TD1-2 : Flash et ActionScript « avancé » V2.1.1

Document en ligne : mickael.martin.nevot.free.fr

1 Généralités

Ce TD s'inscrit dans la continuité du `TD1 : Flash et ActionScript`.

Vous devez développer les bases d'un jeu d'aventure de chasse au trésor 2D mono-joueur en Flash et ActionScript.

Pour réaliser le moteur 2D, vous devez mettre en place un moteur de *tile set*. Dans ce type de moteur, les niveaux sont créés par assemblage d'une multitude de petites images, appelées *tiles*, formant une image plus grande grâce à un quadrillage, appelé *tile set* (qui est donc défini par la largeur et la hauteur de ses *tiles*).

Vous trouverez les ressources multimédia nécessaires à la réalisation de ce TD sur le site Web de l'enseignant.

2 Préparation

Créez une nouvelle application Flash (répertoire de travail, fichier Flash, etc.) ayant `Application` comme classe de document.

Créez une classe ActionScript `Application` avec un gestionnaire de clavier :

```
package {
    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.events.KeyboardEvent;

    public class Application extends MovieClip {
        public function Application():void {
            super();

            this.addEventListener(Event.ADDED_TO_STAGE, this.addedToStage);
        }

        private function addedToStage(e:Event):void {
            this.removeEventListener(Event.ADDED_TO_STAGE, this.addedToStage);
```

```
            this.stage.addEventListener(KeyboardEvent.KEY_DOWN, keyDown);
        }

        private function keyDown(e:KeyboardEvent):void {
            trace(String.fromCharCode(e.keyCode));
            trace(e.keyCode);
        }
    }
}
```

3 Héros

Créez un clip nommé `Hero` comportant quatre images clefs. Chacune d'entre elles comportant à son tour un clip distinct. Chaque image clef (et clip correspondant) correspond à une orientation du héros :

Figure 1 – Composition du clip du héros

Voici la correspondance entre image clef et orientation du héros :

- 1 : haut ;
- 2 : gauche ;
- 3 : droite ;
- 4 : bas.

Chacun de ces clips a deux calques. Le premier comporte le corps (inamovible) du héros, le second est composé de six images dont chaque image impaire est clef :

Figure 2 – Timeline d'un clip d'orientation du héros

Chaque groupe de deux images est une position (à l'arrêt, un pied en avant, l'autre pied en avant) du héros lors de son déplacement dans une direction : réalisez les graphismes de ces positions en faisant en sorte qu'ils aient une dimension de 30x30 pixels et qu'ils soient alignés au centre :

Figure 3 – Exemple de décomposition de mouvement du héros

Pensez à nommer toutes les occurrences de tous les clips de la même façon :

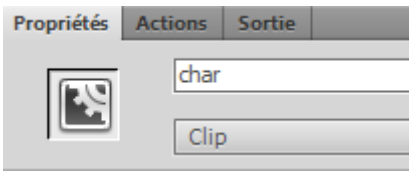

Figure 4 – Nom des occurrences des clips du héros

Créez une classe `Hero` avec son constructeur. Arrêtez la lecture de l'objet ainsi que l'ensemble des `MovieClip` qui le compose à sa création :

```
package {
    import flash.display.MovieClip;

    public class Hero extends MovieClip {
        // On déclare en public le nom d'occurrence des MovieClip.
        public var char:MovieClip;

        public function Hero():void {
            super();

            // On arrête la lecture de l'objet.
            stop();

            // On arrête la lecture de tous les MovieClip de l'objet.
            for (var i:uint = 0; i < this.numChildren ; ++i){
                if (this.getChildAt(i) is MovieClip) {
                    (this.getChildAt(i) as MovieClip).stop();
                }
            }
        }
    }
}
```

Liez le clip `Hero` à la classe `Hero` :

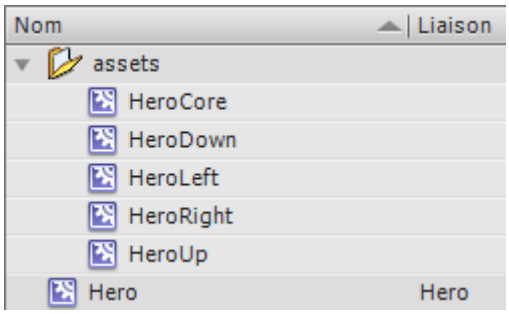

Figure 5 – Bibliothèque de clips pour le héros

4 Tiles

Créez maintenant un clip nommé `Tile` comportant au moins dix images clefs contenant chacune une image de 30x30 pixels (mettez tous les éléments traversables dans les premières images clefs) :

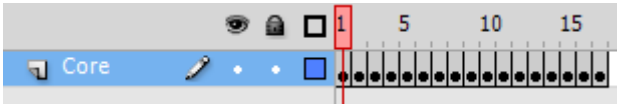

Figure 6 – Exemple de clip `Tile`

Pour cela, vous pouvez vous servir des éléments graphiques fournis sur le site Web de l'enseignant :

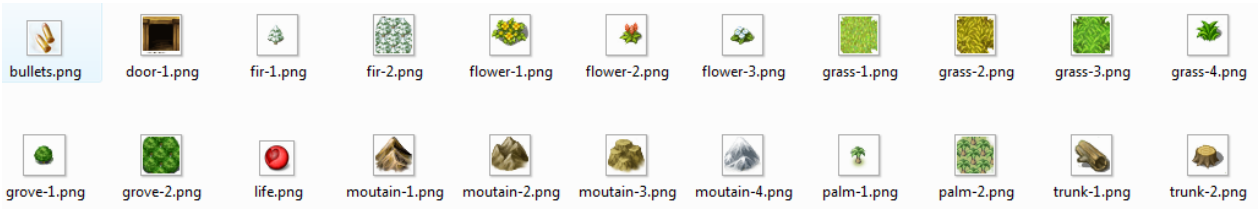

Figure 7 – Exemple d'éléments graphiques de *tiles*

Créez une classe `Tile` avec son constructeur et liez le clip `Tile` à la classe `Tile`. De nouveau, arrêtez la lecture de l'objet à sa création.

5 Création du niveau de jeu

Faites en sorte que la scène principale de l'application Flash ait une dimension de 450x480 pixels, qu'elle soit en 24 images par seconde et qu'elle ait une couleur de fond verte :

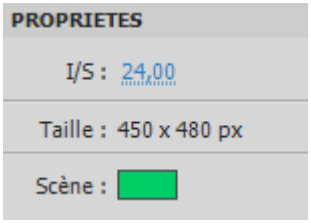

Figure 8 – Propriétés de la scène principale

Puis, dans la classe `Application`, créez un attribut de type tableau qui contient la carte de jeu. Pour ce faire, le tableau doit être multidimensionnel (15x16 éléments) et contenir des entiers qui doivent correspondre aux images clefs du clip `Tile`. Par exemple :

```
private var map:Array =
                [[16, 12, 16, 12, 12, 17, 12, 16, 16, 12, 17, 12, 17, 12, 17],
                [16, 1, 1, 1, 1, 2, 1, 1, 1, 12, 12, 12, 1, 1, 16],
                [16, 1, 1, 1, 1, 1, 1, 1, 1, 4, 5, 1, 1, 1, 16],
                [16, 1, 1, 1, 1, 1, 1, 1, 1, 17, 12, 1, 1, 1, 16],
                [16, 1, 5, 4, 1, 1, 1, 1, 1, 16, 1, 1, 1, 1, 17],
                [17, 1, 5, 5, 6, 1, 1, 16, 1, 16, 1, 1, 1, 1, 17],
                [17, 1, 5, 6, 6, 1, 1, 1, 1, 1, 1, 15, 15, 1, 16],
                [17, 1, 1, 7, 6, 1, 1, 1, 1, 2, 1, 1, 1, 1, 16],
                [17, 1, 11, 7, 7, 1, 11, 11, 1, 2, 1, 1, 3, 1, 12],
                [17, 1, 1, 7, 7, 1, 1, 1, 1, 2, 1, 3, 3, 3, 12],
                [17, 1, 9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 12],
                [17, 1, 9, 1, 1, 8, 8, 1, 1, 1, 1, 1, 1, 1, 12],
                [17, 1, 9, 1, 1, 8, 8, 1, 1, 1, 1, 1, 1, 1, 12],
                [17, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 10, 10, 10, 12],
                [12, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 13],
                [13, 13, 13, 14, 14, 14, 12, 12, 17, 12, 17, 12, 12, 13, 13]];
```

Il faut ensuite afficher la carte à chaque lancement de jeu. Pour cela créez une méthode qui parcourt l'ensemble du tableau et qui, pour chaque élément, crée un objet `Tile`, le positionne et enfin, l'affiche :

```
private function buildLevel():void {
    // On calcule les limites du tableau de la carte.
    var nbYLoop:uint = this.map.length;
    var nbXLoop:uint = (this.map[0] as Array).length;
```

```
    // On parcourt le tableau de la carte.
    for(var y:uint = 0 ; y < nbYLoop ; ++y) {
        for(var x:uint = 0 ; x < nbXLoop ; ++x) {
            // On récupère le numéro de tile courant qui
            // correspond à l'élément graphique correspondant.
            var tileNum:uint = this.map[y][x];
            // On crée un nouveau Tile.
            var newTile:Tile = new Tile();

            // On positionne le Tile nouvellement créé
            // (chaque tile a une dimension de 30x30 pixels).
            newTile.y = y * 30;
            newTile.x = x * 30;

            // On affiche le bon élément graphique.
            newTile.gotoAndStop(tileNum);

            // On ajoute le nouveau Tile à la liste d'affichage.
            this.stage.addChild(newTile);
        }
    }
}
```

6 Placement et déplacement du héros

Dans la classe `Application`, créez un attribut de type `Hero`.

Faites une méthode qui, au lancement de l'application, instancie l'attribut de type `Hero`, le positionne, l'oriente et l'affiche :

```
private function buildHero():void {
    this.hero = new Hero();
    this.hero.x = 45;
    this.hero.y = 45;
    this.hero.gotoAndStop(3);

    this.stage.addChild(this.hero);
}
```

Modifiez maintenant la méthode appelée par le gestionnaire de clavier pour prendre en compte le déplacement de `Hero` en faisant en sorte qu'il soit possible de se déplacer à la fois horizontalement et verticalement mais pas dans deux directions opposées. Pensez également à changer l'orientation de `Hero` :

```
private function keyDown(e:KeyboardEvent):void {
    var keyDown:uint = e.keyCode;

    // Déplacement horizontal.
    if (keyDown == Keyboard.LEFT) {
        this.hero.x -= 30;
        this.hero.gotoAndStop(2);
    } else if (keyDown == Keyboard.RIGHT) {
        this.hero.x += 30;
        this.hero.gotoAndStop(3);
    }
```

```
    // Déplacement vertical.
    if (keyDown == Keyboard.UP) {
        this.hero.y -= 30;
        this.hero.gotoAndStop(1);
    } else if (keyDown == Keyboard.DOWN) {
        this.hero.y += 30;
        this.hero.gotoAndStop(4);
    }
}
```

En l'état actuel, le déplacement de `Hero` est fonctionnel mais son animation de mouvement n'est pas satisfaisante puisqu'elle ne s'arrête pas lorsque `Hero` est immobile. Pour corriger cela et maîtriser l'animation, créez une méthode dans `Hero`, comme pour le `TD1 : Flash et ActionScript`, qui fasse en sorte d'avancer image par image à chaque nouvelle *frame* de rafraîchissement de `Hero`. De plus, une fois arrivée à la dernière image du clip, l'animation doit reprendre sur la première image et arrêter ce traitement de l'animation :

```
public function move():void {
    // On prépare le mouvement de Hero.
    this.addEventListener(Event.ENTER_FRAME, this.enterFrame);
}

private function enterFrame(event:Event):void {
    if (this.char.currentFrame == this.char.totalFrames) {
        this.char.gotoAndStop(1);
        this.removeEventListener(Event.ENTER_FRAME, this.enterFrame);
    } else {
        this.char.nextFrame();
    }
}
```

Pensez à appeler cette méthode dans `Application` à chaque déplacement de `Hero` (et à sa construction).

7 Gestion des collisions

Créez une nouvelle méthode dans la classe `Application` qui vérifie si un élément de la carte est franchissable en fonction de l'entier qui caractérise le *tile* et en fonction de la position du héros sur la carte :

```
/**
 * Vérifie si le tile est franchissable.
 * @return true si oui, false sinon.
 */
private function isWalkable(x:Number, y:Number):Boolean {
    var numFrame:uint = this.map[Math.floor(y / 30)][Math.floor(x / 30)];

    if (numFrame < 8) {
        // Franchissable.
        return true;
    } else {
        // Obstacles.
        return false;
    }
```

```
}
```

Pensez à appeler cette méthode dans `Application` à chaque déplacement horizontal ou vertical.

8 Chasse au trésor

Dans `Application`, créez un attribut de type tableau qui contient la carte des trésors de jeu (invisible).

Puis, modifiez la méthode de création de niveau pour qu'elle génère aléatoirement la carte des trésors de telle sorte que chacune des cases franchissables ait 10 % de chance de contenir un trésor.

Puis, créez une méthode qui, en fonction d'un élément de la carte au trésor, affiche s'il contient un trésor ou non et supprime la présence du trésor dans la carte.

Enfin, modifiez la méthode appelée par le gestionnaire de clavier pour prendre en compte la demande de fouille et l'appel de la méthode ci-dessus lorsque l'on appuie sur la touche `Espace` :

```
Pas de trésor
Pas de trésor
Trésor trouvé !
Pas de trésor
Pas de trésor
Pas de trésor
```

Figure 9 – Résultat d'une fouille de trésor

9 Améliorer l'application

L'application comporte maintenant ses bases fonctionnelles et devrait ressembler à cela :

Figure 10 – Exemple d'application finale

Réalisez une ou plusieurs de ces fonctionnalités (ou autres) :

- modification de l'affichage des *tiles* qui ont été fouillés ;
- matérialisation et comptabilisation des trésors découverts ;
- gestion de personnages non joueurs avec lesquels le héros peut dialoguer (très semblable au système de chasse au trésor) ;
- gestion d'ennemis, de combat et de la vie du héros ;
- gestion de bonus récupérables par le héros ;
- création de plusieurs niveaux et de *tiles* spéciaux servant de points de passage entre eux ;
- création de niveaux avec de grandes cartes et gestion du déplacement du décor ;
- intégration de sons et de musiques au jeu ;
- prévoir un mode de triche ;
- etc.