jug, M. Schadd, *Icga Journal* **32**, 233 (2009).

74. J. Maynard Smith, *Evolution and the Theory of Games.* (Cambridge University Press, 1982).

75. T.-Y. Lin, *et al.*, *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 2117–2125.

76. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Proceedings of the IEEE* **86**, 2278 (1998).

77. J. Long, E. Shelhamer, T. Darrell, *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 3431–3440.

78. M. Hessel, *et al.*, Podracer architectures for scalable reinforcement learning (2021).

79. J. Bradbury, *et al.*, JAX: composable transformations of Python+NumPy programs (2018).

80. M. Lanctot, *et al.*, *CoRR* **abs/1908.09453** (2019).

81. Master of the Flag, `http://www.jayoogee.com/masteroftheflag/game.html`. Accessed: 2022-05-16.