Projet : Créez un jeu de plateau tour par tour en JS

https://saijitsu.github.io/thesurvivorssword

Nicolas TURLET

Énoncé du projet:

Ce projet consiste à créer un jeu en ligne en JavaScript dans lequel 2 joueurs évoluent chacun leur tour pour s'affronter dans un duel à mort.

Etape 1: Le jeu va générer une carte pour le duel

1. des **cases vides**
2. des **cases inaccessibles**
3. un nombre limité d'**armes** (4 maximum) sera placé aléatoirement et pourra être récolté par les joueurs qui passeraient dessus
4. **Les joueurs**

Etape 2: Les placements

Doit exister au moins **4 types d'arme** dans le jeu, avec des d**égâts différents**. L'**arme par défaut qui équipe les joueurs doit infliger 10 points de dégâts**. Chaque arme a u**n nom et un visuel associé**.

Le **placement** des deux joueurs est lui aussi **aléatoire** sur la carte au chargement de la partie. Ils **ne doivent pas se toucher** (ils ne peuvent pas être côte à côte).

A chaque tour, un joueur peut se **déplacer** d'**une à trois cases** (**horizontalement** ou **verticalement**) avant de terminer son tour. Il ne peut évidemment pas passer à travers un obstacle.

Si un joueur passe sur une case contenant une arme, il **laisse son arme actuelle sur place et la remplace par la nouvelle**.

Etape 3 : le combat !

Si les **joueurs se croisent sur des cases adjacentes** (horizontalement ou verticalement), un **combat à mort** s'engage.

Lors d'un combat, le fonctionnement du jeu est le suivant :

- ❏ Chacun attaque à son tour (**tour par tour**)
- ❏ Les dégâts infligés dépendent de l'arme possédée par le joueur (**dégâts en fonction de l'arme**)
- ❏ Le joueur peut choisir **d'attaquer** ou de se **défendre** contre le prochain coup
- ❏ Lorsque le joueur se **défend, il encaisse 50% de dégâts en moins** qu'en temps normal
- ❏ Dès que les points de **vie** d'un joueur (initialement à 100) tombent **à 0** , celui-ci a perdu. **Un message s'affiche et la partie est terminée**.

Livrables: Code HTML/CSS/JS et Jquery du projet

Compétences:

- ★ Mettre en oeuvre la bibliothèque jQuery dans une application web
- ★ Concevoir une architecture d'application JavaScript réutilisable
- ★ Développer une application JavaScript orientée objet (JSOO)


THE LEGEND OF
THE SURVIVOR'S SWORD
Player 1 : Link
Heal Points:
100/100
Equiped Weapon: PEASANT SWORD Power: 10
Player 2: Dark Link
Heal Points:
100/100
Equiped Weapon: PEASANT SWORD Power: 10
Hey! Listen!
RESTART GAME
msi

Compatibilité

Compatible avec les navigateurs actuels dans leurs dernières versions (Ordinateur et Mobile). Il s’agit d'offrir aux utilisateurs le meilleur contenu en fonction de leur appareil de navigation.

Versioning

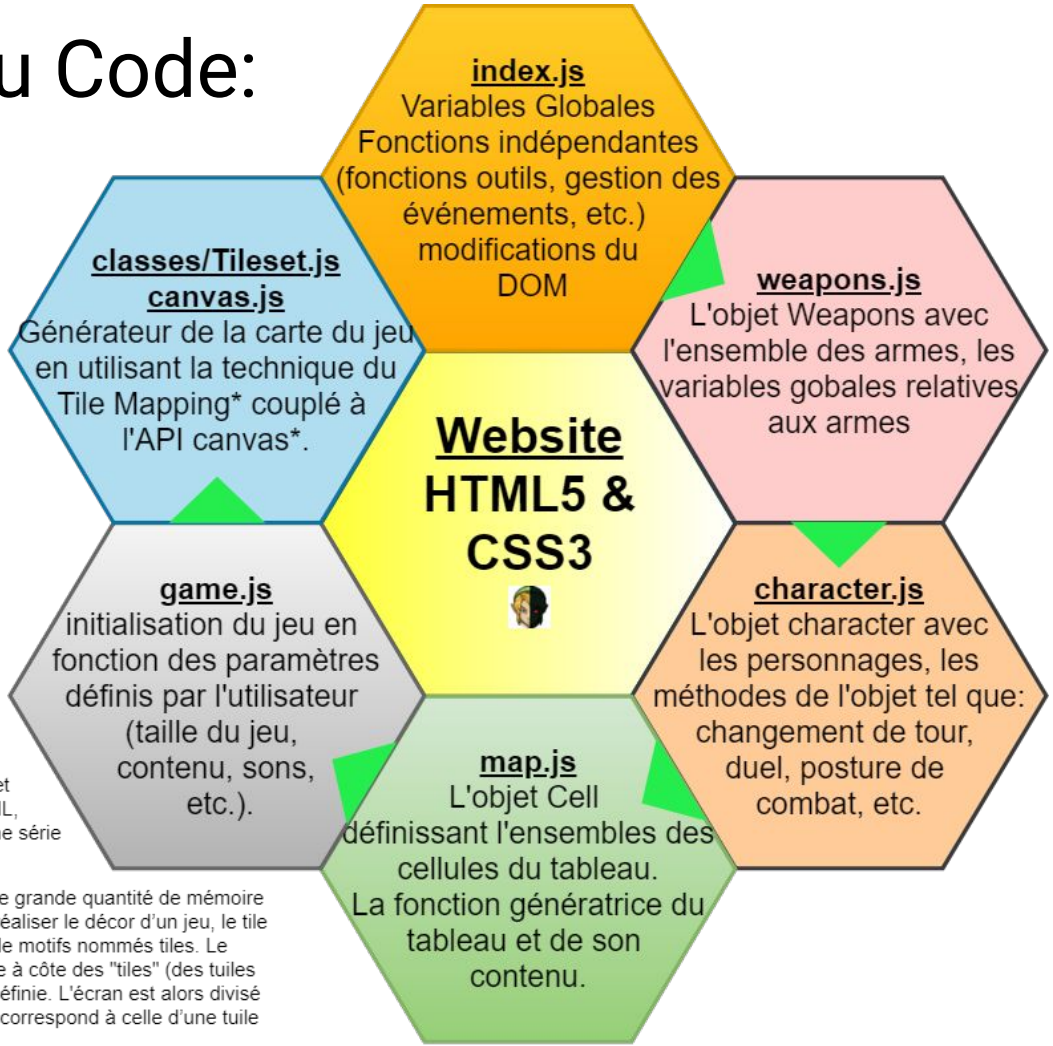
Versioning: Anglais

Langages et Librairie

Javascript (ES5), HTML5, CSS3

Jquery (DOM & Interface)

Architecture du Code:



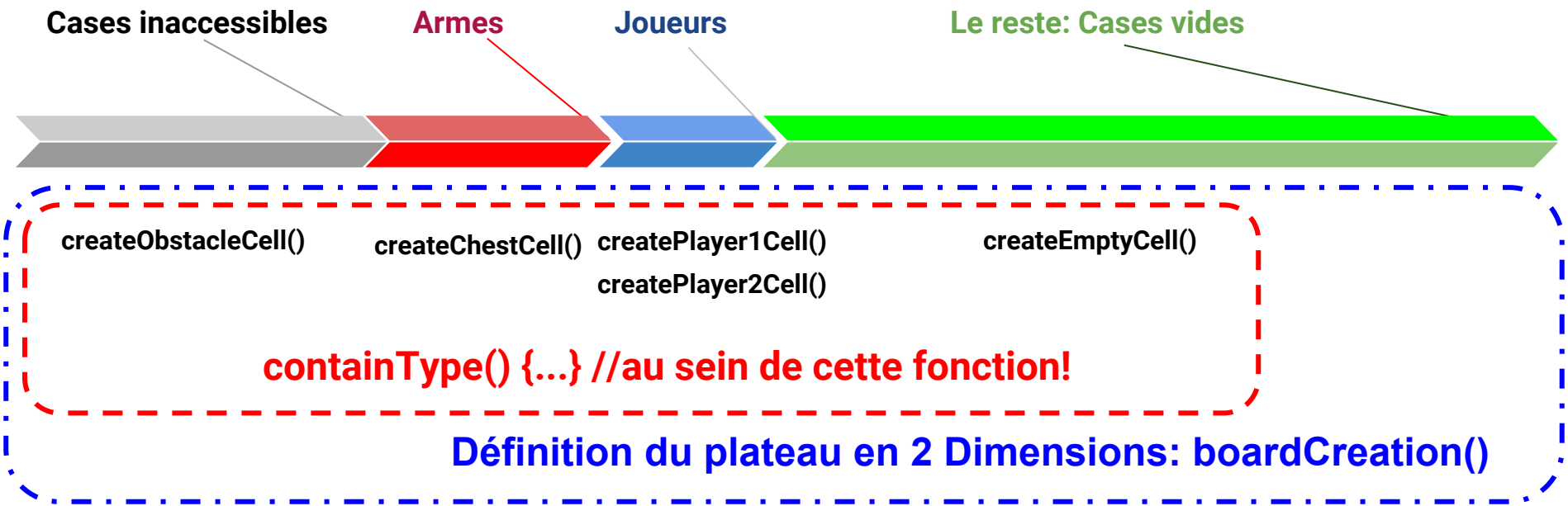
Canvas: zone de dessin dont la *hauteur* et la *largeur* sont définies dans du code HTML, JavaScript permet d'accéder à l'aire via une série complète de fonctions de dessins

Tile Mapping: Afin d'éviter de gaspiller une grande quantité de mémoire avec l'utilisation de grandes images pour réaliser le décor d'un jeu, le tile mapping va reconstruire le décor à partir de motifs nommés tiles. Le concept de Tile Mapping est de placer côte à côte des "tiles" (des tuiles en anglais) dans une fenêtre de taille prédéfinie. L'écran est alors divisé en une grille dont la taille de chaque case correspond à celle d'une tuile (les images étant des **sprites**).

Etape 1: Le jeu va générer une carte pour le duel

A l'initialisation du jeu, va être généré un tableau avec un nombre de cases déterminées (***totalCells***) résultant des paramètres de l'utilisateur.

La fonction ***createRandomCellList()*** va générer un tableau (***randomList***) qui va ensuite être mélangé. L'intérêt est de générer le plateau de jeu (board) en limitant le nombre de calculs nécessaires à sa création afin d'optimiser les performances.



La fonction **boardCreation()** va alors pouvoir générer un plateau de jeu (**board**) classique avec des coordonnées **x pour les colonnes** et **y pour les lignes**. La fonction va récupérer les **valeurs** de ***randomList*** en fonctions des besoins grâce à un algorithme permettant l'identification des cases:

currentCellPosition = x + y * board.length

Exemple: les 10 premières valeurs seront des cases obstacles, les 4 suivantes des coffres contenants des armes, les deux suivantes le joueurs 1 et 2, et le reste sera composé de cases vides.

Création board 2D:

Utilisation d'une boucle for pour alimenter le contenu:

définition du contenu:

alimentation cellList, pour conserver les valeurs X et Y qui ne change pas dans l'objet cell:

```
function boardCreation() {
  // Board creation
  board = new Array(columns);
  for (var i = 0; i < rows; i++) {
    board[i] = new Array(rows);
  }
  // Board Contain
  for (y = 0; y < (board.length); y++) {
    for (x = 0; x < (board.length); x++) {
      currentCellPosition = x + y * board.length;
      // create the object and store a reference to the cell object
      var containTypeCell = containType(x, y, board.length, board, currentCellPosition);
      board[y][x] = containTypeCell;
      cellList.push(containTypeCell);
    }
  }
  console.log(board);
}
```

Application de la méthode:

```
47,54,50,6,69,33,40,72,55,76,1,57,7,87,97,38,95,65,15,63,84,75,8,73,20,19,94,22,66,23,2,4,28,85,  index.js:94
83,9,70,49,46,79,14,41,32,25,10,78,56,92,91,3,93,81,62,12,18,86,30,37,36,34,26,31,77,74,5,60,16,58,98,0,35,29
,45,88,52,64,17,71,53,11,13,42,21,68,80,89,67,27,96,99,59,90,61,48,82,43,39,24,44,51

▼Array(10)                                                                                    map.js:34
  ▶0: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶1: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶2: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶3: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▼4: Array(10)
    ▶0: Cell {contain: 1, numberCell: 40, y: 4, x: 0, freeCell: false, …}
    ▶1: Cell {contain: 0, numberCell: 41, y: 4, x: 1, freeCell: true, …}
    ▶2: Cell {contain: 0, numberCell: 42, y: 4, x: 2, freeCell: true, …}
    ▶3: Cell {contain: 0, numberCell: 43, y: 4, x: 3, freeCell: true, …}
    ▶4: Cell {contain: 0, numberCell: 44, y: 4, x: 4, freeCell: true, …}
    ▶5: Cell {contain: 0, numberCell: 45, y: 4, x: 5, freeCell: true, …}
    ▶6: Cell {contain: 0, numberCell: 46, y: 4, x: 6, freeCell: true, …}
    ▶7: Cell {contain: 1, numberCell: 47, y: 4, x: 7, freeCell: false, …}
    ▶8: Cell {contain: 0, numberCell: 48, y: 4, x: 8, freeCell: true, …}
    ▶9: Cell {contain: 0, numberCell: 49, y: 4, x: 9, freeCell: true, …}
     length: 10
    ▶__proto__: Array(0)
  ▶5: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶6: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶7: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶8: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
  ▶9: (10) [Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell, Cell]
   length: 10
  ▶__proto__: Array(0)
```

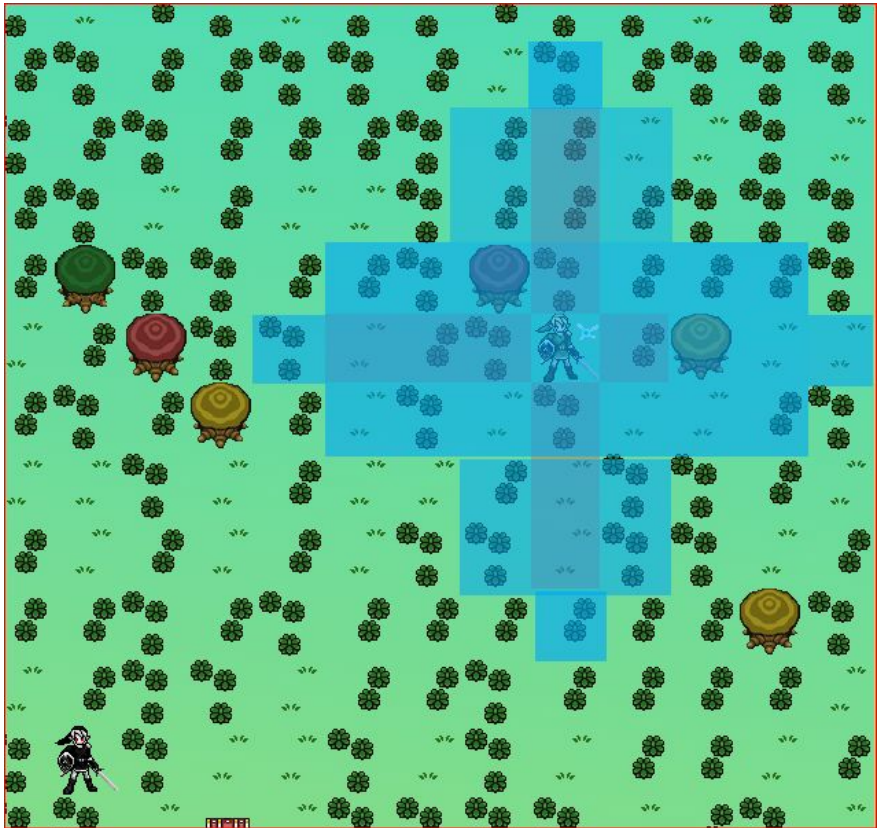
nombre de la cellule =
7 + 4 * 10 = 47

Colonne = 7
Ligne = 4
plateau = 10x10 (longueur = 10)

Etape 2: Les placements

9 types d'arme existent dans le jeu, avec des **dégâts différents**. L'**arme par défaut qui équipe les joueurs inflige 10 points de dégâts**. Chaque arme a u**n nom et un visuel associé**.

Le **placement** des deux joueurs est lui aussi **aléatoire** sur la carte au chargement de la partie (grâce à l'utilisation du tableau ***randomList***).



A. Placements des joueurs

Les joueurs ne peuvent pas apparaître l'un à côté de l'autre:

La fonction **characterNear()** gère le risque de contact:

Vont être vérifier les si les valeurs des **36 cases en bleu** ne correspondent pas à **randomList[valeur retenu pour le Joueur 1]**, si aucune correspondance n'est trouvée, alors le **joueur 2** va être placé sur la carte.

Le cas échéant la méthode de character **changeDropArea()** va définir un nouvelle emplacement pour le Joueur 2 en vérifiant encore **characterNear()** et ce jusqu'à ce qu'aucun contact n'ai été trouvé.

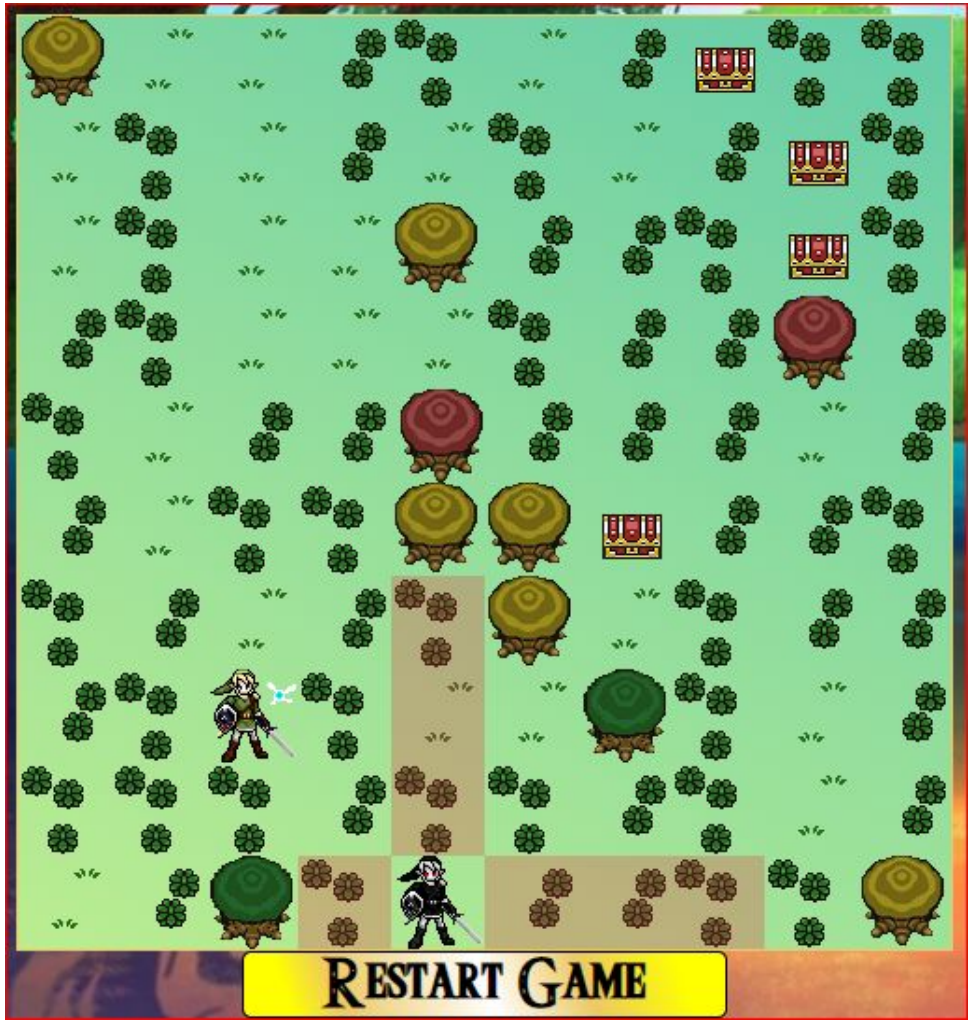
B. Le déplacements des joueurs

A chaque tour, un joueur peut se **déplacer** d'**une à trois cases** (**horizontalement** ou **verticalement**) grâce à la méthode de character **tripArea()** le tour se termine à dès qu'une case valide est sélectionnée.

Il ne peut évidemment pas passer à travers un obstacle, **tripArea()** va vérifier une à une les coordonnées des cases à gauche du joueurs, puis en bas, à droite et enfin en haut à l'aide d'une boucle ***for***.

A chaque début de tour la Méthode **tripArea()** alimente l'Array **highLightning[]** avec les valeurs des cellules ou le joueur peut se déplacer.

En fin de tour, la fonction **clearCurrentPlayerHighLightning()** va supprimer la surbrillance des cases ou le déplacement est possible.



C. La gestion des armes ramassées

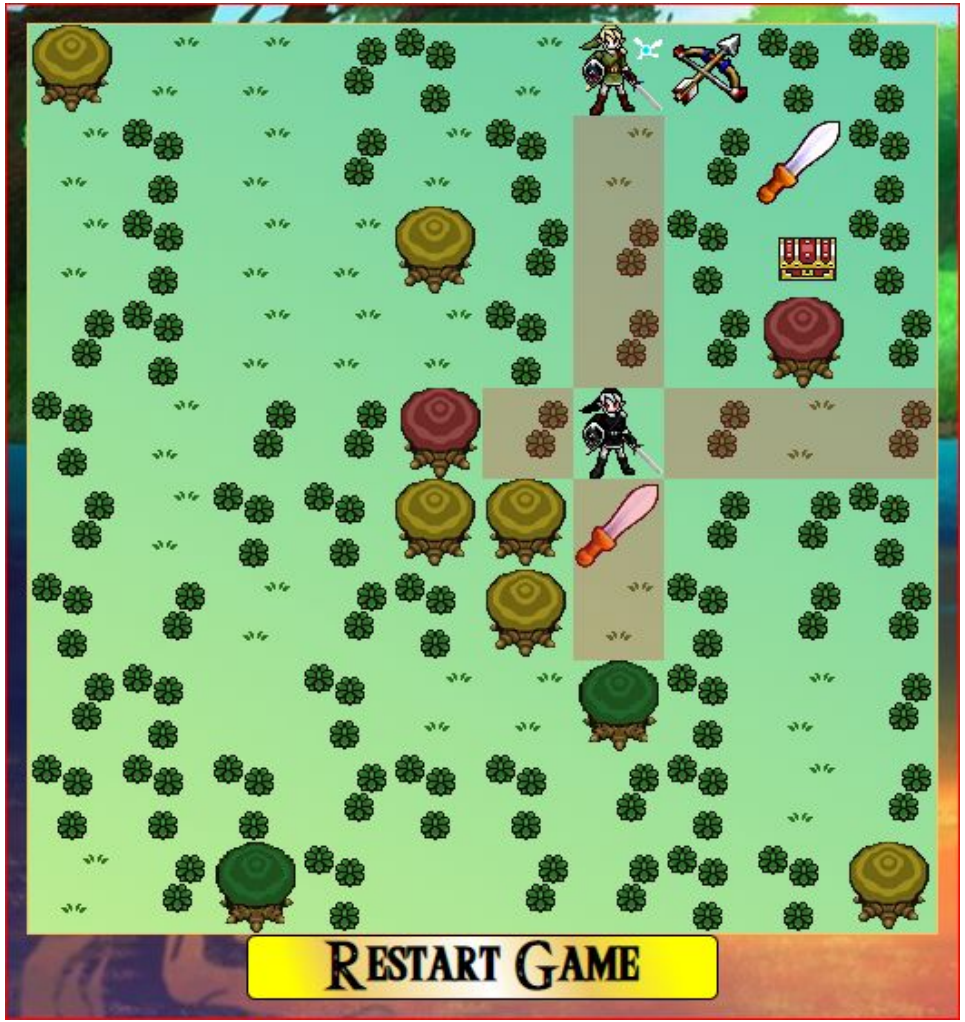
Si un joueur tombe sur une case contenant une arme, l'écouteur d'événement va utiliser la fonction **clickChestCell()** pour générer le changement d'arme.

Le visuel de coffre étant possible grâce à l'objet Weapons et sa propriété: **this.worn = worn qui va distinguer les armes déjà équipées et les autres.**

Le joueur **laisse son arme actuelle sur place et la remplace par la nouvelle**.

this.weapon = weapon; // Il va équiper l'arme trouvée dans le coffre
this.weaponToDeposited = weaponToDeposited; // Il va stocker l'arme à déposer au prochain déplacement.

L'arme sera déposée au prochain tour via les fonctions **clickEmptyCell()** si la case est vide ou **clickChestCell()** s'il tombe à nouveau sur un coffre.



Etape 3 : le combat !

Si les **joueurs se croisent sur des cases adjacentes** (horizontalement ou verticalement) avec la méthode de character **playersCollision()**, vont être vérifiées à chaque fin de tour(**Character.prototype.changeOfPlayerSTurn()**) les cases du haut, bas, droite et gauche. S'il y a un contact à alors un **combat à mort** s'engage (**Character.prototype.duel()**).

Lors d'un combat: **fight()**, le fonctionnement du jeu est le suivant :

- ❏ Chacun attaque à son tour (**tour par tour: Character.prototype.changeOfPlayerSDuelTurn()**):

- ❏ Les dégâts infligés dépendent de l'arme possédée par le joueur (**dégâts en fonction de l'arme**):

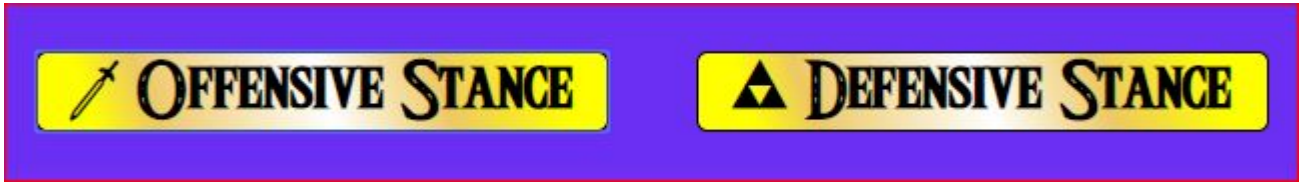
On va vérifier la puissance de l'arme du joueur à chaque fois qu'il va infliger un dommage au joueur adverse.

```
Character.prototype.dommageDeal = function () {
  return this.weapon.power;
};
```

❏ Le joueur peut choisir **d'attaquer** ou de se **défendre** contre le prochain coup:

Character.prototype.isDefensiveStance() // On va vérifier à chaque tour la posture de combat du joueur

Character.prototype.isOffensiveStance()

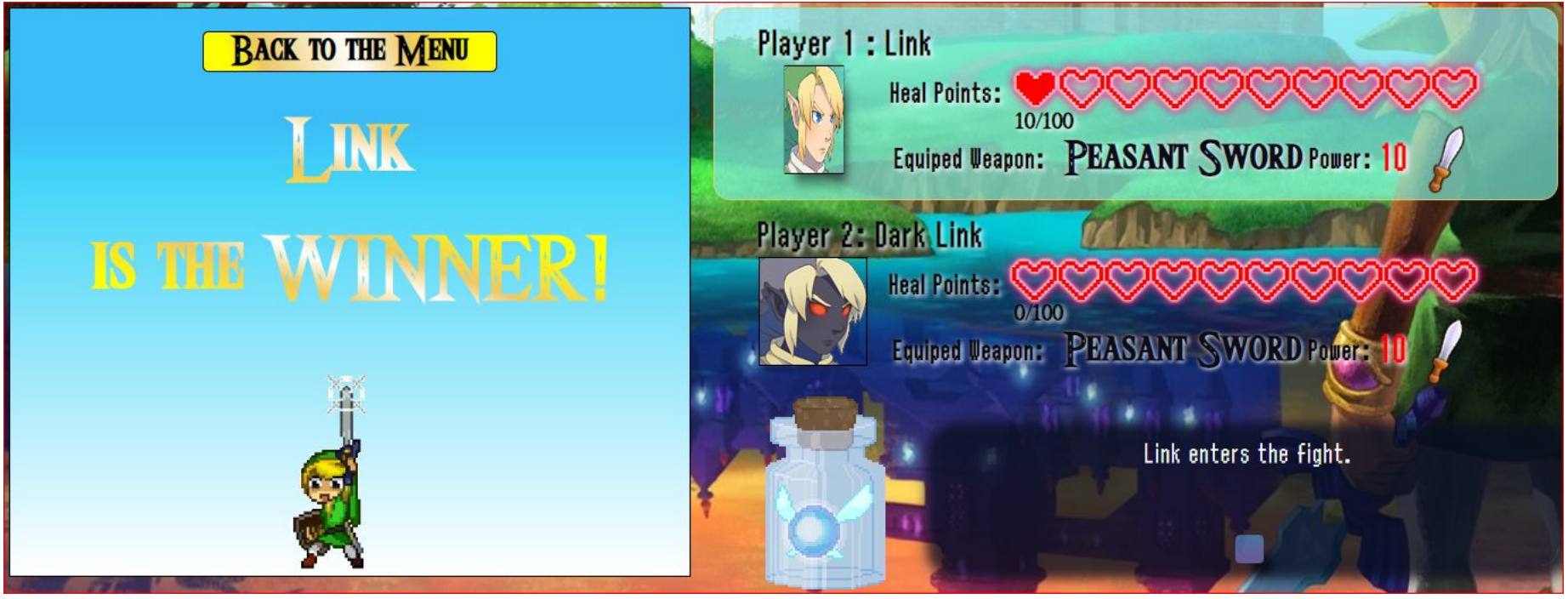


❏ Lorsque le joueur se **défend, il encaisse 50% de dégâts en moins** qu'en temps normal:

Application d'un agorithme simple: opponentPlayer.heal = opponentPlayer.heal - currentPlayer.dommageDeal() / 2

❑ Dès que les points de **vie** d'un joueur (initialement à 100) tombent **à 0** , celui-ci a perdu. **Un message s'affiche et la partie est terminée**:

```
function deadOpponent() {
  updateStatistics()
  changeTrack(victoryMusic)
  setTimeout(function () {
    shakeBottleImage()
    $("#chat-text").text(opponentPlayer.name + " is unconscious! " +
currentPlayer.name + " is the winner!")
  }, 2000);
}
```



API CANVAS

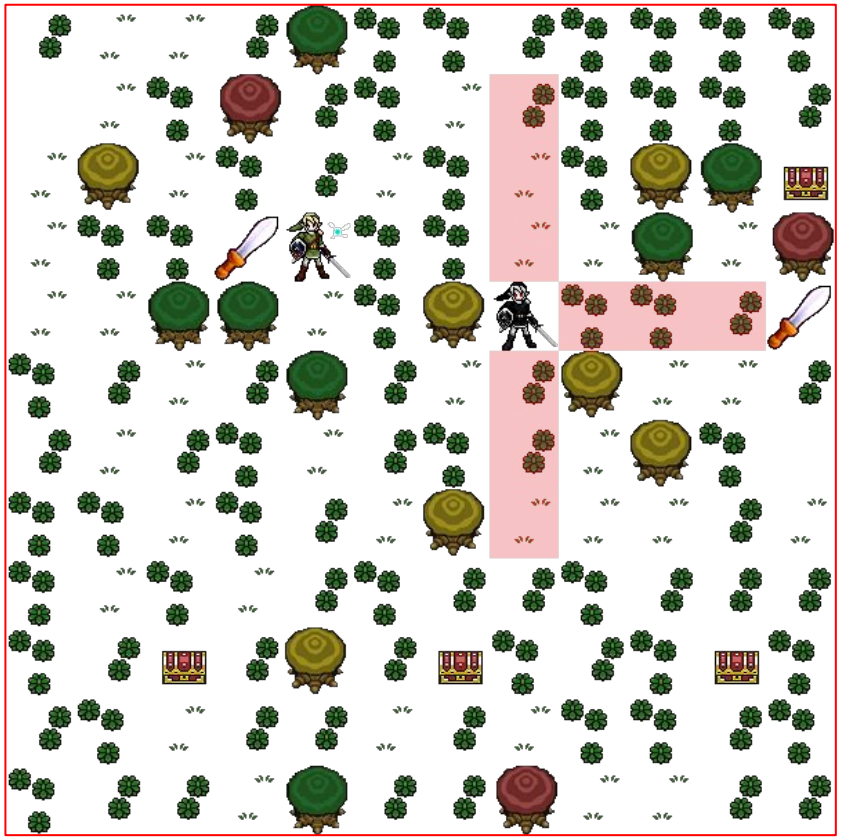
➔ Une Actualisation à chaque tour sur le plateau!

- ❏ il s'agit d'un standard développé par le W3C
- ❏ il fonctionne très bien en osmose avec les autres standards (HTML, JavaScript)
- ❏ il est performant et accéléré matériellement sur la plupart des navigateurs et systèmes
- ❏ il est bien pris en charge sur les mobiles

➔ Couplé avec du Tile Mapping

Le **board** nous offre un définition du monde schématique avec des Cellules. Nous allons donner une **Tile** (tuile) pour chaque type de Cellules. L'ensemble du plateau va tenir sur un **TileSet** qui ne se chargement qu'une fois pour un taille infime:

ici 18 Tile, pour un plateau pouvant en contenir jusqu'à 400 Tile!



Fonctionnalités /Feature

- ★ **Sons:** Musique de menu, aventure, duel, victoire et voix de Navi pour les annonces console.
- ★ **Console:** donne des indications sur la situation de jeu, le joueur actuel, la vie de la cible, les positions, etc.
- ★ **Box:** indiquant le joueur en cours avec des effets visuels spécifique à chaque joueur.
- ★ **Design en fonction du joueur en cours:** changement pour les duels, la fée dans la bouteille, le message de victoire.
- ★ **Background du plateau différent à chaque partie: getGradiantBackground()**
- ★ **Responsive: Flex Box!**

Compétences:

★ Mettre en oeuvre la bibliothèque jQuery dans une application web

Les fonctions essentielles à l'interface visuelle : victory(), currentPlayerIs(), fight(), updateStatistics() utilisent Jquery. C'est également le cas pour le son du jeu, l'initialisation du jeu, etc.

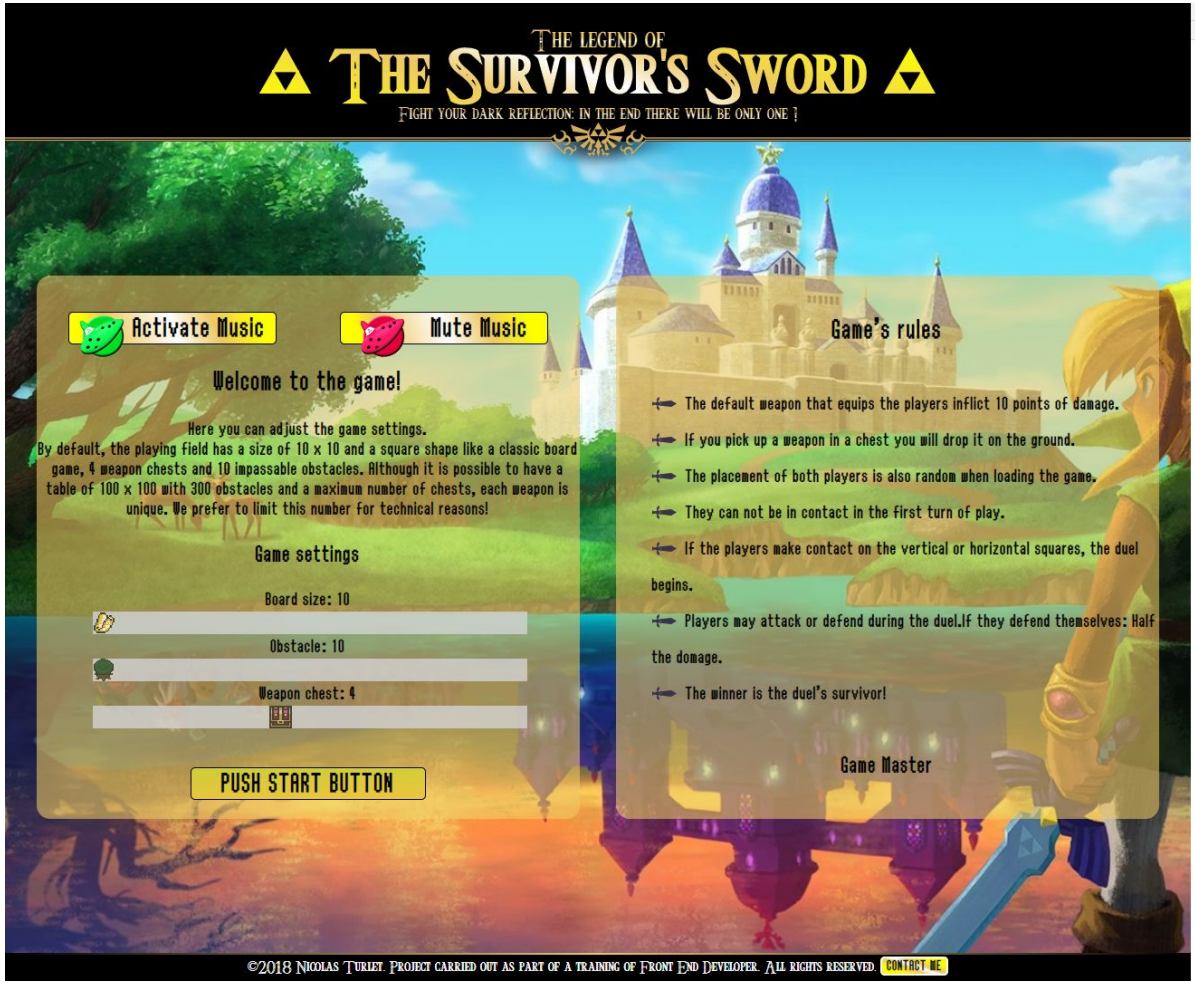
★ Concevoir une architecture d'application JavaScript réutilisable

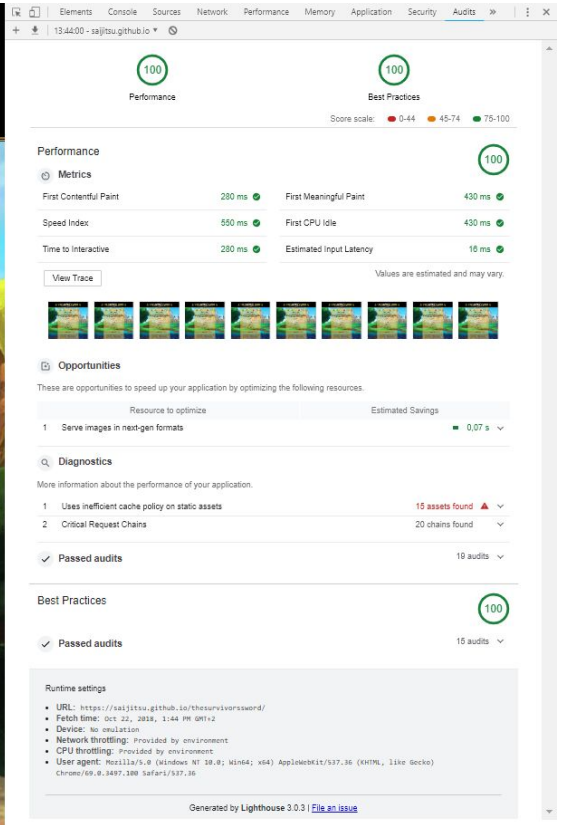
L'architecture est conçue dans cet objectif:
index.js offre des fonctions outils utilisées dans l'ensemble de l'application.
canvas.js ou **Tileset.js** ne peuvent voir les images utilisées changées facilement.
Les objets **Map, Character, Weapons** sont facilements réutilisables dans un jeu de plateau par exemple.

★ Développer une application JavaScript orientée objet (JSOO)

L'ensemble des objets **Map, Character, Weapons** coopèrent ensemble et chaque objet va être en mesure de communiquer avec les autres. **Map, Character, Weapons** constituent des entités indépendantes avec un rôle distinct.

Les performances du website





DÉMONSTRATION !