 **Introdução à Programação**

Mestrado em Multimédia

# TP9. Card game “Sueca”

António Coelho

The main purpose of this tutorial is to work on data abstractions and reinforce the use of iteration for problem solving.

More specifically we will explore the following learning objectives:

* Use iteration to solve problems involving task repetition;
* Apply lists and tuples for problem solving.

It is recommended to read the following supporting texts before the tutorial:

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

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

# The card game “Sueca”

In this tutorial, we will develop a version of the “[Jogo da Sueca](https://en.wikipedia.org/wiki/Sueca_(card_game))” (Swedish game) where four players play in teams of two. In this version of the game one of the players will be the "human" and the other 3 will be controlled by the computer.

In this game, each player has a hand of 10 cards which is going to be drawn, card by card, over 10 turns, until players run out of cards to play. At the end of the game it is necessary to count the number of points of each team of 2 players through the cards won in the game (total of 120 points).

In this game cards 8, 9 and 10 are not used, so the deck has 40 cards in total.   
The rank of each card has the following number of points:

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

Note that ranks 2 through 6 have no value, while rank 7 is worth 10 points. The faces 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: the rank (one character or a number) and the suit (one character).

The suit is represented by the initial letter: “D” - Diamonds; "H" - Hearts; "S" - Spades; "C" - Clubs.

As an example, analyse the following player’s hand:

**hand = [(3, "H"),("A","H"), (2, "D"), ("K","C"), (7, "S")]**

# Game structure

This game has some complexity, so it will be useful to use decomposition to better structure the game into subproblems:

1. "Card" abstraction: functions to create cards and access the "rank" and "suit";
2. Rank of a card: returns the value (points) of a card;
3. Draw: Create a specific card from the deck.
4. Score: calculate the score of a set of cards;
5. Highest card: determine the highest card of a suit in a hand of cards;
6. Create the Deck of Cards: Create a list of 40 cards with all 10 ranks of the 4 suits;
7. Shuffle the cards: shuffle the deck of cards at random;
8. Player Interaction: Show the player's hand and ask the player to play a card;
9. Artificial Intelligence: The card played by other players, controlled by the computer.
10. The game: finally, the game algorithm ...

# “Card” abstraction

A card is represented by a tuple of two values: the rank and the suit.

**(rank, suit)**

Function **create\_card(rank, suite)** rerturns a tuple from two parameters: the rank and the suit. It is the constructor of abstraction “Card”.

# Card abstraction(tuple)

def create\_card(rank, suite):

# returns a tuple with both values

return (rank, suite)

To access the two values of a card, two selectors of the abstraction “Card” were created.:

def rank(card):

return card[0]

def suit(card):

return card[1]

# Score of a card

Function **points(rank)** returns an integer corresponding to the points of the rank of the card according to the table above. This function has only one parameter, the rank of the card, which can be either an integer or a character.

Review the code of the function **points(rank)**.

def points(rank):

if rank == "A":

return 11

elif rank == "Q":

return 2

elif rank == "J":

return 3

elif rank == "K":

return 4

elif rank == 7:

return 10

else:

return 0

# Draw a card

Function **draw\_card(value)** has only one parameter, the value of the card, and returns the corresponding rank number or face (character) according to the following table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Rank | “A” | 2 | 3 | 4 | 5 | 6 | 7 | “Q” | “J” | “K” |

Review the code of the function **draw\_card(value)**.

def draw\_card(value):

if value == 1:

return "A"

elif value == 8:

return "Q"

elif value == 9:

return "J"

elif value == 10:

return "K"

else:

return value

Determine the score of 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 **score(cards)** has only one parameter, which is a list of cards (tuples):

def score(cards):

score = 0

for card in cards:

score += points(rank(card))

**return score**

# Determine the highest card of a given suit

When a player has to assist a suite, they usually try to play the highest card (highest number of points) in their "hand" that corresponds to the suit of the play. The function **highest\_card (hand, suit\_played)** has two parameters:

* **hand**, which is a list of cards (tuples);
* **suit\_played**, which is the character corresponding to the suit being played (“H”, “D”, “C” or “S”).

This function returns only the number / face of the card. When there is no card of this suit returns **False**.

Review the code of the function **highest\_card (hand, suit\_played):**

**def highest\_card(hand, suit\_played):**

**max\_points = -1 # less than the minimum - cards 2 to 6 have 0 points**

**max\_card = ""**

**# the hand of the player**

**for card in hand:**

**if suit(card) == suit\_played:**

**if points(rank(card)) > max\_points:**

**max\_card = rank(card)**

**max\_points = points(card)**

**# if the player has no cards of that suit...**

**if max\_points >= 0:**

**return max\_card**

**else:**

**return False**

# Create the deck of cards

The deck of cards is composed by 40 cards, including 10 distinct cards for each of the 4 suits.

Function **create\_deck()** has no parameter:

**def create\_deck():**

**# create a list of 40 values (a deck of cards)**

**deck = list(range(40))**

**# define the 4 suits**

**suits = ["D", "H", "S", "C"]**

**# creates the deck of cards**

**i = 0**

**for suit in suits: # for each of the 4 suites**

**for c in range(10): # for each of the 10 cards of each suit**

**deck[i] = create\_card(draw\_card(c+1), suit) # creates the card**

**i += 1**

**return deck**

# Shuffle the deck of cards

In most card games, it is required to shuffle the deck of cards, introducing “chance”. This is accomplished by changing the position of the cards in the deck.

Function **shuffle\_cards(deck, times)** has as parameters the deck of cards and the number of times to shuffle. This function is a modifier and so it does not return any value.

**def shuffle\_cards(deck, times):**

**for i in range(times):**

**# randomly select the cards to change positions**

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

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

**# change card position**

**temp = deck[i]**

**deck[i] = deck[j]**

**deck[j] = temp**

# Interacting with the user

The User Interaction (UI) is a very important task is all softwares, and has it will influence the users’ experience (UX). In games it is fundamental.

Function **show\_hand(hand)** shows the hand of the player, displaying one card per line.

**def show\_hand(hand):**

**# Challenge: Improve interaction**

**print ("Your hand:")**

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

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

This function is quite simplistic and could be improved. Take this as a challenge!

Next, we need to ask the player for the card to play.

Function **joga\_jogador (cartas, mao)** has 2 parameters:

* **cards**, a list up to 4 cards (tuples) regarding the current cards being played;
* **hand**, the hand of the player (list of tuples);

At the end, this function returns the card selected by the player, removing it from the player’s hand:

**def play\_player (cards, hand):**

**show\_hand(hand)**

**n=0**

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

**ans = input ("Which card do you want to play? (1 to " + str(len(hand)) + "): ")**

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

**n = int(ans)**

**else:**

**n = 0**

**card = hand[n-1]**

**del hand[n-1]**

**return card**

# Artificial Intelligence

Three of the players are controlled by the computer. Developing a simple algorithm can enhance the experience for the human player.

The algorithm goes through the following steps:

1. Check the suit of the play or if it is the player who plays first. In this case selects a random card;
2. Select the highest card of the suit of the play;
3. If you do not have any card of that suit try to “cut” the hand with a trump card;
4. If you don't have a trump card either, play the first card of your hand.

Function **play\_NPC(cards, hand, suit\_played, trump\_suit)** has the following parameters:

* **cards**, which is a list of 4 cards (tuples) for the play;
* **hand**, which is the player's list of cards (tuples) – the hand;
* **suit\_played,** the suit of the play;
* **trump\_suit,** the trump suit.

Analyse the code of the function **joga\_NPC(cartas, mao, naipe\_jogada, naipe\_trunfo)**:

**def play\_NPC (cards, hand, suit\_played, trump\_suit):**

**# verifies the trump suit**

**if trump\_suit == "":**

**trump\_suit = random.choice(["D", "H", "S", "C"]) # random suit... Improve...**

**# Has to assist the suit**

**rank\_to\_play = highest\_card(hand, suit\_played)**

**if rank\_to\_play != False:**

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

**if rank(hand[i]) == rank\_to\_play and suit(hand[i]) == suit\_played:**

**del hand[i]**

**break**

**return create\_card(rank\_to\_play, suit\_played)**

**else:**

**# if this player does not have cards of that suit**

**play\_trump = highest\_card(hand, trump\_suit)**

**if play\_trump != False:**

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

**if rank(hand[i]) == play\_trump and suit(hand[i]) == trump\_suit:**

**del hand[i]**

**break**

**return create\_card(play\_trump, trump\_suit)**

**else:**

**# returns the first card of the hand... Challenge: Improve...**

**card = hand[0]**

**del hand[0]**

**return card**

|  |  |
| --- | --- |
| 🗬 | *Computer-controlled players are referred to as “Non Player Character” and are often referred to as NPC.*  *The area of Artificial Intelligence (AI) develops several algorithms that promote “intelligent” behaviour, both by simulating heuristics used by specialists, as well as through Machine Learning itself.*  *These algorithms are often used in digital games for NPC control.*  *Challenge (optional): Try to find a better algorithm for the* **play\_NPC** *function. (this challenge has a high degree of difficulty)*  *Hint: The algorithm must consider who is winning and proceed in order to optimize the team points.* |

# The card game “Sueca”

Finally, function **sueca()** develops the game algorithm, starting with the setup, and then through the game loop.

The setup of the game creates a deck of cards and shuffles the cards. Then the game's trump card is selected from the first card in the deck and finally the 40 deck cards are distributed to the 4 players, as a 10 card hand. The two lists **cards\_player\_team** and **cards\_opponents** accumulate the results of each team so that the final score can be calculated and the winner team identified.

**def sueca():**

**# 1. creates the deck of cards**

**deck = create\_deck()**

**# 2. shuffles the cards (ex. 100 times)**

**shuffle\_cards(deck, 100)**

**# shose the trump suit**

**trump\_suit = suit(deck[0])**

**# 3. creates the hands of the 4 players (the first player is the human)**

**game = [0, 0, 0, 0]**

**for i in range(4):**

**game[i] = deck[i\*10 : i\*10 + 10]**

**cards\_player\_team = []**

**cards\_opponents = []**

The game loop runs through a counted (**for**) cycle since the game consists of exactly 10 turns.

**# 4. Play the game - 10 rounds**

**player = 0 # the player**

**inicial\_player = 0 # the player that starts each round**

**for i in range(10):**

The algorithm of each turn is as follows:

1. Visualization of each turn.

**# show the game**

**print("\n\*\* Play " + str(i+1) + " (the trump suit is " + trump\_suit + ") \*\* ")**

1. Each of the 4 players plays a card sequentially to complete each turn. The list of 4 cards and the suit of the play, which will be defined by the player who plays first, is initialized. Then, in a 4-iterated counted (for) cycle, each player's play is saved, each player's play is displayed and the play's 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(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. After the four players make their move, the card list is completed. First it is required to check to see if someone played trump and, if so, wins the player with the highest card in the trump suit. If there are no trump cards, the one with the highest card of the played suit wins. This updates the player variable with the winner of the play, which will be the next to start to play in the next turn.

**# the 4 players choose the card to play**

**cards = ["", "", "", ""]**

**suit\_played = "" # indicates that the first player to start chooses the suit**

**for j in range (4):**

**# human or computer?**

**if player == 0: # human player**

**cards[player] = play\_player(cards, game[player])**

**else:**

**cards[player] = play\_NPC(cards, game[player], suit\_played, trump\_suit)**

**print ("Player " + str(player+1) + " played : " + str(cards[player]))**

**# updates the suit to play and the player to play next**

**if suit\_played == "":**

**suit\_played = suit(cards[player])**

**player = (player + 1) % 4 # player to the right**

1. At the end of each turn, the cards are added to the winner's team, so that at the end it is possible to count the score.

**# verifies who wins the play and adds the cards to the corresponding team**

**card\_trump\_suit = highest\_card(cards, trump\_suit)**

**if card\_trump\_suit != False:**

**# wins who has the highest trump card**

**for j in range(len(cards)):**

**if rank(cards[j]) == card\_trump\_suit and suit(cards[j])==trump\_suit:**

**player = j**

**break**

**else:**

**# highest card of the suit played**

**played\_card = highest\_card(cards, suit\_played)**

**for j in range(len(cards)):**

**if rank(cards[j]) == played\_card and suit(cards[j])== suit\_played:**

**player = j**

**break**

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

**if score(cards\_player\_team) > 60:**

**print("You win! " + str(score(cards\_player\_team)) + " points. Congratulations!")**

**elif score(cards\_opponents) > 60:**

**print("You lose! The other team scored " + str(score(cards\_opponents)) + " points.")**

**else:**

**print("It's a draw - 60 points for each team.")**

Test the game with the following function call:

**Sueca()**

# Final challenge

One of the features of digital games is that they have very intuitive user interfaces. Improve the player information displayed to enhance interaction.

An optional high-level challenge is to improve the computer's IA algorithm…

# Bibliography

* 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/>