Textual RPG

Callback functions

Exercise 1:

Create a function named `processEnemies`. This function should accept two parameters: an array of enemy objects and a callback function.

The structure of an enemy object in the array should be like this:

```
{
  id: 12,
  name: "Enemy name",
  health: 100
}
```

Your `processEnemies` function should perform the following tasks:

1. Check if the first argument is an array. If not, throw an error with the message 'Expected an array of enemies'.
2. For each object in the array, check if it contains the properties `name` and `health`. If any object does not contain these properties, throw an error with the message `Enemy the ID ${enemy.id} is not in the right structure` (the id property will always be).
3. If the object structure is correct, clone the array to avoid mutating the original one: `const newEnemies = JSON.parse(JSON.stringify(enemies));`.
4. Loop over each object in the cloned array, and pass each object to the callback function.
5. Return the new array.

Exercise 2:

Create a function named `processQuests`. This function should accept two parameters: an array of quest objects and a callback function.

The structure of a quest object in the array should be like this:

```
{
  id: 5,
  name: "Quest name",
  experience: 100
}
```

Your `processQuests` function should perform the same tasks as in Exercise 1, but with quests. Adjust the error messages accordingly.

Exercise 3:

Create a function named `processPlayers`. This function should accept two parameters: an array of player objects and a callback function.

The structure of a player object in the array should be like this:

```
{
  id: '45e3e'
  name: "Player name",
  health: 100,
  level: 1,
  location: "forest",
  inventory: ["sword", "health potion"]
}
```

Your `processPlayers` function should perform the same tasks as in the previous exercises, but with players. The structure verification should check for `name`, `health`, `level`, `location`, and `inventory`. Adjust the error messages accordingly.

For each of these functions, remember to pass a callback function that will be applied to each object in the respective array.

For instance, you could test the `processPlayers` function with a callback that increments each player's health by 10 like this:

```
processPlayers(players, player => {
    player.health += 10;
    return player;
});
```

This will return a new array of players, each with their health incremented by 10, without modifying the original `players` array.