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.

3 Tilemaps

Une tilemap est une carte constituée d'éléments de pavage de taille égale (les tiles) ; l'agencement des tiles permet de construire une carte cohérente et de fait d'obtenir une carte en utilisant moins de mémoire qu'avec une seule image pour toute la carte ; cette section présente le principe d'une tilemap statique, d'une tilemap dynamique et de la génération procédurale de tilemap.

3.1 Tilemap statique

Une tilemap statique est une carte dont la composition est définie avant le chargement du jeu ; la carte peut varier en fonction des actions du joueur et en fonction du temps mais le joueur n'a pas d'action directe sur la composition de la carte ; il est courant d'utiliser un fichier PNG et un fichier JSON pour définir une tilemap ; le fichier PNG définit des tiles et le fichier JSON définit l'organisation de ces tiles pour construire la carte désirée.

L'utilitaire Tiled [5] permet de réaliser des tilemaps en générant un fichier JSON à partir d'un fichier PNG ; Tiled est compatible avec différents moteurs de jeu ; dans un fichier JSON de tilemap, les tiles sont agencés en couche, selon différents paramètres ; on peut également spécifier des positions clés comme le point de départ du joueur dans la carte, ou encore la position intiiale de personnages non joueurs ; l'organisation en couches permet de définir un ordre d'affichage des éléments de la carte (objets, décors, personnages joueurs et non joueurs) ; le fichier PNG définit un ensemble de tiles appelé tileset qui est indexé pour construire des cartes à partir de tableaux de ces indexes ; une carte correspond donc à une superposition de matrices d'indexes ; pour une matrice de taille $n \times m$ et des tiles de $a \times b$ pixels, on obtient une carte de $na \times mb$ pixels.

A la différence d'un atlas, les tiles sont espacés entre eux ; dans le tileset d'introduction de Tuxemon, les tiles sont espacés du bord par un pixel et entre eux par deux pixels ; Fig. 11 présente le tileset et le programme suivant présente les 50 premiers tiles du tileset dont le résultat est présenté Fig. 12.

5. `https://www.mapeditor.org/`.

```
let cnv = document.getElementById("myCanvas");
let ctx = cnv.getContext("2d");
let tlm;
let tls;
let tls_elts = [];

function onload_tlm () {
  if(this.status == 200) {
    tlm = JSON.parse(this.responseText);
    tls = new Image();
    tls.src = tlm["tilesets"][0]["image"];
    tls.onload = function() {
      let tls_imageheight = tlm["tilesets"][0]["imageheight"];
      let tls_imagewidth = tlm["tilesets"][0]["imagewidth"];
      let tls_margin = tlm["tilesets"][0]["margin"];
      let tls_spacing = tlm["tilesets"][0]["spacing"];
      let tls_tileheight = tlm["tilesets"][0]["tileheight"];
      let tls_tilecount = tlm["tilesets"][0]["tilecount"];
      ctx.beginPath();
      ctx.fillStyle = "#FF0000";
      ctx.fillRect(0, 0, tls_imagewidth, 102);
      ctx.closePath();
      let canvas = document.createElement('canvas');
      canvas.height = tls_imageheight;
      canvas.width = tls_imagewidth;
      let context = canvas.getContext('2d');
      context.drawImage(tls, 0, 0, tls.width, tls.height);
      for(let ih = tls_margin; ih < tls_imageheight; ih+=(tls_tileheight+tls_spacing)) {
        for(let iw = tls_margin; iw < tls_imagewidth; iw+=(tls_tileheight+tls_spacing)) {
          let canvasImageData = context.getImageData(iw, ih,
                                        tls_tileheight, tls_tileheight);
          let canvasData = canvasImageData.data;
          tls_elts.push(canvasImageData);
          if(tls_elts.length <= 50) {
            ctx.putImageData(canvasImageData, iw, ih);
          }
        }
      }
    }
  }
}
let xobj = new XMLHttpRequest();
xobj.onload = onload_tlm;
xobj.overrideMimeType("application/json");
xobj.open("GET", "./tilemaps/tuxemon-town.json", true);
xobj.send();
```

FIG. 11 – Tileset d'introduction de Tuxemon.

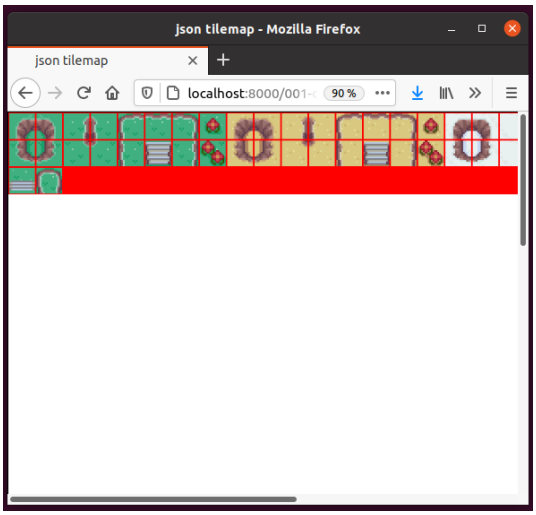

FIG. 12 – 50 premiers pavés du tileset de Tuxemon.

Ayant découpé le tileset en tiles, il s'agit d'utiliser les tableaux d'indexes nommés `layers` dans le fichier JSON pour reconstitué la tilemap ; Fig. **??** présente le résultat de l'exécution du programme ci-dessous, pour les deux premières couches avec la fonction `putImageData` ne tenant pas compte des transparences.

```
let layer0_data = tlm["layers"][0]["data"];
let layer0_height = tlm["layers"][0]["height"];
let layer0_width = tlm["layers"][0]["width"];
let layer1_data = tlm["layers"][1]["data"];
let layer1_height = tlm["layers"][1]["height"];
let layer1_width = tlm["layers"][1]["width"];
let layer0_data_i = 0;
for(let ih = 0; ih < layer0_height; ih += 1) {
  for(let iw = 0; iw < layer0_width; iw += 1) {
    if(layer0_data[layer0_data_i] > 0)
      ctx.putImageData(tls_elts[layer0_data[layer0_data_i]-1],
                       iw*tls_tileheight, ih*tls_tileheight);
    layer0_data_i += 1;
  }
}
let layer1_data_i = 0;
for(let ih = 0; ih < layer1_height; ih += 1) {
  for(let iw = 0; iw < layer1_width; iw += 1) {
    if(layer1_data[layer1_data_i] > 0)
      ctx.putImageData(tls_elts[layer1_data[layer1_data_i]-1],
                       iw*tls_tileheight, ih*tls_tileheight);
    layer1_data_i += 1;
  }
}
```

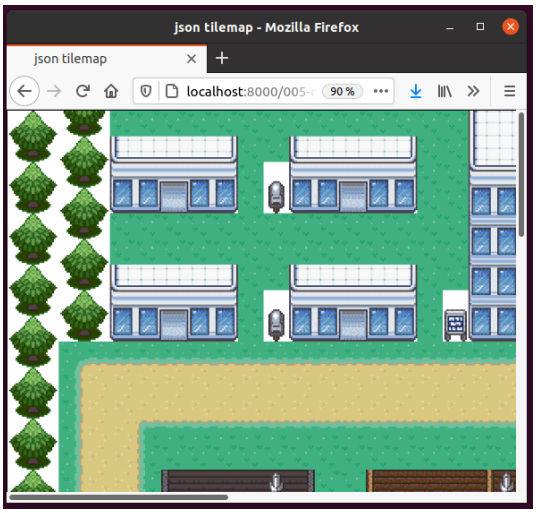

FIG. 13 – Reconstruction avec putImageData.

On peut résumer la solution précédente à l'articulation entre un premier double-for pour découper les tiles et une second double-for pour reconstruire la tilemap.

```
for(let ih = tls_margin; ih < tls_imageheight; ih+=(tls_tileheight+tls_spacing)) {
  for(let iw = tls_margin; iw < tls_imagewidth; iw+=(tls_tileheight+tls_spacing)) {
    let canvasImageData = context.getImageData(iw, ih,
                                    tls_tileheight, tls_tileheight);
    let canvasData = canvasImageData.data;
    tls_elts.push(canvasImageData);
  }
}
...
for(let ih = 0; ih < layer0_height; ih += 1) {
  for(let iw = 0; iw < layer0_width; iw += 1) {
    if(layer0_data[layer0_data_i] > 0)
      ctx.putImageData(tls_elts[layer0_data[layer0_data_i]-1],
                       iw*tls_tileheight, ih*tls_tileheight);
    layer0_data_i += 1;
  }
}
// layers1, ...
```

Afin de superposer les couches en utilisant la transparence, on remplace l'articulation précédente par l'articulation suivante pour obtenir Fig. 14.

```
for(let ih = tls_margin; ih < tls_imageheight; ih+=(tls_tileheight+tls_spacing)) {
  for(let iw = tls_margin; iw < tls_imagewidth; iw+=(tls_tileheight+tls_spacing)) {
    let canvas2 = document.createElement('canvas');
    canvas2.height = tileset_tileheight;
    canvas2.width = tileset_tileheight;
    let context2 = canvas2.getContext('2d');
    let canvasImageData = context.getImageData(iw, ih,
                                    tls_tileheight, tls_tileheight);
    context2.putImageData(canvasImageData, 0,0);
    tls_elts.push(canvas2);
  }
}
...
for(let ih = 0; ih < layer0_height; ih += 1) {
  for(let iw = 0; iw < layer0_width; iw += 1) {
    if(layer0_data[layer0_data_i] > 0)
      ctx.drawImage(tls_elts[layer0_data[layer0_data_i]-1],
                       iw*tls_tileheight, ih*tls_tileheight);
    layer0_data_i += 1;
  }
}
// layers1, ...
```

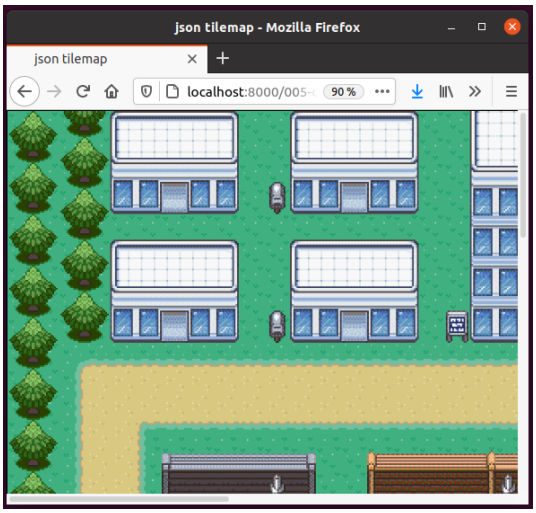

FIG. 14 – Reconstruction avec drawImage.

3.2 Tilemap dynamique

Une tilemap dynamique est une carte dont la composition est modifiable par le joueur pendant le jeu ; le joueur peut agir directement sur la carte ; en guise d'exemple, misa va pouvoir ajouter des arbres sur une carte ; le programme js est divisé en classes, lesquelles sont associées à des fichiers distincts :

- le fichier `SpriteAnime.js` gère l'animation des sprites
- le fichier `Character.js` gère l'animation et la position d'un personnage
- le fichier `TreeGrid.js` gère l'ajout d'arbres selon une grille dans le décor
- le fichier `prg.js` gère l'ensemble du programme en utilisant les classes définies dans les fichiers précédents

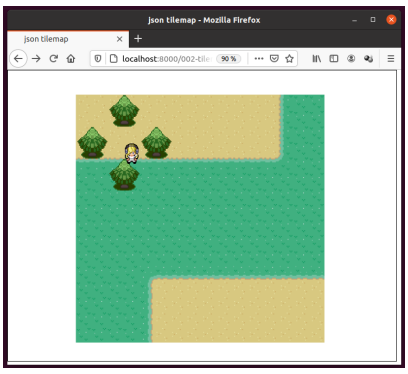

FIG. 15 – Avec 4 arbres.

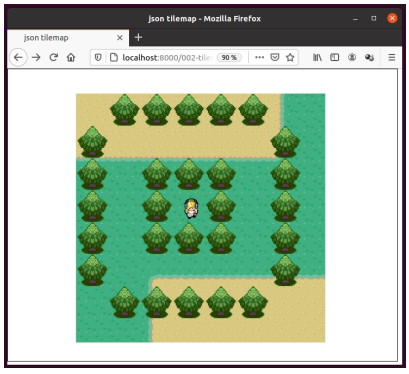

FIG. 16 – Avec 28 arbres.

Le joueur peut déplacer Misa et planter des arbres face à Misa en appuyant sur la touche `a`.

Le fichier `SpriteAnime.js` définit la classe SpriteAnime et permet de gérer une animation ; ses paramètres sont :

- le contexte du canvas (nommé `ctx`) dans lequel prendre les images pour construire l'animation
- le fichier JSON (nommé `json`) associé au contexte pour obtenir les coordonnées des images considérées
- le préfix des identifiants des images (nommé `prefix`) dans le fichier JSON
- le premier numéro des images (nommé `first_id`)
- le dernier numéro des images (nommé `last_id`)
- le numéro de la première image pour l'animation (nommé `cur_id`)
- l'évènement associé à l'animation (nommé `e`)
- le décallage en x associé à la fenêtre dans laquelle sera l'animation (nommé `dx`)
- le décallage en y associé à la fenêtre dans laquelle sera l'animation (nommé `dy`)

La fonction `getImage` permet de récupérer le canvas associé à l'image de l'animation en cours.

```
export default class SpriteAnime {
  constructor(ctx, json, prefix, first_id, last_id, cur_id, e, dx, dy) {
    this.event_code = e;
    this.dx = dx;
    this.dy = dy;
    this.animestep = cur_id;
    this.animeseq = [];
    for(let i = first_id; i < last_id+1; i += 1) {
      let filename = prefix;
      if(i < 10) { filename += ".00"+i; }
      else if(i < 100) { filename += ".0"+i; }
      else { filename += "."+i; }
      let x = json["frames"][filename]["frame"]["x"];
      let y = json["frames"][filename]["frame"]["y"];
      let w = json["frames"][filename]["frame"]["w"];
      let h = json["frames"][filename]["frame"]["h"];
      let canvasImageData1 = ctx.getImageData(x, y, w, h);
      let canvas2 = document.createElement('canvas');
      canvas2.width = w;
      canvas2.height = h;
      let context2 = canvas2.getContext('2d');
      context2.putImageData(canvasImageData1, 0,0);
      this.animeseq.push(canvas2);
    }
  }
  getAnime() {
    return this.animeseq[this.animestep];
  }
}
```

Le fichier `Character.js` définit la classe Character et permet de gérer position et animation d'un personnage ; dans ses données internes, le tableau `animes` regroupe des objets de type `SpriteAnime` ; la variable `animeId` définit le `SpriteAnime` en cours ; la fonction `next_step` met-à-jour la position du personnage (lignes 22 à 30) et l'animation associée à l'évènement (nommé `event_code`) ; si l'évènement est différent du précédent, l'animation est réinitialisée (lignes 12 à 19) ; la variable `anime_freq_step` réduit la fréquence des animations.

```
1  export default class Character {
2    constructor(anime_freq) {
3      this.last_event = null;
4      this.animes = [];
5      this.anime_freq = anime_freq;
6      this.anime_freq_step = anime_freq;
7      this.posx = 100;
8      this.posy = 100;
9      this.animeId = 0;
10   }
11   next_step(minx, miny, maxx, maxy, event_code) {
12     if(event_code != this.last_event) {
13       this.animestep = 0;
14       for(let i = 0; i < this.animes.length; i += 1) {
15         if(this.animes[i].event_code == event_code) {
16           this.animeId = i;
17           this.last_event = event_code;
18         }
19       }
20     } else {
21       if(this.anime_freq_step <= 0) {
22         this.animes[this.animeId].animestep += 1;
23         if(this.animes[this.animeId].animestep >=
24            this.animes[this.animeId].animeseq.length)
25              this.animes[this.animeId].animestep = 0;
26         this.posx += this.animes[this.animeId].dx;
27         this.posy += this.animes[this.animeId].dy;
28         if(this.posx < minx) this.posx = minx;
29         if(this.posx > maxx) this.posx = maxx;
30         if(this.posy < miny) this.posy = miny;
31         if(this.posy > maxy) this.posy = maxy;
32         this.anime_freq_step = this.anime_freq;
33       } else { this.anime_freq_step -= 1; }
34     }
35   }
36   getImage() {
37     if(this.animeId >= this.animes.length) { return new Image(); }
38     return this.animes[this.animeId].getAnime();
39   }
40 }
```

Le fichier `TreeGrid.js` définit la classe TreeGrid et permet de gérer une grille d'occupation du sol ; une valeur 1 dans le tableau `grid` indique la présence d'un arbre ; le décallage de 64 pixels (lignes 21 à 27) est approximatif.

```
1  export default class TreeGrid {
2    constructor(x, y, dx, dy) {
3      this.dx = dx; this.dy = dy;
4      this.nbc = Math.floor(x/dx);
5      this.nbl = Math.floor(y/dy);
6      this.maxx = this.nbc*this.dx; this.maxy = this.nbl*this.dy;
7      this.grid = new Array(this.nbc);
8      this.cnv = document.createElement('canvas');
9      this.cnv.width = dx; this.cnv.height = dy;
10     this.ctx = this.cnv.getContext("2d");
11     for(let i = 0; i < this.nbc; i += 1) {
12       this.grid[i] = [];
13       for(let j = 0; j < this.nbl; j += 1) this.grid[i].push(0);
14     }
15   }
16   addTree(character) {
17     let posx = character.posx;
18     let posy = character.posy;
19     switch(character.animeId) {
20       case 0: // right
21         posx = character.posx + 64; break;
22       case 1: // Left
23         posx = character.posx - 64; break;
24       case 2: // Back
25         posy = character.posy - 64; break;
26       case 3: // Front
27         posy = character.posy + 64; break;
28     }
29     let nx = Math.floor(posx/this.dx);
30     let ny = Math.floor(posy/this.dy);
31     if(this.grid[nx][ny] == 0) { this.grid[nx][ny] = 1; }
32   }
33   draw(ctx, mapdx, mapdy) {
34     if(this.cnv == null) return;
35     for(let i = 0; i < this.nbc; i += 1) {
36       for(let j = 0; j < this.nbl; j += 1)
37         if(this.grid[i][j] == 1)
38           ctx.drawImage(this.cnv, mapdx+i*this.dx, mapdy+j*this.dy);
39     }
40   }
41 }
```

L'import des classes est réalisé lignes 1 à 3 ; `tlm`, `tls` et `tls_elts` correspondent respectivement au fichier JSON, le tileset et les images du tileset ; la carte est reconstruite dans le canvas `map_cnv` dont le contexte est `map_ctx` ; les coordonnées `map_ori_x` et `map_ori_y` définissent la position de la carte dans le canvas principal.

```
1  import SpriteAnime from "./SpriteAnime.js";
2  import Character from "./Character.js";
3  import TreeGrid from "./TreeGrid.js";
4
5  let cnv = document.getElementById("myCanvas");
6  let ctx = cnv.getContext("2d");
7  let map_cnv = document.createElement('canvas');
8  map_cnv.width = cnv.width;
9  map_cnv.height = cnv.height;
10 let map_ctx = map_cnv.getContext("2d");
11 let tlm;
12 let tls;
13 let tls_elts = [];
14 let map_ori_x = 140;
15 let map_ori_y = 50;
16 let tlm_loaded = 0;
17 let atlas_loaded = 0;
18 let misa_character = new Character(1);
19 let new_trees = new TreeGrid(16*32, 16*32, 66, 66);
```

La fonction `onload_tlm` (page 39) reconstruit la carte et la dessine dans le contexte `map_ctx` du canvas `map_cnv`.

La fonction `onload_atlas` (page 41) définit les animations associées aux déplacements de Misa.

La fonction `keydown_fun` (page 41) définit les touches associées aux déplacements de Misa et les touches associées au placement d'un nouvel arbre ; selon la configuration, la touche `a` provoquera l'évènement `KeyA` ou `KeyQ`.

```
function onload_tlm () {
  if(this.status == 200) {
    tlm = JSON.parse(this.responseText);
    tls = new Image();
    tls.src = window.location.pathname+"assets/tilemaps/"+tlm["tilesets"][0]["image"];
    tls.onload = function() {
      let tls_imageheight = tlm["tilesets"][0]["imageheight"];
      let tls_imagewidth = tlm["tilesets"][0]["imagewidth"];
      let tls_margin = tlm["tilesets"][0]["margin"];
      let tls_spacing = tlm["tilesets"][0]["spacing"];
      let tls_tileheight = tlm["tilesets"][0]["tileheight"];
      let tls_tilecount = tlm["tilesets"][0]["tilecount"];
      let tlm_height = tlm["height"];
      let tlm_width = tlm["width"];
      let canvas = document.createElement('canvas');
      canvas.height = tls_imageheight;
      canvas.width = tls_imageheight;
      let context = canvas.getContext('2d');
      context.drawImage(tls, 0, 0, tls.width, tls.height);
      for(let ih = 1; ih < tls_imageheight; ih += (tls_tileheight+2)) {
        for(let iw = 1; iw < tls_imagewidth; iw += (tls_tileheight+2)) {
          let canvas2 = document.createElement('canvas');
          canvas2.height = tls_tileheight;
          canvas2.width = tls_tileheight;
          let context2 = canvas2.getContext('2d');
          let canvasImageData = context.getImageData(iw, ih, tls_tileheight,
                                                     tls_tileheight);
          context2.putImageData(canvasImageData, 0,0);
          tls_elts.push(canvas2);
        }
      }
      let canvasImageData = context.getImageData(0, 1+32*7+6*2, 64+2, 64+2);
      new_trees.ctx.putImageData(canvasImageData, 0,0);
      let layer0_data = tlm["layers"][0]["data"];
      let layer0_height = tlm["layers"][0]["height"];
      let layer0_width = tlm["layers"][0]["width"];
      let layer0_data_i = 0;
      let layer0_ix = map_ori_x;
      let layer0_iy = map_ori_y;
      for(let ih = 0; ih < layer0_height; ih += 1)
        for(let iw = 0; iw < layer0_width; iw += 1) {
          if(layer0_data[layer0_data_i] > 0)
            map_ctx.drawImage(tls_elts[layer0_data[layer0_data_i]-1],
              layer0_ix +iw*tls_tileheight, layer0_iy+ih*tls_tileheight);
          layer0_data_i += 1;
        }
    }
  }
  tlm_loaded = 1;
}
```

```
function onload_atlas () {
  if(this.status == 200) {
    let json_infos = JSON.parse(this.responseText);
    let spritesheet = new Image();
    spritesheet.src = "./assets/atlas/"+json_infos["meta"]["image"];
    spritesheet.onload = function() {
      let canvas1 = document.createElement('canvas');
      canvas1.width = json_infos["meta"]["size"]["w"];
      canvas1.height = json_infos["meta"]["size"]["h"];
      let context1 = canvas1.getContext('2d');
      context1.drawImage(spritesheet, 0,0,canvas1.width,canvas1.height);
      misa_character.animes.push(new SpriteAnime(context1, json_infos,
        "misa-right-walk",0,3,3,"ArrowRight",5,0));
      misa_character.animes.push(new SpriteAnime(context1, json_infos,
        "misa-left-walk",0,3,3,"ArrowLeft",-5,0));
      misa_character.animes.push(new SpriteAnime(context1, json_infos,
        "misa-back-walk",0,3,3,"ArrowUp",0,-5));
      misa_character.animes.push(new SpriteAnime(context1, json_infos,
        "misa-front-walk",0,3,3,"ArrowDown",0,5));
      atlas_loaded = 1;
    }
  }
}

window.addEventListener('keydown', keydown_fun, false);
function keydown_fun(e) {
  switch(e.code) {
    case "ArrowDown":
    case "ArrowUp":
    case "ArrowLeft":
    case "ArrowRight":
      misa_character.next_step(50, 50, 16*32-150, 16*32-150, e.code);
      break;
    case "KeyQ":
      new_trees.addTree(misa_character);
      break;
  }
}
```

Sont chargés le fichier JSON correspondant à la carte (lignes 1 à 5) puis celui correspondant au personnage de Misa (lignes 7 à 11).

La fonction `update` appelle la fonction `draw` pour redessiner l'ensemble de la scène, qui correspond à replacer la carte, puis les arbres, puis Misa.

```
1  let xobj_map = new XMLHttpRequest();
2  xobj_map.onload = onload_tlm;
3  xobj_map.overrideMimeType("application/json");
4  xobj_map.open("GET", "./assets/tilemaps/more_trees.json", true);
5  xobj_map.send();
6
7  let xobj_misa = new XMLHttpRequest();
8  xobj_misa.onload = onload_atlas;
9  xobj_misa.overrideMimeType("application/json");
10 xobj_misa.open("GET", "./assets/atlas/misa.json", true);
11 xobj_misa.send();
12
13 function draw() {
14   ctx.clearRect(0, 0, cnv.width, cnv.height);
15   let canvasImageData = map_ctx.getImageData(0, 0, cnv.width, cnv.height);
16   ctx.putImageData(canvasImageData, 0,0);
17   new_trees.draw(ctx, map_ori_x, map_ori_y);
18   ctx.drawImage(misa_character.getImage(),
19                 map_ori_x+misa_character.posx,
20                 map_ori_y+misa_character.posy);
21 }
22
23 function update(timestamp) {
24   draw();
25   requestAnimationFrame(update);
26 }
27 requestAnimationFrame(update);
```

3.3 Génération procédurale de tilemap

La génération procédurale est la création d'éléments de jeu à l'aide de générateurs ; elle implique la définition d'une procédure de création ; les éléménts de jeu concernés sont classiquement les niveaux du jeu, les personnages, les sons, les scénarios, les dialogues, les textures ; elle permet d'obtenir une grande quantité d'éléments de jeu.

Les Fig. 17 et 18 présentent deux cartes utilisant une génération procédurale de forêt ; les arbres sont placés aléatoirement dans la carte, à l'exception de la position de Misa ; Alg. 1 présente la procédure associée ; la fonction `initRandom` présente le programme js associé.

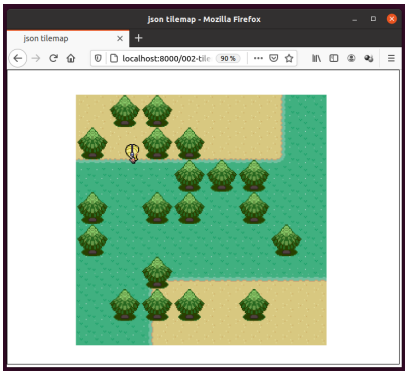

FIG. 17 – Génération d'arbres dans la carte.

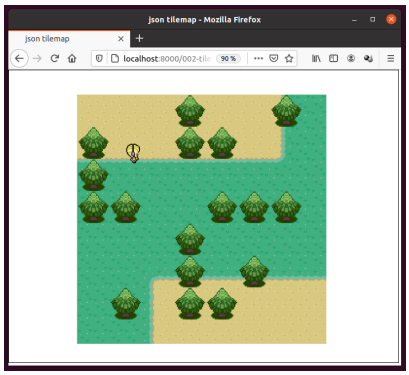

FIG. 18 – Génération d'arbres dans la carte.

```
1 fonction generate ( x_i, y_i ) :
2 |   a ← random ( ) ×nb_cols × nb_lines ;
3 |   for a times do
4 |   |   add-tree-at ( random ( ) ×nb_cols, random ( ) ×nb_lines) ;
5 |   remove-tree-at ( x_i/dx, y_i/dy) ;
```

Alg. 1: Génération de forêts.

```
1  initRandom(posx, posy) {
2    let nbrt = Math.floor(Math.random()*(this.nbc*this.nbl));
3    for(let i = 0; i < nbrt; i += 1) {
4      let rx = Math.floor(Math.random()*this.nbc);
5      let ry = Math.floor(Math.random()*this.nbl);
6      this.grid[rx][ry] = 1;
7    }
8    let nx = Math.floor(posx/this.dx);
9    let ny = Math.floor(posy/this.dy);
10   this.grid[nx][ny] = 0;
11 }
```