Tetris AI

Kim Anna

acoryk1@outlook.com

November 1, 2025

1 Introduction

Tetris is a well-known puzzle game where the player controls falling tetrominoes, aiming to complete horizontal lines without gaps. Once a line is filled, it disappears and rewards the player with points. As the level increases, the falling speed grows, and the available maneuvering space shrinks. From a machine learning perspective, Tetris is an excellent environment for experimentation due to the following properties:

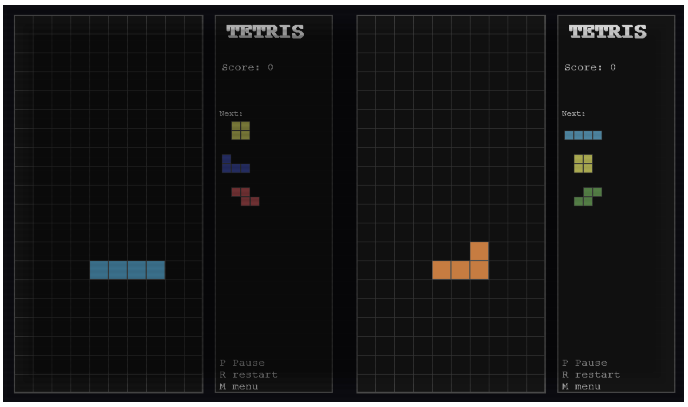

Figure 1: Tetris dual mode.

- **Huge State Space:** The game has an enormous number of possible board configurations (estimated around 2^{200}), making exact optimization infeasible and suitable for approximate or neural approaches [1].
- **Clear State–Action–Reward Interface:** Each game state (board + active piece) and action (move, rotate, drop) can be easily encoded for learning algorithms such as DQN, CNN, or LSTM agents.
- **Simple yet Nontrivial Visual Representation:** The 2D grid structure makes it computationally efficient for convolutional models while maintaining nontrivial gameplay complexity.
- **Balanced Complexity:** The game rules are simple, yet achieving high performance requires long-term strategy, making Tetris a perfect testbed for comparing heuristic and neural AI agents [2].

Several studies have explored reinforcement learning and neural network approaches in Tetris [**?** 3, 4].

2 Project Objectives

- **Development of a heuristic Tetris-playing bot.** Design a bot based on a heuristic evaluation system that analyzes all possible moves and selects the optimal one using predefined parameters — penalties for height, holes, and uneven surfaces, as well as rewards for cleared lines. These parameters will be tuned empirically. The bot's decision-making process will be used to generate datasets for subsequent supervised learning experiments.
- **Investigation and comparison of different neural network architectures for Tetris.** Experiment with several neural network models potentially suitable for this task, including Long Short-Term Memory networks (LSTM), Convolutional Neural Networks (CNN), and Deep Q-Networks (DQN). Evaluate their effectiveness, applicability, and actual advantages compared to the heuristic approach.

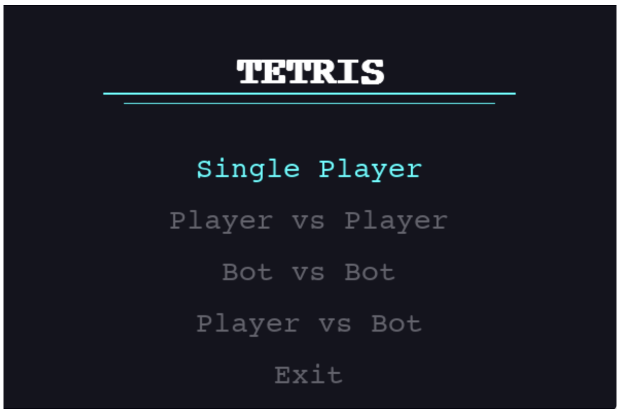

Figure 2: Menu with game mode selection.

- **Development of a game interface and interaction system for bots and players.** Implement a user interface and underlying logic supporting:
  - matches between two bots;
  - autonomous play for individual bots;
  - human participation, either solo or against another human or a bot.

3 Heuristic-based AI

The bot component implements a heuristic-based AI for Tetris.

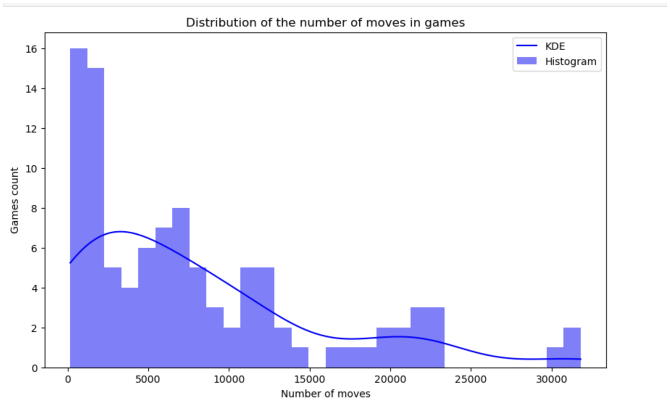

Figure 3: Heuristic bot results

It iterates through all valid rotations and horizontal positions of the current piece, simulates its drop, and evaluates the resulting board state using a linear combination of simple features:

- Aggregate height — the total height of all columns;
- Holes — the number of empty cells beneath filled blocks;
- Bumpiness — the difference in heights between adjacent columns;
- Cleared lines — the number of lines removed in a single move.

This approach represents a classic heuristic strategy, commonly used in early Tetris AIs before the rise of reinforcement learning methods. As a result, the bot plays in a stable and human-like manner, making intelligent decisions without any data training or neural networks.

4 LSTM

LSTM model for Tetris takes as input a sequence of game states (e.g., the positions of pieces and the game board over recent steps) and processes them through an LSTM layer, which captures temporal dependencies between moves. Based on the hidden state of the last step, the model predicts the probabilities of actions (left, right, rotate, drop) through a fully connected layer. This way, the network considers the history of the game, not just the current state, and selects the most strategically optimal action. Initially, LSTM was considered as the main

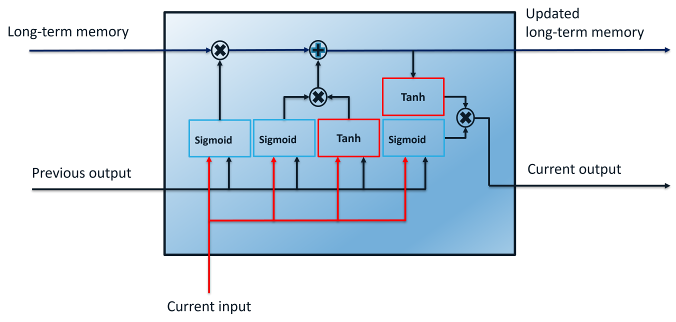

Figure 4: An LSTM network relates the input data window to outputs with layers. Instead of just one layer, LSTMs often have multiple layers. [5]

model to preserve the context of previous games, but since the data was generated by a heuristic bot, it was repetitive and did not give the expected results. It was kept only for experimental purposes.

5 CNN

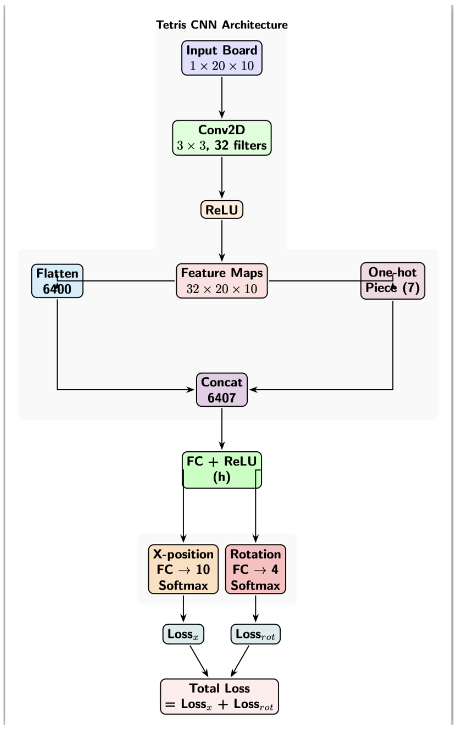
Tetris CNN Architecture
Input Board $1 \times 20 \times 10$
Conv2D 3×3, 32 filters
ReLU
Flatten 6400
Feature Maps $32 \times 20 \times 10$
One-hot Piece (7)
Concat 6407
FC + ReLU (h)
X-position FC $\rightarrow$ 10 Softmax
Rotation FC $\rightarrow$ 4 Softmax
Loss$_x$
Loss$_{rot}$
Total Loss = Loss$_x$ + Loss$_{rot}$

Figure 5: Tetris CNN Architecture

CNN model for Tetris takes the current game board and the piece type as input. The board is processed through convolutional layers to extract spatial features, which are then combined with a one-hot encoding of the piece type. The combined features pass through a fully connected layer and produce two outputs: horizontal movement probabilities and rotation probabilities. This allows the network to choose optimal actions based on both the board layout and the current piece. CNN performed well because it effectively captures both the spatial structure of the game board and the current piece type.

6 Tetris environment for DQN

Developing a training environment for Tetris presented a significant challenge. One of the most difficult aspects was designing and tuning the reward system. When the reward structure was too simple, the agent quickly reached a local optimum. Conversely, an overly complex or poorly balanced reward system could cause the agent to avoid exploration or even seek to lose quickly.

7 DQN

Deep Q-Network (DQN) combines Q-learning with deep neural networks and is effective in environments with large or continuous state spaces, like Tetris. Using a CNN as a feature extractor allows the network to efficiently understand spatial patterns on the board.

Training was challenging; initially, more complex variants like double DQN and dueling DQN were tried, but these increased training time without clear gains. Returning to a standard DQN with lookahead gave better results.

8 Results

Throughout this project, the main focus was on developing a DQN agent capable of learning to play Tetris autonomously. The classical DQN and its improved version, dueling DQN, demonstrated some progress. However, the results did not fully meet expectations due to limited time and computational resources. Each increase in model complexity required significantly more training time and resources.

To achieve better results within a shorter timeframe, a lookahead strategy was implemented. This approach allowed the agent to make decisions by evaluating future game states several steps ahead.

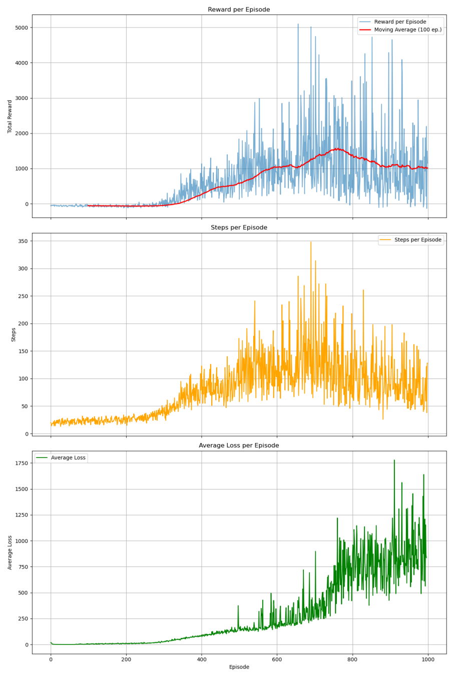

Figure 6: Tetris DQN agent training progress.

9 Possible improvements.

In the future, I plan to develop a hybrid model that combines the strengths of all three approaches used in this project. The idea is to build a more advanced DQN architecture based on CNNs and enhanced with LSTMs, enabling the agent not only to interpret the current game state but also to leverage temporal dependencies and remember previous actions.

References

[1] Scherrer, B. et al. Approximate Policy Iteration Schemes: A Comparison of Empirical and Theoretical Results. *Journal of Machine Learning Research*, 16:1449–1480, 2015.

[2] Algorta, S. and Simsek, O. The Game of Tetris in Machine Learning. *arXiv preprint arXiv:1905.01652*, 2019.

[3] Shailaja, M. et al. Playing Tetris with Reinforcement Learning. *International Journal for Research in Applied Science and Engineering Technology*, 2022.

[4] Lundgaard, N. and McKee, B. Reinforcement Learning and Neural Networks for Tetris. *Norwegian University of Science and Technology*, 2017.

[5] https://apmonitor.com/pds/index.php/Main/LongShort

[6] https://docs.pytorch.org/tutorials/