# *Big Points*
# Uma Análise Baseada na Teoria dos Jogos

Mateus M. F. Mendonça

5 de junho de 2017

# 1 Trabalho de Conclusão de Curso

## 1.1 Introdução

ada *nó* é um *estado*. *Pódio* e *escada* querem dizer mesma coisa.

## 1.2 Fundamentação Teórica

Teoria dos jogos é o estudo do comportamento estratégico interdependente[1], 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 tentam vencer individualmente (SPANIEL, 2011).

### 1.2.1 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, p. 2). No século seguinte, o matemático e filósofo Augustin Cournot fez uso da teoria dos jogos para estudos relacionados à política. Mais recentemente, em 1913, Ernst Zermelo publicou o primeiro teorema matemático da teoria dos jogos (SARTINI et al., 2004, p. 2).

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 (PRAGUE, 2004, p. 2), introduzindo uma noção abstrada sobre jogo estratégico e **estratégia mista**[2]. Em 1928, John von Neumann demonstrou que todo jogo finito[3] de **soma zero**[4] com duas pessoas possui uma solução em estratégias mistas. Em 1944, Neumann publicou um trabalho junto a Oscar Morgenstern introduzindo a teoria dos jogos na área da economia e matemática aplicada (SARTINI et al., 2004, p. 2–3).

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

[2]Estratégia mista é um conjunto de estratégias puras associadas a uma distribuição de probabilidade(FIGUEIREDO, 2001).

[3]Jogos finitos são aqueles onde cada participante se depara com um conjunto finito de escolhas, ou seja, eles escolhem suas estratégias dentro de um conjunto finito de alternativas(FIGUEIREDO, 2001).

[4]Um jogo soma zero é um jogo no qual a vitória de um jogador implica na derrota do outro.

### 1.2.2 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.

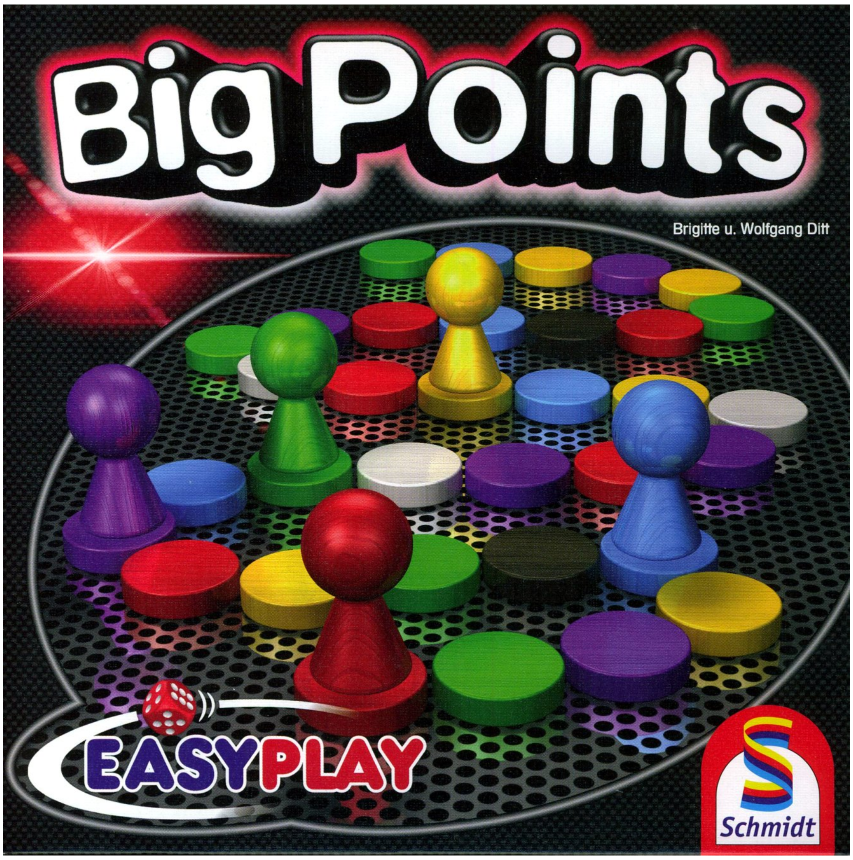
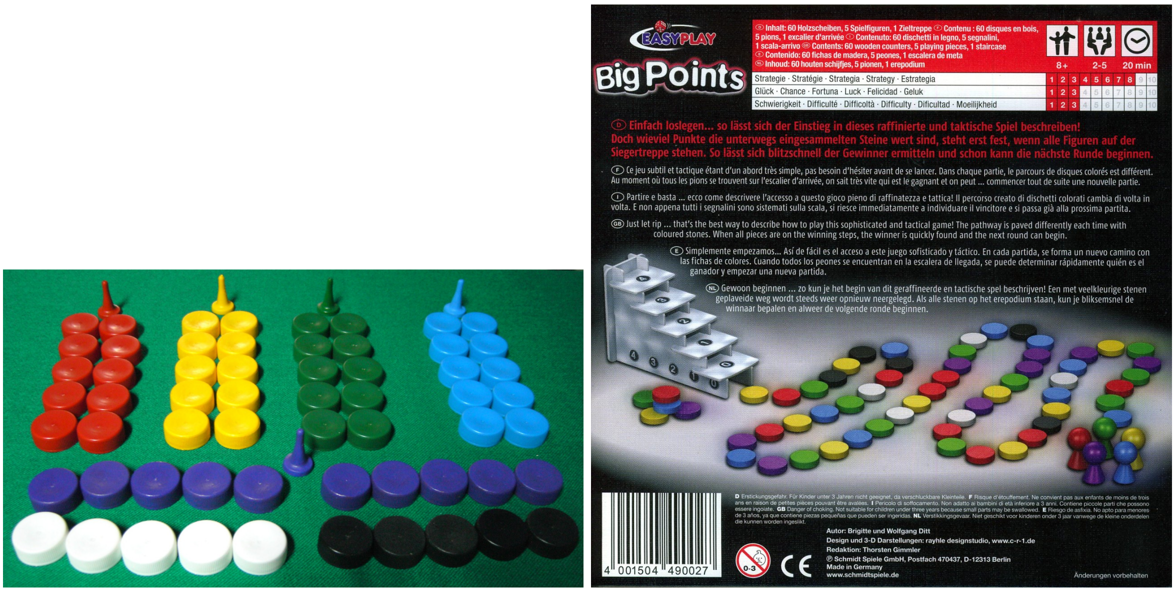



Figura 1 – Caixa do jogo **Big Points**

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[5], o segundo peão, três

[5]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.




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

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

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[6] 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.

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[7] 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

[6] 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.

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

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.

$$\begin{aligned}
Pontuacao &= n_r \cdot (n_c - p_r) + n_g \cdot (n_c - p_g) + n_b \cdot (n_c - p_b) \\
Pontuacao &= 2 \cdot (3 - 3) + 1 \cdot (3 - 2) + 3 \cdot (3 - 1) \qquad \text{(e.q. Exemplo de pontuação)} \\
Pontuacao &= 7
\end{aligned}$$

## 1.3 Metodologia

### 1.3.1 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*.

#### 1.3.1.1 Bit Fields

Dentro da estrutura `State` foi declarado duas estruturas anônimas[8] 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*

[8]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`.

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[9]; `_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 $i$[10]; `_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.

```
struct State
{
    // Cinco cores, quatro discos
    struct {
        // 5 cores * 4 discos (1bit pra cada)
        // 4 cores * 5 discos (1bit pra cada)
        ll _tabuleiro :20;

        // 0..5 posições possíveis (3bits) * 5 peões
        ll _peao :15;

        // 0..6 posições (3bits) * 5 peões
        ll _escada :15;
    };

    struct {
        // 0..6 discos (3bits) * 5 cores * 2 jogadores
        ll _jogadores :30;

        // Jogador 1 ou Jogador 2
        ll _atual :1;
    };
```

O cálculo para determinar os *bits* necessários para armazenar as informações de

[9]Cinco cores e quatro discos ou quatro cores e cinco discos.

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

cada variável foi realizado da seguinte forma:

$$
\begin{aligned}
\_\texttt{tabuleiro} &= n_c \cdot n_d \\
\_\texttt{tabuleiro} &= \max(4 \cdot 5,\ 5 \cdot 4) \\
\_\texttt{tabuleiro} &= 20 \textit{ bits}
\end{aligned}
\qquad \text{(e.q. \textit{bits} de \_tabuleiro)}
$$

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

$$
\begin{aligned}
\_\texttt{peao} &= \lceil \log_2(n_d + 1) \rceil \cdot n_p \\
\_\texttt{peao} &= \max\left(\lceil \log_2(4+1) \rceil \cdot 5,\ \lceil \log_2(5+1) \rceil \cdot 4\right) \\
\_\texttt{peao} &= \max(3 \cdot 5,\ 3 \cdot 4) \\
\_\texttt{peao} &= 15 \textit{ bits}
\end{aligned}
\qquad \text{(e.q. \textit{bits} de \_peao)}
$$

Na segunda equação, e.q. *bits* de _peao, 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 $peao(p_i) = n_d + 1$.

$$
\begin{aligned}
\_\texttt{escada} &= \lceil \log_2(n_c + 1) \rceil \cdot n_p \\
\_\texttt{escada} &= \lceil \log_2(6) \rceil \cdot 5 \\
\_\texttt{escada} &= 15 \textit{ bits}
\end{aligned}
\qquad \text{(e.q. \textit{bits} de \_escada)}
$$

A equação e.q. *bits* de _escada 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[11].

$$
\begin{aligned}
\_\texttt{jogadores} &= \lceil \log_2(n_d + 1) \rceil \cdot n_c \cdot n_j \\
\_\texttt{jogadores} &= \max(\lceil \log_2(5+1) \rceil \cdot 4 \cdot 2,\ \lceil \log_2(4+1) \rceil \cdot 5 \cdot 2) \\
\_\texttt{jogadores} &= \max(3 \cdot 4 \cdot 2,\ 3 \cdot 5 \cdot 2) \\
\_\texttt{jogadores} &= 30 \textit{ bits}
\end{aligned}
\qquad \text{(e.q. \textit{bits} de \_jogadores)}
$$

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

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

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 $\lceil \log_2(6) \rceil = 3 bits$

$$\begin{aligned} \texttt{\_atual} &= \lceil \log_2(2) \rceil \\ \texttt{\_atual} &= 1\ bit \end{aligned} \qquad \text{(e.q. } bits \text{ de \_atual)}$$

#### 1.3.1.2 Funções de Acesso

A estrutura possui um construtor que atribui valores às variáveis através de RAII[12], dessa forma não se faz necessário nenhuma extra implementação. Todas as variáveis possuem um valor padrão, verdadeiro para qualquer tamanho de tabuleiro $t_i$, onde $4 \leqslant t_i \leqslant 20$.

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

int tabuleiro (int pos) const {
    return (_tabuleiro & (1<<pos))>>pos;
}

void settabuleiro (int pos, int available) {
    _tabuleiro = (_tabuleiro & ~(1<<pos)) | ((available&1)<<pos);
}
```

#### 1.3.1.3 Comparador

### 1.3.2 Programação dinâmica

- Duas funções para melhor entendimento da DP e regras do jogo
- Explicação da DP e da função Play (função para realizar as jogadas)

### 1.3.3 Scrum

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

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

valor. Inicialmente, tal método de organização e gerenciamento do projeto foi aplicado para o desenvolvimento do sistema em questão (SCHWABER; SUTHERLAND, 2016). O *kanban* do waffle.io 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[13]. Por fim, para trabalhar em outra *branch*, foi sempre necessário atualizá-la em relação à mestre[14].

Quantidade de partidas

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

(e.q. Quantidades de Partidas Distintas)

### 1.3.4 Análise do jogo

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 repetidos 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 reduzido, temos

[13] $ git merge
[14] $ git rebase

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 1.

Tabela 1 – 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.

### 1.3.5 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[^dynamic_programing] onde os estados são armazenados em uma *hash*, temos que o número de estados distintos varia entre 17 e 25.

#### 1.3.5.1 Estado do jogo

Para escrever a programação dinâmica capaz de

#### 1.3.5.2 Verificação dos estados

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

# Referências


CUBBI; MAGGYERO; FRUDERICA. *RAII*. ⟨http://en.cppreference.com/w/cpp/language/raii⟩. Accessed May 31, 2016.

FIGUEIREDO, R. S. *Teoria dos Jogos*: Conceitos, formalização matemática e aplicação à distribuição de custo conjunto. [S.l.]: Universidade Federal de São Carlos, 2001.

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⟩.

SARTINI, B. A. et al. *Uma Introdução a Teoria dos Jogos*. 2004.

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

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