Coursework of Foundations of Computing and Technology, Programming Strand: The Sueca card game

1 The Sueca card game

The coursework of Foundations of Computing and Technology, programming strand (FCTP), explores the card game *Sueca*, popular in the Portuguese speaking world[1].

A tricks-based card game, Sueca is played by 4 players who form two pairs competing against each other. Sueca is usually played at a squared table; the players that form a pair sit opposite from each other (Fig. 1). The aim is to win tricks containing valuable cards which are worth points; the pair that makes most points wins.

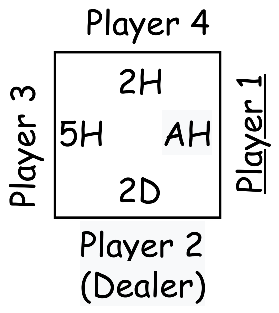

Figure 1: Arrangement of players in a Sueca game, played clockwise, and a Sueca trick started by player 1 (underlined).

A sueca card deck comprises 40 cards. Each card has a suit, either clubs, diamonds, hearts, or spades (denoted C, D, H, and S respectively), and a rank, either 2, 3, 4, 5, 6, queen, jack, king, 7, ace (denoted as 2, 3, 4, 5, 6, Q, J, K, 7, A, in the input data respectively) whose face value varies and is precisely in this ascending order; 2 is the lowest rank, the *ace* is the highest. There are some peculiarities: the 7 (*manilha* in Portuguese) is the second highest rank and the Jack is more valuable than the Queen.

Cards are worth points based on their ranks as per table 1a. The two pairs compete for the 120 points at stake in the 10 rounds of each game. The deck is shuffled and distributed equally; each player gets 10 cards. The suit of the last card of the dealer player is the *trump*. The player to the right of the dealer (next player, anti-clockwise) is player 1 who starts the game and is paired with player 3 competing against the pair formed by players 2 (the dealer) and 4, as illustrated in Fig. 1.

The rule of sueca are as follows:

1. The game is played either clockwise or anti-clockwise. Figure. 1 highlights a trick played clockwise, the setting considered for the implementation (see below).

2. Players take turns playing cards, following the lead suit whenever possible.

[1]Sueca means *Swedish* in Portuguese, but this popular team-based game has nothing to do with Sweden. You may know more about it by reading the games's Wikipedia's entry.

Ace	11
7	10
King	4
Jack	3
Queen	2
All others	0

(a) Ranks and points in sueca.

Trump Card:			7D
AH	2D	5H	2H
AC	3D	4C	KC
AS	2S	4S	3S
AD	4H	QD	6D
4D	7S	3H	JD
7D	5D	7C	3C
KS	KD	QS	5S
7H	QC	QH	2C
JH	JC	KH	6C
5C	6S	6H	JS
Score:		76 – 44	

(b) A sueca game with their trump *seven of diamonds* and its ten rounds

Trump Card:			7D
AH	2D	5H	2H
AC	3D	4C	KC
AS	2S	4S	6D
3S	6H	7S	5S
7C	3C	2C	4D
AD	4H	QD	JD
5D	QC	3H	7D
6C	KD	JC	5C
7H	QS	KH	6S
JH	JS	QH	KS

(c) A sueca game with an *illegal cut* in round 3.

Table 1: Distribution of points in Sueca, and one legal and one illegal sample game. Each trick in the sample games above comprises four cards in the order in which the cards were played.

3. Players may play any card when unable to follow the lead suit. If they play a trump in response to a non-trump lead suit, then the move is called a *cut* (*corte* in Portuguese) as the trump cuts the lead suit — any trump card is more valuable in a trick than any other card of other suits, even if their value in points is 0.

4. Not following the lead suit when holding such cards is illegal and considered an act of cheating, which, if detected, results in a victory by 120 points to the opposing pair. Table 1c illustrates cheating — in round 3, player 2 illegally threw 6D onto the table, in response to the lead card AS with spades in his hand.

5. The player who wins the current trick starts the next round. A trick is won by the highest card with respect to the lead suit and the trump; a trump is always higher than any other non-trump card; when a trick includes more then one trump, then the usual ranking on trumps applies (the highest-ranked trump wins).

Figure 1 illustrates the first round of the Sueca game described in table 1b:

- The dealer pulled out the *7 of diamonds* as the trump.
- Player 1 starts the game by throwing the ace of hearts (AH). As the game is played clockwise, player 2 plays next; unable to follow the lead suit, players does a cut by throwing the 2 of diamonds (2D), a trump; player 3 plays the five of hearts (5H) and player 4 the 2 of Hearts (2H). The trick's highest card is the two of diamonds, the trump thrown by player 2 who wins the round, bringing a total of 11 points to pair B.

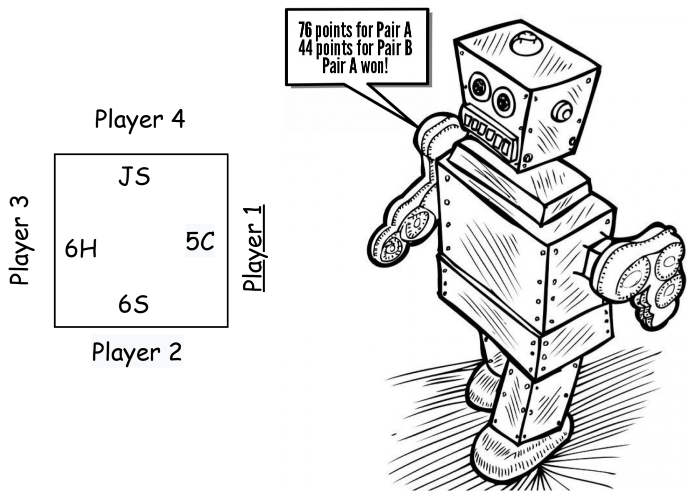

Figure 2: Verdict of envisaged robot referee and scorer in last round of the sueca game of table 1b

- Player 2 starts the next round with the ace of clubs (AC); the next player, player 3, plays the *three of diamonds* (3D), which is followed by the four of clubs (player 4) and the King of clubs (player 1); the highest card is 3D, a trump, played by player 3, bringing a total of 15 points to pair A. Player 3 starts the next round, and the games proceeds as described in table 1b.
- When 10 rounds have been played, after all cards in players' hands have been thrown into the table, the game is over; points are counted from the tricks won by each team, using the ranks to points mapping of table 1a. The pair with most points wins. The game of table 1b was won by the first pair (pair A) with 76 points against 44.

2 The assignment

FCTP's coursework assignment intends to contribute to a robot referee which observes sueca games as they are played by people, and even robot players, checks that the games conform to the rules and then says who won and the final score when the game finishes, as illustrated in Fig.2. This is done to automate scoring and tackling cheating in sueca — sueca games can become competitive at times and pairs often cheat by resorting to illegal *cuts*. The FCTP coursework assignment will contribute to the first stage of this project. It aims to build a python implementation, called *sueca_scorer*, which takes sueca games described in text files and says which pair won along with the points awarded by each pair, and identifies situations of invalidity with the given sueca games.

The scorer may be called from the command line. For the game described in table 1b encoded in the file 'game1.sueca' we would have:

```
$ python sueca_scorer.py game1.sueca
Pair A won the given sueca game.
Score: 76 | 44
```

Option '-c' is provided to show, in addition to the final result, the cards held by the players in the given game:

```
$ python sueca_scorer.py -c game1.sueca
Pair A won the given sueca game.
Score: 76 | 44
Player's cards in the sueca game
Player 1: AH, KC, 4S, QD, 3H, 3C, 5S, QH, KH, 5C
Player 2: 2D, AC, 3S, 6D, JD, 7D, KS, 2C, 6C, 6S
Player 3: 5H, 3D, AS, AD, 4D, 5D, KD, 7H, JH, 6H
Player 4: 2H, 4C, 2S, 4H, 7S, 7C, QS, QC, JC, JS
```

Option '-g' shows the tricks of the given game in the order in which the cards were played:

```
$ python sueca_scorer.py -g game1.sueca
Pair A won the given sueca game.
Score: 76 | 44
Trump: 7D | Diamonds
1: AH 2D 5H 2H
2: AC 3D 4C KC
3: AS 2S 4S 3S
4: AD 4H QD 6D
5: 4D 7S 3H JD
6: 7D 5D 7C 3C
7: KS KD QS 5S
8: 7H QC QH 2C
9: JH JC KH 6C
10: 5C 6S 6H JS
```

Trump Card:			5S
2D	4D	3D	AD
2C	5C	4C	AC
AH	2H	7H	5H
7C	JC	3C	QS
7D	KD	JD	5D
6H	3H	KH	2S
AS	5S	JS	4S
6D	QD	3S	7S
QC	KC	6S	KS
4H	QH	6C	JH
Score:		60 – 60	

Table 2: A sueca game which results in a draw.

Games may draw, as illustrated in the game of table 2. In such cases, an appropriate message, declaring such games as draws, should be outputted by sueca_scorer:

```
$ python sueca_scorer.py game2.sueca
The game resulted in a draw.
Score: 60 | 60
```

The user is informed when the following erroneous situations occur: (i) unrecognised option, (ii) missing sueca game file, or (iii) inexistent game file:

```
$ python sueca_scorer.py -d game1.sueca
Invalid option -d
```

```
Usage: python sueca_scorer.py [-c | -g] <game_file>
$ python sueca_scorer.py
A game file is missing
Usage: python sueca_scorer.py [-c | -g] <game_file>
$ python sueca_scorer.py game9.sueca
Could not find the game file 'game9.sueca'
Usage: python sueca_scorer.py [-c | -g] <game_file>
```

sueca_scorer should be able to detect invalid games. For example, an invalid game may be created by modifying the game of table 1b, declaring the 4 of diamonds as the trump card in a file 'game1a.sueca':

```
$ python sueca_scorer.py game1a.sueca
The game is invalid
Player 2 (dealer) does not hold trump card 4D
```

All cards of a sueca game should belong to the 40 card deck. For instance, a variant with an invalid card may be obtained by modifying the 4 of spades (4S) of the third round of the game of table 1b to an 8 of spades (8S):

```
$ python sueca_scorer.py game1b.sueca
The game is invalid
Card '8S' is invalid!
Invalid rank symbol: 8 (line 4 of game file)
```

The implementation should detect cheating. From the game of table 1b, a cheating situation can be created if, in round 3, player 2 plays 6D instead of 3S, not adhering to the obligation of following the lead suit, which could lead to the game of table 1c:

```
$ python sueca_scorer.py game1c.sueca
The game is invalid
Player 2 played illegal card 6D in round 3 with respect to lead suit Spades
Pair A won the given sueca game.
Score: 120 | 0
```

Sueca tricks should not include cards which have already been played. From the game of table 1b, we can create such an invalid situation by changing card AS in round 3 to AC, already played in round 2, detectable by sueca_scorer:

```
$ python sueca_scorer.py game1d.sueca
The game is invalid
Card AC of round 3 has already been played in round 2
```

Valid Sueca games must contain 10 tricks or rounds. An invalid game may be created by removing the last two rounds from the game of table 1b, an invalid situation which should be detected by sueca_scorer:

```
$ python sueca_scorer.py game1e.sueca
The game is invalid
Game file 'game1e.sueca' is incomplete. A complete game takes 10 rounds;
the given game includes 8 rounds only.
```

3 The implementation

The following describes aspects of the sueca_scorer implementation which submissions are required to comply to. Deviations, with respect to the definitions that follow, from the case-sensitive names of classes, functions and exceptions, as well as return types of functions, are to be considered incorrections when it comes to marking.

3.1 Suits and Ranks

Suits and ranks of cards are to be represented as strings. A suit is either one of the following single character strings: “H” (Hearts), “C” (Clubs), “S” (spades) or “D” (diamonds). Ranks are also single character strings: “A” (Ace), “2” (two), “3” (three), “4” (four), “5” (five), “6” (six) and “7” (seven or manilha), “Q” (Queen), “J” (Jack) and “K” (King). Python definitions dealing with suits and ranks should be placed in file ‘sueca_suits_ranks.py’; this includes all functions, and their auxiliary definitions, that follow.

The following functions are required:

- $valid\_suit(s)$ and $valid\_rank(r)$ take a string and return a boolean indicating whether the given string is either a valid suit or rank string, respectively, in accordance with the string-based representations mentioned in the previous paragraph.
- Because suits are represented in the implementation in their abbreviated forms (as either “H”, “C”, “S” or “D”), function $suit\_full\_name(s)$ is used to yield the full name of any given suit s — either “Hearts”, “Clubs”, “Spades” or “Diamonds”. Exception $ValueError$ should be raised to indicate that the given suit is invalid – e.g. $suit\_full\_name(''P'')$ should raise “ValueError” with the message “invalid suit symbol P”.
- $rank\_points(r)$ gives the points associated with any given rank r as per the table 1a.
- $rank\_higher\_than(r1, r2)$ checks whether any rank $r1$ is higher than another rank $r2$. This captures the total order that exists between the different ranks as for any two ranks $r1$ and $r2$ $rank\_higher\_than(r1, r2)$ is either true or false. This order reflects the points associated with the different ranks (as per table 1a), but also the implicit order that exists between the cards that are worth 0 points — for instance, 3 is higher than 2, and 4 is higher than 3 and so on.
- The last two functions should raise the exception $ValueError$ to indicate that the given rank is invalid – e.g. calling these functions on the string $''8''$ should raise $ValueError$ with the message “invalid rank symbol 8”.

From an implementation of the definitions above, it should be possible to recreate the following interaction in a python console session in a python console session loaded with the module ‘sueca_suits_ranks.py’:

```
>>> valid_suit("C")
True
>>> valid_suit("S")
True
>>> valid_suit("P")
False
>>> valid_rank("3")
True
>>> valid_rank("7")
True
>>> valid_rank("8")
False
>> rank_points("A")
11
>> rank_points("7")
10
>> rank_points("5")
0
>>> rank_points("9")
ValueError: Invalid rank symbol: 9
>>> rank_higher_than("3","2")
True
>>> rank_higher_than("6","3")
True
>>> rank_higher_than("K","Q")
True
>>> rank_higher_than("J","7")
False
>>> rank_higher_than("4","4")
False
>>> rank_higher_than("8","7")
ValueError: Invalid rank symbol: 8
```

3.2 Cards

Cards have a two-character string representation made-up of a rank and a suit (e.g. “KS” denotes king of spades). Class *Card* represents Sueca cards. Card-related Python definitions are to be placed in file called ‘sueca_cards.py’, which should include the definitions that follow.

The following function is required:

- *parseCard*(*cs*) (not a function of class *Card*) takes a card represented as a string *cs* and returns an object of class *Card*. Custom exception *CardInvalid* is raised if *cs* is not a valid string representation of a sueca card (e.g. "9D" would produce a *CardInvalid* exception).

Class *Card* should include the following:

- The required constructor.
- Function *points*(*self*) returns the points associated with the current card *self*, the points of the card's rank.
- Function *higher_than*(*self*, *other*, *s*, *t*) says whether the *Card* instance corresponding to *self* is higher than the *Card* instance corresponding to *other* with respect to the lead suit *s* and the trump suit *t*.
- Function *show*(*self*) gives the string representation of the card (e.g. "7C").

From an implementation of the definitions above, it should be possible to recreate the following interaction in a python console session loaded with the module 'sueca_cards.py':

```
>>> parseCard("2C").show()
'2C'
>>> parseCard("8C").show()
sueca_cards.CardInvalid: Card '8C' is invalid!
Invalid rank symbol: 8
>>> parseCard("QSD").show()
sueca_cards.CardInvalid: Card 'QSD' is invalid!
A card string representation must contain 2 characters only
>>> parseCard("2C").points()
'2C'
>>> parseCard("KS").points()
4
>>> parseCard("9D").points()
sueca_cards.CardInvalid: Card '9D' is invalid!
Invalid rank symbol: 9
>>> parseCard("KS").higher_than(parseCard("2C"), "S", "D")
True
>>> parseCard("KS").higher_than(parseCard("JS"), "S", "D")
True
>>> parseCard("KS").higher_than(parseCard("2D"), "S", "D")
False
>>> parseCard("7S").higher_than(parseCard("2C"), "C", "D")
False
>>> parseCard("7Q").higher_than(parseCard("2C"), "C", "D")
sueca_cards.CardInvalid: Card '7Q' is invalid!
Invalid suit symbol: Q
```

3.3 Tricks

Class *Trick* represents tricks. A trick's string representation is made up of the trick's cards represented as strings separated by spaces (e.g. "AH 2D 5H 2H"). All python definitions which concern tricks should be placed in file 'sueca_tricks.py'.

The following functions are required:

- *parseTrick*(*ts*) takes a trick represented as a string and returns an object of class *Trick*. Exception *ValueError* is to be raised if the given string *ts* is not a valid representation of a sueca trick, which must comprise four valid sueca cards.
- *parseGameFile*(*fname*) takes the name of a text file, similarly in form to the games of tables 1b and 1c[2], and returns the parsed trump card and list of tricks.

Class *Trick* should include the following:

- The required constructor.
- Function *points*(*self*) gives the points of the current trick.
- Function *trick_winner*(*t*) takes a trump suit *t* and yields the trick's winning player, a number between 1 and 4, in the order in which the cards of the trick were played. For example, the trick "AH 2D 5H 2H" is won by the player who played 2D if the trump is diamonds, hence, in this case, 2 should be returned.
- Function *show*(*self*) gives the trick's string representation (e.g. "7C JC 6C 2S").

From an implementation of the definitions above, it should be possible to recreate the following interaction in a python console session loaded with the module 'sueca_tricks.py':

```
>>> parseTrick("AH 2D 5H 2H").show()
'AH 2D 5H 2H'
>>> parseTrick("AH 2D 5H 8H").show()
sueca_cards.CardInvalid: Card '8H' is invalid!
Invalid rank symbol: 8
>>> parseTrick("AS 2S 7S JS 5S").show()
ValueError: A trick string must comprise four cards only; the given trick
is: AS 2S 7S JS 5S
>>> parseTrick("AH 2D 5H 2H").points()
11
>>> parseTrick("AS 2S 7S JS").points()
24
>>> parseTrick("AH 2D 5H 2H").trick_winner("D")
2
>>> parseTrick("AS 2S 7S JS").trick_winner("D")
```

[2]Sample game files can be found in the sample games zip file, available on FCTP's NOW learning room as supplementary coursework materials.

```
1
>>> parseTrick("5C 6S 6H JS").trick_winner("D")
1
>>> tc, ts = parseGameFile('game1.sueca')
>>> tc.show()
'7D'
>>> ts[0].show()
'AH 2D 5H 2H'
>>> ts[1].show()
'AC 3D 4C KC'
>>> ts[-1].show()
'5C 6S 6H JS'
```

Above, file 'game1.sueca' is the game file corresponding to the game of table 1b, which is given in the zip file with the sample games available from NOW.

3.4 Games

Class *Game* represents sueca games with a trump card and their 10 tricks. Definitions which concern sueca games should be placed in the file 'sueca_games.py'.

Class *Game* should include the following:

- The required constructor.
- Function *gameTrump*(*self*) returns the game's trump card, an instance of class *Card* above.
- Function *score*(*self*) returns a pair with the points won by each pair in the current game.
- Function *playTrick*(*self*, *t*) adds the given trick *t* (an instance of class *Trick* above) to the current game. The following exceptions should be raised: (i) *CardAlreadyPlayed* if a trick contains a card played in a previous round of the game; (ii) *DealerDoesNotHoldTrumpCard* if player 2 (the dealer) did not actually hold the game's trump card; and (iii) *IllegalCardPlayed* if a card played in some round is illegal with respect to the lead suit, which caters for the illegal cuts problem illustrated in the game of table 1c.
- Function *cardsOf*(*self*, *p*) returns a list of cards (instances of class *Card* above) held by the player *p* (a number from 1 to 4) in the current game. Exception "ValueError" should be raised if the given player number is invalid.
- Function *gameTricks*(*self*) returns a list with the tricks (instances of class *Trick* above) played in the game.

From an implementation of the definitions above, it should be possible to recreate the following interaction in a python console session loaded with the module 'sueca_games.py':

```
>>> tc, ts = parseGameFile('game1.sueca')
>>> g = Game(tc)
>>> g.gameTrump().show()
'7D'
>>> g.playTrick(ts[0])
>>> g.score()
(0, 11)
>>> g.cardsOf(1)[0].show()
'AH'
>>> g.cardsOf(2)[0].show()
'2D'
>>> g.cardsOf(5)[0].show()
ValueError: Invalid player number 5
>>> g.gameTricks()[0].show()
'AH 2D 5H 2H'
>>> g.playTrick(ts[1])
>>> g.score()
(15, 11)
>>> for t in ts[2:]:
...    g.playTrick(t)
...
>>> g.score()
(76, 44)
>>> g.gameTricks()[-1].show()
'5C 6S 6H JS'
>>> g.cardsOf(1)[-1].show()
'5C'
>>> g.cardsOf(2)[-1].show()
'6S'
```

3.5 Command line interface and outputs

File ‘sueca_scorer.py’ will act as the main (or controller) module and deal with the command line interface and outputs described in section 2.

The following function is required:

- $runGame(fname, showCards, showGame)$ takes the name of a game file, and two booleans, which are false by default, indicating whether information concerning the player’s cards or the games’s tricks (see section 2 for further information), respectively, are to be shown on the console. This function parses the game file and runs the game, indicating score and winning pair (see section 2 for further information). The following exceptions are raised: (i) $GameFileCouldNotBeFound$, if the given game file is not found; and (ii) $SuecaGameIncomplete$ if the given game file includes less than the required 10 rounds for a complete sueca game.

From an implementation of the definitions above, it should be possible to establish the following in a python console session loaded with the module 'sueca_scorer.py':

```
>>> runGame('game1.sueca')
Pair A won the given sueca game.
Score: 76 - 44
>>> runGame('game1.sueca', showCards = True)
Pair A won the given sueca game.
Score: 76 - 44
Player's cards in the sueca game
Player 1: AH, KC, 4S, QD, 3H, 3C, 5S, QH, KH, 5C
Player 2: 2D, AC, 3S, 6D, JD, 7D, KS, 2C, 6C, 6S
Player 3: 5H, 3D, AS, AD, 4D, 5D, KD, 7H, JH, 6H
Player 4: 2H, 4C, 2S, 4H, 7S, 7C, QS, QC, JC, JS
>>> runGame('game1.sueca', showGame = True)
Pair A won the given sueca game.
Score: 76 - 44
Trump: 7D - Diamonds
1: AH 2D 5H 2H
2: AC 3D 4C KC
3: AS 2S 4S 3S
4: AD 4H QD 6D
5: 4D 7S 3H JD
6: 7D 5D 7C 3C
7: KS KD QS 5S
8: 7H QC QH 2C
9: JH JC KH 6C
10: 5C 6S 6H JS
>>> runGame('game2.sueca')
The game resulted in a draw
Score: 60 - 60
>>> runGame('game9.sueca')
sueca_scorer.GameFileCouldNotBeFound: Could not find the game file 'game9.sueca'
>>> runGame('game1a.sueca')
sueca_games.DealerDoesNotHoldTrumpCard: Player 2 (dealer) does not hold trump card 4D
>>> runGame('game1c.sueca')
The game is invalid
Player 2 played illegal card 6D in round 3 with respect to lead suit Spades
Pair A won the given sueca game.
Score: 120 | 0
```

Section 2 describes the outputs of the command-line interface.

4 How to go about it

sueca_scorer should be developed incrementally in a bottom-up fashion. starting from the module ‘sueca_suits_ranks.py’ (section 3.1). This should be followed by ‘sueca_cards.py’ (section 3.2), ‘sueca_tricks.py’ (section 3.3), ‘sueca_games.py’ (section 3.4) and ‘sueca_scorer.py (section 3.5), as each of these modules are built on top of one another in this specific order. Modules should be tested as they are developed using the console-based tests provided for each section.

5 The Submission

The submission should include the developed python sueca_scorer code made-up of the five files mentioned above. Submissions may be undertaken by either individual students or students working in pairs. By default, submissions are individual. Submission pairs must be registered and are subject to vetting by the module team. All developed files are to be submitted according to the instructions given on NOW.