Etape 1 : génération de la carte

Bouton start pour lancer toutes ces fonctions, function startGame

function **generateTable** : double For

function **displayTable** : s’appuie sur generateTable, affiche la grille

Objet **joueur** : nom, santé, arme, visuel  
héritage pour 2 joueurs différents

function **spawnPlayers** :2 joueurs, pas côte à côte. Au hasard parmi les coordonnées de generateTable

function **startWeapons** : ajoute l’objet arme de départ à chaque objet joueur

Objet **arme** : nom, dégâts, visuel  
héritage pour 4 armes différentes

function **spawnWeapons** : 2 armes parmi les 3 autres types que défaut, pas sur cases joueurs

2 objets armes : armes de départ, et les autres.

function **blockSquare** : Math, 20% de chance, chaque case sans joueur ou arme soit un obstacle

function **emptySquare** : le reste des cases disponibles

Un canvas qui va inclure l’espace de jeu.

generateTable : génère un tableau à deux niveaux avec 2 boucles For, itération, de 10 x 10.  
X et Y (deux niveaux du tableau) en arguments de la fonction, pour le nombre de lignes et colonnes.  
return ce tableau à double niveau. Possibilité de créer un plateau plus grand.

displayTable : affiche le tableau dans le navigateur.  
double boucle For pour itérer dans le tableau de generateTable, et afficher lignes et colonnes avec la balise Table.

spawnPlayers : itère à travers le tableau d’objets joueurs pour les ajouter sur la map. Possibilité d’avoir plus de 2 joueurs.

spawnWeapons : idem. Un tableau d’objets armes pour pouvoir en rajouter.

blockSquare : utilisation de math.floor et math.random x10 sur les cases sans joueur ni arme, si chiffre entre 0 et 8, la case reste vide et donc disponible pour le mouvement.

emptySquare : utile ?

Une fonction englobante createGame() ?

Etape 2 : les mouvements

function **detectMovement** : itération sur la grille, chaque case vide ou avec une arme est disponible au mouvement

function **displayMovement** : affiche les cases disponibles au mouvement

function **doMovement** : déplace le personnage en fonction du click sur une case disponible

Retire joueur de la case d’origine et l’ajoute à la case d’arrivée

function **switchWeapon** : si un joueur passe sur une arme, elle remplace l’arme qu’il a en main

Ajout de cette arme dans son objet, et de la sienne sur la case

detectMovement : itération sur la grille pour connaitre les cases disponibles au mouvement, par rapport à la position du joueur dont c’est le tour. 1 à 3 cases de mouvement horizontal ou vertical en ligne droite. Un obstacle bloque le mouvement.  
Argument : le joueur, pour sa position.  
Return : les valeurs disponibles au mouvement (tableau ?)

displayMovement : utilise le return de detectMovement pour affiche les cases disponibles au mouvement.

doMovement : gère le déplacement du joueur à l’aide d’un clique sur la case disponible souhaité.  
Retire le joueur de la case d’origine et l’ajoute à la case d’arrivée.  
Si (If statement) le joueur passe par une case avec une arme, appelle function switchWeapon().  
Itération sur les cases disponibles au mouvement pour détecter s’il y a une arme (nouvelle fonction) ?  
Le tour passe au joueur suivant, avec detectMovement().

Une fonction englobant detect / display / doMovement et switchWeapon, appelée doTurn().  
Si les joueurs ne sont pas juxtaposés à la fin d’un tour, passe au joueur suivant en bouclant sur elle-même. Sinon passe sur la function doFight().

Joueur, argument de doTurn() ou de chaque fonction imbriquée ?

function **detectFight** : à la fin de chaque mouvement, détection pour voir si un autre joueur adjacent. Si oui, début combat.

Une seule issue : victoire d’un joueur, déplacements impossibles

Etape 3 : le combat

function **startGame** : refait apparaitre le bouton start, function startGame

function **endFight** : si un joueur atteint 0 points de vie, la partie s’arrête

Un message déclare l’autre vainqueur

function **defend** : évite 50% des dégâts de l’adversaire, pas d’attaque

function **attack** : inflige dégâts à l’adversaire

function **doFight** : joueur qui termine son mouvement démarre.

Choix entre attaquer et défendre

Retrait des points de vie

detectFight : si false à la fin de la fonction doTurn, prochain tour de mouvement. Si true, appelle de la fonction doFight().

doFight : deux choix avec les touches du clavier A (attack) et D (defend). Ajout d’un attribut à l’objet Joueur (statut : A ou D), booléen ?  
Si A, appelle la fonction attack(). Retrait des points de vie à l’adversaire. Si D pour l’adversaire, 50% de dégâts en moins.  
Si D, appelle la fonction defend().

endFight : prend en argument l’attribut santé des joueurs, si un joueur tombe à 0 ou en-dessous, la partie s’arrête et l’adversaire est déclaré vainqueur.

Ou version plus évolutive :

endFight fait appelle à removePlayer() qui retire le sprite du joueur à 0 de santé, puis regarde combien de joueurs il reste, s’il en reste plus d’un, le jeu continue, sinon le dernier est vainqueur.

Fait appel à endGame() qui bloque le canvas (plus de déplacement possible), et fait apparaitre le bouton « Start a New Game ».

Ce bouton fait appel à startGame qui lance une nouvelle partie et fait disparaitre le bouton Start a New Game.

Aller plus loin :

En plus des sprites pour les personnages, armes, cases.  
Plus animation avec la librarie jQuery.animateSprite.

Un son au moment du départ du jeu. « Start », juste après que tout ait été généré.

Ajouter des sons pour les armes, à chaque attaque.

Un petit jingle au moment de la victoire d’un joueur.

Un HUD avec :  
Un compteur de points de vie pour chaque joueur  
Le type d’arme possédé et ses dégâts