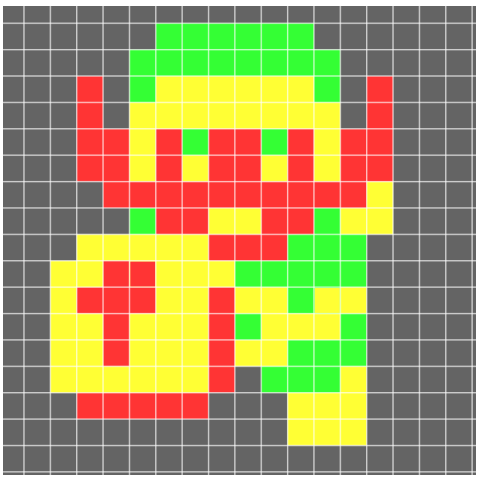
Louis Masson

Pixel Art



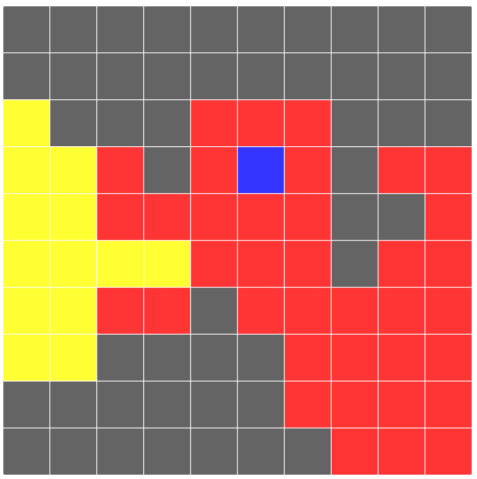


Table des matières

Introduction

Dans les premiers jeux vidéos, il était impossible de fournir des images de haute résolution. Limité par des plateformes relativement peu puissantes et ayant peu de mémoire comparé à aujourd'hui, les graphismes où les pixels étaient encore discernables ont donné naissance à un nouveau style de dessin : le pixel art. Aujourd'hui, le pixel art est encore beaucoup utilisé dans les jeux-vidéo, on appelle ce courant le néo-pixel art. Le but ici est de développer un outil facilitant la création de ce genre d'œuvre voir même d'en faire un outil pratique pour concevoir des jeux-vidéo ou des animations.

Organisation

Pour n'importe quel projet d'envergure, il est nécessaire d'avoir une bonne organisation et de la tenir. Il a donc été décidé d'utiliser un script de gestion des tâches développé par mes soins à l'aide de *Google Sheets* pour répondre le plus précisément à toutes nos attentes. Nous avons trouvé intéressant le fait de suivre la vitesse nécessaire à l'accomplissement de chaque tâche, afin grâce à un simple calcul de statistique pour pouvoir estimer le temps nécessaire pour finir les futures tâches.

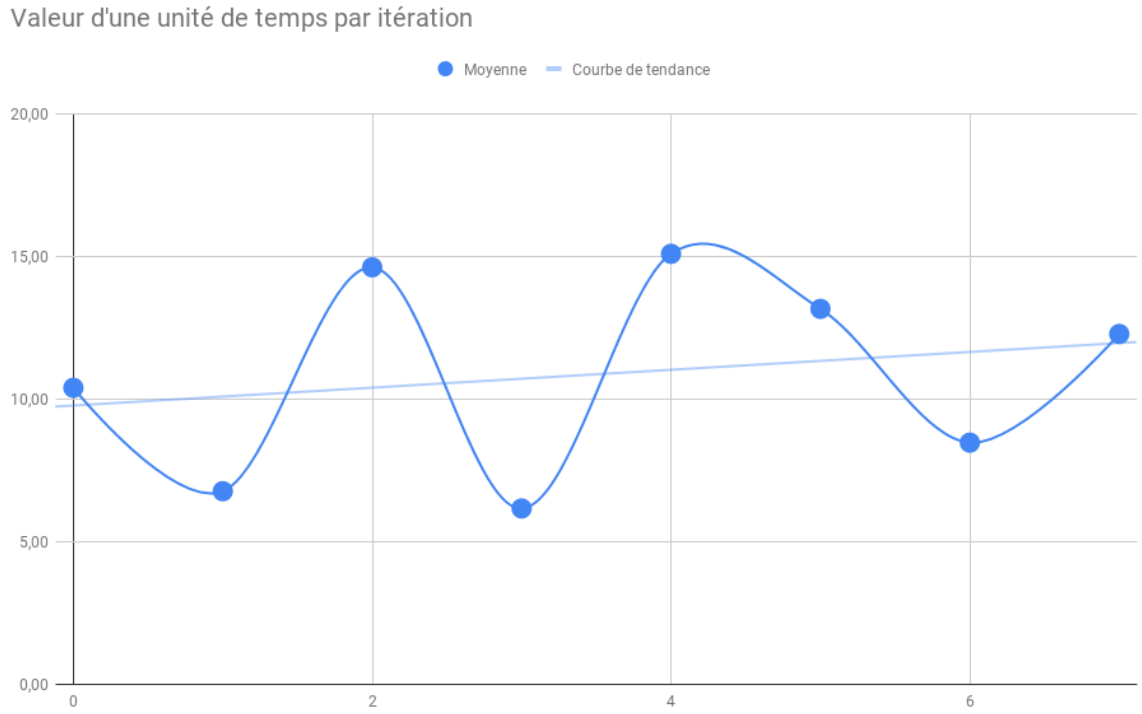
État	Description	Priorité
Fait	Réfactoring	0
Fait	Ajouter une interface pour les couleurs personalisées	1
Fait	Ajouter la possibilité de dessin en laissant le clique enfoncé	1
Fait	Fusionner interface couleur personalisées et dessin avec clique enfoncé	1,1
Fait	Ajouter la possibilité de créer des animations (regrouper des sprites dans un ordre précis)	2

À chaque tâche est attribuée une priorité (chaque semaine, une nouvelle feuille de tâche est créée et donc les priorité sont remises à 0).

Temps relatif	Date début	Date fin	Temps total (min)	Temps pour 1 relatif (min)	Relatif moyen (min)
4	06/05 09:18:15	06/05 09:40:14	21,98	5,50	8,47
8	06/05 10:14:50	06/05 10:41:57	27,10	3,39	
2	06/05 09:48:30	06/05 10:14:37	26,12	13,06	
1	06/05 10:51:08	06/05 11:02:11	11,07	11,07	
14	08/05 18:25:24	08/05 20:36:09	130,77	9,34	

On attribue aussi un temps relatif aux autres tâches, à la fin de chacune un temps pour une relative est donc calculée par une simple division (Système de temps relatif inspiré du principe expliqué dans le livre *Agile software development principles patterns and practices*[1]).

Nous pouvons alors en tirer des informations à propos de la production de l'équipe, même si dans ce cas le projet a été mené sur une durée assez courte (une dizaine de semaines).



[1] ***Agile software development principles patterns and practices*** **est un livre écrit par Robert Cecil Martin parlant de différents principes de développement logiciel.**

Choix des technologies

Pour ce projet, il a été décidé de concevoir une page Web, ce qui permet une meilleur accessibilité et une implémentation de l'interface plus facile. De plus, les cours d'ISN sont seulement portés sur le Javascript, le HTML et le CSS. Il aurait été plus difficile de s'orienter vers des technologies inconnues de mon partenaire.

Architecture du projet

Voici les éléments que l'on trouve à la racine du projet:

- LICENSE **précisant la licence à laquelle est soumise tout le code du projet**
- index.html **qui contient le code html de la page**
- style/ **contenant tous les fichiers CSS**
- src/ **qui englobe tout le code de production**
- modules/ **permet de stocker toutes les dépendances**
- test/ **le dossier pour le programme testant les classes du code de production**

La licence à laquelle est soumise le projet est la licence MIT, qui permet à qui que ce soit de réutiliser le code, de le vendre, de l'afficher sur internet… La seule obligation est d'inclure le fichier LICENCE **contenant lui-même toutes ces conditions.**

Une histoire de Sprite

Pour modéliser une image en pixel art, il aurait été difficile d'utiliser directement l'image stockée par un canvas (les pixels auraient été trop petits). Il a donc été décidé de créer une classe nommée Sprite**. Cette classe est une interface liant toutes les composantes liées à l'image:**

- SpriteDataHandler **pour la gestion des données (état des pixels)**
- SpritePixelColorHandler **pour la gestion de la couleur des pixels**
- SpritePositionCalculator **qui sert à calculer les positions**
- SpritePixelDrawer **pour modifier l'état des pixels**
- SpriteDrawer **pour dessiner le sprite sur le canvas**

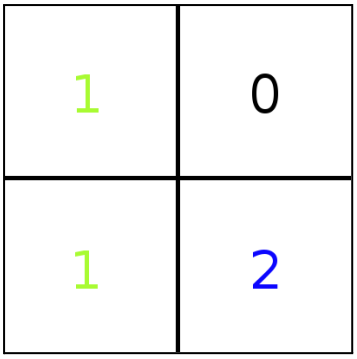
SpriteDataHandler

Cette classe stocke l'id de la couleur des pixels sous forme de tableau et aussi les dimensions du Sprite **(**width **et** height**).**

SpritePixelColorHandler

Sa fonction est de gérer les couleurs utilisées dans le Sprite**, elle contient le tableau** pixelColors **qui va représenter toutes les couleurs utilisées par le Sprite, et l'id de la dite couleur est tout simplement son indice dans ce même tableau. Il y a aussi les variables** drawingColor **et** cleaningColor.

On remarque que cette classe à deux responsabilités: le stockage des couleurs utilisées et la mémorisation des couleurs sélectionnées pour dessiner et effacer.



Nous pouvons penser à une fragmentation de la classe pour garantir une meilleure cohésion des variables d'instance à l'intérieure des fonctions.

SpritePositionCalculator

Cette classe permet d'effectuer tous les calculs relatifs à la position sur un sprite. Comme calculer l'indice d'un pixel avec ces coordonnées x et y et déduire sur quel pixel se trouve une position sur un canvas.

SpritePixelDrawer

Elle permet tout simplement de dessiner sur le Sprite à l'aide de la couleur de dessin et la "gomme". Il serait peut-être plus raisonnable de ranger les variables des couleurs de dessin et d'effaçage dans cette classe plutôt que dans SpritePixelColorHandler**.**

SpriteDrawer

Cette classe est l'unité permettant de dessiner le sprite sur le canvas, elle va dessiner chaque pixel un par un.

Initialisation du programme

Lors du chargement de la fenêtre, une fonction anonyme est lancée (dans le fichier ./src/initialisation.js**). Elle va appeler d'autres sous fonctions qui vont initialiser des parties précises du programme (les éléments de l'interface, le sprite, le canvas, …).**

À l'heure où j'écris ces lignes, il existe 3 variables globales, qui sont respectivement:

- canvas**: contient l'élément canvas**
- sprite**: tout simplement une instance de Sprite**
- isDrawing**: permet de se souvenir si l'utilisateur est en train de dessiner un trait ou non).**

Dessin sur le canvas/détection de cliques

La déclaration de tous les listeners du canvas se font dans la fonction initialiserCanvas()**. Voici les résultats attendu pour dessiner sur le Sprite:**

1. **Lors d'un clique sur le canvas, le pixel cliqué doit se dessiner**

2. **Lors d'un maintien du clique gauche et un glissement de la souris sur le canvas, un trait doit se dessiner en accord avec la position du curseur.**
3. **Si le curseur quitte le canvas lors du dessin d'un trait, et que le bouton est relâché en dehors, revenir sur le canvas ne déclenche pas le dessin d'autres traits.**
4. **Si le curseur quitte le canvas et y revient tout en gardant le clic gauche enfoncé, un trait doit toujours se dessiner sous le curseur.**

La règle 1 est facile à mettre en œuvre, il suffit de détecter lorsque l'utilisateur clique sur le canvas (événement 'mousedown'**), si oui il suffit de dessiner le pixel concerné.**

L'événement 'mousemove' **symbolise un mouvement de la souris sur un élément. Il faut alors rattacher cet événement au dessin du pixel sous la souris lorsque le clique droit est pressé. On utilise la variable** isDrawing **qui va être affectée à** true **lorsque** 'mousedown' **est lancé sur le canvas et affectée à** false **lorsque l'événement** 'mouseup' **survient sur toute la page. Après ça, toutes les règles sont respectées.**

Bogues répertoriés

À ce jour deux cas ont été détectés et pas réparés:

- **Le clique droit permet aussi de dessiner sur le sprite**
- **Valider les dimensions du canvas sans avoir mis aucune valeur causera une erreur critique**

La première est sûrement causée par le fait que 'mousedown' **et** 'mouseup' **sont aussi appelés lors d'un clic droit. On pourrait régler ce problème en vérifiant à chaque fois quel bouton est appuyé ou relâché.**

L'autre est due à coup sûr à une injection de valeur de mauvais type dans les attributs du canvas (même si cela reste une conjecture).

Améliorations possible

Un projet n'est jamais terminé, il est toujours possible d'améliorer des fonctionnalités, d'en rajouter de nouvelles.

Idées d'améliorations:

- **Pouvoir sauvegarder/importer des sprites**
- **Ajouter de la gestion des animations (pouvoir créer des animations à partir de plusieurs sprites et ainsi générer des spritesheets pouvant être utilisé hors de la page)**
- **Améliorer l'interface et le style de la page pour la rendre plus esthétique et ergonomique.**

Annexe

./LICENCE

```
MIT License

Copyright (c) 2019 Louis Masson & Maxence Houguet

Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the "Software"), to deal in the Software
without restriction, including without limitation the rights to use, copy, modify,
merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice shall be included in all copies or
substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

./src/initialisation.js

```js
let canvas;
let sprite;

let isDrawing = false;

// Louis et Maxence
window.onload = function() {
    initialiserInterfaces();
    initialiserSprite();
    initialiserCanvas();
    sprite.draw();
};

// Louis et Maxence
function initialiserInterfaces() {
    initialiserInterfaceDimensionsCanvas();
    initialiserInterfaceDimensionsSprite();
    initialiserCouleurInterface();
    initialiserCouleursPersonnalisees();
    initialiserClearSprite();
    initialiserFillSprite();
}
```

```
// Maxence
function initialiserSprite(){
    if(Number(hauteurSpriteInput.value) < 1 || Number(largeurSpriteInput.value) < 1 ){
        sprite = new Sprite(10, 10);
    }
    else{
        sprite = new
Sprite(Number(largeurSpriteInput.value),Number(hauteurSpriteInput.value));
    }
}

// Louis
function initialiserCanvas() {
    canvas = document.getElementById("ecran");

    canvas.addEventListener('mousedown', canvasDrawOnListener);
    document.body.addEventListener('mouseup', canvasDrawOffListener);

    canvas.addEventListener('mousedown', canvasDrawListener);
    canvas.addEventListener('mousemove', canvasDrawListener);

    setTargetContext(canvas.getContext("2d"));
    background(new Color(0));
}

// Louis
function canvasDrawOnListener() {
    isDrawing = true;
}

// Louis
function canvasDrawOffListener() {
    isDrawing = false;
}

// Louis
function canvasDrawListener(evt) {
    if(isDrawing) {
        let rectangle = evt.target.getBoundingClientRect();
        let positionOnCanvas = new Vector(evt.clientX - rectangle.left, evt.clientY -
rectangle.top);

        let positionOnSprite = sprite.getPixelPositionByCanvas(positionOnCanvas.x,
positionOnCanvas.y, canvas.width, canvas.height);

        sprite.drawPixel(positionOnSprite.x, positionOnSprite.y);
        sprite.draw();
    }
}
```

./src/Sprite/Sprite.js

```
/*
 *
 *  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
 *    Attention: Les accesseurs ont été retirés afin de
 *               raccourcir la taille du fichier.
 *
 *  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *
*/
'use strict';

class Sprite {
    constructor(width=0, height=0) {
        this.dataHandler = new SpriteDataHandler(width, height);
        this.positionCalculator = new SpritePositionCalculator(this);
        this.pixelColorHandler = new SpritePixelColorHandler(this);
        this.pixelDrawer = new SpritePixelDrawer(this);
        this.drawer = new SpriteDrawer(this);
    }

    draw() {
        this.drawer.draw();
    }

    drawPixel(x, y) {
        this.pixelDrawer.drawPixel(x, y);
    }

    clearPixel(x, y) {
        this.pixelDrawer.clearPixel(x, y);
    }

    addColor(color) {
        this.pixelColorHandler.addColor(color);
    }

    copy() {
        let sprite = new Sprite();

        sprite.dataHandler = this.dataHandler.copy();
        sprite.pixelColorHandler = this.pixelColorHandler.copy(sprite);

        return sprite;
    }
}
```

./src/Sprite/SpriteDataHandler.js

```
'use strict';
```

```
class SpriteDataHandler {
    constructor(width=0, height=0) {
        this.width = width;
        this.height = height;
        this.data = [];

        for(let i = 0; i < width*height; i++)
                this.data.push(0);
    }

    copy() {
        let dataHandler = new SpriteDataHandler();

        dataHandler.width = this.width;
        dataHandler.height = this.height;
        dataHandler.data = this.data.slice();

        return dataHandler;
    }
}
```

./src/Sprite/SpritePositionCalculator.js

```
'use strict';

class SpritePositionCalculator {
    constructor(sprite) {
        this.sprite = sprite;
    }

    getIndex(x, y) {
        if(this.areGetIndexParametersIncorrect(x, y))
                return 0;

        return y*this.sprite.getWidth() + x;
    }









    getPixelPositionByCanvas(x, y, canvasWidth, canvasHeight) {
        if(this.areGetPixelPositionByCanvasParametersIncorrect(x, y, canvasWidth,
canvasHeight))
                return new Vector(0, 0);
```

```
        let xScale = canvasWidth/this.sprite.getWidth();
        let yScale = canvasHeight/this.sprite.getHeight();

        return new Vector( Math.floor(x/xScale),
                           Math.floor(y/yScale) );
    }

    areGetIndexParametersIncorrect(x, y) {
        if(x < 0 || x >= this.sprite.getWidth())
               return error("x has a wrong value ! (" + x + ")", true, console.warn);
        if(y < 0 || y >= this.sprite.getHeight())
               return error("y has a wrong value ! (" + y + ")", true, console.warn);

        return false;
    }

    areGetPixelPositionByCanvasParametersIncorrect(x, y, canvasWidth, canvasHeight) {
        if(this.sprite.getData().length <= 0)
               return error("There's no content in the this.sprite.", true,
console.warn);
        if(x < 0 || x >= canvasWidth)
               return error("x has a wrong value ! (" + x + ")", true, console.warn);
        if(y < 0 || y >= canvasHeight)
               return error("y has a wrong value ! (" + y + ")", true, console.warn);

        return false;
    }
}
```

./src/Sprite/SpritePixelColorHandler.js

```
'use strict';

class SpritePixelColorHandler {
    constructor(sprite) {
        this.drawingColor = PixelColors.BLACK;
        this.cleaningColor = PixelColors.GRAY;

        this.pixelColors = [];
        this.initPreselectionColors();
    }


    initPreselectionColors() {
        this.addColor(new Color(100)); // PixelColors.GRAY
        this.addColor(new Color(255, 255, 255)); // PixelColors.WHITE
        this.addColor(new Color(0)); // PixelColors.BLACK
        this.addColor(new Color(52, 52, 255)); // PixelColors.BLUE
        this.addColor(new Color(52, 255, 52)); // PixelColors.GREEN
        this.addColor(new Color(255, 52, 52)); // PixelColors.RED
        this.addColor(new Color(255, 255, 52)); // PixelColors.YELLOW
    }
```

```
    copy(sprite=this.sprite) {
        let pixelColorHandler = new SpritePixelColorHandler(sprite);

        pixelColorHandler.pixelColors = this.pixelColors;
        pixelColorHandler.drawingColor = this.drawingColor;
        pixelColorHandler.cleaningColor = this.cleaningColor;

        return pixelColorHandler;
    }

    addColor(color) {
        this.pixelColors.push(color);
    }

    getPixelColor(pixelColorId) {
        if(pixelColorId >= this.pixelColors.length)
               return error('this.drawingColor has a wrong value!', new Color(255));
        return this.pixelColors[pixelColorId];
    }

    getDrawingColorId() {
        return this.drawingColor;
    }

    getCleaningColorId() {
        return this.cleaningColor;
    }


    getLastColorId() {
        return this.pixelColors.length - 1;
    }
}
```

./src/Sprite/SpritePixelDrawer.js

```
'use strict';

class SpritePixelDrawer {
    constructor(sprite) {
        this.sprite = sprite;
    }

    drawPixel(x, y) {
        this.sprite.setPixel(this.sprite.getDrawingColorId(), this.sprite.getIndex(x,
y));
    }

    clearPixel(x, y) {
        this.sprite.setPixel(this.sprite.getCleaningColorId(), this.sprite.getIndex(x,
y))
    }
}
```

./src/Sprite/SpriteDrawer.js

```
'use strict';

class SpriteDrawer {
    constructor(sprite) {
        this.sprite = sprite;
    }

    // auteurs: Louis et Maxence
    draw() {
        let largeurPixel = canvas.width/this.sprite.getWidth();
        let hauteurPixel = canvas.height/this.sprite.getHeight();

        for(let x = 0; x < this.sprite.getWidth(); x++) {
               for (let y = 0; y < this.sprite.getHeight(); y++) {
                      stroke(new Color(255));
                      fill(
      this.sprite.getPixelColor(this.sprite.getData()[this.sprite.getIndex(x, y)])
                           );

                      rect( x*largeurPixel, y*hauteurPixel,
                           largeurPixel, hauteurPixel );
               }
        }
    }
}
```

./src/Sprite/PixelColors.js

```
'use strict';

// enum
const PixelColors = {
    GRAY: 0,
    WHITE: 1,
    BLACK: 2,
    BLUE: 3,
    GREEN: 4,
    RED: 5,
    YELLOW: 6,
}
```