# O jogo da Sueca

Neste tutorial, iremos desenvolver uma versão do jogo da Sueca, onde quatro jogadores se constituem em equipas de dois. Nesta versão do jogo, um dos jogadores será "humano" e os outros 3 serão controlados pelo computador.

Neste jogo, cada jogador tem uma lista de 10 cartas (uma mão), que se vai reduzindo ao longo de 10 jogadas, até os jogadores ficarem sem cartas. Ao terminar o jogo, é necessário contabilizar o número de pontos de cada equipa (de 2 jogadores), através das cartas ganhas (total máximo, 120 pontos).

Neste jogo, não se utilizam as cartas 8, 9 e 10, tendo cada carta o seguinte valor em pontos:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Carta | "A" | 2 | 3 | 4 | 5 | 6 | 7 | "Q" | "J" | "K" |
| Pontos | 11 | 0 | 0 | 0 | 0 | 0 | 10 | 2 | 3 | 4 |

|  |  |
| --- | --- |
| **Eye** | Observe que as cartas numéricas de 2 a 6 não têm qualquer valor, enquanto que a carta 7 vale 10 valores. As figuras são representadas por caracteres maiúsculos: "A", "Q", "J" e "K". |

Cada "mão" de um jogador é uma lista de cartas, onde cada carta é representada por um tuplo (*tuple*) com 2 elementos (um par): o número/figura da carta e o naipe (um caracter).

O naipe é representado pela letra inicial: "O" – ouros; "C" – copas; "E" – espadas; "P" – paus.

A título de exemplo, analise a seguinte mão de um jogador, representada por uma lista de pares:

mao = [(3, "C"),("A","C"), (2, "O"), ("K","P"), (7, "E")]

|  |  |
| --- | --- |
| **Head with gears** | Porque se terá optado por utilizar uma lista para a "mão" do jogador e um tuplo para a "carta"? |

Caso não saiba como jogar o jogo da Sueca, leia o seguinte artigo da Wikipedia:

* [O Jogo da Sueca](https://pt.wikipedia.org/wiki/Sueca_(jogo_de_cartas))

# Estruturação do jogo

Este jogo apresenta uma certa complexidade, pelo que será recomendável utilizar os princípios da abstração e da decomposição em subproblemas, para o estruturar:

1. Abstração "carta": criar funções para criar objetos do tipo “carta” e aceder à "figura" e ao "naipe";
2. Pontos de uma carta: retorna o valor de uma carta;
3. Dar uma carta: criar uma carta específica do baralho;
4. Pontuação: calcular e devolver a pontuação de um conjunto de cartas;
5. Carta mais alta: determinar a carta mais alta de um naipe, num conjunto de cartas;
6. Criar o Baralho de cartas: criar uma lista de 40 cartas, com todas as 10 figuras dos 4 naipes;
7. Baralhar as cartas: fazer uma sequência de trocas cartas, de forma aleatória;
8. Mostrar as cartas: visualizar as cartas.
9. Interação com o Jogador: Mostrar a mão do jogador e pedir a carta a jogar;
10. Inteligência Artificial: Jogadas dos outros jogadores, controlados pelo computador.
11. Jogo da Sueca: finalmente o ciclo do jogo...

Este jogo utilizará uma interface com o utilizador baseada em texto, na linha de comando.

|  |  |
| --- | --- |
| **Head with gears** | Dedique uns momentos de reflexão sobre o conteúdo de cada um dos 11 pontos referidos, tentando justificar a sua existência. No entanto, não se preocupe caso não encontre justificação para todos eles, pois irá encontrá-la na parte restante do tutorial. |

# Abstração Carta

Uma carta é representada por um tuplo de dois valores (um par): a figura e o naipe.

(figura, naipe)

A função cria\_carta(figura, naipe) retorna um tuplo a partir dois parâmetros: a figura e o naipe. Trata-se de um construtor da abstração Carta.

def cria\_carta(figura, naipe):

return (figura, naipe)

Para aceder aos dois valores de uma carta, foram criados dois seletores da abstração Carta:

def figura(carta):

return carta[0]

def naipe(carta):

return carta[1]

# Pontos de uma carta

A função pontos()retorna um número inteiro correspondente ao valor de uma carta, de acordo com a tabela indicada. Esta função possui apenas um parâmetro, carta, que tanto pode ser um número inteiro como um caráter.

Analise o código da função pontos() onde se optou por utilizar uma estrutura condicional de seleção múltipla.

def pontos(figura):

if figura == "A":

return 11

elif figura == "Q":

return 2

elif figura == "J":

return 3

elif figura == "K":

return 4

elif figura == 7:

return 10

else:

return 0

|  |  |
| --- | --- |
| **Head with gears** | Onde se encontram as opções que selecionam as cartas de 2 a 6? |

# Determinar figura correspondente ao número de ordem

A função da\_figura()tem apenas um parâmetro, o número de ordem da carta, e retorna o número ou a figura (caráter) correspondente, de acordo com a seguinte tabela:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| nº de ordem | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Carta | "A" | 2 | 3 | 4 | 5 | 6 | 7 | "Q" | "J" | "K" |

Analise o código da função. Mais uma vez foi utilizada uma estrutura condicional de seleção múltipla.

def da\_figura(numero):

if numero == 1:

return "A"

elif numero == 8:

return "Q"

elif numero == 9:

return "J"

elif numero == 10:

return "K"

else:

return numero

# Determinar a pontuação de um conjunto de cartas

No final do jogo é necessário contar a pontuação total das cartas ganhas por cada equipa.

A função pontuacao() tem apenas um parâmetro, que é uma lista de cartas, e faz o somatório dos pontos de todas as cartas:

def pontuacao(cartas):

soma = 0

for carta in cartas:

soma += pontos(figura(carta))

return soma

# Determinar a maior carta de um determinado naipe

Quando um jogador tem que "assistir" uma jogada, normalmente tenta verificar qual a maior carta (maior número de pontos) da sua "mão" correspondente ao naipe da jogada. A função carta\_maior() tem dois parâmetros:

* mao, que é uma lista de cartas;
* naipe\_jogada, que é um caracter correspondente ao naipe da jogada ("C", "O", "P" ou "E").

Esta função retorna apenas o numero/figura da carta. Quando não existe nenhuma carta desse naipe retorna False.

Analise o código da função e repare na estrutura de ciclo contado que percorre a mão de cartas.

def carta\_maior(mao, naipe\_jogada):

# inicializacao

pontos\_maximo = -1 # menor que minimo - zero valores para cartas de 2 a 6

carta\_maximo = ""

# percorre a mao do jogador

for carta in mao:

if naipe(carta) == naipe\_jogada and pontos(figura(carta)) > pontos\_maximo:

carta\_maximo = figura(carta)

pontos\_maximo = pontos(carta\_maximo)

# valor de retorno

if pontos\_maximo >= 0:

return carta\_maximo

else:

return False

# Criar o Baralho de cartas

O baralho de cartas é composto por 40 cartas, contemplando 10 figuras de cartas para cada um dos 4 naipes.

A função cria\_baralho() não tem qualquer parâmetro:

def cria\_baralho():

baralho = []

for naipe in ["O", "C", "P", "E"]: # para cada um dos 4 naipes

for c in range(1, 11): # para cada uma das 10 cartas

baralho.append(cria\_carta(da\_figura(c), naipe))

return baralho

# Baralhar as cartas

Em qualquer jogo de cartas é geralmente necessário baralhar as cartas, trocando-as de posição.

A função baralha\_cartas(baralho, vezes) tem como parâmetros um baralho (uma lista de cartas) e o número de vezes que se pretende baralhar as cartas:

def baralha\_cartas(baralho, vezes):

for i in range(vezes):

#troca cartas

i = random.randint(0, len(baralho)-1)

j = random.randint(0, len(baralho)-1)

temp = baralho[i]

baralho[i] = baralho[j]

baralho[j] = temp

|  |  |
| --- | --- |
| **Head with gears** | Percebe-se agora porque se criou o baralho com uma lista - para o poder baralhar!  Poderia também ter sido utilizado um tuplo?  Observe também o algoritmo de troca das cartas.  temp = baralho[i]  baralho[i] = baralho[j]  baralho[j] = temp  Porque foi utilizada uma variável local (temp) ? |

# Interação com o Jogador

Na interação com o jogador, é essencial proporcionar-lhe a informação necessária para poder fazer a jogada (output) e um modo de interação simples para atuar sobre o jogo (input).

## Output

Para este propósito, iremos criar a função mostra\_mao() que visualiza a mão do jogador em modo texto, mas numa configuração atrativa:

A sua mao:

1 2 3 4 5 6 7 8 9 10

+--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+

| e| | p| | p| | p| | c| | c| | o| | o| | o| | o|

|K | |A | |J | |Q | |5 | |2 | |A | |7 | |4 | |2 |

+--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+

Esta função começa por ordenar a lista de cartas por naipe e por figura, de forma a facilitar a escolha do jogador sobre a carta a jogar.

Para tal foi necessário criar duas cadeias de caracteres com uma ordenação por naipe e uma ordenação por figura. E foi criada uma função numero() que retorna um valor numérico inteiro para cada carta. Esta função possibilitará a comparação das cartas ao utilizar o método sort(),que será utilizada para ordenar as listas de cartas.

# mostra cartas

naipes = "OCPE"

figuras = "23456QJK7A"

def numero(t):

return naipes.index(naipe(t))\*100+figuras.index(str(figura(t)))

|  |  |
| --- | --- |
| **Head with gears** | Para entender melhor a ordenação com esta função, calcule o valor das seguintes cartas:  (2, "O"), (3, "O"), (4, "O"), (4, "C"), (4, "P"), (4, "E")  E porque será necessário utilizar a função str() para a figura? |

Observe agora a função mostra\_mao().

def mostra\_mao(mao):

print ("A sua mao:")

# ordena a mao

mao.sort(key=numero, reverse = True)

# numeracao: 1 2 3 4 5 6 7 8 9 10

txt = ""

for i in range (len(mao)):

txt += " " + str(i+1) + " "

print (txt)

# bordo superior: +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ txt = ""

for i in range (len(mao)):

txt += "+--+ "

print (txt)

# naipe: | e| | p| | p| | p| | c| | c| | o| | o| | o| | o|

txt = ""

for carta in mao:

txt += "| "+ naipe(carta).lower() + "| "

print (txt)

# figura: |K | |A | |J | |Q | |5 | |2 | |A | |7 | |4 | |2 |

txt = ""

for carta in mao:

txt += "|"+ str(figura(carta)) + " | "

print (txt)

# bordo inferior: +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+

txt = ""

for i in range (len(mao)):

txt += "+--+ "

print (txt)

|  |  |
| --- | --- |
| **Eye** | Observe em primeiro lugar a utilização do método [sort()](mailto:https://www.w3schools.com/python/ref_list_sort.asp) das listas.  É feita uma ordenação inversa (do maior para o menor) com o atributo reverse = True.  E como os elementos da lista têm como origem uma abstração definida pelo utilizador (Abstração Carta), atribui-se a função numero(), definida anteriormente, para fazer a comparação numérica entre dois desses elementos.  Observe também a forma como são desenhadas as cartas, com caracteres, e "linha a linha". |

## Input

Este subproblema foca-se na interação com o jogador "humano". Em primeiro lugar mostra-se a "mão do jogador", chamando a função mostra\_mao(), para que este possa escolher a sua jogada.

Em seguida, lê-se o número da carta (número de ordem na mão do jogador) que este pretende jogar.

A função joga\_jogador()tem como único parâmetro mao, que é a lista de cartas (tuplos) referente à mão do jogador.

No final, a função retorna a carta selecionada, apagando-a da sua mão. Esta função é assim um modificador!

def joga\_jogador (mao):

# visualização da mão do jogador

mostra\_mao(mao)

# valida o input do jogador para ser um número válido

n=0

while not (n > 0 and n <= len(mao)):

resp = input ("Qual a carta que pretende jogar? (1 a " + str(len(mao)) + "): ")

n = int(resp)

# retorna a carta e elimina-a da mão do jogador

carta = mao[n-1]

del mao[n-1]

return carta

|  |  |
| --- | --- |
| **Head with gears** | Na condição do ciclo while comparou-se o número lido com o número de cartas da "mão" do jogador. Porquê?  Utilizou-se del para eliminar um elemento da lista. Que outra solução poderia utilizar? |

# Inteligência Artificial (IA)

Três dos jogadores são controlados pelo computador. A Inteligência Artificial (IA) estuda um conjunto de técnicas que permitem programar um computador para simular um comportamento "inteligente". Não vamos explorar estas técnicas a fundo, mas apenas desenhar um algoritmo que siga regras de jogo simples, para potenciar um desafio interessante ao jogador humano.

|  |  |
| --- | --- |
| **Head with gears** | O primeiro passo para criar as regras do jogador não humano (NPC) é pensarmos como é que nós jogamos...  Simule uma ou duas rondas de jogadas e analise a forma como decide a carta a jogar...  Verifique se é semelhante ao algoritmo que é enunciado a seguir. |

O algoritmo define-se com os seguintes passos:

1. Verificar o naipe da jogada ou se é o jogador a jogar em primeiro lugar.   
   Neste último caso seleciona uma carta aleatória;
2. Selecionar a carta mais alta do naipe da jogada;
3. Caso não possua nenhuma carta do naipe da jogada, tentar "cortar" a jogada com um trunfo;
4. Caso também não tenha um trunfo, jogar a primeira carta da sua mão.

A função joga\_NPC()tem como parâmetros:

* cartas, que é uma lista de 4 cartas referente à jogada;
* mao, que é a lista de cartas do jogador;
* naipe\_jogada**,** o naipe da jogada;
* naipe\_trunfo**,** o naipe de trunfo.

Analise o código da função**.**

def joga\_NPC (cartas, mao, naipe\_jogada, naipe\_trunfo):

# verifica o naipe da jogada

if naipe\_jogada == "":

# neste caso é o primeiro a jogar

naipe\_jogada = random.choice(["O", "P", "C", "E"]) # naipe 'a sorte...

# Tenta assistir à jogada

figura\_jogada = carta\_maior(mao, naipe\_jogada)

if figura\_jogada != False:

for i in range(len(mao)):

if figura(mao[i]) == figura\_jogada and naipe(mao[i]) == naipe\_jogada:

del mao[i]

break

return cria\_carta(figura\_jogada, naipe\_jogada)

else:

# caso nao tenha cartas do naipe da jogada

figura\_trunfo = carta\_maior(mao, naipe\_trunfo)

if figura\_trunfo != False:

for i in range(len(mao)):

if figura(mao[i]) == figura\_trunfo and naipe(mao[i]) == naipe\_trunfo:

del mao[i]

break

return cria\_carta(figura\_trunfo, naipe\_trunfo)

else:

# retorna a primeira carta... Desafio: Melhorar...

carta\_outro\_naipe = mao[0]

del mao[0]

return carta\_outro\_naipe

|  |  |
| --- | --- |
| Balloon animal | Os jogadores controlados pelo computador são denominados "*Non Player Character*", sendo frequentemente designados pela sigla NPC.  A área da Inteligência Artificial (IA) desenvolve diversos algoritmos que promovem um comportamento "inteligente", quer pela simulação de heurísticas utilizadas pelos especialistas, como também através da própria aprendizagem computacional (*Machine Learning*).  Estes algoritmos são utilizados frequentemente nos jogos digitais para o controlo dos NPC.  **Desafio (opcional):** Tente encontrar um melhor algoritmo para a função joga\_NPC().  (Este desafio tem grau de dificuldade elevado.)  *Pista: O algoritmo deve considerar quem está a ganhar (a sua equipa ou a adversária) e proceder de forma a otimizar os pontos da equipa.*  *Para tal deverá aceder ao parâmetro* cartas*, com a lista de cartas da jogada, e que não foi utilizado no algoritmo apresentado.* |

# Jogo da Sueca

Finalmente, a função sueca() implementa o jogo, começando pela inicialização, e depois através do ciclo de jogo.

A inicialização cria um baralho, seguindo-se o baralhar das cartas. Depois é selecionado o trunfo do jogo, a partir da primeira carta do baralho, e, finalmente, divide-se as 40 cartas do baralho em 4 mãos de 10 cartas, para os 4 jogadores. As listas cartas\_equipaecartas\_oponente servem para acumular o resultado de cada equipa de forma a se poder contabilizar a pontuação, no final.

def sueca():

# 1. cria baralho

baralho = cria\_baralho()

# 2. baralha as cartas (ex. 100 vezes)

baralha\_cartas(baralho, 100)

# escolhe o trunfo

trunfo = naipe(baralho[0])

# 3. cria as mãos dos 4 jogadores (o primeiro jogador e' o humano)

jogo = [0, 0, 0, 0]

for i in range(4):

jogo[i] = baralho[i\*10 : i\*10 + 10]

cartas\_equipa = []

cartas\_oponente = []

|  |  |
| --- | --- |
| Eye | Observe a forma como o baralho é segmentado em 4 conjuntos de 10 cartas, que são as mãos dos jogadores. |

O ciclo de jogo é executado através de um ciclo contado (for) uma vez que o jogo consiste em exatamente 10 jogadas.

# 4. faz jogo - 10 jogadas

jogador = 0 # comeca o jogador

for i in range(10):

O algoritmo de cada jogada é o seguinte:

1. Visualização da jogada.

# visualizar informação da jogada

print("\n\*\* Jogada " + str(i+1) + " (o trunfo e' " + trunfo + ") \*\* ")

1. Cada um dos 4 jogadores joga uma carta, sequencialmente, de forma a completar uma jogada. Inicializa-se a lista de 4 cartas da jogada e o naipe da jogada, que irá ser definido pelo jogador que joga primeiro. Depois, num ciclo contado (for) com 4 iterações, guarda-se a jogada de cada jogador, visualiza-se a jogada de cada um e atualiza-se o naipe da jogada a partir da carta do primeiro jogador.

# os 4 jogadores escolhem a carta a jogar

cartas = ["", "", "", ""]

naipe\_jogada = "" # indica que o jogador que joga primeiro pode escolher o naipe

for j in range (4):

# joga humano ou computador?

if jogador == 0:

cartas[jogador] = joga\_jogador(jogo[jogador])

else:

cartas[jogador] = joga\_NPC(cartas, jogo[jogador], naipe\_jogada, trunfo)

print ("Jogador " + str(jogador+1) + " jogou : " + str(cartas[jogador]))

# atualiza o naipe da jogada e o próximo jogador a jogar

if naipe\_jogada == "":

naipe\_jogada = naipe(cartas[jogador])

jogador = (jogador + 1) % 4 # jogador à direita

1. Depois de os quatro jogadores jogarem, a lista cartas está preenchida. Primeiro verifica se alguém jogou trunfo e, em caso afirmativo ganha o jogador que tiver a carta mais alta do naipe de trunfo. Caso não haja trunfos, ganha quem tiver a carta mais alta do naipe da jogada. Atualiza-se assim a variável jogador com o vencedor da jogada, que será o próximo a iniciar a jogada.

# verifica-se quem ganha e acrescenta-se as cartas à equipa correspondente

carta\_naipe\_trunfo = carta\_maior(cartas, trunfo)

if carta\_naipe\_trunfo != False:

# ganhou quem colocou trunfo

for j in range(len(cartas)):

if figura(cartas[j]) == carta\_naipe\_trunfo and naipe(cartas[j])==trunfo:

jogador = j

break

else:

# maior carta do naipe jogado

carta\_jogada = carta\_maior(cartas, naipe\_jogada)

for j in range(len(cartas)):

if figura(cartas[j]) == carta\_jogada and naipe(cartas[j])== naipe\_jogada:

jogador = j

break

1. No final de cada jogada acrescenta-se as cartas dessa jogada à equipa do vencedor, para no final ser possível contar a pontuação.

# verifica quem ganha a jogada e acrescenta cartas para a pontuação dessa equipa

if jogador % 2 == 0:

cartas\_equipa.extend(cartas)

print("Jogada para a sua equipa!")

else:

cartas\_oponente.extend(cartas)

print("Jogada para a equipa adversária.")

Após o fim do ciclo de jogo é verificada a pontuação de cada equipa, indicando o vencedor (pontuação superior a 60) ou o empate (caso ambas as equipas obtenham 60 pontos).

# verifica a equipa vencedora

if pontuacao(cartas\_equipa) > 60:

print("Ganhou sua a equipa. " + str(pontuacao(cartas\_equipa)) + " pontos. Parabens!")

elif pontuacao(cartas\_oponente) > 60:

print("Ganhou a equipa adversaria" + str(pontuacao(cartas\_oponente)) + " pontos.")

else:

print("Empataram com 60 pontos.")

Teste o jogo com a seguida chamada da função sueca:

Sueca()

# Desafio final

Uma das características dos jogos digitais é terem interfaces com o utilizador muito intuitivas. Melhore as funções de interação com o jogador humano, através de uma interface gráfica, com o rato, utilizando o módulo p5.

*Pista: Poderá utilizar imagens de baralhos de cartas que existem na internet.   
Para tal basta recorrer à função* [*image()*](mailto:https://p5.readthedocs.io/en/latest/examples/image/load%20and%20display%20image.html) *do p5.*

Um desafio de nível elevado, que se propõe a título opcional, é melhorar o algoritmo de IA do computador.

|  |  |
| --- | --- |
| **Share with person** | Partilhe o **seu** jogo! |