# Proposal for a Comparison of Reward Functions for Tetris Deep Q-Learning

Emilio Kiryakos
*UIC College of Engineering*
*University of Illinois at Chicago*
Chicago, USA
ekirya2@uic.edu

Dev Patel
*UIC College of Engineering*
*University of Illinois at Chicago*
Chicago, USA
dpate366@uic.edu

Markus Perez
*UIC College of Engineering*
*University of Illinois at Chicago*
Chicago, USA
mgperez2@uic.edu

Michael Klimek
*UIC College of Engineering*
*University of Illinois at Chicago*
Chicago, USA
mklim7@uic.edu

Christopher Bejar
*UIC College of Engineering*
*University of Illinois at Chicago*
Chicago, USA
cbeja2@uic.edu

***Abstract*—Our project intends to train multiple agents to play the game of Tetris using Deep Q-Learning in order to 1) evaluate the efficacy of different reward functions for Tetris and 2) build, to our knowledge, the first open-source dataset of Tetris Gymnasium observations (current state, action, reward, next state) for possible use by others in future offline reinforcement learning Tetris projects**

## I. Introduction

Tetris is a popular puzzle game developed by Alexey Pajitnov in 1984, which has players solve the game by stacking block objects in order to clear lines for points, all while intensifying the pace of the gameplay as the game progresses [9].

The game of Tetris has a variety of block shapes and colors with line, square, T-shape, l-shape and zig-zag blocks, all also called ”tetrominoes”. The game is scored based on how many lines were cleared at one time and how fast were the lines were cleared. Players of the game have to strategically think about the placement and rotation of Tetris blocks to be able to maximize the number of lines cleared at once and manage the blocks the player holds and has stored in the background of the game.

As the game progresses, the player places the blocks from bottom to top and the game is over when the any part of a block reaches the top of the stack, also called the ”Skyline” [9].

Reinforcement learning is a major machine learning paradigm which focuses on the study of intelligent agent behavior with the goal of reward maximization, where rewards are scores accumulated from a reward function that rates different behaviors.

Q-learning is a model-free reinforcement learning algorithm first described by Chris Watkins in 1989 [10].

Deep Q-learning is an extension of Q-learning to deep learning which was first shown by Google DeepMind in 2013 [11].

## II. Problem Statement and Objective

Due to the goal of reward maximizing, the design of reward functions is crucial for efficiently and adequately training agents. With a game as complex as Tetris, there are numerous considerations one must deal with. To what degree should we reward individual line clears? How can we properly incentivize going for clears of four rows at a time? How should we handle storing a tetromino? Etc.

With no clear ”correct” reward function, we primarily aim to benchmark and compare different informed reward function designs.

Additionally, in our research for this project, we came across recent advancements in **offline** Deep Q-Learning. Notably, we were interested in the Random Ensemble Mixture (REM) algorithm proposed by Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi in 2020 [12] because of it's performance in classic Atari games and the fact that no one has implemented the algorithm on Tetris before.

While reading the paper, we realized that this is because the offline algorithm requires extremely large datasets of replay experience, of which none exist for Tetris.

Thus, our secondary objective is to create a dataset for this purpose. While we certainly won't be able to make one of a desirable size, hopefully we can lay the building blocks for any interested parties to build on in the future.

## III. Literature Review

Using Machine Learning to create a agent to play for you is not an innovative idea. Tetris has been played using Reinforcement Learning techniques as early as 2008 [1]. There have also been those who have their findings posted online like Nicholas Lundgaard and Brian McKee from the University of Oklahoma [2] or Matt Stevens and Sabeek Pradhan from Stanford University [3].

### A. Others' Implementations and Notes

Lundgaard and McKee made an excellent point about the reasoning for choosing Reinforcement Learning vs other forms of Machine Learning. The point out that it's not a good game to have supervised learning because it is very difficult to tell what was a good move and what was not [2]. Tetris is just one of those games where you don't think one move at a time, but many steps at once.

### B. Others' Perspective and Motivations

Stevens and Pradhan had a similar reason for wanting to implement Machine Learning algorithms to plat Tetris. They state that "...neural networks can have remarkable performance at game playing, using a minimal amount of prior information about the game" [3], which definitely lends itself well to Deep Reinforcement learning where there is very little prior information. They also state that their motivation in their attempt was to implement a neural network in a way that would be fun, which is very similar to our motivation as well.

### C. Reinforcement Learning Techniques and Differences

**The following information is referenced from "Q-Learning" and "Deep Q-Learning" respectively. [13], [14]**

Traditional Reinforcement Learning involves an agent that will interact with its environment and receive either positive or negative "rewards" for performing certain actions. The agent tries to improve its behavior based on its rewards.

Q-Learning is an implementation of Reinforcement Learning that involves a Q table in which all of the agent's actions and states are assigned a predicted reward value called a Q value. It is a method of Reinforcement Learning that is very close to how Humans actually learn as well. Q-Learning is typically used in small environments that can make good use of the Q Table due to a small number of inputs and states. It is also able to fix mistakes made during training and make it very improbable for that mistake to happen again.

Deep Q-Learning is a similar implementation of Q-Learning but instead of using a Q Table, it uses two neural networks. The reason for this is because the implementation of a Deep Q-Learning network involves some variability. Having a second neural network makes the model more stable and leads to better outcomes. The first neural network is typically used to adjust the parameters of the network while the second is typically used to compute that variability, but has frozen parameters. This ensures that the model is working as intended. You would typically use Deep Q-Learning when you have many states and inputs where using Q-Learning's Q Table may be impractical.

## IV. Data

The data within the project is the experience replay data (our training data), our predicted Q-Values from our Deep Q-Network, our target Q-Values from our Target Network, and the calculated loss from the predicted and target Q-Values. The experience replay data will contain current state, action, reward, and next state data from which our rewards will be calculated from our reward functions and the actions will be selected by our $\epsilon$-greedy algorithm. Q-values are values determining the quality of state, action pairs. Our predicted Q-values will be used for action selection and be subject to change as our Deep Q-Network's weights are adjusted to minimize the difference between predicted and target Q-values. Our target Q-values define the relationship between current Q-values, the immediate reward, and the discounted future rewards. As our agents train, we will record average reward, cumulative reward, loss, and q-value data for each training episode to use as metrics for our analysis.

## V. Methods and Feasibility

In order to facilitate the long training times which will be required for our agents and to allow for easy cooperation, we have decided to use a persistent virtual machine for this project.

The first step of the project will be environment setup (i.e. how will we run the game? how will we get the game states?). For this step, we have identified three different possibilities:

1) **PyGame [4]:** We use PyGame to create a Tetris clone and create all necessary functions ourselves
2) **PyGame [4] and Gymnasium [5]:** We use PyGame to create a Tetris clone and use Gymnasium, a fork of OpenAI's Gym, for it's useful reinforcement learning features (Env, a markov decision process (MDP) representation class, and it's API)
3) **Gymnasium [5] and Tetris Gymnasium [6]:** We use Gymnasium, a fork of OpenAI's Gym, for it's useful reinforcement learning features (Env, a markov decision process (MDP) representation class, and it's API) and Tetris Gymnasium, a modular and adjustable Tetris environment integrated with Gymnasium

Of these approaches, we plan to use **(3)** for simplicity's sake but, will use **(2)** in the case that there are any technical issues or limitations.

Once we have our environment setup, we can go about the process of creating our agent. As mentioned in our literature review, Deep Q-Learning presents numerous benefits for the task of agentic game playing over traditional Q-Learning and other methods so we have chosen it as our approach. As Deep Q-Learning takes advantage of deep neural networks, we have decided to use **PyTorch** [7] and **Keras** [8] for our deep learning tasks due to their reputation as premier deep learning libraries.

During agent training, we plan to use learning curves (plots of average and/or cumulative reward against training episodes), loss function plots, and Q-value plots to measure the efficacy of a given reward function. Once we have the data for a reward function, we will move on to the next as time permits.

On the topic of feasibility, there are many factors at play, from the complexity of Tetris to the performance of our hardware. Thus, while we believe we can accomplish our goal, we are prepared to adjust our target deliverable to one model with "good" performance alongside it's experience replay data if our initial aim proves to be too ambitious for our time constraint or if we run into significant difficulties.

## VI. Time and Team Assignment

This proposal is comprised of seven sections which were worked on as follows:

- **Introduction:** Emilio Kiryakos and Markus Perez
- **Problem Statement and Objective:** Emilio Kiryakos and Markus Perez
- **Literature Review:** Christopher Bejar
- **Data:** Michael Klimek and Markus Perez
- **Methods and Feasibility:** Markus Perez
- **Time and Team Assignment:** Dev Patel and Markus Perez
- **References:** Christopher Bejar and Emilio Kiryakos

For which, every group member reviewed and gave feedback on each section

Given a project presentation date of 12/03/24, we have given ourselves a deadline of 11/29/24 for the technical aspects of the project, 12/02/24 for any presentation materials, and 12/10/24 for our final report drafting.

We have identified the following sequential steps needed to complete the project and have assigned them among ourselves:

1) **Identifying a VM provider and VM setup:** Christopher Bejar
2) **Setup and testing of game enviroment in VM:** Emilio Kiryakos
3) **Implementing of Deep Q-Network:** Markus Perez
4) **Implementing Experience Replay with data storage:** Dev Patel
5) **Implementing $\epsilon$-Greedy Algorithm:** Michael Klimek
6) **Implementing Target Network and training loop:** Markus Perez
7) **Implementing evaluation metric recording and plotting:** Emilio Kiryakos
8) **Testing our implementation on a simple toy Gymnasium enviroment:** Emilio Kiryakos
9) **Starting and monitoring training on Tetris Gymnasium enviroment:** Michael Klimek
10) **Designing and addition of additional reward functions:** Markus Perez
11) **Starting and monitoring of training with different reward functions:** Dev Patel
12) **Analysis of recorded metrics and plots:** Christopher Bejar

We plan to finish each setup and algorithm implementation step within a couple of days after the completion of the previous one in order to maximize the amount of time for training our agents. While we don't anticipate additional tasks, we are prepared to adjust these steps and roles as needed.

For the project presentation, we plan to each make slides on and discuss our respective tasks.

Lastly, for the final report, we plan to split up tasks as follows:

- **Introduction:** Michael Klimek
- **Problem Definition:** Emilio Kiryakos
- **Data and Experimental Design:** Markus Perez
- **Results and Analysis:** Christopher Bejar
- **Remarks and Future Works:** Markus Perez
- **Novelty Statement:** Emilio Kiryakos
- **Contribution Summary:** Dev Patel

## References

[1] szityu01, "Reinforcement learning agent plays Tetris," YouTube, May 28, 2008. https://www.youtube.com/watch?v=Uf5d3TwiiqI

[2] N. Lundgaard and B. Mckee, "Reinforcement Learning and Neural Networks for Tetris." Available: https://www.idi.ntnu.no/emner/it3105/materials/neural/lundgaard-tetris-ann.pdf

[3] M. Stevens and S. Pradhan, "Playing Tetris with Deep Reinforcement Learning." Available: http://vision.stanford.edu/teaching/cs231n/reports/2016/pdfs/121_Report.pdf

[4] Pygame, "Pygame Front Page." Available: https://www.pygame.org/docs/

[5] Farama Foundation, "Gymnasium Documentation." Available: https://gymnasium.farama.org/

[6] Max-We, "Tetris-Gymnasium." Available: https://github.com/Max-We/Tetris-Gymnasium?tab=readme-ov-file

[7] PyTorch, "PyTorch documentation." Available: https://pytorch.org/docs/stable/index.html

[8] Keras, "Keras 3 API documentation." Available: https://keras.io/api/

[9] Tetris, "About Tetris®", Tetris, 1985-2024. Available: https://tetris.com/about-us.

[10] C. John and C. H. Watkins "Learning from Delayed Rewards." Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[11] Volodymyr Mnih et al., "Playing Atari with Deep Reinforcement Learning." Available: https://arxiv.org/pdf/1312.5602

[12] R. Agarwal, D. Schuurmans and M. Norouzi, "Reinforcement Learning and Neural Networks for Tetris." Available: https://arxiv.org/pdf/1907.04543v4

[13] K. Chanda, "Q-Learning," GeeksforGeeks, Feb. 06, 2019. https://www.geeksforgeeks.org/q-learning-in-python/

[14] A. Gupta, "Deep Q-Learning," GeeksforGeeks, Jun. 13, 2019. https://www.geeksforgeeks.org/deep-q-learning/