Adventure Game

Game description

As the title indicates, this is an adventure game. I hope you will have a lot of fun implementing this project and that you will at the same time learn a lot from it. This project will definitely require from you time, efforts and research. As you will be building it from scratch. This time, unlike previous project, you will get no code from me. But I will instead explain to you how to build it in steps. Note that this is <u>your</u> game. Feel free to give it your own flavour once you have done everything I asked.
The game starts again by asking to the user to identify him/herself based on a name. Once identified, the character of the game will land somewhere in a virtual map (we don't see the map, but it is implemented in the backend of the game i.e. 'behind the scenes'). All the information according to the character and its current state in the game will be printed in the Standard Output Stream.

The primary objective of the character is to kill all the monsters that are present in his world (represented by a 2D map). At the start of the game, the character has 5 lifes and lands on a specific bloc of the by you generated 2D map. Then the character has the choice to take any direction (left, right, up down) that leads to another block that is inside the borders of this map. This means that if the character is currently on the left border of the map he should have only 3 possibilities left (up, down or right), so he cannot go to the left anymore.

In each block it is possible to meet an enemy (monster). The character doesn't know in advance where the monsters are located. Any time the character enters a territory (block) where a monster is present, then he has the choice to either run away, or to start a battle against that monster. If he chooses to run away, the monster can follow him and there is a big chance that the character will lose 1 life (but not necessary). If instead he chooses to face the enemy, a battle will start.
The battle isn't a physical fight. But it's a 'Rock Paper Scissor' game between you (the character) and the monster. You as the character of the game make a choice and the monster generates a random choice. The winner of this game is the winner after 3 times. (For those of you who don't know about the 'Rock Paper Scissor' game : … you had no childhood ! Here is a link to play this game : https://www.rpsgame.org/).
If the monster wins the game, he stays on the current block and the character loses 1 life and has to leave the current block back to its previous position. If the character instead wins the game, he gets 1 extra life and the monster disappears.

After every turn, the character has to choose a new direction to continue its quest of destroying all the monsters of the map. But the character is not a robot with infinite power supply. After 5 turns, the character needs to sleep. But if on the 5th turn he lands on a territory where an enemy is located, then during its sleep it will lose 2 lifes.
The only location of the map where it is impossible to cross a monster is at the character's home.

Implementation

The game will be entirely played in back-end. And the inputs will be throug the classical Standard Input Stream.
Before the start of the game, the user enters the name of the character. Then, when he hits enter, the game starts:

1) Generating the map

1.1) Multi-dimensional list

In order to generate a map, we need to know how to make multi-dimensional lists or matrices (don't use dictionnaries for this game).
A multi-dimensional list is a list that has lists as elements.

Take for example following list:
[[0,1,2,3], [4,5,6,7], [8,9,7,4]]
This list represents a matrix of 3 lignes and 4 columns. In order to simplify the visualisation of this list we can write it down this way:

```
[ [0,1,2,3],
  [4,5,6,7],
  [8,9,7,4] ]
```

You can now see that for example the number 8 is located on the 3rd row and the first column. This means the row 2 and column 0. So in order to access that element through indexation, we need to index it 2 times ! See this code below :

```
>>> lst = [ [0,1,2,3], [4,5,6,7], [8,9,7,4] ]
```

What will be according to you the result of following line ?:

```
>>> lst[2]
```

In order to find the result of this code, we can for example count the number of elements of lst. How many elements does this list contain ? Know that every element of a list should be seperated by a comma ','. In this case lst doesn't have 12 elements, but only 3 ! Each of these 3 elements is also a list. This means that following command: `lst[2]` will return as value: `[8,9,7,4]` and not 2 ! Make sure to understand this before continuing.

But what if we want to reach number 8 ? How should we index it ?
Well, we now know that lst[2] is a list. So we can index that list as well by doing :

```
>>> lst[2][0]
```

This means that we first access the list `[8,9,7,4]` and then its element on the first position (`[0]`).
Train by playing with this until you fully understand how it works. This is crucial for this project.

1.2) The structure of a map

The world of this game will also be represented by a multi-dimensionnal list. Conside the map below:

	0	1	2	3	4	5	6	7
0	‘M’				‘M’			
1								
2		‘M’		‘P’				
3					‘M’			
4							H	
5				‘M’				

This is a map with dimensions 6x8. How will we represent such a map in form of a list ?
Well, we know that there are 6 lines and 8 columns. So we know that there will be a big list containing 6 elements, each of them being a list of 8 elements (Try to implement it !)
But this time, instead of having numbers as elements, the lists contain strings. An empty string (‘’) means that this block is empty. The string ‘P’ represents the character of the game: Pablo. And ‘M’ represents a monster. In this case we have for example: lst[0][0] == ‘M’ and lst[4][7] == ‘’.

1.3) Generate an empty map

How to generate a map by having as a given only the dimansions ?
This is the function you need to implement :

```
def generate_empty_map(nb_rows, nb_cols):
    """
    Return a multi-dimentional list, with given nb of rows and columns.
    All elements should be empty strings --> ""
    :param nb_rows: int representing the nb of to be generated rows
    :param nb_cols: int representing the nb of to be generated columns
    :return: multi-dim list containign empty strings.
    """
    pass
```

The map that will be returned by this function will have an empty string at every position. Not monsters nor characters yet on the game. Remove the ‘pass’ and start implementing this function. Make tests for this function (assert statements) in a seperate file. DO IT! Otherwise I can guarantee you that you’ll suffer in this project ! So do it for this function directly when you finish it. The same applies for all other functions of your project.

2) Insert monsters at random locations

This function will require from you a bit of reflexion and eventually some internet research.

```
def insert_monsters_into(map):
    """
    Insert monsters randomly in positions of given map (list) by changing the value at
that position into
    the string: 'M'. Every position of the map has 20% chance to get a monster.
    :param map: a list as is returned by generateEmptyMap()
    :return: the new map (with the added monsters)
    """
    pass
```

In order to insert monsters at random locations, you will need to pass by every cell (block) of the given map (that will be generated by generate_empty_map()). Each position has 20% chance to get a monster inserted. It is your turn now to come out with a correct algorithm for this problem. (You don’t necesseraly need to implement the whole logic inside the body of this function. Feel free to use auxilary functions implemented somewhere else and used in this function.)
Hint: You don’t necesseraly need the random module for this function. But it will work if you do so.

3) Build the character's home

The home of the character of the game is placed at a random position in the map. For this you'll need to get the information of the dimension of the map (the number of lines and columns) from the given parameter 'map' and select a random number for each of these two ranges:

```
def make_home(map):
    """
    Inserts a string 'H' on a random position in the given map. This position may not
contain any monster.
    :param map: a list as is returned by generateEmptyMap()
    :return: the new map (with the added home)
    """
    pass
```

4) The Battle

As stated above, the battle is a rock-paper-scissor battle between Pablo and a monster. You will therefore need to implement a function that simulates such a 3-times game. You ask an input from the user. The user can only choose between 3 things: 'Rock', 'Paper' or 'Scissor'. Everytime the user inputs something else, you should tell him that his input is illegal and give him another chance to input correctly.
Once the program got a valid input, the monster makes a choice. His choice will be a random selection from one of those three things (use the random module). That choice will be printed on the screen and it will calculate who won this round (or a draw). There are in total 3 rounds anytime such a battle takes place. The winner of the game will be the one that won the most rounds. In case of a total draw (even after the 3 rounds), the character will lose 1 life and the monster doesn't die. If it's a draw the user doesn't lose a life and the monster still doesn't die. In order to kill this monster the character will need to come back to this territory and fight the monster again.

```
def create_battle(map, char, monster_position):
    """
    Create a "Pierre-Papier-Ciseaux duel" between the user and a monster.
    :param map: list containing the entire map
    :param char: data-structure holding information about the player's character
    :param monster_position: tuple representing the position of the monster
    :return: the new map (with the eventual changes)
    """
    pass
```

5) Start of the game

At the start, the user enters a name for this character. Once done the character lands somewhere on a random place in the map.
Feel free to chose how to represent your character. But note that the character has following attributes:

- **He is currently pointing to a specific direction**
  The character points to a specific direction according the the last move he did. At the start of the game, the character will be pointing to the right. But then, when he makes his first move, his position will eventually change. (For example after moving left he'll be pointing to the left). This is important because you should not allow the character to go back to the location he was before his last move.
- **He has a number of lifes left**
- **He has a number of turns already passed**
  Because as it is stated in the intro, every 5 turns the character needs to rest. You therefore need to store the current number of turns passed.

Summary of the functions to implement:

Here is a set of to implement functions. Feel free of course to implement other (auxiliary) functions. By the way you have to make minimum two other new functions besides the ones below:

- **generate_empty_map(nb_rows, nb_cols)**
  Generates a map and returns it.
- **insert_monsters_into(map)**
  Inserts monsters in the map. Every position has 20% chance to get a monster.
- **make_home(map)**
  Insère le string 'M' à un endroit au hasard dans la map (sans monstre)
- **create_battle(map, char, monster_position)**
  Create the battle between the character and the monster at the given position.
- **startGame()**
  In this function the whole game will take place. This time you will get no code and not even pseudo-code. You will have to implement the entire algorithm yourself. Know that if you can do this on your own, you will have successfully processed the entire logics behind computer programming. Have fun while coding;)

  The game starts by asking for a name as input, then generates a map of dimension 10x10 and then inserts the monsters in th map.
  Then, after every new turn the program prints the current position of the character and the current number of turns passed. Then the user can choose where to go next (according to the possibilities left). Etc. etc. until ending the game.
  If the character enters a position where a monster is located, the program informs him and launches the battle. From the start of the game the program informs the character about the location of his home.

Some practicalities:

- **The character can not leave the map**
- **The character can not get back to the last position he was in**
  Of course he can get back to positions he has been before, but not the last one. So he can not make direct 180° turns. So if he is for example pointing left, he can not make his new movement to the right.
- **The home of the character is the only location where it is impossible to find a monster.**
- **The game is done once the character has no lifes anymore, or that he killed all the monsters.**
- **The monsters do not move. They stay on their initial positions until dying (or finishing the game)**

Happy coding!