Latvijas Universitāte

Faculty of Computer Science

Course code and title: DatZ3169: Qualification Project

AI-Enhanced Tetris Optimal Strategy Learning and Gameplay Optimization

Created by: Reza Ghasempour(rg21003)

Supervisor: Ali Ghasempour, CyberWiseSpace CEO

Riga 2024

Abstract

This comprehensive documentation details the development and functionality of a multifaceted application comprising two integral components: a captivating Tetris game and an AI model trained using genetic learning to achieve exceptional proficiency in playing Tetris. The Tetris segment boasts a user-centric interface, blending classic gameplay mechanics with contemporary visual enhancements to deliver an immersive gaming experience. Through meticulous design and implementation, this segment encompasses diverse features, including block manipulation, scoring algorithms and user-interaction elements.

In parallel, the AI component intricately outlines the architecture and methodologies employed in leveraging genetic learning principles. It explicates the genetic algorithm framework, elucidating the genetic representation of gameplay strategies, fitness functions, mutation, crossover techniques, and evolutionary strategies instrumental in optimizing the AI's performance. The documentation meticulously narrates the AI's training journey, encompassing data acquisition, model evaluation, and iterative evolutionary processes, showcasing the AI's learning trajectory and progressive skill enhancement over training cycles.

This documentation acts as a comprehensive guide, furnishing detailed insights, code snippets, and technical specifications for both the Tetris game and the genetic learning-based AI model. By encapsulating intricate development details, it enables a deeper understanding, facilitates implementation, and encourages further advancements in both gaming experiences and AI-driven gameplay strategies.

Keywords: Tetris game, AI, genetic learning, gameplay mechanics, user interface, genetic algorithms, strategy optimization, training process, skill enhancement.

AI-Uzlabotais Tetris: optimāla stratēģijas apguve un spēles optimizācija

Anotācija

Šī dokumentācija detalizēti apraksta divdaļīgas lietotnes izstrādi un funkcionalitāti: pievilcīgu Tetris spēli un mākslīgā intelekta (AI) modeli, kas apmācīts, izmantojot genētiskās mācīšanās metodes, lai sasniegtu izcilu prasmi spēlēt Tetris. Tetris sadaļa piedāvā lietotājam draudzīgu interfeisu, kurā tiek apvienoti klasiskie spēles mehānismi ar mūsdienīgām vizuālajām uzlabojumiem, lai nodrošinātu iespaidīgu spēles pieredzi. Ar rūpīgu dizainu un īstenošanu šajā sadaļā ir iekļautas dažādas funkcijas, tostarp bloku manipulācija, rezultātu algoritmi un lietotāju mijiedarbības elementi.

Paralēli AI sadaļa detalizēti apraksta arhitektūru un metodoloģijas, kas izmantotas, lai izmantotu genētiskās mācīšanās principus. Tas izskaidro genētisko algoritmu struktūru, izklāsta spēles stratēģiju genētisko reprezentāciju, piemērotības funkcijas, mutāciju, krustpunktu tehnikas un evolucionārās stratēģijas, kas ir svarīgas AI izlīdzināšanā. Dokumentācija rūpīgi apraksta AI apmācības ceļojumu, iekļaujot datu ieguvi, modeļa novērtēšanu un iteratīvus evolucionārus procesus, demonstrējot AI mācīšanās trajektoriju un progresīvo prasmju uzlabošanos apmācības ciklu laikā.

Šī dokumentācija kalpo kā visaptverošs ceļvedis, sniedzot detalizētus ieskatu, kodu fragmentus un tehniskos specifikācijas gan Tetris spēlei, gan genētiskās mācīšanās balstītajam AI modelim. Ieguldot sarežģītus izstrādes datus, tā veicina dziļāku sapratni, atvieglo ieviešanu un veicina turpmāku progresu gan spēļu pieredzēs, gan AI vadītajās spēles stratēģijās.

Atslēgas vārdi: Tetris spēle, AI, genētiskā mācīšanās, spēles mehānisms, lietotāja interfeiss, genētiskie algoritmi, stratēģijas optimizācija, apmācības process, prasmju uzlabošana.

Table of Contents

List of Figures

Glossary

Acronym	**Definition**
HTML	Hyper Text Markup Language is the standard markup language for documents designed to be displayed in a web browser.
JSON	JavaScript Object Notation is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.
PY	Python is a high-level, general-purpose programming language.
JS	JavaScript is a programming language and core technology of the World Wide Web, alongside HTML and CSS.
AI	Artificial Intelligence makes it possible for machines to learn from experience, adjust to new inputs and perform human-like tasks.
CSS	Cascading Style Sheets is a style sheet language used for specifying the presentation and styling of a document written in a markup language such as HTML or XML.
CORS	Cross-Origin Resource Sharing is a mechanism for integrating applications.
API	Application Programming Interface is a way for two or more computer programs to communicate with each other. It is a type of software interface.
UI	User Interface is the point of human-computer interaction and communication in a device.

1. INTRODUCTION

1.1. Purpose

This application aims to entertain users with a captivating Tetris gaming experience while showcasing the potential of AI through genetic learning. It provides a platform to witness AI mastery in Tetris, fostering understanding of AI concepts and their application in gameplay strategies.

1.2. Scope

This application's scope involves developing a feature-rich Tetris game with a user-friendly interface and creating an AI model using genetic learning techniques specifically tailored to excel at playing Tetris. It includes designing gameplay elements for the Tetris game and implementing a genetic algorithm framework for the AI's learning process. The aim is to provide an engaging gaming experience while showcasing the AI's mastery in playing Tetris through genetic learning.

1.3. Overview

This document is structured into four comprehensive chapters. The first chapter provides an overarching view, detailing the overall concept and purpose of the application. The second chapter delves into the software's description, offering insights into its functionalities and components, focusing on both the Tetris game and the AI model developed using genetic learning.

Moving forward, the third chapter elaborates on the software's design, presenting a detailed breakdown of the architecture, algorithms, and methodologies employed in creating the Tetris game and implementing the genetic algorithm framework for AI training.

Finally, the fourth chapter focuses on testing, discussing the methodologies, approaches, and results obtained from testing the application's functionality, robustness, and performance. This comprehensive structure aims to provide a thorough understanding of the application's development, design, and evaluation processes.

2. OVERALL DESCRIPTION

This documentation outlines an application featuring a Tetris game and an AI model trained using genetic learning for Tetris gameplay. The Tetris game offers a basic yet engaging experience, while the AI, though functional, requires optimization. The documentation covers the purpose, design, functionalities, and ongoing optimization efforts for both the Tetris game and AI model, aiming to provide an understanding of their development and potential enhancements.

2.1. AS IS, Client, Business requirement

This documentation was commissioned by the University of Latvia to present an application comprising a Tetris game and an AI model trained using genetic learning for Tetris gameplay. The Tetris game, while functional, features limited functionalities. The AI model, although operational, currently lacks optimization in its code. The project primarily serves research purposes for the university, emphasizing academic exploration rather than immediate business application or commercial value.

2.2. Product vision

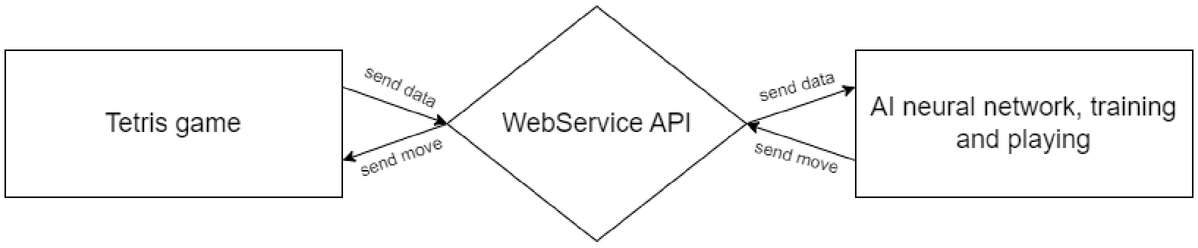
The ultimate vision for this project is to develop an AI system capable of emulating human-like gameplay in Tetris using neural network architecture without employing reinforcement learning techniques. The primary goal is to create an AI model that comprehends, strategizes, and plays Tetris with a behavioral resemblance to human gameplay patterns and decision-making processes. The emphasis lies in harnessing neural network capabilities to facilitate organic learning and gameplay mimicry, achieving a level of AI proficiency in Tetris that aligns closely with human intuition and strategy, all while eschewing reinforcement learning methodologies.

2.3. System users

Player: The primary user of the system is the human player, engaging with the Tetris game component. The player interacts directly with the game interface, manipulating falling blocks, strategizing movements, and aiming to achieve high scores. The player's role involves enjoying the gaming experience and utilizing Tetris mechanics to clear lines.

AI Model: The secondary user is the AI model developed for playing Tetris. This entity functions autonomously within the system, using genetic learning and neural network architecture to analyze gameplay patterns, make decisions on block placements, and optimize strategies. The AI model's purpose is to mimic human-like gameplay behaviors without reinforcement learning, continually learning and evolving its gameplay approach based on past experiences and genetic algorithm iterations.

2.4. System Structure Model



Pic. 1. System Structure Model

2.5. General Restrictions

Hardware Limitations: The software's performance may be affected by the hardware capabilities of the device running the application. Lower processing power or limited memory might impact the gaming experience or AI performance.

Optimization Constraints: Due to the ongoing nature of code optimization for the AI model, the software might encounter performance issues or suboptimal gameplay strategies, impacting the AI's efficiency and responsiveness.

Non-commercial Use: As commissioned by the University of Latvia for research purposes, the software is intended for academic exploration and not for commercial or profit-driven applications.

Limited Features: The Tetris game component might lack advanced features commonly found in commercial gaming applications due to its focus on research rather than extensive gameplay functionalities.

No Reinforcement Learning: The AI model's design intentionally excludes reinforcement learning techniques, which might limit its learning speed or potential in comparison to models utilizing such methodologies.

Compatibility Concerns: Compatibility issues may arise when running the software on certain operating systems or devices due to specific software dependencies or requirements.

2.6. Assumptions and Dependencies

Assumption of Stable Environment: The software assumes a stable and consistent operating environment for optimal performance, free from abrupt interruptions or system inconsistencies.

Dependence on Software Libraries: The application relies on specific software libraries and frameworks for its functionality, assuming their availability, compatibility, and proper functioning.

Assumption of User Familiarity: It assumes users possess a basic understanding of Tetris gameplay mechanics and AI concepts, facilitating a smoother interaction with the application.

Dependence on Hardware Capabilities: The software's performance is dependent on the capabilities of the hardware running the application, with better hardware potentially enhancing the gaming experience and AI efficiency.

Assumption of Data Integrity: It assumes the integrity and accuracy of the data used for training the AI model, considering it to be representative of Tetris gameplay scenarios.

Dependency on Development Tools: The software development process depends on specific programming languages, development environments, and tools for implementation and testing. Changes or limitations in these tools might impact the software development workflow.

3. SOFTWARE REQUIREMENTS SPECIFICATIONS

3.1. Introduction

For this section we will go through each file and describe the functionality within each file. This project uses Pytorch and neural networks to train the AI in a similar Tetris game written in python and after we get the inputs from the saved model and send them to the Tetris javascript for running.

3.2. Functional Requirements

3.2.1. Overview

There will be 11 files that we will look through, the files are:

1. **config.json:** this file is a Json file that contains the general variables we will use in our python files
2. **loadconfig.py:** this file is a python file that enables us to load Json file into python files.
3. **index.html:** this file contains the main html file for Tetris game which runs the JavaScript files.
4. **style.css:** this file contains the style and design for Tetris game which in JavaScript.
5. **teris.py:** this file contains a Tetris game similar logically to JavaScript version, but it doesn't have graphical interface and it is used for training the AI.
6. **tetris_ai.py:** this file contains the neural network, layers and weights used by our AI**.**
7. **train.py:** this file contains different adjustment functions such as mutation, population choosing etc., and it connects to tetris.py and tetris_ai.py to train the AI and save the best MODEL.
8. **MODEL:** this file is our best saved agent after training which we will use is our tetris.js.
9. **tetris.js:** this file contains a Tetris game made is JavaScript canvas with user interface and it is where we will inputs go for our MODEL to play the Tetris.
10. **web_service.py:** this file gets all the necessary information for AI from tetris_ai.py to play the game and sends that info to test.js.
11. **test.js:** this file handles requests and answers between webservice.py and tetris.js.

3.2.2. config.json

1. **Configuration Loading:**

Requirement: Upon system startup, the software shall read and parse the config.json file to extract configuration parameters.

Details: The system should handle the retrieval of the config.json file, parse its contents, and store the configuration parameters for later use.

2. **Game Board Dimensions and Random Seed Initialization:**

Requirement: The system shall utilize the rows and columns values from the configuration to determine the dimensions of the game board. The seed value shall initialize the random number generator for consistent pseudo-random behavior.

Details: The software should interpret the rows and columns values to set the size of the game board. Additionally, the seed value should initialize the random number generator to maintain reproducibility.

3. **AI Model Configuration:**

Requirement: The system shall utilize the specified parameters (population_size, mutation_strength, mutation_rate, number_of_generation, games_per_agent, top_percent, max_ticks, idiot_percent_chosen) to configure the genetic algorithm of the AI model.

Details: The software should apply the provided parameters from the config.json file to set up the genetic algorithm used by the AI model.

4. **Adaptive Mutation Rates:**

Requirement: The system shall adjust the mutation rate based on generation numbers specified in the mutation_rates object.

Details: The software should dynamically modify the mutation rate according to the defined generation numbers and corresponding mutation rates in the mutation_rates object within the configuration file.

5. **Error Handling for Invalid Configuration:**

Requirement: In case of an invalid or missing config.json file or invalid parameters within it, the system shall handle errors in loadconfig.py.

Details: The software should provide appropriate error messages if the config.json file is missing or contains invalid parameters.

3.2.3. loadconfig.py

1. **Configuration Loading:**

Requirement: Upon system initialization, the software should systematically attempt to access and load the configuration data present in the config.json file.

Details: The system shall utilize an explicit loading mechanism at startup to retrieve and read the configuration details stored within the designated config.json file.

2. **Default Configuration Management:**

Requirement: In scenarios where the config.json file is inaccessible or contains erroneous data, the system shall employ a predetermined default configuration set.

Details: If the system encounters difficulties in accessing or interpreting the content of the config.json file (such as file absence, incorrect format, or missing fields), it will autonomously resort to a predefined set of default configuration values, ensuring continued system functionality.

3. **Comprehensive Configuration Extraction:**

Requirement: The system shall demonstrate proficiency in accurately extracting various configuration parameters from the JSON-formatted data loaded from the config.json file.

Details: The software shall intricately decipher the JSON structure, specifically retrieving and assigning crucial configuration elements, including but not limited to:

Number of rows and columns for the game board

Seed value used for initializing the random number generator

Population size for the AI model

Parameters indicating mutation strength and rate

Total number of generations to iterate through

Count of games played per agent

Percentage of top-performing agents to retain

Maximum allowable ticks within the AI's decision-making process

Percentage of "idiot" agents chosen for comparison

Mutation rates for each generation to optimize genetic algorithm progression.

4. **Effective Error Handling Mechanism:**

Requirement: If any complications arise during the loading or parsing process of the config.json file, the system shall promptly notify the user by printing a descriptive warning message.

Details: In the event of loading or parsing errors (e.g., file corruption, syntax issues, missing or invalid data), the software will issue a clear and informative warning message to alert the user. Subsequently, the system will seamlessly revert to utilizing default configuration values to prevent any disruption to its operations.

3.2.4. Index.html

1. **Game Statistics Display:**

Requirement: The interface should display the game statistics, including the current points and timer.

Details: The HTML structure should contain elements (<div> or <span>) labeled as "Points" and "Time," with corresponding values that update dynamically during gameplay.

2. **Button Mapping Information:**

Requirement: The page should feature a section that delineates the keyboard controls for playing the game.

Details: An HTML division (<div>) titled "Button Mapping" must encompass an unordered list (<ul>) containing individual list items (<li>) specifying the actions associated with respective keys (e.g., Spacebar, Right Arrow/D, Left Arrow/A, Down Arrow/S, R).

3. **Game Canvas Section:**

Requirement: The HTML file should contain a designated section for the game canvas and any supplementary canvases.

Details: Inside the canvasContainer div, there should be a primary canvas (tetrisCanvas) and an extra canvas (extraCanvas) designed to facilitate the game's visual elements. The tetrisCanvas should serve as the primary game area, while the extraCanvas is meant for additional game components.

4. **Game Over Overlay Functionality:**

Requirement: The system should display an overlay with pertinent messages and a reset button upon the completion of the game.

Details: Upon game completion (game over state), the HTML file should reveal an overlay containing a message (e.g., "Game Over") and a reset button (resetButton). Clicking the reset button should invoke the game reset functionality (resetGame()).

5. **External Script Inclusion:**

Requirement: The HTML file should link external JavaScript libraries and files required for game functionality.

Details: The HTML file must reference external resources such as the jQuery library (jquery-3.6.0.min.js), an external JavaScript file (test.js), and the primary Tetris game script (tetris.js) for proper game execution and functionality.

3.2.5. style.css

1. **Global Styles for Body:**

Requirement: The CSS file should apply global styles to the body element.

Details: The body element's styles should set margin to zero, prevent scrolling, use flexbox to arrange elements vertically (flex-direction: column), center align items, utilize the entire viewport height (height: 100vh), and set a background color (background-color: #f0f0f0).

2. **Canvas Styling:**

Requirement: The CSS file should style all canvas elements.

Details: The styles for canvas elements should include a 2px solid border with a light gray color (border: 2px solid #ddd).

3. **Extra Canvas Styles:**

Requirement: The CSS file should style the extra canvas specifically.

Details: The extra canvas should have a top margin of 20 pixels (margin-top: 20px).

4. **Game Over Overlay Styling:**

Requirement: The CSS file should style the game over overlay and its components.

Details: The overlay should cover the entire viewport (position: absolute; top: 0; left: 0; width: 100%; height: 100%;) with a semi-transparent white background (background-color: rgba(255, 255, 255, 0.8)). Initially, it should be hidden (display: none).

5. **Button Mapping Section Styling:**

Requirement: The CSS file should style the button mapping section.

Details: The button mapping section should be positioned to the left (left: 20px) and vertically centered (transform: translateY(-50%)). It should have padding, a light gray background, border radius, and a subtle box shadow for styling.

6. **Stats Section Styling:**

Requirement: The CSS file should style the stats section.

Details: The stats section should be positioned to the top left (position: absolute; top: 20px; left: 20px;), use flexbox for column display, and align items to the start. It should have a color of dark gray (color: #333) and a font size of 1.5rem.

7. **Styling for Titles, Lists, and Texts:**

Requirement: The CSS file should style various textual elements.

Details: Styles should be defined for headings (h2), lists (ul), list items (li), and specific text displays, setting appropriate font sizes, margins, and colors for readability and consistency.

3.2.6. tetris.py

1. **Game Initialization:**

Requirement: The system must initialize a grid-based play area for AI-driven Tetris gameplay.

Details: It should allow configurable settings to define the grid size (rows and columns) for the AI's game board.

2. **Tetrimino Shapes:**

Requirement: The system must include a variety of Tetrimino shapes like L, T, Square, Ugly L, Line, etc., for AI-driven gameplay.

Details: Each Tetrimino shape should have unique patterns and rotations to engage the AI player with diverse block configurations.

3. **Piece Movement:**

Requirement: The system must enable the AI to maneuver Tetrimino pieces by moving them left, right, or rotating them clockwise.

Details: Tetrimino pieces should be programmatically movable or rotatable by the AI until they encounter obstacles or reach the boundaries of the game board.

4. **Collision Detection:**

Requirement: The system must incorporate collision detection algorithms to prevent Tetrimino pieces controlled by the AI from surpassing grid boundaries or overlapping existing blocks.

Details: Upon collision, the Tetrimino pieces must be correctly positioned and locked within the grid to ensure accurate gameplay for the AI.

5. **Game Mechanics:**

Requirement: The system must implement a tick mechanism to control the descent pace of Tetrimino pieces for AI-controlled gameplay.

Details: Tetrimino pieces should descend systematically, row by row, simulating the AI's decision-making for optimal piece placement.

6. **Line Clearing:**

Requirement: The system should manage the clearing of completed lines on the grid during AI-driven gameplay.

Details: Upon line completion, the system should clear the lines, allowing subsequent blocks above to cascade down, maintaining game integrity for the AI player.

7. **Scoring System:**

Requirement: The system may include a scoring system based on various factors (e.g., completed lines, time intervals, etc.) for AI performance evaluation.

Details: If applicable, the scoring mechanism could evaluate and record AI performance metrics during and at the end of each game session, reflecting AI gameplay proficiency.

8. **Game Over Condition:**

Requirement: The system should define game-ending conditions for the AI, triggered when Tetriminos cannot enter the designated play area due to collisions or other criteria.

Details: Specific conditions will signal the conclusion of the game session, indicating situations where the AI can no longer place Tetriminos within the playable space.

9. **AI Decision-Making:**

Requirement: The system should simulate intelligent decision-making processes for the AI to manage Tetrimino movements (left, right, rotate, do nothing).

Details: The AI should employ algorithms or logic to strategically maneuver Tetrimino pieces, considering optimal placement and rotation based on the current game state and available Tetriminos.

10. **Game State Representation:**

Requirement: The system must visually represent the game state, showcasing the current grid layout with Tetrimino pieces, empty spaces, and completed lines for AI-driven gameplay.

Details: Real-time updates of the game state should reflect the AI's decisions and actions, visually presenting the evolving game board and Tetrimino configurations to observers or for analysis purposes.

3.2.7. tetris_ai.py

1. **Neural Network Architecture:**

Requirement: The system must present a neural network model composed of four linear layers.

Details: Each layer should consist of specific input and output dimensions designed for the network's structure.

2. **Model Initialization:**

Requirement: The system must initialize the neural network model with predetermined input and output dimensions for each layer.

Details: The model's architecture must be set up with precise input and output dimensions aligning with the Tetris AI requirements.

3. **Gradient Calculation Control:**

Requirement: The system should disable gradient calculation for the neural network layers.

Details: Preventing gradient calculation is essential to maintain the model's architecture without allowing further learning or adjustment.

4. **Layer Bias Initialization:**

Requirement: The system must initialize the bias of each neural network layer with zeros.

Details: Setting bias values to zeros ensures a neutral starting point for network computations.

5. **Layer Weight Initialization:**

Requirement: The system should initialize the weights of each neural network layer with uniform random values ranging from -0.5 to 0.5.

Details: Utilizing uniform random values within the specified range for weight initialization supports the network's variability and adaptability.

6. **Weight Mutation Method:**

Requirement: The system must offer a method to mutate the weights of the neural network based on a specified mutation rate.

Details: This functionality enables controlled adjustments to the network's weights, affecting its learning and adaptability.

7. **Activation Function - Sigmoid:**

Requirement: The system must apply a sigmoid activation function to the output of the first three linear layers in the neural network model.

Details: Sigmoid activation facilitates non-linearity in the network's initial layers, enhancing its capacity to capture complex relationships in data.

8. **Activation Function - Softmax:**

Requirement: The system should apply a softmax activation function to the output of the fourth linear layer along the second dimension.

Details: Softmax activation normalizes the output, providing probabilities across the second dimension, suitable for classification or probability-based output.

9. **Model Output Retrieval:**

Requirement: The system shall return the output of the neural network model when provided with an input.

Details: Upon receiving input data, the system should produce corresponding output based on the processed information through the neural network.

10. **Model Instance Creation:**

Requirement: The system must allow the creation of an instance of the neural network model.

Details: This functionality enables the instantiation of the Tetris AI neural network model, facilitating its utilization within the Tetris AI system.

3.2.8. train.py

1. **Module and Configuration Import:**

Requirement: The system must import necessary modules and configuration parameters.

Details: Importing essential modules and configuration parameters is crucial for the proper execution of the training script.

2. **Agent Mutation Function:**

Requirement: The system shall define a mutation function to generate a new agent by mutating a random parameter of an existing agent.

Details: This function facilitates the creation of new agents by altering specific parameters of existing agents through mutation processes.

3. **Value Normalization Function:**

Requirement: The system shall define a function to normalize a value within a given range.

Details: Normalization functionality is necessary to scale and map values to a specified range, ensuring consistency in data processing.

4. **Action Probability Determination Function:**

Requirement: The system shall define a function to determine the action with the highest probability.

Details: This function aids in decision-making by selecting actions based on their respective probabilities, optimizing the agent's moves.

5. **Agent Population Representation Class:**

Requirement: The system shall define a class to represent a population of agents.

Details: The class structure enables the management and handling of multiple agents within the evolutionary simulation.

6. **Evolutionary Simulation Representation Class:**

Requirement: The system shall define a class to represent the evolutionary simulation.

Details: This class encapsulates the functionalities and procedures related to running the evolutionary simulation of the Tetris AI.

7. **Initialization of Simulation Components:**

Requirement: The system shall initialize components, including a population of agents, scores, generation number, score history, and mutation parameters.

Details: Initializing these components forms the groundwork necessary for conducting the evolutionary training process.

8. **Mutation Parameter Adjustment Method:**

Requirement: The system shall provide a method to dynamically adjust mutation parameters based on recent performance.

Details: This method enables adaptive modification of mutation parameters, optimizing the evolutionary process based on current performance trends.

9. **Simulation Generation Execution Method:**

Requirement: The system shall provide a method to execute a generation of the evolutionary simulation.

Details: This method triggers the execution of a single generation in the evolutionary training process.

10. **Agent Selection Method for Next Generation:**

Requirement: The system shall provide a method to select agents for the subsequent generation based on defined criteria.

Details: This method determines the selection process for choosing agents to propagate to the next generation, influencing the evolution of the AI.

11. **Best Agent Retrieval Method:**

Requirement: The system shall provide a method to retrieve the best agent (with the highest score) from the population.

Details: This method identifies and retrieves the agent with the most optimal performance within the current population.

12. New Generation Creation Method:

Requirement: The system shall provide a method to generate a new population of agents for the next evolutionary iteration.

Details: This method generates a new set of agents based on the selection criteria, forming the subsequent generation.

13. Random Seed Initialization:

Requirement: The system shall set the random seed to ensure reproducibility of results.

Details: Setting the random seed guarantees that subsequent runs of the training script will yield the same results, aiding in reproducibility.

14. Evolutionary Simulation Initialization:

Requirement: The system shall initialize the evolutionary simulation components and configurations.

Details: Initialization includes setting up parameters and structures required for the evolutionary process.

15. Best Agent Retrieval from Initial Population:

Requirement: The system shall retrieve the best agent and its score from the initial population.

Details: This step ensures the evaluation of the initial population's performance and identification of the best agent.

16. Simulation Execution for Specified Generations:

Requirement: The system shall execute the simulation for the specified number of generations.

Details: Running the simulation over a predefined number of generations allows the evolution of the AI over multiple iterations.

17. Generation Number and Best Score Display:

Requirement: The system shall print the current generation number and the best score achieved.

Details: Displaying generation progress and the best score provides essential information during the simulation run.

18. Periodic State Saving of Best Agent:

Requirement: The system shall save the state of the best agent every 50 generations.

Details: Periodic saving preserves the state of the most successful agent for future reference or continued training.

19. New Generation Creation Execution:

Requirement: The system shall create a new generation of agents for the evolutionary simulation.

Details: This step initiates the creation of a new set of agents to progress to the subsequent evolutionary stage.

20. Score Reset for New Generation:

Requirement: The system shall reset scores for the new generation of agents.

Details: Resetting scores ensures a fresh start for the newly created generation, avoiding influence from prior iterations.

21. Final State Saving of Best Agent:

Requirement: The system shall save the state of the best agent after completing all generations.

Details: Saving the final state of the best agent allows retention of the most successful AI configuration achieved through the entire evolutionary process.

3.2.9. MODEL

1. **Model Saving Method:**

Requirement: The system shall provide a method to save the trained AI model to a file.

Details: This functionality allows the preservation of the trained AI model's configuration and parameters for future usage.

2. **Saved Model File Contents:**

Requirement: The saved model file shall contain all parameters of the trained model, encompassing weights and biases of each layer.

Details: Ensuring the comprehensive storage of model parameters guarantees the model's integrity and accurate representation when loaded.

3. **File Format Compatibility:**

Requirement: The system shall ensure that the saved model file is in a format compatible with the application for seamless reloading.

Details: Compatibility with the application ensures that the saved model file can be effortlessly reloaded without any issues.

4. **File Security Measures:**

Requirement: The system shall ensure that the saved model file is in a secure, non-editable format to prevent unauthorized modifications.

Details: Implementing security measures safeguards the integrity and authenticity of the trained model, preventing unauthorized alterations.

5. **Model Loading Method:**

Requirement: The system shall provide a method to load the saved model file back into the application.

Details: This functionality enables the retrieval and integration of the previously saved AI model into the application for further usage.

6. **Retained Learned Knowledge:**

Requirement: The loaded model shall retain all acquired knowledge and should not necessitate additional training.

Details: Retaining learned knowledge ensures that the model, once loaded, retains its previously acquired intelligence and capabilities.

7. **Consistent Performance Post-Loading:**

Requirement: The system shall ensure that the loaded model performs in the same manner as it did at the time of saving.

Details: Ensuring consistency in performance post-loading guarantees that the model operates identically to its state at the time of saving, maintaining reliability and accuracy.

3.2.10. tetris.js

1. **AI Interaction Handling:**

Requirement: The system shall establish a communication interface with an external AI service to determine the best move for the Tetris game.

Details: The system shall send the current game board state to the AI service, requesting the next move.

2. **Interpretation of AI Responses:**

Requirement: The system shall interpret responses received from the AI service as instructions for Tetris gameplay actions, including moving left, moving right, rotating, or maintaining the current position.

Details: Decoded responses will directly influence the movement and actions of the Tetris pieces within the game.

3. **Tetris Piece Generation and Control:**

Requirement: Upon the start of a new round, the system shall initialize a Tetris piece at the top of the game board.

Details: Each round begins by placing a new Tetris piece at the top for user or AI manipulation.

4. **Tetris Piece Movement:**

Requirement: The system shall facilitate the downward movement of the active Tetris piece by one row per game tick.

Details: Tetris pieces shall descend continuously, allowing for player or AI-controlled movements.

5. **Piece Manipulation Options:**

Requirement: The system shall enable the active Tetris piece to perform left, right, or 90-degree rotation actions.

Details: These actions are essential for both user and AI interaction to control the Tetris piece.

6. **Piece Locking and Line Clearing:**

Requirement: The system shall lock the active Tetris piece in place when it reaches the bottom or collides with existing blocks.

Details: Locking ensures the placement of the Tetris piece and triggers line clearance for completed rows on the game board.

7. **Game Termination Conditions:**

Requirement: The system shall conclude the game session when a new Tetris piece cannot be accommodated at the top of the game board due to blockage.

Details: Game termination signifies the completion of the current session, prompting the final score display.

8. **Interactive Web-Based Display:**

Requirement: The system shall render the game board, active Tetris piece, and score on a web-based user interface.

Details: This display will allow users to visualize and interact with the game elements during gameplay, updating after each move or rotation.

3.2.11. web_service.py

1. **Server Initialization:**

Requirement: The system shall initialize a Flask application server upon system startup.

Details: This server is the backbone for handling AI requests and responses.

2. **Cross-Origin Resource Sharing (CORS):**

Requirement: The system shall enable CORS for all routes within the Flask application server.

Details: CORS activation will facilitate communication between the server and external entities.

3. **AI Model Initialization:**

Requirement: The system shall initialize an instance of the Tetris AI model.

Details: This model acts as the AI decision-making component for the Tetris game.

4. **Loading Pre-trained Weights:**

Requirement: The system shall load pre-trained weights into the Tetris AI model from a designated file named 'MODEL'.

Details: These weights form the basis of the AI model's decision-making process.

5. **Game State Processing:**

Requirement: The system shall process incoming POST requests targeted at the '/tetris_ai' endpoint.

Details: Extracting game state data from the JSON payload allows the system to prepare data for AI model input.

6. **Data Conversion for AI Model:**

Requirement: The system shall convert the extracted game state data into a suitable format for the Tetris AI model.

Details: This conversion prepares the game state data for input into the AI model for decision making.

7. **AI Command Interpretation:**

Requirement: The system shall pass the processed game state data to the Tetris AI model for decision making.

Details: The AI model output will be interpreted as a Tetris move command for gameplay.

8. **Output Representation:**

Requirement: The system shall convert the AI model's output into a string representation of the determined move command.

Details: This representation ensures a clear and communicative response to the POST request.

9. **Response to Requests:**

Requirement: The system shall return the string representation of the move command as the response to incoming POST requests at '/tetris_ai'.

Details: The response will contain the Tetris move command determined by the AI model.

10. **Flask Server Run Configuration:**

Requirement: The system shall execute the Flask application server in debug mode when the script is run as the main program.

Details: Debug mode execution facilitates development and troubleshooting during the script's primary execution.

3.2.12. test.js

1. **Variable Initialization:**

Requirement: The system shall initialize a variable named result_v with a default value of "Default value".

Details: This initialization sets up a placeholder for storing results or error messages.

2. **Function Definition:**

Requirement: The system shall define a function named get_move(data) that accepts a single argument data.

Details: This function is responsible for handling AJAX requests and processing responses.

3. **Synchronous AJAX Request:**

Requirement: The get_move(data) function shall make a synchronous AJAX POST request to the URL "http://localhost:5000/tetris_ai".

Details: The synchronous request is designed to communicate with a specific endpoint for Tetris AI functionality.

4. **Request Payload and Content Type:**

Requirement: The AJAX request sent by get_move(data) shall include the data argument as the request payload with a content type set to "application/json".

Details: The payload contains data necessary for Tetris AI decision making, formatted as JSON for compatibility.

5. **Response Handling:**

Requirement: Upon a successful AJAX request, the get_move(data) function shall store the received response in the result_v variable.

Details: Storing the response allows subsequent actions or processes to access the received data.

6. **Error Handling:**

Requirement: In the event of an error during the AJAX request, the get_move(data) function shall store the error message in the result_v variable.

Details: Capturing and storing error messages ensures visibility and handling of issues during communication with the Tetris AI endpoint.

7. **Return Value:**

Requirement: The get_move(data) function shall return the value stored in the result_v variable.

Details: The returned value may contain the response received from the Tetris AI endpoint or an error message, providing an output for subsequent use or display.

3.3. Non-Functional Requirements

3.3.1. Performance

Response Time: The system shall respond to AI move commands within 100 milliseconds to maintain fluid gameplay.

Scalability: The system shall efficiently handle a minimum of 1 game without performance degradation.

3.3.2. Reliability

Availability: The system shall maintain an uptime of 99.9%, ensuring constant availability for user interactions.

Error Handling: There are fail safe implemented in the system in case user interrupt webservice the game with continue as usual but with a slight performance drop.

3.3.3. Usability

User Interface Clarity: The system shall feature an intuitive and visually appealing interface, including clear instructions for seamless user interaction.

Game State Display: The system shall present the game state components (board, current piece, score) prominently and comprehensively to enhance user understanding.

3.3.4. Security

Data Integrity: The system shall not provide encryption between the data moved between webservice and Tetris game so there is internal vulnerability in case access to computer is compromised.

Error Message Security: The system shall avoid exposing sensitive information in error messages or logs to prevent potential security breaches.

3.3.5. Maintainability

Code Documentation: The system shall have well-documented and structured code, enabling easier understanding and maintenance for developers.

Modifiability: The system architecture shall allow for seamless updates or enhancements, facilitating the integration of new features or improvements to the AI model.

3.3.6. Portability

Browser Compatibility: The system shall function uniformly across major web browsers (Chrome, Firefox, Safari, Edge) without discrepancies in performance or functionality.

Device Adaptability: The system shall adapt responsively to one device, ensuring proper functionality and optimal user experience on the desktop platform.

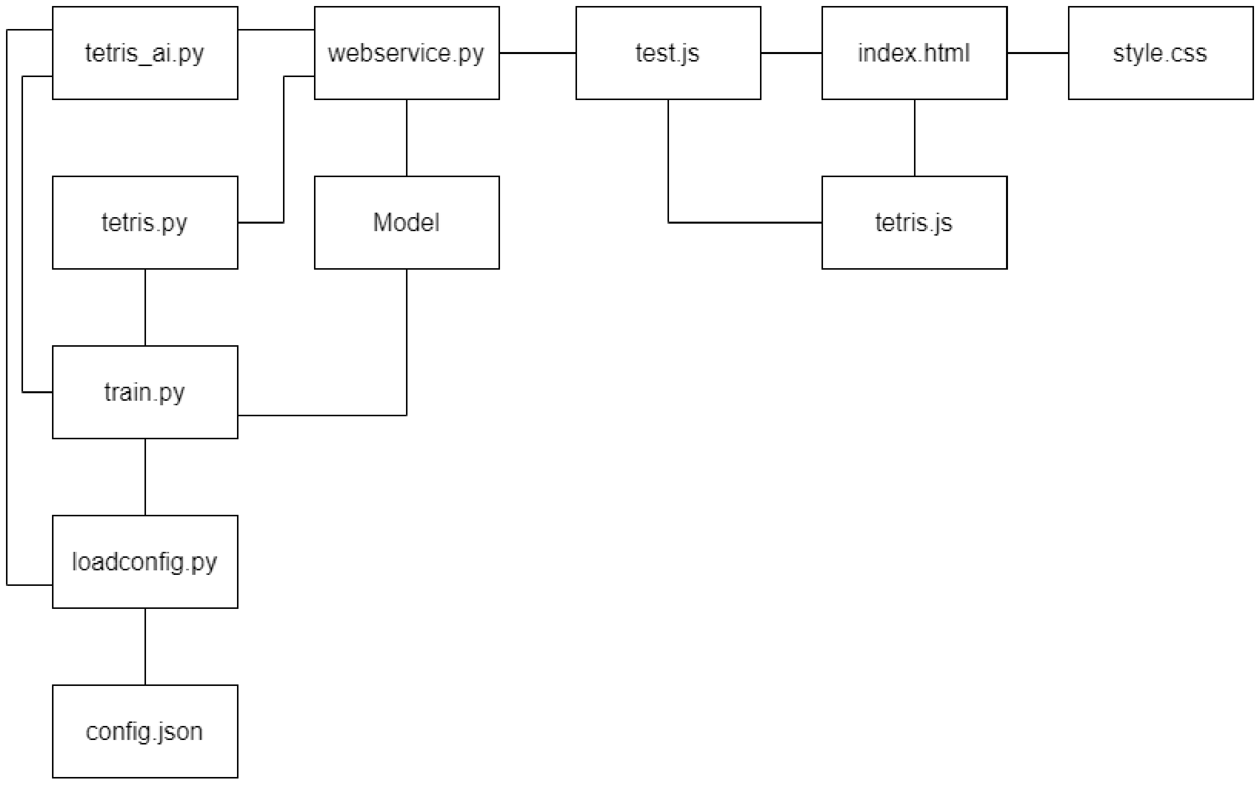
4. SOFTWARE DESIGN SPECIFICATIONS

4.1. System Architecture and Interactions

Overview

This section of this software design specification document aims to provide a comprehensive understanding of how the various components of the system function and interact with each other. This section delves into the details of the system's internal structure, clarifying the relationships and dependencies among different modules, classes, functions, and components within the software. By giving insight about the system architecture and illustrating the flow of data and control, this section offers an insightful overview that aids in comprehending the software's operational mechanisms and facilitates further comprehension of this document. This section is divide into two sections one the AI training and methods used, the second section is a brief explanation about the JavaScript Tetris.

4.1.1. File Structure Model



Pic. 2. File Structure Model

4.1.2. Tetris AI

AI Neural Network

The AI of the Tetris game is based on a neural network model implemented using PyTorch. The neural network is a type of machine learning model that is inspired by the human brain. It consists of interconnected layers of nodes, or "neurons", which can learn to recognize patterns in data.

The `Net` class in the code defines the architecture of our neural network. It has four layers, each of which is a linear (or fully connected) layer. This means that each neuron in a layer is connected to all neurons in the previous layer.

The input to the network is the current state of the Tetris game, which includes the board configuration, the position of the current piece, and the shape and rotation of the current piece. The output of the network is a decision on what action to take (move left, move right, rotate, or stay in place).

The `forward` method defines how an input is transformed into an output as it passes through the network. Each layer applies a linear transformation (multiplication by a weight matrix and addition of a bias vector) to its input, followed by an activation function. The activation function introduces non-linearity into the model, allowing it to learn more complex patterns.

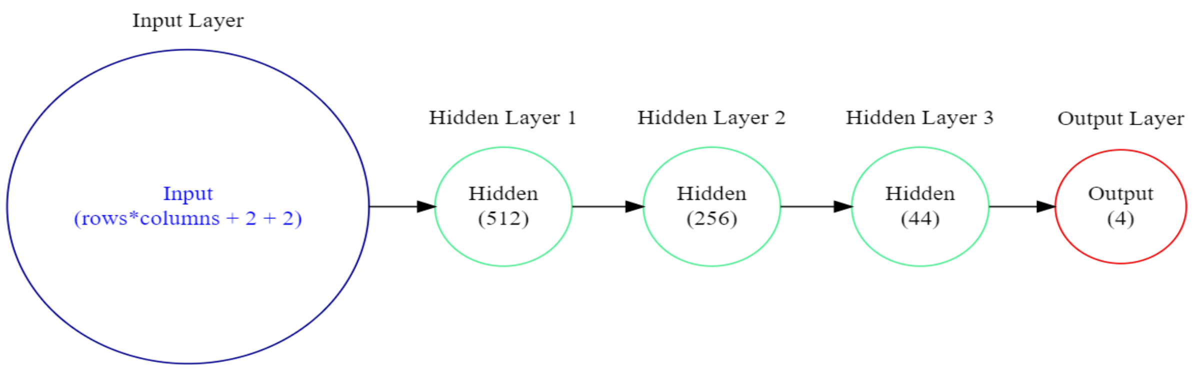
The first three layers use the sigmoid activation function, which squashes the input into a range between 0 and 1. This can be interpreted as the probability of a neuron "firing" or activating.

The final layer uses the softmax activation function, which also squashes its input into a range between 0 and 1, but in such a way that the outputs sum to 1. This makes it suitable for use in a multi-class classification problem, where you want to assign probabilities to different classes (in this case, the different actions the AI can take).

The `mutate` method allows for the introduction of random variations into the weights of the network. This is a form of genetic algorithm, where the idea is to mimic the process of natural selection in order to find the best model.

The weights of the network are initialized with uniform random values between -0.5 and 0.5, and the biases are initialized to zero. The weights and biases are the parameters of the model that are learned during training.

In the `__main__` block, an instance of the `Net` class is created. This is our AI model, ready to be trained and used to play Tetris.



Pic. 3. Simplified Version of Neural Network

Training AI

AI model is being trained using a game of Tetris implemented in Python as a training environment. here how it works:

1. **Game Initialization**: For each agent in the population, a new game of Tetris is initialized. The game is implemented in Python and the game state is represented as a 2D grid.
2. **Game Play**: The agent plays the game for a certain number of steps (ticks). At each step, the current game state is passed through the agent's neural network, which outputs a set of action probabilities. The action with the highest probability is chosen and performed in the game.
3. **Game End**: The game ends when the tick function returns False, which happens when the Tetris grid is filled up such that no new piece can be added.
4. **Score Calculation**: After the game ends, the agent's score is calculated based on the game's scoring rules. This score is used to evaluate the performance of the agent.
5. **Multiple Games**: Each agent plays multiple games of Tetris, and the agent's final score is the mean score over all games it played.

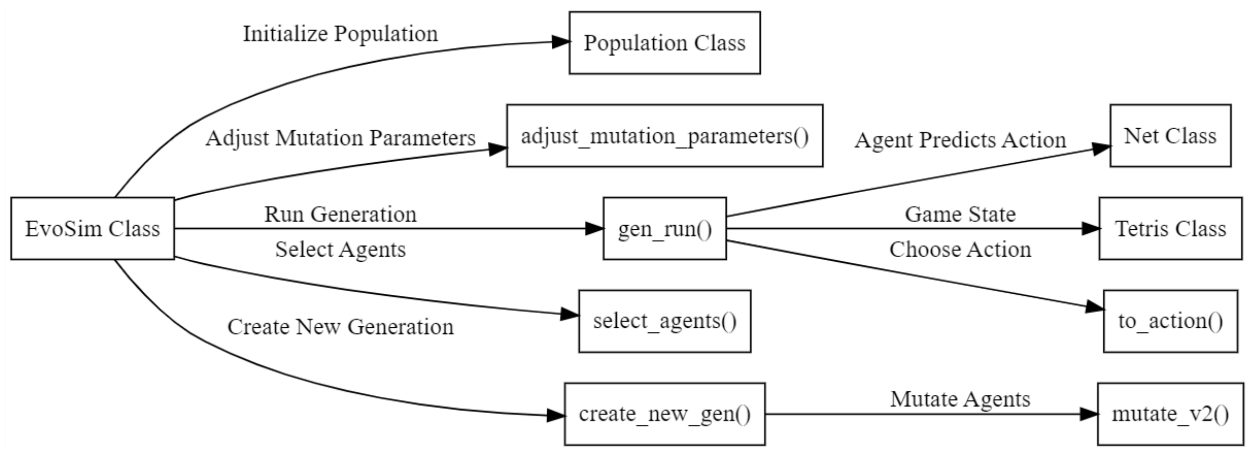
This process is repeated for each agent in the population and for each generation of the evolutionary algorithm. The game of Tetris provides a complex and dynamic environment for training the AI, as it involves strategic decision-making and planning ahead.

Below is the methodologies used for making sure to announce randomness to AI movement to make sure the AI doesn’t get stuck:

1. **Imports**: The necessary modules and configuration parameters are imported.
2. **Mutation Function**: `mutate_v2` is a function that creates a new agent by mutating a given agent. It randomly selects a parameter of the agent's neural network and adds noise to it. The amount of noise is determined by the mutation rate and strength.
3. **Normalization and Action Selection Functions**: `normalize` is a function that normalizes a value within a given range. `to_action` is a function that determines the action with the highest probability.

4. **Population Class**: This class represents a population of agents. It initializes a list of `Net` instances, where `Net` is the neural network that represents an agent.
5. **EvoSim Class**: This class represents the evolutionary simulation. It initializes a population of agents, scores, generation number, score history, and mutation parameters. It has several methods:
    - `*adjust_mutation_parameters*`: Adjusts mutation parameters based on recent performance.
    - `*gen_run*`: Runs a generation of the simulation. Each agent plays a number of games of Tetris, and their scores are calculated.
    - `*select_agents*`: Selects agents for the next generation based on their scores.
    - `*get_best*`: Returns the best agent (the one with the highest score).
    - `*create_new_gen*`: Creates a new generation of agents by mutating the selected agents.
6. **Main Execution**: If the script is being run directly, it sets the random seed for reproducibility, initializes the evolutionary simulation, and runs it for a specified number of generations. Every 50 generations, it saves the state of the best agent. After all generations have been run, it saves the state of the best agent.

This script is a basic implementation of a genetic algorithm for training a population of agents to play Tetris. The agents are neural networks that take the game state as input and output action probabilities. The agents are evolved over generations by mutation and selection based on their performance in the game.



Pic. 4. Data Flow Representation in train.py

Tetris Game (python)

The provided code is a Python class definition for a Tetris game. The class `Tetris` has several methods that handle the game logic, including moving and rotating pieces, checking for collisions, clearing completed lines, and calculating the score.

The `__init__` method is the constructor for the `Tetris` class. It initializes the game state, including the game grid, the current piece's position, shape, and rotation, the game over flag, the count of completed lines and ticks, and the previous action.

The `from_json` static method creates a new `Tetris` game from JSON data. It reads the game state from the JSON data and returns a new `Tetris` object with that state.

The `to_numpy` method converts the game state to a numpy array. It normalizes the game state data and concatenates it into a single numpy array.

The class sends data by returning it from methods. For example, the `to_numpy` method returns the game state as a numpy array. The `calculate_score` method returns the current score. The `tick` method returns a boolean indicating whether the game is over.

The `count_holes` method counts the number of holes in the game grid. A hole is defined as an empty space that is below a filled block in the same column.

The `calculate_pillar_height_difference` method calculates the difference in height between adjacent pillars. A pillar is defined as a column of filled blocks in the game grid.

The `tutorial_points` method calculates the tutorial points, which is twice the number of times the piece has been rotated.

The `calculate_score` method calculates the score based on the number of lines completed, the number of ticks, the tutorial points, the number of holes, and the difference in pillar height.

The `collision_with_move` method checks if a move would result in a collision. It checks for collisions with the game board boundaries and other Tetriminos.

The `clear_completed_lines` method clears completed lines from the game grid. A line is completed if all blocks in the line are filled.

The `on_piece_collision` method handles a collision with a piece. It saves the current piece to the grid, clears any completed lines from the grid, and spawns a new piece.

The `move_down` method moves the current piece down. If the piece would collide at the new position, it handles the collision.

The `handle_input` method handles a user input. It moves or rotates the piece based on the input.

The `handle_input` method allows an external entity (like a human player or an AI agent) to interact with the game by performing actions. The `tick` method advances the game state by one tick, which can also result in changes to the game state (like moving the current piece down).

The class receives data primarily through method parameters. For example, the `handle_input` method receives an action that should be performed (like moving the piece to the left, right, or rotating it). The `from_json` method receives a JSON object that represents a game state, and it initializes a new `Tetris` instance with that state.

The `save_piece` method saves the current piece to the grid. It saves each block of the piece to the corresponding position in the grid.

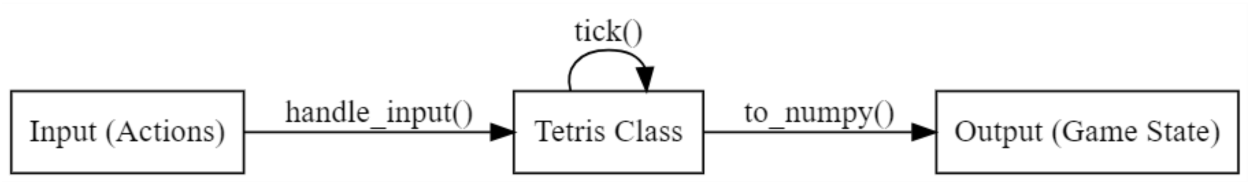
The `spawn_new` method spawns a new Tetrimino. It randomly selects a new shape and sets it as the current piece. If the new Tetrimino would collide with existing blocks, it ends the game.

The `tick` method advances the game by one tick. It handles the user's input, moves the current piece down, and increments the tick count.

The `to_str` method converts the game state to a string. It creates a string representation of the game board with the current piece drawn on it.

The `__repr__` method defines the string representation of the game. It uses the `to_str` method to convert the game state to a string.

The `Tetris` class can save the game state to a JSON object using the `to_json` method, and load it from a JSON object using the `from_json` method. This allows the game state to be saved and loaded, which can be useful for training AI agents over multiple sessions.



Pic. 5. Data Flow Representation in tetris.py

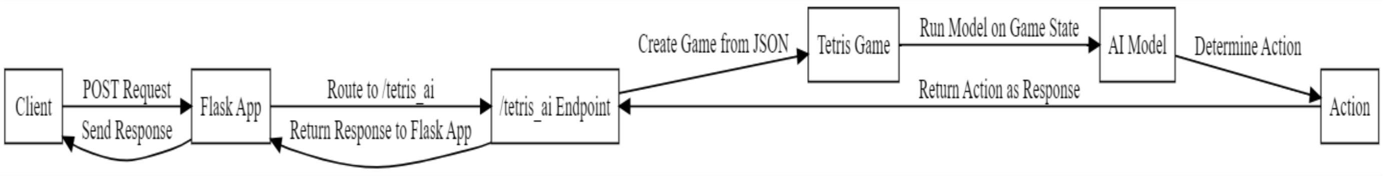
In summary, the `Tetris` class in `tetris.py` receives data through method parameters, sends data by returning it from methods, and allows the game state to be saved and loaded through JSON. It provides an interface for external entities to interact with the game.

4.1.3. WebService API

This how data flow works in web_service.py

1. **Client Sends Request**: A client sends a POST request to the `/tetris_ai` endpoint of the Flask app. The request contains JSON data representing the current state of a Tetris game.
2. **Flask Routes Request**: The Flask app routes the request to the `tetris_ai` function.
3. **Create Tetris Game**: The `tetris_ai` function retrieves the JSON data from the request and uses it to create a `Tetris` game instance using the `Tetris.from_json(data)` method.
4. **Run AI Model**: The `tetris_ai` function then converts the game state to a numpy array using the `game.to_numpy()` method and passes it to the `run_model(x)` function.
5. **Initialize AI Model**: The `run_model(x)` function initializes a `Net` instance (the AI model) and loads the trained weights from the `./MODEL` file.
6. **Predict Action**: The `run_model(x)` function runs the AI model on the game state and gets the output. It then determines the action with the highest probability using the `to_action(y)` function.
7. **Convert Action to String**: The `run_model(x)` function converts the action to a string using the `action_to_str(action)` function and returns it.
8. **Return Response**: The `tetris_ai` function returns the action as a response to the client, along with a 200 status code.

So, the data flows from the client to the Flask app, then to the `tetris_ai` function, then to the `run_model(x)` function, and finally back to the client as a response.



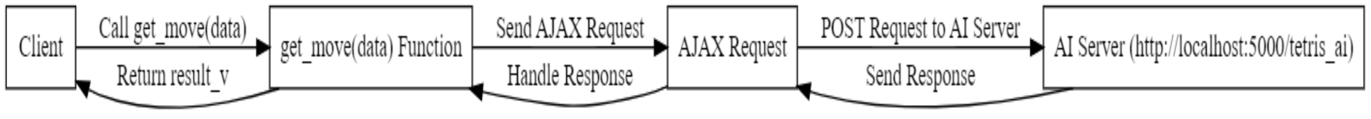
Pic. 6. Data Flow Representation in Web Service

This section explains how data moves from server to Tetris game

1. **Initialize `result_v`**: A variable `result_v` is initialized with a default value.
2. **Call `get_move(data)`**: The `get_move(data)` function is called with some data. This data represents the current state of a Tetris game.

3. **Send AJAX Request**: The `get_move(data)` function sends an AJAX request to the AI server at `http://localhost:5000/tetris_ai`. The request is a POST request and the data is sent as JSON.
4. **Server Processes Request**: The AI server receives the request, processes the data, and sends back a response. The response is the move that the AI has decided to make.
5. **Handle Server Response**: The `get_move(data)` function handles the server's response. If the request is successful, the response is stored in `result_v`. If there's an error, the error message is stored in `result_v`.
6. **Return Result**: The `get_move(data)` function returns the value of `result_v`, which is either the move from the AI or an error message.

So, the data flows from the `get_move(data)` function to the AI server and then back to the `get_move(data)` function.



Pic. 7. Request and Data Handling

4.1.4. Tetris Game (JavaScript)

Index.html file for the Tetris game contains the structure and layout of the game interface. The game elements include the stats display, button mapping, canvas container, and game over overlay. The necessary JavaScript files for the game functionality are also included.

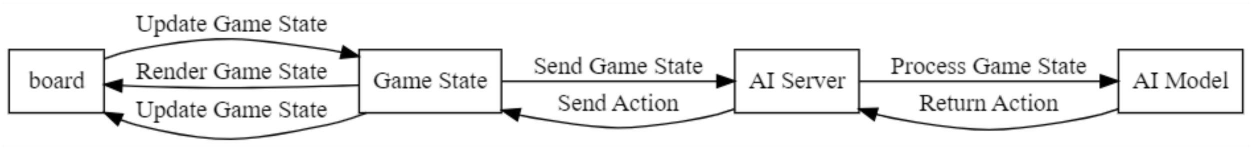
The Tetris game is a combination of a front-end JavaScript game and a back-end AI server. Here's a general breakdown of its components and data flow:

Components:

1. **Front-End Tetris Game (JavaScript)**: This is the visual part of the game that users interact with. It's responsible for rendering the game board, handling user input, and updating the game state.
2. **Back-End AI Server (Python Flask)**: This server uses a trained AI model to decide the best move given the current game state. It exposes an endpoint (`/tetris_ai`) that the front-end game can send requests to.
3. **AI Model (Python PyTorch)**: This is a neural network model that has been trained to play Tetris. It takes the current game state as input and outputs a probability distribution over possible moves.

Data Flow:

1. **Game State to AI Server**: When it's the AI's turn to play, the front-end game sends the current game state to the AI server as a JSON object.
2. **AI Server to AI Model**: The AI server converts the game state to a format that the AI model can understand (a numpy array), and then runs the model on this data.
3. **AI Model to AI Server**: The AI model outputs a probability distribution over possible moves. The AI server chooses the move with the highest probability and converts it to a string.
4. **AI Server to Front-End Game**: The AI server sends the chosen move back to the front-end game as a response to the original request.
5. **Front-End Game Updates**: The front-end game updates its state based on the move chosen by the AI, and then renders the new state to the screen.



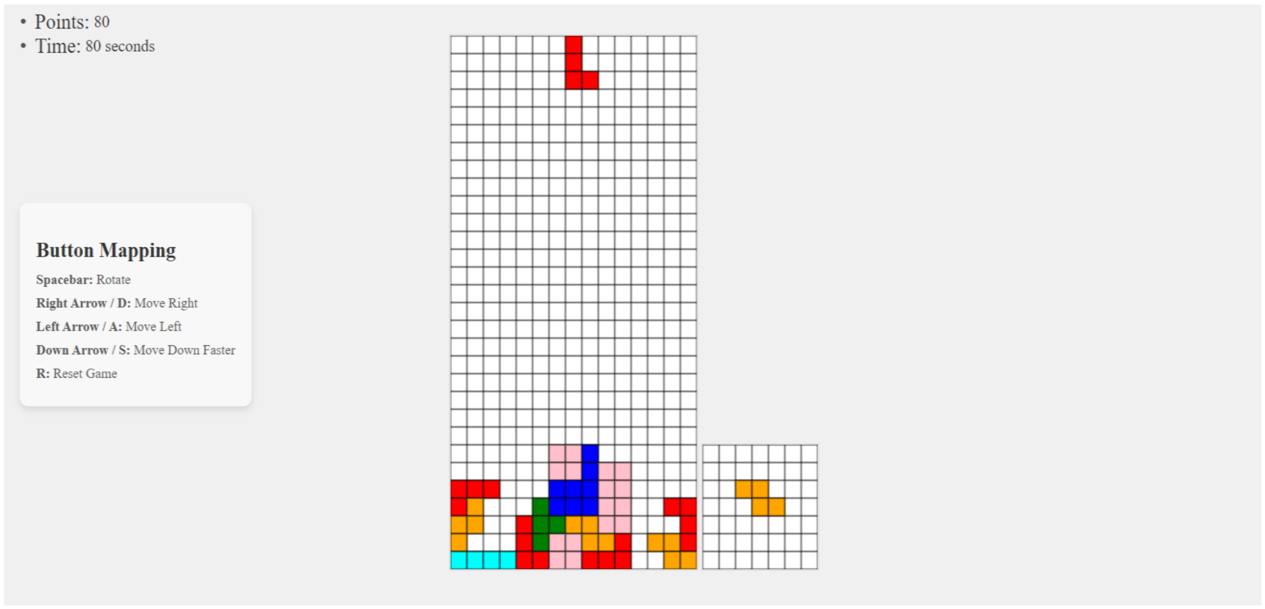
Pic. 8. Interaction model between server and Tetris game

4.2. User Interface (UI)

This section covers our software's user interface design. It delves into layout, visual elements, and user interactions. Stay tuned for insights into our user-friendly and visually engaging interface.

4.2.1 General Overview

Our Tetris game features a game board where pieces fall, a section displaying the next piece, a point tracker, a timer, and a button mapping section. Dive in for a closer look at these integral parts of our engaging gameplay experience.



- Points: 80
- Time: 80 seconds

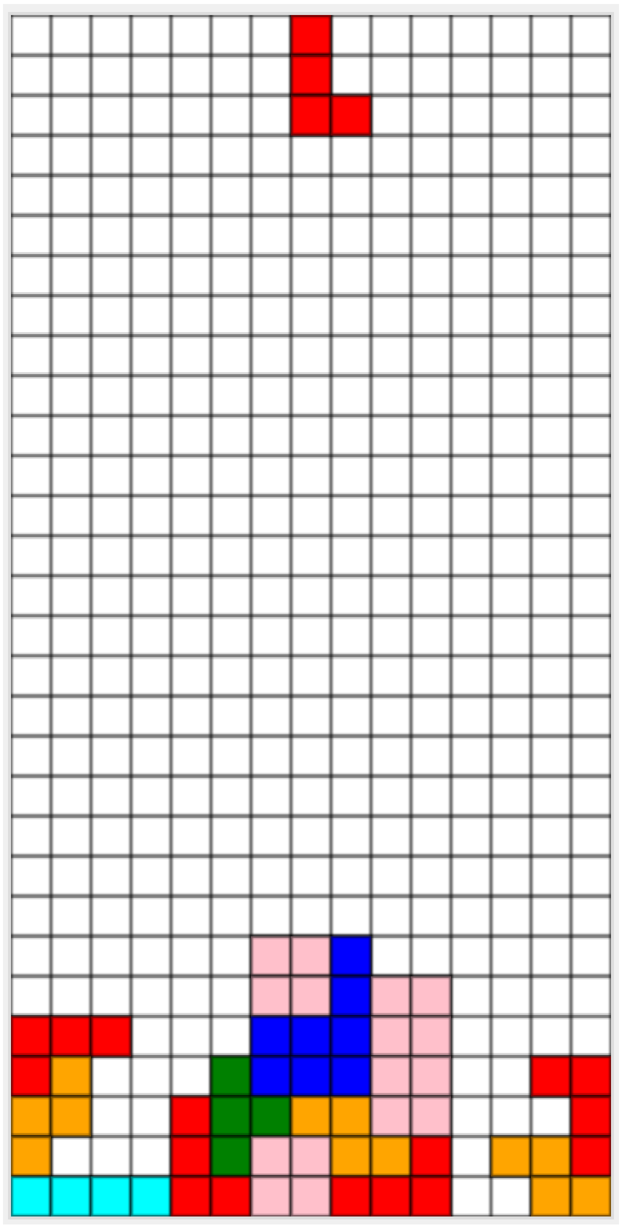
Button Mapping

Spacebar: Rotate
Right Arrow / D: Move Right
Left Arrow / A: Move Left
Down Arrow / S: Move Down Faster
R: Reset Game

Pic. 9. General UI

4.2.2 Gameboard

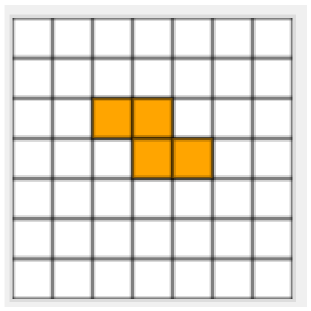
The game board serves as the primary area where Tetriminos fall, and players strategize their moves. It's a grid-based layout where pieces descend from the top. Players aim to manipulate these pieces to create complete horizontal lines, causing them to disappear and score points. The game board is the central space where the core gameplay unfolds, demanding quick thinking and precise placement from the player.



Pic. 10. Game Board

4.2.3 Next Piece Board

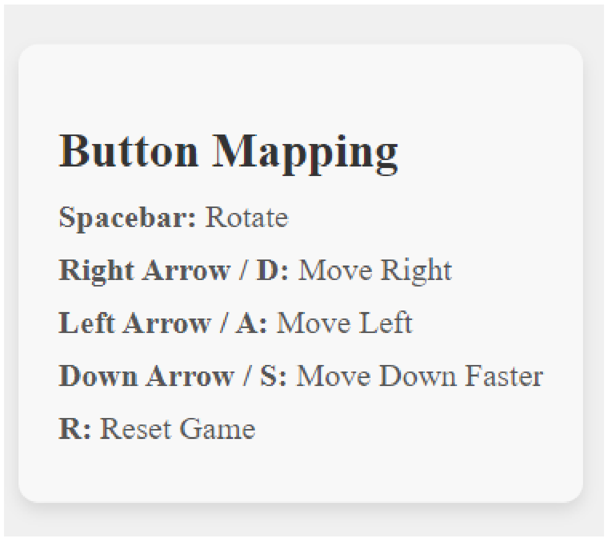
The "Next Piece" board is a designated section within the Tetris interface that displays the upcoming Tetrimino shape that will descend onto the main game board after the current piece is placed.



Pic. 11. Next Piece Board

4.2.4. Button Mapping

The "The Button Mapping" section provides guidance or information about gameplay mechanics, controls, or strategies to help players understand how to play the game effectively.



Button Mapping

Spacebar: Rotate

Right Arrow / D: Move Right

Left Arrow / A: Move Left

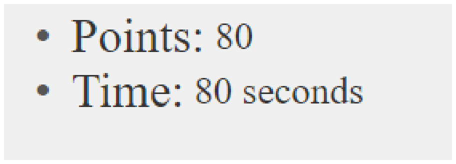
Down Arrow / S: Move Down Faster

R: Reset Game

Pic. 12. The Button Mapping

4.2.5. Timer/Point

The "Timer" tracks gameplay time, adding urgency, while "Points" display the player's score, reflecting their progress and achievements in the game.

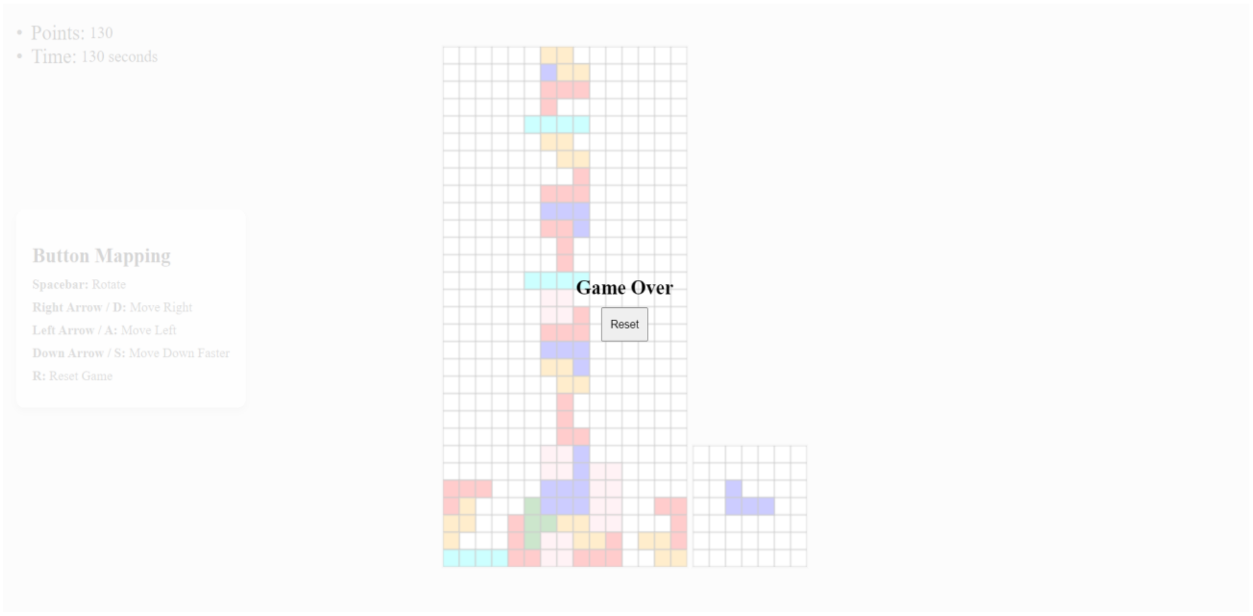


- Points: 80
- Time: 80 seconds

Pic. 13. Timer/Points

4.2.6. Game Over

The "Game Over" page displays at the end of the game, indicating the conclusion of the gameplay. It showcases a message stating "Game Over" and typically offers an option or button to restart the game.



Pic. 14. Game Over overlay

5. TESTING

5.1. Introduction

Our testing process revolves around thoroughly examining the Tetris game's functionalities to guarantee its smooth and intended operation. We aim to ensure that each aspect, from gameplay to user interaction, aligns with design specifications and delivers an enjoyable experience.

Our objective is to conduct comprehensive tests on the game's components, including piece movements, user controls, and interactive elements. This meticulous approach aims to identify and address any issues or unexpected behaviors that could affect the game's performance.

NOTE: Due to the nature of the implemented AI testing won't be implemented for AI and its training ground, the reasoning for this action will be explained in conclusion.

5.2. Unit Test

There are 5 test cases for this part in the folder testing. You just need to have a browser and just run the desired test index.html

Test Case 1

Game Over Test

Description: Test to ensure that the game detects when it's over, either by reaching the top or when pieces stack up.

Input: Simulate a scenario where the game board is filled or a tetrimino reaches the top.

Expected Output: The game should stop the loop, show a "Game Over" message, and prevent further moves.

Test Case 2

Lines Completed Test

Description: Validate that completed lines are properly cleared from the board.

Input: Create a scenario where one lines is completely filled.

Expected Output: The filled lines should be cleared, and the lines_completed variable should increment.

Test Case 3

Move Down Test

Description: Ensure that tetriminos move down correctly upon each iteration.

Input: Trigger a move_down action and check the tetrimino's new position.

Expected Output: Tetrimino should move down until it collides or reaches the bottom of the board.

Test Case 4

Movement Test

Description: Verify the movement functionalities (left, right and moving down instantly) of the tetriminos.

Input: Trigger move_left and move_right and move_down_fast actions separately.

Expected Output: Tetriminos should move horizontally without colliding with other pieces or exceeding board boundaries and pieces going down should collide and and don't pass other pieces.

Test Case 5

Rotate Test

Description: Test the rotation of tetriminos.

Input: Trigger rotation and check the tetrimino's new orientation.

Expected Output: Tetriminos should rotate within the bounds without collisions.

This testing document outlines the different test cases you should perform and their expected outcomes. You can execute these tests systematically to verify that the Tetris game functions as expected in different scenarios.

5.3. Integration Testing

Integration testing ensures that different parts of the code work together seamlessly.

Test Scenario 1: Rendering Integration

Objective: Verify that rendering functions work together as expected.

Test Steps:

Simulate rendering of Tetriminos on the game board.

Validate that the background grid and Tetriminos are drawn correctly.

Confirm that the next Tetrimino is rendered in the designated area.

Expected Result: All rendering functions work without errors or overlapping rendering elements.

Test Scenario 2: Game Logic Integration

Objective: Confirm that game logic and user interactions synchronize properly.

Test Steps:

Simulate user inputs (keyboard events) for moving, rotating, and dropping Tetriminos.

Verify that the game logic processes these inputs accurately and updates the game board.

Check for proper collision detection and Tetrimino placement.

Expected Result: User inputs interact correctly with the game logic, updating the board and Tetrimino positions accurately.

5.4. User Interface Testing

User interface testing focuses on validating user interactions and game state transitions.

Approach:

Test Scenario 1: Keyboard Input Handling

Objective: Validate the handling of various keyboard inputs for Tetris actions.

Test Steps:

Simulate key presses for moving, rotating, and dropping Tetriminos.

Ensure that each key press triggers the corresponding game action without delay or errors.

Expected Result: All valid keyboard inputs perform the intended game actions accurately.

Test Scenario 2: Game State Transition

Objective: Verify the game state transitions (e.g., game start, game over, game reset).

Test Steps:

Test game start by initializing the game and confirming the first Tetrimino's appearance.

Trigger game over conditions and check if the game stops correctly.

Validate game reset functionality.

Expected Result: Game state transitions occur as expected without unexpected behavior or crashes.

RESULT

Tetris Game Development

The development process involved creating a complete Tetris game using JavaScript. This encompassed implementing fundamental functionalities essential for gameplay, such as tetrimino movement mechanisms, rotation mechanisms to fit the shapes into the game board, collision detection algorithms to prevent overlapping shapes, game board management for keeping track of filled rows, an extra canvas to shows next piece, and a well-designed user interface (UI) offering intuitive controls and visual representations for a seamless gaming experience.

AI Neural Network Integration

A key highlight of this project was the successful integration of an Artificial Intelligence (AI) neural network into the Tetris game. This neural network was engineered to autonomously learn and play the game. The integration involved designing and implementing a neural network architecture capable of understanding the game dynamics, strategizing moves, and making decisions based on learned patterns.

AI Training and Performance

After rigorous training sessions within the designed training environment, the AI demonstrated initial gameplay proficiency. It showcased the ability to make moves and decisions based on learned strategies. However, it's important to note that the current performance level of the AI is preliminary. The AI's gameplay, while promising, is not yet at a stage where it can consistently achieve high scores or play the game indefinitely without further training and optimization.

Future Optimization and Training

To enhance the AI's capabilities and gameplay proficiency, future plans involve extensive optimization and training iterations. This includes refining the neural network architecture, fine-tuning learning algorithms, augmenting training datasets, and possibly exploring reinforcement learning techniques. These efforts aim to elevate the AI's gameplay to a level where it can adeptly navigate the Tetris environment, achieve higher scores, and potentially play the game indefinitely.

CONCLUSION

The successful development of this Tetris game platform integrated with an AI neural network marks a significant milestone in the pursuit of AI-driven gameplay. With a fully functional platform now established, there are ample opportunities for further advancements and enhancements in the AI's learning and gameplay capabilities.

Future Enhancements for AI Training

Moving forward, the focus will be on refining the AI's learning process through several innovative strategies. One proposed approach involves initially training the agent with a single piece and gradually introducing and "unlocking" new Tetrimino pieces as generations progress. This stepwise learning technique aims to facilitate a more structured learning curve for the AI, allowing it to adapt progressively to increasingly complex gameplay scenarios.

Additionally, the game board size in traditional Tetris stands at 10x20. By considering alternative board sizes, such as a reduced board size, the training process can potentially expedite. This modification seeks to optimize training time and computational resources while ensuring substantial learning progress.

Storing and Reusing Best Models

A strategy to retain and reuse the best-performing AI models from certain generations will be implemented. By storing these optimal models and reintegrating them into subsequent training sessions, the AI can benefit from prior successful strategies. This methodology aims to foster continuous improvement by leveraging past successful learning patterns.

Absence of Explicit Testing in AI Development

Regarding the absence of comprehensive testing procedures in the AI development phase, it's important to note that the nature of AI training involves iterative learning and self-correction. As the AI learns from vast datasets and experiences, the testing paradigm differs from conventional software testing methodologies. AI models continually adapt and evolve based on trial and error within the training environment. Consequently, the focus remains on iterative training and optimization rather than conventional software testing procedures.

The established platform presents an exciting avenue for exploring innovative techniques in AI-driven gameplay. The proposed strategies for enhancing the AI's training process aim to accelerate its learning curve and elevate its gameplay proficiency, paving the way for more sophisticated and adept AI-controlled gaming experiences.

Useful links and Resources

https://github.com/uvipen/Tetris-deep-Q-learning-pytorch

https://medium.com/mlearning-ai/reinforcement-learning-on-tetris-707f75716c37

https://github.com/nuno-faria/tetris-ai

https://www.askforgametask.com/tutorial/machine-learning/ai-plays-tetris-with-cnn/

https://youtu.be/QOJfyp0KMmM?si=Eo441aLh06Hhlhhz

Tools and Libraries Used

https://pytorch.org/

https://github.com/features/copilot

https://chat.openai.com/

https://www.w3schools.com/graphics/canvas_intro.asp