TP4 - Space Invaders

Dans ce TP vous allez mettre en oeuvre les connaissances acquises lors du TP précédent. Ce TP est susceptible d'être évalué, il sera donc à remettre dans sa totalité.

Le Space Invaders ou Galaga est un jeu où la diplomatie n'est pas une solution. Vous incarnez un vaisseau dont l'objectif est de détruire des aliens. Votre vaisseau évolue en bas de l'écran et peut tirer des missiles. Les aliens quant à eux évoluent en haut de l'écran en cherchant à rejoindre le bas de l'écran pour vous détruire.

Énoncé

On veut réaliser un jeu de type Space Invaders :
Un Alien se déplace en haut de l'écran, de manière horizontale de gauche à droite ;
Un vaisseau se déplace en bas de l'écran, de manière horizontale de gauche à droite ;
Le vaisseau peut tirer des missiles à la verticale ;
Si un missile rencontre un Alien, celui-ci est détruit et un autre Alien surgit à gauche de l'écran.
Le jeu est composé d'images dans une grille de 9 cases de largeur et 5 de hauteur. Chacune de ces images sera contenue dans une case de 100 pixels de haut par 100 pixels de large. Vous devrez disposer ces images dans un système de grid, de flex ou de tableau. L'image blanche est par défaut affichée dans toutes les cases. Le vaisseau, les missiles, les aliens et les explosions viendront remplacer les images blanches en fonction de l'évolution de la partie.

Exemples de Ressources

OU

Étape 1 : Création du plateau de jeu

Créez un fichier HTML qui ne contient pour l'instant que :
Une balise div vide avec un id contenu ;
Un bouton "jouer" qui permet de lancer une partie ;
L'import vers un fichier JavaScript app.js.
Dans le fichier app.js, créez 3 variables :

```
var largeur = 9;
var hauteur = 5;
var plateau = [];
```

Créez une fonction *initPlateau* qui se lance au chargement de la page (événement load).
Cette fonction va récupérer le contenu de la div HTML et créer un tableau HTML (balise tr et td).

La taille du tableau se fera en fonction des 2 variables précédemment définies.
Dans chaque cellule du tableau, vous ajouterez l'image vide.png fournie en entrée du TP.
Vous devriez à la fin vous retrouver avec une page toute blanche et juste un bouton qui permettra de démarrer une partie.

Étape 2 : Ajout de votre vaisseau

Nous allons maintenant ajouter notre vaisseau sur le plateau de jeu puis lui permettre de se déplacer.
Créez un nouveau fichier JavaScript : `vaisseau.js` (et importez-le dans votre HTML !).
Dans ce fichier, créez 2 variables qui détermineront l'emplacement de votre vaisseau :

```
var posShipY = hauteur -1;
var posShipX = (largeur-1)/2;
```

Le vaisseau sera donc positionné en bas de notre plateau de jeu et au milieu. En effet, pour des simplicités de code, les coordonnées du tableau commenceront en haut à gauche.
Créez une fonction *lancerPartie* qui remplacera l'image vide à la position définie par les coordonnées par l'image de votre vaisseau.
Attachez le déclenchement de cette fonction au clic sur le bouton Jouer, directement dans le code HTML ou dans le fichier app.js.
Maintenant, quand vous appuyez sur le bouton « Jouer », votre vaisseau apparaît à l'écran.

Étape 3 : Déplacer le vaisseau

Nous allons maintenant implémenter le déplacement du vaisseau.
Créez une nouvelle fonction *deplacerVaisseau*. Cette fonction prend en paramètre un événement clavier (event). Pour savoir sur quelle touche de l'ordinateur l'utilisateur a appuyé.
Nous allons utiliser la propriété code des events dont voici la documentation détaillée :
https://developer.mozilla.org/fr/docs/Web/API/KeyboardEvent/code
Dans notre cas, nous voudrons par exemple déplacer l'utilisateur d'une case à gauche s'il a appuyé sur la touche gauche de votre clavier :

```
switch (event.code) {
case "ArrowLeft":
effaceShip();
posShipX--;
afficheShip();
break;
}
```

effaceShip et *afficheShip* sont 2 fonctions qui respectivement : remplaceront l'image du vaisseau par l'image vide et remplaceront l'image vide par l'image du vaisseau.
Il faut maintenant attacher notre fonction aux événements du clavier utilisateur dans votre fichier *app.js* :
window.addEventListener("keydown", deplacerVaisseau);
Votre vaisseau doit pouvoir maintenant se déplacer de droite à gauche.

Attention : vous devez gérer le cas où l'utilisateur essaye de déplacer son vaisseau au-delà de la limite du plateau pour que la variable posShipX ne devienne pas plus grande que la largeur ou négative. Avec un if peut-être ?

Étape 4 : Afficher un missile

Créez un nouveau fichier missile.js.
Dans ce fichier, déclarez 2 variables qui représenteront l'emplacement de notre missile :

```
var posMissileX ;
var posMissileY ;
```

Écrivez une fonction *pewPew* qui représentera un tir de votre missile.
Cette fonction doit :
Initialiser les 2 variables *posMissileX* et *posMissileY* pour que le missile démarre sa course juste au-dessus de votre vaisseau.

Puis elle doit modifier le plateau pour afficher le missile à l'emplacement correspondant.
Modifiez maintenant votre code de la fonction *deplacerVaisseau* pour appeler cette fonction *pewPew* lorsque vous appuyez sur la flèche du haut.
Pour l'instant, votre missile ne se déplace pas encore. Nous allons mettre ça en oeuvre dans la prochaine étape !!!

Étape 5 : Déplacer le missile

Créez une nouvelle fonction : *deplacementMissile(y, x)*
Cette fonction va effacer l'image de missile des coordonnées x, y puis va écrire une image de missile aux coordonnées y-1 (souvenez-vous les coordonnées 0 sont en haut à gauche).
Pour que le comportement de déplacement continue tout seul, cette fonction va se rappeler elle-même.
À la fin de votre fonction *deplacementMissile*, ajoutez le code suivant :

```
setTimeout(function(){
deplacementMissile(x,y-1);
},200);
```

La fonction *setTimeout* permet de décaler l'appel pour donner l'impression de déplacement. Si on appelait directement la fonction *deplacementMissile* sans le *setTimeout*, on ne verrait même pas le missile à l'écran. Voici la documentation si vous souhaitez en savoir plus :
https://developer.mozilla.org/fr/docs/Web/API/setTimeout

Une autre solution est l'utilisation de *setInterval* qui déclenche une fonction à intervalle régulier :
https://developer.mozilla.org/en-US/docs/Web/API/setInterval

En dehors de la fonction *setTimeout*, on rappelle notre fonction (*deplacementMissile*) en faisant monter le missile vers le haut en réduisant les coordonnées y de 1.
C'est ce qu'on appelle la récursivité en algorithme, un concept qui peut être un peu compliqué à appréhender au début.
Appelez maintenant cette méthode *deplacementMissile* dans votre fonction *pewPew()*. Vous devriez maintenant voir votre missile se déplacer à l'écran !

Étape 6 : Placer l'alien

Dans un nouveau fichier nommé alien.js, créez une nouvelle fonction : *initAlien()*.
Cette fonction permet de positionner un alien en haut à gauche de l'écran, vous devrez pour ce faire utiliser 2 variables de position de l'alien initialisées à 0 :

```
var posAlienX ;
var posAlienY ;
```

Créez une fonction *deplaceAlien(x, y)* dont l'objectif sera d'effacer l'alien de sa position actuelle et de le faire apparaître à la position [x, y] transmise en paramètre.
Pour que le comportement de déplacement continue tout seul, cette fonction va se rappeler elle-même.

Étape 7 : Déplacer l'alien

Dans alien.js, créez la fonction *parcoursGrille()* qui utilise les fonctions *setTimeout* ou *setInterval* pour déplacer l'alien.
Ce dernier aura un cycle que vous devrez gérer en utilisant les conditions :
L'alien se déplace d'abord de gauche à droite.

Dès que l'alien a atteint le bord du plateau de jeu, il se déplace verticalement sur la ligne inférieure.

L'alien se déplace ensuite de droite à gauche.
Cette fonction va utiliser la fonction *deplaceAlien(x, y)* afin de :
Effacer l'image de l'alien à la position actuelle (*posAlienX*, *posAlienY*) ;

Mettre à jour la position de l'alien en fonction de sa direction (droite ou gauche) ;

Afficher l'alien à sa nouvelle position ;

Rappeler la fonction elle-même après un certain temps (si vous utilisez le *setTimeout*) pour créer un déplacement continu.
Il faut maintenant appeler la fonction *parcoursGrille()* dans *initAlien()* et également appeler *initAlien()* lorsque vous démarrez une partie dans votre fichier app.js.

Étape 8 : Gérer les collisions entre les missiles et l'alien

Nous allons maintenant gérer les collisions entre les missiles et l'alien. Pour cela, nous devons vérifier si les positions des missiles et de l'alien sont les mêmes à chaque étape du déplacement des missiles.
Modifiez la fonction *deplacementMissile(y, x)* dans le fichier missile.js pour vérifier les collisions.
Pensez également à gérer le cas ou l'alien atteint le bas de l'écran. Vous pouvez ajouter un système de vie ou retirer des points au joueur.

Étape 9 : Comptage des points

Nous allons maintenant ajouter un système de comptage des points. Chaque fois qu'un alien est détruit, le score du joueur augmente.
Ajoutez une variable pour stocker le score dans votre fichier app.js.
Créez une fonction pour mettre à jour le score.
Ajoutez un élément HTML pour afficher le score dans votre fichier HTML.
Enfin, appelez la fonction *updateScore()* chaque fois qu'un alien est détruit. Ajoutez cet appel de fonction à l'intérieur de la condition de collision dans la fonction *deplacementMissile(y, x)* du fichier missile.js.

Avec ces étapes, vous devriez maintenant avoir un jeu Space Invaders fonctionnel avec un déplacement d'alien, un comptage de points et une gestion des collisions entre les missiles et l'alien.