O Impacto da Função de Recompensa no Treinamento de Agentes DQN em Jogos 2D

Lucas M. Tiago[1], **Daniel D. Abdala**[2]

[1]Faculdade de Computação – Universidade Federal de Uberlândia (UFU)
Caixa Postal 15.064 – 91.501-970 – Uberlândia – MG – Brazil

lucas.mpt@ufu.br, abdala@ufu.br

Abstract. *The development of intelligent agents in games has attracted growing interest, due to recent advances in artificial intelligence. Although algorithms like Deep Q-Network (DQN) exhibit remarkable performance, their implementation is not a trivial task. This paper analyzes the framework for creating and training these agents, highlighting the reward function as the critical component to be adapted to each environment. To validate this analysis, three distinct agents were created, trained and evaluated.*

Resumo. *O desenvolvimento de agentes inteligentes em jogos tem atraído crescente interesse, devido aos recentes avanços em inteligência artificial. Embora algoritmos como Deep Q Network (DQN) apresentem notável desempenho, sua implementação não é uma tarefa trivial. Este trabalho analisa o framework de criação e treinamento desses agentes, destacando a função de recompensa como o componente crítico a ser adaptado em cada ambiente. Para validar essa análise, três agentes distintos foram criados, treinados e avaliados.*

1. Introdução

O aprendizado por reforço (Reinforcement Learning - RL) [Sutton and Barto 2018] é uma abordagem proeminente para treinar modelos de inteligência artificial (IA) em jogos eletrônicos, destacando-se de outros métodos pela interação constante entre agente e ambiente. Ao explorar e interagir com o ambiente continuamente, os agentes recebem recompensas ou penalidades dependendo das ações tomadas. Jogos são amplamente utilizados como base de testes em várias áreas da tecnologia, sendo uma das principais para validar o desempenho e eficiência de algoritmos de IA [Hu et al. 2024, Shao et al. 2019], pois representam ambientes complexos com objetivos e regras bem definidos.

Trabalhos como o de [Mnih et al. 2015] comprovaram a eficiência do algoritmo Deep Q-Learning (DQN) em diversos jogos de Atari. No entanto, o foco do estudo não foram as etapas do treinamento, mas sim a comprovação da eficiência do algoritmo de forma genérica em vários casos. Em contrapartida, este trabalho busca mostrar que a função de recompensa é o componente mais importante a ser adaptado ao jogo no qual o agente será treinado, tornando o treinamento para qualquer jogo 2D uma tarefa direcionada.

2. Metodologia

Foi utilizada a linguagem Python, devido à sua ampla adoção na área de IA e à grande disponibilidade de bibliotecas úteis como o PyTorch [PyTorch Foundation 2023]. A Figura 1 apresenta as principais etapas de execução.

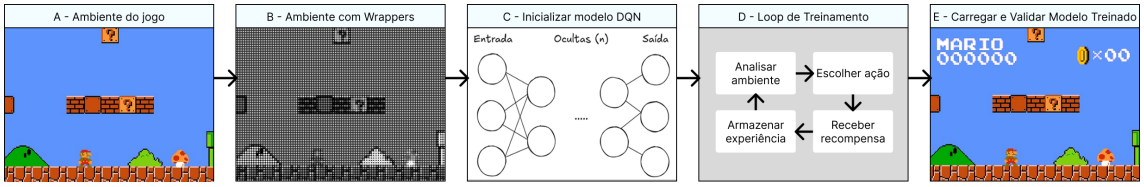


Figura 1. Diagrama representando as etapas de execução do código.

2.1. A - Ambiente do Jogo

O ambiente de treinamento é inicializado utilizando a biblioteca Gym [Brockman et al. 2016] a uma taxa de 60 quadros por segundo, juntamente com os espaços de ação (movimentos possíveis) e de observação (imagens da tela). Essas observações passam por um pré-processamento antes de serem enviadas à rede neural.

2.2. B - Wrappers de Ambiente

Para acelerar o treinamento e otimizar o código, foram implementados Wrappers no ambiente, os quais reduzem a resolução geral do jogo, normalizam e ajustam os valores de cada pixel, da recompensa e espaços de ações do jogo. Dentre todos os Wrappers, o mais difícil de se adaptar é o **GameReward( )** $\in [-15, 15]$: como alguns ambientes não possuem sistema de recompensa próprio, torna-se necessário criar uma função que calcula e retorna esse valor. Contudo, essa não é uma tarefa trivial, já que cada jogo possui uma série de objetivos únicos que devem ser alcançados.

2.3. C - Inicialização do Modelo DQN

Define-se a arquitetura da rede DQN e os hiperparâmetros, listados na Tabela 1. A instanciação do algoritmo foi feita por meio da distribuição Xavier Uniforme, enquanto os vieses (bias) receberam o valor constante de 0,01.

Tabela 1. Camadas da rede DQN.

Camada	Tipo	Formato da Saída
Input	Imagem	(4, 84, 84)
Conv1	Convolucional	(32, 20, 20)
Conv2	Convolucional	(64, 18, 18)
FC1	Densa	512
Fluxos da Dueling DQN		
Value Stream	Densa	1
Advantage Stream	Densa	12 (Ações)

2.4. D - Loop Principal

O treinamento ocorre em um loop iterativo de episódios, onde cada episódio representa uma única tentativa do agente de terminar a fase. Foi utilizado o algoritmo Dueling DQN [Wang et al. 2016] com ϵ-greedy e armazenamento em memória de replay, onde a memória é atualizada apenas após um número mínimo de passos (warm-up).

2.5. E - Carregar e Validar Modelo do Arquivo

Após o treinamento, o agente é salvo juntamente com seus pesos e a estrutura das camadas, para posteriormente ser avaliado no mesmo ambiente com wrappers.

3. Experimentos e Resultados

Para padronização, os mesmos valores de hiperparâmetros foram utilizados em todos os experimentos para os Wrappers, indicados na Tabela 2.

Tabela 2. Hiperparâmetros de ambiente utilizados nos experimentos.

Hiperparâmetro	Valor
Ações Nulas Máximas (noopMax)	30
Pulo de Frames (frameSkip)	4
Redimensionamento de Imagem (size)	84x84 pixels
Empilhamento de Frames (k)	4 frames

Para cada experimento, utilizou-se um jogo diferente: Super Mario Bros., pela fácil implementação; Sonic the Hedgehog, similar ao anterior; e R-Type, um jogo de nave diferente dos dois primeiros. Os parâmetros de treinamento e as telas dos jogos podem ser vistos na Figura 2. Vale ressaltar que o ϵ representa a probabilidade do agente escolher uma ação aleatória e o Decaimento ϵ, o número de épocas até o ϵ atingir o valor mínimo.

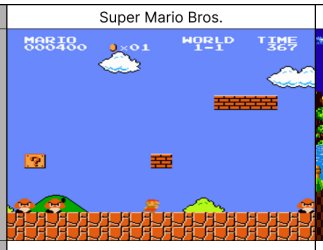
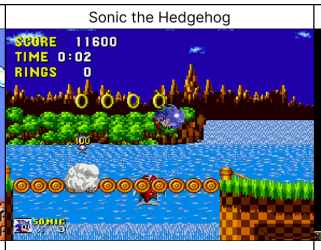
Nome do jogo	Super Mario Bros.	Sonic the Hedgehog	R-TYPE
Imagem do jogo			
Taxa de Aprendizagem (Learning Rate)	0,001	0,001	0,0005
Tamanho do Lote (Batch Size)	64	64	128
Capacidade da Memória	100.000	100.000	100.000
Epsilon Inicial	0,85	0,01	1,0
Epsilon Final	0,001	0,01	0,01
Decaimento do Epsilon	500	0	6.000
Fator de Desconto	0,99	0,99	0,99
Atualização da Rede Alvo (por época)	50	50	100
Passos de Aquecimento (Warm-up)	1000	2000	1000

Figura 2. Hiperparâmetros de treinamento e seus respectivos jogos.

3.1. Super Mario Bros - NES

O ambiente utilizado foi do Super Mario Bros., um jogo de plataforma em que o objetivo é avançar do começo da fase até o final sem morrer, movendo-se da esquerda para a direita enquanto evita obstáculos. A biblioteca Gym-Super-Mario-Bros [Kauten 2018] possui função de recompensa própria, a qual assume que o objetivo principal do agente é chegar ao fim no menor tempo possível e sem morrer:

$$\left.\begin{aligned} V &= x_1 - x_0 \\ C &= c_0 - c_1 \end{aligned}\right\} \quad \text{Recompensa} = V + C + D$$

- V: Deslocamento horizontal; quanto mais para a direita andar maior o valor de V.
- C: Diferença do tempo do relógio no jogo; para cada segundo passado $C = -1$.
- D: Penalidade por morte: se o agente está vivo, $D = 0$; caso contrário $D = -15$.

A função de recompensa dita o comportamento do agente, pois é ela quem indica ao agente que ele deve se mover para a direita, penalizando mortes e inatividade.

3.2. Sonic the Hedgehog - Sega Genesis

O ambiente escolhido foi o do Sonic the Hedgehog, que também é um jogo de plataforma 2D, onde o objetivo é chegar ao final de cada fase no menor tempo, coletando a maior quantidade de anéis. Além disso, o jogo possui obstáculos que só podem ser ultrapassados se o personagem estiver em uma velocidade mínima, dificultando severamente o avanço. Utilizou-se a biblioteca Gym Retro [Nichol et al. 2018], a qual possui uma variável da pontuação do jogo, que não deve ser utilizada diretamente como recompensa. Por isso, foi necessário implementar uma função que englobasse todos os objetivos principais:

$$\left.\begin{aligned} V &= \max(x_1 - x_0, -1) \\ A &= a_1 - a_0 \\ P &= p_1 - p_0 \end{aligned}\right\} \quad \text{Recompensa} = V.Wv + A.Wa + P.Wp + S + D$$

- W: Peso das variáveis, para este exemplo: $Wv = 0,2$, $Wa = 0,1$ e $Wp = 0,05$.
- V: Deslocamento horizontal, mas com um limite inferior de $V = -1$.
- A: Variação do número de anéis em relação ao loop anterior.
- P: Pontuação atual do agente p_1 subtraída da pontuação p_0 anterior.
- S: Penalidade por ficar parado, se $x_1 > \max(x)$ então $S = 0$, caso contrário se o agente passar mais de 100 loops com $x_1 < \max(x)$ então $S = -0,5$.
- D: Penalidade por morte: se o agente está vivo $D = 0$; caso contrário $D = -5$

Essa função de recompensa busca direcionar o agente a sempre aumentar seu $\max(x)$ enquanto coleta o máximo de anéis e derrota o máximo de inimigos, mas sem punir o agente caso ele tenha que andar para a esquerda devido aos loopings presentes nas fases, os quais exigem uma velocidade mínima para serem ultrapassados.

3.3. R-Type - Arcade

O jogo R-Type foi selecionado por ser relativamente diferente dos anteriores: é um jogo de tiro (shooter) onde os obstáculos se movimentam em direção ao personagem. O objetivo é sobreviver ao crescente número de inimigos na tela, atirando nos inimigos e desviando dos projéteis. Também utilizou-se a biblioteca Gym Retro, a função de recompensa teve de ser implementada manualmente, mas, diferente de Sonic, o agente não precisa avançar em direção aos obstáculos; ele apenas precisa sobreviver e desviar.

$$\left.\begin{aligned} P &= p_1 - p_0 \end{aligned}\right\} \quad \text{Recompensa} = A + P + D$$

- A: Um valor constante por sobrevivência, $A = 0,2$ a cada loop vivo.
- P: Variação da pontuação, aumenta ao matar inimigos.
- D: Penalidade por morte: se o agente está vivo $D = 0$; caso contrário $D = -5$.

A recompensa foi difícil de balancear, já que um P muito alto faz o agente matar o máximo de inimigos antes de morrer, enquanto um A alto faz com que o agente fique vivo, mas sem matar muitos inimigos; consequentemente, isso dificulta a sobrevivência.

3.4. Resultados

Na Tabela 3 seguem os resultados de treinamento dos experimentos, evidenciando que os agentes obtiveram progresso no jogo. Porém estes resultados são iniciais e carecem de uma melhor análise. As implementações estão no repositório [Tiago 2025a, Tiago 2025b].

Tabela 3. Resultados de desempenho dos agentes após o treinamento.

Métrica Avaliada	Mario	Sonic	R-Type
Quantidade de Épocas	10000	2000	500
Recompensa Média Acumulada	2427	4502	257
Perda Média	1,4228	4,5629	2,0157
Fase Máxima Alcançada [1,3]	2	3	1
Tempo Médio de Sobrevivência	53s	118s	21s

4. Conclusão

A função de recompensa é crucial para que o agente consiga aprender a finalizar o jogo, pois define o comportamento esperado. Embora os agentes tenham progredido, para um bom treinamento, seria necessária uma análise exaustiva dos hiperparâmetros e da função de recompensa de cada ambiente, pois este trabalho testou apenas uma amostra limitada. Para estudos futuros, recomenda-se a análise da função de recompensa em ambientes mais complexos, como jogos tridimensionais [Chaplot et al. 2016] ou espaços de ação contínuos.

Referências

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

Chaplot, D. S. et al. (2016). Transfer learning for deep reinforcement learning in 3d environments. In *NIPS 2016 Workshop on Deep Reinforcement Learning*.

Hu, C., Zhao, Y., Wang, Z., Du, H., and Liu, J. (2024). Games for artificial intelligence research: A review and perspectives. *IEEE Transactions on Artificial Intelligence*.

Kauten, C. (2018). Super mario bros. for openai gym. PyPI.

Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nichol, A., Hesse, C., Choi, J., and Schulman, J. (2018). Gotta learn fast: A new benchmark for generalization in rl. arXiv preprint arXiv:1804.03720.

PyTorch Foundation (2023). Pytorch: An open source machine learning framework. PyTorch Blog.

Shao, K., Tang, Z., Zhu, Y., Li, N., and Zhao, D. (2019). A survey of deep reinforcement learning in video games. arXiv preprint arXiv:1912.10944.

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

Tiago, L. M. P. (2025a). Implementação de agente dqn no gym super mario bros. GitHub. Disponível em: `https://github.com/Luc4smp/dqn-agente-mario`.

Tiago, L. M. P. (2025b). Implementação de um agente dqn para o gym retro. GitHub. Disponível em: `https://github.com/Luc4smp/dqn-agent-retro`.

Wang, Z. et al. (2016). Dueling network architectures for deep reinforcement learning. In *Proc. 33rd International Conference on Machine Learning*, pages 1995–2004. PMLR.