Modélisation et géométrie discrète

Compte-rendu TP7

Animation

Louis Jean
Master 1 IMAGINE
Université de Montpellier

14 novembre 2023

Table des matières

1 Introduction

L'animation dans un environnement web, en particulier en utilisant JavaScript, est un domaine captivant qui combine à la fois la créativité et la rigueur technique. Ce concept joue un rôle essentiel dans des applications variées telles que les interfaces utilisateur interactives, les jeux vidéo et les visualisations de données. Dans ce TP, j'ai plongé dans le monde de l'animation JavaScript en me concentrant sur la dynamique d'une balle animée, à travers une base de code **JavaScript** déjà fournie.

2 Animation simple

Au départ, il était demandé de faire "tomber" une balle sur le sol (une droite). Voici ce que j'ai écrit pour obtenir l'effet escompté.

```
window.onload = function () {
    let d = new Displayer(document.getElementById("
        ↪ display"));
    let b = new Ball(40, 40, 40);
    d.drawBall(b);
    let width_canvas = d.getCanvas().width, height_canvas
        ↪  = d.getCanvas().height;
    d.drawLine(0, height_canvas, width_canvas,
        ↪ height_canvas);
    let animation = new Animation(50,20);
    let speedY = 30;
    animation.nextStep = function(t) {
        b.y += speedY;
        if(b.y > height_canvas - b.w) b.y = height_canvas
            ↪  - b.w;
        d.init();
        //d.g2d.clearRect(0,b.y - b.w - 2*stepY,2*b.w,2*b
            ↪ .w); // On peut faire ça pour éviter de
            ↪ redessiner la ligne à chaque fois, mais un
            ↪ peu imprécis
        d.drawLine(0, height_canvas, width_canvas,
            ↪ height_canvas);
        d.drawBall(b);
    }
    animation.run();
}
```

Figure 1: Code réalisant l'animation simple

Et voici le rendu après animation, avec la balle qui se pose sur le sol.

Figure 2: Fin de l'animation simple

3 Squash and stretch

Pour réaliser l'effet de squash and stretch bien connu en animation, pratique pour rajouter de la "vie" à son animation, j'ai préféré passer par une figure d'ellipse plutôt qu'un cercle. J'ai donc modifié en accord ma classe **Displayer**, ainsi que la fonction de dessin d'une ellipse.

```
class Ellipse {
    x;
    y;
    wx;
    wy;
    constructor(x, y, wx, wy) {
        this.x = x;
        this.y = y;
        this.wx = wx;
        this.wy = wy;
    }
}
```

Figure 3: Création de la classe Ellipse

```
drawEllipse(p) {
    let x = p.x,
        y = p.y,
        wx = p.wx,
        wy = p.wy;
    this.g2d.beginPath();
    this.g2d.ellipse(x, y, wx, wy, 0, 0, Math.PI * 2,
        ↪ true);
    this.g2d.fillStyle = this.obg;
    this.g2d.fill();
    this.g2d.strokeStyle = this.fg;
    this.g2d.stroke();
}
```

Figure 4: Fonction de dessin pour l'ellipse

```
window.onload = function () {
    let d = new Displayer(document.getElementById("
        ↪ display"));
    let x_pos = 0;
    let y_pos = 0;
    let rayonX = 40;
    let rayonY = 40;
    let e = new Ellipse(x_pos, y_pos, rayonX, rayonY);
    let speedX = 5;
    let speedY = 0;
    let gravity = 0.98;
    let amortissementY = -0.9;
    let amortissementX = -1;
    let width_canvas = d.getCanvas().width, height_canvas
        ↪  = d.getCanvas().height;
    d.drawLine(0, height_canvas, width_canvas,
        ↪ height_canvas);
    let animation = new Animation(500,10);
    animation.nextStep = function(t) {
        speedY += gravity;
        let stretchFactor = 1 + Math.abs(speedY) / 100;
            ↪ // Facteur d'étirement de la balle, à
            ↪ ajuster si on veut
        e.x += speedX;
        e.y += speedY;
        if(e.y < height_canvas - e.wy) {
            e.wx = rayonX / stretchFactor;
            e.wy = rayonY * stretchFactor;
            squashSteps = 0;
        }
        else {
            e.y = height_canvas - e.wy;
            speedY *= amortissementY; // Rebond
            e.wx = rayonX;
            e.wy = rayonY;
        }
        if(e.x > width_canvas - e.wx) {
            speedX *= amortissementX; // Retour en arriè
                ↪ re si on dépasse les x ou l'origine
            amortissementX = -1 * amortissementX;
        }
        if(e.x < 0) {
            amortissementX = -1 * amortissementX;
            speedX *= amortissementX
        }
        d.init();
        d.drawLine(0, height_canvas, width_canvas,
            ↪ height_canvas);
        d.drawEllipse(e);
    }
}
```


Figure 5: Code réalisant l'effet de squash and stretch

J'ai pris en compte les rebonds sur le sol et sur les murs. Quand la balle descend ou monte, elle s'étire verticalement, et quand elle touche le sol, elle s'écrase sur elle-même. Voici un aperçu du rendu de l'animation.

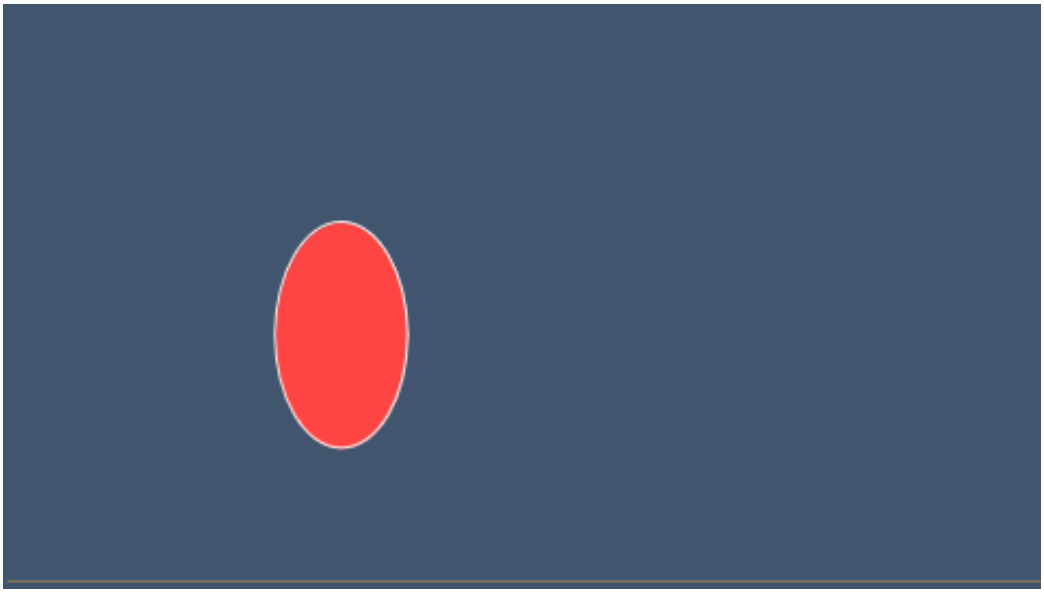

Figure 6: Balle étirée

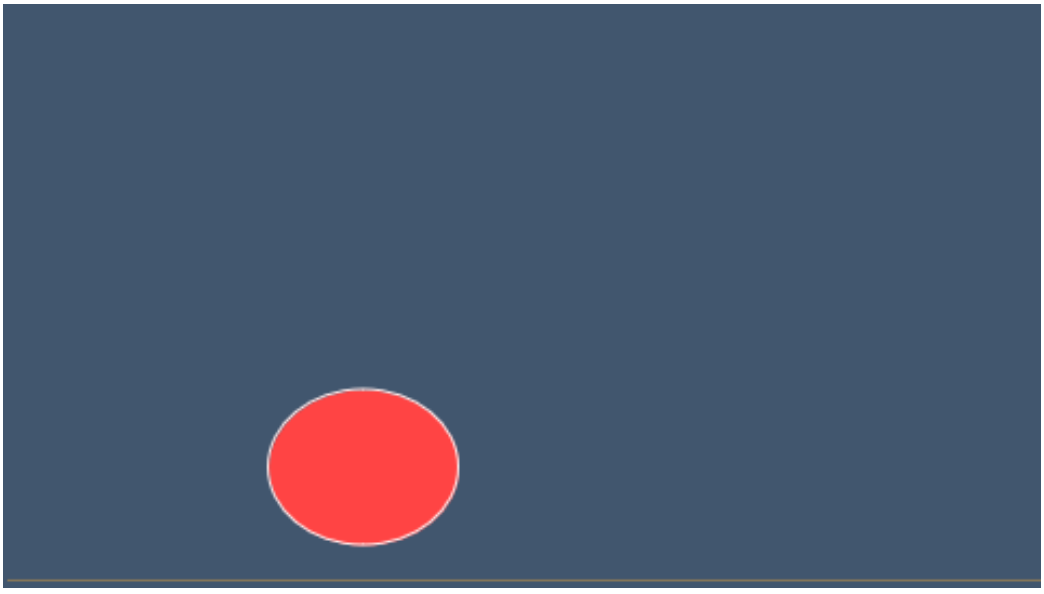

Figure 7: Balle écrasée

4 Arrêt et redémarrage de l'animation

Il fallait pouvoir arrêter et redémarrer l'animation à son bon vouloir. Pour cela j'ai rajouté un bouton HTML, et implémenté une fonction **stop** dans la classe **Animation**.

```
<input type="button" id="start_stop_btn" value="Arrêter/d
    ↪ émarrer l'animation">
```

Figure 8: Création du bouton HTML

```
let StartStopBtn = document.getElementById("
    ↪ start_stop_btn");
let isRunning = true;
StartStopBtn.addEventListener("click", function() {
    if(isRunning) {
        animation.stop();
        isRunning = false;
    }
    else {
        animation.run();
        isRunning = true;
    }
});
```

Figure 9: Bout de code pour ajouter la logique d'appui sur le bouton

```
stop() {
    window.cancelAnimationFrame(this.id);
}
```

Figure 10: Fonction permettant l'arrêt de l'animation, dans la classe **Animation**

Figure 11: Bouton permettant d'arrêter et de démarrer l'animation

5 Affichage du framerate

Pour afficher le framerate, j'ai modifié les attributs de la classe **Animation** et la fonction **animate**, puis j'ai fait une fonction d'affichage du dit framerate sur la canvas.

```
class Animation {
    id = 0;
    step = 0;
    start = 0;
    previousTimeStep = 0;
    framerate = 0;
```

Figure 12: Mise à jour de la classe **Animation**

```
this.framerate = 1/(delta/1000);
```

Figure 13: Calcul du framerate dans la méthode **animate**

```
drawFramerate(framerate) {
    this.g2d.fillStyle = "black";
    this.g2d.font = "20px Arial";
    this.g2d.fillText(`${framerate.toFixed(2)} FPS
        ↪ `,10,30);
}
```

Figure 14: Dessin du framerate

Figure 15: Affichage des FPS

6 Rembobinage de l'animation

Pour rembobiner l'animation, j'ai tenté des choses, notamment un stockage dans un tableau de chaque variable utile, mais n'ai malheureusement pas réussi l'implémentation, car à chaque fois que j'appuyais sur le bouton pour rembobiner, l'animation se stoppait quelques temps puis reprenait au même endroit.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.