Graphisme et Sprites. Callback. setTimeout, clearTimeout, setInterval. Modèle MVC etc..

PREMIERE PARTIE : animation simple. L'objet ici est de manipuler des graphismes pour réaliser une petite animation et comprendre comment réaliser une animation graphique simple en Javascript.

1. Récupérez le document javascript_débutant.pdf sur la page Moodle. Lisez-le (facultatif si vous avez suivi les cours UBS – normalement).

2. Commencez avec un nouveau fichier et créez un canvas comme vu dans le TD1. Vous pouvez avoir accès à sa taille en utilisant le code suivant :

```
cahier = document.getElementById("dessin").getContext("2d");
cahier.largeur = document.getElementById("dessin").width;
cahier.hauteur = document.getElementById("dessin").height;
```

   Faites-en sorte que le canvas ait un fond jaune (utilisez fillRect). Tout le TD aura lieu dans des canvas exclusivement.

3. Il est possible de gérer les clicks souris en interceptant les évènements souris de la manière suivante :

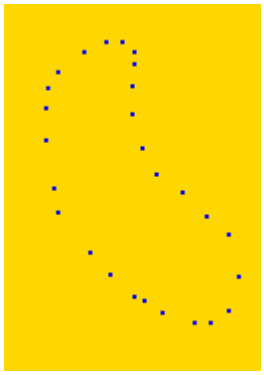
```
function captureClick(event)
{
  console.log("x="+event.pageX+" y="+event.pageY)
}

document.addEventListener("click", captureClick);
```

4. Vérifiez que les coordonnées s'affichent bien, à chaque click souris, dans la console. La fonction *captureClick* est appelée à chaque click souris. Le système passe en paramètre *event* qui est un objet avec des propriétés. *pageX* et *pageY* donnent la coordonnée par rapport à la page web (et pas par rapport au canvas). Pour calculer les coordonnées par rapport aux coordonnées du canvas, utiliser le code suivant :

```
function captureClick(event)
{
  var x = event.pageX - event.target.offsetLeft
  var y = event.pageY - event.target.offsetTop
  console.log("x="+x+" y="+y)
}
```

   Vérifiez que les coordonnées sont cohérentes par rapport à votre canvas (minimum et maximum).

5. Réaliser le code qui permet de dessiner des points en cliquant avec la souris (utiliser fillRect) :



6. On décide de créer une région de type ‘bouton’ dans le canvas. Créez un rectangle rouge de taille 120 par 40 et affichez le texte "effacer" :

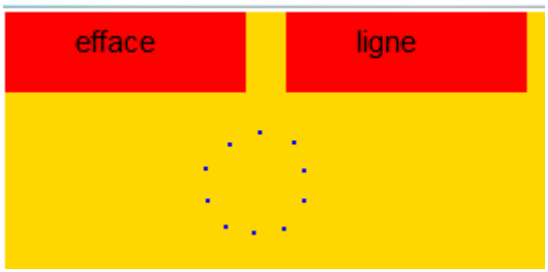
Faites en sorte qu’un click sur ce rectangle efface le canvas complètement (le remet en jaune) et efface donc les points dessinées. Attention, le bouton ne disparait pas. Conseil : fabriquez une fonction *dessineEcran* qui affiche le fond jaune et le bouton. Détectez la position du click dans la fonction *captureClick* et si les coordonnées se trouvent dans le bouton, redessinez l’écran.

7. Créez un tableau pour conserver les points. Chaque click de la souris rajoute un nouveau point dans le tableau. Si vous le souhaitez, vous pouvez ou bien créer un seul tableau qui contient les coordonnées x et y des points, ou bien créer un *tableauX* qui contient les coordonnées x et un *tableauY* qui contient les coordonnées y. Utilisez une variable *nbpoints* qui indique le nombre de points contenus dans le tableau. *nbpoints* représente à la fois le nombre d’éléments du tableau et l’indice du prochain élément à rajouter. "efface" ne se contente pas d’effacer l’écran, il efface aussi le tableau des points.

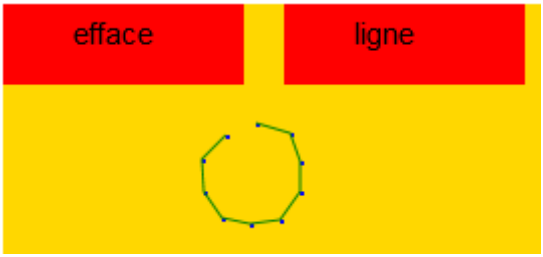
8. Vérifiez que votre tableau se remplit correctement en affichant le tableau dans la console.

9. Réalisez un second bouton, "ligne" qui dessine une ligne entre les points déjà dessinées.

Par exemple : vous tracez un cercle point par point :



Puis quand vous cliquez sur le rectangle "ligne", le cercle devient :

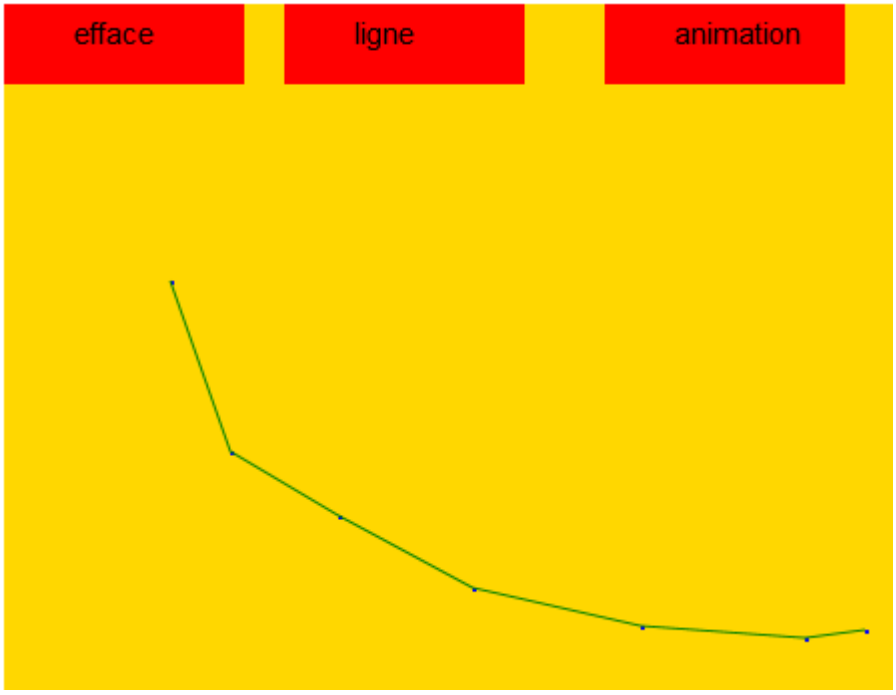


10. Récupérez l'image du chat sur Moodle :

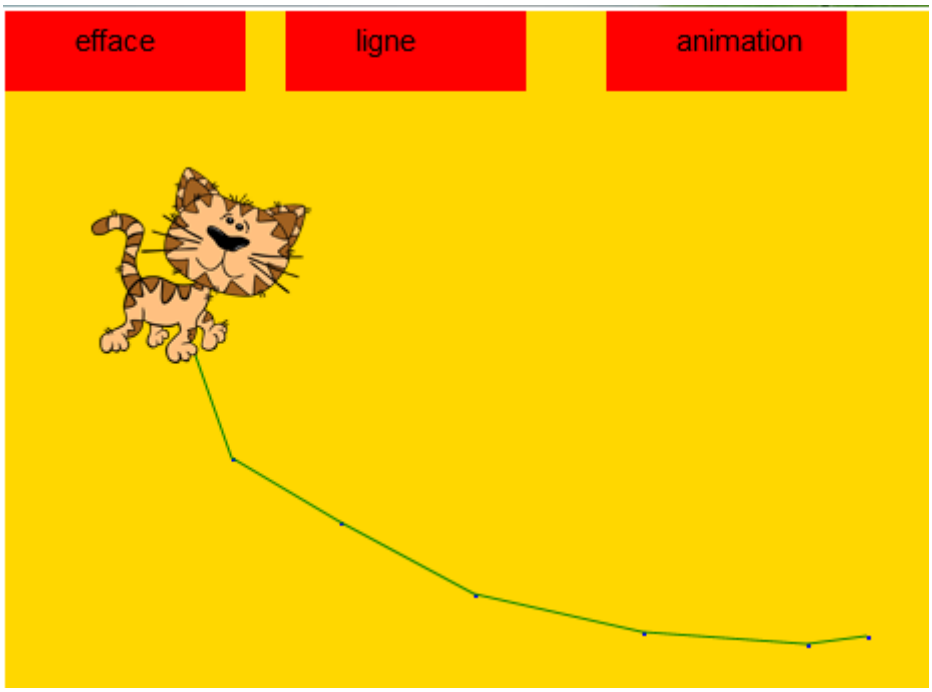
11. On souhaite réaliser un bouton supplémentaire "animation" qui déclenche l'animation suivante : le chat se déplace et suit la trajectoire que vous avez dessiné. Commencez par créer un bouton en plus :

12. Fabriquez une fonction *dessineChat(x,y)* qui affiche le chat à la position (x,y) sur le canvas.

13. Créez une fonction *dessineChatAuPoint(n)* qui dessine le chat à la position du point d'indice n de votre tableau (par exemple *dessineChatAuPoint(0)* dessine le chat sur le premier point conservé dans votre tableau). Gérez le décalage : le chat s'affiche à partir de sa position d'image en haut à gauche. On souhaite que le centre du chat soit affiché sur le point. Vérifiez que cela fonctionne en affichant le chat sur la position 0 quand vous cliquez sur "animation" (pour l'instant, pas d'animation).



Après avoir cliqué sur "animation" :



14. Pour réaliser cette animation, l'idée est de réafficher le chat à intervalles réguliers, en faisant avancer l'indice de sa position. On souhaite appeler *dessineChatAuPoint(0)*, puis, un peu plus tard *dessineChatAuPoint(1),* etc.. Jusqu'au dernier point. Évidemment, entre chaque affichage, il ne faut pas oublier d'effacer l'écran.

    Lancez cette fonction en cliquant sur le bouton animation. Vous aurez sans doute besoin d'une variable supplémentaire (globale) qui indique à quel indice se trouve le chat. Faites attention : ne demandez pas au chat d'aller plus loin que le dernier point de la ligne !

Dans votre animation, le chat doit sauter d'un point à l'autre puis stopper sur le dernier point.

15. Récupérez l'image du chat endormi ainsi que le son du chat (renommez le avec l'extension .mp3)

16. Modifiez votre code pour que le chat s'endorme à l'arrivée.
17. Faites-en sorte que le chat miaule au départ.
18. Que se passe-t-il si vous cliquez pendant le déplacement du chat ? Comment proposez-vous d'interdire les clicks pendant l'animation (et évidemment les autoriser après l'animation) ?

SECONDE PARTIE

Dans le TD qui suit, vous allez réaliser un programme du genre *candy crush* simplifié :

Le principe est le suivant : vous partez d'une configuration sans alignement de 3 bonbons de la même couleur (vertical ou horizontal). Dans l'image précédente, il n'y a pas de configuration de trois bonbons de la même couleur, contiguës vertical ou horizontal.

L'utilisateur peut clicker sur deux bonbons côte à côte. Et seulement sur deux bonbons voisins. Ces deux bonbons échangent leur place (avec une animation) si et seulement si, cet échange résulte en

une nouvelle configuration avec trois (ou plus) bonbons de la même couleur contiguës et aligné (vertical ou horizontal). Sinon, les bonbons n'échangent pas leur place.

Les configurations de bonbons côte-à-côte (verticales et horizontales) de 3 ou plus disparaissent alors, et laissent un trou dans l'espace de jeu.

Les bonbons des colonnes au-dessus des trous tombent alors (avec une animation graphique) pour combler ces espaces vides. La grille est complétée par le haut avec des animations jusqu'à ce que la grille soit pleine.

Il est possible que le déplacement des bonbons réalise d'autres alignements de bonbons, qui, à leurs tours disparaissent etc..

L'utilisateur ne reprends la main que quand la grille s'est stabilisée et qu'il n'y a plus de mouvements.

Un score qui compte des points (le nombre de bonbons disparus) est mis à jour.

Il est absolument primordial de bien réfléchir à l'organisation du programme avant de le réaliser.

1. Les animations graphiques ne sont possibles que quand le navigateur a la main (vous utilisez *setInterval* ou *setTimeout*). Commencez par énumérer les cas d'animation : ce seront les cas où votre programme Javascript stoppe et ne fonctionne que par appel asynchrone de code par *timer*.

2. Une méthode de programmation courante (et particulièrement en Web), consiste à utiliser un modèle MVC ou modèle Modèle Vue Contrôleur. Le modèle contient la représentation du problème ou les données. La vue sert à afficher le modèle sur l'écran. Le contrôleur sert à faire le lien entre l'interface utilisateur, la modification du modèle et demande à la vue de se mettre à jour. Ces trois objets ont leurs méthodes propres et leur état propre.

   Par exemple, le modèle d'un système d'information d'entreprise de vente de pizza peut-être un objet qui contient un tableau et des méthodes (le stock des pizza). La vue est un objet qui fabrique une représentation graphique de la base de donnée du modèle (des graphiques, des photos des pizzas, l'interface graphique etc) et le contrôleur gère les clicks de l'utilisateur, ou bien les évènements de mise à jour en temps réel etc.. Les trois sont bien séparés, ce qui permet de faire évoluer la base de données (ou de la changer) SANS modifier la vue ou le contrôleur. Même chose pour la vue et le contrôleur. Séparer les trois permet également de bien spécifier QUI fait QUOI et simplifie l'implémentation (parce qu'on sait où on va ☺)

   Réalisez des dessins qui expliquent ce que vous mettez dans chaque objet. Ce que chaque objet peut faire ou pas. À quelles données il a accès.

   Vous allez créer ces 3 objets qui sont distincts et permettent de mettre en œuvre le mécanisme du jeu. Vous allez devoir réfléchir à ce que vous souhaitez mettre dans le modèle, ce qu'est la vue et ce que fait le contrôleur. Faites un dessin qui met en scène ces trois objets. Indiquez comment ils communiquent (évidemment par appel de méthodes, mais

quelles méthodes). Avec des flèches, proposez un scénario qui permet de réaliser le jeu du candy crush.

Pensez bien au problème des animations graphiques : normalement, c'est au contrôleur de gérer les demandes de réaffichage, donc c'est le contrôleur qui s'occupe de la gestion des appels asynchrones à la mise à jour graphique.

Faites une liste des évènements (click, début du jeu etc..) et montrez comment votre organisation gère chaque évènement.

Vous rendrez donc un dossier avec cette étude préparatoire : cette étude ne contient pas de code. Une conception MVC se fait AVANT de faire du code. Le code implémente ensuite strictement cette spécification. Cette méthode assure une bonne séparation de l'interface, des évènements et du système d'information.

Cette étude contient vos dessins, votre organisation et structure de données et les cas d'appels ou de communication des objets modèle, vue et contrôleur. Indiquez bien quelles animations gère le contrôleur – et quand.

Imaginez bien ce que vous devrez implémenter pendant que vous réalisez l'analyse.