Monster game, game requiriments

Functional requirements.

Game kickoff

- The game will start asking for the player's name

    - The name must be a single word, starting with a capital letter and then just letters, without numbers or special characters.

    - The player must decide to not select any name or just cancel the prompt; in that case the game will assign a default name, like for example: "Anonymous"

- Then it will then generate a random number of monsters (between 1 and 3). A console log message will notify the player the number of monsters to combat.

- Then it will ask the user for a name for each monster. The game will provide a default name for each monster, that the user can accept or modify.

    - The monsters' name can contain just letters and can end with a single number (Ex: monster1, monster2; this names can be used as default names). No other character is allowed. The end number is optional. The monster's name must be a single word.

- The Player will start the game with:

    - A health of 100 points

- 2 Healing Potions (each one will heal the Player with 70 points)

- The monsters will start the game with a health of 100 points each.

Game general rules

- There will be a number of rounds until either the Player dies or all the monster dies.

- Each round starts with a Player fighting a Monster. Just pick up one of the monsters.

- The Player and the Monster will fight until one of them dies.

- After killing a Monster a new fight against a new Monster will start, until no more monsters are alive (or the player dies), that will end the game.

- Each round will start with a Player action and then will end with a Monster action, if the Monster is still alive.

    - If the Player kills the monster, then the round will end up and a new fight will start (if there are still monsters to fight agains).

Game rounds:

- The Player will be asked to pick up the next action:

  - Attack

  - Consume a Healing Potion

- If the Player decide to attack, he will cause a variable damage to the Monster between 10 & 20 points (a random number).

- The Monster will die if after the Player attack, his health is 0 o below.

- If the Player decide to consume a Healing Potion, his health will be restored with 40 points

- After executing the Player's action, the game will automatically execute the Monster action, if he is alive.

- If the monster was killed by the Player, the round will end up and a new round will start against a new Monster (if there are still any alive).

- The monster will automatically attack the Player. The monster's attack will cause a variable damage to the Player between 10 & 20 points (a random number).

- At the end of the round, if the Player is still alive, the game will show in the console log the results of the round

    - The status of the player: name, the damage made, his health and the number of remaining healing potion

    - The number of remaining monsters

- The status of the Monster: name and health

```
Round status
Anonymous - D: 18, H: 85, P: 2
There is still 2 monsters alive
Monster2 - D: 15, H: 82
```

Game END

- In the console log it will appear the final result

- "Victory! {Player's name} defeated all the monsters (and a victory unicode character)", if the Player killed all the monsters

- "{Player's name} died! (and a skull unicode character)", if the Monsters killed the Player

- Also will appear a set of statistics:

- Number of Player attacks

- Total damage made by the Player

- Number of Healing Potions consumed

- Number of Monster attacks

- Total damage made by the Monsters

```
Anonymous died! ☠
**********************
    GAME STATISTICS
**********************
Player weak attack: 7
Monster attack: 7
Player heals: 0
Total damage made by the Player: 95
Total damage made by the Monsters: 106
```

• And GAME OVER message
• This 3 messages, will appear in the console every 2 seconds

NONE FUNCTIONAL REQUIREMENTS.

• Create a brand new repository in GITHUB for the project.	
◦ Add my github user (xavi12p) as member of your repository	
• There is a due date to deliver the project. There is a second date (cut-off date) to deliver it, but you will receive a penalization for each day missed from the due date. (5 points over 100).	
• In case of fatal errors, the project will be returned to the student to fix the fatal errors. Every time the project is returned, you will be penalized with 10 points (over 100).	
• Split your code in layers: UI & Business Logic. You can use the MVC pattern.	Dividir el código en dos grandes bloques. 1 que se encargue de todo lo que pase en pantalla. Y otro que se encargue de bussiness logic.
◦ The UI layer will be responsible of the User Interface	
◦ The Business Logic layer will be responsible of the Core Game (the business rules of the game).	
▪ You must decouple any UI call from the Business Logic Layer	
• Use as many theoretical concepts as possible. <u>The more you use, the better grade you will get.</u>	En la segunda o tercera pasada. Refgactoring
• All the configuration game values, like for example the player's health or the min/max attack, must be coded once in the code only.	

• All the configuration game values must be easily located in the code.	
• Write DRY code (DRY stands for: Don't Repeat Yourself)	
• Name the variables and functions in english	
• Comment out your code in english	
• The project must run in Firefox	
• The project must run in the VSCode LiveServer	
• The code must be formatted as per the Javascript best practices.	
• The application cannot raise any error in the console.	
• Only the requested log info must appear in the console. Any debug information must be removed.	
• Test the Business Logic with JEST	
• Don't use languages structures/concepts that you have not covered yet in the course.	

GRADING:

- Functional requirements: 40 %
- None functional requirements: 30%
- Advanced JS usage: 30%
- Use of english: Extra of 10%

Avanced JS Functionality	Value	Check
JS Basics - Use of constants	Low	
JS Basics - Strict Equality comparisons	Low	
JS Basics - Conditional Operator (?)	Low	
JS Basics - Nullish Coalescing (??) or (||)	Medium	
JS Basics - Break / Continue loops	Low	
JS Basics - Switch statement	Low	
Function - default value in params	Medium	
Function - Some Function with comments (explaining parameters and return values)	Low	
Function - Arrow Functions / Function Expressions	High	
Function - Callback functions	High	
Object - Properties with square brackets	Low	
Object - Loop with FOR IN	High	
Object - spread operator (i.e. clone object)	Medium	
Object - Objects with methods	Medium	
Object - Constructor functions	High	
Predefined Objects - Use at least 2 Math methods	Low	
Predefined Objects - String - Use `	Low	

Avanced JS Functionality	Value	Check
Predefined Objects - Use Regexp	Low	
Predefined Objects - Use Date & Time	Low	
Arrays - Use at least 2 array methods	Medium	
Arrays - Loop with FOR OF	High	
Arrays - Rest Parameter	Low	
Arrays - Complex methods as map or reduce	Medium	
Arrays - Destructuring	Medium	
Map & Set - Use of Map	High	
Map & Set - Use of Set	High	
Scheduling - Use setTimer or setInterval	Medium	

Primero hacerlo funcionar y luego revisarlo y hacer REFACTORING.

22 DE NOVIEMBRE.