EasyChair Preprint

№ 13437

Piece by Piece: Assembling a Modular Reinforcement Learning Environment for Tetris

Maximilian Weichart and Philipp Hartl

July 20, 2024

Piece by Piece: Assembling a Modular Reinforcement Learning Environment for Tetris

Maximilian Weichart[1] and Philipp Hartl[1]

Abstract: The game of Tetris is an open challenge in machine learning and especially Reinforcement Learning (RL). Despite its popularity, contemporary environments for the game lack key qualities, such as a clear documentation, an up-to-date codebase or game related features. This work introduces *Tetris Gymnasium*, a modern RL environment built with *Gymnasium*, that aims to address these problems by being modular, understandable and adjustable. To evaluate *Tetris Gymnasium* on these qualities, a Deep Q Learning agent was trained and compared to a baseline environment, and it was found that it fulfills all requirements of a feature-complete RL environment while being adjustable to many different requirements. The source-code and documentation is available at on GitHub[2] and can be used for free under the MIT license.

Keywords: Tetris, Reinforcement Learning, Gymnasium, Library, Software Engineering

1 Introduction

Tetris is a game that is known around the world for its simple yet engaging design, and at the same time, Reinforcement Learning (RL) has been an active area of research for the last couple of years. A popular approach to advancing the field has been to develop agents that can beat human expert agents in various games, in hopes of extrapolating and generalizing the concepts learned from solving these problems and to applying them to problems in the real world. To that end, the Atari game suite has been widely used as a benchmark for RL algorithms. However, while significant progress has been made, a few games, such as *Pitfall*, continue to pose substantial challenges [Ec19]. Tetris is another such game that remains an unsolved problem in RL [LL20]. Its NP-hard nature [As20], combined with stochastic elements, sparse rewards, and the need for exploration and long-term planning, make it particularly challenging. Tetris's intuitive game principle and modest computational requirements render it a compelling setting for generating and evaluating novel RL approaches in a particular difficult setting.

2 Related work

Due to Tetris' popularity, there exist numerous RL environments for it, which can be categorized into two types: Those which use an emulator to run the binaries of the original

1 University of Regensburg, Faculty of Informatics and Data Science, Universitätsstraße 31, Germany, maximilian.weichart@stud.uni-regensburg.de; philipp.hartl@informatik.uni-regensburg.de

2 `https://github.com/Max-We/Tetris-Gymnasium`

games[Ka24], and those which implement their own game engine and APIs for usage [Co24; Ng24; Ru22]. Unfortunately, several of the existing projects are outdated, rendering them incompatible with the current Python versions and libraries. There exist up-to-date environments, but these often lack a clear documentation for their source code and on how to use them [Bu24]. Furthermore, most of the environments, especially those which are running the original game binaries via an emulator, lack key-game mechanics, such as the hold-functionality (see Sect. 2.1), and are impractical to customize. The goal of this work is to offer a comprehensive RL environment for the game of Tetris which solves the main problems of existing approaches and serves as a go-to implementation for other researchers to test their algorithms with. Based on *Gymnasium* [Br16; To24], which is the de facto standard framework for creating and using RL environments, we introduce a modularized, easily understandable and adjustable implementation of Tetris called *Tetris Gymnasium*.

2.1 Tetris concepts

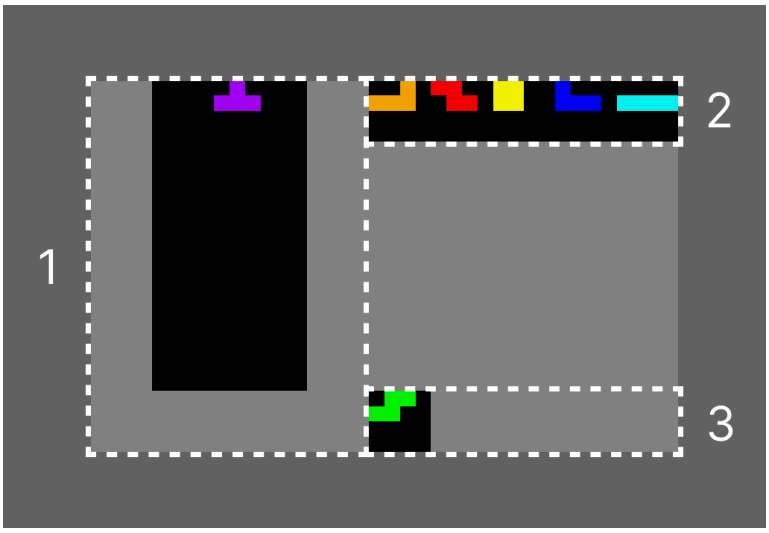

Fig. 1: Matrix (1) Queue with random generator (2) Holder (3)

A game of Tetris mainly consists of three components which are displayed in Fig. 1. These include the matrix ("board"), on which the Tetrominoes ("pieces") move, a queue which displays the incoming Tetrominoes, and a hold-function, which allows the agent (player) to swap out Tetrominoes during the game. The Tetrominoes can have different shapes and colors, and the order in which they appear is determined by a random generator. While playing the game, agents can score points by clearing lines (rows). The formula for calculating the score varies from game to game, and many versions of Tetris include special combos, such as those defined in the Tetris Design Guidelines[3].

3 https://ia804609.us.archive.org/27/items/2009-tetris-variant-concepts_202201

2.2 Reinforcement learning

By the definition of Sutton; Barto [SB18], RL can be expressed as a Markov Decision Process (MDP) where an agent interacts with an environment, as depicted in Fig. 2. At each iteration, the agent performs an action a_t based on a state s_t and subsequently receives a new state s_{t+1} and reward r_{t+1} from the environment. The goal of the agent is to maximize the cumulative rewards obtained over time.

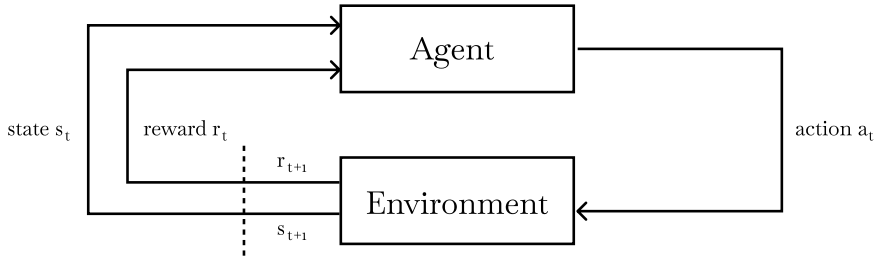

Fig. 2: The RL-loop consisting of an agent interacting with an environment. This figure is adapted from Sutton; Barto [SB18].

Tetris Gymnasium implements the environment from the RL-loop via a standardized API based on *Gymnasium*, formalizing the game of Tetris and offering ways to adjust the observations (the information available to the agent about a state), rewards and actions in a modular way.

3 Implementation

The following sections will introduce the technical implementation of *Tetris Gymnasium* and how it ensures its modularity, understandability and adjustability. In this context, Python was used as the programming language, to enable compatibility with *Gymnasium*.

3.1 Tetromino and Matrix

A fundamental design decision in *Tetris Gymnasium* is to represent the Tetrominoes and the matrix as NumPy arrays, making features of the game easily adjustable to different requirements, e.g. new Tetrominoes or matrix dimensions. The Tetrominoes are therefore represented as a 2-dimensional arrays, just like the playable matrix which a 2D-array is of shape (h, w) and can be adjusted in height h and width w size via parameters exposed by

the environment. Additionally, there exists a horizontal padding and a padding p on the bottom of the matrix, resulting in the matrix being of shape $(h + p, w + 2p)$. The padding p is a design choice, which eliminates the need to handle index-out-of-range errors when encountering edge cases of Tetrominoes moving over the edges of the matrix, resulting in an more robust and generalizable game logic. Each of the values in the array indicates a pixel of the game, which can be free space (= 0), padding(= 1), or a Tetromino($\geq$ 2).

3.2 Queue, Holder and Randomizer

As displayed in Fig. 1, the Tetris Design Guidelines define the components next queue, hold queue and random generator, which are implemented as separate classes `TetrominoQueue`, `TetrominoHolder` and `Randomizer`. The `TetrominoQueue` samples the upcoming Tetrominoes via a `Randomizer`, which is sampling from a bag by default. The `TetrominoHolder` lets the agent swap out the active Tetromino, resetting its position. All three components are used by the environment and can be modified, e.g. by increasing the length of the queue or changing the sampling algorithm of the randomizer, without affecting the rest of the game engine.

3.3 Environment and Game Engine

The *Gymnasium* environment API[4] defines a set of methods and attributes that every environment shall implement, which includes definitions for the observation and action space as well as methods like `step()` and `reset()` to interact with the environment.

In *Tetris Gymnasium*, the `Tetris`-class implements the `gymnasium.env` interface, making the environment compatible with *Gymnasium*. It also includes the Tetris game engine which is composed of attributes such as data structures for the matrix and Tetrominoes, and methods e.g. for collision-detection or moving Tetrominoes. Following the best practices introduced by *Gymnasium*, the environment may be extended and modified using various pre-defined- or custom-wrappers[5], offering a modular way to adjust the environment.

3.4 Documentation

The presented library aims to be easily understandable for its users, including beginners, and therefore offers three types of documentation. Firstly, the `examples` directory of the project includes self-contained scripts that showcase potential use-cases for the library. Secondly, the repository implements a collection of pre-commit-hooks similar to the official *Gymnasium* code-base, configured with a linter, formatter and a docstring parser, which

4 `https://gymnasium.farama.org/api/env/`
5 `https://gymnasium.farama.org/api/wrappers/`

should lead to a homogeneous code-base. Thirdly, the docstrings are combined with optional markdown files via *Sphinx* [6] to offer an extensive documentation on the web[7].

4 Evaluation

To showcase its simple interoperability with standard RL algorithms and libraries, we compared our implementation with the `ALE/Tetris-v5` environment from the official *Gymnasium* library for training a Deep Q Learning (DQN) agent [Mn13]. Both environments have been set up in the same way using the `dqn_atari.py`[8] script from the *CleanRL* project [Hu21]. The only adjustments made to the script were specifying the environment-id and lowering the number of time steps. The game-configuration for *Tetris Gymnasium* closely resembles the one for `ALE/Tetris-v5`, including gravity, identical Tetrominoes and matrix dimensions. Both the environments have been trained successfully, and the results were logged[9] in *Weights and Biases*.

While this evaluation does not focus on the quality of the resulting DQN-agents, it can be seen that the agents improve in the game in both cases, validating that the environments are functioning properly. Furthermore, this evaluation shows that while both *Gymnasium*-based environments can be directly integrated and used in the training process, *Tetris Gymnasium* offers additional options for customizing the game, where other environments don't. The documentation for *Tetris Gymnasium* also includes detailed information about these individual components and configurations, making it more accessible and understandable for end-users. This level of detail and customization is not present in `ALE/Tetris-v5` or other previously mentioned Tetris environments.

5 Conclusion

This paper introduces *Tetris Gymnasium*, a RL environment that aims to solve the limitations of other Tetris environments by being modular, understandable and adjustable, eliminating the need to spend significant time on implementing the game and APIs themselves. We have briefly shown that the environment offers advantages over other solutions and can be easily integrated into existing projects.

The development of *Tetris Gymnasium* is an ongoing process, and the current iteration of the environment also has its limitations, such as the lack of advanced scoring mechanisms (T-Spins), rendered frames not being upscaled, and the library not being published on the Python Package Index yet. Furthermore, we also plan to further incorporate feedback from

6 `https://www.sphinx-doc.org/en/master/`

7 `https://max-we.github.io/Tetris-Gymnasium/`

8 `https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/dqn_atari.py`

9 Tetris Gymnasium: `https://api.wandb.ai/links/go-apps-github/45n4shht`, Baseline: `https://api.wandb.ai/links/go-apps-github/16c8bmlr`

researchers using the library in their work and plan to evaluate the ease of use in more detail. It would be interesting to implement the rules and mechanics of a multiagent-version of Tetris, such as Tetr.io[10] and to consider integrating *Tetris Gymnasium* into the official *Gymnasium* library.

References

[As20] Asif, S.; Coulombe, M.; Demaine, E. D.; Demaine, M. L.; Hesterberg, A.; Lynch, J.; Singhal, M.: Tetris is NP-hard even with $O(1)$ rows or columns, 2020, DOI: 10.48550/arXiv.2009.14336, arXiv: 2009.14336[cs], URL: http://arxiv.org/abs/2009.14336, visited on: 05/13/2024.

[Br16] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W.: OpenAI Gym, 2016, DOI: 10.48550/arXiv.1606.01540, arXiv: 1606.01540[cs], URL: http://arxiv.org/abs/1606.01540, visited on: 05/13/2024.

[Bu24] Butera, J.: jaybutera/tetrisRL, original-date: 2017-08-11T23:43:54Z, 2024, URL: https://github.com/jaybutera/tetrisRL, visited on: 05/17/2024.

[Co24] Cox, M.: michiel-cox/Tetris-DQN, original-date: 2019-10-05T12:44:03Z, 2024, URL: https://github.com/michiel-cox/Tetris-DQN, visited on: 05/13/2024.

[Ec19] Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; Clune, J.: Go-Explore: a New Approach for Hard-Exploration Problems. CoRR abs/1901.10995, 2019, arXiv: 1901.10995, URL: http://arxiv.org/abs/1901.10995.

[Hu21] Huang, S.; Dossa, R. F. J.; Ye, C.; Braga, J.: CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms, 2021, DOI: 10.48550/arXiv.2111.08819, arXiv: 2111.08819[cs], URL: http://arxiv.org/abs/2111.08819, visited on: 05/13/2024.

[Ka24] Kauten, C.: Kautenja/gym-tetris, original-date: 2018-05-24T23:56:21Z, 2024, URL: https://github.com/Kautenja/gym-tetris, visited on: 05/13/2024.

[LL20] Liu, H.; Liu, L.: Learn to Play Tetris with Deep Reinforcement Learning. 2020.

[Mn13] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M.: Playing Atari with Deep Reinforcement Learning, 2013, DOI: 10.48550/arXiv.1312.5602, arXiv: 1312.5602[cs], URL: http://arxiv.org/abs/1312.5602, visited on: 05/13/2024.

[Ng24] Nguyen, V.: uvipen/Tetris-deep-Q-learning-pytorch, original-date: 2020-03-29T10:35:44Z, 2024, URL: https://github.com/uvipen/Tetris-deep-Q-learning-pytorch, visited on: 05/13/2024.

[Ru22] Russell, T.: tristanrussell/gym-simpletetris, original-date: 2022-03-28T09:50:36Z, 2022, URL: https://github.com/tristanrussell/gym-simpletetris, visited on: 05/13/2024.

[SB18] Sutton, R. S.; Barto, A. G.: Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA, 2018, ISBN: 0262039249.

[To24] Towers, M.; Terry, J. K.; Kwiatkowski, A.; Balis, J. U.; de Cola, G.; Deleu, T.; Goulão, M.; Kallinteris, A.; KG, A.; Krimmel, M.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Tan, A. J. S.; Younis, O. G.: Gymnasium, original-date: 2022-09-08T01:58:05Z, 2024, URL: https://github.com/Farama-Foundation/Gymnasium, visited on: 05/13/2024.

10 https://tetr.io/