UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Lana Bertoldo Rossato

MODELAGEM MATEMÁTICA E DESENVOLVIMENTO DE UM AGENTE BASEADO EM MODELOS MARKOVIANOS

Santa Maria, RS
2020

Lana Bertoldo Rossato

MODELAGEM MATEMÁTICA E DESENVOLVIMENTO DE UM AGENTE BASEADO EM MODELOS MARKOVIANOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

ORIENTADOR: Prof. Joaquim Vinicius Carvalho Assunção

472

Santa Maria, RS
2020

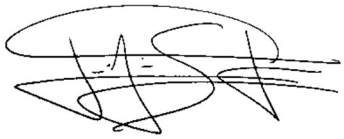
Lana Bertoldo Rossato

MODELAGEM MATEMÁTICA E DESENVOLVIMENTO DE UM AGENTE BASEADO EM MODELOS MARKOVIANOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

Aprovado em 30 de setembro de 2020:

Joaquim Vinicius Carvalho Assunção, Dr. (UFSM)
(Presidente/Orientador)



Luís Alvaro de Lima Silva, Dr. (UFSM)

Felipe Martins Muller, Dr. (UFSM)

Santa Maria, RS
2020

RESUMO

MODELAGEM MATEMÁTICA E DESENVOLVIMENTO DE UM AGENTE BASEADO EM MODELOS MARKOVIANOS

AUTORA: Lana Bertoldo Rossato
ORIENTADOR: Joaquim Vinicius Carvalho Assunção

Nos últimos anos, é possível ver o avanço da Inteligência Artificial aplicada aos mais diversos jogos, sejam eles determinísticos ou estocásticos. Muitas técnicas, apesar de alcançarem o resultado esperado, requerem um considerável poder computacional e não são viáveis em plataformas mais básicas (e.g., smartphones de baixo custo). Com isso, há a necessidade de criação de técnicas de peso leve. Esse trabalho utiliza Cadeias de Markov no desenvolvimento de um agente para um jogo de cartas. A modelagem foi feita para representar a ordem natural das forças e separa em três módulos de acordo com as modalidades do jogo, sendo elas: Truco, Envido e Flor. A tomada de decisão é feita com base em partidas jogadas anteriormente, reforçando as decisões tomadas e melhorando-as. Para avaliar o modelo, diversas rodadas de testes foram feitas, variando o oponente e as bases de dados. Esse processo resultou em matrizes de aprendizado para cada modo do jogo. Assim, elas foram avaliadas levando em consideração as características principais do modelo e também dos oponentes.

Palavras-chave: Cadeias de Markov. Modelagem Matemática. Inteligência Artificial. Agentes para Jogos.

ABSTRACT

MATHEMATICAL MODELING AND DEVELOPMENT OF AN AGENT BASED ON MARKOVIAN MODELS

AUTHOR: Lana Bertoldo Rossato
ADVISOR: Joaquim Vinicius Carvalho Assunção

In the last years, it is possible to see the advance of Artificial Intelligence applied to the most diverse games, whether they are deterministic or stochastic. Many techniques, despite achieving the expected result, require considerable computational power and are not viable on more basic platforms (e.g., low-cost smartphones). Thus, there is a need to create light weight techniques. This work uses Markov Chains in the development of an agent for a card game. The modeling was performed to represent the natural order of the forces and separates it into three modules according to the game's modalities, being: Truco, Envido and Flor. Decision making is based on games played previously, reinforcing the decisions made and improving them. To evaluate the model, several rounds of tests were made, varying the opponent and the databases. This process resulted in learning matrices for each game mode. Thus, they were evaluated considering the main characteristics of the model and also of the opponents.

Keywords: Markov Chains. Mathematical Modeling. Artificial Intelligence. Game Agents.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Jogos digitais sempre foram um caso de estudo interessante, seja para encontrar padrão no comportamento de jogadores (THURAU; HETTENHAUSEN; BAUCKHAGE, 2006; BENMAKRELOUF; MEZGHANI; KARA, 2015) ou para reconhecer gestos de entrada para o jogo (KRATZ; SMITH; LEE, 2007). Além de serem amplamente utilizados como meio de diversão e estarem em praticamente todos os lugares, eles ainda representam problemas do mundo real e suas soluções podem ser usadas ou adaptadas para estes problemas (BALAN; OTTO, 2017). Em particular, os jogos de cartas proporcionam um ambiente interessante à pesquisa, visto que há diversas adversidades, como visão parcial do jogo, pelas cartas do oponente que não são visíveis e distribuição aleatória, quando as cartas são embaralhadas no começo do jogo.

Jogos de cartas são diferentes de jogos determinísticos, que se tem visão total do jogo, como o Xadrez, Go e Damas. Esse cenário se tornou um desafio (NIKLAUS et al., 2019) e ainda faz com que jogadores humanos tenham vantagem sobre os agentes artificiais do estado da arte. No entanto, diversas tentativas foram feitas e há avanços em jogos como Texas hold'em Poker (AMBEKAR et al., 2015; WATSON et al., 2008) e StarCraft II (SYNNAEVE; BESSIERE, 2016), fazendo com que a diferença de desempenho entre humanos e agentes artificiais diminua.

Nesse contexto, a modelagem Markoviana utilizada deste trabalho tenta prever a chance que o jogador tem de vencer, dada somente as cartas da sua mão, já que não é possível ver as cartas do oponente. O parâmetro de probabilidade de vitória já se mostrou efetivo em trabalhos anteriores (PAULUS, 2019; AMBEKAR et al., 2015), reforçando então a base desse trabalho.

A aprendizagem por reforço é amplamente utilizada em vários contextos, entre eles o meio de jogos (VINYALS et al., 2017; ANDERSEN; GOODWIN; GRANMO, 2017). Aliado a técnica Markoviana (XU; SHI; WANG, 2018), o objetivo do aprendizado por reforço é obter o máximo valor de recompensa cumulativa no processo de Markov, fazendo com que o modelo aprenda com a sua ação. Nesse caso, a recompensa ou punição é por jogada bem sucedida.

Assim como em (AMBEKAR et al., 2015), este trabalho também utiliza a probabilidade de vitória como meio de auxiliar nas futuras jogadas. Porém, como em (WATSON et al., 2008), essa probabilidade é baseada em conhecimento prévio. Apesar de ambos (AMBEKAR et al., 2015; WATSON et al., 2008) também terem como alvo um jogo de cartas, as técnicas utilizadas diferem da aplicada nesse trabalho, porém mantendo o mesmo objetivo. Ainda, o estudo de (SYNNAEVE; BESSIERE, 2016) sobre o jogo StarCraft II pode ser encaixado no mesmo problema deste trabalho, visto que ambos os jogos tem informações parciais. Da mesma maneira que (XU; SHI; WANG, 2018), a técnica de aprendizado por

reforço aliada a técnica de Markov também é utilizada nesta pesquisa, utilizando o mesmo esquema de ação e recompensa.

O presente trabalho está organizado como segue. No capítulo 2 são apresentados alguns trabalhos relacionados e como se assemelham ou diferenciam deste. O capítulo 3 descreve o background sobre os meios utilizados nesse trabalho, sendo eles o jogo de Truco, os Modelos Markovianos e a Aprendizagem por Reforço. As seções explicam alguns conceitos iniciais sobre as técnicas utilizadas e dá uma visão geral sobre o jogo de Truco e suas regras. O capítulo 4 apresenta a modelagem do agente, mostrando o espaço de estados e o modelo para cada modo de jogo. O capítulo 5 apresenta de forma geral a implementação do agente, juntamente com testes realizados e resultados obtidos. O capítulo 6 conclui o trabalho apresentado, dando também um panorama sobre os trabalhos futuros.

2 TRABALHOS RELACIONADOS

Há diversos jogos que se assemelham ao jogo de Truco por diversos quesitos. Alguns são semelhantes por também conterem variáveis aleatórias, como jogos em que cartas são embaralhadas ou onde há lançamento de dados. Outros por também contarem com a visão parcial do jogador, como jogos onde o que cada jogador tem só é revelado ao final de uma rodada, mão ou partida. Com isso, o jogo de Poker se torna um dos principais comparativos e objeto de pesquisa associada, uma vez que é mais conhecido do que o jogo de Truco, e por sua vez mais utilizado como ferramenta de pesquisa nesse meio. Outra similaridade é que o blefe também existe no Poker, tirando assim o foco que seria totalmente nas cartas e colocando parte dele também nas decisões do jogador, como por exemplo, se as cartas da mão do jogador oferecem a possibilidade do blefe pela possível falta de "cartas boas" na mão do adversário.

Conforme estudos de (AMBEKAR et al., 2015), saber onde e como apostar é importante para o sucesso no jogo. Uma aplicação foi desenvolvida para ajudar os jogadores a apostar com inteligência utilizando mineração de dados e probabilidade estatística no processo. O sistema dá ao jogador a probabilidade de vencer baseado nas cartas disponíveis, os únicos recursos visíveis e à disposição no momento. Dados de treino e teste foram utilizados para prever a classe em que as cartas da mão se encaixam usando um classificador Bayesiano juntamente com probabilidades estatísticas. Após vários experimentos utilizando árvores de decisão, eles obtiveram que o experimento com algoritmo Bayesiano foi o que mais aumentou a acurácia, chegando a 92.13%. Por fim, a importância do trabalho é mostrada pela comparação com a técnica apresentada e as técnicas que vêm sido usadas.

Tendo como objetivo deduzir o perfil do jogador com base nos dados de suas jogadas, (BENMAKRELOUF; MEZGHANI; KARA, 2015) partiram do pressuposto de que as características do jogador influenciam no seu desempenho e isso permite identificar um perfil. Para atingir tal objetivo, os autores utilizaram regressão linear múltipla, a fim de analisar os dados e para encontrar as características do jogador, e técnicas de clustering, mais especificamente o K-means, para extrair grupos de jogadores e identificar os perfis. Após um pré-processamento dos dados, os autores estudaram a correlação entre diversas características e selecionaram as variáveis para a futura análise. Assim, o modelo de regressão linear foi aplicado e mostrou que há efeitos significativos em variáveis consideradas dependentes, como acessos ao jogo e missões. Com o clustering foi possível identificar três tipos de jogadores: iniciante, intermediário e avançado, sendo eles grupos distintos de jogadores que dependem do comportamento e ações no jogo.

Para desenvolver um agente para jogos de cartas, (PAULUS, 2019) utilizaram uma junção de técnicas de casos e clusters aplicados ao reuso de ações do jogo presentes em

casos recuperados de uma consulta feita a um agente de Raciocínio Baseado em Casos (CBR) projetado para jogar cartas. Tendo o suporte do algoritmo K-means, critérios de reuso são aplicados a grupos presentes na base onde as ações de jogo são relacionadas com o estado em que foram feitas. O modelo de reuso tem dois passos e foi implementado em conjunto com diferentes combinações de critérios onde as ações e clusters foram escolhidos por meio do reuso da maioria, escolha da ação/cluster por métodos aleatórios, ações/clusters com maior probabilidade de vitória, ou escolha do cluster que proporciona maior ganho de pontos. Para avaliar essas propostas, agentes virtuais implementados com diferentes políticas foram submetidos a diferentes testes. Como resultado, no geral, obteve que os 3 dos 4 melhores agentes utilizavam de alguma forma a probabilidade de vencer.

Utilizando Cadeias Ocultas de Markov, (ZHOU; HUANG; WANG, 2004) propõem um novo método para reconhecimento de expressões faciais em jogos interativos. Para tal, a modelagem levou em consideração as partes do rosto a serem analisadas em uma expressão e essa modelagem embutida proporciona melhor desempenho e melhores estimativas iniciais. Como treinamento, diversas imagens foram usadas e nos testes o algoritmo apresentou bons resultados. Analisando pessoas que já estavam no conjunto de treinamento, alcançou 92% de sucesso, enquanto que em rostos novos, alcançou 84% de sucesso. Usando a mesma ferramenta, (KRATZ; SMITH; LEE, 2007) estudaram o reconhecimento de gestos que serviam de entrada para jogo. O modelo se mostrou rápido no reconhecimento do gesto, mas no treino não obteve bom desempenho quando a necessidade era treinamento em tempo de jogo. A acurácia do reconhecimento depende diretamente do tempo gasto com treinamento e do número de estados do modelo.

Ainda utilizando Cadeias Ocultas de Markov, (OKABE; SAITO; NAKAJIMA, 2005) objetivam gerar desenhos feitos a partir de um pincel. O modelo foi treinado junto com o algoritmo de Baum-Welch e estima qual será o próximo traço com base no traço feito anteriormente. Os dois estilos de pintura usados são diferentes e o modelo consegue alternar entre eles, levando em consideração aspectos importantes da pintura, como a fricção do pincel contra o papel. Isso mostra a eficácia do algoritmo proposto, tendo em consideração também as peculiaridades de cada estilo utilizado na aprendizagem. Já (THURAU; HETTENHAUSEN; BAUCKHAGE, 2006) estudaram o reconhecimento rápido e robusto de comportamentos e ações no mundo virtual dos jogos de futebol. Para reconhecer o comportamento do time, o trabalho também utiliza Cadeias Ocultas de Markov e realiza o treinamento para cada ação de interesse. Várias séries de experimentos revelaram que essa estrutura realmente produz um desempenho rápido e robusto.

O presente trabalho utiliza da modelagem Markoviana, como visto em (ZHOU; HUANG; WANG, 2004; KRATZ; SMITH; LEE, 2007; THURAU; HETTENHAUSEN; BAUCKHAGE, 2006), para reconhecer os padrões de jogadas anteriores. Essa identificação de padrões em jogos, também explorada em (BENMAKRELOUF; MEZGHANI; KARA, 2015) com o uso de clusters, utiliza a probabilidade de vitória como embasamento para as decisões

tomadas dentro do jogo (AMBEKAR et al., 2015). Essa probabilidade já se provou eficiente em (PAULUS, 2019), dentro do mesmo ambiente de jogo de cartas. A seguir, serão introduzidos alguns dos conceitos visto nos trabalhos anteriores e utilizados no presente trabalho.

3 BACKGROUND

Esta seção apresenta primeiramente o jogo de Truco, suas regras básicas e algumas definições importantes para o entendimento do trabalho. Fala também sobre os modelos Markovianos e seus conceitos principais, servindo como um conhecimento básico do tópico. Por fim, também são apresentadas algumas definições sobre aprendizagem por reforço, importantes para este projeto.

3.1 O JOGO DE TRUCO

Dentre as variações presentes no Brasil, as regras utilizadas nesse trabalho são as do Truco Gaudério ou Gaúcho, mais comumente jogado nas regiões do sul da América do Sul. Nele é utilizado o baralho espanhol e podem jogar 2, 3, 4, ou 6 pessoas, podendo também formar duplas ou trios. Cada jogador recebe três cartas e o objetivo é ganhar mais pontos que o adversário, tendo um limite que varia, podendo ser 12, 24 ou 30 pontos ou qualquer limite definido pelo grupo de jogadores naquele momento. Cada rodada consiste em cada jogador mostrar uma carta e o vencedor é quem tiver a carta mais forte. Cada mão, conjunto de três rodadas, vale um ponto, mas esse valor pode mudar com o pedido de Truco. Nele, o valor da mão pode ser aumentado, conforme mostrado na Tabela 3.1, e estes pontos refletem diretamente na pontuação final. Esse aumento é atingido gradativamente, havendo aceites e contrapropostas, chamadas de Retruco e Vale 4.

Tabela 3.1 – Pedidos e Contrapropostas

Pedido	1ª Contraproposta	2ª Contraproposta
Envido	Real Envido	Falta Envido
Envido	Falta Envido	-
Real Envido	Falta Envido	-
Falta Envido	-	-
Flor	Contra Flor	Contra Flor e Resto
Truco	Retruco	Vale 4

Fonte: Próprio autor.

O Envido e a Flor são disputas paralelas ao jogo que acrescentam pontos no placar final e ambos ocorrem no início da mão, antes de jogar a primeira carta. No Envido, um número é calculado a partir das cartas de cada jogador e quem tiver o maior número vence. Ele é calculado pela soma dos valores das cartas de mesmo naipe adicionando 20, esse último se dá justamente por terem o mesmo naipe. O valor de cada carta corresponde ao

número dela, exceto pelas cartas 10, 11 e 12, que valem zero. Como no Truco, no Envido também é possível aumentar o valor da aposta aceitando o pedindo e contrapropondo, como visto na Tabela 3.1. O Falta Envido não só aumenta a pontuação como vale os pontos que faltam para o jogador que está ganhando ganhar a partida. Ele pode não ganhar o jogo imediatamente, mas terá uma vantagem grandiosa sobre o adversário. Além disso, os pedidos de Real Envido e Falta Envido podem ser realizados mesmo sem o pedido de Envido ter acontecido anteriormente. Na Flor, o jogador precisa ter as três cartas com o mesmo naipe. Caso os dois jogadores atendam a essa condição, a disputa pode ter seus pontos elevados com o Contra Flor e o Contra Flor Resto, vide Tabela 3.1, esse último com as mesmas regras do Falta Envido. Para esses dois casos, a pontuação é calculada da mesma forma do Envido.

A Tabela 3.2 ilustra a força das cartas nos pedidos de Truco, Envido e Flor, sendo a coluna de Truco também válida para as jogadas normais do jogo.

Tabela 3.2 – Relação de força das cartas

Força	Truco	Envido/flor
1º	$1\spadesuit$	Quaisquer duas cartas do mesmo naipe (20 pontos)
2º	$1\clubsuit$	$7*$
3º	$7\spadesuit$	$6*$
4º	$7\diamond$	$5*$
5º	$3*$	$4*$
6º	$2*$	$3*$
7º	$1\diamond, 1\heartsuit$	$2*$
8º	$12*$	$1*$
9º	$11*$	$10*, 11*, 12*$ (0 pontos)
10º	$10*$	
11º	$7\heartsuit, 7\clubsuit$	
12º	$6*$	
13º	$5*$	
14º	$4*$	

Fonte: Próprio autor.

Durante toda a partida e em qualquer parte do jogo, o blefe é encorajado, visto que uma das características do jogo é a sua visão parcial. Por exemplo, no Envido é comum fazer o pedido com pouca pontuação imaginando que o oponente sentirá receio e negará os pontos. Assim, muitas vezes o jogador acaba por entregar os pontos que possivelmente poderia ter ganho se não tivesse recuado. Maiores informações e regras mais detalhadas podem ser encontradas em (WINNE, 2017).

A próxima seção falará sobre modelos Markovianos e suas noções básicas. Elas são necessárias para um melhor entendimento do trabalho e de como eles podem se relacionar com o jogo de Truco apresentado nesta seção.

3.2 MODELOS MARKOVIANOS

Um modelo Markoviano (STEWART, 2009) é um tipo especial de processo estocástico e pode ser aplicado a praticamente qualquer sistema, como visto em (OKABE; SAITO; NAKAJIMA, 2005; THURAU; HETTENHAUSEN; BAUCKHAGE, 2006; KRATZ; SMITH; LEE, 2007; ZHOU; HUANG; WANG, 2004). Ele representa os estados possíveis do sistema e a probabilidade de um estado mudar para outro. Essa mudança ocorre quando uma força atua sobre o modelo, sendo geralmente uma ação, e essa variável pode ser discreta ou contínua. O sistema só pode ocupar um estado por vez e a transição tem custo de tempo zero, ocorrendo de forma instantânea.

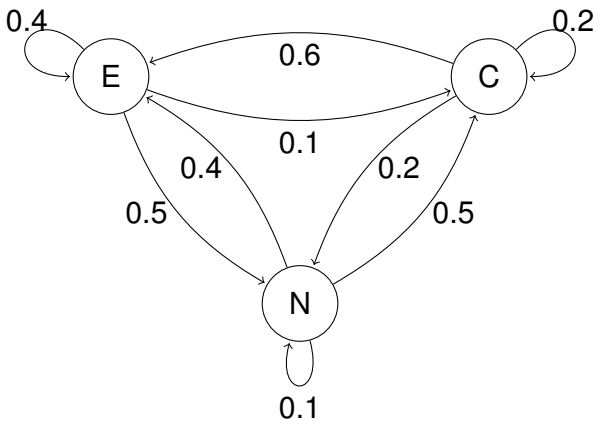
Sendo um processo estocástico, o Markoviano também é representado por variáveis aleatórias $\{x(t), t \in T\}$, sendo $x(t)$ uma variável aleatória e definida dentro de um espaço de probabilidade em um tempo t. O parâmetro t representa o tempo e, assim, $x(t)$ representa o valor dessa variável em determinado tempo t. T é o conjunto onde se encontra t, podendo ser $T = \{0, 1, 2, 3, ...\}$ quando o processo é discreto, e $T = \{t : 0 < t < +\infty\}$ quando o processo é contínuo.

Podemos dizer que um modelo genérico é Markoviano se ele satisfaz a propriedade de Markov. Essa propriedade diz que a transição entre estados só pode ocorrer levando em consideração o estado atual, sem a interferência dos estados ocupados anteriormente. Assim, o modelo evolui e assume diferentes valores ao longo do processo e, uma vez que ele muda muito pouco ou nada, de acordo com o limite de precisão estabelecido, ele chegou em seu estado estacionário.

Os estados de uma Cadeia de Markov de Tempo Discreto, como a utilizada nesse trabalho, são representados por um conjunto de variáveis $x_0, x_1, x_2, ..., x_n$ nos tempos 0, 1, 2, ..., n, definindo o conjunto temporal T. O conjunto de probabilidades que leva um estado x_n até todos os outros estados em um tempo t tem soma total correspondente a 1. Normalmente, esse modelo é representado por meio de um grafo, onde os vértices representam o conjunto de variáveis e as arestas representam as probabilidades de um estado ir para outro estado. Essa mesma informação pode ser representada em formato matricial, onde as linhas e colunas representam os estados e cada intersecção representa sua respectiva probabilidade.

A Figura 3.1 mostra um exemplo simples de uma Cadeia de Markov. Nela há três estados e eles representam três possibilidades de tempo, *Ensolarado*, *Nublado* e *Chuvoso*. A possibilidade do tempo passar de Nublado para Ensolarado é de 40%, de Nublado para Chuvoso é de 50%, e de permanecer Nublado é de 10%, e assim cada estado tem a sua probabilidade de passar para o outro estado, ou manter-se no estado atual. Dado um dia, as probabilidades diretas de passar de um estado para outro são as mesmas probabilidades presentes nas transições. Para dois dias ou mais, é preciso usar a equação de Chapman Kolmogorov.

Figura 3.1 – Exemplo simples de Cadeia de Markov



Fonte: Próprio autor.

A Cadeia apresentada na Figura 3.1 pode ser representada também pela matriz de probabilidades da Tabela 3.3. Desse modo, se o tempo estiver Nublado, a probabilidade de evoluir (t) para Chuvoso é de 50%. No próximo dia (t+2), é de 19%. Depois de alguns dias a probabilidade se mantém estável em 24%. Isso é útil no jogo de Truco tendo em vista que, por exemplo, ele tem três turno e na 3ª mão é necessário considerar as probabilidades das duas mãos anteriores.

Tabela 3.3 – Tabela representativa da Figura 3.1

	N	C	E
N	0.1	0.5	0.4
C	0.2	0.2	0.6
E	0.5	0.1	0.4

Fonte: Próprio autor.

A seguinte seção apresentará os conceitos de aprendizagem por reforço, a mesma utilizada pelo agente. Ela alimentará as matrizes da Cadeia de Markov, fazendo com que o agente aprenda a partir de seus jogos e isso reflita na sua base de dados.

3.3 APRENDIZAGEM POR REFORÇO

Para um humano aprender a jogar qualquer tipo de jogo, além de aprender as suas regras, ele precisa jogar diversas vezes para entender sua dinâmica. Do mesmo modo, a aprendizagem por reforço visa repetir o mesmo cenário ou um cenário parecido para reforçar as características desse cenário na aprendizagem. Buscando melhorar o desempenho,

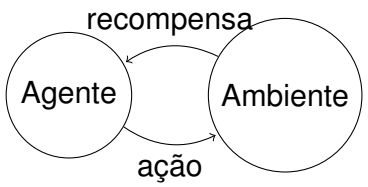
o modelo passa por diversas ações e experiências para fixar um ideal. Com isso, padrões acabam se formando, mostrando um caminho que leva ao resultado esperado.

Com o tempo, é desejável que o agente saiba qual atitude tomar em determinada situação, baseado no tempo e na repetição de experimentos. Além disso, ele também precisa relacionar as ações que geram maior recompensa em cada situação que lhe é apresentada e evitar as que não geraram tanta recompensa ou geraram penalidades. Esse processo de reforço continua até que o agente seja capaz de formular a sua própria opinião para aqueles casos em que foi treinado.

Jogos digitais apresentam uma oportunidade exploratória para o aprendizado por reforço, visto que a possibilidade de realizar treinos e testes é ampla e de fácil alcance, precisando de quatro (XU; SHI; WANG, 2018) variáveis: o espaço de estados, o espaço de ações, a função de recompensa e o modelo de aprendizagem. Esse modelo de aprendizado é amplamente utilizado (VINYALS et al., 2017; ANDERSEN; GOODWIN; GRANMO, 2017; XU; SHI; WANG, 2018) e apresenta bons resultados

Na Figura 3.2, é possível ver como o agente e o ambiente interagem. Esse processo é repetido diversas vezes até alcançar um estado onde a aprendizagem já não é mais necessária.

Figura 3.2 – Modelo de aprendizagem por reforço



Fonte: Próprio autor.

O próximo capítulo aborda a modelagem do agente e todo o seu desenvolvimento. Ele explora e utiliza os conceitos descritos neste capítulo, sendo eles a modelagem Markoviana e a aprendizagem por reforço, bem como o jogo de Truco.

4 MODELAGEM

Ao longo do jogo, é possível perceber que há diferentes modos que são dependentes entre si e fazem o jogo fluir. Eles são responsáveis por agregar mais pontos ao placar final dos jogadores e controlam:

- Truco: A decisão de chamar, aceitar ou negar uma aposta. Além disso, dependendo da pontuação, pode levar a um aumento ou uma desistência. Para efeitos de representação, aqui descrito como Γ.
- Jogar Carta: A ordem das cartas a serem jogadas em cada rodada. Por exemplo, um jogador com $4*, 4*, 1\spadesuit$ nunca deve jogar dois $4s$ seguidos. Aqui será representado como Ξ.
- Envido: A decisão de chamar, aceitar, negar, aumentar ou desistir da aposta. Representado neste trabalho como Φ.
- Flor: Similar a Φ, exceto que os pontos da Flor são baseados em três cartas e o jogador que tem a Flor sempre irá chamá-la. Será descrito neste trabalho como Ψ.

A decisão de separar a modelagem do jogo em quatro modos diferentes veio do objetivo de criar uma representação eficiente em termos de armazenamento e memória. Desse modo também é possível controlar de forma mais eficiente a posterior aprendizagem, pois a modularização do modelo faz com que, por exemplo, o erro de um não influencie o desempenho de outro, tornando também mais simples qualquer ajuste ou melhoria.

Cada Cadeia de Markov foi adaptada e implementada como duas matrizes, sendo que uma contém o número de vitórias e a outra o número de derrotas. Desse modo, a probabilidade também foi dividida em duas. Essa divisão foi feita porque uma variável de vitória não seria suficiente, sendo necessário também saber a quantidade de derrotas que as jogadas obtiveram. Isso porque a proporção pode variar bastante, visto que a quantidade de partidas jogadas com uma configuração de cartas não é a mesma para todas. Como uma estratégia de início, essas matrizes foram pré-preenchidas levando em consideração a probabilidade direta de vitória e derrota de acordo com a força das cartas. Isso foi feito visando principalmente a otimização dos testes, visto que seriam necessárias muitas partidas para preencher as matrizes com pelo menos um jogo por combinação, chegando ao máximo de 252004 partidas na maior matriz.

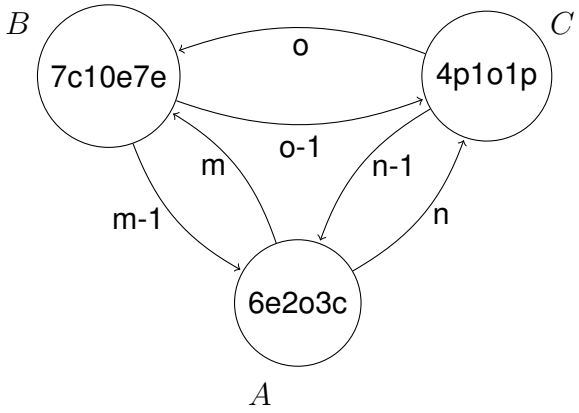
As matrizes são dispostas de forma que as linhas representam a configuração de cartas do atual jogador e as colunas representam o oponente. Cada intersecção linha-coluna representa a chance de vitória ou derrota do conjunto de cartas do jogador atual perante as cartas do oponente. O valor mais alto representa a vitória ou derrota garantida

e esse valor diminui conforme a chance de vencer ou perder também é reduzida, chegando a 1, que representa a derrota ou vitória certeira. A utilização de números inteiros ao invés de probabilidade se deve ao fato de que seria muito custoso atualizar as matrizes sempre que uma partida fosse jogada. A atualização percorreria posições que seriam pouco usadas, devido a aleatoriedade do embaralhamento das cartas, se tornando uma operação desnecessária naquele momento. Modificar somente os focos e calcular as probabilidades só quando requisitadas reduz o número de operações, gerando melhor desempenho.

Todas as representações de Cadeia de Markov são baseadas e atualizadas através de aprendizado por reforço, fazendo com que o modelo aprenda através do treino e repetição de jogadas. Assim, o agente joga partidas contra si mesmo, onde os dois tem a mesma base de dados.

A Figura 4.1 mostra um breve modelo que será expandido posteriormente. Nela é feita uma representação de três conjuntos de cartas que refletem possíveis mãos dos jogadores e suas chances de vitória em um provável confronto direto com as outras. Visto que todo estado tem chance contra os outros dois, baseado na Tabela 3.2, as probabilidades de vitória são praticamente iguais, sofrendo pouca variação. Essa semelhança se deve a alternância que as cartas podem sofrer na ordem do jogo, modificando o seu resultado final. Assim, o estado A tem uma chance m de vencer o estado B e esse tem $m - 1$ de vencer o estado A. Já para o estado C, A tem uma probabilidade n de vencê-lo e $n - 1$ de perder para ele. Por fim, o estado C tem uma chance o sobre o estado B e este tem uma chance $o - 1$ sobre C.

Figura 4.1 – Modelo geral de Markov adaptada para o jogo Truco



Fonte: Próprio autor.

A implementação do modelo foi feita no formato matricial e separando o número de vitórias do número de derrotas, sendo simples e de fácil controle. Cada atualização das matrizes representa a 10% do valor máximo inicial de cada uma, ou seja, o maior número de vitórias/derrotas da fase de pré-preenchimento para cada matriz. Desse modo, os avanços seguem sempre o mesmo padrão de atualização. Isso impede que somente

uma configuração se destaque, fazendo com que todas cresçam na mesma proporção. Além disso, a cada vez que um oponente blefa com um mesmo jogo, a chance do bot aceitar cai em 10% do total.

A seguinte seção apresentará os espaços de estados da modelagem. Cada modo do jogo utiliza um destes espaços de estados em sua implementação de acordo com a necessidade de cada um. Eles refletem a quantidade mínima de estados necessários sem que informações sejam perdidas ou duplicadas.

4.1 ESPAÇO DE ESTADOS

O espaço de estados compreende todos os estados possíveis para o contexto de cada modo de jogo. Neste trabalho, os estados de cada espaço podem ser as combinações de cartas, os pontos ou a força das cartas. Os modos, que serão apresentados posteriormente, tem as suas necessidades e cada um ocupa um dos espaços descritos nesta seção.

4.1.1 Modo de Cartas (Truco)

Levando em consideração que o baralho utilizado no jogo tem 40 cartas, o número de combinações dessas 40 cartas dispostas 3 a 3 é de $C(40,3) = \frac{40!}{(40-3)!*3!} = 9880$ estados. Como exceção, conforme visto na Seção 3.1, algumas cartas tem a mesma força independente do naipe, sendo elas as de número 2, 3, 4, 5, 6, 10, 11, 12. Assim, por exemplo, a carta 5 de copas tem a mesma força que a carta 5 de espada. As cartas de número 1 e 7 tem três diferenças com relação ao naipe, sendo que dois deles tem forças iguais. Assim, temos 14 níveis e $C(14,3) = \frac{14!}{(14-3)!*3!} = 364$ estados.

Contudo, nesse caso, as repetições precisam ser contadas (e.g. 2c, 2e, e 2o é uma combinação válida), resultando em $C(14,3) = \frac{(14+3-1)!}{(14-1)!*3!} = 560$ estados. Porém, algumas não são válidas devido ao valor único de algumas cartas[1]. Combinações que contém duas cartas únicas são retiradas desse total, sendo elas um total de 56. Exemplificando, essas combinações são as que contém a mesma carta única duas vezes e qualquer outra carta junto, incluindo ela mesma, retirando também combinações que contem três vezes a mesma carta única. Além disso, é necessário retirar combinações em que há três cartas 1 e três cartas 7 não identificadas pelo naipe, visto que elas só são duas e uma terceira torna a mão inválida. Por fim, o número de combinações inválidas totaliza 58, o que faz com que o número de possíveis estados seja $560 - 58 = 502$.

[1] Valores únicos são: Ás de espadas, Ás de copas, Sete de espadas e Sete de ouro.

4.1.2 Modo de Pontos (Envido e Flor)

Se levássemos em consideração somente a pontuação final do Envido, teríamos 13 estados, indo de 20 a 33 pontos, duas cartas de pontuação 0 e as duas cartas mais valiosas, respectivamente. No entanto, as cartas envolvidas na jogada podem fazer diferença, visto que quem é pé já viu a carta jogada pelo adversário, e essa pode fazer parte da sua combinação de envido. Portanto, levando em consideração as cartas envolvidas e não somente a pontuação final, o número de estados passa a ser $C(8,2) = \frac{8!}{(8-2)!*2!} = 28$, mais um equivalente a pontuação de duas cartas que tem pontuação 0, totalizando 29 estados.

Já na Flor, teríamos 18 estados indo de 20 a 38 pontos. Porém, pelo mesmo motivo apresentado anteriormente, é preciso avaliar as três cartas da mão. Sendo assim, há $C(8,3) = \frac{8!}{(8-3)!*3!} = 56$ estados, mais 8 combinações em que as cartas de valor 0 estão presentes mais de uma vez, totalizando 64 estados.

4.1.3 Modo de Força (Jogar carta)

Para jogar uma carta, é preciso avaliar não só a sua força, mas também a ordem em que elas são jogadas, visto que uma mesma combinação pode perder e ganhar de outra somente alterando a ordem de jogada. Por exemplo, as combinações 4c, 5c, 6c e 4e, 5e, 6e podem variar entre derrota e vitória se a primeira carta jogada for um 4 ou um 6. Com isso em vista, as cartas foram separadas entre baixas, médias e altas, tendo seis diferentes possibilidades de organizar as jogadas. Uma matriz auxiliar, além das de vitória e derrota, é responsável por prever a possível classe em que a carta jogada pelo oponente se encaixa.

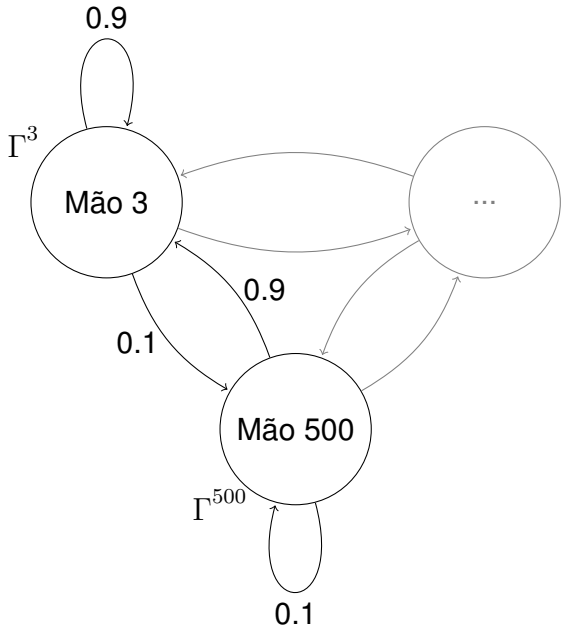
O padrão do modo de cartas não foi repetido no modo de Jogar Carta, pois não seria eficiente para este caso, visto que indica somente a probabilidade de ganhar e perder e jogo. O que falta é indicar a ordem em que as cartas devem ser jogadas para alcançar a vitória.

4.2 MODELO PARA O MODO TRUCO

Seguindo o espaço de estados no modo de Truco, a Figura 4.2 mostra a representação compacta do modelo implementado, o qual reflete o jogo. São mostrados em primeiro plano 2 estados, sendo eles Γ^3 e Γ^{500}, e em segundo plano um estado fictício, representando todos os restantes. Cada estado é composto pela junção de três cartas, sendo que o estado Γ^3 corresponde as cartas 4-4-6 e o estado Γ^{500} corresponde as cartas 7o-7e-1e. Cada Γ^i, sendo $1 \leq i \leq 502$, é um estado composto por 501 transições, indo de Γ^1 até

Γ^{502}. Cada transição contém um peso que é levado em consideração na hora de aceitar ou negar um pedido. Na cadeia de vitória, cada peso de Γ^j até Γ^k, sendo $1 \leq j \leq 502$ e $1 \leq k \leq 502$, é a quantidade de vitórias que o primeiro teve sobre o segundo. Já na cadeia de derrota, o peso de Γ^j até Γ^k representa o número de derrotas do primeiro sobre o segundo.

Figura 4.2 – Representação adaptada da Markov para o modo Truco



Fonte: Próprio autor.

Na Figura 4.2, as transições envolvendo o estado mais à direita representam as diversas transições que estes estados fazem com os outros 500. A junção destas é vista como o vetor $\Gamma_{3,j}$, para as transações que tem como origem o estado Γ^3 e como o vetor $\Gamma_{500,j}$ para as transações que saem do estado Γ^{500} apresentados na Tabela 4.1. Cada linha da tabela representa o estado atual, ou seja, a combinação de cartas do atual jogador. As colunas representam o próximo estado, ou seja, a mão do oponente, aquela que o jogador atual precisa vencer. Na prática, durante as execuções, a Markov é dinâmica e é feito um recorte da matriz, onde somente as mãos de interesse são levadas em consideração.

O modelo mostrado na Figura 4.2 é uma representação dos dados mostrados na Tabela 4.1. Sendo assim, a tabela é uma representação fiel dos dados utilizados nas fases de treino e teste. Pela tabela, é possível observar que qualquer $\Gamma_{i,j}$ é maior ou igual a um $\Gamma_{k,j}$, sendo $i > k$. Portando, utilizando os estado da Figura 4.2, $\Gamma_{3,j} < \Gamma_{500,j}$.

As reais probabilidades são calculadas a partir dos possíveis estados que a mão do oponente pode ter. Por exemplo, se o oponente só tem uma carta jogada, são selecionadas todas as combinações de cartas que ele pode ter. Se ele jogou um 4, são separadas todas as combinações que tem um 4 na sua composição, como 4-4-4, 4-5-6, 4-4-7, 4-2-3.

Tabela 4.1 – Matriz de contagem representando parte da Markov para o modo Truco

$\Gamma_{i,j}$	$\Gamma_{i,1}$	$\Gamma_{i,2}$	$\Gamma_{i,3}$	...	$\Gamma_{i,500}$	$\Gamma_{i,501}$	$\Gamma_{i,502}$
$\Gamma_{1,j}$	37	36	35	...	3	2	1
$\Gamma_{2,j}$	38	37	36	...	4	3	2
$\Gamma_{3,j}$	39	38	37	...	5	4	3
...	...	...	...	...	...	...	...
$\Gamma_{500,j}$	71	70	69	...	37	36	35
$\Gamma_{501,j}$	72	71	70	...	38	37	36
$\Gamma_{502,j}$	73	72	71	...	39	38	37

Fonte: Próprio autor.

Após, todas as vitórias e derrotas para esse grupo são contabilizadas separadamente. Em seguida, o cálculo da probabilidade é feito como $\frac{v}{v+d}$ e $\frac{d}{v+d}$, sendo v o número de vitórias e d o número de derrotas. Seguindo o exemplo, sendo $v = 5$ e $d = 69$, a probabilidade de vitória partindo do estado Γ^3 é de $\frac{v}{v+d} = \frac{5}{5+69} = 0.1$ e a de derrota $\frac{d}{v+d} = \frac{69}{5+69} = 0.9$. Já partindo do estado Γ^{500}, a probabilidade de vitória é de $\frac{v}{v+d} = \frac{69}{69+5} = 0.9$ e a de derrota é de $\frac{d}{v+d} = \frac{5}{69+5} = 0.1$. Depois disso, é feita uma conferência da probabilidade de vitória, sendo que ela deve ser maior do que a mínima estipulada. Por fim, uma "roleta" de probabilidades é executada, onde a vitória e a derrota podem ser sorteadas. Se for vitória, o jogador chama/aceita o pedido de Truco, se for derrota, ele não pede/nega o pedido.

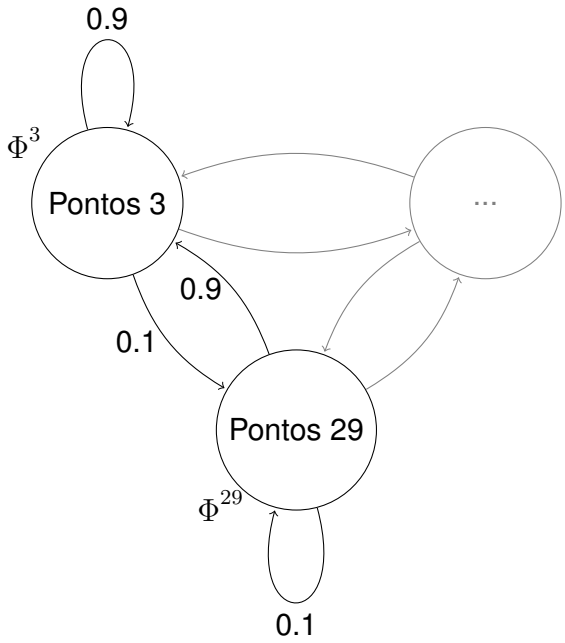
4.3 MODELO PARA O MODO ENVIDO

Conforme o espaço de estados do modo Envido apresentado anteriormente, a Figura 4.3 mostra uma reprodução simplificada da cadeia implementada. Na figura, são vistos dois estados principais, Φ^3 e Φ^{29}, e um estado fictício que representa os restantes. Neste caso, cada estado é composto pela junção da pontuação de duas cartas e aqui elas são independentes dos naipes. O estado Φ^3 é formado pelas cartas de pontuação 0[2] e 2 e o estado Φ^{29} pelas cartas 6 e 7. Cada Φ^i, sendo $1 \leq i \leq 29$, é um estado composto por 28 transições, indo de Φ^1 até Φ^{29}. Cada transição é representada por um peso, sendo este levado em consideração quando é preciso chamar, aceitar ou negar o pedido de Envido. Na matriz de vitória, cada transição de Φ^j até Φ^k, sendo $1 \leq j \leq 29$ e $1 \leq k \leq 29$, é a quantidade de vitórias que a primeira pontuação teve sobre a segunda. Já na matriz de derrota, o peso de Φ^j até Φ^k representa o número de derrotas da primeira sobre a segunda.

Na Figura 4.3, as transições que envolvem o estado mais à direita representam as demais transições dos dois estados principais com os outros 27 estados. A união destas

[2] Cartas de pontuação 0 no Envido são as cartas 10, 11 e 12.

Figura 4.3 – Representação adaptada da Markov para o modo Envido



Fonte: Próprio autor.

transições pode ser vista como o vetor $\Phi_{3,j}$, para as que vêm do estado Φ^3, e como o vetor $\Phi_{29,j}$ para as transações que tem como origem o estado Φ^{29}, conforme apresentado na Tabela 4.2. Cada linha da tabela representa o estado atual, ou seja, a combinação de cartas de Envido do atual jogador. As colunas representam o próximo estado, ou seja, a pontuação do oponente, aquela que o jogador atual precisa vencer. Na prática, durante as execuções, a Markov é dinâmica e é feito um recorte da matriz, onde somente as pontuações de interesse são levadas em consideração.

A cadeia mostrada na Figura 4.3 é uma representação dos dados mostrados na Tabela 4.2. Essa tabela é uma representação fiel dos dados utilizados nas fases de treino e teste, assim como no Truco. Pela tabela, é possível observar que qualquer $\Phi i, j$ é maior ou igual a um $\Phi_{k,j}$, sendo $i > k$. Portando, utilizando os estado da Figura 4.3, $\Phi_{3,j} < \Phi_{29,j}$.

As reais probabilidades são calculadas a partir dos possíveis estados que a mão do oponente pode ter. Por exemplo, se o jogador começa como pé e é o que pedirá Envido, ele já saberá uma carta do oponente e pode considerar esta como possível carta de Envido do adversário. Então, todas as pontuações que contém esta carta têm que ser consideradas. Se o oponente tiver jogado a carta 3, por exemplo, as opções 23, 35 e 37 são algumas das que serão analisadas. Após, todas as vitórias e derrotas para esse grupo são contabilizadas separadamente. Em seguida, o cálculo da probabilidade é feito igual mostrado no modo Truco, como $\frac{v}{v+d}$ e $\frac{d}{v+d}$, sendo v o número de vitórias e d o número de derrotas. Conforme o exemplo, sendo $v = 2$ e $d = 24$, a probabilidade de vitória partindo do estado Φ^3 é de $\frac{v}{v+d} = \frac{2}{2+24} = 0.1$ e a de derrota $\frac{d}{v+d} = \frac{24}{2+24} = 0.9$. Já partindo do estado

Tabela 4.2 – Matriz de contagem representando parte da Markov para o modo Envido

$\Phi_{i,j}$	$\Phi_{i,1}$	$\Phi_{i,2}$	$\Phi_{i,3}$	...	$\Phi_{i,27}$	$\Phi_{i,28}$	$\Phi_{i,29}$
$\Phi_{1,j}$	0	0	0	...	0	0	0
$\Phi_{2,j}$	14	13	12	...	3	2	1
$\Phi_{3,j}$	15	14	13	...	4	3	2
...	...	...	...	...	...	...	...
$\Phi_{27,j}$	24	23	22	...	13	12	11
$\Phi_{28,j}$	25	24	23	...	14	13	12
$\Phi_{29,j}$	26	25	24	...	15	14	13

Fonte: Próprio autor.

Φ^{29}, a probabilidade de vitória é de $\frac{v}{v+d} = \frac{24}{24+2} = 0.9$ e a de derrota é de $\frac{d}{v+d} = \frac{2}{24+2} = 0.1$. Depois disso, a probabilidade de vitória é analisada a fim de confirmar que está acima da probabilidade mínima estabelecida. Por fim, uma "roleta" de probabilidades é executada, onde a vitória e a derrota podem ser sorteadas. Se for vitória, o jogador chama/aceita o pedido de Envido, se for derrota, ele não pede/nega o pedido.

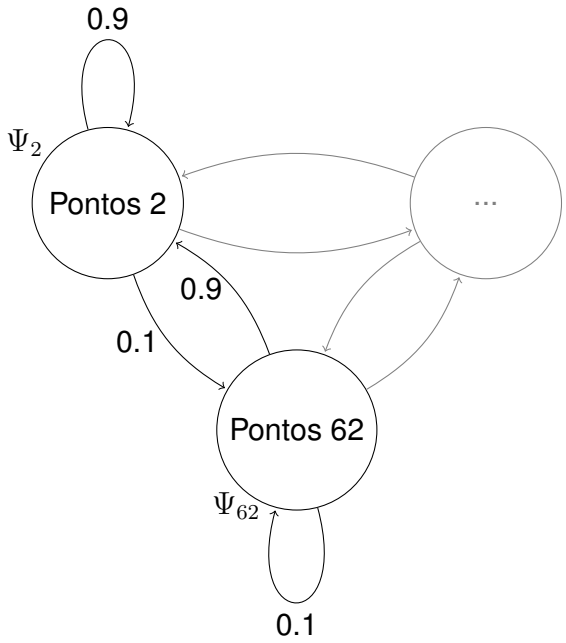
4.4 MODELO PARA O MODO FLOR

De acordo com o espaço de estados do modo Flor mostrado anteriormente, é apresentado na Figura 4.4 uma versão mais simples do modelo executado. Na figura, são observados dois estados principais, Ψ^2 e Ψ^{62}, e um estado em segundo plano representando os demais. A composição dos estados se dá da mesma maneira apresentada no modo Envido, porém com uma carta a mais. Assim, o estado Ψ^2 é formado pelas cartas de pontuação 0-0-2[2] e o estado Ψ^{62} pelas cartas 3-6-7. Cada Ψ^i, sendo $1 \leq i \leq 64$, é um estado composto por 63 transições, indo de Ψ^1 até Ψ^{64}. Cada transição é representada por um peso, sendo este levado em consideração quando é preciso chamar, aceitar ou negar o pedido de Contra Flor. As transições das matrizes de vitória e derrota acontecem da mesma forma que no modo Envido, sendo que as transições vão de 1 até 64.

Na Figura 4.4, as transições que abrangem o estado mais a direita mostram algumas das transições que os outros dois estados tem com os 61 estados restantes. A junção destas transições pode ser vista como o vetor $\Psi_{2,j}$, para as que vêm do estado Ψ^2, e como o vetor $\Psi_{62,j}$ para as transações que tem como origem o estado Ψ^{62}, como apresentado na Tabela 4.3. O funcionamento da tabela é idêntico ao apresentado no modo Envido e Truco.

A cadeia mostrada na Figura 4.4 é uma representação dos dados mostrados na Tabela 4.3. Pela tabela, é possível observar que qualquer $\Psi_{i,j}$ é maior ou igual a um $\Psi_{k,j}$, sendo $i > k$. Portando, utilizando os estados da Figura 4.4, $\Psi_{3,j} < \Psi_{62,j}$.

Figura 4.4 – Representação adaptada da Markov para o modo Flor



Fonte: Próprio autor.

Tabela 4.3 – Matriz de contagem representando parte da Markov para o modo Flor

$\Psi i, j$	$\Psi_{i,1}$	$\Psi_{i,2}$	$\Psi_{i,3}$	...	$\Psi_{i,62}$	$\Psi_{i,63}$	$\Psi_{i,64}$
$\Psi_{1,j}$	0	0	0	...	0	0	0
$\Psi_{2,j}$	19	18	17	...	3	2	1
$\Psi_{3,j}$	20	19	18	...	4	3	2
...	...	...	...	...	...	...	...
$\Psi_{62,j}$	34	33	32	...	18	17	16
$\Psi_{63,j}$	35	34	33	...	19	18	17
$\Psi_{64,j}$	36	35	34	...	20	19	18

Fonte: Próprio autor.

As reais probabilidades são calculadas a partir dos possíveis estados que a mão do oponente pode ter. Por exemplo, se o jogador começa como pé e é o que chamará a Flor, ele já saberá uma carta do oponente. Então, todas as pontuações com esta carta têm que ser consideradas a partir disso. Se o oponente tiver jogado a carta 3, por exemplo, as opções 023, 345 e 137 são algumas das que serão analisadas. Após, o cálculo da probabilidade é feito conforme apresentado nos modos anteriores. Seguindo o exemplo, sendo $v = 3$ e $d = 33$, a probabilidade de vitória partindo do estado Ψ^2 é de $\frac{v}{v+d} = \frac{3}{3+33} = 0.1$ e a de derrota $\frac{d}{v+d} = \frac{33}{3+33} = 0.9$. Já partindo do estado Ψ^{62}, a probabilidade de vitória é de $\frac{v}{v+d} = \frac{33}{33+3} = 0.9$ e a de derrota é de $\frac{d}{v+d} = \frac{3}{33+3} = 0.1$. Por fim, o algoritmo segue como é mostrado na seção do modo Envido.

5 EXPERIMENTOS E RESULTADOS

Nesse capítulo são apresentadas algumas informações sobre a implementação e funcionamento geral dos scripts, bem como um caso de jogo exemplificado. Também são apresentados os cenários e rodadas de testes e, por fim, é feita uma análise da evolução das matrizes nos diferentes experimentos.

5.1 IMPLEMENTAÇÃO

A implementação do agente foi feita na linguagem Python, fazendo operações de busca em arquivos de texto e arquivos JSON. Como ambiente de execução, foi utilizada a plataforma desenvolvida em (PAULUS, 2019), que simula o ambiente de jogo de dois jogadores. Essa plataforma foi implementada na linguagem Java, havendo a necessidade de ligar as duas linguagens. Para facilitar a comunicação, foi feita uma classe em Java que receber as requisições do jogo, como *chamarTruco* ou *aceitarTruco*. Para realizar o processamento da consulta, a classe com as informações do estado atual do jogo foi alterada para o formato JSON, a fim de preservar os dados no momento do envio e ser de fácil manipulação. Então, através da classe *Runtime* do Java, foi possível fazer a comunicação externa e executar o código em Python. Por fim, depois de executar o código do agente, a classe de transição recebe o resultado da requisição e retorna para a aplicação.

Primeiramente, o agente recebe o arquivo JSON enviado pela classe de transição e o converte para um dicionário, tendo assim uma busca fácil entre os elementos. Após leitura das matrizes, o algoritmo define a combinação ou pontuação do jogador atual e a do oponente, sempre mantendo a consistência entre as várias possibilidades do oponente e as cartas do jogador atual. Feito isso, são somados os números de vitórias e derrotas das configurações do jogador atual *versus* as possíveis combinações do adversário. Depois, é calculado o percentual através do valor destas duas variáveis e há uma verificação da porcentagem de ganho para garantir que ela seja maior do que o valor estipulado inicialmente, de acordo com cada pedido. Por fim, há uma roleta de probabilidades em que as opções são vitória e derrota, de acordo com os valores calculados anteriormente. Se o resultado dessa roleta for vitória, o valor retornado será *true*, significando que o agente deseja chamar ou aceitar a requisição, e se o resultado for derrota, o valor retornado será *false*, significando que o agente não deseja chamar ou aceitar a requisição.

Cada modo de jogo tem o seu próprio script, sendo o script do modo Envido apresentado no Apêndices A e os demais no GitHub da autora[1], e eles seguem um mesmo

[1] https://github.com/lanabr/TCC

padrão, para fins de organização e modularidade. Há também o script que faz a atualização das matrizes com os dados finais de cada rodada e cada partida. Ele extrai as matrizes dos modos que aconteceram naquela rodada e analisa o ganhador e o perdedor, separando os resultados entre as respectivas matrizes dos modos de jogo. Por exemplo, se não houve pedido de Contra Flor naquela rodada, não há necessidade de atualizar as matrizes. Cada modo é atualizado em 10% do valor máximo inicial de cada um, porém há situações em que não se tem conhecimento total das cartas do oponente. Isso acontece quando a partida acaba antes das três rodadas e as cartas não são mostradas completamente, tornando incompletos os dados de Truco, Jogar Carta, Flor e Envido. Desse modo, toda vez que isso acontece, o valor de 10% é divido entre as possibilidades daquele modo. Por exemplo, se só houve duas rodadas, há a terceira carta ausente para o modo Truco. Assim, todas as cartas possíveis são combinadas a essas duas cartas e a pontuação é dividida entre essas possibilidades.

As matrizes foram dispostas de modo que o sentido de crescimento fosse do mais fraco para o mais forte. Isso não se aplica as matrizes que correspondem ao fluxo normal de jogo, mas gerou mudanças nas que correspondem ao Envido, Flor e Truco. Nas duas primeiras, o espaço de estados corresponde a junção das cartas do respectivo pedido, resultando em junções como 0-2, 1-3, 3-6 e 5-7 para o Envido, e 0-4-5, 2-3-6, 3-5-7 e 4-6-7 para a Flor. A ordem, tanto nas colunas quanto nas linhas, reflete a soma dessa junção de cartas. Então, por exemplo, a combinação 1-2 está posicionada antes da 0-5, pois a sua soma é menor e por isso deve vir antes.

Já para o Truco é preciso considerar a força de cada carta presente na mão do jogador. Também é preciso balancear o peso de cada carta, visto que, se a mão for 4-4-1e, o 1 de espada não pode elevar demasiadamente a força do conjunto, uma vez que as outras duas cartas são as com a menor força do baralho. Portanto, uma solução foi atribuir um valor baixo e sequencial a cada carta e logo após aplicar a fórmula $\frac{f_1+f_2+f_3}{3}$, em que f_n representa a força da carta n, sendo que $\{1 \leq n \leq 3\}$ para encontrar a força da mão por rodada. A Tabela 5.1 apresenta esses valores para cada carta. Como exemplo, o conjunto de cartas 11-12-2 tem a força por rodada igual a $\frac{6+7+9}{3} = 7,33$. Do mesmo modo, a combinação 10-12-3 tem a mesma força por rodada de $\frac{5+7+10}{3} = 7,33$ da combinação anterior. Isso acontece porque as cartas estão balanceadas, da mesma maneira que o 10 é menos forte do que o 11, o 3 é mais forte do que o 2 e isso gera um equilíbrio. Sendo assim, a matriz de Truco foi disposta de acordo com esses valores de força e cresce da combinação mais fraca, 4-4-4, para a mais forte, 7e-1p-1e.

A seção a seguir apresenta um exemplo de jogo detalhado, conforme foi apresentado de forma breve anteriormente. Nele são explicados os principais caminhos e possíveis decisões do algoritmo, ilustrando o seu funcionamento também com auxilio de imagens.

Tabela 5.1 – Relação de força para o posicionamento das mãos na matriz de Truco

Carta	Valor
$1\spadesuit$	14
$1\clubsuit$	13
$7\spadesuit$	12
$7\diamond$	11
$3*$	10
$2*$	9
$1\diamond, 1\heartsuit$	8
$12*$	7
$11*$	6
$10*$	5
$7\heartsuit, 7\clubsuit$	4
$6*$	3
$5*$	2
$4*$	1

Fonte: Próprio autor.

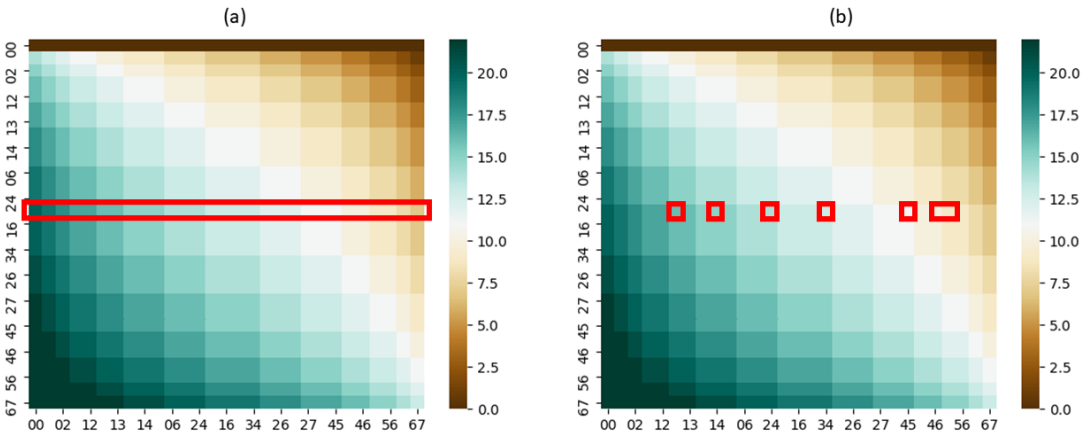
5.2 EXEMPLO DE JOGO

Como exemplo inicial, temos as cartas 7 de ouro, 12 de paus e 7 de paus para o jogador 1 e as cartas 7 de espada, 2 de espada e 4 de espada para o jogador 2. Considerando que o jogador 1 é mão e o jogador 2 é pé, o primeiro passo a ser dado é o pedido de Envido ou Flor por parte do jogador 1. O agente analisa primeiro se há alguma combinação de Envido em suas cartas. No caso apresentado, essa combinação existe com as cartas 12 de paus e 7 de paus, somando um total de 27 pontos. Depois disso, ele analisa a carta já jogada do oponente, se houver. Neste exemplo, o adversário é o pé da rodada, então não há nenhuma carta a ser analisada. Em um exemplo secundário, se fosse ao contrário, sendo o jogador 1 como pé e o jogador 2 como mão e o jogador 2 já tivesse jogado uma carta, ela seria verificada com o intuito de considerá-la como uma pista dos possíveis pontos de Envido do jogador. Essa seria a única informação disponível no momento e, assim, deve ser levada em consideração.

Continuando com o exemplo inicial, a combinação do jogador 1 é 0-7, formada pelas cartas 12 e 7, e para o jogador 2 são consideradas todas as 29 possibilidades de combinação de Envido, pois não se tem nenhuma pista de qual possa ser. Então, o conjunto de possíveis Envidos do jogador 2 é igual a {0-0 ... 0-7, 1-2 ... 1-7, 2-3 ... 2-7, 3-4 ... 3-7, 4-5 ... 4-7, 5-6, 5-7, 6-7}. A partir disso, são somados os valores da matriz de vitória para as intersecções entre a combinação do jogador 1 com cada uma das combinações do jogador 2, e o mesmo acontece para a matriz de derrota. Exemplificando, são somadas as posições de $\Phi_{13,0}$ até $\Phi_{13,28}$, como mostrado na Figura 5.1 (a). Em outra situação, se

uma carta já tivesse sido jogada, por exemplo a carta 4 de espada, o cenário de busca seria diferente, conforme representado na Figura 5.1 (b), onde somente algumas colunas seriam selecionadas. As somas de vitória e derrota para o cenário do exemplo inicial são iguais a 384 e 364, respectivamente.

Figura 5.1 – (a) Matriz de Envido onde é mostrada a linha onde estão os dados pesquisados pelo agente (b) Matriz de Envido onde são mostradas as células pesquisadas pelo agente



Fonte: Próprio autor.

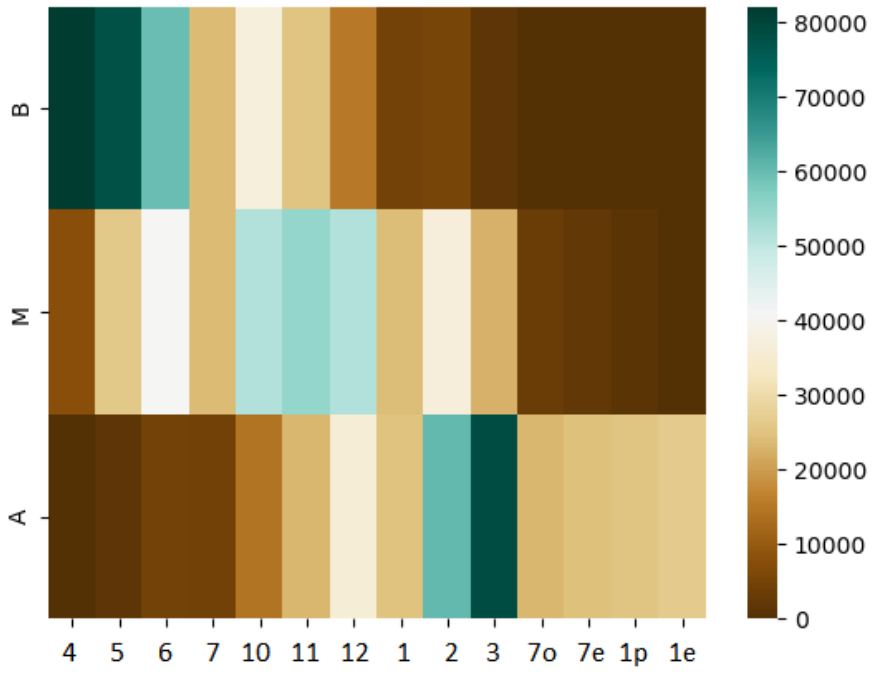
Após, é feito o cálculo da probabilidade pelas fórmulas $\frac{v}{v+d}$ e $\frac{d}{v+d}$, conforme já apresentadas anteriormente, resultando em $v = 0.5$ e $d = 0.5$. Então, é feita uma conferência da probabilidade de vitória, onde ela precisa ser maior do que o valor mínimo estipulado inicialmente, que é de 0.4. Se o índice de vitória for menor do que este valor, o agente automaticamente não aceita. No caso deste exemplo, ele é maior, então o algoritmo segue com a execução. Por fim, é feita uma roleta de probabilidades, onde uma lista é preenchida de acordo com os valores de derrota e vitória calculados anteriormente. Dessa lista é retirado aleatoriamente um valor entre G e P, Ganhar ou Perder. Se o valor for G, o agente faz o pedido de Envido, e se for P isso não acontece. Neste caso, será considerado que ele optou por aceitar e então, caso o oponente aceite, ele irá perder, pois sua pontuação é menor que a pontuação do adversário.

Neste momento podem haver os pedidos de Real Envido e Falta Envido, onde quem os faria seria o jogador 2 como contraproposta. O algoritmo segue do mesmo modo descrito anteriormente, sendo a única diferença o valor mínimo da porcentagem de vitória, que são de 0.6 para o Real Envido e e 0.8 para o Falta Envido. Esse aumento serve como uma confiança no pedido, pois é preciso ter, de alguma forma, uma certeza de que a chance de ganhar é alta o suficiente, pois mais pontos estão sendo disputados com esses pedidos. Se houver a possibilidade do pedido de Flor, ele seria feito no lugar do pedido de Envido e ocorreria da mesma maneira.

Seguindo com o jogo, o próximo passo é do jogador 1. O algoritmo leva em conta a ordem em que as cartas foram jogadas e o nível de cada uma. A ordem se divide em 1ª, 2ª e 3ª, e os níveis se dividem em Baixa, Média e Alta. Ao todo, são 6 possibilidades, sendo elas Baixa-Média-Alta, Baixa-Alta-Média, Média-Baixa-Alta, Média-Alta-Baixa, Alta-Baixa-Média e Alta-Média-Baixa. Então, para a primeira rodada, as possibilidades de jogo para o jogador 1 são as seis descritas anteriormente e, por consequência, a lista de combinações possíveis para o jogador 2 são as mesmas seis, já que não há carta jogada de nenhum jogador. Então, ambos jogadores tem o seu conjunto de opções como sendo {B-M-A, B-A-M, M-B-A, M-A-B, A-B-M, A-M-B}.

Agora, em um cenário onde fosse o jogo ao contrário e o jogador 2 fosse mão e já tivesse jogado uma carta, esta seria levada em consideração na hora de prever a sua ordem de jogo. Por exemplo, se essa primeira carta jogada fosse o 4 de espada, o agente anteciparia o nível dessa carta como baixo. Essa previsão é feita através de uma matriz que armazena as cartas e os níveis dessas a partir de dados de partidas já jogadas pelo agente. O nível da carta jogada pelo oponente será o nível em que ela mais se encaixou ao longo das partidas. Essa matriz auxiliar é mostrada na Figura 5.2 em seu estado final de aprendizagem, cujo processo de evolução é descrito na Seção 5.3 e a sua análise é feita na Seção 5.4. Com isso, as possíveis combinações do jogador começariam com a letra B, formando o conjunto {B-M-A, B-A-M}.

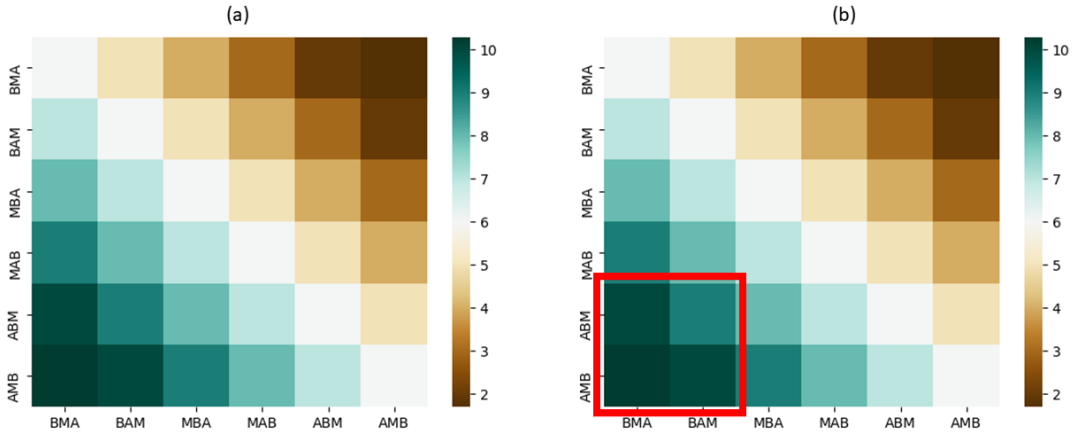
Figura 5.2 – Matriz de cartas e níveis usada para prever o nível da carta do oponente



Fonte: Próprio autor.

Após isso, seguindo com o exemplo inicial, as somas de vitórias e derrotas são colocadas em vetores. Cada combinação do jogador 1 ocupa um espaço neste vetor e a soma relacionada a ele é colocada na respectiva posição. Isso acontece nos vetores de vitória e derrota. Cada posição tem a soma da intersecção entre a combinação relativa do jogador 1 com as combinações do jogador 2. Como todas as combinações estão sendo consideradas em ambos os lados, toda a matriz mostrada na Figura 5.3 (a) será analisada.

Figura 5.3 – (a) Matriz de jogo a ser analisada na primeira rodada (b) Matriz de jogo a ser analisada na segunda rodada



Fonte: Próprio autor.

Então, para escolher qual carta jogar, a porcentagem de ganho é somada de acordo com a primeira carta de cada combinação do jogador 1, pois é a única jogada que importa no momento. Por exemplo, os valores de vitória e derrota das combinações B-M-A e B-A-M serão unidos em um só, já que ambos possuem a primeira carta sendo a Baixa. Após, é feito o cálculo da probabilidade pelas fórmulas $\frac{b}{b+m+a}$, $\frac{m}{b+m+a}$ e $\frac{a}{b+m+a}$, sendo que b representa o número de vitórias quando a primeira carta foi a baixa, o m representa da mesma forma para a carta média e o a representa a carta alta. Neste caso, o valor para a carta baixa será 0.1, para a carta média será 0.1 e para a carta alta será 0.8. Por fim, o valor referente a cada probabilidade é distribuído em uma lista e ela passará a conter as letras B, M e A, representando as divisões de nível Baixa, Média e Alta, respectivamente. Um desses valores será retirado aleatoriamente da lista e equivale a carta a ser jogada pelo jogador 1, sendo que B equivale a sua carta baixa, M a sua carta média e A a sua carta alta.

Nesta primeira rodada, o ganhador 1 jogará a sua carta alta, o 7 de ouro, e o jogador 2 jogará a carta 4 de espada. Assim, o jogador 1 ganhou a rodada, visto que, pela Tabela 3.2, a sua carta é mais forte que a do oponente. Na próxima rodada, o mesmo processo ocorre novamente, mas com algumas adaptações. Na hora da formação das combinações do jogador 1, precisa-se considerar a carta já jogada por ele. Então, sabendo que a carta

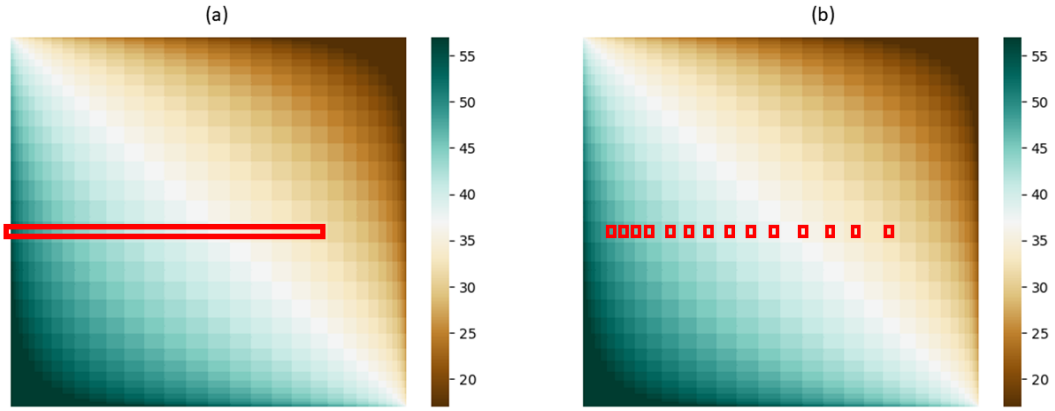
jogada foi a alta, as opções que restam são {A-B-M, A-M-B}. A Figura 5.3 (b) mostra a área que será analisada nesta rodada. Já para a combinação do jogador 2, a sua carta jogada na primeira rodada agora é levada em consideração e o número de possibilidades também é diminuído. Analisando essa carta, as combinações diminuem para duas, sendo elas B-M-A e B-A-M, visto que o agente infere que a carta 4 de espada jogada por ele é a sua carta baixa. Na terceira rodada, esse processo não é mais preciso, pois só há uma carta para jogar e ela será lançada a mesa inevitavelmente.

O pedido de Truco pode ser feito a qualquer momento e em qualquer rodada. Para analisar se deve pedir ou não pedir, aceitar ou não aceitar, primeiramente o agente analisa quais são as suas chances. Se a requisição for feita na terceira rodada, ele verificará se perdeu ou empatou as duas primeiras. Se for algum desses casos, ele não pede ou não aceita, já que estará entregando pontos ao adversário, pois não terá a mínima chance de ganhar esses pontos que está apostando. Já se o pedido tiver sido feito na primeira ou segunda rodada, não há esse impedimento e o algoritmo segue. Então, supondo que o pedido tivesse sido feito na segunda rodada pelo jogador 1, as combinações seriam feitas resultando em 7p-12-7o para o jogador 1 e em uma lista para o jogador 2. Alguns exemplos da lista de combinações do jogador 2 são {4-4-4, 4-4-11, 4-10-10, 4-6-7c, 4-6-1o, 4-7p-3, 4-11-2, 4-1o-1c, 4-12-1e, 4-2-1e, 4-1p-1e}. Esta lista relaciona a carta já jogada pelo jogador 2 com todas as outras disponíveis, excluindo as cartas únicas se elas estiverem na mão do jogador 1.

Após, são somados os números de vitórias e derrotas de cada combinação de jogador 1 e jogador 2, ou seja, todas as combinações $\Gamma_{270,i}$, sendo que i são as combinações do jogador 2 apresentadas anteriormente (uma representação da área das combinações dita anteriormente está na Figura 5.4 (a)). Em seguida, utilizando as fórmulas $\frac{v}{v+d}$ e $\frac{d}{v+d}$ já apresentadas anteriormente, é feito o cálculo da porcentagem de vitórias e derrotas, resultando em 0.6 para vitória e 0.4 para derrota. Depois disso, é feita uma conferência da porcentagem de vitória e ela tem que ser maior do que o valor definido inicialmente, de 0.5. Por fim, uma lista é preenchida de acordo com a porcentagem de vitórias e derrotas. Essa lista é composta por G e P, sendo Ganhos e Perdas, respectivamente. Um valor é retirado aleatoriamente da lista, sendo que, se esse valor retirado for G, o agente pede ou aceita a chamada de Truco, e se for P ele não pede ou não aceita o pedido. Essa chamada pode evoluir para ReTruco e Vale Quatro, sendo que a única diferença pro agente é o valor mínimo de vitória, que é de 0.7 para o ReTruco e 0.8 para o Vale Quatro.

Já em um outro cenário onde o jogador 1 pede Truco na terceira rodada, temos então que a lista de mãos possíveis do jogador 2 passa a ser {4-4-2, 4-5-2, 3-6-2, 4-7-2, 4-10-2, 4-11-2, 4-12-2, 4-1-2, 4-2-2, 4-2-3, 4-2-7o, 4-2-7e, 4-2-1p, 4-2-1e}, considerando que na segunda rodada ele jogou a carta 2 de espada (as cartas são dispostas na combinação de acordo com sua a força). A matriz com os pontos de busca diminui, tornando-a mais seletiva, como é mostrado na Figura 5.4 (b).

Figura 5.4 – (a) Matriz de Truco a ser analisada na segunda rodada (b) Matriz de Truco a ser analisada na terceira rodada



Fonte: Próprio autor.

A próxima seção apresenta os experimentos feitos e alguns resultados derivados deles. Foram realizadas três rodadas de testes com diferentes propósitos e configurações, sendo que duas delas geraram matrizes de aprendizagem distintas que são analisadas mais adiante. Também são apresentados os agentes de (PAULUS, 2019), usados para comparação e validação do presente trabalho.

5.3 EXPERIMENTOS

No total, 3 rodadas de testes serão descritas nessa seção, sendo utilizadas diferentes configurações no decorrer do processo. Inicialmente, foram feitos testes em que os dois jogadores da partida utilizavam a técnica apresentada neste trabalho, jogos do tipo *Markov x Markov*, e as cartas foram distribuídas de forma aleatória. Foram realizadas 5000 partidas e as matrizes resultantes da aprendizagem desses jogos foram salvas em média a cada 200 partidas. Cada conjunto de 200 jogos foi realizado em uma média de 6 horas. Mesmo parecendo uma quantidade muito pequena de partidas rodadas em tanto tempo, esses números podem ser ainda mais expressivos se comparados com os testes rodados posteriormente. As matrizes resultantes mostram a evolução do agente e como ele está tomando suas decisões. Com esse grupo de jogos iniciais, temos um conjunto de matrizes treinadas e que tem informações de 5000 partidas jogadas em um espaço de armazenamento muito pequeno e de fácil busca e manipulação. Esses testes ocorreram na plataforma citada na seção anterior, assim como todos os testes apresentados nesta seção.

Nas próximas rodadas de testes serão utilizados os quatro melhores agentes desenvolvidos em (PAULUS, 2019). Eles foram enumerados resumidamente como 1º, 2º, 3º e 4º de acordo com a sua classificação final no respectivo trabalho. Também, Paulus trabalha com duas bases, onde uma delas é resultante do processo de aprendizado, chamada de Imitação, possuindo um total de 27515 casos. A outra, chamada de Baseline, abrange os casos coletados inicialmente, contendo 3195 mãos de Truco. Ambas as bases serão utilizadas em momentos diferentes, a fim de avaliar se a quantidade de casos disponíveis para o oponente afeta a jogabilidade.

De forma geral, os agentes de Paulus, ao receberem uma consulta, computam a similaridade entre o caso recebido e os grupos de casos da base utilizada. Depois de encontrar o grupo mais similar, é computada pela segunda vez a similaridade, mas desta vez são usados os casos pertencentes ao grupo identificado anteriormente. Após isso, são retornados os casos mais similares e as decisões tomadas nesses casos são reusadas através de um modelo de reuso em duas etapas. A identificação dos grupos é feita utilizando o algoritmo K-Means, a fim de reconhecer e explorar os diversos cenários de decisão presentes nos casos armazenados na base. No modelo de reuso em duas etapas, são aplicados dois critérios diferentes sobre os dados, sendo o primeiro sobre a escolha de um grupo de casos e o segundo na escolha da ação de jogo que deve ser reusada.

Sendo assim, o 1º bot, o qual teve o melhor desempenho, utiliza a união de dois critérios de reuso, a probabilidade de vitória e o número de pontos, respectivamente na escolha do cluster e das ações de jogo. Assim, é escolhida a ação de jogo que gerou maior ganho nas mãos dos casos que estão inseridos no grupo que possui a maior probabilidade de vitória, o que gera uma segurança a mais nas suas decisões. A grande quantidade de casos na base mais evoluída possibilitou o crescimento do desempenho deste agente, proporcionando assim o maior número de vitórias em relação aos oponentes. Este agente obteve um maior número de vitórias em jogos contra agentes considerados conservadores, com ações de jogo mais cautelosas, dando prioridade às jogadas consideradas seguras.

O 2º bot, que teve o segundo melhor desempenho, utilizou o critério de reuso de número de pontos na escolha de suas ações de jogo. Este agente não considera os agrupamentos de casos na escolha da sua jogada. Essas características fazem com que o agente tenha uma postura bastante agressiva. A utilização de um *threshold* de similaridade mais alto tende a diminuir o número de casos recuperados que contém ações de jogo feitas em cenários diferentes de jogo. Um *threshold* de similaridade alto faz com que sejam recuperadas somente jogadas muito similares a uma dada consulta e, como o número de jogadas possíveis é muito alto, isso faz com que os casos recuperados diminuam. Em algumas situações, essa técnica faz com que os resultados sejam melhores do que os do 1º bot. Porém, há outros contextos em que a política implementada sugere ações de jogo mais incertas, podendo não obter total sucesso.

O 3º bot, ficando com o terceiro melhor desempenho, escolheu os cluster e ações de jogo que lhe forneciam uma maior probabilidade de vitória. O comportamento do agente tende a ser mais conservador do que os agentes descritos anteriormente, que possuem características mais agressivas. O aumento de casos na base proporcionou benefícios para esse agente, assim como para o 1º agente. Isso se deve provavelmente a utilização de um critério de probabilidade, visto que um número maior de amostras gera um aumento na precisão das escolhas para esse método. Mesmo com a implementação de uma política considerada conservadora, o agente alcançou a terceira maior diferença de pontuações em vitória.

Por último, o 4º bot, com o quarto melhor desempenho, utilizou o critério de probabilidade de vitória diretamente na escolha das suas ações de jogo. Isso o tornou também bastante conservador, como o 3º agente, e, além disso, ele leva em conta ações de jogo empregadas em estados provavelmente diferentes. Com o aumento de casos na base, o agente teve uma melhora em seu desempenho, pelo mesmo motivo do 3º agente, visto que ambos utilizam o critério de probabilidade de vitória. O mesmo critério utilizado neste agente foi implementado também pelo 1º, mas quando usada sozinha não obteve o mesmo êxito.

A segunda rodada de testes engloba partidas onde as configurações também foram aleatórias. Neste contexto, foram realizados 2000 jogos, sendo 500 com cada um dos 4 bots. Aqui, as matrizes foram salvas a cada 100 partidas e cada conjunto de 25 partidas durou em média 6 horas e meia. Comparando com a rodada de testes anterior, em que não haviam os jogadores de Paulus, essa rodada foi mais de 8x mais lenta. O agente que utilizava a técnica descrita neste trabalho utilizou as matrizes iniciais e a base de casos dos quatro melhores agentes foi a Imitação, sendo essa a resultante do processo de aprendizado.

Jogando contra o agente mais forte dentre os quatro, o bot que utiliza a técnica apresentada neste trabalho venceu 11,40% das partidas jogadas, como mostrado na Tabela 5.2. Dentre estes, mais de 50% das vitórias aconteceram quando ele começou como pé na partida, ou seja, ele jogou a sua primeira carta com base na carta do oponente. Isso se deve ao fato de que, quando se é o pé da rodada, a carta do adversário já está na mesa, ou seja, a sua decisão é baseada na carta que o oponente já jogou. Assim, é possível escolher entre a carta que vence a carta já jogada ou a que vai de acordo com a estratégia do jogador. A ação pode ter várias reações, podendo também levar a derrota no jogo, sendo que, do total de 100% de derrotas, as que se deram com o jogador sendo pé representa o equivalente a 48,31%.

Contra o 2º bot, a porcentagem de vitória foi de 7,00%, sendo consideravelmente menor em relação ao primeiro bot. O índice de vitória na posição de pé também diminuiu, mas por outro lado, a porcentagem de vitória sendo mão subiu. Já contra o 3º bot, a porcentagem de vitória foi de 10,80%, se mantendo entre o 1º e 2º agentes. Por fim, contra

Tabela 5.2 – Vitórias da técnica adaptada de Markov contra os melhores agentes de (PAULUS, 2019) com base treinada

Agentes	Vitórias totais	Vitórias sendo mão	Vitórias sendo pé
1º	11,40%	36,84%	63,16%
2º	7,00%	48,57%	51,43%
3º	10,80%	44,44%	55,56%
4º	17,00%	57,65%	42,35%

Fonte: Próprio autor.

o 4º bot, o percentual de vitória foi de 17,00%, o maior entre os quatro agentes. Porém, o nível de vitória por posição se inverteu, sendo que a maior porcentagem foi com a posição de mão, com 57,65% das vitórias. Essa diferença provavelmente se deve ao fato de que a escolha do 4º bot, como pé, não foi a melhor possível, fazendo com que um agente com a mesma característica conservadora conseguisse tirar proveito dessa situação, assumindo uma postura agressiva em comparação às jogadas do 4º bot.

É possível perceber também que o percentual de vitórias contra o 2º agente foi o mais baixo. Isso mostra que o agente apresentado nesse trabalho não se adaptou ao estilo de jogo mais agressivo, característico do segundo agente. Também, seguindo a mesma linha do 2º agente, o 1º agente provocou uma porcentagem bem inferior de vitórias devido ao seu estilo de jogo, tirando proveito das jogadas mais seguras provenientes da implementação modelada. Por outro lado, o maior percentual de vitória deu-se com o 4º agente. Com isso, percebe-se que o agente apresentado neste trabalho teve um melhor desempenho jogando contra um bot mais conservador, que é provavelmente o seu próprio estilo de jogo.

É importante dizer que os bots de (PAULUS, 2019) não aprendem ao longo das partidas jogadas. A base utilizada não é alterada conforme os agentes ganham ou perder, sendo totalmente estática. A única aprendizagem feita é a do agente que teve a sua implementação descrita neste trabalho, atualizando as matrizes conforme as rodadas de testes e o propósito de cada uma.

Tão importante quanto o número total de vitórias, analisar o progresso dessas vitórias nos dá um parâmetro sobre a evolução do agente ao longo dos jogos. A Tabela 5.3 mostra essa evolução em um intervalo de 100 partidas contra cada agente. Abaixo são descritos alguns motivos para tal desempenho e o porquê esse resultado é esperado.

Contra o 1º agente, por exemplo, foram 12 partidas ganhas nos primeiros 100 jogos, havendo um aumento de 1 vitória na marca das 200 partidas. Agora, na marca das 300 partidas, houve um decréscimo no número de vitórias, reduzindo a 7 partidas. Isso provavelmente se deve ao fato de que as combinações de cartas dadas nessas 100 partidas ainda não tinham sido exploradas anteriormente, fazendo com que o agente acessasse

Tabela 5.3 – Evolução do número de vitórias nos testes com os melhores agentes de (PAULUS, 2019)

Agentes	100 partidas	200 partidas	300 partidas	400 partidas	500 partidas
1º	12	13	7	16	9
2º	10	6	2	10	7
3º	7	14	11	12	10
4º	21	19	15	16	14

Fonte: Próprio autor.

configurações de jogo ainda não atualizadas, com o preenchimento inicial, ficando em desvantagem. Na marca das 400 partidas, o número de vitórias aumentou consideravelmente, sendo o maior até então, mas decaiu nas 100 partidas seguintes, possivelmente pelo mesmo motivo anterior. Isso acontece porque a função de atualização só age na combinação exata da partida, não mudando as combinações semelhantes. Por exemplo, se a combinação de carta é 4c-7p-11o para o jogador atual e 5e-10c-3e para o oponente, a atualização só será feita nessa específica intersecção da matriz.

Contudo, há casos em que a atualização é feita em partes. Por exemplo, se neste caso só houvesse a informação das cartas 4c-7p-11o e 5e-10c, a atualização seria em todas as intersecções cuja combinação incluísse as cartas 4, 7 e 11 na linha e 5 e 10 na coluna. Com isso, haveriam 13 combinações nas colunas, o que totaliza também 13 intersecções. No entanto, essa atualização ainda é bem restrita e continua sendo pontual, não estendendo para as intersecções vizinhas.

Essa variação se manteve nas partidas com todos os quatro agentes, tendo altos e baixos em todas as marcações. Isso mostra que não foi específico de um só agente adversário, de sua característica agressiva ou conservadora. Porém, um comportamento diferente foi percebido com o 4º agente. O maior número de vitórias ocorreu nas primeiras 100 partidas, com 21 vitórias. Nos conjuntos de 100 partidas posteriores, o número de vitórias não alcançou esse mesmo valor, mas ainda assim ocorreram oscilações. Isso pode ter acontecido devido ao balanceamento das matrizes iniciais, sendo mais pertinente às jogadas do agente adversário.

Por fim, uma terceira rodada de testes foi realizada, utilizando a base de dados inicial em ambos os lados. Foram feitos 50 jogos com cada bot nas mesmas condições descritas nos testes anteriores. No caso dessa rodada de testes, cada conjunto de 25 jogos demorou cerca de 3 horas para a conclusão.

A Tabela 5.4 mostra os resultados dessa rodada de testes. Nela é possível ver que, ao contrario da segunda rodada de testes, o agente foi mais forte em jogos contra o 3º agente. Contudo, ele também tem um estilo de jogo conservador, não desviando do índice anterior. Entretanto, ao contrário do teste anterior, o 2º agente não foi o que levou a menos

vitórias, ficando atrás de outros dois agentes. Levando em consideração a característica de que o 1º bot tem vantagem sobre agentes com estilo de jogo conservador, é possível afirmar que a técnica descrita neste trabalho opta por ações de jogo conservadoras e seguras, tomando suas escolhas sem ser agressivo e arriscado.

Também, é possível notar que, neste caso, as bases utilizadas fizeram diferença no resultado final. A porcentagem de vitória aumentou para todos os agentes, considerando os dados da Tabela 5.2, dando destaque para as partidas jogadas contra o 3º agente, alcançando quase 50% de vitória. O 2º agente, ao contrário da segunda rodada de testes, proporcionou o segundo maior índice de vitória, que anteriormente era o índice mais baixo. Isso contraria a ideia de que o maior índice de vitória se dá contra jogadores que também compartilham a estratégia conservadora.

Tabela 5.4 – Vitórias da técnica adaptada de Markov contra os melhores agentes de (PAULUS, 2019), ambos com base inicial

Agentes	Vitórias totais
1º	14%
2º	32%
3º	42%
4º	24%

Fonte: Próprio autor.

A próxima seção faz uma análise das matrizes resultantes do aprendizado do agente, mostrando de forma visual suas mudanças. Da mesma forma, são feitas comparações entre as aprendizagens de ambas as rodadas de testes, levando em consideração também a quantidade de jogos necessárias para alcançar tal desempenho.

5.4 ANÁLISES E HEATMAPS

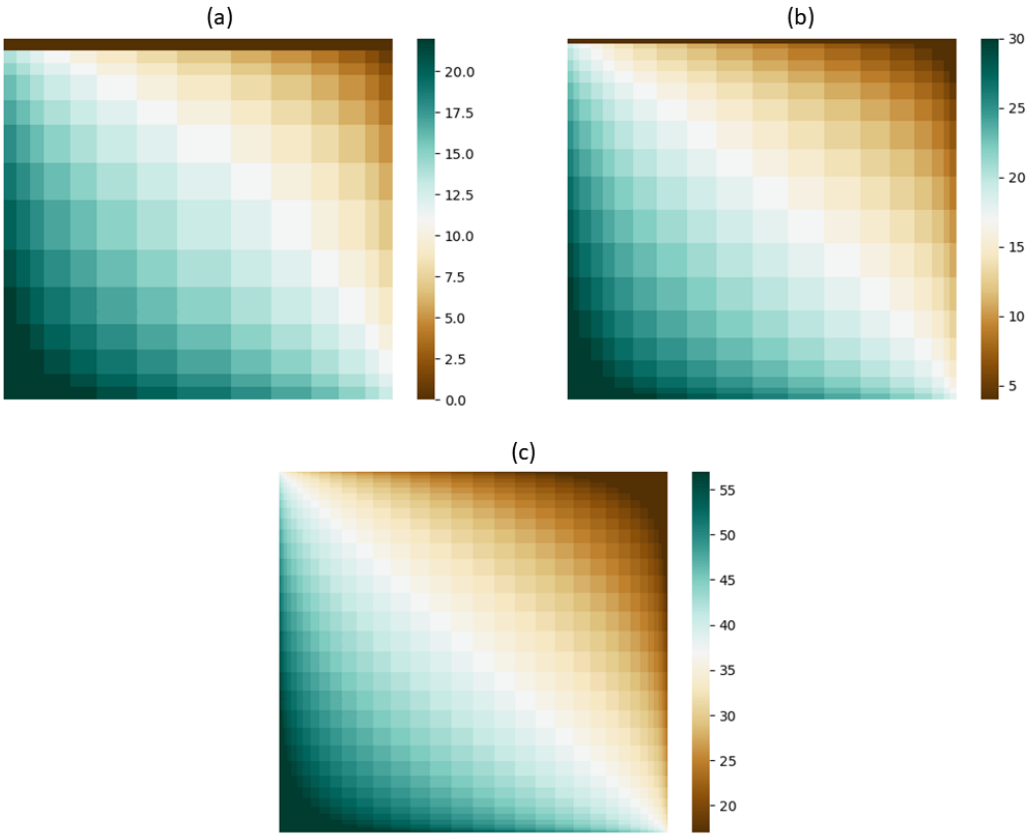
Durante a primeira rodada de testes, o agente aprendeu com as suas ações no jogo e, ao final das 5000 partidas, foi possível observar um conjunto de matrizes que mostravam essa evolução em todos os modos de jogo. Após, durante a segunda rodada de testes, o agente aprendeu com as ações de jogo do seu oponente. Ao final das 2000 partidas, as matrizes iniciais do agente que utilizou a técnica adaptada de Markov tomou quatro rumos diferentes. Cada conjunto de 500 jogos com cada um dos quatro bots evoluiu de maneira diferente as matrizes iniciais, tendo como reflexo o modo de jogo dos quatro bots mais vencedores de (PAULUS, 2019).

Inicialmente, a Figura 5.5 mostra as matrizes iniciais dos modos Envido (a), Flor (b) e Truco (b) em heatmaps. Essas representações seguem o mesmo padrão, onde o eixo

X se refere às combinações do oponente e o eixo Y se refere às combinações do jogador atual. Por exemplo, algumas combinações que aparecem nos eixos do modo Envido são 0-0, 1-3, 2-6 e 4-5, sendo que cada número se refere ao valor da carta que compõe a combinação de cartas do Envido. Isso se repete para o modo Flor, sendo que algumas das combinação são 0-1-2, 0-3-5, 1-4-6 e 3-5-6, contendo uma carta a mais em comparação ao Envido.

Já para o modo Truco, as combinação que aparecem nos eixos são, por exemplo, 4-7c-11, 6-12-2 e 3-7o-1e, sendo que cada número representa cada carta da mão do jogador. Elas estão dispostas na combinação de acordo com a sua força e, dessa maneira, tendo como exemplo a combinação 6-12-2, o 2 vem depois do 6 porque é mais forte. As letras que aparecem depois dos números 1 e 7 representam o naipe da carta, sendo que a letra "c" representa o naipe de copas, a letra "o" o naipe de ouro, a letra "e" o de espada e a letra "p" o naipe de paus. Isso acontece porque as cartas 1 e 7 diferem de força conforme os naipes e precisam ser identificadas por eles.

Figura 5.5 – Heatmaps iniciais dos modos (a) Envido, (b) Flor e (c) Truco

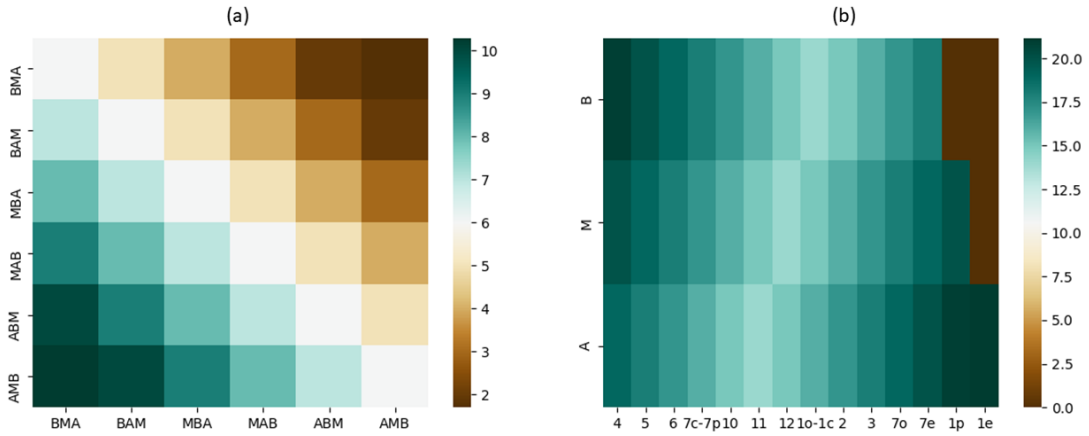


Fonte: Próprio autor.

A Figura 5.6 mostra a matriz inicial do modo Jogar Carta (a) e a matriz de quantidade (b). O eixo X da matriz do modo de Jogar Carta representa as combinação do oponente e o eixo Y representa as combinações do jogador atual. Essas combinações

levam em consideração a ordem em que as cartas são jogadas pelos jogadores e também o nível delas. Considerando que os níveis de uma carta podem ser Baixa, Média e Alta, as possíveis ordens de jogo são Baixa-Média-Alta (BMA), Baixa-Alta-Média (BAM), Média-Baixa-Alta (MBA), Média-Alta-Baixa (MAB), Alta-Baixa-Média (ABM) e Alta-Média-Baixa (AMB).

Figura 5.6 – Heatmaps iniciais do modo (a) Jogar Carta e (b) da matriz de quantidades



Fonte: Próprio autor.

Todos esses cinco heatmaps seguem o mesmo padrão de cor, sendo que, quanto mais forte o tom de marrom, menor é o número de vitórias e quando mais forte é o tom de azul, maior é o número de vitórias. É possível ter uma base desses valores altos e baixos de vitória pela legenda de cor na direita de cada heatmap. As combinações de menor força para ambos os jogadores está no canto superior esquerdo e as combinações de maior força estão no canto inferior direito.

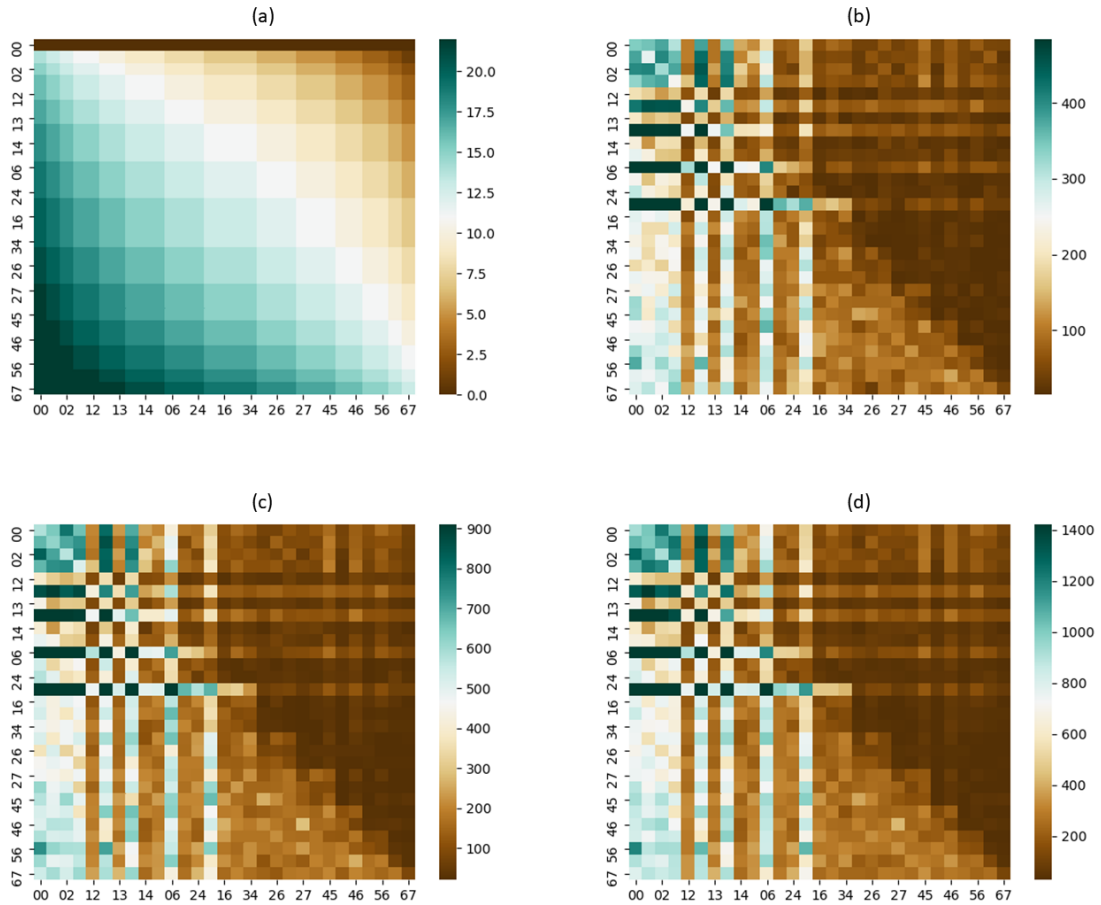
Do mesmo modo, o canto inferior esquerdo representa as vitórias mais certas e as mais óbvias, sendo que representa uma combinação de cartas muito forte do jogador atual e muito fraca do oponente. Já o canto superior direito reflete o contrário, ou seja, uma derrota certa. As combinações nesse canto são as mais fracas para o jogador atual e as mais fortes para o oponente. O meio dos heatmaps caracteriza combinações iguais para ambos os jogadores e a cor branca reflete a neutralidade de um confronto entre elas, onde qualquer um dos jogadores pode vencer a disputa.

A matriz de quantidades na Figura 5.6 (b) cruza os níveis que as cartas podem se encaixar com todas as cartas do baralho. A matriz difere do padrão das demais devido ao seu formato e o modo de preenchimento. O eixo X se refere às cartas do jogo, representadas pelo seu número e, no caso das cartas 1 e 7, pelo seu naipe também. O eixo Y se refere aos níveis que a carta pode ter, sendo eles Baixo, Médio e Alto. A matriz segue um padrão de valores maiores nas duas pontas, para evidenciar as cartas extremas jun-

tamente com o seu respectivo nível, e valores menores no intervalo central, distribuindo o nível médio entre a maioria das cartas.

A primeira rodada de testes resultou em 25 conjuntos de matrizes salvas. Ambos os modos são compreendidos por uma matriz de vitória e outra de derrota. Aqui serão apresentadas somente as matrizes de vitória, visto que elas são o alvo da análise. Alguns heatmaps correspondentes ao modo Envido são mostrados na Figura 5.7. É possível ver que há uma grande mudança já na imagem (b), que corresponde à marcação das 1800 partidas jogadas pelo agente. Em (b) é possível notar uma prevalência dos tons de marrom, principalmente na metade direita do heatmap, provavelmente significando uma falta de ocorrência dessas cartas na mão de ambos os jogadores e consequentemente uma falta de atualização.

Figura 5.7 – Heatmaps do modo Envido correspondentes às evoluções (a) 1, (b) 9, (c) 17 e (d) 25



Fonte: Próprio autor.

Já, por outro lado, outras combinações foram bem exploradas, aumentando consideravelmente o número de vitórias, onde a mesma cor, que na imagem (a) corresponde

ao valor de 20, na imagem (b) corresponde a pouco mais de 400. Nas imagens (c) e (d), correspondente a marca dos 3400 e 5000 jogos respectivamente, não houveram muitas mudanças drásticas, apenas um aumento nos valores de vitória, como pode ser visto na barra de legenda. Por outro lado, esse aumento foi universal, pois o padrão de cores não se alterou conforme o números de vitórias aumentou.

É interessante notar também que a área que concentra a maior probabilidade de vitória não é a que o jogador tem as maiores pontuações, mas sim a que o oponente tem as menores pontuações. Essa área mantém uma probabilidade alta em sua maioria, mas com algumas colunas possivelmente inexploradas na atualização. Mesmo o jogador tendo, teoricamente, altas chances de vitória no Envido, algumas adversidades fazem com que isso não reflita nas matrizes, podendo ter explicação na aleatoriedade da resposta da ação.

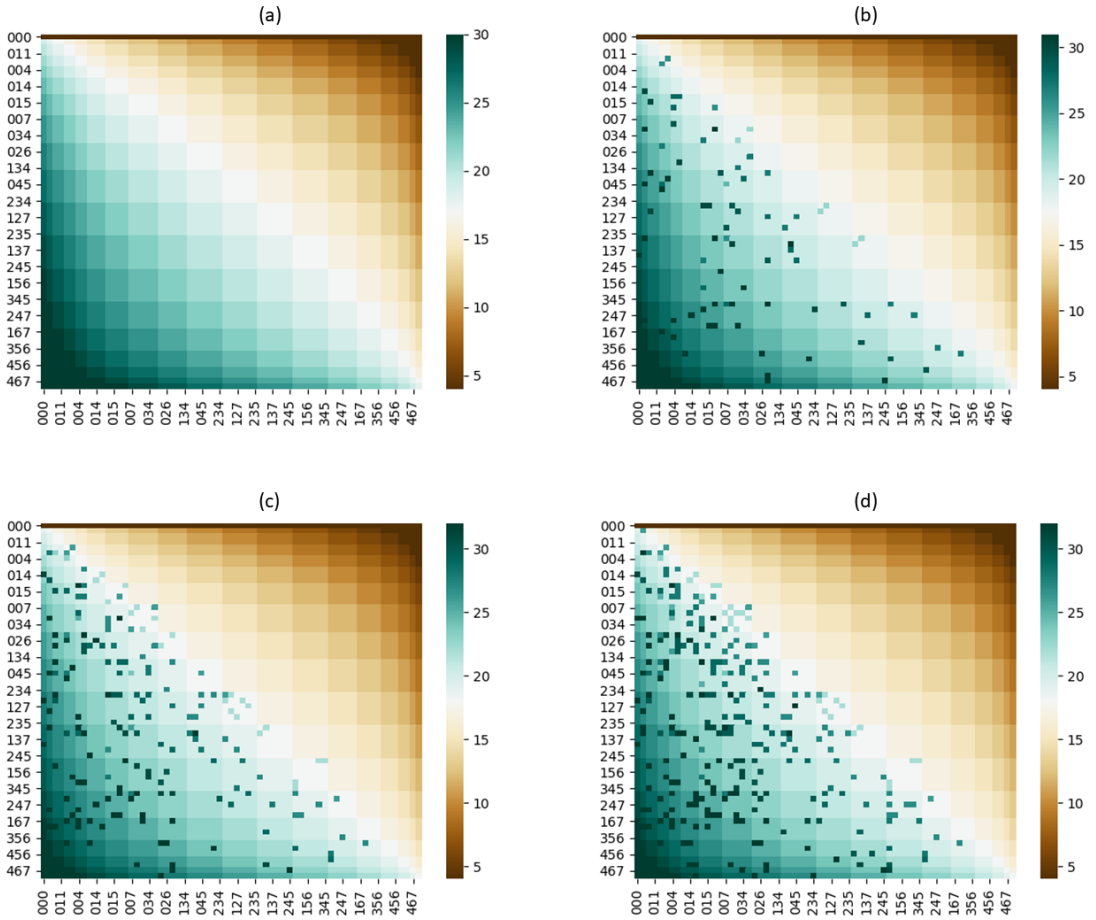
Outro fato a ser notado é que, inicialmente, a primeira linha da matriz foi tida como derrota certa em qualquer situação. Porém, no decorrer da evolução, esse pensamento foi se mostrando equivocado, visto que a combinação se tornou um valor relativamente alto, estando em um tom de azul no canto superior esquerdo no heatmap da Figura 5.7 (d).

Já para o modo Flor, é possível ver alguns heatmaps na Figural 5.8. A evolução das matrizes mostra que o intervalo dos valores pode não ter mudado, mas alguns pontos específicos aumentaram o seu valor durante as atualizações. Visto que os pedidos de Contra Flor não são muito recorrentes no jogo, as mudanças das matrizes desse modo não são tão drásticas como mostradas anteriormente no modo de Envido. Assim, é possível ver modificações mais pontuais e diretas, feitas nas combinações das mãos específicas dos jogadores. Neste caso, uma grande quantidade de jogos não iria resultar em modificações extremas na matriz, visto que a tendência é aumentar somente o número de atualizações pontuais.

Na Figura 5.9 são mostrados alguns heatmaps do modo Truco. Eles mostram que há uma prevalência desde o início, na imagem (b), de tons de marrom, representando valores mais baixos. Ao longo da evolução é possível ver que essa "mancha"marrom se espalha pela matriz. Isso acontece porque as combinações que estão sendo aceitas são apenas as mais certas, e por consequência são essas as combinações atualizadas. Os valores mais baixos vão ficando cada vez mais distantes dos mais altos e esses vão se agrupando no canto inferior esquerdo da matriz, que representa as combinações mais fortes do jogador atual e mais fracas do oponente.

No modo de Truco, uma função de atualização mais abrangente seria o ideal a ser utilizado, visto que são mais de 252 mil combinações possíveis. Para preencher cada intersecção pelo menos uma vez seria necessária uma quantidade inviável de partidas, fazendo-se indispensável uma função de atualização em área. Assim, combinações de mãos com a mesma força ou com cartas semelhantes seriam também incrementadas, mesmo não sendo a mão exata do jogo. Isso faria com que não fosse frequente o acesso a combinações não exploradas em um contexto onde já ocorreram várias partidas.

Figura 5.8 – Heatmaps do modo Flor correspondentes às evoluções (a) 1, (b) 9, (c) 17 e (d) 25

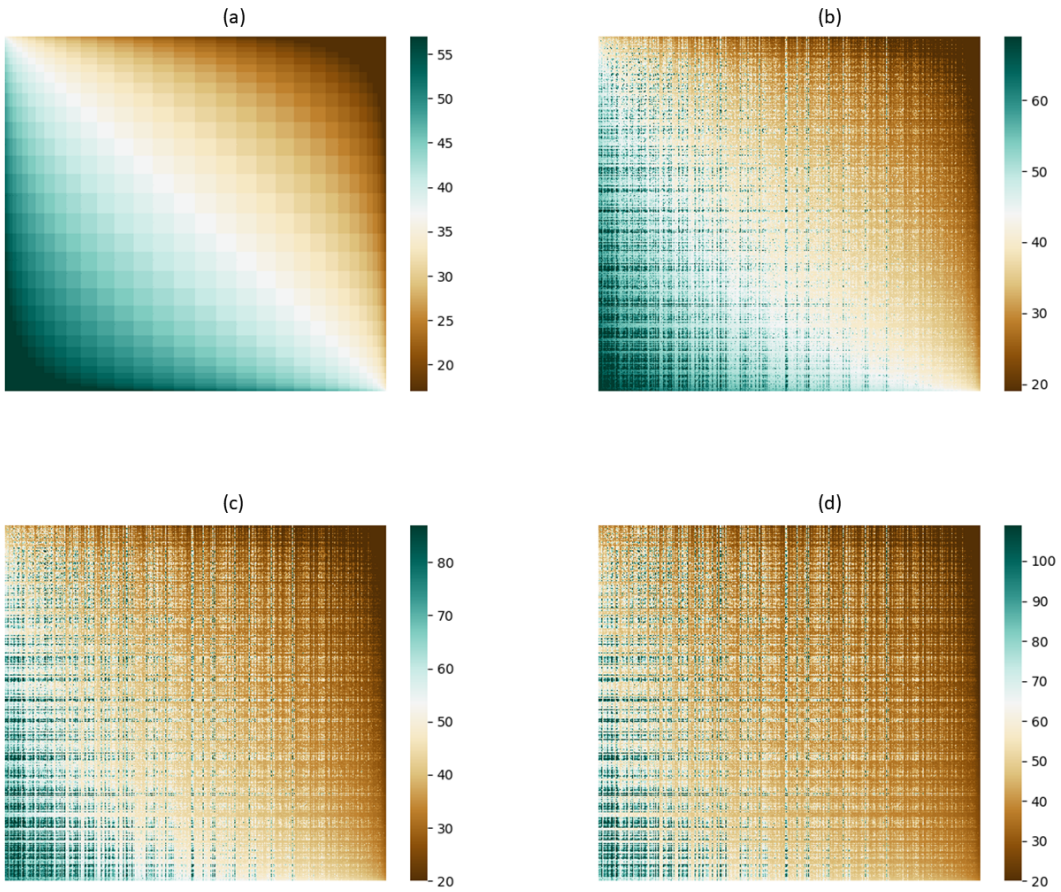


Fonte: Próprio autor.

Além disso, como o Truco é um jogo de blefe, nem sempre a chance de vitória será acima da média. A escolha de uma porcentagem de vitória mínima muito alta fez com que o agente só aceitasse o pedido de Truco em situações muito seguras, ou seja, praticamente nenhuma situação. Portanto, isso fez com que a cor, quase que em sua totalidade, do heatmap da matriz na Figura 5.9 (d) fosse marrom.

Por fim, resultantes ainda da primeira rodada de testes, a Figura 5.10 apresenta alguns heatmaps do modo de Jogar Carta. É possível observar que, já na imagem correspondente com a marca das 1800 partidas, a matriz se encontra bem diferente da inicial. A metade de cima está completamente em tons de marrom, representando as jogadas que começam com as cartas mais baixas. Entre elas estão situações em que a primeira carta jogada é a carta baixa, e a primeira e a segunda cartas jogadas são as cartas média e baixa, respectivamente. Isso mostra que essas ações não são as mais inteligentes, conforme o aprendizado do agente.

Figura 5.9 – Heatmaps do modo Truco correspondentes às evoluções (a) 1, (b) 9, (c) 17 e (d) 25

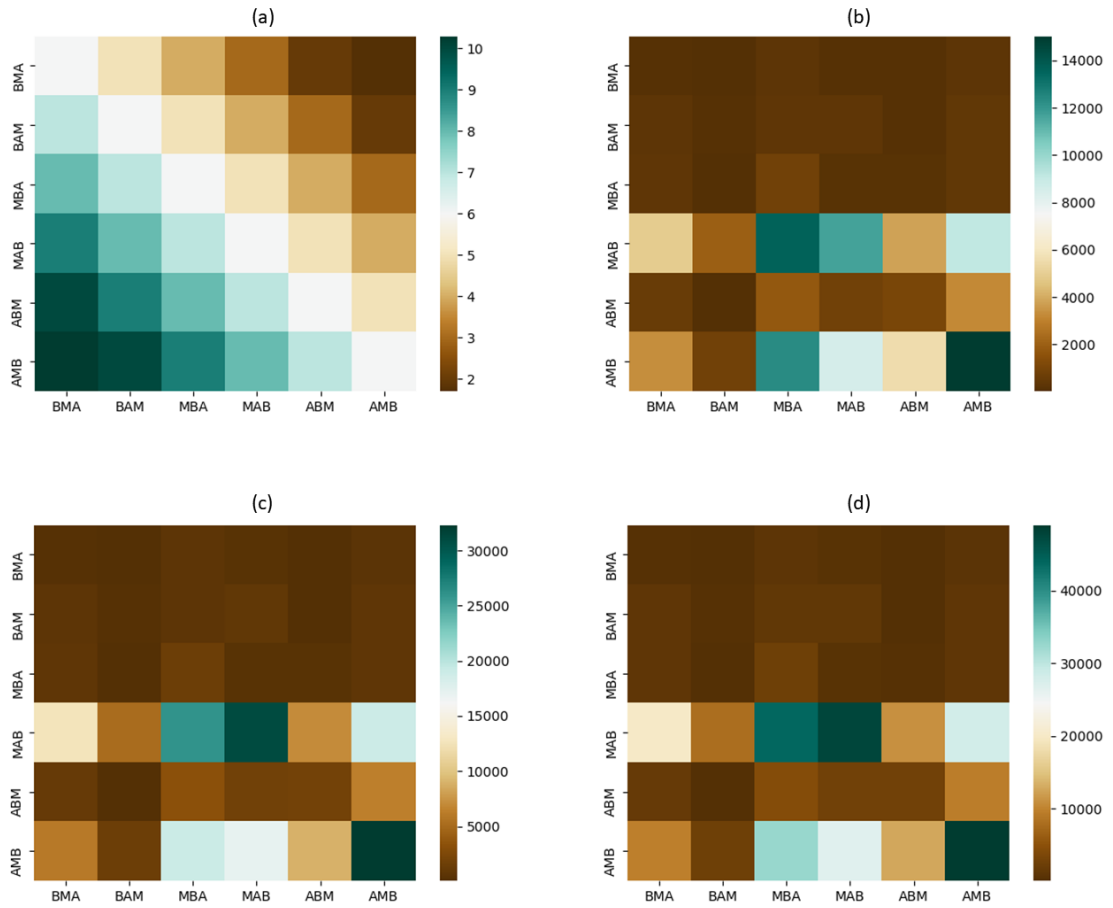


Fonte: Próprio autor.

Já das escolhas mais inteligentes, sobressaem-se 6 na metade inferior das imagens. Algumas dessas situações são as combinações onde o jogador atual jogou as suas cartas na ordem Alta-Média-Baixa e o oponente jogou as suas na ordem Média-Baixa-Alta, Média-Alta-Baixa e Alta-Média-Baixa. Outro cenário mais vitorioso é visto quando o jogador jogou as suas cartas na ordem Média-Alta-Baixa e o adversário jogou as suas na ordem Média-Baixa-Alta, Média-Alta-Baixa e Alta-Média-Baixa. Esses cenários levam à vitória do jogador, ou pelo menos ao empate, com base no nível de suas cartas e na ordem que ele as jogou.

De forma geral, durante o processo de aprendizagem, o agente aprendeu que é melhor começar jogando uma carta de nível pelo menos médio, na tentativa de garantir uma vitória já na primeira rodada. Após isso, ele lança a segunda carta mais alta, ou a própria mais alta, para garantir a vitórias da segunda rodada para si e não deixar o jogo seguir para uma terceira rodada, onde o resultado é incerto. É preferível garantir a vitória

Figura 5.10 – Heatmaps do modo Jogar Carta correspondentes às evoluções (a) 1, (b) 9, (c) 17 e (d) 25

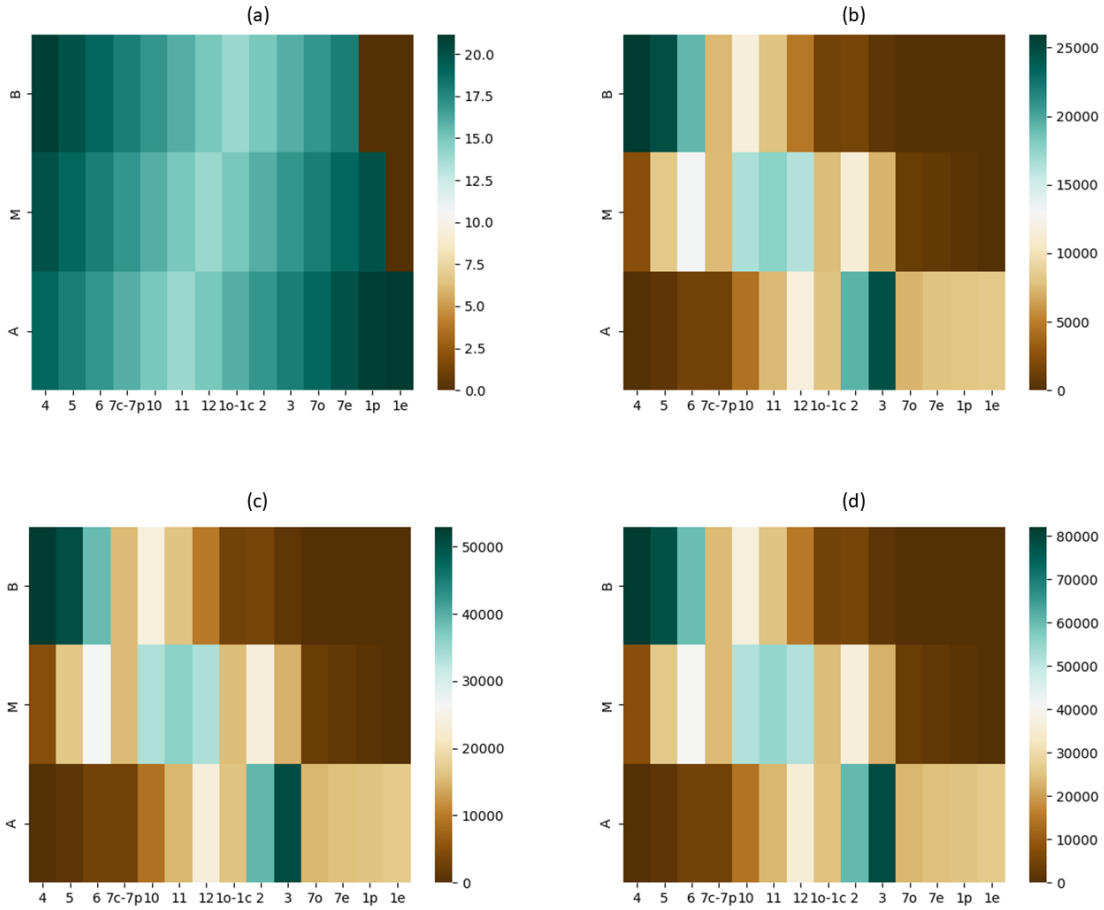


Fonte: Próprio autor.

do que deixar ela passar e consequentemente perder a partida inteira. Portanto, jogar uma carta baixa nas primeiras rodadas não é algo inteligente a ser feito, visto que é improvável que o oponente jogue uma carta mais baixa.

Além disso, a Figura 5.11 mostra a evolução da matriz auxiliar, sendo possível ver que ela se manteve praticamente inalterada durante a sua atualização. Além disso, as previsões se mostraram condizentes com a realidade, onde as cartas 4, 5 e 6 são as mais baixas e as cartas 7 de copas e 7 de paus podem variar entre baixas e médias. As cartas 10, 11 e 12 são consideradas médias, tendo também as cartas 1 de ouro e 1 de copas como possíveis médias e altas, e as cartas 2 e 3 são consideradas altas. Porém, há cartas mais altas que as cartas 1 e 2, mas elas não tiveram um grande destaque, provavelmente por não terem sido jogadas com tanta frequência. Isso pode acontecer em situações em que o jogador deseja guardar a carta para usar em uma situação mais pertinente, onde ela é mais necessária. Essas cartas são o 7 de ouro (7o), o 7 de espada (7e), o 1 de paus (1p)

Figura 5.11 – Heatmaps da matriz auxiliar correspondentes às evoluções (a) 1, (b) 9, (c) 17 e (d) 25



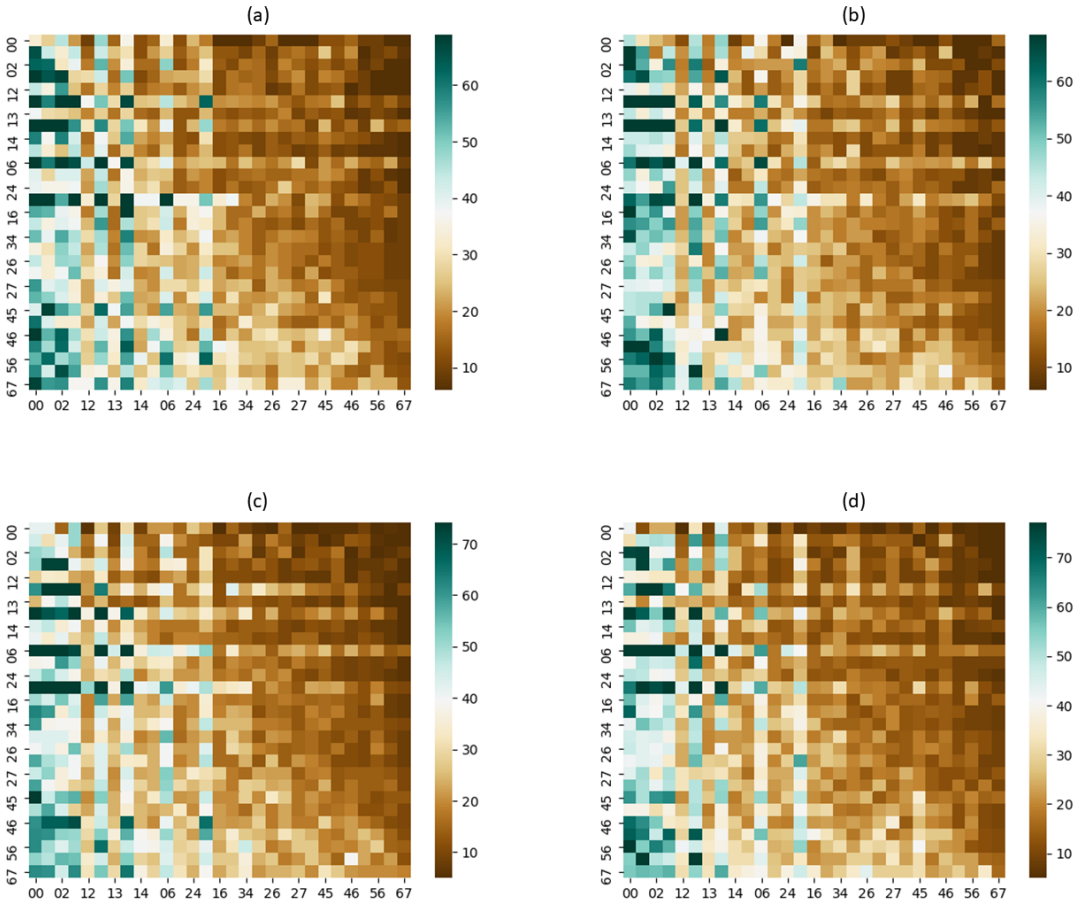
Fonte: Próprio autor.

e o 1 de espada (1e) e os seus valores para o nível alto se mantiveram abaixo da média, mas ainda assim mais altos se comparados aos outros dois níveis.

Na segunda rodada de teste, houveram 4 evoluções diferentes das matrizes, cada uma corresponde a partidas que ocorreram com um dos 4 bots descritos na seção anterior. A Figura 5.12 apresenta os heatmaps das matrizes finais do modo Envido das 4 evoluções. As imagens (a), (b), (c) e (d) mostram os heatmaps resultantes de partidas jogadas com os 1º, 2º, 3º e 4º agentes, se repetindo para os outros modos que serão apresentados posteriormente. É possível ver que elas são bastante parecidas, onde na metade direita se encontram pontos de diferentes tons de marrom, e na metade esquerda estão concentrados os pontos verdes.

Comparando com o heatmap de Envido da primeira rodada de testes, na Figura 5.7, o padrão se mantém em todas as aprendizagens, sendo o lado direito em tons de marrom e o lado esquerdo contendo pixels azuis. Porém, é notável que existe mais variação de

Figura 5.12 – Heatmaps do modo Envido correspondentes à evolução dos agentes (a) 1, (b) 2, (c) 3 e (d) 4

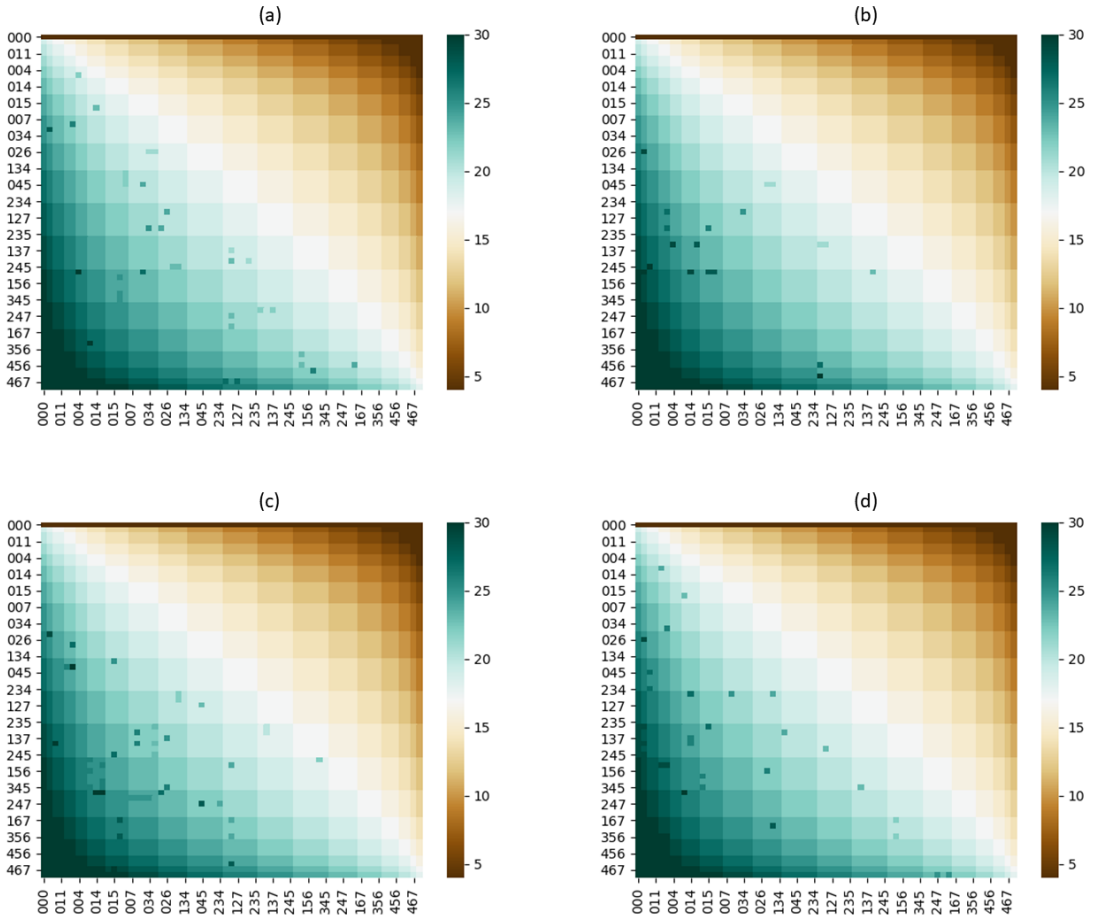


Fonte: Próprio autor.

tom nas matrizes da Figura 5.12. Por fim, levando em consideração a quantidade de partidas que levaram a cada matriz final, é visto que, mesmo com poucas partidas jogadas, a aprendizagem derivada da segunda rodada de testes levou a praticamente o mesmo resultado alcançado na primeira rodada. É interessante notar também que, mesmo o oponente tendo características de jogo diferentes, podendo ser mais agressivo ou conservador, o resultado final foi praticamente o mesmo nas cinco aprendizagens.

A Figura 5.13 apresenta, para o modo Flor, as matrizes finais das 4 aprendizagens. É possível observar que cada uma tem atualizações em lugares diferentes e também não há um padrão. Isso reforça a ideia de que, mesmo com uma grande quantidade de jogos, há a possibilidade de coincidir com uma combinação que ainda não foi modificada e está com o preenchimento inicial. Isso enfatiza também a necessidade de uma função de atualização em área, uma vez que o que se espera é que, em um certo número de partidas jogadas, já não se encontrem mais os valores da matriz inicial.

Figura 5.13 – Heatmaps do modo Flor correspondentes à evolução dos agentes (a) 1, (b) 2, (c) 3 e (d) 4

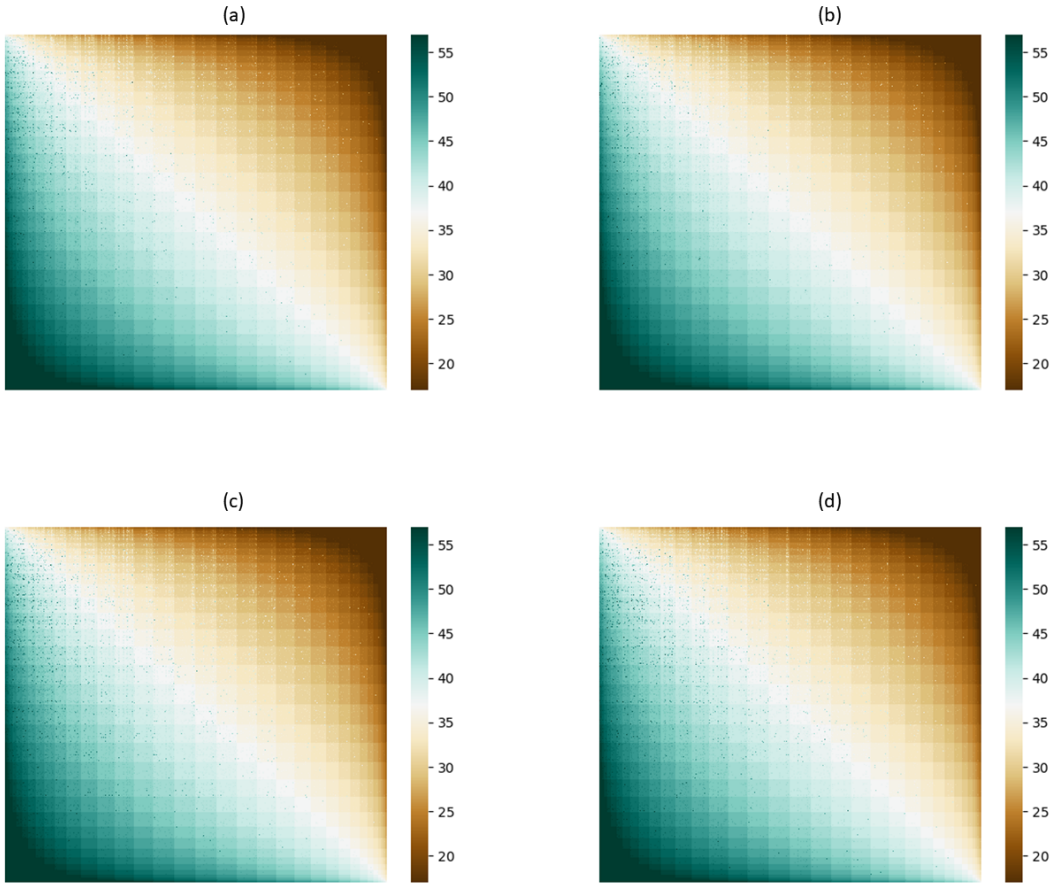


Fonte: Próprio autor.

Comparando com os heatmaps da Figura 5.8, é visto que, se houvesse um conjunto maior de partidas, as matrizes apresentadas na Figura 5.13 poderiam evoluir para o mesmo caminho das matrizes da primeira rodada de testes, com mais "pontinhos" pelo heatmap. Como os pedidos de Contra Flor não são muito frequentes, isso não aconteceu tendo somente 500 partidas com cada bot. Da mesma forma, o número de 4096 combinações possíveis é muito alto e, em adição a isso, a chance de acontecer uma Flor para ambos os jogadores e evoluir para o Contra Flor é muito baixa.

A Figura 5.14 apresenta os heatmaps das matrizes finais das 4 aprendizagens para o modo Truco. Entre as quatro matrizes não é possível ver grande diferença, visto que há pouca mudança da matriz inicial. É possível ver somente uma leve alteração onde há alguns pixels em um tom de azul mais forte na diagonal inferior dos heatmaps, e pixels mais claros na diagonal superior, onde a cor predominante é a marrom. Esse padrão aconteceu nas 4 matrizes finais, independente da característica de jogo do oponente.

Figura 5.14 – Heatmaps do modo Truco correspondentes à evolução dos agentes (a) 1, (b) 2, (c) 3 e (d) 4



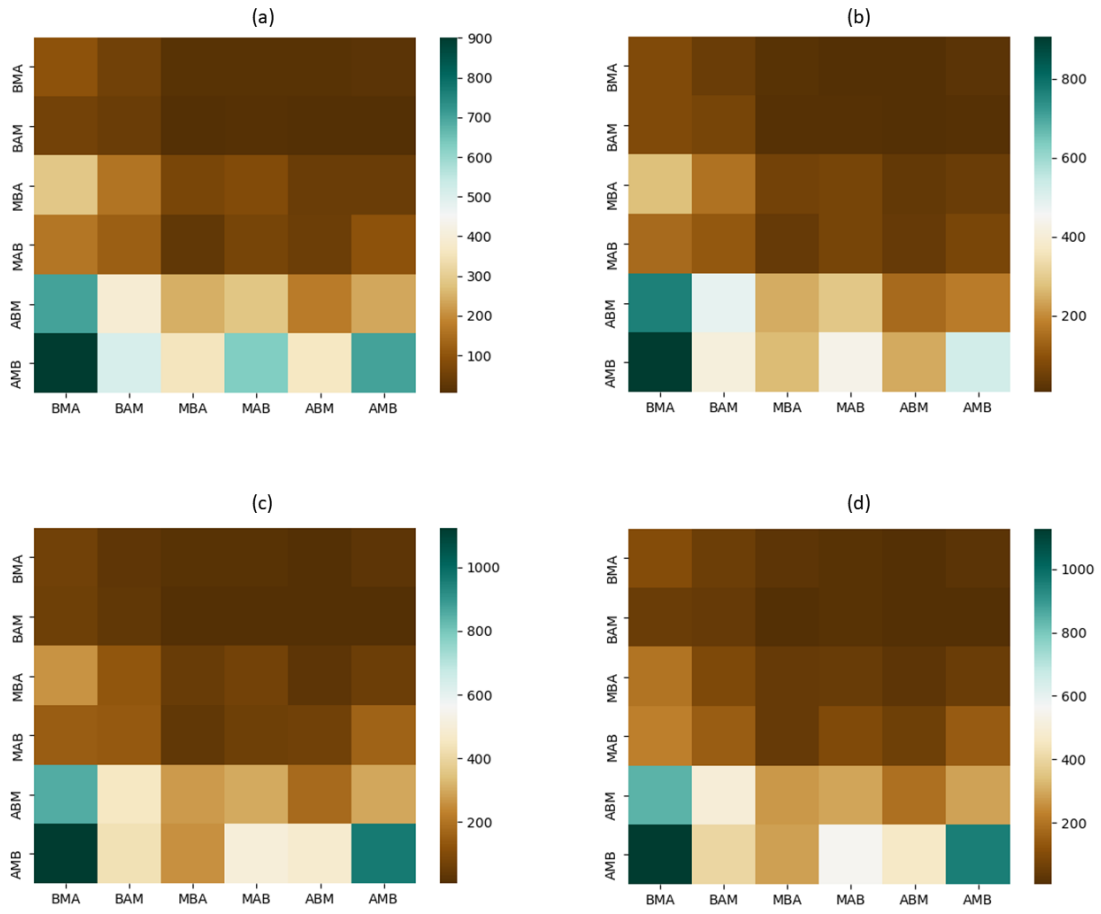
Fonte: Próprio autor.

Comparando com a matriz da Figura 5.9 (d), é possível ver que há uma grande diferença, principalmente na quantidade de pixels de cor marrom. A julgar pelo aparecimento de cores mais claras na parte marrom do heatmap, as matrizes da segunda rodada de testes não seguiriam o mesmo padrão apresentado nas matrizes da primeira rodada. Isso pode ser concluído também pelo avanço violento do tom marrom sobre o azul na Figura 5.9. Seriam necessários também mais partidas jogadas nos moldes da segunda rodada de testes, a fim de investigar qual seria o caminho seguido pela aprendizagem com os 4 agentes.

A Figura 5.15 exibe os heatmaps das matrizes finais do modo Jogar Carta das 4 evoluções da segunda rodada de testes. É possível observar que todas seguem um mesmo padrão de ter mais da metade do heatmap da cor marrom, ao contrário da Figura 5.10 (d), onde a quarta linha ainda possuía jogadas de valor mais alto. Todas as 4 aprendizagens têm 3 combinações fortes em comum, sendo que as que evoluíram com os agentes

conservadores são ainda mais semelhantes.

Figura 5.15 – Heatmaps do modo Jogar Carta correspondentes à evolução dos agentes (a) 1, (b) 2, (c) 3 e (d) 4



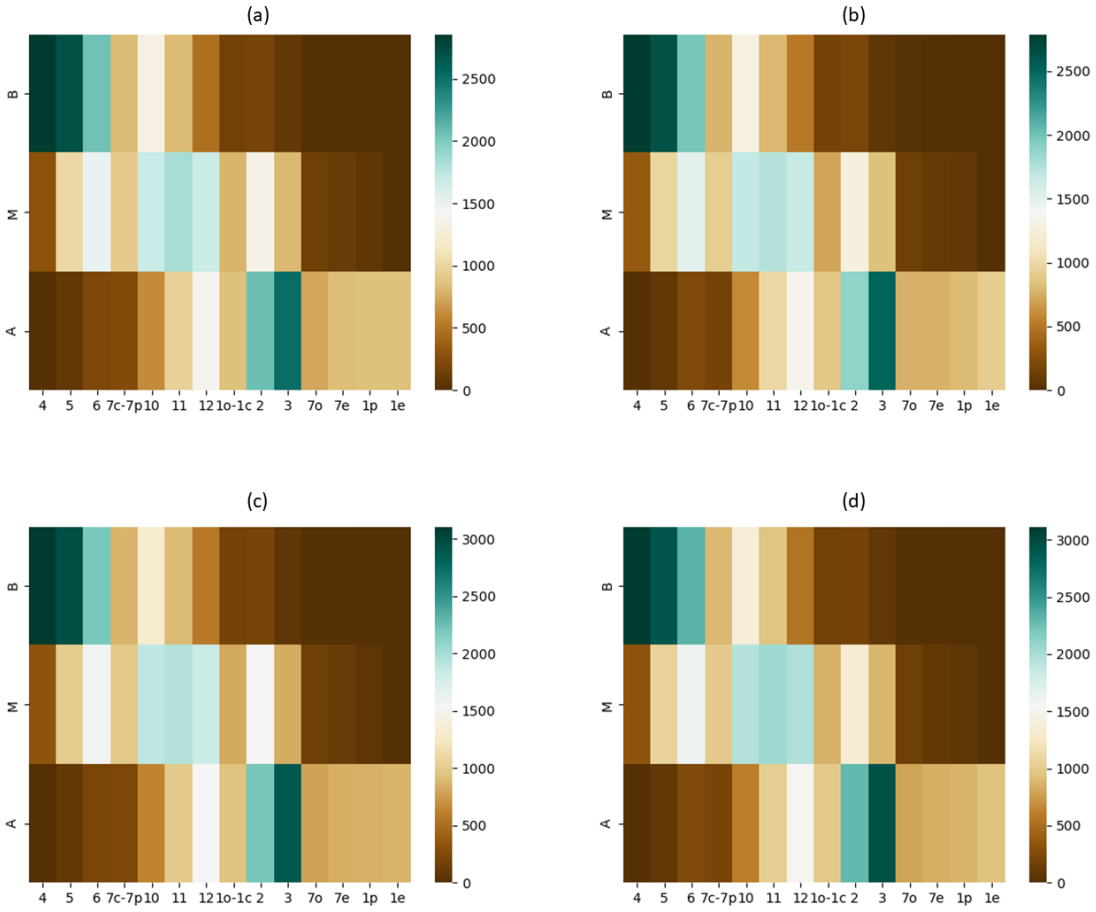
Fonte: Próprio autor.

As ordens de jogo Alta-Baixa-Média e Alta-Média-Baixa para o jogador atual e Baixa-Média-Alta para o oponente, juntamente com a ordem Alta-Média-Baixa para ambos os jogadores, são os destaques igualmente fortes em todas as quatro aprendizagens. Comparando com a Figura 5.10, somente a última combinação é idêntica e as duas primeiras sequer estão em um tom claro. Provavelmente essas combinações foram menos exploradas em um caso e mais em outro, mesmo sendo óbvias até para um jogador iniciante.

Por fim, a Figura 5.16 mostra os heatmaps das matrizes auxiliares finais das 4 evoluções. A partir delas é possível ver que, comparando com a Figura 5.11 (d), todas as aprendizagens seguiram para um mesmo desfecho. Mesmo com uma quantidade bem abaixo de partidas, as matrizes da segunda rodada de testes alcançaram o mesmo padrão das matrizes da primeira rodada. Todas elas concluíram que as cartas mais baixas são

os 4, 5 e 6, as médias, em sua maioria, são os 10, 11 e 12, e as altas são os 2 e 3, provavelmente os mais utilizados pelos jogadores.

Figura 5.16 – Heatmaps da matriz auxiliar correspondentes à evolução dos agentes (a) 1, (b) 2, (c) 3 e (d) 4



Fonte: Próprio autor.

O último capítulo deste trabalho apresenta conclusões sobre o mesmo, falando sobre os principais pontos levantados ao longo do desenvolvimento. Da mesma forma, são apresentados os trabalhos futuros, pensando em melhorias que podem ser implementadas a fim de elevar a porcentagem de vitórias do modelo.

6 CONCLUSÃO

O desenvolvimento descrito neste TCC é um trabalho inicial de análise e implementação da estrutura base. Da mesma forma, abrirá caminho para outros trabalhos, servindo como uma ideia inicial para futuras pesquisas e aprimoramentos. Utilizando a técnica adaptada de Cadeias de Markov, temos, ao invés de uma grande infraestrutura, arquivos de texto simples e leves, que podem ser atualizados de maneira fácil e rápida. Estes arquivos contém matrizes, onde as linhas e colunas representam possibilidades de jogos. Também, o custo de memória e uso computacional são mínimos, necessitando de pouco poder de processamento.

Ao longo dos testes, foram percebidas algumas questões que, se melhoradas, elevariam a capacidade de vitória da técnica descrita neste trabalho. Como possível forma de resolver o problema da necessidade de uma grande quantidade de jogos no preenchimento das matrizes, uma solução seria a atualização por área, ao invés de somente em um ponto específico. Com isso, uma jogada influencia nas jogadas similares ao seu redor, diminuindo assim a necessidade de ter um jogo por configuração de cartas ou pontos. Essa necessidade pode fazer com que, em um avançado número de partidas, ainda aconteça de ser a primeira vez que uma configuração de cartas para os dois jogadores será escolhida. Ajustar a atualização fará com que sejam necessários menos jogos para preencher de forma constante toda a matriz, além de proporcionar uma resposta mais rápida por parte do agente, pois uma atualização corresponde a vários cenários.

A função de atualização em área pode ser feita levando em consideração os níveis das cartas, onde podem ser utilizados os níveis descritos no modo de Jogar Carta. Também podem ser criados novos níveis, o que faria com que cada nível abrangesse menos cartas, aumentando a precisão da similaridade. Assim, as cartas das mãos do jogo atual podem ser substituídas pelas cartas de mesmo nível e assim as combinações similares também seriam atualizadas. Essa atualização pode ser adicionada ao modelo sem a necessidade de reestruturação, sendo de fácil incorporação ao modelo já existente, facilitando assim a sua implementação.

O decaimento no número de vitórias, mostrado na Seção 5.3, pode ter como motivo a busca em pontos inexplorados da matriz, fazendo com que o ponto levantado anteriormente seja ainda mais evidente. Essa estratégia pode ser comparada a utilizada em (PAULUS, 2019), onde foi feita uma função de similaridade entre os jogos. Comparando com os agentes de (PAULUS, 2019), seria de extrema importância a implementação de uma função de similaridade. Ela ajudaria tanto na etapa de buscar a ação de jogo, encontrando mãos similares a fim de melhorar a acurácia da ação, quanto na atualização, com o propósito de torná-la mais efetiva e abrangente. Da mesma forma que a função de atualização, a função de similaridade por ser feita considerando os níveis descritos no modo de

Jogar Carta ou com a criação de novos níveis, a fim de aumentar a acurácia da escolha da ação de jogo.

Outra questão que pode ser levantada é que o peso inicial para a atualização é arbitrário e único, ou seja, não se modificou conforme o número de partidas jogadas. Também, esse valor se torna inapropriado quando a divisão na atualização é muito grande. Por exemplo, para o modo Truco, quando se tem a informação de somente duas cartas para cada jogador. O número de combinações possíveis é, basicamente, de 169 e o valor a ser divido entre elas é somente 73, não sendo suficiente para gerar alguma mudança. Esse ajuste no peso pode ser feito considerando a média dos valores de vitória ou até mesmo uma porcentagem sobre o índice de crescimento. Sendo assim, o ajuste desse peso faz parte dos trabalhos futuro deste projeto.

Conforme dito anteriormente na Seção 3.2, a equação de Chapman Kolmogorov permite calcular a probabilidade para n passos adiante. Neste trabalho isso não foi feito, ou seja, a probabilidade das rodadas anteriores não é levada para a próxima rodada. Portanto, no futuro, isso é algo importante de ser implementado, pois aumentará a precisão na escolha da carta a ser jogada pelo agente.

Durante a análise feita na Seção 5.3 dos resultados provenientes da segunda rodada de testes, foi visto que, em jogos com o 4º agente, o padrão de vitórias se inverteu. Em jogos contra o 1º, 2º e 3º agentes, o número de vitórias sendo pé foi maior do que sendo mão. Já com o 4º agente, houve uma inversão nesse padrão e o número de vitórias sendo mão foi maior. A causa dessa inversão não é clara, podendo ter diversas explicações, sendo uma delas o conservadorismo de ambos os agentes. Uma análise mais aprofundada dos arquivos de logs está entre os trabalhos futuros dessa pesquisa, sendo de grande importância para entender o comportamento do agente perante outros modelos.

Na Seção 5.2 é apresentado um exemplo de jogo e nele são mostradas as porcentagens mínimas de vitória para os modos de Envido e Truco e as suas contra propostas. Esses são valores definidos arbitrariamente e permanecem estáticos, não sendo alterados conforme a aprendizagem do agente. Futuramente, estes valores devem ser alterados para se atualizarem conforme análise de logs dos jogos, fazendo com que se adaptem e não sejam desproporcionais a realidade do jogo. Uma forma de adaptar esses valores é fazer uma análise dos logs das partidas jogadas pelo agente e verificar o seu nível de aceitação perante as disputas de cada modo.

Em adição ao que já foi colocado, outro detalhe que explica o baixo número de vitórias é que, para obter uma vitória completa, é preciso vencer diversas mãos ao longo da partida. Então, é possível que, mesmo perdendo a partida, o agente tenha ganhado várias mãos, apenas não o suficiente para obter a vitória. O jogo, da maneira que foi implementado, dá a vitória para o primeiro jogador que completar 24 pontos. Com isso, é possível que o perdedor possa ter conseguido até 23 pontos, não fazendo dele um jogador completamente ruim.

Mesmo todas as limitação citadas e diversas melhorias pensadas, o agente conseguiu uma porcentagem de vitória de 28% sobre os agentes de (PAULUS, 2019), que corresponde a 56/200 partidas. São resultados satisfatórios considerando a complexidade do jogo e o número de interações, sem treinamento anterior ou propagação de aprendizagem. Além de tudo, o poder computacional necessário é ínfimo ($O(n)$), sendo n o número de estados alcançáveis, permitindo a portabilidade do modelo. Por fim, como trabalhos futuros estão todas as melhorias apresentadas ao longo desse capítulo, como utilizar um método de atualização que propague a probabilidade, a fim de atingir os estados semelhantes, e trabalhar no ajuste da taxa de aprendizagem, que anteriormente era fixo em 10%. Da mesma forma, uma função de similaridade também está na lista de trabalhos futuros, juntamente com testes contra jogadores humanos.

REFERÊNCIAS BIBLIOGRÁFICAS

AMBEKAR, G. et al. Anticipation of winning probability in poker using data mining. In: **2015 International Conference on Computer, Communication and Control (IC4)**. [S.l.: s.n.], 2015. p. 1–6.

ANDERSEN, P.-A.; GOODWIN, M.; GRANMO, O.-C. Towards a deep reinforcement learning approach for tower line wars. **Proceedings of the 37th SGAI International Conference on Artificial Intelligence, Cambridge, UK, 2017, Artificial Intelligence XXXIV, 2017**, 2017. Disponível em: <http://dx.doi.org/10.1007/978-3-319-71078-5>.

BALAN, S.; OTTO, J. **Business Intelligence in Healthcare with IBM Watson Analytics**. 1st. ed. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2017. ISBN 1548829897.

BENMAKRELOUF, S.; MEZGHANI, N.; KARA, N. Towards the identification of players' profiles using game's data analysis based on regression model and clustering. In: **2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)**. [S.l.: s.n.], 2015. p. 1403–1410.

KRATZ, L.; SMITH, M.; LEE, F. J. Wiizards: 3d gesture recognition for game play input. In: **Proceedings of the 2007 Conference on Future Play**. New York, NY, USA: Association for Computing Machinery, 2007. (Future Play 07), p. 209212. ISBN 9781595939432. Disponível em: <https://doi.org/10.1145/1328202.1328241>.

NIKLAUS, J. et al. Survey of artificial intelligence for card games and its application to the swiss game jass. In: **2019 6th Swiss Conference on Data Science (SDS)**. [S.l.: s.n.], 2019. p. 25–30.

OKABE, Y.; SAITO, S.; NAKAJIMA, M. Paintbrush rendering of lines using hmms. In: **Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia**. New York, NY, USA: Association for Computing Machinery, 2005. (GRAPHITE 05), p. 9198. ISBN 1595932011. Disponível em: <https://doi.org/10.1145/1101389.1101405>.

PAULUS, G. B. **Casos e clusters no desenvolvimento de políticas de reuso para tomada de decisão em jogos de cartas**. 2019. 130 p. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Maria, Santa Maria, 2019.

STEWART, W. J. **Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling**. [S.l.]: Princeton University Press, 2009.

SYNNAEVE, G.; BESSIERE, P. Multiscale bayesian modeling for rts games: An application to starcraft ai. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 8, n. 4, p. 338–350, 2016.

THURAU, C.; HETTENHAUSEN, T.; BAUCKHAGE, C. Classification of team behaviors in sports video games. In: **18th International Conference on Pattern Recognition (ICPR'06)**. [S.l.: s.n.], 2006. v. 1, p. 1188–1191.

VINYALS, O. et al. Starcraft II: A new challenge for reinforcement learning. **CoRR**, abs/1708.04782, 2017. Disponível em: <http://arxiv.org/abs/1708.04782>.

WATSON, I. et al. Improving a case-based texas hold'em poker bot. In: **2008 IEEE Symposium On Computational Intelligence and Games**. [S.l.: s.n.], 2008. p. 350–356.

WINNE, L. L. **Truco**. [S.l.]: Ciudad Autónoma de Buenos Aires Ediciones Godot, 2017.

XU, M.; SHI, H.; WANG, Y. Play games using reinforcement learning and artificial neural networks with experience replay. In: **2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)**. [S.l.: s.n.], 2018. p. 855–859.

ZHOU, X.; HUANG, X.; WANG, Y. Real-time facial expression recognition in the interactive game based on embedded hidden markov model. In: **Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.** [S.l.: s.n.], 2004. p. 144–148.

APÊNDICE A – ALGORITMO DO MODO ENVIDO

```
1 import json as js
2 import sys
3 import random as rd
4 import math
5
6
7 class Envido:
8     def __init__(self):
9         self.pontosEnvido = 0 # meus pontos de envido
10        self.combinacaoEnvido = None # minha combinação de envido
11        self.possiveisCombEnvido = [] # possiveis combinações de
    envido do adversario
12
13        self.ordem = [] # ordem das cartas (menor para maior)
14        self.codificacao = [] # codificação das cartas
15
16        self.trucoDescription = {} # classe trucoDescription
17        self.pasta = "C:/Users/LanaR/eclipse-workspace/
    clustercbrgamer/src/markov/"
18
19        self.probLimiteGanhoEnvido = 0 # probabilidade minima de
    ganho do envido
20
21        self.cabecalhoMatProbEnvido = [] # cabeçalho das matrizes
    de markov
22        self.matrizEnvidoVitoria = [[]] # matriz de probabilidades
     de vitoria
23        self.matrizEnvidoDerrota = [[]] # matriz de probabilidades
     de derrota
24
25    def main(self):
26        """
27        Método inicial que é chamado de fora da classe e executa
    as operações
28        """
29        self.trucoDescription = js.load(open(self.pasta + '
```

```
    trucoDescription.json', 'r'))
30          self.probLimiteGanhoEnvido = float(sys.argv[2])
31          self.pontosEnvido = self.trucoDescription['
    pontosEnvidoRobo']
32          self.ordem = ["4", "5", "6", "7", "10", "11", "12", "1", "
    2", "3", "7o", "7e", "1p", "1e"]
33          self.codificacao = [1, 2, 3, 4, 6, 7, 8, 12, 16, 24, 40,
    42, 50, 52]
34
35          self.extraindoMatrizes()
36          self.defineMinhaCombEnvido()
37          self.chamaEnvido()
38
39      def extraindoMatrizes(self):
40          """
41          Método que extrai dos arquivos as matrizes de vitória e
    derrota e coloca nas variáveis da classe
42          """
43          vitoria = open(self.pasta + 'matrizEnvidoVitoria.txt', 'r'
    )
44          derrota = open(self.pasta + 'matrizEnvidoDerrota.txt', 'r'
    )
45          matrizVit = []
46          matrizDer = []
47          self.cabecalhoMatProbEnvido = vitoria.readline().split()
48          cabecalho = derrota.readline()
49
50          linhasVit = vitoria.readlines()
51          linhasDer = derrota.readlines()
52
53          for i in range(len(linhasVit)):
54              matrizVit.append(linhasVit[i].split())
55
56          for i in range(len(linhasDer)):
57              matrizDer.append(linhasDer[i].split())
58
59          self.matrizEnvidoVitoria = matrizVit
60          self.matrizEnvidoDerrota = matrizDer
61
```

```
62          vitoria.close()
63          derrota.close()
64
65      def definePossiveisCombEnvido(self, primeiraCarta):
66          """
67          Método que encontra as possíveis combinações de cartas/
    pontos do adversário
68          :param primeiraCarta: primeira carta do humano, a partir
    dela são encontradas as combinações
69          """
70          if primeiraCarta is not None:
71              for numeroComb in range(0, 8):
72                  if numeroComb != primeiraCarta:
73                      self.possiveisCombEnvido.append(self.
    arrumaCombinacao(primeiraCarta, numeroComb))
74                  if numeroComb == 0 and primeiraCarta == 0:
75                      self.possiveisCombEnvido.append(self.
    arrumaCombinacao(primeiraCarta, numeroComb))
76
77          elif primeiraCarta is None:
78              self.possiveisCombEnvido = self.cabecalhoMatProbEnvido
    .copy()
79
80      def ajustaPrimeiraCartaHumano(self, primeiraCarta):
81          """
82          Método que ajusta a primeira carta do jogador ao padrão
83          :param primeiraCarta:
84          :return:
85          """
86          if primeiraCarta is not None:
87              primeiraCarta = self.ordem[self.codificacao.index(
    primeiraCarta)]
88              naipes = ["o", "e", "p"]
89
90              if primeiraCarta[-1] in naipes: primeiraCarta = int(
    primeiraCarta[:-1])
91              if int(primeiraCarta) >= 10: primeiraCarta = 0
92              else: primeiraCarta = int(primeiraCarta)
93
```

```
94              return primeiraCarta
95
96      def defineMinhaCombEnvido(self):
97          """
98          Método que analisa quais cartas tem naipes iguais e define
        pontos e combinação dessas cartas.
99          """
100         carta1 = 0
101         carta2 = 0
102
103         cartaBaixa = self.ordem[self.codificacao.index(self.
        trucoDescription['cartaBaixaRobo'])]
104         cartaMedia = self.ordem[self.codificacao.index(self.
        trucoDescription['cartaMediaRobo'])]
105         cartaAlta = self.ordem[self.codificacao.index(self.
        trucoDescription['cartaAltaRobo'])]
106
107         naipes = ["o", "e", "p"]
108
109         if cartaBaixa[-1] in naipes: cartaBaixa = int(cartaBaixa
        [:-1])
110         if int(cartaBaixa) >= 10: cartaBaixa = 0
111         else: cartaBaixa = int(cartaBaixa)
112
113         if cartaMedia[-1] in naipes: cartaMedia = int(cartaMedia
        [:-1])
114         if int(cartaMedia) >= 10: cartaMedia = 0
115         else: cartaMedia = int(cartaMedia)
116
117         if cartaAlta[-1] in naipes: cartaAlta = int(cartaAlta
        [:-1])
118         if int(cartaAlta) >= 10: cartaAlta = 0
119         else: cartaAlta = int(cartaAlta)
120
121         if cartaBaixa + cartaMedia + 20 == self.pontosEnvido:
122             carta1 = cartaBaixa
123             carta2 = cartaMedia
124         elif cartaBaixa + cartaAlta + 20 == self.pontosEnvido:
125             carta1 = cartaBaixa
```

```
126             carta2 = cartaAlta
127         elif cartaMedia + cartaAlta + 20 == self.pontosEnvido:
128             carta1 = cartaMedia
129             carta2 = cartaAlta
130 
131         carta1 = self.avaliaCarta(carta1)
132         carta2 = self.avaliaCarta(carta2)
133 
134         self.combinacaoEnvido = self.arrumaCombinacao(carta1,
    carta2)
135 
136     def arrumaCombinacao(self, carta1, carta2):
137         """
138         Método que arruma a combinação de acordo com a ordem das
    cartas
139         :param carta1: primeira carta
140         :param carta2: segunda carta
141         :return: combinação na ordem certa
142         """
143         if carta1 > carta2:
144             return str(carta2) + str(carta1)
145         elif carta1 <= carta2:
146             return str(carta1) + str(carta2)
147 
148     def avaliaCarta(self, valorCarta):
149         """
150         Método que avalia o valor da carta
151         :param valorCarta: número da carta
152         :return: valor de carta, se for menor do que 10, e 0 se
    for igual ou maior do que 10
153         """
154         if valorCarta >= 10: return 0
155         else: return valorCarta
156 
157     def markovEnvido(self):
158         """
159         Método que executa todas as operações
160         :return: falso se não tem pontos de envido, se não
    alcançou o mínimo ou se os calculos
```

```
161                 não forem positivos para a questão, e true se os
    resultados derem positivo.
162         """
163         if self.pontosEnvido < 20: return False
164 
165         primeiraCarta = self.ajustaPrimeiraCartaHumano(self.
    trucoDescription['primeiraCartaHumano'])
166 
167         self.definePossiveisCombEnvido(primeiraCarta)
168 
169         ganhosPossiveisEnvido = 0
170         perdasPossiveisEnvido = 0
171         listaVitoriaDerrota = []
172 
173         # contabiliza derrotas e vitorias
174         for comb in self.possiveisCombEnvido:
175             ganhosPossiveisEnvido += int(self.matrizEnvidoVitoria[
    self.cabecalhoMatProbEnvido.index(self.combinacaoEnvido)][self.
    cabecalhoMatProbEnvido.index(comb)])
176             perdasPossiveisEnvido += int(self.matrizEnvidoDerrota[
    self.cabecalhoMatProbEnvido.index(self.combinacaoEnvido)][self.
    cabecalhoMatProbEnvido.index(comb)])
177 
178         if ganhosPossiveisEnvido == 0 and perdasPossiveisEnvido ==
     0:
179             ganhosPossiveisEnvido = 1
180             perdasPossiveisEnvido = 1
181 
182         # calcula porcentagem
183         porcentagemGanhos = ganhosPossiveisEnvido / (
    ganhosPossiveisEnvido + perdasPossiveisEnvido)
184         porcentagemPerdas = perdasPossiveisEnvido / (
    ganhosPossiveisEnvido + perdasPossiveisEnvido)
185 
186         porcentagemGanhos = round(porcentagemGanhos, 1)
187         porcentagemPerdas = round(porcentagemPerdas, 1)
188 
189         if primeiraCarta is not None and primeiraCarta > 5:
190             if porcentagemGanhos - 0.2 > 0:
```

```
191                 porcentagemGanhos -= 0.2
192                 porcentagemPerdas += 0.2
193
194         if self.probLimiteGanhoEnvido < porcentagemGanhos:
195             pass
196         else:
197             return False
198
199         # insere na lista para randomizar
200         for i in range(int(porcentagemGanhos * 10)):
201             listaVitoriaDerrota.append("G")
202
203         for i in range(int(porcentagemPerdas * 10)):
204             listaVitoriaDerrota.append("P")
205
206         # escolhe randomico da lista
207         escolheJogo = rd.choice(listaVitoriaDerrota)
208
209         # se escolheu G retorna true, se não false
210         if escolheJogo == "G": return True
211         else: return False
212
213     def chamaEnvido(self):
214         """
215         Método que verifica o resultado das operações. Só esse
    método pode usar a função print()
216         """
217         if self.markovEnvido(): print("true")
218         else: print("false")
219
220 if __name__ == "__main__":
221     e = Envido()
222     e.main()
```