Assignment 2

COMP9021, Session 1, 2015

Aims: The purpose of the assignment is to:

- design and implement an interface based on the desired behaviour of an application program;
- practice the use of functions encapsulated in a class and some builtin data structures, lists in particular;
- develop problem solving skills.

Submission

Your program will be stored in a file named `solitaire.py`. Upload your file using WebCMS. Assignments can be submitted more than once: the last version is marked. Your assignment is due by May 10, 11:59pm.

Assessment

All functions of the interface except `generated_sequence_of_jumps()` will be automatically tested for a maximum mark of 5. **If and only if you successively pass all tests for all following functions**:

- `game.set_configuration()`
- `game.set_complement_configuration()`
- `game.get_configuration()`
- `game.get_configuration_length()`
- `game.configuration_is()`
- `game.apply_jumps()`

then the function `generated_sequence_of_jumps()` will be automatically tested for a maximum mark of 4. Up to 1 mark will reward good readability of the source code, good comments when needed, and reasonable complexity of the underlying logic. So if you fail at least one test for at least one of the functions above, then the maximal mark you can score for this assignment is 6.

Passing all tests for all functions above is a very easy task. On the other hand, the function `generated_sequence_of_jumps()` is significantly harder to implement well, and may be impossible to implemented perfectly, so you might have to tackle the open problem of making it more and more powerful... I am likely to request you to submit test cases which you think are particularly hard to pass and that few students would pass, with possibly bonus marks as a reward.

For each test, the automarking script will let your program run for 30 seconds.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

You should of course look for resources on the Internet, and it might be valuable to play with the Solitaire widget provided in week 6.

Board positions

We refer to the positions of the board as follows:

```
      37 47 57
   26 36 46 56 66
15 25 35 45 55 65 75
14 24 34 44 54 64 74
13 23 33 43 53 63 73
   22 32 42 52 62
      31 41 51
```

This system of reference is also used in the implementation of the Solitaire widget provided in week 6.

Very important

The functions:

- `get_configuration()`
- `get_configuration_length()`
- `configuration_is()`

do not print out anything; they only **return** values (list, int, boolean, respectively).

Expected behaviour for the easy part (including all functions for which you need to pass all tests in order for your program to be tested further)

```
$ python
...
>>> from solitaire import *
>>> game = Solitaire()
>>> game.get_configuration()
[]
>>> game.display_configuration()
    O O O
  O O O O O
O O O O O O O
O O O O O O O
O O O O O O O
  O O O O O
    O O O
>>> game.set_configuration([35, 43, 44, 8, 45, 46, 55])
Invalid configuration.
>>> game.get_configuration()
[]
>>> game.set_configuration([35, 43, 44, 45, 46, 55])
>>> game.get_configuration()
[46, 35, 45, 55, 44, 43]
>>> game.get_configuration_length()
6
>>> game.set_configuration([])
>>> game.get_configuration()
[]
>>> game.set_configuration([46, 43, 43, 35, 44, 55, 45, 55, 46, 46])
>>> game.get_configuration()
[46, 35, 45, 55, 44, 43]
>>> game.configuration_is([44, 55, 55, 55, 35, 46, 44, 55, 45])
False
>>> game.configuration_is([44, 55, 55, 55, 35, 46, 44, 55, 45, 43, 12])
False
>>> game.configuration_is([44, 55, 55, 55, 35, 46, 44, 55, 45, 43])
True
>>> game.display_configuration()
    O O O
  O O X O O
O O X X X O O
O O O X O O O
O O O X O O O
  O O O O O
    O O O
```

```
>>> game.set_configuration([44])
>>> game.display_configuration()
    O O O
  O O O O O
O O O O O O O
O O O X O O O
O O O O O O O
  O O O O O
    O O O
>>> game.set_complement_configuration([44])
>>> game.get_configuration()
[37, 47, 57, 26, 36, 46, 56, 66, 15, 25, 35, 45, 55, 65, 75, 14, 24, 34, 54, 64,
 74, 13, 23, 33, 43, 53, 63, 73, 22, 32, 42, 52, 62, 31, 41, 51]
>>> game.display_configuration()
    X X X
  X X X X X
X X X X X X X
X X X O X X X
X X X X X X X
  X X X X X
    X X X
>>> game.set_complement_configuration([51, 37, 2, 31, 44, 37, 51, 57, 144])
>>> game.display_configuration()
    O X O
  X X X X X
X X X X X X X
X X X O X X X
X X X X X X X
  X X X X X
    O X O
>>> game.set_configuration([35, 43, 44, 45, 46, 55])
>>> game.apply_jumps([[45, 65], [43, 45], [35, 55], [65, 55], [46, 44]])
Invalid sequence of jumps.
>>> game.apply_jumps([[45, 65], [43, 45], [35, 55], [65, 45], [46, 44]])
>>> game.get_configuration()
[44]
>>> game.display_configuration()
    O O O
  O O O O O
O O O O O O O
O O O X O O O
O O O O O O O
  O O O O O
    O O O
```

```
>>> game.set_configuration([35, 43, 44, 45, 46, 55])
>>> game.display_jumps([[45, 65], [43, 45], [35, 55], [65, 45], [46, 44]])
    O O O
  O O X O O
O O X O O X O
O O O X O O O
O O O X O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O X X O X O
O O O O O O O
O O O O O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O O O X X O
O O O O O O O
O O O O O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O O X O O O
O O O O O O O
O O O O O O O
  O O O O O
    O O O

    O O O
  O O O O O
O O O O O O O
O O O X O O O
O O O O O O O
  O O O O O
    O O O
>>> game.apply_reverse_jumps([[45, 65], [40, 45], [35, 55], [65, 45],
 [46, 44]])
Invalid sequence of jumps.
```

```
>>> game.apply_reverse_jumps([[45, 65], [43, 45], [35, 55], [65, 45],
 [46, 44]])
>>> game.get_configuration()
[46, 35, 45, 55, 44, 43]
>>> game.apply_jumps([[45, 65], [43, 45], [35, 55], [65, 45], [46, 44]])
>>> game.display_reverse_jumps([[45, 65], [43, 45], [35, 55], [65, 45],
 [46, 44]])
    O O O
  O O X O O
O O O X O O O
O O O O O O O
O O O O O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O O O X X O
O O O O O O O
O O O O O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O X X O X O
O O O O O O O
O O O O O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O X O O X O
O O O X O O O
O O O X O O O
  O O O O O
    O O O

    O O O
  O O X O O
O O X X X O O
O O O X O O O
O O O X O O O
  O O O O O
    O O O
```

```
>>> game.set_configuration([35, 43, 44, 45, 46, 55])
>>> game.display_configuration()
    O O O
  O O X O O
O O X X X O O
O O O X O O O
O O O X O O O
  O O O O O
    O O O
>>> game.list_possible_next_jump()
[[45, 47], [45, 25], [45, 65], [44, 42]]
>>> game.set_complement_configuration([45, 44, 54, 53, 63])
>>> game.display_configuration()
    X X X
  X X X X X
X X X O X X X
X X X O O X X
X X X X O O X
  X X X X X
    X X X
>>> game.list_possible_next_jump()
[[47, 45], [56, 54], [25, 45], [65, 45], [65, 63], [24, 44], [74, 54],
 [33, 53], [42, 44], [51, 53]]
```

Remarks

It should be clear from the previous that:

- `set_configuration()` accepts duplicates, and numbers in any order;
- `set_complement_configuration()` accepts duplicates, and incorrect numbers on non numbers (which are just ignored);
- `get_configuration()` lists the numbers in the canonical order given by the way we refer to the positions of the board, from top to bottom and from left to right;
- `list_possible_next_jump()` lists the pairs of numbers in the canonical lexicographic order given by the way we refer to the positions of the board, from top to bottom and from left to right.

Expected behaviour for the function `generated_sequence_of_jumps()`

```
>>> game.set_complement_configuration([37])
>>> game.apply_jumps(game.generated_sequence_of_jumps())
>>> game.get_configuration_length()
1
>>> game.set_configuration([35, 45, 34, 44])
>>> game.display_configuration()
    O O O
  O O O O O
O O X X O O O
O O X X O O O
O O O O O O O
  O O O O O
    O O O
>>> game.apply_jumps(game.generated_sequence_of_jumps())
>>> game.get_configuration_length()
1
>>> game.set_configuration([35, 45, 34, 44])
>>> game.apply_jumps(game.generated_sequence_of_jumps(goal = [23]))
>>> game.configuration_is([23])
True
>>> game.set_configuration([35, 45, 34, 44])
>>> game.apply_jumps(game.generated_sequence_of_jumps(goal = [51]))
>>> game.configuration_is([51])
False
>>> game.set_configuration([26, 36, 56, 66])
>>> game.display_configuration()
    O O O
  X X O X X
O O O O O O O
O O O O O O O
O O O O O O O
  O O O O O
    O O O
>>> game.apply_jumps(game.generated_sequence_of_jumps())
>>> game.get_configuration_length()
2
```

Remarks

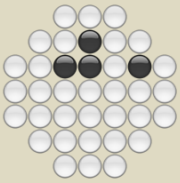
- The optional `goal` argument of `generated_sequence_of_jumps()` is an arbitrary sequence of valid board positions;
- when the optional `goal` argument of `generated_sequence_of_jumps()` is not provided, the aim of that function is to produce a sequence of jumps resulting in the least number of pegs remaining on the board (1 in the best case, which of course is not always possible).

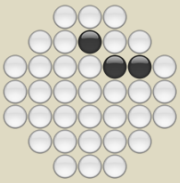
Nicer outputs

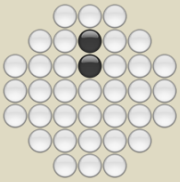
To avoid all too predictable problems, we output the letters `O`'s and `X`'s, but for development purposes, you might prefer to output unicode characters (of code 26AB and 26AA). Then you will be able to interact with your program as follows from the command line (or from within Idle, but the output is not as clean as from the command line):

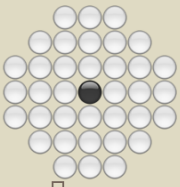
```
>>> from solitaire import *
>>> game = Solitaire()
>>> game.set_configuration([35, 43, 44, 45, 46, 55])
>>> game.display_configuration()
```

```
>>> game.display_jumps([[45, 65], [43, 45], [35, 55], [65, 45], [46, 44]])
```









```
>>>
```