Computer Science

COMPSCI 130 S2
Assignment TWO

Due: **11 pm Monday 28th October 2019**
Worth: **10% of the final mark**

Introduction

Solitaire (is also known as Klondike) is a game which is played with a standard 52-card deck, without Jokers. An example is shown in Figure 1.

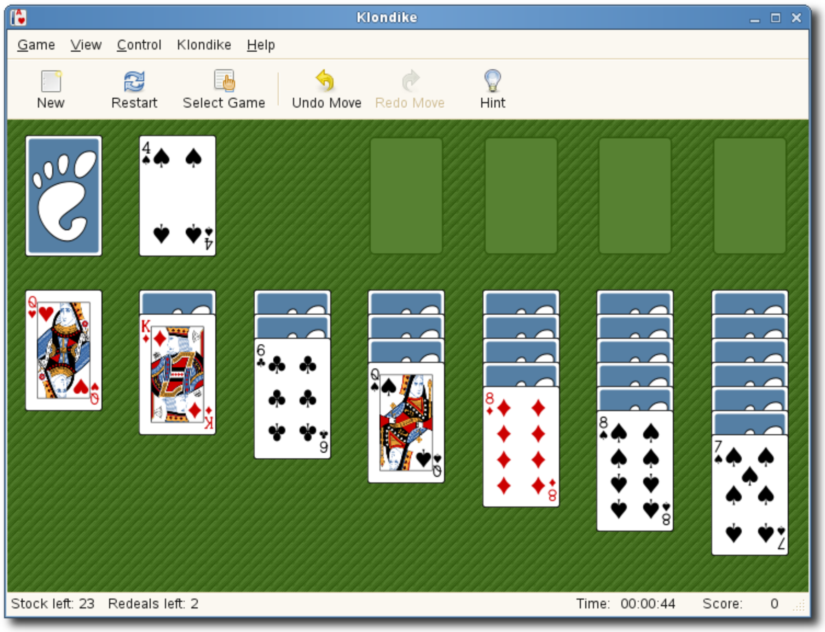

Figure 1 A GNOME version of Klondike [1]

After shuffling, seven piles of cards are laid from left to right. Each pile begins with one upturned card. From left to right, each pile contains one more card than the last. The first and left-most pile contains a single upturned card, the second pile contains two cards (one downturned, one upturned), the third contains three (two downturned, one upturned), and so on, until the seventh pile which contains seven cards (six downturned, one upturned). The piles should look like the figure to the right at the beginning of every game.

The four foundations (light rectangles in the upper right of Figure 1) are built up by suit from Ace (low in this game) to King, and the tableau piles can be built down by alternate colors, and partial or complete piles can be moved if they are built down by alternate colors also. Any empty piles can be filled with a King or a pile of cards with a King. The aim of the game is to build up a stack of cards starting with two and ending with King, all of the same suit. Once this is accomplished, the goal is to move this to a foundation, where the player has previously placed the Ace of that suit. Once the player has done this, they will have "finished" that suit, the goal being to finish all suits, at which time the player would have won (see https://en.wikipedia.org/wiki/Klondike_%28solitaire%29).

In this assignment, you are going to implement this game with simplified game rule using Python.

Warning

DO NOT SUBMIT SOMEONE ELSE'S WORK:

- The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help.
- Under no circumstances should you take or pay for an electronic copy of someone else's work. This will be penalized heavily.
- Under no circumstances should you give a copy of your work to someone else
- The Computer Science department uses copy detection tools on the files you submit. If you copy from someone else, or allow someone else to copy from you, this copying will be detected and disciplinary action will be taken.
- To ensure you are not identified as cheating you should follow these points:
  - Always do individual assignments by yourself.
  - Never give another person your code.
  - Never put your code in a public place (e.g., forum, your web site).
  - Never leave your computer unattended. You are responsible for the security of your account.
  - Ensure you always remove your USB flash drive from the computer before you log off.

Your name, UPI and other notes

- All files should also include your name and ID in a comment at the beginning of the file.
- All your files should be able to be compiled without requiring any editing.
- All your files should include good layout structure

The simplified game rule

After shuffling, *n* piles of cards are laid from top to bottom. The (0) first pile begins with one upturned card and *n*-1 downturned cards. The other piles (the tableau piles) are empty are empty at the beginning. An example is shown below.

```
0: 6 * * * * * * * * *
1:
2:
3:
```

The tableau piles can be built down according to the numbers of the cards. Partial or complete tableau piles can be moved if they are built down in order. An example is shown below.

```
0: 2 * * * * * *
1: 5 4 3
2: 6
3:
```

to

```
0: 2 * * * * * *
1:
2: 6 5 4 3
3:
```

The empty pile can be filled with any card. If the player do not want to /cannot move the card of the first pile (0) to any other tableau piles, the player can get a second card and put the original one to the back of the first pile (0). An example is shown below.

```
0: 6 * * * * * * * * *
1:
2:
3:
```

to

```
0: 7 * * * * * * * * *
1:
2:
3:
```

or

```
0: 7 * * * * * * * *
1: 6
2:
3:
```

The aim of the game is to build up a stack of cards starting from card *n*-1 to card 0. In order to become familiar with the template program supplied. Your assignment is divided into several sections for ease of completion. Please complete the assignment in order of the sections.

Section 1: The class Deque (20 marks)

The class **Deque** is used to implement a pile of cards. The summary of class methods are shown below:

```
class Deque:
    def __init__(self):
    def add_front(self, item):
    def add_rear(self, item):
    def remove_front(self):
    def remove_rear(self):
    def size(self):
    def peek(self):
    def peeklast(self):
    def printall(self, index):
```

def __init__(self):
You should use a python **List** data structure to implement the deque. The list's name should be **items**.

def add_front(self, item):
This method adds a new item to the deque (at list[**SIZE**-1])

def add_rear(self, item):
This method adds a new item to the deque (at list[0])

def remove_front(self):
This method removes and returns an item from the deque (at list[**SIZE**-1])

def remove_rear(self):
This method removes and returns an item from the deque (at list[0])

def size(self):
This method returns the size of the deque

def peek(self):
This method returns the front item of the deque (at list[**SIZE**-1])

def peeklast(self):
This method returns the rear item of the deque (at list[0])

def printall(self, index):
This method prints out all the items of the deque with the order beginning from the rear item. The items should be separated by ' '. However, if **index** is **0**, all the items besides the first one (the rear item at list[0]) should be hidden by '*'.

For example: if q.item is [4, 5, 6, 7, 8], the expected output for "q.printall(1)" should be "4 5 6 7 8" and the expected output for "q.printall(0)" should be "4 * * * *"

In section 1, you are going to complete the implementation of this Deque class.

You can test your code at coderunner (assignment 2, question 1).

Section 2: display(self) of class Solitaire (10 marks)

You are going to implement the **display** function for the game solitaire. The class **Solitaire** should include the **__init__(self, ncards)** and **def display(self)**. You can write the function **__init__(self, ncards)** as below.

```
class Solitaire:
  def __init__(self, ncards):
    self.t = []
    self.__CardNo = len(ncards)
    self.__ColNo = (self.__CardNo // 8) + 3
    self.__ChanceNo = self.__CardNo * 2
    for i in range(self.__ColNo):
      self.t.append(Deque())
    for i in range(self.__CardNo):
      self.t[0].add_front(ncards[i])

  def display(self):
    ......
```

def display(self) is going to display the game layout. The game should have a number (**self.__ColNo)** of card piles. Each card pile is stored as a **deque.** If **self.__ColNo** is 4, 4 rows of numbers should be printed out. However, the first row will display the first item only, the others are hidden by '*'. Thus, you can implement this function by **self.t[i].printall(i)**. Each line should start with the pile number, then ':'. The last is the list items.

An example is shown below:

deque 0 is [6, 5, 4]
deque 1 is [3, 2, 1, 0]
deque 2 is [14, 13, 12, 11]
deque 3 is [10, 9, 8, 7]

The display should be:

```
0: 6 * *
1: 3 2 1 0
2: 14 13 12 11
3: 10 9 8 7
```

In section 2, you should implement the full program including both the classes **Solitaire** and **Deque**.

You can test your code at coderunner (assignment 2, question 2).

Section 3: move(self, c1, c2) of class Solitaire (15 marks)

You are going to implement the **move** function for the game solitaire. The class **Solitaire** should include the **__init__(self, ncards), def display(self)** and **move(self, c1, c2)**.

The function **move(self, c1, c2)** moves the card from a pile to another pile. There are three types of moves.

Codition 1 (c1 = c2 = 0): move a card from the top of the first pile (rear of the list) to the bottom of the first pile (front of the list). This move is always valid.

```
0: 6 2 1
1: 3 2 1 0

After move(0, 0),
0: 2 1 6
1: 3 2 1 0
```

Codition 2 (c1 = 0, c2 > 0): move a card from the top of the first pile (rear of the list) to the bottom of any other pile c2 (front of the list). This move is valid only when the number of moving card (*N1*) is less than the number of the card (*N2*) at the bottom of the destination pile by one (i.e., *N2* = *N1* + 1).

```
0: 4 8
1: 7 6 5

After move(0, 1),
0: 8
1: 7 6 5 4
```

Codition 3 (c1 > 0, c2 > 0): move the whole pile of cards from c1 to the bottom of any other pile c2 (front of the list). This move is valid only when the number of card (*N1*) on the top of the moving pile is less than the number of the card (*N2*) at the bottom of the destination pile by one (i.e., *N2* = *N1* + 1).

```
0: 8
1: 7 6 5
2: 4 3 2

After move(2, 1),
0: 8
1: 7 6 5 4 3 2
2:
```

In section 3, you should implement the full program including both the classes **Solitaire** and **Deque**.

You can test your code at coderunner (assignment 2, question 3).

Section 4: class Solitaire – full program (30 marks)

You are going to implement the complete program of the game solitaire. The class **Solitaire** should include the following methods:

```
class Solitaire:
  def __init__(self, ncards):
    self.t = []
    self.__CardNo = len(ncards)
    self.__ColNo = (self.__CardNo // 8) + 3
    self.__ChanceNo = self.__CardNo * 2
    for i in range(self.__ColNo):
      self.t.append(Deque())
    for i in range(self.__CardNo):
      self.t[0].add_front(ncards[i])

  def display(self):

  def move(self, c1, c2):

  def IsComplete(self):

  def play(self):
    print("****************************************NEW
GAME**************************************")
    for game_iter in range(self.__ChanceNo):
      self.display()
      print("Round", game_iter+1, "out of", self.__ChanceNo, end = ": ")
      col1 = int(input("Move from row no.:"),10)
      print("Round", game_iter+1, "out of", self.__ChanceNo, end = ": ")
      col2 = int(input("Move to row no.:"),10)
      if col1 >= 0 and col2 >= 0 and col1 < self.__ColNo and col2 < self.__ColNo:
        self.move(col1, col2)
      if (self.IsComplete() == True):
        print("You Win in", game_iter+1, "steps!")
        break;
      else:
        if game_iter+1 == self.__ChanceNo:
          print("You Loss!")
    print()
```

You can write the **play(self)** function according to above provide code. You are going to implement the **IsComplete(self)** function to finish the whole program. This is the function to check whether the player can win the game. The player can win the game if all of below are satisfied:

1. No card on the first pile
2. All the cards are on the one of other piles
3. All the cards are in decreasing order (it should be always true if there is no illegal move)

This function returns **True** if the player win the game and **False** otherwise.

You can test your code at coderunner (assignment 2, question 4).

Section 5: Creative Extension (30 marks)

Extend your program in a "cool" way, e.g. using a more attractive graphical representation, more functionalities, some simple AI etc. Please explain your extension in the file A2cool.pdf and submit a separate source file titled A2cool.py. The markers will evaluate your submission using the following criteria:

- Explanation (how detailed, clear and insightful is your explanation?)
- Novelty and creativity (how creative / interesting / novel / "cool" is your solution?)
- Technical difficulty (what is the complexity of your solution in terms of software development and/or algorithm development skills reflected in your solution?)

Submission

Submit your assignment online via the assignment dropbox (https://adb.auckland.ac.nz/) at any time from the first submission date up until the final date. You will receive an electronic receipt. Submit ONE A2.zip file containing the following files:

1. **A2.py** (Python source file with the solution for sections 1-4)
2. **A2cool.py** (Python source file with the solution for section 5)
3. **A2cool.pdf** (pdf-file with your explanations for the extension described in section 5)

Remember to include your name, UPI and a comment at the beginning of each file you create or modify. You may make more than one submission, but note that every submission that you make replaces your previous submission. Submit ALL your files in every submission. Only your very latest submission will be marked. Please double check that you have included all the files required to run your program and A2cool.pdf in the zip file before you submit it.

Note 1. We will only mark your submitted code. Your testing results on coderunner will not be counted. It is created for you to test your program only.
Note 2: Your program must compile and run to gain any marks. We recommend that you check this on the lab machines before you submit. Your solution in A2.py must run on CodeRunner.

Marking Details

TOTAL: 110 marks

Section 1: The class Deque	**20 marks**
Display the hidden numbers	5
Display all numbers	5
Manage the movement within one deque	5
Manage the movements among a few deques	5

Section 2: : display(self) of class Solitaire	**10 marks**
All numbers of all piles can be displayed correctly (1)	2.5
All numbers of all piles can be displayed correctly (2)	2.5
All numbers of all piles can be displayed correctly (3)	2.5
All numbers of all piles can be displayed correctly (4)	2.5

Section 3: move(self, c1, c2) of class Solitaire	**15 marks**
Move from pile 0 to pile 0	3
Move from pile 0 to pile 1	3
Move from pile 0 to pile 2	3
Move from pile 0 to pile 3	3
Move from any pile to any valid pile	3

Section 4: full program	**30 marks**
Win in 3 steps	3
Win in 5 steps	3
Lose in 6 steps	3
Win in 6 steps	3
Win in 15 steps	3
Lose in 20 steps	3

Win in 16 steps	3
Lose in 30 steps	3
Win in 24 steps (1)	3
Win in 24 steps (2)	3

Section 5: creative extension	**30 marks**
Explanation (how detailed, clear and insightful is your explanation in the file A2cool.pdf?)	10
Novelty and creativity (how creative / interesting / novel / “cool” is your solution?)	10
Technical difficulty (what is the complexity of your solution in terms of software development and/or algorithm development skills reflected in your solution?)	10

Section 6: Coding style	**5 marks**
Appropriate comments	3
Easy to read	2