Minimax Strikes Back

Quentin Cohen-Solal
LAMSADE, Université Paris-Dauphine, PSL, CNRS
Paris, France
quentin.cohen-solal@dauphine.psl.eu

Tristan Cazenave
LAMSADE, Université Paris-Dauphine, PSL, CNRS
Paris, France
tristan.cazenave@dauphine.psl.eu

ABSTRACT

Deep Reinforcement Learning reaches a superhuman level of play in many complete information games. The state of the art algorithm for learning with zero knowledge is AlphaZero. We take another approach, Athénan, which uses a different, Minimax-based, search algorithm called Descent, as well as different learning targets and that does not use a policy. We show that for multiple games it is much more efficient than the reimplementation of AlphaZero: Polygames. It is even competitive with Polygames when Polygames uses 100 times more GPU (at least for some games). One of the keys to the superior performance is that the cost of generating state data for training is approximately 296 times lower with Athénan. With the same reasonable ressources, Athénan without reinforcement heuristic is at least 7 times faster than Polygames and much more than 30 times faster with reinforcement heuristic.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**; **Game tree search**; **Heuristic function construction**; Planning for deterministic actions; Multi-agent planning; Neural networks.

KEYWORDS

Minimax ; Reinforcement Learning ; Zero Learning ; Games ; Tree Search

ACM Reference Format:
Quentin Cohen-Solal and Tristan Cazenave. 2023. Minimax Strikes Back. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Monte Carlo Tree Search (MCTS) [3, 13, 19] and its refinements [6, 7, 24] are the current state of the art in complete information games search algorithms. Historically, at the root of MCTS were random and noisy playouts. Many such playouts were necessary to accurately evaluate a state. Since *AlphaGo* [24] and *AlphaZero* [25] it is not the case anymore. More precisely, strong policies and evaluations are now provided, by neural networks that are trained with Reinforcement Learning. These evaluations are stronger and faster to calculate.

In AlphaGo and its descendants the policy is used as a prior in the PUCT bandit to explore first the most promising moves advised by the neural network policy. In addition, the neural evaluations replace the playouts. Moreover, in AlphaGo, before the reinforcement learning process, data from matches played between humans are used during a supervised learning process. It is not the case with the latest version, i.e. AlphaZero, where a very high level of play can be achieved without the use of knowledge. For example, AlphaZero has surpassed the level of the program Stockfish 8 in Chess (Grandmaster level) [25]. In this paper, we advocate that with reinforcement learning, MCTS might not be the best algorithm anymore. Minimax algorithms are serious challengers when equipped with a strong evaluation function from reinforcement learning.

In this paper, we show that minimax-based algorithms are competitive with MCTS-based algorithms, and even superior for at least many games. More precisely, we make a comparison between a recent Minimax-based reinforcement learning framework, called *Athénan*, with AlphaZero, the state of the art of reinforcement learning, which had not been done before. Unlike the AlphaZero approach based on MCTS, Athénan uses two search algorithms, which are variants of Minimax: *Descent* used during the learning process and *Unbounded Minimax* used after the learning process.

The remainder of the paper is organized as follows. The second section deals with related work. In particular, the section 2.2 presents the two learning algorithms we use in this paper: Athénan and AlphaZero, but also the open source reimplementation of the latter named *Polygames*. This section also compares their characteristics. Sections 3 experimentally compares the two learning algorithms. Section 4 is a discussion about the results and Section 5 concludes the article. Important note: Details of experiments, removed for a reason of space, are described in Technical Appendix [12], which includes the description of the used games, details about the used neural architectures, the learning parameters, algorithms, other experiments, and additional performance curves and tables.

2 BACKGROUND

2.1 Related Work

There are many search algorithms for perfect information games. The two standard algorithms are Monte Carlo Tree Search and Minimax with $\alpha\beta$ pruning.

MCTS has its roots in computer Go [13]. It was theoretically defined with the UCT algorithm [19] that converges to the Nash equilibrium and uses a well defined bandit, Upper Confidence Bounds (UCB), which minimizes the cumulative regret at each node [2]. Theoretical bandits were soon replaced with empirical bandits, giving better results. First the RAVE algorithm [15] improved greatly on UCT for the games of Go and Hex [9]. A later refinement is GRAVE that improves on RAVE for many different games [6] and is used for General Game Playing, for example in Ludii [4]. MCTS was combined with neural networks in AlphaGo, surpassing professional level in the game of Go [24]. The search algorithm used in AlphaGo is MCTS with PUCT, a bandit that uses the policy given by the neural network to bias the moves to explore. Later AlphaGo was redesigned to learn from zero knowledge, leading to AlphaGo Zero

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

[26]. It was then applied to other games, namely Shogi and Chess, with the more general AlphaZero program [25]. Many teams have replicated the AlphaZero approach for Go and for other games: Elf/OpenGo [28], Leela Zero [23], Crazy Zero by Rémi Coulom, KataGo [29], Galvanise Zero [14], and Polygames [8].

As we use Polygames as a sparring partner, we will give more details about it. Polygames (MIT License) replicates the AlphaZero approach and has been successfully applied to many games. There are multiple innovations in Polygames. It can train neural networks with an architecture independent of the size of the board. To do so it uses a fully convolutional policy, meaning that there is no dense layer between the last convolutional planes and the policy. The value head is also independent of the size of the board since it uses global average pooling before the dense layers connected to the evaluation output neuron. It is much more difficult to train a network for Hex 19 (board size 19×19) or Havannah 10 than training it on a smaller size board. Polygames did succeed in these games by scaling its neural networks trained on smaller sizes to the difficult board sizes. It played on the Little Golem game server and beat the best players at these games that were considered too difficult for *Zero Learning* (i.e. learning without using knowledge except the game rule). Other innovations of Polygames include a pool of neural networks during self-play matches in order to avoid catastrophic forgetting.

The other kinds of algorithms used in computer games are the $\alpha\beta$ family of algorithms, whose apogee took place with Deep Blue [5], the first program beating a world chess champion. $\alpha\beta$ dominated the field of perfect information games until the advent of MCTS in 2006. Still, many current strong Chess programs use $\alpha\beta$ [16]. The latest versions are combined with NNUE neural networks [22]. There has been a lot of research on the optimizations of $\alpha\beta$ [21]. Many of them deal with move ordering since move ordering can drastically improve the search time of $\alpha\beta$ [18]. In [27], $\alpha\beta$ was also combined with a policy within a reinforcement learning architecture and it reaches a good level at Hex (the policy is used to prune actions in order to reduce the branching factor).

The search algorithms we use to learn and play games are close to Unbounded Best-first Minimax Search [20]. There is very little study on this algorithm and it seems little or not applied in practice, except in the work of [10]. In that work, variants and improvements of Unbounded Minimax are proposed with several complementary techniques of zero learning that do not require the use of policies. The proposed overall architecture, called *Athénan* (or also *the Descent framework*), exceeds the state-of-the-art level of play at the game of Hex (size 11, 13, and 19) and other games. In the context of the experiments of [10], Athénan is the best zero learning approach not using a policy: in particular, replacing the used variant of Unbounded Minimax, called *Descent Minimax* (or Descent for short), by Unbounded Minimax, by $\alpha\beta$, or by MCTS (with UCT) gives less good results. Moreover, in the experiments of that work, another variant of Unbounded Minimax, called *Unbounded Minimax with Safe decision*, is shown better than Unbounded Minimax and than $\alpha\beta$ for confrontations ("What is the best search algorithm for winning a game?" is a different question than "What is the best search algorithm to learn faster?"). In [11], it has been proved that, with enough time, Descent Minimax and Unbounded Minimax find the best game strategy (multiplayer generalizations are also proposed).

2.2 Deep Reinforcement Learning Algorithms Compared in this Paper

We detail in this section the two zero learning frameworks used in the experiments of this article.

2.2.1 Polygames Learning Algorithm. Polygames uses its search algorithm, MCTS with PUCT, to generate matches, by playing against itself. It uses the information from these matches to update its neural network. This neural network is used by the search algorithm to evaluate states by a value and by a policy (i.e. a probability distribution on the actions playable in that state). For each finished match, the network is trained to associate with each state of the state sequence of this match the result of the end of that match (which is -1 for a loss, 0 for a draw, and +1 for a win). It is also trained, at the same time, to associate with each state a particular policy. In that "target" policy, the probability of an action is proportional to $N^{1/\tau}$ where N is the number of times this action has been selected in the search from that state and τ is a parameter. Note that, during each Polygames learning process, several games are performed in parallel and their evaluations are batched in order to be evaluated in parallel on the GPU.

2.3 More Details about AlphaZero/Polygames

During a learning process using AlphaZero (and thus Polygames), as long as there is time left, a new match phase is performed. A phase consists of a match against oneself, where in each turn the move to be played is decided after carrying out a search with MCTS + PUCT. MCTS is similar to Unbounded Minimax. The first main

Function descent_iter(s, S, T, f_θ, f_t)
  if terminal(s) **then**
    $S \leftarrow S \cup \{s\}$
    $v(s) \leftarrow f_t(s)$
  else
    if $s \notin S$ **then**
      $S \leftarrow S \cup \{s\}$
      foreach $a \in$ actions(s) **do**
        if terminal($a(s)$) **then**
          $S \leftarrow S \cup \{a(s)\}$
          $v(s,a) \leftarrow f_t(a(s))$
          $v(a(s)) \leftarrow v(s,a)$
        else
          $v(s,a) \leftarrow f_\theta(a(s))$
    $a_b \leftarrow$ best_action(s)
    $v(s, a_b) \leftarrow$ descent_iter($a_b(s), S, T, f_\theta, f_t$)
    $a_b \leftarrow$ best_action(s)
    $v(s) \leftarrow v(s, a_b)$
  return $v(s)$

Function descent($s, S, T, f_\theta, f_t, \tau$)
  t = time()
  while time()$- t < \tau$ **do** descent_iter(s, S, T, f_θ, f_t)
  return S

Algorithm 1: *Descent* algorithm (symbol definitions in Table 1).

Symbols	Definition
actions (s)	action set of the state s for the current player
terminal (s)	true if s is an end-game state
$a(s)$	state obtained after playing the action a in the state s
time ()	current time in seconds
S	states of the partial game tree (and keys of the transposition table T)
T	transposition table (contains state labels as v or P)
$P(s)$	target policy of state s computed from the search data
D	learning data set
τ	search time per action
$t_{\max}$	chosen total duration of the learning process
$v(s)$	value of state s from the game search
$v(s,a)$	value obtained after playing action a in state s
$f_\theta(s)$	adaptive evaluation function (of non-terminal game tree leaves ; first player point of view)
$f_t(s)$	evaluation of terminal states, e.g. game gain (first player point of view)
action_selection(s, S, T)	decides the action to play in the state s depending on the partial game tree, i.e. on S and T
update(f_θ, D)	updates the parameter θ of f_θ in order for $f_\theta(s)$ is closer to v for each $(s,v) \in D$

Table 1: Index of symbols

difference is that the value of a state is not the minimax value in the partial game tree but the average of the leaves in the subtree starting from that state. The second main difference is that the tree is constructed not in choosing states of higher value, but states optimizing the value plus an exploration term depending on the policy of the neural network and the number of selection of actions during the search. After the match, the match state sequence data is added to the previous data (only the most recent data points are kept). Periodically, training is performed from a sample of this data set. The main part of this algorithm is described in Algorithm 2.

2.3.1 Athénan Standard Learning Algorithm. Athénan, the learning framework of [10], is based on a variant of Unbounded Minimax called *Descent Minimax* (or *Descent* for short), which consists in exploring the sequences of actions until terminal states. In comparison, Unbounded Minimax and MCTS explore the sequences of actions only until reaching a leaf state. An iteration of Descent (an analyzed sequence of actions) thus consists in a deterministic complete simulation of the rest of the game. The exploration is thus deeper while remaining a best-first approach. This allows the values of terminal states to be propagated more quickly to (shallower) non-terminal states. Descent is formally described in Algorithm 1.

Unlike Polygames, the learned target value of a state is not the end-game value but its minimax value in the partial game tree built during the match. This information is more informative, since it directly contains part of the knowledge acquired during the previous matches. In addition, contrary to Polygames, learning is carried out for each state of the partial game tree constructed during the searches of the match (not just for each state of the states sequence of the played match). In other words, with Polygames, there is one learning target per search whereas with Athénan, there are several learning targets per search. Therefore, there is no loss of information with Athénan: all of the information acquired during the search is used during the learning process. As a result, Athénan generates a much larger amount of data for training from the same number of played matches than AlphaZero / Polygames. Thus, unlike the state of the art which requires to generate matches in parallel to build its learning dataset, this approach does not require the parallelization of matches (and the parallelization of Athénan is not done in the experiments of this article).

During confrontations, the used search algorithm is Unbounded (Best-First) Minimax with Safe decision, denoted UBFM$_s$. It is a variant of Unbounded Best-First Minimax which performs the same search. More precisely, it iteratively extends the best sequence of actions in the partial game tree (i.e. it adds at each iteration the leafs of the principal variation of the partial game tree). Note that, on the one hand, the best action sequence generally changes after each extension. On the other hand, in general, the worse the evaluation function is, the wider the exploration is. The difference between them is as follows: with Unbounded Minimax, the action to play, chosen after each search, is the one with the best value, while with this variant, the chosen action is the one that is the most explored.

Finally, this approach is optionally based on a *reinforcement heuristic*, that is to say an evaluation function of terminal states more expressive than the classical gain of a game (i.e. +1 / 0 / −1). The best proposed general reinforcement heuristics in [10] are *scoring* and the *depth heuristic* (the latter favoring quick wins and slow defeats).

Note that this approach does not use a policy, so there is no need to encode actions. Consequently, this avoids the learning performance problem of neural networks for games with large

Function AlphaZero_main_algorithm($t_{\max}$, τ)
- $t_0 \leftarrow$ time()
- **while** time()$-\, t_0 < t_{\max}$ **do**
  - **for** $k \in \{1, \ldots, K\}$ **do**
    - $s \leftarrow$initial_game_state()
    - $S \leftarrow \emptyset$
    - $T \leftarrow \{\}$
    - $G \leftarrow \{s\}$
    - **while** $\neg$terminal(s) **do**
      - S, $T \leftarrow$mcts(s, S, T, f_θ, f_t, τ)
      - $a \leftarrow$action_selection(s, S, T)
      - $s \leftarrow a(s)$
      - $G \leftarrow G \cup \{s\}$
    - $D \leftarrow \{(s', (f_t(s), P(s'))) \mid s' \in G\}$
  - update(f_θ, D)

Algorithm 2: Main algorithm of AlphaZero (see Table 1 for the definitions of symbols ; K is the number of matches performed between two updates, some of these matches are executed in parallel ; G is the sequence of states of the current match).

number of actions (i.e. very large output size). In addition, although Athénan does not performed matches in parallel, it batches all the child states of an extended state together to be evaluated at one time on the GPU [10] (with Descent and Unbounded Minimax).

2.4 More Details about Athénan

During a learning process using Athénan, as long as there is time left, a new match phase is performed. A match phase consists of a match against oneself, where in each turn the move to be played is decided after carrying out a search with Descent. The move to be played after the search is chosen according to an action selection method, depending on the result of the search. In these experiments, the used action selection method is the *ordinal law* (actions are chosen randomly according to the order of their value) [10] with the exploitation parameter ϵ' chosen at random uniformly between 0 and 1 each time a new action must be decided. After each match phase, the data from the associated partial game tree is added to the previous data (here, only the data of the last 100 matches are kept). Then, a training phase is carried out from a sample of this data set. Specifically, *smooth experience replay* is used [10]. The main part of this algorithm is described in Algorithm 3. The full formalization is described in [10].

```
Function Athénan_main_algorithm(t_max, τ)
    t_0 ← time()
    while time() − t_0 < t_max do
        s ← initial_game_state()
        S ← ∅
        T ← {}
        while ¬terminal(s) do
            S, T ← descent(s, S, T, f_θ, f_t, τ)
            a ← action_selection(s, S, T)
            s ← a(s)

        D ← {(s, v(s)) | s ∈ S}
        update(f_θ, D)
```

Algorithm 3: Main algorithm of Athénan (see Table 1 for the definitions of symbols).

3 COMPARISON OF ZERO REINFORCEMENT LEARNING ALGORITHMS

In this section, we experimentally compare the two learning algorithms Polygames (see Section 2.2.1) and Athénan (see Section 2.3.1). First, in the context of 8 games, we compare the data efficiency of the two algorithms, i.e. the amount of data generated during the self-play matches which are learned in order to self-improve. Second, we compare the win performances of the two algorithms in the same context (in particular, the algorithms use the same resources). They are rated against MCTS. Then, a longer training is performed on Hex 13 and the algorithms are evaluated against Mohex 2.0 [17], the best publicly available Hex program. Finally, the Polygames networks, that have won numerous medals during the TCGA 2020

layer #	C-network	R_1-network	R_2-network
1	conv. + ReLU	convolution	convolution
$\cdots$	conv. + ReLU	2 res. blocks	8 res. blocks
$N-2$	conv. + ReLU	1×1 conv.	dense + ReLU
$N-1$	dense + ReLU	dense + ReLU	dense + ReLU
N	dense layer	dense layer	dense layer

Table 2: Description of 3 neural architectures of value networks, called C-network, R_1-network, and R_2-network. Each residual block is composed of a ReLU followed by a convolution followed by a ReLU followed by a convolution. Output contains one neuron. Other parameters are: kernel is 3×3, filter number is F, number of neurons in dense layers is D, padding is used with R_i-network but not with C-network.

Game	F	D	Game	F	D
Surakarta	132	845	Breakthrough	132c	477
Othello	132	477	Outer-Open-Gomoku	132	111
Hex 13	132	155	Havannah 8	132	111
Connect6	132	65	Havannah 10	132	65

Table 3: The filter number in convolutional layers and the number of neurons in dense layers of the R_2-networks used with Athénan, detailed for the 8 games.

tournament, confront Athénan networks that have used drastically less computational power for their learning processes. In each of these experiments, Athénan is strongly better than Polygames.

3.1 Technical Details

We expose in this section the technical details common to the experiments of Sections 3.2, 3.3, and 3.4. Recall that full details of experiments of this paper are in Technical Appendix [12].

3.1.1 Parameters. For each learning process with Athénan, the batch size of the stochastic gradient descent B is 3000, *smooth experience replay* is used with the following parameters: $\mu = 100$ and $\delta = 3$. The neural architecture is the same for each game: a R_2-network (see Table 2). The number of parameters in each neural network is of the order of $5 \cdot 10^6$. This implies that the number of filters F and number of dense neurons D are different for each game. The corresponding numbers are described in Table 3.

The action distribution used during the learning process is the ordinal law [10]. It is used with a uniform random variable between 0 and 1 as exploration parameter (the variable value changes after each search performed for determining the next action to play; therefore no simulated annealing is used).

Network architectures used for Polygames are adaptations of the architecture being used with Athénan, in order to add a policy while keeping an analogous number of parameters in the neural network (see the Supplementaries document for the details).

Evaluations of Section 3.3 are performed against the basic MCTS algorithm based on UCT (it uses 160 rollouts). For each learning process (i.e. each learned neural network), each evaluation consists of 400 games (200 in first player and 200 in second player).

3.1.2 Computational Resources.

In this section, we present the used computational resources for the experiments of this paper.

For the performed training runs and confrontations, we use the following hardware: GPU Nvidia Tesla V100 SXM2 32 Go, 2 to 10 CPU (processors Intel Cascade Lake 6248 2.5GHz) on RedHat. There is an exception, for the performed confrontations against Polygames tournament networks (confrontations of Section 3.5), we use the following hardware: GeForce GTX 1080 Ti, 2 to 8 CPU (Intel(R) Xeon(R) CPU E5-2603 v3 1.60GHz) on Ubuntu 18.04.5 LTS.

Athénan programs (Descent Minimax, Unbounded Minimax, ...) are coded in Python (using tensorflow 1.15). Games and Search in Polygames are coded in C/C++. For confrontations, Polygames num_actor parameter is 8 (threads doing MCTS).

3.2 Comparison of Generated Learning Data

In this section, we experimentally compare the number of state data, the number of state evaluations, and the number of neural network evaluations performed during an Athénan training and a Polygames training, each during 15 days. In total, 8 trainings were carried out with Athénan and 5 with Polygames for each of the following games: Connect6, Outer-Open-Gomoku, Hex 13, Havannah 8, Havannah 10, Othello, Breakthrough, and Surakarta.

We start by comparing the number of evaluations. In average, the neural network evaluations of Athénan is 12.7 times smaller than that of Polygames. In addition, the average number of state evaluations of Athénan is 2.5 times smaller than that of Polygames. In other words, Polygames is more efficient to perform evaluations. However, this is not an intrinsic characteristic, because this difference is mainly explained by two facts. First, Athénan is coded in Python but searches and game mechanisms for Polygames are coded in C/C++, which allows it to be 2 to 5 times faster (the speed difference depends on the game ; for example in Python, at Othello, game calculations take more than 82% of the time of the learning process). Second, Polygames performs many matches in parallel, whereas Athénan, in its implementation, is purely sequential (except for evaluating the children of a state: these evaluations are simultaneously performed on the GPU). Thus, the matches parallelization of Polygames gives a potentially larger number of evaluations for the same period of time (Polygames evaluations are also simultaneously performed on the GPU). The detailed numbers for each game are described in Table 4. In summary, in this experiment, Polygames performs more evaluations but it could be counterbalanced by implementing Athénan in C/C++ or by parallelizing it.

Now we compare *learned states*: the number of state data used during the learning process. Athénan generates 296 times more learned states than Polygames (despite performing fewer evaluations as we saw in the previous paragraph). This is due to the fact that Athénan uses tree learning: it learns all the data generated during the search, i.e. it learns all the partial game tree build during the search. By contrast, Polygames / AlphaZero only learns a summary of this search, namely a policy and a state value for the state analyzed during the search. Note that since determining a policy for a state requires that its children be sufficiently explored, it does not seem possible to learn a policy for each state of the search (i.e. for each state of the partial game tree). In other words, it does not seem possible to perform tree learning for the policy with AlphaZero / Polygames. The same remark applies for the state value. Indeed, since the learning target for the state value is the endgame value, it would be the same for all the states of the tree, which is more likely to negatively impact the training in creating over-fitting than improving learning. In other words, naively modifying AlphaZero / Polygames to use a terminal tree learning and tree learning for the policy in order to decrease the cost of data generation should not improve performance. However, it is possible to change the learning target, i.e. replace the endgame value by the search state value and thus to perform classic tree learning. This has been studied in the context of MCTS without policy, and the results are much worse than with Descent Minimax and tree learning (i.e. with Athénan standard algorithms) [10].

In conclusion, the cost of state data generation is 50 to 700 times better with Athénan than with Polygames depending on the game (296 times better on average over the tested games), despite the fact that it performs 2.5 times fewer states evaluations. Moreover, recall that using a language other than Python with Athénan would further improve its performance, most likely by a factor of at least 2. Note also that the matches with Athénan are not parallelized (unlike AlphaZero / Polygames), and parallelizing them would also increase the number of data.

3.3 Win comparison with Same Resources

In this section, we compare the learning performances of Athénan with the learning performances of Polygames, with respect to the win percentages against MCTS.

This comparison is notably based on the *gain* of using Athénan rather than Polygames, which is the difference in their win performances:

Definition 1. The *gain of using the algorithm A rather than the algorithm A'* is $\frac{1}{2}\left((w_A - l_A) - (w_{A'} - l_{A'})\right)$ where w_A (resp. l_A) is the win (resp. loss) percentage of A.

Several trainings, each during 15 days, have been performed for each of the following games: Surakarta, Hex 13, Connect6, Outer-Open-Gomoku, Breakthrough, Othello 8, Havannah 8, Havannah 10. In total, 5 repetitions were performed with Polygames and 8 repetitions were performed with Athénan for each game (4 repetitions without reinforcement heuristic and 4 repetitions with reinforcement heuristic). As the number of repetitions is small, we use the following advanced statistical evaluation procedure: *stratified bootstrap confidence interval* [1] which allows one to evaluate learning processes over several tasks even with a low number of repetitions.

The final global performances of the learning processes based on Athénan and Polygames against a 160-rollouts MCTS with UCT (i.e. without any knowledge nor learned policy) are described in Figure 1. The details for each of the 8 games are described in Figure 2. The curves describing the evolution of the performances of the two algorithms throughout the training are described in Figure 3.

In conclusion, the performances of Athénan are much better than those of Polygames. The performance superiority of Athénan is even more marked when a reinforcement heuristic is used (we already knew that reinforcement heuristic is an improvement of the combination of Descent Minimax and tree learning, i.e. an essential component of Athénan [10]). In particular, on the one hand, the gain of using Athénan without reinforcement heuristic rather than

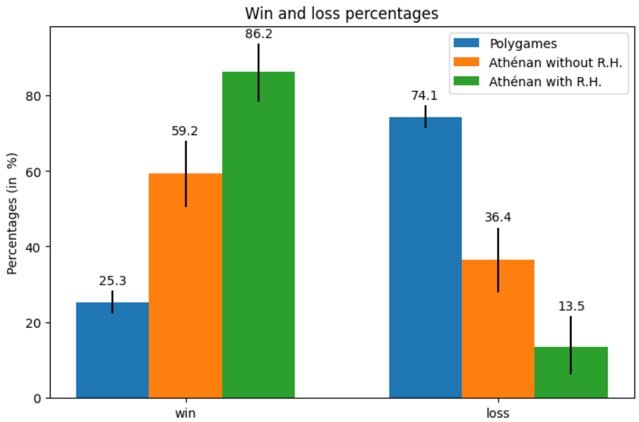

Figure 1: Performance of Athénan (resp. Polygames) against MCTS with UCT at the end of the 15 days of training averaged over the 8 games. Their stratified bootstrap confidence intervals are indicated by the black lines. Athénan results are detailed in function of the use of a reinforcement heuristic (abbreviated R.H.).

Polygames is 35.8% (see Def. 1). On the other hand, the gain of using Athénan with reinforcement heuristic rather than Polygames is 60.75%. Regarding learning speed, Athénan without reinforcement heuristic achieves in only 2 days the performance of Polygames after 15 days of training, i.e. there is a factor of 7. Moreover, Athénan with reinforcement heuristic achieves this performance in much less than half a day, there is a factor of 30 (see the curve in Figure 3).

3.4 Win Comparison with Same Resources during a Long-Term Learning Process

In this section, we compare again the learning performances of Athénan with the learning performances of Polygames with respect to win percentages, but for a longer training (113 days), and only at Hex 13, evaluating them this time against Mohex 2.0 [17], champion program at Hex from 2013 to 2017 at the Computer Olympiads. Mohex 2.0 is the strongest hex program which is freely available.

For this, we have continued the learning processes of the previous section carried out on Hex 13 with Athénan and with Polygames. Then, we have thus evaluated them against Mohex 2.0, at different stages of their learning processes (an evaluation has been performed approximately every 4 days).

The evolution of the average win percentages of Athénan (with and without reinforcement heuristic) against Mohex 2.0 during the training is shown in Figure 4. Athénan with reinforcement heuristic goes rather far beyond the level of Mohex 2.0. Athénan without reinforcement heuristic does not reach the level of Mohex 2.0 but it still manages to beat it in certain positions. On the contrary, none of the learned Polygames networks (combined with the Polygames search algorithm) has succeeded in winning any match against Mohex 2.0, whatever the evaluation moment during their learning process. In other words, the Polygames winning curve is constant and is 0%, with a confidence interval of $[0\%; 0\%]$.

Therefore, in this experiment, learning with *Athénan* is also widely better than with Polygames.

3.5 Comparison versus Tournaments Polygames Networks

In this section, we evaluate Athénan networks against high level Polygames networks, at Breakthrough, at Othello 8 and 10.

The Polygames networks are those having won at Breakthrough and Othello 10 and finished second at Othello 8 in the TCGA 2020 tournament. They have been trained during 7 days with 100 GPU each. The used Athénan networks was trained with only one GPU during 5 days. These trainings was later extended to 30 days.

The results of the confrontation of the 5-day Athénan networks against Polygames networks are described in Figure 5. Although learning with Athénan required 100 times less GPU (1 GPU vs. 100 GPU) and lasted slightly less time (5 days vs. a week), the performance of Athénan is much better for each of the three games.

The results of the confrontation of the 30-day Athénan networks against the same Polygames networks are described in Figure 6. This 30 days experience shows that the Athénan networks continue to improve.

4 FINAL DISCUSSION

In [10], Athénan has been compared to other reinforcement learning algorithms without knowledge that do not use learned policy. In particular, using tree learning gives better results than root learning or terminal learning (the AlphaZero / Polygames learning technique for state values). As a reminder, tree learning learns the entire partial search tree of the analyzed state while root learning and terminal learning only learns the target value for the analyzed state. More precisely, in the experiments of [10], the use of tree learning is always better than the use of terminal learning and for almost all 9 tested games, tree learning improves the final win rate by at least 40%. In this new article, we have seen that tree learning can generate about 296 times more learning data than terminal learning (used by AlphaZero / Polygames). This is one of the reasons that allows Athénan to obtain better results and in particular to learn much faster, especially at the start of the learning process. Note that tree learning could not be applied naturally to AlphaZero / Polygames (see Section 3.2). Moreover, the use of tree learning can lower performance with some search algorithms (for various reasons ; see [10] for details). This is not the case with Descent Minimax, in particular thanks to its exploration which is both very deep and in best first.

The superior performance of Athénan is not only due to the use of tree learning. In [10], Descent Minimax, the central algorithm of Athénan, gives better results than Unbounded Minimax, which is, itself, better than $\alpha\beta$ and Monte Carlo Tree Search (the search algorithm of AlphaZero / Polygames). More precisely, using Descent Minimax with tree learning rather than MCTS with tree learning increases the win rate by at least 40% for all 9 tested games.

This previous study lacked a comparison with the state of the art, which uses a policy (contrary to the studied techniques in [10]). The question then was: do the results against MCTS without policy generalize to the state of the art (i.e. to MCTS with a learned policy)? This article thus fills this gap and allows one to conclude that with a reasonable hardware and an accessible time, Athénan gives much better results than Polygames (see Section 3). In addition, at least in

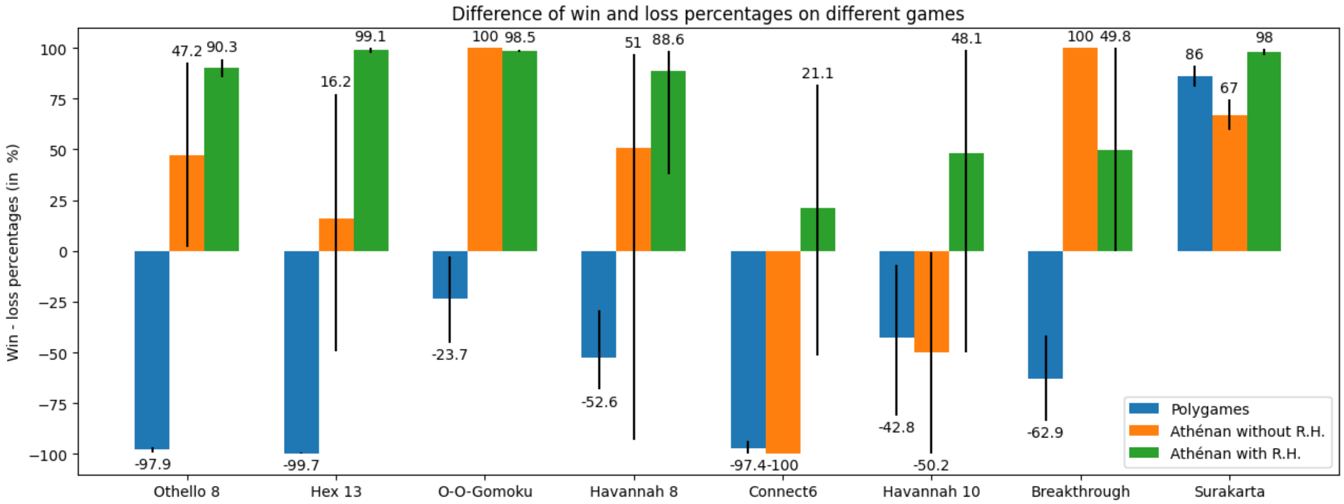

Figure 2: Average win percentages minus loss percentages of Athénan (resp. Polygames) against MCTS with UCT at the end of the 15 days of training for the 8 games. Athénan results are detailed in function of the use of a reinforcement heuristic (abbreviated R.H.). Their bootstrap confidence intervals are indicated by the black lines.

	Connect6	Havannah 10	Havannah 8	Outer-Open-Gomoku	Hex 13	Surakarta	Othello	Breakthrough
Learned states	55	64	111	115	359	442	529	693
Neural evaluation	0,02	0,03	0,05	0,04	0,10	0,11	0,12	0,16
States evaluation	0,37	0,30	0,37	0,49	0,65	0,40	0,10	0,49

Table 4: Ratio of Athénan data over Polygames data for the same learning time and for different games (average over 5 runs for Polygames and 8 runs for Athénan ; data of a run varies by a maximum of ±60% for Polygames and ±20% for Athénan). For example, in Connect6, Athénan learns 55 times more states, makes 50 times less neural evaluations, and makes 3 times less state evaluations.

some context, Athénan with one GPU is even more efficient than Polygames with 100 GPUs (see Section 3.5).

5 CONCLUSION

In [10], a new framework for reinforcement learning without knowledge, called Athénan, has been proposed. In particular, in [10], Athénan has been compared to different standard search algorithms and learning techniques from the literature (which does not use a policy), and it has been shown that Athénan obtains much better performance. However, Athénan has not been compared against the state of the art of reinforcement learning without knowledge, i.e. MCTS combined with a learned policy, the standard entire architecture being called AlphaZero. This lack of comparison is all the more critical as the use of policy in the AlphaZero framework increases the level of play considerably. A comparison with the state of the art AlphaZero is thus essential to know if Athénan is better or if it is only a useful algorithm when a policy cannot be used.

Therefore, in this paper, we have made the first comparison between Athénan and Polygames, a re-implementation of AlphaZero. In particular, we have shown that Athénan has much better performances than Polygames.

Recall that Athénan is a Minimax approach different in many points from AlphaZero. Their basic differences are as follows. Athénan does not use a policy. It is based on Unbounded Minimax variants instead of MCTS. It learns as learning target the (partial) minimax value of states instead of the endgame value. Finally, Athénan learns the values of all the states of the search tree built during the match, while AlphaZero only learns the values of the states of the match (i.e. AlphaZero only learns the data of the states sequence of the match, which is a small subset of states of the game search tree).

In our experiments, we have compared and revealed the cost of generating the learning data. Athénan generates for the same duration 296 times more learned states than Polygames. This result is all the more striking since Athénan performs less than half as many state evaluations than Polygames (because contrary to Polygames, Athénan is programmed in Python and Athénan does not performed matches in parallel).

In addition, we have compared the win rates of the two zero learning algorithms by evaluating them against MCTS with UCT on a large number of games. Athénan obtains much better results than Polygames. In particular, Athénan is about 7 times faster than Polygames without reinforcement heuristic and much more than 30 times faster with reinforcement heuristic. Moreover, we have

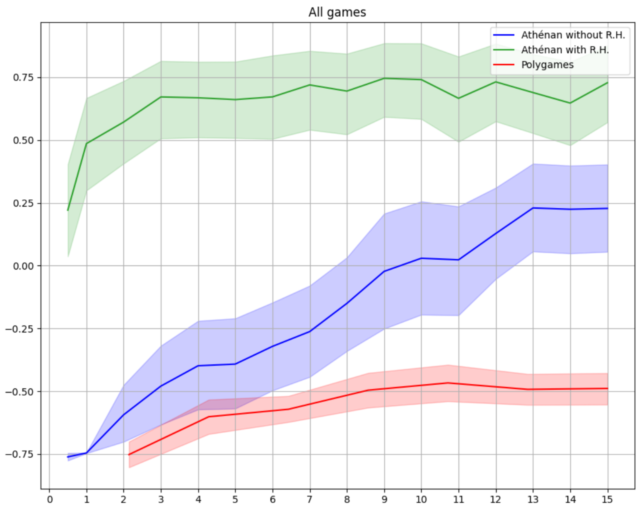

Figure 3: Evolution of average win rates minus average loss rates of Athénan with reinforcement heuristic (with R.H.), of Athénan without reinforcement heuristic, and of Polygames against MCTS with UCT along the 15 days of training and their stratified bootstrap confidence intervals over the 8 games.

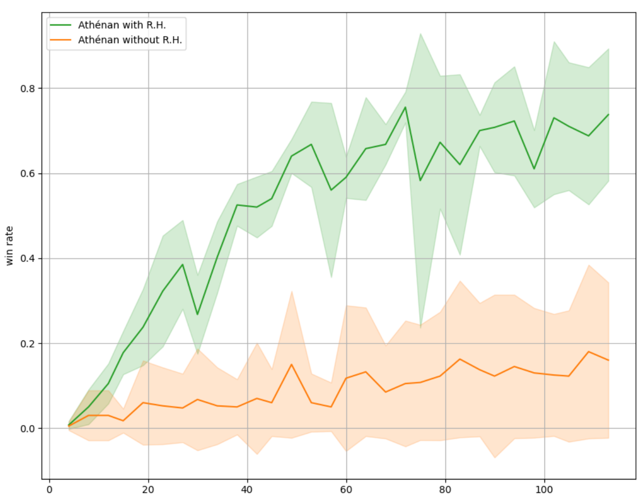

Figure 4: Evolution of average win rates of Athénan with and without reinforcement heuristic (R.H.) against Mohex 2.0, during 113 days of training (there is approximately one evaluation every 4 days ; each evaluation consists of 50 matches in first player and 50 other matches in second player). Shading is the 95% confidence interval.

performed another win rates comparison at Hex 13, in the context of a longer training that lasted 113 days, by evaluating them against Mohex 2.0, the best freely available Hex program. Athénan has obtained again much better results than Polygames.

Finally, we have made a last comparison at Othello 8, Othello 10, and Breakthrough, against top Polygames networks, having won

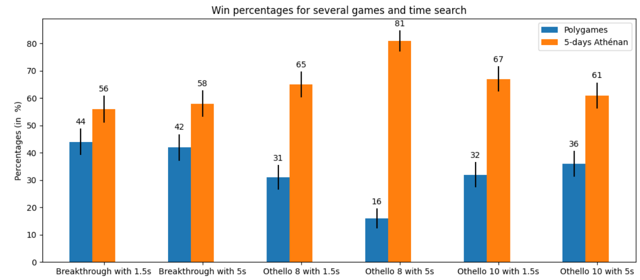

Figure 5: Results of 400 matches between Athénan (5 days of training) and Polygames (using tournaments Polygames networks) at Breakthrough, Othello 8, and Othello 10.

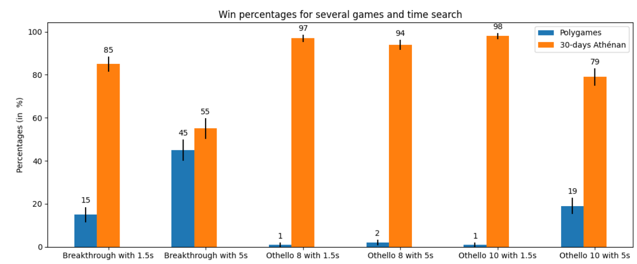

Figure 6: Results of 400 matches between Athénan (30 days of training) and Polygames (using tournaments Polygames networks) at Breakthrough, Othello 8, and Othello 10.

two gold medals and one silver medal at the 2020 TCGA tournaments. These Polygames networks have been trained for a week with over 100 GPUs. It is again Athénan that get the best results on each game, although its training only lasted 5 days and only required the use of half a GPU.

In conclusion, all these experiments show that for many games, reinforcement learning with Athénan is widely more efficient than with Polygames, at least for accessible learning times and reasonable resources use.

Note to conclude that Athénan faced Polygames during the 2020 Computer Olympiad and beat it at Othello 8, Othello 10, and Breakthrough. Athénan also beat other re-implementations of AlphaZero during this competition (at Surakarta and Clobber). In fact, 5 gold medals were won by Athénan for the following games: Othello 10, Breakthrough, Surakarta, Amazons, and Clobber. This was the first time that the same algorithm has won so many gold medals in the same year.

Athénan has again competed in the 2021 Computer Olympiad. This time, it won 11 gold medals (Hex 11, Hex 13, Hex 19, Havannah 8, Havannah 10, Othello 8, Surakarta, Amazons, Breakthrough, Brazilian Draughts, Canadian Draughts; there was no competition at Othello 10 and Clobber). Athénan notably beat Polygames at games where they met (Hex 13, Hex 19, Havannah 8, Havannah 10).

Athénan has again competed in the 2022 Computer Olympiad. This time, it won 5 gold medals (Surakarta, Breakthrough, Canadian Draughts, Santorini, and Ataxx). Thus, Athénan is still the defending champion at 13 games.

ACKNOWLEDGMENTS

Thanks to Nicholas Sowels for proofreading. This work was granted access to the HPC resources of IDRIS under the allocation 2020-AD011011461, AD011011461R1, AD011011461R2, AD011011461R3, and 2020-AD011011714 made by GENCI. This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

REFERENCES

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. 2021. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems* 34 (2021).

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2-3 (2002), 235–256.

[3] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (March 2012), 1–43.

[4] Cameron Browne, Matthew Stephenson, Éric Piette, and Dennis J.N.J. Soemers. 2020. A Practical Introduction to the Ludii General Game System. *Advances in Computer Games. Springer* (2020).

[5] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence* 134, 1-2 (2002), 57–83.

[6] Tristan Cazenave. 2015. Generalized Rapid Action Value Estimation. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015.* 754–760.

[7] Tristan Cazenave. 2016. Playout policy adaptation with move features. *Theor. Comput. Sci.* 644 (2016), 43–52.

[8] Tristan Cazenave, Yen-Chi Chen, Guan-Wei Chen, Shi-Yu Chen, Xian-Dong Chiu, Julien Dehos, Maria Elsa, Qucheng Gong, Hengyuan Hu, Vasil Khalidov, Li Cheng-Ling, Hsin-I Lin, Yu-Jin Lin, Xavier Martinet, Vegard Mella, Jeremy Rapin, Baptiste Roziere, Gabriel Synnaeve, Fabien Teytaud, Olivier Teytaud, Shi-Cheng Ye, Yi-Jun Ye, Shi-Jim Yen, and Sergey Zagoruyko. 2020. Polygames: Improved Zero Learning. *ICGA Journal* 42, 4 (December 2020), 244–256.

[9] Tristan Cazenave and Abdallah Saffidine. 2009. Utilisation de la recherche arborescente Monte-Carlo au Hex. *Revue d'Intelligence Artificielle* 23, 2-3 (2009), 183–202.

[10] Quentin Cohen-Solal. 2020. Learning to Play Two-Player Perfect-Information Games without Knowledge. *arXiv preprint arXiv:2008.01188* (2020).

[11] Quentin Cohen-Solal. 2021. Completeness of Unbounded Best-First Game Algorithms. *arXiv preprint arXiv:2109.09468* (2021).

[12] Quentin Cohen-Solal and Tristan Cazenave. 2023. Minimax Strikes Back: Technical Appendix. *arXiv preprint* (2023).

[13] Rémi Coulom. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games (Lecture Notes in Computer Science, Vol. 4630)*. Springer, 72–83.

[14] Richard Emslie. 2019. Galvanise Zero. https://github.com/richemslie/galvanise_zero.

[15] Sylvain Gelly and David Silver. 2011. Monte-Carlo Tree Search and Rapid Action Value Estimation in computer Go. *Artif. Intell.* 175, 11 (2011), 1856–1875.

[16] Guy Haworth and Nelson Hernandez. 2021. The 20th Top Chess Engine Championship, TCEC20. *ICGA Journal* 43, 1 (2021), 62–73.

[17] Shih-Chieh Huang, Broderick Arneson, Ryan B Hayward, Martin Müller, and Jakub Pawlewicz. 2013. MoHex 2.0: a pattern-based MCTS Hex player. In *International Conference on Computers and Games*. Springer, 60–71.

[18] Donald E. Knuth and Ronald W. Moore. 1975. An Analysis of Alpha-Beta Pruning. *Artif. Intell.* 6, 4 (1975), 293–326. https://doi.org/10.1016/0004-3702(75)90019-3

[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML'06) (LNCS, Vol. 4212)*. Springer, 282–293.

[20] Richard E Korf and David Maxwell Chickering. 1996. Best-first minimax search. *Artificial intelligence* 84, 1-2 (1996), 299–337.

[21] Tony A Marsland. 1987. Computer chess methods. *Encyclopedia of Artificial Intelligence* 1 (1987), 159–171.

[22] Yu Nasu. 2018. Efficiently updatable neural-network-based evaluation functions for computer shogi. *The 28th World Computer Shogi Championship Appeal Document* (2018).

[23] Gian-Carlo Pascutto. 2017. Leela Zero. https://github.com/leela-zero/leela-zero.

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[25] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

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

[27] Kei Takada, Hiroyuki Iizuka, and Masahito Yamamoto. 2019. Reinforcement learning to create value and policy functions using minimax tree search in hex. *IEEE Transactions on Games* 12, 1 (2019), 63–73.

[28] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and C Lawrence Zitnick. 2019. Elf OpenGo: An analysis and open reimplementation of AlphaZero. *arXiv preprint arXiv:1902.04522* (2019).

[29] David J Wu. 2019. Accelerating self-play learning in Go. *arXiv preprint arXiv:1902.10565* (2019).