A red sign with white text

Description automatically generated

**Cardiff University**

School of Computer Science and Informatics

CM3203 – Individual Project Final Report

TetrisAI – CAN A CNN-BASED DQN AGENT PLAY TETRIS?

## Abstract

This research paper explores the development and performance of a Convolutional Neural Network(CNN)-Based Deep Q-Network(DQN) AI agent which is designed to play the game Tetris. Unlike previous CNN models failing to develop or improve upon strategy. This study will explore whether a hybrid CNN model can not only play Tetris but also improve upon and create new strategies.

Using a hybrid framework, combining CNN & DQN, the AI agent was trained for 7000 Tetris game sessions. Each session was simulated within a Tetris, NES 1989 version, environment created for optimising strategy. The training sessions were designed to encourage the AI agent to explore and adapt to the environment, and then later develop and master its own strategies for playing Tetris. This methodology is designed to see if there is potential in a CNN-DQN hybrid to master and create game strategies in Tetris. Something that has previously not been achieved by CNN models alone. Previously, as demonstrated by a standalone CNN model, does not have the ability to play the game Tetris.

Results indicate minor learning progression in terms of scoring consistency and faced challenges to get a consistent line-clearing strategy. Hurdles that were met included hyperparameter tuning complexities and external disruption, graphical issues, creating extra challenges the AI agent had to face when playing Tetris.

The study outcome highlights the potential and limitations of the CNN-DQN hybrid model within a gaming environment, showing there were challenges with strategies and real-time decision-making. The direction of future research would include changing the reward structure, optimising the learning algorithms and creating or finding a new Tetris simulation environment to potentially advance the performance of this AI model within complex gaming environments.

## Acknowledgements

## Table of Contents

1. Introduction………………………………………………………………………………………………….. 4

2. Background………………………………………………………………………………………………….. 5

2.1 Introduction………………………………………………………………………………………………. 5

2.2 Reinforcement Learning………………………………………………………………………………. 5

2.2.1 Exploration vs Exploitation……………………………………………………………… 6

2.2.2 Rewards and Penalties ………………………………………………………………….. 7

2.3 Environment and Agent……………………………………………………………………………….. 7

2.4 Markov Decision Process…………………………………………………………………………….. 8

2.4.1 Policy Function…………………………………………………………………………….. 9

2.4.2 Epsilon-greedy policy…………………………………………………………………….. 10

2.5 Neural Network………………………………………………………………………………………….. 11

2.5.1 How the Neural Network Processes Information………………………………… 12

2.6 Convolutional Neural Network (CNN)…………………………………………………………….. 13

2.7 Deep Q-Network (DQN)……………………………………………………………………………….. 14

2.8 Tetris Environment……………………………………………………………………………………… 15

2.8.1 Action Set…………………………………………………………………………………….. 15

2.8.2 Environment Versions……………………………………………………………………. 15

2.8.3 Game Modes………………………………………………………………………………… 15

2.9 Other Approaches................................................................................................ 16

3. Methodology........................................................................................................... 16

3.1 Introduction......................................................................................................... 16

3.2 Approach.............................................................................................................17

3.3 Specification....................................................................................................... 17

3.3.1 Model Configuration..........................................................................…. 18

3.3.2 Hyperparameters.................................................................................. 18

3.3.3 Environment Setup................................................................................ 19

3.4 Design................................................................................................................. 19

3.4.1 System Architecture.............................................................................. 19

3.4.2 Algorithm Design................................................................................... 20

3.4.3 Integration and Role Within Tetris........................................................... 21

3.5 Implementation................................................................................................... 22

3.5.1 Training Process.................................................................................... 22

3.5.2 Optimisation Techniques....................................................................... 24

3.5.3 Evaluation Metrics................................................................................. 25

3.5.4 Troubleshooting and Tuning................................................................... 25

4. Results................................................................................................................... 26

4.1 General Performance Overview.................................................................. 26

4.2 Detailed Analysis....................................................................................... 27

4.3 Critical Incidents and Turning Points........................................................... 35

4.4 Challenges and Limitations........................................................................ 36

4.5 Summary of Results................................................................................... 36

5. Conclusion............................................................................................................. 37

6. Future Work............................................................................................................ 38

7. Reflection on Learning............................................................................................. 39

References................................................................................................................. 40

## 1 Introduction

Tetris (Anon 2024a.), is a recognised game since its first release in 1985, celebrating nearly 40 years of gameplay with multiple active communities in various versions. This paper will look at the AI challenges seen in the game Tetris, using the NES Tetris 1989 version. Within this version of the game being used at ‘Classic Tetris World Championships’(CTWC)(Anon 2024b.), we have seen human players pushing this game to the limit in recent years, and new techniques created (Rolling(Zwiezen, Z. 2021.)), record after record being broken and a “true” kill screen(Orland, K. 2024.) being reached.

Although the objective of Tetris is simple - clear lines, obtain score points and survive – the game presents a huge gap in skill levels between the bottom and the top players. An example of a Tetris move that creates this gap is advanced techniques like the “T-spin” (Ramos, J. 2019.), with multiple versions of this more, increasing in difficulty, introducing more complexity and strategy to the game. As the piece sequence in a Tetris game is random and unpredictable and there is no guaranteed best or worst move, this creates a base of complexity in the game, making it a great environment to explore the capabilities of the AI model.

Recent AI agents, like StackRabbit(Anon 2024c.), have shown the capabilities of AI models using a move search algorithm(Anon 2024c.), with - before the rolling technique was discovered – and without human limitations by achieving a score of 102 million and reaching level 237[\*4] in 2021. However, using different learning algorithms, there appears to be a gap in AI strategies' effectiveness.

Previous attempts to create Tetris-playing AIs have not all been successful. When using a standalone Convolutional Neural Network (CNN) model, which has previously failed to play Tetris due to not being capable of creating a strategy for long-term and (Deep) Q-Learning which has successfully learnt to play Tetris, shows promise when creating a hybrid model. Hybrid models using Deep Q-learning elements have been shown to benefit other models when integrated. This presents a gap within the AI’s capabilities to play Tetris at a human-standard playing level within an ever-changing and unpredictable environment.

Performing this study aims to bridge the gap by exploring the potential of a hybrid model using a CNN-based Deep Q-Network(DQN) model(Park, J.-H. et al. 2022.). The primary objectives of the research are:

* To determine if a CNN-based DQN AI agent can improve the strategic decision-making capabilities of AI when playing Tetris.
* To determine if the model can beat limitations faced by a previous standalone CNN model in gameplay strategy.
* To evaluate how the model performs when compared to existing AI agents, focussing on long-term strategic planning and adaptability to Tetris.

## 2 Background

**2.1 Introduction**

The background section will contain information about the AI agent like, Reinforcement Learning, Markov Decision Process, and the two algorithms used in this project, DQN & CNN. I will also present information about the Tetris environment and background information surrounding the research question.

**2.2 Reinforcement Learning**

Reinforcement Learning (RL) is a subset of machine learning that focuses on the interaction between both the agent and the environment. The objective of RL is how the agent can maximise the reward that is being received. Unlike supervised learning where you provide the agent with a set of training data, reinforcement learning focuses just on performance and learns from the feedback provided by the agent’s previous actions and experiences.

**Core concepts of RL:**

**Agent and Environment:** In RL, an agent makes its decisions by playing around with an environment. As the agent performs actions the environment will simulate the actions taken, providing the agent with new situations and rewards or penalties. This adopts learning adaptive behaviour within the environment.

**Objective:** The primary objective of RL is to maximise the reward being received. The agent will usually achieve this via exploring, what new actions do and how they impact the environment and exploiting, using known actions that give (high) rewards.

**Exploration vs Exploitation:** Exploration is when the agent tests out new actions to see if the highest potential rewards can be improved. Exploitation is when the agent is using known actions that provide it with the best reward or best future outcome. Implementing an RL strategy that balances these aspects will make it more effective for optimising learning.

**Learning and Decision Making:** RL is a cycle of observing, action, and feedback, repeating these steps over and over allows the agent to adapt and improve its strategies based on all the information it has received. This process works by observing the current state within the environment, deciding an action based on the current policy, performing the action, and then updating the policy based on the reward or penalty received and then a new state is created by the environment.

A diagram of a diagram

Description automatically generated

Figure 1, Simple Reinforcement Learning Loop (Source: Anon)

**Theoretical Foundations:**

**Markov Decision Processes (MDPs):** Many RL problems include the use of MDPs, which will make sure that the outcomes are partly random and partly controlled by the decision maker (AI agent). MDPs are a mathematical framework used for modelling decision-making for situations when outcomes are uncertain.

**Policy Function:** This function will define how an agent behaves within certain situations based on the policy. A policy provides a map, from previous state environments encountered to the actions to be taken within the current state.

**Reward Function:** Within RL, the reward function provides feedback on how good or bad an action was when used in a particular state. This is the feedback received that is used to update the policy, which gives information to the agent about the best and worst moves within the current state.

Reinforcement Learning can be applied in many ways from robotics, where the agent has to manipulate objects (e.g. gripping and lifting a can of coke or 10kg block of metal), to being applied in gaming like chess, where some agents have been trained to play above and identify moves to humans seem illogical at first. In a complex environment such as Tetris, RL will let the agent be able to make strategic decisions and improve the performance and survivability of the agent learnt from the game’s dynamics.

**2.2.1 Exploration vs Exploitation**

A diagram of a triangle

Description automatically generatedIn reinforcement learning, the concepts of exploration and exploitation are a fundamental trade-off. Exploration invites the agent to try out new actions to potentially find rewards that can translate into long-term and yield more rewards, for more effective strategies, like using advanced moves in Tetris, T-spins. Exploitation invites the agent to try and abuse the reward system to yield the highest reward repeatedly, based on the knowledge that has been obtained from previous game states

Figure 2, Graph demonstrating epsilon decay rate as the model moves from exploration to exploitation (Source: Python Lessons)

**Importance of Creating a Balance:**

**Risk of Over-Exploration:** Excessive exploration will stop the agent from effectively using strategies discovered that reap higher rewards, resulting in potential delayed or reduced performance improvements.

**Risk of Over-Exploitation:** Excessive exploitation will hinder the progress and limit the knowledge of the agent due to no or minimal exploration, resulting in using suboptimal policies due to exploiting a strategy that was discovered early in the training session and missing out on discovering new strategies that could yield better rewards.

**Adaptation in Complex Environments:**

The Tetris environment provides a great scenario, due to the randomness of the incoming Tetris pieces, it creates a complex and constantly changing environment. These features of the environment require the agent to be able to maintain a well-rounded balance between;

**Exploring:** Testing new moves and strategies to keep adapting to the unpredictable sequence of Tetris pieces.

**Exploiting:** Repeatedly using the best-known strategies to optimise performance and survivability.

**Flexibility:**

Maintaining a balance between exploration and exploitation, allows the agent to be flexible and respond to the challenges faced by the game’s environment. After many iterations of the environment, the agent will be constantly making adjustments to the strategy being used via continuous learning, developing a near-optimal strategy.

**2.2.2 Rewards and Penalties**

In reinforcement learning one of the foundations needed is rewards and penalties, where the AI agent learns and improves strategy over time. These feedback mechanisms will impact the AI agent directly in terms of what decision and behaviour the agent employs to maximise the rewards and minimise the penalties.

Rewards are given to reinforce behaviours, in the agent, that lead to a desired outcome. This notifies the agent that the action performed was beneficial which encourages the agent to perform the same move again within a similar situation in the future. For example, within a game environment like Tetris, rewarding an agent for a line clearance will then encourage the agent to find a strategy to get the most lines cleared.

Penalties are the opposite of rewards and discourage the agent from performing undesired actions. It informs the agent that the action taken negatively impacts the outcome, encouraging the agent to avoid performing an action similar to that again. Within gaming environments, the most common penalty you will see is if the agent receives a game over, encouraging the agent to stay alive.

Creating a balance between the two types of feedback is crucial for effective learning. Creating unbalanced rewards can encourage the agent to greedy strategies that we don’t want. On the other hand, excessive penalties can cause the agent to be too safe when performing actions. Creating an optimal approach needs a balance between the two types of feedback that drive the agent into strategies that are robust and useable in many game states.

By providing consistent structured feedback in a reinforcement learning system, the agent can adapt to complex environments, adjust its strategies and make better decisions.

**2.3 Environment and Agent**

In reinforcement learning (RL), The environment and the agent are essential and the centre of the learning process. The two are closely linked, providing a framework so that the AI agent can interact and learn about the environment.

**Environment**

The environment holds all the information needed for the agent so that it can interact with it. The environment will provide challenges and game states to the agent, and react to the actions taken by the agent providing new states and rewards. The role of the environment is to provide a simulated real-world scenario of the game Tetris, where there is space for the agent to play and learn through experience. Within the environment, each action that the agent performs causes state changes and will have an impact on the agent’s ability to perform the decision-making process.

**Agent**

The agent is the decision-making entity in RL. Via the environment’s states, it interprets information through sensors or signals and decides what action to take to achieve the objective. The main goal that the agent is trying to achieve is, taking the best action within different states, this is impacted by the feedback; rewards and penalties, from the environment. The agent then uses the feedback to improve strategies and decision-making capabilities over time. The agent controls the learning and adaptability of the AI system, updating its knowledge within the environment constantly, in some ever-changing conditions.

The interaction between the agent and the environment is essential for the reinforcement learning process. The interaction consists of the agent interpreting the state of the current environment, based on the current policy, it will choose an action to perform and then receive feedback, either a reward or penalty. By taking into account the feedback, the agent will be encouraged to develop a policy that can maximise the obtainable reward. Repeating this loop, observing, action, and feedback the agent, in the learning process is going to adapt and optimise the strategies within a complex environment.

**2.4 Markov Decision Process**

A diagram of a diagram

Description automatically generatedA group of circles with letters and numbers

Description automatically generatedThe Markov Decision Process (MDP) is a framework using mathematics and modelled for making decisions within the environment. Some outcomes are partly random, for exploration, and partly controlled by a decision-maker (the agent). MDPs are shaped by using states, actions, rewards, and transitions that are contained within the environment.

Figure 3, Shows Markov Decision Process with probabilities of performing an action. (Source: waldoalvarez)

Figure 4, Simplified version of Markov Decision Process showing a typical day of what student may do. (Source: Berg, E)

**MDP Components:**

**States:** These are the potential conditions that an agent may find itself when interacting with the environment. Every state seen by the agent will provide context based on the decisions that were made and the current understanding of the environment.

**Actions:** In every state, the agent has a choice from a set of actions. When the agent performs an action, this leads to a transition into a new state

**Rewards:** Based on the actions performed, the agent will get rewarded or penalised, examining how successful the previous action was.

**Transitions:** The transition function, based on the chosen action, is how the agent moves from state to state. The function captures the dynamics of the environment, which impacts the agent’s decision-making.

**Decision Process in MDP:**

In an MDP, the decision process works by evaluating the current state of the environment, selecting an action that is deemed to obtain the best future rewards, executing the action, and transitioning to a new state. Repeating this cycle until the environment has reached its final state or the process is restarted. The objective of the agent is to find a policy, and strategy of which actions to choose in certain states, that will yield the best future reward, although there is a factor that will reduce the weight of future rewards making immediate rewards more appealing.

**Utility of MDPs:**

MDPs provide a strong foundation for understanding how optimal decisions are made within environments. They have been applied in a few areas of AI systems and are crucial in certain aspects. MDPs can be found in robotics, agents navigating through an environment automatically, to complex strategy games such as Chess, where the environment conditions are uncertain, but decisions still need to be made.

**2.4.1 Policy Function**

A policy function is how the agent develops a strategy helping it decide on the next move relevant to the current state of the environment. A policy is a map of states of the environment to the actions the agent should perform when in those states.

**Policy Role:**

**Initial Policy:** At first the policy, in reinforcement learning is unknown and random as the agent does not have any information about the environment. During the initial phase, the policy will encourage the agent to explore different actions, what they do and how they interact with the environment.

**Policy Improvement:** As the agent interacts with the environment and receives feedback, based on the actions taken, in the form of rewards and penalties, the agent will update the policy, improving the strategy to try and maximise the cumulative reward. This is an essential process of learning and improvement to the agent’s ability to perform the task at hand.

**Adapting in Complex Environments:**

**Adaptability:** In complex environments, like Tetris, the agent may have to develop and use multiple policies that can be used within different scenarios. By being adaptable to different scenarios this aids the agent in finding strategies that can be optimised for multiple scenarios, enhancing the agent’s performance drastically.

**Strategic Diversity:** Within a complex environment, like Tetris, having multiple strategies may consist of what stack strategy the agent will choose and based on upcoming pieces or the current board state, the agent may adjust how to stack the pieces. The ability to use multiple strategies based on the current environment state is crucial for a high-scoring game, this showcases the policies learnt by the policy function in RL.

**Goal of Optimising Policies:**

The goal of improving a policy function is to find the best strategy that will maximise the reward over time. The agent must be able to balance taking the immediate reward and looking at maximising the future reward. This challenge can be found in an environment like Tetris, due to the unpredictability of incoming pieces.

**2.4.2 Epsilon-Greedy Policy**

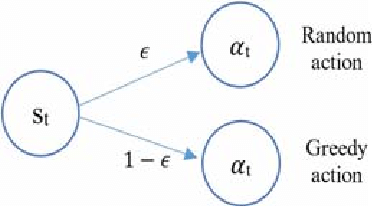
The epsilon-greedy policy is an implementation used with a reinforcement learning framework, designed to balance out exploration and exploitation. By introducing the parameter, epsilon (ϵ) as a probability of performing a random action or 1- ϵ as a probability of performing a greedy action (The best predicted available move). During the early phases of training the ϵ is set high and encourages the agent to explore and gain experience with the environment space.

Figure 5, demonstrates epsilon-greedy action selection

(Source: Utic, Z)

**Functionality and Adaptation:**

**High Epsilon Value:** At the start of training, the higher the epsilon value, the more inclined the agent is to explore the environment, this reduces the agent having tunnel vision for suboptimal strategies. Within environments that require a range of experiences to understand the full range of potential states and actions, it is crucial for the agent to explore.

**Epsilon Decay:** As the agent trains, the ϵ is going to slowly reduce, moving the policy away from exploration and towards exploitation of the learnt strategies that will return the best reward. Making sure the decay rate can balance a good amount of exploration with efficient exploitation.

**Epsilon in Complex System:**

**Adaptability to Randomness:** In environments similar to Tetris, where the challenge is created by unpredictability and complexity, the epsilon-greedy policy allows the agent to constantly adapt by adjusting the balance between exploration and exploitation. Being able to adapt is crucial if the agent wants to develop a strong strategy that can keep up with the ever-changing environment.

**Decay Rate Impact:** Depending on the speed of the epsilon decay this affects the agent’s ability to learn:

**Slow Decay:** Promotes over-exploring. A slow decaying ϵ could cause the agent to over-explore, this delays and/or stops the agent from improving upon better strategies.

**Fast Decay:** Promotes rapid exploitation. A fast-decaying ϵ makes the agent prematurely decide on a strategy which will be suboptimal. This is due to the agent not exploring the environment properly and will not discover better strategies.

The ability of epsilon-greedy policy to adjust the exploration level makes for a great application when the optimal policy is unknown and needs to be discovered by interacting with the environment.

**2.5 Neural Network**

Neural Networks (NNs) are a core part of the foundation of artificial intelligence (AI), with the purpose of simulating the neuron structure of the human brain. As the data processing isn’t done via explicit commands, the neural networks use layers to interconnect nodes (“neurons”), all able to perform simple calculations, to interpret sensory data, like humans but, learn through machine perception and other methods. As the AI agent develops knowledge from input patterns, the response to the environment state will adjust.

**Key Components of Neural Networks:**

**Input Layer:** This layer receives raw data, e.g. Game states in Tetris, Tetris piece configuration, etc. Each neuron within the layer will be linked to one aspect of the input data.

**Hidden Layer(s):** These layer(s) process the inputs into meaningful patterns. The hidden layer can contain one or multiple layers that can receive data from the input layer processing transforming it into usable data for the output layer.

A screenshot of a computer

Description automatically generated**Output Layer:** This layer produces the final output, the decision of the network, which is trying to identify the best move based on the current state of the Tetris environment.

Figure 6, shows the layout of how the basics of a neural network and how it may be structured. (Source: Neutelings, I)

**Learning with Neural Networks:**

A diagram of a function

Description automatically generated**Weights and Biases:** Each connection between nodes found in a neural network is defined via weights, which are given a random initial value at the beginning of training. These weights scale the input data and biases are then added as an additional offset. These two parameters are fundamental and adjust during the training sessions to improve the network’s decision-making. ***weight ⋅ input + bias***

Figure 7, Shows how the each node (neuron) in a neural network is defined (Source: Gibaru, O**)**

**Loss Function:** Measuring the effectiveness of a neural network is done via a loss function which calculates the difference between the network’s predictions and the actual outcomes. Commonly loss functions include mean squared error for regression tasks and cross-entropy loss for classification.

**Backpropagation:** This is a common method used when training neural networks, where the loss is calculated and then propagated back through the network. This is the process that adjusts the weights and biases, intending to minimise the loss and improve the network’s ability to accurately predict the best move.

**2.5.1 How Neural Networks are Applied in this Project**

In this project, the neural networks analyse and make decisions based on the visual input from the data, the game states, provided by the Tetris game environment. This means the AI agent can recognise patterns like Tetris block shapes and predict the outcome of multiple moves, developing strategic decision-making when playing Tetris.

**Role in the Project:**

**Pattern Recognition:** Neural networks create the ability for AI agents to be able to look and interpret the game environment, similarly to how a human player analyses the board.

**Strategic Decision Making:** Processing the visual data, the network learns how to recognise and react to different game board states, this pushes the AI agent to make more informed decisions within its gameplay.

**2.6 Convolutional Neural Network (CNN)**

A diagram of a network

Description automatically generatedConvolutional Neural Networks (CNNs) are a specialisation of deep neural networks that are very effective when analysing visual information. CNNs are designed to learn spatial hierarchies of features via backpropagation, from low-level features (Simple textures and edges) to high-level patterns (the whole object and the object’s parts individually).

Figure 8, Shows how a CNN model is structured (Source: Neutelings, I)

**Basic Capabilities of CNNs:**

**Pattern Recognition:** CNNs are used and effective for recognising patterns found within images. This is achieved via a series of convolutional layers which each applies its own filters to the input, as it moves through each layer extracting features becoming more complex.

**Feature Extraction:** Using a multi-layer structure, CNNs take the raw data and transform it into useful representations. Using this transformation of data is essential when tasks need to understand complex visual inputs.

**Visual Tasks:**

In tasks where there is a lot of visual data and provides useful information, CNNs will thrive in this environment type. Successfully being applied in areas like:

**Imagine and Video Recognition:** Creating an automatic process to identify patterns in images or videos. This could be something like face recognition.

**Medical Image Analysis:** The use of CNNs in this area assists in diagnosis by interpreting a medical scan image and detecting conditions.

**Autonomous Vehicles:** Allowing cars to visually interpret the surrounding area to navigate in a safe manner.

**Relevance to Gaming Environments**

CNNs using their ability to process complex visual information with accuracy and speed can enhance AI gameplay in strategic games. In a game like Tetris, CNNs would be able to interpret how the pieces are placed, assess the current game state, and then predict the impact of moves. This capability is essential for agents that can react in real-time, produce strategies and simulate human standard gameplay. By interpreting the visual data, the network identifies patterns of piece placement and predicts outcomes, CNNs can aid AI agents in making informed decisions to yield the best reward.

**Role in Complex Decision-Making:**

The CNNs' ability to process and interpret complex visual inputs makes them extremely useful in an environment where fast and accurate visual assessments are needed. CNNs support the recognition of the current game states but can aid when thinking about future scenarios, which it can pick up from learning patterns, improving the strategic ability of the agent.

**2.7 Deep Q-Network (DQN)**

Deep Q-Network (DQN) is an approach in reinforcement learning that utilises the combination of the traditional Q-learning algorithm with Deep neural networks. Using DQN gives an agent the ability to handle complex decision-making tasks from high-dimensional state spaces and feature-rich input processing. This method represents a huge advancement in reinforcement learning and allows for improved capabilities than traditional methods.

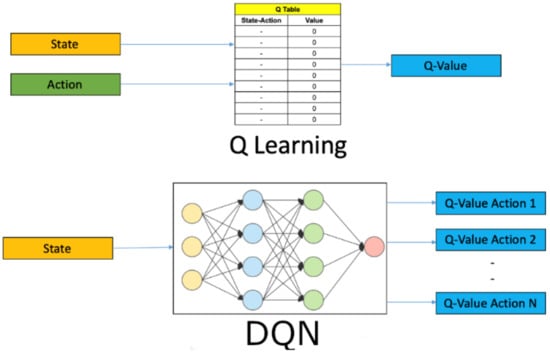


Figure 9, Shows Q-Learning vs. DQN and how Q-Learning is integrated. (Source: Wang, Y. et al)

**Functionality and Advantages:**

**Q-value Estimation:** DQN through the use of deep neural networks to estimate Q-values, representing what the potential long-term rewards can be for each action within the current state.

**Generalisation:** Unlike Q-learning which has problems with large state spaces, DQN thrives within environments with big, continuous, or high-dimensional state spaces like the one presented by the video game Tetris. DQN can do this due to its ability to generalise information from visual or sensory inputs, making it effective within changeable environments, like video games.

**Action Selection:** Balancing between exploration and exploitation, the DQN will employ an epsilon(ϵ)-greedy policy where it will select actions randomly with a probability ϵ to explore the environment, and greedily based on the Q-values with probability 1- ϵ to exploit known information.

**Optimisation:** The network trains by minimising the loss between predicted Q-values and target Q-values, these values are updated using the Bellman equation which refines the policy towards the most optimal decisions.

**Role in Complex Environments:**

In a video game, where the player needs to quickly assess what they are looking at and respond to the continually changing environment, DQN can enhance decision-making dramatically. By processing an input from the game state, in Tetris this could be the Tetris piece placement, DQN can predict the outcomes of multiple actions and select the best one to create a strategy that is going to yield the highest reward.

**Significance in AI Development:**

The implementation of DQN in deep learning with reinforcement learning has demonstrated AI capabilities, where we see developed agents performing at or above human standards in complex tasks. Making for an incredibly powerful tool that can be applied to almost any real-world need, that requires decision-making.

**2.8 Tetris Environment**

The Tetris environment, ‘gym-tetris’, is a recreated environment of the NES Tetris game, 1989 version. The game board consist of a grid that is 20 blocks high and 10 blocks wide. Tetris contains 7-piece shapes; O, I, T, L, J, S and Z. These pieces are used to solve the puzzle, within the game, which is trying to slot all pieces together as efficiently as possible with minimal gaps and maximising the amount of Tetris obtained.

**2.8.1 Action Sets:**

The environment contains two sets of action lists: ‘MOVEMENT’ & ‘SIMPLE\_MOVEMENT’.

SIMPLE\_MOVEMENT: This contains 6 actions,

‘NOOP’, No operation.

‘A’, Rotate left.

‘B’, Rotate right.

‘right’, Move right.

‘left’, Move left.

‘down’ Move down (Soft drop).

MOVEMENT: This contains 12 actions that add slight complexity, combining directional and rotational actions,

['NOOP'], ['A'], ['B'], ['right'], ['right', 'A'], ['right', 'B'], ['left'], ['left', 'A'], ['left', 'B'], ['down'], ['down', 'A'], ['down', 'B'],

**2.8.2 Environment Versions:**

The environment also has 4 different versions: v0, v1, v2, v3. These environments are slightly different from each other via rewards & penalties.

V0: Rewards the agent based on score.

V1: Rewards the agent based on lines cleared.

V2: Rewards the agent based on the score but penalises height.

V3: Rewards the agent based on lines cleared but penalises height.

**2.8.3 Game Modes:**

A-type: Original game, Endless will continue playing until the player dies

B-type: Arcade mode, this may include 999 lines or 150 lines in the game but for the environment, it is 150-line mode. This mode is just a timer to see how quickly the player can clear 150 lines.

**2.9 Other AI Models in Tetris**

Artificial Intelligence (AI) applications within games, like Tetris, can be approached in many ways, that can develop different strategies. This section will be a brief review of other AI Models that have attempted to play Tetris with relevance to the research. Highlighting a standalone model using Convolutional Neural Networks (CNNs) and Deep Q-Networks (DQNs). Reviewing these models should provide some insight into the strengths and limitations of both models.

**Standalone CNN Model**

The use of CNN models has been explored in many areas, like gaming, for the ability to capture visual data and pattern recognition. An example of CNN being applied was documented by Stanford University (Stevens, M. and Pradhan, S.), they tasked a CNN model to interpret and interact with a Tetris game environment. Although CNNs have incredible abilities in image classification and recognition within a strategy game, more specifically Tetris which has presented challenges. A standalone CNN has struggled when it comes to long-term planning and decision-making, vital in Tetris as each state is dependent on the previous action taken. Within the study, they highlight the limitations of standalone CNNs when tasks to managing a sequence of decision-making, which is required to play Tetris, as the model lacks the ability to create a plan for future moves, demonstrating suboptimal performance in the game Tetris.

**Deep Q-Networks (DQN) Model**

Deep Q-Networks (DQN) have been successful when attempting to play Tetris, demonstrating the ability to handle a complex decision-making process, required in a game like Tetris. Based on a study published on ResearchGate, DQNs are effective when they are given high-dimensional state spaces of Tetris with the use of a reward system that encourages line clearing and strategic block placement. Using this approach helps direct reinforcement learning to examine each move and take the most optimal action based on future outcomes. Outperforming usual methods, used for strategic depth and adaptability.

Combining the models, CNNs with DQN, can improve this capability. According to a study from MDPI, CNNs are excellent at extracting high-level features, e.g. patterns, from visual inputs, highly useful for interpreting a complex game environment, like Tetris. Using CNN’s ability to interpret visual data with DQN’s strategic decision-making, creating a hybrid model has the potential to have a better understanding of the game states. This should create a better overall assessment of which moves are good and which are bad.

## 3 Methodology

**3.1 Introduction**

The methodology will outline the approach used to implement and evaluate the AI model described within the Background. Detailing the steps taken to apply the theoretical aspects of neural networks, Convolutional Neural Networks(CNNs) and Deep Q-Networks (DQN), into a functional model that can interact with the Tetris game environment effectively. The focus of the study is to fill the gap in hybrid AI models in a complex environment, like Tetris.

The primary aim of this research is to evaluate the effectiveness of a CNN-based DQN hybrid model when set the task to play the game Tetris. Addressing the model’s ability to learn from and adapt to the many complex game states produced by Tetris and making strategic decisions to potentially reach the average human Tetris player’s abilities. The focus of the research will be how the CNN-based DQN model can adapt to a Tetris game environment by looking at learning efficiency, decision-making accuracy, and adaptability to the game dynamics.

The following sections will dive deeper into the technical aspects of implementation, training, frameworks, and any challenges faced throughout the research, hopefully offering insight for future AI research in similar complex environments and how these neural networks interact with each other.

**3.2 Approach**

This section will outline the methodologies used to create a functional Convolutional Neural Network (CNN) and Deep Q-Network (DQN) hybrid model within a Tetris game environment. The approach is structured to deploy the model and demonstrate the capabilities of the theoretical model to make real-time decisions.

**Model Integration and Setup:**

The first step of the model is initialising the CNN to process the visual input retrieved from the Tetris game environment, transforming raw pixel data into a structured format. The CNN’s architecture has been structured to extract detailed features from the game’s visuals, this includes the current piece shape and what the board currently looks like. Once the features have been extracted via the CNN’s layers, the DQN part of the model will take control of evaluating the possible inputs based on previously learnt Q-values, from previous games, to choose what the AI agent thinks are the most optimal moves. The CNN and DQN models are interconnected which is very crucial, this link between the models allows for data, from the CNN, to be transformed into usable data for the DQN. This provides a smooth transition for data and a simulation of a real-world scenario, where the AI agent will need to adapt quickly to new information being received.

**Data Handling and Preprocessing:**

The AI model begins playing Tetris without any prior knowledge of the game. Due to Tetris being a game where every game state will be unique after the first 10 – 15 pieces, based on the order of Tetris pieces/placement, the data preprocessing methods are crucial and must be robust. Preprocessing is performed by normalising the input data, via Batch Normalisation, to keep consistency for the neural networks when they are interpreting the game environment. Using this optimising strategy, we can optimise the speed and accuracy of the neural networks.

**Simulation and Training Environment:**

The AI model is trained via a simulated environment of NES Tetris 1989, using ‘gym-tetris’ environment, starting every Tetris game from the beginning each time it dies. This allows the agent to interact with the environment, teaching the agent the fundamentals via repeated interaction playing the game, with different scenarios produced by the randomness of the Tetris piece order which will develop into strategic planning.

**Parameter Tuning and Optimisation:**

The key hyperparameters include the learning rate, discount factor (Gamma), and epsilon decay for epsilon-greedy policy, which need to be tuned to balance exploration & exploitation. The model makes use of the Adam optimiser to improve network training, and adjustments of weights and biases to make the predictions better, reflected by a Huber loss function, this combines the components of mean squared error and mean absolute error, depending on error size. Regular updates, via Target Update, are carried out to the target network making sure the model’s learning is stable and improving.

**Evaluation and Monitoring:**

The AI’s performance is being evaluated by metrics like the number of lines cleared and the score obtained. These metrics provide quantitative feedback based on strategy effectiveness. Also visually assessing the AI's performance in real-time to determine what strategy it is adopting and how well the agent is interacting with the environment. Evaluating the agent in this way gives me insights into how the model’s decision-making process is progressing.

**3.3 Specification**

**3.3.1 Model Configuration**

**Input specifications**

**Preprocessing:** Game states of Tetris are captured as RGB images, that get converted into greyscale, reducing computational demands. The images are resized for consistency to 160x144 pixels, creating a standard input size for the neural network. The pixel values are normalised between a range of 0 and 1 to make neural network processing easier, improving data handling and model efficiency.

**Format:** The neural network receives an input via a single-channel (grayscale) image with the dimensions 160x144. The pre-processed images then get wrapped in a batch dimension and then fed into the neural network for interpretation.

**Output specifications**

**Decision Output:** The output of the neural network is a table of Q-values, each value represents the potential actions that are available in the current game state. The amount of outputs is equal to the action space of the Tetris environment, determined by the actions in the ‘JoypadSpace’.

**Interpretation:** Every Q-value represents the predicted future rewards that are available based on the action that the agent has chosen to perform. The action that represents the highest Q-value is deemed the optimal action within the current state.

**3.3.2 Hyperparameters**

**Learning Rate:** Set to 0.006, this decides the step size at each loop repeated, as it moves towards a minimum of the loss function

**Discount factor (Gamma):** Set to 0.85, this discounts future rewards as part of the Q-value calculation, creating a balance of what immediate versus long-term rewards are important.

**Epsilon Values:** At the beginning epsilon (ϵ) starts at 1.0, encouraging the agent to explore the environment, and decays over time to 0.01, encouraging exploitation of policies learnt from exploring. The decay rate is managed via epsilon-decay value, set to 850, this makes sure there is a balance between adequate exploration and exploitation.

**Batch size:** Set to 256, this is the number of training samples to collect before the model’s internal parameters can be updated.

**Replay Memory Size:** Set to 50,000, this stores transitions obtained throughout training. This is how the network learns from past experiences

**Target network update frequency:** Set to update 20 episodes. This helps stabilise the learning process of Q-values, reducing the relation between the target and current Q-values.

**3.3.3 Environmental Setup**

**Hardware Requirements**

**Primary Device:** The code is meant for systems with CUDA-compatible GPUs installed, which helps training by accelerating the process, using parallel computing capabilities. The code will check if there is a GPU present and available.

**GPU Utilisation:** Once a CUDA-capable GPU is detected and available, TensorFlow and PyTorch operations are then configured and executed on the GPU. This provides faster matrix operations and backpropagation steps

**Fallback Option:** If there is no GPU available, the model is told to use the CPU instead which will slow down the training efficiency.

**Software Dependencies**

**Python Libraries:**

**Torch:** Used for building and training the neural network models.

**Numpy:** Used for numerical operations, e.g. processing state images and other data manipulation

**PIL (Pillow):** Used for image processing tasks, like resizing and greyscale conversion

**Gym-tetris:** Custom OpenAI Gym environment designed for Tetris, simulated the game for training the AI.

**Nes\_py:** an NES emulator, which enables communication, by the Gym to run different games, like Tetris within a Gym environment.

**Other Dependencies:** Other libraries used are matplotlib to retrieve visual data like plotting the training progress over episodes.

**Configuration Variable:**

**‘DISPLAY’ Settings:** ‘os.environ[‘DISPLAY’] = ‘ 10.212.3.1xx:10.0’. This is a line of code used for remote rendering, redirecting the output to different X11 display servers. The ‘xx’ represents the GPU number being used via SuperComputing Wales, in this case, the GPU name format will look like ‘\_ccs20xx’. This is only used if you want to run the model in a non-local environment.

**Environment Configuration:** Using environment variables, like GPU settings or display settings, allows the model to be able to run in multiple computational environments without major modifications to the code.

**3.4 Design**

**3.4.1 System Architecture**

**Convolutional Neural Network (CNN)**

**Input Layer:** CNN receives the pre-processed game state as an image, which is converted into grayscale and resized to a consistent dimension (160x144 pixels). Simplifying the input format will help reduce computational complexity and load, allowing the network to put more focus on the essential features.

**Convolutional Layers:**

**First Convolutional Layer:** Starts by applying 16 filters with a kernel size of 5x5 and a stride of 2. This layer will capture the basic patterns, for example, Tetris piece shapes and edges of objects/board.

**Second Convolutional Layer:** Increases the depth to 32 filters, this is capturing more complex patterns like how to arrange the Tetris pieces and complete potential lines.

**Third Convolutional Layer:** Continuing with 32 filters, this layer further develops the features from game states into a higher-level representation, used for evaluating potential moves.

**Batch Normalisation:** This stabilises the learning process by normalising the output from the previous convolutional layer, each layer is normalised. This step stabilises the learning process and speed of training within the deep networks, enhancing training.

**Activation Functions:** The activation function used within the network is Rectified Linear Unit(ReLU). ReLU will be applied after each of the convolutional layers, making sure that positive values are only passed forward. ReLU helps by introducing non-linearity, aiding the network to learn more complex patterns and interactions with the game environment.

**Deep Q-Network (DQN)**

**Integration with CNN:**

Within the context of this project, the DQN will receive processed visual inputs from the CNN, which provide an understanding of the current game state. The CNN will extract and interpret the space of the game board and dependencies like the arrangement and the types of Tetris blocks, and output the predicted Q-values for each of the actions available

**Output Layer:** Once the processing is completed after going through the three layers of convolutional layers, the processed data is then flattened and passed to a fully connected linear layer, the bridge from CNN to DQN. This layer takes the high-level features extracted from the CNN and changes it into a suitable format, a set of Q-values, which represent the predicted reward for actions.

**Learning Process:** By employing a Q-learning algorithm which is made to constantly update the policy network based on the feedback received from the environment, based on the actions the agent is performing.

**3.4.2 Algorithm Design**

**Reinforcement Learning Strategy:**

**Q-values:** Within a DQN setup the neural network will learn to predict Q-values, which is an estimation of the total reward that is potentially obtainable by the AI agent over the future after performing certain actions within certain states.

**Action Selection:** Balancing between exploration and exploitation, the DQN will employ an epsilon(ϵ)-greedy policy where it will select actions randomly with a probability ϵ to promote exploring the environment, and greedily based on the Q-values, from previous experiences selects the action based on the highest Q-values with probability 1- ϵ to exploit known information.

**Optimisation:** The network trains by minimising the loss between predicted Q-values and target Q-values, these values update using the Bellman equation which refines the

**Reward System:**

The agent during the training sessions receives two forms of feedback: Reward (Positive Reinforcement) and Penalties (Negative Reinforcement). These rewards and penalties are what shape the agent’s learning process and encourage strategies where it can clear more lines and survive for longer.

**Rewards:**

* **Clearing a line:** the agent will receive points for each line cleared. This reward can be increased by clearing two lines and doubling the reward. The same logic applies and will receive a reward for:

single line clear: base reward,

Double line clear: base reward \*2,

Triple line clear: base reward \*3,

Tetris (four lines cleared at once): base reward \*4.

* **Score**: obtained in the game; the score reward is based on the in-game score counter, you can improve the score with soft and/or hard drops or by clearing lines.

**Custom Enhancements:** I have adjusted the line clearance reward, instead of receiving 1 reward point, the default of the environment, I adjusted it to be 100 reward points per line to encourage and push for a strategy that clears lines.

**Penalisations:**

* **Game over:** This penalisation comes into effect when the game ends and the agent has to restart a new one. This is a custom penalty I created for the environment, encouraging the agent to try and play for survivability.
* **Height penalisations:** This penalisation should discourage the agent from creating towers of blocks or creating a “bumpy” surface.

**3.4.3 Integration and Role within Tetris**

**Integration of CNN and DQN**

**Visual Processing and Feature Extraction by CNN:** CNN is very important when processing the game state images. These state images get converted into grayscale and resized to a consistent dimension. By performing this preprocessing step computational demands are lowered, allowing CNN to focus on extracting meaningful features and patterns from the visual data

**Data Flow to DQN:** Once processing of the visual input is completed after going through the three layers of convolutional layers, the processed data is then flattened and passed to a fully connected linear layer, the bridge from CNN to DQN. This layer takes the high-level features extracted from the CNN and changes it into a suitable format, a set of Q-values, which represent the predicted reward for actions.

**DQN’s Role in Decision Making:**

**Usage of CNN Outputs:** CNN provides DQN with an output of the processed visual inputs. The visual inputs contain information, which creates an understanding of the current game state. The information can be how the Tetris blocks are placed on the board and what Tetris piece is currently in play and where it can go.

**Decision-Making Strategy:** Based on CNN’s output, which provides a set of Q-values, DQN will evaluate each of the outcomes when performing a move. DQN will select the action that it has predicted to receive the highest reward, improving immediate and future rewards.

**Adaptation:** DQN adjust the strategy being used in real-time and learns from previous consequences of actions, e.g. rewarded for clearing lines or penalised for a game over.Using this learning and adaptation process should develop and improve strategies to help with the complex environment of Tetris

**Impact of AI Performance:**

**Better Decision-Making:** This integrated approach allows the AI agent to make better decisions quicker, improving the ability to respond to the game's scenarios changing quickly.

**Strategic Adaptation:** By using real-time feedback and experience to update the strategy, the agent learns to navigate the Tetris environment more effectively

**3.5 Implementation**

**3.5.1 Training Process**

**Initialisation**

**Network Setup:** Both CNN and DQN are initialised with their specific layer configurations. Each convolutional layer in the CNN uses a defined number of filters, kernel size, and stride, each followed by batch normalisation layers. DQN is initialised with a fully connected linear output layer which interprets the flattened features from CNN into the action space.

**Weight initialisation:** Weights of both networks are initialised using PyTorch’s default initialisations, which are randomised to help break symmetry.

**Environment Preparation:** The Tetris game environment is initialised using the ‘gym-tetris’ package, wrapped with custom functionality to handle game actions and rewards.

**Data loading and Preprocessing**

**Screen Capture:** Each state of the Tetris game, containing information about the Tetris board and current piece in play, is captured as an RGB image

**Preprocessing Steps:**

**Grayscale Conversion:** The RGB images are converted to grayscale images, reducing computational demands and simplifying the input while keeping necessary information.

**Resizing:** Images are resized to 160x144 pixels creating consistency of the input dimensions.

**Normalisation:** Pixel values get normalised between a range of 0 – 1, making training a faster and more efficient process

**Batch Dimension Addition:** A batch dimension is added to pre-processed images before making it into the network, making sure it follows the input requirements of PyTorch’s batch processing

**Training Loop**

**Loop Mechanics:** The training loop runs for a number of episodes, determined before the training session and depending on whether a GPU is available or not, each episode represents 1 full game of Tetris

**Data Input:** Within an episode the current state (pre-processed screen image) is input into the CNN to extract features, collected at each timestep, which gets passed via the connected layer to the DQN to evaluate each action available.

**Forward Pass:**

**Feature Extraction:** CNN processed input states, extracting the relevant features, like the current game board state.

**Q-value Estimation:** DQN estimates Q-values of each action possible within the current state provided by the extracted features.

**Action Selection:** The agent selects an actionusing an ϵ - greedy policy, where ϵ decreases over time. Creating a balance between exploration (random action) and exploitation (chosen action based on highest Q-values).

**Reward Observation and Next State:** The agent executes the action within the environment, receiving a new state and reward for the last action, which is then observed.

**Loss Calculation:** loss is calculated via the Huber loss function, combining the Mean Squared Error (MSE) and Mean Absolute Error (MAE) loss function benefits, depending on the error value, between predicted Q-value and target Q-values, which get updated using the Bellman equation.

**Backpropagation:** The loss gradient is computed, and backpropagation is performed to update the weight values.

**Batching Processing:** Training data is sampled from the replay memory; this contains the previous state transitions. This should store the most successful transitions as training samples, stabilising the learning.

**Periodic Updates:** The target network weights are updated with policy networks, and the time between updates is predetermined.

**Stopping Conditions**

**Maximum Episodes:** A limit on the training episodes to make sure the training does not continue running until stopped.

**Performance Plateau:** If the strategy or performance metrics, like cleared lines, start to show no improvements over time, suggesting training further is unlikely to get better results.

A diagram of a system

Description automatically generated

Figure 10, This diagram illustrates the workflow of the AI model used to play Tetris, beginning within the Tetris Environment and capturing the current game state.

**3.5.2 Optimisation Techniques**

Multiple techniques were implemented to improve the efficiency of learning and stabilising the neural network:

**Replay Memory:** Replay memory that can store 50,000 transitions helps the model learn from previous experiences, storing the observed transitions during gameplay. This gives the AI a diverse set of situations to learn from by replaying the previous actions, states, and outcomes.

**Target Network:** To keep stability in the learning, a target network is employed, duplicating the value network but parameters don’t change. The target network’s weight is updated every 20 episodes.

**Batch Normalisation:** This is applied to each output of every convolutional layer to normalise the activations. Stabilising the outputs of each activation throughout training, batch normalisation improves the speed and stability.

**Gradient Clipping:** During backpropagation, gradient clipping prevents the gradient from exploding by capping the gradient at a maximum value. This stops the update to the weights is not too big, keeping training stable.

**Epsilon-Greedy Strategy Adjustment:** The epsilon value for the epsilon-greedy policy begins at 1.0, promoting exploration and decays to a minimum of 0.01 balancing exploration with exploitation. This makes sure the agent does not focus on a suboptimal policy and explores multiple alternatives.

**Learning Rate and Optimiser:** PyTorch’s Adam optimizer is used with a learning rate of 0.003. Adam adjusts the learning rate for each parameter, aiding in finding optimal updates for the weights.

**3.5.3 Evaluation Metrics**

To monitor the training process, the metrics introduced are:

**Score:** This metric directly shows the score of each game the AI agent played and the ability to play within each episode by presenting the final score of the game. A higher score will indicate a better performance in the decision-making process, like line clearances and piece placement.

**Lines cleared:** This metric shows how many lines the AI agent cleared in each episode, being the fundamental metric due to Tetris' primary objective in the game, clearing lines. More lines cleared in a game; shows signs the agent is learning how to play Tetris effectively.

**Loss Monitoring:** Training loss is outputted to keep track of the training process. This shows whether the agent is learning effectively whether that’s too fast or too slow. Decreasing loss values results in the model improving predictions of Q-values.

**Observation of Training:** By observing the agent while it is training during random episodes, information like the agent’s current strategy and performance can be seen. These observations aid in understanding the agent's strategy adjustments that metrics alone cannot be reflected in visual graphs

**3.5.4 Troubleshooting and Tuning**

This section describes the challenges I have faced throughout the process and what solutions I applied to the problems to fix them.

**Common Challenges**

A few issues were faced before the training phase;

**Checkpoint Loading Errors:** The model was unable to load the checkpoints correctly.

**Customer Reward Wrapper Issues:** Custom rewards were not processed correctly. Providing the agent with the incorrect reward value.

**Sudden loss increase:** Loss suddenly increased due to a random graphical issue causing the environment to blackout completely.

**Environment Graphical issues:** This seemed to be caused by keeping the environment for extended periods collectively during a training session. Rendering the environment as useless for a human to interpret.

**Diagnostic Tools and Techniques**

**Log analysis:** Printing logs of loss value to monitor the stability of the training and used for other error messages during the time of the problem.

**Debugging Sessions:** Sessions of testing out the checkpoint saving and loading to identify the fault in the process

**Tuning Strategies**

**Checkpoint Handling:** Changing the checkpointing logic to improve file handling and that the model was correctly saved to the file

**Reward Wrapper Fix:** Adding logs for rewards to see if the agent is retrieving rewards and penalties correctly.

**Resolutions of Issues**

**Checkpoint Issue:** This was solved by changing the logic of how and what the model saved to the file.

**Reward Wrapper:** Correctly defining the rewards when calling upon the class

**Loss Stability:** This error fixed itself after a few episodes once the environment reset and wasn’t completely blacked out, returning to normal loss values.

**Environment Graphics:** This issue was unfixed but was managed via watching episodes of games in set periods due to the environment after long periods of the window being open, and not minimised, which would seem to result in graphical issues. There were also some incidents of a random blackout of graphics that affected the model's performance.

**Error handling**

Introducing error handling mechanisms, produced a clearer message about the error and easier to address the issues as soon as they became a problem.

## 4 Results

**4.1 General Performance Overview**

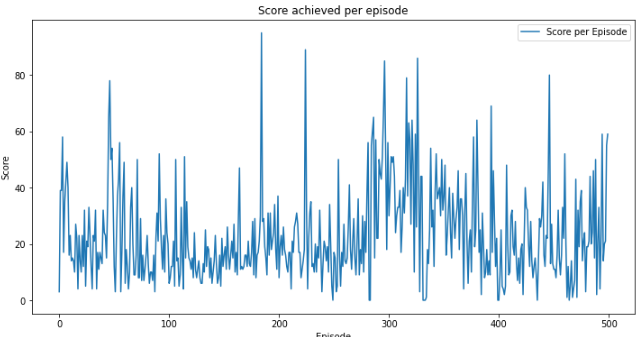
Evaluating the performance of the AI agent over the 7000 Tetris games, the model showed some improvement in scoring ability consistency and not so much improvement in the line-clearing department. The average score per episode was around an average of 45 in the first 1000 games to compared to the final 1000 games which had an average of 55-60 score points, showing that there was a process of learning and adaptation of some sort.

**Score Development:** Throughout the training, there were continuous fluctuations in scores obtained that were becoming a bit more consistent towards the end of training, with a score of around 120 being the maximum score obtained.

**Line Clearance Development:** Although the score obtained showed some improvements, line clearance did not follow the same improvement with there being random Tetris games where the agent would clear a line. The agent achieved the highest lines cleared around game 780 – 800, this indicates the agent exploring going for multiple lines in an episode which would have yielded the most reward throughout training.

**Challenges and Learning Curves:** The model struggled to keep a consistent performance throughout the training. This is demonstrated by the fluctuations of scores and sparse randomness of lines cleared in the scattered episodes

**4.2 Detailed Analysis**

**Games 1 – 1000**

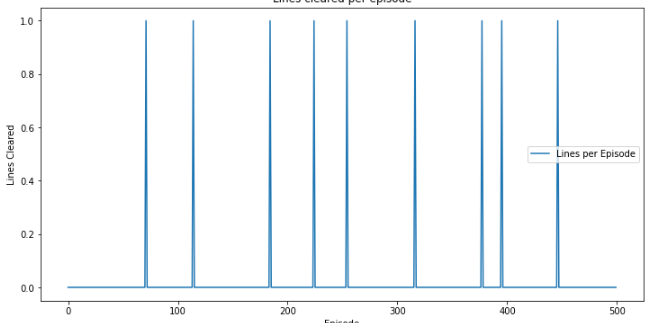
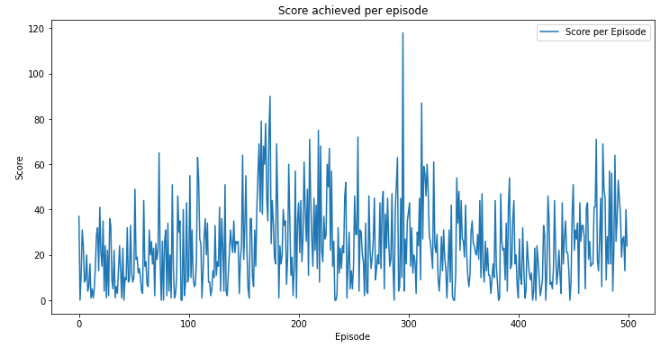
Games 1 – 1000 highlighted the model exploring the environment, with scores ranging from 0 – 120 and clearing 1 line in multiple instances and two lines in one instance (See Figure 14). This shows the model exploring the environment and is a promising indication of an effective strategy being developed.

Figure 11, Games 1 – 500 score

We can see the first 500 Tetris games there is steady exploration with trying to clear lines. Showing potential effective piece placement.

Figure 12, Games 1 – 500 lines



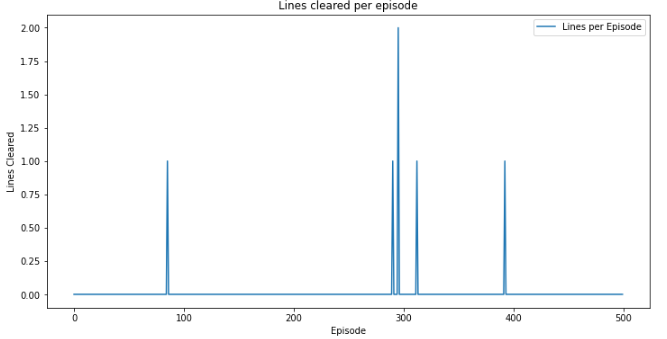
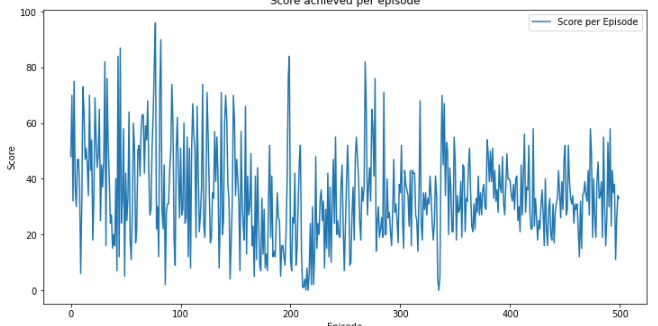
On the other hand, games 501 – 1000, show a slowdown in exploring the strategy to clear lines but with a little more fluctuation in the score, this was the Agent exploring different strategies at this point and seeing if the score was the better reward. Although we see fewer lines cleared in this half of the phase, we do see a game where the agent managed to clear 2 lines and this was also the top-scoring game of the first 1000 games, which would have yielded the most reward by far. Indicating there are still adjustments to the decision-making process to be made.

Figure 14, Games 501 – 1000 lines

Figure 13, Games 501 – 1000 score

**Games 1001 – 2000**

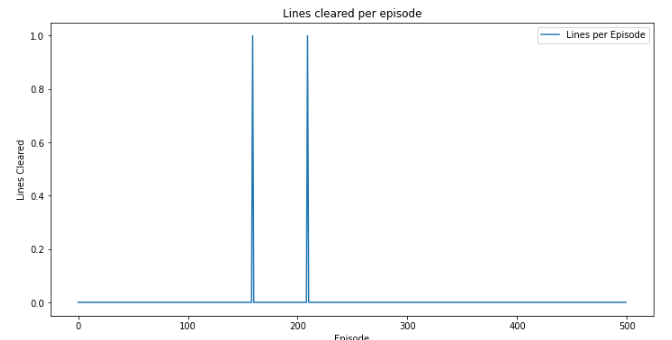
During the next 1000 games, 1001 – 2000, There again is a variation in the score obtained but is contained within the range of 0 – 100. This at this stage is most likely due to the randomness of the Tetris game but the games where we see some peak scores is demonstrating some of the AI model adaptability capabilities. We also see during this set of games that the agent achieved more consistency by producing more high-scoring games indicating some improvement in stack management

Figure 15, Games 1001 - 1500 score

Figure 16, Games 1001 – 1500 lines

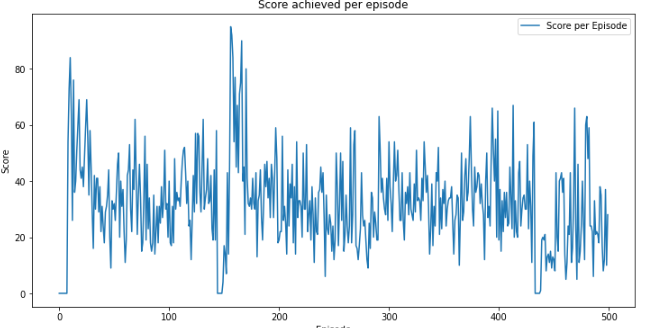
In the second half of these 1000 games, we can see that the agent started declining in the score obtained with a range between 0 – 80. Even hitting a score of 0 multiple times suggests new strategies are being used that aren’t working.

Figure 17, Games 1501 – 2000 score

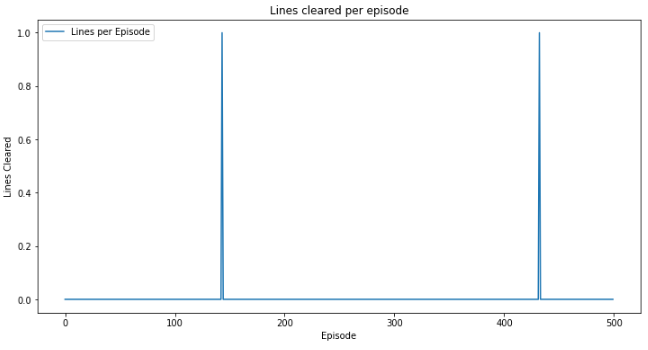
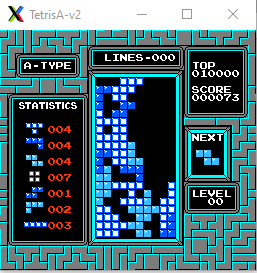


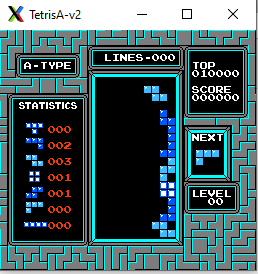
Figure 18, Games 1501 – 2000 lines

**Tetris Game 1012:**

This game shows the agent has the ability to manage to stack Tetris pieces, but it becomes uneven and high, and the agent is not looking to particularly clear lines. Shown by the gaps in stacking and stacking towards one side

Figure 19, Tetris Game 1012

**Tetris Game 1013 & 1657:**

**Game 1013:**

tetris game 1013 p2
This game here demonstrated a strategy change from the previous game, where the agent stacks up on one side to the top and then decides to start stacking in the middle and towards the left. Showing some strategy to try and survive as long as it can without clearing a line.

Figure 21, End of Tetris Game 1013

Figure 20, Start of Tetris Game 1013

**Game 1657:**

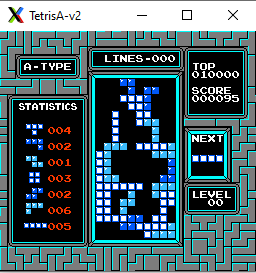
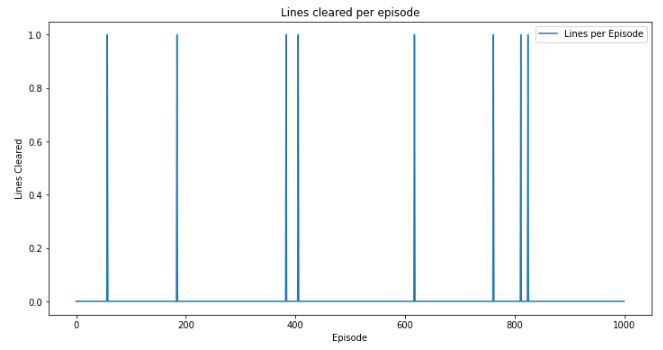
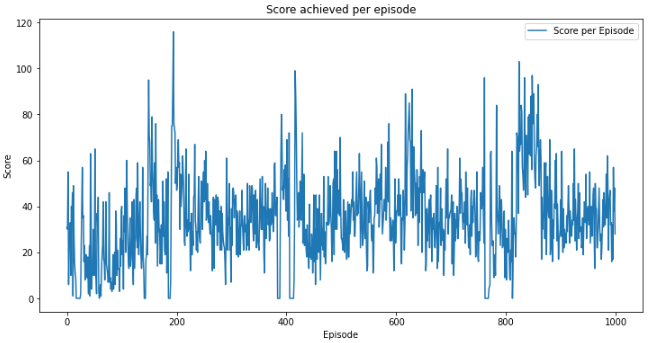
We can see in this game if the agent focused more on the reward for clearing a line it could of easily, especially in game 1657 (See Figure 22), clear a line or two receiving a reward of 100 points for each line. Which it thought was not the current optimal strategy for surviving and getting score points.

Figure 22, Tetris Game 1657

**Games 2001 – 3000**

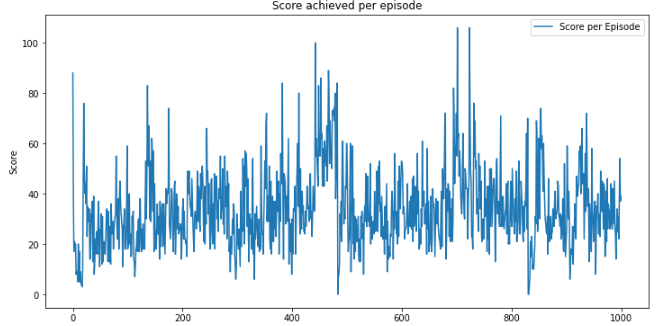
Figure 23, Games 2001 - 3000 score

During the games 2001 – 3000, there is again a lot of variation with more peak games getting closer to a score of 100 but still lying within the range of 20 – 80 points. There are no clear improvements but suggesting there is some degree of learning happening as the top score games are peaking higher and a little more often.

We also see that the agent has decided to look at a line-clearing strategy again with the occasional line being cleared in a few games. We can also see that where the game score peaks are, they seem to be around the same games as when a line is cleared. As we can see the agent is still struggling with clearing lines consistency.

Figure 24, Games 2001 - 3000 lines

No Tetris games were available during this session due to graphical issues rising early in the environment.

**Games 3001 – 5000**

Just like the previous 1000 games the games 3001 – 5000 show a lack of improvement to the model's ability to play Tetris, in fact games 3001 – 4000 to 4001 – 5000 show a decline in the model's ability to stay consistent and score points.

Figure 25, Games 3001 - 4000 score

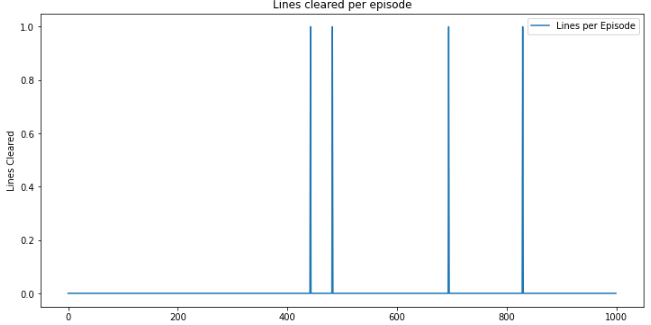
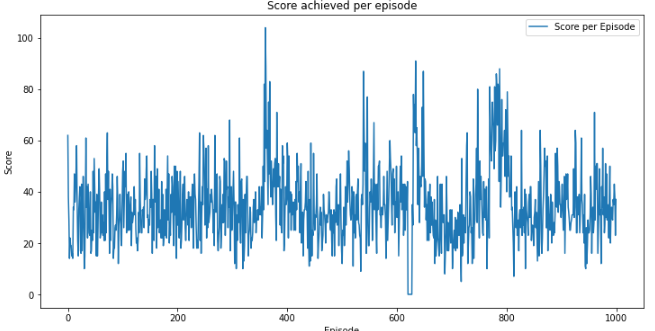
****

Figure 25, Games 3001 - 4000 lines

**Games 4001 – 5000**



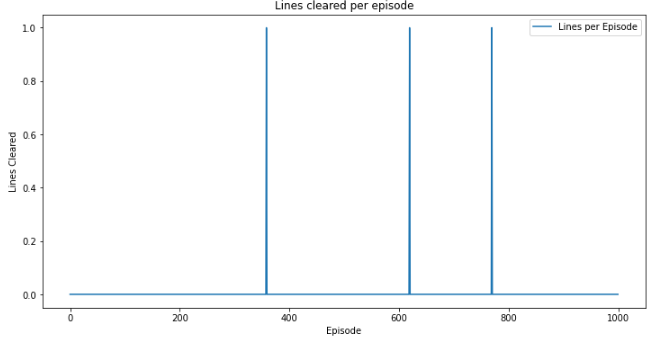
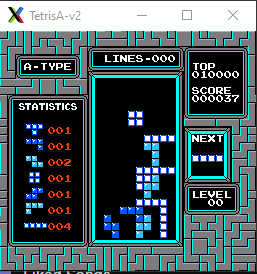


Figure 26, Games 4001 - 5000 score

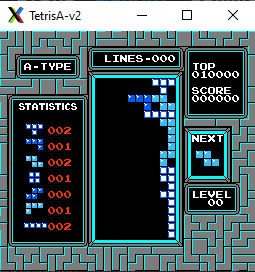
We can also see even the consistency in line clearance is declining again and showing no real improvements in the strategy being used to clear lines to try and survive longer.

Figure 26, Games 4001 - 5000 score

**Tetris Game 3586**

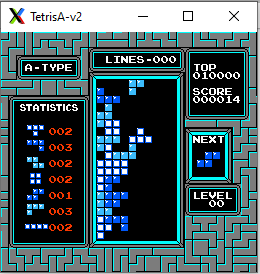
During game 3586, despite the poor performance in the graphs, it was, however, playing with the environment and performed a t-insert, highlighted in Figure 27 to fill in a gap. Showing some indication of stack management and reducing the gaps but does not focus on this as we can see from the rest of the environment state.

Figure 27, Tetris Game 3586 showing a t-insert

**Tetris Game 4365**

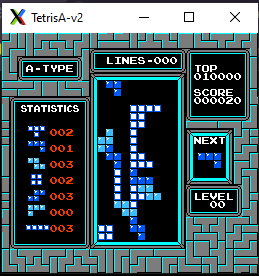
In game 4365 we see the agent revert back to single towers up the side and then building across the top, resulting in a short game and not a lot of reward points accumulated

Figure 28, Tetris Game 4365

**Tetris Game 4699**

In game 4699 we see the same approach as game 4365 but has built on the strategy making the tower thicker to survive for longer.

Figure 29, Tetris Game 4699

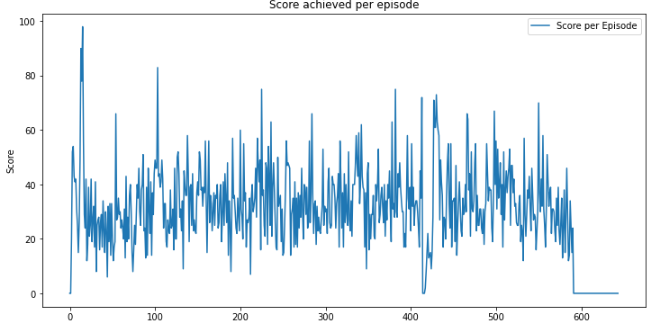
**Tetris Game 4859**

In game 4859 we see the agent going back to a strategy of skinny towers, showing no signs of development apart from building up from the side, the agent is building from the middle of the board

Figure 30, Tetris Game 4859

**Games 5001 - 6000**

**Tetris Games 5001 - 5645**



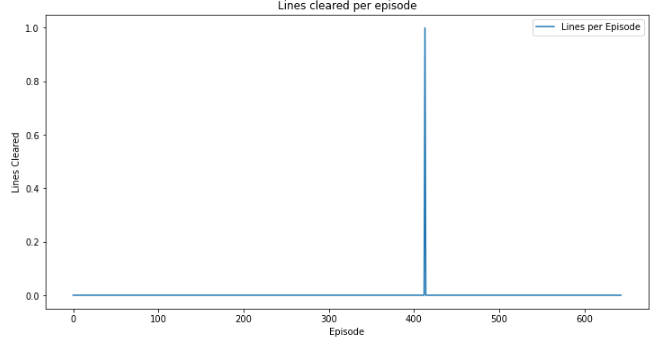
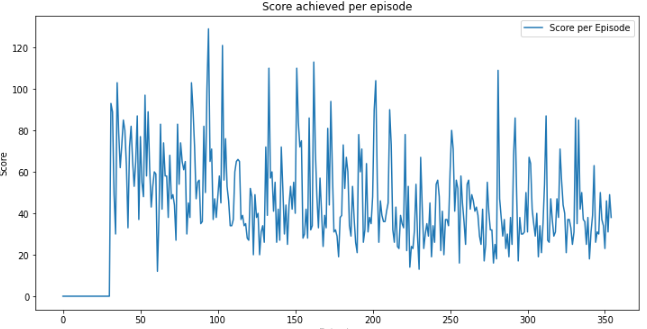
During this part of the training, we can see no improvements to score or lines, just a few peak spikes in score. However, it is worth noting that around game 600, there was a graphical issue with the environment that blackout the environment, and caused the loss after resetting the training period manually, to spike to values of 1000+, after a few games of Tetris the agent recovered and was back to normal on the loss calculations.

Figure 31, Games 5001 - 5645 score

Figure 31, Games 5001 - 5645 lines

**Tetris Games 5646 - 6000**

We can see at the start of the session that the issue had an impact on the performance of the agent getting a score of 0 for the first 30 games or so.

Another thing to note, after the graphical blackout, the agent on average started doing better in terms of score but did not clear a single line.

Figure 32, Games 5646 - 6000 score

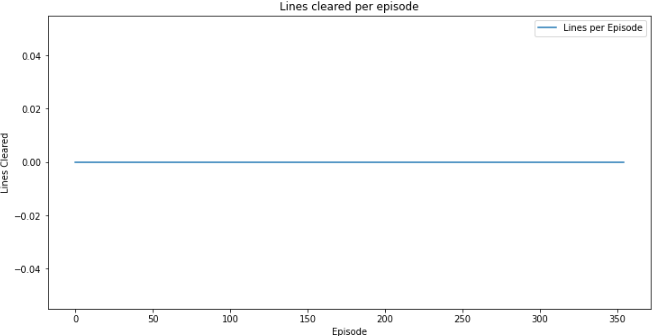


Figure 33, Games 5646 - 6000 lines

**Tetris Game 5296 Tetris Game 5414**

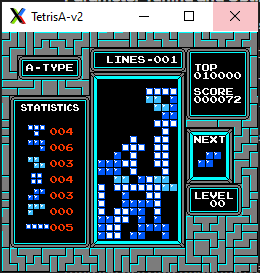
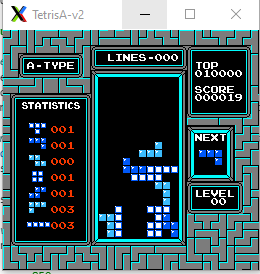


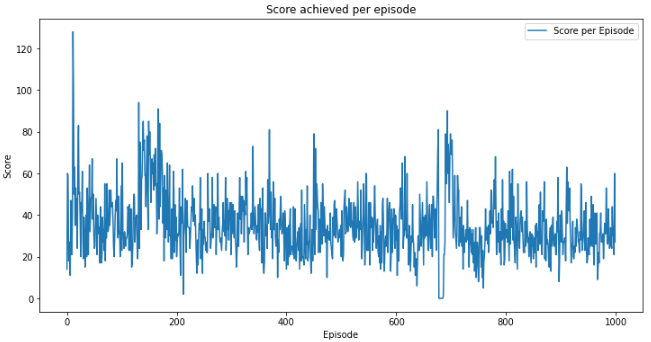
Figure 35, Tetris Game 5414

Figure 34, Tetris Game 5296

In both games 5296 and 5414, just before the graphical blackout, we can see a new strategy being tested, which appears to be creating a platform after some time.

Game 5296 does however show the agent has cleared a line but did not try going for a second one even with multiple opportunities before it covered the left side.

**Games 6001 – 7000**



In the final 1000 games, 6001 – 7000, we can see a slow decline in score performance and still 0 improvement in creating a strategy to clear lines.

Figure 36, Games 6001 - 7000 score

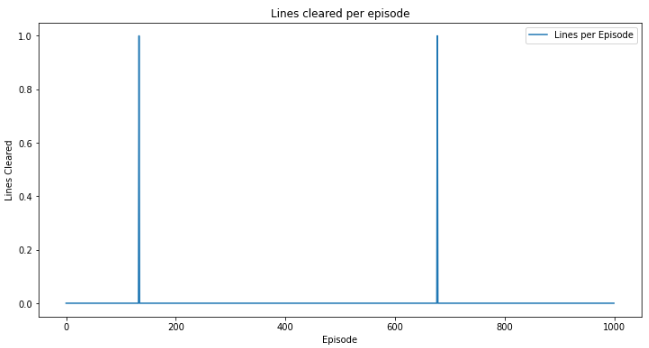
Around the time we see the agent clear a line, it is reflected in the score and around the time or after for a period of time the score is improved and then returns to low-scoring games again.

Figure 37, Games 6001 - 7000 lines

**Tetris Game 6029** **Tetris Game 6835**

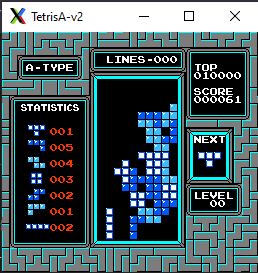
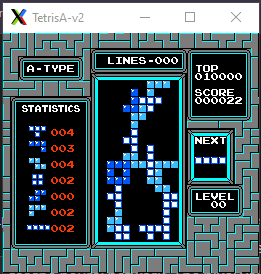


Figure 39, Tetris Game 6835

Figure 38, Tetris Game 6029

Here we can see two different strategies, one that is using a slightly more optimal piece placement in the grand scheme, 6029, but it is still not optimised placement to look into the future and clear lines. Whereas game 6835, shows very suboptimal placement and in fact creates a new platform where the agent had multiple pieces it could have used to clear it, but instead decided on a skinny tower upwards.

**4.3 Critical Incidents and Turning Points**

The progress of developing an AI model that is able to play Tetris showed a couple of incidents and turning points that changed the overall performance of the model. Here are some incidents and challenges highlighted that capture the model’s adaptability to such a complex learning environment during the training process.

**Games 780 - 800 Two Lines Cleared:** During the first 1000 games of training there was a huge turning point demonstrating, at early doors, that the AI is capable of clearing lines and creating some sort of strategy. Around games 780 – 800, we see a game where the agent was able to clear two lines in one game. This was the best-performing game in terms of line clearing.

**Game 1013 Strategy Shift:** The observation of game 1013 shows a strategy shift from the previous game 1012, one that was not positive in terms of Tetris gameplay, where the AI agent started stacking up one side with a skinny tower. This shows the capabilities of the AI that it can adapt and come up with new strategies whether they are effective or not.

**Graphical Blackout Games 5600 – 5700:** The biggest challenge that impacted the AI agent’s training sessions was a graphical blackout that happened around the games 5600 – 5700, this issue was solved by simply restarting the environment and starting a new training session. This caused the agent, temporarily to have a huge spike in loss values, reaching 1000+, causing a disruption in the training sessions for around 50 – 100 games. The agent was able to recover from this and tested the resilience of the AI’s learning algorithms and that the environment was more unstable than originally thought.

**Inconsistent Performance Games 3001 – 5000:** During the period of these games, the AI showed a decline in performance, in terms of score maximisation and line clearing. An issue with the tuning of the learning algorithm or that the agent is unable to properly develop a strategy within a complex environment like Tetris.

**Adaptive Learning:** The agent after the graphical blackout, showed recovery in strategy. It returned to the environment to achieve scores that were higher than the last 1000 Tetris games. The agent was still unable to refine the strategy into a line-clearing one demonstrated by the zero lines cleared in the period of games 5646 – 6000.

Understanding these turning points during the training gives valuable insights. It reflects some potential and limitations of the current algorithms when placed to learn in a complex environment like Tetris.

**4.4 Challenges and Limitations**

Creating a CNN and DQN hybrid model to play Tetris has presented a few challenges and limitations, especially at the average standard of a human.

**Model Limitations:** One limitation observed, that had a big impact, was the inconsistency of the AI at clearing lines, even when it gave two types of rewards for scoring points in-game & clearing a line. This shows a potential gap within the model’s ability to balance between the two rewards which are both already interconnected in Tetris. Although the agent attempted multiple strategies, the model struggled to create a strong line-clearing strategy. This could be an indication of the learning algorithm’s ability to learn from past game states effectively.

**Methodological Challenges:** Tuning the model’s hyperparameters to an optimal balance between exploration and exploitation proved to be challenging. This can be linked to the complexity and randomness of Tetris and predicting a long-term beneficial move to be complex. Although the agent had a high tilt towards exploration, it was still unable to effectively explore strategies. On top of this, maintaining stable learning caused a problem but this was highlighted by the loss values rising and episode performances not staying consistent.

**External Factors:** One of the training sessions was disrupted by graphical issues. The main graphical issue was a blackout around games 5600 – 5700, which led to a temporary sharp spike in loss values. Not only was training hindered but the training environment was not consistent due to graphical issues, restricting gameplay observation later in episodes.

Evaluation Limitations: The evaluation metrics focussed on the score and lines cleared, with the restructure of the rewards, more metrics could be included to capture the strategic depth and provide a better understanding of the model’s behaviour.

**Future Improvements:** Taking note of these challenges, a re-evaluation of the reward structure, better refinement of the learning algorithms and enhancements to the Tetris environment.

**4.5 Summary of Results**

**Score and Line Clearance Trends:**

**Score Improvement:** There was a small upward trend in scoring ability comparing the start to the finish, showing some ability to be able to achieve a higher score. This upward trend was stopped when the agent stopped improving, starting to decline hitting a score of 100 less often.

**Line Clearance:** Although the AI model showed some ability in scoring points, the model was incredibly inconsistent when it came to clearing lines. The highest number of lines cleared was achieved in the first 1000 games, around 780 – 800 but this performance was not enough to create the behaviour of clearing lines. Shows some possible limitations in balancing the two objectives of Tetris, even with a reward structure that was more focused towards clearing lines.

**Adaptability and Learning Progression**

**Exploration and Adaptation:**  The beginning of training was heavily focused on exploring how the game works with multiple strategies which the agent started to refine in later episodes. The model struggled to develop a strategy that was consistent and effective at clearing lines, showing some inability within the learning algorithm when using past game states to learn from.

**External Disruptions:** Training received disruptions, all related to graphical issues, one where the model had a spike in loss value, temporarily hindering learning progress. This shows the model’s resilience and ability to recover and continue learning after disruptions.

**Strategic Depth and Decision-Making:**

**Strategy Formulation:** During the training sessions, the AI showed the ability to test out different strategies and switch between them, which would be extremely useful in Tetris if you could apply them properly. The developed strategies unfortunately did not create consistency and did not improve the gameplay of the model, showcasing some possible challenges with the AI decision-making process within an environment like Tetris.

**Overall Learning Curve:** Overall, the learning curve shows improvement in some areas, like scoring game points but less improvement in areas like clearing lines and effective stacking. The randomness of the performance metrics indicates the model's ability to adapt to some aspects of Tetris but, still lacks the understanding needed to play effectively with the randomness of the environment.

## 5 Conclusion

The research has shown the complex yet feasible application of a hybrid model, using CNN and DQN, to play Tetris. During the 7000 games, the AI model showed some minor improvements, with its ability to score points, becoming slightly more consistent at achieving games with higher scores as the training progressed, but the AI struggled majorly when trying to clear lines, one of the core objectives of Tetris.

**Impactful Outcomes:**

**Learning Progression:** The model showed a slight learning curve, with the score fluctuating consistently but did have a slight improvement of achieving a slightly higher score on average, from the first 1000, 45, to the last 1000 games, around 55-60. Indicating some learning progression for a scoring strategy.

**Strategy Development:** Despite the AI model engaging in the initial exploration and attempting to adapt to the environment, the model still displayed a consistent struggle and was unable to develop an effective line-clearing strategy. This is clearly demonstrated by the scattered line clearances across all the Tetris games, potentially highlighting a limitation in strategic gameplay adaptation.

**Adaptability to Setbacks:** The model when faced with a setback was incredibly resilient and accidentally tested from an external disruption, the graphical blackout which impacted games 5600 – 5700, temporarily causing a huge spike in the loss value and calculations. However, the agent was able to recover and continue with the learning.

**Impact of Hyperparameter Tuning:** Due to the complex nature of Tetris and pairing this with challenges found in tuning hyperparameters like epsilon values for exploration and exploitation. This demonstrated some difficulty is keeping a balance between the immediate rewards and long-term strategy.

**Relation to Research Objectives:**

The objective of the research was to evaluate the effectiveness of a CNN-based DQN hybrid model when navigating the Tetris game environment. There was minor success in achieving this objective, shown by the models capable of improving scoring ability slightly but not capable of a consistent line-clearing strategy which is the game's primary objective.

Another objective was to evaluate how the model can adapt to the unpredictable, dynamic and, complex environment. The findings suggest the model can adapt its strategies to obtain more score points but the ability to pair scoring with a complex line-clearing strategy was underdeveloped.

**Implications of the Research**

This research highlights what the potential and limitations are when using deep learning models such as CNNs and DQN within gaming environments that require the ability to perform strategic planning and real-time decision-making. The insights provided by the AI’s performance and the challenges faced with a line-clearing strategy can contribute to other developments in reinforcement learning applications, especially with how the reward structures and learning algorithms can be optimised for a complex and dynamic game environment like Tetris.

## 6 Future Work

The progression of the AI model when learning to play Tetris has highlighted some potential oversights by me, that could help in further research and improvement of the model. Here are some points that I think could potentially enhance the performance of this AI model

**Enhanced Data Utilisation:** I believe a deeper analysis of the game state, achieved by retrieving game board data, such as each individual block cell on the Tetris game board, which was previously not achievable due to the ‘gym-tetris’ environment restricting access to board data. By providing more information about the game state and interpreting each cell differently from one another, the decisions by the AI model could be more informed as it has more information about the game state. Providing the model with a more detailed game state could enable the model to look at planning for the future more effectively.

**Reward System Reconstruction:** Currently the reward structure, which is focussed towards scoring points and clearing lines, may see some improvements to the AI model with a reconstruction. Although these two rewards are the main objective of Tetris, introducing some sub-objectives for the model could be beneficial and improve overall gameplay performance. Based on the ‘Enhanced Data Utilisation’ and getting access to the game board data, providing rewards for actions like not leaving a gap when placing a piece could help with optimising piece placement overall and in return should result in creating a strong strategy that clears more lines naturally and more reward points being accumulated.

**Learning Algorithm Optimisation:** The learning algorithm could benefit from a re-evaluation that enhances the ability to learn from previous game experiences more effectively. This could include adjusting the balance between exploration-exploitation more dynamically, based on the model’s performance. A technique of creating a simplified version of the Tetris game and gradually adding more Tetris pieces into the mix could prove to be beneficial as the model would then learn how to deal with each piece more effectively and how to integrate the more complex pieces into the strategy over time.

## 7 Reflection on Learning

At the beginning of this project, my knowledge and understanding of the AI subject was very minimal and basic, having an idea of the basics like reinforcement learning, neural networks etc. Now after interactions with people, I would say that I can explain how not only the foundations work but also the extra complicated components. This was not easy by any means needing a better understanding of frameworks like PyTorch.

The idea of AI systems has always been an exciting topic that I wanted to look into and being able to dive into such a topic that I could teach myself and create an AI that was able to learn to some extent was exciting. Now with the deeper knowledge I have gained about AI systems and the importance they are showing in the real world, gaining this experience will only help me towards a career in AI if the opportunity comes.

During the coding process, there was a lot of looking at PyTorch documents and learning about the different neural network structures which took some time to understand, especially creating a hybrid AI model and how to interlink the models and all the equations for topics like Huber loss, weight and biases, etc. but as I progressed through the project these equations became easier to understand.

One of the hardest parts of this project was when tuning the hyperparameters. When performing this stage of the project, I did not realise how hard it would be to optimise the parameters of the neural network. While observing gameplay or evaluating the metrics to gain some insight, it was often unclear what parameters needed tuning. The difficulty of creating a hybrid neural network and how time-consuming tuning parameters on a small scale has provided an understanding of the amount of tweaking some of these bigger AI models have gone through before the models start learning effectively.

Overall, this was a challenging project from start to finish but has provided me with some fun, and more interest in the topic of AI and has considerably improved my knowledge of AI systems. This project has also made me want to improve on the current model, and potentially create my own Tetris environment for it to train within and has made me feel more comfortable if I were to apply for a job in the AI field.

## References

Anon 2024a. Classic Tetris World championship [Online] Available at: https://en.wikipedia.org/wiki/Classic\_Tetris\_World\_Championship#:~:text=The%20contestants%20play%20the%201989,display%20to%20improve%20viewer%20experience [Accessed: 10 May 2024].

Anon 2024b. Tetris [Online] Available at: https://en.wikipedia.org/wiki/Tetris [Accessed: 10 May 2024].

Anon 2024c. StackRabbit: Shattering Tetris Records & Embracing the Future [Online] Available at: https://www.toolify.ai/ai-news/stackrabbit-shattering-tetris-records-embracing-the-future-1731802#:~:text=of%20the%20game.-,How%20Does%20StackRabbit%20Work%3F,based%20on%20all%20possible%20pieces [Accessed: 10 May 2024a].

Anon. *Simple Reinforcement Learning Loop* . [Image] Available at: https://ekababisong.org/key-concepts-of-modern-reinforcement-learning/ [Accessed: 2 May 2024b].

Berg, E. *Markov Decision Process*. [Image]

<https://optimization.cbe.cornell.edu/index.php?title=Markov_decision_process>[Accessed: 2 May 2024]

Gibaru, O. *Definition of a neuron*. [Image]

<https://www.oliviergibaru.org/courses/ML_NeuralNetwork.html>[Accessed: 3 May 2024]

Neutelings, I. *Neural Network Structure*. [Image]

<https://tikz.net/neural_networks/> [Accessed: 2 May 2024]

Neutelings, I. *Convolutional Neural Network Structure*. [Image]

<https://tikz.net/neural_networks/> [Accessed: 2 May 2024]

Orland, K. 2024. 34 years later, a 13-year-old hits the NES tetris ‘kill screen’ [Online] Available at: https://arstechnica.com/gaming/2024/01/someone-has-finally-beaten-nes-tetris/ [Accessed: 10 May 2024].

Python Lessons. Epsilon Greedy strategy in Deep Q Learning. [Image]

Available at: https://www.youtube.com/watch?v=HNVlAvwEvPw [Accessed: 4 May 2024].

Park, J.-H. et al. 2022. Deep reinforcement learning-based DQN agent algorithm for Visual Object Tracking in a virtual environmental simulation [Online] Available at: https://www.mdpi.com/2076-3417/12/7/3220 [Accessed: 10 May 2024].

Ramos, J. 2019. Tetris Advanced Guide: T-spins, Perfect clears, and combos [Online] Available at: https://www.polygon.com/guides/2019/3/6/18244590/tetris-t-spin-perfect-clear-combo-guide [Accessed: 10 May 2024].

Stevens, M. and Pradhan, S. [Online] Available at: https://cs231n.stanford.edu/reports/2016/pdfs/121\_Report.pdf [Accessed: 10 May 2024].

Utic, Z. 2018. *ε-greedy action selection* . [Image]

<https://www.researchgate.net/publication/325049488_Off-Policy_Q-Learning_Technique_for_Intrusion_Response_in_Network_Security>[Accessed: 2 May 2024]

waldoalvarez 2017. *Example of a Markov Decision Process*. [Image]

<https://commons.wikimedia.org/wiki/File:Markov_Decision_Process.svg>[Accessed: 2 May 2024]

Wang, Y. et al. 2021. *Q-learning vs. DQN*. [Image]

[https://www.mdpi.com/1999-5903/13/10/261#](https://www.mdpi.com/1999-5903/13/10/261)[Accessed: 2 May 2024]

Zwiezen, Z. 2021. Nes Tetris players call it ‘rolling,’ and they’re setting New World Records [Online] Available at: https://kotaku.com/nes-tetris-players-call-it-rolling-and-theyre-setting-1846767518 [Accessed: 10 May 2024].