4Hammer: a board-game reinforcement learning environment for the hour long time frame

Massimo Fioravanti
DEIB – Politecnico di Milano
Via G. Ponzio 34/5, I-20133 Milano, Italy
massimo.fioravanti@polimi.it

Giovanni Agosta
DEIB – Politecnico di Milano
Via G. Ponzio 34/5, I-20133 Milano, Italy
giovanni.agosta@polimi.it

Abstract

Large Language Models (LLMs) have demonstrated strong performance on tasks with short time frames, but struggle with tasks requiring longer durations. While datasets covering extended-duration tasks, such as software engineering tasks or video games, do exist, there are currently few implementations of complex board games specifically designed for reinforcement learning and LLM evaluation. To address this gap, we propose the *4Hammer* reinforcement learning environment, a digital twin simulation of a subset of *Warhammer 40.000* —a complex, zero-sum board game. *Warhammer 40.000* features intricate rules, requiring human players to thoroughly read and understand over 50 pages of detailed natural language rules, grasp the interactions between their game pieces and those of their opponents, and independently track and communicate the evolving game state.

1 Introduction

Machine Learning (ML) techniques have been able to tackle tasks with progressively longer time horizons [7], yet many tasks commonly performed by humans are still beyond the abilities of machines to perform. Games have long attracted research interest in artificial intelligence [5, 14, 13] as they provide structured environments with clearly defined rules within which an agent can perform actions as well as variable time horizon length. The two main categories of games relevant to ML are video games [4, 13, 15] and board games [16, 17, 10, 9]. The former relieve players of the mental burden of tracking the game state by providing user interfaces that present necessary information on-demand and enforce valid player actions. In contrast, board games typically provide physical components for maintaining game states, yet frequently leave tracking transient elements—such as turn order—to the players themselves, thus increasing the difficulty of playing them for an ML agent.

It is worth noting that board games are not an homogeneous class. Abstract games, such as chess and go, have simple rules but emerging complexity, while other games have an increased cognitive cost arising from large amount of rules and components – f.i., Magic: The Gathering has hundreds of different cards, each presenting its own tiny set of rules which are added to the baseline rules of the game, while *Warhammer 40.000* has many different pieces (miniatures) which behave according to their own rules variations. For human players, physical game components (e.g., cards reporting the rules) and/or on intangible qualities, such as their theme, are used to help in dealing with this complexity [6]. While abstract games offer a precise rules environment, thanks to their abstraction from reality, some commercial board games exhibit rules that are informal, incomplete, contain inaccuracies or implicitly rely on the player's understanding of the real world.

Consequently, learning and playing a new off-the-shelf board game from start to finish constitutes a complex, interactive human task that can span multiple hours, requiring players to navigate environments characterized by rule uncertainty and sometime even resolve disputes arising from differing interpretations of the game state.

Preprint. Under review.

It is worth noting that large amounts of rules make digital implementations of such games difficult, therefore large non-abstract board games digital implementations targeting ML are few. Many ML projects only offer abstract board games that are easy to implement [8, 11, 9].

In this paper we introduce *4Hammer*, the first digital implementation of *Warhammer 40.000* [3], a off-the-shelf board with large amounts of textual and situational rules and player tracked game state. Although *Warhammer 40.000* is not as universally known as chess or poker, *Warhammer 40.000* is by far the most widely played game in the category of miniature war games, with at least 300.000 games played in two-day tournaments since August 2023 [2]. Thus, an implementation amenable to ML would constitute an important stepping stone towards the automated solution of games with large rules sets.

4Hammer can be used both with and without a graphical engine to render the top down view of the game state, can serialize textual and tensor representations of the game state and thus be used both with LLMs and traditional reinforcement learning techniques. We demonstrate both capabilities with an experimental campaign.

The rest of this paper is organized as follows. In section 2, we briefly introduce *Warhammer 40.000* and its "Combat Patrol" variant, which is the focus of the proposed implementation. In section 4, we explain the architecture of the *4Hammer* implementation and its integration with graphical engines, LLMs, and Reinforcement Learning libraries, while in section 5 we provide an experimental assessment. Finally, in section 6 we draw some conclusions and highlight future directions.

2 Background

Warhammer 40,000 *Warhammer 40.000* is a non-deterministic, two-player, zero-sum tactical miniature board game. Each player selects various game pieces, known as *units*, each possessing distinct rules and abilities. These units are deployed onto a three-dimensional battlefield, where players alternate turns to maneuver their units strategically, engage opposing forces, and complete specific objectives.

Warhammer 40.000 presents several characteristics that pose challenges for machine learning approaches:

- **Extended Timeframe:** A single match can span several hours, with tournament rounds typically lasting between one and three hours. While part of this time is devoted to physically moving game pieces on the board, which does not happen in a digital implementation, the time spent in decision-making is still much longer than in abstract games.
- **Extensive Core Rules:** The core rules are detailed in a freely available, 60-page natural-language document, supplemented by an additional 35-page document covering advanced game mechanics. Not all rules apply in every match; f.i., rules related to vehicles might be irrelevant in games where no vehicle-type units are present.
- **Extensive Faction-Specific Rules:** The complete game offers hundreds of unit options, each possessing unique rules. Human players are typically expected to know the rules governing their chosen units and must communicate these rules to opponents upon request.
- **Player-Tracked Game State:** Many game effects persist across multiple actions or turns. While some rules explicitly instruct players on how to track such effects—such as placing specific tokens—other temporary effects require players to independently monitor and remember the current state. E.g., units typically move only once per turn, and it is the player's responsibility to remember which units have already moved.

Combat Patrol: *Warhammer 40.000* includes several game variants, called *modes*, one of which, Combat Patrol, serves as an introductory variant. This mode significantly reduces the complexity by limiting the combinations and total number of units that players select before gameplay, although the core rules remain unchanged.

3 The *4Hammer* environment

The *4Hammer* environment is a digital implementation of *Warhammer 40.000*'s Combat Patrol game mode. Its rules are encoded in *Rulebook*[1], a domain-specific language explicitly designed for creating reinforcement learning datasets. The graphical interface of the environment is powered by the Godot game engine [1], released under MIT license.

The *4Hammer* environment is designed recognizing that the field of ML is in continuous flux, and thus environments should support multiple use cases. In particular, we identified the following use cases, which can be combined as needed:

- **Perfect Information Reinforcement Learning:** The entire game state is directly accessible as observations provided to the reinforcement learning algorithm.
- **Imperfect Information Reinforcement Learning:** The game state is observed indirectly via a rendered, bird's-eye graphical view of the game board. Agents must independently track and maintain additional information that isn't explicitly provided.
- **State with Rules:** The full game rules are accessible, and agents select actions from a provided list of valid options.
- **State Only:** The formal rules implementation is disregarded, and agents directly manipulate the underlying state data structure as a working memory ("scratch pad") to carry out actions in the game.

Implemented systems The *4Hammer* environment currently implements 6 of the 35 available Combat Patrol factions, comprising a total of 22 distinct units, along with all necessary rules to use these units. This simplification does not detract significantly from the game, since the factions are never used all at the same time – rather, each player chooses a different faction.

3.1 Limitations

Rules not directly relevant to these selected units have not been implemented. Additionally, the physical game is originally played on a 44x30-inch board, where units can occupy continuous 3D positions. In *4Hammer*, we have discretized the board into 1-inch squares, modifying movement rules accordingly to accommodate this quantization. The original three-dimensional environment has also been simplified into a two-dimensional representation, removing obstacles and rendering all game elements as two-dimensional objects. Moreover, the optional secondary objective mechanics have been omitted from our implementation.

The omitted rules interfere with the game play at the tactical level where minute distances of where models are placed on the table are relevant but have little effect on the strategic level of the game, which focuses more on selecting the right game components to achieve objectives such as trying to eliminate the opponent game pieces. In our experiments, we have not observed neural networks or LLMs achieving good results at the strategic level; therefore, the finer tactical level is irrelevant. When the models will be able to perform well on the strategic level, the tactical level can be expanded to include all rules.

4 Architecture

Figure 1 shows the architecture of *4Hammer* from a build system point of view. The game simulation libraries are compiled into a shared library, and wrappers for Python, `C++` and Godot are generated. From the rules library, the handwritten graphical godot components we build the graphical engine. Python script can instead dynamically load the rules library. *4Hammer* can be found at https://github.com/rl-language/4Hammer .

4.1 Game simulation libraries

The Stats, State, and Rules are the core simulation components and contain the code to run it without the graphical engine. They supports bidirectional interoperability with Python, C and C++, provide

[1] https://github.com/rl-language/rlc/

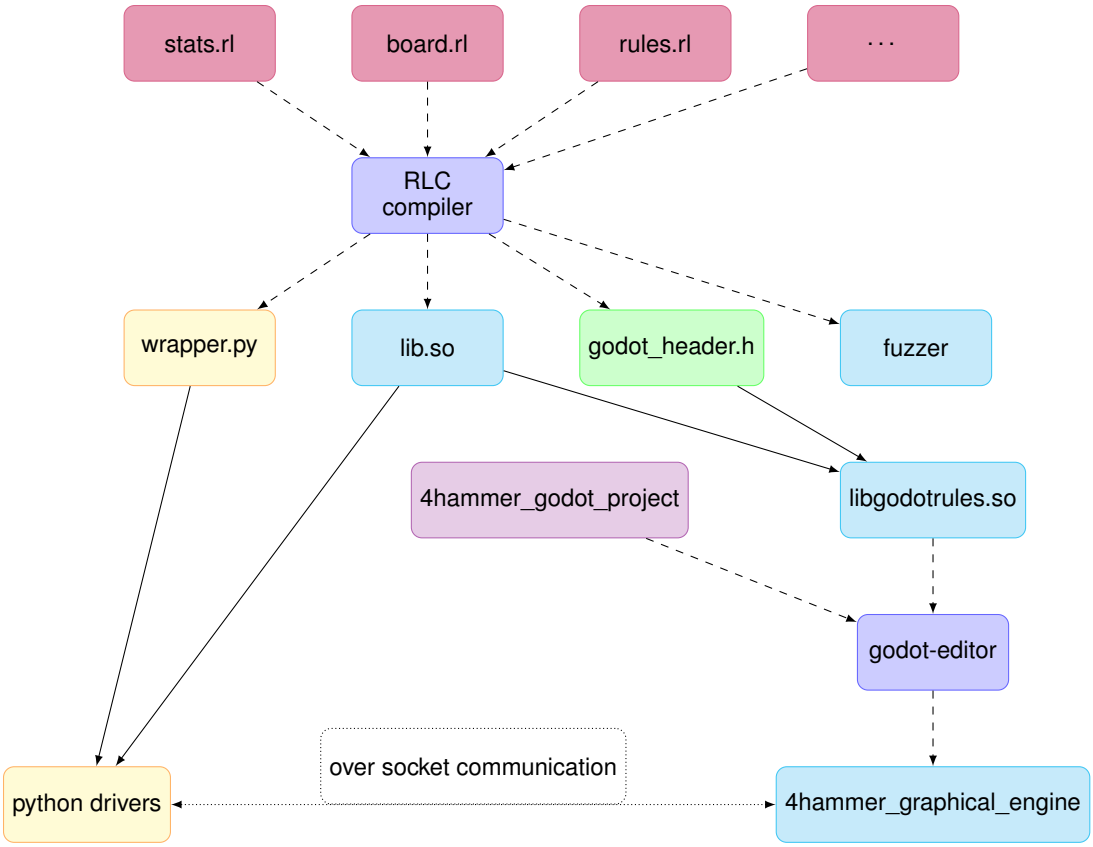

Figure 1: Hierarchical representation of the components in *4Hammer*. Purple components are implemented exclusively in *Rulebook*, while yellow components represent Python source files. The component labeled `4hammer_godot_project` comprises Godot engine files responsible for graphical elements. Cyan-colored nodes represent binary objects such as executables and libraries. Dashed edges indicate input-output relationships, illustrating inputs processed by tools (`rlc` or the Godot editor) to produce outputs. Solid lines represent composition, illustrating how various artifacts are combined to produce new components. Dotted lines indicate network communication occurring at runtime.

serialization to tensor encoding, textual encoding and binary encoding, and native handling of hidden information. These components are implemented in *Rulebook*, a domain-specific language that provides the mentioned features.

Each of these libraries is modular, depending only on required components, and can therefore be independently reused.

Stats The Stats library contains declarative data for all units implemented within *4Hammer*. This library can easily be expanded by adding rules for additional units.

Board The Board library defines all essential elements needed to represent the game state, excluding game sequences. All scalar values and containers within this library are annotated with their minimum and maximum possible values, enabling validation of game states. Being written purely in *Rulebook*, textual, binary, and tensor representations are automatically generated. Consequently, the game state can be seamlessly provided either to LLMs through a textual representation or used directly with traditional reinforcement learning methods.

Rules The Rules library encapsulates all gameplay sequences required to play a complete match from start to finish. For example, Figure 2 illustrates the control flow graph for the single_attack sequence, executed potentially hundreds of times per game when a player's unit interacts with

opposing units. Ellipses represent entry and exit points of sequences, solid boxes indicate atomic decisions made by players (e.g., allocating damage to game components), and dashed boxes represent nested sequences called within the current sequence. Arrows indicate potential transitions between nodes based on the game's state during simulation.

Sequences are reusable and composable, as demonstrated by Figure 2, where the roll_dice subsequence is utilized multiple times. This reusability facilitates the composition of new environments beyond those described in this document, for example, different game modes.

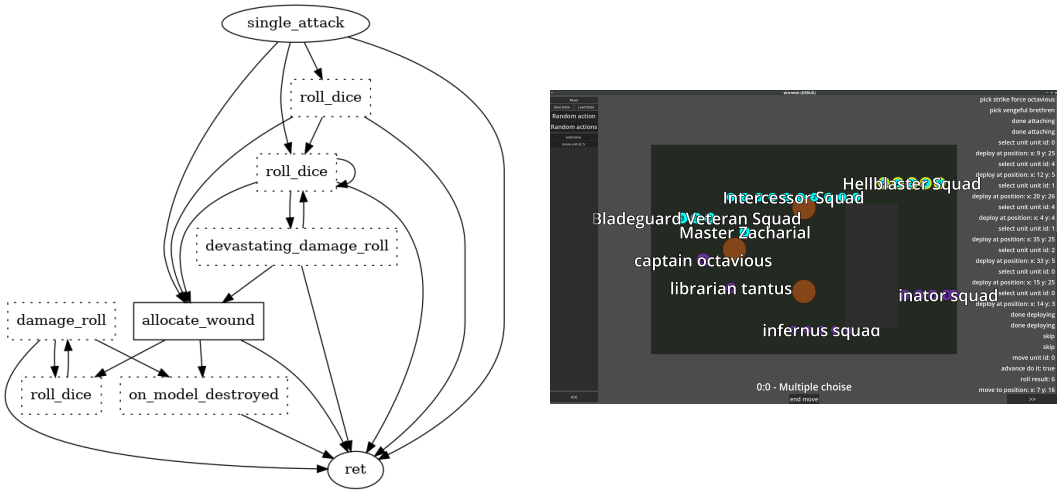

Figure 2: The single_attack game sequence (left) and the engine-level skirmish (right).

Compilation Outputs From the *Rulebook* source files, several outputs are automatically generated: a library implementing the rules, a fuzzer for automated testing, a Python wrapper library, and a wrapper header file for integration with the Godot engine.

4.2 Graphical Engine

Built upon the *Rulebook* libraries, *4Hammer* incorporates an auto-configuring graphical engine developed using Godot. The Godot engine dynamically links to the compiled *Rulebook* library (lib.so). At runtime, Godot scripts interpret and render the current game state, enabling modifications or extensions to *Rulebook* libraries without requiring changes to graphical components.

The graphical engine can load and visually represent any valid game state and operates entirely client-side within a web browser.

Figure 2 demonstrates the graphical user interface provided by the 4Hammer graphical engine.

4.3 Game Drivers

The final component of *4Hammer* comprises Game drivers, which load and run the game components independently of the graphical engine, allowing headless execution. Additionally, these drivers can connect over a network to control and manage execution within the graphical environment. The drivers are written in Python for ease of modification by the user.

5 Experimental Results

The primary goal of our experimental validation is to demonstrate that the *4Hammer* environment is robust enough to serve as a reliable training environment. Consequently, we do not focus on creating a state-of-the-art machine learning model to master the environment. Instead, we use the *Rulebook* compiler framework's built-in implementation of PPO [12], running on an Ubuntu machine equipped with an NVIDIA RTX 4070 GPU and an Intel Core i7-9700K CPU @ 3.60GHz to evaluate the headless mode. For validating the graphical engine driver, we employ Gemini 2 Flash.

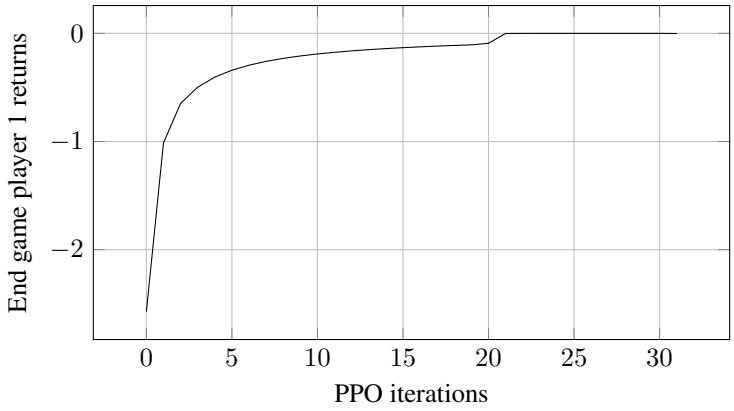

Figure 3: Average score obtained by the player 0 while training on single_shooting_maximize.rl

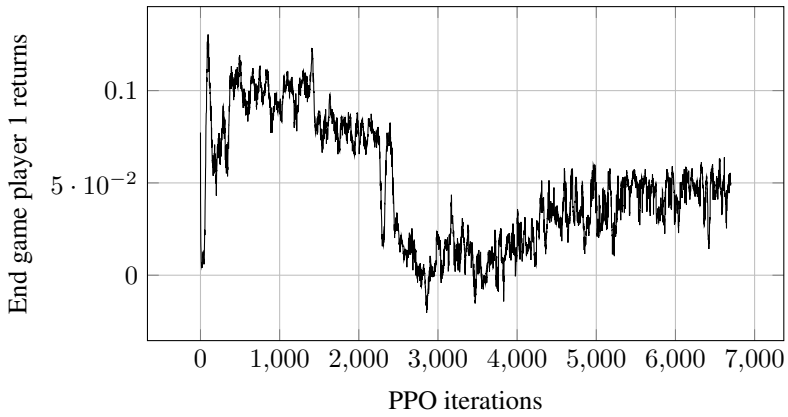

Figure 4: Average score obtained by the agent during training on `single_turn.rl`.

Headless Mode We provide two example RL files describing simplified subsets of the full game. The first, **single_shooting_maximize.rl**, enables an agent to select one of two available units, with the selected unit subsequently attacking a third target unit. The agent's objective is to maximize inflicted damage. Figure 3 presents the average score achieved using the command `rlc-learn`, with a learning rate set to 0.00001 and 1000 steps per environment.

The second example, **single_turn.rl**, simulates an entire game but limits each player to a single turn instead of the standard five. Unit selections are predetermined rather than chosen by the players. Additionally, the original scoring mechanism has been adjusted, as achieving significant score improvements within a single turn is not feasible. Instead, both players aim to maximize the difference in the number of remaining game pieces at the end of the match. Figure 4 illustrates the average reward achieved during training, showing a significantly more irregular trend due to the direct impact of opponent actions on the player's score. These results were generated using rlc-learn with a learning rate of 0.00001, 5,000 steps per environment, and active league play.

Graphical Engine Mode To validate the graphical engine and Python driver mechanisms, we utilized Gemini 2 Flash[2] to select actions. A python script keeps the internal state of the game and at each move provides the full textual serialized state to Gemini, along with all the valid moves in that state. Gemini selects a move among them, then the python script forwards the move to the graphical engine, displaying it on screen.

While Gemini actions were fairly poor, it was fairly competent at reading correctly the textual game state. As an example, in figure 5 we report the message that pushed Gemini to not reroll some dices when given the possibility.

[2] https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash

```
The game is in the Fight phase, specifically in the Charge phase where
the von Ryan's Leapers (unit 6) are attempting to charge the Infernus
Squad (unit 2). The current player is true (player 2 - tyranids).
The von Ryan's Leapers are already eligible to charge.
The question is whether to use any stratagems or abilities to
influence the charge roll.

Given that the current roll is 3 + 4 = 7, which seems high enough to reach
the target, and there's no visible way to improve this without spending
command points,  the best action is to skip any further actions and let
the charge proceed.
```

Figure 5: Gemini 2 Flash inner monologue when considering if it should try rerolling a pair of dices. The decision required the model to understand which pieces belonged to the opponent and which did not, what was the decision to perform in the moment, the previous result that was being prompted to reroll, what was the cost of rerolling, and if that cost was worth it or not.

6 Conclusions

In this paper, we introduce *4Hammer*, a reinforcement learning environment specifically designed to test the performance of machine learning techniques on tasks in the field of board games with hour-long horizons. We presented the modular architecture of *4Hammer*, capable of accommodating diverse scenarios from traditional reinforcement learning methods to LLM-driven interaction over network protocols. We then experimentally validated our environment under conditions representative of its intended use cases.

Future work for *4Hammer* includes leveraging advanced reinforcement learning techniques to achieve superhuman-level gameplay performance.

Acknowledgments and Disclosure of Funding

This work is partially supported by the Italian Ministry of Enterprises and Made in Italy (MIMIT) under the program "Accordi per l'innovazione nella filiera del settore automotive", through the grant "Piattaforma ed ecosistema cooperativo, C- ITS ETSI standard per la mobilità digitale integrata", numero F/340043/01-04/X59, CUP B49J24001210005, finanziato a valere del Bando MISE – ACCORDI PER L'INNOVAZIONE NEL SETTORE AUTOMOTIVE D.M. 31/12/2021 e DD 10/10/2022

References

[1] Godot. https://github.com/godotengine/godot.

[2] Statcheck databse. https://www.stat-check.com/the-meta.

[3] Warhammer 40.000 rules. https://www.warhammer-community.com/en-gb/downloads/warhammer-40000/.

[4] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*, pages 600–603. IEEE, 2020.

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

[6] George Skaff Elias, Richard Garfield, and K. Robert Gutschera. *Characteristics of Games*. The MIT Press, 2012.

[7] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh,

David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long tasks, 2025. [Online].

[8] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

[9] Shreya Patankar, Charutosh Usakoyal, Pruthviraj Patil, and Kaustubh Raut. A survey of deep reinforcement learning in game playing. In *2024 MIT Art, Design and Technology School of Computing International Conference (MITADTSoCiCon)*, pages 1–5. IEEE, 2024.

[10] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

[11] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. Ludii – the ludemic general game system. In G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, editors, *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 411–418. IOS Press, 2020.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. [Online].

[13] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

[14] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[15] Konstantinos Souchleris, George K Sidiropoulos, and George A Papakostas. Reinforcement learning in game industry—review, prospects and challenges. *Applied Sciences*, 13(4):2443, 2023.

[16] István Szita. Reinforcement learning in games. In *Reinforcement Learning: State-of-the-art*, pages 539–577. Springer, 2012.

[17] Konstantia Xenou, Georgios Chalkiadakis, and Stergos Afantenos. Deep reinforcement learning in strategic board game environments. In *European Conference on Multi-Agent Systems*, pages 233–248. Springer, 2018.