Project 2: Text Monster Game Using Python, students will use casting, Boolean expressions, lists and while loops to create a text-based adventure game! ## Overview This game takes place in a three story dungeon. The user has to traverse the levels in search of the prize. Along the way they collect items and fight monsters. On each move the user has seven possible commands: `left`, `right`, `up`, `down`, `grab`, `fight`, `help`. If the input is invalid (not one of these commands,) the game will let the user know. Otherwise, the game will execute the user's command. The goal of the game is to collect the prize guarded by the boss monster. ## Details ### Behavior ```python What would you like to do? left You see sword What would you like to do? grab Picked up sword What would you like to do? left You see nothing What would you like to do? left You see monster What would you like to do? fight You defeated the monster! What would you like to do? left You see stairs down What would you like to do? down You see nothing What would you like to do? right You see stairs up What would you like to do? left You see nothing What would you like to do? left Sorry cant go that way. What would you like to do? up Cannot go that way. What would you like to do? right You see nothing What would you like to do? up Cannot go that way. What would you like to do? right You see stairs up What would you like to do? up You see nothing What would you like to do? right You see nothing What would you like to do? right You see nothing What would you like to do? right You see nothing What would you like to do? up Cannot go that way. What would you like to do? left You see nothing What would you like to do? ``` #### The game has three floors * Each floor is made up of five rooms, arranged in a line from left to right. * A room can contain: a sword, a monster, magic stones, up-stairs, down-stairs or nothing. #### At the start of the game, the user is placed in one of the rooms #### Movement * The user 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 user can also move upstairs or downstairs if the room contains an up-staircase or a down-staircase. #### Contents of rooms * The game prints out the contents of the current room after every command. * The user 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. #### Monsters guard some rooms * The user 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 user can exit in the direction from which they came. * If the user 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 the boss monster. ### Implementation Details * The game should be implemented using lists. #### User Items * Use a list to keep track of the user's items. * At the beginning of the game it should be empty. * A maximum of three items can be held at once. #### Monsters * There should be three regular monsters placed throughout the game. * These require a sword to defeat. * There should be a boss monster in the room just before the room that contains the prize. * This monster requires magic stones and a sword to defeat. #### Movement Implementation * The user can only go up if there is an up-staircase * The user can only go down if there is down-staircase. * The program should not allow the user to run past a monster, * The program should not allow the user to go up, down or past bounds of the game. #### Win/Lose * The game is won when the player grabs the prize. * The game is lost if the user * 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. ```python floor_1 = ['nothing', 'nothing', 'stairs up'] floor_2 = ['nothing', 'nothing', 'stairs up'] floor_3 = ['prize', 'nothing', 'nothing'] ``` The above code has each floor being its own lists. Feel free to use a different implementation, but this should work for our purposes. #### User Position It will be useful to keep track of the user's position through a variable. ```pythin user_room = 0 user_floor = floor_1 ``` This would put the user at the position of the first room of the first floor. #### Validating User Input You will need to check the input of the user to make sure it is valid for the current game state: ```python if user_input == "down": current_room = user_floor[user_room] if current_room != "stairs down": print("Can't go downstairs; there are no stairs.") ``` ## Extra Credit * Add the command `run`, which allows a player to run past a monster instead of fighting. This should work 40% of the time. (Hint: Research the random library.) * Implement the board using nested lists (each item of the list is a list.)