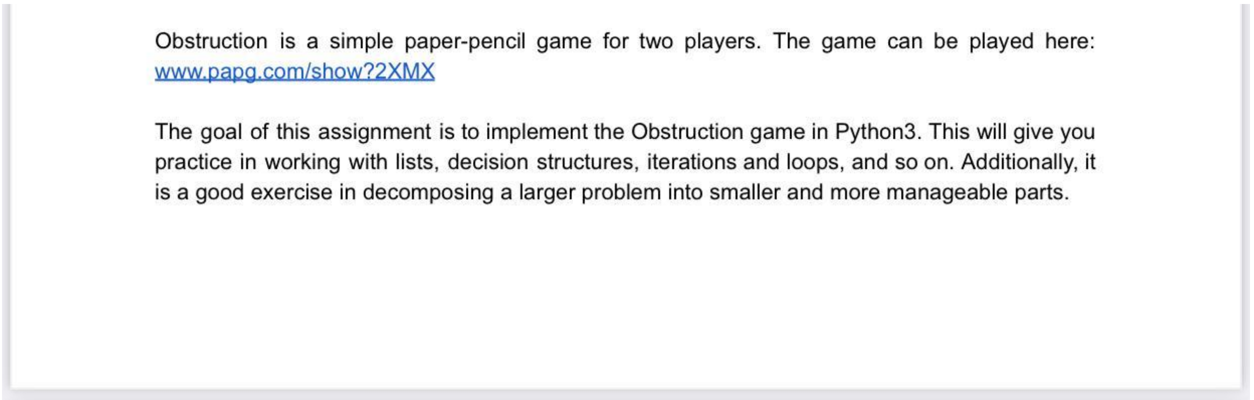
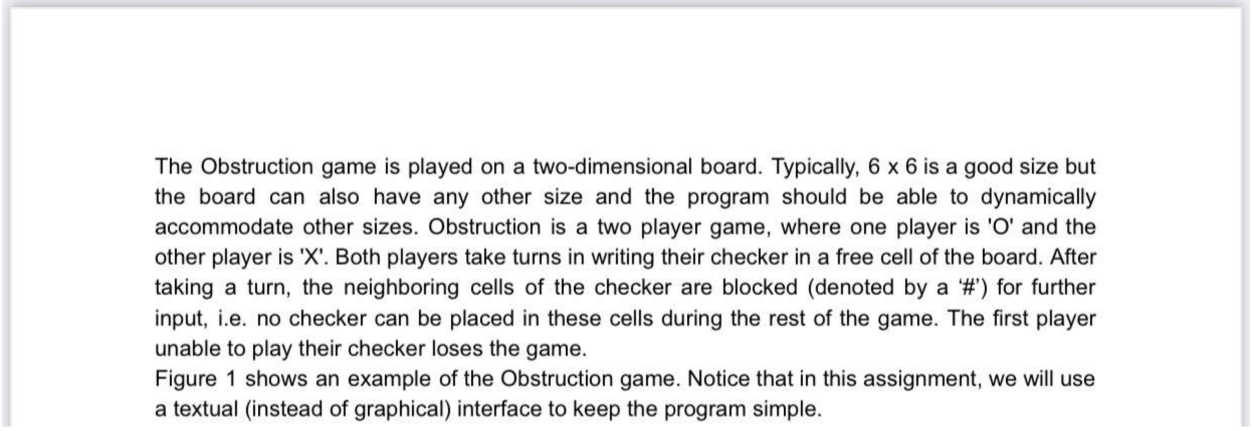
Assignment of Obstruction Game

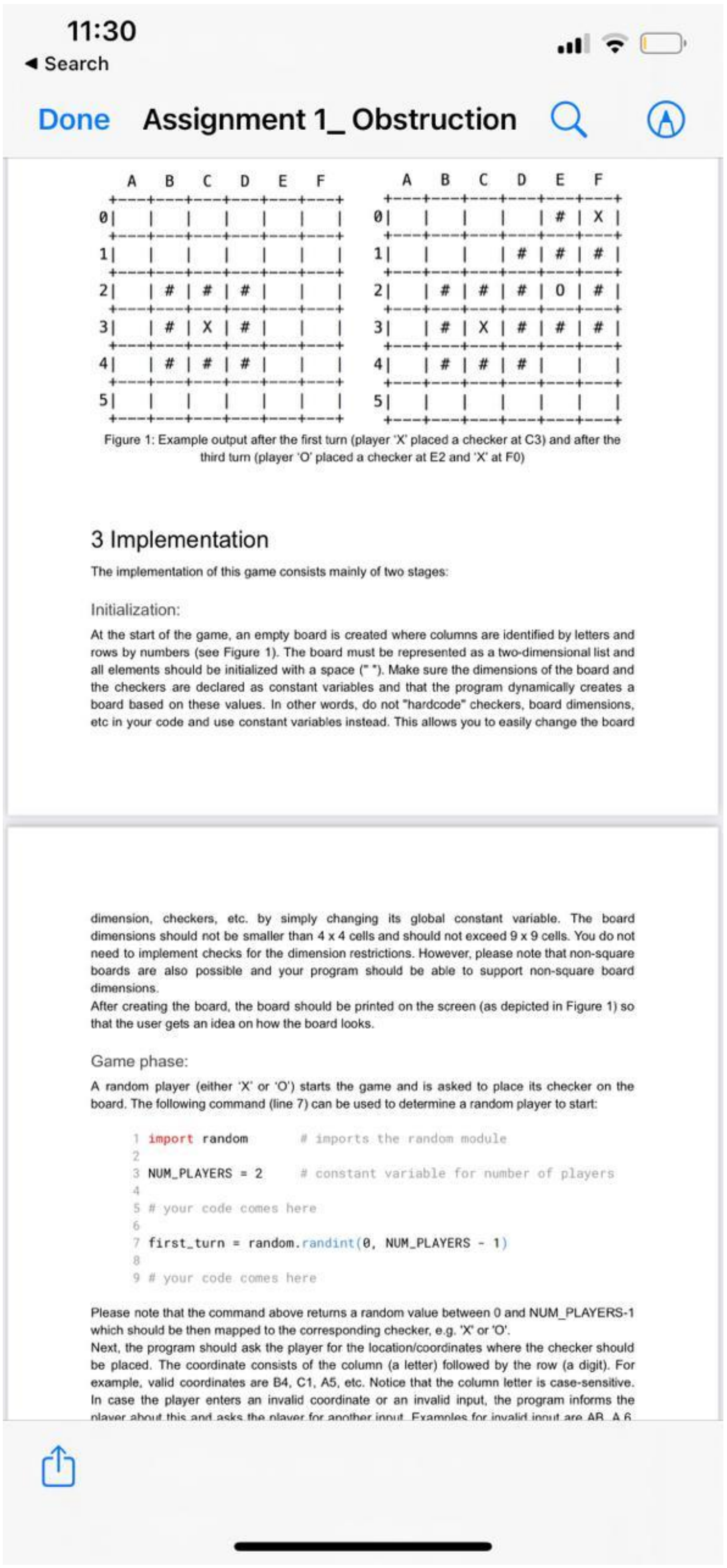


Obstruction is a simple paper-pencil game for two players. The game can be played here: www.papg.com/show?2XMX

The goal of this assignment is to implement the Obstruction game in Python3. This will give you practice in working with lists, decision structures, iterations and loops, and so on. Additionally, it is a good exercise in decomposing a larger problem into smaller and more manageable parts.



The Obstruction game is played on a two-dimensional board. Typically, 6 x 6 is a good size but the board can also have any other size and the program should be able to dynamically accommodate other sizes. Obstruction is a two player game, where one player is 'O' and the other player is 'X'. Both players take turns in writing their checker in a free cell of the board. After taking a turn, the neighboring cells of the checker are blocked (denoted by a '#') for further input, i.e. no checker can be placed in these cells during the rest of the game. The first player unable to play their checker loses the game.
Figure 1 shows an example of the Obstruction game. Notice that in this assignment, we will use a textual (instead of graphical) interface to keep the program simple.



```
    A   B   C   D   E   F            A   B   C   D   E   F
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
 0|   |   |   |   |   |   |       0|   |   |   |   | # | X |
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
 1|   |   |   |   |   |   |       1|   |   |   | # | # | # |
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
 2|   | # | # | # |   |   |       2|   | # | # | # | O | # |
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
 3|   | # | X | # |   |   |       3|   | # | X | # | # | # |
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
 4|   | # | # | # |   |   |       4|   | # | # | # |   |   |
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
 5|   |   |   |   |   |   |       5|   |   |   |   |   |   |
  +---+---+---+---+---+---+        +---+---+---+---+---+---+
```

Figure 1: Example output after the first turn (player 'X' placed a checker at C3) and after the third turn (player 'O' placed a checker at E2 and 'X' at F0)

3 Implementation

The implementation of this game consists mainly of two stages:

Initialization:

At the start of the game, an empty board is created where columns are identified by letters and rows by numbers (see Figure 1). The board must be represented as a two-dimensional list and all elements should be initialized with a space (" "). Make sure the dimensions of the board and the checkers are declared as constant variables and that the program dynamically creates a board based on these values. In other words, do not "hardcode" checkers, board dimensions, etc in your code and use constant variables instead. This allows you to easily change the board dimension, checkers, etc. by simply changing its global constant variable. The board dimensions should not be smaller than 4 x 4 cells and should not exceed 9 x 9 cells. You do not need to implement checks for the dimension restrictions. However, please note that non-square boards are also possible and your program should be able to support non-square board dimensions.

After creating the board, the board should be printed on the screen (as depicted in Figure 1) so that the user gets an idea on how the board looks.

Game phase:

A random player (either 'X' or 'O') starts the game and is asked to place its checker on the board. The following command (line 7) can be used to determine a random player to start:

```
1 import random        # imports the random module
2
3 NUM_PLAYERS = 2      # constant variable for number of players
4
5 # your code comes here
6
7 first_turn = random.randint(0, NUM_PLAYERS - 1)
8
9 # your code comes here
```

Please note that the command above returns a random value between 0 and NUM_PLAYERS-1 which should be then mapped to the corresponding checker, e.g. 'X' or 'O'.

Next, the program should ask the player for the location/coordinates where the checker should be placed. The coordinate consists of the column (a letter) followed by the row (a digit). For example, valid coordinates are B4, C1, A5, etc. Notice that the column letter is case-sensitive. In case the player enters an invalid coordinate or an invalid input, the program informs the player about this and asks the player for another input. Examples for invalid input are AB, A 6,

te that the command above returns a random value between 0 and NUM_PLAYERS-1
uld be then mapped to the corresponding checker, e.g. 'X' or 'O'.
program should ask the player for the location/coordinates where the checker should be placed. The coordinate consists of the column (a letter) followed by the row (a digit). For example, valid coordinates are B4, C1, A5, etc. Notice that the column letter is case-sensitive. In case the player enters an invalid coordinate or an invalid input, the program informs the player about this and asks the player for another input. Examples for invalid input are AB, A 6, 1A, 32, A;5, a6, etc. or coordinates out of the board dimensions.
After a valid user input, the program adds the checker to the board, "blocks" the neighboring cells (denoted by a '#') and asks the other user for input. This procedure repeats until the game is won by any of the players. The board should be printed after each turn with the updated cells. Before printing the updated board, the screen should be cleared by using the following command (line 6):

```
1 import random       # imports the random module
2 import os           # imports the os module
3
4 # your code comes here
5
6 os.system("clear")  # clears the screen
```

After each turn, the program checks if further turns are possible and if not, the game ends. The program should inform the user that the game is over and which player won.

4 Grading

Description	Score (/15)
Initialization of the board	1
Using global variables (constants) for board dimensions, checkers, etc.	1
Printing of the board and board labels, as depicted in Figure 1	2
Randomly selecting first player and alternating turns	2
Checking for invalid user input (letter followed by number)	2
Checking for invalid coordinates (repetitive input of same cell, out of board, etc))	2
Correctly placing the checker and "blocking" the neighboring cells	2
Detecting the end of the game and terminating the game after a win	2
Using comments and readability of code (style, variable naming, etc.)	1