SIMULADOR DE BATALHAS POKEMON

Eduardo Dalmás Faé, João Pedro Kuhn Braun
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre
{edfae, jpkbraun}@inf.ufrgs.br

ABSTRACT

Esta pesquisa aborda uma simulação simplificada de batalhas pokemons, o treinamento de agentes por aprendizado por reforço nesse ambiente utilizando os seguintes algoritmos: A2C, DQN, PPO e RPPO, sendo esses testados em combates entre si para medição do desempenho alcançado. Os algoritmos foram treinados utilizando uma função de recompensa simples, e uma mais elaborada, integrando recompensas a curto prazo. Eles foram divididos em grupos e colocados para disputar combates entre si, e foi observado, com exceção do A2C, que a função de recompensa mais elaborada capturou melhor as nuâncias de uma batalha, acelerando a convergência desses agentes. A simulação atual ainda traz muitas simplificações, possibilitando futuras melhorias em muitos aspectos nesta pesquisa.

***K*eywords** Aprendizado por Reforço · Batalha Pokemon · Simulação Simplificada

1 Introduction

Máquinas podem pensar Turing (1950)? Alan Turing se fez essa pergunta em 1950, e desde então muitos cientistas se propuseram a pesquisar e dedicar as suas vidas para esse tema e suas variações. Em 1959, cunhou-se o termo *machine learning* (aprendizado de máquina) devido à pesquisa de Samuel (1959), na qual um agente foi treinado para jogar damas, um dos primeiros passos da época na direção de computadores se tornarem capazes de resolver/jogar jogos desenvolvidos por humanos.

Desde então, muita pesquisa foi realizada na área de aprendizado de máquina e aprendizado por reforço. Entre muitos marcos nesses campos de conhecimento, podemos citar Deep Blue Campbell et al. (2002), IA que foi capaz de vencer o campeão de xadrez mundial da época, Garry Kasparov. Mais tarde, AlphaGo Silver et al. (2016) e AlphaZero Silver et al. (2017) vieram para derrotar o campeão mundial de Go, Lee Sedol, e propor uma técnica para treinar um agente para jogar com desempenho superhumano diversos jogos de tabuleiro, sem necessidade de aprendizado supervisionado ou anotações de especialistas para aprender boas jogadas, respectivamente.

Indo além de jogos que podem ser representados por um domínio discreto, também houveram grandes contribuições, como DQN para jogar Atari Mnih et al. (2015), AlphaStar Vinyals et al. (2019) e OpenAI-5 OpenAI et al. (2019). DQN (*Deep-Q Network*) é muito semelhante a *Q-learning* tabular, porém no lugar da tabela de transições para cada par de ação e estado, uma rede neural se torna a responsável por aprender essa representação e, a partir de um estado, decidir a ação que deve ser tomada. Ela foi utilizada pela primeira vez para jogar diversos jogos do Atari, em 2013, resultando em uma publicação mais elaborada em 2015. AlphaStar e OpenAI-5 também tratam de jogos representados por um domínio contínuo, sendo o primeiro responsável por alcançar domínio altamente competitivo em StarCraft II e o segundo por vencer dos campões mundiais do mundial de Dota 2 no ano de 2019.

Sabemos que as aplicações de aprendizado por reforço estão distantes de estarem restritas a jogos. É importantíssimo salientar que muito dessa pesquisa exposta demonstra uma prova de conceito, até onde vai o nosso alcance, o tamanho de nosso horizonte. Após descobrirmos o que é possível no universo de jogos e simulações, abstraímos esse conhecimento e o aplicamos em diversas áreas como a robótica, a qual se beneficia muito de treinos de agentes em simulações para futura aplicação no mundo real, o treinamento de grandes modelos de linguagem Ouyang et al. (2022), em que através

do *feedback* humano é possível melhorar a qualidade das respostas desses modelos através de apredizado por reforço, entre muitas outras.

Dessa forma, enxergamos a simulação de batalhas entre pokemons como uma oportunidade de provar um conceito. Esse jogo tem características únicas que o fazem ser um tópico interessante de ser pesquisado. O combate em turnos não é trivial, pois a ação dos dois jogadores ocorre simultaneamente, o jogo tem muitas características a serem levadas em conta, então tem um grande espaço de estados, e está sempre em constante desenvolvimento, tornando-se desafiador de sua modelagem estar atualizada. Não somente a comunidade científica se beneficia da exploração desse domínio, como também a sua grande comunidade, que tem interesse nesses assuntos.

2 Conceitos Básicos

Pokemon é uma franquia da Nintendo que começou como um jogo no ano de 1996, que com o passar dos anos evoluiu para muitas outras categorias de conteúdo, tornando-se mundialmente reconhecida. Entre muitas das dinâmicas acerca do jogo, que trata desde narrativa de uma história até explorações de cenários e segredos, pode-se afirmar que uma das mecânicas centrais que traz vida ao jogo são as batalhas entre pokemons.

Pokemons são seres vivos nesse mundo de fantasia que se assemelham com animais, e treinadores desses pokemons os capturam e usam para batalhar entre si, dinâmica essa importantíssima e muito levada a sério nesse mundo. Esses seres vivos, no contexto da batalha, possuem tipos, quatro movimentos, cada um possuindo tipos e poderes, atributos que ditam a sua "qualidade", etc. A batalha ocorre em turnos em que, simultaneamente, cada treinador pokemon decide se deseja trocar o pokemon atual para outro de sua equipe, se possível, ou atacar com um de seus 4 movimentos válidos. Caso um único treinador decida trocar, a troca ocorre primeiro, caso ambos troquem, a ordem não importa, e se ambos pokemons atacarem, executa sua ação primeiro o mais veloz.

Seus tipos são importantes de serem levados em consideração também no contexto da batalha. Existem 18 tipos no mundo pokemon, cada um deles possuindo características positivas ou negativas em relação aos outros, como duplicar, cortar pela metade ou anular o dano causado pelo ataque de um oponente. Além disso, quando um pokemon ataca utilizando um movimento categorizado com seu próprio tipo, o poder deste golpe é multiplicado por um fator de 50%, dinâmica relevante para o combate.

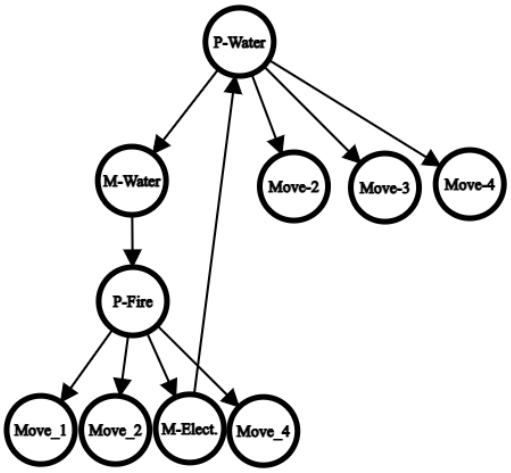
3 Trabalhos Relacionados

Batalha Pokemon Trabalhos anteriores já modelaram ambientes de batalha pokemon para treino e teste de agentes. Simões et al. (2020) definiram um ambiente simplificado de pokemon para iniciar seus experimentos, reduzindo enormemente a complexidade do jogo para tal. Para o treino de seus agentes, foram utilizados dois algoritmos, WPL (*Weighted Policy Learner*) Abdallah and Lesser (2014) e GIGAWoLF (*Generalized Infinitesimal Gradient Ascent Win or Learn Fast*) Bowling (2004) em cima de uma configuração de times pokemons semi-aleatória, para então serem testados em um ambiente determinístico criado pelos próprios autores, para descobrir se os agentes aprenderam a executar uma ação ruim a curto prazo, trocar de pokemon, para uma possível vitória num horizonte mais distante. O presente trabalho utilizou de um ambiente mais complexo, além de definir mais agentes, com diferentes algoritmos para seus treinamentos.

Padronização de Batalhas Houveram tentativas de padronizar o cenário geral de batalha e modelação de um ambiente pokemon, além de permitir a integração de agentes treinados com simuladores de batalhas pokemon grandemente utilizados hoje em dia, como Pokemon Showdown. Entretanto, trabalhos como os de Sahovic falham hoje em executar esse proposta com êxito, visto alguns problemas de uso e falta de documentação adequada. Esta pesquisa não integra o agente treinado com simuladores para interação e teste com outros jogadores, porém traz uma padronização não só de um ambiente de batalha pokemon através do uso do PettingZoo Terry et al. (2021), mas também traz padronização no treino e uso de agentes nesse ambiente.

Montagem de Equipe Uma etapa anterior ao combate em batalhas é a montagem de uma equipe adequada de pokemons. Reis et al. (2021) acompanham o *meta game* atual do simulador de batalha, Pokemon Showdown, para restringir as possibilidades de diferentes opções para criar equipes, focando em estratégias voltadas para o jogo competitivo quando nessa criação. Este trabalho, no intuito de generalizar para o agente poder batalhar em toda e qualquer situação em que ele se encontrar, foca em um *meta game* parcial ao gerar os times dos agentes, mas ainda mantém uma aleatoriedade forte, propiciando generalização para os mais diferentes cenários.

Figure 1: Exemplo da estratégia de movimentos de cobertura. Na imagem, é evidente que um pokemon do tipo fogo é frágil contra água, então, a estratégia de nossa configuração *meta aware* traz, como um possível movimento de cobertura, o tipo fogo ter um ataque elétrico, que é forte contra água, para poder se defender de adversários fortes contra ele.



4 Metodologia

Inicialmente, nosso foco foi desfazer gradualmente as simplificações no ambiente de batalha pokemon realizadas por Simões et al. (2020), usando seu trabalho de base para o início da pesquisa. Reintroduzimos tipos duplos para os pokemons, que agora quando gerados podem representar toda a tipagem encontrada nesse universo, e criamos uma nova configuração para o ambiente de batalha pokemon, chamando-a de *meta aware*. Tal configuração reflete uma das possíveis estratégias usadas por jogadores de alto nível em batalhas pokemon, que consiste em escolher movimentos da (s) tipagem (ns) dele próprio e, para os faltantes, selecionar o que chamamos de movimentos de cobertura, como segue na Figura 1. Para um pokemon do tipo X com fraqueza Y, pode ser selecionado então um golpe caracterizado como Z, o qual Y é fraco contra, seguindo uma linha de raciocínio semelhante a "O inimigo de meu inimigo é meu amigo" (um golpe poderoso contra um inimigo poderoso contra mim deve ser importante).

O código é original de 2020, utilizando-se da versão 1.x do TensorFlow para realizar o treinamento de seus agentes, com os algoritmos WPL Abdallah and Lesser (2014) e GIGAWoLF Bowling (2004). Após múltiplas tentativas de adaptação do código para alguma versão 2.x do TensorFlow ou de execução em retrocompatibilidade de alguma versão mais recente com o código antigo, optamos por utilizar a biblioteca de Python PettingZoo Terry et al. (2021), visto que possui muita padronização para treinamento multiagente.

Treinamos quatro agentes com algoritmos diferentes da biblioteca Stable-Baselines3 Raffin et al. (2021), sendo estes PPO (*Proximal Policy Optimization*) Schulman et al. (2017), DQN (*Deep-Q Networks*) Mnih et al. (2013), RPPO (*Recurrent* PPO), uma versão do PPO que utiliza de LSTMs (*Long Short-Term Memory*) Hochreiter and Schmidhuber (1997) para implementar recorrência, e A2C, uma versão de algoritmo *actor-critic* da implementação de Mnih et al. (2016) que, na implementação dentro do Stable-Baselines3, não é assíncrono que nem a versão A3C. A escolha desses algoritmos para o treino dos agentes foi devido ao conhecimento de alguns, vistos em sala de aula a respeito de seu funcionamento, e também ao seu bom desempenho evidenciado na literatura, tendo cada um treinado por 1000000 de interações com o ambiente.

Além disso, no ambiente simplificado de pokemon original foi feita uma engenharia de recompensa, mudando a dinâmica óbvia de uma recompensa de +1 quando o agente obtém uma vitória e -1 quando derrotado para uma equação que leva em consideração alguns retornos imediatos, sendo estes o dano causado e recebido por seus pokemons e um retorno positivo quando um pokemon adversário é derrotado. Segue, na Equação (1), a descrição matemática do cálculo da recompensa. Observa-se que o retorno para o agente tende a ser mais positivo do que negativo, o que pode ocasionar recompensas positivas mesmo em cenários de derrota, como no caso de um agente X derrotar 1 pokemon do adversário e deixar o segundo com metade de seus pontos de vida, perdendo a partida e recebendo, assim, +0,5 pontos de recompensa.

$$r_t = \sum_{i=1}^{2} \left(\Delta h_{i,t}^{\text{enemy}} + \Delta k_{i,t}^{\text{enemy}} \right) - \sum_{j=1}^{2} \Delta h_{j,t}^{\text{ally}}, \qquad -2 \leq r_t \leq 4 \tag{1}$$

Também houve a necessidade de adicionar um fechamento novo no combate pokemon: o empate. Numa batalha tradicional do jogo não simplificado, não há o empate, pois em algum momento ela vai ter um fim e algum jogador terá a vitória. No entanto, foi observado durante o treinamento e teste dos agentes que, durante o combate, duas situações inconvenientes podem ocorrer: ambos os jogadores decidem que a melhor ação é trocar o seu pokemon para sempre e, como esse movimento válido não causa dano, a batalha não se encerra, ou ambos não obtiveram conhecimento o suficiente para generalizar para todos os cenários, não aprendendo que alguns movimentos são completamente anulados contra alguns tipos, fazendo com que ninguém cause dano, tornando o jogo infinito. Devido a essas exceções, foi necessária a implementação do empate, que ocorre quando ambos os agentes, num combate, executarem consecutivamente quatro ações que não causem dano nenhum em seu oponente, sendo esta mecânica somente implementada para os agentes treinados com a fórmula tradicional da recompensa, ocasionando menos um de retorno para ambos os agentes.

Visto essa situação, optamos por treinar os quatro agentes na configuração do ambiente de batalha como *meta aware*, com a função de recompensa original e a função de recompensa simplificada, um ponto para a vitória, menos um para a derrota do agente. Assim, foram treinados oito agentes, que para a avaliação de seus desempenhos foram colocados para batalhar entre si, exceto contra si mesmo, em grupos de quatro divididos pela função de recompensa que foi utilizada para treiná-los. Após isso, cada algoritmo lutou contra sua versão de recompensa alternativa, na tentativa de medir se o agente com a recompensa simples ou a original teria um melhor desempenho. Para cada teste, foram utilizadas duas configurações de ambiente, a nossa, *meta aware*, e uma configuração determinística, em que o jogador um começa com uma grande vantagem contra seu oponente, e se corretamente aproveitada deve garantir uma taxa de vitória de 100%. Para todos os jogos, os agentes ocuparam a posição de jogador 1 e também de jogador 2, resultando então em 64 jogos no total.

5 Resultados e Discussão

Houveram no total 64 jogos, dos quais 16 foram as versões com recompensa padrão *versus* recompensa mais elaborada, e 24 de competição interna para cada recompensa, resultando em 48 jogos. Em todas as tabelas, há o sufixo "-old", indicando que foi utilizada a função de recompensa encontrada pelos autores de Pokemon Battle Simulator Simões et al. (2020), enquanto "-new" indica a função de recompensa padrão, de mais um para vitória e menos um para derrotas ou empates. Para cada jogo citado, foram realizados 10000 batalhas de teste.

Table 1: Resultados dos jogos com a função de recompensa antiga, no ambiente determinístico.

Jogador 1	Jogador 2	V	D	L
A2C-old	DQN-old	7560	**0**	2440
A2C-old	PPO-old	7509	**0**	2491
A2C-old	RPPO-old	7549	**0**	2451
DQN-old	A2C-old	**10000**	**0**	0
DQN-old	PPO-old	**10000**	**0**	0
DQN-old	RPPO-old	**10000**	**0**	0
PPO-old	A2C-old	**10000**	**0**	0
PPO-old	DQN-old	**10000**	**0**	0
PPO-old	RPPO-old	**10000**	**0**	0
RPPO-old	A2C-old	7574	**0**	2426
RPPO-old	DQN-old	7548	**0**	2452
RPPO-old	PPO-old	7455	**0**	**2545**

Podemos observar que, na Tabela 1 os agentes DQN e PPO foram capazes de generalizar, mesmo partindo de um treinamento em uma configuração diferente (pois todos agentes foram treinados somente na configuração *meta aware*), para todos os jogos determinísticos. A vantagem entregue ao jogador um garante a sua vitória caso jogue adequadamente. Sendo assim, dois agentes falharam na generalização para esse cenário, sendo estes A2C e RPPO. Apesar disso, é interessante que não houve nenhum empate, todos os caminhos levaram aos encerramentos das partidas.

Na configuração em que os agentes treinaram, como observamos na Tabela 2, os resultados foram diferentes comparativamente com a configuração determinística. RPPO e A2C ocuparam a primeira e segunda posição, respectivamente, em

Table 2: Resultados dos jogos com a função de recompensa antiga, no ambiente *meta aware*.

Jogador 1	Jogador 2	V	D	L
A2C-old	DQN-old	5231	99	4670
A2C-old	PPO-old	4986	116	4898
A2C-old	RPPO-old	4766	143	5091
DQN-old	A2C-old	4626	99	5275
DQN-old	PPO-old	4660	110	5230
DQN-old	RPPO-old	4293	**145**	**5562**
PPO-old	A2C-old	4860	98	5042
PPO-old	DQN-old	5272	121	4607
PPO-old	RPPO-old	4774	119	5107
RPPO-old	A2C-old	5187	123	4690
RPPO-old	DQN-old	**5526**	116	4358
RPPO-old	PPO-old	5171	91	4738

relação a sua taxa de vitórias. Apesar de terem falhado em generalizar para casos de vitória garantida, no ambiente em que foram todos treinados igualmente, esses dois se desempenharam melhor. É possível que RPPO, devido ao uso de LSTM em seu funcionamento, tenha se adaptado melhor para casos adversos ao invés de garantidamente vantajosos, assim como o algoritmo de *actor-critic* não pudesse avaliar tão bem um estado de muita vantagem, mas pôde obter maior êxito em combates "semelhantes" aos vistos durante o treinamento.

Ressalta-se também que, o maior vencedor, RPPO, obteve o maior número de empates, que mesmo sem a informação se foi um empate "bom" (a melhor ação realmente seria trocar de pokemon, o oponente acompanhou e houve empate após muitas trocas) ou "ruim" (o agente foi incapaz de causar dano repetidamente pois não aprendeu ainda a generalizar), devido a ele ter maior estatística em ambos os casos, parecem ter sido "bons" empates.

Table 3: Resultados dos jogos com agentes novos contra agentes antigos, no ambiente determinístico.

Jogador 1	Jogador 2	V	D	L
A2C-new	A2C-old	7504	**0**	2496
A2C-old	A2C-new	**10000**	**0**	0
DQN-new	DQN-old	0	**0**	**10000**
DQN-old	DQN-new	**10000**	**0**	0
PPO-new	PPO-old	**10000**	**0**	0
PPO-old	PPO-new	7437	**0**	2563
RPPO-new	RPPO-old	**10000**	**0**	0
RPPO-old	RPPO-new	**10000**	**0**	0

Com exceção do PPO, todos os agentes que treinaram com a recompensa antiga obtiveram desempenho melhor ou tão bom quanto os agentes novos, para cada algoritmo, como enxergamos na Tabela 3. No âmbito geral, talvez a recompensa simplificada tenha sido incapaz de, num mesmo número de passos treinados, fazer com que os agentes se aproveitassem do cenário vantajoso. Ainda assim, segue inconclusivo, visto que o cenário de vantagem força um tipo específico, então outros cenários de vantagem poderiam ter resultados diferentes.

Table 4: Resultados dos jogos com agentes novos contra agentes antigos, no ambiente *meta aware*.

Jogador 1	Jogador 2	V	D	L
A2C-new	A2C-old	6796	863	2341
A2C-old	A2C-new	2387	841	6772
DQN-new	DQN-old	0	**880**	**9120**
DQN-old	DQN-new	**9184**	815	1
PPO-new	PPO-old	4182	233	5585
PPO-old	PPO-new	5494	236	4270
RPPO-new	RPPO-old	4738	364	4898
RPPO-old	RPPO-new	4781	366	4853

Observando a Tabela 4, vemos que o agente A2C com recompensa simples teve um desempenho melhor que o antigo. No entanto, para todos os outros, a versão da função de recompensa encontrada por engenharia de recompensa parece ter tido desempenho melhor ou semelhante, sendo este último o caso do RPPO, inconclusivo. Forçando uma generalização,

a engenharia de recompensa funcionou e na maioria dos casos, treinar com esse *feedback* mais elaborado, enxergando um pouco a curto prazo, fez sentido e tornou os agentes mais aptos a batalharem. No entanto, não é verdade que existe uma função de recompensa absoluta, que para todo algoritmo treinado utilizando-a ele terá um desempenho melhor, visto o comportamento do ator-crítico. A própria função de recompensa pode ser um hiperparâmetro que pode ser elaborado para cada agente individualmente.

Table 5: Resultados dos jogos com a função de recompensa nova, no ambiente determinístico.

Jogador 1	Jogador 2	V	D	L
A2C-new	DQN-new	0	**10000**	0
A2C-new	PPO-new	**10000**	0	0
A2C-new	RPPO-new	**10000**	0	0
DQN-new	A2C-new	0	**10000**	0
DQN-new	PPO-new	**10000**	0	0
DQN-new	RPPO-new	**10000**	0	0
PPO-new	A2C-new	0	**10000**	0
PPO-new	DQN-new	0	**10000**	0
PPO-new	RPPO-new	**10000**	0	0
RPPO-new	A2C-new	0	0	**10000**
RPPO-new	DQN-new	0	0	**10000**
RPPO-new	PPO-new	6164	0	3836

Até o momento, não houve um único empate nos cenários determinísticos, porém com exceção do RPPO, todos, ou como jogador um ou como jogador dois, empataram, e quando o fizeram, foi 100% dos jogos, como evidenciado na Tabela 5. O desempenho geral também piorou, pois vemos que o RPPO, por exemplo, mesmo com a vitória garantida na posição de jogador um, perdeu 100% das vezes na maioria dos casos. Para o ambiente na configuração vantajosa, nenhum agente foi capaz de garantir a sua vitória absoluta, o que pode indicar que não houve generalização para esse cenário a partir da função de recompensa simples.

Table 6: Resultados dos jogos com a função de recompensa nova, no ambiente *meta aware*.

Jogador 1	Jogador 2	V	D	L
A2C-new	DQN-new	119	**4791**	5090
A2C-new	PPO-new	5720	1049	3231
A2C-new	RPPO-new	6163	885	2952
DQN-new	A2C-new	5207	4667	126
DQN-new	PPO-new	7689	2211	100
DQN-new	RPPO-new	**8203**	1708	89
PPO-new	A2C-new	3114	1087	5799
PPO-new	DQN-new	120	2096	7784
PPO-new	RPPO-new	5029	387	4584
RPPO-new	A2C-new	2999	890	6111
RPPO-new	DQN-new	101	1696	**8203**
RPPO-new	PPO-new	4581	390	5029

Diferente dos melhores agentes observados na Tabela 2, a Tabela 6 demonstra resultados diferentes. DQN e A2C obtiveram a maior taxa de vitórias, respectivamente, enquanto PPO e RPPO tiveram desempenho semelhante, sendo o primeiro marginalmente melhor. Uma função de recompensa diferente resultou, para um treino de mesmo número de passos, hiperparâmetros idênticos e com os mesmos agentes avaliados, resultados qualitativos diferentes, o que pode ser evidência que a função de recompensa pode ser vista como um hiperparâmetro capaz de fazermos um *fine-tuning* para cada agente.

6 Conclusão

Após todo o exposto nesta pesquisa, percebemos que uma engenharia de recompensa parece fazer-se necessária. Uma função de recompensa simples demais falha em capturar toda a nuância de uma batalha pokemon, mesmo que em um cenário simplificado comparado ao jogo original. No entanto, tal função deve ser pensada com cuidado para não dar oportunidade do agente divergir para um comportamento incorreto durante o seu treinamento, ou ficar preso em pensamentos a curto prazo, sendo incapaz de levar a batalha até o seu fim.

Existem também alguns limitadores. O tempo destinado para a pesquisa estava dentro de um semestre letivo, que foi só parcialmente utilizado para o desenvolvimento desta pesquisa. Como estamos lidando com agentes de aprendizado por reforço, esse tempo fica ainda mais restrito, que para todo agente a ser treinado, para cada configuração diferente de hiperparâmetros, há um tempo razoável de espera para a coleta de resultados. Também houveram situações inesperadas, como a atualização de uma biblioteca tornar um código, ainda que comentado, inutilizável, ocasionando na tentativa prolongada de uma refatoração e consequente decisão de descarte de boa parte do trabalho que poderia, na situação ideal, ser reusado.

Para desenvolvimento futuro, surgem diversas oportunidades. O ambiente simplificado ainda tem um longo caminho para se assemelhar ao jogo original, que possui um estado observável muito maior a ser levado em consideração. Limitação de uso de um mesmo movimento, adição de mais quatro pokemons na equipe de cada jogador, movimentos que causam efeitos negativos, atributos, itens, entre outros, entram todos nessa categoria. Há a possibilidade do uso de ferramentas de explicabilidade, ganhando entendimento de quais *features* do estado da batalha são as mais relevantes para a tomada de decisão de cada agente. Foram utilizados somente quatro algoritmos para treinar os agentes, existem muitos mais que podem ser testados no ambiente para gerar mais dados de conflitos. Isso e muito mais pode ser feito como trabalho futuro, agregando na evolução da pesquisa.

References

Sherief Abdallah and Victor Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal Of Artificial Intelligence Research*, 33:521–549, 1 2014. doi: 10.1613/jair.2628. URL http://arxiv.org/abs/1401.3454http://dx.doi.org/10.1613/jair.2628.

Michael Bowling. Convergence and no-regret in multiagent learning. *Advances in Neural Information Processing Systems*, 17, 2004.

Murray Campbell, A. Joseph Hoane, and Feng Hsiung Hsu. Deep blue. *Artificial Intelligence*, 134:57–83, 1 2002. ISSN 0004-3702. doi: 10.1016/S0004-3702(01)00129-1. URL https://www.sciencedirect.com/science/article/pii/S0004370201001291.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997. ISSN 08997667. doi: 10.1162/NECO.1997.9.8.1735. URL https://ieeexplore.ieee.org/abstract/document/6795963.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013. URL https://arxiv.org/pdf/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature 2015 518:7540*, 518:529–533, 2 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2 2016. URL https://arxiv.org/pdf/1602.01783.

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 12 2019. URL https://arxiv.org/pdf/1912.06680.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 3 2022. ISSN 10495258. URL https://arxiv.org/pdf/2203.02155.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Simao Reis, Luis Paulo Reis, and Nuno Lau. Vgc ai competition - a new model of meta-game balance ai competition. *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2021-August, 2021. ISSN 23254289. doi: 10.1109/COG52621.2021.9618985.

Haris Sahovic. Poke-env: pokemon ai in python. URL https://github.com/hsahovic/poke-env.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 7 1959. ISSN 0018-8646. doi: 10.1147/RD.33.0210. URL https://ieeexplore.ieee.org/document/5392560.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov Openai. Proximal policy optimization algorithms. 7 2017. URL https://arxiv.org/pdf/1707.06347.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature 2016 529:7587*, 529:484–489, 1 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 12 2017. URL https://arxiv.org/pdf/1712.01815.

David Simões, Simão Reis, Nuno Lau, and Luís Paulo Reis. Competitive deep reinforcement learning over a pokémon battling simulator. *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 4 2020.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

A M Turing. Computing machinery and intelligence. *Computing Machinery and Intelligence. Mind*, 49:433–460, 1950.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature 2019 575:7782*, 575:350–354, 10 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z. URL https://www.nature.com/articles/s41586-019-1724-z.