



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Nelson Luiz Joppi Filho
Igor May Wensing

**Implementando Monte Carlo Counterfactual Regret Minimization e Selfplay para
Heads Up No-Limit Leduc Poker: Uma Abordagem de Experimentos Incrementais**

Florianópolis, Santa Catarina – Brasil

2023

Nelson Luiz Joppi Filho

Igor May Wensing

**Implementando Monte Carlo Counterfactual Regret Minimization e Selfplay para
Heads Up No-Limit Leduc Poker: Uma Abordagem de Experimentos Incrementais**

Trabalho de Conclusão de Curso submetido ao curso de
Ciência da Computação do Centro Tecnológico da
Universidade Federal de Santa Catarina como requisito
parcial para a obtenção do título de Bacharel em Ciência
da Computação.

Orientador: Prof. Elder Rizzon Santos, Dr.

Florianópolis, Santa Catarina – Brasil

2023

Nelson Luiz Joppi Filho

Igor May Wensing

**Implementando Monte Carlo Counterfactual Regret Minimization e Selfplay para Heads Up
No-Limit Leduc Poker: Uma Abordagem de Experimentos Incrementais**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em
Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em Ciência da
Computação.

Local [inserir local da defesa], [dia] de [mês] de [ano].

Coordenação do Curso

Banca examinadora

Prof. Elder Rizzon Santos, Dr.

Orientador

Prof. Mauro Roisenberg, Dr.

Prof. Alexandre Gonçalves Silva, Dr.

Profa. Cristina Meinhardt, Dra.

Universidade Federal de Santa Catarina

Florianópolis, 2023.

Dedicamos este trabalho às nossas famílias.

“No jogo da vida como no poker, a habilidade supera a sorte a longo prazo.”
(autor desconhecido)

RESUMO

Este trabalho apresenta o desenvolvimento de uma Inteligência Artificial (IA) para jogar Heads Up No-Limit Leduc Poker, utilizando uma abordagem de Aprendizado por Reforço com foco no *Monte Carlo Counterfactual Regret Minimization* (MCCFR). O objetivo da pesquisa foi criar uma IA capaz de competir efetivamente em um ambiente de jogo com informação imperfeita, simulando cenários reais de jogo. O método adotado envolveu uma progressão de experimentos, começando com jogos simples como Pedra, Papel e Tesoura, avançando para o Kuhn Poker, e finalmente implementando a IA no contexto mais complexo do Heads Up No-Limit Leduc Poker. Durante estas etapas, o algoritmo de IA foi continuamente refinado e testado contra a IA DeepStack, uma referência em jogos de poker, para avaliar sua eficácia e ajustar estratégias. Os resultados indicaram uma evolução positiva na capacidade da IA de maximizar ganhos e minimizar perdas, aproximando-se do equilíbrio de Nash. Além disso, uma interface web interativa desenvolvida possibilita jogos entre humanos e a IA, servindo como ferramenta de aprendizado e pesquisa. O estudo reconhece limitações, como a dependência do volume de iterações para o sucesso do algoritmo e sugere melhorias e expansões futuras, incluindo técnicas de redução de variância e adaptação para o poker em escala maior.

Palavras-chave: Aprendizado por Reforço; Monte Carlo Counterfactual Regret Minimization; Inteligência Artificial; Heads Up No-Limit Leduc Poker; Teoria dos Jogos; Simulação.

ABSTRACT

This study focuses on the application of Reinforcement Learning techniques in Artificial Intelligence (AI) to play Heads Up No-Limit Leduc Poker, emphasizing the use of Monte Carlo Counterfactual Regret Minimization (MCCFR). The research aims to integrate game theory and machine learning concepts to develop an AI capable of tackling challenges in scenarios with imperfect information. The incremental approach led to experiments from the simple game of Rock, Paper, Scissors, advancing to Kuhn Poker, and culminating in Heads Up No-Limit Leduc Poker. In these stages, the AI was refined and tested, achieving strategies close to Nash equilibrium. The model's validation was conducted through simulations against the professional AI DeepStack, highlighting significant advances in the AI's ability to optimize gains and reduce losses. Furthermore, a developed interactive web interface enables games between humans and AI, serving as a tool for learning and research. The study acknowledges limitations, such as the dependence on the volume of iterations for the algorithm's success, and suggests future improvements and expansions, including variance reduction techniques and adaptation for poker on a larger scale.

Keywords: Reinforcement Learning; Monte Carlo Counterfactual Regret Minimization; Artificial Intelligence; Heads Up No-Limit Leduc Poker; Game Theory; Simulation.

LISTA DE FIGURAS

Figura 1 - Arquitetura do Libratus.....	18
Figura 2 - Aplicação da resolução de sub-jogos na blueprint original do Libratus.....	19
Figura 3 - Abstração feita por Pluribus.....	21
Figura 4 - Regret minimization.....	22
Figura 5 - Arquitetura em alto nível; Estratégia tabular; Árvore de partida.....	29
Figura 6 - Descrição de um infostate.....	31
Figura 7 - Cálculo de estratégia e soma de estratégias.....	32
Figura 8 - Representação dos infostates para cada nodo decisão de ambos os pontos de vista, conforme a partida da figura 5.....	32
Figura 9 - Exemplo de atualização da estratégia e dos arrependimentos.....	33
Figura 10 - Estrutura Trie para armazenar palavras.....	35
Figura 11 - Estrutura Trie para armazenar histórico de jogadas.....	35
Figura 12 - Ambiente de simulação containerizado da aplicação distribuída.....	47
Figura 13 - Gráfico de ganho acumulado 4kk vs DeepStack.....	48
Figura 14 - Gráfico de ganho acumulado 20kk vs DeepStack.....	50
Figura 15 - Gráfico de ganho acumulado 100kk vs DeepStack.....	52
Figura 16 - Gráfico de ganho acumulado DeepStack, 4kk, 20kk e 100kk.....	53

LISTA DE QUADROS E TABELAS

Quadro 1 – Qualidades das mãos de poker, da mais forte para a mais fraca.....	7
Quadro 2 - Comparação entre IA's dos trabalhos 3.1, 3.2 e 3.3.....	27
Tabela 1 – Resultado do primeiro experimento de Pedra, Papel e Tesoura.....	38
Tabela 2 – Resultado do segundo experimento de Pedra, Papel e Tesoura.....	39
Tabela 3 – Resultado do terceiro experimento de Pedra, Papel e Tesoura.....	39
Tabela 4 – Resultado do treinamento via CFR para Kuhn Poker.....	42
Tabela 5 – Comparativo do equilíbrio de Nash teórico vs CFR.....	42
Tabela 6 – Resultado do treinamento via MCCFR para Kuhn Poker.....	43
Tabela 7 – Comparativo do equilíbrio de Nash teórico vs MCCFR.....	43
Tabela 8 – Dados estatísticos das 20.000 partidas entre 4kk e DeepStack.....	48
Tabela 9 – Dados estatísticos complementares das 20.000 partidas de 4kk e DeepStack.....	49
Tabela 10 – Dados estatísticos das 200.000 partidas entre 20kk e DeepStack.....	50
Tabela 11 – Dados estatísticos das 200.000 partidas entre 100kk e DeepStack.....	52
Tabela 12 – Dados estatísticos das 400.000 partidas entre todas as estratégias e DeepStack.....	54

LISTA DE ABREVIATURAS E SIGLAS

CFR	<i>Counterfactual Regret Minimization</i>
MCCFR	<i>Monte Carlo Counterfactual Regret Minimization</i>
IA	Inteligência artificial
CPU	<i>Central Processing Unit</i>
HUNL	<i>Heads-up no-limit</i>
DT	Distância do transportador
EV	<i>Expected value</i>
mbb	milésimos de big blinds
VMPJ	Valor médio de jogo para o primeiro jogador
AIVAT	<i>Action-informed value assessment tool</i>

SUMÁRIO

1. INTRODUÇÃO.....	3
1.1. OBJETIVO GERAL.....	5
1.2. OBJETIVOS ESPECÍFICOS.....	5
2. FUNDAMENTAÇÃO TEÓRICA.....	6
2.1. POKER TEXAS HOLD'EM.....	6
2.1.1. Regras do Texas Hold'em Poker.....	6
2.1.2. Características do Texas Hold'em Poker.....	8
2.2. IA APLICADA A JOGOS.....	9
2.2.1. Métodos de Monte Carlo.....	9
2.2.2. Teoria dos Jogos.....	9
2.2.2.1. Equilíbrio de Nash.....	10
2.2.3. Algoritmos de aproximação de equilíbrio de Nash.....	10
2.2.3.1. Regret Matching.....	11
2.2.3.2. CFR (Counterfactual Regret Minimization).....	11
2.2.3.3. MCCFR (Monte Carlo Counterfactual Regret Minimization).....	12
2.2.4. Exemplos de IA em Outros Jogos.....	13
2.3. APRENDIZADO DE MÁQUINA.....	14
2.3.1. Aprendizado Supervisionado.....	14
2.3.2. Aprendizado Não Supervisionado.....	15
2.3.3. Aprendizado por Reforço.....	15
2.3.3.1. Exploration vs Exploitation.....	16
2.3.3.2. Selfplay.....	16
2.3.4. Redes neurais e aprendizado profundo.....	17
3. TRABALHOS RELACIONADOS.....	18
3.1. SAFE AND NESTED SUBGAME SOLVING FOR IMPERFECT-INFORMATION GAMES.....	18
3.2. SUPERHUMAN AI FOR MULTIPLAYER POKER.....	20
3.3. COMBINING DEEP REINFORCEMENT LEARNING AND SEARCH FOR IMPERFECT-INFORMATION GAMES.....	22
3.4. OPPONENT MODELING AND EXPLOITATION IN POKER USING EVOLVED RECURRENT NEURAL NETWORKS.....	23
3.5. HEADS-UP LIMIT HOLD'EM POKER IS SOLVED.....	23
3.6. EQUILIBRIUM FINDING FOR LARGE ADVERSARIAL IMPERFECT-INFORMATION GAMES.....	24
3.7. LEARNING USEFUL FEATURES FOR POKER.....	24
3.8. AUTO-ENCODER NEURAL NETWORK BASED PREDICTION OF TEXAS POKER OPPONENT'S BEHAVIOR.....	25
3.9. CONSIDERAÇÕES.....	26

4. DESENVOLVIMENTO.....	28
4.1. ARQUITETURA E MODELAGEM.....	28
4.1.1. Parâmetros de entrada.....	30
4.1.2. Métricas de avaliação.....	33
4.1.3. Otimizações.....	34
4.1.3.1. Uso do formato Pickle.....	34
4.1.3.2. Uso da estrutura de dados Trie.....	34
4.1.3.3. Uso de ferramentas de profiling: line_profiler.....	36
4.1.3.4. Uso da implementação PyPy de Python.....	36
4.2. INTERFACE WEB.....	36
5. EXPERIMENTOS.....	37
5.1. PEDRA, PAPEL E TESOURA.....	37
5.1.1. Regras do jogo.....	37
5.1.2. Experimentos e resultados.....	38
5.1.2.1. Primeiro experimento:.....	38
5.1.2.2. Segundo experimento:.....	38
5.1.2.3. Terceiro experimento:.....	39
5.2. KUHN POKER.....	40
5.2.1. Regras do jogo.....	40
5.2.2. Experimentos e resultados.....	41
5.3. HUNL LEDUC POKER.....	44
5.3.1. Regras do jogo.....	45
5.3.2. Ambiente de simulação DeepStack.....	46
5.3.3. Resultados dos testes.....	47
5.3.3.1. Experimentos envolvendo a estratégia gerada após 4kk de iterações:.....	48
5.3.3.2. Experimentos envolvendo a estratégia gerada após 20kk de iterações:.....	50
5.3.3.3. Experimentos envolvendo a estratégia gerada após 100kk de iterações:.....	51
5.3.3.4. Panorama dos testes.....	53
6. CONCLUSÃO.....	56
6.1. TRABALHOS FUTUROS.....	57
REFERÊNCIAS.....	58
APÊNDICE A.....	63

1. INTRODUÇÃO

Em 1997, o supercomputador Deep Blue (CAMPBELL; HOANE; HSU, 2002) tornou-se a primeira máquina a vencer um campeão mundial de xadrez num torneio com regras de tempo oficiais, derrotando o lendário jogador Garry Kasparov em uma sequência de jogos. Quase duas décadas depois, em 2016, AlphaGo (SILVER et al., 2016) realizou um feito equivalente para o jogo Go, um jogo com muito mais possibilidades de jogadas do que o xadrez, entrando, juntamente com Deep Blue, para a lista de grandes marcos atingidos pela inteligência artificial.

Jogos como xadrez e Go apresentam uma semelhança fundamental: a todo momento ambos jogadores possuem total conhecimento das informações do jogo, conceito chamado de informação perfeita. Alguns jogos, porém, não possuem esta característica, são jogos de informação imperfeita, em que os jogadores podem não ter todas as informações necessárias sobre o jogo, como bridge, batalha naval e poker. Neste caso, métodos tradicionais como busca são insuficientes para a criação de um programa que os joguem profissionalmente.

Neste trabalho focaremos no jogo de Texas Hold'em Poker, que apresenta elementos de sorte, mas que a longo prazo são ofuscados pela habilidade de cada jogador. Já existem pesquisas para esta modalidade de poker que conseguiram criar programas capazes de atingir nível sobre-humano em poucas horas, como em Brown N. e Sandholm T. (2019) porém um total de 12.400 horas de CPU em servidores foram necessárias para seu treinamento. Nossa motivação então é a criação de um programa que utilize ordens de magnitude menos recursos, mas que consiga ter um desempenho melhor do que um jogador médio não profissional.

A estratégia deste jogo não é simples de ser implementada com métodos convencionais, então uma opção que temos é o uso de inteligência artificial para tentar simular o pensamento humano, a qual possui duas grandes abordagens, a simbólica e a conexionista, definidas a seguir (BAJADA, 2019).

A inteligência artificial simbólica é a abordagem clássica, onde as etapas de solução do problema são descritas à máquina por uma pessoa, de forma que seu comportamento é verificável e previsível, o que é muito útil para sistemas que não podem contar com imprevistos, como o de um avião em voo, por exemplo. Existem alguns métodos incluídos aqui, como os de dedução lógica (se isto acontecer, então faça aquilo); de árvores de busca, que criam árvores de possibilidades do que pode acontecer e encontram a melhor solução; de heurística, que não tentam verificar uma árvore de possibilidades completamente, mas sim aproximar uma boa solução em um tempo significativamente mais rápido que o de uma busca

exaustiva; sendo as árvores de busca, e heurísticas muito utilizadas em jogos como xadrez (CAMPBELL; HOANE; HSU, 2002).

Já a inteligência artificial conexionista não depende necessariamente do conhecimento do programador nem de um modelo do problema, como na inteligência artificial simbólica. Ela consegue aprender por conta própria mediante treinamento e dados, gradualmente construindo seu próprio modelo do problema, e melhorando gradualmente seu desempenho. O exemplo mais comum desta abordagem são as redes neurais artificiais, nas quais camadas de nodos são conectadas com pesos entre si, simulando o funcionamento dos neurônios no cérebro. Uma rede neural aprende por meio de repetidas atualizações dos pesos entre os nodos, de modo que quanto mais dados o modelo obter, mais atualizações ocorrerão nos pesos, e melhor essa rede ficará para realizar previsões com novos dados (BAJADA, 2019).

Além da inteligência artificial, outra ideia importante para este trabalho é a ciência de dados, um “conceito para unificar estatística, análise de dados, informática, e seus métodos relacionados”, de modo a “entender e analisar fenômenos atuais” com dados (HAYASHI et al., 2013, tradução nossa). Tal conceito auxiliará na coleta de dados relevantes para análise, desenvolvimento, e visualização de desempenho de nosso programa.

Nossa proposta é utilizar uma abordagem de aprendizagem de máquina que se encaixa na abordagem conexionista da inteligência artificial, uma excelente ferramenta para o reconhecimento de padrões, que pode ser muito bem aplicada no contexto de poker para analisar o comportamento de adversários. Melhorando a capacidade de leitura de jogo do programa, podendo aprender jogando contra si mesma e observando jogos de poker passados. Utilizaremos o aprendizado de máquina em um dos componentes de nossa IA (Inteligência Artificial), juntamente com métodos que procurem em tempo real a melhor jogada para os sub-jogos que a IA encontrará enquanto joga.

1.1. OBJETIVO GERAL

Este trabalho visa criar um programa para o jogo de Texas Hold'em Poker, que utilize aprendizado de máquina para identificar padrões nas jogadas dos adversários, com um sistema de cálculo de probabilidades para auxiliar o programa no processo de tomada de decisões.

1.2. OBJETIVOS ESPECÍFICOS

- Analisar o estado da arte relacionado às técnicas de análise de dados e aprendizado de máquina aplicados ao poker.
- Comparar diferentes tipos e arquiteturas de redes neurais, visando encontrar a melhor arquitetura para nosso problema.
- Desenvolver um programa que faça o cálculo de probabilidades relacionadas ao poker, e que utilize uma rede neural para encontrar padrões de jogada no adversário, conforme a experiência adquirida.
- Delinear e executar um experimento para avaliar a evolução de nosso programa, competindo diferentes versões deste entre si e comparando seus resultados ao longo do tempo.

2. FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo é fornecer a fundamentação teórica necessária para o desenvolvimento do nosso modelo que joga poker. Primeiramente é apresentado, na seção 2.1, o Texas Hold'em Poker, incluindo suas regras (2.1.1), características (2.1.2), e a estratégia envolvida no jogo (2.1.3).

Depois disso, na seção 2.2, são apresentados conceitos importantes associados à aplicação de inteligência artificial em jogos. Isso inclui o método de Monte Carlo e suas variações (2.2.1), equilíbrio de Nash (*Nash Equilibrium*, 2.2.2), e teoria dos jogos (2.2.3), conceitos fundamentais para a tomada de decisões em jogos. Também apresentamos exemplos de aplicações de IA bem sucedidas (2.2.4), tanto para poker quanto para outros jogos, a fim de entender aspectos relevantes que possam ser aplicados ao nosso problema.

Por fim, é explicado o conceito de aprendizado por reforço (2.3.1), foco desta implementação, comparando-o com aprendizado supervisionado e não supervisionado. Depois é explicada a questão de *exploration* vs *exploitation* (2.3.3), bem como conceitos como *selfplay* (2.3.4.1) e o uso de aprendizado profundo no contexto do aprendizado por reforço (2.3.4).

2.1. POKER TEXAS HOLD'EM

2.1.1. Regras do Texas Hold'em Poker

O Texas Hold'em é uma das modalidades mais populares e amplamente jogadas de poker. Se trata de um jogo com um baralho padrão de 52 cartas, que pode acomodar de dois a dez jogadores, porém geralmente jogado com seis. Cada rodada do jogo começa com dois jogadores à esquerda do *dealer* colocando uma aposta obrigatória conhecida como *blind*. O jogador imediatamente à esquerda do *dealer* coloca o *small blind*, que é geralmente metade da aposta mínima, enquanto o jogador à esquerda do *small blind* coloca o *big blind*, que é igual à aposta mínima. Nos referiremos a estes dois jogadores pelo valor de seu *blind*.











Após os *blinds* serem postados, cada jogador recebe duas cartas privadas, distribuídas com a face para baixo. Segue-se uma rodada de apostas, chamada *pre-flop*, começando pelo jogador à esquerda do *big blind*. Os jogadores podem escolher *call* (igualar a aposta atual), *raise* (aumentar a aposta) ou *fold* (desistir da mão).

Em seguida, o *dealer* coloca três cartas comunitárias com face para cima no centro da mesa. Isso é conhecido como o *flop*. Outra rodada de apostas ocorre, começando pelo *small*

bling. Depois disso, uma quarta carta comunitária, conhecida como *turn*, é colocada na mesa, seguida por outra rodada de apostas. Finalmente, a quinta e última carta comunitária, conhecida como *river*, é colocada na mesa, seguida pela última rodada de apostas.

Após a conclusão da última rodada de apostas, se ainda houver dois ou mais jogadores, a mão vai para um *showdown*. Os jogadores revelam suas cartas privadas e vence o jogador que possuir a melhor combinação de cinco cartas dentre suas duas cartas privadas e as cinco cartas comunitárias. Segue abaixo a ordem de qualidade das mãos, da melhor para a pior:

Quadro 1 – Qualidades das mãos de poker, da mais forte para a mais fraca.

Nome	Descrição	Exemplo
Royal Flush	A melhor mão possível no poker, consiste em 10, J, Q, K, A do mesmo naipe.	
Straight Flush	Cinco cartas em sequência do mesmo naipe, como 4, 5, 6, 7, 8 de paus.	
Quadra	Quatro cartas do mesmo valor, como quatro A, mais uma carta desempate.	
Full house	Três cartas do mesmo valor mais duas cartas de outro valor igual, como três Q e dois 6.	
Flush	Cinco cartas do mesmo naipe, sem formar sequência.	
Straight	Cinco cartas em sequência, mas não do mesmo naipe.	
Trinca	Três cartas do mesmo valor, como três K.	
Dois pares	Duas cartas de um valor, mais duas cartas de outro valor, com uma carta desempate, como 9, 9, 5, 5, 3.	
Um par	Duas cartas do mesmo valor, com três cartas desempate.	
Carta mais alta	Nenhuma das mãos acima, vence pela carta mais alta na mão, como um ás.	

Fonte - Autoria própria

O jogador com a melhor mão ganha o pote. Se, a qualquer momento, todos os jogadores desistirem, exceto um, o jogador remanescente ganha o pote sem a necessidade de um *showdown*.

2.1.2. Características do Texas Hold'em Poker

O Texas Hold'em Poker é um jogo que incorpora elementos de chance e estratégia. Apesar da distribuição das cartas ser aleatória e estar além do controle dos jogadores, a habilidade desempenha um papel significativo a longo prazo. Neste jogo, essa é envolvida na capacidade de tomar decisões corretas com base em informações incompletas, avaliar as probabilidades de vitória e manipular os oponentes por meio de apostas estratégicas.

A incerteza é uma característica fundamental no Texas Hold'em Poker. A cada mão, os jogadores devem tomar decisões sem conhecer as próximas cartas distribuídas ou as cartas dos outros jogadores. Isso cria um elemento de risco, exigindo suposições e estimativas. Essa incerteza, por sua vez, destaca a importância da gestão de risco e do cálculo de probabilidades no jogo.

A gestão de risco e o cálculo de probabilidades também são aspectos cruciais. Jogadores devem avaliar o risco em cada decisão de apostar, aumentar, igualar ou desistir, considerando o tamanho do pote, as ações dos adversários, a força da mão e a quantia a arriscar. Calcular probabilidades é essencial para decidir quando apostar, aumentar, igualar ou desistir, levando em conta eventos como completar uma mão vencedora ou a probabilidade de um oponente ter uma mão melhor.

O blefe é outra parte crucial do jogo. Um jogador pode tentar persuadir os outros de que possui uma mão forte quando, na verdade, possui uma mão fraca, ou vice-versa. O blefe pode ser uma estratégia eficiente para ganhar os potes quando a mão do jogador não é suficientemente forte para vencer. Entretanto, o blefe é uma estratégia arriscada, pois pode resultar em grandes perdas se o jogador for pego, logo, é importante usar o blefe somente na hora certa.

Em jogos presenciais, a estratégia do poker também envolve a leitura dos outros jogadores. Isso pode incluir a tentativa de identificar "*tells*" ou hábitos que podem indicar a qualidade da mão de um jogador, bem como tentar prever as ações futuras de um jogador com base nas suas ações passadas. A capacidade de ler com sucesso os outros jogadores, pode dar a um jogador uma vantagem significativa no jogo.

2.2. IA APLICADA A JOGOS

2.2.1. Métodos de Monte Carlo

Os métodos de Monte Carlo são uma classe de algoritmos computacionais que dependem de amostragens aleatórias repetidas para obter resultados numéricos. O nome vem dos cassinos de Monte Carlo, onde os jogos de azar ilustram os fenômenos aleatórios centrais para esses métodos.

No contexto dos jogos, os métodos de Monte Carlo podem ser usados para estimar a probabilidade de diferentes resultados e informar a tomada de decisões. Por exemplo, em um jogo de poker, um dos métodos de Monte Carlo pode ser usado para estimar a probabilidade de ganhar com uma mão particular. Isso é feito simulando diversos jogos distintos em que um jogador específico possui esta mão, e contando a proporção de jogos que resultam em uma vitória para este. (RUBINSTEIN; KROESE, 2016).

Os métodos de Monte Carlo também são úteis para resolver problemas que são complexos ou de alta dimensão. Por exemplo, eles podem ser usados para estimar o valor esperado de uma ação em um jogo de poker, que depende do resultado de todas as possíveis sequências futuras de cartas e ações dos oponentes. Isso seria extremamente difícil de calcular exatamente, mas pode ser estimado de forma eficiente com Monte Carlo.

2.2.2. Teoria dos Jogos

A teoria dos jogos é um ramo da matemática que estuda situações de interação estratégica, onde o resultado para um jogador depende das decisões de todos. Ela inclui conceitos como jogos de soma zero, onde o ganho de um jogador é a perda do outro, e equilíbrio de Nash.

Esta é fundamental para entender a estratégia e a tomada de decisões no poker e em outros jogos (OSBORNE; RUBINSTEIN, 1994). Por exemplo, ela pode ajudar a entender como os jogadores devem ajustar suas estratégias em resposta às ações dos outros jogadores, e como a estrutura do jogo (como a ordem das ações e a informação disponível para os jogadores) afeta a estratégia ótima.

Ela também oferece ferramentas analíticas poderosas para examinar tanto a eficiência quanto a justiça em contextos de jogos. Ao utilizar conceitos como equilíbrio de Nash, a teoria dos jogos permite a análise de estratégias em que nenhum jogador tem incentivo para mudar unilateralmente, indicando eficiência no sentido de que as escolhas dos jogadores

convergem para uma solução estável. Além disso, a teoria dos jogos aborda questões de justiça ao explorar jogos cooperativos, nos quais os participantes podem formar coalizões e negociar para alcançar distribuições de recursos mais equitativas. A capacidade de modelar interações estratégicas e avaliar resultados alternativos faz da teoria dos jogos uma ferramenta versátil para analisar não apenas as dinâmicas eficientes, mas também as questões éticas de justiça em diversos cenários de jogos (BINMORE, 2007).

2.2.2.1. *Equilíbrio de Nash*

O equilíbrio de Nash é um conceito da teoria dos jogos que descreve uma situação em que nenhum jogador pode melhorar seu resultado alterando sua estratégia unilateralmente, desde que as estratégias dos outros jogadores permaneçam inalteradas. Em outras palavras, é um estado de estabilidade onde nenhum jogador tem incentivo para desviar da sua estratégia atual. (NASH, 1951).

No contexto do poker, um jogador está em equilíbrio de Nash se ele ou ela está jogando de uma maneira que o melhor contra-jogo possível do(s) adversário(s) não pode esperar ganhar a longo prazo. Isso significa que a estratégia do jogador é ótima no sentido de que não pode ser “explorada” por um adversário que conhece a estratégia do jogador.

Encontrar o equilíbrio de Nash em um jogo de poker é um problema desafiador devido à complexidade e ao tamanho do jogo. No entanto, existem algoritmos que podem encontrar aproximações do equilíbrio de Nash para jogos simplificados de poker. Esses algoritmos podem ser úteis para desenvolver estratégias de poker que são robustas contra uma variedade de oponentes.

2.2.3. **Algoritmos de aproximação de equilíbrio de Nash**

Para compreender plenamente a dinâmica de equilíbrio de Nash em jogos de informações imperfeitas, como o poker, utilizamos o algoritmo de MCCFR. No entanto, para entendê-lo em profundidade, é essencial introduzir primeiro o CFR e, ainda antes, o *Regret Matching*. Vamos explorar essa progressão detalhadamente, desde o conceito inicial de *Regret Matching* até a complexidade do MCCFR, enfatizando como cada um desses algoritmos representa um avanço em relação ao anterior.

2.2.3.1. *Regret Matching*

O *Regret Matching* é uma técnica aplicada em jogos que requerem apenas uma decisão, como o jogo de Pedra, Papel e Tesoura. Em contraste, o CFR é usado em jogos mais complexos com múltiplos pontos de decisão, como o poker, onde ele incorpora o *Regret Matching* para calcular as estratégias ao longo de sua execução. Este algoritmo é utilizado para minimizar os arrependimentos de ações realizadas. Ele calcula o arrependimento de não ter escolhido determinadas ações e seleciona ações futuras proporcionalmente aos arrependimentos positivos, reduzindo o arrependimento ao longo do tempo e aumentando o valor médio ganho no jogo.

No entanto, apenas minimizar os arrependimentos ao longo do tempo pode deixar o jogador vulnerável a ser “explorado” por oponentes que compreendem o processo de *Regret Matching*. Para lidar com essa questão, o *Regret Matching* é combinado com o *selfplay* para se aproximar de um Equilíbrio de Nash ao longo do tempo, já que ao jogar repetidamente contra si, o algoritmo começa a explorar e compreender suas próprias fraquezas e a evitar ser “explorado” pelos oponentes. Esse processo iterativo de autoaprendizagem e adaptação é fundamental para a convergência do algoritmo para um Equilíbrio de Nash, onde nenhuma ação unilateral pode melhorar o resultado para o jogador, dada a estratégia dos demais jogadores (NELLER; LACTOT, 2013).

2.2.3.2. *CFR (Counterfactual Regret Minimization)*

O CFR (*Counterfactual Regret Minimization*), vai um passo além do *Regret Matching*, e simula todos os cenários possíveis em um jogo com múltiplas situações de escolhas. O que envolve considerar todas as combinações de ações tanto para o jogador atual quanto para o oponente, levando em conta todas as possíveis ramificações do jogo (NELLER; LACTOT, 2013).

Após começar a percorrer a árvore de nodos, o algoritmo chega em nodos terminais, que representam o final de um jogo. Nesses nodos terminais, a recompensa para aquele estado é calculada e repassada para seu nodo antecessor, onde é calculada a recompensa média dos nodos filhos. Essa recompensa média é usada para atualizar os arrependimentos associados aos nodos filhos, fazendo com que decisões com maiores recompensas possuam um arrependimento maior que outros nodos piores.

Após atualizar os arrependimentos dos nodos filhos, o algoritmo passa a recompensa média calculada pelo nodo pai para o seu "avô" na árvore do jogo. Esse processo de atualização de arrependimentos e passagem de recompensas médias ocorre repetidamente, percorrendo todos os nodos da árvore de jogo, até que o algoritmo tenha explorado completamente todos os cenários e retornado ao nodo inicial. Esta atualização de arrependimentos de um nodo proporcional à recompensa de seus nodos filhos é o *Regret Matching*. A diferença é que o CFR atualiza os arrependimentos dos nodos também com base na probabilidade de se alcançar esse nodo ao longo do jogo.

Essa abordagem permite que o algoritmo CFR aprenda e otimize as estratégias de decisão à medida que simula e avalia as possíveis sequências de ações em um jogo, convergindo, eventualmente, para um equilíbrio de Nash.

2.2.3.3. MCCFR (*Monte Carlo Counterfactual Regret Minimization*)

A principal diferença entre o algoritmo de *Monte Carlo Counterfactual Regret Minimization* (MCCFR) e o *Counterfactual Regret Minimization* (CFR) reside no método de amostragem adotado. No MCCFR, em vez de considerar todas as possíveis combinações de ações em um determinado jogo, o algoritmo utiliza uma abordagem de amostragem Monte Carlo para avaliar cenários. Em outras palavras, em vez de analisar todas as decisões viáveis em cada nodo da árvore de decisão, o MCCFR realiza estimativas com base em um conjunto limitado de amostras aleatórias (LANCTOT et al., 2009).

Essa estratégia de amostragem permite que o MCCFR lide de maneira mais eficiente com jogos complexos e extensos, reduzindo consideravelmente a carga computacional em comparação com o CFR. Em jogos com uma árvore de decisão extensa, o CFR se torna impraticável, uma vez que exige a avaliação de todos os nodos. Nesse sentido, o MCCFR surge como uma solução viável e eficaz. Além disso, outra distinção crucial é que o MCCFR incorpora aleatoriedade no processo de aprendizado. Ao invés de considerar todas as ações igualmente, o MCCFR utiliza amostras aleatórias ponderadas para ajustar as estratégias. Essa introdução de aleatoriedade torna o algoritmo mais robusto e imprevisível, o que é uma característica desejável ao lidar com oponentes humanos ou agentes sofisticados.

Vale ressaltar que, assim como ocorre com o CFR, existem diferentes variações do MCCFR. O algoritmo específico que serviu como base para nossa implementação é o mesmo utilizado pelo Pluribus. Nossa implementação escolhe um jogador específico para percorrer a árvore de jogadas possíveis e simula todas as ações possíveis apenas para esse jogador em

particular (BROWN, 2019). Optamos pelo uso da versão do MCCFR implementada no Pluribus devido aos resultados promissores alcançados com recursos computacionais relativamente limitados.

2.2.4. Exemplos de IA em Outros Jogos

A inteligência artificial tem sido aplicada com sucesso em uma variedade de jogos. A Google's DeepMind, por exemplo, desenvolveu o AlphaGo (SILVER et al., 2016), um programa de IA que derrotou o campeão mundial de Go, um jogo complexo. O AlphaGo utilizou uma combinação de aprendizado profundo e aprendizado por reforço para aprender a jogar Go a um nível humano.

Da mesma forma, a IBM criou o Deep Blue (CAMPBELL, 2002), que derrotou o campeão mundial de xadrez Garry Kasparov. O Deep Blue adotou uma abordagem baseada em busca e avaliação para jogar xadrez, em vez de aprendizado de máquina. Mais recentemente, a DeepMind desenvolveu o AlphaZero (SILVER et al., 2017), que utilizou aprendizado profundo por reforço para aprender a jogar xadrez, bem como Go e shogi, a um nível sobre-humano, apenas jogando contra si.

No contexto do poker, o aprendizado profundo por reforço pode ser usado para aprender estratégias complexas e adaptativas que são difíceis de obter com métodos tradicionais. Por exemplo, um agente pode usar o aprendizado profundo por reforço para aprender a blefar ou a responder ao blefe dos oponentes, o que seria difícil de programar explicitamente.

Esses exemplos demonstram o potencial da IA para resolver problemas complexos e fornecem um contexto para a aplicação da IA em jogos, incluindo abordagens baseadas em busca, aprendizado profundo e aprendizado por reforço. No entanto, a aplicação da IA em jogos complexos e de informação incompleta, como o poker, apresenta desafios adicionais.

No "*Handbook of Reinforcement Learning and Control*" (ZHANG et al., 2021), os autores apresentam uma visão geral das teorias e algoritmos de aprendizado por reforço multiagente (MARL, sigla em inglês) e exploram sua aplicação em problemas de decisão complexos e sequenciais, como o jogo de poker, o qual tem sido frequente objeto de estudo na área de MARL, devido à sua natureza compatível com o assunto.

Programas avançados de poker, como DeepStack (MORAVČÍK, 2017) e Libratus (BROWN, 2017), conseguiram êxito no HUNL Texas Hold'em Poker, uma versão do Texas Hold'em Poker onde apenas dois jogadores participam da mesa e não há limite de aposta,

onde o limite seria a própria banca do jogador. Pluribus (BROWN, 2019), construído sobre o sucesso do Libratus, conseguiu superar os melhores jogadores humanos no Texas Hold'em Poker sem limites com seis jogadores. Os autores atribuem o sucesso desses programas a várias técnicas, incluindo abstração e decomposição do sub-jogo para jogos de informação imperfeita em grande escala, o uso de *Monte Carlo Counterfactual Regret Minimization*, *selfplay* e busca limitada em profundidade. No entanto, apesar desses sucessos empíricos, a compreensão teórica dos algoritmos MARL ainda é um desafio, destacando a necessidade de pesquisa futura nessa área.(ZHANG et al., 2021).

2.3. APRENDIZADO DE MÁQUINA

O aprendizado de máquina, é uma área da ciência da computação cujo foco é o desenvolvimento de técnicas para que computadores possam aprender a partir de dados, generalizar este aprendizado, e tomem decisões com base nele, sem a necessidade de serem programados de forma explícita para cada tarefa específica. O objetivo é capacitar as máquinas a reconhecer padrões, fazer previsões e tomar decisões com base nesses padrões. (EL NAQA; MURPHY, 2015).

O aprendizado por reforço, foco do nosso trabalho, é uma categoria dentro do Aprendizado de Máquina, assim como aprendizado supervisionado e não supervisionado (explicadas a seguir). No entanto, ao contrário das outras técnicas, o aprendizado por reforço se difere ao não precisar de um conjunto de dados de treinamento pré-definido, pois aprende através da interação com um ambiente. Toda vez que há essa interação, um novo dado com informações sobre o quão boas ou ruins foram as ações tomadas é gerado, podendo ser usado para que o modelo aprenda. (SUTTON; BARTO, 2018).

2.3.1. Aprendizado Supervisionado

O aprendizado supervisionado é um tipo de técnica em que ensinamos um computador a aprender a partir de exemplos rotulados, onde há descrição relacionada ao dado. Por exemplo: mostramos para a máquina várias fotos de animais, e dizemos qual o nome de cada animal. Após ver muitos exemplos que relacionam a foto com o respectivo animal, a máquina começa a perceber padrões e a relacionar as características dos animais com seus respectivos nomes. Assim, quando mostramos uma nova foto de um animal, a máquina consegue dizer qual é o seu nome com base no que ela aprendeu. Nesse tipo de aprendizado, temos rótulos,

que fornecem respostas corretas para a máquina para cada exemplo (imagem). A máquina utiliza essas informações para ajustar seu modelo e melhorar suas previsões. (EL NAQA; MURPHY, 2015).

2.3.2. Aprendizado Não Supervisionado

O aprendizado não supervisionado é um tipo de técnica em que ensinamos um computador a encontrar padrões em dados sem dar a ele nenhum rótulo. Deixamos a máquina explorar e descobrir coisas por conta própria. Ela olha para os dados e tenta agrupar as informações de maneiras que façam sentido. Como não há rótulos, não ensinamos a máquina o que está certo ou errado, damos liberdade a ela para explorar os dados e encontrar estruturas por conta própria. Ela pode agrupar os dados de acordo com características semelhantes, descobrir relações entre diferentes variáveis, entre outras coisas. (EL NAQA; MURPHY, 2015).

2.3.3. Aprendizado por Reforço

Ao contrário dessas outras formas de aprendizado de máquina explicadas, não são fornecidos dados de treinamento com as respostas corretas no aprendizado por reforço. Em vez disso, o agente aprende por meio da experiência, tentando diferentes ações e recebendo recompensas ou punições com base nesses resultados. Assim, ele precisa descobrir quais ações resultam em recompensas positivas e quais resultam em recompensas negativas. Isso requer uma exploração ativa do ambiente e a busca por estratégias que levem a melhores resultados. (SUTTON; BARTO, 2018).

No aprendizado por reforço, o agente interage com o ambiente em etapas de tempo. A cada etapa, o agente observa o estado atual do ambiente, escolhe uma ação dentre as possíveis, realiza essa ação, recebe uma recompensa (ou punição, que é uma recompensa negativa) e o ambiente transita para um novo estado. O objetivo do agente é aprender uma política, que é uma estratégia que mapeia estados para ações, maximizando assim a soma das recompensas ao longo do tempo. (SUTTON; BARTO, 2018).

2.3.3.1. *Exploration vs Exploitation*

Exploration e *Exploitation* são dois conceitos fundamentais no aprendizado por reforço. *Exploration* refere-se ao agente tentando ações diferentes para aprender mais sobre o ambiente. *Exploitation* refere-se ao agente escolhendo a ação que acredita ser a melhor com base na informação atual. O equilíbrio entre *Exploration* e *Exploitation* é crucial para o sucesso do aprendizado por reforço. (KAELBLING; LITTMAN, 1996).

A *Exploration* é necessária porque o agente precisa tentar várias ações para descobrir quais delas são boas. No início, quando o agente tem pouco conhecimento sobre o ambiente, é benéfico explorar mais para ganhar informações. No entanto, a *Exploration* tem um custo, pois o agente pode receber recompensas menores ao tentar ações desconhecidas. (KAELBLING; LITTMAN, 1996).

A *Exploitation* é necessária porque o agente precisa usar o conhecimento que já adquiriu para obter a maior recompensa possível. À medida que o agente ganha mais experiência e seu conhecimento sobre o ambiente se torna mais preciso, é benéfico explorar menos e “explorar” mais. (KAELBLING; LITTMAN, 1996).

O desafio no aprendizado por reforço é encontrar o equilíbrio certo entre *Exploration* e *Exploitation*. Se o agente explorar demais, ele pode perder oportunidades de obter recompensas altas com as ações que já sabe que são boas. Se o agente explorar muito pouco, ele pode ficar preso em uma estratégia subótima e perder a chance de descobrir ações melhores.

2.3.3.2. *Selfplay*

O *selfplay* (auto-jogo) é uma técnica de aprendizado por reforço na qual um agente aprende a melhorar suas habilidades jogando contra si (TESAURO, 1995). Esta abordagem é especialmente útil em jogos de estratégia, como xadrez, Go e poker, onde a gama de possíveis situações de jogo é vasta e a melhor estratégia depende das ações do adversário (ZHANG et al., 2021). Um exemplo famoso de *selfplay* é o AlphaGo (SILVER et al., 2016), desenvolvido pela DeepMind.

No *selfplay*, o agente começa com uma estratégia inicial, que pode ser completamente aleatória ou baseada em algum conhecimento prévio, e então joga diversos jogos contra si mesmo, atualizando sua estratégia com base nos resultados de cada jogo. (SUTTON; BARTO, 2018). A cada jogo realizado, o agente tem a oportunidade de explorar novas estratégias e aprender com seus erros. Por exemplo, se o agente tenta uma nova ação e perde o jogo como

resultado, ele aprende que essa ação pode não ser a melhor escolha nessa situação específica. Da mesma forma, se o agente tenta uma nova ação e ganha o jogo, ele aprende que essa ação pode ser uma boa escolha.

No contexto do poker, um agente de aprendizado por reforço pode usar *selfplay* para aprender estratégias complexas e adaptativas. Por exemplo, o agente pode aprender quando é melhor blefar, como responder ao blefe de um oponente, ou como ajustar sua estratégia com base no estilo de jogo do oponente. Isso seria extremamente difícil, senão impossível, de programar explicitamente, mas pode ser aprendido efetivamente através do *selfplay*. (BROWN; SANDHOLM, 2019).

2.3.4. Redes neurais e aprendizado profundo

Redes neurais são uma técnica inspirada no cérebro humano. Elas simulam o funcionamento de neurônios, e a maneira como são ativados, organizado-os em camadas. Cada neurônio recebe inputs, realiza cálculos em sua entrada e produz um output, que é passado para outros neurônios. (ABIODUN et al., 2018). Esta abordagem é geralmente utilizada em problemas mais complexos, que envolvem um *aprendizado profundo* dos dados. O aprendizado supervisionado, não supervisionado e por reforço podem utilizar redes neurais como componente principal ou complementar, mas não são essenciais, já que outros algoritmos e abordagens também podem ser empregados. As redes neurais são apenas uma das ferramentas disponíveis para modelar e resolver problemas complexos de aprendizado de máquina. (ABIODUN et al., 2018).

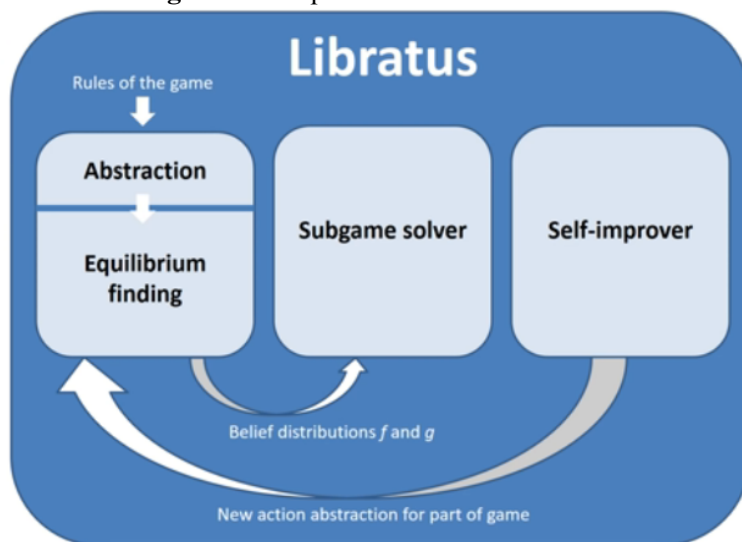
3. TRABALHOS RELACIONADOS

Nesta seção buscamos apresentar uma seleção de trabalhos envolvendo a criação de modelos baseados em aprendizado por reforço e métodos de busca que jogam poker, assim como trabalhos com abordagens diferentes de aprendizado neste contexto. Dos trabalhos escolhidos, três possuem maior relevância e similaridade ao nosso, expostos nos tópicos 3.1, 3.2 e 3.3.

3.1. SAFE AND NESTED SUBGAME SOLVING FOR IMPERFECT-INFORMATION GAMES

Em Brown N. e Sandholm T. (2017), os autores apresentam Libratus, a primeira IA capaz de derrotar com significância estatística jogadores profissionais em partidas de HUNL Texas Hold'em Poker, uma modalidade de Texas Hold'em Poker disputada apenas entre dois oponentes. Para enfrentar o desafio de criar uma IA capaz de jogar poker, um jogo com um número imenso de possibilidades para computar uma estratégia completa, os desenvolvedores criaram três módulos que compõem o Libratus.

Figura 1 - Arquitetura do Libratus



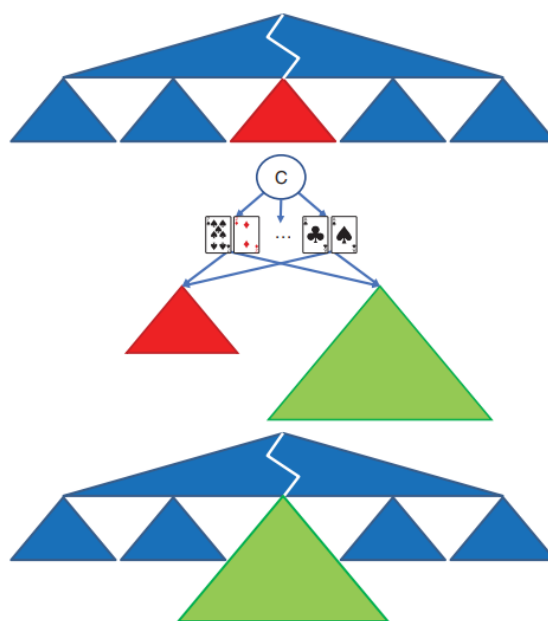
Fonte: vídeo com explicação do autor¹.

O primeiro módulo agrupa jogadas parecidas (como apostar 500 ou 501 fichas), cria uma abstração do jogo, que é uma versão simplificada do jogo original, e que facilita o

¹ Disponível em: <<https://youtu.be/2dX0lwaQRX0>>. Acesso em: 29 jun. 2023.

cálculo de uma estratégia. Em seguida utiliza a mesma para encontrar uma estratégia (também chamada de *blueprint* pelos autores) que segue um equilíbrio de Nash aproximado do poker, por meio do CFR, um algoritmo de *selfplay*. A estratégia resultante é muito detalhada na primeira metade do jogo (*pre-flop* e *flop*) e muito simplificada no fim (*turn* e *river*), mas não existe problema nesta simplificação final, pois além de computarmos este *blueprint* antes do jogo começar, Libratus faz uso da resolução de sub-jogos em tempo real enquanto joga, que o auxilia para encontrar boas jogadas em rodadas finais do jogo, explicada no módulo a seguir.

Figura 2 - Aplicação da resolução de sub-jogos na blueprint original do Libratus



Fonte: (BROWN; SANDHOLM, 2018)².

O segundo módulo é a resolução de sub-jogos. Aqui a IA identifica o estado atual do jogo (sub-jogo), e calcula uma estratégia muito melhor de acordo com a situação atual do jogo, inserindo sua nova estratégia dentro do *blueprint* calculado no primeiro módulo. Aqui os autores apresentam duas técnicas que se diferenciam neste aspecto das antigas, sendo elas *Reach Subgame Solving* e *Nested Subgame Solving*. No caso de *Reach Subgame Solving*, a partir de um sub-jogo específico é levado em consideração as possíveis alternativas de resultado em outros sub-jogos para o oponente, atualizando a estratégia de acordo. Isso dificulta a “exploração” da estratégia por parte dos oponentes, pois as decisões tomadas dependerão menos do sub-jogo atual em que estamos e mais da estratégia geral do jogo. Já

² Identificação de um sub-jogo (em vermelho) dentro do *blueprint* inicial da IA, e aplicação de resolução de sub-jogos para encontrar uma estratégia mais detalhada para o sub-jogo atual (em verde) e inseri-la na *blueprint* original.

Nested subgame solving é uma forma de atualizar a estratégia, de modo que o sub-jogo que atual possua um peso muito maior. Esta abordagem permite respostas mais precisas a uma ampla gama de ações do oponente e pode ser usada para resolver modelos mais refinados à medida que o jogo progride na árvore de decisão. São apresentados experimentos utilizando ambas as técnicas mencionadas anteriormente.

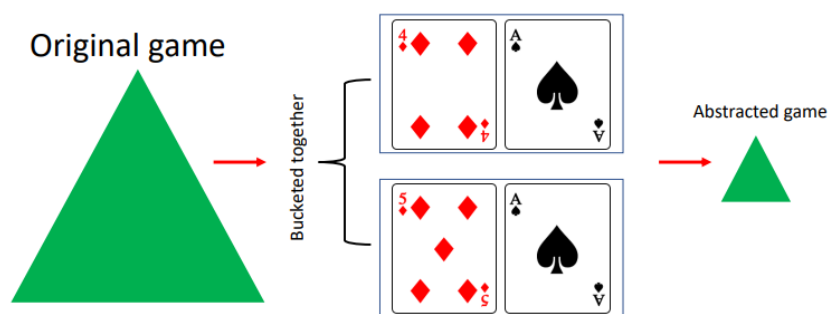
O terceiro módulo é o auto-aperfeiçoador, aonde a IA revê situações que o adversário criou, nas quais a IA possivelmente não possui um *blueprint* adequado, e atualiza este buscando encontrar uma estratégia superior nestas situações, e consequentemente, uma estratégia mais próxima de um equilíbrio de Nash. Estas situações acontecem quando um oponente não aposta valores dentro da abstração, fazendo com que a aposta seja arredondada para um valor próximo do que no *blueprint*, com isso o adversário pode explorar imprecisões no arredondamento da IA. Na prática, o auto-aperfeiçoador funcionava à noite, durante a competição em que Libratus jogou contra humanos profissionais, a IA adicionava algumas ações à abstração com base nas escolhas mais frequentes dos oponentes e na distância dessas ações em relação às existentes na abstração. Uma vez selecionada uma ação, uma estratégia era calculada para ela de forma semelhante à resolução de um sub-jogo. A partir desse momento, se um oponente escolhesse essa ação (ou algo próximo), a estratégia do sub-jogo resolvido era utilizada. Reduzindo ao longo da competição o erro de arredondamento na tradução das ações.

3.2. SUPERHUMAN AI FOR MULTIPLAYER POKER

O artigo "*Superhuman AI for multiplayer poker*" apresenta o Pluribus, uma IA desenvolvida pelos mesmos autores do artigo "*Safe and Nested Subgame Solving for Imperfect-Information Games*" (BROWN; SANDHOLM, 2019), o qual apresenta Libratus. Ambos modelos se baseiam em técnicas que acham um equilíbrio de Nash para dois jogadores, porém, ao contrário de Libratus, Pluribus foi treinado também para jogos com mais de dois jogadores, e mesmo não tendo garantias teóricas de achar um Equilíbrio de Nash com mais de dois jogadores, ainda sim obtém ótimos resultados com múltiplos jogadores.

Pluribus foi treinado jogando contra cinco cópias de si mesmo, usando uma abstração do jogo de poker **figura 1**, que serve para reduzir a complexidade do jogo ao agrupar situações similares, assim como uma versão modificada de *Monte Carlo Counterfactual Regret Minimization* (MCCFR), que se trata de um algoritmo de minimização de arrependimento para encontrar um equilíbrio aproximado.

Figura 3 - Abstração feita por Pluribus

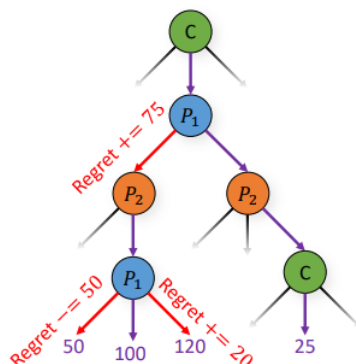


Fonte: slides de apresentação do autor sobre o algoritmo de CFR profundo.

O algoritmo começa selecionando um jogador e atualiza o arrependimento apenas para esse jogador. As ações são escolhidas em proporção à quantidade de arrependimento positivo. Cada ação tem um valor de arrependimento atualizado ao longo do tempo. As ações são inicialmente escolhidas aleatoriamente, pois todas têm o mesmo arrependimento.

À medida que o jogador avança, ele revisa cada decisão que poderia ter tomado e compara o resultado que obteve com o que teria obtido em situações hipotéticas diferentes. Isso permite ao jogador calcular o arrependimento, o qual é a diferença entre o que ele realmente ganhou e o que ele poderia ter ganho. Essa diferença é então adicionada ao valor de arrependimento da ação que não foi tomada.

Para auxiliar o MCCFR, foi incluído um método chamado *pruning*, que reduz a exploração de ações que consistentemente oferecem uma recompensa negativa. Isso permite que o bot se aprofunde mais nas partes mais úteis e interessantes do jogo. Além de ser utilizado no treinamento do modelo, este método também roda em tempo real, durante o jogo, para buscar uma estratégia melhor.

Figura 4 - Regret minimization

Fonte: slides de apresentação do autor sobre o algoritmo de CFR profundo³.

Um aspecto notável deste trabalho é o fato de Pluribus ter sido treinado em apenas oito dias em um servidor com 64 núcleos, com um total de 12,400 CPU *core hours*, e exigiu menos de 512 GB de memória, demonstrando assim que as suas demandas computacionais eram significativamente menores do que as das IAs anteriores. Em seguida, ele foi testado contra cinco jogadores profissionais de poker ao longo de 12 dias, durante os quais foram jogadas 10.000 mãos, onde Pluribus demonstrou-se estatisticamente superior aos demais.

O artigo destaca que o Pluribus não utiliza redes neurais e sua política é modificada exclusivamente por meio de uma variação de CFR, o que difere de muitas abordagens recentes de IA. O comportamento da IA revelou estratégias surpreendentes, desafiando a sabedoria convencional do poker.

3.3. COMBINING DEEP REINFORCEMENT LEARNING AND SEARCH FOR IMPERFECT-INFORMATION GAMES

O artigo Brown N. e Sandholm T. (2020) apresenta uma abordagem inovadora para criar modelos generalistas de IA para jogos de informação imperfeita, como o poker. Os autores apresentam o *Recursive Belief-based Learning* (ReBeL), uma IA que combina aprendizado por reforço e busca. ReBeL contribui para o campo de jogos de IA, demonstrando que é possível criar um único modelo independente do jogo, capaz de jogar qualquer jogo de informação imperfeita de soma zero para dois jogadores, tendo apenas as regras do jogo. Ela toma decisões considerando a distribuição de probabilidade de diferentes crenças que cada jogador pode ter sobre o estado atual do jogo, chamado de estado de crença

³ Disponível em: <<https://icml.cc/media/icml-2019/Slides/4443.pdf>>. Acesso em: 29 jun. 2023.

pública ou *public belief system* (PBS). Em outras palavras, o ReBeL pode avaliar as chances de seu oponente pensar que ele tem, por exemplo, um par de ases.

Os autores demonstram a eficácia do ReBeL aplicando-o ao HUNL Texas Hold'em Poker e a uma variante do xadrez chamada *dark chess*. Em ambos os jogos, o ReBeL atingiu um alto nível de desempenho, derrotando jogadores humanos experientes no poker e alcançando um desempenho sobre-humano no *dark chess*.

Em conclusão, o artigo apresenta um avanço significativo no campo de jogos de IA, demonstrando o potencial da combinação de aprendizado por reforço e busca para jogos de informação imperfeita, além de introduzir uma IA capaz de lidar com uma ampla variedade de jogos e potencialmente contribuir para aplicações do mundo real.

3.4. OPPONENT MODELING AND EXPLOITATION IN POKER USING EVOLVED RECURRENT NEURAL NETWORKS

O artigo "*Opponent modeling and exploitation in poker using evolved recurrent neural networks*", LI X. e MIIKKULAINEN R. (2018), apresentam uma abordagem de poker que se concentra em “explorar” as fraquezas dos oponentes em vez de seguir estratégias de equilíbrio de Nash.

O autor introduz os agentes *Adaptive System of Holdem* (ASHE), construídos com redes neurais recorrentes evoluídas, com a capacidade de modelar e se adaptar aos oponentes. Esses agentes foram eficazes em derrotar outros agentes baseados em equilíbrio de Nash em partidas um-contra-um. O artigo também destaca o uso de técnicas como Estimativa de Utilidade Recursiva e *Pattern Recognition Trees* (PRTs) avançadas, abrindo caminho para novas pesquisas em jogos de informação imperfeita.

3.5. HEADS-UP LIMIT HOLD'EM POKER IS SOLVED

O artigo "*Heads-up limit hold'em poker is solved*" BOWLING M. et al. (2015), apresentam um avanço significativo na área da inteligência artificial e teoria dos jogos. Os autores desenvolveram o algoritmo CFR+, que resolveu efetivamente o jogo de *heads-up limit* Texas Hold'em Poker (HULHE), uma variante desafiadora do poker de informação imperfeita. O CFR+ aproxima um equilíbrio de Nash, onde nenhuma estratégia pode ser melhorada unilateralmente pelos jogadores.

A implementação do algoritmo exigiu um enorme poder computacional e resultou em uma estratégia altamente eficaz contra os oponentes. Os autores acreditam que esse avanço tem implicações amplas para a tomada de decisões em cenários complexos e pode ter aplicações em áreas como segurança e medicina.

3.6. EQUILIBRIUM FINDING FOR LARGE ADVERSARIAL IMPERFECT-INFORMATION GAMES

No artigo Brown N. (2020), descreve um novo algoritmo que utiliza a distância do transportador (DT) para calcular abstrações sensíveis ao potencial em jogos de informação imperfeita, com ênfase no Texas Hold'em Poker. A abordagem leva em consideração como a equidade das informações privadas de um jogador pode mudar ao longo do jogo, levando em conta a revelação de novas cartas públicas. Ao contrário de abordagens tradicionais, que consideram apenas a equidade final no final do jogo, o algoritmo proposto é capaz de distinguir estados diferentes que têm potenciais de mudança de equidade diferentes.

A utilização da distância do transportador (DT) é introduzida como uma métrica para medir a distância entre distribuições de equidade em diferentes estados do jogo. Essa métrica é utilizada para agrupar estados similares durante o processo de abstração. O artigo fornece um exemplo detalhado do método, aplicando-o em um jogo de poker simplificado e, em seguida, no jogo completo do Texas Hold'em Poker. Os resultados mostram uma melhoria estatisticamente significativa em relação a algoritmos de abstração anteriores.

Os autores também discutem os desafios computacionais associados à abordagem proposta, destacando o alto custo computacional para calcular a DT exata. Eles apresentam uma heurística para aproximar a DT, que acelera significativamente o processo de cálculo, fornecendo uma boa aproximação do valor real.

No final, são apresentadas possíveis direções futuras para a pesquisa, incluindo o desenvolvimento de heurísticas mais precisas e/ou rápidas para aproximar a DT, a extensão da abordagem para outras fases do jogo e a aplicação do algoritmo em outros jogos além do poker.

3.7. LEARNING USEFUL FEATURES FOR POKER

No artigo "*Learning Useful Features For Poker*" de Arjun Nagineni, da Universidade do Texas em Austin, (NAGINENI; NOVAK; VAN DE GEIJN, 2018), os autores apresentam

um estudo sobre a utilização de redes neurais para extrair características valiosas de um jogo de poker. O objetivo é usar essas características para “explorar” o comportamento dos oponentes. A pesquisa baseia-se em trabalhos anteriores de Xun Li sobre redes de modelagem de oponentes, com um novo foco na “exploração” de diferentes representações do estado do jogo dentro da rede. Tradicionalmente, as características de entrada são feitas manualmente usando o conhecimento comum do poker. No entanto, essa pesquisa visa permitir que o computador determine quais características são essenciais a partir do estado bruto do jogo de poker, eliminando assim a necessidade de características feitas manualmente. O objetivo do projeto é provar que as redes neurais, que têm se mostrado melhores para extrair características visuais em diversas aplicações, também podem se destacar na representação de jogos de poker.

Esse estudo se concentra apenas na representação do estado do jogo, deixando a “exploração” do jogo real com a nova representação para trabalhos futuros. O modelo desenvolvido foi capaz de representar a maioria do estado do jogo de poker (exceto os tamanhos das apostas) ao comprimi-lo substancialmente por meio da aprendizagem de características, sem perda significativa de informações. Verificou-se que a representação comprimida é semelhante, mas melhor do que as características de entrada feitas manualmente, pois contém informações mais detalhadas. O artigo conclui sugerindo que o modelo possa ser expandido no futuro para incluir os tamanhos das apostas e ser testado em jogos reais de poker.

3.8. AUTO-ENCODER NEURAL NETWORK BASED PREDICTION OF TEXAS POKER OPPONENT’S BEHAVIOR

Neste artigo, Wang S. et al. (2022), propõem o uso de redes neurais *auto-encoder* para prever o comportamento de oponentes no Texas Hold'em Poker, um jogo caracterizado por informações imperfeitas. Ele propõe um método de previsão de comportamento de oponentes com base em redes neurais *auto-encoder*. Utiliza dados históricos do *International Computer Poker Tournament* para modelar o comportamento de jogadores de Texas Hold'em Poker e treinar um modelo de rede neural *auto-encoder* que pode ser aplicado em diferentes estágios do jogo.

A estrutura e os parâmetros do modelo de rede *auto-encoder* são estudados experimentalmente, utilizando o algoritmo de convergência de gradiente conjugado quantizado (SCG) e um classificador softmax. O método demonstra prever com precisão as

informações de comportamento do oponente em pouco tempo, fornecendo uma assistência valiosa para a tomada de decisão do agente de Texas Hold'em Poker. A rede neural *auto-encoder*, com sua capacidade de expressão e compressão, é especialmente vantajosa ao lidar com dados de mão de alta dimensão e dispersos. O modelo de previsão de comportamento do oponente construído pode fornecer rapidamente informações de previsão precisas ao agente, auxiliando-o na tomada da melhor decisão.

O estudo conclui que o modelo proposto possui uma alta taxa de precisão na previsão de comportamento em cada estágio do Texas Hold'em Poker, superando outros métodos, e seu tempo de execução é menor, tornando-se uma ferramenta altamente eficiente para a tomada de decisões no jogo.

3.9. CONSIDERAÇÕES

Para desenvolvermos nosso trabalho, analisamos diversos sistemas avançados, incluindo Libratus, Pluribus e ReBel. Pluribus, notável por seu desempenho eficiente com recursos computacionais limitados, demonstra a viabilidade de desenvolver um sistema de jogo de poker sem grandes investimentos financeiros. ReBel, por outro lado, é um sistema notável que emprega aprendizado profundo em seu algoritmo, representando um avanço significativo na aplicação de inteligência artificial em jogos complexos. Além disso, Libratus é um marco importante, servindo como um ponto de referência para avaliar as melhorias do Pluribus ao longo do tempo. Esta comparação é crucial para compreender o progresso e os avanços do Pluribus em relação ao seu antecessor.

Quadro 2 - Comparação entre IA's dos trabalhos 3.1, 3.2 e 3.3.

	Libratus (2017)	Pluribus (2019)	ReBel (2020)
Número de jogadores (considerando a IA)	2	6	2
Abstração de informações	Sim	Sim	Não
Taxa de vitória contra jogadores profissionais mili big blinds por mão (mbb)	147 (± 39) em 120.000 mãos	48 (± 25) em 10.000 mãos	165 (± 69) em 7,500 mãos
Custo de treinamento	25.000.000 horas de processamento de CPU (3.000.000 para testes), custo de \$100.000 no supercomputador Bridge.	12.400 horas de processamento de CPU, custo de \$144 num serviço de Cloud conhecido como AWS.	?
Usa aprendizado profundo	Não	Não	Sim
Garantia de convergência para equilíbrio de Nash	Sim	Sim (2 jogadores) Não (3+ jogadores)	Sim
Inovações ou Características Únicas	Pioneiro em Abstração de Jogo para limitar a complexidade do jogo.	Primeira IA a vencer jogadores profissionais de poker em mesas de 6 jogadores.	Traz uma abordagem mais genérica para resolução de jogos imperfeitos, porém mantendo ótima performance.
Depth limited search	Não	Sim	Sim

Fonte: Elaboração própria a partir de artigos dos autores. (BROWN; SANDHOLM, 2017) (BROWN; et al., 2020) (BROWN; SANDHOLM, 2019) (BROWN; SANDHOLM, 2018).

4. DESENVOLVIMENTO

Nossa implementação foi fortemente influenciada pelo trabalho realizado em Pluribus, um modelo que alcançou resultados notáveis utilizando recursos computacionais modestos. Adaptando as estratégias eficazes do Pluribus para um contexto de jogo de dois jogadores, optamos por incorporar e ajustar as principais ideias desse modelo. O cerne da nossa abordagem é a aplicação de técnicas de convergência de equilíbrio de Nash, empregando o método MCCFR no contexto do jogo de HUNL Leduc Poker. Nosso objetivo é otimizar uma estratégia de jogo que maximize o retorno em fichas por partida.

A implementação do MCCFR desempenhou um papel crucial ao permitir que nossa solução aproximada fosse eficiente. Isso possibilitou que o agente aprendesse estratégias robustas sem a necessidade de explorar exaustivamente todas as possibilidades de jogadas. Para avaliar o sucesso de nossa implementação, utilizamos métricas específicas, detalhadas no capítulo 5, como o EV e o tempo de treinamento. Essas métricas fornecem *insights* valiosos sobre a eficiência da aprendizagem e a eficácia das estratégias desenvolvidas pelo agente.

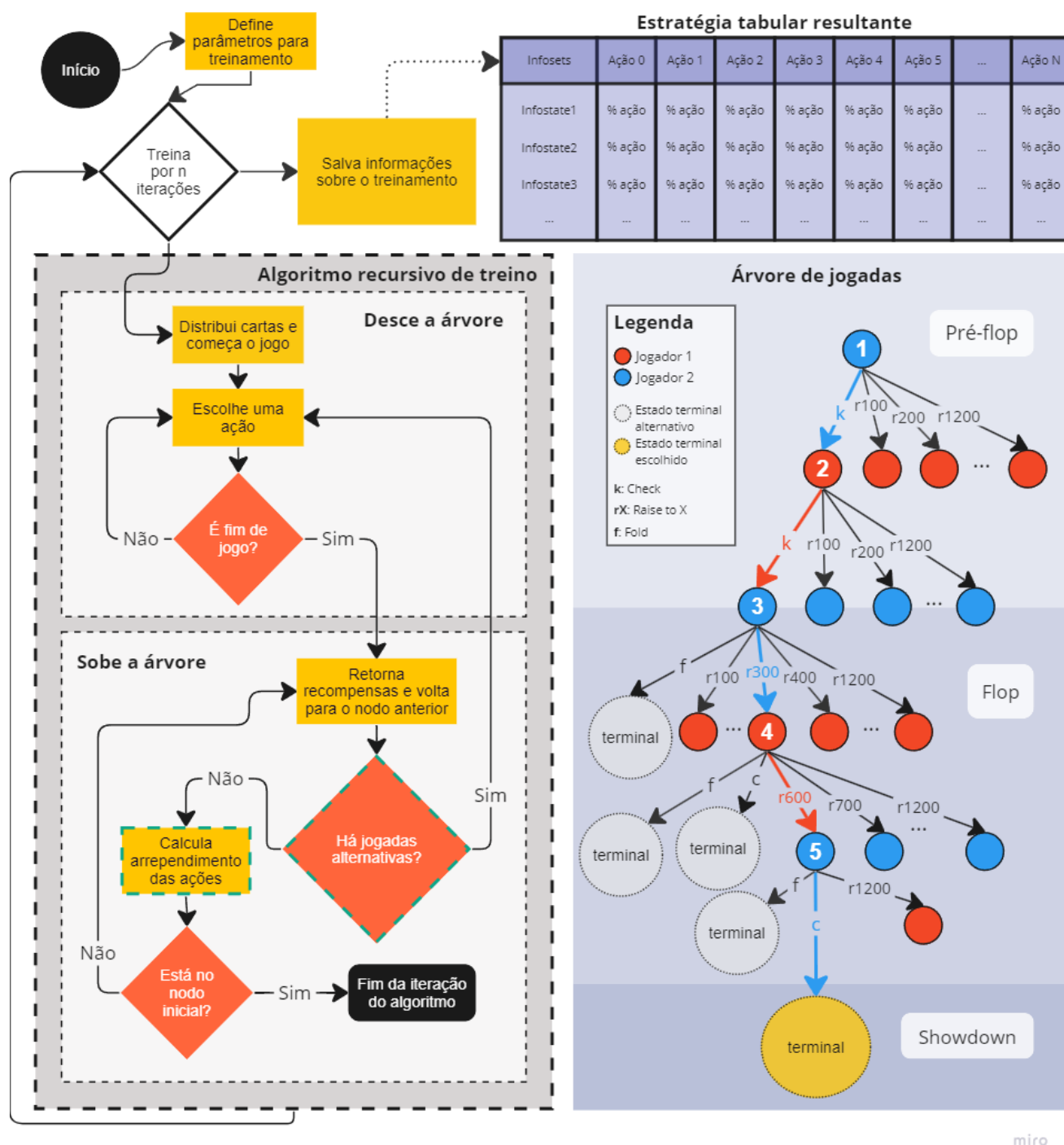
Uma característica distintiva de nossa abordagem é o desenvolvimento incremental do algoritmo MCCFR. Começamos com conceitos básicos, como o *Regret Matching*, e avançamos gradualmente para a implementação completa do MCCFR, sem recorrer a frameworks existentes. Essa abordagem detalhada nos proporcionou uma compreensão profunda do funcionamento interno do algoritmo e a capacidade de ajustá-lo de forma otimizada para o nosso ambiente específico.

4.1. ARQUITETURA E MODELAGEM

Em termos gerais, nossa arquitetura final aplica o algoritmo MCCFR, iterando sobre a árvore de jogadas para atualizar os valores dos arrependimentos e das estratégias de cada nodo. No entanto, realizar este processo apenas uma vez inevitavelmente resultaria em uma estratégia final subótima. Portanto, iteramos esse processo múltiplas vezes, permitindo que a estratégia evolua gradualmente ao longo das iterações. A seguir explicaremos outras etapas de nossa implementação.

A **figura 5** apresenta uma visão geral do funcionamento do algoritmo de treinamento, o formato da tabela de estratégias resultante pós-treinamento e também a árvore de jogadas de uma partida de HUNL Leduc.

Figura 5 - Arquitetura em alto nível; Estratégia tabular; Árvore de jogadas



Fonte: autoria própria

4.1.1. Parâmetros de entrada

Antes do início da execução do algoritmo, a customização de diversos parâmetros são permitidas, incluindo:

- **Número de iterações:** Define quantas vezes o algoritmo irá fazer o processo de jogo e atualização de valores, começando no nodo inicial da árvore, indo até o fim da árvore diversas vezes, e voltando para o nodo inicial.

- **Algoritmo:** Permitimos o uso tanto do CFR quanto do MCCFR. As diferenças na implementação são relativamente pequenas, mas possuem uma grande diferença na hora de percorrer a árvore e atualizar os valores dos nodos.

- **Cartas:** Para jogar HUNL Leduc utilizamos apenas 6 cartas, um par de ases, um par de reis, e um par de damas. Porém, qualquer combinação de cartas poderia ser utilizada, inclusive todas as cartas de um baralho comum.

- **Número de fichas inicial:** Como o teste central de nossa implementação foi contra o DeepStack, que joga com 1200 fichas, utilizamos este número para a criação de nossos modelos, porém qualquer outro valor era possível também.

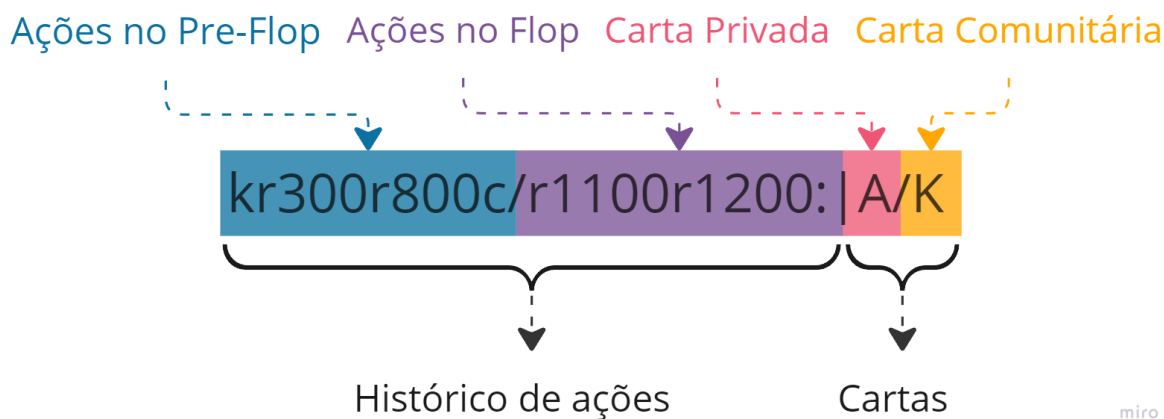
- **Seed:** O processo de execução do algoritmo envolve o uso de números aleatórios inúmeras vezes, como na hora de escolher uma ação para um usuário. Um número fixo de *seed* nos permite executar nosso algoritmo múltiplas vezes e obter sempre o mesmo resultado para o mesmo conjunto de parâmetros, o que é essencial para reprodutibilidade dos resultados, e para realizar modificações e otimizações no algoritmo, mantendo confiança de que a lógica final permanecia inalterada.

- **Fase de exploração:** Adicionamos uma lógica em nossa implementação que fazia com que para as primeiras n iterações (definidas pelo número do parâmetro), a árvore de jogadas fosse percorrida atualizando os valores de arrependimentos, mas sem atualizar os valores de estratégias. A motivação disto é evitar que comecemos dando preferências para estratégias não ótimas, porém ao realizar testes notamos resultados inferiores ao utilizar a fase de exploração, então decidimos por deixá-lo sempre como 0.

- **Estratégia fixa:** Ao treinar um modelo novo utilizando nossa implementação, fazemos com que ele seja tanto o jogador 1 quanto o jogador 2, em todas as iterações. Desta forma ele consegue identificar fraquezas em sua estratégia e corrigi-las. Porém, outra possibilidade que adicionamos é a de uma estratégia nova poder ser treinada jogando contra outra que treinamos anteriormente. Desta forma conseguimos ver quão “exploitáveis” são nossas estratégias treinadas.

Após a definição dos parâmetros e o início da execução, cada iteração começa com o embaralhamento das cartas para simular uma situação de jogo real. Em seguida, o algoritmo MCCFR/CFR é iniciado, percorrendo a árvore de jogadas a partir do nodo raiz. Ao alcançar um nodo específico, resgatamos o *infostate* da situação atual, composto pelo histórico de jogadas, o qual é o identificador do nodo, concatenado com nossa carta privada, e com a carta pública, caso houver, como visto na **figura 6**. Identificamos as ações possíveis naquele momento e selecionamos uma aleatoriamente, conforme a estratégia atual para esse nodo. É importante destacar que a tabela de estratégias vai se expandindo à medida que a exploração dos jogos avança através das iterações do treinamento. À medida que mais jogadas são exploradas e mais nodos são visitados, a tabela de estratégias se enriquece progressivamente, refletindo a evolução e refinamento das estratégias em resposta às diferentes situações de jogo

Figura 6 - Descrição de um infostate

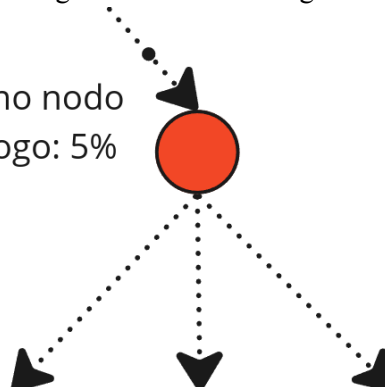


Fonte: autoria própria

A estratégia de um nodo é determinada pela soma dos arrependimentos positivos associados a ele, conforme demonstrado na **figura 7**. A cada recálculo da estratégia atual, a soma de estratégias é atualizada, convergindo ao longo das iterações em direção ao equilíbrio de Nash. Após a iteração final, normalizamos os valores da soma de estratégias de cada nodo para garantir que estejam compreendidos entre 0 e 1. Esse processo assegura que a soma de todas essas estratégias para um mesmo nodo seja igual a 1. Então salvamos um dicionário com chaves sendo *infostates*, e com valores sendo a soma de estratégias final, devidamente normalizada. Esses dados são então salvos em um arquivo *Pickle*, representando a estratégia final do nosso modelo treinado.

Figura 7 - Cálculo de estratégia e soma de estratégias

Chance de chegar no nodo
desde o início do jogo: 5%

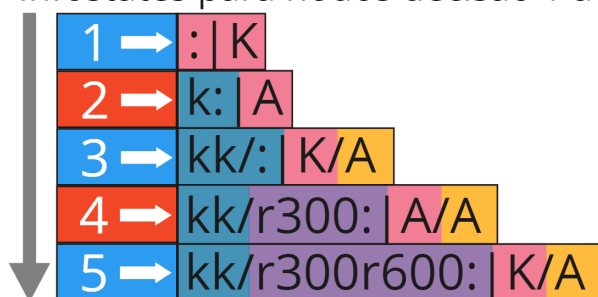


Soma de arrependimentos	525	-100	350
Estratégia atual calculada	$525 / (525 + 350) = 60\%$	0%	$350 / (525 + 350) = 40\%$
Soma de estratégias	Soma de estratégias anterior + $(0.6 * 0.05)$	Soma de estratégias anterior + $(0 * 0.05)$	Soma de estratégias anterior + $(0.4 * 0.05)$

Fonte: autoria própria

Após escolher a ação, movemos para um novo nodo na árvore, que representa o histórico com a adição da ação escolhida, e trocamos o jogador atual. Se o nodo não for terminal, o processo de descida na árvore continua normalmente, com novas ações selecionadas para cada jogador, e com a verificação da condição do nodo (se é terminal ou não). No caso de o novo nodo ser terminal, calculamos a recompensa e retornamos ao nodo anterior.

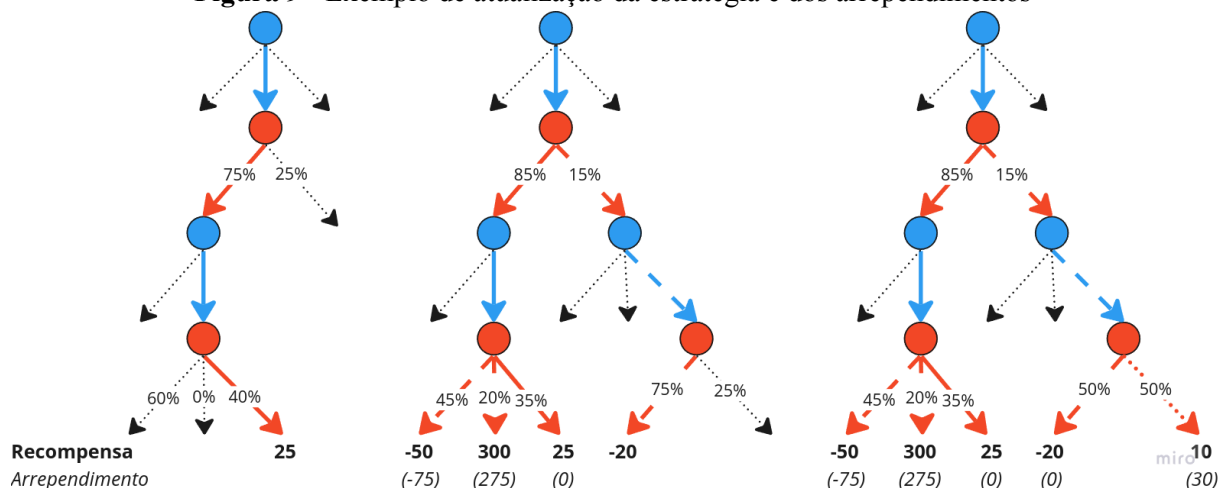
Infostates para nodos decisão 1 a 5

**Figura 8** - Representação dos *infostates* para cada nodo decisão de ambos os pontos de vista, conforme a partida da **figura 5**.

Para calcular os arrependimentos dos nodos vizinhos, é necessário realizar jogadas alternativas em cada um deles, descendo na árvore até atingir um estado terminal e retornando o valor da recompensa adquirida, conforme mostrado na **figura 9**. Uma vez que retornamos ao nodo raiz e todos os seus nodos imediatos tiverem passado por jogadas alternativas,

calculamos os arrependimentos, atualizamos as estratégias e concluímos a iteração do algoritmo.

Figura 9 - Exemplo de atualização da estratégia e dos arrependimentos



Fonte: autoria própria

4.1.2. Métricas de avaliação

Como métrica de avaliação optamos por utilizar a notação mbb, que é um padrão para benchmarks de IAs no contexto de poker. Esta notação se refere a "milésimos de *big blinds*" por mão, que é uma forma de normalizar ganhos ou perdas independentemente dos níveis de *stakes*. Essa métrica é particularmente útil para comparar a performance de diferentes algoritmos de inteligência artificial em termos de eficácia e estratégia, permitindo avaliações consistentes mesmo quando o tamanho do *big blind* varia. Além disso, a mbb permite que pesquisadores e desenvolvedores meçam o desempenho de suas IAs de poker em relação a um "jogador neutro" (0 mbb), que nem ganha, nem perde dinheiro no longo prazo. Assim, uma IA com uma taxa de 5 mbb estaria ganhando 5 milésimos de um *big blind* por mão jogada, indicando uma habilidade superior em relação aos adversários enfrentados. Essa abordagem quantitativa é essencial para identificar avanços significativos na área de inteligência artificial aplicada ao jogo de poker.

4.1.3. Otimizações

Houveram algumas otimizações que trouxeram melhorias ao código, algumas com resultados modestos, mas outras com melhoras consideráveis. A seguir apontaremos as principais otimizações realizadas, em ordem de impacto.

4.1.3.1. *Uso do formato Pickle*

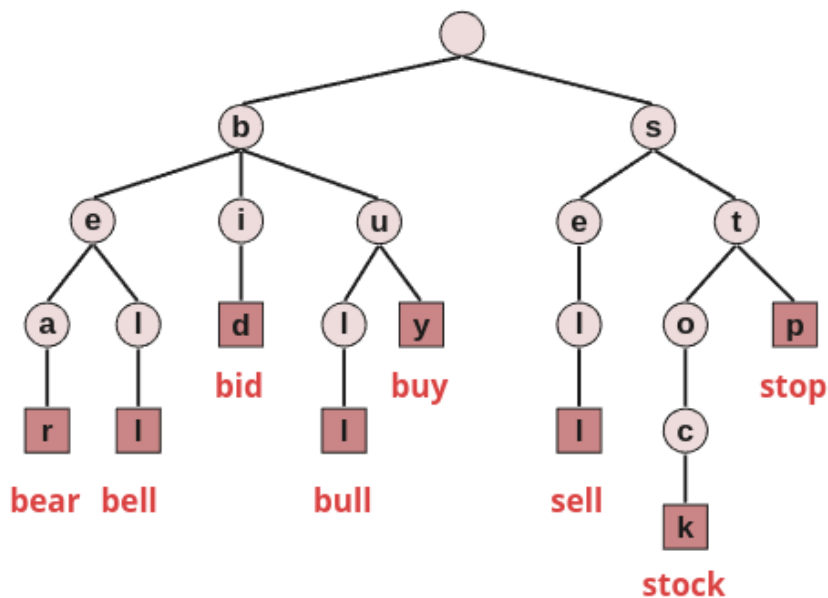
Inicialmente, os modelos finais gerados após os treinamentos eram salvos em arquivos JSON. No entanto, ao realizar a transição para o formato *Pickle*, uma melhora no uso do espaço de armazenamento de cerca de 20% foi observada.

Além disso, ao salvar o modelo no formato *Pickle*, notamos uma melhoria no tempo de salvamento. No entanto, como o modelo é salvo apenas uma vez ao final do treinamento, essa melhoria tem um impacto relativamente pequeno.

4.1.3.2. *Uso da estrutura de dados Trie*

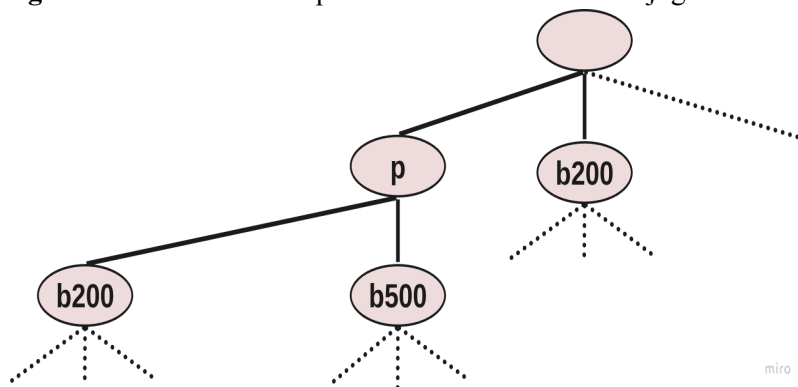
A Trie é uma estrutura de dados em forma de árvore usada para armazenar e buscar sequências de caracteres de maneira eficiente, como pode se ver na **figura 10**. Em nosso código com implementação da estrutura Trie, os nodos são identificados individualmente pela sequência de ações realizadas. Por exemplo:

- **“” (vazio)**: Nodo raiz, antecessor de todos, ele é vazio, pois não há jogadas anteriores a ele.
- **“p”**: Nodo que representa o histórico após o primeiro jogador realizar um passe. Este nodo é uma ramificação do nodo raiz.
- **“pb200”**: Nodo que representa o histórico onde o primeiro jogador realizou um passe e o segundo jogador fez uma aposta de 200 fichas. Este nodo é uma ramificação do nodo “p”.
- **“pb500”**: Nodo que representa o histórico onde o primeiro jogador realizou um passe e o segundo jogador fez uma aposta de 500 fichas. Este nodo também é uma ramificação do nodo “p”.

Figura 10 - Estrutura Trie para armazenar palavras

Fonte: autoria própria

Vemos que a Trie é especialmente útil em nosso caso para representar nodos com um prefixo idêntico em uma árvore. Desta forma, se estivermos no nodo “p”, podemos acessar o nodo “pb200” somente com a ação final “b200”, não precisando desta forma calcular o histórico inteiro para aquele nodo, somente a última ação.

Figura 11 - Estrutura Trie para armazenar histórico de jogadas

Fonte: autoria própria

Devido à organização hierárquica e à eliminação de operações para o nodo atual, o treinamento de MCCFR apresenta uma melhora no tempo de execução de aproximadamente 10% em comparação com o uso de um dicionário convencional.

4.1.3.3. *Uso de ferramentas de profiling: line_profiler*

O *line_profiler* é uma ferramenta de *profiling* do Python que identifica partes do código que causam gargalos no desempenho do programa. Ao analisar a execução linha por linha, fornece informações detalhadas sobre o tempo de execução de cada linha de código. Utilizando o *line_profiler*, descobrimos que 51% do tempo era gasto no treinamento dos modelos, e outros 47% eram consumidos pelo cálculo de uma métrica simples ao final de cada iteração. Identificamos que uma soma de listas usada para o cálculo da métrica era a responsável pelo gargalo. Para solucionar o problema, introduzimos uma variável global para a métrica, atualizada durante a execução do algoritmo. Essa alteração simples reduziu o tempo de execução pela metade.

4.1.3.4. *Uso da implementação PyPy de Python*

PyPy é uma implementação alternativa ao interpretador Python que utiliza a compilação JIT (*Just-in-Time*) para melhorar o desempenho. Através desta compilação, o PyPy consegue otimizar e acelerar a execução de código Python. Ao migrarmos nosso código para o PyPy, observamos um aumento muito significativo, de quase 5 vezes na velocidade de execução, sem precisar realizar nenhuma alteração no código.

4.2. INTERFACE WEB

Criamos uma interface web interativa, tanto o *front-end* quanto o *back-end*, que facilita a interação direta entre um jogador humano e nossa inteligência artificial. Esta plataforma visa oferecer uma experiência mais aprofundada e abrangente tanto do jogo HUNL Leduc quanto das capacidades e estratégias da nossa IA. Também apresenta um painel de estatísticas, para que o usuário possa acompanhar o histórico de partidas jogadas contra a IA, e ter uma ideia de como é o desempenho de ambos. Mais informações de utilização estão disponíveis no link do repositório encontrado no apêndice.

5. EXPERIMENTOS

Neste capítulo, apresentamos a fundamentação prática de nosso trabalho, estruturada em uma sequência de provas de conceito que gradualmente constroem a nossa implementação final do algoritmo de MCCFR. Estruturamos o capítulo em três partes principais, refletindo as etapas incrementais do nosso processo experimental, das quais as duas primeiras foram inspiradas nos exemplos dados por Neller T. e Lanctot M., 2013. A primeira parte aborda o jogo Pedra, Papel e Tesoura, introduzindo o algoritmo de *Regret Matching*. A segunda parte avança para o jogo Kuhn Poker, onde o CFR e o MCCFR são explorados e adaptados. Por fim, a terceira parte aborda o jogo HUNL Leduc Poker, aplicando o MCCFR em um contexto de informações e ações mais complexo e demonstrando a aplicabilidade do nosso algoritmo. Para cada jogo, detalhamos suas regras, nossa abordagem e os resultados alcançados, oferecendo uma visão clara do desenvolvimento e eficácia do nosso modelo.

5.1. PEDRA, PAPEL E TESOURA

O jogo de Pedra, Papel e Tesoura é uma ótima opção para validar o algoritmo de *Regret Matching* devido à sua natureza de soma zero com informação perfeita, onde as regras são claras e cada jogada tem uma resposta ideal claramente definida. Essas características o tornam um ambiente adequado para testar a eficácia do algoritmo antes de aplicá-lo a jogos mais complexos com informação imperfeita.

Para validar nossa implementação, realizamos três experimentos com diferentes cenários e estratégias no jogo de Pedra, Papel e Tesoura. Como um *Regret Matching* funcional é crucial para desenvolver qualquer variação do CFR, precisamos confirmar seu funcionamento adequado.

5.1.1. Regras do jogo

O jogo Pedra, Papel e Tesoura é amplamente conhecido e jogado, e possui regras simples. Os jogadores escolhem uma das três opções: pedra, papel ou tesoura, ao mesmo tempo. As regras são:

- Pedra vence a tesoura (pedra esmaga a tesoura).
- Tesoura vence o papel (tesoura corta o papel).
- Papel vence a pedra (papel embrulha a pedra).

Porém, caso ambos os jogadores escolham a mesma opção, acontece um empate. Em nossa implementação, vitórias, empates e derrotas são representadas por um ganho para o jogador de +1, 0 e -1, respectivamente.

5.1.2. Experimentos e resultados

A seguir são apresentados experimentos que demonstram a capacidade da nossa implementação de *Regret Matching* em se adaptar a diferentes estratégias e configurações do jogo de Pedra, Papel e Tesoura.

5.1.2.1. Primeiro experimento:

Treinamos nosso algoritmo via *selfplay* com 100.000 iterações jogando a versão descrita anteriormente de Pedra, Papel e Tesoura, esperando que o mesmo atinja uma estratégia com chances iguais de jogar qualquer opção, já que se tiver uma tendência a jogar uma opção mais que as outras, isto poderia facilmente ser “explorado” por um adversário que se adapte a sua estratégia (BROWN; SANDHOLM, 2019). Ao rodarmos nosso algoritmo na situação apresentada, a estratégia final de equilíbrio converge para uma distribuição muito próxima do ótimo teórico.

Métrica de Avaliação: Convergência da estratégia para uma distribuição igual de 33.33% para cada opção (Pedra, Papel e Tesoura).

Tabela 1 – Resultado do primeiro experimento de Pedra, Papel e Tesoura.

Estratégia	Pedra	Papel	Tesoura
Equilíbrio de Nash teórico	33,33%	33,33%	33,33%
Nosso resultado:	33,54%	33,21%	33,25%

Fonte - Autoria própria

5.1.2.2. Segundo experimento:

Aqui, jogamos o jogo padrão de Pedra, Papel e Tesoura com 100.000 iterações, mas fixamos a estratégia de nosso oponente para uma estratégia fixa inclinada a escolher Tesoura um pouco mais do que outras opções.

Nesse contexto, nossa estratégia deve se adaptar para “explorar” o adversário e jogar Pedra com probabilidade de 100% (BROWN; SANDHOLM, 2019). É importante notar que não estamos em risco de ser “explorados”, uma vez que nosso oponente possui uma estratégia fixa e não se adapta à nossa estratégia, que claramente poderia ser “explorada” por uma estratégia que tenha preferência por jogar papel.

Métrica de Avaliação: Convergência da estratégia para “explorar” a estratégia fixa do oponente, jogando Pedra com 100% de probabilidade.

Tabela 2 – Resultado do segundo experimento de Pedra, Papel e Tesoura.

Estratégia	Pedra	Papel	Tesoura
Equilíbrio de Nash teórico	100%	0%	0%
Nosso resultado:	99,82%	0,01%	0,17%

Fonte - Autoria própria

5.1.2.3. Terceiro experimento:

No terceiro cenário, ambos jogadores possuem estratégias dinâmicas, como no primeiro cenário, mas as regras do jogo de Pedra, Papel e Tesoura são levemente modificadas. Agora, se qualquer jogador escolher Tesoura, o vencedor ganhará 2 pontos de recompensa e o perdedor perderá 2, ao invés de 1. Como resultado, a estratégia ótima de ambos jogadores converge para jogar Pedra e Papel com probabilidade de 40% cada e Tesoura com 20%, o que representa um equilíbrio de Nash para esse cenário, como demonstrado em Brown N. (2020).

Métrica de Avaliação: Convergência da estratégia para jogar Pedra e Papel com 40% de probabilidade e Tesoura com 20%, resultando em um equilíbrio de Nash.

Tabela 3 – Resultado do terceiro experimento de Pedra, Papel e Tesoura.

Estratégia	Pedra	Papel	Tesoura
Equilíbrio de Nash teórico	40%	40%	20%
Nosso resultado:	39,89%	39,35%	20,75%

Fonte - Autoria própria

Nos experimentos com Pedra, Papel e Tesoura, o objetivo era desenvolver uma estratégia para nossa IA que evitasse perdas em expectativa. O primeiro experimento mostrou a IA aprendendo a estratégia ideal, o segundo confirmou sua habilidade de se adaptar contra um oponente com estratégia fixa, e o terceiro testou sua resposta a mudanças nas regras. Em todos, a IA demonstrou progresso significativo em direção ao objetivo de alcançar uma estratégia que não resulte em perdas esperadas.

5.2. KUHN POKER

Um próximo passo estratégico, além do simples jogo de Pedra, Papel e Tesoura, é implementar o jogo de Kuhn Poker. Apresentado por Kuhn H., 1950, este jogo é amplamente utilizado no contexto acadêmico. Nesse novo contexto, o algoritmo de *Regret Matching* em si não é mais suficiente devido à natureza extensiva do Kuhn Poker, onde uma partida é composta por turnos. Por isso, implementamos os algoritmos CFR e MCCFR, e os avaliamos por meio de experimentos detalhados que serão discutidos a seguir. Além disso, outro aspecto importante da avaliação do CFR com o Kuhn Poker é que o jogo possui um padrão de equilíbrio de Nash conhecido, o qual será explicado com detalhes também a seguir.

5.2.1. Regras do jogo

O Kuhn Poker é uma versão extremamente simplificada do poker. O jogo é jogado com um baralho de três cartas: uma dama (Q), um rei (K) e um ás (A). Cada jogador recebe uma carta, que é privada, e há uma rodada de apostas. Durante a rodada de apostas, os jogadores podem escolher entre passar ou apostar uma unidade (KUHN, 1950). Sendo todas as possibilidades de jogos as listadas a seguir:

- Se ambos os jogadores passarem, o jogador com a carta mais alta ganha 1 ponto, e seu oponente perde 1 ponto.
- Se um jogador apostar e o outro passar, o jogador que apostou ganha 1 ponto, e seu oponente perde 1 ponto.
- Se ambos os jogadores apostarem, o jogador com a carta mais alta ganha 2 pontos, e seu oponente perde 2 pontos.

5.2.2. Experimentos e resultados

Para testar nossa implementação de CFR no Kuhn Poker, pode-se verificar se está ocorrendo a convergência para um equilíbrio de Nash, o que pode ser feito ao verificar se os seguintes requisitos (KUHN, 1950) são atendidos pela estratégia final:

O primeiro jogador, em um equilíbrio de Nash, deve:

- Perder a uma taxa de $-1/18$ pontos por rodada, e consequentemente, o segundo jogador deve esperar ganhar a uma taxa de $+1/18$. Demonstrando uma desvantagem posicional do jogador que começa.
- Escolher livremente uma probabilidade $\alpha \in [0, 1/3]$ com a qual apostará ao ter uma dama. Caso ele passe e o segundo jogador aposte, ele deve sempre desistir
- Ao ter um ás, deve apostar com a probabilidade 3α . Caso ele passe e o segundo jogador aposte, ele deve sempre pagar
- Sempre passar quando tiver um rei, e caso o segundo jogador apostar após esse passe, ele deve pagar com a probabilidade $\alpha + 1/3$

O segundo jogador tem uma única estratégia de equilíbrio:

- Sempre apostar ou pagar ao ter um ás
- Ao ter um rei, passar se possível, caso contrário pagar com a probabilidade de $1/3$
- Ao ter uma dama, nunca pagar e apostar com a probabilidade de $1/3$.

A seguir, os requisitos foram comparados para uma estratégia em equilíbrio de Nash, com o resultado da implementação do CFR no Kuhn Poker, após 100.000 iterações. Na tabela 1, a primeira letra denota a mão do jogador atual, e as letras subsequentes representam a sequência de ações passadas dos jogadores que levaram o jogo até o estado atual. Por exemplo:

- **Q**: O jogador possui uma dama e não jogadas antes da atual.
- **Kb**: O jogador possui um rei e o primeiro jogador realizou uma aposta (*bet* em inglês, por isso o “b”).
- **Apb**: O jogador possui um ás, começou-se passando, e em seguida o segundo jogador fez uma aposta.

Tabela 4 – Resultado do treinamento via CFR para Kuhn Poker.

Estado do jogo	Estratégia em equilíbrio de Nash		Nossa estratégia (CFR)	
	Passar	Apostar	Passar	Apostar
Q	$1 - \alpha$	α	73,67%	26,32%
Qb	100%	0	99,99%	0,00%
Qp	66,67%	33,33%	65,81%	34,18%
Qpb	100%	0%	99,99%	0,00%
K	100%	0%	99,97%	0,02%
Kb	66,67%	33,33%	66,10%	33,89%
Kp	100%	0%	99,98%	0,01%
Kpb	$2/3 - \alpha$	$1/3 + \alpha$	40,10%	59,90%
A	$1 - 3\alpha$	3α	20,16%	79,83%
Ab	0%	100%	0,00%	99,99%
Ap	0%	100%	0,00%	99,99%
Apb	0%	100%	0,00%	99,99%

Fonte - Autoria própria

Na tabela 2, a comparação do valor médio de jogo para o primeiro jogador (VMPJ) em uma estratégia em equilíbrio de Nash foi feita com o VMPJ de nossa estratégia encontrada por meio do CFR:

Tabela 5 – Comparativo do equilíbrio de Nash teórico vs CFR

	Estratégia em equilíbrio de Nash	Nossa estratégia (CFR)
VMPJ	-0,05	-0,053

Fonte - Autoria própria

Observa-se que nossa estratégia se aproxima muito de um equilíbrio de Nash para $\alpha = 0,2632$, o que indica fortemente que a implementação CFR está funcionando. Além disso, o VMPJ também ficou consideravelmente próximo do valor esperado em um equilíbrio de Nash.

Após isto, foi realizado o mesmo experimento para validar o algoritmo de MCCFR, por meio de 100.000 iterações. O resultado pode ser visualizado nas tabelas comparativas a seguir:

Tabela 6 – Resultado do treinamento via MCCFR para Kuhn Poker.

Estado do jogo	Estratégia em equilíbrio de Nash		Nossa estratégia (MCCFR)	
	Passar	Apostar	Passar	Apostar
Q	$1 - \alpha$	α	98,16%	1,84%
Qb	100%	0	99,99%	0.00%
Qp	66,67%	33,33%	64,62%	35,37%
Qpb	100%	0%	99,99%	0,01%
K	100%	0%	99,99%	0,01%
Kb	66,67%	33,33%	64,66%	35,33%
Kp	100%	0%	99,99%	0,01%
Kpb	$2/3 - \alpha$	$1/3 + \alpha$	64,16%	35,83%
A	$1 - 3\alpha$	3α	94,96%	5,03%
Ab	0%	100%	0,01%	99,99%
Ap	0%	100%	0,01%	99,99%
Apb	0%	100%	0,01%	99,99%

Fonte - Autoria própria

E a seguir, a comparação do valor médio de jogo para o primeiro jogador (VMPJ) em uma estratégia em equilíbrio de Nash, com o VMPJ da estratégia encontrada por meio de MCCFR:

Tabela 7 – Comparativo do equilíbrio de Nash teórico vs MCCFR

	Estratégia em equilíbrio de Nash	Nossa estratégia (MCCFR)
VMPJ	-0,05	-0,056

Fonte - Autoria própria

Assim como a estratégia resultante do algoritmo CFR, a estratégia gerada pelo MCCFR se aproximou consideravelmente de um equilíbrio de Nash para $\alpha = 0,0184$, e o VMPJ também se aproximou do valor ótimo em uma estratégia de equilíbrio de Nash.

A proximidade das estratégias da nossa IA ao equilíbrio de Nash em Kuhn Poker indica eficácia na tomada de decisões estratégicas. No entanto, isso não garante vitórias constantes, pois o equilíbrio de Nash em jogos como o poker implica em evitar perdas a longo prazo, não em vencer todas as partidas. Assim, embora a IA esteja otimizada para jogar competitivamente, vitórias em todas as situações não são garantidas.

5.3. HUNL LEDUC POKER

Concluindo nossa série progressiva de experimentos, agora com o jogo de HUNL Leduc Poker. A versão básica foi concebida por Southey et al. (2012), porém a versão que implementamos representa um salto significativo em complexidade devido ao elemento de *No-Limit*, que permite apostas de qualquer tamanho, aumentando exponencialmente o espaço de decisões possíveis.

O HUNL Leduc Poker, por ser mais simples que o poker completo, porém mais complexo que o Kuhn Poker, é utilizado em pesquisas para equilibrar complexidade e acessibilidade. Ele desafia os algoritmos a administrar um espaço de ação e informação muito maiores, tornando o MCCFR essencial para o nosso estudo. O MCCFR é adaptado para navegar pelo universo extenso do *No-limit*, através de métodos de amostragem que tornam viável o cálculo de estratégias ótimas sem explorar cada possibilidade individualmente.

Para testar a robustez do nosso modelo MCCFR, conduzimos simulações contra uma versão simplificada da DeepStack AI (MORAVČÍK et al., 2017), uma inteligência artificial avançada projetada para o poker. Esse oponente de IA, que incorpora estratégias de nível profissional, serve como um parâmetro de comparação para avaliar a nossa abordagem.

A escolha do HUNL Leduc Poker para nossos experimentos não é apenas prática, mas estratégica. Apesar de sua escala reduzida em relação ao poker tradicional, ele retém complexidades essenciais valiosas para testar teorias de jogos e aprimorar algoritmos. A eficácia do MCCFR neste contexto sugere potenciais aplicações em jogos mais complexos e situações do mundo real onde decisões e informações imperfeitas são a norma.

5.3.1. Regras do jogo

As regras do Leduc Poker variam dependendo do estudo em questão (MÜNCH, 2017; MORAVČÍK et al., 2017; SOUTHEY et al., 2012). A versão que implementamos está conforme a versão utilizada pelo DeepStack, para fins de testagem e validação.

Baralho e Blinds:

O jogo é jogado com um baralho reduzido de seis cartas, consistindo em dois exemplares de cada uma das três classificações diferentes: damas (Q), reis (K) e ases (A). Dois jogadores competem um contra o outro (*heads up*). Cada jogador começa com uma pilha igual de fichas, especificamente 1200 fichas. Antes do início da mão, cada jogador posta um *blind* obrigatório de 100 fichas.

Distribuição das Cartas e Rodadas de Apostas:

Cada jogador recebe uma carta privada. Existem duas rodadas de apostas no jogo: uma após a distribuição das cartas privadas e outra após o *flop*. Após o *flop*, uma única carta comunitária é colocada na mesa.

Ações Possíveis:

- Passar (*check*): Se não houver aposta, um jogador pode passar sem apostar.
- Apostar (*bet/raise*): Iniciar ou aumentar uma aposta com o valor mínimo de $\max(\text{blind}, \text{pote})$.
- Pagar (*call*): Igualar a aposta do oponente.
- Desistir (*fold*): Abandonar a mão e perder as fichas já apostadas.

Hierarquia das Mãos:

Um par Qualquer (QQ, KK, AA) é a mão mais forte, formada pela sua carta privada e pela carta comunitária.

Caso não exista um par, a carta mais alta define a qualidade:

- dama (Q): A carta mais fraca.
- rei (K): Mais forte que a dama.
- ás (A): Mais forte que o rei e a dama.

Showdown:

Após a segunda rodada de apostas, os jogadores revelam suas cartas.

O jogador com a melhor mão ganha o pote.

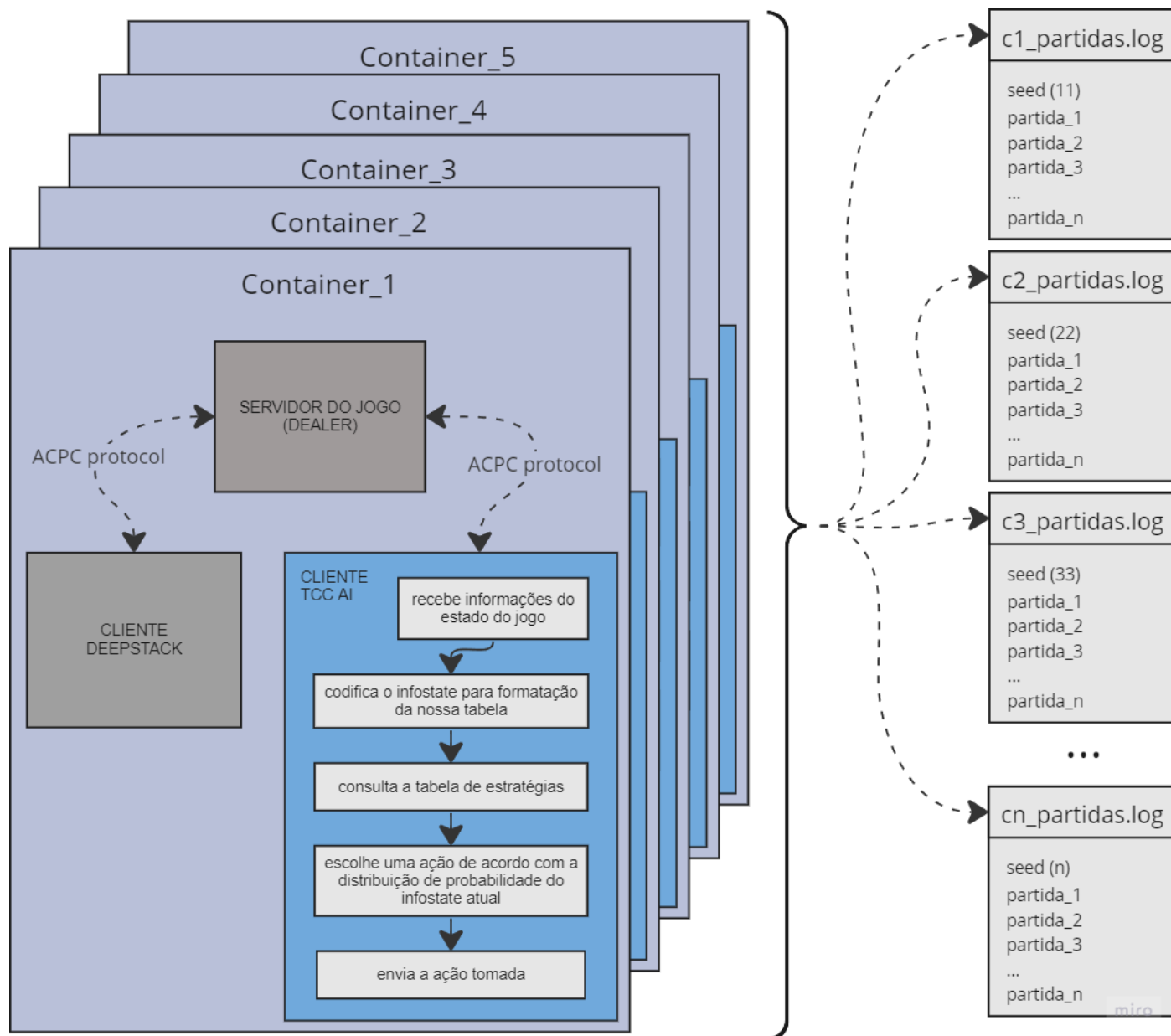
Em caso de empate, seja por ambos terem a mesma carta privada, o pote é dividido igualmente.

5.3.2. Ambiente de simulação DeepStack

Durante a fase de teste do nosso modelo, realizamos várias rodadas de simulações em um ambiente distribuído fornecido pela equipe que desenvolveu o DeepStack. Este ambiente inclui uma versão adaptada da DeepStack IA (MORAVČÍK et al., 2017) especificamente para o HUNL Leduc Poker e segue o protocolo ACPC (*Annual Computer Poker Competition*), que é um padrão para competições de poker entre programas.

Para realizar as simulações, optamos pela containerização da aplicação distribuída utilizando Docker **figura 12**. A containerização foi escolhida por dois motivos principais: primeiro, para gerenciar as dependências de forma eficiente, garantindo que as simulações pudessem ser executadas em diferentes sistemas sem a necessidade de configurações adicionais; segundo, para facilitar a execução paralela das simulações. Utilizamos de 5 a 10 *containers* em paralelo, o que permitiu a realização das simulações de maneira eficaz e escalável. Essa abordagem maximizou o uso dos recursos computacionais disponíveis e reduziu o tempo total necessário para a execução das simulações. Cada *container* estava associado a uma *seed* para controle na distribuição de cartas.

Figura 12 - Ambiente de simulação containerizado da aplicação distribuída.



Fonte - Autoria própria

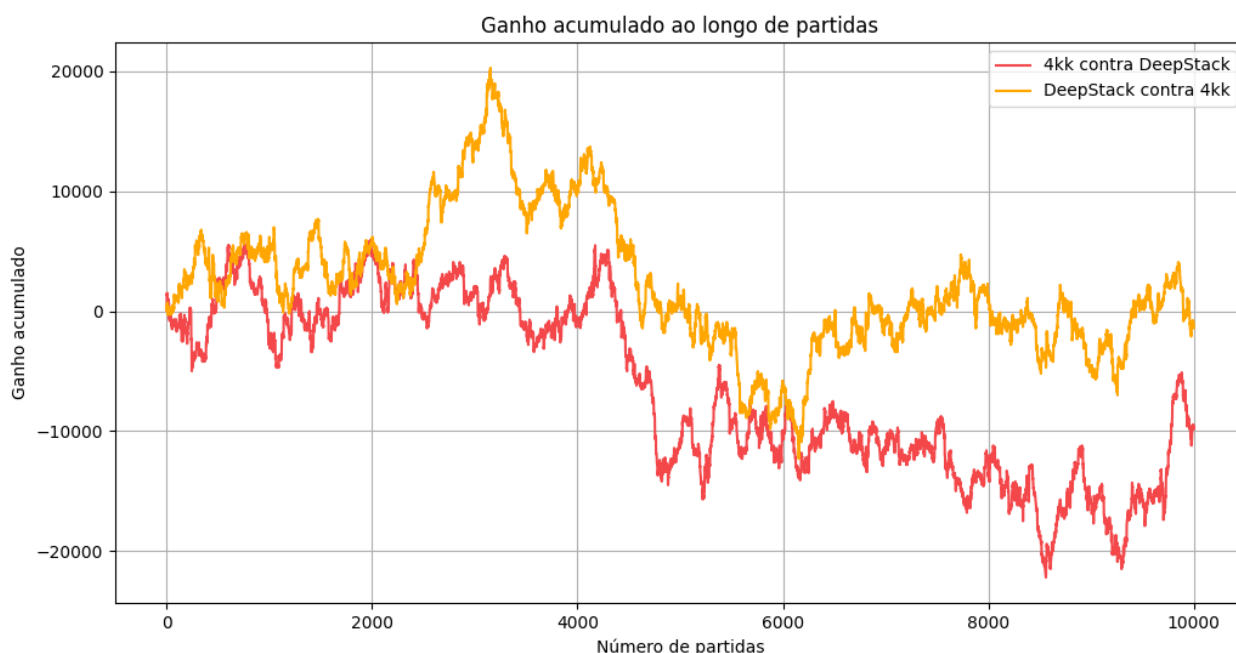
5.3.3. Resultados dos testes

Esta seção apresenta os resultados obtidos das simulações efetuadas no ambiente distribuído. Inicialmente, são discutidos os resultados dos testes preliminares utilizando a estratégia de 4kk (4.000.000 de iterações), conforme descrito no tópico 5.3.4.1. Em seguida os resultados para a estratégia de 20kk e 100kk, apresentados respectivamente nos tópicos 5.3.4.2 e 5.3.4.3. Finalmente, no tópico 5.3.4.4, é exposto um comparativo geral dos resultados.

5.3.3.1. Experimentos envolvendo a estratégia gerada após 4kk de iterações:

Os dados a seguir apresentam os resultados de 20.000 partidas. Destas, 10.000 partidas colocam o resultado do nosso treinamento de 4.000.000 de iterações contra o DeepStack, representados pela cor verde. Nas outras 10.000 partidas, representadas pela cor dourada, a distribuição das mãos se faz invertida, visando que os oponentes recebam as cartas uns dos outros do primeiro conjunto de simulações. Este método coloca as duas IAs nas mesmas situações, com o propósito de minimizar o fator de sorte nos resultados.

Figura 13 - Gráfico de ganho acumulado 4kk vs DeepStack



Fonte - Autoria própria

Tabela 8 – Dados estatísticos das 20.000 partidas entre 4kk e DeepStack.

Estratégia	Total de Jogos	Taxa de Vitória	Taxa de Empate	Taxa de Derrota	Margem Média de Vitória	Margem Média de Derrota	Desvio Padrão	Valor Esperado
4kk	10.000	42,39%	13,94%	43,67%	172,71	169,89	237,40	-0,98
DeepStack	10.000	42,27%	13,87%	43,86%	171,02	165,14	232,70	-0,14

Fonte - Autoria própria

A estratégia de 4kk, quando comparada ao DeepStack, apresentou uma taxa de vitória, empate e derrota muito parecidas, com apenas uma diferença decimal. Quanto às margens médias de vitória e derrota, a estratégia de 4kk registrou uma margem média de vitória um pouco mais elevada (172,71) em comparação com o DeepStack (171,02), sugerindo uma eficiência marginalmente superior na maximização das vitórias. No entanto, a mesma estratégia registrou perdas com margens maiores, indicando possíveis riscos maiores nas derrotas. O desvio padrão foi levemente maior para a estratégia de 4kk, indicando uma maior variação nos resultados e um potencial aumento no risco de perdas.

A diferença mais pronunciada entre as duas estratégias surgiu no cálculo do valor esperado (EV). A estratégia de 4kk iterações teve um EV negativo maior (-0,98) comparado ao do DeepStack (-0,14), o que sugere uma eficácia reduzida em termos de retorno esperado por partida, a métrica mais importante na avaliação de estratégias de jogo. Portanto, apesar de a estratégia de 4kk iterações e o DeepStack apresentarem taxas de vitória, empate e derrota próximas, o EV negativo mais acentuado da estratégia de 4kk iterações indica um desempenho inferior do ponto de vista financeiro a longo prazo. A causa principal deste resultado negativo se deve à margem média de derrota ser maior para a estratégia de 4kk, e após uma análise exploratória dos dados da simulação foi possível resgatar as seguintes informações:

Tabela 9 – Dados estatísticos complementares das 20.000 partidas de 4kk e DeepStack.

	% de folds	perda média em folds	% de check/calls	perda média em check/call
4kk	71,51	-134,39	28,41	-256,72
DeepStack	72,02	-133,36	27,92	-245,38

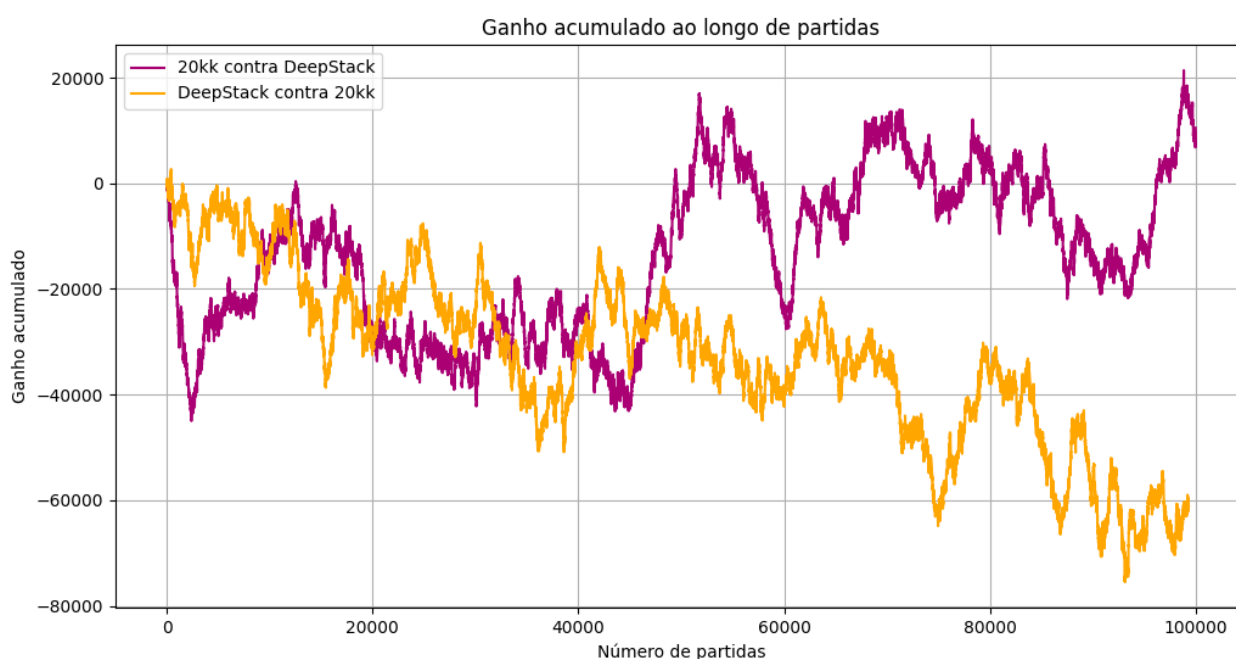
Fonte - Autoria própria

Portanto, a principal razão para a margem média de perda, é a perda média em situações de *check/call*. Isso sugere que a estratégia de treinamento de 4kk está frequentemente chegando a confrontos com mãos menos favoráveis, indicando a necessidade de um treinamento mais extenso para aprimorar e ajustar as decisões de ação que levam a esse resultado.

5.3.3.2. Experimentos envolvendo a estratégia gerada após 20kk de iterações:

Para esta segunda bateria de testes, decidimos aumentar a quantidade de partidas simuladas para reduzir a variância e então alcançar resultados mais representativos. Os dados a seguir apresentam os resultados de aproximadamente 200.000 partidas seguindo o mesmo método de inversão de mãos do experimento passado, porém utilizando a estratégia de **20.000.000** de iterações.

Figura 14 - Gráfico de ganho acumulado 20kk vs DeepStack



Fonte - Autoria própria

Tabela 10 – Dados estatísticos das 200.000 partidas entre 20kk e DeepStack.

Estratégia	Total de Jogos	Taxa de Vitória	Taxa de Empate	Taxa de Derrota	Margem Média de Vitória	Margem Média de Derrota	Desvio Padrão	Valor Esperado
20kk	100.000	41,23%	14,73%	44,02%	165,02	154,32	214,59	0,1050
DeepStack	99.329	43,86%	14,85%	41,27%	154,79	165,99	215,76	-0,6161

Fonte - Autoria própria

Primeiramente é importante salientar que devido a um problema inesperado com o ambiente de simulação containerizado, não foi possível executar 671 partidas do conjunto de

partidas com mão invertida. Os dados das partidas rodadas até o corrimento do problema não foram afetados. Como essa falta de 0,67% do segundo grupo de dados é mínima e provavelmente não tem um impacto significativo na comparação dos resultados, decidimos não repetir as simulações e considerá-la válida.

O primeiro aspecto notável do resultado é que a proporção de vitórias, empates e derrotas das duas estratégias, independente se na situação direta ou inversa, apresentam constância. O 20kk manteve uma taxa de vitória de 41,23% na primeira sequência e 41,27% na outra, 14,73% e 14,85% de empates, e 44,02% e 43,86% de derrotas, indicando que a taxa de vitória possui uma relação maior com a estratégia do que com a sorte nas mãos.

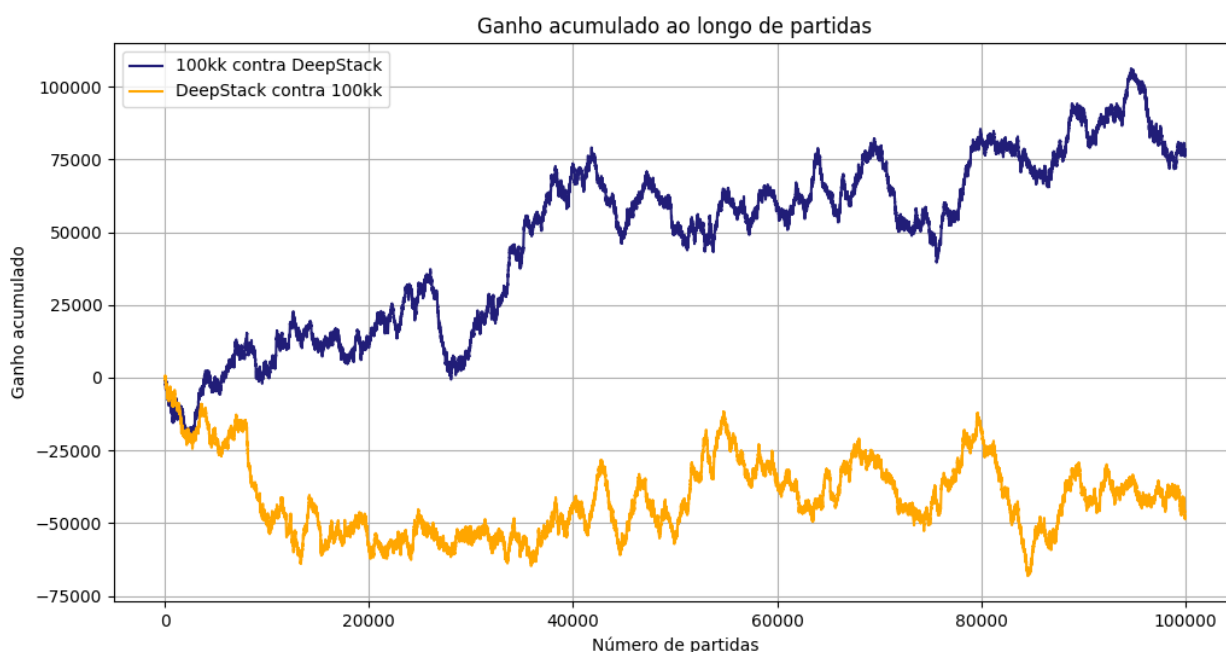
Em relação às margens médias de vitória e derrota, a estratégia de 20kk exibe uma capacidade ligeiramente superior de maximizar os ganhos em suas vitórias, evidenciada por uma margem média de vitória de 165,03 em comparação a 154,80 do DeepStack quando invertemos as posições. No entanto, esta mesma estratégia sofre derrotas com margens menores, sugerindo uma gestão de risco mais eficiente nas perdas.

A volatilidade dos resultados, medida pelo desvio padrão, permaneceu comparável entre as duas estratégias, insinuando uma consistência na dispersão dos resultados obtidos por ambas as estratégias. Entretanto, a análise do valor esperado nos oferece a perspectiva mais reveladora desta comparação. A estratégia de 20kk demonstrou um EV positivo de 0,105, enquanto que o DeepStack na situação inversa apresentou um EV negativo consideravelmente maior, de -0,616. Isso sugere que, apesar de pequenas variações nas taxas de vitória, empate e derrota, a estratégia de 20kk tem um desempenho financeiro superior no longo prazo, considerando os dados da amostra.

As causas subjacentes a este EV negativo para o DeepStack, quando comparado à estratégia de 20kk, são multifacetadas, mas provavelmente tem forte relação com o fato da amostra que, mesmo grande, não alcança significância estatística devido a sua alta variância.

5.3.3.3. *Experimentos envolvendo a estratégia gerada após 100kk de iterações:*

Finalmente, para esta última série de testes, mantivemos a quantidade de partidas simuladas em 200.000, pois aumentar esse número tornaria impraticável o tempo de simulação. Os dados a seguir apresentam os resultados dessas 200.000 partidas, empregando o mesmo método de inversão de mãos utilizado nos experimentos anteriores, porém aplicando a estratégia de **100.000.000** de iterações.

Figura 15 - Gráfico de ganho acumulado 100kk vs DeepStack

Fonte - Autoria própria

Tabela 11 – Dados estatísticos das 200.000 partidas entre 100kk e DeepStack.

Estratégia	Total de Jogos	Taxa de Vitória	Taxa de Empate	Taxa de Derrota	Margem Média de Vitória	Margem Média de Derrota	Desvio Padrão	Valor Esperado
100kk	100.000	40,90%	14,78%	44,30%	168,67	153,97	220,17	0,7800
DeepStack	100.000	44,19%	14,83%	40,97%	155,15	168,50	220,50	-0,4780

Fonte - Autoria própria

Novamente, é evidenciado a constância na proporção de vitórias, empates e derrotas das duas estratégias, independente se na situação direta ou inversa. O 100kk manteve uma taxa de vitória de 40,9% na primeira sequência e 40,975% na outra, 14,78% e 14,83% de empates, e 44,3% e 44,19% de derrotas, reforçando que a taxa de vitória possui uma relação maior com a estratégia do que com a sorte nas mãos.

As margens médias tanto de vitória quanto derrota também se mostram estáveis, independente se na situação direta ou inversa, onde o 100kk, quando vitorioso, assegura uma margem mais ampla (168,67 contra 155,15), e quando derrotado, apresenta uma margem mais

estreita (153,97 contra 168,50) em comparação ao DeepStack. Este dado indica que o 100kk tem uma vantagem dual: maximiza ganhos e minimiza perdas de forma mais efetiva que o seu oponente, o que corrobora para que o EV seja positivo para o 100kk.

A proximidade dos desvios padrão sinaliza uma variação de resultados similar entre as duas estratégias, o que implica uma consistência no nível de risco associado a ambas.

Contudo, a discrepância mais significativa se dá nos valores esperados. A estratégia de 100kk apresenta um valor esperado positivo (0,78), enquanto que o DeepStack, ao competir contra o 100kk, detém um valor esperado negativo (-0,478). Esta diferença indica que o 100kk supera o DeepStack em termos de rentabilidade esperada, dentro dos limites de significância da amostra.

Concluindo, a estratégia de 100kk revela-se mais vantajosa contra o DeepStack, com base na maior taxa de vitória e um valor esperado substancialmente mais benéfico. Por outro lado, o DeepStack, apesar de sua eficácia em acumular vitórias contra o 100kk, enfrenta um desafio em termos de retorno financeiro, evidenciado por um valor esperado negativo que sinaliza um desempenho monetário menos favorável.

5.3.3.4. Panorama dos testes

Encerrando a série de experimentos, este tópico propõe uma comparação panorâmica dos resultados alcançados. A tabela a seguir agrupa os resultados das simulações passadas e adiciona um novo conjunto de simulações do DeepStack jogando contra si para definir um baseline entre a linhas de ganhos:

Figura 16 - Gráfico de ganho acumulado DeepStack, 4kk, 20kk e 100kk.

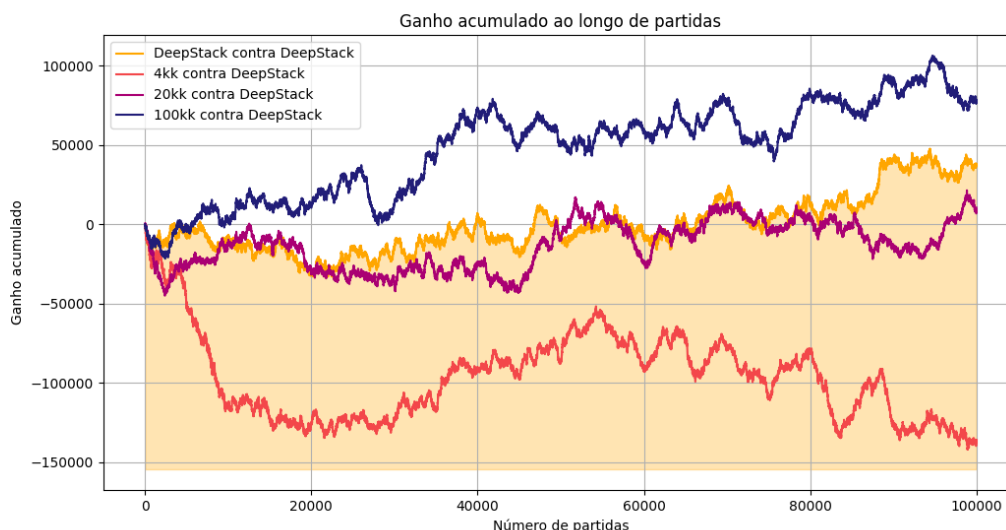


Tabela 12 – Dados estatísticos das 400.000 partidas entre todas as estratégias e DeepStack.

	DeepStack	4kk	20kk	100kk
Tamanho da amostra	100.000	100.000	100.000	100.000
Taxa de vitória	0,42634	0,41444	0,4123	0,40907
Taxa de empate	0,14845	0,14641	0,14733	0,14788
Taxa de derrota	0,42521	0,43915	0,44029	0,44305
Margem média de vitória	159,710	174,621	165,027	168,665
Margem média de derrota	159,248	167,892	154,327	153,969
Desvio padrão	216,672	234,427	214,598	220,165
Valor esperado	0,37699	-1,3599	0,105	0,77999
Erro padrão	0,685	0,741	0,679	0,696
mili big blinds por mão (mbb/g)	3,76 ± 6,85	-13,6 ± 7,41	1,05 ± 6,79	7,79 ± 6,96

Fonte - Autoria própria

Na série de testes realizada, observamos uma melhora evolutiva no desempenho da nossa IA em confronto com o DeepStack. Inicialmente, com a estratégia de 4 milhões de iterações (4kk), a IA apresentou um desempenho bastante inferior ao do DeepStack. Isso foi evidenciado por uma perda média de -13,59 mbb/g em comparação com o ganho médio do DeepStack de 3,76 mbb/g.

À medida que aumentamos o número de iterações para 20 milhões (20kk), a IA demonstrou uma melhoria significativa. Esta estratégia alcançou um ganho médio de 1,05 mbb/g, ainda abaixo do DeepStack, porém com uma melhora significativa em relação a estratégia de 4 milhões de iterações (4kk).

A estratégia mais avançada, com 100 milhões de iterações (100kk), obteve o melhor resultado. Neste cenário, a IA alcançou um ganho médio de 7,79 mbb/g, superando o DeepStack, que registrou o ganho médio de 3,7 mbb/g. Este resultado sugere a eficácia da estratégia de 100kk em termos de rentabilidade a longo prazo.

Os resultados sugerem que, com um treinamento mais extenso e aprimorado, nossa IA pode apresentar uma evolução contínua, culminando em uma abordagem estratégica que tem potencial para rivalizar com o desempenho do DeepStack. Contudo, deve-se considerar que, no contexto dos jogos de poker, elementos de variabilidade e aleatoriedade ainda têm um

papel significativo, o que limita a possibilidade de assegurar sucesso absoluto em todas as situações.

6. CONCLUSÃO

O projeto teve como objetivo central o desenvolvimento de uma inteligência artificial capaz de competir contra outras IAs no jogo de HUNL Leduc Poker, utilizando o algoritmo de MCCFR. Embora tenhamos obtido êxito nesse propósito, com a implementação do algoritmo e resultados promissores nos testes realizados, é necessário reconhecer as limitações e desvios em relação aos objetivos inicialmente delineados. A decisão de focar no HUNL Leduc Poker em detrimento do HUNL Texas Hold'em Poker foi motivada por razões práticas e de escopo. Optamos pelo anterior devido ao seu número substancialmente menor de estados em comparação com o HUNL Texas Hold'em Poker, o que permitiu que concentrássemos nossos esforços no desenvolvimento do algoritmo MCCFR sem a complexidade adicional associada à abstração de estados no contexto do poker mais abrangente devido à quantidade maior de turnos de aposta e cartas.

Decidimos também não incorporar redes neurais ou cálculos de probabilidades baseado em redes bayesianas em nosso algoritmo, inspirados pela arquitetura do Pluribus, que obteve bons resultados sem depender desses componentes. Ao invés disto, nos concentramos na implementação e otimização do MCCFR para atingir uma estratégia próxima a um equilíbrio de Nash, resultando em um modelo pouco “exploitável” pelo adversário. Essa decisão estratégica não apenas alinhou nosso trabalho com as abordagens bem-sucedidas observadas no Pluribus, mas também influenciou diretamente a metodologia dos experimentos. Os resultados desses experimentos destacaram uma melhora significativa na performance do algoritmo à medida que o número de iterações aumentava, o que culminou em uma estratégia mais robusta e eficiente quando submetida a 100 milhões de iterações. Essa robustez foi evidenciada nos confrontos simulados contra o DeepStack, ressaltando a competência do algoritmo desenvolvido. A estratégia com 100 milhões de iterações exibiu uma margem de vitória superior, e apresentou um valor esperado positivo.

Contudo, como todo estudo, este também possui suas limitações. As principais residem na dependência do número de iterações para a eficácia do algoritmo, nas limitações computacionais enfrentadas, e principalmente, na significância estatística dos testes, que está abaixo de 95% para o intervalo positivo, e indica que com testes com amostras maiores ou com técnicas de redução de variância os resultados poderiam ter sido outros.

6.1. TRABALHOS FUTUROS

Para trabalhos futuros, sugerimos várias melhorias e expansões que podem aprimorar significativamente a eficiência do nosso modelo atual. Uma delas é a implementação de uma técnica de redução de variância, como o AIVAT (BURCH, 2018), que pode permitir a avaliação dos modelos com menos simulações de partidas, mantendo a significância estatística dos resultados. Essa técnica poderia reduzir drasticamente a quantidade de dados necessários para testar a eficácia de um modelo.

Outra adição valiosa seria integrar uma funcionalidade de busca em tempo real ao algoritmo existente. Isso complementaria a estratégia pré-computada e estaria alinhada com as abordagens utilizadas nas melhores IAs para jogos atualmente disponíveis. Essa melhoria permitiria que o modelo se adaptasse dinamicamente durante o jogo, aumentando sua capacidade de responder a estratégias adversárias inesperadas.

Além disso, o uso de linguagens compiladas, como Cython, poderia melhorar significativamente o tempo de execução do algoritmo. Isso tornaria o processo de treinamento e simulação mais eficiente, o que é crucial para desenvolvimentos e testes rápidos.

Uma capacidade que também deve ser explorada é a habilidade do modelo de identificar e se adaptar a comportamentos adversários que desviem do equilíbrio de Nash, similar ao que é observado no modelo Pluribus. Isso permitiria uma adaptação contínua às estratégias dos oponentes ao longo do jogo, tornando o modelo mais resiliente e versátil em cenários de jogo variados.

Finalmente, visando expandir a aplicabilidade do modelo para o jogo completo de HUNL Poker, sugerimos o desenvolvimento de técnicas de abstração para agrupar mãos semelhantes, conhecidas como *bucketing*. Essa abordagem permitiria categorizar mãos com base em padrões de apostas no *pre-flop* e em combinações específicas de cartas nas fases subsequentes do jogo. A categorização das cartas da mesa com base na probabilidade de completar sequências ou *flushes* e no valor relativo das cartas também parece promissor. Além disso, a criação de uma função de mapeamento para agrupar os nodos finais de cada fase do jogo em clusters facilitaria o gerenciamento de estratégias sem a necessidade de criar uma única árvore para todo o jogo, algo inviável em um ambiente de computação padrão. Essas melhorias e expansões podem abrir caminho para avanços significativos na capacidade e aplicabilidade do nosso modelo de IA em contextos de poker mais complexos.

REFERÊNCIAS

SUTTON, Richard S.; BARTO, Andrew G. Reinforcement learning: An introduction. MIT press, 2018. Disponível em:

<<https://books.google.com/books?hl=pt-BR&lr=&id=uWV0DwAAQBAJ&oi=fnd&pg=PR7&ots=mivLv533n0&sig=9IKMIVSen27Oq4suSNzaoDI4gEo>>. Acesso em 27 jun. 2023.

EL NAQA, Issam; MURPHY, Martin J. What is machine learning?. Springer International Publishing, 2015. Disponível em:

<https://link.springer.com/chapter/10.1007/978-3-319-18305-3_1>. Acesso em 27 jun. 2023.

BROWN, Noam; SANDHOLM, Tuomas. Safe and nested subgame solving for imperfect-information games. Advances in neural information processing systems, v. 30, 2017. Disponível em:

<<https://proceedings.neurips.cc/paper/2017/file/7fe1f8abaad094e0b5cb1b01d712f708-Paper.pdf>>. Acesso em 22 jun. 2023.

BROWN, Noam et al. Combining deep reinforcement learning and search for imperfect-information games. Advances in Neural Information Processing Systems, v. 33, p. 17057-17069, 2020. Disponível em:

<<https://proceedings.neurips.cc/paper/2020/hash/c61f571dbd2fb949d3fe5ae1608dd48b-Abstract.html>>. Acesso em 22 jun. 2023.

LI, Xun; MIIKKULAINEN, Risto. Opponent modeling and exploitation in poker using evolved recurrent neural networks. In: Proceedings of the Genetic and Evolutionary Computation Conference. 2018. p. 189-196. Disponível em

<<https://dl.acm.org/doi/abs/10.1145/3205455.3205589>>. Acesso em 22 jun. 2023.

BAJADA, Josef. Symbolic vs connectionist a.i. 2019. Towards Data Science. 2019. Disponível em:

<<https://towardsdatascience.com/symbolic-vs-connectionist-a-i-8cf6b656927>>. Acesso em 22 jun. 2023.

BROWN, Noam; SANDHOLM, Tuomas. Superhuman AI for multiplayer poker. *Science*, v. 365, n. 6456, p. 885-890, 2019. Disponível em:

<<https://www.science.org/doi/10.1126/science.aay2400>>. Acesso em 22 jun. 2023.

BOWLING, Michael et al. Heads-up limit hold'em poker is solved. *Science*, v. 347, n. 6218, p. 145-149, 2015. Disponível em <<https://www.science.org/doi/10.1126/science.1259433>>. Acesso em 22 jun. 2023.

NAGINENI, Arjun; NOVAK, Gordon; VAN DE GEIJN, Robert. Learning Useful Features For Poker. 2018. Disponível em <<https://nn.cs.utexas.edu/downloads/papers/nagineni-thesis18.pdf>>. Acesso em 22 jun. 2023.

WANG, Shuai et al. Auto-encoder neural network based prediction of Texas poker opponent's behavior. *Entertainment Computing*, v. 40, p. 100446, 2022. Disponível em <<https://www.sciencedirect.com/science/article/abs/pii/S1875952121000434?via%3Dihub>>. Acesso em 22 jun. 2023.

MORAVČÍK, Matej et al. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, v. 356, n. 6337, p. 508-513, 2017. Disponível em <<https://www.science.org/doi/abs/10.1126/science.aam6960>>. Acesso em 08 nov. 2023.

GIBSON, Richard G. Regret minimization in games and the development of champion multiplayer computer poker-playing agents. 2014. Disponível em <<https://era.library.ualberta.ca/items/15d28cbf-49d4-42e5-a9c9-fc55b1d816af>>. Acesso em 22 jun. 2023.

BILLINGS, Darse et al. The challenge of poker. *Artificial Intelligence*, v. 134, n. 1-2, p. 201-240, 2002. Disponível em <<https://www.sciencedirect.com/science/article/pii/S0004370201001308>>. Acesso em 22 jun. 2023.

COSTA, Alexandre Marangoni. A Study on Neural Networks for Poker Playing Agents. 2019. Tese de Doutorado. PUC-Rio. Disponível em <<https://www.maxwell.vrac.puc-rio.br/48011/48011.PDF>>. Acesso em 22 jun. 2023.

CAMPBELL, Murray; HOANE, A. Joseph; HSU, F. hsiung. Deep blue. Artificial Intelligence, v. 134, n. 1, p. 57–83, 2002. ISSN 0004-3702. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0004370201001291>>. Acesso em 22 jun. 2023.

HAYASHI, Chikio et al. (Ed.). Data Science, Classification, and Related Methods: Proceedings of the Fifth Conference of the International Federation of Classification Societies (IFCS-96), Kobe, Japan, March 27–30, 1996. Springer Science & Business Media, 2013. Disponível em: <<https://link.springer.com/book/10.1007/978-4-431-65950-1>>. Acesso em 22 jun. 2023.

SILVER, David et al. Mastering the game of Go with deep neural networks and tree search. nature, v. 529, n. 7587, p. 484-489, 2016. Disponível em: <<https://www.nature.com/articles/nature16961%7D>>. Acesso em: 22 jun. 2023

BROWN, Noam. Equilibrium finding for large adversarial imperfect-information games. PhD thesis, 2020. Disponível em: <<http://reports-archive.adm.cs.cmu.edu/anon/2020/CMU-CS-20-132.pdf>>. Acesso em 22 jun. 2023.

TESAURO, Gerald. TD-Gammon: A Self-Teaching Backgammon Program. 1995. Disponível em: <https://link.springer.com/chapter/10.1007/978-1-4757-2379-3_11>. Acesso em 22 jun. 2023.

ZHANG, K., YANG, Z., & Başar, T.. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In K.G. Vamvoudakis, Y. Wan, F.L. Lewis, & D. Cansever (Eds.), Handbook of Reinforcement Learning and Control (páginas 321-384). Studies in Systems, Decision and Control, Vol. 325. Cham: Springer. DOI: 10.1007/978-3-030-60990-0_12. Disponível em : <https://link.springer.com/chapter/10.1007/978-3-030-60990-0_12>. Acesso em 22 jun. 2023.

RUBINSTEIN, Reuven Y.; KROESE, Dirk P. Simulation and the Monte Carlo method. John Wiley & Sons, 2016. Acesso em: 22 jun. 2023.

SILVER, D.; ANTONOGLU, I.; SIFRE, L.; LILLICRAP, T. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. Disponível em:

<<https://arxiv.org/abs/1712.01815>>. Acesso em: 22 jun. 2023.

KAELBLING, Leslie Pack; LITTMAN, Michael L.; MOORE, Andrew W. Reinforcement learning: A survey. Journal of artificial intelligence research, v. 4, p. 237-285, 1996.

Disponível em: <<https://www.jair.org/index.php/jair/article/view/10166/24110>>. Acesso em: 28 jun. 2023.

ABIODUN, Oludare Isaac et al. State-of-the-art in artificial neural network applications: A survey. Heliyon, v. 4, n. 11, p. e00938, 2018. Disponível em:

<<https://www.sciencedirect.com/science/article/pii/S2405844018332067>>. Acesso em 28 jun. 2023.

BROWN, Noam; SANDHOLM, Tuomas. Superhuman AI for heads-up no-limit poker:

Libratus beats top professionals. Science, v. 359, n. 6374, p. 418-424, 2018. Disponível em

<<https://www.science.org/doi/full/10.1126/science.aao1733>>. Acesso em 29 jun. 2023.

NASH, John. Non-cooperative games. Annals of Mathematics, v. 54, n. 2, p. 286-295, 1951.

OSBORNE, Martin J.; RUBINSTEIN, Ariel. A course in game theory. MIT press, 1994. Disponível em:

<<https://sites.math.rutgers.edu/~zeilberg/EM20/OsborneRubinsteinMasterpiece.pdf>>. Acesso em 29 jun. 2023.

BINMORE, Ken. Game theory: a very short introduction. OUP Oxford, 2007. Disponível em

<<https://ideas.repec.org/b/oxp/obooks/9780199218462.html>>. Acesso em 29 jun. 2023.

NELLER, Todd W.; LANCTOT, Marc. An introduction to counterfactual regret minimization. In: Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013). 2013. Disponível em

<<http://cs.gettysburg.edu/~tneller/modelai/2013/cfr/cfr.pdf>>. Acesso em 07 nov. 2023.

LANCTOT, Marc et al. Monte Carlo sampling for regret minimization in extensive games. Advances in neural information processing systems, v. 22, 2009. Disponível em <https://papers.nips.cc/paper_files/paper/2009/hash/00411460f7c92d2124a67ea0f4cb5f85-Abstract.html>. Acesso em 07 nov. 2023.

KUHN, H. W. Simplified Two-Person Poker. In: KUHN, H. W.; TUCKER, A. W. (Org.). Contributions to the Theory of Games. Vol. 1. Princeton University Press, 1950. p. 97-103.

SOUTHEY, Finnegan et al. Bayes' bluff: Opponent modelling in poker. arXiv preprint arXiv:1207.1411, 2012. Disponível em <<https://poker.cs.ualberta.ca/publications/UAI05.pdf>>. Acesso em 08 nov. 2023.

MÜNCH, Bc Martin. Algorithms for Playing Multi-player Simplified Poker. 2017. Disponível em <<https://core.ac.uk/download/pdf/84834175.pdf>>. Acesso em 08 nov. 2023.

BURCH, Neil et al. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2018. Disponível em <<https://doi.org/10.1609/aaai.v32i1.11481>>. Acesso em 13 nove. 2023.

APÊNDICE A

Aqui estão disponibilizados os repositórios com as implementações do projeto:

Repositório contendo a implementação do algoritmo de treinamento e interface Web:

- <https://github.com/njoppi2/poker-ml>

Repositório contendo o ambiente de simulação DeepStack:

- <https://github.com/IgorMayWensing/DeepStack-Leduc-Container>