SUPINFO Academic Dept.

Algorithms and programming in Python

Second Mini - Project

SOS Game

Version 1.0
Last update: 13/12/2018
Use: Students/Staff
Author: Laurent GODEFROY/PN

SUMMARY

1 PREAMBLE

This exam must be realized by groups of two students. In the single case where the number of students in the promotion is an odd number, one and only one group of three students is allowed.

Any form of plagiarism or using codes available online or any other type of support, even partial, is prohibited and will cause a 0, a cheater mention, and even a disciplinary board.

This mini-project will be presented by a project defence. Your passing time will be announced by your campus.

The projects defences are also made up of groups of two. This will take **20 minutes** in which you will show your examiner the proper functioning of your program with a demo. If you have not implemented the entire project, you will expose the functioning parts.

To support your presentation, you must prepare a PowerPoint file type, in which you will explain parts of the code that you judge important and significant. It is not necessary to send your file to your examiner. He will discover it day of your viva. A communication that specifies all those informations will be sent early January.

An **indicative** scale is given to you in the last part of the topic.

2 GENERALITIES ABOUT THIS PROJECT

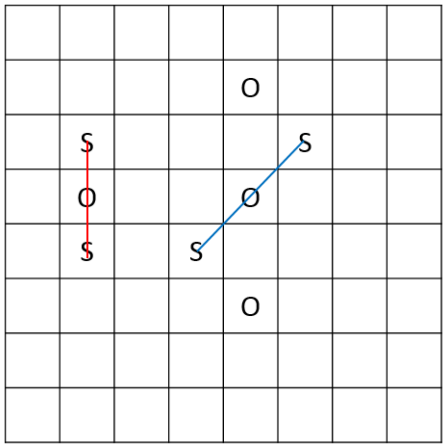
Important note: no code is requested in this sub part, where we will explain the rules.

The objective of this mini-project is to write in Python a program to play the reflection and logic game “SOS GAME”. This game is also available here at Google Play (produced by Gasos Technology).

Two players confront each other on a square board of dimension n, i.e. which contains n lines and n columns. At the beginning, the board is empty.

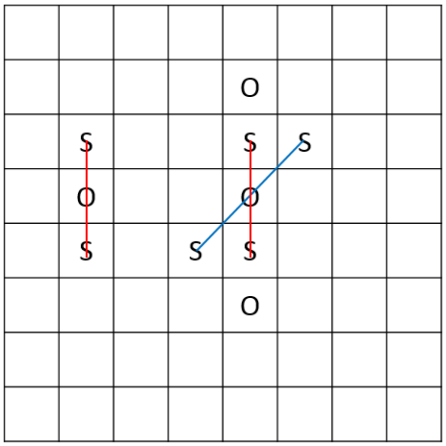
At each turn, a player will write on an empty square the letter “S” or the letter “O”. They have total liberty for their choice. The objective is to do vertical, horizontal or diagonal alignments of SOS sequences. To do this, the player may use indistinctly his letters or those of his opponent.

We will keep a trace for each alignment using colors like below:

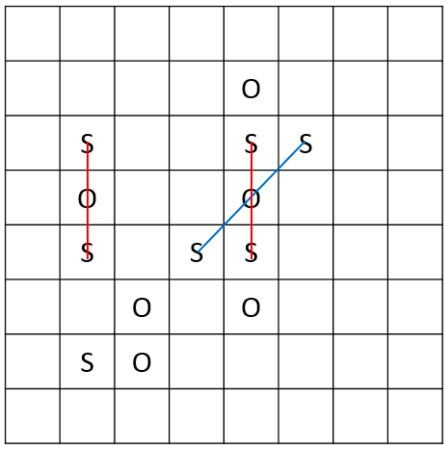


If a player does an alignment just after writing a letter, his score will increase by 1 and must play again. Otherwise, his score doesn't change and his opponent plays.

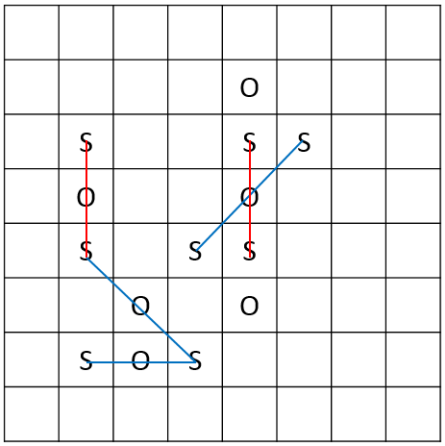
Note that a single letter can be used for many alignments, like the O on the fourth line and fifth column of this example:



Write a letter can cause simultaneously several alignments. If we are in this situation:



If the blue player puts an S on the seventh line and fourth column, he does two alignments:



His score will increase of two.

When the board is filled, the winner is the player with more alignments. A draw is possible.

Several readings may be useful to understand the subject. Take the time to well understand before starting coding.

In the third part, we will implement the algorithms for the game.

In the fourth part, we will develop the graphical interface.

Finally, in the fifth part, we will propose some bonus and possible extensions for this game.

3 ALGORITHMS TO PLAY THIS GAME

It is highly recommended to read this part before starting to code.

3.1 DATA STRUCTURES

The board will be modeled by a two-dimensional list of integers. Each element of this list represents a square of the board. An empty square will be represented by 0, an 'S' by 1 and an 'O' by 2.

The players will be represented by an integer equal to 1 or 2.

The players' scores will be saved in a list of two integers.

3.2 ALGORITHMS

Important notes:

- We can possibly implement subroutines in addition to those asked.
- Instead calculate regularly the dimension of the list, we will prefer to pass it as a parameter of our subroutines.

Notations of the subroutines' parameters that we will implement:

- « board »: two-dimensional list of integers equals to 0, 1 or 2 that represent the board of the game.
- « n »: strictly positive integer equal to the number of rows and columns of the « board ».
- « player »: integer that represents the player whose the turn it is.
- « scores »: list of two integers that represents the score of the two players.
- « i »: any integer.
- « j »: any integer.
- « lines »: two-dimensional list whose items are lists of two tuples of two integers. Those lists of two tuples represent the coordinates of two points, the beginning and the ends of an alignment.
- « l »: integer equal to 1 or 2, that represents an « S » or an « O ».

Implement the following subroutines in a file called « sosAlgorithms.py »:

- A function « newBoard(n) » that returns a two-dimensional list, which represents the initial state of a board with **n** rows and **n** columns.
- A function « possibleSquare(board,n,i,j) » that returns **True** if **i** and **j** are the coordinates of a square where a player can write a letter. Otherwise it returns False.
- A procedure « updateScoreS(board,n,i,j,scores,player,lines) » that assume that the player **player** wrote the letter « S » on a square of coordinates **i** and **j**. It is looking the possible « SOS » alignments that the player could have spawned by writing this letter. Then it updates the score of the player **player** and the list **lines**.

- A procedure « updateScoreO(board,n,i,j,scores,player,lines) » that assumes that the player **player** wrote a letter « O » on a square of coordinates **i** and **j**. It is looking the possible « SOS » alignments that the player could have spawned by writing this letter. Then it updates the score of the player **player** and the list **lines**.
- A procedure « update(board,n,i,j,l,scores,player,lines) » that updates the board by assigning the value **l** to the square of coordinates **i** and **j**. Depending on the case, the procedure call one of the two previous procedures. When this procedure occurs, the list **lines** is empty.
- A function « winner(scores) » that returns a string which indicates the result of the game.

4 A GRAPHICAL INTERFACE

It is highly recommended to read this part before starting to code. We will use the graphical library Pygame. Using another library won't be taken into account.

Note 1: in this part we will use the same data structures of the previous part. The parameters will have the same meaning, but in addition we will have the parameter « mySurface » which indicates the surface where we draw.

Note 2: you are free to design the graphics as you want as long as the features are available. The screenshots are just indicatives.

Note 3: all the events are triggered by the mouse and not by the keyboard.

4.1 THE DISPLAYS PROCEDURES

The goal of this part is to implement the procedures that permit to display the board and the players' scores.

Initially the empty board will be displayed as follow:

Each empty square is divided by two sides. The player whose turn it is will click on the left side to write the letter « S » on this square or on the right side to write the letter « O ».
The arrow at the right of the scores indicates the player whose turn it is.

After some turns, here is what the board should look like:

Note that after having a new letter, we won't redraw all the board. We just write the letter and eventually draw new alignments. We use the color of the actual player.

Finally, when the game will be over, the result will be displayed:

Implement the following subroutines in a file called « sosGUI.py »:

- A procedure « drawBoard(mySurface,n) » that draws the initial board.
- A procedure « displayScore(mySurface,n,scores) » that displays the scores of the two players.
- A procedure « displayPlayer(mySurface,n,player) » that indicates to the player **player** that it's his turn to play.
- A procedure « drawCell(mySurface,board,i,j,player) » that draw the content of the square of coordinates **i** and **j** with the color of the player **player**.
- A procedure « drawLines(mySurface,lines,player) » that draws lines which represent the eventual alignments. Those informations are in the list **lines**.
- A procedure « displayWinner(mySurface,n,scores) » that draws the result of the game.

4.2 THE GAME PROCEDURE

In the previous file, implement the following subroutines:

- A function « selectSquare(mySurface,board,n) » that permits to choose a square and a letter, and that returns those datas in a tuple **(i,j,l)**.
- A procedure « gamePlay(mySurface,board,n,scores) » that manages the entire game.

4.3 THE MAIN PROCEDURE

In the previous file, implement the following subroutines:

- A procedure « SOS(n) » which creates a graphical window, initializes the data structures and manages the entire game.

5 BONUS

You are free to implement zero, one or many of the following bonus.

5.1 SAVE A GAME

Add an option that permits to save the actual state of a game in text file.
Implement the possibility to resume a game saved in a text file.

5.2 MULTI-PLAYERS

Implement a play mode where more than two players can play.

5.3 IA RANDOM

Implement a solo player mode against the computer. The computer plays randomly.

5.4 IA LIGHTLY CLEVER

Implement a play mode against the computer. The computer plays as follow:

- If an « SOS » is possible, the computer makes it.
- Otherwise, when it's possible, it writes an « S » or an « O » to prevent the player to make an alignment.

6 INDICATIVE SCALE

This scale may change and is only **indicative**.

- Part 3 : 15 points
- Part 4 : 25 points

- Bonus : 10 points

It makes a total of 40 points, and your project defence is evaluated on 20 points. The grade of 60 points will be proportionally brought to a grade based on 20 points.