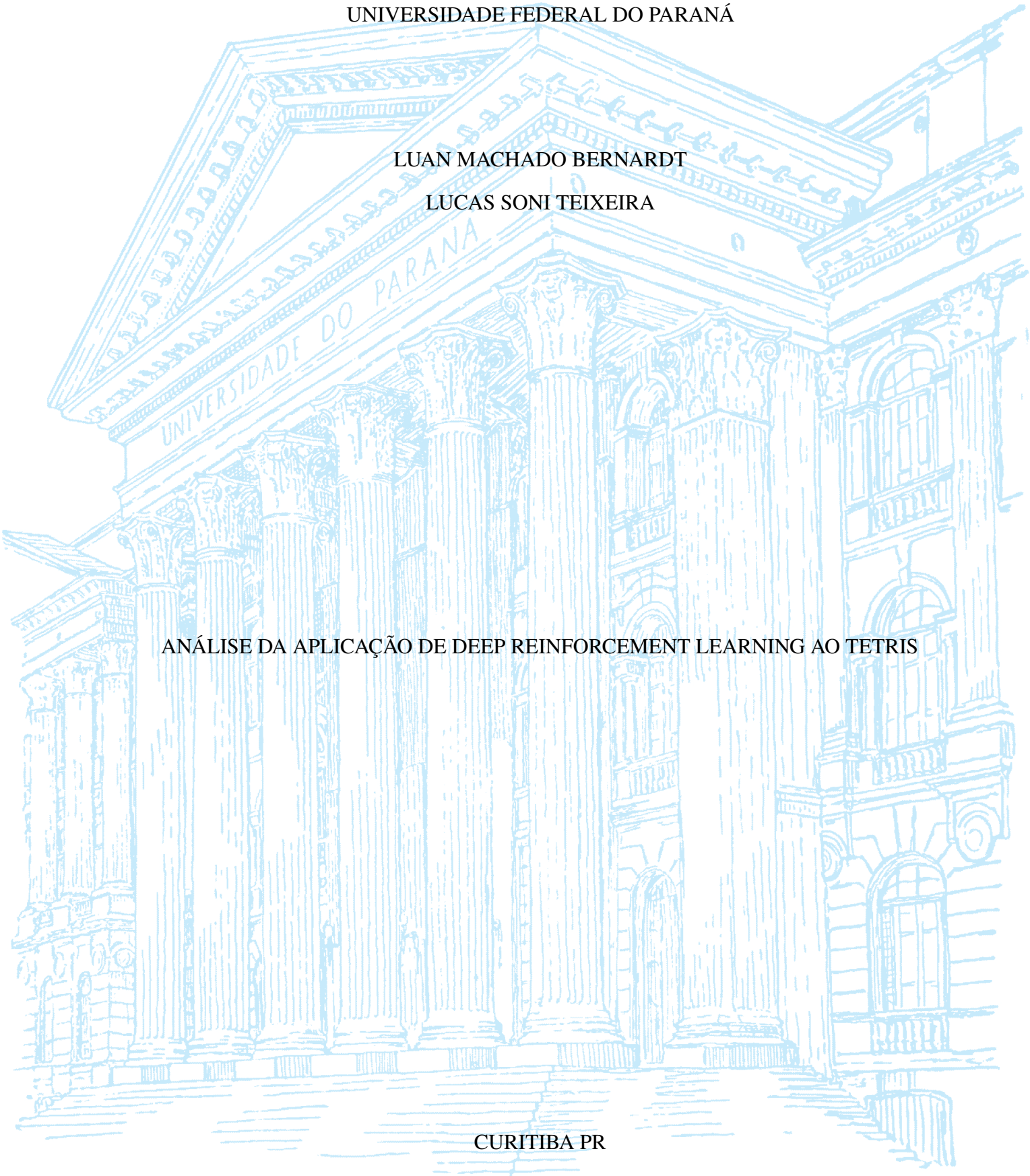
UNIVERSIDADE FEDERAL DO PARANÁ

LUAN MACHADO BERNARDT

LUCAS SONI TEIXEIRA



ANÁLISE DA APLICAÇÃO DE DEEP REINFORCEMENT LEARNING AO TETRIS

CURITIBA PR

2025

LUAN MACHADO BERNARDT
LUCAS SONI TEIXEIRA

ANÁLISE DA APLICAÇÃO DE DEEP REINFORCEMENT LEARNING AO TETRIS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Eduardo Jaques Spinosa.

CURITIBA PR

2025

AGRADECIMENTOS

Gostaríamos de expressar uma profunda gratidão ao Prof. Dr. Eduardo Jacques Spinosa, nosso querido orientador, por ter nos emprestado a sua vasta experiência e gastado o seu precioso tempo, nos guiando com paciência e compreensão durante esse árduo processo.

A nossos familiares e amigos, agradecemos por sempre terem nos apoiado durante essa etapa, do início ao fim. Sempre nos incentivando a chegar ao final dessa caminhada.

RESUMO

Este trabalho investiga a aplicação de Aprendizado por Reforço Profundo (Deep Reinforcement Learning), especificamente com o algoritmo Deep Q-Network (DQN), para analisar a adaptabilidade de um agente autônomo no jogo Tetris. O objetivo central é compreender como o desempenho e a convergência do aprendizado são impactados por alterações nos parâmetros fundamentais do ambiente, como a topologia do tabuleiro e a estrutura da função de recompensa. Para isso, o estudo parte de um cenário de controle (baseline), um pouco diferente do cenário convencional do tetris, com um tabuleiro de dimensões 20x10, onde a pontuação é calculada de forma quadrática e a informação da próxima peça não é visível para o agente. A partir deste ponto, o desempenho do agente é sistematicamente comparado em cenários experimentais distintos, que incluem a alteração das dimensões do tabuleiro para 20x20 (tornando-o mais largo) e 30x10 (tornando-o mais alto), bem como a modificação da função de recompensa para um modelo linear. A análise comparativa dos resultados demonstrou que a topologia do ambiente é o fator mais crítico para o aprendizado, onde o aumento da altura do tabuleiro (30x10) se mostrou benéfico, enquanto o aumento da largura (20x20) prejudicou significativamente o desempenho. Concluiu-se que, embora o agente se adapte, sua estratégia fundamental de limpezas de linha única permanece robusta em todos os cenários, sendo apenas sutilmente aprimorada pela função de recompensa linear.

Palavras-Chave: Redes neurais, aprendizado por reforço, Tetris, inteligência artificial, jogos digitais.

ABSTRACT

This work investigates the application of Deep Reinforcement Learning (DRL), specifically using the Deep Q-Network (DQN) algorithm, to analyze the adaptability of an autonomous agent in the game of Tetris. The main objective is to understand how performance and learning convergence are impacted by alterations to the environment's fundamental parameters, such as the board's topology and the reward function's structure. To this end, the study starts from a control scenario (baseline), which is slightly different from the conventional Tetris setup, featuring a 20x10 board, a quadratic scoring function, and no next-piece visibility for the agent. From this reference point, the agent's performance is systematically compared across distinct experimental scenarios, including altering the board dimensions to 20x20 (making it wider) and 30x10 (making it taller), as well as modifying the reward function to a linear model. The comparative analysis of the results demonstrated that the environment's topology is the most critical factor for learning, where increasing the board's height (30x10) proved beneficial, while increasing its width (20x20) significantly hindered performance. It was concluded that although the agent adapts, its fundamental strategy of single-line clears remains robust across all scenarios, being only subtly improved by the linear reward function.

Keywords: Neural networks, reinforcement learning, Tetris, artificial intelligence, digital games.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DQL	*Deep Q-Learning*
DRL	Aprendizado por Reforço Profundo, em inglês *Deep Reinforcement Learning*
FA	Função de Ativação
IA	Inteligência Artificial
RL	Aprendizado por Reforço, em inglês *Reinforcement Learning*
RN	Rede Neural Artificial

SUMÁRIO

1 INTRODUÇÃO

Os jogos sempre foram um excelente campo de testes para a inteligência artificial (IA), servindo como uma plataforma para desenvolver e avaliar novos algoritmos. O Tetris é um caso clássico: um jogo com regras simples, mas com uma profundidade estratégica que o torna um desafio para uma IA. É nesse cenário que este trabalho se desenvolve, utilizando o Tetris para explorar o aprendizado de um agente utilizando Aprendizado por Reforço Profundo (*Deep Reinforcement Learning* - DRL).

Normalmente, vemos agentes de IA sendo treinados para atingir a performance máxima em um ambiente com regras fixas. A motivação deste trabalho, no entanto, foi um pouco diferente: o que acontece quando as "regras do mundo" mudam? Em vez de apenas criar um jogador de Tetris que alcança altas pontuações, o foco foi analisar como ele se comporta ao ser desafiado com variações no próprio ambiente em que joga.

O objetivo deste TCC, portanto, foi investigar a adaptabilidade de um agente baseado no algoritmo *Deep Q-Network* (DQN) (Mnih et al. (2013)) quando confrontado com diferentes versões do ambiente do Tetris. Para isso, o projeto seguiu alguns passos centrais:

- Primeiro, desenvolvemos um agente de referência, treinado para jogar em um cenário base que definimos (um tabuleiro de 20x10, com um sistema de pontuação quadrático).
- Em seguida, testamos o desempenho deste mesmo agente em ambientes modificados, um com o tabuleiro mais largo (20x20), outro mais alto (30x10), e um cenário final com uma regra de pontuação linear.
- Por fim, comparamos os resultados de cada experimento para entender como essas mudanças no ambiente influenciaram a capacidade de aprendizado e a estratégia final do nosso agente.

Este documento foi organizado para guiar o leitor através de todas as etapas desta investigação. O Capítulo 2 estabelece o referencial teórico, cobrindo os conceitos essenciais do jogo Tetris, do Aprendizado por Reforço e das arquiteturas de Redes Neurais utilizadas. Em seguida, o Capítulo 3 detalha os Materiais e Métodos, descrevendo a implementação do agente, a definição do ambiente e a metodologia experimental. No Capítulo 4, são apresentados e discutidos os resultados obtidos em cada um dos cenários testados, com uma análise comparativa do desempenho do agente. Por fim, o Capítulo 5 encerra este trabalho com as conclusões gerais, as limitações encontradas e sugestões para futuras investigações na área.

2 REFERENCIAL TEÓRICO

2.1 TETRIS

O jogo Tetris foi criado em meados dos anos 80 pelo engenheiro soviético Alexey Pajitnov. Inspirado por um quebra-cabeça chamado "Pentominos", Tetris é um jogo de encaixe, no qual deve-se posicionar peças geométricas chamadas de *tetrominos*, de modo que todas as posições de uma linha estejam ocupadas por blocos. As peças possuem formas variadas (Figura 2.1) e são compostas sempre por quatro blocos, podendo ser rotacionadas e encaixadas na base do campo de jogo.

A jogabilidade do Tetris se concentra no jogador manipular as peças em descenso no campo de jogo, que é uma matriz vertical, em sua versão padrão, de 20 linhas por 10 colunas. Essas peças surgem no topo do tabuleiro de maneira centralizada e gradualmente "cae", isto é, têm seu valor no eixo Y (vertical) subtraído. Os movimentos aos quais uma peça pode ser submetida são um conjunto discreto, sendo alguns deles:

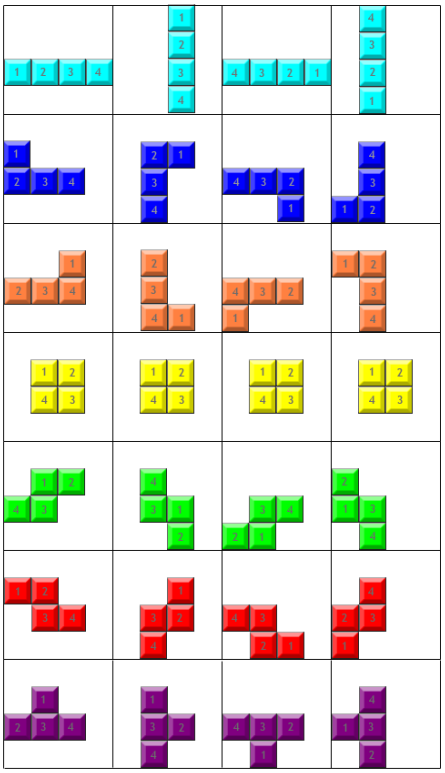
> **Horizontal**: A peça pode ser movida uma célula da matriz para a esquerda ou para a direita, caso haja pelo menos uma célula livre para o lado escolhido.
>
> **Rotação**: A peça pode ser rotacionada em quatro angulações diferentes, 0º (direção inicial da peça), 90º, 180º e 270º, desde que haja espaço suficiente em todas as direções para acomodar a nova disposição dos 4 blocos do *tetromino*.
>
> **Descenso**: Posicionando a peça na posição da matriz com o menor valor no eixo Y (vertical) o possível, até que esteja em contato em sua face inferior com outra peça ou com a borda da matriz, a posição no eixo X (horizontal) e a rotação são mantidas. Essa classe de movimentos não foi implementada em nossos experimentos, pois possuem impacto significativo apenas em jogadores humanos, tendo em vista que sua única funcionalidade é acelerar visualmente o jogo.

Tanto os movimentos horizontais quanto os rotacionais são sempre seguidos de um efeito de gravidade, responsável por subtrair a altura da peça na matriz em uma unidade. Apenas um desses movimentos pode ser realizado por quadro do jogo.

O objetivo do Tetris é o posicionamento dos *tetrominos* de maneira a preencher completamente uma ou mais linhas horizontais. Quando uma linha é preenchida, ela é eliminada, e as linhas superiores descem para preencher o espaço vazio. O jogo termina, com a derrota do jogador, apenas quando o campo de jogo é preenchido com um ou mais blocos em uma posição do topo, impedindo a movimentação de novas peças.

A geração de peças utiliza um método de randomização conhecido como *7-bag* ou *bag-style*. Este algoritmo funciona de maneira análoga a um sorteio sem reposição: as sete peças únicas (*tetrominos*) são colocadas em uma "sacola" (*bag*) virtual. As peças são então sorteadas aleatoriamente desta sacola, uma a uma, até que ela fique vazia. Somente após todas as sete peças terem sido utilizadas, a sacola é reabastecida com as sete peças novamente, e o processo de sorteio se repete. Esse método garante que o jogador receba uma distribuição equilibrada, evitando longas sequências de uma única peça e assegurando que, em um intervalo de 14 sorteios, cada tipo de peça apareça no mínimo uma e no máximo duas vezes.

Figura 2.1: Possíveis formatos e rotações dos *tetrominos*.



Em sua versão original[1], a pontuação do jogador é calculada pelo somatório das distâncias de queda de cada peça, ou em caso de eliminação de linha, do bônus recebido por limpar 1, 2, 3 ou 4 linhas simultaneamente (sendo o último chamado de Tetris), cujos valores são respectivamente 100, 300, 500, 800. A distância de queda é a diferença entre a altura do tabuleiro e a posição na matriz da célula mais alta pertencente à peça.

2.1.1 Complexidade e Estratégia

Tetris é um jogo com uma dificuldade crescente. A medida que o jogador avança, a velocidade de queda das peças aumenta, exigindo reações mais rápidas e melhor planejamento. Demaine et al. (2003) mostraram que Tetris é um problema computacional completo, relacionado à teoria da NP-completude, onde maximizar o número de linhas completas, o total de linhas eliminadas simultaneamente e a quantidade de peças inseridas antes do fim do jogo são problemas NP-completos.

Estratégias de jogo envolvem o gerenciamento eficiente do espaço, a previsão de peças futuras e o posicionamento cuidadoso para evitar a criação de lacunas, especialmente nas linhas inferiores do campo de jogo, e aumentar a altitude das peças posicionadas, o que acarretaria em menos espaço para manusear e posicionar as seguintes.

2.2 APRENDIZADO POR REFORÇO (*REINFORCEMENT LEARNING*)

O aprendizado por reforço (RL, do Inglês *Reinforcement Learning*) é uma vertente do aprendizado de máquina especializada em como um agente aprende a tomar decisões, utilizando

[1]A versão oficial do Tetris, que serviu como base para este trabalho pode ser encontrada no site: https://play.tetris.com/

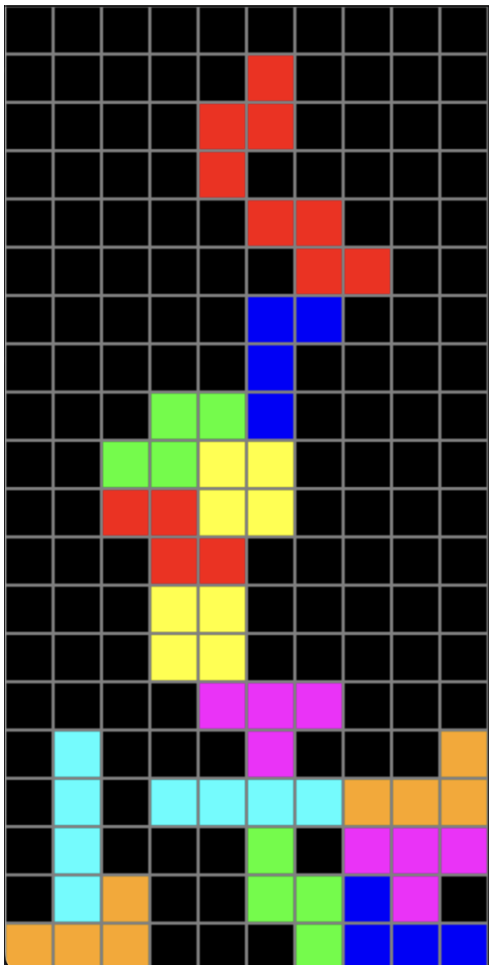


Figura 2.2: Imagem de um jogo em andamento representando as peças em campo e o tabuleiro

a abordagem de tentativa e erro com o objetivo de maximizar recompensas cumulativas. Esse processo permite à máquina aprender, interagindo com um ambiente incerto e potencialmente complexo, recebendo *feedback* em forma de recompensas ou punições, baseadas nas suas ações (IBM, 2024). O RL é eficiente para problemas de decisão sequenciais em cenários desconhecidos, sendo extremamente útil para aplicações do mundo real, onde os resultados são não determinísticos e dinâmicos, podem variar de acordo com o tempo. (Li, 2019)

O processo de aprender por reforço gira em torno da ideia de que um agente interage com algum ambiente para alcançar um objetivo, efetuando ações e recebendo um retorno para otimizar e aprimorar sua capacidade de tomada de decisão com o passar do tempo. Este processo é frequentemente ilustrado pelo ciclo agente-ambiente, como demonstrado na Figura 2.3, onde o agente toma ações que modificam o ambiente e recebe em troca informações sobre o novo estado e uma recompensa.

2.2.1 Conceitos-chave

Os elementos básicos do funcionamento RL, os quais integram o fluxo são de acordo com Sutton e Barto (2018) e IBM (2024):

> **Agente**: O tomador de decisões, aquele que vai aprender, realiza ações e interage com o ambiente em que foi inserido. É capaz de fazer escolhas e lidar com os resultados sem instrução direta de um agente humano.

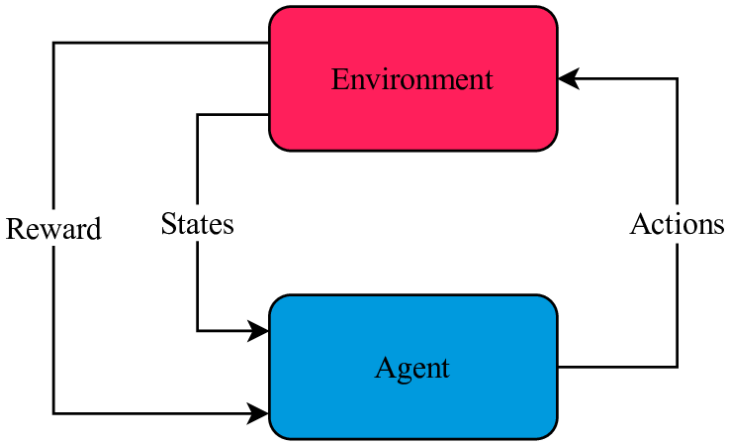


Figura 2.3: O ciclo de interação do Aprendizado por Reforço. (Benjamin Remman, 2021)

Ambiente: Cenário com o qual o agente interage. Fornece informações a respeito de seu estado atual e responde às ações do agente.

Ação: Todos os possíveis movimentos que o agente pode tomar com base no estado atual. Espaço de ações denomina todas as ações disponíveis para o agente em dado estado, pode ser discreto, quando há um número máximo de ações possíveis, como é o caso do Tetris (mover para os lados e rotacionar), ou contínuo, quando existem infinitas possibilidades, por exemplo, a movimentação de um robô, que pode mover-se para todas as direções em qualquer angulação.

Estado: Uma instância específica do ambiente em que o agente se encontra em dado momento. É utilizado pelo agente para decidir qual a próxima ação a tomar.

Recompensa: *Feedback* do ambiente baseado na ação tomada. Podem ser positivas ou negativas, respectivamente encorajando e punindo a decisão, guiando o agente em direção ao objetivo.

Além desses conceitos, existem ainda outras importantes definições do RL, que guiam e influenciam diretamente na capacidade de tomada de decisão do agente, sendo elas:

Política: A estratégia usada pelo agente para determinar a próxima ação baseada no estado atual. Pode variar de uma simples função até um processo computacional mais complexo, sempre mapeando estados do ambiente a uma das ações disponíveis no espaço.

Função de Recompensa: Uma função que retorna um sinal de *feedback* escalar baseado no estado após uma ação.

Função de Valor: Uma estimativa das recompensas futuras que um agente pode esperar receber de um dado estado ou de uma ação tomada em um dado momento. Em contrapartida à função de recompensa, que calcula uma recompensa imediata, a função de valor aponta o ganho a longo prazo, considerando todos os possíveis estados subsequentes e suas respectivas recompensas. Deve-se buscar maximizar esse valor.

Modelo do Ambiente: Uma representação de um ambiente que ajuda a planejar, prevendo estados e recompensas futuras.

2.2.2 Como o Agente Aprende

O aprendizado do agente ocorre através de um ciclo de interação contínuo e dinâmico com o ambiente. Este processo não é apenas um conjunto de partes distintas, mas um sistema de feedback profundamente interdependente, onde a eficácia depende do alinhamento de todos os componentes. O ciclo pode ser detalhado no seguinte passo a passo:

1. **Observação do Estado:** O ciclo se inicia com o agente observando o estado atual do ambiente, que chamaremos de S_t. Em nosso caso, isso seria a configuração do tabuleiro do Tetris em um dado momento.

2. **Seleção da Ação:** Com base no estado S_t e em sua política atual (sua estratégia de tomada de decisão), o agente escolhe uma ação A_t do conjunto de todas as ações possíveis. No início do treinamento, essa escolha pode ser aleatória (exploração), mas com o tempo, o agente aprende a escolher ações que ele acredita que levarão a melhores resultados (explotação).

3. **Interação com o Ambiente:** O agente executa a ação A_t no ambiente. Por exemplo, move o tetromino para a esquerda.

4. **Recebimento de *Feedback*:** Após a ação, o ambiente transita para um novo estado, S_{t+1}, e retorna ao agente um *feedback* numérico, a recompensa R_{t+1}. Essa recompensa avalia quão boa foi a ação A_t tomada no estado S_t. Uma linha completa pode gerar uma recompensa positiva, enquanto criar um buraco pode gerar uma recompensa negativa (punição).

5. **Atualização do Conhecimento:** Este é o passo crucial do aprendizado. O agente utiliza o resultado da sua interação — a tupla de experiência $(S_t, A_t, R_{t+1}, S_{t+1})$ — para atualizar seu conhecimento interno. Ele ajusta sua política ou sua função de valor para que, no futuro, quando se encontrar no estado S_t novamente, seja mais provável que escolha uma ação que leve a uma recompensa maior.

Este ciclo de observar → agir → receber *feedback* → aprender se repete milhares ou milhões de vezes. A cada iteração, o agente refina sua estratégia com o objetivo de maximizar a soma total de recompensas que ele espera receber, tornando-se progressivamente mais competente na tarefa. O sucesso desse processo, contudo, depende criticamente do design da função de recompensa, pois um sistema de incentivos mal projetado pode levar a comportamentos indesejados, mesmo com um algoritmo de aprendizado sofisticado.

Um dos desafios que surgem no RL, e não em outros tipos de aprendizado de máquina, é o dilema entre exploração e explotação. Para maximizar sua recompensa, um agente deve preferir ações que já tentou no passado e delas obteve resultados lucrativos. Por outro lado, para descobrir novas ações possivelmente até melhores, ele precisa tentar ações que não foram previamente selecionadas. O agente precisa explorar o que já sabe para obter recompensa, mas também precisa explorar para fazer melhores escolhas no futuro (Sutton e Barto, 2018).

2.3 *Q-LEARNING*

É um algoritmo do Aprendizado por Reforço cuja função é treinar o agente para qualificar suas possíveis ações baseando-se no estado atual. Tem como um de seus objetivos encontrar a política ótima que maximiza o total de recompensa acumulada após sucessivos passos (Watkins e Dayan, 1992).

2.3.1 Função Q-valor

A função Q-valor representa a recompensa total acumulada obtida ao executar uma ação A em um estado S. Essa função mapeia um par ação-estado a um valor numérico que significa o valor a longo prazo de A executada em S. Esse valor é utilizado pelo agente para decidir qual ação tomar para maximizar sua recompensa no futuro.

Essa recompensa futura é calculada recursivamente utilizando a equação de Bellman (Spano et al., 2019):

$$Q(S_t, A_t) = (1 - \alpha) \cdot Q'(S_t, A_t) + \alpha \cdot (R_{t+1} + \gamma \cdot max_a Q(S_{t+1}, a)) \tag{2.1}$$

Onde:

$Q(S_t, A_t)$: O Q-valor atual para o par ação-estado (S_t, A_t).

α: A taxa de aprendizado, um valor $(0 \leq \alpha \leq 1)$ que controla o quanto uma nova informação influencia no Q-valor atual. Quanto mais perto de um, mais peso possuem os novos aprendizados.

R_{t+1}: O *feedback* imediato do ambiente após executar a ação A_t no estado S_t.

γ: O fator de desconto, um valor $(0 \leq \gamma \leq 1)$ que determina a importância de recompensas futuras. Um γ próximo de 0 torna o agente imediatista, se preocupando apenas com recompensas precoces.

$max_\alpha(S_{t+1}, a)$: A maior recompensa futura esperada que pode ser obtida no próximo estado S_{t+1} tomando a ação a.

A equação de Bellman, portanto, serve como a regra de atualização que guia o aprendizado do agente. Em um ciclo contínuo, o agente executa uma ação, observa a recompensa e o novo estado, e então utiliza essa equação para refinar sua estimativa sobre a qualidade daquela ação (o Q-valor). Com o tempo, os Q-valores convergem, permitindo que o agente simplesmente escolha a ação com o maior valor Q em qualquer estado para maximizar sua recompensa.

2.4 REDES NEURAIS

Redes Neurais ou Redes Neurais Artificiais são uma classe de modelos computacionais desenvolvidos para mimetizar o funcionamento do cérebro humano em tarefas como processar e organizar informação, detectar padrões. São compostas por múltiplas camadas interconectadas de unidades de processamento, chamadas de neurônios, que processam uma entrada e geram uma saída.

A unidade básica de uma rede neural artificial é um neurônio. Cada neurônio recebe um conjunto de valores como entrada, executa uma operação, normalmente a soma ponderada desses valores mais um valor de viés (*bias*), resultando em um único valor de saída (Hagan et al., 1996).

2.4.1 Conceitos chave

Uma RN possui três tipos de camadas:

Camada de Entrada: primeira camada, por onde os dados entram na rede.

Camadas Ocultas: são as camadas situadas entre a camada de entrada e a de saída, nelas ocorrem a maior parte do processamento dos dados, formam o núcleo de uma rede neural artificial. Os neurônios nelas situados recebem um conjunto de pesos e produzem uma saída aplicando uma função de ativação. Uma RN pode ser composta de várias camadas ocultas totalmente conectadas entre si, isto é, um neurônio de uma camada recebe *inputs* de todos os neurônios da anterior.

Camada de Saída: retorna o resultado do processamento das camadas internas. É a última camada de uma rede.

O fluxo e a transformação dos valores de entrada são influenciados por três parâmetros essenciais:

Pesos: se referem a força com a qual duas unidades de processamento estão conectadas. Determinam a influência que os valores de entrada possuem no resultado. Alterações nesses valores impactam diretamente na performance da RN.

Biases: são valores utilizados para ajustar o valor de saída de um neurônio, permitindo à rede aprender padrões mais complexos que não seriam percebidos somente com os valores de entrada.

Função de Ativação: é uma função matemática aplicada a saida de um neurônio, determinando se o mesmo deve ser ativado ou não, a partir da soma ponderada dos valores de entrada adicionada do viés. As FAs são classificadas em vários tipos como Lineares, Sigmóides, entre outras, com destaque para a Unidade Linear Retificada (ReLU), que foi utilizada em nossa implementação.

2.4.1.1 Função ReLU

É um tipo de função de ativação caracterizada por ser muito simples e eficiente computacionalmente. Devido a sua simplicidade e performance elevada é considerada por Dubey et al. (2022) como a FA estado-da-arte. A função ReLU consiste em retornar a própria entrada caso seja positivo ou 0, se negativo.

$$f(x) = max(0, x) \tag{2.2}$$

Além disso, é amplamente utilizada na literatura para o treinamento de agentes para aplicações com pixels, como é o caso do Tetris, devido ao fato de possibilitar a representação de características e padrões de maneiras mais robustas. Desativando todos os neurônios com valores de ativação negativos, a dispersão entre camadas aumenta, auxiliando o agente na percepção de padrões em cenários onde as entradas são visuais e de dimensão considerável (Mnih et al., 2013) .

2.5 APRENDIZADO PROFUNDO (*DEEP LEARNING*)

O Aprendizado Profundo é outra vertente do aprendizado de máquina caracterizado pelo uso de redes neurais. O que o diferencia dos demais e dá profundidade ao método refere-se à presença de várias camadas ocultas na arquitetura da RN, comumente dezenas e até milhares, o que habilita os modelos a extraírem informações abstratas e complexas da entrada por meio da hierarquia de conceitos.

A hierarquia de conceitos (*Hierarchy of concepts*) é o nome dado ao fluxo no qual a máquina aprende características mais complicadas da entrada a partir de outras mais simples.

Camadas internas mais no início detectam padrões mais simples e os passam para as seguintes, que os combinam e transformam, produzindo padrões de mais alto nível em camadas mais profundas. Um exemplo pode ser encontrado na área de reconhecimento de imagens, onde as características mais básicas de uma imagem, como linhas, contornos e formas, são detectadas pelos conjuntos de neurônios iniciais da RN e são combinadas mais a fundo para detectar padrões e reconhecer objetos (Nielsen, 2019).

2.5.1 Aprendizado por Reforço Profundo (*Deep Reinforcement Learning)*

O Aprendizado por Reforço Profundo (DRL, do inglês *Deep Reinforcement Learning*) é uma metodologia de Inteligência Artificial que combina o Aprendizado por Reforço com Redes Neurais Profundas. Ao interagir iterativamente com um ambiente e tomar decisões que visam maximizar recompensas cumulativas, permite que os agentes aprendam estratégias sofisticadas. Graças ao DRL, que utiliza a capacidade do aprendizado profundo de extrair características complexas de dados não estruturados, como pixels brutos ou entradas de câmera, os agentes podem aprender diretamente estratégias a partir de entradas sensoriais de alta dimensionalidade.

O DRL depende fortemente de algoritmos como o Q-learning, métodos de gradiente de política e sistemas de ator-crítico. Os conceitos de redes de valor, redes de política e trade-offs de exploração-explotação são cruciais. As aplicações do DRL são diversas e incluem robótica, jogos, bancos e saúde. Seu uso, desde o sucesso pioneiro em jogos de Atari, mais tarde do programa AlphaGo (uma IA treinada com DRL que jogava o jogo eletrônico de tabuleiro Go) e em desafios do mundo real, enfatiza sua versatilidade e poder. Amostragem de eficácia, táticas exploratórias e considerações de segurança são dificuldades. A colaboração visa impulsionar o DRL de forma responsável, prometendo um futuro que mudará a maneira como decisões são tomadas e problemas são resolvidos (Arulkumaran et al., 2017).

3 MATERIAIS E MÉTODOS

Nesta seção, são detalhados os materiais e a metodologia usados para o desenvolvimento do agente de aprendizado por reforço para o jogo Tetris. Aborda-se a definição do problema, as ferramentas computacionais utilizadas, a implementação do agente e, por fim, a metodologia experimental para avaliação de seu desempenho.

3.1 O PROBLEMA

O problema abordado neste trabalho consiste em treinar um agente para jogar Tetris com o objetivo de maximizar sua pontuação. Do ponto de vista do aprendizado por reforço, isso significa encontrar uma política de ações ótima que mapeie os estados do jogo para as ações que resultarão na maior recompensa acumulada. O desafio se encontra no tamanho do espaço de estados do Tetris e na natureza de recompensa espalhada e atrasada do jogo, onde as consequências de uma ação podem não ser imediatamente evidentes.

3.2 AMBIENTE DE SIMULAÇÃO E FERRAMENTAS

O desenvolvimento do projeto foi realizado usando a linguagem Python, na versão 3.13, muito utilizada para desenvolvimento de RN's e treinamentos similares ao nosso projeto. Foram utilizadas as seguintes bibliotecas em conjunto:

- Pygame: Biblioteca utilizada para renderizar a parte visual do jogo Tetris, com o objetivo de ser possível acompanhar o desempenho do agente durante o treinamento.
- NumPy: Biblioteca utilizada para representação do estado do jogo e manipulação de números e da matriz (representação do jogo em formato de dados para a rede neural artificial).
- PyTorch: Biblioteca utilizada para o desenvolvimento das redes neurais (MLP e CNN), que fazem parte do agente DQN. Essa foi a principal biblioteca do projeto e grande parte da lógica utilizada para o funcionamento é providenciada por essa biblioteca.
- Matplotlib: Biblioteca utilizada para plotagem de alguns gráficos que representam visualmente alguns dados do jogo, como score, linhas completadas durante o jogo, etc.

3.3 IMPLEMENTAÇÃO

A versão do Tetris implementada possui algumas modificações em relação à original, definida na seção 2.1. A primeira delas é a remoção do movimento de *hard drop*, pois não influenciaria positivamente no aprendizado de um agente, tendo em vista que é um elemento do jogo que não altera o estado final do tabuleiro, pois apenas permite posicionar uma peça mais rapidamente, proporcionando uma jogabilidade mais veloz, porém sem alterações positivas significativas na pontuação.

Houve também alterações na função de pontuação do jogo, que foi utilizada apenas a fim de medir desempenho do agente durante uma instância do jogo. A bonificação por eliminação passou a ser calculada de maneira quadrática em relação ao número de linhas completas simultaneamente, sendo o bônus 100, 400, 900, 1600, para 1, 2, 3 ou 4 linhas respectivamente.

$$linhas\_completas^2 * 100 \tag{3.1}$$

A premiação pela profundidade de posicionamento de uma peça passou a ser a metade. Além disso, novas variáveis foram incluídas nesse cálculo, como recompensas negativas por criar buracos e aumentar a *bumpiness*, que são atributos que serão explicados a seguir na subseção *3.3.1.3*.

Essa nova função para o cálculo da pontuação foi definida com o intuito de deixar eliminações de linha mais significativas na pontuação final e de ser possível identificar em jogos que o agente não eliminou linhas, se os posicionamentos foram melhores ou piores em relação a heurística. Isto é, em um jogo no qual o agente não eliminou linhas, porém posicionou peças de maneira mais otimizada, ocupando mais células inferiores no tabuleiro, terá uma pontuação final maior que um jogo em que o agente posicionou as peças majoritariamente nas células superiores.

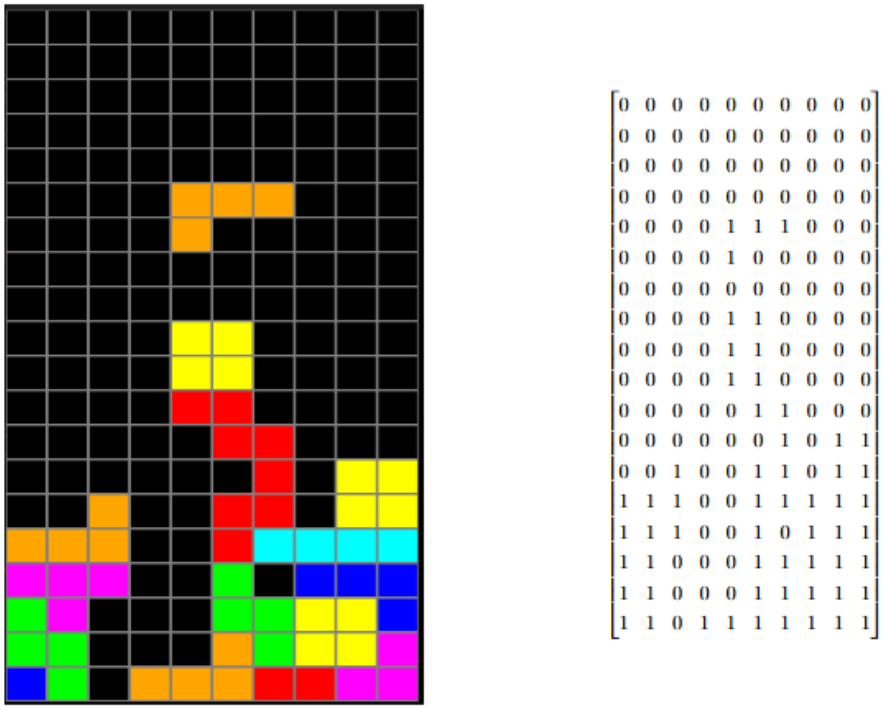
3.3.1 Definição do Agente de Aprendizado por Reforço

3.3.1.1 Estado

Um estado na implementação é a configuração do tabuleiro do jogo em dado momento, o que inclui tanto o posicionamento das peças fixas (aquelas que já foram integradas à matriz, caíram anteriormente e foram posicionadas pelo jogador) quanto o da peça em movimento, bem como as posições vazias do tabuleiro.

Como a cor das peças no Tetris possui uma função meramente visual e não influencia na jogabilidade, todas as peças podem ser representadas com o mesmo valor, optou-se por codificar os estados do jogo em matrizes binárias, com valores 0 ou 1, que apontam respectivamente para um espaço vazio no *grid* e uma posição de peça, podendo ser tanto peças já integradas quanto a da vez.

Figura 3.1: Estado do jogo e sua respectiva representação matricial.



$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

3.3.1.2 Ação

Para definir quais seriam as ações no Tetris, um jogo que possui um número relativamente pequeno de controles, uma abordagem parece óbvia, mapear cada movimento possível de uma peça em um passo do jogo. Tanto Antonietto (2023) quanto Mnih et al. (2013) optaram por essa abordagem em suas análises.

Também aplicando DQL ao Tetris, Antonietto (2023) utiliza como ações os possíveis movimentos do Tetris, direcionais, rotacionais e *hard drops*, sem o uso de qualquer espécie de heurística, as ações são escolhidas apenas por seus *scores*, ao final, todavia, não existem resultados significativos em relação a eliminação de linhas, elemento fundamental do Tetris.

Mnih et al. (2013), não aplica o DQL ao Tetris, mas a alguns jogos de Atari como *Pong* e *Space Invaders*, assim como o primeiro, sem qualquer uso de heurística ou informações específicas dos jogos e utilizando apenas o conjunto de ações de cada jogo. Esse, por sua vez, obtém resultados significativos, desempenhando melhor que seres humanos especialistas em alguns desses jogos.

No entanto, essa abordagem não se mostrou eficiente nos experimentos inciais, pois o agente não demonstrou a capacidade em associar movimentos a recompensas da maneira esperada, portanto não foi utilizada para os experimentos finais.

A ação utilizada de forma definitiva foi uma similar à que é chamada de ação agrupada em Stevens e Pradhan (2016), que utilizou o DRL para treinar um agente para jogar Tetris, em sua configuração padrão. Os resultados utilizando essa abordagem de ação são comparados com os de ações simples, as mesmas presentes no jogo, expondo uma superioridade em performance da primeira.

Portanto, a definição de ação utilizada no trabalho é um estado final do tabuleiro ao posicionar uma determinada peça. Para cada peça da vez, todos os possíveis estados finais para

a mesma são gerados, isto é, todas as possíveis posições no eixo X (horizontal) em todas as rotações de maneira que a peça esteja integrada às demais posicionadas previamente.

3.3.1.3 Recompensa

Cada ação definida acima deve possuir uma recompensa associada, de modo que possa ser avaliada e escolhida no DRL, pois a cada iteração do treinamento o agente escolhe a ação que irá tomar com base nesse valor, optando pela com maior recompensa.

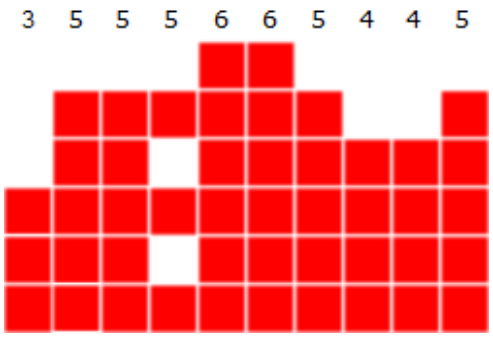
Yiyuan (2013) implementou um agente que joga Tetris em tempo real, escolhendo suas ações com base em uma heurística que utiliza informações específicas do Tetris, como o número de linhas completas, irregularidade, altura agregada e buracos no tabuleiro. A recompensa é calculada pela equação abaixo que utiliza os quatro atributos acima, em dado estado final do jogo.

$$R(s) = agg\_height_s * a + bumpiness_s * b + cleared\_lines_s * c + holes_s * d \tag{3.2}$$

Para cada estado quatro características (do inglês *features*) são calculadas cada uma multiplicada por uma constante que representa o peso da mesma na recompensa final, sendo elas:

agg_height: *aggregate height* ou altura agregada é a soma das alturas das colunas da matriz em cada estado do jogo. A altura de uma coluna, no contexto do Tetris, é a célula da matriz não vazia pertencente à coluna com maior valor no eixo Y (vertical). É ideal que o agente tente minimizar esse valor, pois mantendo o jogo concentrado em alturas mais baixas, significa mais espaço para o posicionamento de peças no decorrer do jogo.

Figura 3.2: Exemplo de *aggregated height*. Yiyuan (2013)

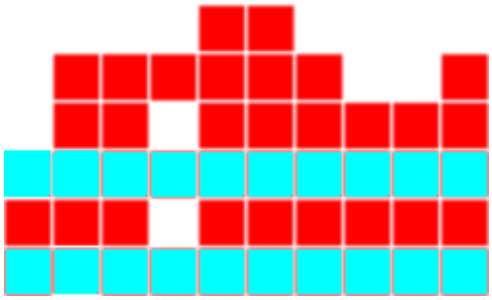


bumpiness: representa a irregularidade do tabuleiro em dado estado. É calculada pela soma das diferenças absolutas de altura entre duas colunas adjacentes. Um menor valor de *bumpiness* significa uma configuração da matriz mais achatada e estável, possibilitando o preenchimento de espaços vazios sem deixar buracos em baixo.

Figura 3.3: Exemplo de tabuleiro muito irregular, com *bumpiness* alta. Yiyuan (2013)

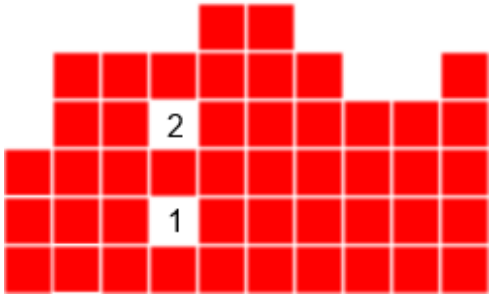
cleared_lines: *cleared lines* é a variável que indica o número de linhas eliminadas por estado. A eliminação de linhas é o objetivo primordial do Tetris, logo esse valor deve ser maximizado pelo agente, pois além de bonificar pontos, diminuirá a altura de todas as colunas da matriz em pelo menos 1.

Figura 3.4: Exemplo de linhas a serem eliminadas. Yiyuan (2013)



holes: contabiliza os buracos no tabuleiro. No Tetris, um buraco se refere a algum espaço vazio sob uma peça na mesma coluna, o que torna as linhas abaixo impossíveis de serem totalmente preenchidas e consequentemente eliminadas. O agente deve minimizar o número de buracos para manter o tabuleiro com todos os espaços acessíveis maximizando o número de linhas completas.

Figura 3.5: Exemplo de buracos no tabuleiro. Yiyuan (2013)



As constantes servem para declarar o quanto o valor de uma das *features* influenciam na recompensa. Os valores utilizados na implementação foram similares aos de Yiyuan (2013),

que por meio de um Algoritmo Genético chegou em resultados considerados, pelo autor, ótimos. Respectivamente, os valores das constantes **a**, **b**, **c** e **d** são:

$$a = -0.510066;\ b = -0.184483;\ c = 0.760666;\ d = -0.35663; \tag{3.3}$$

Nota-se que as características a serem minimizadas possuem uma constante negativa e aquela a ser maximizada é positiva.

Optou-se por utilizar essa função com esses valores como base no trabalho, pois demonstraram ser eficientes em guiar um agente a tomar decisões em tempo real e parecem ser um bom ponto de partida para uma função de recompensa para um agente de DQN.

3.3.2 Arquitetura da RN

Neste trabalho, foram exploradas duas arquiteturas de redes neurais artificiais, representando uma evolução na complexidade e adequação ao problema.

3.3.2.1 Modelo Inicial: Multi-Layer Perceptron (MLP)

A primeira implementação do agente utilizou uma rede neural *Multi-Layer Perceptron* (MLP) para aproximar a função Q. Nesta abordagem, o estado do jogo, representado pela matriz de 20x10 posições do tabuleiro, é "achatado" (do inglês *flattened*) em um vetor de uma única dimensão de 200 entradas. Este vetor alimenta uma sequência de camadas densas que, por fim, resultam em um vetor de saída com os valores Q para cada ação possível. A hipótese inicial era que uma MLP seria capaz de mapear os estados do tabuleiro para as ações ótimas. No entanto, esta arquitetura possui uma limitação significativa para o problema do Tetris: a completa perda da informação topológica. Ao converter a matriz 2D em um vetor 1D, a relação de vizinhança entre os blocos é descartada, dificultando para a rede o reconhecimento de padrões espaciais essenciais, como linhas horizontais prontas para serem completadas ou campos não preenchidos no tabuleiro.

3.3.2.2 Modelo Otimizado: Convolutional Neural Network (CNN)

Para superar a limitação da perda de informação espacial da MLP, a arquitetura do agente foi evoluída para uma Rede Neural Convolucional (CNN). Diferente da MLP, a CNN processa a entrada em seu formato de matriz 2D, tratando o tabuleiro do Tetris como uma imagem, pois é um tipo especializado de RN para processar informações com topologia do tipo *grid* (Goodfellow et al., 2016). Esta abordagem é naturalmente mais adequada ao problema. As camadas convolucionais aplicam filtros (*kernels*) que percorrem a matriz de entrada, permitindo que a rede aprenda a identificar e extrair características espaciais hierárquicas. Padrões simples, como blocos individuais ou pares de blocos, podem ser detectados nas camadas iniciais, enquanto padrões mais complexos, como a forma de peças específicas, a presença de linhas completas ou a formação de "buracos" para encaixar peças, podem ser aprendidos nas camadas mais profundas.

A arquitetura da RN que implementamos neste trabalho foi construída em PyTorch, e funciona de maneira que a rede recebe os dados do jogo como um "pacote" de informações (um *tensor*) com o formato (2, H, W), onde H e W são a altura e a largura do tabuleiro. O diferencial aqui é que usamos dois canais de entrada: um canal mostra o estado atual do tabuleiro e o outro mostra a peça que está caindo. Assim, a rede consegue analisar o campo e a peça ao mesmo tempo.

A estrutura em si foi montada com as seguintes camadas:

- Primeira camada de convolução (Conv2d): Ela usa 16 filtros com *kernel* de 3x3. Também usamos *stride* de 1 e *padding* de 1, o que na prática significa que o "mapa" do jogo não diminui de tamanho nesta etapa.
- Segunda camada de convolução (Conv2d): Para encontrar padrões mais detalhados nos dados da primeira camada, esta segunda usa 32 filtros, também com *kernel* de 3x3 e as mesmas configurações de *stride* e *padding*.
- Camada de processamento (Linear): Depois das convoluções, todos os mapas de características são "achatados" para formar um único vetor. Esse vetor então alimenta uma camada totalmente conectada com 128 neurônios para processar essas informações.
- Camada de saída (Linear): No final, uma última camada produz um vetor com a quantidade exata de saídas que nosso agente tem de ações. Cada saída é o valor Q estimado para uma ação específica.

Para ajudar a rede a aprender de forma eficiente, a função de ativação ReLU é usada logo após as camadas de convolução e a primeira camada linear. Isso introduz uma não-linearidade que é essencial para o modelo conseguir aprender as relações complexas do Tetris. Com essa estrutura, a ideia é que a rede consiga construir um entendimento muito melhor do estado do jogo e, com isso, aprender a tomar decisões mais inteligentes.

3.4 METODOLOGIA EXPERIMENTAL

3.4.1 Cenários

Foram realizados experimentos em 5 cenários diferentes, que se diferenciam pelas dimensões do tabuleiro, função de pontuação ou pela presença da informação sobre a próxima peça. A diversidade de cenários tem como objetivo medir a adaptabilidade do agente durante seu aprendizado, identificando suas limitações e pontos fortes.

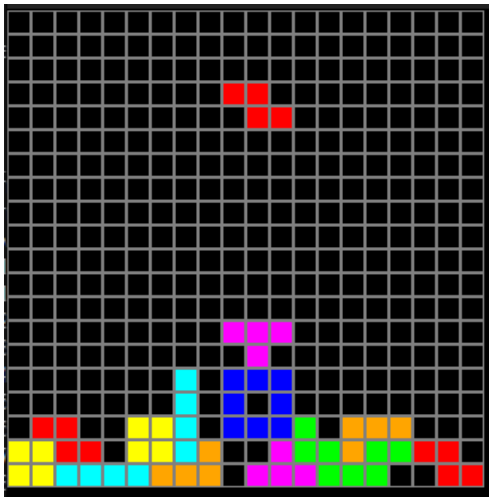
3.4.1.1 Cenário Controle

O cenário controle ou *baseline* é a implementação do Tetris em seu estado padrão, ou seja, com as dimensões originais (20x10), sem a informação da próxima peça e com a pontuação de eliminações de linhas utilizando a função quadrática, como definido na seção de Implementação.

3.4.2 Cenário Tabuleiro 20x20

Cenário que se distingue do *baseline* unicamente pela dimensão do tabuleiro, que passa a ser 10 colunas mais largo. Esse aumento em 2x do número de colunas no tabuleiro ocasiona um espaço de ações com o dobro do tamanho.

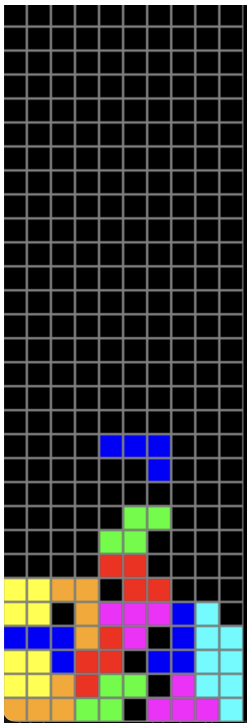
Figura 3.6: Cenário Tabuleiro Largo com dimensões 20x20.



3.4.3 Cenário Tabuleiro 30x10

Cenário que se distingue do *baseline* pela dimensão do tabuleiro, que passa a ser 10 linhas mais alto. Apesar de não aumentar o espaço de ações esse incremento na altura faz com que o agente tenha mais tempo e espaço para posicionar as peças nas posições com as maiores recompensas e demore mais para perder.

Figura 3.7: Cenário Tabuleiro Alto com dimensões 30x10.



3.4.4 Cenário Estrutura de Recompensa Linear

Nesse cenário a função de recompensa foi alterada para não bonificar o agente com uma pontuação quadrática com base no número de linhas eliminadas simultaneamente, conforme 3.3. Utilizou-se então a função originalmente implementada ao Tetris definido na seção 2.1,

que possui valores pré-definidos para cada caso. Espera-se então que o agente não desenvolva estratégias para limpar linhas simultâneas tão facilmente, e por consequência, que haja menos ocorrências de eliminações desse tipo.

4 EXPERIMENTOS E ANÁLISE DE RESULTADOS

Este capítulo apresenta e analisa os resultados dos experimentos realizados para avaliar o desempenho e a adaptabilidade do agente de *Deep Reinforcement Learning*. A análise parte de um cenário de controle (*baseline*), que serve como ponto de referência para avaliar o impacto de modificações sistemáticas no ambiente do jogo. Cada experimento foi executado 10 vezes de forma independente, com cada execução durando 1000 episódios, para garantir a robustez estatística dos dados. Episódio é uma iteração completa do jogo, desde a o início até o fim do jogo, quando não for mais possível o posicionamento de novas peças.

4.1 ANÁLISE DO CENÁRIO DE CONTROLE (*BASELINE*)

O primeiro passo da análise consiste em caracterizar o comportamento do agente no cenário de controle. Este cenário utiliza um tabuleiro de 20x10, uma função de recompensa quadrática e não oferece visibilidade da próxima peça. Os resultados desta configuração base servirão como a principal métrica de comparação para todas as variações subsequentes.

Uma observação inicial importante ao analisar as Figuras 4.1 e 4.2 é a clara semelhança visual entre as curvas de Recompensa Média e de Linhas Completadas. Embora seus formatos sejam quase idênticos, é crucial esclarecer que as métricas não são as mesmas, mas sim fortemente correlacionadas.

Conforme detalhado na Seção 3.3.1.3, a recompensa (*Reward*) utilizada para treinar o agente não é simplesmente o número de linhas, mas sim uma função heurística ponderada que considera quatro características do tabuleiro: altura agregada, rugosidade (*bumpiness*), número de buracos e, principalmente, o número de linhas completadas. A forte correlação visual entre os gráficos ocorre porque o componente “linhas completadas” possui o maior peso positivo nesta função. Portanto, ele atua como o principal fator do valor da recompensa. Quando o agente aprende a limpar mais linhas, o valor total da recompensa aumenta de forma quase diretamente proporcional, resultando em curvas de aprendizado com formatos muito similares. As outras características da função de recompensa atuam como penalidades que refinam a estratégia do agente, mas o sinal positivo predominante vem da conclusão de linhas.

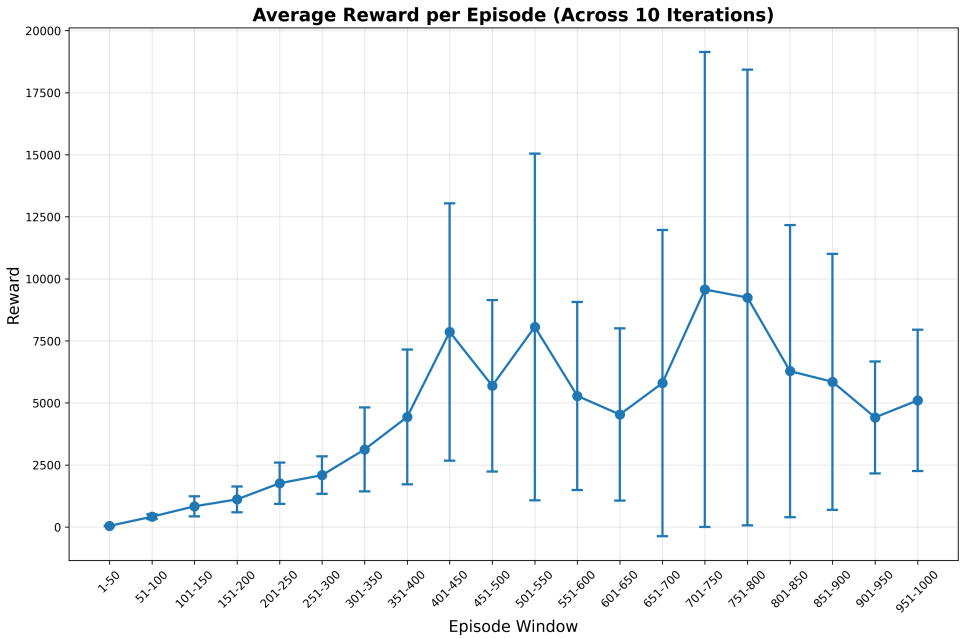


Figura 4.1: Média de recompensa por episódio no cenário *baseline*.

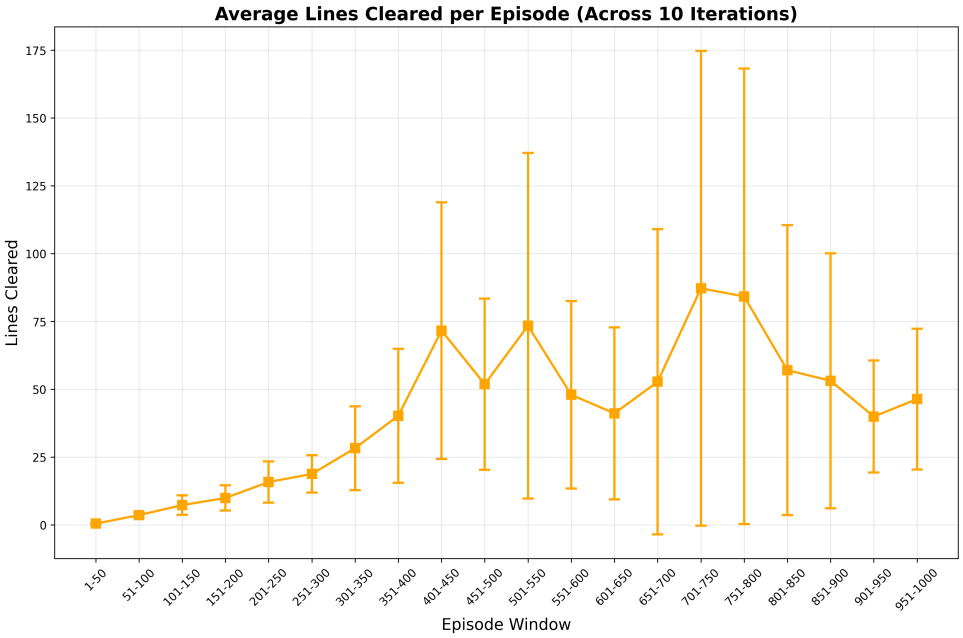


Figura 4.2: Média de linhas completadas por episódio no cenário *baseline*.

A análise das Figuras 4.1 e 4.2, que ilustram a recompensa média e o número de linhas completadas, demonstra uma clara tendência de aprendizado, embora com fases distintas. O agente exibe um progresso notavelmente consistente ao longo dos primeiros 450 episódios, partindo de um desempenho próximo de zero e melhorando de forma estável. Após este marco, o aprendizado continua, mas de maneira mais volátil, com flutuações significativas de desempenho até atingir o pico máximo na janela de 701-750 episódios.

A tendência geral ascendente, apesar da instabilidade, valida que o agente está efetivamente aprendendo a otimizar a função de recompensa e que sua habilidade se traduz diretamente em uma maior capacidade de limpar linhas. Em ambos os gráficos, a alta variabilidade entre as

execuções (barras de erro longas) e a queda de desempenho após o pico são características que evidenciam a instabilidade inerente a algoritmos de DRL (Henderson et al. (2018)).

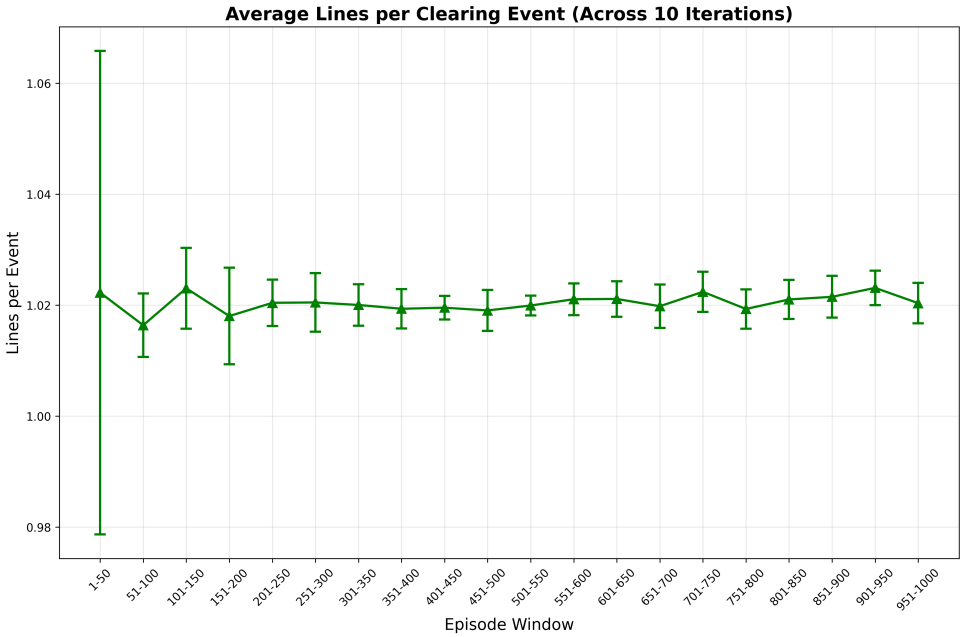


Figura 4.3: Média de linhas por evento de limpeza no cenário baseline.

A Figura 4.3, por sua vez, oferece uma visão mais detalhada sobre a estratégia adquirida. O gráfico mostra que, após uma variação inicial, a média de linhas por evento de limpeza se estabiliza em um valor muito próximo de 1.02, sem uma tendência de crescimento. Este é um resultado revelador, pois indica que, embora o agente aprenda a limpar mais linhas no geral (como visto na Figura 4.2), ele não aprende a complexa estratégia de empilhar peças para realizar limpezas múltiplas (como Duplas, Triplas ou Tetris). Sua política de jogo foca em uma estratégia reativa e de curto prazo: limpar linhas únicas assim que a oportunidade surge.

Em síntese, no cenário padrão, o agente aprende com sucesso a tarefa de sobreviver e maximizar a recompensa, tornando-se eficiente em limpar linhas. No entanto, a estratégia aprendida é simples e imediatista. Este comportamento servirá como um ponto de referência crucial para avaliar o impacto das modificações de ambiente nos próximos cenários.

4.2 ANÁLISE DO IMPACTO DA TOPOLOGIA DO TABULEIRO

Nesta seção, investiga-se como a alteração nas dimensões do tabuleiro impacta o aprendizado do agente. Comparamos o *baseline* (20x10) com um cenário de tabuleiro mais largo (20x20) e outro mais alto (30x10).

4.2.1 Resultados do Cenário de Tabuleiro Largo (20x20)

Para o cenário com o tabuleiro mais largo (20x20), os resultados obtidos para as três métricas principais são apresentados nas Figuras 4.4, 4.5 e 4.6.

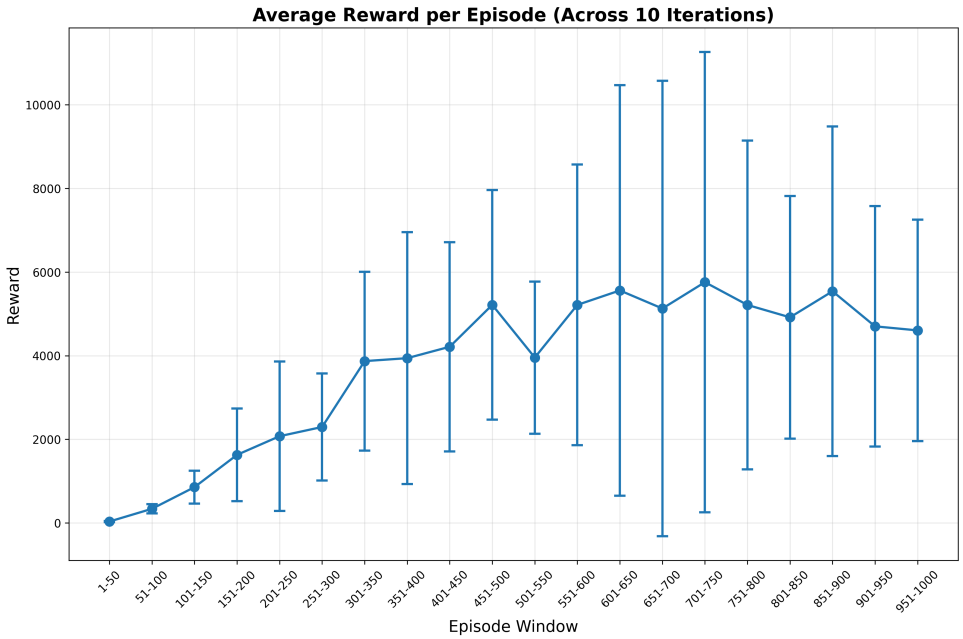


Figura 4.4: Média de recompensa por episódio no cenário 20x20.

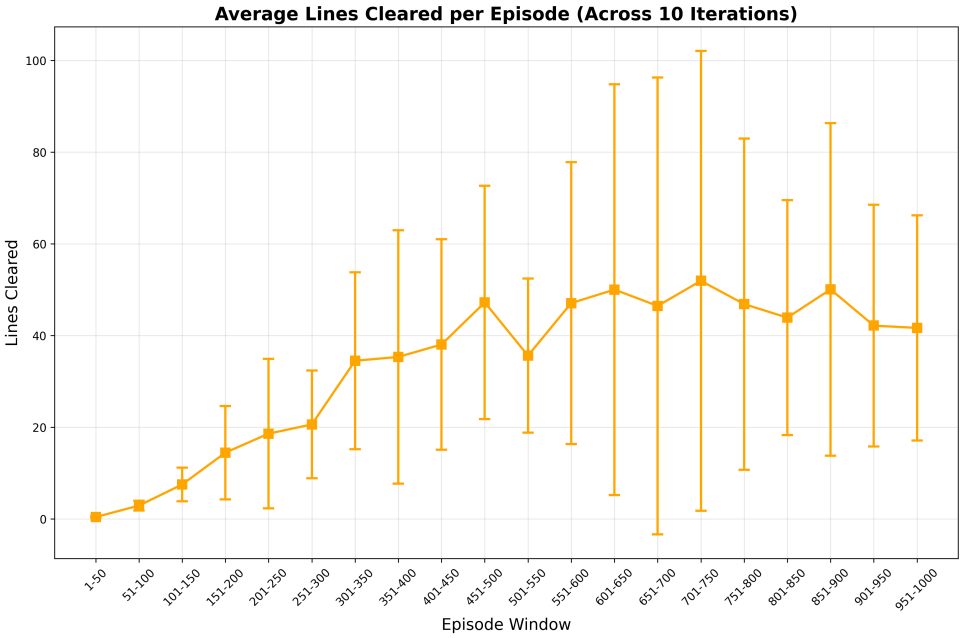


Figura 4.5: Média de linhas completadas por episódio no cenário 20x20.

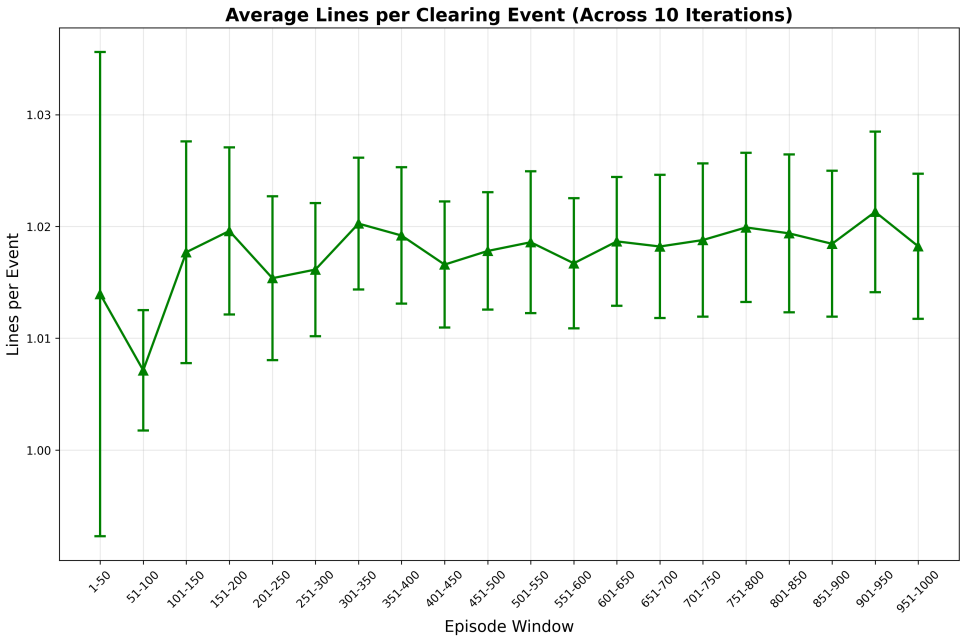


Figura 4.6: Média de linhas por evento de limpeza no cenário 20x20.

4.2.2 Resultados do Cenário de Tabuleiro Alto (30x10)

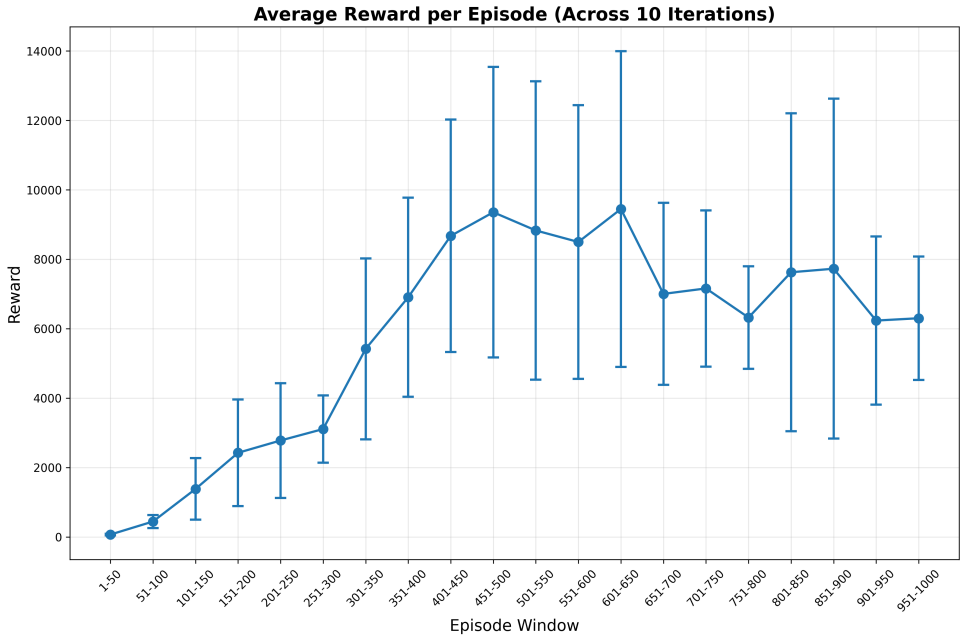


Figura 4.7: Média de recompensa por episódio no cenário 30x10.

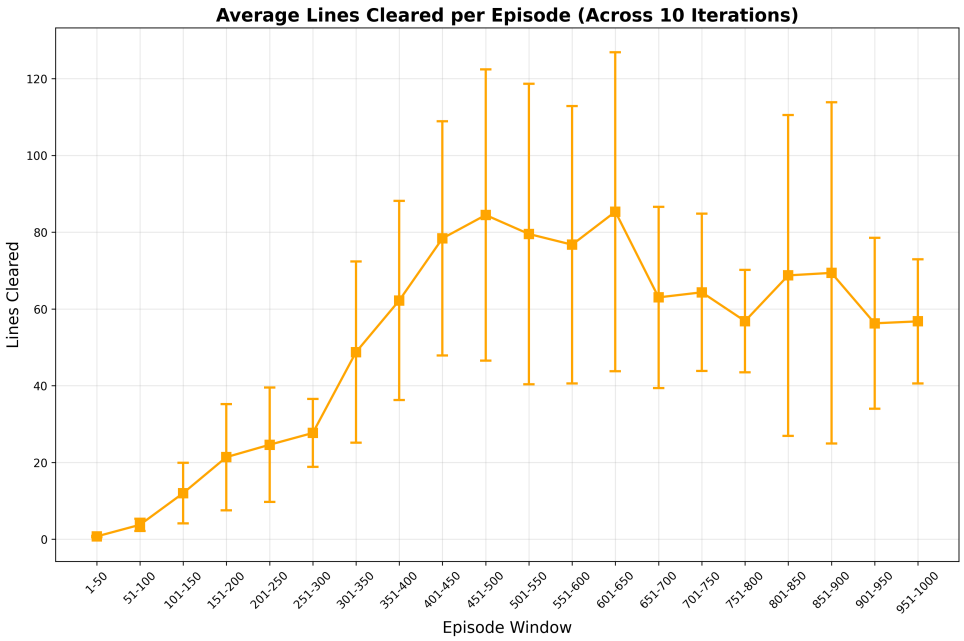


Figura 4.8: Média de linhas completadas por episódio no cenário 30x10.

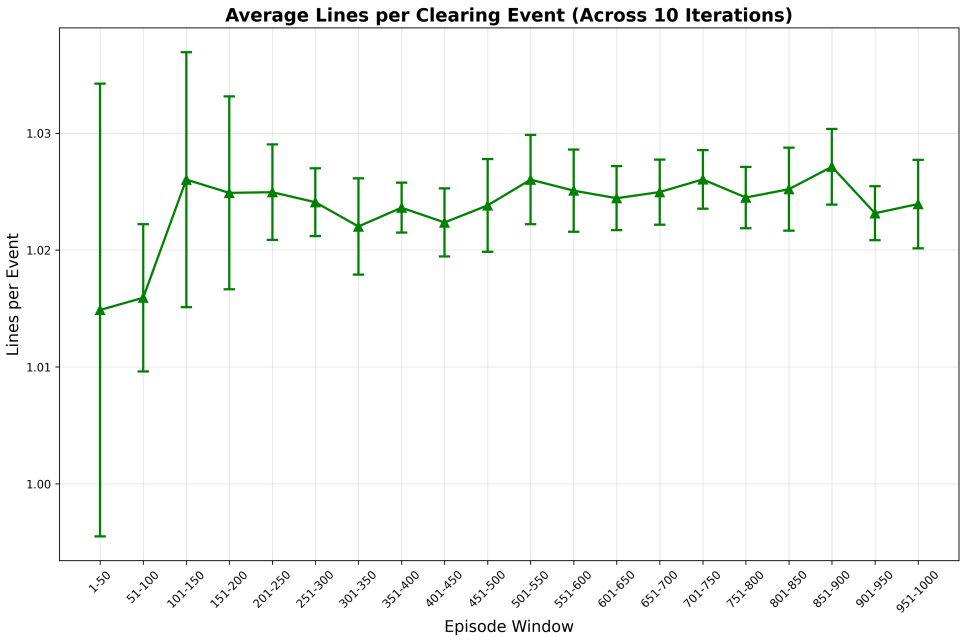


Figura 4.9: Média de linhas por evento de limpeza no cenário 30x10.

Tabela 4.1: Média de linhas eliminadas por cenário a cada 100 episódios.

Episódios	***Baseline***	**20x20**	**30x10**
1–100	2.055	1.651	**2.254**
101–200	8.633	10.981	**16.705**
201–300	17.304	19.605	**26.161**
301–400	34.287	34.915	**55.491**
401–500	61.756	42.634	**81.442**
501–600	60.705	41.343	**78.144**
601–700	46.954	48.242	**74.189**
701–800	**85.727**	49.395	60.591
801–900	55.102	46.984	**69.079**
901–1000	43.173	41.911	**56.520**

Tabela 4.2: Média de eliminações simultâneas em cada cenário por iteração.

Linhas Simultâneas	***Baseline***	**20x20**	**30x10**
1 linha	40,710.20	33,113.80	**45,690.40**
2 linhas	859.30	651.70	**1,136.30**
3 linhas	0.30	0.60	**1.40**
4 linhas	0	0	0

4.2.3 Análise Comparativa das Topologias

A comparação entre o cenário *baseline* (20x10) e o cenário do tabuleiro largo (20x20) revela resultados contraintuitivos. Embora o espaço lateral adicional possa sugerir uma maior facilidade para o agente manobrar as peças, os dados indicam um desempenho geral inferior em todas as métricas principais.

Conforme visto na Figura 4.4, o pico de recompensa média no tabuleiro 20x20 é significativamente inferior ao alcançado no *baseline* (Figura 4.1). Essa queda de desempenho é diretamente espelhada na Figura 4.5, onde o número máximo de linhas completadas também é significativamente menor. Uma possível explicação para esta dificuldade é o aumento expressivo do espaço de estados. Um tabuleiro com o dobro da largura torna o problema de exploração muito mais complexo, dificultando para o agente encontrar e convergir para políticas de jogo eficazes dentro do mesmo número de episódios de treinamento.

A análise da estratégia, apresentada na Figura 4.6, é particularmente importante. Assim como no cenário *baseline*, a média de linhas por evento de limpeza permanece estável em torno de 1.02, sem qualquer tendência de crescimento. Isso sugere que o espaço adicional não incentivou ou permitiu que o agente desenvolvesse estratégias mais sofisticadas, como o empilhamento planejado para limpezas múltiplas. A política aprendida continua a ser reativa e focada na eliminação imediata de linhas únicas.

De forma distinta, o cenário com o tabuleiro mais alto (30x10) apresentou um desempenho muito mais robusto. As Figuras 4.7 e 4.8 mostram que o agente atinge um pico de performance comparável ao do *baseline* e muito superior ao do cenário 20x20. Curiosamente, o agente parece atingir este pico de forma mais acelerada, por volta da janela de 451-500 episódios, enquanto no *baseline* o pico ocorre mais tardiamente. Isso sugere que a maior altura do tabuleiro oferece mais margem para erro, permitindo que o agente sobreviva por mais tempo em cada episódio e consolide seu aprendizado mais rapidamente, sem a penalidade de um

topo iminente, o que justifica as maiores médias de linhas eliminadas pertencerem ao cenário 30x10, como visto na Tabela-4.1. No que tange à estratégia, a Figura 4.9 aparenta revelar que o comportamento do agente no tabuleiro alto permanece inalterado, com a média de linhas por evento de limpeza também estabilizada em torno de 1.02, no entanto, se analisadas as informações na Tabela-4.2, nota-se que o tabuleiro 30x10 eliminou linhas simultâneas de maneira mais consistente, apresentando em média 1,136 eliminações duplas e 1.40 triplas a cada 100 episódios.

Em resumo, a topologia do ambiente demonstrou um impacto profundo no aprendizado. Alargar o tabuleiro (20x20) prejudicou significativamente o desempenho ao expandir a complexidade da exploração. Em contraste, aumentar a altura (30x10) se mostrou benéfico, acelerando a convergência para um desempenho similar ao do *baseline*. No entanto, é notável que nenhuma das alterações na topologia foi suficiente para induzir o desenvolvimento de uma estratégia de jogo mais complexa, com o agente mantendo o foco e a predominância em limpezas de linha única em todos os cenários.

4.2.4 Análise Comparativa das Topologias

A comparação dos três cenários de topologia revela que aumentar a largura do tabuleiro (20x20) prejudicou significativamente o desempenho, provavelmente devido ao aumento expressivo do espaço de estados. Em contraste, aumentar a altura (30x10) se mostrou benéfico, permitindo ao agente atingir um pico de performance comparável ao do *baseline* e de forma mais acelerada, como visto na Tabela 4.1. Apesar das diferenças de desempenho, a Tabela 4.2 indica que a estratégia fundamental do agente não se alterou, com a predominância massiva de limpezas de linha única em todos os cenários.

4.3 ANÁLISE DO IMPACTO DA ESTRUTURA DE RECOMPENSA

Esta seção foca em entender como a estrutura do incentivo afeta o aprendizado, comparando a recompensa quadrática (*baseline*) com a recompensa linear.

4.3.1 Resultados do Cenário de Recompensa Linear

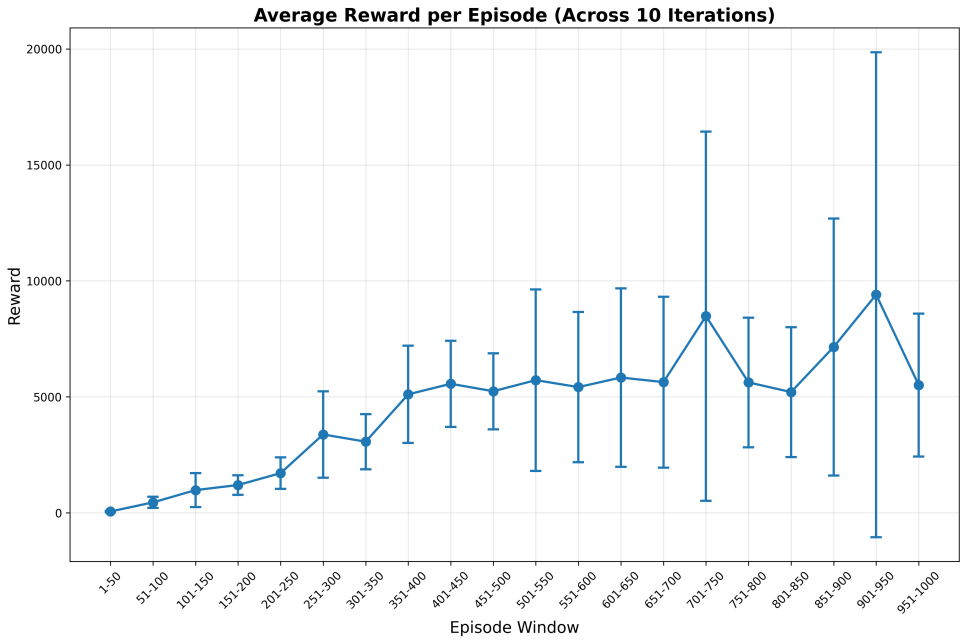


Figura 4.10: Média de recompensa por episódio no cenário com recompensa linear.

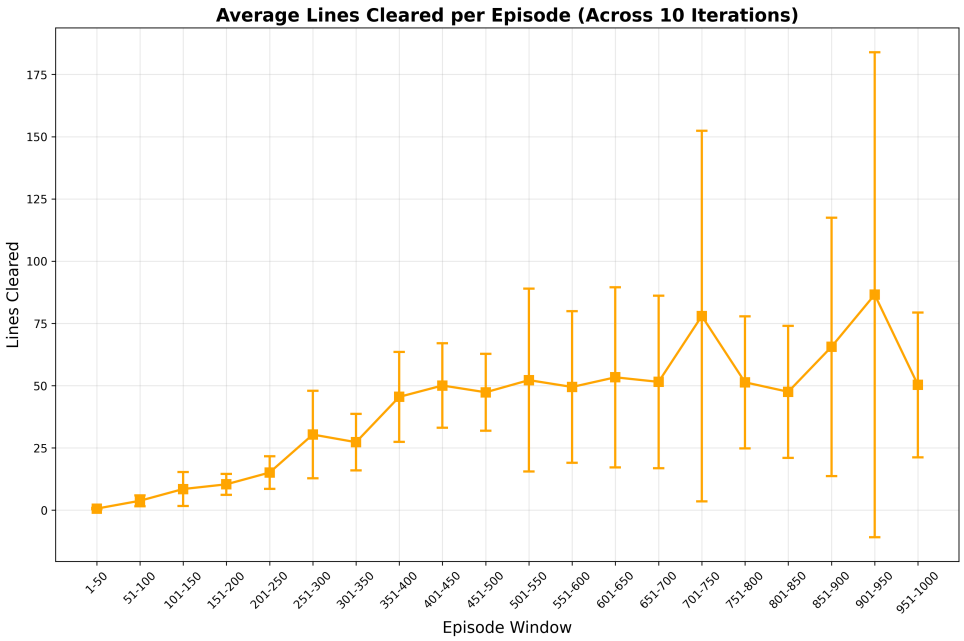


Figura 4.11: Média de linhas completadas por episódio no cenário com recompensa linear.

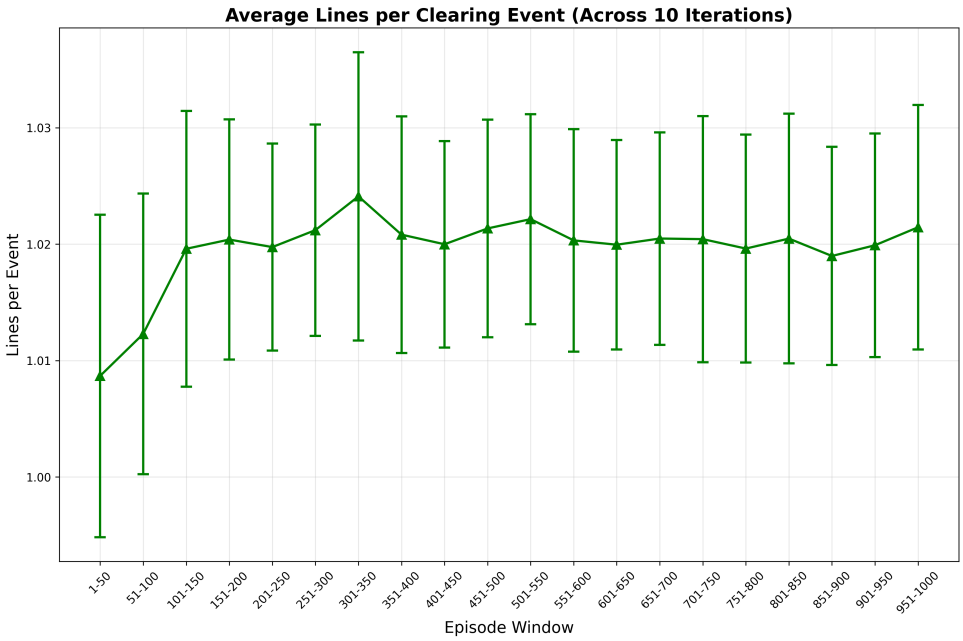


Figura 4.12: Média de linhas por evento de limpeza no cenário com recompensa linear.

Tabela 4.3: Média de linhas eliminadas por função de recompensa a cada 100 episódios.

Episódios	***Baseline***	**Linear**
1–100	2.055	**2.181**
101–200	8.633	**9.414**
201–300	17.304	**22.719**
301–400	34.287	**36.376**
401–500	**61.756**	48.683
501–600	**60.705**	50.831
601–700	46.954	**52.421**
701–800	**85.727**	64.646
801–900	55.102	**56.548**
901–1000	43.173	**68.415**

Tabela 4.4: Média de eliminações simultâneas para cada função de recompensa.

Linhas Simultâneas	***Baseline***	**Linear**
1 linha	**40,710.20**	39,359.00
2 linhas	859.30	**931.00**
3 linhas	0.30	**0.80**
4 linhas	0	0

4.3.2 Análise Comparativa das Estruturas de Recompensa

A análise visual dos gráficos de aprendizado (Figuras 4.10 a 4.12) para o cenário de recompensa linear revela um comportamento de treinamento mais errático em comparação com o *baseline*. Embora a tendência geral ainda seja de crescimento, a curva é menos estável. No entanto, uma análise mais aprofundada através dos dados tabulados oferece uma perspectiva mais completa e surpreendente.

A Tabela 4.3 mostra que o agente com recompensa linear, apesar da instabilidade, supera o *baseline* em número de linhas limpas nas fases iniciais (1-400 episódios) e finais (801-1000 episódios) do treinamento, indicando que o aprendizado, embora volátil, é eficaz.

O ponto mais interessante reside na análise da estratégia, via Tabela 4.4. Enquanto o gráfico da média de linhas por evento (Figura 4.12) parece idêntico ao do cenário base, a tabela revela que o agente com recompensa linear executou, em média, mais limpezas de duas e três linhas. Isso sugere que a ausência de uma recompensa quadrática massiva para limpezas de três a quatro (se comparadas a 900 e 1600 na quadrática, respectivamente) linhas pode ter incentivado o agente a adotar uma política marginalmente mais eficaz em criar limpezas duplas, prolongando sua vida e consequentemente limpando mais linhas com o passar do tempo, um comportamento sutil não visível na média geral dos gráficos. Portanto, a mudança para uma recompensa linear, embora não tenha transformado a estratégia fundamental do agente, parece ter refinado seu comportamento de maneira positiva.

4.4 DISCUSSÃO GERAL DOS RESULTADOS

A análise conjunta dos diferentes cenários experimentais permite traçar conclusões abrangentes sobre a sensibilidade e a adaptabilidade do agente DQN no ambiente do Tetris. Os resultados demonstram que, embora o algoritmo de aprendizado seja sempre o mesmo, seu desempenho é profundamente influenciado por cada uma das modificações propostas no ambiente, revelando quais fatores são mais críticos para o sucesso do treinamento.

A modificação com o impacto mais significativo foi a da topologia do tabuleiro. O cenário de tabuleiro largo (20x20) teve o efeito mais negativo, prejudicando severamente a capacidade do agente de atingir o desempenho do *baseline*. Isso evidencia que a expansão do espaço de estados horizontal, que aumenta a complexidade do problema de exploração, é um desafio maior para o agente do que o simples aumento da altura. Em contraste, o cenário de tabuleiro alto (30x10) teve o impacto mais positivo, não só permitindo que o agente alcançasse um desempenho comparável ao do cenário controle, mas também acelerando a convergência do aprendizado.

Um dos achados mais consistentes deste trabalho diz respeito à estratégia do agente. Em todos os cenários, a política de jogo aprendida se mostrou extremamente robusta, porém simplista. A média de linhas por evento de limpeza permaneceu praticamente inalterada, indicando uma forte tendência do agente em convergir para uma estratégia reativa de limpar linhas únicas assim que possível. No entanto, a análise detalhada da Tabela 4.4 revelou a nuance importante de que o cenário de recompensa linear, ao remover o incentivo extremo para limpezas de quatro linhas, permitiu que o agente executasse mais limpezas duplas e triplas. Isso sugere que, embora a estratégia dominante seja robusta, ela é sutilmente sensível à estrutura de incentivos.

Com base nos resultados, pode-se concluir que o design do ambiente é um fator tão ou mais crítico para o sucesso do treinamento de um agente de DRL quanto a própria arquitetura do agente. Para jogos como o Tetris, um ambiente que pune severamente erros (tabuleiro baixo) ou que torna a exploração excessivamente complexa (tabuleiro largo) pode impedir o aprendizado eficaz. Além disso, para induzir estratégias mais sofisticadas e de longo prazo, como o planejamento para um "Tetris", uma função de recompensa simples, mesmo que bem ponderada, pode ser insuficiente. Seriam necessários métodos mais avançados para superar a tendência do agente em convergir para a solução mais simples e imediata.

5 CONCLUSÃO

Este trabalho se propôs a investigar a capacidade de adaptação de um agente de Aprendizado por Reforço Profundo, baseado no algoritmo *Deep Q-Learning* (DQL), quando confrontado com variações de ambiente do jogo Tetris. O objetivo central foi compreender como alterações na topologia do tabuleiro e na estrutura da função de recompensa impactariam o desempenho, a velocidade de aprendizado e a estratégia final do agente. Para tal, um agente de referência foi treinado em um cenário de controle (*baseline*) e seu desempenho foi comparado com o de cenários com tabuleiros mais largos e mais altos, e com uma função de recompensa linear, a qual é usada para a pontuação do jogo originalmente, em vez de quadrática.

A análise comparativa dos experimentos gerou conclusões significativas. Foi demonstrado que a topologia do ambiente é um fator crítico que influencia diretamente a eficácia do treinamento. O aumento da largura do tabuleiro (20x20) teve o impacto mais negativo, prejudicando o desempenho do agente ao aumentar exponencialmente a complexidade do problema de exploração. Em contraste, o aumento da altura (30x10) mostrou-se positivo, não só permitindo que o agente atingisse um desempenho comparável ao do *baseline*, mas também acelerando a convergência de seu aprendizado. A mudança na função de recompensa para um modelo linear, por sua vez, embora tenha resultado em curvas de treinamento mais voláteis, revelou uma melhora sutil na estratégia do agente, que passou a executar mais limpezas simultâneas, de duas e três linhas.

Um dos achados mais consistentes e importantes deste estudo foi a robustez da estratégia simplista e reativa do agente em todos os cenários, pois apesar das variações no ambiente e no desempenho geral, o agente consistentemente convergiu para uma política de limpar linhas únicas de forma imediata, sem desenvolver a capacidade de planejamento de longo prazo necessária para criar jogadas de alto valor, como os "Tetrises". Isso sugere que, para a metodologia aplicada, a convergência para uma solução "boa o suficiente", porém simples, é uma tendência forte.

É importante também reconhecer as limitações deste trabalho, visto que o treinamento foi conduzido por iterações com um número fixo de 1000 episódios, e é possível que estratégias mais complexas pudessem emergir com um período de treinamento mais extenso. Adicionalmente, os mesmos hiperparâmetros do agente foram utilizados em todos os cenários. Acreditamos que uma otimização individual para cada ambiente poderia levar a resultados distintos, potencializando o desempenho em cada caso específico.

Com base nas conclusões e limitações, diversas direções para trabalhos futuros podem ser tomadas, como a investigação do impacto de um treinamento mais longo, e a otimização de hiperparâmetros por cenário, as quais consideramos como passos naturais. Outra linha também promissora seria a implementação de técnicas de recompensa mais sofisticadas (*reward shaping*) para incentivar ativamente o desenvolvimento de estratégias complexas. Por fim, a implementação do cenário com visibilidade da próxima peça, planejada inicialmente, permanece como um próximo passo crucial para entender o impacto da disponibilidade de informação no planejamento do agente.

Concluímos que, no campo do DRL, a interação entre agente e ambiente é uma via de mão dupla, onde a configuração dos cenários e ambientes no qual o agente está inserido é tão importante quanto a heurística e algoritmo de inteligência artificial do mesmo.

REFERÊNCIAS

Antonietto, A. M. (2023). Aprendizado por reforço profundo aplicado a tetris com entradas não processadas. Relatório técnico, UFPR.

Arulkumaran, K., Deisenroth, M. P., Brundage, M. e Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6).

Benjamin Remman, S. (2021). The reinforcement learning loop.

Demaine, E. D., Hohenberger, S. e Liben-Nowell, D. (2003). Tetris is hard, even to approximate. Em *Computing and Combinatorics: 9th Annual International Conference, COCOON 2003 Big Sky, MT, USA, July 25–28, 2003 Proceedings 9*. Springer.

Dubey, S. R., Singh, S. K. e Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503.

Goodfellow, I., Bengio, Y. e Courville, A. (2016). *Deep Learning*. MIT Press.

Hagan, M. T., Demuth, H. B. e Beale, M. H. (1996). *Neural Network Design*. PWS Publishing Company.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. e Meger, D. (2018). Deep Reinforcement Learning that Matters. Em *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

IBM (2024). Reinforcement Learning. `https://www.ibm.com/br-pt/think/topics/reinforcement-learning`.

Li, Y. (2019). Reinforcement learning applications. *arXiv preprint arXiv:1908.06973*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. e Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nielsen, M. A. (2019). *Neural Networks and Deep Learning*. Determination Press.

Spano, S., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Matta, M., Nannarelli, A. e Re, M. (2019). An efficient hardware implementation of reinforcement learning: The q-learning algorithm. *Ieee Access*, 7.

Stevens, M. e Pradhan, S. (2016). Playing tetris with deep reinforcement learning. *Convolutional Neural Networks for Visual Recognition CS23, Stanford Univ., Stanford, CA, USA, Tech. Rep*.

Sutton, R. S. e Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition.

Watkins, C. J. e Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4).

Yiyuan, L. (2013). Tetris ai – the (near) perfect bot.