Minesweeper Manual

1. Input and Outputs (Sample Run)

Random inputs

Table:

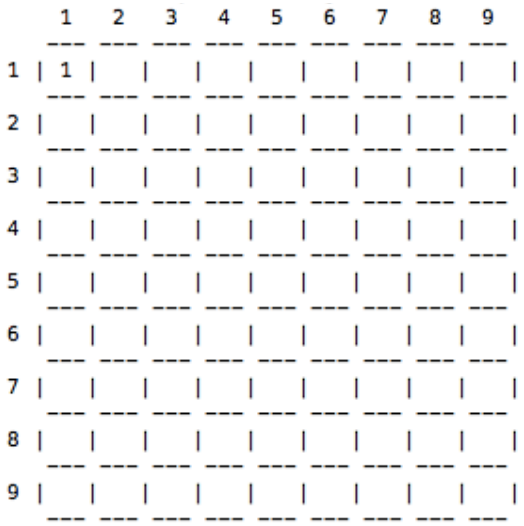
a table we generate using random sample function. It should be 11×11 and we randomly choose 10 cell assign the value 9 to them from the center 9×9 part. The value 9 indicates the cell should be treat as a bomb.

User input and outputs

I will use black words to represent the exact output of the program, use red words to show the exact input and blue words to describe the program.

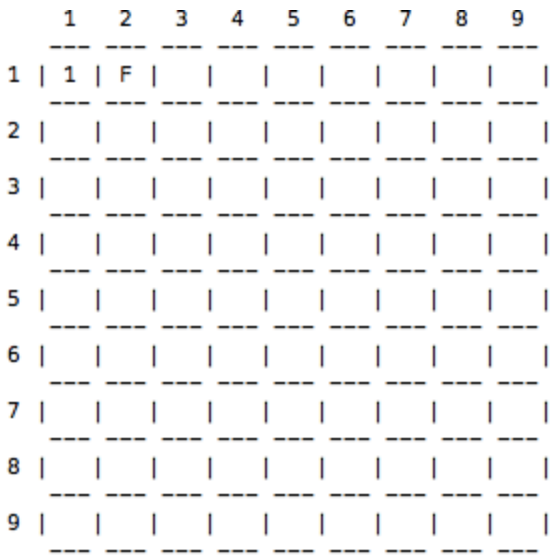
```
$ minesweeper_hw.py
Welcome to this Minesweeper Game!
Also print some instructions here
Here is the board:
The program will print a nice table. Since it is a little bit large,
we don't show it here
Let's begin!
Please enter the row index first (choose one in 1 ~ 9),
Then type in column index (choose from 1 ~ 9).
If you want to put or remove a flag, please type in 'f' before the
cell, e. g, f12.
```

```
Please enter your move now:123
Invalid input! You should only type in 2 digits or 'f' plus 2 digits.
Please enter your move again: f232
Invalid input! You should type in 2 or 3 characters.
Please enter your move again: wew
Invalid input! You should only type in 2 digits or 'f' plus 2 digits.
Please enter your move again: 11
```



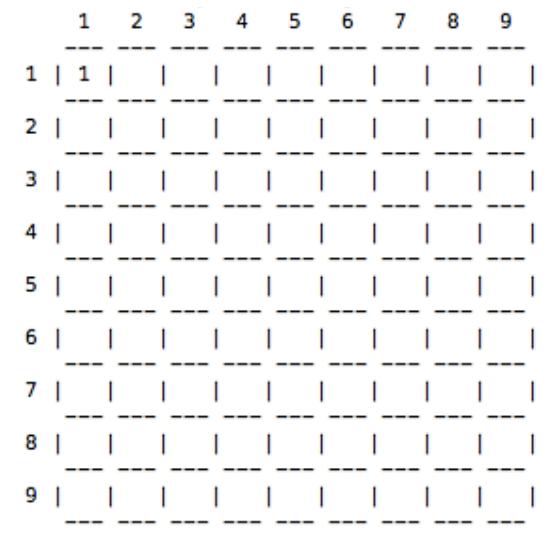
The program will should the board after each step and will be changed according to the user’s action.

Please enter your move now: f12



If the user type in a “f” before the two-digit position, the certain cell will be “flagged”.

Please enter your move now: f12

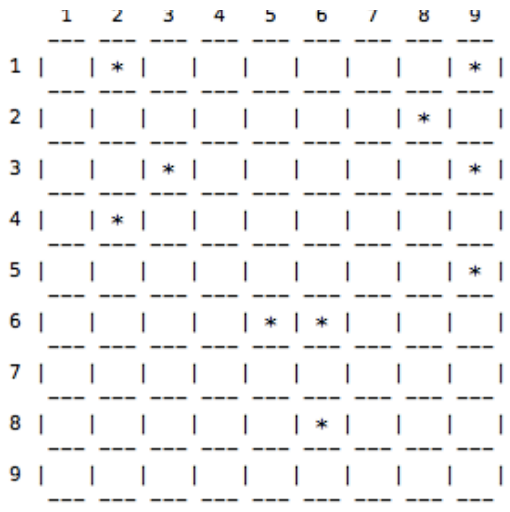


If the user type in a “f” before the two-digit position which has been flagged before, the flag will be removed.

Please enter your move now :12

You lose.

```
Here is the position of mines
```




```
Do you want to restart the game? please choose 'yes' or 'no' yes
If type in yes, the program will restart a new game which means it
will generate another random table and redo all above.
Do you want to restart the game? please choose 'yes' or 'no' ewfes
Invalid input! please type 'yes' or 'no'. no
Goodbye!
```

If all of the cells without bomb have been revealed, the user will win. The program will print “you win!”, ask whether the user want to restart the game.

2. Variables

Table a table we generate using random sample function. It should be 11×11 and we randomly choose 10 cell assign the value 9 to them from the center 9×9 part. The value 9 indicates the cell should be treat as a bomb.

Base a list includes 81 elements from the class Cell. We generate this list

according to the table. It contains all information we need of each element and will be updated every time after the user move.

Move a list whose length is 2 or 3. It record what the user wants to do.

3. Module usage

numpy a fundamental package for scientific computing with Python.
Use numpy.array( ) and numpy.zeros( ) to create the table.
random This module implements pseudo-random number generators for various distributions.
Use random.sample( ) to randomly generate positions of mines.

4. What does this program DO

- **Create an invisible table with 10 randomly assigned bombs**
  - Obtain 10 bombs' indexes
    - Use np.zeroes((11,11),int) to create an empty 11×11 table
    - Define an empty list to store the index of each cell in the table (here the index is the sequence of the cell in the table, as the first and the last rows and columns are not cells. It will simplify the later step for calculating bombs near a cell without bomb.)
    - Use list comprehension to store indexes in a list of lists, each list containing one row with all columns
    - Loop through the above list, and append all indexes to the empty

list. Thus, 10 bombs' indexes are obtained.

- Place 10 bombs
  - To obtain bombs' positions, use random.sample(emptylist, 10) to randomly choose 10 positions to place bombs, and cast the list to a numpy array.
  - Simply calculate the row and column indexes of each bomb in the 11*11 table.
  - Put bombs into the table, marked as integer 9, and return the table.

- **Set up a class Cell to give properties to each cell**
  - Define a special function: __init__ (self, nrow, ncol, table). The function contains the following information (initial values in the parentheses):
    - The row index of the cell (self.row = nrow)
    - The column index of the cell (self.col = ncol)
    - Whether the cell is revealed or not (self.visible = False)
    - Whether the cell is flagged (self.flag = False)
    - The position of a cell in the table, indicated by row index and the column index (self.bomb = table[nrow][ncol])
  - Define a function to calculate the number of bombs near each cell (num_nearby(self, table)). In this part, the previous 11*11 table will be helpful)
    - The initial number of bombs near each cell is 0.
    - If the cell contains a bomb, the function will return integer 9.
    - If the cell does not contain a bomb, and if the row and column indexes are both in range(1,10), loop through its nearby 8 cells to see how many bombs are around, and return the number of bombs

nearby. (As we use 11*11 table, it equals to add a frame, which is made up of integer zeros, to the required 9*9 table. Thus, each element in the center 9*9 table is equal, and we do not need to specify the edge and corner elements.)

- **Create a list of cells containing properties from class Cell**
  - Define an empty list named "base".
  - Loop through each cell without the outer zeros frame by row and column indexes in the 11*11 table, and append the Cell properties to the empty list.
  - Return the list containing properties of each cell
- **Check whether the user's input is valid.**
  - If the length of user's input is not 2 or 3, tell him the it's invalid, and what he should type in. Return False.
  - If the length of user's input is 2
    - Try to cast his input into 2 integers, if it causes a ValueError, tell him the it's invalid, and what he should type in. Return False.
    - If one number of the user's input is not between 1 and 9, tell him the it's invalid, and what he should type in. Return False.
  - If the length of the user's input is 3
    - If the first element he types in is not "f", which indicates a flag, tell him the it's invalid, and what he should type in. Return False.
    - Again, if one number of the user's input is not between 1 and 9, tell him the it's invalid, and what he should type in. Return False.
  - Return True
- **Check whether the cell chosen by user is available**

- Use list comprehension to separately assign each cell’s visible and flag status to two lists (visible_status and flag_status).
- Set a toggle ava = True. User’s input has been checked valid here.
- If the length of user’s input is 2, which means the user wants to reveal this cell
  - Calculate the cell’s position
  - If the cell is revealed or flagged, tell the user and set ava=False
- If the length of user’s input is 3, which means the user wants to flag this cell
  - If the total number of flags equals 10, the user cannot put a flag anymore and set ava=False
  - If the total number of flags does not reach 10, calculate the cell’s index. If the cell is revealed, tell the user and set ava=Flase
- Return the toggle ava

- **Ask the user to type in their action, check the input and return the position list they want to go.**
  - Ask the user to type in his action use raw_input, and split his command into a list. Use the previous function to check whether it is valid. While it is not a valid input, repeat the raw_input step
  - Check whether the input cell is available. While it is not available, repeat the raw_input step
  - Return a list of input which is valid and available.
- **Check whether the game is end or not.**
  - Set a toggle judge=True
  - If the length of the user’s input is 2

        - Calculate the cell’s index and see whether it is a bomb. If it is, tell the user he loses the game and set judge=False.
    - Use list comprehension to separately assign each cell’s visible status, position, and the index for all cells without bombs to three lists (visible_status, bomb, and index).
    - If the visible status of all the indexes in the index list is True, tell the user that he wins, and set judge=False.
    - Return judge
- **Update the base list after each step**
    - If the length of the user’s input is 2
        - Calculate the cell’s index and update the visible status of the cell to True
    - If the length of the user’s input is 3
        - Calculate the cell’s index and change its flag status
    - Return the base list
- **Print the updated table**
    - Use list comprehension to separately assign each cell’s visible status, flag status, and the number of its nearby bombs to three lists (visible_status, flag_status, and nearby).
    - Loop through each cell. Print the updated board according to each cell’s properties
        - If the visible status is True, print the number of bombs nearby in the cell
        - Else if the flag status is True, print a symbol “F” in the cell
        - Else print nothing in the cell

- **Check whether the user wants to play again**
  - Use raw_input to ask the user whether he wants to play again (The user needs to type in “yes” or “no”)
  - While the user types in something other than “yes” or “no”, ask him again.
  - If he types in “yes”, return True. Else, return False.
- **Combine all the other functions and start the game**
  - Set a toggle: program = True (Whether the user wants to restart the program)
  - While program is True
    - Create a table and use the table to create a base list
    - Print some instructions and the blank board
    - Set another toggle: game = True (Whether the game is end or not)
    - While game is True
      - Update the base list and check whether the game is over or not.
      - If the user did not touch a bomb, print the updated board.
      - If the user revealed a bomb, set game=False, and print the bombs on the board.
      - Ask the user whether he wants to restart. If he wants to quit, set program=False, and end the program. Otherwise, he can play again.
- **Automatically run the program**