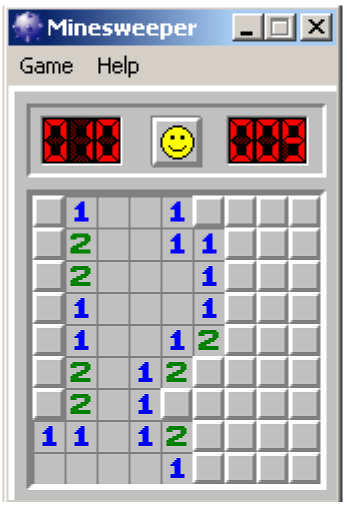
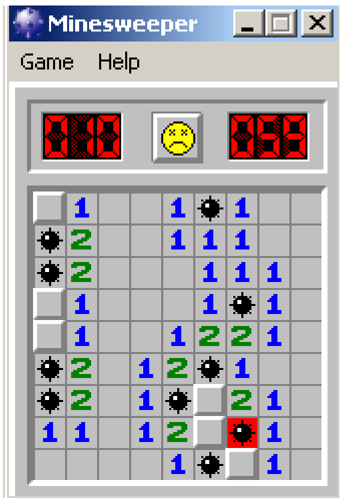
Programming Project 06 (30%)

The purpose of this project is to work with classes.

Assignment Overview

Minesweeper is a popular single-player computer board game that comes with Windows Operating System. The object of the game is to clear an imaginary minefield without stepping on a mine.





Game description
The game is played on a rectangular grid (Board) of undistinguished squares – (covered cells), where some of them might contain mines.
The grid size, and the number of mines, are set by the player. In our project, legal boards range between 1x2 and 20x50 (*rows X columns*), and the legal number of the mines on a board, range between 1 to (*rows X columns*) -1. The mines are randomly placed on the grid.

The game is played by revealing (uncovering) covered squares on the board.
- If a square containing a mine is revealed, the game is over.
- If the square does not have a mine, then a digit might appear in the square, indicating the number of adjacent squares, that contain mines.
- But, if there are no mines in any of the adjacent squares surrounding this square, then a Ripple effect takes control, and an automatic revealing process of the surrounding squares starts to operate (see below).

The game is won when all the cells without mines are revealed.
The above game pictures are only for illustrative purposes.

Task

Your task is to implement a simple version of the minesweeper game in Python. Before implementing the game, it might be wise to play the game for a few times, to help understand the project. It might prove inspiring to look at : http://en.wikipedia.org/wiki/Minesweeper_(video_game).

Use the wp-proj06Start.py file as a 'dear' skeleton, and as a guiding API that must be obeyed.

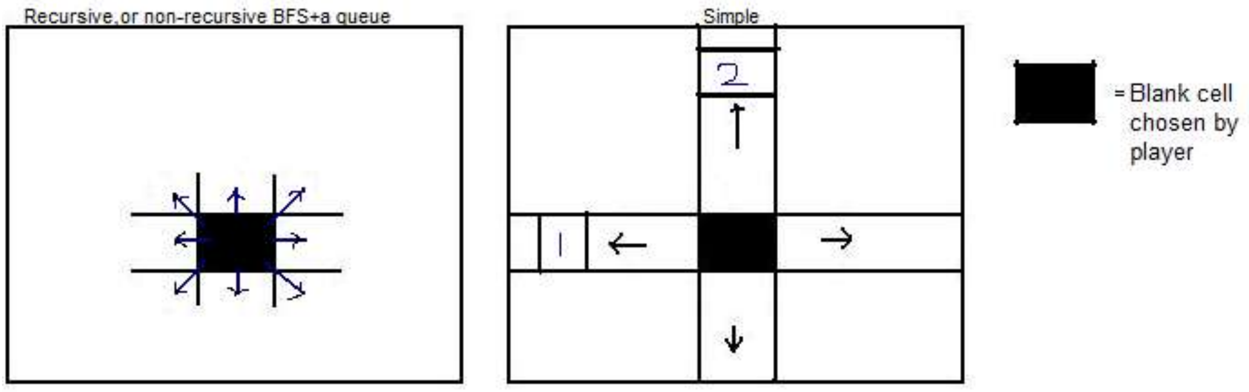
The Ripple Effect

A cell without adjacent mines around it, is called a blank cell, and it is marked by 0.
It is 'explosion-safe' to uncover all its (covered) adjacent neighbors. After doing that to the blank chosen by the user, for each new blank revealed in the process, the uncovering of all its adjacents will go again, and so on. It will stop only when there will be no more blanks to uncover around them. This is the Ripple effect. It fuels on blank cells, and on the 'explosion-safe' processing of all the adjacents of a blank cell. It is born natural recursive, but in the project we ask for the recursion to start the uncovering with the cell one row above the blank cell, and to move around the blank clockwisely, uncovering each cell amongst its adjacents. If any of the cells revealed in the process is itself a blank cell (marked 0), it is also rippled before continuing to the other cells, recursively.

More ways to Ripple

Besides the recursion, we suggest two other non recursive ways to implement the Ripple effect :
- a basic simple ripple, that 'moves' only along the row and the column of the initial given blank cell, so the adjacent squares diagonally positioned to the initial blank, are left uncovered.

- a BFS ripple, which uses a breadth first search using a queue. The adjacent squares to the uncovered blank are added to a queue, and each of them which is also a blank (marked 0), adds its adjacent cells to the end of the queue, if they are not there already. (Skip the cells that are already in the queue, as in a regular BFS implementation).


Recursive.or non-recursive BFS+a queue
Simple
2
1
= Blank cell chosen by player

Each of the three ways is rewarded differently :
- projects using the basic ripple start from **90%**
- recursion starts from **100%**
- bfs starts from **105%**

General Discussion

Basically, the user initiates a new game on the command line by setting the board's sizes and the number of mines. From now on, each step in the game is accompanied by a print display that includes the board current situation, followed by a group of dialogued options:

```
$python wp-proj06.py -r 4 -c 8 -m 12
   0  1  2  3  4  5  6  7
0  H  H  H  H  H  H  H  H
1  H  H  H  H  H  H  H  H
2  H  H  H  H  H  H  H  H
3  H  H  H  H  H  H  H  H
Game status: NotStarted
Available actions: (1) Save | (2) Exit | (3) Move
```

The user advances the game by choosing between the available options.

A started game can be 'freezed', its current board saved to a file, and later reloaded for its revived continuation. This means that there are file format specifications to follow. Then, a frozen game can be initiated from the command line, using the right options. All these IO operations are carefully watched by a battalion of Exceptions (classes), suggested in the skeleton, that just wait for you to implement and use them in the project.

The basic step of the game is done by the user, when he chooses a square to uncover, by giving its row/column coordinates. Then the program complies by uncovering this square. Uncovering a minefield square has its perils, and a lot of things may happen, the worst: uncovering a mine, the best: uncovering a blank cell and triggering the Ripple effect which is a master of automatic uncovering of all the adjacent squares around blank squares. The usual thing that happens, is to uncover a cell marked with a number between 1- 8, that represents the N of mines in his adjacent surroundings.
Treating adjacent surroundings of cells is important here. There are three sizes of adjacent surroundings, depending on their in-center cell position. Corner cells have 3 adjacents, on-edges-but-not-on-corners cells have 5, and mid-board cells have 8 adjacents.

More dialog cycles examples :

```
Enter selection: 3
Enter row then column (space separated): 1 2
   0  1  2  3  4  5  6  7
0  H  H  H  H  H  H  H  H
1  H  H  5  H  H  H  H  H
2  H  H  H  H  H  H  H  H
3  H  H  H  H  H  H  H  H
Game status: InProgress
Available actions: (1) Save | (2) Exit | (3) Move
Enter selection: 1
Enter filename: game1.txt
Save operation done
   0  1  2  3  4  5  6  7
0  H  H  H  H  H  H  H  H
1  H  H  5  H  H  H  H  H
```

```
2  H  H  H  H  H  H  H  H
3  H  H  H  H  H  H  H  H
Game status: InProgress
Available actions: (1) Save | (2) Exit | (3) Move
Enter selection: 2
Goodbye :)

$python wp-proj06.py -i game1.txt
   0  1  2  3  4  5  6  7
0  H  H  H  H  H  H  H  H
1  H  H  5  H  H  H  H  H
2  H  H  H  H  H  H  H  H
3  H  H  H  H  H  H  H  H
Game status: InProgress
Available actions: (1) Save | (2) Exit | (3) Move
Enter selection: 3
Enter row then column (space separated): 3 0
   0  1  2  3  4  5  6  7
0  H  H  H  H  H  H  H  H
1  H  H  5  H  H  H  H  H
2  H  H  H  H  H  H  H  H
3  1  H  H  H  H  H  H  H
Game status: InProgress
Available actions: (1) Save | (2) Exit | (3) Move
Enter selection: 3
Enter row then column (space separated): 3 7
   0  1  2  3  4  5  6  7
0  H  H  H  H  H  H  H  H
1  H  H  5  H  H  H  H  H
2  H  H  H  H  H  H  H  H
3  1  H  H  H  H  H  H  *
Game status: Lose
Available actions: (1) Save | (2) Exit
Enter selection:
```

Program Specifications:

Printing the Board

The Board print format includes the cells contents, and their row/column indexing:
The serial numbers of the rows and the columns of the game board, start from 0, and will be displayed each time the game board is printed. The 1st column holds the row numbers, and is aligned with the respective rows, and the 1st row holds the column numbers of the board, and is aligned with the respective columns. It should look like this:

```
  0  1  2  3...
0 H  H  H  H
1 H  H  1  *
….
```

where 'H' marks hidden cells (even in winning situations), and the asterisk ‘*’, or a digit from 0-8, mark different in-game values for the uncovered cells.

The print format for the whole board is 2 characters per cell, left-aligned, one space-separated: "%-2s %-2s ... %-2s" , as the number of the columns, and another for the row serial number.

The first has an empty string as the first argument to the print format (first “%-2s”) and the others are the column numbers.

```
   0  1  2  3  ...
```

This line precedes the numbered game board lines themselves.

The Machinery behind the interactive sequence of Prompts/Prints/Dialogs for each step:

1) print the board, (as described above)

2) print
"Game status: %s", where %s is one of **'NotStarted', 'InProgress', 'Win', 'Lose'**

3) print available actions, which can be one of the following (depending on the state):
"Available actions: (1) Save | (2) Exit | (3) Move" # for in progress and not started
"Available actions: (1) Save | (2) Exit" # for win/lose

4) ch = raw_input("Enter selection: "). The input should be exactly a digit, no whitespace or other characters around it.

5) if ch == '1', do the following:
    5.1) raw_input = "Enter filename: "
    5.2) try to save the board with the filename
    5.3) if succeeds, print "Save operation done", else (error) print "Save operation failed"

6) if ch == '2', print "Goodbye :)" and return (None)

7) if ch == '3' and not win/lose status, do the following:
    7.1) move = raw_input("Enter row then column (space separated): ")
    7.2) take the input and convert to integers, then make move and update game status. The input is exacly in the format “%d %d” with no other characters or whitespace.
    7.3) if any error happened while doing 7.2, print "Illegal move values"

8) if not one of the legal choices, print "Illegal choice"

9) go to 1

Command line arguments for game start, or game continuation:

r, rows –N of rows (integer), default 1
c, columns –N of columns (integer), default 2, (smallest legal board is : 1x2)
m, mines – N of initial mines to be scattered (integer), default 1
i, input – name of input file

You must use argparse in order to specify the arguments names and handle them from the command line. All arguments should be converted to file-descriptor/integer by argparse itself, using the 'type' argument to add_argument.
Also use the 'default' argument to set defaults.
Error prints are in the function description (main())

More restrictions:
Don't use globals, or code outside functions :
You can add classes, methods, inner classes, but don't
add any (unindented) executing line, outside the methods/functions.

Deliverables

The file to deliver is wp-proj06.py

בהצלחה