Adventure Time

Idea

A small adventure game. How large the board is and "easy" or "hard" mode are predefined by command line arguments. Bombs and treasures are randomly placed on the board, while other positions are empty. Player can choose position on the board he wants to go. At the beginning he has m (which is computed according to the number of bombs) lives, he'll lose one life every time he steps on a bomb. The ultimate goal of the game is to find all treasures before dead. The game ends when player is dead or he has found all treasures.

Topics included

1. **Error handling**
   When game starts, the program successively prompts the user to make selections. To make sure invalid user input doesn't cause the program to crash, I use try/except. For example, the input is expected to be numbers which will be cast into int afterwards, however the user types a letter, then except ValueError can be used to handle this issue.
2. **Classes/objects**
   I mainly defined three classes in this program, namely Board, Player, Game.
3. **Numpy**
   Generate the board by randomly select elements with replacement from a given set, according to a given probability; generate a 1D array of zeros.
4. **Command line arguments**
   The board size n and mode (1: easy; 2: hard) are predefined by command line arguments. For example, we run the program by "$ adventure.py 4 1"
   As we have constraints that n must be an integer from 4 to 9, mode must be 1 or 2, if not satisfied, it will print out

   Usage: python adventure.py <Board size> <Level (easy/hard)>

   <Board size>: Board size should be an integer from 4-9.

   <Level (easy/hard)>: Enter number for difficulty Level 1:easy, 2:hard.

Inputs and outputs (Sample run)

- **Inputs**

Command line arguments: As mentioned in previous section.
Player selections: Every time the game is started, the program prompts the player to make selections.
Quit: If player types in quit, the system quits the entire program.

- **Outputs (Sample run)**

```
In [1]: runfile('C:/Users/Admin/Desktop/MS/301/Final_Program/
final_2.py', args='4 2', wdir='C:/Users/Admin/Desktop/MS/301/
Final_Program')
Welcome to Adventure Time!

You have 6 life remained
You found 0 treasures there're still 4 treasures
You stamped on 0 bombs there're still 9 bombs
Your current score is  0  points.

    1   2   3   4
   --- --- --- ---
1 |   |   |   |   |
   --- --- --- ---
2 |   |   |   |   |
   --- --- --- ---
3 |   |   |   |   |
   --- --- --- ---
4 |   |   |   |   |
   --- --- --- ---

Select a valid position or enter "quit" to quit the game.

Where would you like to go? Please type in two integers representing
row/column: 33

Haha! You found a treasure !


Your current position is: ( 3 , 3 )

There're 1 treasures around you! Good luck!
There're 2 bombs around you! Be careful!

You have 6 life remained
You found 1 treasures there're still 3 treasures
You stamped on 0 bombs there're still 9 bombs
Your current score is  8  points.

    1   2   3   4
   --- --- --- ---
1 |   |   |   |   |
   --- --- --- ---
2 |   |   |   |   |
   --- --- --- ---
3 |   |   | $ |   |
   --- --- --- ---
4 |   |   |   |   |
   --- --- --- ---

Select a valid position or enter "quit" to quit the game.

Where would you like to go? Please type in two integers representing
row/column: 22

Oh,no! You stepped on a bomb!
```

```
Congratulations! You've found all treasures!

The true map is:

    1   2   3   4
   --- --- --- ---
1 | * | * | o | * |
   --- --- --- ---
2 | o | * | $ | * |
   --- --- --- ---
3 | $ | * | $ | o |
   --- --- --- ---
4 | $ | * | * | * |
   --- --- --- ---
You have 2 life remained
You found 4 treasures there're still 0 treasures
You stamped on 4 bombs there're still 5 bombs
Your current score is  0  points.

Would you like to play again? Yes/No  Your selection: No
```

List of tasks/functions

1. **Print the board**

   Take in the board array and print the $n \times n$ configuration. The thing on the board is represented by a 1D array. The correspondence between (row, col) position and index is shown below. Eg. a 4×4 board, the 10^{th} element (index 9) of this 1D array represents the thing in position (3,2).

```
    1   2   3   4
   --- --- --- ---
1 | 0 | 1 | 2 | 3 |
   --- --- --- ---
2 | 4 | 5 | 6 | 7 |
   --- --- --- ---
3 | 8 | 9 |10 |11 |
   --- --- --- ---
4 |12 |13 |14 |15 |
   --- --- --- ---
```

2. **The Board class**

   (1) Initialize the board: size n, 1D array representing things on the board, a list which records visited positions, i.e. positions the player has already chosen.

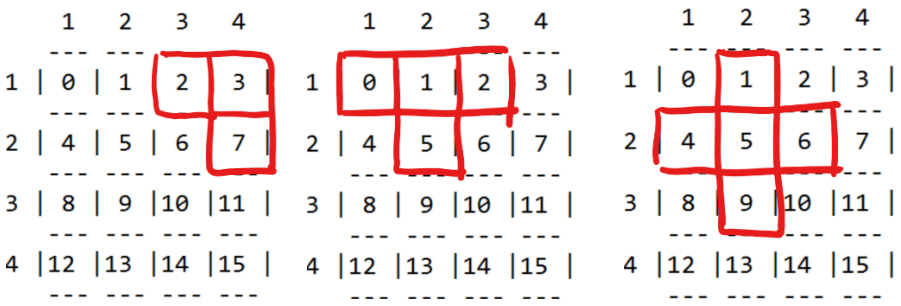
   (2) Create a board with n*n cells, containing bombs/treasures or nothing, the proportion of which differs from game mode easy/hard.

   (3) Record the amount of treasures and bombs.

   (4) Judge if all treasures have been found.

   (5) Based on visited positions, decide the board showing to the player, just reveal the things on positions already chosen.

(6) After player selects a position (row, col), return the corresponding serial number on the board, i.e. the index of this cell.
(7) After player selects a position (row, col), return the "thing" in that cell.
(8) Judge if the position is available: after player selects a position, return False if he has already selected this position once, return True if the position hasn't been selected before and add the position to visited list.
(9) Give player hints: Tell player what his current position is and the number of treasures & bombs around him. Here, “around” means the cross centered by the current position, as shown below.



3. The Player class

(1) Initialize the player, which is the *main character* in the game. The player has 4 attributes: number of lives left, number of treasures/bombs he has found/stepped on, and the steps he has taken. These variables are updated if needed each time the player makes a move.
(2) Calculate player’s score based on number of treasures/bombs he has found/stepped on, and the steps he has taken. The original score is zero.
(3) Be able to choose a position: Try to select a position on current board, return False if it's already selected once, return True if success and update other variables if needed.
(4) Print the player’s current status: lives left, # treasures have been found and not been found yet, # bombs have been found and not found yet, current score.
(5) Decide if the player is dead: return True if there's no life left.

4. InvalidInput class

Define a new exception class called InvalidInput to indicate the input is invalid.

5. QuitError class

Define a new exception class called QuitError to raise exception for “quit” command in the input.

6. The Game class

(1) Initialize a game: given board size n and play mode, initialize the board and the player, the original life of the player equals ceil(0.6*number of bombs).
(2) Restart a game: when one round of game ends, the program asks the player if he would like to play again. If yes, start the game again.

(3) Judge if the game ends: The game ends when player is dead or he has found all treasures.

(4) Prompt the player to choose a position, return player's choice (row, col) if it's valid. Raise QuitError (quit the entire game) if player types in "quit", raise InvalidInput if the player types in other thing instead of two integers like"42" representing row/col, or the position is out of range.

(5) Play game: use a while loop, prompt the player to choose a position until the game is over. After player makes a move, check if the position is valid by function mentioned in (4)

   i. If valid, check if the position has been selected once
      a. If yes, print the message and ask again.
      b. If no, go ahead to next steps.
   ii. If invalid, print the invalid message and ask the player to select again.

   At each round, print the player's status, the board with "things" he already found, and hints. When the game ends, reveal the true board and ask the player if he would like to play again. If yes, start the game again. If no, exit the program.

7. Main function

Check if command arguments are valid:

i. If yes, initialize the game and start to play.

ii. If not, display the usage message.