 **Introdução à Programação**

Mestrado em Multimédia

# TP9. Jogo da Sueca

António Coelho

O objetivo principal deste tutorial é o de trabalhar as abstrações de dados e reforçar a utilização da iteração para a resolução de problemas.

Mais especificamente iremos explorar os seguintes objetivos de aprendizagem:

* Utilizar a iteração para a resolução de problemas que envolvam repetição de tarefas;
* Aplicar listas e tuplos na resolução de problemas.

Recomenda-se a leitura dos seguintes textos de apoio antes da realização do tutorial:

How to Think Like a Computer Scientist: Learning with Python 3ed; Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, 2012

* Capítulo 9 - [Tuples](http://www.openbookproject.net/thinkcs/python/english3e/tuples.html)
* Capítulo 11 - [Lists](http://www.openbookproject.net/thinkcs/python/english3e/lists.html)

# O jogo da Sueca

Neste tutorial iremos desenvolver uma versão do jogo da Sueca, onde quatro jogadores jogam 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 para jogar. Ao terminar o jogo é necessário contabilizar o número de pontos de cada equipa de 2 jogadores, através das cartas ganhas no jogo (total de 120 pontos).

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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Carta | “A” | 2 | 3 | 4 | 5 | 6 | 7 | “Q” | “J” | “K” |
| Valor | 11 | 0 | 0 | 0 | 0 | 0 | 10 | 2 | 3 | 4 |

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: 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:

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

# Estruturação do jogo

Este jogo apresenta uma certa complexidade, pelo que será útil utilizar a decomposição para estruturar melhor o jogo em subproblemas:

1. Abstração “carta”: criar funções para criar cartas 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 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 algoritmo do jogo...

# Abstração Carta

Uma carta é representada por um tuplo de dois valores: 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(carta)** retorna um número inteiro correspondente ao valor da carta, de acordo com a tabela acima. Esta função possui apenas um parâmetro, a carta, que tanto pode ser um número inteiro como um caracter.

Analise o código da função **pontos(figura)**.

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

# Dar uma carta

A função **da\_carta(valor)** tem apenas um parâmetro, o número de ordem da carta, e retorna o número ou a figura (caracter) 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 **da\_carta(valor)**.

def da\_carta(valor):

if valor == 1:

return "A"

elif valor == 8:

return "Q"

elif valor == 9:

return "J"

elif valor == 10:

return "K"

else:

return valor

# 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(cartas)** tem apenas um parâmetro, que é uma lista de cartas (tuplos):

**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(mao, naipe\_jogada)** tem dois parâmetros:

* **mao**, que é uma lista de cartas (tuplos);
* **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 **carta\_maior(mao, naipe\_jogada)**:

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

**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:**

**if pontos(figura(carta)) > pontos\_maximo:**

**carta\_maximo = figura(carta)**

**pontos\_maximo = pontos(carta\_maximo)**

**# nao tem cartas desse naipe**

**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():**

**# cria uma lista de 40 valores**

**baralho = list(range(40))**

**# define os 4 naipes**

**naipes = ["O", "C", "P", "E"]**

**# constroi o baralho**

**i = 0**

**for naipe in naipes: # para cada um dos 4 naipes**

**for c in range(10): # para cada uma das 4 cartas**

**baralho[i] = cria\_carta(da\_carta(c+1), naipe) # cria carta**

**i += 1**

**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 e o número de vezes que se pretende trocar 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**

# Mostrar as cartas

Na interação com o jogador é essencial proporcionar a este a informação necessária para poder fazer a jogada. Para este propósito iremos criar a função **mostra\_mao(mao)** que mostra a mão do jogador (uma carta por cada linha), e a função **mostra\_carta(carta)** que mostra a carta jogada por cada jogador.

**def mostra\_mao(mao):**

**# pode ser melhorado**

**print ("A sua mao:")**

**for i in range (len(mao)):**

**print(str(i+1) + " - " + str(mao[i]))**

Ambas as funções são muito simples e podem ser melhoradas. Fica como desafio!

# Interação com o Jogador

Este subproblema foca-se na interação com o jogador “humano”. Em primeiro lugar mostra-se a “mão do jogador” para que este possa escolher a sua jogada. Em seguida lê-se o número da carta que este pretende jogar.

A função **joga\_jogador (cartas, mao)** tem como parâmetros:

* **cartas**, que é uma lista de 4 cartas (tuplos) referente à jogada;
* **mao**, que é a lista de cartas (tuplos) do jogador;

No final, a função retorna a carta selecionada, apagando-a da sua mão:

**def joga\_jogador (cartas, mao):**

**mostra\_mao(mao)**

**n=0**

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

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

**if len (resp) > 0 and resp[0] in ["1", "2", "3", "4", "5", "6", "7", "8", "9"]:**

**n = int(resp)**

**else:**

**n = 0**

**carta = mao[n-1]**

**del mao[n-1]**

**return carta**

# Inteligência Artificial

Três dos jogadores são controlados pelo computador. Desenvolver um algoritmo simples para potenciar um desafio interessante ao jogador humano.

O algoritmo passa pelos seguintes passos:

1. Verifica o naipe da jogada ou se é o jogador a jogar em primeiro lugar. Neste 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(cartas, mao, naipe\_jogada, naipe\_trunfo)** tem como parâmetros:

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

Analise o código da função **joga\_NPC(cartas, mao, naipe\_jogada, naipe\_trunfo)**:

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

**# verifica o naipe da jogada**

**if naipe\_jogada == "":**

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

**# Tenta assistir 'a 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**

|  |  |
| --- | --- |
| 🗬 | *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 e proceder de forma a otimizar os pontos da equipa.* |

# Jogo da Sueca

Finalmente a função **sueca()** desenvolve o algoritmo do 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\_equipa** e **cartas\_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. 50 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 = []**

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(cartas, 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 proximo jogador a jogar**

**if naipe\_jogada == "":**

**naipe\_jogada = naipe(cartas[jogador])**

**jogador = (jogador + 1) % 4 # jogador 'a 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 a jogada e acrescenta-se as cartas 'a 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 pontuacao 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 adversaria.")**

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 visualização de informação ao jogador para incrementar esta interação.

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

# Bibliografia

* How to Think Like a Computer Scientist: Learning with Python 3ed; Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, 2012   
  *Available online at:*  <http://www.openbookproject.net/thinkcs/python/english3e/>
* Python Programming Language – Official Website.   
  *Available online at:* <http://www.python.org/>