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

Maximilian Weichart and Philipp Hartl

May 26, 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, such as Chess [Si17] or Dota 2 [Op19], in hopes of extrapolating and generalizing the concepts learned from solving these problems and to apply them to problems in the real world. Tetris qualifies as an interesting challenge to work on, as it can be an NP-hard problem [As20] and has a rich history of both traditional and RL approaches [AŞ19] to build upon.

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 games[Ka24], and those which implement their own game engine and APIs for usage [Co24; Ng24; Ru22b]. However, several of the existing projects are outdated, making them incompatible with current Python versions and libraries.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

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`

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.

To ensure better comparability of RL algorithms and to ease the transfer from one problem to another, OpenAI has released Gym [Br16], which is currently maintained under the name *Gymnasium* by the Farama Foundation [To24]. It offers environments for various RL problems via a standardized API and is fully open-source. Based on *Gymnasium*, this work presents a modularized, easily understandable and adjustable implementation of Tetris called *Tetris Gymnasium*. To ensure a low entry hurdle even for beginners, it comes with an extensive documentation of both the open-sourced code and its potential use-cases. Compared to other Tetris environments, *Tetris Gymnasium* is fully customizable and up-to-date, employing best practices from software engineering.

2.1 Tetris concepts

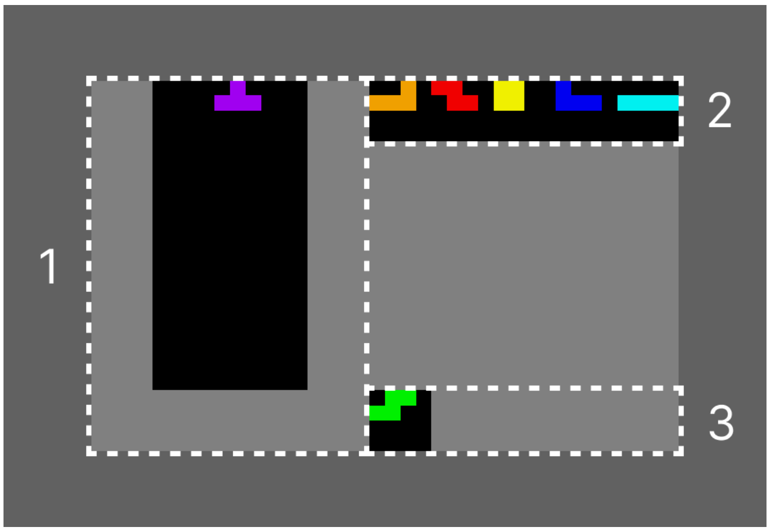

Fig. 1: Matrix (1) Queue with random generator (2) Holder (3)

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 Russell et al. [Ru22a], in RL the agent finds themselves in a Markov Decision Process (MDP), which consists of:

- **States** S: A set of states that an agent can find themselves in
- **Actions** $A(s)$: A set of actions that an agent can choose from in a given state
- **Transition Model** $P(s'|s,a)$: Transition model from one state to another
- **Reward Function** $R(s,a,s')$: Reward function

Tetris Gymnasium is based on *Gymnasium* and offers a way to model Tetris as an MDPs via a standardized API, formalizing the problem to be approachable for RL methods.

3 Implementation

The following sections will introduce the technical implementation of *Tetris Gymnasium* and how it ensures its modularity, understandability and adjustability.

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 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 test the practical quality of the environment, the *Tetris Gymnasium* has been used to train a Deep Q Learning (DQN) agent [Mn13]. The training script is based on `dqn_atari.py`[8] from the *CleanRL* project [Hu21], which offers various self-contained training scripts using *Gymnasium*, and can be found under `examples/train_cnn.py`. The only adjustments that have been made were lowering `total_timesteps` to 500,000 and changing the `environment_id`-string to the respective environment.

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

As a comparison to *Tetris Gymnasium*, the `ALE/Tetris-v5` environment from the official *Gymnasium* library has been chosen as a baseline. The training script tries to represent a real-world usage of the environment as closely as possible and includes an extensive setup for training a DQN-agent with multiple wrappers and detailed logging via the *Weights and Biases* platform. By using *Tetris Gymnasium* in this setting and by opting to make no significant adjustments of the code provided by the *CleanRL* project, it should be tested if the library fulfills the requirements that a feature-complete RL environment should offer.

Both the `ALE/Tetris-v5` and *Tetris Gymnasium* environment were successfully trained on an accelerated machine and the results have been 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, being able to clear rows more frequency over the training process, validating that both environments could be to train a DQN-agent.

Overall, it can be shown that *Tetris Gymnasium* can be directly integrated and used in the training process for an RL agent without any special adjustments. Additionally, to the features of the baseline environment `ALE/Tetris-v5`, *Tetris Gymnasium* offers many options for directly customizing the rewards, observation space, action space and Tetris-related game-features.

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 shown that the environment offers these 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. Besides overcoming these limitations, 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

[AŞ19] Algorta, S.; Şimşek, Ö.: The Game of Tetris in Machine Learning, 2019, arXiv: `1905.01652[cs]`, URL: `http://arxiv.org/abs/1905.01652`, visited on: 03/18/2024.

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

10 `https://tetr.io/`

[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.

[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.

[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.

[Op19] OpenAI; Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Dębiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Józefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J.; Petrov, M.; Pinto, H. P. d. O.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; Zhang, S.: Dota 2 with Large Scale Deep Reinforcement Learning, 2019, DOI: 10.48550/arXiv.1912.06680, arXiv: 1912.06680[cs, stat], URL: http://arxiv.org/abs/1912.06680, visited on: 05/13/2024.

[Ru22a] Russell, S. J.; Norvig, P.; Chang, M.-w.; Devlin, J.; Dragan, A.; Forsyth, D.; Goodfellow, I.; Malik, J.; Mansinghka, V.; Pearl, J.; Wooldridge, M. J.: Artificial intelligence: a modern approach. Pearson, Harlow, 2022, ISBN: 978-1-292-40113-3.

[Ru22b] 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.

[Si17] Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; Hassabis, D.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017, DOI: 10.48550/arXiv.1712.01815, arXiv: 1712.01815[cs], URL: http://arxiv.org/abs/1712.01815, visited on: 05/17/2024.

[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.