

Ivan de Jesus Pereira Pinto

Inteligência Artificial aplicada ao Jogo de Dominó

São Luís - MA

2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Pereira Pinto, Ivan de Jesus.

Inteligência Artificial aplicada ao Jogo de Dominó /
Ivan de Jesus Pereira Pinto. - 2018.
47 p.

Orientador(a): Luciano Reis Coutinho.

Monografia (Graduação) - Curso de Ciência da
Computação, Universidade Federal do Maranhão, São Luís -
MA, 2018.

1. Aprendizado de Máquina. 2. Inteligência
Artificial. 3. Teoria dos Jogos. I. Reis Coutinho,
Luciano. II. Título.

Ivan de Jesus Pereira Pinto

Inteligência Artificial aplicada ao Jogo de Dominó

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Luciano Reis Coutinho.

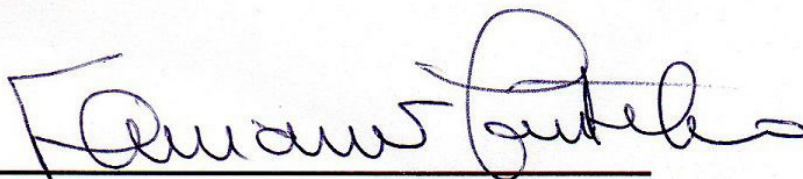
São Luís - MA

2018

Inteligência Artificial aplicada ao Jogo de Dominó

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

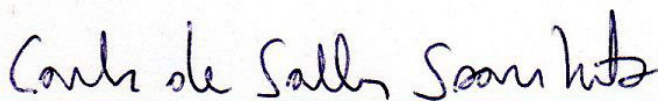
Trabalho Aprovado em 22/01/2018



Prof. Dr. Luciano Reis Coutinho.

Orientador

Universidade Federal do Maranhão



Prof. Dr. Carlos de Salles Soares Neto

Examinador

Universidade Federal do Maranhão



Prof. Dr. Tiago Bonini Borchardt

Examinador

Universidade Federal do Maranhão

Agradecimentos

Agradeço aos meu pais, minha Madrinha e Padrinho, minha Tia Selma, e todos na família que me apoiaram.

Agradeço também ao apoio dos colegas, da ajuda do professor orientador Luciano que está comigo nesse trabalho a muito tempo, e um agradecimento especial a Jéssica Cardoso pela companhia.

Resumo

O jogo de Dominó possui diversas características que o tornam um modelo interessante para os campos de Inteligência Artificial (IA), Aprendizado de Máquina (AM) e Teoria dos Jogos (TJ). Sua observabilidade parcial e estocasticidade presente no monte de compra são alguns exemplos. A escassa aplicação no Dominó de técnicas tradicionais da IA e dos campos citados, motivam a realização de um estudo comparativo para esse problema. Abordou-se duas das versões mais comuns de Dominó, **Dominó com Compra** e **Dominó Ponta de Cinco**.

Este trabalho faz o estudo de soluções existente na literatura da IA, mais especificamente Aprendizado por Reforço e Busca. Avalia-se as técnicas derivadas desses campos juntamente com técnicas baseadas em regras através de comparações experimentais. Os resultados revelam melhor performance do algoritmo baseado em Busca, *Perfeita Informação Monte Carlo* (*PIMC*), para o jogo de **Dominó com Compra**, e resultados similares entre as técnicas estudadas no caso do **Dominó Ponta de Cinco**.

Palavras-chaves: Inteligência Artificial, Aprendizado de Máquina, Teoria dos Jogos, Aprendizado por Reforço.

Abstract

The game of Dominoes has many features that make it an interesting model for the fields of Artificial Intelligence(AI), Machine Learning(ML), and Game Theory(GT). Its partial observability and stochasticity present at the boneyard are some examples. The scarce application of traditional techniques from AI and the cited fields in the game of Dominoes motivate a comparative study for this problem. Two of the most common variants of Dominoes are explored, **Draw Dominoes** and **All Five**

This work studies the available solutions in the AI literature, most specifically from Reinforcement Learning and Search. Techniques from these fields are evaluated along with rule-based ones, through experimentals comparisons. The results shows that the best performance in **Draw Dominoes** comes from Search-based algorithm, *Perfect Information Monte Carlo (PIMC)*, and in **All Five** similar results have been obtained between the studied techniques.

Keywords: Artificial Intelligence, Machine Learning, Game Theory, Reinforcement Learning.

Lista de ilustrações

Figura 1 – Conjunto de Dominó duplo-seis ordenado por valor.	15
Figura 2 – Jogador bloqueado	16
Figura 3 – Spinner no centro, peça (2,2)	17
Figura 4 – Processo de Decisão de Markov	21
Figura 5 – Iteração de Política Generalizada	24
Figura 6 – Jogada que teria 2 ações (1 a direita e 1 a esquerda) é resumida somente ao após-estado resultante	25
Figura 7 – Rede Neural Perceptron	25
Figura 8 – Árvore Binária	28
Figura 9 – Árvore Minmax com recompensas nas folhas	31
Figura 10 – Encontrando movimento ótimo	32
Figura 11 – Podagem AlphaBeta	33
Figura 12 – Representação visual das etapas	34
Figura 13 – Comparações entre Jogador-AR vs Jogador-AlphaBeta nas duas varia- ções de dominó	40
Figura 14 – Comparações entre Jogador-AlphaBeta vs Jogador-MCTS nas duas variações de dominó	41

Lista de tabelas

Tabela 1 – Jogador-AR vs Jogador-Regra	39
Tabela 2 – Taxas de Recompensas(Porcentagem) em Torneio Round-Robin no Dominó com Compra. Ordem por linha	42
Tabela 3 – Taxas de Recompensas (Porcentagem) em Torneio Round-Robin do Dominó Ponta de Cinco. Ordem por linha	43

Lista de Algoritmos

3.1	Sarsa com aproximador de função	27
4.1	Minmax	31
4.2	AlphaBeta Algoritmo	32
4.3	BUSCA MCTS	35
4.4	Perfeita Informação Monte Carlo	36
5.1	Algoritmo baseado em regra para Dominó com Compra	38
5.2	Algoritmo baseado em regra para Dominó Ponta de Cinco	38

Lista de abreviaturas e siglas

AR	Aprendizado por Reforço
IP	Imperfeita Informação
IA	Inteligência Artificial
MCTS	<i>Monte Carlo Tree Search</i>
MC	Monte Carlo
MDP	<i>Markov Decision Process</i>
PI	Perfeita Informação
PIMC	Perfeita Informação Monte Carlo
TD	<i>Temporal Difference</i>
UCT	<i>Upper Confidence bound applied to Trees</i>

Sumário

1	INTRODUÇÃO	13
1.1	Objetivos	13
1.2	Organização	14
2	DOMINÓ E TRABALHOS RELACIONADOS	15
2.1	O jogo de Dominó	15
2.2	Regras	16
2.3	Trabalhos Relacionados	18
2.3.1	Complexidade	18
2.3.2	Implementação	18
3	APRENDIZADO POR REFORÇO	20
3.1	Notações	20
3.2	Geral	21
3.2.1	Função Valor	22
3.2.2	Avaliação de Política	22
3.2.3	Melhoramento de Política	23
3.2.4	Iteração de Política	23
3.2.5	Após-Estado	24
3.2.6	Aproximação de Função	24
3.3	Características Binárias	26
3.4	Sarsa	26
4	BUSCA	28
4.1	Busca Clássica	28
4.2	Busca com Adversários	29
4.3	MinMax	30
4.4	Podagem Alpha-Beta	31
4.5	Aprofundamento iterativo	33
4.6	Busca em Árvore Monte Carlo	33
4.7	Busca Monte Carlo em Perfeita Informação	35
5	EXPERIMENTOS	37
5.1	Avaliador Baseado em Regra	37
5.2	Avaliação em Perfeita Informação	38
5.2.1	Treinamento em Diferença Temporal	39

5.2.2	Comparações em Perfeita Informação	39
5.3	Avaliação em Imperfeita Informação	41
5.3.1	Comparações em Imperfeita Informação	41
6	CONCLUSÃO E TRABALHOS FUTUROS	44
	REFERÊNCIAS	45

1 Introdução

O uso de jogos como área de aplicação para técnicas de *Inteligência Artificial* (IA) é uma prática bem estabelecida na comunidade científica (YANNAKAKIS; TOGELIUS, 2015). Jogos de estratégia como Xadrez, Go e Bridge ofertam ambientes simples mas desafiadores, motivando a criação de algoritmos cada vez mais robustos (LEVY, 1988; HERIK; UITERWIJK; RIJSWIJCK, 2002). De um lado, Xadrez e Go compartilham características similares: são jogos onde todas as peças são visíveis para os jogadores durante a partida. Bridge no entanto, é um jogo no qual os jogadores não sabem quais são as cartas na mão do oponente até o momento em que elas são reveladas. Com base nessa características, Xadrez e Go são chamados jogos de *Perfeita Informação*, enquanto jogos como Bridge são chamados de *Imperfeita Informação*. Essa distinção é importante pois ela naturalmente leva a diferentes problemas e abordagens dependendo do tipo do jogo em questão (SCHAEFFER; HERIK, 2002).

Este trabalho tem como ambiente de estudo o jogo de Dominó, muito conhecido no Brasil e em toda América latina como um jogo recreativo e competitivo. Ele é considerado um jogo de imperfeita informação, com diversas variações em suas regras e componentes.

1.1 Objetivos

Este trabalho tem como objetivo principal:

- A implementação de técnicas do campo da Inteligência Artificial a variações comuns do jogo de Dominó e suas subseqüente avaliações através de confrontos diretos entre as técnicas .

Especificamente temos os seguintes 3 tipos de abordagens que serão estudadas para o desenvolvimento das IAs:

- **Aprendizado de Máquina:** Serão desenvolvidos jogadores de Dominó por meio do Aprendizado por Reforço (AR)
- **Busca:** Serão desenvolvidos jogadores com algoritmos de busca conhecidos.
- **Regras:** Serão desenvolvidos jogadores que utilizam conhecimento humano explícito do jogo, em forma de regras.

Por fim, o propósito maior não se trata de criar uma IA que jogue bem Dominó, mas sim de avaliar como as técnicas consideradas estado da arte, em outros jogos de perfeita e imperfeita informação, lidam com o Dominó.

1.2 Organização

O trabalho é organizado do seguinte modo: No capítulo 2 tem-se uma introdução ao problema que será abordado neste trabalho, o jogo de Dominó. As regras das variações abordadas para este trabalho são apresentadas, sendo definido duas variações de Dominó para estudo, bem como uma breve análise da complexidade e os desafios impostos por eles. Um levantamento dos trabalhos relacionados é apresentado em seguida.

No capítulo 3 aborda-se a técnica de Aprendizado por Reforço e sua modelagem como solução ao jogo de Dominó. No capítulo 4 aborda-se as técnicas de que se utilizam de busca, também chamadas de *planejamento*.

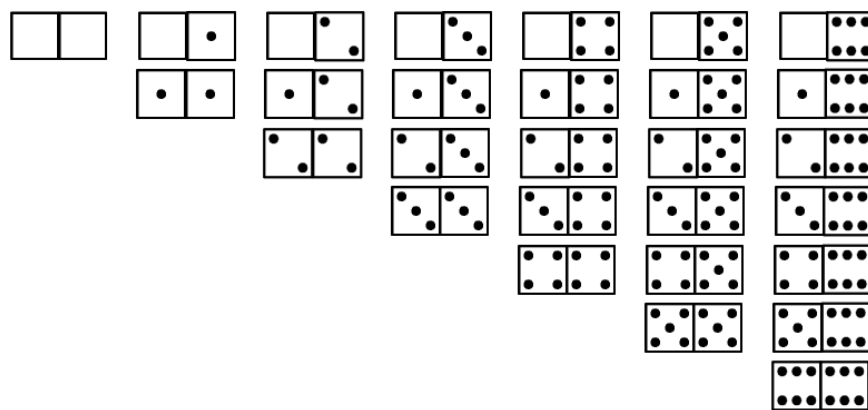
O capítulo 5 traz a toda a experimentação realizada com o objetivo de avaliar as técnicas estudadas nas seções anteriores. São analisados as duas variações de Dominó em nas diferentes versões de perfeita informação e imperfeita informação, para as quais também são desenvolvidas soluções baseadas em regras. Por fim, termina-se com as conclusões do trabalho no capítulo 6.

2 Dominó e Trabalhos Relacionados

2.1 O jogo de Dominó

Nesta tese, estuda-se um jogo muito popular no Brasil e América Latina, o *Dominó* (KELLEY; LUGO, 2003). Com sua origem sendo traçada ao Egito Antigo e China Medieval, a história desse jogo é longa e sujeita a algumas controvérsias quanto ao local de surgimento. Nos tempos atuais, o *Dominó*¹ é jogado em níveis amador e profissional, com competições oficiais sendo organizadas por clubes e federações ao redor do mundo (I.P.; MEDIA, 2016). Para jogá-lo é necessário um *conjunto de dominós*, onde cada peça² de Dominó é um retângulo 1x2, com frente e costa, onde a frente é dividido em duas seções, e as costas sem qualquer divisão ou elementos, com sua única utilidade sendo esconder a frente. As duas seções da frente são marcadas por *pontos*, os quais indicam um valor segundo sua quantidade (1 ponto equivale ao valor 1), e no caso de não te-los, se considera o valor 0. Um conjunto tradicional de dominós contém 28 peças, chamado de *Duplo-6* onde cada peça pode ter valores variando de 0 a 6 nas duas seções, e todos são únicos quanto a combinação das seções, como ilustrado na Figura 1. Não é incomum jogar com conjuntos maiores de peças, como o *Duplo-9* que contém 55 peças, ou o *Duplo-12* com 91 peças.

Figura 1 – Conjunto de Dominó duplo-seis ordenado por valor.



Fonte – Acervo do Autor

¹ De agora em diante, o termo **Dominó** será usado para designar o jogo em si, e **dominó** para se referir a uma peça do jogo.

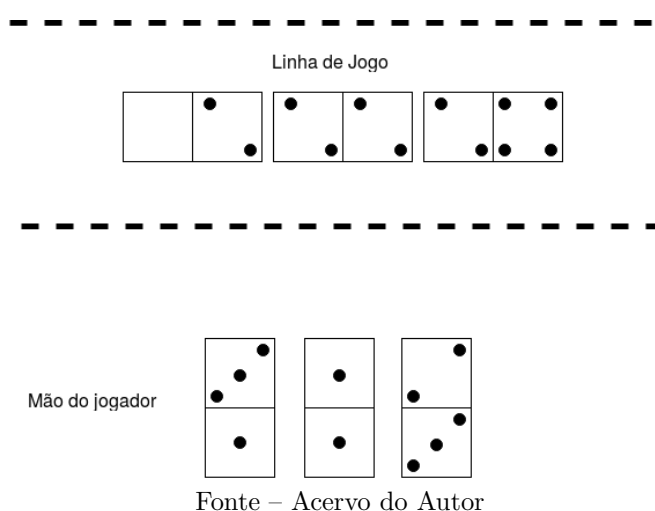
² um **dominó** também será referido como pedra,peça.

2.2 Regras

Dominó, em sua variante mais tradicional, é um jogo competitivo onde duas pessoas jogam com um conjunto *Duplo-6*. Todos os dominós são postos virados pra baixo (de costa) e embaralhados. Cada jogador toma para si 7 peças, constituindo suas *mãos* iniciais. As peças restante são mantidas viradas para baixo, e formam o *monte de compra*. Para facilitar na implementação dos experimentos, considerou-se o *monte de compra* como um conjunto ordenado, o que não modifica significativamente a jogabilidade. Cada jogador pode ver as peças em sua *mão*, mas não pode ver as frentes das peças do oponente, só se podendo saber a quantidade de peças que ele possui.

Como primeiro movimento, um dos jogadores escolhe um dominó de sua mão, e o põe na *mesa* virado para cima. Em seguida, os jogadores vão em turnos pondo as peças nas *extremidades*, um de cada vez, de tal modo a formar uma *linha de jogo*. A *linha de jogo* consiste de uma sequência de dominós, onde cada peça obrigatoriamente deve ter os lados equivalendo em valor (número de pontos) com os lados adjacentes dos dominós vizinhos. Logo, combina-se o lado equivalente da peça jogada à extremidade. Caso não se tenha peças na mão que combinem com as extremidades, como ilustrado na Figura 2, diz-se estar *bloqueado*, e se passa a vez. O jogo termina quando um dos jogadores acabar de jogar todos os dominós em sua mão, ou nenhum dos jogadores puder jogar mais alguma peça.

Figura 2 – Jogador bloqueado



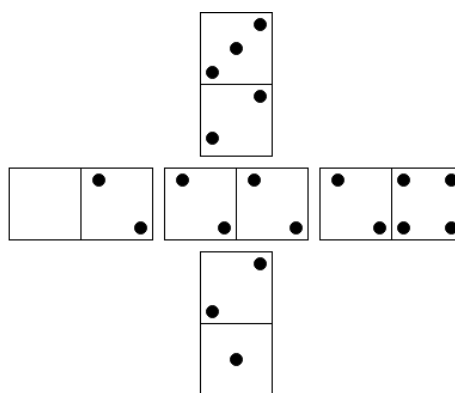
Essas são as regras mais simples e presentes na maioria das variações do Dominó. Descreve-se a seguir as regras de duas variações mais comuns ao jogo de Dominó que serão estudadas nesse trabalho:

Dominó com Compra: Nesta versão, quando o jogador é *bloqueado*, é obrigatório que se retire peças do *monte de compra*, uma de cada vez, até que venha uma peça jogável, ou não haver mais peças pra comprar. No segundo caso, passa-se a vez para o

oponente. Quando todas as peças da mão de um dos jogadores tiverem sido jogadas, ou quando ambos não puderem mais jogar nenhuma peça, o jogo termina.

A pontuação ocorre somente no fim da partida. O jogador que esvaziou primeiro sua mão é declarado vencedor, e tem como pontuação a soma dos pontos na mão do oponente, contando os dois lados de cada peça. Caso o jogo termine por não haver peças para jogar, decide-se o vencedor pelo seguinte critério de desempate: o jogador que tiver o dominó com menor valor vence (no caso de empate, vence quem tiver o menor valor de lado individual (ex. peça (0,2) vence peça (1,1)). Sua pontuação é também decidida como a soma dos pontos da mão do oponente.

Figura 3 – Spinner no centro, peça (2,2)



Fonte – Acervo do Autor

Ponta de Cinco (All Five): Essa versão de Dominó é similar ao **Dominó com Compra** pelo fato de haver compra de peças caso não se tenha dominó jogável. No entanto, ele possui duas diferenças fundamentais: (1) a primeira peça dupla a ser jogada (conhecida como *spinner*) vai poder ser combinada por 4 lados como demonstrado na Figura 3. Isso ocorre somente na primeira dupla entre todos os jogadores, adicionando um número maior de possíveis jogadas; (2) a segunda diferença é quanto a pontuação. Caso a soma das 4 (ou 2 caso uma peça dupla não tiver sido jogada) for divisível por 5, recebe-se essa soma como pontuação nessa rodada. Essa segunda regra é a que caracteriza o nome do jogo.

Ainda se recebe recompensa no final da partida pelas mesmas condições do **Dominó com Compra**, mas com uma pequena diferença: a recompensa é aproximada para o maior e mais próximo múltiplo de 5. As pontuações parciais pelas rodadas oferecem uma nova dinâmica de jogo, onde diferentes estratégias como priorizar a pontuar pela partida, e não somente ao final, são possíveis.

Em torneios ou partidas online geralmente se decide o vencedor como aquele que alcança primeiro uma quantidade específica de pontos. As mais comuns são as múltiplas de

50, como 100, 150, etc. Neste trabalho não se considera esse fator, já que só a quantidade de pontuação acumulada é o suficiente para saber se um jogador é superior.

Dominó possui diversas variações além das 2 estudadas neste trabalho. Pode-se ter mais de dois jogadores, e ao invés de somente competitivo, o jogo pode ser cooperativo entre equipes. O uso do monte de compras pode ou não ser permitido, e pode-se ter mais de um *spinner*. A contagem da pontuação pode variar, como por exemplo, o Ponta de Cinco pode ser Ponta de 3 ou Ponta de 7, etc. O trabalho se limita a versão de 2 jogadores com um conjunto de dominó *Duplo-6*, nas duas variações cujas regras foram explicadas.

2.3 Trabalhos Relacionados

2.3.1 Complexidade

A complexidade teórica do jogo de Dominó foi estudada por [Demaine, Ma e Waingarten \(2014\)](#). Eles provaram que o jogo competitivo de 2 jogadores usando um conjunto de dominós infinitos tem complexidade de *PSPACE-completo*³, quando analisado como jogo de perfeita informação. No caso de imperfeita informação, os autores mencionam, sem provas, de que o jogo poderia ser difícil como *NEXPTIME*⁴.

Visto de um ponto mais prático, existem muitos aspectos que tornam jogar Dominó um problema computacionalmente difícil. Alguns deles foram analisados por [Myers \(2014\)](#). Ele discute o alto grau de ramificação da árvore de decisões do jogo, determinado pelo número de possíveis movimentos em 1 turno. Um segundo aspecto discutido é a natureza aleatória do ato de comprar peças do monte, tornando o jogo estocástico, e cada compra adiciona probabilidades de possíveis mãos, dificultando a análise do problema. Por último, Myers menciona a não-observabilidade da mão do oponente e o monte de compra, problemas associados a jogos de imperfeita informação. Em ([ARMANINO, 1977](#)), computa o número de diferentes configurações em que as peças podem ser distribuídas entre os jogadores, obtendo um número da ordem de 10^{42} possibilidades.

Todos aspectos citados qualificam o Dominó como um ambiente desafiador para estudar e aplicar técnicas de IA.

2.3.2 Implementação

Do ponto de vista de implementação, o Dominó vem sendo estudado desde 1950 ([SMITH, 1973](#)) ([PERVIN, 1962](#)). Esses primeiros trabalhos tinham como objetivo criar programas de computadores capazes de jogar dominó ao nível profissional. Essencialmente

³ São problemas ditos mais difíceis que os famosos NP. Jogos NP são aqueles que precisamos encontrar uma solução para resolvê-los, como o cubo de Rubik. Nos jogos PSPACE precisamos encontrar um movimento que garanta vitória considerando todos os possíveis movimentos do oponente

⁴ Não podem ser solucionados em tempo logarítmico ([HARTMANIS; IMMERMANN; SEWELSON, 1985](#)).

eles faziam uso de estratégias explicitamente codificadas como conjunto de regras, que chegavam a apresentar certo grau de adaptação.

Artigos científicos mais recentes seguem caminhos similares no uso de regras. Antonio, Filho e Costa (2008) apresentam uma metodologia para escolha de ações de jogos baseadas em uma função de avaliação manualmente ajustada. Em um trabalho mais recente (ANTONIO; FILHO; COSTA, 2013), os mesmos autores experimentam com um algoritmo genético para encontrar a melhor estratégia para selecionar ações. Cruz, Guimarães e Takahashi (2013) comparam múltiplas estratégias dos vários conjuntos de regras propostos. Garza (2006) faz um estudo de avaliação de estratégias individuais para a versão de Dominó para 4 jogadores no contexto cooperativo.

A dissertação de Myers (MYERS, 2014) estuda o **Dominó Ponta de Cinco** em uma versão minimalista, com 3 peças em cada mão, e 4 no monte de compra. O objetivo é calcular o movimento ótimo nessa situação. Para tal, o autor utiliza métodos da *Teoria dos Jogos* para modelar diversos tipos de oponentes no algoritmo de Busca, e o combina com simulações *Monte Carlo* para tomada de decisão em imperfeita informação.

3 Aprendizado por Reforço

3.1 Notações

Nessa seção, introduz-se os elementos teóricos essenciais no jogo de Dominó que serão utilizados para essa e futuras seções. Tem-se as seguintes considerações:

- Ambiente: O mundo em si, que no caso é o jogo de Dominó
- Agente: São essencialmente os jogadores de Dominó, atuadores no ambiente
- Estado: Representa a informação relevante do ambiente disponível para acesso ao jogador. Cada estado deve ser distinguível dos demais, já que contém uma única combinação de informação. Estados podem ser terminais ou não-terminais, que no primeiro caso é quando chega-se em um estado ao qual não se pode sair (ex. quando o jogador ou seu oponente não tem mais peças na mão). Também podem ser objetivos ou não-objetivos, se tratando de um estado que deseja-se estar (ex. um estado que represente a situação de vitória) ou não.

No Dominó, do modo que foi modelado, existem dois tipos de estados: completo e parcial. Estados completos contém informação sobre a nossa mão, a mão do oponente, as duas extremidades e o monte de compra. A linha de jogo não é necessária, pois não adiciona nenhuma informação relevante que o agente possa usar para chegar em um estado desejável (ex. vitória). O estado parcial a que se refere, se trata do jogo de Dominó em sua versão original, onde não se sabe a mão do oponente, nem o monte de compra. Nessa versão, tem-se a mão do jogador, as duas extremidades, e o número de peças na mão do oponente e no monte de compra como fonte de informação.

- Ação: mecanismo disponibilizado pelo ambiente (jogo) para os agentes transitarem pelos estados. Nos Dominós as ações disponíveis para o jogador dependem do estado em que ele está atualmente. Há 3 tipos de ações: jogar, comprar e passar.

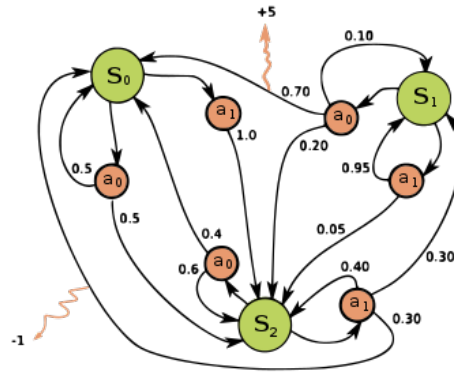
Jogar se trata de retirar uma peça na sua mão onde um dos seus lados case com uma das duas extremidades, ou 4 no caso do spinner. *Comprar* ocorre quando não se tem nenhuma peça jogável na mão, e se tem peças no monte de compra. Compra-se até que se tenha uma peça jogável. *Passar* ocorre quando não se tem peça jogável na mão, e nem quando todas as peças foram compradas.

- Recompensa: Se trata do valor obtido em um dado estado resultante de uma ação. No caso do **Dominó Ponta de Cinco**, recompensas podem ser obtidas quando as

extremidades tem uma soma total múltipla de 5. Em ambas as versões do Dominó, uma recompensa é obtida no término do jogo. Além disso, toda recompensa positiva que um jogador recebe, o outro recebe a mesma quantia como negativa, tornando o jogo como zero-soma. Todos os estados em que as situações citadas anteriormente não ocorrem, a recompensa é considerada como zero.

3.2 Geral

Figura 4 – Processo de Decisão de Markov



Fonte – (WALDOALVAREZ, 2017)

Aprendizado por Reforço (AR) é baseado na ideia de um agente descobrir o que fazer para alcançar seu objetivo a partir de iterações com o ambiente. Formalmente, o agente tem de resolver um problema na forma de um processo de decisão de Markov (MDP) (THIE, 1983). Qualquer problema que pode ser modelado como um MDP pode ser, a princípio, tratável com métodos do Aprendizado por Reforço.

Um MDP é uma 5-tupla (S, A, P, r, λ) , descrito como:

- S : conjunto de estados, com o estado s_0 como inicial.
- A : Conjunto de ações.
- $P_a(s, s')$: Função de transição de estado. Ele define a probabilidade de ser mover para um novo estado s' dado que se executou a ação a do estado atual s .
- r_t : Recompensa imediata que é recebida no tempo atual t
- λ : Fator de desconto, que permite diferenciar a importância das recompensas mais próximas, das mais distantes.

Todos os elementos (fora o fator de desconto) são ilustrados na Figura 4. Além deles, tem-se $\pi(s)$, também chamada de política, que faz um mapeamento do estado s para uma

ação a . Uma política é geralmente formalizada como uma distribuição de probabilidade que expressa a tendência de selecionar algumas ações quando em um estado particular i.e. $\pi(s, a) \equiv \text{Pr}[a_t = a | s_t = s]$. Seguir uma política π significa então tomar ações de acordo com a distribuição $\pi(a, s)$. Com base nisso, o agente tenta aprender uma política dita ótima que maximiza a recompensa acumulada, também chamada de retorno, no ambiente. Enquanto pequenos MDPs podem ser resolvidos a partir de programação dinâmica ou programação linear, ao se escalar em tamanho sua resolução vai ser tornando inviável pelos dito métodos. Aprendizado por Reforço, no entanto, tem tido bastante sucesso em gerar políticas eficientes para grandes MDPs(JIA, 2011).

Entre os métodos para solucionar tarefas de Aprendizado por Reforço, um dos mais utilizados em prática são os chamados métodos de Aprendizagem por diferença temporal (*TD-Learning*), que são populares pelas boas performances que possuem em grandes MDPs, pela sua intuitividade, e por não precisar aprender uma função de transição de estado. Os seus principais componentes são explicados nas próximas seções.

3.2.1 Função Valor

Na literatura, existem vários métodos computacionais para solucionar tarefas de Aprendizado por Reforço (SUTTON; BARTO, 1998a; SZEPEŠVÁRI, 2010; WIERING; OTTERLO, 2012). Apesar de suas diferenças, virtualmente todos são baseados em se estimar *funções de valor* sendo as principais: A *função valor-estado* $V^\pi(s) \equiv E_\pi[R_t | s_t = s]$, onde E_π é o valor esperado, R_t é o valor médio que se espera de recompensa do estado atual até o fim do episódio, conhecido como retorno. Logo a equação implica no retorno esperado do estado atual s dado que se segue uma política π ; A *função valor-ação* $Q^\pi(s, a) \equiv E_\pi[R_t | s_t = s, a_t = a]$, que por sua vez representa o retorno esperado do estado atual s , tomando-se a ação a , e seguindo-se a política π em sequência.

Essas *funções de valor* portanto medem quão bom é estar em um dado estado, ou tomar uma ação em um dado estado, quando seguindo uma política particular.

3.2.2 Avaliação de Política

Em geral, avaliação de política é o processo de se estimar a *função valor-estado* V^π para uma dada política π . Em *TD Learning*, avaliação de política é feita progressivamente ao calcular diferenças entre as estimativas V^π em 2 tempos, o atual e o próximo, e se faz uso dessas diferenças para atualizar a estimativa atual. A fórmula de atualização é definida como:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \quad (3.1)$$

onde α é o parâmetro *tamanho do passo*, uma pequena fração positiva que controla a taxa de aprendizado. Resumidamente, a racionalidade é que após uma ação a_t , tem-se

nova informação sobre a recompensa atual, obtida do estado s_t para o s_{t+1} . Essa nova informação leva a uma melhor estimativa sobre o retorno esperado do estado s_t , que é obtido ao adicionar a recompensa observada r_{t+1} , o estimado retorno esperado para o próximo estado, $V(s_{t+1})$. A subtração dessa estimativa com a estimativa para o estado atual $V(s_t)$ é também conhecida como Erro TD.

3.2.3 Melhoramento de Política

O melhoramento de política se trata do problema da escolha de uma política π de modo que ela no próximo tempo π_{t+1} seja melhor ou pelo menos igual a política no tempo atual. O modo mais simples de se garantir essa proposição é considerando:

$$\pi(s) = \operatorname{argmax}_a q(s, a) \quad (3.2)$$

onde a política escolhe de maneira gulosa a próxima ação.

3.2.4 Iteração de Política

A *iteração de política* é o uso de avaliação de política juntamente com *melhoramento de política* com o propósito de computar uma política ótima para o problema de controle. Em vez de se aprender uma função para o valor do estado, aprende-se uma função de estimação *valor-ação* Q^π para uma política comportamental π . A fórmula de atualização será:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [R_{t+1} + \gamma Q(s_{t+1}, a_t) - Q(s_t, a_t)] \quad (3.3)$$

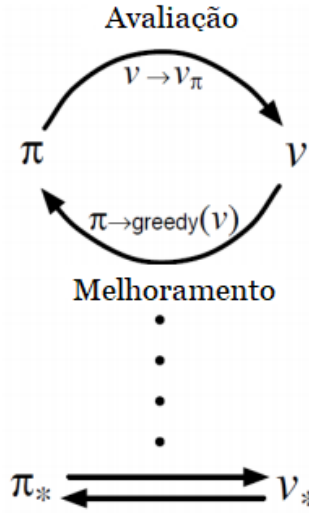
Onde a ação a é selecionada através da política π no estado s .

Considerar a política π de maneira gulosa a faz escolher as melhores ações, que vão levar a estados de melhor avaliação. Ao final ambos convergem para a política ótima π^* e *função valor-estado* ótima V^* . Esse relacionamento é chamado de *iteração de política generalizada*, e é ilustrada na figura 5.

Este método só tem resultados práticos em ambiente pequenos, como *gridworlds*. Pode-se tomar um tempo infinito para convergir em ambientes mais complexos. Grande parte desse problema se dá por não ser possível explorar todos os estados, e assim não saber uma estimação significativa da *função valor-ação* de um estado que nunca foi visitado.

Esse é um problema significativo no Aprendizado por Reforço, chamado de *multi-armed bandits* (GITTINS; GLAZEBROOK; WEBER, 2011). O problema trata, em suma, da questão de *exploration x exploitation*, ou seja, o quanto se deve dedicar a descobrir ações que podem ser boas no estado atual versus a agir de maneira gulosa tomando as decisões que atualmente são consideradas melhores. Uma das técnicas mais comuns para lidar com esse problema é utilizar a política *e-greedy*, que age de maneira gulosa com uma probabilidade $1 - e$, e de maneira aleatória com probabilidade e .

Figura 5 – Iteração de Política Generalizada



Fonte – Adaptado de (SUTTON; BARTO, 1998b, Figura 4.7)

3.2.5 Após-Estado

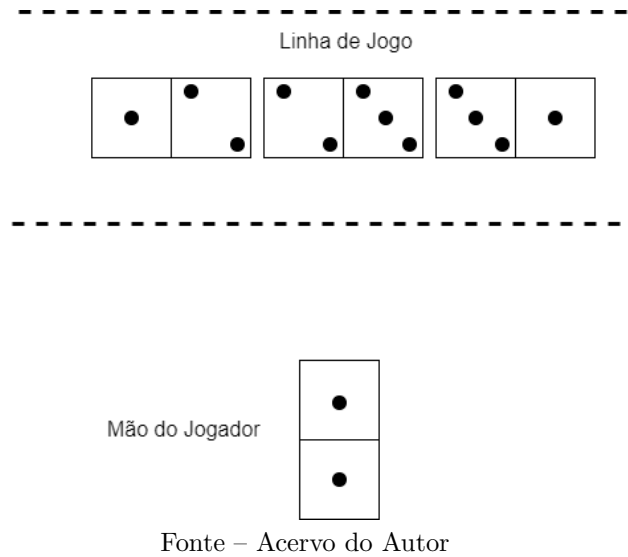
Existem outros casos de funções valores fora as de estado e ação. Jogos baseados em turnos permitem ver o estado parcial resultante da ação tomada, antes do oponente executar a sua ação, por meio de um simples simulador. Pode-se então utilizar o valor desse estado parcial, chamado após-estado, no lugar da *função valor-ação*. A vantagem dessa abordagem é demonstrada na Figura 6. Nela se vê que ao simular a jogada, é possível avaliar somente o após-estado, pois é idêntico não importando onde a peça for jogada, ao invés de ter que avaliar duas ações da função Q que resultariam nesse mesmo após-estado.

A função após-estado $f_a(s)$ representa então o estado após se executar a ação a , e antes do oponente realizar a sua. A *função valor-ação* deve ser modificada para acomodar esse conceito, demonstrado na sua nova forma $\hat{Q}(s, a) \equiv \hat{V}(f_a(s))$, que nos permite reduzir mapeamentos para ações que resultariam no mesmo após-estado.

3.2.6 Aproximação de Função

É necessário que se represente as funções *valor-estado* e *valor-ação* por algum meio. O uso de tabelas que armazenam todos os valores possíveis dessas funções é uma forma simples de representá-las, mas vai se tornando inadequada pelo crescimento exponencial a medida que os ambientes vão se tornando maiores. Enquanto jogos mínimos como o jogo-da-velha tem um número de estados armazenável em uma tabela, outros jogos que também aparentam simples, como poker, tem acima de bilhões de estados (JOHANSON, 2013). Portanto é necessário utilizar métodos aproximados como forma de representar as

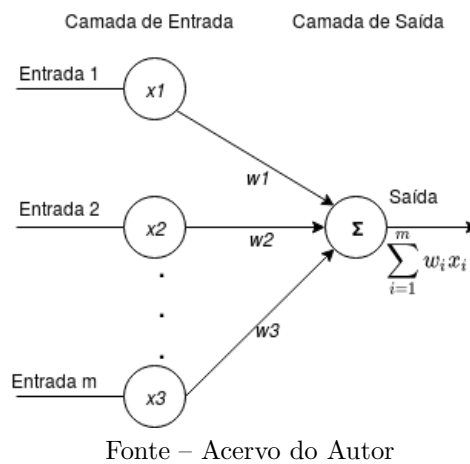
Figura 6 – Jogada que teria 2 ações (1 a direita e 1 a esquerda) é resumida somente ao após-estado resultante



funções valor.

Para esse trabalho utilizou-se uma simples rede neural perceptron, ou regressor linear, composto de um conjunto de nós de entradas e seus pesos. A entrada é definida por um vetor de *características binárias*, que é combinado com os pesos por produto escalar gerando em sua saída uma estimativa da *função valor-após-estado*. Sua arquitetura é mostrada na Figura 7, contendo somente a camada de entrada e o valor de saída. Esse simples modelo já obteve grande sucessos, como no jogo de *skat* (BURO et al., 2009), e sua performance depende primariamente das características escolhidas.

Figura 7 – Rede Neural Perceptron



3.3 Características Binárias

As características binárias que servem de entrada para o aproximador de função podem ser descritas para o jogo de perfeita e imperfeita informação. Elas devem conter as informações mais relevantes para um agente em aprendizado. Primeiramente definiu-se as características comuns a ambas as variações de Dominó e as versões de perfeita e imperfeita informação para o jogador em questão e seu oponente:

- Características para cada peça possível na mão do jogador (de 0 - 28).
- Características para a quantidade de duplas na mão do jogador (de 0 a 7).
- Característica condicional para caso a peça a ser jogada forçar o oponente a comprar.
- Características para cada peça possível nas extremidades (0 a 6 para cada extremidade aberta).

As características particulares ao **Dominó Ponta de Cinco**;

- Características para as pontuações possíveis da peça a ser jogada (5,10,15,20,25+).
- Característica indicando se há um *spinner* no estado atual.

As características particulares ao jogo em perfeita informação;

- Características para cada peça possível na mão do oponente (de 0 - 28).
- Características para cada peça possível no monte de compra (de 0 - 28).

As características particulares ao jogo em imperfeita informação;

- Características para cada número possível de peças na mão do oponente (de 1 - 21).
- Características para cada número possível de peças no monte de compra (de 0 - 14).

3.4 Sarsa

Métodos de controle para encontrar uma política ótima foram desenvolvidos como online e offline, chamados *Sarsa* e *Q-Learning* respectivamente. *Sarsa* é chamado online porque seu processo de decisão é dependente da política sendo seguida, estando ciente dos custos graças aos passos de exploração. *Sarsa* é o algoritmo escolhido para método de controle neste trabalho, com sua fórmula de atualização sendo a mesma definida em (3.3). O pseudo-algoritmo é descrito como:

Algoritmo 3.1 Sarsa com aproximador de função

```
1: função SARSA
2: Inicialize  $\theta$  parâmetros
3: Repita (para cada episódio)
4:   Inicialize estado  $s_0$ 
5:   Repita (para cada passo  $t$  no episódio)
6:     Selecione ação  $a$  da política  $\pi(s_t)$ 
7:     Execute  $a$ , observe novo estado  $s_{t+1}$ , recompensa  $r$ 
8:      $\delta_t = r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)$ 
9:      $\theta_{t+1} = \theta_t + \alpha \delta_t$ 
10:     $s_t = s_{t+1}$ 
11: fim função
```

Onde θ é a função aproximadora e seus parâmetros são os pesos da rede neural. $\hat{Q}(s_t, a_t)$ vai ser então a estimativa *após-estado* da rede neural sobre a ação a no estado s no tempo t .

4 Busca

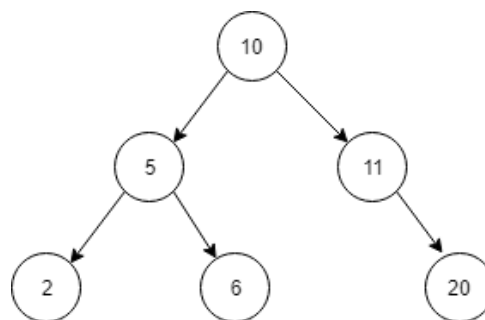
4.1 Busca Clássica

Em muitos problemas, pode ser necessário e prudente que o agente trate de buscar no espaço de estados possíveis o seu particular estado objetivo. Se é dado um modelo completo do ambiente para o agente, ele pode buscar uma sequência de ações que levem ao estado desejado antes de ter que realizar sua ação atual. Esses tipos de buscas também são conhecidas como *planejamento* na Inteligência Artificial (BONET; GEFFNER, 2001).

As buscas mais clássicas são aquelas que podem ser representadas por grafos. Nelas, o agente tenta descobrir um caminho que condiz com a particularidade de seu problema, como buscar o caminho mais curto, ou de menor custo, para um nó alvo.

Vários tipos de busca podem surgir ao se modificar as estruturas que contém os estados. Árvores são um tipo especial de grafo, onde cada nó pode ter um ou mais filhos, mas todos obrigatoriamente tem somente um pai. Um dos exemplos mais comuns são as árvores binárias, onde cada nó pode ter no máximo 2 filhos, e os novos nós são geralmente inseridos na esquerda se menor que o no folha atual ou a direita caso contrário, gerando uma árvore como a representada na Figura 8.

Figura 8 – Árvore Binária



Fonte – Acervo do Autor

Dois dos algoritmos de busca mais comuns para árvores são as buscas em profundidade e largura. No primeiro caso, a busca explora o máximo possível em cada ramo antes de retornar (*backtracking*), enquanto que no segundo a busca ocorre por níveis, onde o nível pode ser, por exemplo, todos os filhos do nó raiz, e após completa-lo explora-se o nível 2 com todos os filhos dos filhos do nó raiz.

As buscas citadas até agora são consideradas *busca desinformada*, pois não utilizam qualquer tipo de conhecimento sobre o problema em que buscam. Em jogos complexos a

busca pode se tornar muito extensa, tendo que se considerar um limite de profundidade à busca, e tratar o nó alcançado nesse limite como um nó folha, utilizando uma função de avaliação $H(s)$ que determina o valor heurístico desse estado. Esse método também é chamado *Busca Informada*, e o valor heurístico é uma medida da rentabilidade dessa posição. Buscas também podem ser mais rápidas e precisas ao serem auxiliadas por um *conhecimento direcionador* que guia o modo como ela é realizada, tal como escolher o próximo nó a se explorar dentre vários, ou decidir que ramos abandonar nessa exploração.

A busca clássica tem as mais diversas aplicações em campos além da Inteligência artificial, e pode ir além de estruturas como grafos e árvores. A introdução básica vista aqui serve como os blocos iniciais para se entender como a busca pode ser aplicada nos Dominós.

4.2 Busca com Adversários

Ambientes como jogos na maioria das vezes possuem múltiplos jogadores. A busca clássica no entanto, não modela o comportamento de outros possíveis agentes. Para incorporar esse novo aspecto, é necessário auxílio de uma abordagem que considere interação desses agentes, o que é o caso para a Teoria dos jogos.

A Teoria dos jogos tem ramos nos campos da Economia e Matemática, e na Inteligência artificial sua contribuição é evidente na área de sistemas multi-agente. Os agentes são ditos *auto-interessados*, ou seja, cada um tem seus próprios interesses e motivações, que vão levar em conta quando interagirem com outros agentes. Algumas das principais noções que a Teoria dos Jogos nos dá sobre o ambiente são de suma importância para este trabalho, em especial as seguintes noções: Jogos **zero-soma**, **Equilíbrio Nash** e Jogos de **Perfeita Informação** e **Imperfeita Informação**

Jogos de **zero-soma** se referem aqueles que o ganho de um jogador é a perda de outro. Mais especificamente, a soma dos ganhos e perdas dos jogadores deve sempre resultar em 0. Como exemplo, em um jogo de xadrez, existe um vencedor e um perdedor, e se considerar a vitória ter valor 1 e a derrota valor -1, a soma no final será 0. Nesse trabalho considera-se o Dominó de dois jogadores como sendo soma-zero, onde a pontuação final do vencedor é recebida negativamente pelo perdedor, idem para a pontuação no meio do jogo para o **Dominó Ponta de Cinco**.

Equilíbrio Nash(NASH et al., 1950) define o movimento ótimo que um jogador pode fazer no estado atual. Para jogos competitivos de 2 jogadores, o jogador 1 deve levar em consideração a melhor jogada que o jogador 2 pode fazer enquanto esse jogador 2 considera o raciocínio do jogador 1. Quando os jogadores calcularam a melhor ação a se tomar, considerando que o oponente está seguindo esse mesmo pensamento, diz-se que o jogo está no Equilíbrio Nash, onde nenhuma jogador se beneficia de mudar sua estratégia.

Jogos ditos de **Perfeita Informação** são aqueles onde o agente tem toda a informação disponível sobre as ações e estados diferentes desde o princípio do jogo até o estado atual. Uma visão menos formal é que se trata de uma situação onde o agente tem informação o suficiente para poder decidir sobre um movimento ótimo. Jogos de **Imperfeita Informação** no entanto, são todos os jogos que não condizem com a declaração anterior. Neles, informações sobre o estado ou as ações tomadas possuem elementos escondidos. Essa situação acaba por dificultar a tomada de decisões ótimas do agente pela falta de informações essenciais.

Outros fatores importantes para se destacar são o fato do jogo poder ser determinístico, onde uma ação em um dado estado s_1 sempre vai resultar no estado s_2 , e não-determinístico, como em jogos que existem elementos de chance(jogar de dados). Jogos podem ser baseados em turnos, onde cada ação é tomada em sequência, ou simultâneo, onde todas as ações são tomadas ao mesmo tempo na rodada. E por fim, jogos podem ter múltiplos jogadores, onde nem todos são opositores, e em alguns sendo possível construir coalizões(BRANDENBURGER, 2007).

No que diz respeito as duas variações de Dominó neste trabalho, ambas são por turnos, contendo elementos não-determinísticos. São estudadas as versões de **Perfeita** e **Imperfeita** informação. A seguir definiu-se os principais algoritmos de buscas utilizado para o Dominó.

4.3 MinMax

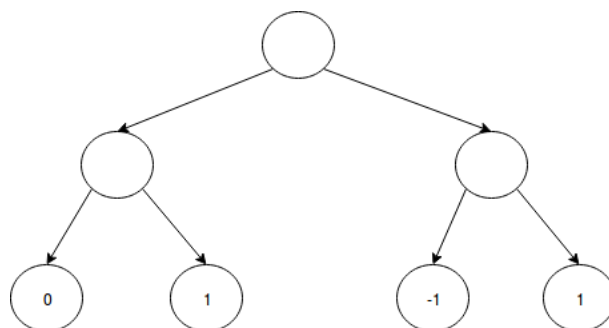
Algoritmos de busca tem sido componentes essenciais nos avanços da IA em jogos. Para jogos de 2 jogadores o algoritmo de *minimax*, ou alguma variação sua, são uma das principais técnicas utilizadas. Um exemplo clássico é no xadrez, onde o uso de variantes do *minimax* em conjunto com funções de avaliação heurísticas geraram a primeira IA capaz de vencer um campeão humano(CAMPBELL; HOANE; HSU, 2002).

O *minimax* é uma busca em árvore não-binária, que explora todas as possíveis ações dos jogadores a partir do estado atual até uma profundidade definida, ou um tempo limite estabelecido. Como exemplo, na Figura 9 um árvore de busca é construída, com as recompensas representadas nos nós folhas, sendo +1 se o jogador em questão tenha vencido, -1 se tiver perdido, e 0 em caso de empate.

O algoritmo é dado em 4.1. Ele é garantido de obter o equilíbrio *Nash* para jogos determinísticos de 2 jogadores, se para o jogo em questão esse equilíbrio existir. A função AVALIACAO retorna a recompensa particular do jogo caso for um nó folha, caso contrário ela retorna uma avaliação heurística sobre o valor do nó/estado alcançado.

Para melhor entender como o algoritmo está calculando o movimento ótimo, a

Figura 9 – Árvore Minmax com recompensas nas folhas



Fonte – Acervo do Autor

Algoritmo 4.1 Minmax

```

1: função MINMAX(jogadorMax,s,d)
2:   se s for terminal or d = 0 retorne AVALIACAO(s)
3:   se jogadorMax
4:     melhor_r := -inf
5:     para cada próximo estado s' de s
6:       r = MINMAX(True,s',d-1)
7:       melhor_r = MAX(melhor_r,r)
8:   senao
9:     melhor_r := +inf
10:    para cada próximo estado s' de s
11:      r = MINMAX(False,s',d-1)
12:      melhor_r = MIN(melhor_r,r)
13:   retorne melhor_r
14: fim função

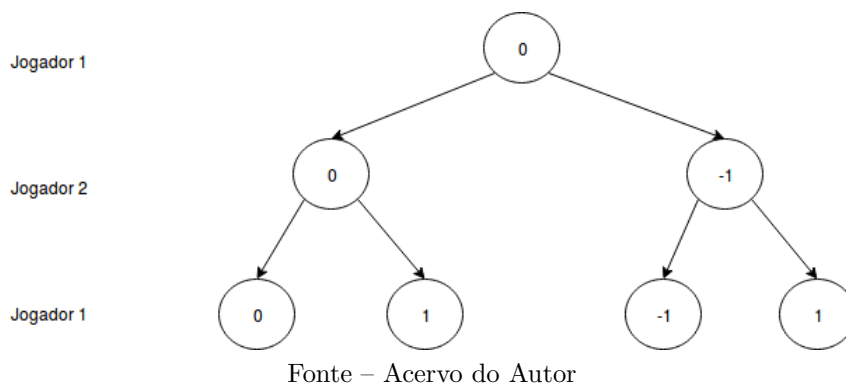
```

Figura 10 foi montada com um exemplo básico de jogo, onde as recompensas são 1,-1,0 pela vitória, derrota e empate respectivamente. Uma busca tendo limite de profundidade 3 é realizada pelo jogador 1. Ele tem a escolha de poder maximizar os resultados vindo dos nós nível do jogador 2. O jogador 2 também tem o mesmo raciocínio, o que equivale a minimizar a recompensa dos nós nível do jogador 1. Por isso, dentre 0 e 1 o jogador 2 escolhe 0, e dentre -1 e 1 ele escolhe -1. O jogador 1 só pode decidir entre 0 e -1, e para maximizar seus resultados ele escolhe o primeiro. A jogada considerada ótima para o jogador 1 é aquela que leva ao empate, considerando que o oponente também está jogando de maneira ótima.

4.4 Podagem Alpha-Beta

A busca *minmax* pode ser considerada uma abordagem bruta-força, pois considera todos os nós possíveis que podem ser tomados. Existem métodos que tentam diminuir essa busca exaustiva por meio de algum conhecimento a priori. O algoritmo *alphabeta* é uma versão do *minmax* que se utiliza de conhecimento em nós já visitados para podar certos

Figura 10 – Encontrando movimento ótimo



ramos futuros na busca. Ele retorna a mesma ação que o *minimax*, mas muitas vezes em bem menos tempo.

Algoritmo 4.2 apresenta o pseudo código do *alphabeta*. Vê-se que a grande diferença para o anterior é que são passados as variáveis *alpha* e *beta*, que servem para cortar a busca no momento em que se sabe quando um ramo vai ser efetivamente pior do que os já explorados, evitando assim um esforço adicional.

Algoritmo 4.2 AlphaBeta Algoritmo

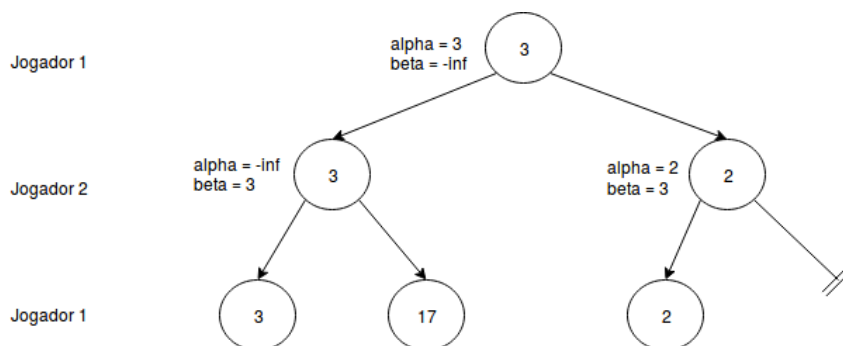
```

1: função AB(jogadorMax,s,d,alpha,beta)
2:   se s for terminal or d = 0 retorne AVALIACAO(s)
3:   se jogadorMax
4:     melhor_r := -inf
5:     para cada próximo estado s' de s
6:       r = AB(True,s',d-1,alpha,beta)
7:       melhor_r = MAX(melhor_r,r)
8:       alpha = MAX(melhor_r,alpha)
9:       se alpha >= beta
10:        retorne melhor_r
11:   senao
12:     melhor_r := +inf
13:     para cada próximo estado s' de s
14:       r = AB(False,s',d-1,alpha,beta)
15:       melhor_r = MIN(melhor_r,r)
16:       beta = MIN(melhor_r,beta)
17:       se alpha >= beta
18:        retorne melhor_r
19:   retorne melhor_r
20: fim função
  
```

Na Figura 11 tem-se o funcionamento do *alphabeta* em prática. Note que a visita aos nós sempre se dá da esquerda para a direita.

Para melhor entendimento e detalhes em possíveis melhoramentos adicionais o (KNUTH; MOORE, 1975) é indicado.

Figura 11 – Podagem AlphaBeta



Fonte – Acervo do Autor

4.5 Aprofundamento iterativo

Um ultimo melhoramento que deve ser comentado é o do aprofundamento iterativo. Em casos reais, a busca em árvore tem que ser feita sobre limitações de tempo, que podem diferir de jogo pra jogo, mas geralmente é em torno de uns poucos segundos. Para alguns jogos a busca a certos ramos pode ocupar muito tempo se ele for particularmente longo, evitando que outras soluções boas e mais curtas para explorar, porém a direita desses ramo, seja encontrada. Aprofundamento iterativo auxilia nesse problema ao ir gradualmente aumentando a profundidade limite que deve ser explorada, começando de 1. Foi demonstrado por (KORF, 1985) que o aumento adicional na complexidade não vai ser tão impactante. Na prática, muitas *engines* de xadrez e outros jogos limitados por tempo utilizam esse método para melhorar sua performance. Nesse trabalho a busca *minmax* com *alphabeta* são feitas utilizando esse melhoramento.

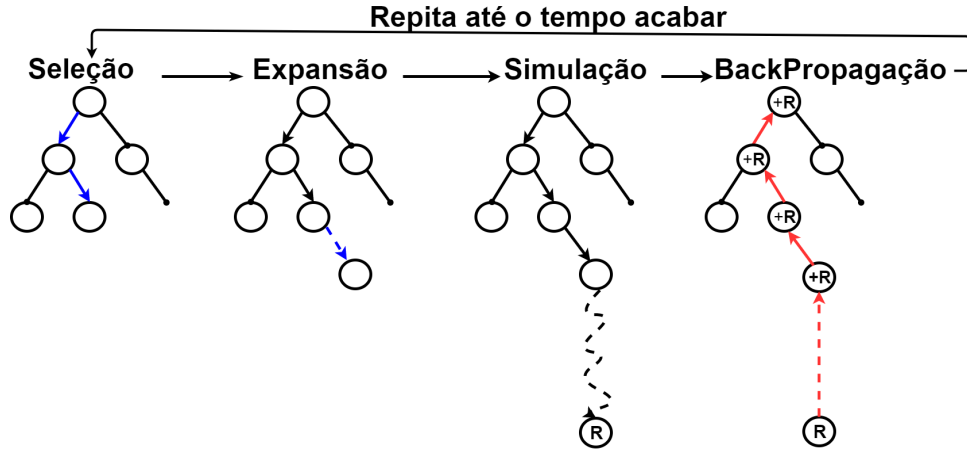
4.6 Busca em Árvore Monte Carlo

Métodos Monte Carlo (MC) possuem a fascinante característica de encontrarem respostas a problemas por meio de amostragem aleatória e simulações. A Busca em Árvore Monte Carlo (*MCTS*) aqui descrito, faz completo uso dessa habilidade dos métodos MC para construir uma árvore de busca eficiente.

A popularidade do *MCTS* vem de sua eficiência no jogo de *Go* (GELLY; SILVER, 2011), em especial no sucesso recente do *AlphaGo* (SILVER; HASSABIS, 2016) que o utiliza junto com redes neurais profundas. Mas não é somente no *Go* que *MCTS* tem se destacado. Em outros jogos (WINANDS, 2017) (WINANDS; BJORNSSON; SAITO, 2010) essa técnica tem tido significativo sucesso, e sua aplicabilidade se estende até mesmo em jogos de imperfeita informação (STURTEVANT, 2008) (COWLING; POWLEY; WHITEHOUSE, 2012).

Essencialmente um algoritmo de simulação, ele constrói uma árvore de busca por

Figura 12 – Representação visual das etapas



Fonte – Acervo do Autor

partes, divididas como:

1. Seleção: Nessa fase, apenas nós já adicionados podem ser visitados. Ela lida com o problema de *exploration vs exploitation* aplicando a política UCT(KOCSIS; SZEPESVÁRI,), adaptada para problemas com adversários(GELLY; SILVER, 2011), onde se tem a garantia de convergência para a solução *minmax*(KOCSIS; SZEPESVÁRI,). A equação para decidir qual o próximo nó a se visitar é dada como:

$$Q(s, a)^{(+)} = Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}}. \quad (4.1)$$

onde $Q(s, a)$ é a *função valor-ação*, ou seja, o valor espera para tomar a ação a no estado s . $N(s)$ nos dá o número de vezes que o nó com o estado s já foi visitado, e $N(s, a)$ denota o número de vezes que o nó filho alcançado do estado s com a ação a foi visitado. C é uma constante experimental que pode ser ajustada de acordo com o problema. A seleção para quando encontrar algum nó terminal, ou quando alcançar um nó onde todos os filhos ainda não foram explorados.

2. Expansão: Essa fase decide quantos filhos serão adicionados ao nó da árvore alcançado na fase de seleção. Para este trabalho considerou-se que somente 1 nó é adicionado por iteração.
3. Simulação: A fase de simulação ocorre logo após a expansão. Uma política de simulação é utilizada para se jogar a partir do nó expandido. Para o Dominó escolheu-se utilizar uma política aleatória. Essa escolha é comum, e traz o benefício das propriedades de *Monte Carlo*. Simula-se até encontrar um estado-nó terminal, de onde se extrai a recompensa.

4. Retro-Propagação: É a etapa onde se propaga a recompensa alcançada para todos os nós visitados na seleção e o nó expandido. Cada nó então vai tirar a média da recompensa adquirida até agora, o que pode ser visto como cada nó atualizando sua *função valor-estado*. Além disso, estatística como contagem de visitas para cada nó também é atualizada.
5. Melhor-Ação: Quando o limite máximo de iterações for alcançada, ou o tempo disponível acabar, a ação a ser tomada pode ser eleita por vários critérios, os mais comuns sendo:
 - a) Nó mais visitado
 - b) Nó de maior valor

O funcionamento completo é descrito no Algoritmo 4.3.

Algoritmo 4.3 BUSCA MCTS

```

1: função MCST(estado  $s_0$ )
2:   Crie um nó raiz  $n_0$  de  $s_0$ 
3:   Enquanto há tempo disponível
4:      $NoSelecionado = Selecao(n_0)$ 
5:      $NoExpandido = Expansao(NoSelecionado)$ 
6:      $r = Simulacao(NoExpandido)$ 
7:     RetroPropagação( $r$ )
8:   retorne MelhorAcao( $n_0$ )
9: fim função
  
```

O *MCTS* pode ser considerado também como um método do Aprendizado por Reforço, onde a árvore sendo construída pode ser vista como uma *função valor-ação* parcial (GELLY; SILVER, 2011). Também é notado os conceitos de estimação de valor e melhoramento de política na construção da árvore.

4.7 Busca Monte Carlo em Perfeita Informação

Todas as buscas apresentadas até agora lidam com o jogo em perfeita informação, que é uma forma que facilita a análise e solução do jogo, mas não o representa em sua forma original. Deve-se tratar o jogo com os seus elementos de imperfeita informação. Uma das técnicas mais utilizadas para isso é a busca *PIMC* (Monte Carlo em Perfeita Informação) por meio de *determinização* (YOON et al., 2008).

Determinização se trata de transformar elementos ocultos e incertos de um jogo em elementos determináveis. No caso dos Dominós, se trata de atribuir valores as peças da mão do oponente e do monte de compra, de modo que o jogo se torne de perfeita informação.

O *PIMC*(LONG et al., 2010) se utiliza de *determinizações* para buscar por uma solução. Como não se sabe quais os valores corretos dos elementos ocultos, cria-se determinizações de maneira uniforme, que no caso do Dominó seriam peças aleatórias e válidas. Com o jogo em Perfeita Informação, pode-se aplicar as técnicas discutidas anteriormente. Itera-se repetidas vezes a solução de perfeita informação em estados determinizados, salvando o valor esperado da ação que é retornado pela técnica. Esse processo só termina quando se ultrapassar um tempo estipulado ou um número máximo de iterações. No fim, elege-se como ação final aquela que possuir maior valor esperado acumulado durante as simulações. A geração de soluções aleatórias que convergem para uma solução real é o que caracteriza esse método como *Monte Carlo*. O funcionamento completo está descrito no Algoritmo 4.4.

É possível se utilizar de inferência para diminuir a quantidade de determinizações que sabe-se serem falsas. Enquanto não é o objetivo desse trabalho se aprofundar nas técnicas de inferência e modelagem de oponentes(BILLINGS et al., 1998), inferência básica pode ser aplicada de modo trivial. Ao procurar por meios de saber que peças o oponente tem em sua mão, a única situação que garantidamente nos revela essa informação acontece na hora da compra ao monte de compra pelo oponente. Se, por exemplo, o oponente comprar uma quantidade n de peças, e jogar a peça (2,5) com o 5 sendo o elemento casando com a antiga extremidade, então sabe-se com certeza absoluta de que o oponente não tinha nenhuma peça em sua mão com o valor 5, e que nenhuma das n peças compradas tem o valor 5. Logo, no processo de determinização, as peças de valor 5 não precisam ser postas na mão do oponente, elas devem estar com certeza no monte de compra.

Por fim, apesar do *PIMC* possuir deficiências já conhecidas e estudadas como em (LONG et al., 2010), ele obteve sucesso em construir jogadores de alto nível para jogos como *phantom go* (BORSBOOM et al., 2007), GIB do bridge(GINSBERG, 2001), *skat*(BURO et al., 2009).

Algoritmo 4.4 Perfeita Informação Monte Carlo

```

1: função PIMC(Estado  $s$ ,  $N$ , ValoresAcoes  $val$ )
2:   para cada ação  $a$  em PossiveisAcoes( $s$ )
3:      $val[a] = 0$ 
4:   para  $i$  de 1 a  $N$  faça
5:      $d = \text{Determinizacao}(s)$ 
6:     para cada ação  $a$  em  $d$ 
7:        $val[a] += \text{SolucaoPI}(d,a)$ 
8: fim função

```

5 Experimentos

Nesta seção será avaliada as técnicas de Aprendizado por Reforço e Busca no contexto do jogo de Dominó. A avaliação se dará nas formas de Perfeita Informação e Imperfeita informação. A motivação para se estudar o primeiro caso é principalmente para ter uma ideia sobre a complexidade aplicada do jogo, e para gerar um oráculo para propósito de *benchmarking*. Uma outra motivação é de avaliar a performance dessas técnicas numa forma mais simples (PI), e posteriormente compara-las com o caso real (IP), para se perceber a diferença entre resultados.

A medida de avaliação se dará pela fórmula:

$$taxa_de_avaliação_jogador1 = \frac{R.C.jogador1}{R.C.jogador1 + R.C.jogador2} \quad (5.1)$$

onde $R.C.$ é a recompensa acumulada que o jogador1 obteve contra seu oponente. Essa medida será aplicada em todas as comparações pela seção. Ela basicamente nos dá a taxa da proporção de um dos jogadores na soma das recompensas acumuladas de ambos os jogadores.

5.1 Avaliador Baseado em Regra

Para melhorar o *benchmarking* das abordagens tomadas nesse trabalho, seria ideal que fossem comparadas com programas conhecidos derivados de conhecimento humano. No entanto, não foi encontrado nenhuma *engine* ou programa de dominó *opensource* muito significativo que poderia ser utilizado para as comparações. Além disso, a maioria das implementações encontradas eram: produtos comerciais fechados; programas que possuíam pequenas variações nas regras que impossibilitavam comparações diretas; produzidos sem muita clareza na técnica utilizada, dificultando uma comunicação entre programas.

A partir de livros e sites de Dominós foi identificado algumas regras básicas seguidas pelos jogadores humanos. Assim foi desenvolvido uma IA com regras para as variações estudadas neste trabalho. Os algoritmos são descritos como:

A estratégia no Algoritmo 5.1 para o **Dominó com Compra**, é de se livrar das peças de muito valor em sua mão, dando prioridade a duplas. Desse modo, mesmo se ele perder a partida, vai possivelmente ter minimizado essa perda.

No caso do Algoritmo 5.2 do **Dominó Ponta de Cinco**, ele pode ser visto como um algoritmo guloso que tenta escolher a peça que vai lhe dar mais pontuação na rodada ao compara-las com o *MAXPONTUACAO*. Caso não tenha como pontuar, chama-se a

Algoritmo 5.1 Algoritmo baseado em regra para Dominó com Compra

```

1: função REGRACOMPRA( maoJogador , monteCompra)
2:   se PEDRAJOGAVEL(maoJogador)
3:     maiorPedra = nulo
4:     para cada pedra em maoJogador
5:       se DUPLA(pedra)
6:         retorne pedra
7:       senao
8:         maiorPedra = MAXPEDRA(maiorPedra,pedra)
9:     retorne maiorPedra
10:  senao
11:    pedra = COMPRA(monteCompra)
12:    retorne pedra
13:
14: fim função

```

Algoritmo 5.2 Algoritmo baseado em regra para Dominó Ponta de Cinco

```

1: função REGRAPONTA5( maoJogador , monteCompra)
2:   se PEDRAJOGAVEL(maoJogador)
3:     melhorPedra = nulo
4:     maiorPontuacao = 0
5:     para cada pedra em maoJogador
6:       melhorPedra = MAXPONTUACAO(melhorPedra,pedra)
7:     se VALOR(melhorPedra) > 0
8:       retorne melhorPedra
9:     senao
10:      retorne REGRACOMPRA(maoJogador, monteCompra)
11:  senao
12:    pedra = COMPRA(monteCompra)
13:    retorne pedra
14:
15: fim função

```

mesma função do algoritmo de compra, que vai escolher uma dupla ou a peça de maior valor.

Ambas as estratégias são comuns em partidas humanas, e utilizadas em algum grau por iniciantes e profissionais.

5.2 Avaliação em Perfeita Informação

Nessa seção será apresentado os experimentos realizados com o jogo em Perfeita Informação. A técnica de Aprendizado por Reforço será usada para treinar um agente chamado *jogador-AR*. Além dele serão avaliados o algoritmo baseado em regras, chamado *jogador-Regra*, e as técnicas de busca, *jogador-AlphaBeta* e *jogador-MCTS* respectivamente. Todos os *jogadores* tem versões para as duas variações de Dominó estudadas nesse trabalho.

5.2.1 Treinamento em Diferença Temporal

Para o *jogador-AR*, escolheu-se treina-lo com o *Sarsa* por meio do *auto-jogo*. No *auto-jogo*, o agente joga contra um clone seu, que utiliza as mesmas configurações e função aproximadora, sem nenhum auxílio humano. A vantagem de se utilizar dessa forma de treinamento deriva de não estar dependente do nível de jogo do oponente, pois para conseguir uma política ótima real, seria necessário um oponente que teoricamente jogasse de modo ótimo. Se, por exemplo, o jogador-AR treinasse somente contra o jogador-Regra, ele só aprenderia a jogar contra esse oponente.

As configurações dos parâmetros para o *Sarsa* foram as seguintes: 0.07 como taxa de exploração ϵ , 0.99 para o fator de desconto γ .

Avaliar o agente no auto-treino é uma tarefa difícil. Contra um oponente fixo bastaria salvar a recompensa acumulada em cada episódio e verificar o crescimento em média. No entanto, no auto-jogo o agente está treinando contra um oponente que está em constante desenvolvimento, se tornando melhor ao mesmo tempo. Logo decidiu-se avaliar a política resultante após o treinamento, em confronto direto contra o *jogador-Regra*. Os resultados são mostrados na Tabela 1, onde a taxa de avaliação é apresentada em porcentagem.

Tabela 1 – Jogador-AR vs Jogador-Regra

Jogo	AR	RE
Dominó com Compra	60%	40%
Dominó Ponta de Cinco	52%	48%

Fonte – Acervo do Autor

Percebe-se pela taxa de treinamento que o *jogador-AR* desenvolveu uma estratégia superior no **Dominó com Compra** contra o *jogador-Regra*. Para o **Dominó Ponta de Cinco** no entanto, a taxa de aprendizagem converge para uma recompensa acumulada média tendendo a 50%, indicado que o jogador-AR possui o mesmo nível do jogador-Regra para essa versão. Alguns possíveis motivos para esse resultado no **Dominó Ponta de Cinco** podem ser: O jogador de regra dessa versão ser melhor que o desenvolvido para o de compra, ou talvez esse jogo ser de mais difícil aprendizado que o de compra, ou até mesmo que talvez as estratégias possíveis sejam limitadas. Essa questão será mais explorada nos próximos experimentos.

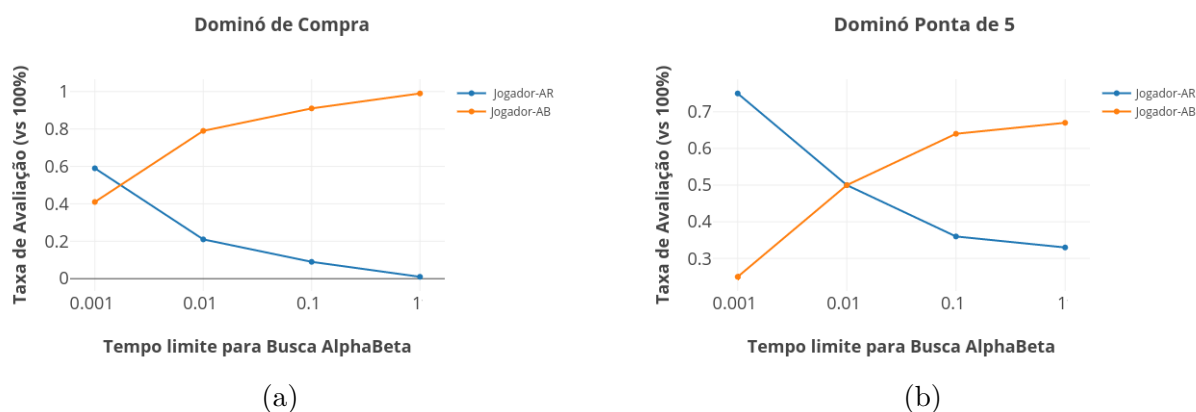
5.2.2 Comparações em Perfeita Informação

Comparou-se inicialmente o jogador-AR versus o jogador-AlphaBeta em ambos os jogos de Dominó. Como sabe-se que o jogador-AR é melhor que o jogador-Regra no

Dominó com Compra, e possui o mesmo nível no **Dominó Ponta de Cinco**, pode-se salvar um pouco de esforço em não comparar os jogadores de regra com as demais técnicas.

O experimento é feito com o jogador-AlphaBeta tendo seu tempo disponível para realizar a busca sendo variado. Variou-se no tempos de 0.001, 0.01, 0.1, e 1 segundos, e pra cada variação teve-se 1000 partidas de comparação contra o jogador-AR, com a proporção de recompensa acumulada como fator avaliativo.

Figura 13 – Comparações entre Jogador-AR vs Jogador-AlphaBeta nas duas variações de dominó



Fonte – Acervo do Autor

Na Figura 13 vê-se o resultado do experimento. No jogo de **Dominó com Compra**, o jogador-AR tem a princípio uma taxa melhor que o jogador-AlphaBeta, mas seu desempenho rapidamente se torna inferior com o aumento de tempo disponível para o jogador-AlphaBeta. A conclusão óbvia é que nessa versão do dominó, o algoritmo de busca *alphabeta*, quando em perfeita informação, vence de modo contundente contra o jogador-AR desenvolvido, e certamente o de regra.

No jogo de **Dominó Ponta de Cinco**, o resultado é bem diferente. A avaliação nos mostra que o jogador-AlphaBeta tem melhor desempenho, mas a diferença entre ambos é muito menor. Com o máximo de tempo disponível o jogador-AlphaBeta só consegue 67% na taxa de avaliação sobre o jogador-AR. Isso leva a suspeita de que essa variação não exige um planejamento muito grande de ações, sendo que só acumular pontuações quando a situação permite já é uma das melhores estratégias, como evidenciado na proximidade entre o jogador-AR e o jogador-Regra.

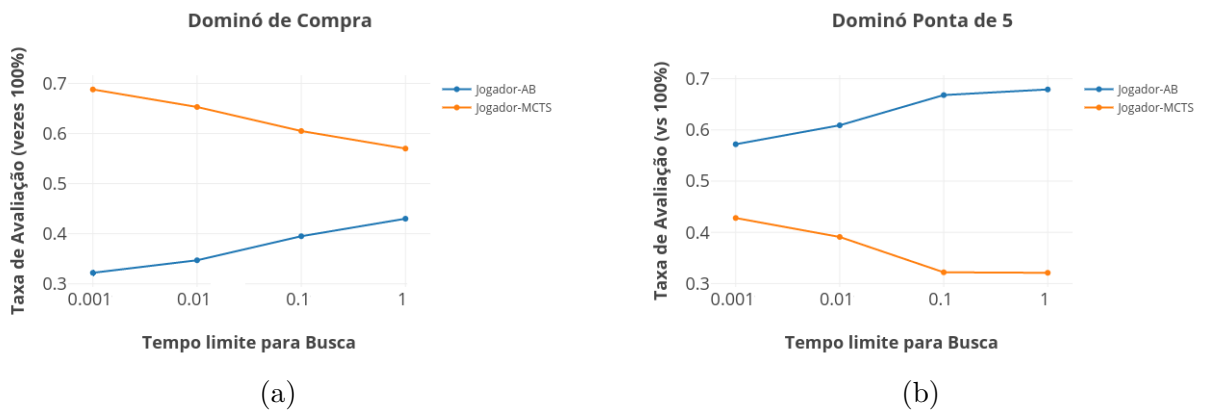
O próximo experimento é entre os jogadores de busca, jogador-AlphaBeta e jogador-MCTS. Variou-se os tempos disponíveis as buscas, de 0,001 a 1 segundo, para ambos os jogos.

O resultado do experimento é demonstrado na Figura 14. No jogo de **Dominó com Compra**, o jogador-MCTS tem um domínio inicial relevante sobre o jogador-AlphaBeta.

Esse domínio vai diminuindo com mais tempo disponível para buscar, no entanto ainda persiste. Uma possível explicação para esse fenômeno, é de que o jogador-AlphaBeta com mais tempo vai gradualmente se aproximando do equilíbrio *Nash*. O jogador-MCTS como explicado anteriormente tem garantias de convergência para o *minmax*, mas parece saber aproveitar bem melhor que o jogador-AlphaBeta quando se tem limites no orçamento de tempo disponível.

No entanto, no experimento com o **Dominó Ponta de Cinco** o mesmo não se repetiu. Vê-se que o jogador-AlphaBeta tem uma vantagem sobre o jogador-MCTS. Uma explicação para isso pode ser que as recompensas que ocorrem no meio do jogo interferem ou dificultam nas simulações *Monte Carlo*, tornando os nós com uma estimativa muito imprecisa. O *alphabeta* foi superior nessa versão de Dominó, se demonstrado a melhor técnica.

Figura 14 – Comparações entre Jogador-AlphaBeta vs Jogador-MCTS nas duas variações de dominó



Fonte – Acervo do Autor

5.3 Avaliação em Imperfeita Informação

Nessa seção avalia-se o jogo em sua forma original, de imperfeita informação, onde as peças do oponente e o monte de compra tem seus valores desconhecidos. Se treinará novamente o agente de Aprendizado por Reforço nos dois jogos de Dominó, e se avaliará o jogador baseado em regra e a técnica de *PIMC*, sob a alcunha *jogador-PIMC*.

5.3.1 Comparações em Imperfeita Informação

O jogador-AR foi treinado com as características definidas para imperfeita informação e com os mesmos parâmetros da seção anterior. O treinamento se deu do mesmo modo, por auto-jogo.

O jogador-PIMC tem 2 parâmetros principais que só podem ser escolhidos através de experimentação: A quantidade de iterações para geração de determinizações, e o quanto de tempo investir na solução de perfeita informação para cada uma dessas determinizações. No caso, para o jogo de **Dominó com Compra** a melhor solução é o jogador-MCTS, e para o **Dominó Ponta de Cinco** é o jogador-AlphaBeta. Para ambos, foi avaliado diversos números de iterações e tempos para as buscas, e chegou-se a conclusão que cerca de 100 determinizações com cada busca tendo 0.007 unidades de tempo disponível é o limite em termos de performance que o PIMC consegue alcançar.

Para avaliação dos jogadores foi realizado um torneio *round-robin* (todos contra todos) de 1000 partidas. Além do jogador-AR, jogador-Regra, jogador-PIMC, foram adicionados mais dois jogadores: jogador-Aleatório e jogador-PI. O jogador-Aleatório faz somente jogadas aleatórias, e o jogador-PI vê o jogo como perfeita informação, o que é proibido pelas regras, mas é utilizado aqui como modo de avaliação. Esses dois jogadores servem como avaliadores extremos: um corresponde a um modo amador de se jogar, e o outro a um oponente 'invencível', ou oráculo.

As comparações para o **Dominó com Compra** são demonstradas na Tabela 2, que é lida como a taxa da proporção de recompensa acumulada dos jogadores das colunas versus os jogadores das linhas.

Tabela 2 – Taxas de Recompensas(Porcentagem) em Torneio Round-Robin no Dominó com Compra. Ordem por linha

Nomes	Aleatório	Regra	AR	PIMC	PI
Aleatório	X	X	X	X	X
Regra	55%	X	X	X	X
AR	56%	52%	X	X	X
PIMC	69%	66%	65%	X	X
PI	99%	95%	95%	80%	X

Fonte – Acervo do Autor

Percebe-se que dentre os resultados apresentados, o jogador-Aleatório tem o pior desempenho, e o jogador-PI domina em todas comparações, como era de se esperar para ambos. Pode-se inferir pelos resultados que o jogador-PI é imbatível no tempo delimitado para sua busca, e com poucos indícios de que tenha algum modo de vencê-lo. O jogador-AR teve uma queda de desempenho se for comparar com sua versão em perfeita informação. Ele acaba por ter um nível similar ao jogador-Regra, mostrando que essa versão do Dominó não é uma tarefa simples de aprendizado. O jogador-PIMC tem os melhores resultados

como uma técnica válida para o jogo original, tendo uma taxa acima de 60% sobre o jogador-Regra e o jogador-AR.

Para o jogo de **Dominó Ponta de Cinco** o cenário de perfeita informação voltou a surgir. Vê-se nos resultados da Tabela 3 que as diferenças entre os jogadores AR, Regra, e PIMC são pequenas. Muito provavelmente todos eles estão convergindo para estratégias similares, que seriam parecidas as regras do jogador-Regra, onde elas dão prioridades a fazer pontuação por turnos, ao invés de ganhar a partida ao final. O jogador-Aleatório novamente foi o pior, e o jogador-PI os venceu com vantagens similares.

Tabela 3 – Taxas de Recompensas (Porcentagem) em Torneio Round-Robin do Dominó Ponta de Cinco. Ordem por linha

Nomes	Aleatório	Regra	AR	PIMC	PI
Aleatório	X	X	X	X	X
Regra	66%	X	X	X	X
AR	66%	51%	X	X	X
PIMC	69%	55%	54%	X	X
PI	81%	68%	68%	65%	X

Fonte – Acervo do Autor

6 Conclusão e Trabalhos Futuros

Nessa monografia foi realizado o estudo e aplicação de técnicas computacionais da Inteligência Artificial, de modo a avaliar a eficiência delas nas variações propostas do jogo de Dominó. Os resultados obtidos para o jogo de **Dominó com Compra** mostram que, se tratado como perfeita informação, algoritmos de busca tem larga vantagem sobre uma aplicação padrão de Aprendizado por Reforço, que é o *Sarsa* com função aproximadora linear. Na versão original do jogo, a técnica de busca *PIMC* se mostrou superior a aplicação pura do Aprendizado por Reforço, mas sem a vantagem relatada anteriormente. Avaliações com jogadores aleatório e de regra cementam a técnica do *PIMC* como a melhor para essa versão, só perdendo para as de perfeita informação, obviamente.

No **Dominó Ponta de Cinco**, os resultados em perfeita informação foram bem diferentes: a vantagem do algoritmo de busca não é nem de perto como na versão discutida anteriormente, sendo que ela consegue aproximadamente o dobro de recompensa acumulada das técnica de Aprendizado por Reforço e Regra, por volta de 66%. Presume-se então que a estratégia proposta para o jogador de Regra já é provavelmente uma das melhores a se tomar. Esse pensamento é reforçado com o resultado na versão original do jogo, onde o *jogador-pimc* teve resultados aproximados as técnicas de regra e Aprendizado por Reforço.

Conclui-se que a busca é superior no **Dominó com Compra**, e no **Dominó Ponta de Cinco** as regras propostas se mostram forte o suficiente para lidar com os demais jogadores, que provavelmente aprendem uma estratégia similar.

Espera-se que este trabalho contribua positivamente as pesquisas relacionadas a Inteligência artificial aplicada ao jogo de Dominó, e aos jogos de Imperfeita Informação no geral. As técnicas discutidas e implementadas neste trabalho podem ser de auxílio a futuras pesquisas envolvendo este tema.

Trabalho futuros incluem um estudo do efeito de técnicas de inferência podem ter no *PIMC*, o uso de outras técnicas de determinização como *Information Set SMCTS*(COWLING; POWLEY; WHITEHOUSE, 2012), *Recursive-PIMC*(FURTAK; BURO, 2013), e de Aprendizado por Reforço como *Deep-Q Learning*(MNIH et al., 2015) e *Asynchronous Actor-Critic*(MNIH et al., 2016). Outras versões conhecidas do Dominó podem ser estudadas, como o Dominó quatro pontas(ANTONIO et al., 2011), conhecido pela sua complexidade, bem como versões com múltiplos jogadores, onde é necessário aprender a cooperar em equipe.

Referências

- ANTONIO, N. S. et al. Optimization of an evaluation function of the 4-sided dominoes game using a genetic algorithm. In: IEEE. *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. [S.l.], 2011. p. 24–30. Citado na página 44.
- ANTONIO, N. S.; FILHO, C. F. F. C.; COSTA, M. G. F. Proposta de uma heurística para o jogo de domino de 4 pontas. In: *Proc. VII Brazilian Symp. Comput. Games Digit. Entertain., Comput. Track*. [S.l.: s.n.], 2008. p. 24–30. Citado na página 19.
- ANTONIO, N. S.; FILHO, C. F. F. C.; COSTA, M. G. F. Optimization of an evaluation function of the four-sided dominos game using a genetic algorithm. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 5, n. 1, p. 33–43, 2013. Citado na página 19.
- ARMANINO, D. C. *Dominoes*. [S.l.]: Cornerstone Library, 1977. Citado na página 18.
- BILLINGS, D. et al. Opponent modeling in poker. In: *AAAI/IAAI*. [S.l.: s.n.], 1998. p. 493–499. Citado na página 36.
- BONET, B.; GEFFNER, H. Planning as heuristic search. *Artificial Intelligence*, Elsevier, v. 129, n. 1-2, p. 5–33, 2001. Citado na página 28.
- BORSBOOM, J. et al. A comparison of monte-carlo methods for phantom go. In: *Proc. BeNeLux Conf. Artif. Intell., Utrecht, Netherlands*. [S.l.: s.n.], 2007. p. 57–64. Citado na página 36.
- BRANDENBURGER, A. Cooperative game theory. *Teaching Materials at New York University*, 2007. Citado na página 30.
- BURO, M. et al. Improving state evaluation, inference, and search in trick-based card games. In: *IJCAI*. [S.l.: s.n.], 2009. p. 1407–1413. Citado 2 vezes nas páginas 25 e 36.
- CAMPBELL, M.; HOANE, A. J.; HSU, F.-h. Deep blue. *Artificial intelligence*, Elsevier, v. 134, n. 1-2, p. 57–83, 2002. Citado na página 30.
- COWLING, P. I.; POWLEY, E. J.; WHITEHOUSE, D. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 4, n. 2, p. 120–143, 2012. Citado 2 vezes nas páginas 33 e 44.
- CRUZ, A. R. da; GUIMARÃES, F. G.; TAKAHASHI, R. H. Comparing strategies to play a 2-sided dominoes game. In: IEEE. *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*. [S.l.], 2013. p. 310–316. Citado na página 19.
- DEMAINE, E. D.; MA, F.; WAINGARTEN, E. Playing dominoes is hard, except by yourself. In: _____. *Fun with Algorithms: 7th International Conference, FUN 2014, Lipari Island, Sicily, Italy, July 1-3, 2014. Proceedings*. Cham: Springer International Publishing, 2014. p. 137–146. Citado na página 18.

- FURTAK, T.; BURO, M. Recursive monte carlo search for imperfect information games. In: IEEE. *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. [S.l.], 2013. p. 1–8. Citado na página 44.
- GARZA, A. G. D. S. Evaluating individual player strategies in a collaborative incomplete-information agent-based game playing environment. In: IEEE. *2006 IEEE Symposium on Computational Intelligence and Games*. [S.l.], 2006. p. 211–216. Citado na página 19.
- GELLY, S.; SILVER, D. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, Elsevier, v. 175, n. 11, p. 1856–1875, 2011. Citado 3 vezes nas páginas 33, 34 e 35.
- GINSBERG, M. L. Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, v. 14, p. 303–358, 2001. Citado na página 36.
- GITTINS, J.; GLAZEBROOK, K.; WEBER, R. *Multi-armed bandit allocation indices*. [S.l.]: John Wiley & Sons, 2011. Citado na página 23.
- HARTMANIS, J.; IMMERMANN, N.; SEWELSON, V. Sparse sets in np-p: Exptime versus nexptime. *Information and Control*, Elsevier, v. 65, n. 2-3, p. 158–181, 1985. Citado na página 18.
- HERIK, H. J. van den; UITERWIJK, J. W. H. M.; RIJSWIJCK, J. van. Games solved: Now and in the future. *Artificial Intelligence*, v. 134, n. 1, p. 277–311, jan. 2002. ISSN 0004-3702. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0004370201001527>>. Citado na página 13.
- I.P., S.; MEDIA. *Domino Club and Organisation Links*. 2016. <<http://www.domino-play.com/LinksClubs.htm>>. Accessed: 2017-04-03. Citado na página 15.
- JIA, Q.-S. On state aggregation to approximate complex value functions in large-scale markov decision processes. *IEEE Transactions on Automatic Control*, IEEE, v. 56, n. 2, p. 333–344, 2011. Citado na página 22.
- JOHANSON, M. Measuring the size of large no-limit poker games. *arXiv preprint arXiv:1302.7008*, 2013. Citado na página 24.
- KELLEY, J. A.; LUGO, M. *The Little Giant Book of Dominoes*. [S.l.]: Sterling, 2003. ISBN 1402702906. Citado na página 15.
- KNUTH, D. E.; MOORE, R. W. An analysis of alpha-beta pruning. *Artificial intelligence*, Elsevier, v. 6, n. 4, p. 293–326, 1975. Citado na página 32.
- KOCSIS, L.; SZEPESVÁRI, C. Bandit based monte-carlo planning. In: SPRINGER. *ECML*. [S.l.]. v. 6, p. 282–293. Citado na página 34.
- KORF, R. E. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, Elsevier, v. 27, n. 1, p. 97–109, 1985. Citado na página 33.
- LEVY, D. N. L. *Computer Games I, II*. New York, NY: Springer, 1988. ISBN 978-1-4613-8716-9. Disponível em: <<http://dx.doi.org/10.1007/978-1-4613-8716-9>>. Citado na página 13.

- LONG, J. R. et al. Understanding the success of perfect information monte carlo sampling in game tree search. In: *AAAI*. [S.l.: s.n.], 2010. Citado na página 36.
- MNIH, V. et al. Asynchronous methods for deep reinforcement learning. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2016. p. 1928–1937. Citado na página 44.
- MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Research, v. 518, n. 7540, p. 529–533, 2015. Citado na página 44.
- MYERS, M. M. *Outperforming Game Theoretic Play with Opponent Modeling in Two Player Dominoes*. Tese (Master's Thesis) — Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2014. Citado 2 vezes nas páginas 18 e 19.
- NASH, J. F. et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, USA, v. 36, n. 1, p. 48–49, 1950. Citado na página 29.
- PERVIN, Y. Algorithmization and programming of the game of dominoes. *Problems of Cybernetics*, Pergamon Press, v. 111, p. 957–972, 1962. Citado na página 18.
- SCHAEFFER, J.; HERIK, H. J. van den. Games, computers, and artificial intelligence. *Artificial Intelligence*, v. 134, n. 1, p. 1–7, jan. 2002. ISSN 0004-3702. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0004370201001655>>. Citado na página 13.
- SILVER, D.; HASSABIS, D. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 2016. Citado na página 33.
- SMITH, M. H. A learning program which plays partnership dominoes. *Communications of the ACM*, ACM, v. 16, n. 8, p. 462–467, 1973. Citado na página 18.
- STURTEVANT, N. R. An analysis of uct in multi-player games. In: SPRINGER. *International Conference on Computers and Games*. [S.l.], 2008. p. 37–49. Citado na página 33.
- SUTTON, R. S.; BARTO, A. G. *Introduction to Reinforcement Learning*. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981. Citado na página 22.
- SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press Cambridge, 1998. v. 1. Citado na página 24.
- SZEPESVÁRI, C. *Algorithms for Reinforcement Learning*. [S.l.]: Morgan & Claypool, 2010. v. 4. (Synthesis Lectures on Artificial Intelligence and Machine Learning, v. 4). Citado na página 22.
- THIE, P. R. *Markov decision processes*. [S.l.]: Comap, Incorporated, 1983. Citado na página 21.
- WALDOALVAREZ. *Markov Decision Process*. 2017. Disponível em: <https://commons.wikimedia.org/wiki/File:Markov_Decision_Process.svg>. Citado na página 21.
- WIERING, M.; OTTERLO, M. van (Ed.). *Reinforcement Learning: State-of-the-Art*. Berlin: Springer, 2012. v. 12. (Adaptation, Learning, and Optimization, v. 12). ISBN 978-3-642-27644-6 978-3-642-27645-3. Disponível em: <<http://link.springer.com/10.1007/978-3-642-27645-3>>. Citado na página 22.

- WINANDS, M. H. Monte-carlo tree search in board games. *Handbook of Digital Games and Entertainment Technologies*, Springer, p. 47–76, 2017. Citado na página 33.
- WINANDS, M. H.; BJORNSSON, Y.; SAITO, J.-T. Monte carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 2, n. 4, p. 239–250, 2010. Citado na página 33.
- YANNAKAKIS, G. N.; TOGELIUS, J. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, Institute of Electrical and Electronics Engineers (IEEE), v. 7, n. 4, p. 317–335, dec 2015. Disponível em: <<https://doi.org/10.1109%2Ftciaig.2014.2339221>>. Citado na página 13.
- YOON, S. W. et al. Probabilistic planning via determinization in hindsight. In: *AAAI*. [S.l.: s.n.], 2008. p. 1010–1016. Citado na página 35.