Explorando Combos em Jogos de Luta Através de Reinforcement Learning

1st Rodrigo Peixe Oliveira
Ciência da Computação
UNIFESP
São José dos Campos, Brasil
rodrigo.peixe@unifesp.br

2nd Rennam Victor Cabral de Faria
Ciência da Computação
UNIFESP
São José dos Campos, Brasil
rennam.faria@unifesp.br

I. Resumo

Jogos de luta apresentam grande complexidade, devido ao número de personagens, opções e possíveis interações entre eles. Uma dessas interações presente em quase todos os jogos de luta é o combo. Devido aos fatores descritos, é quase impossível conhecer todos os combos possíveis de cada personagem em cada personagem, principalmente antes do jogo ser lançado, e alguns combos, mesmo de jogos antigos, continuam sendo descobertos até hoje. Como combos são uma parte essencial do balanceamento, diretamente influenciando a recompensa do jogador ao aproveitar aberturas do oponente, eles precisam ser rigorosamente testados, e empresas grandes de jogos de luta costumam ter inúmeros *testers* para avaliar as possibilidades. Um dos problemas que pode surgir é o combo infinito, que impede o oponente de jogar até o fim da partida, reduzindo a interatividade e a diversão. O artigo [1] explorou a possibilidade de utilizar algoritmos evolucionários para tentar encontrar combos infinitos em jogos de luta. Neste artigo, iremos explorar a ideia de usar algoritmos de aprendizado por reforço com o mesmo objetivo, utilizando tanto A2C quanto PPO, e comparando os resultados com o artigo citado.[1][2]

II. Introdução e Motivação

Os jogos de luta, como *Street Fighter Alpha 3*, são caracterizados por sua complexidade e profundidade estratégica, onde os jogadores precisam dominar uma ampla variedade de movimentos, combos e táticas para serem bem-sucedidos. Uma das principais dificuldades no desenvolvimento desses jogos é garantir que eles sejam equilibrados, proporcionando uma experiência justa e competitiva para todos os jogadores, independentemente do personagem escolhido. Isso se torna especialmente importante no cenário atual, em que torneios desses jogos são transmitidos para o mundo todo e apresentam prêmios milionários.

No entanto, testar e balancear esses jogos pode ser uma tarefa extremamente desafiadora e demorada, devido à enorme quantidade de combinações possíveis de movimentos e interações entre personagens. Como são jogos que normalmente exigem habilidade e podem ser jogador de diversas formas, os desenvolvedores nem sempre sabem exatamente como eles serão jogados no futuro. Os jogadores exigem uma variedade crescente de personagens e habilidades, o que cria um desafio cada vez maior para os desenvolvedores, que buscam criar jogos justos e divertidos, tanto para os jogadores quanto para os espectadores.

Uma parte importante do balanceamento de jogos de luta são os combos. Eles normalmente exigem algum tipo de abertura, mas, após iniciados, permitem que o atacante desfira uma sequência de golpes no oponente, geralmente com pouca ou nenhuma interação deste, dependendo apenas da capacidade do atacante de executá-lo. Assim, combos influenciam como um personagem pode punir outros por erros, contribuindo diretamente para o quão forte ou fraco cada personagem é. Combos são resultados das interações das mecânicas do jogo e nem sempre são previstos pelo desenvolvedor. Um caso especial, normalmente indesejado, de combo é o combo infinito, que impede o oponente de jogar pelo resto da duração da partida.

A introdução de técnicas avançadas de inteligência artificial (IA), como o aprendizado por reforço (*Reinforcement Learning* - RL) e algoritmos genéticos, oferece uma oportunidade promissora para abordar esses desafios. O aprendizado por reforço, especificamente os métodos *Advantage Actor-Critic* (A2C), e *Proximal Policy Optimization* (PPO), têm se mostrado eficazes em aprender estratégias complexas em ambientes dinâmicos e de alta variabilidade, como é o caso dos jogos de luta. Além disso, os algoritmos genéticos, que simulam o processo de seleção natural, podem ser usados para explorar e otimizar combos de movimentos, encontrando soluções que humanos poderiam levar anos para descobrir.

Este trabalho propõe a criação de uma IA especializada em explorar e identificar combos em *Street Fighter Alpha 3* utilizando aprendizado por reforço com A2C, PPO e comparando seu desempenho com um algoritmo genético criado em [1]. A motivação para este estudo é dupla: por um lado, desejamos demonstrar a eficácia de técnicas avançadas de IA na exploração e otimização de combos em jogos de luta; por outro, buscamos fornecer uma ferramenta poderosa para desenvolvedores de jogos, que pode ser utilizada para testar e balancear seus jogos de forma mais eficiente e precisa.

A escolha do jogo *Street Fighter Alpha 3* foi feita por

[1]Código disponível em: https://github.com/rpeixe/gym-exploring-combos
[2]Vídeo mostrando os melhores combos disponível em: https://youtu.be/RttH05aGbQA

dois motivos. Primeiro, é um jogo para o qual possuímos a versão de *Game Boy Advance* (GBA). Essa versão pode ser emulada no ambiente do *Stable-Retro*, um *fork* mais atualizado da biblioteca *Gym Retro* da *OpenAI* que usaremos no trabalho. Segundo, esse jogo introduziu uma mecânica inédia à série *Street Fighter* chamada de *Fighting Styles*, ou *ISMs*. Essa mecânica adicionou variedade para o jogo, mas foi infame por introduzir a possibilidade de executar combos infinitos, o que ilustra o problema discutido anteriormente.

Através deste estudo, esperamos contribuir para o avanço do uso de IA no desenvolvimento de jogos eletrônicos, oferecendo uma solução inovadora para um problema antigo e complexo. Além disso, pretendemos fornecer *insights* valiosos sobre as capacidades e limitações das técnicas de aprendizado de máquina no contexto de jogos de luta, potencialmente influenciando futuras pesquisas e aplicações na área.

III. Conceitos Fundamentais

Para entender o desenvolvimento da IA para explorar combos em *Street Fighter Alpha 3* utilizando aprendizado por reforço, é essencial compreender alguns conceitos fundamentais. Nesta seção, abordaremos os principais conceitos que sustentam nosso trabalho.

A. Aprendizado por Reforço

O aprendizado por reforço (RL) é uma área do aprendizado de máquina onde um agente aprende a tomar decisões através da interação com um ambiente, visando maximizar uma recompensa acumulada ao longo do tempo. Os principais componentes do RL são [2]:

- **Agente**: A entidade que toma ações no ambiente.
- **Ambiente**: O mundo com o qual o agente interage.
- **Estado**: A representação da situação atual do agente no ambiente.
- **Ação**: As decisões ou movimentos que o agente pode realizar.
- **Recompensa**: O feedback recebido após a execução de uma ação, que pode ser positivo ou negativo.
- **Política**: A estratégia que o agente segue para decidir quais ações tomar com base no estado atual.
- **Episódio**: Uma série de interações entre o agente e o ambiente, que em algum momento chega ao fim.

B. Advantage Actor-Critic (A2C)

O método Advantage Actor-Critic (A2C) é uma abordagem de aprendizado por reforço que combina os métodos de *actor-critic* com uma vantagem (*advantage*) para melhorar a eficiência do aprendizado. Ele consiste em dois principais componentes:

- **Ator (*Actor*)**: A parte da rede neural que seleciona as ações com base na política atual.
- **Crítico (*Critic*)**: A parte da rede neural que avalia a qualidade das ações tomadas pelo ator, utilizando a função de valor.
- **Vantagem (*Advantage*)**: A diferença entre a função de valor de um estado-ação específico e o valor esperado daquele estado, ajudando a reduzir a variância e acelerar o aprendizado.

O A2C é considerado um algoritmo **on-policy**, o que significa que ele aprende e melhora diretamente a política que está sendo seguida pelo agente durante o processo de tomada de decisão. Essa característica contrasta com os métodos **off-policy**, que podem aprender uma política diferente daquela que está sendo seguida pelo agente [3].

C. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) é um algoritmo de aprendizado por reforço que foi desenvolvido para superar algumas das limitações de métodos como o A2C, proporcionando maior estabilidade e eficiência durante o treinamento. O PPO busca melhorar a política através de pequenas atualizações iterativas, garantindo que cada nova política não se desvie excessivamente da anterior, o que é feito por meio da imposição de uma **penalidade de clipping** em grandes mudanças na política.

O PPO também é um algoritmo **on-policy**, como o A2C, o que significa que ele otimiza a política que está sendo seguida diretamente. A principal diferença está na forma como o PPO controla as atualizações da política, buscando um equilíbrio entre exploração e estabilidade durante o processo de aprendizado [4] [5].

D. Algoritmos Genéticos

Os algoritmos genéticos (GA) são técnicas de otimização inspiradas nos processos de seleção natural e evolução biológica. Eles são usados para encontrar soluções aproximadas para problemas complexos através de operações genéticas como seleção, cruzamento e mutação. Os principais componentes dos AG são [6]:

- **População**: Um conjunto de possíveis soluções para o problema.
- **Cromossomo**: Uma representação codificada de uma solução dentro da população.
- **Função de Aptidão (*Fitness Function*)**: Uma função que avalia a qualidade de cada solução.
- **Seleção**: O processo de escolher os melhores indivíduos da população para reprodução.
- **Cruzamento (*Crossover*)**: A combinação de dois cromossomos para criar uma nova solução.
- **Mutação**: A alteração aleatória de um cromossomo para introduzir diversidade genética.

]

E. Mecânicas de Jogos

Mecânicas de jogos são os sistemas que o jogo apresenta com os quais o jogador pode interagir. As mecânicas do jogo também podem interagir entre si. Exemplos disso em jogos de luta são mecânicas de golpes (apertar um determinado botão realiza um determinado golpe), de vida (receber golpes reduz o seu total de vida) e de rodadas (ter seu total de vida reduzido a zero faz com que o jogador perca a rodada) [7].

F. Balanceamento de Jogos

O balanceamento de jogos é o processo de ajustar as mecânicas de jogo para garantir que nenhum personagem ou estratégia seja excessivamente dominante. Isso envolve testes extensivos para identificar e corrigir desequilíbrios, proporcionando uma experiência justa e competitiva para todos os jogadores. Ferramentas baseadas em IA podem automatizar partes deste processo, ajudando a identificar desequilíbrios de forma mais eficiente [8].

G. Jogo de Luta

Um gênero de jogo eletrônico em que um personagem controlado pelo jogador luta contra outro, controlado ou por IA ou por outro jogador. Geralmente apresenta diferentes golpes e movimentos especiais, que podem ser encadeados para criar *combos* [9].

H. Frame

Jogos eletrônicos funcionam com base em *frames*, que é a unidade de tempo que representa instantes discretos em que a entrada (*input*) do jogador é lida, a lógica do jogo é calculada, e a imagem da tela é desenhada. O número de *frames* por segundo do jogo é chamado de *framerate*. Um *timestep* do ambiente equivale a um *frame* do jogo [10]

I. Hit Stun

A duração de tempo (em *frames*) após um personagem ser acertado em que ele não pode se defender ou realizar nenhuma ação (com exceções dependendo do jogo) é chamada de *hit stun* (ou *hitstun*) [11]

J. Combo

Enquanto um personagem está em *hit stun*, ele pode continuar a receber outros golpes, o que prolonga o estado de *hit stun*. Uma sequência de golpes encadeada dessa forma é chamada de *combo*. Uma sequência normalmente é chamada de *combo* quando tem pelo menos dois acertos, e dura até o fim do *hit stun* do oponente [12]

IV. Trabalhos Relacionados

As inspirações para este trabalho foram [1] e [13], que buscaram o mesmo objetivo mas [1] utilizou um algoritmo evolutivo e [13] utilizou Modelos Ocultos de Markov (*Markov Hidden Models* - HMM). Observando que a maioria das aplicações de IAs em jogos digitais atualmente utilizam AR, decidimos testar aplicar essa técnica ao problema. Uma outra diferença importante dos artigos é que, neles, foi utilizado um ambiente de jogo feito especialmente para a pesquisa, uma vez que não era possível fazer a integração para obter as informações necessárias de jogos reais, como explicado em [13]. Com a criação de tecnologias mais modernas, como o *Gym Retro*, isso se tornou possível, o que nos motivou a testar em jogos reais.

Uma outra aplicação de IA em jogo de luta, mais especificamente Street Fighter, é [14]. O objetivo dessa foi vencer o modo de um jogador do jogo *Street Fighter II Special Champion Edition* do *Sega Genesis*. Foi utilizado o algoritmo A2C de aprendizado por reforço, que alcançou resultados favoráveis, conseguindo vencer a campanha, o que nos motiva a testar essa abordagem.

Um outro trabalho relacionado é [15], que testou diversas abordagens em três jogos diferentes. O objetivo deste também foi vencer o modo de um jogador. Em especial, foi testado o jogo de luta *Mortal Kombat 3* com três diferentes modelos: NEAT (*NeuroEvolution of Augmenting Topologies*), PPO (*Proximal Policy Optimization*) e *Curiosity*. Desses, PPO conseguiu os melhores resultados, se mostrando uma opção atraente para jogos de luta.

V. Objetivo

Neste trabalho, o objetivo principal é utilizar Reinforcement Learning para tentar encontrar e executar o combo mais longo possível, e comparar diferentes algoritmos quanto a eficiência e eficácia na tarefa. Utilizaremos tanto A2C quanto PPO como algoritmos de RL, e compararemos com o algoritmo genético (GA) de [1]. Com isso, esperamos conseguir encontrar bons resultados, e possivelmente encontrar combos infinitos que existem no jogo em questão, além de ter uma indicação dos algoritmos mais indicados para a tarefa.

VI. Metodologia Experimental

Como o nosso trabalho é baseado em um ambiente de jogo eletrônico, *Street Fighter Alpha 3*, e considerando os trabalhos anteriores realizados nessa área, como os trabalhos [16], [17] e [18] que possuiram ótimos resultados, optamos por desenvolver nossa IA utilizando o modelo de RL, sendo presentes em grande parte das IA usadas tanto para jogos eletrônicos e jogos de tabuleiro.

A popularidade do uso desse modelo em jogos é pela forma de se aprender, utilizando probabilidade e aleatoriedade, pelo conceito de *exploration vs. exploitation* e também pelo fato que não é necessário rotular as suas entradas/saídas. A ideia de *exploration vs. exploitation* é explorar caminhos inexplorados em um ambiente, e utilizar o seu conhecimento, gerado pelos caminhos já explorados, para gerar o máximo de recompensa, tendo assim um equilíbrio entre exploração e uso do conhecimento. Em que, no contexto de jogo, se encaixa perfeitamente na forma que é feito, na qual o jogo é o ambiente e a IA deve aprender as ações que geram melhores resultados no ambiente. Outro motivo para usá-lo é que, na maioria dos jogos possui um campo de possibilidades muito grande ou infinitas, sendo assim inviável tentar rotular todo estado para um outro tipo de modelo conseguir executar. Referência: [19] e [20]

Para o desenvolvimento do código utilizamos bibliotecas existentes para desenvolver os modelos de RL em jogos, sendo os principais *Stable-Baselines3* e *Stable-Retro* (um *fork* mais atualizado do *Gym Retro* da *OpenAI*). Utilizando-nas, é possível montar o ambiente e utilizar outras funcionalidades que elas nos oferecem, tais como funções de treinos e testes já implementadas pela *Stable-Baselines3*. Para a escolha de

modelagem da IA, utilizaremos os algoritmos A2C e PPO, que já possuem resultados positivos em trabalhos anteriores.

Para que a IA possa modificar o ambiente e gerar as sequências de combos, foi utilizado o *integration tool* do *Stable-Retro*. Esta ferramenta, é comumente utilizada para implementar IA em jogos de consoles antigos através de emuladores, como os consoles GBA, Atari e SNES. Em nosso caso utilizamos ROM para o console GBA. Além disso, ela possui funcionalidades para o funcionamento da IA, como coleta de dados da partida através dos valores da memória. Dessa forma, podemos mapear exatamente o lugar da memória que está armazenada certas informações, fornecendo elas para que a IA seja treinada, e dependendo das ações executadas poderá gerar recompensas ou não, observando as alterações de informações do jogo pela memória.

Por fim, na criação da estrutura do código, iniciamos a resolver problemas que afetam o treinamento da nossa IA, como impedir que o agente pressione o botão de provocação, que impede o personagem de agir por um tempo. Após isso, seguimos criando o ambiente para que a IA possa executar os combos, o treino e o teste, e tunanmos os hiperparâmetros, tal como taxa de aprendizado, número de passos, coeficiente de entropia e os restantes. No final, salvamos o modelo de melhor resultado.

A. Domínio da Aplicação

O jogo que iremos usar para os experimentos é a versão para Game Boy Advance do Street Fighter Alpha 3. É um jogo de luta 2D um contra um com diversos personagens e modos. Nos concentraremos no modo de treino, pois estamos interessados apenas nos combos, e utilizaremos o personagem Ryu por ser bem conhecido e balanceado. Nesse jogo, também é possível escolher entre três estilos de luta para cada personagem: X-ISM, A-ISM e V-ISM. Eles diferem principalmente em como é possível utilizar a barra de recursos na luta. Escolhemos o estilo V-ISM para os experimentos, pois ele possibilita fazer combos mais facilmente e é infame por possibilitar combos infinitos em condições específicas.

Dentro das lutas, é possível andar, pular, defender e desferir uma variedade de ataques e movimentos especiais, além de provocar (o que não tem uso prático). Dentre os ataques, existem:

- **Ataques normais**: soco forte, soco médio, soco fraco, chute forte, chute médio e chute fraco. Esses ataques variam se o personagem está de pé, agachado ou no ar, totalizando 9 ataques normais diferentes.
- **Command normals**: são uma combinação de uma direção com um ataque básico e são específicos de cada personagem.
- **Movimentos Especiais**: são executados pressionando uma sequência de direções seguida por um determinado botão de ataque. Eles são variados e podem ser um golpe diferente, arremessar uma bola de fogo ou simplesmente se mover pelo mapa, por exemplo.
- Dependendo do estilo de luta escolhido, é possível gastar recursos para fazer uma ação específica.
- **Arremesso**: há a possibilidade de agarrar e arremessar o oponente (o que não é útil pois não é possível continuar o combo).

Quando um personagem atinge outro com um ataque, algumas coisas acontecem:

- A vida do inimigo é reduzida por um determinado valor.
- O personagem que acertou o golpe fica incapaz de agir normalmente até a animação do golpe terminar, estado que chamamos de recovery. Porém, dependendo do golpe e situação, é possível cancelar o recovery em determinados outros golpes (normalmente movimentos especiais), permitindo que outro ataque seja desferido mais rapidamente que o normal. Em especial, o estilo de luta V-ISM permite entrar em um modo (chamado custom combo) por um tempo que habilita o personagem a cancelar o recovery de qualquer golpe em qualquer outro golpe.
- O inimigo fica incapaz de agir por um determinado tempo, estado que é chamado de hitstun. Enquanto ele está em hitstun, ele não pode se defender, podendo receber mais golpes, criando o que chamamos de combo.
- O inimigo pode ou não ser arremessado para o ar, dependendo do ataque e situação. Se o hitstun acabar enquanto ele está no ar, ele pode se recuperar (chamado de air tech). Se ele atingir o chão, ele entra em uma animação de se levantar (chamada de ground tech). Em ambos os casos, o inimigo fica intangível até poder agir novamente, o que significa que acabou o combo do oponente.

As batalhas ocorrem em lugares chamados de estágios, que são planos e possuem duas paredes (visíveis ou não) nas extremidades que impedem os lutadores de passar. Há uma variedade de estágios, mas a diferença é meramente estética. Os lutadores sempre começam no centro do estágio a uma determinada distância. Combos são mais fáceis e consideravelmente mais perigosos quando o inimigo está perto da parede, pois ela impede que ele seja arremessado para longe do agressor. Por causa disso, nos experimentos, decidimos começar os episódios com o inimigo de costas para a parede, a fim de facilitar os combos do agente de RL.

B. Pré-processamento

Para a criação do ambiente de aprendizado e integração entre o código e o emulador, utilizamos a biblioteca Stable-Retro. Ela cria um ambiente através do Gymnasium, carrega o jogo em um determinado ponto inicial (chamado e estado) e apresenta métodos para reiniciar o jogo a partir desse ponto, dar um passo do jogo (avançar um frame) utilizando uma determinada ação (os botões apertados) e retornando uma imagem da tela do jogo, além de sinalizar recompensas e términos a partir de condições definidas no cenário. Além disso, ele permite criar wrappers para modificar o ambiente, alterando a observação, ação ou recompensa. Nós transformamos os ambientes criados em ambientes vetorizados através do Stable Baselines3, que nos permite trabalhar com vários ambientes em paralelo para acelerar o aprendizado.

Nós utilizamos os seguintes wrappers:

- **GrayScaleObservation (Gymnasium)**: transforma a observação em escala de cinza, reduzindo o número de canais e acelerando o trabalho da CNN.
- **ResizeObservation (Gymnasium)**: diminui a resolução da imagem, acelerando o trabalho da CNN.
- **TimeLimit (Gymnasium)**: aplica um limite de tempo a cada episódio, lidando com situações em que o agente fica preso sem fazer nada.
- **VecFrameStack (Stable Baselines3)**: empilha observações passadas para ajudar o agente a entender o sentido de movimentos (por exemplo, quando um personagem está no ar, isso ajuda a saber se está subindo ou descendo).
- **VecTransposeImage (Stable Baselines3)**: transpõe a imagem para ficar do formato esperado pela CNN.
- **ActionMemory (implementado)**: altera o espaço de observação e inclui uma memória das ações passadas para ajudar o agente a aprender a fazer movimentos especiais, que precisam de uma sequência específica de botões, o que não é captado pela imagem da tela
- **FilterAction (implementado)**: para impedir que o agente aperte o SELECT, que é um botão que ativa uma animação de provocação e não tem utilidade em combos.

No final, o modelo de aprendizado vai receber a saída da imagem (transformada) do jogo após passar pela CNN, juntamente com o vetor achatado das últimas ações tomadas pelo agente. A saída será um vetor de ação, representando os botões apertados, que será utilizado pelo ambiente para dar o próximo passo.

C. Reconhecimento de Padrões

A tarefa que a IA deve realizar é encontrar e executar o combo mais longo, isto é, que dura o maior número de frames contínuos com o oponente em hitstun. A recompensa será determinada com base no cenário dado no treinamento, sendo que cada cenário pode alterar o estado inicial da luta e a forma de avaliação da recompensa. Experimentaremos alguns casos, como dar a recompensa pelo tempo total do combo, ou baseada na combinação do número de golpes somados com o tempo do combo. No entanto, todos os cenários compartilham o objetivo comum de encontrar e realizar o combo mais longo possível.

A principal biblioteca a ser utilizada no modelo de RL será Stable Baselines 3, que oferece implementações de algoritmos de RL em PyTorch, possuindo implementações já feitas para algoritmos, treinamento, customizações e entre outras funcionalidades. Adicionalmente, utilizamos Torch e TensorBoard para análise dos resultados em gráficos e salvar parâmetros de treinamento, Optuna para otimização de hiperparâmetros, ela está implementada porém o processo de encontrar os hiperparâmetros é muito demorado, e Argparse, para que o usuário passe os parâmetros que deseja treinar ou utilizar valores por padrão que adicionamos.

Com a biblioteca Stable Baselines 3, utilizamos algoritmos já implementado por ela, inicialmente o Advantage Actor Critic (A2C), que é um algoritmo que utiliza de ator, aquele que decide as ações a serem tomadas no ambiente, e o crítico, que avalia a eficácia dessas ações, gerando assim recompensa para a IA. Então, experimentamos com Proximal Policy Optimization (PPO), um algoritmo de RL que ajusta gradualmente a política do agente conservadoramente, dando pequenas mudanças na política e sem que afaste muito da antiga, permitindo uma exploração mais estável e controlada, porém mais lenta, entretanto este algoritmo já demonstrou bons resultados em outros projetos de RL, como [21] e [22].

Como citado anteriormente, utilizamos alguns métodos e técnicas para aprimorar o aprendizado da IA e a velocidade dela Restrição de ações: Implementado pelo FilterAction, ele limita a quantidade geral de ações que a IA pode fazer em um jogo, assim deixando o processo de treinamento mais rápido. Tempo máximo em época: Implementado pelo TimeLimit, usa o valor padrão de 600 timesteps máximo para cada época de treinamento, para garantir que a IA não fique presa em uma época infinitamente. Acelerar a velocidade do ambiente: Deixando o processo de treinamento mais rápido, mantendo a precisão da IA na computação das ações. Vectorized Environments (Stable Baselines3): É uma opção do Stable Baselines3 que possibilita a opção de executar vários ambientes independentes de treinamento em um único ambiente para que a IA possa treinar rapidamente.

Além deles, utilizamos uma CNN já implementada por padrão nos modelos do Stable Baselines3 (chamada de Nature CNN) para processar as imagens em vetores achatados. Adicionalmente, buscamos os melhores parâmetros e salvamos modelos durante o treinamento para preservar os que geram as melhores recompensas e evitar problemas como o overfitting.

Para os algoritmos de RL, foram utilizadas as seguintes condições de parada: acabar o limite de tempo de 600 frames (10 segundos) ou o contador de combo diminuir (o que significa que o combo acabou). Como recompensa, a cada frame que o inimigo está em um combo, é contabilizado 1 de recompensa. Assim, no fim do episódio, a recompensa total é igual ao total de frames que o combo durou. A cada 3000 passos, são rodados 10 episódios de validação para encontrar os pontos que o agente se sai melhor. Durante essa validação, é usada o modo estocástico, já que o intuito é explorar e tentar conseguir uma grande variedade de combos para encontrar um único combo maior possível, mesmo que não seja consistente. Após o término do treino, foram executados 100 episódios de teste (sem aprendizado) de forma estocástica, de modo a avaliar o modelo final, e o melhor resultado, assim como a média, foi salvo.

Para o algoritmo evolucionário, como quisemos usar exatamente o fitness do artigo [1], não há limite de tempo. Ao invés disso, cada indivíduo é uma sequência de inputs do controle com tamanho 600, durando 600 frames assim como o RL. Foi guardado o combo mais longo gerado por cada indivíduo para comparação com os algoritmos de RL. Para calcular a fitness, foi utilizada a seguinte fórmula. O algoritmo completo está descrito no artigo [1]. Zeros se refere ao número de frames entre um combo e outro.

$$F = \sum_{i=1}^{n} f(i), \quad f(i) = \begin{cases} \frac{\text{ComboSize}}{(1+\text{LeftZeroes})^2} & \text{if right of longest} \\ \frac{\text{ComboSize}}{(1+\text{RightZeroes})^2} & \text{if left of longest} \\ \text{ComboSize} & \text{if the longest} \end{cases} \tag{1}$$

Nosso principal meio de validação e comparação será via quantidade de recompensa (que equivale à duração do combo) e taxa de função de perda, e essas análises serão facilitadas pelos gráficos que serão gerados pela etapa de pós-processamento, com esses gráficos podemos analisar o tempo que certos treinamento demoram a atingir certa quantidade de recompensa e comparar a quantidade de recompensas média que dois tipos de treinamento possuíram determinando o mesma quantidade de tempo (TimeSteps).

Os resultados gerados pelo treinamento e as características dos resultados foram comparadas entre si e com o modelo de algoritmo genético, adaptado do artigo [1]. Dessa forma, poderemos avaliar a eficácia da IA em relação a outras, demonstrando quão bem ela se compara e onde se destaca.

D. Pós-processamento

O estudo dos resultados será baseado nos dados retornados pelos treinamentos realizados, com um foco detalhado em avaliar o desempenho do modelo na execução dos combos. Esses dados retornados podem ser vistos pelo terminal em que está sendo executada a IA ou em gráficos via TensorBoard, que armazenará os resultados do treino, assim sendo um ótimo meio de análise dos dados e comparação entre eles. Nele terá informações de função de perda (LossFunction), média de recompensa por períodos (MeanReward) e dentre outros. As comparações serão sistematizadas para garantir precisão, utilizando a mesma quantidade de timesteps para uma análise consistente.

A comparação de dados será feita com outros modelos e algoritmos para avaliar a eficácia da nossa abordagem. Inicialmente, iremos comparar nossos dados de resultados com um trabalho anterior que usou Algoritmo Genético (GA), [1], então poderemos avaliar a diferença de resultados. Também, iremos comparar resultados obtidos do nosso próprio código com diferentes algoritmos, tal como utilizar os algoritmos A2C e PPO, e analisaremos as mudanças que impactam o desempenho. Por fim, comparamos resultados obtidos por diferentes cenários, verificando quais deles possuem maior quantidade de recompensa. Investigando as razões por trás das ações tomadas e suas características de treinamento.

VII. Resultados e Discussões

Na análise dos treinamentos, apresentaremos gráficos gerados pela biblioteca TensorBoard, apresentaremos os melhores resultados obtidos por cada algoritmo e como o tempo de execução para cada um deles. O principal objetivo é apresentar os resultados e analisá-los, buscando informações, características e particularidades que cada algoritmo possa agregar para o estudo. Nas análise de gráficos, será focado principalmente na função de perda (Loss Function), e na recompensa obtida (Mean Reward), que equivale à quantidade de frames total do combo. No GA, será analisada a fitness (o que mostra como o algoritmo está evoluindo) e o combo máximo obtido em cada geração, que pode ser mais diretamente comparado com os algoritmos de RL.

Para os algoritmos baseados em RL, haverá um maior número de comparação entre eles. Isso se deve ao fato de que, por usarem o mesmo modelo de IA, é mais fácil de fazer comparações detalhadas, do que com modelos distintos, no caso da GA. Sendo assim a geração de informações entre esses dois será mais precisa e detalhada, explicando o porquê cada algoritmo teve tal padrão em seus treinos. Além de que, os parâmetros usados para ambos foi muito similares, no que facilita para comparação.

No entanto, para o trabalho baseado em GA, que também será feita a comparação de resultado, foi necessário realizar uma adaptação para possibilitar uma comparação adequada com os algoritmos implementados em nossa IA. Isso se deve ao fato de que, além de utilizar uma modelagem distinta de RL, o código do GA foi feito para operar em um ambiente diferente do nosso, usando uma réplica do jogo Street Fighter 2 desenvolvida na engine Game Maker pelos autores. Para permitir uma comparação válida, foi adaptado o código GA para rodar no jogo que estamos analisando, no ambiente do Stable Retro, para que seja possível comparar com os nossos resultados em RL. Vale ressaltar que os resultados obtidos serão distintos dos gerados no trabalho original, pois a GA adaptada opera em um ambiente diferente. Contudo, os cálculos de fitness são os mesmos usados no trabalho anterior.

A. A2C

O algoritmo de A2C foi o primeiro a ser testado como treinamento, ele é famoso por ser um dos mais básicos para se usar em RL. O A2C é um variante do algoritmo Asynchronous Advantage Actor Critic (A3C), em que utiliza diversos trabalhadores para processar e eliminar a necessidade de um replay de buffer. Porém, ambos usam ideias do Actor-Critic, na qual possui uma política onde um agente (ou "ator") executa as ações, enquanto uma função de valor (ou "crítico") avalia essas ações, recompensando a IA de acordo com o impacto no ambiente e o critério de recompensa escolhido.

Os treinos foi o mais rápido entre os algoritmos de RL até completar a determinada quantidade de TimeSteps do treino, como esperado, pois se trata de um dos algoritmos mais básicos e rápidos usados na biblioteca do StableBaslines3, caso formos comparar com outros algoritmos de RL [23]. Com os resultados não foram os melhores, com uma média de recompensa baixa, além de uma Policy e Value loss que se estabiliza por volta de 4 Milhões de TimesSteps.

A explicação de tanto o Policy loss quanto o Value loss se converge rapidamente vem do cálculo da função crítica base, que usa o método de Temporal Difference (TD), que se utiliza dos valores atuais para estimar e ajustar valores futuros de Policy loss e Value loss, antes mesmo que termine o treinamento diferenciando do método de Monte Carlos. Por

fim, a dificuldade em alcançar uma alta recompensa pode ser atribuída a diversos fatores. Tal como, a forma síncrona que se usa, podendo trazer velocidade ou lentidão à IA, já que tanto o ator e crítico devem depender de suas atualizações, ou também pode ser causado pelos agentes estarem operando em versões desatualizadas dos parâmetros, diferentes do ambiente atual, causando imperfeições entre a comunicação entre os agentes, o que pode ser válido, tendo em vista que o tempo de cada época é variável, em que reseta toda vez em que o oponente se recupera do *hit stun*.

TABLE I
DURAÇÃO MÁXIMA DE COMBO - A2C

	Duração (frames)
Média	147.35
Melhor	284.00

TABLE II
TEMPO DE EXECUÇÃO - A2C

Timesteps	**Tempo de Execução (h)**
2M	01:00
4M	02:12
6M	03:36
8M	05:00
10M	06:30

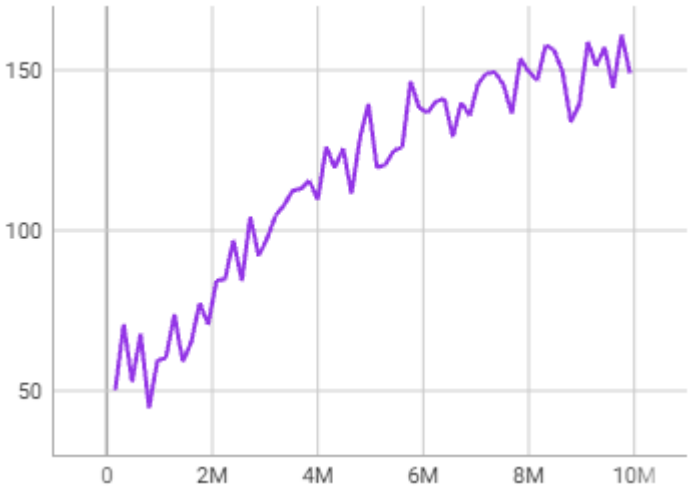

Fig. 1. A2C - Recompensa média durante treino

B. PPO

O PPO é categorizado como método de gradiente de política, ou seja, ele atualiza sua política em pequenos passos(step size), com o objetivo de alcançar a solução máxima para o ambiente em que se está treinando. Em casos de que o algoritmo percebe que não está recebendo altas recompensas ou ainda está no começo do treinamento, ele aumenta o tamanho das atualizações até achar um caminho ótimo para se seguir.

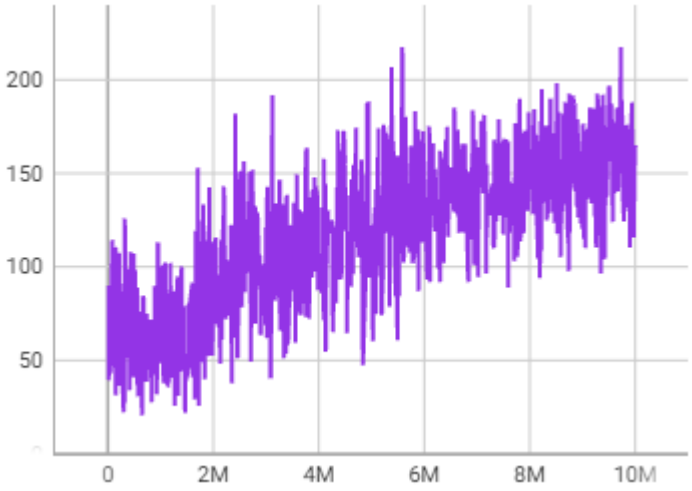

Fig. 2. A2C - Recompensa média durante validação

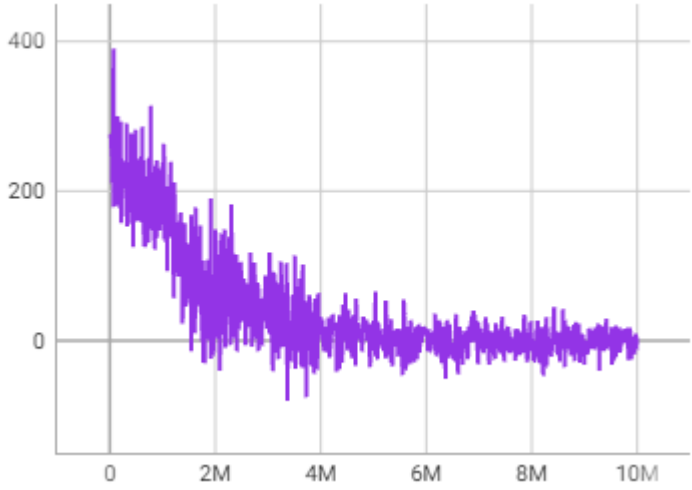

Fig. 3. A2C - Policy Loss

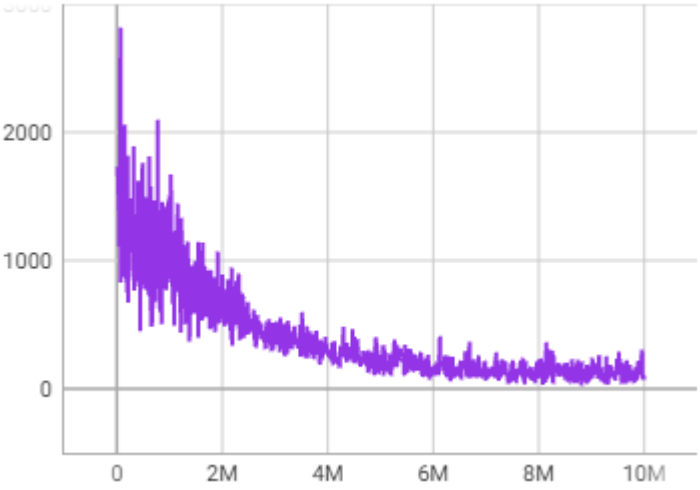

Fig. 4. A2C - Value Loss

Pelo fato de ser um algoritmo mais complexo é esperado que chegue em resultados melhores. Combinando conceitos de múltiplos trabalhadores em A2C e uma região de confiança para o ator, que ajusta dinamicamente de acordo com a quantidade de recompensa e Policy Loss.

Assim gerando ótimos resultados, tanto pelas recompensas melhores que A2C e uma estabilidade muito mais rápida dos parâmetros de Policy e Value Loss. Podendo ser explicado pelo fato de ele ter encontrado, por sorte, o melhor caminho promissor nos treinos iniciais, sendo assim pouca necessidade de fazer grandes atualizações de política. Com um adendo de que o tempo de execução é maior do que o A2C por causa que é um algoritmo mais complexo e por causa das recompensas maiores, ou seja, mais tempo executando o combo.

Observando o gráficos de Recompensa Média podemos notar um alto grau de crescimento de recompensas nos treinos iniciais, que diminui gradualmente a partir de certo ponto e se torna mais inconstante, com alguns altos e baixos, porém ainda em crescimento. Nesse momento o algoritmo está aplicando o conceito de Clipping, fazendo recortes especializados no caminho em que a política está seguindo para encontrar ações melhores, um processo lento pois o objetivo é não se afastar muito de sua melhor recompensa, assim garantindo um processo mais estável.

Isso se encaixa perfeitamente para o problema que estamos tratando. Na qual limitamos para que use apenas um personagem e no melhor estado inicial, posicionado no canto direito da tela. Essa abordagem nos permite gerar resultados muito mais rápidos e precisos, focando no maior combo possível dentro dessas condições específicas.

TABLE III
DURAÇÃO MÁXIMA DE COMBO - PPO

	Duração (frames)
Média	249.22
Melhor	511.00

TABLE IV
TEMPO DE EXECUÇÃO - PPO

Timesteps	**Tempo de Execução (h)**
2M	01:36
4M	03:24
6M	05:18
8M	07:12
10M	09:12

C. GA

Foi feita a comparação dos resultados gerados pelos algoritmos em RL com o trabalho anterior usando GA. O objetivo dessa comparação é avaliar qual dos modelos é mais eficaz para otimizar o desempenho em tempos de tempo de execução de combos em jogos de luta. Essa análise visa complementar nosso estudo, ajudando a determinar qual abordagem oferece

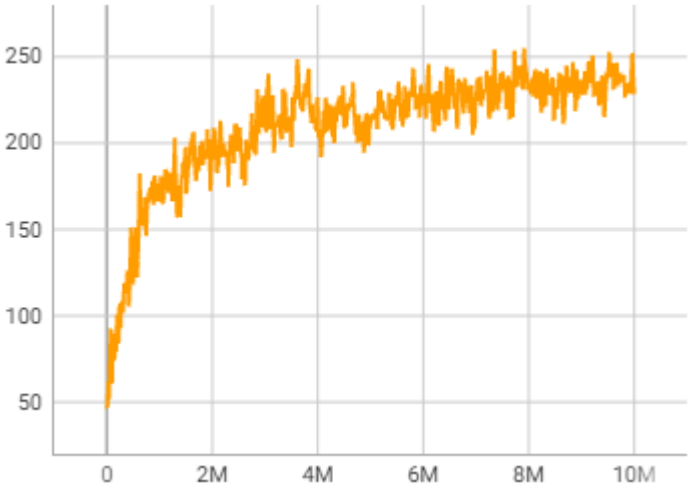

Fig. 5. PPO - Recompensa média durante treino

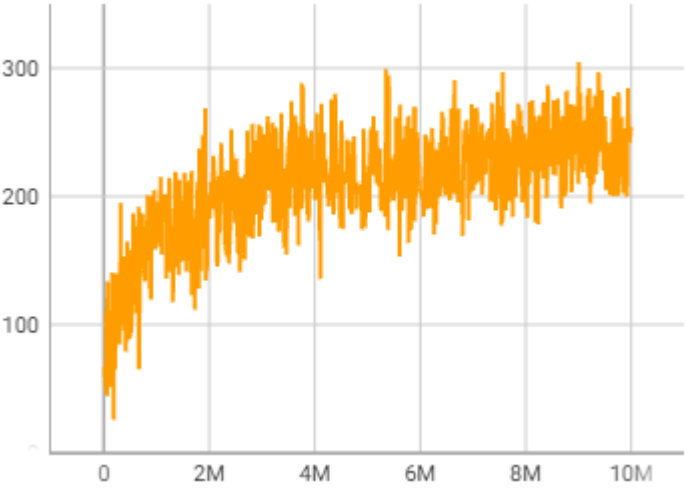

Fig. 6. PPO - Recompensa média durante validação

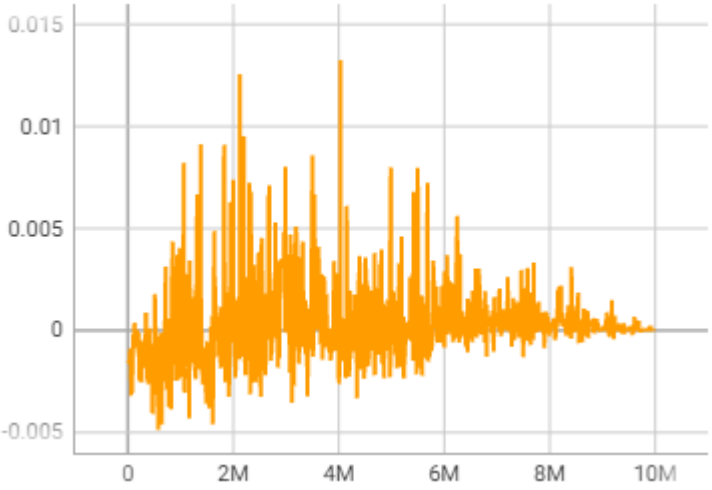

Fig. 7. PPO - Policy Loss

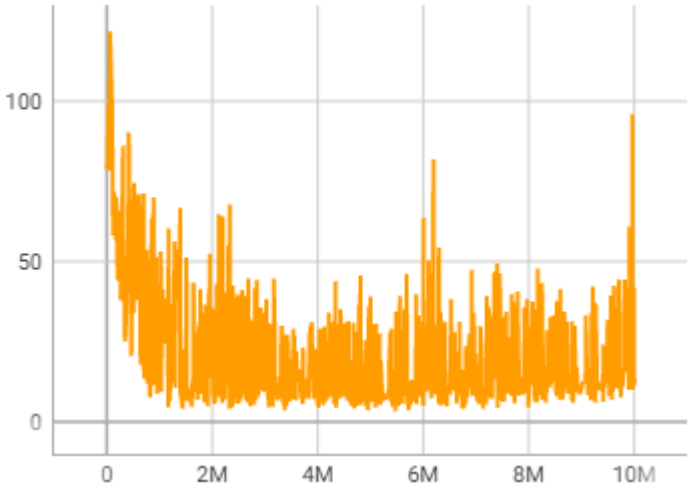

Fig. 8. PPO - Value Loss

TABLE V
TEMPO DE EXECUÇÃO - A2C VS PPO

Número de passos (*timesteps*)	**Tempo de Execução (h)**	
	A2C	***PPO***
2M	01:00	01:36
4M	02:12	03:24
6M	03:36	05:18
8M	05:00	07:12
10M	06:30	09:12

melhores resultados e maior eficiência na busca por bons combos.

A adaptação feita é capaz de compararmos a duração máxima e média do modelo de GA com os dados de recompensa de RL. Assim, podendo gerar análises mais perceptíveis entre dois modelos diferentes.

Os dados de duração média de combo gerado pelo modelo GA são comparáveis aos de recompensa obtidas pelo algoritmo PPO. Observa-se um grande salto na duração média nos treinos iniciais, seguido por uma variação considerável ao longo do tempo, sem um grande crescimento do tempo do combo. Esse comportamento pode indicar que o modelo encontrou a sequência de combo máximo local com maior facilidade em comparação com outros modelos, mas teve dificuldade de caminhar em direção ao máximo global.

Esse fase de convergência é pelos processos de cruzamento e mutação terem dificuldade em encontrar ações que possuem impacto significativo no ambiente. Isso ocorre porque o algoritmo não possui conhecimento explícito sobre quais pontos devem ser particionados ou explorados. Como resultado, possui essa dificuldade em aprimorar em casos muitos específicos de ações, levando a uma perda de variabilidade genética.

Algo curioso ocorre por volta da geração 650, em que a fitness máxima aumenta, mas a duração máxima do combo, não. Observando o cromossomo, ele obtém combo máximo de apenas 156 frames, ao invés de 470 frames como encontrado anteriormente. Isso ocorre porque a fórmula usada para o cálculo de fitness (Equação 1) recompensa outros combos além do mais longo encontrados (para favorecer o crossover), dividindo a duração total desses por 1 mais número de frames entre um combo e outro. O problema é que isso assume que entre quaisquer dois combos vai ter no mínimo um intervalo de 1 frame, o que não é o caso. Se o personagem acertar o oponente no frame exato em que ele se recupera do *hit stun*, ele poderia se defender, o que significa que o combo acabou e um novo começou, mas a função de fitness calcula como se fosse um único grande combo. Isso poderia ser evitado aumentando o valor pelo qual combos adicionais são divididos por números maiores do que 1 mais o número de frames do intervalo.

TABLE VI
DURAÇÃO MÁXIMA DE COMBO - GA

	Duração (frames)
Média	218.56
Melhor	470.00

TABLE VII
TEMPO DE EXECUÇÃO - GA

Geração	**Tempo de Execução (h)**
200	00:42
400	01:18
600	01:54
800	02:36
1000	03:12

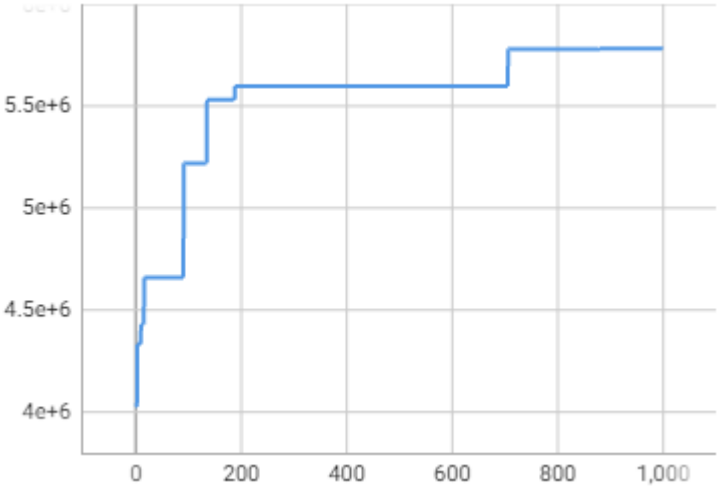

Fig. 9. GA - Melhor fitness por geração

VIII. CONCLUSÕES E TRABALHOS FUTUROS

Tanto o PPO quanto o GA se mostraram muito eficazes em encontrar combos, se aproximando do que é possível dentro do jogo, enquanto o A2C teve um desempenho inferior. O GA encontrou muito rápido o seu melhor combo, mas ficou preso no máximo local e não conseguiu mais progredir, enquanto

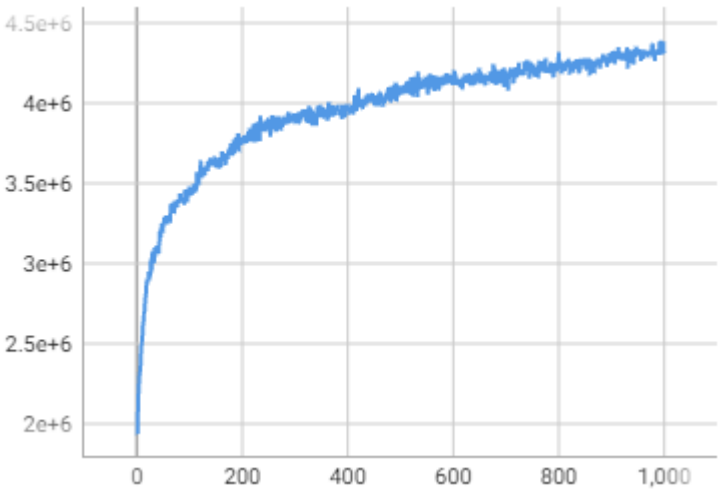

Fig. 10. GA - Fitness média por geração

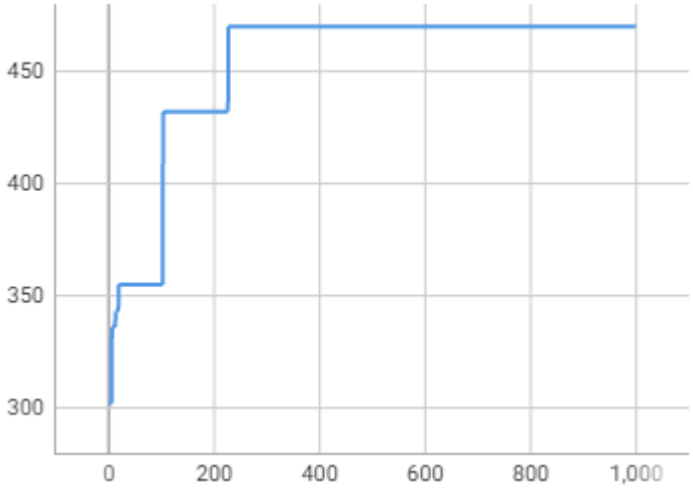

Fig. 11. GA - Duração de combo mais longo por geração

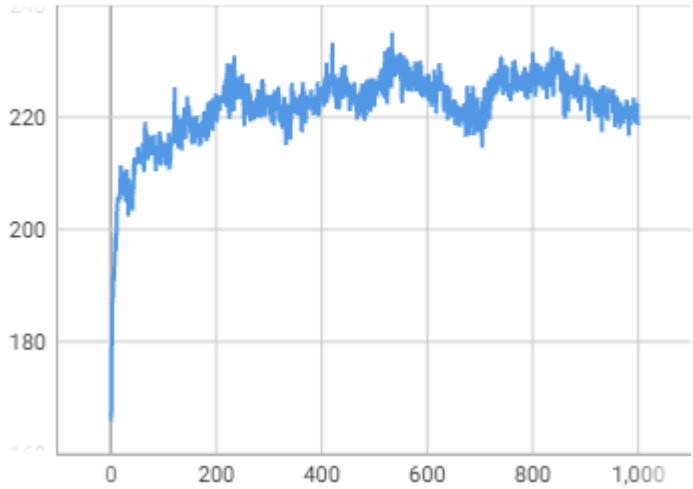

Fig. 12. GA - Duração de combo média por geração

TABLE VIII
DURAÇÃO DE COMBO MÁXIMO - A2C VS PPO VS GA

	Duração de combo máximo (frames)		
	A2C	***PPO***	***GA***
Média	147.35	249.22	218.56
Maior	284.00	511.00	470.00

o PPO demorou muito mais para executar, mas progrediu constantemente até ultrapassar os demais.

Levando em conta a motivação inicial de criar modelos para testar jogos de luta e ajudar no balanceamento, seria recomendado utilizar o GA durante o desenvolvimento para testar conforme faz ajustes no jogo. Após alcançar uma versão em que pretende fazer mudanças com menos frequência, é interessante utilizar o PPO para detectar problemas que o GA pode ter deixado passar.

Este trabalho focou apenas no aspecto de duração do combo para reduzir o escopo do projeto, mas é possível facilmente adicionar objetivos como dano, geração de recursos ou qualquer outro aspecto do combo, desde que seja uma informação acessível a partir do ambiente. Foi utilizado um jogo existente rodando no emulador para testar o funcionamento em jogos reais, mas a implementação pode ser ainda mais fácil caso haja acesso ao código fonte do jogo, o que é o caso quando se está desenvolvendo o jogo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] G. L. Zuin, Y. P. A. Macedo, L. Chaimowicz e G. L. Pappa, “Discovering combos in fighting games with evolutionary algorithms,” em “Proceedings of the Genetic and Evolutionary Computation Conference 2016,” pp. 277–284.
[2] J. Laitano “Conceitos para entender reinforcement learning,” 2021.
[3] T. Simonini “Advantage Actor Critic (A2C),” 2022. https://huggingface.co/blog/deep-rl-a2c
[4] T. Simonini “Proximal Policy Optimization (PPO),” 2022. https://huggingface.co/blog/deep-rl-ppo
[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, “Proximal Policy Optimization Algorithms,” em “arXiv:1707.06347,” 2017.
[6] GeeksForGeeks. Genetic algorithms. https://www.geeksforgeeks.org/genetic-algorithms/
[7] Ludopedia. Mecânicas. https://ludopedia.com.br/mecanicas
[8] Technopedia. Game balance. https://www.techopedia.com/definition/27041/game-balance
[9] Technopedia. Fighting game. https://www.techopedia.com/definition/27154/fighting-game
[10] The Fighting Game Glossary. Frame. https://glossary.infil.net/?t=Frame
[11] The Fighting Game Glossary. Hit Stun. https://glossary.infil.net/?t=Hit%20Stun
[12] The Fighting Game Glossary. Combo. https://glossary.infil.net/?t=Combo
[13] G. Zuin e Y. Macedo. “Attempting to discover infinite combos in fighting games using hidden markov models,” em SBGames, pp 149—157, 2015.
[14] M. Hasan “Street Fighter II is hard, so I trained an AI to beat it for me,” 2020.
[15] J. J. Luo, B. Guo, C. Han, C. Shearer, J. Qu, e K. Osborne. “An exploration of neural networks playing video games,” 2019.
[16] I. Oh, S. Rho, S. Moon, S. Son, H. Lee e J. Chung, ”Creating Pro-Level AI for a Real-Time Fighting Game Using Deep Reinforcement Learning,” in IEEE Transactions on Games, vol. 14, no. 2, pp. 212-220, Jun 2022.
[17] K. Shao, Z. Tang, Y. Zhu, N. Li, e D. Zhao, ”A Survey of Deep Reinforcement Learning in Video Games,” ArXiv, Dez 2019.

[18] C. Berner *et al*. "Dota 2 with Large Scale Deep Reinforcement Learning," ArXiv, vol. abs/1912.06680, 2019.
[19] Basics of Reinforcement Learning for LLMs. https://cameronrwolfe.substack.com/p/basics-of-reinforcement-learning
[20] Epsilon-Greedy Algorithm in Reinforcement Learning. https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/
[21] OpenAI. OpenAI Five defeats Dota 2 world champions. https://openai.com/index/openai-five-defeats-dota-2-world-champions/
[22] Kristensen, J. Theiss, Burelli e Paolo, "Strategies for using proximal policy optimization in mobile puzzle games," em "Proceedings of the 15th International Conference on the Foundations of Digital Games 2020," pp. 1–10.
[23] Mehta e Deepanshu, "State-of-the-Art Reinforcement Learning Algorithms," em "International Journal of Engineering and Technical Research," V8, 2020.