Uma Abordagem de Aprendizado por Reforço para Osu!

Lucas Gabriel Lopes Cruz
UNIFESP (Universidade Federal de São Paulo)
São José dos Campos, Brasil
lucas.cruz@unifesp.br

Gabriel Henrique Silva
UNIFESP (Universidade Federal de São Paulo)
São José dos Campos, Brasil
gabriel.henrique07@unifesp.br

***Abstract*—Este projeto visa aplicar técnicas de aprendizado por reforço (RL) e aprendizado por imitação para criar uma inteligência artificial (IA) capaz de jogar o jogo de ritmo Osu! em alto nível. O método empregado separa o Agente em duas tarefas, mover-se pela tela na direção da próxima nota e clicar na nota em um tempo adequado. Ambas as tarefas puderam ser otimizadas ao limite, levando a um alto nível de performance. Porém, o Agente atua sobre um ambiente simulado do jogo real e por isso ainda deve haver a possibilidade de adaptá-lo ao jogo real através da análise da imagem do jogo.**

***Index Terms*—aprendizado por reforço, inteligência artificial, jogos de ritmo, osu!**

I. Introdução

Osu! é um jogo de ritmo no qual os círculos de acerto aparecem como notas durante o tempo de execução de uma música, e o objetivo é clicar nos círculos no momento apropriado e na ordem correta, com a ajuda de aneis chamados círculos de aproximação que se aproximam dos círculos de acerto para indicar visualmente o tempo. A jogabilidade principal é inspirada no jogo de ritmo para Nintendo DS Osu! Tatakae! Ouendan e sua sequência Elite Beat Agents. Outros tipos de notas exigem que o jogador clique e mantenha pressionado enquanto move o cursor. Se o jogador errar muitos círculos, ele não conseguirá tocar a música e deverá tentar novamente. O objetivo deste artigo é descrever a construção de um ambiente utilizando a biblioteca Pygame e a API do Gymnasium para simular o jogo real, bem como a aplicação de algoritmos de aprendizado por reforço para treinar o agente nesse ambiente. O código completo do projeto está disponível no repositório do GitHub: https://github.com/Alkhiat-L/OsuAI.

A. Motivação

A motivação para esse projeto se deu pela busca de desafios interessantes e diferentes para esse tipo de IA. O jogo Osu! foi escolhido por ser um jogo conhecido por ambos os autores, além de ser um jogo popular conhecido mundialmente dentre os jogos de ritmo, possuindo mais de um 20 milhões de jogadores ao redor do mundo. Portanto, nada melhor do que um ambiente do qual já estávamos familiarizados e do qual existe uma alta gama de conteúdos disponíveis livremente na internet.

II. Trabalhos Relacionados

A. Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller arXiv:1312.5602

Apresentação do primeiro modelo de aprendizagem profunda para aprender com sucesso políticas de controle diretamente a partir de informações sensoriais de alta dimensão usando aprendizagem por reforço. O modelo é uma rede neural convolucional, treinada com uma variante de Q-learning, cuja entrada são pixels brutos e cuja saída é uma função de valor que estima recompensas futuras.

B. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm (2017)

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis arXiv:1712.01815

Generaliza a abordagem em um único algoritmo AlphaZero que pode alcançar, tabula rasa, desempenho sobre-humano em muitos domínios desafiadores. Partindo de um jogo aleatório e sem nenhum conhecimento de domínio, exceto as regras do jogo, o AlphaZero alcançou em 24 horas um nível de jogo sobre-humano nos jogos de xadrez e shogi (xadrez japonês), bem como no Go, e derrotou de forma convincente um programa campeão mundial em cada caso.

C. OpenAI Gym

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba arXiv:1606.01540

OpenAI Gym é um kit de ferramentas para pesquisa de aprendizagem por reforço. Inclui uma coleção crescente de problemas de benchmark que expõem uma interface comum e um site onde as pessoas podem compartilhar seus resultados e comparar o desempenho dos algoritmos. Este whitepaper discute os componentes do OpenAI Gym e as decisões de design tomadas no software.

D. Stable-Baselines3: Reliable Reinforcement Learning Implementations

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, Noah Dormann https://www.jmlr.org/papers/volume22/20-1364/20-1364.pdf

Stable Baselines 3 fornece implementações de código aberto de algoritmos de aprendizagem por reforço profundo (RL) em Python. As implementações foram comparadas com bases de código de referência e testes unitários automatizados cobrem 95% do código. Os algoritmos seguem uma interface consistente e são acompanhados por extensa documentação, simplificando o treinamento e a comparação de diferentes algoritmos RL. Documentação, exemplos e código-fonte estão disponíveis em https://github.com/DLR-RM/stable-baselines3.

III. Contexto

Para criar uma IA para o "Osu!" é essencial entender como o jogo funciona e como seria possível replicá-las num ambiente simulado. Portanto, a seguir, apresentamos o jogo "osu!" e alguns conceitos básicos para compreender o artigo.

A. Jogabilidade

O "Osu!" é bastante simples em sua essência: a jogabilidade consiste em realizar movimentos e cliques em círculos que aparecem na janela do jogo. As formas mais comuns de os jogadores realizarem essas ações são por meio do mouse e do teclado ou de uma mesa digitalizadora e um teclado. Como as notas são e onde elas aparecem são definidos em um "beatmap" criado pela comunidade de jogadores.

B. Elementos básicos

Observando a figura 1 podemos classificar cada um dos elementos do jogo:

- Barra de vida (Parte superior da tela na forma de uma barra), afetada por uma taxa de esgotamento passiva e uma penalidade por errar o tempo de uma nota. O jogador falha no beatmap se a barra de vida chegar a zero.
- Notas (Meio da tela), os objetos que o jogador deve clicar no momento certo. O anel ao redor da nota indica o momento a ser clicada.
- Pontuação (Abaixo da barra de vida), um cálculo de todos as notas que o jogador acertou.
- Precisão (Esquerda da Pontuação), a porcentagem de notas atingidas pelo jogador em um momento perfeito.
- Combo (Direita da Pontuação), o número de objetos atingidos consecutivamente.

C. Beatmaps

Para criar uma simulação eficiente de beatmaps, é crucial compreender a estrutura dos arquivos .osu. Ao desenvolver uma simulação, devemos nos concentrar principalmente na seção HitObjects, que fornece dados essenciais sobre os elementos interativos do jogo. O arquivo .osu é dividido em várias seções, cada uma com um propósito específico. Embora nem todas sejam diretamente relevantes para uma simulação de IA, é importante ter uma visão geral de sua estrutura. As seções

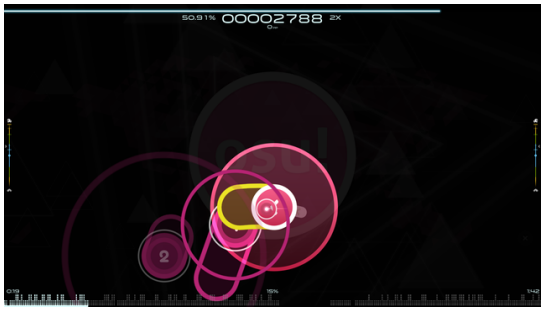

Fig. 1. Tela do jogo osu! durante uma música

incluem informações gerais, dados do editor, metadados, configurações de dificuldade, eventos, pontos de temporização e cores. A seção HitObjects é o coração do arquivo .osu para fins de simulação. Ela contém detalhes sobre cada objeto que aparece na tela, incluindo sua posição, tempo de ativação e tipo. Ao criar uma simulação. Informações técnicas sobre a estrutura do HitObjects:

Formato geral:

$$x, y, tempo, tipo, hitSound, objectParams, hitSample \tag{1}$$

- x,y: Coordenadas do objeto na tela - tempo: Momento de ativação em milissegundos - tipo: Identificador do tipo de objeto - objectParams: Parâmetros específicos do objeto

Para sliders:

$$curveType|curvePoints, slides, ...length, edgeSounds, edgeSets \tag{2}$$

- curveType: Forma do slider - curvePoints: Coordenadas dos pontos de curva (formato x:y) - slides: Número de repetições - length: Comprimento em pixels

Para spinners:

$$endTime \tag{3}$$

D. Aprendizado por Reforço (RL)

O Aprendizado por Reforço (RL) é uma área do aprendizado de máquina cujo objetivo é treinar agentes de IA a tomar decisões em ambientes dinâmicos visando maximizar recompensas cumulativas ao longo do tempo. Nesse processo, o agente interage continuamente com o ambiente, realizando ações, observando os estados resultantes e recebendo recompensas ou penalidades. Com base nessas interações, o agente desenvolve uma política, que é uma estratégia que mapeia estados para ações, utilizando a experiência acumulada. Por meio de métodos como o Q-learning ou Redes Neurais Profundas, o agente ajusta sua política para melhorar continuamente seu desempenho e alcançar melhores resultados em termos de recompensas recebidas.

E. OpenAI Gym

Anunciada em 2016, a Gym é uma biblioteca Python de código aberto projetada para facilitar o desenvolvimento de algoritmos de aprendizado por reforço. Seu objetivo era padronizar como os ambientes são definidos na pesquisa de IA,

tornando a pesquisa publicada mais facilmente reproduzível e fornecendo aos usuários uma interface simples para interagir com esses ambientes. Em 2022, novos desenvolvimentos do Gym foram transferidos para a biblioteca Gymnasium.

IV. Objetivo

O objetivo deste projeto é desenvolver uma inteligência artificial (IA) de alto desempenho capaz de jogar o jogo de ritmo Osu!, utilizando técnicas de aprendizado por reforço (RL). Para alcançar isso, o projeto visa construir um ambiente simulado do jogo usando Pygame e a API do Gymnasium, aplicar algoritmos de RL para treinar um agente nesse ambiente, separando as tarefas do agente em movimentação na tela e cliques precisos nas notas. O projeto busca otimizar essas tarefas para atingir uma performance elevada, com a perspectiva de eventualmente adaptar o agente treinado para jogar o jogo real através da análise de imagem, demonstrando assim a aplicação bem-sucedida de técnicas de IA em um jogo de ritmo popular e desafiador.

V. Metodologia experimental

A. Pré-processamento

Para que fosse possível simular o jogo real foi necessário utilizar uma função para importar o arquivo .osu para dentro do nosso ambiente. Isso nos levou a estudar a documentação desse tipo de arquivo e a guardar as informações dentro de uma lista de Notas recuperadas a partir do beatmap, como explicado anteriormente na seção III-C.

B. Ambiente Pygame

Nesta etapa, criamos um ambiente de Osu!, usando a biblioteca gráfica Pygame, implementamos as mecânicas básicas do jogo, como surgimento de notas, tipos de notas, etc. Num ambiente como o Pygame, não estamos limitados ao tempo do jogo real e podemos realizar testes várias vezes mais rápidos do que num mapa do jogo. Ademais, como o ambiente foi pensado para ser utilizado dentro do google colab para acelerar o treinamento.

C. Utilização do Gym

Na terceira etapa, focamos na construção da estrutura necessária para o Aprendizado por Reforço. Para isso, tomamos como inspiração o Framework Gym desenvolvido para a criação de ambiente de Aprendizado por Reforço que possui um padrão bem estruturado. Sendo assim, foram desenvolvidas entradas como as seguintes: reset(): Iniciar um novo jogo step(action): Processar um ”frame” da gameplay render(): Visualizar o estado do jogo

D. Algoritmos de IA

Utilizamos de dois tipos de modelos de IA de modo a separar as tarefas do agente, um para movimentação e um para clicar, capazes de receber dados do Osu!, utilizando dos algoritmos já existentes pela biblioteca stable baselines, sendo esses o: PPO (Proximal Policy Optimization), A2C (Advantage Actor Critic), TD3 (Twin Delayed DDPG), DQN(Deep Q-Network) e DDPG(Deep Deterministic Policy Gradient).

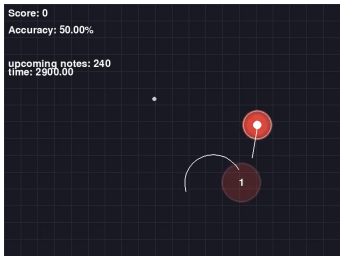

Fig. 2. Captura da tela da nossa versão do jogo

E. Função de recompensa e Wrappers

Projetamos uma função de recompensa para cada uma das tarefas, considerando, para o modelo de movimentação, a distância do mouse até a nota e, para o modelo de clicar, a diferença de tempo entre o clique e o tempo perfeito. Para realizar isso utilizamos de Wrappers para o ambiente Gym, os quais sobreescrevem funções internas do ambiente de acordo com a necessidade. Assim, foram criados 3 Wrappers para cada modelo, Observation, Action e Reward.

F. Implementação final do Ambiente

Inicialmente, utilizamos o jogo real para a implementação do modelo. Contudo, esse método se mostrou inviável devido às limitações da escala de tempo do jogo. Por esse motivo que decidimos criar uma réplica do jogo “Osu!” com as mecânicas necessárias ao invés de utilizar o jogo real, utilizando a biblioteca Pygame em Python. Essa escolha foi feita porque esse tipo de ambiente permite a manipulação do tempo de jogo, através da função step(), e a vetorização do ambiente, possibilitando um treinamento mais rápido e eficiente.

Desse modo, o jogo conta com todas as mecânicas básicas do original, permitindo a importação dos mesmos “beatmaps” (arquivos de mapa do jogo criados pelos usuários em formato *.osu). Além disso, esse método possibilita a realização de testes no próprio jogo.

O ambiente, presente no arquivo osupy/OsuPy.py do repositório do projeto, foi projetado para incluir a função step(action), que apresenta a estrutura típica para um modelo de Aprendizado por Reforço (RL).

Para a utilização do ambiente, utilizamos a biblioteca gymnasium e desenvolvemos a classe OsuPyEnv, que é uma implementação personalizada do ambiente de jogo. Dessa forma, ele pode ser utilizado dentro do arquivo osupy/model.py chamando a API do Gym e utilizando a biblioteca stable_baselines3, a qual disponibiliza algoritmos de RL e redes que possuem compatibilidade com a API do Gym, além de alta possibilidade de customização.

G. Espaço de Observação

O espaço de observação utilizado é formado pelas seguintes variáveis:

1) Modelo de movimentação:

- mouse_x: um float de 0 a 1 que contém a posição relativa do eixo x do mouse

- mouse_y: um float de 0 a 1 que contém a posição relativa do eixo y do mouse
- next_note_x: um float de 0 a 1 que contém a posição relativa do eixo x da próxima nota
- next_note_y: um float de 0 a 1 que contém a posição relativa do eixo y da próxima nota

2) Modelo de clicar:

- time_to_next_note: um espaço discreto que pode ser 1 quando a próxima nota está a menos de 50 ms do tempo perfeito de clicar e 0 quanto está a mais de 50 ms.
- curve_remaining: um espaço discreto que pode ser 1 quando a nota atual é uma curva e 0 quando não é.
- click: um espaço discreto que pode ser 1 quando o mouse está pressionado atualmente e 0 quando não está.

H. Espaço de Ação

O espaço de ação utilizado é formado pelas seguintes variáveis:

1) Modelo de movimentação:

- delta_x: um float de -3 a 3 que contém o deslocamento relativo ao eixo x a ser realizado.
- delta_y: um float de -3 a 3 que contém o deslocamento relativo ao eixo y a ser realizado.

2) Modelo de clicar:

- click: espaço discreto que pode ser 1 se o botão do mouse vai ser pressionado ou 0 se não.

I. Função de Recompensa

1) Modelo de movimentação: A função de recompensa do modelo de movimentação é uma fórmula baseada na distância do mouse até a nota, com perda adicional da recompensa se estiver nas bordas da janela ou se não fizer nenhum movimento.

2) Modelo de clicar: A função de recompensa do modelo de clicar é baseada nos acertos dos círculos, no progresso dos sliders e nos acertos consecutivos, para recompensar a constância, com perda de recompensa nos erros, aumentando a perda nos erros consecutivos.

VI. Avaliação dos resultados

Para a avaliação dos resultados iremos considerar tanto o resultado final da recompensa por episódio da melhor versão de cada um dos algoritmos.

A. Comparação de Modelos para o Componente Mover

Para o componente ”Mover” do nosso agente de Osu!, realizamos testes com diversos modelos de aprendizado por reforço, especificamente escolhidos por sua relevância para o tipo de ação contínua necessária para o movimento do cursor. Os modelos testados foram:

- PPO (Proximal Policy Optimization)
- DDPG (Deep Deterministic Policy Gradient)
- A2C (Advantage Actor-Critic)
- TD3 (Twin Delayed DDPG)

Após extensivos testes e comparações, identificamos que o PPO apresentou o melhor desempenho geral, seguido pelo DDPG como segunda melhor opção.

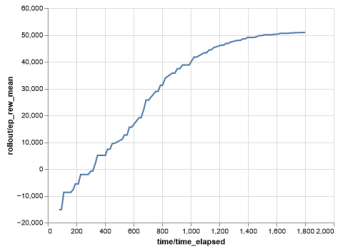

Fig. 3. Gráfico de Recompensa média por episódio vs Tempo do Modelo PPO para o Mover

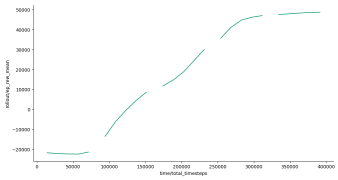

Fig. 4. Gráfico de Recompensa média por episódio vs Tempo do Modelo DDPG para o Mover

1) Analise do Modelo PPO para o Mover: Proximal Policy Optimization (PPO) é um algoritmo de aprendizado por reforço utilizado em problemas de controle e jogos. O PPO é conhecido por ser uma abordagem eficiente e estável, combinando as vantagens de métodos baseados em gradiente de política (como A3C) com a simplicidade de técnicas de otimização. Ele busca melhorar uma política diretamente, maximizando uma função objetivo, enquanto mantém as mudanças na política dentro de limites seguros. Isso é feito utilizando uma técnica chamada clipping, que evita grandes atualizações nos parâmetros da política, o que poderia resultar em oscilações e ineficiência no treinamento.

No gráfico 3, vemos que a recompensa média por episódio está aumentando de forma constante ao longo do tempo, o que é um bom sinal de que o agente está aprendendo e melhorando seu desempenho na tarefa. Desse modo, quando testado esse modelo apresentou um bom resultado alcançando uma recompensa por episódio de $51003.34 + / - 0.19$.

2) Analise do Modelo DDPG para o Mover: O Deep Deterministic Policy Gradient (DDPG) é um algoritmo de aprendizado por reforço profundo, projetado para lidar com problemas de controle contínuo, onde as ações que o agente pode tomar não são discretas, mas sim contínuas. Ele combina ideias dos métodos de gradiente de política (como o DPG) com o aprendizado Q (como o DQN), sendo particularmente eficaz em ambientes com espaços de ação contínuos.

No gráfico 5, vemos que a recompensa média por episódio consegue um bom impulso que chega a quase se igualar aos resultados do PPO chegando em uma recompensa por episódio de $50245.72 + / - 21.97$. Porém, seja pelo próprio formato do ddpg, ele aparenta ser mais instável que o PPO quando analisado pela gravação feita de um episódio de cada um. Enquanto o PPO se move de forma mais suave o DDPG parece se mover o mais rápido possível, algumas vezes passando do ponto.

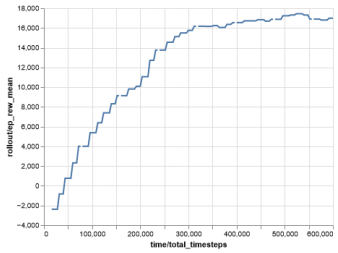

Fig. 5. Gráfico de Recompensa média por episódio vs Tempo do Modelo A2C para o Mover

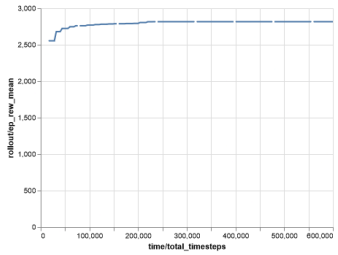

Fig. 6. Gráfico de Recompensa média por episódio vs Tempo do Modelo TD3 para o Mover

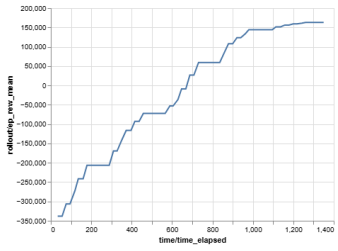

Fig. 7. Gráfico de Recompensa média por episódio vs Tempo do Modelo PPO para o Mover

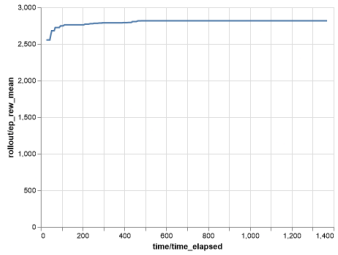

Fig. 8. Gráfico de Recompensa média por episódio vs Tempo do Modelo A2C para o Clicker

3) Analise do Modelo A2C para o Mover: O Advantage Actor-Critic (A2C) é um algoritmo de aprendizado por reforço profundo que combina as vantagens dos métodos baseados em valor e em política. Ele é uma variante do método Actor-Critic, onde o ator e o crítico são treinados simultaneamente, mas com algumas modificações que tornam o algoritmo mais eficiente e estável.

O A2C, ao contrário dos bons resultados do PPO e do DDPG apresentou um resultado que não consegue acompanhar as notas do jogo, chegando a uma recompensa de apenas 22833.55 +/- 38.75 por episódio.

4) Analise do Modelo TD3 para o Mover: O Twin Delayed Deep Deterministic Policy Gradient (TD3) é um aprimoramento do algoritmo DDPG (Deep Deterministic Policy Gradient), projetado para melhorar a estabilidade e a eficiência em problemas de controle contínuo. O TD3 aborda algumas das limitações do DDPG, como a alta variância e o desvio do valor estimado da função Q, introduzindo técnicas específicas para mitigar esses problemas.

O TD3 obteve um péssimo resultado em comparação com todos os outros, de 2816.00 +/- 0.00. É possível que ele fosse melhor dependendo de algum hiper parâmetro, porém com o PPO resultando num resultado tão alto, não será necessário.

B. Comparação de Modelos para o Componente Clicker

- PPO (Proximal Policy Optimization)
- A2C (Advantage Actor-Critic)
- DQN (Deep Q-Network)

1) Analise do Modelo PPO para o Clicker: No gráfico 7, é possível perceber que houve um bom treinamento do modelo utilizando do modelo PPO. Ademais, vendo posteriormente o video gravado de um episódio mostrou que a acurácia do modelo chegou a 99,5% e uma recompensa média por episódio de 166235,56.

2) Analise do Modelo A2C para o Clicker: No gráfico 8, o modelo usando A2C não foi capaz de alcançar o PPO.

3) Analise do Modelo DQN para o Clicker: O Deep Q-Network (DQN) é um algoritmo de aprendizado por reforço que combina técnicas de aprendizado profundo com o método tradicional de Q-learning, permitindo que agentes aprendam a tomar decisões em ambientes complexos com grandes espaços de estados e ações. Ele foi introduzido por uma equipe da DeepMind em 2013 e tornou-se famoso por superar o desempenho humano em vários jogos do Atari.

O DQN também não chegou próximo ao valor do PPO.

VII. Conclusão

Este trabalho apresentou uma implementação bem-sucedida de um agente de aprendizado por reforço para jogar o jogo rítmico ”Osu!”. Através da criação de um ambiente de jogo personalizado utilizando Pygame e a biblioteca Gymnasium, foi possível treinar modelos de IA para realizar as duas

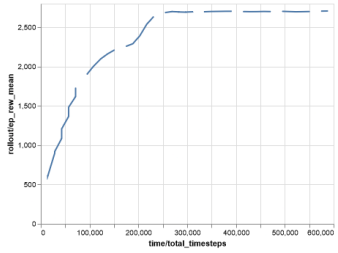

Fig. 9. Gráfico de Recompensa média por episódio vs Tempo do Modelo DQN para o Clicker

principais tarefas do jogo: movimentação do cursor e cliques precisos. Para o componente de movimentação, foram testados quatro algoritmos diferentes: PPO, DDPG, A2C e TD3. O modelo PPO demonstrou o melhor desempenho geral, alcançando uma recompensa média por episódio de 51003,34 ± 0,19. O DDPG também apresentou resultados promissores, chegando a uma recompensa de 50245,72 ± 21,97, embora com movimentos menos suaves comparados ao PPO. No que diz respeito ao componente de cliques, três algoritmos foram avaliados: PPO, A2C e DQN. Novamente, o PPO se destacou, atingindo uma impressionante acurácia de 99,5% e uma recompensa média por episódio de 166235,56. Além disso, é com grande satisfação que relatamos o sucesso na combinação dos modelos de movimentação e cliques. Ao integrar os dois componentes em um único agente, conseguimos alcançar uma notável acurácia de 95% no jogo completo. Este resultado demonstra não apenas a eficácia de cada componente individual, mas também a sinergia obtida ao combiná-los em um sistema unificado. Estes resultados demonstram o potencial da aplicação de técnicas de aprendizado por reforço em jogos rítmicos, oferecendo insights valiosos sobre a eficácia de diferentes algoritmos em tarefas que requerem precisão e timing.

VIII. Trabalhos Futuros

Apesar dos resultados promissores obtidos, há diversas direções para expandir e aprimorar este trabalho:

- Testes em Mapas Variados: Avaliar o desempenho dos modelos em uma gama mais ampla de ”beatmaps”, incluindo diferentes níveis de dificuldade e estilos musicais.
- Comparação com Jogadores Humanos: Realizar um estudo comparativo entre o desempenho do agente de IA e jogadores humanos de diferentes níveis de habilidade.
- Análise de Transferência de Aprendizado: Investigar a capacidade de transferência de aprendizado entre diferentes mapas ou até mesmo entre jogos rítmicos similares.
- Adaptação para o Jogo Original: Explorar maneiras de aplicar os modelos treinados diretamente no jogo ”Osu!” original, superando as limitações de escala de tempo inicialmente encontradas.

Estas direções de pesquisa futura não apenas aprimorariam o desempenho e a aplicabilidade do agente de IA em ”Osu!”, mas também contribuiriam para o campo mais amplo de aprendizado por reforço em jogos e tarefas que requerem coordenação motora fina e timing preciso.

References

[1] M. Schmid, M. Moravčík, N. Burch, R. Kadlec, J. Davidson, K. Waugh, N. Bard, F. Timbers, M. Lanctot, G.Z. Holland, E. Davoodi, A. Christianson, and M. Bowling, ”Student of Games: A unified learning algorithm for both perfect and imperfect information games,” arXiv preprintarXiv:112.3178, 21.

[2] D. Silver, J. Schrittwiser, and K. Simonyan, ”Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,” arXiv preprint arXiv:1712.01815, 2017.

[3] A. Plaat, ”Deep Reinforcement Learning, a textbook,” arXiv preprint arXiv:2201.02135, 2023.

[4] S. J. Russell, P. Norvig, R. S. Wazlawick, and V. D. de Souza, Inteligência Artificial [Artificial Intelligence] (Portuguese Edition). Rio de Janeiro, Brazil: Person, 2004

[5] V. Mnih et al., ”Playing Atari with Deep Reinforcement Learning,” arXiv:1312.5602 [cs.LG], Dec. 2013.

[6] H. van Hasselt, M. Hessel, and J. Aslanides, ”When to use parametric models in reinforcement learning?,” in Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), 2019.

[7] V. Mnih et al., ”Asynchronous Methods for Deep Reinforcement Learning,” in Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 1928-1937.

[8] Brockman, G., et al. (2016). OpenAI Gym.

[9] Stable Baselines3: A set of improved implementations of reinforcement learning algorithms in PyTorch.