Resolução do Jogo de Blackjack via Deep Q-Learning no Ambiente Gymnasium Exame de Inteligência Artificial para Robótica Móvel (CT-213)

João Lucas Rocha Rolim
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
joao.rolim@ga.ita.br

Samir Nunes da Silva
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
samir.silva@ga.ita.br

Samuel Afonso de Souza Cavalcante
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
samuel.cavalcante@ga.ita.br

I. Introdução e Motivação

Blackjack ou Vinte e Um é um jogo comum em cassinos, no qual se utiliza, geralmente, de 1 a 8 baralhos de 52 cartas e se joga contra um *dealer* (a banca ou cassino). Nesse jogo, o objetivo é puxar cartas cuja soma é maior que a mão do *dealer*, sem que se supere o número 21. Assim, se o jogador ou o *dealer* tiverem soma acima de 21, termina-se o jogo e aquele cuja soma ficou abaixo de 21 vence [1].

O jogo de *Blackjack* é inerentemente probabilístico, já que não se controla quais cartas serão puxadas. No entanto, o jogador pode decidir se continua puxando uma carta (ação chamada de "*hit*") ou se para de puxar (ação chamada de "*stick*"). Assim, a escolha de qual ação tomar dadas as condições atuais do jogo impacta diretamente na probabilidade de vitória, de forma que a tomada de ações pode ser dirigida para gerar mais chances de vitória levando em conta o conhecimento sobre o jogo.

Tais características do *Blackjack* abrem portas para o uso de algoritmos de aprendizado na busca de maximizar as taxas de vitória contra o *dealer*. Para justificar tal afirmação, no contexto de geração de alto desempenho em jogos via inteligências artificiais, destaca-se o trabalho da empresa britânica DeepMind (cuja organização matriz é o Google) e de autoria de Mnih et al., por meio do qual atingiu desempenho super-humano em jogos de Atari, como no famoso *Pong*, através do algoritmo *Deep Q-Networks* (DQN) de *Reinforcement Learning* [2]. Tal algoritmo utiliza redes neurais profundas para estimar a função ação-valor, assim estabilizando o algoritmo livre de modelo *Q-learning*.

Logo, no presente artigo, buscou-se implementar um modelo para jogar partidas de *Blackjack* treinado via *Deep Q-Learning*, baseando-se em uma estruturação prévia do problema no ambiente *Gymnasium* [3] [4], na linguagem *Python*. O ambiente do problema em questão corresponde à versão descrita no Exemplo 5.1 do livro de Sutton e Barto [5] e a Figura 1 mostra o jogo de *Blackjack* ocorrendo nesse ambiente. Como há forte componente estocástico nos resultados possíveis, sendo impossível obter sucesso em 100% das partidas, buscou-se atingir um nível de acerto maior que um modelo com política aleatória, alcançando-se, com alta probabilidade, média de acerto igual ou maior que jogadores humanos.

Fig. 1. Jogo de *Blackjack* ocorrendo no ambiente *Gymnasium*.

II. Fundamentação Teórica

A. Regras do Jogo de Blackjack

O cenário envolvido em um jogo de *Blackjack* pode variar de acordo com a modalidade e quais das regras são adotadas. Neste artigo, restringiu-se a complexidade das partidas a fim de simplificar o modelo a ser determinado. Nesse sentido, como não há necessidade de incluir outros jogadores, a "mesa" de jogo inclui apenas 1 jogador - cujas decisões são regidas pelo modelo a ser aprendido - e o *dealer* - no caso, papel realizado pelo código do jogo e que condiciona o andamento da partida, ao mesmo tempo que atua como oponente para o jogador. Além disso, as ações possíveis do jogador se restringem a *hit*

e *stick* e utiliza-se um baralho de cartas infinito, cujas retiradas são feitas com reposição.

A pontuação de cada carta, em maior parte, se baseia simplesmente no número correspondente à carta, de 2 a 9. Já as cartas de figuras - rei *(K)*, valete *(J)* e rainha *(Q)* - valem 10 pontos, enquanto o ás *(A)* vale 11 pontos se a soma da mão com o ás valendo 11 pontos for menor ou igual a 21, do contrário, vale apenas 1 ponto.

Quanto ao fluxo da partida, o jogo se inicia com o *dealer* revelando em seu lado do campo uma carta de face para cima (com seu valor à mostra) e uma de face para baixo (com seu valor oculto), enquanto o jogador tem duas cartas de face para cima. Em seguida, o jogador pode realizar uma de duas ações: dar *hit*, adicionando uma carta do baralho ao seu lado da mesa e somando seu valor às cartas anteriormente em sua posse; e dar *stick*, parando de puxar cartas e estabelecendo o valor da soma de suas cartas que será sua pontuação. A partir do início da tomada de decisão, o jogador pode repetir essa escolha até que ele escolha *stick* ou até que sua soma ultrapasse o valor 21, automaticamente garantindo-o uma derrota.

Dado que o jogador escolheu parar de puxar cartas com *stick*, o *dealer* revela sua carta virada para baixo e, se a soma dos valores de suas cartas for menor que 17, puxa cartas até sua soma ser 17 ou maior. Caso o *dealer* não supere o valor de 21 (caso em que automaticamente perde a partida), sua pontuação é comparada com a do jogador, e a mais próxima de 21, sem excedê-lo, garante o vencedor. Caso as pontuações sejam iguais, considera-se empate.

Nesse contexto, ainda é possível que as primeiras duas cartas do jogador possuam pontuação 21 (um Ás e uma carta de valor 10). O *dealer*, com isso, revela sua segunda carta, e, caso ele também não possua duas cartas de soma 21 (caso de empate), o jogador automaticamente ganha. Nesse caso, a combinação das duas cartas iniciais com soma 21 é chamado de *natural blackjack*.

B. Deep Q-Learning

A abordagem utilizada para o estudo do problema se baseia no método de *Deep Q-Learning (DQN)*, que utiliza o algoritmo *Q-Learning* juntamente de redes neurais profundas para estimativa de sua função ação-valor.

1) Q-Learning: O algoritmo *Q-Learning*, segundo Sutton [5], é um exemplo de *Temporal Difference (TD) Learning*, que consiste em um método de aprendizagem livre de modelo que utiliza *bootstrapping*, isto é, que começa o aprendizado com uma estimativa inicial para o valor estimado pelo retorno médio V_s, em que s representa o estado atual. No entanto, para que um aprimoramento guloso da política $\pi(s)$ não necessite de um modelo do processo decisório de Markov (MDP - Markov Decision Process) do sistema, utiliza-se como atualização gulosa da política a estimativa da função ação-valor $Q(s,a)$, conforme mostrado na Equação 1, em que A representa o conjunto de todas as ações possíveis. Nesse contexto, define-se a função ação-valor condicionada à política π por meio da Equação 2, na qual t representa um certo passo de tempo. No contexto do algoritmo, um passo de tempo se refere a um passo de um episódio, referente a uma jogada, por exemplo, e sendo um episódio uma partida de Blackjack completa.

$$\pi'(s) = argmax_{a \in A} Q(s,a) \tag{1}$$

$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a] \tag{2}$$

Tal algoritmo também se baseia no conceito de política ε-*greedy* (política ε-gulosa), que se refere à atualização de política que garante exploração contínua do espaço de políticas. Tal método, de simples implementação, escolhe a ação gulosa com probabilidade $1-\varepsilon$ e escolhe uma ação aleatoriamente com probabilidade ε. Assim, sendo m o número total de ações que podem ser tomadas, a atualização de política ε-*greedy* é dada pela Equação 3.

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{m} + 1 - \varepsilon, \ a = argmax_{a' \in A} Q(s,a') \\ \frac{\varepsilon}{m}, \ caso\ contrário \end{cases} \tag{3}$$

Ademais, *Q-Learning* se trata de uma forma de aprendizado *off-policy*, na qual o modelo aprende uma política π enquanto executa a política μ, ou seja, aprende π através de experiências amostradas de μ. No caso específico de *Q-Learning*, a política de comportamento μ é ε-*greedy* e a política alvo ótima π é gulosa. Mesmo não sendo necessário reduzir ε ao longo do aprendizado, já que *Q-learning* aprende a política ótima diretamente, optou-se por uma taxa de decaimento de ε pequena de tal maneira que a atualização desse valor é representada pela Equação 4, em que ε_{t+1} representa o valor no passo de tempo $t+1$, ε_t o valor no passo de tempo t e $\Delta\varepsilon$ a variação de ε a cada passo.

$$\varepsilon_{t+1} = max(\varepsilon_f, \varepsilon_t + \Delta\varepsilon) \tag{4}$$

O algoritmo, ainda, utiliza a equação de otimalidade de Bellman com amostragem para compor a atualização da estimativa da função ação-valor $Q(S_t,A_t)$, a qual é feita através da Equação 5, na qual α é a taxa de aprendizado do modelo, R_{t+1} é a recompensa no passo de tempo $t+1$ e γ é o fator de desconto. Tanto α quanto γ são escolhidos como constantes do modelo.

$$\begin{aligned} Q(S_t,A_t) = & Q(S_t,A_t) \\ & + \alpha(R_{t+1} + \gamma max_{a' \in A} Q(S_{t+1},a') \\ & - Q(S_t,A_t)) \end{aligned} \tag{5}$$

Com isso, chega-se no seguinte pseudocódigo para implementação do algoritmo de *Q-Learning* [6]:

0: Inicializar Q(s,a) arbitrariamente.
0: Inicializar μ = ε-*greedy*(Q).
0: **for** cada episódio **do**:
0: Inicializar estado S
0: A $\sim \mu(a|S) = \varepsilon$-*greedy*(Q)
0: **for** cada passo do episódio **do**:

0: Tomar ação A, observar R, S'
0: Atualizar Q(S,A) através da Equação 5
0: S = S'
0: **end for**
0: Até o fim do episódio
0: **end for**=0

2) Definição de Deep Q-Learning: Há várias formas de se criar um aproximador de função, dentre elas: combinação linear de *features*, uso de funções não-lineares e uso de redes neurais. Em particular, o algoritmo de *Deep Q-Learning* consiste no uso de uma rede neural profunda como aproximador de função para estimar a função ação-valor $q_\pi(s,a)$. Tal aproximador é caracterizado por um conjunto de parâmetros w que são aprimorados de forma a respeitar a Equação 6. Suas grandes vantagens são a resolução de processos decisórios de Markov com muitos estados, considerando, por exemplo, espaços de ações e estados contínuos, e a generalização de seu aprendizado para estados próximos.

$$\hat{q}(s,a,\mathbf{w}) \approx q_\pi(s,a) \tag{6}$$

Para seu funcionamento, o algoritmo de *Deep Q-Learning* se utiliza de dois conceitos essenciais, além daqueles apresentados para *Q-Learning*: *experience replay* e *fixed Q-targets*.

Experience replay se refere à prática de armazenar transições na forma $(S_t, A_t, R_{t+1}, S_{t+1})$ em um *replay buffer* D de tamanho definido e, em seguida, para cada atualização no treinamento, amostrar um *mini-batch* aleatório, de tamanho definido, de transições (s,a,r,s') de D. Isso, em particular, torna os dados de treinamento da rede neural não-sequenciais, ajudando a estabilizá-lo [5].

Por sua vez, *fixed Q-targets* se trata da criação de uma cópia da rede durante o treinamento e fixação de seus pesos em w^-. Com isso, treina-se a rede utilizando a função de custo descrita pela Equação 7. Tal prática auxilia a tornar os dadas não-estacionários, o que ajuda na convergência da rede neural [5].

$$L_i(\mathbf{w}) = E_{s,a,r,s' \sim D_i}[(r + \gamma max_{a' \in A} \hat{q}_\mathbf{w}(s',a') - \hat{q}_\mathbf{w}(s,a))^2] \tag{7}$$

Portanto, com todos os conceitos apresentados, é possível construir o algoritmo de *Deep Q-Learning*. No entanto, é necessário também realizar o *tuning* dos hiperparâmetros do modelo, os quais são: a taxa de aprendizado α, o fator de desconto γ, o valor inicial, a taxa de decaimento e o valor final do fator ε da política ε-*greedy*, e o tamanho do *buffer* e dos *mini-batches* do *experience replay*). Além disso, é fundamental a utilização de uma *reward engineering* que auxilie no aprendizado do modelo durante os passos de um episódio, e a definição de uma arquitetura funcional de rede neural para ser utilizada na aproximação da função ação-valor do modelo, ou seja, a criação de uma rede neural suficientemente complexa que estime bem o valor da função, mas que não demore muito para processar seus cálculos.

III. Implementação

A implementação do algoritmo *Deep Q-Learning* foi realizada na linguagem *Python* e a simulação do jogo *Blackjack* foi feita no ambiente disponível na biblioteca *Gymnasium*.

A. Agente e Rede Neural

Primeiramente, desenvolveu-se a classe do agente `DQNAgent` (*Deep Q-Network Agent*) com seus hiperparâmetros. Nessa classe, criou-se a rede neural para estimativa da função ação-valor através da API *Keras* da bibliotecas *Tensorflow*. A rede foi feita com uma camada de *input* que recebe uma entrada de dimensão 3 (igual ao número de dimensões do espaço de observações), duas *hidden layers* de 24 neurônios cada com ativações ReLU (Rectified Linear Unit) e uma camada de *output* com 2 neurônios, a qual retorna os valores estimados da função ação-valor para cada uma das duas ações possíves: *hit* e *stick*. O sumário da rede neural implementada é mostrado na Figura 2. Ademais, utilizou-se *Adam optimizer* como método para a descida de gradiente estocástica da rede e escolheu-se como função de custo o *Mean Squared Error* (erro quadrático médio). Não se utilizou a função de custo da Equação 7 referente *Fixed Q-targets* porque a rede neural do problema é relativamente simples e já suficiente para gerar boas estimativas da função ação-valor.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 24)                96

 dense_1 (Dense)             (None, 24)                600

 dense_2 (Dense)             (None, 2)                 50

=================================================================
Total params: 746 (2.91 KB)
Trainable params: 746 (2.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 2. Sumário da rede neural implementada para o algoritmo de *Deep Q-Learning* no contexto do jogo de Blackjack.

Também implementou-se na classe do agente funções para retornar as ações gulosa e ε-*greedy*, e outras para realizar o *experience replay*, para decair o valor de ε e para salvar e carregar os pesos de treinamentos anteriores. Com isso, o agente ficou pronto para ser treinado.

B. Treinamento e Reward Engineering

Pàra o treinamento do agente no jogo de *Blackjack*, utilizou-se o algoritmo *Deep Q-Learning* juntamente de uma *reward engineering* para melhorar seu aprendizado ao longo dos passos dos episódios, totalizando 2000 episódios de treinamento. Além disso, definiu-se os hiperparâmetros do modelo conforme a Tabela I.

TABELA I
VALORES DOS HIPERPARÂMETROS DO MODELO.

Hiperparâmetro	Valor
Taxa de aprendizado α	0,01
ε inicial	1,0
$\Delta\varepsilon$	0,001
ε final	0,1
Fator de desconto γ	1,0
Tamanho do *replay buffer*	4096
Tamanho dos *mini-batches*	32

A *reward engineering* foi criada levando em conta os objetivos intermediários e finais de um jogo de Blackjack e tomou como base as recompensas padrões do ambiente *Gymnasium*: +1,5 se o jogo for vencido com *natural blackjack*, +1 se for vencido normalmente, 0 se ocorrer empate e -1 se o modelo perder a partida [3].

Nesse contexto, primeiramente ampliou-se a escala das recompensas em 100 vezes para se utilizar valores mais discrepantes dependendo das condições nas quais o modelo tomasse uma ação, sem depender de muitas casas decimais. Além disso, buscou-se valores para que o agente aprendesse mais rapidamente mesmo com uma taxa de aprendizado pequena. Escolheu-se, assim, três formas de recompensar ou punir o modelo: recompensa pelo valor final da mão, limiar abaixo do qual deve realizar um *hit* a depender da carta à mostra do *dealer* e incentivo por realizar *hit* na presença de um ás inicial.

A recompensa pelo valor final da mão se baseou no fato de que se a próxima soma da mão do agente for superior a 21 quando o jogo terminar, ele deve ser punido pelo quão distante essa soma ficou de 21 (primeiro caso). Além disso, ele é recompensado por quão perto sua soma, quando menor ou igual a 21, ficou do limite de estourar (segundo caso). Essa recompensa é modelada, no primeiro caso, pela Equação 8, e, no segundo caso, pela Equação 9, nas quais r_{new} representa a recompensa atualizada, r_{old} a recompensa antiga e *nt* (*next total*) a próxima soma da mão do jogador.

$$r_{new} = r_{old} - 10 \cdot (nt - 21) \tag{8}$$

$$r_{new} = r_{old} + 50 \cdot \left(\frac{nt}{21}\right)^4 \tag{9}$$

Por sua vez, a recompensa pelo limiar abaixo do qual se deve realizar a ação de *hit* leva em conta a carta à mostra do *dealer* e define recompensas para cada um dos valores dessa carta, a depender de quão longe do limiar definido o total da mão do jogador está. Tal recompensa leva em conta estratégias baseadas nas estatísticas do jogo de *Blackjack* [7], e é modelada pela Equação 10 se o agente realizar a ação de *stick* e pela Equação 11 se adotar a ação de *hit*. Nelas, *ht* (*hit threshold*) é o vetor *zero-based* [17, 11, 13, 8, 8, 8, 16, 16, 16, 17] e *dc* (*dealer card*) é a pontuação correspondente à carta à mostra na mão do *dealer*.

$$r_{new} = r_{old} - 5 \cdot (ht[dc-1] - 1 - total) \tag{10}$$

$$r_{new} = r_{old} + 5 \cdot (ht[dc-1] - total) \tag{11}$$

Finalmente, o incentivo por realizar *hit* na presença de um ás inicial consiste na ideia de que é muito vantajoso aumentar a pontuação na presença de um ás inicial [7], já que ele sempre irá se configurar como um ás usável, isto é, valendo 11 pontos. Nesse caso, a atualização de recompensa é dada pela Equação 12.

$$r_{new} = r_{old} + 20 \tag{12}$$

Para verificar como as recompensas do modelo evoluíram ao longo dos episódios de treinamento, criou-se gráficos da recompensa e da média móvel exponencial das recompensas com constante $\alpha_{mme} = 0.02$. A média móvel exponencial é definida por meio da Equação 13, em que mme_{new} é a nova média móvel, mme_{last} é a última média móvel calculada antes da nova, r é a recompensa obtida em um novo episódio e α_{mme} é a constante da média móvel que define quanto o valor de recompensa atual influencia na média móvel.

$$mme_{new} = (1 - \alpha) \cdot mme_{last} + \alpha \cdot r \tag{13}$$

C. Avaliação do Modelo

A avaliação do modelo é feita dentro de 500 episódios e com base na comparação entre uma política aleatória e a política do modelo treinado via *Deep Q-Learning*. Para tal comparação, são usadas as métricas de *Win Percentage* (porcentagem de vitórias), *Tie Percentage* (porcentagem de empates), *Loss Percentage* (porcentagem de derrotas), *Overflow Count* (contagem de quantas vezes o jogador passou de 21) e *Overflow Amount* (soma de todas as diferenças a 21 das pontuações finais do jogador nos episódios). Além disso, obteve-se o gráfico da média móvel exponencial das recompensas padrões do ambiente *Gymnasium* ao longo dos episódios de avaliação para comparação entre as políticas.

IV. RESULTADOS E DISCUSSÕES

A. Treinamento

A Figura 3 mostra as recompensas obtidas pelo agente ao longo de seus 2000 episódios de treinamento via *Deep Q-Learning*, os quais demoraram em média 50 minutos em um *notebook* Lenovo Legion 5 82QJ0000BR com processador Amd Ryzen 7 Series 5000. Nota-se que os resultados são muito ruidosos, já que o jogo de *Blackjack* é naturalmente estocástico, sendo muito difícil verificar o aprimoramento do agente ao longo dos episódios. Nesse sentido, a visualização da média móvel exponencial das recompensas ao longo do treinamento, mostrada na Figura 4, é muito mais apropriada. O gráfico da média móvel indica que houve aumento das recompensas do agente com o treinamento e que ocorreu um crescimento praticamente contínuo da recompensa média entre 125 e 650 episódios, aproximadamente. Após isso, a média móvel oscilou próxima de 40, não tendo melhorias significativas em relação a esse valor ao fim do treinamento.

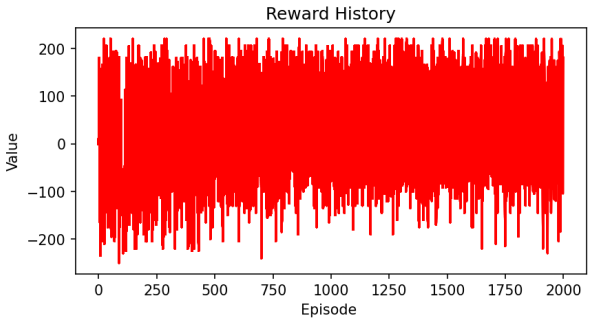

Fig. 3. Recompensas ao longo dos 2000 episódios de treinamento do agente via *Deep Q-Learning*.

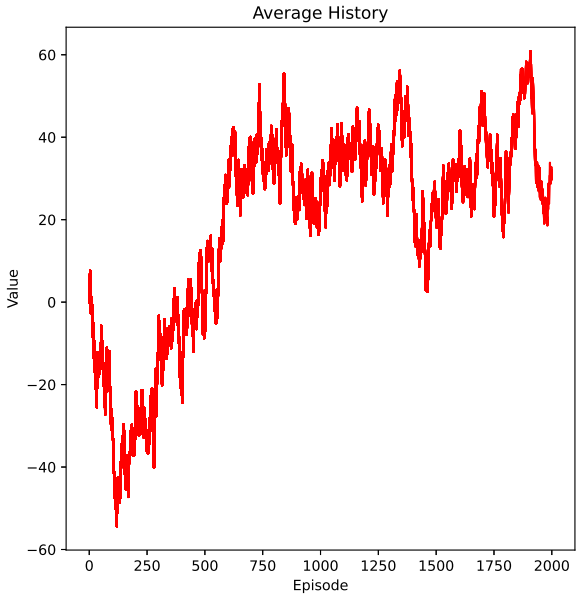

Fig. 4. Média móvel exponencial das recompensas ao longo dos 2000 episódios de treinamento do agente via *Deep Q-Learning*.

B. *Avaliação da Política Aleatória*

Os resultados das métricas para a política aleatória no jogo de *Blackjack* são mostradas na Tabela II. Além disso, o gráfico da média móvel exponencial das recompensas padrões do ambiente *Gymnasium* é mostrado na Figura 5.

TABELA II
MÉTRICAS OBTIDAS PARA A POLÍTICA ALEATÓRIA NO JOGO DE BLACKJACK.

Métrica	Valor
Win Percentage	29,7%
Tie Percentage	3,6%
Loss Percentage	66,7%
Overflow Count	143
Overflow Amount	687

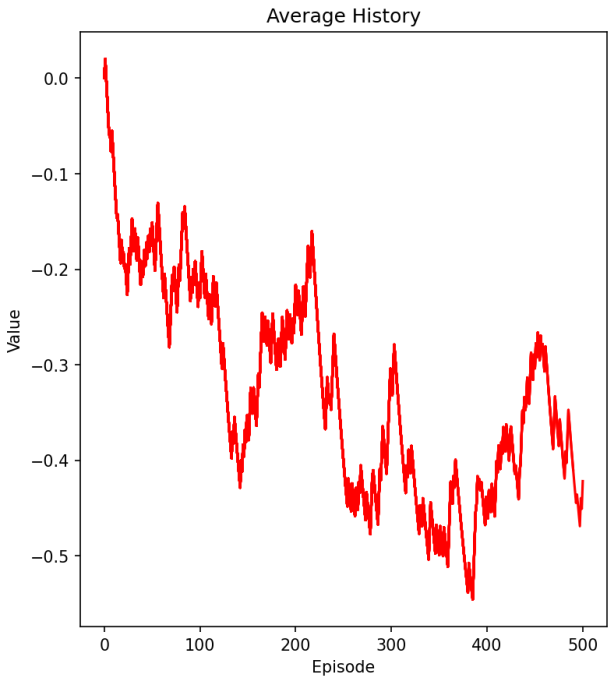

Fig. 5. Média móvel exponencial das recompensas padrões do ambiente *Gymnasium* ao longo da avaliação da política aleatória no jogo de *Blackjack*.

C. *Avaliação do Modelo treinado via* Deep Q-Learning

Os resultados das métricas para a política do agente treinado via *Deep Q-Learning* no jogo de *Blackjack* são mostradas na Tabela III. Ademais, o gráfico da média móvel exponencial das recompensas padrões do ambiente *Gymnasium* é mostrado na Figura 6.

TABELA III
MÉTRICAS OBTIDAS PARA A POLÍTICA ALEATÓRIA NO JOGO DE BLACKJACK.

Métrica	Valor
Win Percentage	41,3%
Tie Percentage	8,2%
Loss Percentage	50,5%
Overflow Count	30
Overflow Amount	46

D. *Comparações nas Avaliações das Políticas*

Comparando-se as métricas das tabelas II e III, verifica-se que houve melhoria substancial de todos os valores após o treinamento do agente com o algoritmo de *Deep Q-Learning*.

A porcentagem de vitória aumentou de 29,7% para 41,3%, representando uma grande melhoria. Deve-se levar em conta a probabilidade de vitória no jogo de *Blackjack* pode ser tão alta quanto 42,22% [8], ou seja, o modelo treinado foi, na média, praticamente ótimo em termos de números de vitórias.

Por sua vez, a porcentagem de empates aumentou de 3,6% para 8,2%. Tal resultado também representa um excelente aprimoramento, tendo-se em mente que, em média, 8,48% dos jogos de *Blackjack* terminam em empate [8], ou seja, o modelo

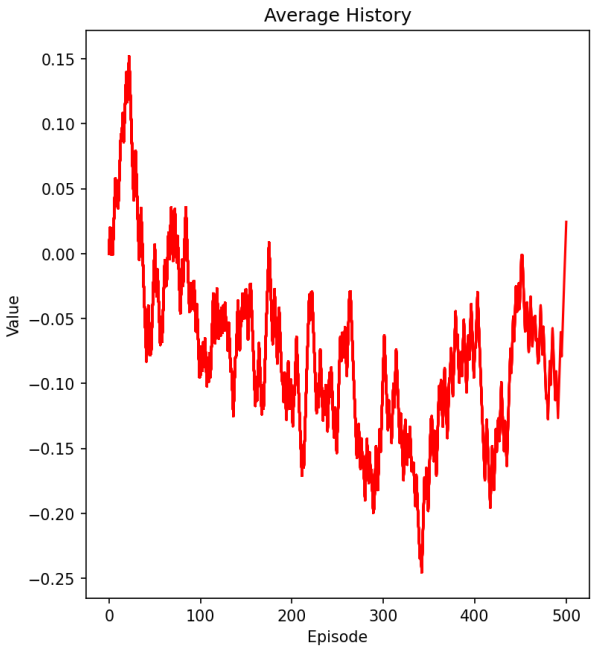

Fig. 6. Média móvel exponencial das recompensas padrões do ambiente *Gymnasium* ao longo da avaliação da política do agente treinado via *Deep Q-Learning* no jogo de *Blackjack*.

quase atingiu o máximo de empates possível em seus jogos, reduzindo bastante a taxa de derrotas, lembrando também que a taxa de vitórias foi praticamente máxima.

Já a taxa de derrotas diminuiu de 66,7% para 50,5%, sendo esse valor resultado dos grandes aumentos nas taxas de vitória e empate. Na média, a probabilidade de se perder um jogo de *Blackjack*, considerando a máxima taxa de vitória possível, é de 49,10% [8]. Assim, conclui-se que o modelo performou perto da otimalidade em termos de taxas de vitória, derrota e empate.

As métricas de *Overflow Count* e *Overflow Amount* também indicam o sucesso do treinamento do agente no jogo. *Overflow Count* diminuiu de 143 para 30, representando uma redução de 79,0% na quantidade de vezes que o agente possuiu soma total de sua mão maior que 21 ao longo de 500 jogos. Ainda, *Overflow Amount* diminuiu de 687 para 46, indicando redução de 93,3% na soma de todas as diferenças a 21 das pontuações finais do jogador nos episódios ao longo de 500 jogos, significando que, em média, o agente ficou muito mais próximo de atingir 21 em suas partidas.

Por fim, a comparação entre os gráficos das Figuras 5 e 6 mostra claramente que a média móvel exponencial das recompensas padrões do modelo treinado via *Deep Learning* ao longo dos 500 jogos de avaliação foi superior à da política aleatória, o que reflete os resultados já encontrados nas métricas de porcentagem de vitória, derrota e empate.

V. Conclusões

Tanto a etapa de treinamento quanto a de avaliação do agente treinado através do algoritmo de aprendizado por reforço *Deep Q-Learning* indicam sua melhoria substancial no jogo de *Blackjack*, atingindo desempenho praticamente ótimo em termos de taxas de vitória, empate e derrota contra o *dealer*, quando comparadas com as estatísticas gerais do jogo.

O presente trabalho é importante no sentido de mostrar a capacidade que os algoritmos de aprendizado por reforço livres de modelo atingiram quando unidos à área de redes neurais, permitindo o aprendizado e resolução, em tempo hábil, de vários jogos complexos e até mesmo inerentemente estocásticos, como é o caso do *Blackjack*.

Por último, o trabalho pode ser estendido ao se considerar outras formas de *reward engineering* para melhorar a velocidade de aprendizado do agente. Por exemplo, pode-se levar em conta casos intermediários mais complexos de ás usável e considerar outras heurísticas usadas por jogadores profissionais tendo em mente a soma total da mão do jogador. Além disso, a extensão pode ocorrer no sentido de utilizar outras variantes do jogo de *Blackjack* e também um número limitado de baralhos no jogo, o que adicionaria mais graus de complexidade tanto na forma como a *reward engineering* seria feita como no progresso do treinamento do agente.

Referências

[1] Wikipédia, "Blackjack — wikipédia, a enciclopédia livre," 2023, [Online; accessed 11-julho-2023]. [Online]. Available: https://pt.wikipedia.org/w/index.php?title=Blackjack&oldid=65907982

[2] V. M. et al., "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[3] Gymnasium Documentation, "Blackjack," 2022, made with Sphinx and @pradyunsg's Furo [accessed 11-julho-2023]. [Online]. Available: https://www.gymlibrary.dev/environments/toy_text/blackjack/

[4] ——, "Solving blackjack with q-learning," 2022, copyright © 2022 Farama Foundation [Online; accessed 11-julho-2023]. [Online]. Available: https://gymnasium.farama.org/tutorials/training_agents/blackjack_tutorial/

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018. [Online]. Available: http://www.incompleteideas.net/book/RLbook2020.pdf

[6] M. R. O. A. Maximo, "Slides da disciplina do ita de ct-213: Inteligência artificial para robótica móvel," 2023.

[7] Grosvenor Cassino, "Blackjack strategy: Hit or stand," 2021. [Online]. Available: https://blog.grosvenorcasinos.com/blackjack-strategy-hit-or-stand

[8] Mr Blackjack, "Blackjack odds & house edge explained," 2021. [Online]. Available: https://www.onlinegambling.com/blackjack/odds/