faire un jeu en javascript/HTML5

D'abord quelques exemples de ce que l'on peut faire :

`http://www.playescapegoat.com/`
`http://hexgl.bkcore.com/`
`https://developer.mozilla.org/fr/demos/detail/bananabread`

1 La balise canvas ; fonctions de dessin

On utilisera une balise canvas, apparue avec HTML5, qui est une surface de pixels sur laquelle on peut dessiner des formes géométriques, des textes, des images, etc... La partie algorithmique (dessiner, faire des animations dans le canvas...) est gérée par javascript.

1. **Fonctions de dessin sur le canvas :**
   Nous devons, avant de voir comment réaliser un petit jeu 2D, voir quelle sont les fonctions permettant de dessiner, de charger des images, etc... Pour cela je vous laisse suivre ce tutoriel jusqu'à la partie 4 (les images). La partie sur les dégradés nous intéresse moins mais vous pouvez la faire quand-même si vous le souhaitez.

   `https://www.alsacreations.com/tuto/lire/1484-introduction.html`

   En résumé vous devez savoir après cela :
   — Récupérer le contexte graphique du canvas d'id canvas : `ctx=document.getElementById('canvas').getContext("2d`
   — Dessiner des formes simples et des images à l'aide des méthodes de ctx :
   `beginPath(), moveTo(x,y), lineTo(x,y), closePath(), stroke(), fill(),`
   `fillRect(x,y,hauteur,largeur), clearRect(x,y,hauteur,largeur), arc(x,y,rayon,angle1,angle2,boolSe`
   `bezierCurveTo(x1,y1,x2,y2,x3,y3), quadraticCurveTo(x1,y1,x2,y2),`
   `drawImage(img,x,y)`
   et des propriétés de ctx :
   `fillStyle,strokeStyle, lineWidth, lineJoin, lineCap,`
   `miterLimit, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, ...`
   — Faire des dégradés.
2. Exercice : dessiner sur le canvas, en trait rouge épais, MMI grâce aux fonctions lineTo(), moveTo(), stroke(), etc... Rajoutez au dessus de cela une image de votre choix (utilisation de drawImage()).

2 Animation d'une image à l'aide de rotate() et translate()

Ces deux fonctions sont des méthodes de l'objet ctx, le contexte graphique. Elles agissent sur le contexte graphique lui-même, qu'elle permettent de translater ou de tourner d'une certaine quantité. Ainsi si l'on tourne grâce à `rotate()` le contexte graphique du canvas de $-\frac{\pi}{4}$ radians, et que l'on trace ensuite une image en (0,0) elle doit apparaître en haut à gauche (son coin haut gauche tout en haut à gauche du canvas), tournée de $-\frac{\pi}{4}$ radians.

Notez que vous disposez également de la méthode `scale()` pour redimensionner le contexte graphique du canvas.

A tout moment vous pouvez sauver le contexte graphique dans un état donné (couleur de la ligne, épaisseur, angle et position...) grâce à la méthode `save()` de ctx et le restaurer par la suite dans le même état grâce à la méthode `restore()`.

1. Créez une page web comprenant un élément canvas. Vous pouvez prendre le fichier `index.htm` sur moodle et le fichier css associé `main.css`.
2. Dans un fichier intitulé `js.js`, on va afficher une image dans le canvas et l'animer. Le principe est de créer une image et quand elle charge (onload), on lui associe une fonction boucle() que l'on relance toutes les `fps` secondes (à l'aide de `setInterval()`). Celle-ci redessine l'image, mais dans une position nouvelle... Cherchez d'abord à afficher une image dans le canvas, sans qu'elle ne bouge (pas besoin de `setInterval()` à cette étape, donc). Une image (mario..) est disponible sur moodle, mais vous pouvez choisir l'image que vous voulez.
3. Maintenant, cherchez à faire en sorte que cette image soit progressivement translatée vers la droite.
4. Faites s'arrêter l'image du personnage sur le bord droit. Puis faites la rebondir sur les bords.
5. Arrêtez le mouvement du personnage, et essayez de l'afficher au milieu du canvas, essayez ensuite de l'afficher tourné de $-\frac{\pi}{4}$ radians en (0,0), puis (0,10), puis (0,50) etc... pour comprendre comment fonctionnent les fonctions translate() et rotate(). Il faut faire cette question en utilisant ces deux fonctions.
6. Enfin, essayez de faire en sorte que l'image tourne sur elle-même tout en avançant. Essayez de la faire grossir quand elle va vers la droite, rapetisser quand elle va vers la gauche (il faut utiliser .scale()).

3 Gestion des événements clavier

Un fichier javascript `class.clavier.js` vous est donné ; cela permet de créer des objets de type Clavier possédant des propriétés haut, bas, droite, gauche valant true si l'on va dans la appuie sur la flèche correspondante ; ces valeurs sont repositionnées à false dés que l'utilisateur relève le doigt. **Utilisez ce fichier pour animer votre personnage au clavier** (en incrémentant sa position en fonction de la touche sur laquelle on a appuyé...). Il suffit d'instancier un objet de la classe Clavier (`var clavier = new Clavier();`). Ensuite on peut utiliser l'objet `clavier` pour savoir si l'on a appuyé sur une ou plusieurs touches directionnelles... (`{if (clavier.droite){ // Ici ce qu'on fait quand on a appuyé sur la`

4 Les sprites

Arrivé à ce point, votre personnage peut bouger, dans chacune des directions, en étant contrôlé au clavier. Toutefois il reste fixe... Une des images qui vous est donnée sur e-campus représente une "sprite sheet", une feuille d'icônes présentant les différentes positions du personnage (mario) qui sera utilisé. Votre objectif ici est d'utiliser celle-ci pour animer votre personnage. Vous observez que cette image comprend plusieurs lignes, une ligne correspondant à une des directions selon laquelle le personnage peut se déplacer, et plusieurs colonnes, correspondant aux différentes positions successives qu'aura notre personnage au cours de son déplacement. Vous devez :

1. Apprendre à utiliser `drawImage()` plus finement. Jusqu'ici vous utilisiez 3 ou 5 paramètres d'entrée, maintenant, il faudra en utiliser 7 ou 9. On ne voudra en effet pas afficher toute l'image, mais seulement une partie de l'image, correspondant à la position du personnage. Les 4 paramètres supplémentaires permettent de définir quelle partie (un rectangle défini par son point haut gauche et ses dimensions...) de l'image on souhaite afficher. Apprenez à utiliser `drawImage()`. Pour cela chercher, sans l'animer pour l'instant, à afficher mario, dans l'une ou l'autre des positions définies sur la sprite sheet.
2. Ceci fait, faites en sorte que mario, quand il va à droite et en haut, se déplace en utilisant les positions successives définies dans la sprite sheet. Pour cela, il faudra utiliser deux variables : l'une, positionnée en fonction de la touche sur laquelle le joueur a appuyé, permettra de choisir à quelle ligne on va chercher la position adéquate. La deuxième permet de choisir la colonne, à savoir la position du personnage. Cette variable s'incrémente à chaque déplacement et revient à 0 quand elle dépasse le nombre maximum de positions disponibles (8 ici).
3. Que faire pour la gauche ? Il n'y a pas de position définie pour aller à gauche... Il suffit en fait de prendre les images de mario allant vers la droite et de retourner le repère (`ctx.scale(-1,1)`). On peut s'aider d'un booléen qui dit dans quel sens le repère doit être dessiné...

5 gestion des sauts

Pour gérer les sauts, on peut utiliser un booléen (appelons-le saut), qui vaut `false` en temps normal et `true` dès qu'on appuie sur la flèche haute. On utilisera également une variable compteur qu'on mettra, par exemple, à 40 au début du saut et qu'on fera décroître d'un en un à chaque nouveau passage dans la fonction boucle(). Lorsque ce compteur revient à 0, on remet saut à false. On utilise enfin une variable posy qui gère la position verticale du personnage et dont la valeur dépend du compteur de saut. Il faut qu'au sommet du saut (donc quand le compteur vaut 20), posy soit minimal (posy sera négatif, car si l'on souhaite sauter vers le haut, il faut penser que les ordonnées sont d'autant plus petites qu'on est haut dans le canvas...). On peut par exemple utiliser la fonction $x \mapsto -x^2$ ou encore $x \mapsto -x^4$. *posy* vaudra alors par exemple `100-(20-compt_saut)*(20-compt_saut)/4`. Si l'on ne veut pas que le personnage saute verticalement, il faut également penser à incrémenter posx (si l'on va vers la droite) ou le décrémenter (si l'on va vers la gauche), quand le compteur de saut n'est pas nul. Il faut également éviter de remettre le compteur de saut à 40 à chaque appui sur la flèche haute ; le saut ne se déclenchera que si l'on appuie vers le haut ET que saut n'est pas déjà à true (ou que le compteur de saut vaut 0). Mettez en place ce système ; testez également avec d'autres fonctions, comme $x \mapsto -x^4$. Essayez de faire sauter moins haut votre personnage, ou plus haut.

6 Défilement du paysage

On dispose d'au moins deux drawImage(), l'un pour le fond, l'autre pour le personnage. On peut vouloir :

— Déplacer le personnage dans un décor fixe. On a vu comment faire : on trace le décor en (0,0), on translate le repère lié au canvas à l'endroit où l'on souhaite dessiner le personnage (et cette translation dépend d'une ou deux variables posx et posy dont la valeur évolue à chaque nouveau passage dans la fonction boucle()), on dessine le personnage, on remet le repère à sa place initiale.

— Que le décor défile continûment : il faut alors translater le repère (de plus en plus vers la gauche si l'on veut avoir l'impression que le personnage défile vers la droite) avant de tracer le décor. On peut ensuite remettre en place le repère et tracer le personnage.

— Que le décor défile quand le personnage arrive, par exemple, tout à fait à droite : le décor défile vers la gauche, le personnage se retrouve à gauche du canvas, plus loin dans l'image de fond (cf les premiers Zelda par ex). Pour cela, il nous faut une variable `defil` qui indique à quel niveau de défilement on en est (si posX est supérieur à (la largeur du canvas -la largeur du personnage)*(defil+1) on augmente defil... Inversement si posx $<$ defil (largeur du canvas - largeur du personnage) on décrémente defil... On peut également utiliser un compteur,

qu'on met à 40 à chaque fois qu'on augmente defil, et qui sert à faire défiler progressivement le fond vers la gauche. Essayez de mettre en place ce système.

7 gestion des collisions

Arrivé ici, tout doit marcher, sauf que s'il y a une image de fond, le personnage peut traverser les obstacles, dépasser du canvas... Vous pouvez déjà essayer de l'empêcher de sortir du canvas (à l'aide de conditionnelles, on bloque le mouvement du personnage au delà d'une certaine position). Mais on ne peut pas multiplier les if()... Il nous faut donc des méthodes plus sophistiquées.

Dans le cas d'une collision avec le pointeur de souris (ou veut détecter quand le pointeur de souris entre ou non dans une zone donnée) : on peut utiliser le canal alpha (la transparence), en faisant en sorte que les obstacles aient une transparence différente de ce qui n'est pas obstacle. En jquery, il nous faut utiliser la fonction getImageData() pour récupérer les informations des pixels (R,V,B,A) dessinés sur un canvas. Exemple :

```
$('#canvas').mousemove( function(evt) {
   souris.x = evt.pageX;  // - éventuellement une certaine quantité dépendant de la pos du canvas dans le
   souris.y = evt.pageY;  // on peut utiliser la fonction jquery offset() ...

   image_data = ctx.getImageData(souris.x, souris.y, 1, 1); // Récupération d'un seul pixel
   pixel  = image_data.data;
   if (pixel[3]<98) {    // si la transparence de ce pixel est < 98...

}
});
```

Mais pour la collision entre un personnage et un obstacle dans le cas de l'utilisation d'une grande image pour le fond, on utilise d'autres méthodes. Il est possible, à l'aide de toute une série de conditionnelles (if..) d'empêcher le personnage d'aller dans certaines zones du canvas (en bloquant le déplacement si l'on voit que celui-ci le mêne dans un obstacle). Une gestion plus propre sera vue en TD, on utilisera une classe Obstacle contenant des méthodes permettant de détecter des collisions point/rectangle et rectangle/rectangle (ce qui nous intéresse a priori, le personnage étant une image rectangulaire, et les obstacles étant souvent des rectangles).

1. Si ce n'est pas fait, essayer d'empêcher le personnage de dépasser du canvas (à l'aide de simples if)
2. On va maintenant utiliser la classe obstacle, que je vous donne sur e-campus. Elle est écrite en orienté objet, il faudra donc créer des instances de la classe Obstacle (ex : `var o= new Obstacle(100,100,100,50);`) et ensuite ces instances pourront utiliser les méthodes de la classe obstacle (ex : `o.collision(100,20,20,10)` pour tester une collision entre o, un obstacle rectangulaire, et un autre rectangle, le personnage, défini par les paramètres d'entrée). En aucun cas on ne peut appeler directement collision()... Je vous fournis également une image de fond toute simple, rectangle.png, fond blanc avec deux rectangles noirs. Faites en sorte que Mario se déplace avec cette image comme fond.
3. Faites s'afficher, à tout moment, la position de mario dans la console (par rapport au coin haut gauche de l'image de fond). Rappel : `console.log()` permet d'écrire dans la console...
4. **collision point-rectangle** : on utilise ici la classe Obstacle. On utilise les méthodes `getDistanceX()` et `getDistanceY()` de cette classe. Ces deux méthodes donnent respectivement les distances horizontale (suivant les abscisses) et verticale (suivant les ordonnées) entre le point passé en paramètre et l'objet appelant (l'obstacle). Créez deux instances de la classe obstacle correspondant aux deux rectangles noirs. A l'aide des deux méthodes décrites ici, empêchez le mouvement du personnage s'il s'approche d'un obstacle (distance nulle dans une des directions et trop petite dans l'autre...).
5. **collision rectangle-rectangle** : on utilise la méthode `collision()`. Elle renvoie un booléen qui vaut true si les deux rectangles - l'objet appelant et celui défini par les paramètres d'entrée - se chevauchent, false sinon. C'est ce qui nous servira le plus souvent, le personnage et les obstacles étant souvent tous deux des rectangles... Utilisez cette méthode pour bloquer le déplacement de votre personnage près des deux rectangles.
6. Faites la même chose, mais les obstacles sont cette fois tous ajoutés dans un tableau `obstacles` qui contiendra tous les obstacles. Les tests de collision se font donc grâce à une boucle for qui parcourt ce tableau et vérifie que le personnage ne rencontrera aucun obstacle. En pratique, les obstacles sont souvent définis dans un fichier (texte, xml, ...).

Contraintes du jeu (2024/2025)

Vous devez créer un jeu en vous appuyant sur ce qui a été vu. Une correction vous est remise. Votre jeu devra être rendu **lundi 19 décembre 2024 au plus tard** et respecter les contraintes suivantes :

1. Le jeu doit être absolument basé sur ce qui a été vu en cours, je dois reconnaître la structure de ce qui vous a été donné (setInterval, fonction boucle, ...).

2. Le jeu ne doit pas se passer dans le monde de Mario : il faudra changer de fond et de personnage ! Il faut utiliser les fonds faits avec M. Clech. Il doit donc y avoir plusieurs couches de fond qui glissent à des vitesses différentes. Il faut utiliser une spritesheet. Vous pouvez récupérer des images sur internet, vous n'êtes pas obligés de les créer. Attention, certaines spritesheets disponibles sur internet sont mal conçues et poseront problèmes (les vignettes doivent être dans une grille avec des cases qui ont toutes la même largeur et même hauteur).
3. Votre jeu doit être hébergé sur o2switch (ou ailleurs d'ailleurs si vous avez un autre hébergeur). Il faut rendre dans l'espace que je vais créer sur moodle un rapport pdf d'une page ou deux expliquant de façon très générale ce que vous avez réussi à faire et pas réussi à faire et dans lequel vous faites le bilan de votre travail. Il faut donner le lien où tester le jeu.
4. A préciser...

Barème : non respect de la contrainte 1. : 0. Originalité sur 6, jouabilité/fonctionnement sur 6 (est-ce que le jeu est fluide, ne bloque pas à certains endroits, est-ce qu'il est intéressant ...), avancement technique sur 8 (plus votre jeu met en oeuvre des concepts avancés plus la note est proche de 8).