

UNIVERSIDADE FEDERAL DE MINAS GERAIS

Instituto de Ciências Exatas

Ciências Atuarias

Pedro Gomes Mundim Bar

Orientador: Prof. Dr. Gilcione Nonato Costa

UM ESTUDO PROBABILÍSTICO DO JOGO DE BURACO

Belo Horizonte

2023

# Sumário

Introdução .....	3
O jogo de buraco.....	3
Um breve panorama .....	3
Sequências válidas.....	3
Preparação.....	4
Rodadas e turnos.....	4
Pegando o morto .....	4
Fim da partida .....	4
Pontuação.....	5
O Algoritmo que determina a melhor jogada.....	7
Passos a serem seguidos .....	8
Probabilidade de comprar uma carta no monte .....	9
Separando as cartas em jogos .....	10
O valor esperado de pontos .....	14
Combinações possíveis de serem feitas .....	15
Probabilidades de transição .....	17
Probabilidade da batida .....	30
Probabilidade de ter as cartas.....	32
Implementando o algoritmo no R.....	35
O que pode ser aproveitado na prática.....	35
Resultados .....	39
Conclusão .....	41

# Introdução

Esse trabalho propõe um estudo probabilístico do jogo de buraco, um jogo de cartas popular, porém sem nenhuma literatura relevante e acessível sobre a probabilidade dentro do jogo. O ponto principal da pesquisa reside no desenvolvimento de um algoritmo matemático capaz de determinar a jogada mais otimizada em cada rodada de uma partida de buraco, considerando as informações disponíveis para o jogador em determinado momento do jogo.

Além de elaborar o algoritmo num campo teórico, o trabalho visa implementá-lo, a fim de proporcionar uma análise empírica da eficácia do modelo proposto. E, com o algoritmo implementado, simular algumas partidas reais, tanto para testá-lo em situações reais de jogo quanto para coletar dados relevantes sobre o próprio jogo de buraco, enriquecendo a compreensão das estratégias e padrões de jogo.

O estudo não apenas contribui para a compreensão mais aprofundada do jogo de buraco, mas também destaca a aplicação prática de métodos probabilísticos e algoritmos matemáticos no contexto de jogos de cartas. A abordagem oferece uma perspectiva interessante sobre a tomada de decisões estratégicas em ambientes dinâmicos e incertos.

Para se compreender a metodologia do projeto e o funcionamento do algoritmo é preciso entender de forma aprofundada o funcionamento do jogo.

## O jogo de buraco

### Um breve panorama

O jogo de Buraco é um jogo de cartas, jogado com dois baralhos de 52 cartas, em que duas duplas jogam em turnos com o objetivo de fazer sequências de cartas do mesmo naipe, cada sequência com sua devida pontuação, e competem para ter o maior número de pontos ao final da partida.

### Sequências válidas

No jogo de buraco o jogador pode pegar uma sequência de cartas válidas em sua mão e a colocar na mesa, ela passa a ser visível para todos os jogadores e seu parceiro pode adicionar cartas a essa sequência. No entanto nem todas as combinações de cartas são válidas. Para que o jogador possa descer um jogo ou complementar um jogo existente ele precisa seguir as seguintes regras:

1. As cartas devem ser do mesmo naipe;
2. As cartas devem estar na ordem do baralho (ás, dois, três, quatro, cinco, seis, sete, oito, nove, dez, valete, dama, rei, ás);

3. O dois pode funcionar como um coringa, substituindo qualquer carta de qualquer sequência, mesmo que não seja do mesmo naipe da sequência. Só pode haver um coringa por sequência;

## **Preparação**

No começo do jogo as duplas sentam-se intercaladas e de forma com que nenhum jogador consiga ver as cartas de nenhum outro jogador. Em seguida, 11 cartas são distribuídas aleatoriamente para cada um dos 4 jogadores, e somente o próprio jogador conhece suas cartas, essas são chamadas as “mãos” dos jogadores. Além disso, são distribuídas, também aleatoriamente 22 cartas em duas pilhas, de 11 cartas cada uma, com suas faces viradas para baixo e desconhecidas por todos os jogadores, esses são os “mortos”. Assim, restam 38 cartas, por consequência aleatórias, com suas faces viradas para baixo e desconhecidas por todos os jogadores, esse é o “monte”. Com as mãos distribuídas, os mortos separados e o monte no centro da mesa o jogo está pronto para começar.

## **Rodadas e turnos**

A cada rodada todos os jogadores têm um turno. Em seu turno o jogador compra uma carta no monte ou todas as cartas do “lixo” (local onde as cartas são descartadas), decide se vai descer alguma sequência válida de cartas para a mesa ou complementar alguma sequência que já esteja na mesa e, ao fim da sua jogada, descarta uma carta para o lixo.

O turno do primeiro jogador a jogar na primeira rodada é um pouco diferente. Como não há cartas no lixo, o jogador compra uma carta do monte e pode decidir se vai descartá-la e comprar uma nova carta ou prosseguir a jogada com essa carta.

## **Pegando o morto**

O primeiro jogador de cada dupla a ficar sem nenhuma carta na mão “bate”. O jogador que bateu pega um dos mortos para ser a sua nova mão. o jogador pode bater antes de descartar uma carta na sua jogada e nesse caso ele segue jogando depois de pegar o morto. Ele também pode bater ao descartar uma carta e nesse caso ele pega as cartas do morto, porém só volta a jogar na próxima rodada

## **Fim da partida**

A partida de buraco pode acabar de duas formas:

- 1.Caso a dupla tenha uma sequência de 7 cartas sem nenhum coringa, uma “canastra limpa”, e já tiver pegado o morto, o jogador da dupla que ficar sem nenhuma carta na mão faz a batida final e a partida acaba.


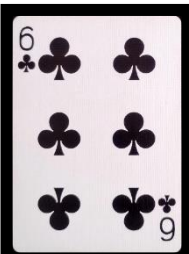

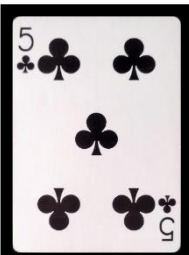

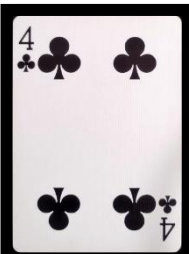
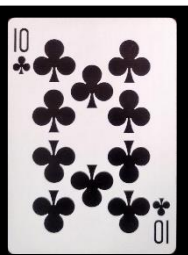
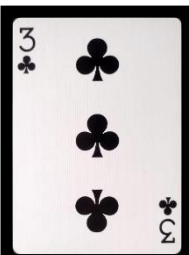
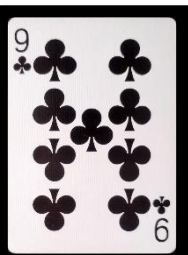
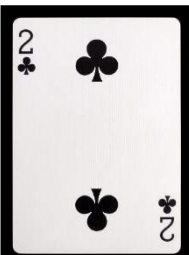
- 2.Caso as cartas do monte acabem, um dos mortos se torna o novo monte, caso não haja nenhum morto na mesa quando as cartas do monte acabarem a partida também acaba. Se há uma carta encartada no lixo, ou seja, uma carta que complementa algum

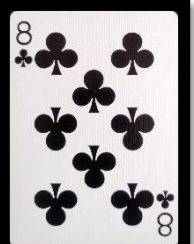
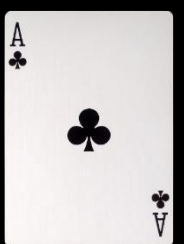
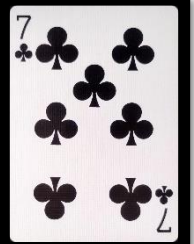
jogo da mesa, na rodada em que as cartas do monte acabaram, o próximo jogador pode fazer seu turno antes da partida acabar.

Quando a partida acaba os pontos são contados e anotados, e começa-se uma nova partida. Usualmente a dupla que faz 3000 pontos primeiro é a vencedora.

### Pontuação

Cada carta tem um valor de pontuação conforme quadro abaixo:

	10 pontos		5 pontos
	10 pontos		5 pontos
	10 pontos		5 pontos
	10 pontos		5 pontos
	10 pontos		10 pontos

	10 pontos		15 pontos
	5 pontos		

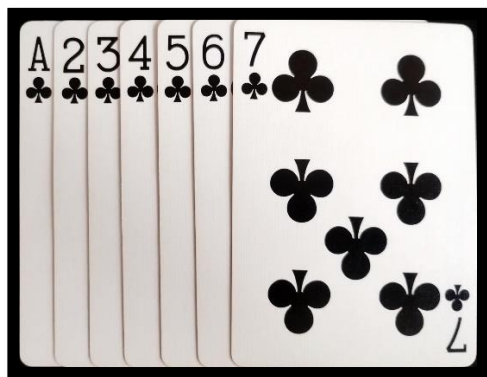
Jogos com 7 ou mais cartas e sem coringa são as canastras limpas, e valem 200 pontos



Jogos com 7 ou mais cartas e com coringa são as canastras sujas, e valem 100 pontos



Caso o coringa esteja na posição do 2, o jogo conta como uma canastra limpa se o coringa for do mesmo naipe que o jogo e como uma canastra suja se o coringa for de um naipe diferente:



Por fim, a batida final vale 100 pontos para a dupla que a realizar e serão deduzidos 100 pontos caso a dupla não pegue o morto.

## O Algoritmo que determina a melhor jogada

Um algoritmo é um conjunto de instruções passo a passo, claramente definido e finito, projetado para realizar uma tarefa específica ou resolver um problema. Essas

instruções descrevem uma sequência de operações que devem ser executadas em uma ordem específica para atingir um objetivo ou produzir um resultado desejado.

Tendo em mente o funcionamento do jogo, a melhor jogada, de um ponto de vista matemático, é aquela que dá a sua dupla a maior probabilidade de ter mais pontos que a dupla adversária ao final da partida. Sendo assim, a ideia central do algoritmo é calcular, para cada diferente jogada possível, o valor esperado de pontos das duas duplas ao final da partida, e fazer a jogada que apresenta maior diferença entre os pontos da sua dupla para a dupla adversária, para que na corrida até os 3000 pontos a sua dupla saia na vantagem. É importante notar que a maior diferença de pontos esperados não se traduz literalmente para a maior probabilidade de vitória, mas é uma aproximação boa o suficiente.

## **Passos a serem seguidos**

Primeiro estabelecemos de forma superficial os passos a serem seguidos no nosso algoritmo para que posteriormente possamos expandir cada um deles e por fim traçá-los novamente de forma mais precisa.

1. Calcular a probabilidade de se comprar cada uma das cartas no monte.
2. Para cada carta comprada você terá uma combinação diferente de cartas na mão. Para cada mão possível, listar as diferentes jogadas que podem ser feitas, ou seja, as diferentes formas de separar as cartas em jogos diferentes e separar uma carta para ser descartada.
3. Para cada jogada diferente, calcular o valor esperado de pontos ao final da partida para cada um desses jogos diferentes e somá-los para encontrar o valor esperado de pontos da sua dupla ao final da partida para aquela jogada.
4. Para cada jogada diferente calcular o valor esperado de pontos ao final da partida da dupla adversária considerando a sua carta que será descartada.
5. Subtrair o valor esperado de pontos da dupla adversária do valor esperado de pontos dos seus jogos, para encontrar o valor esperado da diferença de pontos da sua dupla para a dupla adversária.
6. Pegar, para cada mão diferente, a melhor jogada, ou seja, a jogada que proporciona o maior valor esperado da diferença de pontos da sua dupla para a dupla adversária.
7. Listar as diferentes jogadas que podem ser feitas para a combinação de cartas na sua mão caso você opte por comprar todas as cartas do lixo, repetir os passos 3, 4, 5 para essa mão e pegar a melhor jogada.
8. Calcular se em média as melhores jogadas feitas para cada carta comprada no monte são melhores que as melhores jogada feitas com as cartas compradas no lixo.
9. Com base no resultado do passo 8 estabelecer se a melhor escolha é comprar do monte ou do lixo.
10. Após comprar a carta de onde for melhor, sua mão agora é conhecida por você. Então, basta fazer a melhor jogada, já estabelecida no passo 6, para essa determinada combinação de cartas.

## Probabilidade de comprar uma carta no monte

A probabilidade de comprar uma determinada carta do monte é dada por:  $P_N = N/b$ , onde  $N$  é a quantidade dessa carta no monte e  $b$  é a quantidade total de cartas no monte

$N$  é uma variável aleatória discreta, uma vez que o jogador não tem a informação de quantas cópias de cada carta restam no monte.  $N$  assume os valores  $\{0, 1, 2\}$ , com probabilidade igual à probabilidade de uma das cópias da carta estar no monte vezes a probabilidade da outra cópia da carta estar no monte.

Se o jogador não sabe onde nenhuma das cópias da carta estão, então a probabilidade da cópia 1 estar no monte é igual  $b/\hat{b}$ , onde  $b$  é a quantidade total de cartas no monte e  $\hat{b}$  é o total de cartas desconhecidas pelo jogador, ou seja, as cartas do monte, as cartas dos mortos e as cartas das mãos dos outros jogadores que ainda não foram reveladas. Por consequência a probabilidade da cópia 1 não estar no monte é  $1 - (b/\hat{b})^2$ . As probabilidades para a cópia 2 são exatamente as mesmas.

Sendo assim, a função de distribuição de probabilidade de  $N$  é dada por:

$$f(N) = \begin{cases} \left(1 - (b/\hat{b})\right)^2, & \text{se } N = 0 \\ 2 \times (b/\hat{b}) \times \left(1 - (b/\hat{b})\right), & \text{se } N = 1 \\ (b/\hat{b})^2, & \text{se } N = 2 \\ 0, & \text{para outros valores de } N \end{cases}$$

Se o jogador sabe que uma das cartas não está no monte, então a probabilidade da cópia 1 estar no monte é igual  $b/\hat{b}$ , onde  $b$  é a quantidade total de cartas no monte e  $\hat{b}$  é o total de cartas desconhecidas pelo jogador. Por consequência a probabilidade da cópia 1 não estar no monte é  $1 - (b/\hat{b})^2$ . Porém a probabilidade da cópia 2 estar no monte é 0, pois é sabido que está fora do monte.

Assim, a função de distribuição de probabilidade de  $N$  nesse caso é dada por:

$$f(N) = \begin{cases} \left(1 - (b/\hat{b})\right), & \text{se } N = 0 \\ (b/\hat{b}), & \text{se } N = 1 \\ 0, & \text{para outros valores de } N \end{cases}$$

Por fim, se o jogador sabe que ambas as cópias da carta estão fora do monte é trivial que não há nenhuma cópia da carta no monte. Logo, a função de distribuição de probabilidade de  $P_N = N/b$  é dada por:

$$f(P_N) = \begin{cases} \left(1 - (b/\hat{b})\right)^2, & \text{se } P = 0 \\ 2 \times (b/\hat{b}) \times \left(1 - (b/\hat{b})\right), & \text{se } P = 1/b \\ (b/\hat{b})^2, & \text{se } P = 2/b \\ 0, & \text{para outros valores de } P \end{cases}$$

se o jogador não sabe se alguma cópia da carta está fora do monte.

$$f(P_N) = \begin{cases} \left(1 - (b/\hat{b})\right), & \text{se } P = 0 \\ (b/\hat{b}), & \text{se } P = 1/b \\ 0, & \text{para outros valores de } x \end{cases}$$

se o jogador sabe que uma cópia da carta está fora do monte.

$$f(P_N) = \begin{cases} 1, & \text{se } P = 0 \\ 0, & \text{para outros valores de } x \end{cases}$$

se o jogador sabe que as duas cópias da carta estão fora do monte.

## Separando as cartas em jogos

O número esperado de pontos é calculado de forma separada para cada jogo, sendo assim é preciso separar as cartas na mão e na mesa em jogos diferentes e, para eleger a melhor jogada, é preciso considerar todas as formas possíveis de fazer essa separação.

É importante, primeiramente, estabelecer o que quer dizer separar cartas em jogos. Para esse algoritmo, quando falamos em separar as cartas em jogos estamos indo além de simplesmente agrupar as sequências de cartas que o jogador tem na mão. Separar as cartas em jogos diz respeito a agrupar as cartas da mão, da mesa e as cartas que podem ser compradas no mesmo grupo, para poder estimar o valor de pontos daquele determinado jogo.

Algumas regras nos ajudam a filtrar melhor as formas relevantes de separar os jogos:

1. Cartas de naipes diferentes, com exceção do coringa, sempre ficam em jogos diferentes;
2. Nem sempre cartas do mesmo naipe ficam no mesmo jogo, uma vez que pode ser vantajoso separá-las em jogos diferentes, para buscar uma canastra ou uma batida, por exemplo;
3. As cartas sempre devem estar sempre na ordem do baralho, uma vez que as sequências válidas são somente aquelas que respeitam essa ordem, e não faz sentido ter cartas que não fazem uma sequência válida dentro de um mesmo jogo.
4. Jogos com menos de 3 cartas não fazem sentido, uma vez que sequências de 1 e 2 cartas nem podem ser descidas e, portanto, jogos de 1 ou 2 cartas nunca geram pontos.

5. Os coringas podem participar de todos os jogos;
6. As cópias de uma mesma carta podem participar dos mesmos jogos.

A listagem de jogos tem que ser feita computacionalmente uma vez que são muitas possibilidades. Sendo assim, um código na linguagem de programação R foi implementado para essa e futuras tarefas no trabalho.

Primeiro listamos as cartas que podem formar um jogo, ou seja de um ás ao outro dentro de um mesmo naipe.

```
cartas = c("A", "3", "4", "5", "6", "7", "8", "9", "D", "J", "Q", "K", "A*")
```

Utilizamos o \* em "A\*" para diferenciar os ases, sendo já que o ás pode ser colocado antes do 2 ou depois do K, e deixamos o 2 de fora dessa sequência por ser o coringa e se comportar de forma diferente das outras cartas, podendo fazer parte de qualquer sequência como estabelecido previamente.

Em seguida separamos as cartas na sequência acima de todas as formas possíveis, respeitando as regras de que as cartas devem estar sempre em ordem do baralho e de que deve haver pelo menos 3 cartas em cada um dos jogos. Ou seja, simplificando o problema é como se precisássemos colocar um separador entre as cartas da lista, de forma com que nunca menos de 3 cartas ficassem juntas. Portanto, basta encontrar todas as formas de somar n números inteiros positivos maiores que 3 e menores que 14.

```
permutações_possiveis = list(c(3,11), c(4,10), c(5,9), c(6,8), c(7,7),
c(8,6), c(9,5), c(10,4), c(11,3), c(5,5,4), c(5,4,5), c(4,5,5), c(4,4,6),
c(4,6,4), c(6,4,4), c(3,3,8), c(3,8,3), c(8, 3, 3), c(3,4,7), c(3,7,4),
c(4,3,7), c(4,7,3), c(7,3,4), c(7,4,3), c(3,5,6), c(3,6,5), c(5,3,6),
c(5,6,3), c(6,3,5), c(6,5,3), c(3,3,3,5), c(3,3,5,3), c(3,5,3,3),
c(5,3,3,3), c(3,3,4,4), c(3,4,3,4), c(3,4,4,3), c(4,3,3,4), c(4,3,4,3),
c(4,4,3,3))
```

Com a lista de permutações de números possíveis que obedecem às regras estabelecidas, basta listar, utilizando as cartas, as organizações diferentes em jogos. Note que números indicam o tamanho dos jogos e basta colocar as cartas da nossa lista "cartas", em ordem, até o jogo tenha o tamanho determinado e partir para o próximo jogo.

```
organizações_jogos = list()
for (i in permutações_possiveis){
  if(length(i) == 2){jogo1 = cartas[1:i[1]]
jogo2 = cartas[(i[1]+1):(i[1]+i[2]-1)]
lista = list(jogo1,jogo2)
organizações_jogos= append(organizações_jogos,list(lista))
}
if(length(i) == 3){jogo1 = cartas[1:i[1]]
jogo2 = cartas[(i[1]+1):(i[1]+i[2])]
jogo3 = cartas[(i[1]+i[2]+1):(i[1]+i[2]+i[3]-1)]
lista = list(jogo1,jogo2,jogo3)
```

```

organizações_jogos= append(organizações_jogos,list(lista))
}
if(length(i) == 4){jogo1 = cartas[1:i[1]]
jogo2 = cartas[(i[1]+1):(i[1]+i[2])]
jogo3 = cartas[(i[1]+i[2]+1):(i[1]+i[2]+i[3])]
jogo4 = cartas[(i[1]+i[2]+i[3]+1):(i[1]+i[2]+i[3]+i[4]-1)]
lista = list(jogo1,jogo2,jogo3,jogo4)
organizações_jogos= append(organizações_jogos,list(lista))
}
}

```

Temos, então, uma lista com todas as formas diferentes de organizar em jogos as cartas de um mesmo naipe de um baralho. O primeiro item da lista, por exemplo, nos dá uma forma de separar esses jogos.

```

organizações_jogos[[1]]

## [[1]]
## [1] "A" "3" "4"
##
## [[2]]
## [1] "5" "6" "7" "8" "9" "D" "J" "Q" "K" "A*"

```

No entanto, temos dois baralhos no jogo de buraco, e as cartas dos dois baralhos podem participar do mesmo jogo. Portanto precisamos também considerar a possibilidade de as cartas em cada um desses jogos ser do primeiro ou segundo baralho. Observe que não faz diferença efetiva de qual baralho as cartas são originalmente, porém é relevante diferenciar as cartas entre a primeira a ser comprada e a segunda a ser comprada, para estabelecer uma prioridade na hora de alocar as cartas em determinados jogos.

```

todas_variações = expand.grid(replicate(length(cartas), c(0, 1), simplify
= FALSE))

todas_variações[1,]

##   Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8 Var9 Var10 Var11 Var12 Var13
## 1    0    0    0    0    0    0    0    0    0    0    0    0    0

todas_combinações = list()
for (i in 1:nrow(todas_variações)){
  todas_combinações =
append(todas_combinações,list(paste0(cartas,ifelse(todas_variações[i,]==1
,"*", " "))))
}
todas_combinações[[1]]

## [1] "A " "3 " "4 " "5 " "6 " "7 " "8 " "9 " "D " "J " "Q "
"K "
## [13] "A* "

```

Isso nos dá uma lista com todas as combinações possíveis de cartas de dois baralhos, onde cada vetor é uma combinação possíveis e cada elemento no vetor indica uma carta, sendo o primeiro caractere desse elemento o número da carta e o último caractere indicando a ordem de prioridade.

Para cada uma dessa combinações, fazemos os passos anteriores para ter todas as organizações da mão em jogos diferentes para cada uma das combinações de cartas dos dois baralhos.

```
organizações_jogos = list()
for(h in todas_combinações){
  for (i in permutações_possiveis){
    if(length(i) == 2){jogo1 = h[1:i[1]]
    jogo2 = h[(i[1]+1):(i[1]+i[2]-1)]
    lista = list(jogo1,jogo2)
    organizações_jogos= append(organizações_jogos,list(lista))
    }
    if(length(i) == 3){jogo1 = h[1:i[1]]
    jogo2 = h[(i[1]+1):(i[1]+i[2])]
    jogo3 = h[(i[1]+i[2]+1):(i[1]+i[2]+i[3]-1)]
    lista = list(jogo1,jogo2,jogo3)
    organizações_jogos= append(organizações_jogos,list(lista))
    }
    if(length(i) == 4){jogo1 = h[1:i[1]]
    jogo2 = h[(i[1]+1):(i[1]+i[2])]
    jogo3 = h[(i[1]+i[2]+1):(i[1]+i[2]+i[3])]
    jogo4 = h[(i[1]+i[2]+i[3]+1):(i[1]+i[2]+i[3]+i[4]-1)]
    lista = list(jogo1,jogo2,jogo3,jogo4)
    organizações_jogos= append(organizações_jogos,list(lista))
    }
  }
}
```

Então, basta considerar as cartas da outra metade dos dois baralhos.

Intuitivamente a outra metade, composta pelas exatas mesmas cartas, pode ser organizada das mesmas formas que já listamos para a primeira metade do baralho. Então, listamos todas as combinações entre as formas como podemos organizar a primeira e o segunda metade, levando em consideração que se na primeira metade uma determinada carta vem do primeiro baralho na segunda metade essa mesma carta tem que vir do segundo baralho e vice-versa.

```
a = c(1,1,1,1,1,1,1,1,1,1,1,1)
variações_outrametade = list()
for (i in 1:nrow(todas_variações)){
  variações_outrametade = append(variações_outrametade, list(a-
todas_variações[i,]))
}

todas_combinações_outrametade = list()
```

```

for (i in 1:length(variações_outrametade)){
  todas_combinações_outrametade =
append(todas_combinações_outrametade, list(paste0(cartas, ifelse(variações_
outrametade[i,]==1, "*", " "))))
}

organizações_jogos_outrametade = list()
for(h in todas_combinações_outrametade){
  for (i in permutações_possiveis){
    if(length(i) == 2){jogo1 = h[1:i[1]]
jogo2 = h[(i[1]+1):(i[1]+i[2]-1)]
lista = list(jogo1, jogo2)
organizações_jogos_outrametade=
append(organizações_jogos_outrametade, list(lista))
}
if(length(i) == 3){jogo1 = h[1:i[1]]
jogo2 = h[(i[1]+1):(i[1]+i[2])]
jogo3 = h[(i[1]+i[2]+1):(i[1]+i[2]+i[3]-1)]
lista = list(jogo1, jogo2, jogo3)
organizações_jogos_outrametade=
append(organizações_jogos_outrametade, list(lista))
}
if(length(i) == 4){jogo1 = h[1:i[1]]
jogo2 = h[(i[1]+1):(i[1]+i[2])]
jogo3 = h[(i[1]+i[2]+1):(i[1]+i[2]+i[3])]
jogo4 = h[(i[1]+i[2]+i[3]+1):(i[1]+i[2]+i[3]+i[4]-1)]
lista = list(jogo1, jogo2, jogo3, jogo4)
organizações_jogos_outrametade=
append(organizações_jogos_outrametade, list(lista))
}
}
}

organizações_jogos_2metades = list()
for (i in 1:length(organizações_jogos)){
  nova_organização =
list(organizações_jogos[[i]], organizações_jogos_outrametade[[i]])
  organizações_jogos_2metades =
append(organizações_jogos_2metades, list(nova_organização))
}

```

Agora temos todas as formas possíveis de organizar os jogos de um naipe.

## O valor esperado de pontos

Agora que separamos as cartas em diferentes jogos precisamos encontrar uma forma de calcular o valor esperado de pontos para cada um desses jogos.

No final do jogo o que rende pontos para dupla são as sequências de cartas feitas na mesa. A probabilidade com que o jogo assume a forma de uma determinada sequência

em alguma rodada do jogo depende da probabilidade de que a sua dupla tenha as cartas para fazer aquela sequência e da combinação na rodada anterior.

A situação acima pode ser descrita como uma cadeia de Markov, descrita formalmente no livro “Introduction to Probability Models, Tenth Edition” de Sheldon M. Ross, como

*Seja  $\{X_n, n = 0, 1, 2, \dots\}$  um processo estocástico que assume um número finito ou contável de valores possíveis. A menos que seja mencionado o contrário, este conjunto de valores possíveis do processo será denotado pelo conjunto dos números inteiros não negativos  $\{0, 1, 2, \dots\}$ . Se  $X_n = i$ , então diz-se que o processo está no estado  $i$  no tempo  $n$ . Supomos que sempre que o processo está no estado  $i$ , existe uma probabilidade fixa  $P_{ij}$  de que ele estará no estado  $j$ . Ou seja, supomos que*

$$P\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \dots, X_1 = i_1, X_0 = i_0\} = P_{ij}$$

*para todos os estados  $i_0, i_1, \dots, i_{n-1}, i, j$  e todos os  $n \geq 0$ . Um processo estocástico com essa propriedade é conhecido como uma cadeia de Markov homogênea.*

No caso em estudo, nós temos, para cada jogo a ser feito ou considerado, um processo estocástico  $X_n, n = 0, 1, 2, \dots, t$  em que  $n$  representa cada rodada da partida, sendo  $t$  a última rodada, os estados são as combinações possíveis e as probabilidades de transição são as probabilidades de, estando em uma determinada combinação, ir para outra combinação.

## **Combinações possíveis de serem feitas**

Dentro de cada naipe há uma série de combinações possíveis de serem feitos, e esses serão os estados da nossa cadeia de Markov:

- 12 combinações de 3 cartas: A23, 234, 345, 456, 567, 678, 789, 89D, 9DJ, DJQ, JQK, QKA;
- 11 combinações de 4 cartas: A234, 2345, 3456, 4567, 5678, 6789, 789D, 89DJ, 9DJQ, DJQK, JQKA;
- 10 combinações de 5 cartas: A2345, 23456, 34567, 45678, 56789, 6789D, 789DJ, 89DJQ, 9DJQK, DJQKA;
- 9 combinações de 6 cartas: A23456, 234567, 345678, 456789, 56789D, 6789DJ, 789DJQ, 89DJQK, 9DJQKA;
- 8 combinações de 7 cartas: A234567, 2345678, 3456789, 456789D, 56789DJ, 6789DJQ, 789DJQK, 89DJQKA;
- 7 combinações de 8 cartas: A2345678, 23456789, 3456789D, 456789DJ, 56789DJQ, 6789DJQK, 789DJQKA;
- 6 combinações de 9 cartas: A23456789, 23456789D, 3456789DJ, 456789DJQ, 56789DJQK, 6789DJQKA;

- 5 combinações de 10 cartas: A23456789D, 23456789DJ, 3456789DJQ, 456789DJQK, 56789DJQKA;
- 4 combinações de 11 cartas: A23456789DJ, 23456789DJQ, 3456789DJQK, 456789DJQKA;
- 3 combinações de 12 cartas: A23456789DJQ, 23456789DJQK, 3456789DJQKA;
- 2 combinações de 13 cartas: A23456789DJQK, 23456789DJQKA;
- 1 combinações de 13 cartas: A23456789DJQKA.

Totalizando 78 combinações sem coringa. Além dessas combinações, existem as combinações com coringa. A carta 2 de qualquer naipe pode substituir qualquer carta de qualquer combinação sem coringa. Sendo assim, temos ainda:

- Para cada uma das 12 combinações de 3 cartas, 36 combinações de 3 cartas com coringa, uma vez que cada uma das três cartas pode ser substituída por um coringa para fazer uma combinação diferente, ou seja  $3 \times 12$  combinações possíveis.
- Para cada uma das 11 combinações de 4 cartas, 44 combinações de 4 cartas com coringa;
- Para cada uma das 10 combinações de 5 cartas, 50 combinações de 5 cartas com coringa;
- Para cada uma das 9 combinações de 6 cartas, 54 combinações de 6 cartas com coringa;
- Para cada uma das 8 combinações de 7 cartas, 56 combinações de 7 cartas com coringa;
- Para cada uma das 7 combinações de 8 cartas, 56 combinações de 8 cartas com coringa;
- Para cada uma das 6 combinações de 9 cartas, 54 combinações de 9 cartas com coringa;
- Para cada uma das 5 combinações de 10 cartas, 50 combinações de 10 cartas com coringa;
- Para cada uma das 4 combinações de 11 cartas, 44 combinações de 11 cartas com coringa;
- Para cada uma das 3 combinações de 12 cartas, 36 combinações de 12 cartas com coringa;
- Para cada uma das 2 combinações de 13 cartas, 26 combinações de 13 cartas com coringa;

- Para a combinação de 14 cartas, 14 combinações de 14 cartas com coringa.

Totalizando 520 combinações com coringa, e somando as combinações com e sem coringa encontramos o total de combinações possíveis, 598.

## Probabilidades de transição

Para a construção do algoritmo supomos que sempre que é possível fazer ou complementar um jogo, opta-se por fazer isso. Sendo assim a probabilidade de sair de uma sequência e ir para uma outra maior depende unicamente da probabilidade de possuir as cartas faltante para complementar o jogo daquela forma. Seja  $p_\alpha$  a probabilidade de uma dupla ter dentro daquele jogo um ás até o final da rodada,  $p_2$  um dois,  $p_3$  um três,  $p_4$  um quatro,  $p_5$  um cinco,  $p_6$  um seis,  $p_7$  um sete,  $p_8$  um oito,  $p_9$  um nove,  $p_d$  um dez,  $p_j$  um valete,  $p_q$  uma dama,  $p_k$  um rei e  $p_c$  um coringa.

Algumas probabilidades se repetem na construção da matriz de transição, portanto, para facilitar esse processo, vamos defini-las. Denominamos:

- $\alpha$  a probabilidade de se ter todas as cartas do estado  $j$  que não estão presentes no estado  $i$ .
- $\beta$  a probabilidade de não se ter uma carta que pode ser imediatamente adicionada ao jogo

Ou seja, caso o estado  $i$  seja “567” e o estado  $j$  “45678C” conforme figura abaixo:

$\alpha = p_4 \times p_8$ , a probabilidade de se ter todas as cartas do estado  $j$  que não estão presentes no estado  $i$ , ou seja, a probabilidade de se ter o 4 e o 8. E  $\beta = (1 - p_2) \times (1 - p_3) \times (1 - p_9) \times (1 - p_d)$ , a probabilidade de não se ter uma carta que pode ser adicionada imediatamente no estado  $j$ , ou seja, a probabilidade de não se ter o 2, o 3, o 9 ou o 10, que podem ser colocados na combinação do estado  $j$ . Estabelecidas as probabilidades acima, a probabilidade de ir para um outro estado  $j$  é dada por  $P_{ij}$

Como sempre que é possível fazer ou complementar um jogo opta-se por fazer isso, a probabilidade de ir do estado  $i$  para o estado  $j$  é a probabilidade de encontrar todas as cartas faltantes para ir para o estado  $j$ , porém não encontrar cartas que nos levariam para uma outra combinação ainda maior.

Por exemplo, se temos no estado  $i$  a sequência “567” a probabilidade de ir para o estado  $j$  de combinação “56789” é igual à probabilidade de encontrar o 8 e o 9, porém não encontrar o 4, o 10 ou um coringa, uma vez que essas cartas levariam para um outro estado, portanto:

$$P_{ij} = \alpha \times \beta$$

$$P_{ij} = p_8 \times p_9 \times (1 - p_4) \times (1 - p_d) \times (1 - p_c)$$

De forma semelhante, se temos no estado  $i$  a sequência “567” a probabilidade de ir para o estado  $j$  de combinação “45678C” é igual à probabilidade de encontrar o 8 e o

9, porém não encontrar o 3, o 9, o 2 ou o 10, uma vez que essas cartas levariam para um outro estado, portanto:

$$P_{ij} = \alpha \times \beta P_{ij} = p_4 \times p_8 \times (1 - p_3) \times (1 - p_9) \times (1 - p_2) \times (1 - p_d)$$

Por fim, se temos no estado  $i$  a combinação "567" a probabilidade de ir para o estado  $j$  de combinação "4567C9" é igual à probabilidade de encontrar o 4 e o 9, porém não encontrar o 3, o 8 ou 10, uma vez que essas cartas levariam para um outro estado, portanto:

$$P_{ij} = \alpha \times \beta P_{ij} = p_4 \times p_9 \times (1 - p_3) \times (1 - p_d) \times (1 - p_8)$$

Quando estamos fazendo um novo jogo temos que tomar alguns cuidados especiais. Se nosso estado  $i$  não tem nenhuma carta pode parecer que a fórmula  $P_{ij} = \alpha \times \beta$  nos dá a probabilidade de ir para um estado  $j$  qualquer. Tomamos o exemplo abaixo:

Se estamos indo do estado  $i$  " ", para o estado  $j$  onde temos a sequência "567", temos:

$$P_{ij} = \alpha \times \beta P_{ij} = p_5 \times p_6 \times p_7 \times (1 - p_4) \times (1 - p_8) \times (1 - p_c)$$

Porém, essa fórmula gera um problema quando analisamos todos os casos em conjunto. Imaginemos agora que nosso estado  $j$  é "JQK", então:

$$P_{ij} = \alpha \times \beta P_{ij} = p_j \times p_q \times p_k \times (1 - p_d) \times (1 - p_a) \times (1 - p_c)$$

Se temos as cartas 5, 6, 7, J, Q, K em nossa mão, temos que a probabilidade de ir para "567" é 1, assim como a probabilidade de ir para "JQK" também é 1, o que não é possível. Portanto precisamos estabelecer uma outra variável que estabeleça uma ordem de prioridade para os jogos com o mesmo número de cartas.

Estabelecidos os estados e as probabilidades de transição, conseguimos montar a matriz de transição da nossa cadeia de Markov.

Primeiro listamos os estados, ou seja, as sequências possíveis.

```
#sequencias sem coringa
jogos_possiveis =
c("A23", "A234", "A2345", "A23456", "A234567", "A2345678", "A23456789", "A23456789D", "A23456789DJ", "A23456789DJQ", "A23456789DJQK", "A23456789DJQKA",
"234", "2345", "23456", "234567", "2345678", "23456789", "23456789D", "23456789DJ", "23456789DJQ", "23456789DJQK", "23456789DJQKA",
"345", "3456", "34567", "345678", "3456789", "3456789D", "3456789DJ", "3456789DJQ", "3456789DJQK", "3456789DJQKA",
"456", "4567", "45678", "456789", "456789D", "456789DJ", "456789DJQ", "456789DJQK", "456789DJQKA",
"567", "5678", "56789", "56789D", "56789DJ", "56789DJQ", "56789DJQK", "56789DJQKA",
"A",
```

```

"678", "6789", "6789D", "6789DJ", "6789DJQ", "6789DJQK", "6789DJQKA",
  "789", "789D", "789DJ", "789DJQ", "789DJQK", "789DJQKA",
  "89D", "89DJ", "89DJQ", "89DJQK", "89DJQKA",
  "9DJ", "9DJQ", "9DJQK", "9DJQKA",
  "DJQ", "DJQK", "DJQKA",
  "JQK", "JQKA",
  "QKA")

```

*#sequencias com coringa*

```

jogos_possiveis_coringa = c()
for (i in jogos_possiveis){
  for (j in 1:nchar(i)){
    jogos_possiveis_coringa =
c(jogos_possiveis_coringa, gsub(substr(i,j,j), "C", i))
  }
}

```

*#todos as sequências possíveis*

```

todos_os_jogos = c(jogos_possiveis, jogos_possiveis_coringa)

```

Agora que os estados estão listados, montamos nossa matriz de transição.

```

teste = c()
todos_os_jogos1 = todos_os_jogos
for (jogo in 1:length(todos_os_jogos1)){
  todos_os_jogos_alt = todos_os_jogos1
  for (carta in 1:nchar(todos_os_jogos1[jogo])){
    for (jogo_possivel in 1:length(todos_os_jogos1)){
      todos_os_jogos_alt[jogo_possivel] =
ifelse(grepl(substr(todos_os_jogos1[jogo], carta, carta), todos_os_jogos1[jo
go_possivel]), gsub(substr(todos_os_jogos1[jogo], carta, carta), "", todos_os_
jogos_alt[jogo_possivel]), "X")
    }
  }
  teste = c(teste, todos_os_jogos_alt)
}

```

*#probabilidade de encontrar as cartas*

```

prob = c()
for (jogo in teste){
  prob1 = "1"
  for (carta in 1:nchar(jogo)){
    if (substr(jogo, carta, carta)=="A"){prob1 = paste(prob1, "*pa", sep =
""))}
    if (substr(jogo, carta, carta)=="2"){prob1 = paste(prob1, "*p2", sep =
""))}
    if (substr(jogo, carta, carta)=="3"){prob1 = paste(prob1, "*p3", sep =
""))}
    if (substr(jogo, carta, carta)=="4"){prob1 = paste(prob1, "*p4", sep =

```

```

    ""})
    if (substr(jogo,carta,carta)=="5"){prob1 = paste(prob1,"*p5",sep =
    ""})
    if (substr(jogo,carta,carta)=="6"){prob1 = paste(prob1,"*p6",sep =
    ""})
    if (substr(jogo,carta,carta)=="7"){prob1 = paste(prob1,"*p7",sep =
    ""})
    if (substr(jogo,carta,carta)=="8"){prob1 = paste(prob1,"*p8",sep =
    ""})
    if (substr(jogo,carta,carta)=="9"){prob1 = paste(prob1,"*p9",sep =
    ""})
    if (substr(jogo,carta,carta)=="D"){prob1 = paste(prob1,"*pd",sep =
    ""})
    if (substr(jogo,carta,carta)=="J"){prob1 = paste(prob1,"*pj",sep =
    ""})
    if (substr(jogo,carta,carta)=="Q"){prob1 = paste(prob1,"*pq",sep =
    ""})
    if (substr(jogo,carta,carta)=="K"){prob1 = paste(prob1,"*pk",sep =
    ""})
    if (substr(jogo,carta,carta)=="A"){prob1 = paste(prob1,"*pa",sep =
    ""})
    if (substr(jogo,carta,carta)=="C"){prob1 = paste(prob1,"*pc",sep =
    ""})
    if (substr(jogo,carta,carta)=="X"){prob1 = paste(prob1,"*0",sep =
    ""})
    }
    prob = c(prob,prob1)
  }
}
#probabilidade de não ir para um jogo maior
contador = 0
for (jogo in 1:length(todos_os_jogos1)){
  for (jogo_possivel in 1:length(todos_os_jogos1)){
    contador = contador + 1
    if
    (substr(todos_os_jogos1[jogo_possivel],1,1)=="2"){prob[contador]=paste("(
    1-pa)*",prob[contador])}
    if
    (substr(todos_os_jogos1[jogo_possivel],1,1)=="3"){prob[contador]=paste("(
    1-p2)*",prob[contador])}
    if
    (substr(todos_os_jogos1[jogo_possivel],1,1)=="4"){prob[contador]=paste("(
    1-p3)*",prob[contador])}
    if
    (substr(todos_os_jogos1[jogo_possivel],1,1)=="5"){prob[contador]=paste("(
    1-p4)*",prob[contador])}
    if
    (substr(todos_os_jogos1[jogo_possivel],1,1)=="6"){prob[contador]=paste("(
    1-p5)*",prob[contador])}
    if
    (substr(todos_os_jogos1[jogo_possivel],1,1)=="7"){prob[contador]=paste("(

```

```

1-p6)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="8"){prob[contador]=paste("(
1-p7)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="9"){prob[contador]=paste("(
1-p8)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="D"){prob[contador]=paste("(
1-p9)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="J"){prob[contador]=paste("(
1-pd)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="Q"){prob[contador]=paste("(
1-pj)*", prob[contador])}
    if (substr(todos_os_jogos1[jogo_possivel],1,1)=="C"){
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="2"){prob[contador]=paste("(
1-pa)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="3"){prob[contador]=paste("(
1-pa)*(1-p2)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="4"){prob[contador]=paste("(
1-p2)*(1-p3)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="5"){prob[contador]=paste("(
1-p3)*(1-p4)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="6"){prob[contador]=paste("(
1-p4)*(1-p5)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="7"){prob[contador]=paste("(
1-p5)*(1-p6)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="8"){prob[contador]=paste("(
1-p6)*(1-p7)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="9"){prob[contador]=paste("(
1-p7)*(1-p8)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="D"){prob[contador]=paste("(
1-p8)*(1-p9)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="J"){prob[contador]=paste("(
1-p9)*(1-pd)*", prob[contador])}
    if
(substr(todos_os_jogos1[jogo_possivel],2,2)=="Q"){prob[contador]=paste("(
1-pd)*(1-pj)*", prob[contador])}

```

```

        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="3"){prob[contador]=paste(prob[contador],"*(1-p5)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="4"){prob[contador]=paste(prob[contador],"*(1-p6)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="5"){prob[contador]=paste(prob[contador],"*(1-p7)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="6"){prob[contador]=paste(prob[contador],"*(1-p8)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="7"){prob[contador]=paste(prob[contador],"*(1-p9)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="8"){prob[contador]=paste(prob[contador],"*(1-pd)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="9"){prob[contador]=paste(prob[contador],"*(1-pj)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="D"){prob[contador]=paste(prob[contador],"*(1-pq)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="J"){prob[contador]=paste(prob[contador],"*(1-pk)",sep="")}
        if
        (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possivel])),nchar(todos_os_jogos1[jogo_possivel]))=="Q"){prob[contador]=paste(prob[contador],"*(1-pa)",sep="")}
    }
}

contador = 0
for (jogo in 1:length(todos_os_jogos1)){
  for (jogo_possivel in 1:length(todos_os_jogos1)){
    contador = contador + 1
    if

```

```

(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="K"){prob[contador]=paste(pro
b[contador],"*(1-pa)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="3"){prob[contador]=paste(pro
b[contador],"*(1-p4)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="4"){prob[contador]=paste(pro
b[contador],"*(1-p5)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="5"){prob[contador]=paste(pro
b[contador],"*(1-p6)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="6"){prob[contador]=paste(pro
b[contador],"*(1-p7)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="7"){prob[contador]=paste(pro
b[contador],"*(1-p8)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="8"){prob[contador]=paste(pro
b[contador],"*(1-p9)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="9"){prob[contador]=paste(pro
b[contador],"*(1-pd)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="D"){prob[contador]=paste(pro
b[contador],"*(1-pj)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="J"){prob[contador]=paste(pro
b[contador],"*(1-pq)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="Q"){prob[contador]=paste(pro
b[contador],"*(1-pk)",sep=""})
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])),nchar(todos_os_jogos1[jogo_possivel]))=="C"){
    if
(substr(todos_os_jogos1[jogo_possivel],1,1=="3"){prob[contador]=paste("(
1-pa)*",prob[contador])}
    if

```

```

(substr(todos_os_jogos1[jogo_possivel],1,1)=="4"){prob[contador]=paste("(
1-p2)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="5"){prob[contador]=paste("(
1-p3)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="6"){prob[contador]=paste("(
1-p4)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="7"){prob[contador]=paste("(
1-p5)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="8"){prob[contador]=paste("(
1-p6)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="9"){prob[contador]=paste("(
1-p7)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="D"){prob[contador]=paste("(
1-p8)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="J"){prob[contador]=paste("(
1-p9)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],1,1)=="Q"){prob[contador]=paste("(
1-pd)*",prob[contador])}
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])-1,nchar(todos_os_jogos1[jogo_possivel])-
1)=="3"){prob[contador]=paste(prob[contador],"*(1-p4)*(1-p5)",sep="")}
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])-1,nchar(todos_os_jogos1[jogo_possivel])-
1)=="4"){prob[contador]=paste(prob[contador],"*(1-p5)*(1-p6)",sep="")}
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])-1,nchar(todos_os_jogos1[jogo_possivel])-
1)=="5"){prob[contador]=paste(prob[contador],"*(1-p6)*(1-p7)",sep="")}
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])-1,nchar(todos_os_jogos1[jogo_possivel])-
1)=="6"){prob[contador]=paste(prob[contador],"*(1-p7)*(1-p8)",sep="")}
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])-1,nchar(todos_os_jogos1[jogo_possivel])-
1)=="7"){prob[contador]=paste(prob[contador],"*(1-p8)*(1-p9)",sep="")}
  if
(substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
l])-1,nchar(todos_os_jogos1[jogo_possivel])-
1)=="8"){prob[contador]=paste(prob[contador],"*(1-p9)*(1-pd)",sep="")}

```

```

    if
    (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
1])-1,nchar(todos_os_jogos1[jogo_possivel]) -
1)=="9"){prob[contador]=paste(prob[contador],"*(1-pd)*(1-pj)",sep="")}
    if
    (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
1])-1,nchar(todos_os_jogos1[jogo_possivel]) -
1)=="D"){prob[contador]=paste(prob[contador],"*(1-pj)*(1-pq)",sep="")}
    if
    (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
1])-1,nchar(todos_os_jogos1[jogo_possivel]) -
1)=="J"){prob[contador]=paste(prob[contador],"*(1-pq)*(1-pk)",sep="")}
    if
    (substr(todos_os_jogos1[jogo_possivel],nchar(todos_os_jogos1[jogo_possive
1])-1,nchar(todos_os_jogos1[jogo_possivel]) -
1)=="Q"){prob[contador]=paste(prob[contador],"*(1-pk)",sep="")}
  }
}

#jogos com e sem coringa
contador = 0
for (jogo in 1:length(todos_os_jogos1)){
  for (jogo_possivel in 1:length(todos_os_jogos1)){
    contador = contador + 1
    if
    (nchar(todos_os_jogos1[jogo_possivel])<14){if(grepl("C",todos_os_jogos1[j
ogo_possivel])==FALSE){prob[contador]=paste(prob[contador],"*(1-
pc)",sep="")}}
  }
}

prob_organizado = prob
for (i in 1:length(prob)){
  if (grepl("0",prob[i])){prob_organizado[i] = "0"}
}

matrix_prob = matrix(data = prob_organizado,nrow =
length(todos_os_jogos), byrow = T)

rownames(matrix_prob)=todos_os_jogos
colnames(matrix_prob)=todos_os_jogos

library(flextable)

## Warning: package 'flextable' was built under R version 4.2.3

```

```
flextable(data = as.data.frame(matrix_prob[1:20,1:20]))
```

	A23	A234	A234 5	A234 56	A234 567	A234 5678	A234 56789	A234 56789 D	A234 56789 DJ	A234 56789 DJQ	A234 56789 DJQK	A234 56789 DJQK A	234	2345	23456	23456 7	23456 78	23456 789	23456 789D	23456 789DJ
A23	$1*(1-p4)^*(1-pc)$	$1*p4*(1-p5)^*(1-pc)$	$1*p4*p5*(1-p6)^*(1-pc)$	$1*p4*p5*p6*(1-p7)^*(1-pc)$	$1*p4*p5*p6*p7*(1-p8)^*(1-pc)$	$1*p4*p5*p6*p7*p8*(1-p9)^*(1-pc)$	$1*p4*p5*p6*p7*p8*p9*(1-pd)^*(1-pc)$	$1*p4*p5*p6*p7*p8*p9*(1-pj)^*(1-pc)$	$1*p4*p5*p6*p7*p8*p9*(1-pq)^*(1-pc)$	$1*p4*p5*p6*p7*p8*p9*(1-pk)^*(1-pc)$	$1*p4*p5*p6*p7*p8*p9*(1-pa)^*(1-pc)$	$1*p4*p5*p6*p7*p8*p9*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234	0	$1*(1-p5)^*(1-pc)$	$1*p5*(1-p6)^*(1-pc)$	$1*p5*p6*(1-p7)^*(1-pc)$	$1*p5*p6*p7*(1-p8)^*(1-pc)$	$1*p5*p6*p7*p8*(1-p9)^*(1-pc)$	$1*p5*p6*p7*p8*p9*(1-pd)^*(1-pc)$	$1*p5*p6*p7*p8*p9*(1-pj)^*(1-pc)$	$1*p5*p6*p7*p8*p9*(1-pq)^*(1-pc)$	$1*p5*p6*p7*p8*p9*(1-pk)^*(1-pc)$	$1*p5*p6*p7*p8*p9*(1-pa)^*(1-pc)$	$1*p5*p6*p7*p8*p9*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234 5	0	0	$1*(1-p6)^*(1-pc)$	$1*p6*(1-p7)^*(1-pc)$	$1*p6*p7*(1-p8)^*(1-pc)$	$1*p6*p7*p8*(1-p9)^*(1-pc)$	$1*p6*p7*p8*p9*(1-pd)^*(1-pc)$	$1*p6*p7*p8*p9*(1-pj)^*(1-pc)$	$1*p6*p7*p8*p9*(1-pq)^*(1-pc)$	$1*p6*p7*p8*p9*(1-pk)^*(1-pc)$	$1*p6*p7*p8*p9*(1-pa)^*(1-pc)$	$1*p6*p7*p8*p9*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234 56	0	0	0	$1*(1-p7)^*(1-pc)$	$1*p7*(1-p8)^*(1-pc)$	$1*p7*p8*(1-p9)^*(1-pc)$	$1*p7*p8*p9*(1-pd)^*(1-pc)$	$1*p7*p8*p9*(1-pj)^*(1-pc)$	$1*p7*p8*p9*(1-pq)^*(1-pc)$	$1*p7*p8*p9*(1-pk)^*(1-pc)$	$1*p7*p8*p9*(1-pa)^*(1-pc)$	$1*p7*p8*p9*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234 567	0	0	0	0	$1*(1-p8)^*(1-pc)$	$1*p8*(1-p9)^*(1-pc)$	$1*p8*p9*(1-pd)^*(1-pc)$	$1*p8*p9*(1-pj)^*(1-pc)$	$1*p8*p9*(1-pq)^*(1-pc)$	$1*p8*p9*(1-pk)^*(1-pc)$	$1*p8*p9*(1-pa)^*(1-pc)$	$1*p8*p9*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234 5678	0	0	0	0	0	$1*(1-p9)^*(1-pc)$	$1*p9*(1-pd)^*(1-pc)$	$1*p9*(1-pj)^*(1-pc)$	$1*p9*(1-pq)^*(1-pc)$	$1*p9*(1-pk)^*(1-pc)$	$1*p9*(1-pa)^*(1-pc)$	$1*p9*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234 5678 9	0	0	0	0	0	0	$1*(1-pd)^*(1-pc)$	$1*pd*(1-pj)^*(1-pc)$	$1*pd*pj*(1-pq)^*(1-pc)$	$1*pd*pj*(1-pk)^*(1-pc)$	$1*pd*pj*(1-pa)^*(1-pc)$	$1*pd*pj*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0
A234 5678	0	0	0	0	0	0	0	$1*(1-pj)^*(1-pc)$	$1*pj*(1-pq)^*(1-pc)$	$1*pj*(1-pk)^*(1-pc)$	$1*pj*(1-pa)^*(1-pc)$	$1*pj*(1-pa)^*(1-pc)$	0	0	0	0	0	0	0	0

<b>9D</b>									pc)	pq)*(1- pc)	pk)*(1- pc)	1- pa)*(1- pc)								
<b>A234 5678 9DJ</b>	0	0	0	0	0	0	0	0	0	1*(1- pq)*(1- pc)	1*pq*( 1- pk)*(1- pc)	1*pq*p k*(1- pa)*(1- pc)	1*pq*p k	0	0	0	0	0	0	0
<b>A234 5678 9DJQ</b>	0	0	0	0	0	0	0	0	0	0	1*(1- pk)*(1- pc)	1*pk*( 1- pa)*(1- pc)	1*pk	0	0	0	0	0	0	0
<b>A234 5678 9DJQ K</b>	0	0	0	0	0	0	0	0	0	0	0	1*(1- pa)*(1- pc)	1	0	0	0	0	0	0	0
<b>A234 5678 9DJQ KA</b>	0	0	0	0	0	0	0	0	0	0	0	1*(1- pa)*(1- pc)	1	0	0	0	0	0	0	0
<b>234</b>	0	1*pa*p a*(1- p5)*(1- pc)	1*pa*p a*p5*( 1- p6)*(1- pc)	1*pa*p a*p5*p 6*(1- p7)*(1- pc)	1*pa*p a*p5*p 6*p7*( 1- p8)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*(1- p9)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*p9*( 1- pd)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*p9*p d*(1- pj)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*p9*p d*pj*(1- pq)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*p9*p d*pj*p q*(1- pk)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*p9*p d*pj*p q*pk*( 1- pa)*(1- pc)	1*pa*p a*p5*p 6*p7*p 8*p9*p d*pj*p q*pk*p a*pa	(1-pa)* 1*(1- p5)*(1- pc)	(1-pa)* 1*p5*( 1- p6)*(1- pc)	(1-pa)* 1*p5*p 6*(1- p7)*(1- pc)	(1-pa)* 1*p5*p 6*p7*( 1- p8)*(1- pc)	(1-pa)* 1*p5*p 6*p7*p 8*(1- p9)*(1- pc)	(1-pa)* 1*p5*p 6*p7*p 8*p9*( 1- pd)*(1- pc)	(1-pa)* 1*p5*p 6*p7*p 8*p9*p d*(1- pj)*(1- pc)	(1-pa)* 1*p5*p 6*p7*p 8*p9*p d*pj*(1- pq)*(1- pc)
<b>2345</b>	0	0	1*pa*p a*(1- p6)*(1- pc)	1*pa*p a*p6*( 1- p7)*(1- pc)	1*pa*p a*p6*p 7*(1- p8)*(1- pc)	1*pa*p a*p6*p 7*p8*( 1- p9)*(1- pc)	1*pa*p a*p6*p 7*p8*p 9*(1- pd)*(1- pc)	1*pa*p a*p6*p 7*p8*p 9*pd*( 1- pj)*(1- pc)	1*pa*p a*p6*p 7*p8*p 9*pd*p j*(1- pq)*(1- pc)	1*pa*p a*p6*p 7*p8*p 9*pd*p j*pq*(1- pk)*(1- pc)	1*pa*p a*p6*p 7*p8*p 9*pd*p j*pq*pk (1- pa)*(1- pc)	1*pa*p a*p6*p 7*p8*p 9*pd*p j*pq*pk *pa*pa	0	(1-pa)* 1*(1- p6)*(1- pc)	(1-pa)* 1*p6*( 1- p7)*(1- pc)	(1-pa)* 1*p6*p 7*(1- p8)*(1- pc)	(1-pa)* 1*p6*p 7*p8*( 1- p9)*(1- pc)	(1-pa)* 1*p6*p 7*p8*p 9*(1- pd)*(1- pc)	(1-pa)* 1*p6*p 7*p8*p 9*pd*( 1- pj)*(1- pc)	(1-pa)* 1*p6*p 7*p8*p 9*pd*p j*(1- pq)*(1- pc)
<b>2345 6</b>	0	0	0	1*pa*p a*(1- p7)*(1- pc)	1*pa*p a*p7*( 1- p8)*(1- pc)	1*pa*p a*p7*p 8*(1- p9)*(1- pc)	1*pa*p a*p7*p 8*p9*( 1- pd)*(1- pc)	1*pa*p a*p7*p 8*p9*p d*(1- pj)*(1- pc)	1*pa*p a*p7*p 8*p9*p d*pj*(1- pq)*(1- pc)	1*pa*p a*p7*p 8*p9*p d*pj*p q*(1- pk)*(1- pc)	1*pa*p a*p7*p 8*p9*p d*pj*p q*pk*( 1- pa)*(1- pc)	1*pa*p a*p7*p 8*p9*p d*pj*p q*pk*p a*pa	0	0	(1-pa)* 1*(1- p7)*(1- pc)	(1-pa)* 1*p7*( 1- p8)*(1- pc)	(1-pa)* 1*p7*p 8*(1- p9)*(1- pc)	(1-pa)* 1*p7*p 8*p9*( 1- pd)*(1- pc)	(1-pa)* 1*p7*p 8*p9*p d*(1- pj)*(1- pc)	(1-pa)* 1*p7*p 8*p9*p d*pj*(1- pq)*(1- pc)
<b>2345 67</b>	0	0	0	0	1*pa*p a*(1- p8)*(1- pc)	1*pa*p a*p8*( 1- p9)*(1- pc)	1*pa*p a*p8*p 9*(1- pd)*(1- pc)	1*pa*p a*p8*p 9*pd*( 1- pj)*(1- pc)	1*pa*p a*p8*p 9*pd*p j*(1- pq)*(1- pc)	1*pa*p a*p8*p 9*pd*p j*pq*(1- pk)*(1- pc)	1*pa*p a*p8*p 9*pd*p j*pq*pk (1- pa)*(1- pc)	1*pa*p a*p8*p 9*pd*p j*pq*pk *pa*pa	0	0	0	(1-pa)* 1*(1- p8)*(1- pc)	(1-pa)* 1*p8*( 1- p9)*(1- pc)	(1-pa)* 1*p8*p 9*(1- pd)*(1- pc)	(1-pa)* 1*p8*p 9*pd*( 1- pj)*(1- pc)	(1-pa)* 1*p8*p 9*pd*p j*(1- pq)*(1- pc)
<b>2345 678</b>	0	0	0	0	0	1*pa*p a*(1- p9)*(1- pc)	1*pa*p a*p9*( 1- pd)*(1- pc)	1*pa*p a*p9*p d*(1- pj)*(1- pc)	1*pa*p a*p9*p d*pj*(1- pq)*(1- pc)	1*pa*p a*p9*p d*pj*p q*(1- pk)*(1- pc)	1*pa*p a*p9*p d*pj*p q*pk*( 1- pa)*(1- pc)	1*pa*p a*p9*p d*pj*p q*pk*p a*pa	0	0	0	0	(1-pa)* 1*(1- p9)*(1- pc)	(1-pa)* 1*p9*( 1- pd)*(1- pc)	(1-pa)* 1*p9*p d*(1- pj)*(1- pc)	(1-pa)* 1*p9*p d*pj*(1- pq)*(1- pc)

						pc)	pd)*(1- pc)	pj)*(1- pc)	- pq)*(1- pc)	q*(1- pk)*(1- pc)	q*pk*( 1- pa)*(1- pc)	q*pk*p a*pa					pc)	pd)*(1- pc)	pj)*(1- pc)	- pq)*(1- pc)
<b>2345 6789</b>	0	0	0	0	0	0	1*pa*pa*(1- pd)*(1- pc)	1*pa*pa*pd*(1- pj)*(1- pc)	1*pa*pa*pd*pj*(1- pq)*(1- pc)	1*pa*pa*pd*pj* pq*(1- pk)*(1- pc)	1*pa*pa*pd*pj* pq*pk*(1- pa)*(1- pc)	1*pa*pa*pd*pj* pq*pk* pa*pa	0	0	0	0	0	(1-pa)* 1*(1- pd)*(1- pc)	(1-pa)* 1*pd*(1- pj)*(1- pc)	(1-pa)* 1*pd*pj*(1- pq)*(1- pc)
<b>2345 6789 D</b>	0	0	0	0	0	0	0	1*pa*pa*a*(1- pj)*(1- pc)	1*pa*pa*a*pj*(1- - pq)*(1- pc)	1*pa*pa*a*pj*pq*(1- pk)*(1- pc)	1*pa*pa*a*pj*pq* pk*(1- pa)*(1- pc)	1*pa*pa*a*pj*pq* pk*pa* pa	0	0	0	0	0	0	(1-pa)* 1*(1- pj)*(1- pc)	(1-pa)* 1*pj*(1- - pq)*(1- pc)
<b>2345 6789 DJ</b>	0	0	0	0	0	0	0	0	1*pa*pa*a*(1- pq)*(1- pc)	1*pa*pa*a*pq*(1- pk)*(1- pc)	1*pa*pa*a*pq*p k*(1- pa)*(1- pc)	1*pa*pa*a*pq*p k*pa*p a	0	0	0	0	0	0	0	(1-pa)* 1*(1- pq)*(1- pc)

Com a matriz de transição, utilizamos as equações de Chapman Komogorov, definida no livro "Introduction to Probability Models, Tenth Edition" de Sheldon M. Ross, como:

*Definimos as probabilidades de transição de n passos  $P_{ij}^{(n)}$  como a probabilidade de que um processo no estado i estará no estado j após n transições adicionais. Ou seja,*

$$P_{ij}^{(n)} = P\{X_{n+k} = j | X_k = i\}, \quad n \geq 0, \quad i, j \geq 0$$

*Claro que  $P_{ij}^{(1)} = P_{ij}$ . As equações de Chapman-Kolmogorov fornecem um método para calcular essas probabilidades de transição de n passos. Essas equações são*

$$P_{ij}^{(n+m)} = \sum_{k=0}^{\infty} P_{ik}^{(n)} P_{kj}^{(m)}, \quad \text{para todo } n, m \geq 0, \text{ todos } i, j$$

*E são mais facilmente compreendidas observando que  $P_{ik}^{(n)} P_{kj}^{(m)}$  representa a probabilidade de que, começando em i, o processo irá para o estado j em n + m transições através de um caminho que o leva ao estado k na n-ésima transição. Portanto, somando sobre todos os estados intermediários k, obtemos a probabilidade de que o processo estará no estado j após n + m transições. Formalmente, temos*

$$\begin{aligned} P_{ij}^{(n+m)} &= P\{X_{n+m} = j | X_0 = i\} = \sum_{k=0}^{\infty} P\{X_{n+m} = j, X_n = k | X_0 = i\} \\ &= \sum_{k=0}^{\infty} P\{X_{n+m} = j | X_n = k, X_0 = i\} P\{X_n = k | X_0 = i\} = \sum_{k=0}^{\infty} P_{kj}^{(m)} P_{ik}^{(n)} \end{aligned}$$

*Se denotarmos por  $P^{(n)}$  a matriz de probabilidades de transição de n passos, então a Equação (4.2) afirma que*

$$P^{(n+m)} = P^{(n)} \cdot P^{(m)}$$

No caso em estudo, o número de passos é igual ao número esperado de rodadas, até o fim da partida t, então basta multiplicar nossa matriz de transição por ele mesma t vezes e calcular o valor esperado de pontos de acordo com o estado atual.

## Probabilidade da batida

Para calcular a probabilidade de batida basta calcularmos a probabilidade de que dentro de cada um de nossos jogos, haja apenas sequências válidas de cartas.

Vamos pegar uma organização de jogos qualquer e uma mão qualquer como exemplo:

```

organizaç o_exemplo = list(c("A", "2", "3", "4", "5", "6", "7"),
c("8", "9", "D", "J", "Q", "K", "A*"),
c("A**", "2*", "3*", "4*", "5*", "6*", "7*"), c("8*", "9*", "D*", "J*", "Q*", "K*", "A
**"))

m o_exemplo = c("3", "4", "5", "J", "K")

m o_separada_jogos = lapply(organizaç o_exemplo, function(vetor)
intersect(vetor, m o_exemplo))

m o_separada_jogos

## [[1]]
## [1] "3" "4" "5"
##
## [[2]]
## [1] "J" "K"
##
## [[3]]
## character(0)
##
## [[4]]
## character(0)

todas_sequ ncias = todos_os_jogos[order(nchar(todos_os_jogos))]
cartas_batida = c()
for(jogo in 1:length(m o_separada_jogos)){
  sequ ncia = paste(m o_separada_jogos[[jogo]], collapse = "")
  if(nchar(sequ ncia)>1){
    for (sequ ncia_possivel in todas_sequ ncias){
      if(all(sapply(strsplit(sequ ncia, NULL)[[1]], function(x)
any(grepl(x, sequ ncia_possivel))))){
        cartas_batida <-c(cartas_batida,
setdiff(strsplit(sequ ncia_possivel, NULL)[[1]], strsplit(sequ ncia,
NULL)[[1]]))
        break
      }
    }
  }
}

cartas_batida = paste(cartas_batida, collapse = "")
cartas_batida

## [1] "Q"

prob_batida = c()
prob2 = 1

for (carta in 1:nchar(cartas_batida)){
  if (substr(cartas_batida, carta, carta)=="A"){prob2 =

```

```

paste(prob2, "*pa", sep = "")}
  if (substr(cartas_batida, carta, carta) == "2"){prob2 =
paste(prob2, "*p2", sep = "")}
  if (substr(cartas_batida, carta, carta) == "3"){prob2 =
paste(prob2, "*p3", sep = "")}
  if (substr(cartas_batida, carta, carta) == "4"){prob2 =
paste(prob2, "*p4", sep = "")}
  if (substr(cartas_batida, carta, carta) == "5"){prob2 =
paste(prob2, "*p5", sep = "")}
  if (substr(cartas_batida, carta, carta) == "6"){prob2 =
paste(prob2, "*p6", sep = "")}
  if (substr(cartas_batida, carta, carta) == "7"){prob2 =
paste(prob2, "*p7", sep = "")}
  if (substr(cartas_batida, carta, carta) == "8"){prob2 =
paste(prob2, "*p8", sep = "")}
  if (substr(cartas_batida, carta, carta) == "9"){prob2 =
paste(prob2, "*p9", sep = "")}
  if (substr(cartas_batida, carta, carta) == "D"){prob2 =
paste(prob2, "*pd", sep = "")}
  if (substr(cartas_batida, carta, carta) == "J"){prob2 =
paste(prob2, "*pj", sep = "")}
  if (substr(cartas_batida, carta, carta) == "Q"){prob2 =
paste(prob2, "*pq", sep = "")}
  if (substr(cartas_batida, carta, carta) == "K"){prob2 =
paste(prob2, "*pk", sep = "")}
  if (substr(cartas_batida, carta, carta) == "A"){prob2 =
paste(prob2, "*pa", sep = "")}
  if (substr(cartas_batida, carta, carta) == "C"){prob2 =
paste(prob2, "*pc", sep = "")}
  prob_batida = c(prob_batida, prob2)
}
prob_batida
## [1] "1*pq"

```

No exemplo acima a probabilidade de batida é dada por  $p_q$ , ou seja, a probabilidade de comprar a rainha.

## Probabilidade de ter as cartas

Probabilidade de ter determinada carta em uma rodada é dada pela probabilidade de comprar carta no monte, pegar a carta no lixo ou seu parceiro ter a carta em sua mão.

A função de distribuição de probabilidade de a comprar a carta no monte  $P_N = N/b$  é dada por:

$$f(P) = \begin{cases} \left(1 - (b/\hat{b})\right)^2, & \text{de } P = 0 \\ 2 \times (b/\hat{b}) \times \left(1 - (b/\hat{b})\right), & \text{de } P = 1/b \\ (b/\hat{b})^2, & \text{de } P = 2/b \\ 0, & \text{para outros valores de } P \end{cases}$$

se o jogador não sabe se alguma cópia da carta está fora do monte.

$$f(P) = \begin{cases} \left(1 - (b/\hat{b})\right), & \text{de } P = 0 \\ (b/\hat{b}), & \text{de } P = 1/b \\ 0, & \text{para outros valores de } P \end{cases}$$

se o jogador sabe que uma cópia da carta está fora do monte.

$$f(P) = \begin{cases} 1, & \text{de } P = 0 \\ 0, & \text{para outros valores de } x \end{cases}$$

se o jogador sabe que as duas cópias da carta estão fora do monte.

De forma similar ao cálculo da probabilidade de uma determinada carta estar no monte, podemos fazer o cálculo da probabilidade de uma carta estar na mão do nosso parceiro. Se o jogador não sabe onde nenhuma das cópias da carta estão, então a probabilidade da cópia 1 estar na mão de outro jogador é igual  $c/\hat{b}$ , onde  $c$  é a quantidade total de cartas na mão do jogador e  $\hat{b}$  é o total de cartas desconhecidas pelo jogador, ou seja, as cartas do monte, as cartas dos mortos e as cartas das mãos dos outros jogadores que ainda não foram reveladas. Por consequência a probabilidade da cópia 1 não estar no monte é  $1 - (c/\hat{b})^2$ . As probabilidades para a cópia 2 são exatamente as mesmas.

Sendo assim, a função de distribuição de probabilidade de determinada carta estar na mão de um jogador  $M$  é dada por:

$$f(M) = \begin{cases} \left(1 - (c/\hat{b})\right)^2, & \text{de } M = 0 \\ 2 \times (c/\hat{b}) \times \left(1 - (c/\hat{b})\right), & \text{de } M = 1 \\ (c/\hat{b})^2, & \text{de } M = 2 \\ 0, & \text{para outros valores de } M \end{cases}$$

Se o jogador sabe que uma das cartas não está na mão desse jogador, então a probabilidade da cópia 1 estar na mão de um jogador é igual  $c/\hat{b}$ , onde  $c$  é a quantidade total de cartas na mão desse jogador e  $\hat{b}$  é o total de cartas desconhecidas

pelo jogador. Por consequência a probabilidade da cópia 1 não estar no monte é  $1 - (c/\hat{b})$ . Porém a probabilidade da cópia 2 estar na mão do jogador é 0, pois é sabido que não está lá.

Assim, a função de distribuição de probabilidade de  $M$  nesse caso é dada por:

$$f(M) = \begin{cases} (1 - (c/\hat{b})), & \text{de } M = 0 \\ (c/\hat{b}), & \text{de } M = 1 \\ 0, & \text{para outros valores de } M \end{cases}$$

Por fim, se o jogador sabe que ambas as cópias da carta estão fora da mão do jogador é trivial que não há nenhuma cópia da carta no monte.

Se temos a probabilidade de uma determinada carta estar na mão de um jogador, podemos calcular a probabilidade um jogador descartar uma carta, listamos todas as mãos possíveis e calculamos em quantas dessas mãos a melhor jogada envolve aquele descarte.

Com as probabilidades de comprar a carta no monte ( $P$ ), a probabilidade de a carta estar na mão do seu parceiro ( $M$ ) e a probabilidade de algum dos jogadores adversários descartar a carta ( $D$ ), temos a probabilidade de se ter a carta até o final da rodada como o complemento de nenhuma dessas probabilidades acontecer:

$$1 - ((1 - P)X(1 - M)X(1 - D))$$

Podemos ainda deixar o cálculo mais preciso, uma vez que no decorrer da partida as jogadas e descartes dos outros jogadores nos permitem calcular a probabilidade de que eles tenham determinadas cartas na mão. Para fazer isso, listamos todas as mãos possíveis e calculamos em quantas dessas mãos a melhor jogada envolve aquela compra ou aquele descarte. Por fim, calculamos o percentual dessas mãos, em que aquela era a melhor jogada, que tem determinada carta. Esse percentual nos dá a probabilidade  $theta_x$  de uma carta estar na mão de um jogador, portanto se fazemos isso para todos os jogadores temos a probabilidade de determinada carta estar na mão de algum dos jogadores ( $theta_{total}$ ) e o complemento disso ( $1 - theta_{total}$ ) é a probabilidade de a carta estar no monte ou mortos.

## Tempo até o final do jogo

O tempo esperado até o final da partida, é dado pelo menor valor entre o tempo esperado para dois jogadores da mesma dupla baterem e o tempo esperado para o fim das cartas do monte. Ambas as variáveis dependem de escolhas de todos os jogadores e informações sobre as mãos desses jogadores, tornando o cálculo muito complexo.

Tendo isso em vista, foram jogadas 30 partidas e o número de rodadas foi contabilizado em cada uma delas:

```
n_rodadas =
c(12, 12, 15, 11, 13, 12, 10, 16, 18, 11, 12, 11, 11, 8, 16, 18, 13, 9, 15, 14, 14, 12, 12, 11, 1
```

```
3,12,16,18,18,10)
```

```
mean(n_rodadas)
```

```
## [1] 13.1
```

```
sd(n_rodadas)
```

```
## [1] 2.771157
```

Adotamos a média do número de rodadas na amostra de 30 partidas jogadas, como o número de rodadas esperados no início da partida e a cada rodada que se passa reduzimos em um o valor esperado.

Sendo assim:

$t_n = 13 - n$ , onde  $t$  é o tempo esperado até o final da partida em rodadas e  $n$  é o número de rodadas decorridas.

Apesar de não ser uma medida exata do tempo esperado até o final da partida e não levar em consideração fatores importantes que variam a cada partida, é uma aproximação satisfatória, tendo em vista a pequena a relativa pena variância do número de rodadas a cada partida.

## Implementando o algoritmo no R

### O que pode ser aproveitado na prática

Na teoria temos um algoritmo que determina a melhor jogada a cada rodada, no entanto precisamos analisar a viabilidade de implementá-lo exatamente da forma proposta anteriormente no trabalho. Para isso vamos supor uma partida hipotética de buraco em que as jogadas de um dos jogadores são determinadas pelo algoritmo. Onze cartas são distribuídas para cada um dos jogadores, os mortos e o monte são separados, o primeiro jogador compra uma carta do monte e descarta uma outra carta qualquer. Então é a hora do jogador, cujas jogadas são determinadas pelo algoritmo, jogar.

Ele, primeiramente, precisa calcular a probabilidade das cartas na mão do primeiro jogador, listando cada mão possível e calculando em quais dessas mãos o jogador descartaria a carta descartada. Em seguida calcular a quantidade esperada de pontos de um jogo, para cada jogo da mão, para cada forma de organizar a mão em jogos, para cada carta diferente que ele pode comprar.

Utilizando um método de contagem simples, encontramos o número de combinações possíveis para a mão do primeiro jogador: Como é a primeira rodada da partida há 93 cartas desconhecidas, uma vez que 11 das 104 são conhecidas pois estão na nossa mão, e 12 dessas 93 cartas desconhecidas estão na mão do jogador, já que ele

começou o jogo com 11 e comprou 1 carta. Então, basta utilizar a fórmula de combinação:

$$C(n, k) = n! / k! (n - k)!$$

Onde, n é o número total de elementos e k é o número de elementos escolhidos

$$C(93,12) = 93! / 12! (93 - 12)!$$

```
factorial(93)/(factorial(12)*factorial(93-12))
```

```
## [1] 4.165798e+14
```

Fazendo o cálculo encontramos que o número total de combinações possíveis para a mão do jogador é 416.579.843.773.626. Esse número contém algumas mãos iguais, uma vez que consideramos cada carta como única, mesmo havendo duas cópias de várias cartas, no entanto fazer o cálculo dessa facilita cálculos futuros uma vez que todas as mãos têm a mesma probabilidade e não precisamos considerar a probabilidade de cada mão existir em primeiro lugar.

De qualquer forma, temos 416.579.843.773.626 mãos possíveis e cada uma dessas mãos podem ser organizadas em jogos de formas diferentes.

```
permutações_possiveis =list(c(3,11), c(4,10), c(5,9), c(6,8), c(7,7),  
c(8,6), c(9,5), c(10,4), c(11,3), c(5,5,4), c(5,4,5), c(4,5,5), c(4,4,6),  
c(4,6,4), c(6,4,4), c(3,3,8), c(3,8,3), c(8, 3, 3), c(3,4,7), c(3,7,4),  
c(4,3,7), c(4,7,3), c(7,3,4), c(7,4,3), c(3,5,6), c(3,6,5),c(5,3,6),  
c(5,6,3), c(6,3,5), c(6,5,3), c(3,3,3,5), c(3,3,5,3), c(3,5,3,3),  
c(5,3,3,3), c(3,3,4,4), c(3,4,3,4), c(3,4,4,3), c(4,3,3,4), c(4,3,4,3),  
c(4,4,3,3))
```

```
length(permutações_possiveis)
```

```
## [1] 40
```

Temos 40 permutações possíveis de números que nos dão as diferentes formas de organizar as cartas de um naipe de um baralho em jogos.

Cada uma das 13 cartas pode ser de dois baralhos diferentes, então temos  $2^{13}$  formas de organizar as cartas de um naipe de uma metade das cartas do jogo considerando os dois baralhos.

Para cada uma dessas formas de organizar metade das cartas de um naipe há 40 formas de organizar a outra metade da mão com as cartas restantes.

```
n_jogos = c()  
for (i in permutações_possiveis){  
  n_jogos=c(n_jogos,length(i))  
}
```

```
2*mean(n_jogos)
```

```
## [1] 6.05
```

Colocando os coringas de todas as formas possíveis nesses jogos, temos:  $C(24,8) = 24!/8!(24 - 8)!$

```
factorial(24)/(factorial(8)*factorial(24-8))
```

```
## [1] 735471
```

E por fim, para todas essas formas de organizar a mão há em média 6,05 jogos e 12 descartes. Além disso, temos que levar em consideração que é preciso fazer isso para cada um dos 4 naipes e que o cálculo do valor esperado de pontos de um jogo demora em média aproximadamente 9 segundos. Então temos:

$416.579.843.773.626 \times 40 \times 2^{13} \times 40 \times 12 \times 6,05 \times 4 \times 735.471 \times 9 \text{ segundos}$  para fazer o cálculo da probabilidade da mão do primeiro jogador

$92 \times 40 \times 2^{13} \times 40 \times 12 \times 6,05 \times 4 \times 735.471 \times 9 \text{ segundos}$  para fazer o cálculo da melhor jogada para nós

Dessa forma, o tempo necessário para fazer todos os passos da jogada é dado, em anos, por:

```
((416579843773626*40*2^13*40*12*6.05*4*735471*9)+(92*40*2^{13}*40*12*6.05*4*735471*9))/(60*24*365)
```

```
## [1] 1.996905e+25
```

Ou seja, para calcular a melhor jogada de acordo com o algoritmo proposto demoraria  $1.996905 \times 10^{25}$  anos.

Tendo em vista o tempo, de escala astronômica, que levaria para jogar uma partida utilizando o algoritmo para calcular as jogadas de um dos jogadores, fica claro que não é viável utilizá-lo dessa forma. Sendo assim, há duas possibilidades razoáveis para traduzir o estudo teórico em uma aplicação prática: simplificar o algoritmo para que seja possível fazer os cálculos em um tempo viável ou utilizar alguns cálculos feitos no estudo para auxiliar um jogador humano ao invés de substituí-lo.

Convenientemente, as habilidades dos seres humanos e as capacidades do computador se complementam muito bem em um jogo de buraco. Como vimos anteriormente o algoritmo é muito ineficiente em filtrar boas jogadas, tendo que passar por muitas possibilidades o que demanda uma quantidade absurda de tempo para ser feito, enquanto um ser humano filtra de forma muito eficiente essas jogadas. Em contrapartida o ser humano é ineficiente ou incapaz de fazer alguns cálculos de probabilidade de forma rápida suficiente, enquanto o computador, depois de programado para isso, faz esses cálculos em segundos.

## Probabilidade de comprar uma carta do monte

Primeiramente definimos uma variável que contenha um vetor com todas as cartas do baralho:

```
baralho =  
c("AP", "2P", "3P", "4P", "5P", "6P", "7P", "8P", "9P", "DP", "JP", "QP", "KP",  
"AC", "2C", "3C", "4C", "5C", "6C", "7C", "8C", "9C", "DC", "JC", "QC", "KC",  
"AE", "2E", "3E", "4E", "5E", "6E", "7E", "8E", "9E", "DE", "JE", "QE", "KE",  
"AO", "2O", "3O", "4O", "5O", "6O", "7O", "8O", "9O", "DO", "JO", "QO", "KO",  
"AP", "2P", "3P", "4P", "5P", "6P", "7P", "8P", "9P", "DP", "JP", "QP", "KP",  
"AC", "2C", "3C", "4C", "5C", "6C", "7C", "8C", "9C", "DC", "JC", "QC", "KC",  
"AE", "2E", "3E", "4E", "5E", "6E", "7E", "8E", "9E", "DE", "JE", "QE", "KE",  
"AO", "2O", "3O", "4O", "5O", "6O", "7O", "8O", "9O", "DO", "JO", "QO", "KO")
```

O primeiro caracter de cada elemento do vetor indica o número da carta e o segundo caracter indica o naipe da carta.

Então criamos uma nova variável que terá todas as cartas desconhecidas pelo jogador. Antes do jogo começar o jogador não sabe do paradeiro de nenhuma carta, portanto a variável contém todas as cartas do baralho. À medida que o jogo decorre o jogador passa a conhecer as cartas que estão na sua mão, na mesa, no lixo, ou passaram pelo lixo em algum momento e agora estão na mão de outros jogadores, portanto utilizamos uma função para retirá-las da variável criada.

```
baralho_desconhecido = baralho  
  
remover_cartas = function (cartas){  
  for (carta in cartas){  
    baralho_desconhecido <- baralho_desconhecido[-  
match(carta,baralho_desconhecido)]  
  }  
  return (baralho_desconhecido)  
}
```

Como já estabelecido previamente, a probabilidade de comprar determinada carta no monte ( $P$ ) é dada por  $P_N = \hat{N}/\hat{b}$ , onde  $\hat{N}$  é a quantidade de cópias desconhecidas pelo jogador de determinada carta e  $\hat{b}$  é a quantidade de cartas desconhecidas pelo jogador. Sendo assim, podemos utilizar a variável criada para calcular essa probabilidade no R.

```
probabilidade_de_comprar_carta_monte = function (carta){  
  p =
```

```

as.numeric(table(baralho_desconhecido)[carta])/length(baralho_desconhecido)
  return (p)
}

```

De forma similar, a probabilidade de uma carta estar na mão do nosso parceiro é  $c/\hat{b}$ , onde  $c$  é a quantidade total de cartas na mão do parceiro e  $\hat{b}$  é o total de cartas desconhecidas pelo jogador. Portanto:

```

probabilidade_de_comprar_carta_monte = function (carta,tamanho_mao){
  p = as.numeric(table(baralho_desconhecido)[carta])/tamanho_mao
  return (p)
}

```

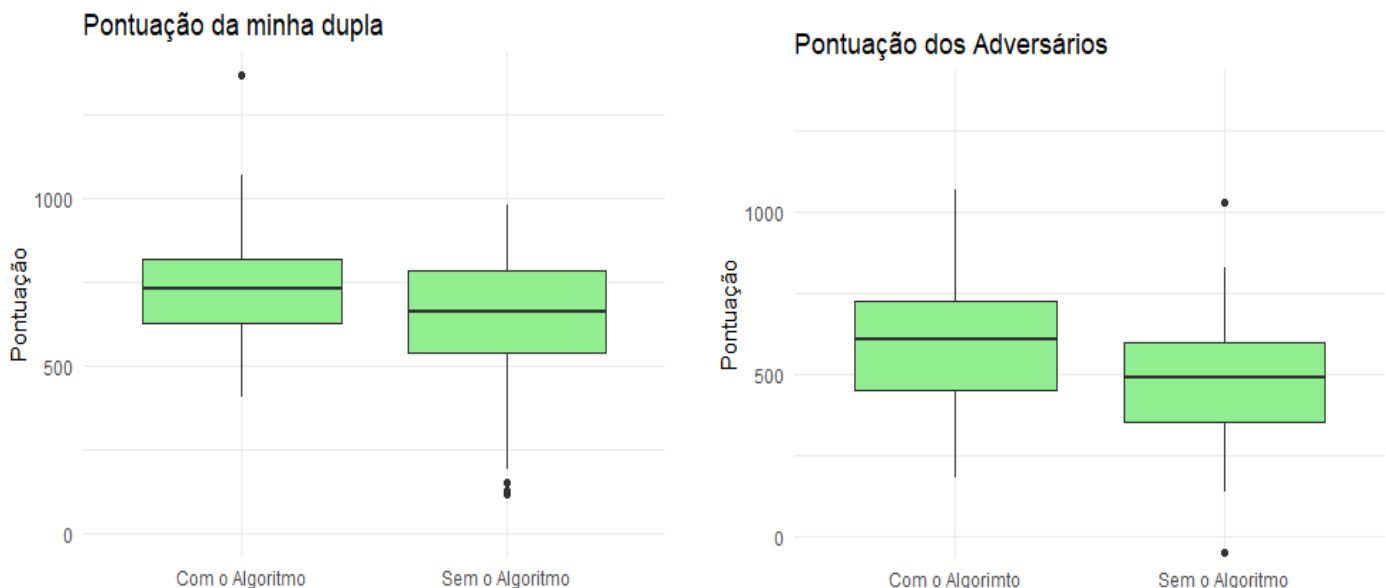
Por fim, para ser possível utilizar na prática, a probabilidade de um adversário descartar determinada carta será calculada como se os demais jogadores descartassem as cartas de maneira aleatória e uniforme. Sendo assim, a probabilidade do descarte de determinada carta é dada por  $c/\hat{b} \times 1/c$  onde  $c$  é a quantidade total de cartas na mão do adversário e  $\hat{b}$  é o total de cartas desconhecidas pelo jogador.

Logo, a probabilidade de se ter determinada carta nessa rodada é  $1 - ((1 - P) \times (1 - M) \times (1 - D))$ .

Com a probabilidade de se ter determinada carta até a próxima rodada, podemos calcular a probabilidade de batida e o valor esperado de pontos no final da partida utilizando os métodos estabelecidos anteriormente, desde que tenhamos uma organização da mão em jogos estabelecidas. Como vimos é muito demorado fazer isso computacionalmente então o próprio jogador estabelece a organização da mão e utiliza os métodos já discutidos para cálculos a probabilidade de batida e o valor esperado de pontos, o que pode ser útil para comparar jogadas, organizações ou descartes possíveis, tal como estimar a quantidade de pontos ao final da partida.

## Resultados

Utilizando as informações de probabilidade de se comprar uma carta e de se ter uma



carta até o final da rodada, de probabilidade de batida e de valor esperado de pontos de determinado jogo, foram jogadas 30 partidas para colher alguns dados sobre a vantagem de se ter essas informações ao longo do jogo.

Analisando os gráficos é possível verificar que a pontuação utilizando o algoritmo para auxiliar no cálculo de algumas probabilidades foi em média maior e com menor variância que a pontuação sem a utilização do algoritmo. No entanto a pontuação dos adversários também foi em média maior quando eu estava utilizando o algoritmo em comparação a quando eu não estava utilizando o algoritmo. Muito provavelmente porque sabendo a probabilidade de batida eu acabo tomando menos riscos durante a partida o que faz com que ela em média dure mais rodadas e, conseqüentemente, as duplas façam mais pontos.

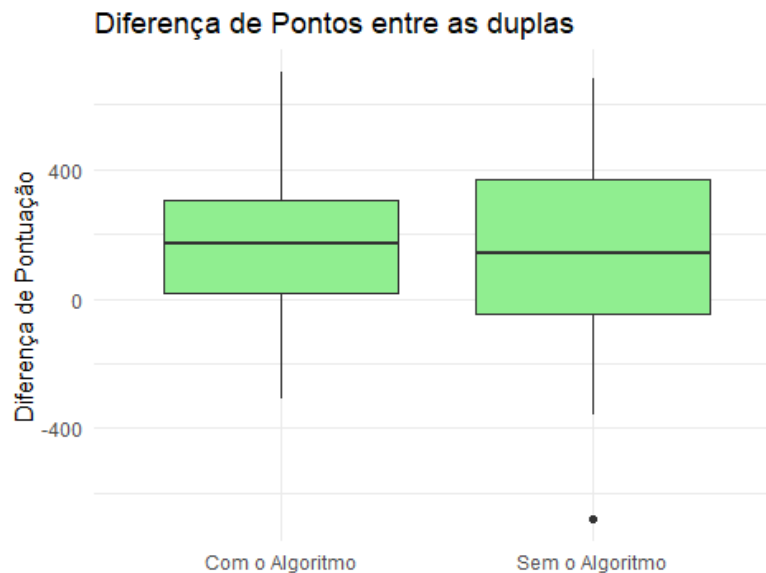
```
wilcox.test(pontos,pontos_algoritmo)

## Warning in wilcox.test.default(pontos, pontos_algoritmo): não é
## possível
## computar o valor de p exato com o de desempate

##
## Wilcoxon rank sum test with continuity correction
##
## data: pontos and pontos_algoritmo
## W = 339.5, p-value = 0.1037
## alternative hypothesis: true location shift is not equal to 0
```

No entanto o teste de wilcox revela que não há evidências suficientes para dizer que as duas amostras de jogos são estatisticamente diferentes.

Quando comparamos a diferença de pontuação entre a minha dupla e a dupla adversária, que nos dá uma melhor ideia sobre o resultado da partida, podemos observar que utilizando o algoritmo em média existe uma diferença positiva maior entre os pontos da minha dupla e os pontos da dupla adversária, e mais importante mais consistentemente essa diferença fica positiva, ou seja, mais consistentemente eu ganho as partidas.



```
wilcox.test(diferença_pontos,diferença_pontos_algoritmo)
```

```
## Warning in wilcox.test.default(diferença_pontos,  
diferença_pontos_algoritmo):  
## não é possível computar o valor de p exato com o de desempate  
  
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: diferença_pontos and diferença_pontos_algoritmo  
## W = 457, p-value = 0.9234  
## alternative hypothesis: true location shift is not equal to 0
```

Novamente o teste de wilcox revela que não há evidências suficientes para dizer que as duas amostras de jogos são estatisticamente diferentes.

## Conclusão

O desenvolvimento do algoritmo matemático para o jogo de cartas “Buraco” representou um avanço significativo em direção à automação e otimização das tomadas de decisão. Ao atingir os objetivos propostos de construir o algoritmo e implementá-lo em uma linguagem de programação para testes, observamos resultados promissores na melhoria da performance estratégica do jogador.

Em uma análise teórica mais profunda, é evidente que alguns pontos podem ser aprimorados a fim de deixar o processo do cálculo menos computacionalmente demandante, a previsão das mãos dos outros jogadores, a organização das mãos, e o próprio cálculo do valor esperado de pontos, poderiam ser refinados. Talvez com as provas necessárias não seja necessário considerar tantas organizações de mãos diferentes quanto estão sendo considerados, por exemplo. Além disso, o tempo esperado para o fim do jogo é uma variável muito importante no algoritmo e uma abordagem estatística para a estimativa dela é muito suscetível a variações aleatórias dentro de uma partida, inerentes da própria natureza do jogo.

Quanto à utilidade prática do projeto, fica claro que essa não passa por uma aplicação direta do algoritmo em uma partida real de buraco, tendo em vista a grandeza impraticável do tempo necessário para se considerar de fato todas as possibilidades dentro do jogo. No entanto, algumas partes do algoritmo funcionam muito bem de forma isolada quando associadas com a habilidade de um jogador humano de filtrar melhor as jogadas pertinentes. A capacidade de se calcular de forma rápida o valor esperado de pontos de uma determinada jogada, por exemplo, pode ser muito útil ao se comparar duas jogadas que parecem plausíveis. Ainda, sempre saber a probabilidade de comprar cada uma das cartas no monte ou a probabilidade de batida caso se compre uma carta do monte, auxilia muito na hora de tomar a decisão de comprar do monte ou do lixo.

Sendo assim, cálculos feitos e métodos utilizados nesse trabalho podem ser utilizados para fazer uma ferramenta excelente de auxílio ao jogador de buraco. Seria possível, por exemplo, fazer um programa que gera um “overlay” (uma camada adicional de informações que é sobreposta à tela principal do jogo) que mostra ao jogador essas informações, de forma a auxiliar na tomada de decisão ao longo da partida.

Em suma, o presente trabalho estabeleceu uma base sólida para aprimoramentos futuros. O equilíbrio entre a complexidade teórica e a eficiência prática é crucial para a viabilidade contínua e a aplicação realista do algoritmo no contexto dinâmico do jogo de “Buraco”. O compromisso contínuo com a pesquisa e desenvolvimento permitirá que o algoritmo evolua, proporcionando uma vantagem estratégica ainda maior no cenário competitivo do jogo.