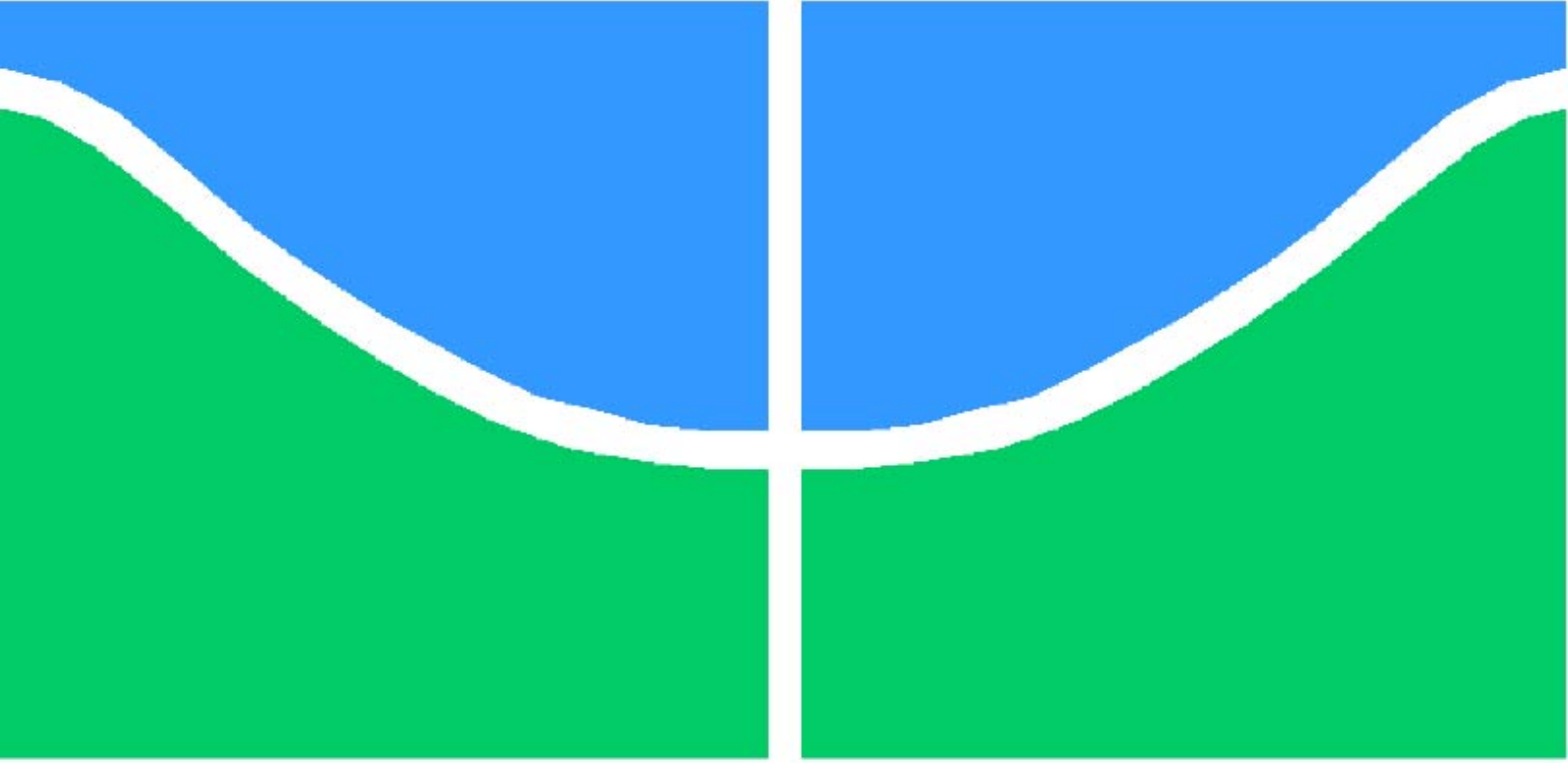


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de *Software*

Big Points: Uma Análise Baseada na Teoria dos Jogos

Autor: Mateus Medeiros Furquim Mendonça

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Coorientador:

Brasília, DF

2016

Mateus Medeiros Furquim Mendonça

Big Points: Uma Análise Baseada na Teoria dos Jogos

Monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de *Software*.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2016

Mateus Medeiros Furquim Mendonça
Big Points: Uma Análise Baseada na Teoria dos Jogos/ Mateus Medeiros Furquim Mendonça. – Brasília, DF, 2016-
51 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Teoria dos Jogos. 2. Análise Combinatória de Jogos. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. *Big Points*: Uma Análise Baseada na Teoria dos Jogos

CDU 02:141:005.6

Mateus Medeiros Furquim Mendonça

Big Points: Uma Análise Baseada na Teoria dos Jogos

Monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de *Software*.

Trabalho aprovado. Brasília, DF, 7 de julho de 2017:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Fábio Macedo Mendes
Convidado 1

Prof. Dra. Carla Silva Rocha Aguiar
Convidado 2

Brasília, DF
2016

Resumo

A Teoria dos Jogos estuda as melhores estratégias dos jogadores em um determinado jogo. Aplicando suas teorias em um jogo de tabuleiro eletrônico, este trabalho propõe analisar o jogo *Big Points* a partir de um determinado estado da partida e, como resultado, identificar as melhores heurísticas para os jogadores e uma possível inteligência artificial.

Palavras-chaves: Teoria dos Jogos, Análise Combinatória de Jogos.

Abstract

Key-words: Game Theory, Combinatorial Game Theory.

Lista de ilustrações

Lista de tabelas

Lista de códigos

*

Lista de abreviaturas e siglas

I.A.	Inteligência Artificial
T.C.J.	Teoria Combinatória dos Jogos
T.E.J.	Teoria Econômica dos Jogos

Lista de símbolos

Símbolos para conjuntos e operações matemáticas

$\varnothing$	Um conjunto sem elementos, conjunto vazio
$\{\ \}$	Delimita conjunto, de forma que $S = \{\}$ é um conjunto vazio
$\forall$	Para cada elemento
$x \in S$	Elemento x pertence ao conjunto S
$x \notin S$	Elemento x não pertence ao conjunto S
$S \subseteq T$	Conjunto S é um subconjunto de T, significa que se $x \in S$ então $x \in T$
$S \cup T$	União entre dois conjuntos $\{x\ ; x \in S \text{ or } x \in T\}$
$S \cap T$	Inteseção entre dois conjuntos $\{x\ ; x \in S \text{ and } x \in T\}$
$S_1 \times \ldots \times S_n$	Produto cartesiano $\{(x_1, \ldots, x_n)\ ; x_i \in S_i (1 \leq i \leq n)\}$
$\sum_{i=1}^{n} x_i$	Somatório de x_1 até x_n de maneira que $\sum_{i=1}^{n} x_i = x_1 + x_2 + \ldots + x_n$
$\prod_{i=1}^{n} x_i$	Produto de x_1 até x_n de maneira que $\prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdot \ldots \cdot x_n$
$A_{p,q}$	Arranjo de p elementos tomados de q a q calculado $A_{p,q} = \frac{p!}{(p-q)!}$
$\binom{p}{q}$	Combinação de p elementos tomados de q a q calculado $\binom{p}{q} = \frac{p!}{q! \cdot (p-1)!}$

Para jogos de soma zero com dois jogadores

σ, τ	Estratégias puras
x	Estratégia mista para o jogador 1
X	Conjunto de todas as estratégias mistas para o jogador 1
y	Estratégia mista para o jogador 2
Y	Conjunto de todas as estratégias mistas para o jogador 2
$P(x, y)$	Ganho do jogador 1

Para jogos não cooperativos com n jogadores

σ_i	uma estratégia pura para o jogador i
S_i	Conjunto de todas as estratégias puras para o jogador i
x_i	uma estratégia mista para o jogador i
X_i	Conjunto de todas as estratégias mistas para o jogador i
$P_i(x_1, \ldots, x_n)$	Ganho do jogador i
$x\|\|x_i'$	Considerando $x = (x_1, \ldots, x_n)$ o conjunto com todas as estratégias dos n jogadores, jogador i substitui a estratégia x_i pela estratégia x_i'

Sumário

1 Introdução

Imagine que um grupo de pessoas concordam em obedecer certas regras e agir de forma individual, ou em grupos menores, sem violar as regras especificadas. No final, suas ações como um todo levará a uma certa situação chamada **resultado**. Os membros deste grupo são chamados de **jogadores** e as regras que eles concordaram em obedecer constitui um **jogo**. Estes conceitos são pequenos exemplos utilizados em análises baseadas na **teoria dos jogos**.

A proposta deste trabalho foi realizar uma destas análise em um jogo de tabuleiro chamado *Big Points*. A motivação que levou à realização deste trabalho foi identificar uma heurística na qual tem-se uma maior chance de ganhar uma partida. Dessa forma, seria possível a implementação de uma inteligência artificial (I.A.) com diferentes dificuldades para jogar contra uma pessoa. Dito isso, o objetivo principal deste trabalho foi analisar várias partidas distintas de uma versão reduzida do jogo.

Uma análise possível para solucionar[1] o jogo é utilizar o teorema *minimax*, onde cada jogador tenta aumentar sua pontuação e diminuir a pontuação do oponente. Os resultados obtidos ao final dessa análise computacional baseadas no teorema *minimax* sugere a possibilidade do jogo completo ser desbalanceado[2], dando ao primeiro jogador uma maior chance de vencer o jogo.

A estrutura do trabalho foi dividida em cinco capítulos, sendo o primeiro esta introdução. O capítulo seguinte (2), Fundamentação Teórica, relata um pouco sobre a história da teoria dos jogos, esclarece alguns conceitos relevantes para o entendimento do trabalho, e explica as regras do próprio jogo. Em seguida, tem-se o capítulo 3, referente à análise e ao desenvolvimento do projeto até sua conclusão, e no capítulo 4 os resultados desta análise são discutidos. Por último, o capítulo 5 onde são feitas as considerações finais do trabalho e são citados alguns possíveis trabalhos futuros em cima do trabalho atual.

[1] Solucionar um jogo é percorrer todas as sua possibilidades de movimento e seus resultados.
[2] É dito um jogo balanceado aquele que a chance dos jogadores de ganhar é a mesma.

2 Fundamentação Teórica

Para um bom entendimento da análise realizada no jogo *Big Points* é preciso ter um conhecimento básico sobre teoria dos jogos e programação dinâmica. A primeira seção deste capítulo conta brevemente sobre a história da teoria dos jogos, com alguns nomes icônicos para esta área. A seção 2.2 explica um pouco sobre os conceitos da teoria dos jogos, mas apenas o necessário para este trabalho. Na seção 2.3, são explicados os conceitos sobre programação dinâmica e, na última seção (2.4), as regras do jogo *Big Points* são explicadas.

2.1 Histórico da Teoria dos Jogos

Pode-se dizer que a análise de jogos é praticada desde o séculco XVIII tendo como evidência uma carta escrita por James Waldegrave ao analisar uma versão curta de um jogo de baralho chamado *le Her* (PRAGUE, 2004). No século seguinte, o matemático e filósofo Augustin Cournot fez uso da teoria dos jogos para estudos relacionados à política[1] .

Mais recentemente, em 1913, Ernst Zermelo publicou o primeiro teorema matemático da teoria dos jogos (ZERMELO, 1913 apud SARTINI et al., 2004). Outros dois grandes matemáticos que se interessaram na teoria dos jogos foram Émile Borel e John von Neumann. Nas décadas de 1920 e 1930, Emile Borel publicou quatro artigos sobre jogos estratégicos[2] [3] [4] (PRAGUE, 2004), introduzindo uma noção abstrada sobre jogo estratégico e estratégia mista.

Em 1928, John von Neumann provou o teorema *minimax*, no qual há sempre uma solução racional para um conflito bem definido entre dois indivíduos cujos interesses são completamente opostos[5] (ALMEIDA,). Em 1944, Neumann publicou um trabalho junto a Oscar Morgenstern introduzindo a teoria dos jogos na área da economia e matemática aplicada[6] (SARTINI et al., 2004). Além destas contribuições, John von Neumann ainda escreveu trabalhos com grande impacto na área da computação, incluindo

[1] COURNOT, A.-A. Recherches sur les principes mathématiques de la théorie des richesses. L. Hachette (Paris), 1838. Disponível em: <http://catalogue.bnf.fr/ark:/12148/cb30280488q>.
[2] BOREL Émile. *The Theory of Play and Integral Equations with Skew Symmetric Kernels*. 1921.
[3] BOREL Émile. *On Games that Involve Chance and the Skill of Players*. 1924.
[4] BOREL Émile. *On Systems of Linear Forms of Skew Symmetric Determinant and the General Theory of Play*. 1927.
[5] NEUMANN, J. von. *Zur Theorie der Gesellschaftsspiele*. [S.l.]: Mathematische Annalen, 1928. 295–320 p.
[6] NEUMANN, J. von; MORGENSTERN, O. *Theory of Games and Economic Behavior*. [S.l.]: Princeton University Press, 1944.

a arquitetura de computadores, princípios de programação, e análise de algoritmos (MIYAZAWA, 2010).

Em 1982, Elwyn Berlekamp, John Conway e Richard Guy publicaram um livro em dois volumes[7] que se tornou uma referência na área da teoria dos jogos combinatorial (GARCIA; GINAT; HENDERSON, 2003) por explicar os conceitos fundamentais para a teoria dos jogos combinatorial.

2.2 Teoria dos Jogos

O campo da teoria dos jogos divide-se em três áreas: 1) Teoria Econômica dos Jogos (T.E.J.), que normalmente analisa movimentos simultâneos de dois ou mais jogadores; 2) Teoria Combinatória dos Jogos (T.C.J.), no qual os jogadores fazem movimentos alternadamente, e não faz uso de elementos de sorte, diferente da T.E.J. que também trata desse fenômeno; e 3) Teoria da Computação dos Jogos

É possível abstrair um jogo de várias maneiras para facilitar sua análise. A **forma extensiva** de um jogo elimina as informações de como jogá-lo e passa a ser representado pelos movimentos possíveis e como o estado do jogo é alterado (JONES, 1980). Um dos jogos mais simples Considere o jogo *Nim* (GARCIA; GINAT; HENDERSON, 2003)

Definição 1. Em jogos com **movimentos simultâneos**, os jogadores devem escolher o que fazer ao mesmo tempo ou, o que leva à mesma situação, as escolhas de cada jogador é escondida de seu oponente. Em qualquer um dos dois casos, o jogador deve escolher sua jogada levando em consideração a possível jogada do outro (CARMICHAEL, 2005).

Definição 2. Jogos de **soma zero** são jogos de puro conflito, no qual apenas um jogador pode vencer o jogo. Em outras palavras, a vitória de um jogador implica na derrota do outro.

2.2.1 Minimax

O teorema minimax provado por John von Neumann é a peça principal da maior parte do trabalho matemático em economia e em atividades onde os atos das decisões são racionais.

Segundo o teorema minimax, há sempre uma solução racional para um conflito entre dois indivíduos cujos interesses são completamente opostos, ou seja, o que é ganho por um lado é perdido pelo outro. Esse é um exemplo da chamada situação soma zero, uma vez que os ganhos dos dois jogadores somam zero.

[7] BERLEKAMP, E. R.; CONWAY, J. H.; GUY, R. K. *Winning Ways for Your Mathematical Plays, Vol. 1*. 1. ed. London, UK: Academic Press, 1982. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1568811306>.

A combinação de estratégias, na qual o máximo dos mínimos é igual ao mínimo dos máximos, chama-se de ponto de equilíbrio do jogo, pois ao escolherem essas estratégias, os jogadores garantem para si um ganho mínimo independente do que o adversário venha a escolher.

Trabalhando com estratégias puras, utilizamos o critério maximin para definir os valores máximo e mínimo do jogo.

Em um jogo de dois jogadores com soma zero é racional para cada jogador escolher a estratégia que maximiza seu ganho mínimo ou que minimize o ganho máximo do adversário, conforme figura 1.

Agora, considerando o uso de estratégias mistas, ou seja, aumentando as possibilidades de escolha, podemos usar o mesmo critério para definir os novos valores máximo e mínimo.

Por exemplo, dois jogadores, na disputa por par ou ímpar, cada um com duas alternativas de escolha. O ganho será representado por 1 e perda por -1. O jogador par obterá ganho se ambos fizerem a mesma escolha, e neste cas ípar

A área de teoria combinatória dos jogos analisa os jogos de uma perspectiva um pouco diferente. É considerado que dois jogadores alternam os movimentos em um jogo que não possui elementos de chance (rolagem de dados, saque de cartas, etc.) e que ambos jogadores possuem informação completa. É dito que os jogadores possuem informação completa se eles tiverem conhecimento de tudo o que está acontecendo no jogo a todo momento (**??**). Ao chegar na vez de algum jogador e ele não tiver nenhum movimento válido para realizar, então aquele jogador é considerado perdedor. Considerando os jogadores *esquerda* e *direita*, podemos representar suas jogadas da maneira descrita em 2.1, onde o jogador *esquerda* possui as jogadas $\{a, b, c, \ldots\}$ e o jogador *direita*, as jogadas $\{f, g, h, \ldots\}$.

$$\{a, b, c, \ldots | f, g, h, \ldots\} \tag{2.1}$$

A Teoria dos Jogos pode ser definida como a teoria dos modelos matemáticos que estuda a escolha de decisões ótimas[8] sob condições de conflito[9]. Os elementos básicos de um jogo são o conjunto de **jogadores**, onde cada jogador possui um conjunto de **estratégias** e, a partir das escolhas de estratégias de cada jogador, temos uma **situação** ou **perfil**. Para cada perfil do jogo, tem-se um resultado no final do jogo. Em termos

[8] É considerado que os jogadores são seres racionais e que possuem conhecimento completo das regras do jogo. Às vezes o jogador também possui informação completa sobre o estado atual e do histórico de jogadas do jogo.

[9] Condições de conflito são aquelas no qual dois ou mais jogadores possuem o mesmo objetivo.

matemáticos é dito que um jogador tem uma **função utilidade**, que atribui um ***payoff***, ou **ganho**, para cada situação do jogo.

Quando essa informação é inserida em uma matriz, tem-se uma **matriz de *payoff***. Em outras palavras, matriz de ganho é a representação matricial dos *payoffs* dos jogadores, onde as estratégia de um jogador estão representadas por cada linha e as de seu oponente estão representadas pelas colunas como mostra a tabela table 1. Além disso o ganho dos jogadores é representado como uma tupla (ou par) de valores, sendo que o primeiro é o ganho do primeiro jogador e o segundo valor, o do segundo jogador.

Tabela 1 – Matriz de ganho.

$P_2 \setminus P_1$	E_{11}	E_{12}
E_{21}	(1,0)	(2,3)
E_{22}	(3,4)	(0,2)

Dessa forma, o primeiro jogador, que é representado por P_1, possui as estratégias E_{11} e E_{12}. Semelhante ao primeiro jogador, tem-se o segundo jogador sendo representado por P_2 e com as estratégias E_{21} e E_{22}. Os valores que se encontram na interseção da estratégia de P_1 e P_2 são os ganhos dos dois jogadores, dessa forma se as estratégias escolhidas forem E_{12} e E_{21}, o primeiro jogador teria perdido com 3 pontos e o segundo jogador venceria com 4 pontos.

De uma forma matemática mais genérica, tem-se o jogador $i \in \{1,2\}$ onde sua estratégia é representada por $\sigma_i \forall \sigma \in S_i$.

2.2.2 Minimax

Como o jogo não possui nenhum elemento dependente da sorte, não serão usados estratégias mistas. O *winning move* não foi analizado devido à complexidade da implementação da análise atual.

e as estratégias σ e τ para o primeiro e segundo jogador, respectivamente. para cada jogador, as estratégias $e \in \{1,2,\cdots,n\}$. Com isso, cada estratégia pode ser representada por E_{je}. Para determinar a pontuação dos jogadores, temos uma função ultilidade $u(E_{1e}, E_{2e})$ tal que retorne uma tupla (a,b) onde a representa o ganho do jogador 1 e b representa o ganho do jogador 2.

2.2.3 Soluções de um jogo

Uma solução de um jogo é uma prescrição ou previsão sobre o resultado do jogo. Dois métodos importantes para encontrar a solução de um estado do jogo são

dominância e **equilíbrio de Nash**.

É dito que uma determinada estratégia é uma **estratégia dominante** quando esta é a única estratégia restante após aplicar a técnica de **dominância estrita iterada**. O encontro das estratégias dos jogadores é chamado de **equilíbrio de estratégia dominante**.

Dominância estrita iterada nada mais é do que um processo onde se eliminam as estratégias que são estritamente dominadas. Obs.: faltou explicar o que é uma estratégia dominada.

Solução estratégica ou **Equilíbrio de Nash** é um conjunto de estratégias para cada jogador onde cada um deles não tem incentivo de mudar sua estratégia se os demais jogadores não o fizerem.

Zero-sum game: a vitória de um jogador implica na derrota do outro. No Big Points, o jogador com maior pontuação vence. Pode-se dar pontuação 1 caso o jogador em questão é o vencedor, e -1 para o jogador que perdeu. Caso haja mais de um jogador com a maior pontuação do jogo, é dado 0 para o payoff dos dois jogadores.

Outra maneira, mais refinada, de demonstrar a vitória e derrota entre os jogadores é calcular a difereça da pontuação entre eles. O jogador com a maior pontuação mantém sua pontuação, e o restante tem sua pontuação subtraída daquela maior pontuação do jogo (dando um resultado negativo).

Backward Induction - As long as every player take turns you can start at the end of the game and make your way to the begin. - One strategy for every decision node

Game Theory the study of strategic interaction among rational decision makers players: people playing the game; each player has a set of strategies strategies: what they will do, how they'll respond payoffs: result of the interaction of strategies

strategy is a set with what decision you will make for every decision making situation in the game

each players is chosen an strategy, these strategies interact, and the game plays out to its conclusion.

rationality and common knowledge

Teoria dos jogos é o estudo do comportamento estratégico interdependente[10], não apenas o estudo de como vencer ou perder em um jogo, apesar de às vezes esses dois fatos coincidirem. Isso faz com que o escopo seja mais abranjente, desde comportamentos no qual as duas pessoas devem cooperar para ganhar, ou as duas tentam se ajudar para ganharem independente ou, por fim, comportamento de duas pessoas que

[10] Estratégia interdependente significa que as ações de uma pessoa interfere no resultado da outra, e vice-versa.

tentam vencer individualmente (SPANIEL, 2011).

2.3 Programação dinâmica

MIMIMIPROGRAM A SODIAMSDOUAHSDOIUG ADIFHGSDKJFHGSD FJGSkfdjgHSD-fas asdasd

2.4 Regras do Big Points

Big Points é um jogo abstrato e estratégico com uma mecânica de colecionar peças que pode ser jogado de dois a cinco jogadores. São cinco peões de cores distintas, que podem ser usadas por qualquer jogador, para percorrer um caminho de discos coloridos até chegar à escada. Durante o percurso, os jogadores coletam alguns destes discos e sua pontuação final é determinada a partir da ordem de chegada dos peões ao pódio e a quantidade de discos adquiridos daquela cor. Ganha o jogador com a maior pontuação.

Figura 1 – Caixa do jogo **Big Points**

(a) Conteúdo do jogo **Big Points** (b) Preparação do jogo **Big Points**

Figura 2 – Organização do jogo **Big Points**

O jogo é composto por cinco peões, como demonstrado na figura 1, um de cada uma das seguintes cores, denominadas **cores comuns**: vermelha, verde, azul, amarela e violeta. Para cada cor de peão, tem-se dez discos, como mostrado na figura 2a, (totalizando cinquenta discos) denominados **discos comuns**, e cinco discos das cores branca e preta (totalizando dez discos) denominados **discos especiais**. Por fim, há um pódio (ou escada) com um lugar para cada peão. A escada determinará a pontuação equivalente a cada disco da cor do peão, de maneira que o peão que ocupar o espaço mais alto no pódio (o primeiro a subir) fará sua cor valer quatro[11], o segundo peão, três pontos e assim por diante, até o último valer zero pontos.

No final da preparação, o jogo ficará parecido com as peças na figura 2b. A preparação do jogo ocorre em algumas etapas envolvendo a posição dos peões, a aleatoriedade do tabuleiro e alguns discos ao lado da escada. A primeira coisa é retirar um disco de cada cor comum e posicioná-los ao lado da escada, estes serão os discos coletados pelo jogador que subir o peão da sua cor para a escada. Em seguida, deve-se embaralhar todos os 55 discos restantes[12] e formar uma fila até a escada, estes são os discos possíveis de serem coletados e onde os peões andam até chegar na escada. Por último, é preciso posicionar os peões no começo da fila de discos, de forma que fique oposto à escada.

[11] No caso de um jogo com menos de cinco peões, a seguinte fórmula se aplica: $Score = N_c - P_{pos}$, onde $Score$ é a pontuação daquela determinada cor, N_c é o número de discos comuns e P_{pos} é a posição do peão no pódio.

[12] 9 discos de cada uma das 5 cores comuns mais 5 discos de cada uma das 2 cores especiais resultando em $(n_{dc} - 1) \cdot n_{cc} + n_{de} \cdot n_{ce} = (10 - 1) \cdot 5 + 5 \cdot 2 = 55$ *discos*, onde n_{dc} é o número de discos comuns, n_{cc} é o número de cores comuns, n_{de} é o número de discos especiais, e n_{ce} é o número de cores especiais.

Após preparar o jogo, deve-se escolher o primeiro jogador de forma aleatória. Na sua vez, cada jogador deve escolher um peão, que não esteja na escada, para movê-lo até o disco à frente mais próximo de sua cor. Caso não haja um disco de sua cor para movê-lo, o peão sobe na escada para a posição mais alta que não esteja ocupada e coleta o disco daquela cor que está ao lado da escada. Em seguida, o jogador escolhe para pegar o primeiro disco disponível[13] à frente ou atrás da nova posição do peão. Caso o disco não esteja disponível, verifique o próximo disco até encontrar um que esteja disponível. Ao encontrar um disco que o jogador possa pegar, retire-o do tabuleiro e coloque-o na mão do jogador atual. A sua vez termina e passa para o próximo escolher um peão e pegar um disco. O jogo segue desta maneira até que todos os peões se encontrem na escada. No final do jogo, conta-se os pontos e ganha o jogador que tiver a maior pontuação.

A pontuação do jogo é dependente da ordem de chegada dos peões na escada e da quantidade de discos de cada cor que o jogador tiver. O primeiro peão que chegou na escada faz com que cada disco de sua cor valha quatro pontos. Os jogadores devem então multiplicar a quantidade de discos daquela cor pelo valor da ordem de chegada do peão da sua cor na escada. Exemplo: se o primeiro jogador tiver dois discos vermelhos, um disco verde e três azuis e a ordem de chegada deles for azul em primeiro lugar, verde logo em seguida e depois o vermelho, sua pontuação será descrita de acordo com a equação , onde n_c é o número de cores do jogo, n_r, n_g e n_b são as quantidades de discos vermelhos, verdes e azuis, respectivamente, que o jogador possui e p_r, p_g e p_b são as posições dos peões vermelho, verde e azul, respectivamente, na escada.

$$\begin{aligned} P &= n_r \cdot (n_c - p_r) + n_g \cdot (n_c - p_g) + n_b \cdot (n_c - p_b) \\ P &= 2 \cdot (3 - 3) + 1 \cdot (3 - 2) + 3 \cdot (3 - 1) \\ P &= 7 \end{aligned} \tag{2.2}$$

[13] É dito disponível aquele disco presente no tabuleiro que não possui um peão em cima.

3 Metodologia

3.1 *Fluxo de Trabalho*

O *framework scrum* é ideal para o desenvolvimento de projetos complexos no qual a produtividade e a criatividade são essenciais para a entrega de um produto de alto valor (SCHWABER; SUTHERLAND, 2016). Inicialmente, tal método de organização e gerenciamento do projeto foi aplicado para o desenvolvimento do sistema em questão. O *kanban* do waffle.io[1] foi utilizado para registrar tarefas devido à sua integração com as *issues* do github. Reuniões com o orientador foram realizadas para discutir aspectos técnicos do jogo, como as estruturas de dados a serem utilizadas para reduzir os dados armazenados, e alguns métodos importantes para agilizar o processamento.

Porém, ao longo do tempo, o esforço para manter a rastreabilidade das tarefas tornou-se muito alto em relação à complexidade do projeto, e ao tamanho da equipe. As tarefas passaram a ser *branchs* locais com nomes significativos, representando a funcionalidade a ser desenvolvida. Após a conclusão da tarefa, testes simples e manuais foram aplicados para então unir à *branch* mestre[2]. Por fim, para trabalhar em outra *branch*, foi sempre necessário atualizá-la em relação à mestre[3].

3.2 Análise do jogo *Big Points*

Para analizar o jogo *Big Points*, é preciso realizar todas as jogadas de todos os jogos possíveis. Cada jogador, na sua vez, deve escolher uma jogada na qual lhe garanta a vitória, se houver mais de uma, escolha a que tiver a maior pontuação. Caso não tenha uma jogada para vencer, o jogador deve minimizar a pontuação do adversário. Após fazer isso para um jogo inicial, os resultados são escritos em um arquivo *csv* para análise. Esse procedimento é repetido para *cada* organização possível do tabuleiro inicial.

Exaurir todas as possibilidades de jogadas é um trabalho computacional imenso e cresce exponencialmente de acordo com o tamanho do jogo. Para um jogo pequeno com apenas dois discos e duas cores comuns (sem especiais) as jogadas possíveis são: mover o peão vermelho e pegar o disco da direita, ou da esquerda; e mover o peão verde e pegar o disco da direita ou da esquerda. Isso gera uma árvore onde cada nó possui quatro filhos e a altura média dessa árvore é quatro, totalizando uma quantidade de estados de aproximadamente $\sum_{h=0}^{4} 4^h \approx 341$. Ao final do cálculo deste jogo

[1] https://waffle.io/mfurquim/tcc
[2] `$ git checkout <to-branch>; git merge <from-branch>`
[3] `$ git rebase <from-branch> <to-branch>`

reduzido, temos que o número de estados distintos varia entre 17 e 25, dependendo do estado inicial do tabuleiro. Devido a este grande número de estados repetidos, escrever o algoritmo fazendo uso de programação dinâmica economizou bastante tempo e processamento.

O jogo seria um jogo balanceado se ambos os jogadores ganharem aproximadamente metade das vezes. Se existem seis jogos diferentes (combinação de duas cores com dois discos cada), o jogo é considerado balanceado se cada jogador ganhar três jogos. Neste caso, temos os jogos $j_i \in \{1122, 1212, 1221, 2112, 2121, 2211\}$, e para cada j_i temos a pontuação máxima e a quantidade de estados distintos, como demonstrado na tabela table 2.

Tabela 2 – Pontuação utilizando Minimax.

Jogo	**Pontuação**	**#Estados**
1122	(2,1)	17
1212	(2,0)	25
1221	(2,1)	25
2112	(2,1)	25
2121	(2,1)	25
2211	(2,0)	17

Em todos as possíveis combinações de tabuleiros iniciais, o primeiro jogador sempre ganha com dois pontos enquanto o segundo jogador consegue fazer no máximo um ponto, na maioria das vezes. Isso torna o jogo desequilibrado.

3.2.1 Quantidade de partidas

$$
\begin{aligned}
Partidas &= (\#J - 1) \cdot \binom{\#D_T}{\#D_W} \cdot \binom{\#D_{L1}}{\#D_K} \cdot \binom{\#D_{L2}}{\#D_R} \cdot \binom{\#D_{L3}}{\#D_G} \cdot \binom{\#D_{L4}}{\#D_B} \cdot \binom{\#D_{L5}}{\#D_Y} \cdot \binom{\#D_{L6}}{\#D_V} \\
Partidas &= 4 \cdot \binom{55}{5} \cdot \binom{50}{5} \cdot \binom{45}{9} \cdot \binom{36}{9} \cdot \binom{27}{9} \cdot \binom{18}{9} \cdot \binom{9}{9} \\
Partidas &= 560'483'776'167'774'018'942'304'261'616'685'408'000'000 \\
Partidas &\approx 5 \times 10^{41}
\end{aligned}
\tag{3.1}
$$

3.3 Estrutura de dados

Devido à enorme quantidade de estados de um jogo reduzido de *Big Points*, foi implementado duas funções para codificar e decodificar a *struct State* para um *long long int*, de forme que ocupe apenas 64 *bits* na memória. Após testar nos limites da

capacidade da variável, percebeu-se um erro quando executado com quatro cores e cinco discos, o que levou à implementação por *bit fields*.

3.3.1 Estado do jogo

Para escrever a programação dinâmica capaz de

3.3.2 Bit fields

Dentro da estrutura `State` foi declarado duas estruturas anônimas[4] utilizando *bit fields*. As duas estruturas servem para garantir a utilização correta dos *bits* quando as variáveis chegarem próximo ao limite da sua capacidade. Essas estruturas possuem variáveis do tipo `unsigned long long int`, que ocupa 64 *bits*. Após a declaração da variável, é declarado a quantidade de *bits* que será utilizado para ela, de modo que `ll _tabuleiro :20` ocupe apenas 20 *bits* da variável `unsigned long long int`, `ll _peao :15` ocupe 15 *bits*, e assim por diante de forma que não ultrapsse os 64 *bits* da variável. Como o comportamento do armazenamento é desconhecido quando a variável é ultrapassada, e para garantir consistência no armazenamento, foi utilizado duas *structs* com, no máximo, uma variável `unsigned long long int` (64 *bits*).

A estrutura `State` possui cinco variáveis: `_tabuleiro`, no qual pode armazenar informações sobre um tabuleiro até 20 discos[5]; `_peao`, que representa a posição $p_i \in \{0, 1, ..., n_d, n_d + 1\}$, onde n_d é o número de discos de cores comuns no jogo e p_i é o peão da cor i[6]; `_escada`, que indica as posições dos peões na escada, sendo a p_i*-ésima* posição de `_escada` é a posição do peao p_i; `_jogadores`, possui informações sobre os discos coletados dos dois jogadores; e por fim, a variável `_atual` que representa o jogador que fará a jogada.

Código 3.1 – Definição da estrutura `State`

```
10 struct State
11 {
12     // Cinco cores, quatro discos
13     struct {
14         // 5 cores * 4 discos (1bit pra cada)
15         ll _tabuleiro :20;
16
17         // 0..5 posições possíveis (3bits) * 5 peões
```

[4] Estruturas anônimas permitem acesso às suas variáveis de forma direta, como por exemplo: `state._tabuleiro` acessa a variável `_tabuleiro` dentro da estrutura anônima, que por sua vez se encontra dentro da estrutura `State`.

[5] Cinco cores e quatro discos.

[6] As cores de peão seguem a ordem RGBYP começando do 0, onde **R***ed* = 0, **G***reen* = 1, **B***lue* = 2, **Y***ellow* = 3, e **P***urple* = 4.

```
18          ll _peao :15;
19
20          // 0..5 posições (3bits) * 5 peões
21          ll _escada :15;
22      };
23
24      struct {
25          // 0..5 discos (3bits) * 5 cores * 2 jogadores
26          ll _jogadores :30;
27
28          // Jogador 1 ou Jogador 2
29          ll _atual :1;
30      };
```

O cálculo para determinar os *bits* necessários para armazenar as informações de cada variável foi realizado será explicado nas subseções seguintes.

3.3.2.1 Cálculo de bits do atributo `tabuleiro`

$$\begin{aligned} \texttt{\_tabuleiro} &= n_c \cdot n_d \\ \texttt{\_tabuleiro} &= 5 \cdot 4 \\ \texttt{\_tabuleiro} &= 20\ bits \end{aligned} \tag{3.2}$$

Na equação 3.2, n_c e n_d são o número de cores e o número de discos do jogo, respectivamente. Seus valores são, no máximo $n_c = 5$ e $n_d = 4$.

3.3.2.2 Cálculo de bits do atributo `peao`

$$\begin{aligned} \texttt{\_peao} &= \lceil \log_2(n_d + 1) \rceil \cdot n_p \\ \texttt{\_peao} &= \lceil \log_2(5 + 1) \rceil \cdot 4 \\ \texttt{\_peao} &= 3 \cdot 4 \\ \texttt{\_peao} &= 15\ bits \end{aligned} \tag{3.3}$$

Na segunda equação, 3.3, o valor de n_d é o número de discos e n_p é o número de peões do jogo, que por sua vez é igual a n_c (número de cores comuns). Cada peão pode estar: fora do tabuleiro, com $peao(p_i) = 0$; em cima de um disco da sua cor, com $peao(p_i) \in \{1, 2, ..., n_d\}$; e na escada, com $peao(p_i) = n_d + 1$.

3.3.2.3 Cálculo de bits do atributo `escada`

$$\begin{aligned} \texttt{\_escada} &= \lceil \log_2(n_p + 1) \rceil \cdot n_p \\ \texttt{\_escada} &= \lceil \log_2(6) \rceil \cdot 5 \\ \texttt{\_escada} &= 15\ bits \end{aligned} \tag{3.4}$$

A equação 3.4 possui as variáveis n_p e n_c com $n_p, n_c \in \{2,3,4,5\}$ e $n_p = n_c$. Cada peão tem um local na escada, que armazena a posição dele de forma que $0 \leqslant escada(p_i) \leqslant n_c$. As situações possíveis são: $escada(p_i) = 0$ quando o peão não estiver na escada; e $escada(p_i) \in \{1,2,3,4,5\}$ sendo a ordem de chegada do peão na escada[7].

3.3.2.4 Cálculo de bits do atributo `jogadores`

$$\begin{aligned} \texttt{\_jogadores} &= \lceil \log_2(n_d + 1) \rceil \cdot n_c \cdot n_j \\ \texttt{\_jogadores} &= \lceil \log_2(4 + 1) \rceil \cdot 5 \cdot 2 \\ \texttt{\_jogadores} &= 3 \cdot 5 \cdot 2 \\ \texttt{\_jogadores} &= 30\ bits \end{aligned} \tag{3.5}$$

A capacidade da variável `_jogadores` é de 30 *bits*, como demonstrado na equação . As variáveis utilizadas nessa equação são: n_d, o número de discos $n_d \in \{1,2,3,4,5\}$; n_c, o número de cores $n_c \in \{1,2,3,4,5\}$; e n_j, o número de jogadores $n_j = 2$. A informação armazenada na mão dos jogadores, para cada disco, vai até o número máximo de discos mais um, pois o jogador pode pegar todos os discos no tabuleiro e o disco adquirido ao mover o peão para a escada. Para armazenar o número seis, são necessários $\lceil \log_2(6) \rceil = 3 bits$

3.3.2.5 Cálculo de bits do atributo `atual`

$$\begin{aligned} \texttt{\_atual} &= \lceil \log_2(2) \rceil \\ \texttt{\_atual} &= 1\ bit \end{aligned} \tag{3.6}$$

3.3.3 Funções de acesso da estrutura `State`

A estrutura possui um construtor que atribui valores às variáveis através de RAII[8], dessa forma não se faz necessário nenhuma extra implementação. Todas as

[7] O primeiro peão p_i a chegar na escada é indicado com $escada(p_i) = 1$.

[8] *Resource Aquisition Is Initialization* é uma técnica de programação que vincula o ciclo de vida do recurso ao da estrutura (CUBBI; MAGGYERO; FRUDERICA,).

variáveis possuem um valor padrão, verdadeiro para qualquer tamanho de tabuleiro t_i, onde $4 \leqslant t_i \leqslant 20$.

Código 3.2 – Construtor da estrutura `State`

```
33      State(int mtabuleiro = (1<<20)-1, int mpeao = 0, int mescada = 0,
34          int mjogadores = 0, int matual = 0) : _tabuleiro(mtabuleiro),
35          _peao(mpeao), _escada(mescada), _jogadores(mjogadores),
36          _atual(matual)
37      {
38      }
```

3.3.3.1 Atributo `tabuleiro`

Código 3.3 – Funções de acesso ao atributo `tabuleiro`

```
41      int tabuleiro (int pos) const {
42          return (_tabuleiro & (1<<pos))>>pos;
43      }
44
45      void settabuleiro (int pos, int available) {
46          _tabuleiro = (_tabuleiro & ~(1<<pos)) | ((available&1)<<pos);
47      }
```

3.3.3.2 Atributo `peao`

Código 3.4 – Funções de acesso ao atributo `peão`

```
50      int peao (int cor) const {
51          return (_peao & (7<<(3*cor)))>>(3*cor);
52      }
53
54      void setpeao (int cor, int pos) {
55          _peao = (_peao&~(7<<(3*cor)))|((pos&7)<<(3*cor));
56      }
57
58      void movepeao (int cor) {
59          setpeao(cor,peao(cor)+1);
60      }
```

3.3.3.3 Atributo `escada`

Código 3.5 – Funções de acesso ao atributo `escada`

```
63      int escada (int cor) const {
64          return (_escada & (7<<(3*cor)))>>(3*cor);
65      }
66
```

```
67      void setescada (int cor, int pos) {
68          _escada = (_escada&~(7<<(3*cor)))|((pos&7)<<(3*cor));
69      }
```

3.3.3.4 Atributo jogador

Código 3.6 – Funções de acesso ao atributo jogador

```
72      int jogador (int jogador, int cor) const {
73          return ((_jogadores>>(15*jogador)) & (7<<(3*cor)))>>(3*cor);
74      }
75
76      void setjogador (int jogador, int cor, int qtd) {
77          _jogadores = (_jogadores & ~(7<<(3*cor + 15*jogador) ))
78              | ((qtd & 7) << (3*cor + 15*jogador));
79      }
80
81      void updatejogador (int player, int cor) {
82          setjogador(player, cor, jogador(player, cor)+1);
83      }
```

3.3.3.5 Atributo atual

Código 3.7 – Funções de acesso ao atributo atual

```
86      int atual () const {
87          return _atual;
88      }
89
90      void updateatual () {
91          _atual ^= 1;
92      }
```

3.3.4 Comparador da estrutura State

Código 3.8 – Comparado da estrutura State

```
95      // Operator to use it in map
96      bool operator<(const struct State& s) const {
97          if (_tabuleiro != s._tabuleiro) return _tabuleiro < s._tabuleiro;
98          if (_peao != s._peao) return _peao < s._peao;
99          if (_escada != s._escada) return _escada < s._escada;
100         if (_jogadores != s._jogadores) return _jogadores < s._jogadores;
101         return _atual < s._atual;
102     }
```

3.4 Programação dinâmica

Programação dinâmica é um método para a construção de algoritmos no qual há uma memorização de cada estado distinto para evitar recálculo, caso este estado apareça novamente. A memorização dos estados do jogo *Big Points* foi feita em uma *hash*, com a chave sendo o estado do jogo e o valor armazenado, a pontuação máxima dos dois jogadores a partir daquele nó.

a melhor jogada para ganhar maximizar seus pontos. Caso não Na vez de cadaCaso a quantidade de jogos vencidos pelo primeiro jogador seja aproximadamente 50%

Para analizar o jogo, é preciso exaurir todas as jogadas possíveis a partir de um jogo inicial. Como

utilizando programação dinâmica[ˆdynamic_programing] onde os estados são armazenados em uma *hash*, temos que o número de estados distintos varia entre 17 e 25.

Devido ao imenso número de jogadas possíveis ao longo do do jogo, decidiu-se utilizar a programação dinâmica para - Duas funções para melhor entendimento da DP e regras do jogo

3.4.1 Função `dp`

A função `dp` possui os casos base para retornar a função,

Código 3.9 – Construtor da estrutura `State`

```
1 #include <bitset>
2 #include <iostream>
3 #include <vector>
4
5 #include "dp.h"
```

3.4.2 Função `play`

Código 3.10 – Construtor da estrutura `State`

```
1 #include <bitset>
2 #include <iostream>
3 #include <vector>
4
5 #include "dp.h"
```

- Explicação da DP e da função Play (função para realizar as jogadas)

3.5 Verificação dos estados

Foi escrito os estados e suas transições em *post-it*s para garantir que a *DP* foi feita corretamente. Os estados

4 Resultados

4.1 Análise Estatística

Estimar quantidade de Jogos que o jogador 1 consegue ganhar, empatar e perder.

o jogo é desbalanceado.

5 Considerações Finais

5.1 Trabalhos futuros

Desenvolvimento de uma I.A. para competir contra um jogador humano.

Referências

ALMEIDA, A. N. de. Teoria dos jogos: As origens e os fundamentos da teoria dos jogos. UNIMESP - Centro Universitário Metropolitano de São Paulo, São Paulo, SP, Brasil. Citado na página 23.

BERLEKAMP, E. R.; CONWAY, J. H.; GUY, R. K. *Winning Ways for Your Mathematical Plays, Vol. 1*. 1. ed. London, UK: Academic Press, 1982. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1568811306>. Citado na página 23.

BOREL Émile. *The Theory of Play and Integral Equations with Skew Symmetric Kernels*. 1921. Citado na página 23.

BOREL Émile. *On Games that Involve Chance and the Skill of Players*. 1924. Citado na página 23.

BOREL Émile. *On Systems of Linear Forms of Skew Symmetric Determinant and the General Theory of Play*. 1927. Citado na página 23.

CARMICHAEL, F. *A Guide to Game Theory*. [S.l.: s.n.], 2005. Citado na página 24.

COURNOT, A.-A. Recherches sur les principes mathématiques de la théorie des richesses. L. Hachette (Paris), 1838. Disponível em: <http://catalogue.bnf.fr/ark:/12148/cb30280488q>. Citado na página 23.

CUBBI; MAGGYERO; FRUDERICA. *RAII*. <http://en.cppreference.com/w/cpp/language/raii>. Accessed May 31, 2016. Citado na página 35.

GARCIA, D. D.; GINAT, D.; HENDERSON, P. Everything you always wanted to know about game theory: But were afraid to ask. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 35, n. 1, p. 96–97, jan. 2003. ISSN 0097-8418. Disponível em: <http://doi.acm.org/10.1145/792548.611900>. Citado 2 vezes nas páginas 23 e 24.

JONES, A. J. *Game Theory*: Mathematical models of conflict. [S.l.: s.n.], 1980. Citado na página 24.

MIYAZAWA, F. K. Introdução à teoria dos jogos algorítmica. UNICAMP, São Paulo, SP, Brasil, 2010. Disponível em: <http://www.ic.unicamp.br/~fkm/lectures/algorithmicgametheory.pdf>. Citado na página 23.

NEUMANN, J. von. *Zur Theorie der Gesellschaftsspiele*. [S.l.]: Mathematische Annalen, 1928. 295–320 p. Citado na página 23.

NEUMANN, J. von; MORGENSTERN, O. *Theory of Games and Economic Behavior*. [S.l.]: Princeton University Press, 1944. Citado na página 23.

PRAGUE, M. H. *Several Milestones in the History of Game Theory*. VII. Österreichisches Symposion zur Geschichte der Mathematik, Wien, 2004. 49–56 p. Disponível em: <http://euler.fd.cvut.cz/predmety/game_theory/games_materials.html>. Citado na página 23.

SARTINI, B. A. et al. *Uma Introdução a Teoria dos Jogos*. 2004. Citado na página 23.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. [S.l.]: Scrum.Org, 2016. Citado na página 31.

SPANIEL, W. *Game Theory 101*: The complete textbook. [S.l.: s.n.], 2011. Citado na página 27.

ZERMELO, E. F. F. *Über eine Anwendung der Mengdenlehre auf die theories des Schachspiels*. 1913. 501–504 p. Citado na página 23.

Anexos

ANEXO A – Regras Originais do Jogo *Big Points*

BigPoints

De Brigitte e Wolfgang Ditt
para 2 a 5 jogadores a partir dos 8 anos

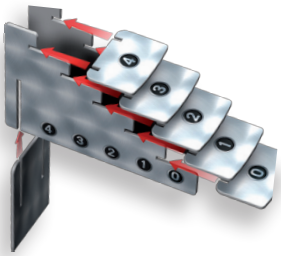
PORTUGUES

O material

- 60 discos em madeira (10 discos de cada umas das seguintes cores : azul, vermelho, amarelo, verde e violeta e ainda 5 brancos e 5 pretos)
- 5 peões : azul, vermelho, amarelo, verde e violeta
- 1 escada de chegada

Conceito do jogo

Os jogadores movem um peão qualquer para o próximo disco da mesma cor do peão. Depois, recolhem o disco situado à frente ou atrás desse peão. O valor dos discos recolhidos depende da ordem dos peões na escada de chegada no fim do jogo.



Antes do primeiro jogo, destacar cuidadosamente as peças do cartão e montar a escada de chegada como mostra a ilustração.

Os preparativos

Formar uma pilha com um disco de cada uma das cores seguintes : azul, vermelho, amarelo, verde e violeta e colocar essa pilha ao lado da escada. (Esses discos destinam-se aos jogadores que coloquem o seus peão na escada de chegada.) Misturar os discos restantes (e claro, os blancos e os pretos) e colocá-los como desejar de maneira a formar um percurso desde a base da escada. A ordem das cores não importa. Posicionar os peões no início do percurso (ver a ilustração à direita).

O desenvolvimento do jogo

Escolher um jogador inicial. Depois, joga-se à vez seguindo o sentido dos ponteiros do relógio. Na sua vez, o jogador escolhe um peão **qualquer**. Coloca-o sobre o disco seguinte cuja **cor** corresponda ao peão escolhido, em direcção à meta. Não é permitido mover um peão para trás.

Depois, o jogodor retira o dico do percurso. Ele pode escolher **entre o disco livre à frente** do peão que acabou de mover, ou **entre o disco primeiro livre atrás** do peão que acabou de mover. Os discos já ocupados não podem ser retirados do percurso. Cada jogador guarda os seus discos (escondidos) na palma da mão até ao final do jogo.

Exemplo:
O jogador move o peão azul para o disco azul seguinte. Em seguida, ele pode ficar com o disco verde que se encontra à frente do peão azul (ilustração de cima), ou com o disco preto que se encontra atrás do peão azul (ilustração de baixo).

- Nota: se, no início, não houver discos livres atrás do peão, o jogador **tem** de ficar com o disco livre seguinte **na direcção do movimento.** Esta regra também se aplica movermos um peão para

um disco à frente da escada de chegada e não haja mais discos livres à frente desse peão; nesse caso, o jogador fica com o último disco livre que se encontre **atrás** do peão.

- Se não houver mais disco nenhum da cor correspondente ao peão, entre este e a escada de chegada, move-se o peão para a escada. O jogador coloca-o no degrau livre mais alto, de seguida pode retirar o disco da cor correspondente da pilha que se encontra ao lado da escada.

Os discos pretos

Se um jogador tirar um disco preto, pode utilizá-lo mais tarde para um turno suplementar:

- No momento em que o jogador decida utilizar um disco preto, ele pode - depois da sua vez - mover outro peão. Ele pode escolher o peão que acabou de mover ou outro peão. Depois, ele retira um disco - segundo as regras descritas anteriormente. Segue-se a vez do jogador seguinte.
- Durante o seu turno suplementar (e exclusivamente nesse), o jogador também pode mover um peão **para trás** colocando-o num disco da cor correspondente.

Não se pode usar mais que um disco preto no mesmo turno. Além disso, um disco preto **não pode usar-se no mesmo turno** em que foi conquistado. Ou seja, o jogador só pode usá-lo no turno seguinte à sua conquista.

Os discos pretos retiram-se do jogo depois de terem sido usados pelos jogadores e não voltam a ser utilizados.

Fim do jogo e pontuação

O jogo acaba quando o último peão é colocado na escada de chegada.

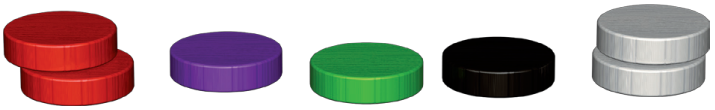
Em seguida, calculam-se os pontos:

- Cada disco vale tantos pontos quantos os indicados no degrau da escada do peão da cor correspondente.
- Os discos pretos não valem nada.
- Cada disco branco vale tantos pontos quanto o número de discos de cores diferentes que o jogador possua.

Exemplo :

No fim do jogo, a escada terá um aspecto como o da ilustração do lado.

O jogador tem os seguintes discos:



A sua pontuação será:

2 x vermelhos (4 pontos cada um) = 8 points

1 x violeta (2 pontos cada um) = 2 pontos

1 x verde (0 pontos cada um) = 0 pontos

1 x preto (0 pontos cada um) = 0 pontos

2 x brancos (além do branco, o jogador possui 4 cores diferentes por isso recebe 4 pontos por cada um) = 8 pontos

No total: 18 pontos

O jogador que obtiver mais pontos ganh o jogo. Em caso de empate, há vários vencedores!

Várias partidas

Como os jogos não são muito longos, podem fazer-se várias partidas. Jogar tantas partidas como o número de jogadores. Em cada uma dessas partidas, começa um novo jogador. Adicionar os resultados das diferentes partidas. O jogador com mais pontos ganha. Em caso de empate, há vários vencedores!