Application of Reinforcement Learning in the Development of Intelligent Agents for the Game of Othello

Valentina Silva de Lacerda[1] **Amanda Gondim de Oliveira**[1]

[1]Universidade Federal Rural do Semi-Árido (UFERSA)
Mossoró – RN – Brasil

valentina.lacerda@alunos.ufersa.edu.br, amandagondim@ufersa.edu.br

Abstract. *This work presents the development of an intelligent agent based on Deep Reinforcement Learning to play the game of Othello. The proposed method employs the Double Deep Q-Network (DDQN) algorithm, with invalid action masking and a linearly decaying exploration policy, aiming to learn effective strategies without prior specialized knowledge. The experimental results indicate an improvement in performance throughout training, with the agent achieving a 94% win rate against a stochastic opponent. It is concluded that, despite the limitations imposed by a stochastic opponent and by the fact that training was conducted only with the agent starting with the black pieces, the agent is capable of achieving superior performance to the random policy under the evaluated conditions.*

Resumo. *Este trabalho apresenta o desenvolvimento de um agente inteligente baseado em Aprendizado por Reforço Profundo para atuar no jogo Othello. O método proposto utiliza o algoritmo Double Deep Q-Network (DDQN), com mascaramento de ações inválidas e política de exploração com decaimento linear, visando aprender estratégias eficientes sem conhecimento prévio especializado. Os resultados experimentais indicam evolução do desempenho ao longo do treinamento, com o agente atingindo taxa de vitória de 94% contra um oponente estocástico. Conclui-se que, apesar das limitações impostas por um adversário aleatório e pelo fato de o treinamento ter sido realizado apenas com o agente iniciando com as peças pretas, o agente é capaz apresentar desempenho superior à política aleatória nas condições avaliadas.*

1. Introdução

O presente artigo tem como objeto de estudo o desenvolvimento de um agente autônomo treinado por meio de Aprendizado por Reforço Profundo para atuar no ambiente do jogo de tabuleiro Othello. O assunto é abordado sob a perspectiva de computação inteligente, focando na capacidade de uma máquina aprender estratégias de forma independente, somente através de interações com o ambiente e da maximização da recompensa, sem necessidade de intervenção humana [Sutton and Barto 2018].

Diante disso, tendo em vista que o jogo Othello foi resolvido no campo teórico, o foco atualmente é na eficiência de aprendizado dos agentes autônomos. A principal contribuição deste trabalho reside na realização de um estudo comparativo sobre o impacto de diferentes estratégias de treinamento, especificamente variando a taxa de aprendizado, a política de exploração e a modelagem da função de recompensa no âmbito de

uma implementação de *Double Deep Q-Network* (DDQN) com mascaramento de ações inválidas aplicada ao Othello.

Na literatura, há exemplos de trabalhos que já abordaram o uso de aprendizado profundo em jogos, como a arquitetura *Q-Network* (DQN) proposta por [Mnih et al. 2015] para jogos Atari, o modelo do AlphaGo [Silver et al. 2016], que empregou o uso de Redes Neurais (CNNs) para processamento do tabuleiro e o [van Hasselt et al. 2016] na implementação de DDQN. Especificamente no contexto do Othello, van der Ree e Wiering [van der Ree and Wiering 2013] investigaram estratégias de Aprendizado por Reforço baseadas em aprendizado contra um oponente fixo e em *self-play*, utilizando diferentes algoritmos de aprendizado. Enquanto esses trabalhos exploram diferentes abordagens de Aprendizado por Reforço no jogo, o presente estudo concentra-se na utilização de DDQN com CNNs, mascaramento de ações inválidas e na comparação entre diferentes configurações de treinamento.

Diante disso, a hipótese que guia este trabalho sustenta que a utilização de Aprendizado por Reforço Profundo, com Redes Neurais Convolucionais para aproximação da função de valor e DDQN para estabilização do treinamento, permite o desenvolvimento de um agente autônomo capaz de apresentar desempenho superior ao de agentes que realizam escolhas aleatórias, sem a necessidade de conhecimento especializado previamente programado.

2. Fundamentação Teórica e Trabalhos Correlatos

Esta seção apresenta os conceitos matemáticos e computacionais que embasam o desenvolvimento do agente inteligente, além de revisar as principais abordagens da literatura que serviram de diretriz para a concepção da arquitetura proposta.

2.1. O Jogo Othello e a Modelagem por Reforço

O Othello é um jogo de tabuleiro de duas dimensões, com tamanho de 8 por 8 casas, que utiliza 64 peças reversíveis com lados de cores pretas e brancas. O objetivo final de uma partida é terminar o confronto com a maior quantidade de peças da sua cor voltada para cima no tabuleiro [Federação Brasileira de Othello 2026]. Entre as abordagens tradicionais de busca aplicadas ao Othello, destaca-se o programa Logistello, baseado em variantes do Minimax e funções de avaliação, que superou o campeão mundial humano em 1997 [Buro 1997, Buro 2002]. Mais recentemente, o Othello em tabuleiro 8×8 foi resolvido do ponto de vista da teoria dos jogos, provando-se computacionalmente que a jogada perfeita por ambos os lados resulta sempre em empate [Takizawa 2023].

Paralelamente aos métodos determinísticos de busca, a Aprendizagem por Reforço (AR) foi explorada para permitir que agentes aprendam estratégias sem conhecimento especialista prévio. Trabalhos fundamentais em AR tradicional para Othello compararam o desempenho de algoritmos como *Q-Learning*, Sarsa e *TD-Learning* em cenários de aprendizado por auto-jogo (*self-play*) versus treinamento contra oponentes fixos [van der Ree and Wiering 2013]. Os resultados apresentados pelos autores indicaram que, enquanto o *TD-Learning* atinge maior generalização quando treinado via *self-play*, algoritmos fora de política (*off-policy*) como o *Q-Learning* tendem a otimizar seu desempenho quando expostos diretamente às dinâmicas dos oponentes contra os quais são testados [van der Ree and Wiering 2013].

A partida inicia com quatro peças dispostas nas diagonais centrais do tabuleiro, divididas entre pretas e brancas. O jogador com as peças pretas realiza o primeiro movimento e os turnos alternam-se consecutivamente, sendo obrigatório passar a vez caso não haja jogadas válidas disponíveis. Um movimento é considerado válido quando o jogador posiciona uma peça em uma casa vazia de modo a cercar, em linha reta ou diagonal, uma ou mais peças adversárias entre a nova peça e outra de sua própria cor conforme demonstrado na figura 1a. Sendo assim, na figura 1b todas as peças inimigas cercadas são revertidas e passam a assumir a cor do jogador do turno atual.



(a) O jogador branco joga a peça A.

(b) Peças viradas pelo jogador.

Figura 1. Dinâmica de jogadas no Othello (Fonte: Federação Brasileira de Othello).

Diante da alta complexidade do jogo e do vasto espaço de estados possíveis, técnicas baseadas unicamente em busca exaustiva apresentam limitações computacionais, motivando também o estudo de métodos baseados em aprendizado e aproximação de funções [Sutton and Barto 2018].

Para mitigar esse desafio, o Aprendizado por Reforço surge como uma vertente em que o agente aprende a tomar decisões otimizadas por meio de interações iterativas de tentativa e erro com o ambiente externo, visando maximizar o acúmulo de um sinal de recompensa numérica ao longo do tempo [Sutton and Barto 2018]. Esse processo de tomada de decisão sequencial é modelado formalmente como um Processo de Decisão de Markov (MDP), definido por cinco elementos: um conjunto de estados (S), um conjunto de ações (A), uma função de transição (P), uma função de recompensa (R) e um fator de desconto (γ) posicionado entre 0 e 1, responsável por calibrar o peso dos ganhos de curto versus longo prazo [Sutton and Barto 2018]. O comportamento do agente é guiado por uma política $\pi(a \mid s)$, e o propósito do aprendizado consiste em encontrar uma política ótima π^* que maximize a recompensa acumulada esperada [Sutton and Barto 2018].

Para mapear e estimar as recompensas de longo prazo, o algoritmo *Q-Learning* calcula iterativamente o valor de uma função de qualidade para cada par estado-ação, gerando o valor Q [Sutton and Barto 2018]. A atualização matemática ocorre com base na experiência adquirida a cada passo através da seguinte equação:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (1)$$

em que α representa a taxa de aprendizado e o termo posicionado entre colchetes define o erro de diferença temporal, responsável por calibrar gradativamente as expectativas do algoritmo com base nos resultados observados no ambiente [Sutton and Barto 2018]. No entanto, a abordagem clássica armazena esses valores em uma estrutura conhecida como Tabela Q, a qual se torna computacionalmente inviável e ineficiente quando o espaço de estados assume grandes dimensões, tornando mandatória a substituição da tabela por aproximadores de funções [Sutton and Barto 2018].

2.2. Aprendizado por Reforço Profundo e Trabalhos Correlatos

O Aprendizado Profundo soluciona a limitação dimensional das tabelas ao extrair representações e características diretamente dos dados brutos de forma automática e hierárquica por meio de múltiplas camadas não lineares [Goodfellow et al. 2016]. No contexto do processamento de estruturas com topologia de grade (como matrizes e tabuleiros), as CNNs superam as limitações das redes totalmente conectadas (as quais destroem a topologia da entrada ao vetorizá-la e aumentam o risco de superajuste) ao utilizarem filtros que operam sob os princípios de interações esparsas e compartilhamento de parâmetros [Goodfellow et al. 2016].

A fusão dos algoritmos de tomada de decisão com os modelos de aproximação de funções baseados em redes neurais profundas consolida o Aprendizado por Reforço Profundo, permitindo o processamento de ambientes de alta dimensão diretamente de sua forma sensorial bruta [Mnih et al. 2015]. O modelo DQN pioneiro dessa área, utilizou CNNs para estimar a função de valor de ação a partir de matrizes de entrada [Mnih et al. 2015]. Para mitigar os problemas de instabilidade e divergência matemática gerados por aproximadores não-lineares, a arquitetura introduziu duas técnicas: a Repetição de Experiência (*Replay Buffer*), que armazena transições antigas e as amostra de forma aleatória para quebrar a correlação temporal dos dados; e a Rede Alvo, que congela temporariamente os parâmetros de uma segunda rede idêntica para calcular os valores dos estados seguintes de forma estável, solucionando o problema do alvo em movimento durante o cálculo do erro [Mnih et al. 2015].

Um exemplo de aplicação de CNNs ao processamento de tabuleiros é o sistema AlphaGo [Silver et al. 2016], em que os pesquisadores mapearam as interseções e as peças do jogo Go como canais e pixels de uma matriz de entrada, permitindo que a rede aprendesse a enxergar estruturas de território e táticas espaciais sem o uso de heurísticas manuais. Contudo, van Hasselt et al. [van Hasselt et al. 2016] demonstraram que o algoritmo DQN clássico manifesta uma tendência a superestimar os valores Q em virtude do uso da mesma rede para selecionar e avaliar as ações. Para corrigir esse viés e evitar a sobrevalorização de estratégias subótimas, os autores propuseram a arquitetura DDQN, a qual desacopla a tomada de decisão utilizando a Rede Principal para selecionar a melhor jogada e a Rede Alvo para avaliar o valor exato dessa escolha [van Hasselt et al. 2016]. Trabalhos recentes também investigam a aplicação de métodos de aprendizado profundo e DDQN ao Othello [Teymorian 2023, Zheng et al. 2021]. Dessa forma, a união do processamento de matrizes de jogos como imagens com a estabilidade e correção algorítmica do DDQN compõe a fundamentação arquitetural adotada para o agente deste trabalho.

3. Metodologia Proposta

Esta seção detalha a abordagem metodológica adotada no desenvolvimento do agente inteligente, compreendendo o ecossistema de ferramentas selecionado, a modelagem matemática do problema, a arquitetura computacional da rede neural e as estratégias experimentais de treinamento.

3.1. Modelagem do MDP e Tecnologias

O projeto foi desenvolvido na linguagem de programação Python, utilizando a biblioteca PyTorch para a definição dos grafos computacionais e gerenciamento de tensores em hardware de aceleração gráfica GPU NVIDIA T4 [Paszke et al. 2019]. A simulação das regras do jogo Othello e o ciclo de iteração do aprendizado baseou-se na interface padronizada da biblioteca Gymnasium, integrada às ferramentas NumPy e Matplotlib para manipulação algébrica e plotagem de dados.

Para viabilizar a aplicação das técnicas de Aprendizado por Reforço Profundo, o ambiente de Othello foi modelado formalmente como um Processo de Decisão de Markov (MDP) através dos seguintes componentes estruturais:

- **Estado (S):** Definido por uma tupla contendo a configuração física do tabuleiro (uma matriz de dimensões 8×8) e a indicação do turno atual. Na matriz, as casas recebem os valores 1 para as peças do agente, -1 para as do oponente e 0 para posições vazias. O agente sempre iniciará a partida, sendo assim, o aprendizado terá o viés das peças pretas.
- **Ações (A):** Espaço discreto composto por 64 possibilidades de escolha correspondentes às coordenadas do tabuleiro, mapeadas linearmente em um vetor unidimensional que varia da ação 0 à ação 63. Aplica-se a técnica de mascaramento de ações, restringindo a seleção às jogadas válidas e impedindo que ações inválidas sejam escolhidas. Quando não há jogadas válidas, o ambiente realiza automaticamente a passagem de turno, mantendo o espaço de ações restrito às 64 casas do tabuleiro. A partida termina somente quando nenhum dos jogadores possui movimentos válidos.
- **Recompensa (R):** O sinal numérico foi estruturado sob duas abordagens independentes, para fins comparativos: a formulação esparsa, na qual são atribuídos valores nulos aos turnos intermediários e, no estado terminal, bônus fixos de 1 para vitória, -1 para derrota e 0 para empate; e a formulação densa, que, além da recompensa esparsa, incorpora bônus imediatos proporcionais à quantidade de peças adversárias viradas em cada jogada. Nesse caso, a recompensa final de cada jogada é obtida pela soma da recompensa esparsa com o número de peças viradas dividido por 100.
- **Fator de Desconto (γ):** Fixado no valor de 0.99 para assegurar uma alta valorização das recompensas de longo prazo, priorizando o planejamento estratégico voltado ao desfecho da partida.

A Figura 2 apresenta o processo de treinamento. Durante cada episódio, as interações com o ambiente geram transições contendo o estado atual, a ação, a recompensa e o próximo estado, armazenadas em um buffer de repetição de experiência com capacidade para 10.000 amostras. Após o encerramento de cada partida, são realizadas

quatro etapas de otimização com lotes de 128 transições amostradas aleatoriamente, reduzindo a dependência temporal entre as experiências. Em paralelo, os parâmetros da rede alvo são sincronizados periodicamente com os da rede principal, a cada 10 episódios.

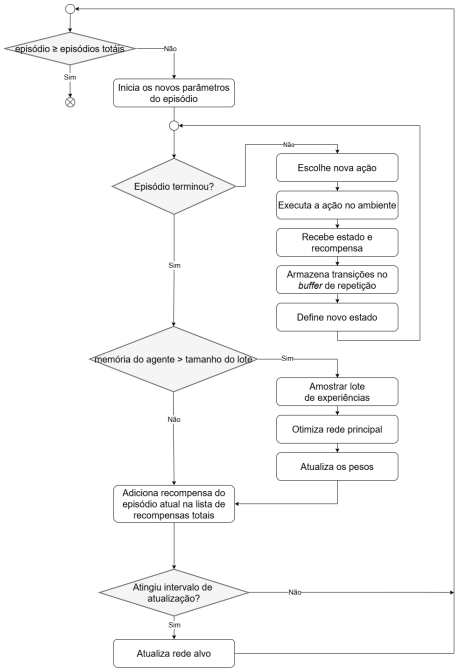


Figura 2. Fluxograma do Processo de Treinamento do DDQN. (Fonte: Autoria própria)

3.2. Arquitetura da Rede Neural

O modelo computacional utiliza CNN para aproximar a função de valor e estimar os valores Q do agente a partir da análise espacial do tabuleiro. A observação do ambiente é alimentada na rede como um tensor tridimensional de dois canais de entrada com dimensões 8×8, em que o primeiro canal mapeia a disposição das peças e o segundo funciona como uma matriz de contexto contendo o identificador do jogador do turno atual.

A extração de características estruturais é realizada por três camadas convolucionais sequenciais equipadas com filtros de tamanho 3×3. A primeira camada convolucional aplica 32 filtros sobre o tensor de entrada, enquanto a segunda e a terceira camadas utilizam 64 filtros cada. O preenchimento de bordas (*padding*) é aplicado em todas as etapas convolucionais para evitar a redução prematura dos mapas de características, garantindo que as regiões de borda e canto do tabuleiro mantenham sua relevância tática no processamento.

Cada bloco convolucional incorpora a função de ativação não linear ReLU, responsável por substituir valores negativos por zero e introduzir a não-linearidade necessária para o aprendizado de padrões complexos. Camadas de agrupamento espacial (*pooling*) foram suprimidas da arquitetura para preservar a integridade das 64 coordenadas geométricas originais e evitar a perda de resolução tática provocada pela redução de amostragem.

O espaço de ações do jogo contempla 64 possibilidades, porém, em determinados estados, nem todas são válidas. Para impedir a seleção de jogadas que não atendem às

regras do Othello, é aplicado um mascaramento de ações inválidas, no qual os valores Q correspondentes a essas ações são atribuídos a $-\infty$. Dessa forma, as ações inválidas são desconsideradas na seleção da ação de maior valor Q. Durante a etapa de exploração da política ϵ-*greedy*, a ação aleatória também é selecionada exclusivamente entre as ações válidas.

A Figura 3 apresenta o fluxo de interação e treinamento do agente DDQN. Na etapa de interação com o ambiente, o estado atual do tabuleiro é convertido na representação de dois canais e encaminhado à Rede Principal, que produz os valores Q para as 64 ações possíveis. Em seguida, o mascaramento de ações inválidas desconsidera as jogadas que não atendem às regras do Othello. A política ϵ-*greedy* utiliza os valores Q resultantes para selecionar a ação. A ação escolhida é executada no ambiente, que retorna o próximo estado e a recompensa. O conjunto dessas informações é armazenado na memória de experiência.

Durante o treinamento, uma amostra aleatória de 128 transições é extraída da memória. O alvo Y_t^{DDQN} é calculado separando a seleção da ação, realizada pela Rede Principal (θ), de sua avaliação, realizada pela Rede Alvo (θ^-), conforme a Equação 2, demonstrada em [van Hasselt et al. 2016]:

$$Y_t^{DDQN} = R_{t+1} + \gamma \cdot Q\left(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-\right) \tag{2}$$

em que R_{t+1} representa a recompensa e $\gamma = 0{,}99$ o fator de desconto. Para transições terminais, o valor futuro é considerado nulo. Durante a seleção da ação no próximo estado, o mascaramento também é aplicado. A otimização da Rede Principal ocorre pela minimização da função de perda *Smooth L1* (*Huber Loss*) entre os valores preditos e o alvo, utilizando o otimizador Adam. Os parâmetros da Rede Alvo são sincronizados com os da Rede Principal a cada 10 episódios.

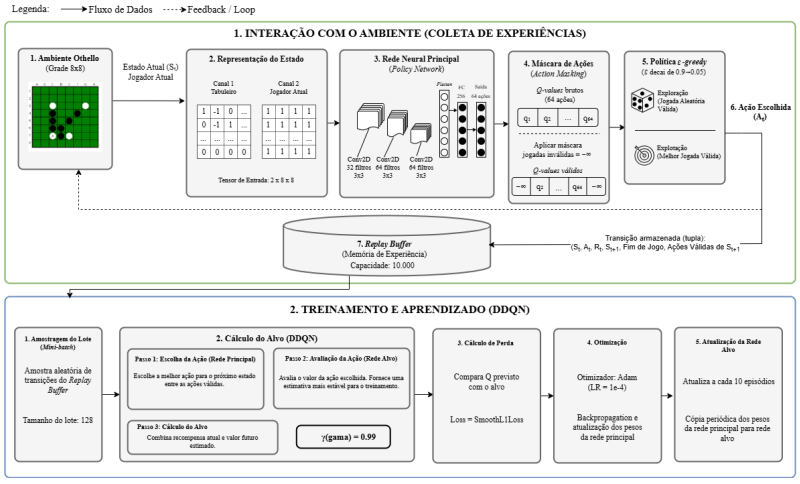


Figura 3. Arquitetura do Agente DDQN para Othello. (Fonte: Autoria própria)

3.3. Estratégia de Exploração e Cenários de Teste

Para solucionar o conflito entre exploração e explotação, adotou-se a política ϵ-*greedy*, na qual o agente seleciona a ação de maior valor Q na maioria das vezes, mas mantém

uma probabilidade ϵ de executar uma escolha aleatória. O parâmetro foi inicializado em 0.9 para forçar decisões exploratórias na fase inicial e permitir a descoberta de variadas configurações do tabuleiro. O decaimento de ϵ foi realizado de forma linear ao longo de 2.000.000 de passos de decisão, até atingir o valor mínimo de 0.05. Assim, nas etapas finais do treinamento, o agente passou a privilegiar a ação de maior valor Q, mantendo 5% de probabilidade de exploração.

Para avaliar o impacto dos hiperparâmetros na estabilidade e no desempenho do agente, adotou-se o algoritmo DDQN. A Rede Principal é utilizada para selecionar as ações e tem seus parâmetros atualizados durante o treinamento. A Rede Alvo, por sua vez, é utilizada para avaliar a ação selecionada no cálculo do valor alvo. Seus parâmetros são mantidos fixos entre as sincronizações e atualizados periodicamente a partir dos pesos da Rede Principal, a cada 10 episódios.

Com o intuito de analisar o comportamento dessa estrutura sob diferentes condições, foram modelados quatro cenários experimentais ao longo de 100.000 episódios cada:

- **Cenário 1:** Utilizou a abordagem de recompensa esparsa, taxa de aprendizado (α) de 1×10^{-4} e um decaimento exponencial rápido para o parâmetro de exploração (taxa de 1.000 passos).
- **Cenário 2:** Expandiu o período de exploração por meio de um decaimento linear longo, contudo elevou a taxa α para 1×10^{-3} para testar os limites de velocidade do algoritmo.
- **Cenário 3:** Manteve a estratégia de exploração por decaimento linear longo, mas reduziu a taxa α para 1×10^{-4} visando uma otimização mais estável.
- **Cenário 4:** Adotou os mesmos parâmetros de controle de passos e aprendizado do Cenário 3, porém substituiu a sinalização esparsa por uma abordagem de recompensa densa.

A Tabela 1 apresenta os hiperparâmetros que permaneceram invariáveis em todas as baterias de treino.

Tabela 1. Hiperparâmetros fixos nos diferentes cenários

Parâmetro	**Valor**	**Descrição**
Número de Episódios	100.000	Duração total da fase de treinamento.
Fator de Desconto (γ)	0.99	Alta valorização das recompensas futuras (visão de longo prazo).
Tamanho do Lote (Batch Size)	128	Quantidade de amostras processadas simultaneamente a cada passo.
Capacidade da Memória	10.000	Tamanho máximo do *Replay Buffer*.
ϵ Inicial e Final	$0.9 \rightarrow 0.05$	Taxa de decaimento da exploração.
Otimização da Rede Alvo	10	A cada 10 episódios

Fonte: Autoria própria

3.4. Protocolo de Avaliação

A avaliação do desempenho do agente foi realizada em duas etapas: métricas obtidas durante o treinamento e métricas aferidas em um ambiente de validação isolado.

Para estabelecer um referencial de comparação (*baseline*), simulou-se um torneio de 1.000 partidas entre dois agentes com políticas estocásticas. O primeiro jogador obteve 44,5% de vitórias, 50,4% de derrotas e 5,1% de empates. Esse resultado foi utilizado como referência para comparar o desempenho dos agentes treinados.

Durante a fase de treinamento contínuo, a principal métrica monitorada foi a evolução da taxa de vitórias ao longo dos episódios. Devido à alta variância inerente ao Aprendizado por Reforço e aos ruídos introduzidos pela política de exploração ϵ-*greedy*, a observação de recompensas em episódios isolados não fornece um panorama de convergência confiável. A aplicação de uma média móvel simples, calculada sobre uma janela estática de 5.000 episódios, permitiu a geração de curvas de aprendizado suavizadas. Nessa etapa, a estabilidade das curvas de aprendizado foi utilizada como indicador de estabilização do desempenho durante o treinamento.

A avaliação definitiva do modelo consistiu no teste prático do agente treinado instanciado a partir do arquivo de pesos salvo gerado no treino (.pth) em confrontos diretos contra um oponente de escolhas aleatórias. O agente foi submetido a um congelamento arquitetural: os parâmetros e pesos sinápticos foram fixados em modo de inferência, garantindo a interrupção da atualização de gradientes. Simultaneamente, o parâmetro de aleatoriedade foi estritamente anulado, forçando o agente a adotar uma política de decisão 100% determinística e gananciosa, pautada exclusivamente nas estimativas de valor Q consolidadas pela Rede Principal. Sob estas condições, o agente disputou 1.000 partidas ininterruptas. Esta etapa reduz a influência da exploração e permite avaliar o desempenho da política aprendida sob uma estratégia determinística contra o oponente aleatório. A Equação 3 demonstra o cálculo usado para obter a taxa de vitória:

$$WR = \left(\frac{NúmerodeVitórias}{TotaldePartidas} \right) \times 100 \tag{3}$$

4. Resultados e Discussões

O monitoramento do desempenho do agente durante a fase de treinamento baseou-se na evolução da taxa de vitórias, analisada por meio de curvas de aprendizado suavizadas para mitigar a alta variância inerente ao processo de Aprendizado por Reforço. Como etapa inicial para a validação empírica do sistema, estabeleceu-se um referencial de desempenho (*baseline*). Para isso, simulou-se um torneio de 1.000 partidas entre dois agentes com políticas de decisão puramente estocásticas, obtendo-se 445 vitórias (44,5%), 504 derrotas (50,%) e 51 empates (5,1%) para o primeiro jogador. A Tabela 2 sintetiza o desempenho comparativo dos cenários experimentais quando submetidos às mesmas condições de validação contra o oponente aleatório.

No Cenário 1, o decaimento rápido de ϵ reduziu a exploração ainda nas etapas iniciais do treinamento. A curva apresentada na Figura 4 mostra a estabilização do desempenho em um estágio mais inicial do aprendizado. Na validação, o modelo obteve 934 vitórias (93,4%), demonstrando desempenho superior à política aleatória.

Tabela 2. Valores das métricas aferidas em um ambiente de validação isolado

Configuração	Vitórias	Derrotas	Empates
Baseline Aleatória	445 (44.5%)	504 (50.4%)	51 (5.1%)
Cenário 1	934 (93.4%)	56 (5.6%)	10 (1.0%)
Cenário 2	722 (72.2%)	243 (24.3%)	35 (3.5%)
Cenário 3	940 (94.0%)	45 (4.5%)	15 (1.5%)
Cenário 4	933 (93.3%)	46 (4.6%)	21 (2.1%)

Fonte: Autoria própria

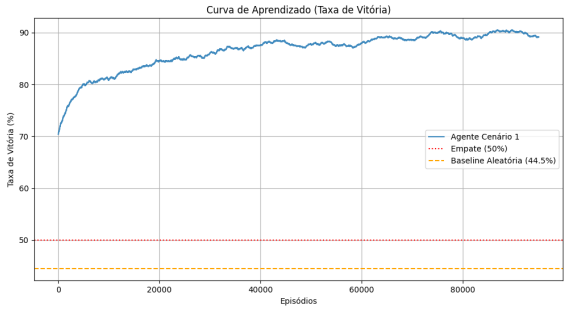


Figura 4. Curva de aprendizado Cenário 1. (Fonte: Autoria própria)

No Cenário 2, a transição linear prolongada permitiu uma evolução gradual do desempenho ao longo dos episódios, e o gráfico da Figura 5 demonstra esse crescimento. Entretanto, a maior taxa de aprendizado esteve associada a maior instabilidade na curva, indicando que as atualizações dos pesos ocorreram de forma mais acentuada. Esse comportamento refletiu-se no desempenho da validação, que apresentou 722 vitórias (72,2%), 243 derrotas (24,3%) e 35 empates (3,5%).

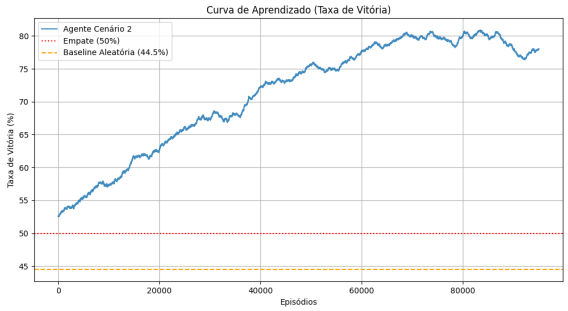


Figura 5. Curva de aprendizado Cenário 2. (Fonte: Autoria própria)

Com a redução da taxa de aprendizado no Cenário 3, os passos de otimização tornaram-se mais sutis, diminuindo a instabilidade do teste anterior. A curva de aprendizado da Figura 6 exibe maior estabilidade, traduzindo-se no melhor índice de validação do experimento, com 940 vitórias (94,0%) e apenas 45 derrotas (4,5%).

Como já é de conhecimento que no contexto do Aprendizado por Reforço, o objetivo do agente reside na maximização do sinal de recompensa. Então, a modelagem da função de recompensa assume um papel fundamental no direcionamento do compor-

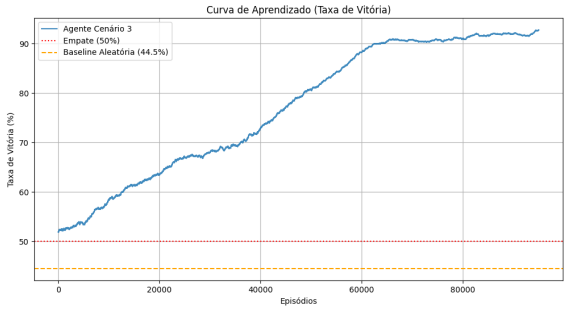


Figura 6. Curva de aprendizado Cenário 3. (Fonte: Autoria própria)

tamento aprendido. Considerando essa premissa, avaliou-se também o agente DDQN utilizando uma abordagem de Recompensa Densa, a Figura 7 demonstra sua curva de aprendizado. O agente do Cenário 4 obteve 933 vitórias (93,3%), 46 derrotas (4,6%) e 21 empates (2,1%). O resultado foi próximo ao obtido pelo Cenário 3, que apresentou 94,0% de vitórias, indicando que a recompensa densa não proporcionou melhoria significativa no desempenho final nas condições avaliadas.

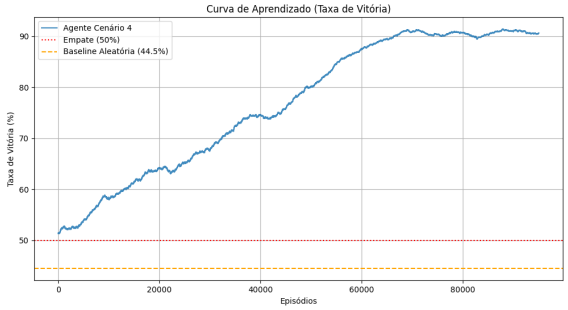


Figura 7. Curva de aprendizado Cenário 4. (Fonte: Autoria própria)

5. Conclusões e Trabalhos Futuros

Os resultados experimentais demonstram que o agente baseado em DDQN apresentou desempenho superior à política aleatória nas condições avaliadas, com o Cenário 3 obtendo a maior taxa de vitória, de 94,0%. A comparação entre os cenários indica que a taxa de aprendizado e a estratégia de exploração influenciaram o desempenho do agente, enquanto a utilização de recompensa densa não apresentou melhoria significativa em relação à formulação esparsa no conjunto de testes realizado.

Contudo, a principal limitação metodológica deste estudo reside no treinamento e na validação do agente exclusivamente contra um oponente estocástico e com o agente atuando como primeiro jogador. Esse cenário restringe a avaliação da capacidade do modelo de desenvolver estratégias defensivas e de lidar com diferentes estilos de jogo. Assim, para trabalhos futuros, sugere-se a implementação de confrontos contra adversários heurísticos e a realização de treinamento por *Self-Play*, permitindo avaliar se o agente é capaz de desenvolver políticas mais robustas diante de estratégias ofensivas e defensivas distintas.

Referências

Buro, M. (1997). The othello match of the year: Takeshi murakami vs. logistello. *ICGA Journal*, 20(3):189–193.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1-2):85–99.

Federação Brasileira de Othello ([2026]). Federação brasileira de othello.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32.

Silver, D., Huang, A., tug, C. J. M., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, K., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

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

Takizawa, H. (2023). Othello is solved. *arXiv preprint arXiv:2310.19387*.

Teymorian, M. (2023). Othello intelligent agent using double deep q-networks. Technical report, Yazd University.

van der Ree, M. and Wiering, M. (2013). Reinforcement learning in the game of othello: Learning against a fixed opponent and learning from self-play. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 108–115.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zheng, Z. et al. (2021). Expert q-learning: Integrating state values from expert examples into deep q-learning. *arXiv preprint arXiv:2106.14642*.