DOCUMENTATION - SPACEREVENGE - PROJET P6

Le jeu est composé de différents fichier comprenant différentes class pour certains :

-> **config.js** => ce fichier contient des variables globales valables pour l'ensemble du programme
-> **main.js** => fichier d'execution des différentes fonctions, créer dans les différents fichiers.

-> **gameboard.js** => génération de la map et de ses éléments : obstacles, armes, joueurs
-> **player.js** => ce fichier inclus une class Player qui inclus tous les paramètres d'un joueur : id, position, id de l'arme possédée par le joueur, état de santé du joueur.
-> **obstacle.js** => ce fichier inclus une class Obstacle qui inclus le paramètre principal et essentiel d'un obstacle c'est à dire sa position
-> **weapon.js** => ce fichier inclus une class Weapon composées des paramètres suivant : id, position, damage, name

-> **style.css** => style global du projet
-> **index.html** => affichage de l'interface de jeu

DESCRITPION FICHIER config.js :

Correspond au nombre de colonnes et de lignes que l'ont souhaite générer
```
let columnGameBoard = 10;
let rowGameBoard =  10;
```

Correspond à l'index initial du premier joueur
```
let indexCurrentPlayer = 0;
```

Le nombre maximum d'obstacles que l'ont souhaite générer sur le plateau
```
let numberOfObstacles = 10;
```

Le nombre maximum d'armes que l'ont souhaite générer sur le plateau
```
let numberOfWeapons = 4;
```

Le nombre maximum de joueurs possible sur le plateau de jeu
```
let numberOfPlayers = 2;
```

Vie de base des joueurs
```
let maxLife = 100;
```

Nombre de déplacements possible
```
let numberMove = 3;
```

Skin de base des joueurs à leur apparition
```
let playerSkin = "alien-fork";
```

Nombres maximum de cellules possibles
```
let numberOfGameBox = rowGameBoard * columnGameBoard;
```

Fonction pour générer un nombre aléatoire pour les armes de joueurs

```
function randomNumberWeapon() {
    return Math.floor(Math.random() * Math.floor(numberOfWeapons));
}
```

Tableau contenant l'ensemble des cellules du plateau de jeu

```
let cellsIndex = [];
```

Tableau contenant les armes générer

```
let cellsWeapons = [];
```

Tableau contenant les obstacles générer

```
let cellsObsctacles = [];
```

Tableau contenant les joueurs

```
let cellsPlayers = [];
```

DESCRIPTION FICHIER gameBoard.js :

Création de la class **Gameboard**

cette class comprends plusieurs paramètres dans le constructor notamment **rows** (ligne) et **columns** (colonnes), **idHtmlTable** (id du plateau que l'ont souhaite générer).

-> **this.rows** = **rows** (ligne)
-> **this.columns** = **columns** (colonnes)
-> **this.idHtmlTable = idHtmlTable** (id du plateau que l'ont souhaite générer)
-> **this.numberOfCell** = **0** (index des cases du jeu de plateau)
-> **this.obstacles** = **cellsObstacles** (contient le tableau comprenant tous les obstacles)
-> **this.weapons** = **cellsWeapons** (contient le tableau comprenant toutes les armes)
-> **this.players** = **cellsPlayers** (contient le tableau comprenant tous les joueurs)
-> **this.indexCurrentPlayer** = **indexCurrentPlayer** (index actuel du joueur)

Création du plateau de jeu avec la fonction **createTheGameBoard()**

La première étape consiste à créer un plateau de jeu conçu de cette façon :
-> 10 colonnes x 10 lignes

Cette fonction effectue l'action suivante :

-> création de la variable **gameboard** qui vas sélectionné l'id du ou des plateau de jeu à générer.

-> nous effectuons une boucle afin de créer 10 lignes (**for(let i = 0; i < this.rows; i++)**)
-> lors de cette boucle nous créons un element html <tr> avec une class « row »
-> ce qui nous donne <tr class='row' > …. </tr>

dans la boucle **for(let i = 0; i < this.rows; i++)** nous effectuons une seconde boucle qui vas cette fois-ci permettre la création des colonnes : **for(let i = 0; i < this.columns; i++)**
-> création de 10 colonnes **for(let i = 0; i < this.columns; i++)**
-> lors de cette boucle nous créons un element html <td> avec une class « cell »
-> ajout d'un id pour chacune des cellules **this.numberOfCell ET** tous les id sont stockés dans un tableau **cellsIndex**

-> **this.numberOfCell** est incrémenter à chaque création d'une cellules
-> ce qui nous donne <td id=**this.numberOfCell** class='cell'></td>

Exemple résultat de la fonction : **createTheGameBoard()**

```
<table class='gameBoard' id='gameBoard'>
    <tr class='row'>
        <td class='cell' id=1></td>
    </tr>
    ....
</table>
```

Fin de la fonction **createTheGameBoard()**

Début génération des obstacles sur le plateau de jeu avec la fonction **generateObstacles()**

-> génération de 10 obstacles avec une position aléatoire sur le plateau de jeu

-> création d'une variable **maxCells** qui contient le nombre max de cellules du plateau
maxCells fait la chose suivante : **rowGameBoard * columnGameBoard -1**
-> *Qu'est que ça fait ?*
-> On multiplie le nombre de lignes sur le plateau de jeu par le nombre de colonnes - 1 pour obtenir le nombre maximum de cellules sur le plateau. On pense à enlever -1 pour ne pas inclure une case en trop (en javascript les tableaux commence tous de 0).

-> création d'une variable **obstacleIndex** à -1
-> *Pourquoi -1 ?*
-> ici on initialise cette variable à -1 pour lui donner son type c'est à dire 'number'. Donc cette variable est de type 'number'

nous effectuons une boucle afin de créer les 10 obstacles **for(let i = 0; i < numberOfObstacles; i++)**
-> Dans cette même boucle nous effectuons également une autre boucle qui est *une boucle **do while**, celle-ci à la particularité d'exécuter une instruction jusqu'à ce qu'une condition de test ne soit plus vérifiée. La condition est testée après que l'instruction soit exécutée, le bloc d'instructions défini dans la boucle est donc exécuté au moins une fois.*

-> do (**obstacleIndex = Math.round(Math.random() * maxCells)**) retourne l'index d'un obstacle
-> *Qu'est ce que ça fait exactement ?*
-> **Math.round :** permet d'arrondir un nombre
-> **Math.random() :** renvoie un nombre entre 0 et 1
-> **maxCells :** nombres maximum de cellules
-> on multiplie un nombre aléatoire entre 0 et 1 avec le nombre maximum de cellules sur le plateau de jeu puis on arrondi le nombre pour obtenir un index d'obstacle

-> while (**!this.cellIsFree(obstacleIndex)**) : tant que la cellule est libre on peut ajouter un obstacle

-> **$("td#"+obstacleIndex).addClass("obstacle");**

une fois que l'ont obtient l'index pour un obstacle on peut lui rajouter la class obstacle pour donner du style à l'obstacle.

-> on déclare la variable **obstacle** qui contient l'objet **Obstacle** avec son paramètre de **position** = **obstacleIndex**

-> et enfin on ajoute la variable obstacle dans un tableau qui contiendra ainsi tout les obstacles présent sur le plateau de jeu

Fin de la fonction **generateObstacles()**

Début génération des armes sur le plateau de jeu avec la fonction **generateWeapons()**

-> génération de 4 armes avec une position aléatoire sur le plateau de jeu

-> cette fonction est très similaire à la précédente

-> création d'une variable **maxCells** qui contient le nombre max de cellules du plateau
-> *Qu'est ce que ça fait ?*
-> Cette variable contient exactement la même chose que dans la fonction précédente.
Rappel : *« On multiplie le nombre de lignes sur le plateau de jeu par le nombre de colonnes - 1 pour obtenir le nombre maximum de cellules sur le plateau. On pense à enlever -1 pour ne pas inclure une case en trop (en javascript les tableaux commence tous de 0).»*

-> création d'une variable **weaponIndex** à -1
-> Pourquoi - 1 ?
-> Ici aussi nous avons besoin de définir le type de cette variable à 'number' c'est pour cela qu'ont lui donne -1.

nous effectuons une boucle afin de créer 4 armes **for(let i = 0; i < numberOfWeapons; i++)**
-> Dans cette même boucle nous effectuons une autre boucle qui est une boucle do while
-> do (**weaponIndex = Math.round(Math.random() * maxCells)**) retourne l'index d'une arme
-> *Qu'est ce que ça fait exactement ?*
-> **Math.round :** permet d'arrondir un nombre
-> **Math.random() :** renvoie un nombre entre 0 et 1
-> **maxCells :** nombres maximum de cellules
-> on multiplie un nombre aléatoire entre 0 et 1 avec le nombre maximum de cellules sur le plateau de jeu puis on arrondi le nombre pour obtenir l'index d'une arme

-> while (**!this.cellIsFree(weaponIndex)**)

-> une fois que l'ont obtient l'index pour une arme :
-> création de la variable **randomWeapon** qui contient la fonction **randomNumberWeapon()**, c'est cette fonction qui permet d'appeler un nombre aléatoire entre 1 et 4.
-> création de la variable **weapon** qui contient l'objet **Weapon**, avec plusieurs paramètres, un **id**, un **index**, et (**randomWeapon**) un nombre aléatoire qui vas permettre de pouvoir générer une arme.
-> ensuite la variable **weapon** est stockée dans un tableau this.weapon

-> ont créer une variable **imgWeapon** qui contient un élément html '**<img />**'
-> ont créer ensuite une autre variable **imgWeaponUrl** qui contient cette fois-ci l'url de l'arme qui vas être générée.
-> ont ajoute une class « **weaponRandom** » à l'élément **imgWeapon**
-> ont ajoute l'attribue src avec l'url **imgWeaponUrl** à l'élément **imgWeapon**
-> et pour finir on affiche l'arme dans une case du plateau

Fin de la fonction **generateWeapons()**

Début ajout des joueurs sur le plateau **addPlayer()**

-> Ajout de 2 joueurs sur le plateau de jeu et affectation d'une arme de base au joueur

-> création d'une variable **maxCells** qui contient le nombre max de cellules du plateau
-> *Qu'est ce que ça fait ?*
-> Cette variable contient exactement la même chose que dans les fonctions précédentes.
Rappel : *« On multiplie le nombre de lignes sur le plateau de jeu par le nombre de colonnes - 1 pour obtenir le nombre maximum de cellules sur le plateau. On pense à enlever -1 pour ne pas inclure une case en trop (en javascript les tableaux commence tous de 0).»*

-> création d'une variable **playerIndex** à -1
-> Pourquoi - 1 ?
-> Ici aussi nous avons besoin de définir le type de cette variable à 'number' c'est pour cela qu'ont lui donne -1.

nous effectuons une boucle afin de créer nos 2 joueurs **for(let i = 0; i < numberOfPlayers; i++)**
-> dans cette boucle nous effectuons une autre boucle do while
-> do : (**playerIndex = Math.round(Math.random() * maxCells)**) : retourne l'index d'un joueur.
-> *Qu'est ce que ça fait exactement ?*
-> **Math.round :** permet d'arrondir un nombre
-> **Math.random() :** renvoie un nombre entre 0 et 1
-> **maxCells :** nombres maximum de cellules
-> on multiplie un nombre aléatoire entre 0 et 1 avec le nombre maximum de cellules sur le plateau de jeu puis on arrondi le nombre pour obtenir l'index d'un joueur

-> while : (**!this.cellIsFree(playerIndex**)) tant que la cellule est libre on peut ajouter un obstacle

-> création d'une variable **firstWeapon** qui contient le nom de l'arme de base affectée au joueur (« Fork »)
-> création d'une variable weapon qui contient l'objet **Weapon** suivis de ses paramètres : **this.weapons.length + 1** (cela signifie que l'ont compte le nombre d'entrée dans le tableau, et on ajoute une nouvelle entrée pour notre arme de base), playerIndex (l'emplacement actuel du joueur), **firstWeapon** (et enfin le nom de l'arme de base affectée au joueur))
-> création d'une variable player qui contient l'objet Player suivis des paramètres suivant : **i** (i correspond à l'id du joueur), **playerIndex** (c'est la position du joueur sur le plateau de jeu), weapon.id (weapon.id correspond à l'id de l'arme associé au joueur)

-> création de la variable **imgWeapon** qui contient un élément html '<img />'
-> on créer une variable **imgWeaponUrl** qui contient l'url de l'image appartenant à l'arme de base générer à la création des joueurs c'est à dire 'fork'.
-> avant d'afficher cela nous devons créer 1 attribut class '**weaponStartThePlayer**'
-> Et un autre attribut src accompagner de la variable **imgPlayerUrl** pour lier l'élément html <img /> à une image.
-> affichage de l'arme dans la même cellule que le joueur

-> **le code est très similaire car nous voulons faire apparaitre notre joueur au même endroit que notre arme.**

-> création de la variable **imgPlayer** qui contient un élément html '<img />'
-> on créer une variable **imgPlayerUrl** qui contient l'url de l'image appartenant à l'alien qui représente un joueur.
-> avant d'afficher cela nous devons créer 1 attribut class '**playerWithStartWeapon**'
-> Et un autre attribut src accompagner de la variable **imgPlayerUrl** pour lier l'élément html <img /> à une image.
-> affichage du joueur dans la même cellule que l'arme

Fin de la fonction **addPlayer()**

Début vérification de la disponibilité des cellules du jeu de plateau **cellIsFree(index)**
-> *Le paramètre index, il correspond à quoi ?*
-> ce paramètre fais référence à **weaponIndex, obstacleIndex, playerIndex**
-> le paramètre index dans cette fonction correspond à l'index de l'arme, joueur ou obstacle.

-> création d'une variable **isfree** que l'ont passe à **true**
-> on part du principe que la case est dispo

```
->   isfree = this.obstacles.filter(ob => ob.position === index).length > 0 ? false : isfree;
->   isfree = this.weapons.filter(wp => wp.position === index).length > 0 ? false : isfree;
->   isfree = this.players.filter(pl => pl.position === index).length > 0 ? false : isfree;
```

-> cela nous permet de filtrer les index pour voir si ils ne sont pas déjà utilisé
-> *Qu'est ce que ça fait exactement ?*
-> *Exemple : vérifie si l'index de l'obstacle est supérieur à 0*
SI** c'est le cas on dit que l'index est pris donc il passe à **false
SINON** on dit que l'obstacle est libre **isfree = true

Fin de la fonction **cellIsFree(index)**

Début vérification des positions des joueurs par rapports aux obstacles **cellIsMovable(index)**

-> création d'une variable **isMovable** que l'on passe à true

```
-> isMovable = this.obstacles.filter(ob => ob.position === index).length > 0 ? false : isMovable;
-> isMovable = this.players.filter(pl => pl.position === index).length > 0 ? false : isMovable;
```

-> cela vas nous permettre de filtrer les index entre les obstacles et les joueurs
Fin de la fonction **cellIsMovable(index)**

Donne la possibilité au joueur de ce déplacer sur le plateau de jeu
moveInDirection(playerPosition, step)

->plusieurs paramètres on été ajoutés a cette fonction
-> **playerPosition** = position du joueur
-> **step** = case du plateau de jeu
-> **colorStep** = couleur sur la case du plateau de jeu

-> déclaration de 2 variables
-> **countStep** = step (on remplace 'i' par countStep et on lui passe l'argument step)
-> **moveInStepIsPossible**

-> première condition de test pour vérifier les cases
if(step < 0) = si step est inférieur à 0
-> on calcule le nombre de case ou le joueur peut se déplacer par rapport au nombres de mouvements possibles.
Sinon
-> si l'ont dépasse le nombres de mouvements possibles ont arrête

-> Tant que les mouvements dans les cases sont possible
-> while(moveInStepIsPossible)

-> dans cette boucle nous allons principalement vérifier les déplacements horizontaux du joueur afin qu'il ne dépasse pas les limites de la map

-> création de 2 variables
-> **gameBoardLine = playerPosition % this.column**
-> **mapLimitation**

-> condition pour vérifier les limites de la map à droite et à gauche du joueur

-> condition pour appliquer du style dans les cases dans lesquelles le joueur peut se déplacer
(Si la position du joueur et la case est dispo et que la mapLimitation est false)
-> if(this.cellIsMovable(playerPosition + countStep) && mapLimitation)
-> création d'une variable
(on ajoute la position du joueur + la case compter pour donner un mouvement finale 'resultMove')
-> resultMove = playerPosition + countStep
-> ajout de style dans la cellule ou le joueur peut se déplacer
-> Sinon
-> break

-> incrémentation des cases dans lesquelles le joueur peut bouger dans la variable countStep

Fin de la fonction **moveInDirection(playerPosition, step)**