Jogo de NIM utilizando Monte Carlo e Machine Learning

Simulação e Métodos de Monte Carlo

José Rodrigues, nº 2019246536

Índice

Introdução

O jogo de nim (conhecido em Portugal como jogo dos palitos ou jogo dos fósforos) consiste numa torre, normalmente composta por 16 de peças (cf. fig.1), em que cada um dos dois jogadores retira peças à vez até não sobrarem peças. O último jogador a retirar peças é o vencedor.

As regras deste jogo são, nomeadamente, cada jogador, no seu turno, tem que retirar pelo menos uma peça, e pode retirar tantas peças quantas quiser desde que pertençam à mesma linha.

Outras variantes incluem torres com mais peças (as linhas são sempre compostas por um número peças pertencentes à sequência de ímpares), o vencedor ser o que faz a penúltima jogada (e não a última) e apenas ser permitido retirar peças da mesma linha que estejam justapostas. Contudo neste trabalho é estudado o caso descrito inicialmente (torre de 16 peças; o vencedor é último a retirar peças; podem-se retirar quaisquer peças da mesma linha).

Figura 1- Torre do jogo de nim de 16 peças.

De modo a facilitar a visualização do jogo, na figura 2 está representado um exemplo de situações que podem ocorrer num jogo de nim.

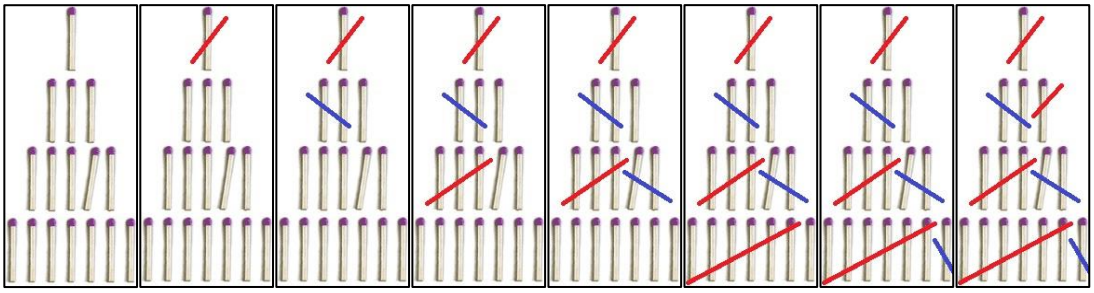


Figura 2- Simulação gráfica do jogo de nim. A vermelho estão marcadas as jogadas do jogador 1 e a azul as jogadas do jogador 2.

Neste caso meramente ilustrativo, o jogador 1 começa por retirar a única peça da primeira fila, o jogador 2 por sua vez retira 2 peças da segunda fila, novamente o jogador 1 retira 3 peças da terceira fila, e assim sucessivamente até que na última jogada o jogador 1 retira a última peça (da 2ª fila) e vence o jogo.

Programa

De forma a criar um adversário virtual que descubra algum tipo de estratégia vencedora simulou-se a realização de vários jogos com 2 jogadores virtuais. A torre inicial considerada é a seguinte:

linha		Nº de peças
0	*[1]*	1
1	*[1, 1, 1]*	3
2	*[1, 1, 1, 1, 1]*	5
3	*[1, 1, 1, 1, 1, 1, 1]*	7

Primeiramente construiu-se um dicionário (uma estrutura do Python que a cada chave faz corresponder um valor), que contempla todas as jogadas (chaves), para cada uma das combinações de peças possíveis, e as respetivas pontuações (valores) atribuídas a cada uma dessas jogadas (obtidas com a simulação, dos diversos jogos, descrita seguidamente). A este dicionário chamou-se *stat*.

Por exemplo, para a jogada inicial, na qual a torre tem a sequência total de peças (1,3,5,7) as jogadas possíveis são, portanto:

- *{(1,3,5,7): {(0, 1): 0, (1, 1): 0, (1, 2): 0, (1, 3): 0, (2, 1): 0, (2, 2): 0, (2, 3): 0, (2, 4): 0, (2, 5): 0, (3, 1): 0, (3, 2): 0, (3, 3): 0, (3, 4): 0, (3, 5): 0, (3, 6): 0, (3, 7): 0}}*

Esta informação lê-se da seguinte forma: para a combinação inicial de peças (1,3,5,7) as jogadas possíveis são (pela ordem apresentada), retirar da primeira linha (linha 0) 1 peça, i.e. (0,1); retirar da 2ª linha (linha 1) 1 peça, i.e. (1,1); etc.; retirar da última linha (linha 3) 7 peças, i.e. (3,7). Note-se que cada uma das jogadas tem o valor correspondente de 0 por se tratar do *stat* antes da simulação.

Para se obter a estratégia vencedora criou-se outro dicionário, *moves*, no qual são colocadas as jogadas feitas pelos jogadores 1 (*moves1*) e 2 (*moves2*) em cada jogo

simulado. Começa-se a simulação com o jogador 1 e inicialmente estas jogadas são feitas utilizando Monte Carlo, ou seja, em cada jogada são sorteados 2 aleatórios, o primeiro, n1, correspondente à linha da torre (número inteiro aleatório entre 0 e 3) e o segundo, n2, correspondente ao número de peças a retirar (número inteiro aleatório entre 1 e o nº de peças na linha n1). Em seguida troca-se para o jogador 2 e repete-se o processo até a torre ficar vazia.

Findo o jogo, às jogadas feitas pelo jogador vencedor adiciona-se 1 ponto e às jogadas associadas ao jogador derrotado subtrai-se 1 ponto, e registam-se estes valores no *stat*. Repete-se esta simulação um número arbitrário de vezes, tendo em conta que quanto maior for o nº de jogos melhor será a estratégia encontrada pelo algoritmo.

Este programa recorre também a Machine Learning no sentido em que em cada turno dos 2 jogadores é avaliado o *stat* (nomeadamente a parte correspondente à combinação de peças desse turno) e escolhe-se como jogada a chave com melhor pontuação. No caso em que, para uma dada combinação de peças, a pontuação seja inferior ou igual a 0 recorre-se novamente a Monte Carlo para se realizar a jogada.

Isto permite que seja obtida a estratégia pretendida, dado que com o decorrer dos jogos surgirão jogadas com maior pontuação que podem constituir uma potencial estratégia vencedora.

Para exemplificar verificam-se as pontuações obtidas para 2 combinações de peças distintas numa simulação de 5000000 jogos:

- *{(0, 0, 0, 2): {(3, 1): 0, **(3, 2): 319682**}}*
- *{(1, 3, 5, 6): {(0, 1): -3, **(1, 1): 177674**, (1, 2): 0, (1, 3): 0, (2, 1): -3, (2, 2): -1, (2, 3): 0, (2, 4): 0, (2, 5): -2, (3, 1): -1, (3, 2): -2, (3, 3): -3, (3, 4): -1, (3, 5): -2, (3, 6): -4}*

No primeiro caso (sobram duas peças na última linha), a melhor jogada é claramente retirar as duas peças que sobram e não apenas uma, pelo que a pontuação da jogada correspondente é 319682 e a pontuação da jogada em que se retira uma peça é nula.

No segundo caso (foi apenas retirada uma peça da última linha), a melhor jogada já não é tão clara, mas neste caso a jogada preferencial corresponde a tirar 1 peça da linha 1, uma vez que as restantes pontuações são nulas ou negativas.

Simulação

Tendo em conta o programa referido anteriormente foi obtida uma determinada pontuação de jogadas simulando 5000000 jogos. Em seguida é apresentado o jogo com um adversário humano.

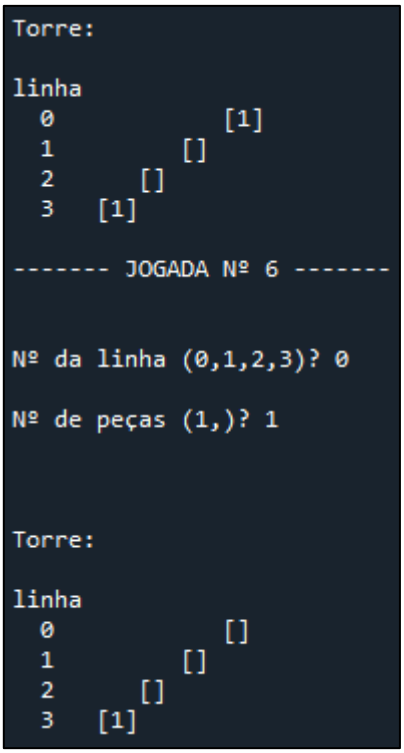
Inicialmente dá-se a opção de escolha para começar o utilizador o jogo ou se começa o computador e apresenta-se a torre inicial (de 16 peças). Neste caso começa o computador que faz a sua jogada (retirar 3 peças da linha 2):

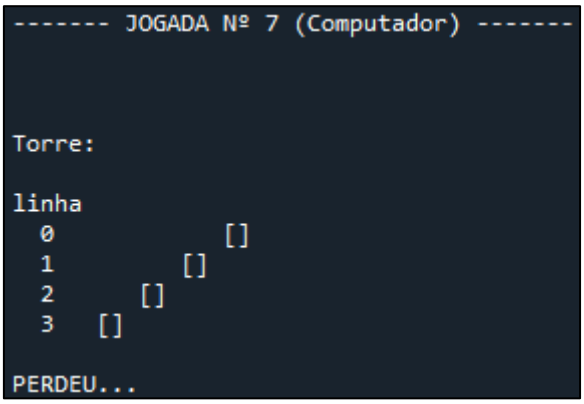
```
Quem começa? ['eu', 'computador'] computador

Torre:

linha
  0           [1]
  1        [1, 1, 1]
  2     [1, 1, 1, 1, 1]
  3  [1, 1, 1, 1, 1, 1, 1]

------- JOGADA Nº 1 (Computador) -------



Torre:

linha
  0           [1]
  1        [1, 1, 1]
  2     [1, 1]
  3  [1, 1, 1, 1, 1, 1, 1]
```

Em seguida o utilizador faz a sua jogada, escolhe a linha e nº de peças que quer retirar (retirou 1 peça da linha 1):

```
------- JOGADA Nº 2 -------


Nº da linha (0,1,2,3)? 1

Nº de peças (1, 2, 3)? 1



Torre:

linha
  0           [1]
  1        [1, 1]
  2     [1, 1]
  3  [1, 1, 1, 1, 1, 1, 1]
```

O jogo continua até chegar ao fim, em que, neste caso, o utilizador fica encurralado e tira a última peça da linha 0 e computador ganha retirando a última peça da torre:





Considerações finais

Este trabalho tinha como principal objetivo simular o jogo de nim utilizando Monte Carlo e obter uma estratégia vencedora recorrendo a Machine Learning, o que se conseguiu com sucesso, dado que se trata de um jogo relativamente básico e, portanto, com um número relativamente reduzido de jogadas possíveis (o que permitiu esta análise num simples computador portátil).

Contudo há alguns aspetos que poderiam ser melhorados no trabalho, nomeadamente, caso o jogador conseguisse vencer o computador (situação pouco provável) atualizar o *stat* de modo a contemplar essas jogadas; a interface entre jogador e computador também poderia ser mais elaborada e interativa, de modo a facilitar a jogabilidade do utilizador, no entanto essa parte do design técnico já não se enquadra propriamente no âmbito da cadeira de Simulação e Métodos de Monte Carlo nem tem propriamente qualquer influencia no objetivo pretendido.

Anexo: Código em Python

```
#%% IMPORTS
from random import randint
from copy import deepcopy

#%%############################## TORRE INICIAL ###################################

number=16
l=int(number**0.5) #nº de linhas de peças
b=2*l-1 #nº de peças da base da pirâmide

def palitos(n):
    p=[]
    for i in range(1,n+1,2):
        p.append(i*[1])
    return p

p=palitos(b)

def torre(x):
    "Print da torre de palitos"
    print('\nTorre:\n\nlinha')
    l=[]
    for i in range(len(x)):
        l.append(i*3)
    l.reverse()
    for i in range(len(l)):
        print(f'  {i}   {(l[i])*" "}{x[i]}')
    return

#%%############################## ESTATÍSTICA ###################################

'COMBINAÇÕES DE PEÇAS POSSÍVEIS'
comb=[]
for i in range(len(p[0])+1):
    for j in range(len(p[1])+1):
        for k in range(len(p[2])+1):
            for w in range(len(p[3])+1):
                comb.append([i,j,k,w])
comb.pop(0)
comb.reverse()

'JOGADAS POSSÍVEIS (em cada combinação de peças): (l,n), com l = nº da linha (a contar a partir do 0) e \
n = nº de peças retiradas'
stat={}
for i in range(len(comb)):
    z=comb[i]
    dic={}
    for j in range(len(z)):
        d=z[j]
        for k in range(1,d+1):
            for w in range(1,min(k,d)+1):
                dic[(j,w)] = 0
            stat[tuple(comb[i])]=dic


#%%############################## SIMULAÇÃO ###################################

def nim(jogos):

    # print('\nLegenda:\nt-turno; j-jogador; l-nº da linha; n-nº de peças retiradas\n')

    for g in range(1,jogos+1):
        t=1 #turno

        moves={} #jogadas
        moves[1]={} #jogadas do jogador 1
        moves[2]={} #jogadas do jogador 2

        # print(f'--- {g}º jogo ---')
        # print('t','j','l','n')

        peças=[]
```

```
    z=[[]]
    p=palitos(b)
    while p!=z*l:
      '1 Jogo'
      peças.append((len(p[0]),len(p[1]),len(p[2]),len(p[3]))) #combinação de peças no momento da jogada

      jogador=1 if t%2!=0 else 2

      x=moves[jogador][t]=max(stat[peças[t-1]], key=stat[peças[t-1]].get)#jogada com mais pontos
      if stat[peças[t-1]][x]>0:
        for i in range(list(x)[1]):
          p[list(x)[0]].remove(1)

      else: #jogada aleatória no caso de não haver melhor jogada
        escolha1=randint(0,len(p)-1) #escolha de linha
        while p[escolha1]==[]:
          escolha1=randint(0,len(p)-1)

        escolha2=randint(1,len(p[escolha1])) #nº de peças retiradas
        for i in range(escolha2):
          p[escolha1].remove(1)

        moves[jogador][t]=(escolha1,escolha2)

      # print(t,jogador,escolha1,escolha2,p)

      t+=1

    # print(f'\nGanha jogador{jogador}\n')

    'Pontuação das jogadas'
    # last player wins, collect statistics:
    for i in moves[jogador]:
     stat[peças[i-1]][moves[jogador][i]]+= 1

    # switch to other player that lost:
    jogador = 2 if jogador == 1 else 1
    for i in moves[jogador]:
     stat[peças[i-1]][moves[jogador][i]] -= 1

  # """
  # print das melhores jogadas
  # """
  # print('\n\nMelhores Jogadas para uma dada combinação de peças (l- nº da linha; n- nº de peças retiradas):')
  # print('\ncomb. de peças  jogada(l,n)  pontuação')
  # for i in range(len(comb)):
  #    best = max(stat[tuple(comb[i])], key=stat[tuple(comb[i])].get)
  #    value = stat[tuple(comb[i])][best]
  #    # if value<0:
  #    #    best = '-'
  #    #    value = ''
  #    print (f"{tuple(comb[i])}:     {best}     {value:>5}")

  return

#%%############################    JOGO    ###################################



# 'DIFICULDADE'
# sdif=['facil','medio','dificil']
# dif=str(input(f'\nDificuldade? {sdif} '))
# while dif not in sdif:
#    print('\nEscreva uma das seguintes opcoes: facil, medio, dificil...')
#    dif=str(input())
# if dif =='facil':
#    nim(10)
# elif dif =='medio':
#    nim(1000)
# elif dif=='dificil':
#    print('loading')
#    nim(1000000)
#    print('done')

# nim(1000000)
'INICIO'
```

```
sstart=['eu','computador']
start=str(input(f'\nQuem começa? {sstart} '))
while start not in sstart:
   print('\nEscreva uma das seguintes opcoes: eu, computador...')
   start=str(input())

'JOGO'
p1=deepcopy(p)
j=0

if start=='eu':
   torre(p1)
   while p1!=[[],[],[],[]]:
      j+=1
      print(f'\n------- JOGADA Nº {j} -------')

      'JOGADOR'
      n1=int(input('\nNº da linha (0,1,2,3)? '))
      n2=int(input(f'Nº de peças {tuple(range(1,len(p1[n1])+1))}? '))

      for i in range(n2):
         p1[n1].remove(1)

      torre(p1)

      if p1==[[],[],[],[]]:
         print('\nVENCEU!!!')
         break
      # print(f'\nEu: \n{p1}')

      j+=1
      print(f'\n------- JOGADA Nº {j} (Computador) -------')
      'COMPUTADOR'
      peças=(len(p1[0]),len(p1[1]),len(p1[2]),len(p1[3]))
      x=max(stat[peças], key=stat[peças].get)#jogada com mais pontos
      if stat[peças][x]>0:
         for i in range(list(x)[1]):
            p1[list(x)[0]].remove(1)

      else: #jogada aleatória no caso de não haver melhor jogada
         escolha1=randint(0,len(p1)-1) #escolha de linha
         while p1[escolha1]==[]:
            escolha1=randint(0,len(p1)-1)

         escolha2=randint(1,len(p1[escolha1])) #nº de peças retiradas
         for i in range(escolha2):
            p1[escolha1].remove(1)

      torre(p1)

      if p1==[[],[],[],[]]:
         print('\nPERDEU...')
         break
      # print(f'\nComputador: {p1}')

elif start=='computador':
   torre(p1)
   while p1!=[[],[],[],[]]:
      j+=1
      print(f'\n------- JOGADA Nº {j} (Computador) -------')
      print('\n')
      'COMPUTADOR'
      peças=(len(p1[0]),len(p1[1]),len(p1[2]),len(p1[3]))
      x=max(stat[peças], key=stat[peças].get)#jogada com mais pontos
      if stat[peças][x]>0:
         for i in range(list(x)[1]):
            p1[list(x)[0]].remove(1)

      else: #jogada aleatória no caso de não haver melhor jogada
         escolha1=randint(0,len(p1)-1) #escolha de linha
         while p1[escolha1]==[]:
            escolha1=randint(0,len(p1)-1)

         escolha2=randint(1,len(p1[escolha1])) #nº de peças retiradas
         for i in range(escolha2):
            p1[escolha1].remove(1)
```

```
torre(p1)

if p1==[[],[],[],[]]:
   print('\nPERDEU...')
   break

j+=1
print(f'\n------- JOGADA Nº {j} -------')
'JOGADOR'
n1=int(input('\nNº da linha (0,1,2,3)? '))
n2=int(input(f'Nº de peças {tuple(range(1,len(p1[n1])+1))}? '))
print('\n')
for i in range(n2):
   p1[n1].remove(1)

torre(p1)

if p1==[[],[],[],[]]:
   print('\nVENCEU!!!')
   break

# print(f'\nEu: {p1}')
```