# The game of "Sueca" (Swedish girl)

In this tutorial, we will develop a version of the "Sueca" game, where four players play in teams of two. In this version of the game, one of the players will be "human" and the other 3 will be controlled by the computer.

In this game, each player has a list of 10 cards (one hand), which is reduced over 10 moves, until the players run out of cards. At the end of the game, it is necessary to count the number of points for each team (of 2 players), through the cards won (maximum total, 120 points).

In this game, cards 8, 9 and 10 are not used, with each card having the following value in points:

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

|  |  |
| --- | --- |
| **Eye** | Note that numeric cards 2 to 6 have no value, while card 7 is worth 10 points. Figures are represented by capital letters: "A", "Q", "J" and "K". |

Each "hand" of a player is a list of cards, where each card is represented by a tuple with 2 elements (a pair): the number / figure of the card and the suit (one character).

The suit is represented by the initial letter: "O" - Ouros (diamonds); "C" - Copas (hearts); "E" - Espadas (swords); "P" - Paus (clubs).

As an example, analyse the following hand of a player, represented by a list of pairs:

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

|  |  |
| --- | --- |
| **Head with gears** | Why have we chosen to use a list for the player's "hand" and a tuple for the "card"? |

If you don't know how to play the Swedish game, read the following Wikipedia article:

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

# Game setup

This game has a certain complexity, so it is recommended to use the principles of abstraction and decomposition into subproblems, to structure it:

1. Abstraction "Carta" (card): create functions to create objects of the type "carta" and access its "figure" and "suit";
2. Points on a card: returns the value of a card;
3. Draw a card: create a specific card from the deck;
4. Scoring: calculating and returning the score for a set of cards;
5. Highest card: determine the highest card of a suit, in a set of cards;
6. Create the Deck of cards: create a list of 40 cards, with all 10 figures of the 4 suits;
7. Shuffle the cards: make a sequence of card exchanges, at random;
8. Show the cards: view the cards.
9. Interaction with the Player: Show the player's hand and ask for the card to play;
10. Artificial Intelligence: Other players' plays, controlled by the computer.
11. game of "Sueca": finally the game loop...

This game will use a text-based user interface on the command line.

|  |  |
| --- | --- |
| **Head with gears** | Dedicate a few moments of reflection on the content of each of the 11 points mentioned, trying to justify its existence. However, don't worry if you can't find justification for all of them, as you'll find it in the rest of the tutorial. |

# Abstraction "Carta" (card)

A card is represented by a tuple of two values (a pair): the figure and the suit.

(figura, naipe)

Function cria\_carta(figura, naipe) returns a tuple from two parameters: the figure and the suit. It is a constructor of abstraction Carta.

def cria\_carta(figura, naipe):

return (figura, naipe)

To access the two values of a letter, two selectors of the Letter abstraction were created:

def figura(carta):

return carta[0]

def naipe(carta):

return carta[1]

# Points of a card

Function pontos()returns an integer corresponding to the value of a card, according to the indicated table. This function has only one parameter, letter, which can be either an integer or a character.

Analyze the code of the pontos () function where we chose to use a conditional multiple selection structure.

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** | Where are the options that select cards 2 to 6? |

# Determine the figure corresponding to the index

Function da\_figura()has only one parameter, the order number of the card, and returns the corresponding number or figure (character), according to the following table:

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

Aanalyze the function code. Once again a conditional multiple selection structure was used.

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

# Determine the score for a set of cards

At the end of the game it is necessary to count the total score of the cards won by each team.

Function pontuacao() has only one parameter, which is a list of cards, and adds the points of all the cards:

def pontuacao(cartas):

soma = 0

for carta in cartas:

soma += pontos(figura(carta))

return soma

# Determine the highest card of a given suit

When a player has to "assist" a move, he usually tries to check which is the highest card (highest number of points) in his "hand" corresponding to the suit of the move. The function carta\_maior() has two parameters:

* mao, a list of cards;
* naipe\_jogada, which is a character corresponding to the suit to play ("C", "O", "P" ou "E").

This function returns only the number / figure of the card. When there is no card of that suit it returns False.

Analyze the function code and notice the counted loop structure that runs through the hand of cards.

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

# Create the Deck of Cards

The deck of cards consists of 40 cards, comprising 10 card figures for each of the 4 suits.

Function cria\_baralho() has no parameters:

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

# Shuffle the deck of cards

In any card game it is generally necessary to shuffle the cards.

Function baralha\_cartas(baralho, vezes) has as parameters a deck (a list of cards) and the number of times you want to shuffle the cards:

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** | You can now see why the deck was created as a list - so you can shuffle it!  Could a tuple also have been used?  Also, analyse the card exchange algorithm.  temp = baralho[i]  baralho[i] = baralho[j]  baralho[j] = temp  Why a local variable was used (temp) ? |

# Player interaction

When interacting with the player, it is essential to provide him with the necessary information to be able to make the move (output) and a simple interaction mode to act on the game (input).

## Output

For this purpose, we will create the function mostra\_mao () that visualizes the player's hand in text mode, but in an attractive configuration:

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 |

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

This function starts by sorting the list of cards by suit and figure, in order to facilitate the player's choice of the card to play.

For this it was necessary to create two character strings with sorted by suit and by figure. And a numero() function was created that returns an integer value for each card. This function will make it possible to compare the cards using the sort () method, which will be used to sort the lists of cards.

# mostra cartas

naipes = "OCPE"

figuras = "23456QJK7A"

def numero(t):

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

|  |  |
| --- | --- |
| **Head with gears** | To better understand the sorting with this function, calculate the value of the following cards:  (2, "O"), (3, "O"), (4, "O"), (4, "C"), (4, "P"), (4, "E")  And why is it necessary to use the function str() for the figure? |

Observe function 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 first the use of the method [sort()](mailto:https://www.w3schools.com/python/ref_list_sort.asp) on the list.  Reverse sorting (from largest to smallest) is done with the attribute reverse = True.  And as the elements of the list originate from a user-defined abstraction (Abstraction "Carta"), the function numero(), previously defined, is assigned to make the numerical comparison between two of these elements.  Also note the way the letters are drawn, with characters, and "line by line". |

## Input

This sub-problem focuses on the interaction with the "human" player. Firstly, the "player's hand" is shown, calling the function mostra\_mao (), so that he can choose his play.

Then, the card number (order number in the player's hand) that the player wants to play is read.

The joga\_jogador() function has as only one parameter mao, which is the list of cards (tuples) referring to the player's hand.

At the end, the function returns the selected card, erasing it from your hand. This function is therefore a modifier!

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** | In the condition of the while cycle, the number read was compared with the number of cards in the player's "hand". Why?  del was used to delete an item from the list. What other solution could you use? |

# Artificial Intelligence (IA)

Three of the players are controlled by the computer. Artificial Intelligence (AI) studies a set of techniques that allow you to program a computer to simulate "intelligent" behaviour. We are not going to explore these techniques in depth, but just design an algorithm that follows simple game rules, to provide an interesting challenge to the human player.

|  |  |
| --- | --- |
| **Head with gears** | The first step in creating the rules of the non-human player (NPC) is to think about how we play...  Simulate one or two rounds of this game and analyse how you decide which card to play ...  Check if that it is similar to the algorithm that is outlined below. |

The algorithm is defined with the following steps:

1. Check the suit of the move or whether it is the player who plays first.
2. In the latter case, select a random suit;
3. Select the highest card of the suit of the move;
4. If you have no card of the suit, try to "cut" the play with an trump;
5. If you also don't have an asset, play the first card in your hand.

Function joga\_NPC()has as parameters:

* cartas, which is a list of 4 cards referring to the current move;
* mao, which is the player's list of cards;
* naipe\_jogada**,** the suit of the move;
* naipe\_trunfo**,** the trump suit.

Analyze the function code**.**

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 | Computer controlled players are called "Non Player Character" and are often referred to by the acronym NPC.  The area of Artificial Intelligence (AI) develops several algorithms that promote "intelligent" behaviour, both by simulating the heuristics used by specialists, as well as through computational learning itself (Machine Learning).  These algorithms are often used in digital games to control NPCs.  **Challenge (optional):** Try to find a better algorithm for the joga\_NPC () function.  (This challenge has a high degree of difficulty.)  *Hint: The algorithm must consider who is winning (your team or the opponent) and proceed in order to optimize the team's points.*  *To do this, you must access the cards parameter, with the list of cards in play, which was not used in the presented algorithm..* |

# Jogo da Sueca

Finally, function sueca() implements the game, starting at startup, and then through the game loop.

The initialization creates a deck of cards, followed by the shuffling of the cards. Then the game's trump card is selected from the first card in the deck, and finally the 40 cards in the deck are divided into 4 10-card hands for the 4 players. Lists cartas\_equipaandcartas\_oponente serve to accumulate the result of each team in order to be able to count the score, at the end.

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 how the deck is segmented into 4 sets of 10 cards, which are the players' hands. |

The game loop is executed through a counted loop (for) since the game consists of exactly 10 moves.

# 4. faz jogo - 10 jogadas

jogador = 0 # comeca o jogador

for i in range(10):

The algorithm for each move is as follows:

1. Display the move number (jogada).

# visualizar informação da jogada

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

1. Each of the 4 players plays a card, sequentially, in order to complete a turn. The list of 4 cards of the play and the suit of the play is initialized, which will be defined by the player who plays first. Then, in a counted loop (for) with 4 iterations, each player's play is saved, each player's play is visualized and the play suit is updated from the first player's card.

# 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. After the four players have played, the card list is filled. First check if someone has played trump and, if so, the player with the highest card of the trump suit wins. If there are no trump cards, whoever has the highest card of the suit in the hand wins. Thus, the player variable is updated with the winner of the play, who will be the next to start the play.

# 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. At the end of each round, the cards from that round are added to the winning team, so that in the end it is possible to count the score.

# 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.")

After the end of the game loop, the score of each team is checked, indicating the winner (score greater than 60) or the tie (if both teams get 60 points).

# 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.")

Test the game with the following function call:

Sueca()

# Final challenge

One of the characteristics of digital games is that they have very intuitive user interfaces. Improve the functions of interaction with the human player, through a graphical interface, with the mouse, using the p5 module.

*Hint: You can use images of decks of cards that exist on the internet.   
To do this, simply use the function* [*image()*](mailto:https://p5.readthedocs.io/en/latest/examples/image/load%20and%20display%20image.html) *from p5.*

A high-level challenge, which is optionally proposed, is to improve the computer's AI algorithm.

|  |  |
| --- | --- |
| **Share with person** | Share o **your** game! |