Story-board

Download de https://github.com/flashline/Atelier-JS-2017 (branch master et done)

Aller sur http://www.pixaline.net/intra/atelier2017/ pour voir les 2 progs dans les dossiers :
shoot/ et easyShoot/.

easyShoot :

Jeter un œil sur index.html et css/index.css .

Théo :
Un des grands principes :
Décomposer un projet en modules plus simples et les modules en fonctionnalités encore plus simples.

Q: Comment peut-on décomposer ce prog en sous-programmes plus simples ?
R: t**arget , canon, ball,** capture des actions utilisateur (**listeners)**

On y va avec **target**...
Choisir le nom de la var qui contiendra l'objet JS permettant de manipuler la html de la cible (target).
1/ **Choisir des noms clairs**, pas y=ax+b mais chargesTotales=CA*coefCharge+chargeFixe
2/ Comme l'idée de partage est très répandue chez les programmeurs, et que plusieurs programmeurs de langues différentes, peuvent travailler sur même projet.
On utilise l'anglais, devenue la langue "naturelle" de l'informatique.

Q: Choix du nom ?
R: **targetElem**

Écrire :
```
/// Target
var targetElem=get("#target"); // see function get() below in js/shoot.js
```
Choisir la fonction pour affichage aléatoire.
TargetRefresh ;
Q: Comment on programme des **actions à intervalles réguliers**
R : setInterval() :
Écrire :
```
window.setInterval(targetRefresh,5000) ;
```

…/

Créer targetRefresh()
Q: Quelle function JS permet de récupérer des nombres aléatoires ?
R: Math.random() ;
Théo : le fonctionnement de Math.random => renvoi un nombre de 0 à 0,999999999....

```
function targetRefresh () {
    var x=Math.floor(Math.random() * (600-20) ); // -20 because right margin is necessary
    var y=Math.floor(Math.random() * 200 ); // 200 because target must be in top part of the scene
    targetElem.style.left=x+"px";
    targetElem.style.top=y+"px";
};
```

Théo :
Les propriétés (variables) de l'objet <element>.style comme <element>.style.**left**, sont les équivalences systématiques des propriétés CSS.

Fin du module target on teste et débogue

Module **canon**
créer var (canonElem) et initialiser canonX et canonY

```
/// Canon
var canonElem=get("#canon");
var canonX=285;
var canonY=315;
```

on verra plus tard comment créer la fonction qui déplace le canon mais on peut déjà la nommer Q : comment ? R : canonMove()

Note : On a besoin du module "listener" **(**capture des actions utilisateur) pour tester canon

Module **listener**
Q: Comment écoute-t-on les événements ?
R: addEventListener()
Détail : document.addEventListener("keydown", <function>);

Écrire :

```
/// Listeners
document.addEventListener("keydown",onKeyDown);
```

Création de la function listener

```
function onKeyDown (e) {
        if (e.keyCode== ??? ) { // left arrow
        }
        else if (e.keyCode== ???) { // right arrow
        }
}
```

Théo : L'objet d'événement *e*
est passé par JS et possède entre autres, la propriété *keyCode*.
Q: Cherchez et trouvez le code correspondant aux flèches et espace.
R: http://keycode.info/ par exemple.

```
function onKeyDown (e) {
        if (e.keyCode== 37) { // left arrow
        }
        else if (e.keyCode== 39) { // right arrow
        }
}
```

Q: Comment n'avoir qu'une seule fonction canonMove pour déplacer dans les 2 sens ?
R: envoyer un paramètre positif OU négatif.

Écrire :

```
var CANON_STEP=6 ; // en début de prog
```

et

```
function onKeyDown (e) {
        //left
        if (e.keyCode==37 ) {
                canonMove(-CANON_STEP);
        }
        //right
        else if (e.keyCode==39 ) {
                canonMove(+CANON_STEP);
        }
        else if (e.keyCode==32  ) {
        }
}
```

Q:
Pourquoi c'est bien d'utiliser une constante comme CANON_STEP plutôt qu'écrire la valeur directement dans la fonction ?
R:
c'est plus facile de modifier des lignes en début de prog plutôt que chercher partout dans un programme.
Théo :
Une constante en programmation est une variable qui ne varie pas durant l'exécution du prog. mais sa valeur peut être changée (réglée) par le programmeur.

Module c**anon** : canonMove() et canonRefresh() ;

Écrire :

```
function canonMove (step) {
        canonX+=step;
        canonRefresh ();
}
function canonRefresh () {
        canonElem.style.left=canonX+"px";
}
```

TEST :
Q: Voir ce qui cloche en testant ?
R: ça déborde !

Module **listeners > onKeyDown**
Écrire :

```
function onKeyDown (e) {
        if (e.keyCode==37 && canonX>0) {
                canonMove(-CANON_STEP);
        }
        else if (e.keyCode==39 && canonX<600-30) { // 30 is the canon's width
                canonMove(+CANON_STEP);
        }
}
```

TEST : Fin du module canon on teste et débogue

Module **ball**
créer var ballElemProto

```
/// Ball
var ballProtoElem=get("#ballProto");
var nextBallNum=0;
```

Q: pourquoi proto ? Quelle particularité a ball par rapport à target et canon ?
R: plusieurs ball en rafale.
Théo : on va créer plusieurs éléments à partir de la prototype.

on verra plus tard comment créer la fonction qui lance une nouvelle ball mais on peut déjà la nommer : Q : comment R : ballShoot()

Écrire :

```
function onKeyDown (e) {
        if (e.keyCode==37 && canonX>0) {
                canonMove(-CANON_STEP);
        }
        else if (e.keyCode==39 && canonX<600-30) { // -30 is the ball's width
                canonMove(+CANON_STEP);
        }
        else if (e.keyCode== 32 ) { // we use space key to shoot a canonball
                ballShoot() ;
        }
}
```

Module **ball** > ballShoot()

On va devoir gérer plusieurs visuels (éléments), plusieurs positions.

Q: Comment résoudre le problème ?

R: Avec l'objet JS de type *Object* .

Théo :

Le type de variable *Object* c'est pas de la POO mais on met un doigt de pied quand même dans la POO façon JS.

Un Object regroupe plusieurs variables qu'on appelle propriétés, dans une seule variable qui est donc du type Object.

On crée l'objet avec var (comme une autre variable)

ex : var ball={} ;

On accède à une propriété en écrivant <nom de l'objet>**.**<nom de la propriété>

ex : ball.elem

On va donc mettre dans un Object "ball" toutes les propriétés d'une balle.

Il y aura un Object ball pour chaque balle.

On passera l'Object ball successivement à toutes les fonctions.

On démarre la function ballShoot.

Au lieu de setInterval on va utiliser une fonction JS plus moderne.

Écrire :

```
function ballShoot () {
    var ball={};
    ball.elem=ballProtoElem.cloneNode();
    ball.x=canonX-5; ball.y=canonY; // ball must start from the top of canon
    get("#main").appendChild(ball.elem);
    ballMove(ball);
};
```

Note : Voir liens web sur cloneNode() et appendChild() si besoin...

Q: Pourquoi et comment donner un id unique à chaque ball.elem
R: c'est la philosophie de html ; l'id d'un élément doit être unique même si dans notre cas ça ne provoquerait pas d'erreur.

Écrire :
```
function ballShoot () {
    nextBallNum++;
    var ball={};
    ball.elem=ballProtoElem.cloneNode();
    ball.elem.id="ball_"+nextBallNum;
    ball.x=canonX-5; ball.y=canonY;
     get("#main").appendChild(ball.elem);
     ballMove(ball);
}
```

créer : `var CANON_BALL_SPEED=2;` // en début de prog
Et Écrire :
```
function ballMove(ball) {
        ball.y-=CANON_BALL_SPEED ;
        ballRefresh(ball);
        var loop=function () { ballMove(ball); }
        window.requestAnimationFrame(loop);
}
```
Et :
```
function ballRefresh (ball) {
        ball.elem.style.left=ball.x+"px";
        ball.elem.style.top=ball.y+"px";
};
```

Q: Quel problème on va avoir si on laisse ballMove comme ça ?
Voir avec l'inspecteur Chrome la création dynamique dans le html des div ball
et ainsi le lien étroit entre html et js
R: les balles ne sont plus visibles mais les fonctions tournent tjrs et les objets s'accumulent en mémoire... jusqu'à "manger" toutes les ressources et ralentir les autres progs !

Théo :
Dans des programmes plus complexes, penser à systématiquement créer une fonction de "nettoyage" dans chaque module qui en a besoin. On peut la nommer par exemple clean() (ou remove() ou clear())
Cette fonction doit selon les cas :
- Supprimer les boucles setInterval par window.clearInterval(<loopId>);
- Enlever les listeners par <object>.removeEventListener("<type event>",<function>)
- Retirer les éléments html inutiles par <element contener>.removeChild(<element>)
- Détruire toutes les références à un objet inutile par <variable>=null ;
  Ainsi le "garbage collector" de JS pourra libérer de la mémoire.
- Exécuter la fonction *clean()* des objets que le module a lui même créés.
  Ainsi le "cleanage" se fait en cascade.

.

Ici on va juste ajouter:

```
function ballMove(ball) {
        ball.y-=CANON_BALL_SPEED ;
        ballRefresh(ball);
        if (ball.y>-30) {
                var loop=function (timestamp) { ballMove(ball); }
                window.requestAnimationFrame(loop);
        }
        else get("#main").removeChild(ball.elem);
}
```

TEST : Fin du module ball on teste et débogue

Notes sur les commentaires (comments)

La règle : ni trop ni trop peu !
Ajouts de commentaires :
En début de module
En début de fonctions avec description des paramètres d'entrée et de sortie.
Cela suffit si les variables et fonctions sont bien nommées.
Exemple façon *javadoc* :

```
/**
* Move the canon to the left or right
*
* @param step       number of pixels.
*                   < 0    for left movement
*                   > 0    for right movement
*                     0    no movement
*/
function canonMove (step) {
        canonX+=step;
        canonRefresh ();
}
```

FIN