Project 2: Text Monster Game

Using Python, Boolean expressions, lists and while loops to create a text-based adventure game!

Overview

This game takes place in a three-story virtual space. Each story (or floor) can have as many rooms as desired for game play. The player must navigate the levels in search of the prize. Along the way they collect items and fight monsters. On each move the player has seven possible commands: `LEFT`, `RIGHT`, `UP`, `DOWN`, `GRAB`, `FIGHT`. If the input is invalid (not one of these commands,) the game will let the player know. Otherwise, the game will execute the player's command. The goal of the game is to collect the prize guarded by the monster.

Details

Behavior Example

```
What would you like to do? left
You see a sword.
What would you like to do? grab
You picked up the sword.
What would you like to do? left
You see nothing.
What would you like to do? left
You see a monster.
What would you like to do? fight
You defeated the monster!
What would you like to do? left
You see stairs going down.
What would you like to do? down
You see nothing.
```

The game has three floors

- Each floor is made up of at least five rooms, arranged in a line from left to right.
- A room can contain a sword, a monster, magic stones, up-stairs, down-stairs, a prize, or nothing.

At the start of the game

- The player should always start in the same room.

Movement

- The player can try to move to the left room or right room.
- If there is no room in that direction, the game should report this.
- The player can also move upstairs or downstairs if the room contains an up-staircase or a down-staircase.
  - The player can only go up if there is an up-staircase
  - The player can only go down if there is a down-staircase.
  - If a player uses a staircase, the room the player enters must contain a staircase in the opposite direction.
- The program should not allow the player to run past a monster.
- The program should not allow the player to go up, down, left, or right past the bounds of the game.

Contents of rooms

- The game prints out the contents of the current room after every command.
- The player can grab swords or magic stones if they walk into a room with them.
- The sword or stones are no longer in the room once grabbed.
- Your virtual world should have a minimum of 4 swords.
- Your virtual world must have a minimum of 1 magic stone.

Monsters guard some rooms

- The player can use a sword to defeat a monster using the `FIGHT` command.
- The sword and monster disappear after fighting.
- If they have no sword, the player can exit in the direction from which they came.
- If the player fights without a sword, they will be defeated and the game will end.
- If they try to walk past a monster, they will be killed and the game will end.
- A sword and magic stones are required to defeat a monster.

Implementation Details

- The game should be implemented using lists.

Player Items

- Use a list to keep track of the player's items.
- At the beginning of the game it should be empty.
- Players can hold up to three items at a time.
- If a player attempts to pick up a fourth item, a warning message should be printed.
  - If a grab attempt fails, the item in question should remain in the room.

Monsters

- There should be three regular monsters placed throughout the game.
- Monsters require a sword to defeat.
- There should be a boss monster in the room just before the room that contains the prize.
- The boss monster requires magic stones and a sword to defeat.

Win/Lose

- The game is won when the player grabs the prize.
- The game is lost if the player:
  - fights a monster without a sword.
  - fights the boss monster without a sword and stones.
  - tries to move past a monster.

Design Considerations

Game Board

The game board is the basis of the game. The following is a way to think of a smaller game board as a set of three lists, one for each floor.

```
floor_1 = ['nothing', 'nothing', 'stairs up']
floor_2 = ['stairs up', 'nothing', 'stairs down']
floor_3 = ['stairs down', 'nothing', 'prize']
```

The above code has each floor being its own list. This is only one possible way to keep track of the game board. A good alternative would be a single nested list storing all floors, but this may be somewhat more difficult for new coders to work with.

Player Position

It will be useful to keep track of the player's position using two variables.

```
player_room = 0
 player_floor = floor_1
```

This would put the player's position in the first room of the first floor.

Validating Player Input

You will need to check the input of the player to make sure it is valid for the current game state:

```
if player_input == "down":
    current_room = player_floor[player_room]
    if current_room != "stairs down":
        print("Can't go downstairs; there are no stairs.")
```

Extra Credit

- Add an 'inventory' command to display the items (if any) that the player is currently carrying.
- Implement **Cardinal Directions** into your code to give more control over direction for the player.
- Include a short description of each room, printed each time a player enters it – and optionally a longer description of the same room, printed only the first time the player enters.
- Advanced: Randomly generate locations of monsters, magic stones and swords. Simple random placement is likely to make the game unplayable, so this will probably require a larger floor plan and careful design of the randomization.