Université Paris 8

Moteurs de Jeu [1]

Nicolas JOUANDEAU
n@up8.edu

septembre 2022

1. A la découverte des fonctions proposées par les moteurs et les jeux possibles.

1 Installation et organisation du code

1.1 Introduction

L'élément HTML5 Canvas est un composant HTML avec un rendu définissable en JavaScript (*i.e.* `js`) ; il s'agit simplement de créer un contexte et d'utiliser l'API HTML5 pour dessiner ; il faut distinguer l'élément Canvas de son contexte ; l'élément est un noeud DOM qui est intégré à la page HTML ; le contexte est un objet avec des propriétés et des méthodes pour dessiner dans l'élément canvas ; le contexte est 2D, 2.5D ou 3D ; un canvas ne peut avoir qu'un seul contexte ; il est possible d'avoir plusieurs contextes, et surtout plusieurs canvas, de les superposer et de ne pas tous les afficher.
`WebGL` est une spécification additionnelle à HTML5 pour le rendu et le chargement d'objets 3D proche de `OpenGL ES 3.0` ; `WebGL` est une spécification d'interface de programmation de 3D dynamique pour les pages et applications HTML5 ; elle a été créée par le Khronos Group ; la version WebGL2 tend vers l'unification du support des navigateurs Web et l'exploitation de fonctionnalités de rendu 3D et de nouvelles fonctions des cartes graphiques telles que l'utilisation de nombreuses sources lumineuses ponctuelles (*i.e.* deferred rendering).
`three.js` est une bibliothèque JavaScript facilitant l'ajout de rendu et le chargement d'objets 3D.

Les atouts du développement `js` sont de permettre :
- le développement de jeux open source
- la mise en oeuvre sur tous les périphériques disposant d'un navigateur Web supportant HTML5

Parmi les moteurs et bibliothèques 2D/3D de jeu, avec une activité notable [2], on trouve :
- Phaser, qui est un moteur de jeu pour navigateurs web ; il est destiné au développement de jeux en javascript sur mobile ; étant Phaser3 intègre la spécification WebGl pour les fonctions 3D dans les applications HTML5, utilise Canvas pour les rendus plus simples, permet de charger images, sons, sprites, tilemaps, données JSON, fichiers XML, de gérer les collisions selon trois systèmes de simluation physique (arcade, physique et mécanique), de définir des groupes d'objets, de créer des animations, d'utiliser un système de particules, de manipuler plusieurs caméras, de gérer des entrées clavier, souris, gamepad et interfaces tactiles, d'intégrer des sons Web Audio et HTML Audio, d'adapter résolution et périphérique d'affichage, de créer et d'utiliser des plugins ; dans les grandes lignes, Phaser est :
  - un moteur (sans éditeur intégré) écrit en `js` et `ts` en 2 millions de lignes.
  - 200k lignes en `js` pour la partie disons standard du moteur.
  - 600k lignes en `js` pour la partie physique du moteur.
  - 1 millions de lignes en `ts` pour les animations spines (squelettes hierarchisés).
- Unreal Engine, qui est un moteur de jeu vidéo propriétaire développé par Epic Games.
- Unity3D, qui est un moteur de jeu vidéo propriétaire développé par Unity Technologies.
- CryENGINE, qui est un moteur de jeu développé par Crytek, spécialisé dans les fps [3].
- Godot, qui est un moteur 2D/3D (avec éditeur intégré) libre et open source, écrit en c et cpp (qqes parties js, java et c#) en 3 millions de lignes.

2. Avec un mode de Licence intéressant tel que GPL ou MIT, ayant permis de développer une grande quantité de jeux ou des jeux importants dans l'histoire du jeu vidéo ou encore des concepts intéressants ou amusants.
3. fps est l'abréviation de first person shooter, *i.*e. jeux de tir à la première personne

Autres moteurs : Aleph One, Allegro, Ardor3D, Axiom Engine, `Babylon.js`, Blender Game Engine, Blend4Web, Build engine, Cafu Engine, ClanLib, Cocos2D, Cocos2D-X, Cocos2D-HTML5, Corona(TM) SDK, Crystal Space, Cube Engine, Cube 2 Engine, DarkPlaces, Delta3d, DXFramework, Ethanon Engine, Exult, GameStart, GDevelop 5, Genesis3D, Genesis Device, Id Tech 1, Id Tech 2, Id Tech 3, Id Tech 4, Ika, IndleLib, Ioquake3, Irrlicht, Java3D, JMonkeyEngine, Jogre, LibGDX, LÖVE, MELHARFI, MelonJS, Moai, Monogame, Nebula Device, NeoAxis, OctaForge, Onscripter, OpenSceneGraph, ORX, Panda3D, PLIB, Polycode, Retribution Engine, Second Life, Sparrow, Sphere, Spring, SpriteKit, Starling, Stratagus, Superpowers, Troll2D, Urho3D, Ursina Engine, V-Play, Verge 3.2, Xenko, Xilon Engine II.

Les programmes de ce support sont testés dans `Chrome` ; la compatibilité avec les autres navigateurs (`Edge`, `Firefox`, `opera`, `Safari`, `Chrome Android`, `Firefox Android`, `Opera Android`, `Safari IOS`, `Samsung Internet`, `WebView Android`) n'a pas été testée.

1.2 Premiers pas avec Canvas

Il s'agit de définir un document HTML, un script `js` et de lancer un serveur Web localement et d'utiliser un navigateur Web pour visualiser le résultat.

On utilise le programme HTML suivant pour dessiner dans un Canvas de 640x480 avec un bord noir.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>premier-pas-avec-canvas</title>
  <style>
    body {margin: 0px; padding: 0px;}
    canvas {border: 1px solid black;}
  </style>
</head>
<body>
  <canvas id="myCanvas" width="640" height="480"></canvas>
  <script type="module" src="./js/myprg.js"></script>
</body>
</html>
```

On utilise le programme `js` suivant pour dessiner dans le Canvas.

```
let cnv = document.getElementById('myCanvas');
let ctx = cnv.getContext('2d');

ctx.font = '64pt Calibri';
ctx.fillStyle = 'blue';
ctx.fillText('Hello', cnv.width/2-100, cnv.height/2-100);

ctx.imageSmoothingEnabled= false;
var img = new Image();
img.src = './assets/mario.png';
img.onload = function() {
  ctx.drawImage(img, cnv.width/2-100, cnv.height/2-100, 200, 200);
};
```

Les fichiers utilisés içi sont :

- Le fichier `index.html` définissant le Canvas et utilisant le script `myprg.js`
- Le fichier `myprg.js` écrivant un texte et affichant une image de Mario ; ce fichier est dans le répertoire `js`
- Le fichier `mario.png` correspondant à une image de Mario ; ce fichier est dans le répertoire `assets`

On lance un serveur Web local dans le répertoire contenant le fichier `index.html` à l'aide de la commande suivante :

```
python3 -m http.server 8000
```

On visualise le résultat en ouvrant la page `http://localhost:8000/` dans un navigateur Web comme le présente Fig. 1.

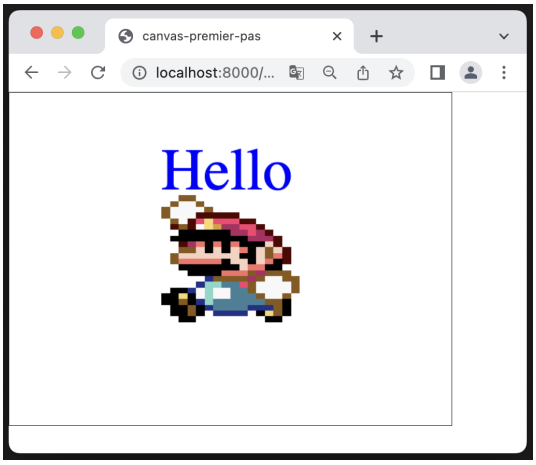

FIG. 1 – Premier programme avec un Canvas.

Au chargement des fichiers, le serveur Web nous indique le code HTTP correspondant à chaque requête :

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [14/Sep/2020 07:12:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Sep/2020 07:12:16] "GET /js/test.js HTTP/1.1" 304 -
127.0.0.1 - - [14/Sep/2020 07:12:16] "GET /img/mario.png HTTP/1.1" 304 -
```

Les principaux codes sont :

- 200 : succès de la requête
- 304 : Document non modifié depuis la dernière requête.
- 404 : Ressource non trouvée.

La balise `title` peut permettre d'être sur que le serveur web charge le bon fichier sans utiliser de cache.

Dans un programme `js`, il sera possible de suivre l'exécution en affichant des valeurs sur la console avec des lignes de ce type :

```
console.log("%d %f %s", 1, 2.0, "trois");
```

1.3 Variables, constantes, classes

`var` est le mot clé historique pour déclarer une variable ; `var` déclare une variable globale si elle est déclarée globalement ; si elle est dans un bloc (ou dans une fonction), cela dépend de son nom ; si elle possède le même nom qu'une variable globale, alors c'est cette variable globale ; si elle ne possède pas le même nom qu'une variable globale, c'est une variable locale au bloc dans lequel elle est déclarée ; avec `var`, il est possible de redéclarer des variables ; tout ceci fait qu'il devient possible de modifier une variable globale, en utilisant le même nom qu'une variable globale existante sans savoir que cette dernière existe ; toute variable de type `var` est considérée comme étant en début de programme ; ainsi il devient possible d'utiliser une variable avant sa déclaration ; par défaut, les variables de type `var` ont la valeur `undefined`.

```
console.log(a);
var a = 1;
```

est équivalent à

```
var a  = undefined;
console.log(a);
var a = 1;
```

`let` est le mot clé introduit pour résoudre les bugs créés par l'utilisation de `var` ; `let` déclare une variable locale, dont la portée est restreinte au bloc dans lequel elle est déclarée ; une variable déclarée avec `let` ne peut pas être redéclarée ; une variable de type `let` ne peut pas être utilisée avant sa déclaration ; utiliser une variable `let` avant son affectation produit une `ReferenceError`.

`const` déclare une constante ; comme son nom l'indique, il n'est pas possible de modifier la valeur d'une constante.

Pour les les fonctions simples correspondant à l'application d'un opérateur ou l'utilisation de `map`, une expression de fonction fléchée permet une syntaxe plus courte.

```
const f1 = (a,b) => a+b;

function f2(a,b) {
  return a+b;
}

let A = [
  [1,2,3,4,5],
  [1,2,3,4,5,6,7,8],
  [1,2,3]
];
let A_size = A.map(A => A.length);
```

Pour arrondir des valeurs, on pourra utiliser `Math.floor` ou `|`.

```
let a = Math.floor(Math.PI);
let b = Math.PI|0;
```

Structurer le code en classes permet de le rendre plus lisible et réutilisable ; sur un code court, l'absence de classe est plus lisible mais dès qu'on dépasse ce qui peut s'afficher à l'écran, structurer en classes est conseillé.

```
let x = 0;
let y = 0;
function move(dx, dy) {
  return [x+dx,y+dy];
}
function update() {
  [x,y] = move(1,1);
  // ...
}
setInterval(update, 200);
```

```
class Pos {
  constructor(nx = 0, ny = 0) {
      this.x = nx;
      this.y = ny;
  }
  move(dx,dy) {
      this.x += dx;
      this.y += dy;
  }
}
let p = new Pos();
function update() {
  p.move(1,1);
  // ...
}
setInterval(update, 200);
```

Avec les classes, il convient de garder à l'esprit que la relation d'héritage est très contraignante en cas de réorganisation du code ; pour définir un héritage, on utilisera les mots clés `extends` et `super`.

```
class Pt2D {
  constructor(x = 0, y = 0) {
      this.x = x;
      this.y = y;
  }
};
class Pt3D extends Pt2D {
  constructor(x = 0, y = 0, z = 0) {
      super(x, y);
      this.z = z;
  }
};
```

On pourra définir des fonctions `static` pour les appeler sans instancier une classe.

```
class A {
  static f(x, y, z) {
    return Math.sqrt(x*x+y*y+z*z);
  }
};
let a = A.f(1.0,2.0,3.0);
```

1.4 Dessin vectoriel avec l'API Canvas

`https://developer.mozilla.org/fr/docs/Web/API/Canvas_API`.

- Pour dessiner un rectangle
  - forme pleine : `fillRect(x, y, largeur, hauteur)`
  - filiforme : `strokeRect(x, y, largeur, hauteur)`
- Pour effacer une zone rectangulaire : `clearRect(x, y, largeur, hauteur)`
- Pour dessiner suivant un trajet
  - Initialiser le trajet : `beginPath()`
  - Finaliser le trajet : `endPath()`
  - Tracer en mode contour : `stroke()`
  - Tracer en mode plein : `fill()`
  - Pendant un trajet en ligne droite :
    - Aller à une position sans tracer : `moveTo(x,y)`
    - Aller à une position en ligne droite : `lineTo(x,y)`
  - Pendant un trajet sur un arc de cercle :
    - Centré sur une coordonnée : `arc(x, y, r, angle1, angle2, ccw)`
    - Reliant deux points selon un rayon : `arcTo(x1, y1, x2, y2, r)`
  - Selon une courbe de Bézier quadratique, partant de la position courante, allant vers une position en utilisant un point de contrôle :
    `quadraticCurveTo(cp1x, cp1y, x, y)`
  - Selon une courbe de Bézier cubique, partant de la position courante, allant vers une position en utilisant deux points de contrôle :
    `bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)`
  - En suivant un tracé rectangulaire : `rect(x, y, largeur, hauteur)`

Consulter la documentation pour plus de fonctions concernant la combinaison des trajets et les trajets SVG.

1.5 Animer Canvas avec setInterval

La fonction `setInterval` permet d'appeler une fonction itérativement selon un délai en millisecondes avec un mimimum à 4ms.

Le programme suivant dessine un cercle dont la position change avec le temps ; la trajectoire suivie par le cercle bleu correspond à des rebonds vers la droite sur un sol fictif en 400 en y ; chaque appel de la fonction `update` demande :

- la MAJ de la position avec `update_pos` (lignes 9 à 16 et ligne 23)
- le dessin du cerle avec `draw` (lignes 17 à 21 et ligne 24)
- la fonction `draw` est appelée toutes les 10ms ou plus, selon la charge de calcul courante.

A chaque instant, le cercle est défini par sa taille `ballSize` et par sa position en ordonnée `posX` et sa position en absisse `posY` ; la variable `vX` définit la vitesse du cercle selon l'axe des ordonnées et la variable `hY` définit la hauteur des rebonds du cercle.

```
1  let cnv = document.getElementById("myCanvas");
2  let ctx = cnv.getContext("2d");
3  let ballSize = 30;
4  let vX = 2.0;
5  let hY = 200.0;
6  let posX = 0;
7  let posY = 400-1.0*Math.abs(hY*Math.sin(0.0));
8
9  function draw() {
10   ctx.clearRect(0, 0, cnv.width, cnv.height);
11   ctx.beginPath();
12   ctx.arc(posX, posY, ballSize, 0, 2*Math.PI);
13   ctx.fillStyle = "#0000FF";
14   ctx.fill();
15   ctx.closePath();
16 }
17 function update_pos() {
18   posX += vX;
19   posY = 400-1.0*Math.abs(hY*Math.sin(Math.PI*posX/60));
20   if(posX >= cnv.width) posX = 0;
21 }
22 function update() {
23   update_pos();
24   draw();
25 }
26 setInterval(update, 10);
```

1.6 Animer Canvas avec requestAnimationFrame

La fonction `requestAnimationFrame` permet d'appeler automatique un callback avant chaque MAJ de la page du navigateur ; pour être appelé avant chaque MAJ, le callback doit rappeler la fonction `requestAnimationFrame` ; le callback prend un argument qui définit le temps en millisecondes écoulé depuis le début de l'animation.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
let ballSize = 30;
let vX = 2.0;
let hY = 200.0;
let posX = 0;
let posY = 400-1.0*Math.abs(hY*Math.sin(0.0));
function draw() {
  ctx.clearRect(0, 0, cnv.width, cnv.height);
  ctx.beginPath();
  ctx.arc(posX, posY, ballSize, 0, 2*Math.PI);
  ctx.fillStyle = "#0000FF";
  ctx.fill();
  ctx.closePath();
}
function update_pos() {
  posX += vX;
  posY = 400-1.0*Math.abs(hY*Math.sin(Math.PI*posX/60));
  if(posX >= cnv.width) posX = 0;
}
let previousTimeStamp = undefined, updateTime = 10, elapsed = 11;
function update(timestamp) {
  if(previousTimeStamp != undefined) {
    elapsed = timestamp-previousTimeStamp;
  }
  if(elapsed > updateTime) {
    previousTimeStamp = timestamp;
    update_pos();
  }
  draw();
  requestAnimationFrame(update);
}
requestAnimationFrame(update);
```

Pour ajouter une trace des positions du cercle comme présenté Fig. 2, on mémorise dans un tableau la position du cercle à chaque appel de `update_pos` lors du premier passage dans la largeur de la fenêtre.

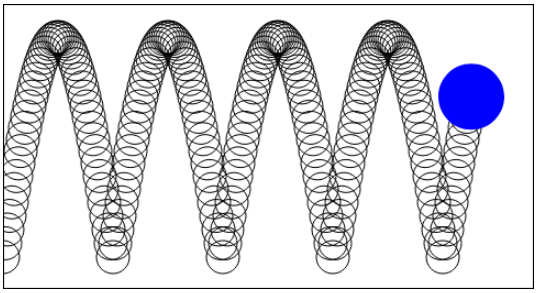

FIG. 2 – Trace des positions du cercle.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
let ballSize = 30;
let vX = 2.0;
let hY = 200.0;
let posX = 0;
let posY = 400-1.0*Math.abs(hY*Math.sin(0.0));
let persistant_on = 1;
let persistant_circles = [[posX,posY]];

function draw() {
  ctx.clearRect(0, 0, cnv.width, cnv.height);
  for (let i = 0; i < persistant_circles.length; i++) {
    ctx.beginPath();
    ctx.arc(persistant_circles[i][0], persistant_circles[i][1],
            ballSize/2, 0, 2*Math.PI);
    ctx.stroke();
    ctx.closePath();
  }
  ctx.beginPath();
  ctx.arc(posX, posY, ballSize, 0, 2*Math.PI);
  ctx.fillStyle = "#0000FF";
  ctx.fill();
  ctx.closePath();
}
function update_pos() {
  posX += vX;
  posY = 400-1.0*Math.abs(hY*Math.sin(Math.PI*posX/100));
  if(posX >= cnv.width) { posX = 0; persistant_on = 0; }
  if(persistant_on == 1) { persistant_circles.push([posX,posY]); }
}
let previousTimeStamp = undefined, updateTime = 10, elapsed = 11;
function update(timestamp) { ... }
requestAnimationFrame(update);
```

1.7 Animer Canvas en suivant la souris

La fonction `addEventListener` permet d'ajouter un callback sur les évènements générés à la souris ; pour chaque évènement `e`, on récupère la position de la souris avec les attributs `clientX` et `clientY` ; les évènements possibles sont `mousedown`, `mousemove`, `mouseup`, `click`, `dblclick`, `mousout`, `mouseover`, `mouseenter`, `mouseleave`, `contextmenu`.

Dans ce cas, l'affichage doit être mis à jour sans délai ; on utiliser la fonction `update` sans prendre en compte le paramètre `timestamp` ; on peut définir `update` sans le paramètre `timestamp`.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
let ballSize = 30, posX = 0, posY = 0;
function draw() {
  ctx.clearRect(0, 0, cnv.width, cnv.height);
  ctx.beginPath();
  ctx.arc(posX, posY, ballSize, 0, 2*Math.PI);
  ctx.fillStyle = "#0000FF";
  ctx.fill();
  ctx.closePath();
}
function update() {
  draw();
  requestAnimationFrame(update);
}
document.addEventListener("mousemove", mousemove_fun);
function mousemove_fun(e) { posX = e.clientX; posY = e.clientY; }
requestAnimationFrame(update);
```

1.8 Amortir le suivi de la souris par interpolaton linéaire

La fonction `lerp` (ligne) permet de réaliser une interpolation linéaire entre deux positions avec un amortissement ; on ajoute à chaque instant au cercle le décallage entre la position de la souris et la position du cercle, multiplié par l'amortissement ; on dessine deux cercles ; l'un en rouge sans la fonction `lerp` et l'autre en bleu avec la fonction `lerp`.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
let ballSize = 30;
let ballX_1 = 0, ballY_1 = 0;
let ballX_2 = 0, ballY_2 = 0;
let posX = 0, posY = 0;
let trace_size = 100;
let trace_mouse = [[posX,posY]];
function drawBall(x,y,size,fillColor) {
  ctx.beginPath();
  ctx.arc(x, y, size, 0, 2*Math.PI);
  ctx.fillStyle = fillColor;
  ctx.fill(); ctx.closePath();
}
function draw() {
  ctx.clearRect(0, 0, cnv.width, cnv.height);
  ctx.beginPath();
  ctx.moveTo(ballX_1,ballY_1);
  ctx.lineTo(posX,posY);
  ctx.strokeStyle = "#FF0000";
  ctx.stroke(); ctx.closePath();
  drawBall(ballX_1, ballY_1, ballSize,"#FF0000");
  drawBall(ballX_2, ballY_2, ballSize/2,"#0000FF");
}
function lerp (start, end, amt){ return (1-amt)*start+amt*end }
function update() {
  ballX_1 += (posX - ballX_1) * 0.1;
  ballY_1 += (posY - ballY_1) * 0.1;
  ballX_2 = lerp(ballX_2, posX, 0.1);
  ballY_2 = lerp(ballY_2, posY, 0.1);
  draw();
  requestAnimationFrame(update);
}
document.addEventListener("mousemove", mousemove_fun);
function mousemove_fun(e) { posX = e.clientX; posY = e.clientY; }
requestAnimationFrame(update);
```

Fig. 3 présente la trace de l'amortissement linéaire des positions du cercle.

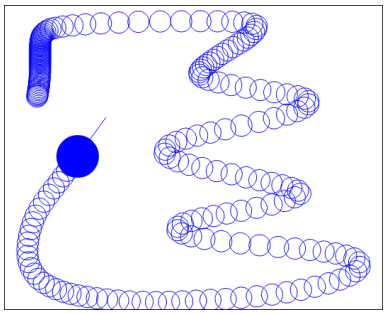

FIG. 3 – Trace des amortissemets des positions du cercle.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
let ballSize = 30, ballX = 0, ballY = 0;
let mouseX = 0, mouseY = 0;
let trace_size = 200;
let trace_mouse = [[ballX,ballY]];
function drawBall(x,y,size,fillColor) { ... }
function draw() {
  ctx.clearRect(0, 0, cnv.width, cnv.height);
  for (let i = 0; i < trace_mouse.length; i++) {
    ctx.beginPath();
    ctx.arc(trace_mouse[i][0], trace_mouse[i][1], ballSize/2, 0, 2*Math.PI);
    ctx.stroke(); ctx.closePath();
  }
  ctx.beginPath();
  ctx.moveTo(ballX,ballY);
  ctx.lineTo(mouseX,mouseY);
  ctx.strokeStyle = "#0000FF";
  ctx.stroke(); ctx.closePath();
  drawBall(ballX, ballY, ballSize,"#0000FF");
}
function update() { ... }
document.addEventListener("mousemove", mousemove_fun);
function mousemove_fun(e) {
  mouseX = e.clientX; mouseY = e.clientY;
  ballX += (mouseX - ballX) * 0.1; ballY += (mouseY - ballY) * 0.1;
  trace_mouse.push([ballX,ballY]);
  if(trace_mouse.length > trace_size) trace_mouse.shift();
}
requestAnimationFrame(update);
```

1.9 Librairie `dat.GUI`

Pendant la phase de développement, il peut être utile de visualiser différentes valeurs de paramètres.

La librairie `dat.GUI` permet de définir des composants graphiques et d'avoir des `controlers` permettant de manipuler une variable pendant l'éxecution du programme ; on place le fichier `dat.gui.js` dans le répertoire `js` et on ajoute la ligne suivante dans le fichier html.

```
<script src="./js/dat.gui.js"></script>
```

Et le fichier `index.html` devient comme suit.

```
<!doctype html>
<html lang="en">
<head>
  <style>
    canvas {border: 1px solid black;}
  </style>
  <script src="./js/dat.gui.js"></script>
</head>
<body>
  <canvas id="myCanvas" width="640" height="480"></canvas>
  <script type="module" src="./js/prg.js"></script>
</body>
</html>
```

La librairie `dat.GUI` est aujourd'hui ici : `github.com/dataarts/dat.gui` ; elle permet d'organiser les `controlers` hiérarchiquement dans des répertoires ; le programme suivant permet de contrôler la vitesse, la couleur, la taille de la balle et la taille de la trace du trajet de la balle du programme présenté à la section 1.6 ; l'ajout d'un controler implique d'utiliser un objet avec des propriétés ; Fig. 4 présente le résultat.

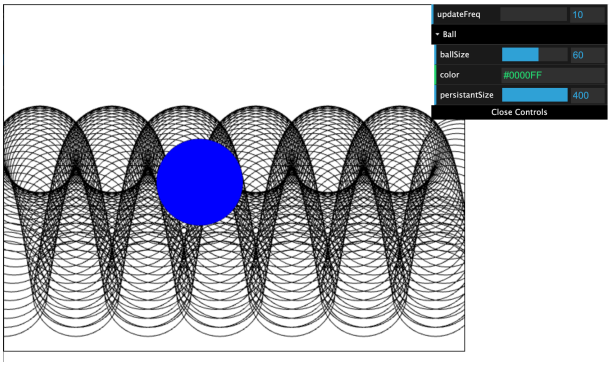

FIG. 4 – Utilisation d'un `controler dat.GUI`.

Pour obtenir une longueur variable, on peut définir un objet `param` avec une propriété `longueur` que l'on souhaite faire varier de 10 à 100 par pas de 1.

```
1  let param = {
2    longueur = 10,
3  };
4  let gui = new dat.gui.GUI();
5  gui.add(param, 'longueur').min(10).max(100).step(1);
```

On pourra définir :

- Une case à cocher à l'aide d'un booléen (`true` si coché et `false` sinon)
- Un menu déroulant à l'aide d'un tableau de valeurs possibles
- Un menu déroulant à l'aide d'un tableau associatif (qui sélectionne la valeur associée ; on peut parler de valeurs nommées)
- Une fonction qui sera appelée à la sélection du composant graphique `dat.GUI`
- Une couleur avec une fenêtre ergonomique de définition de cette couleur
- Un premier répertoire `first values` contenant les deux premiers éléments de cette liste
- Un premier répertoire `last values` contenant les trois éléments suivants de cette liste

On peut renommer le nom d'un élément de l'interface `dat.GUI` avec la fonction `name('new_name')` ; lors de la mise à jour d'une valeur, on peut également demander l'appel d'une fonction prédéfinie avec `onChange()`.

```
1  let param = {
2    valeurBooleennePourCaseACocher : true,
3    menuDeroulant : "un", // ici un est la valeur par défaut
4                          // ce qui suppose de l'avoir par la suite (cf ligne 13)
5    menuDeroulantAssociatif : 1.0,
6    appelDeFonction : fun() { console.log("hello world"); },
7    couleur: "#ffae23",
8  };
9  function another_fun() { console.log("another hello world"); }
10 let gui = new dat.gui.GUI();
11 let fa = gui.addFolder('first values');
12 fa.add(param, 'valeurBooleennePourCaseACocher').onChange(another_fun);
13 fa.add(param, 'menuDeroulant', [ 'un', 'deux', 'trois' ]);
14 let fb = gui.addFolder('last values');
15 fb.add(param, 'menuDeroulantAssociatif', { Un: 1.0, Dix: 10.0, Vingt: 20.0 } );
16 fb.add(param, 'appelDeFonction').name('une fonction');
17 fb.addColor(param, 'couleur');
```

A chaque changement de valeur de la case à cocher ligne 11, la fonction `another_fun` est appelée.

Il est possible d'ajouter automatiquement des fonctions de sauvegarde et de restauration des options choisies avec `gui.remember(param)`.

Dans le cas du programme de la section 1.6, on peut définir une classe `Ball`.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
class Ball {
  constructor(ballSize = 30, vX = 2.0, hY = 200.0, persistantSize = 10) {
    this.ballSize = ballSize;
    this.vX = vX;
    this.hY = hY;
    this.updateFreq = 10;
    this.color = "#0000FF";
    this.posX = 0;
    this.posY = 400-1.0*Math.abs(this.hY*Math.sin(0.0));
    this.persistantSize = persistantSize;
    this.persistant_circles = [];
    if(this.persistantSize > 0) {
      this.persistant_circles.push([this.posX,this.posY]);
    }
  }
  update_pos(cnv) {
    this.posX += this.vX;
    this.posY = 400-1.0*Math.abs(this.hY*Math.sin(Math.PI*this.posX/100));
    if(this.posX >= cnv.width) {
      this.posX = 0;
      this.posY = 400-1.0*Math.abs(this.hY*Math.sin(Math.PI*this.posX/100));
    }
    if(this.persistantSize > 0) {
      this.persistant_circles.push([this.posX,this.posY]);
      while(this.persistant_circles.length > this.persistantSize) {
        this.persistant_circles.shift();
      }
    }
  }
  draw(ctx) {
    for (let i = 0; i < this.persistant_circles.length; i++) {
      ctx.beginPath();
      ctx.arc(this.persistant_circles[i][0], this.persistant_circles[i][1],
        this.ballSize, 0, 2*Math.PI);
      ctx.stroke();
      ctx.closePath();
    }
    ctx.beginPath();
    ctx.arc(this.posX, this.posY, this.ballSize, 0, 2*Math.PI);
    ctx.fillStyle = this.color;
    ctx.fill();
    ctx.closePath();
  }
}
```

```
47 let ball = new Ball();
48 let gui = new dat.gui.GUI();
49 gui.add(ball, 'updateFreq').min(10).max(100).step(1);
50 let ballFolder = gui.addFolder('Ball');
51 ballFolder.add(ball, 'ballSize').min(10).max(100).step(1);
52 ballFolder.add(ball, 'color');
53 ballFolder.add(ball, 'persistantSize').min(10).max(400).step(10);
54
55 function draw() {
56   ctx.clearRect(0, 0, cnv.width, cnv.height);
57   ball.draw(ctx);
58 }
59 let previousTimeStamp = undefined, elapsed = ball.updateFreq+1;
60 function update(timestamp) {
61   if(previousTimeStamp != undefined) {
62     elapsed = timestamp-previousTimeStamp;
63   }
64   if(elapsed > ball.updateFreq) {
65     previousTimeStamp = timestamp;
66     ball.update_pos(cnv);
67   }
68   draw();
69   requestAnimationFrame(update);
70 }
71 requestAnimationFrame(update);
```

L'interfaçage des variables `updateFreq`, `ballSize`, `color` et `persistantSize` avec des composants graphiques permet de faire varier respectivement la vitesse de déplacement de la balle, sa taille, sa couleur et sa trace.