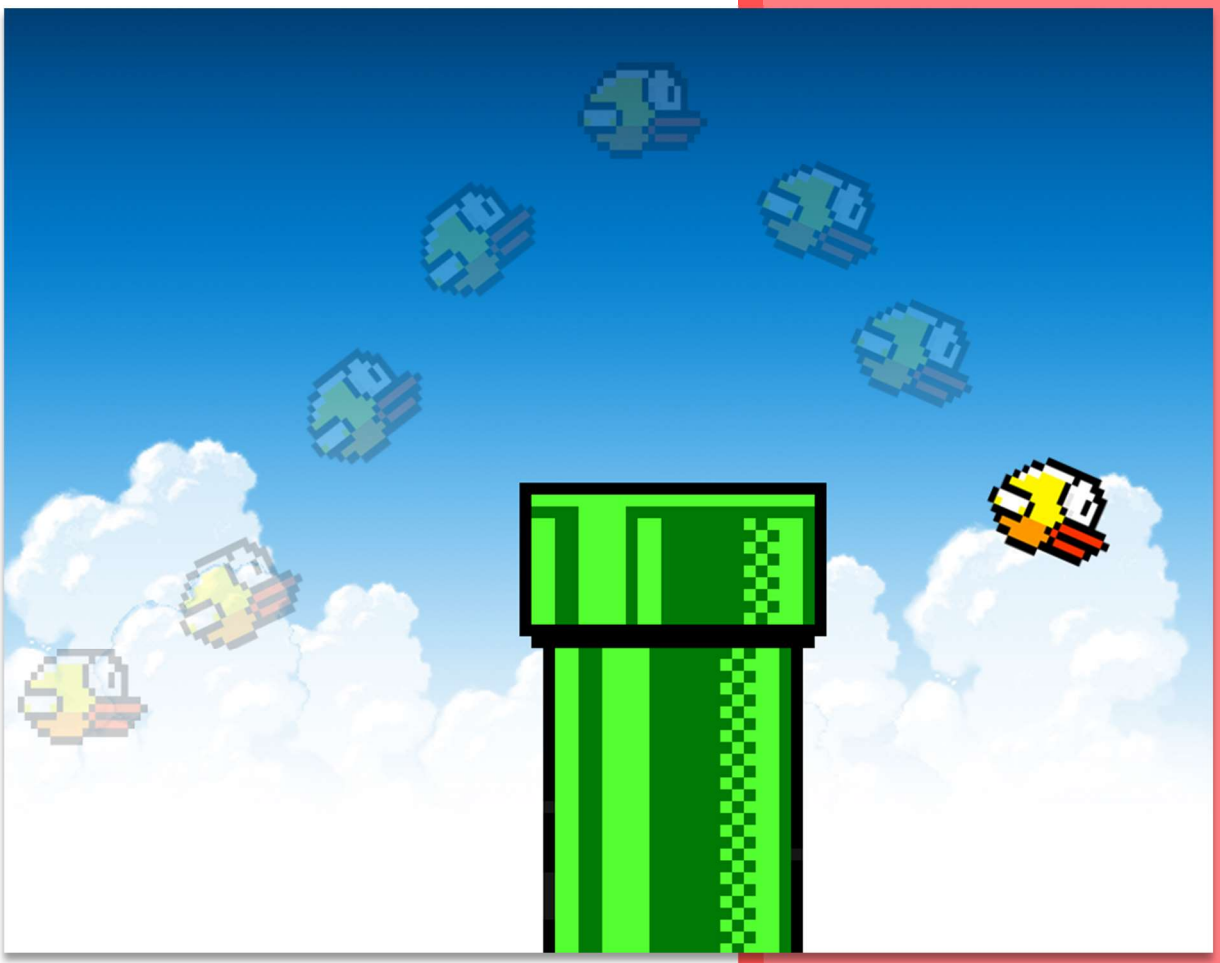
2020

FlappyBird – Documentation technique



Bovay Louis

DIVTEC

05/11/2020

Table des matières

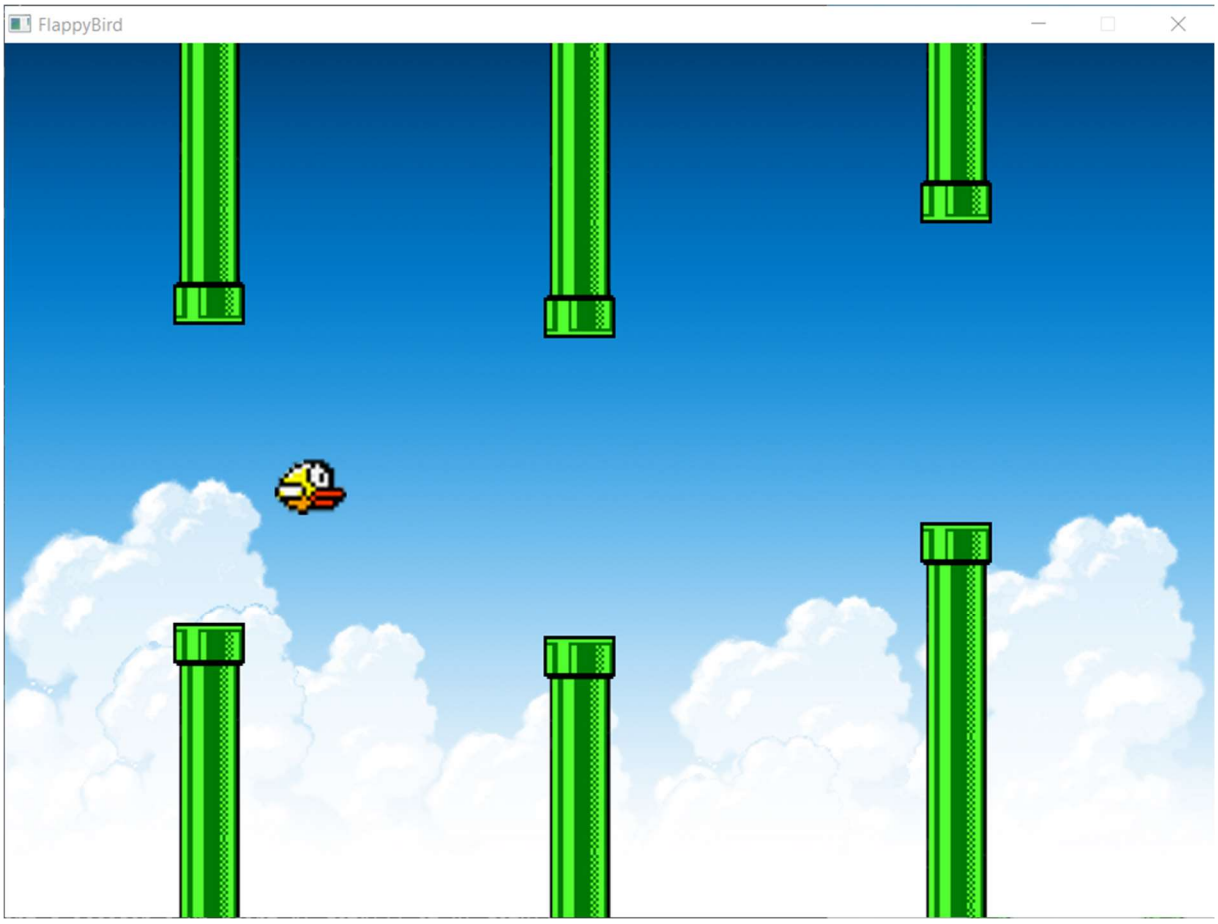
1. Description du projet

Le projet a pour but de réaliser le jeu FlappyBird en programmation orientée objet.

Le but du jeu est de faire battre des ailes un oiseau pour qu'il ne se cogne pas à des tuyaux placés en haut et en bas qui foncent sur lui.

Si l'oiseau touche un tuyau, le plafond ou le sol, il meurt.

A chaque fois qu'il esquive un couple de tuyaux, un point est marqué.



2. Convention de nommage

- Tous les noms de variable sont écrits en CamelCase → `myVariable`.
- Tous les noms de fonction doivent décrire l'action faite dans la fonction et sont écrits en CamelCase → `getRandomNumber()`.
- Tous les noms de classe sont écrits en CamelCase et commence par une majuscule → `SpaceBird.java`.
- L'entièreté du code, à part les commentaires, est en anglais.

3. Installations

- IntelliJ: https://www.jetbrains.com/fr-fr/idea/
- Le SDK JavaFX: https://gluonhq.com/products/javafx/
- Tutoriel d'installation de JavaFX par Jetbrain : https://www.jetbrains.com/help/idea/javafx.html#create-project

Créez un nouveau projet et sélectionnez « JavaFX » dans la liste de gauche.

Cliquez sur le menu déroulant « Project SDK » et ajoutez le SDK téléchargé via le bouton « Add JDK… »

Choisissez un nom de projet ainsi que ça location et voilà : le projet inclura les librairies javafx.

3.1 Configuration du compilateur

Si aucun configuration n'est présente ici :

Il faut en ajouter une dont voici les paramètres :

- Type : Application
- Sélectionnez le fichier `Main.Java` du projet pour « Working Directory »
- Cliquez sur **OK** et l'IDE pourra compiler.

3.2 Configurer un module

Un module est l'ensemble des fichier du projet, il est important d'en avoir un pour faire les liens entre les différents fichiers afin d'éviter des erreur d'entité introuvable ou des mauvaise aide à la complétion.

1. CTRL + ALT + SHIFT + S
2. Onglet module
3. Bouton « + »
4. « New Module »
5. Donnez un titre, choisir le chemin du projet et des fichiers du projet
6. Finish

4. Les classes principales :

Shape, qui hérite de Rectangle représente un rectangle physique en 2 dimensions qui peut se déplace en XY et donner les coordonnées de ses 4 coins grâce à sa variable Area qui permet de générer les coordonnées des quatre coins du rectangle.

Bird, qui hérite de Shape représente un oiseau qui peut voler, tomber, mourir et revivre.

Pipe, qui hérite aussi de Shape, représente un tuyau qui peut détecter si l'oiseau le touche.

PipeCouple représente un couple de tuyau qui peut bouger vers la gauche, revenir instantanément à droite, générer un espace fixe à un endroit aléatoire sur l'axe Y entre ses deux tuyaux et donner un points au joueur.

Score représente un score, il peut stocker les points, les fournir, les remettre à zéro ainsi que les afficher.

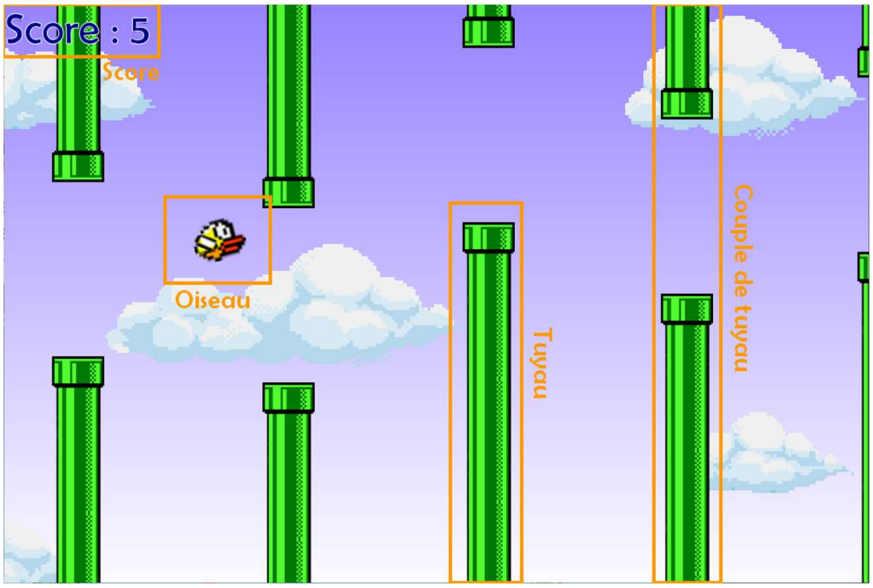
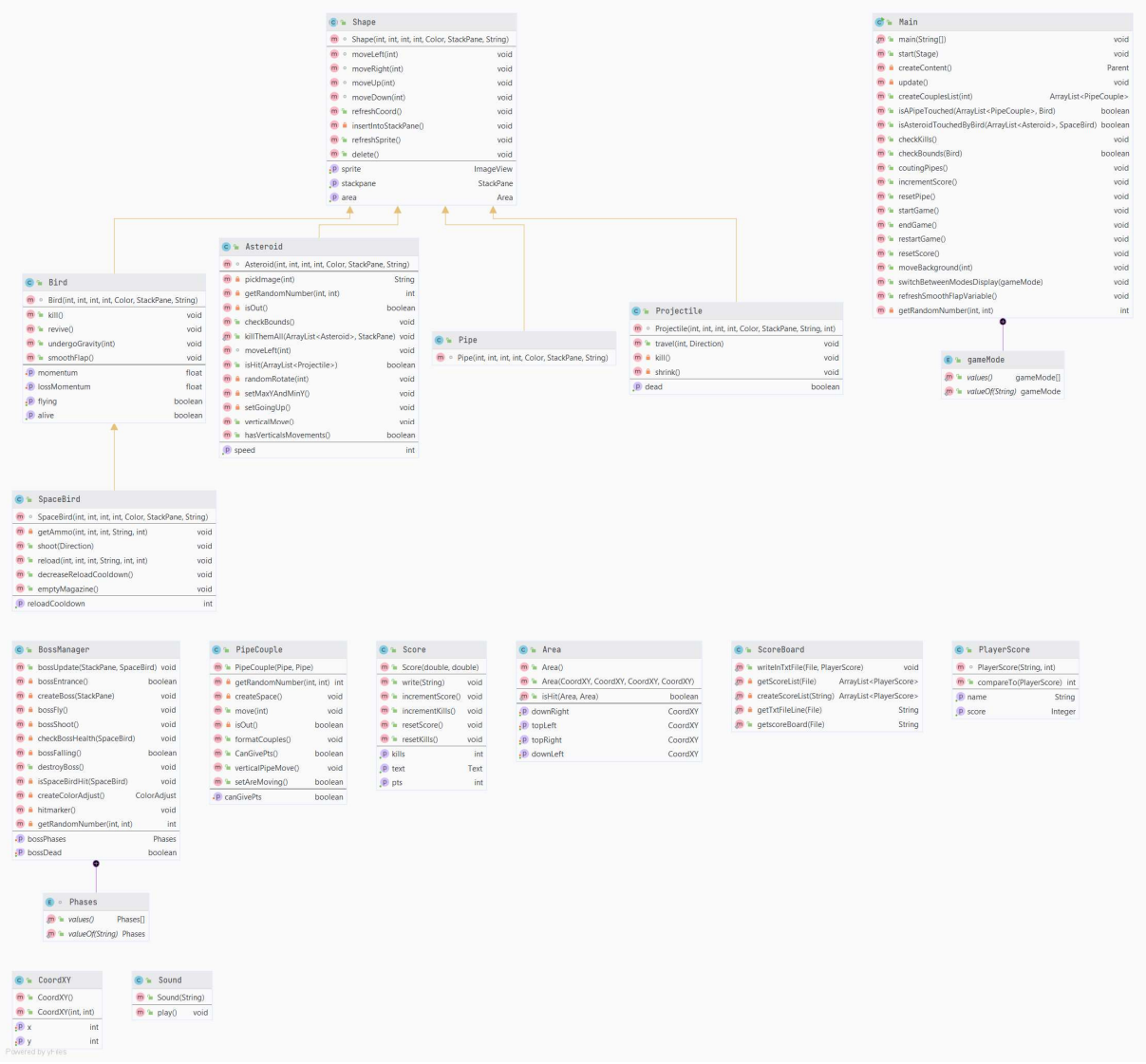


Figure 1 Description des éléments visuels

5. Diagramme de classe

Voilà le diagramme de toutes les classes présentes au sein du projet.

Une version taille réelle est disponible dans le dossier IMG du projet GitHub.



6. Explications supplémentaires

Cette partie est là pour expliquer ce qui ne pourrait pas être facile à comprendre ou pour faciliter l'apprentissage de classes et méthodes utilisées au sein du projet.

Si une fonction / variable est mentionnée ou alors qu'une partie de code est montrée, il sera précisé à chaque début de sujet dans quel `fichier` la trouver.

S'il y a plusieurs fichiers mentionnées, les variables/fonction seront précédées de leur classe.

6.1 Les « fausses scènes »

`Main.java`

En théorie, on pourrait dire que le jeu à trois scènes :

- Le menu principal
- Le jeu
- Fin de partie

En pratique, c'est toujours la même scène qui se joue mais qui change d'état.

Il y a pour ça deux booléen :

```
boolean isGameRunning = false;
boolean isGameStarted = false;
```

`IsGameStarted` sert juste une seule fois pour faire apparaître le menu principal, nous ne sommes pas sensé pouvoir retourner au menu principal, à moins de relancer l'application

`IsGameRunning` sert à savoir si le jeu est en cours ou non, si `isGameRunning` = `true` alors le jeu se joue, si `isGameRunning` = `false` l'écran de score s'affiche et propose de rejouer.

Tout ça sert savoir quand lancer une de ces trois méthodes :

```
public void startGame() {…}
public void restartGame() {…}
public void endGame() {…}
```

Ces trois méthodes servent comme leur nom l'indique à démarrer, redémarrer et finir la partie.

6.2 Les couples de tuyaux

`Pipe.java`
`PipeCouple.java`

Dans le jeu, il y a des couples de tuyaux « `PipeCouple` » qui contiennent deux tuyaux « `Pipe` » chacun et qui bougent de droite à gauche (`PipeCouple.move()`).

Les deux tuyaux présents dans un couple ont un espace fixe de 250, l'espace peut apparaître n'importe où sur leur axe Y (`PipeCouple.createSpace()`).

Arrivé à gauche, ils sont reformatés (`PipeCouple.formatCouples()`) et retrouvent leur état d'origine (à droite, avec un nouvelle espace entre eux et le droit de donner un point) :

```
public void formatCouples() {
    //tuyau du haut
    this.topPipe.setTranslateX(550);
    this.topPipe.setTranslateY(-350);
    //tuyau du bas
    this.bottomPipe.setTranslateX(550);
    this.bottomPipe.setTranslateY(350);
    //Ils peuvent à nouveau donner des points
    this.canGivePts = true;
    // Générer l'espace aléatoire entre les deux tuyaux
    createSpace();
}
```

À chaque fois qu'un couple passe la position Y de l'oiseau, leur état passe de « apte à donner un point » à « non apte à donner un point » (`canGivePts`).

6.3 Le background

`Main.java`

Le background du jeu est composé de deux images, la particularité de ces images c'est qu'elles se suivent parfaitement, il n'y a pas de « coupure » entre elles.

Pour l'animation, on les place côte à côte et on les fait se déplacer de droite à gauche.

Comme les tuyaux, lorsqu'une des deux images se retrouve trop à gauche, on la téléporte tout à droite, puis on la refait passer : le background infini est créé.

```
    if (background.getTranslateX() <= -1000) {
        background.setTranslateX(1000);
```

6.4 La classe Area

```
Area.java
Shape.java
```

La classe Area représente les 4 coins d'une forme de la classe `Shape` qui permettent de former un rectangle.

Un objet Area sera composé de 4 coordonnées XY de la classe CoordXY qui représenteront le coin haut-gauche, haut-droit, bas-gauche et bas-droit.

L'utilité première de cette classe est de détecter les collisions (`Area.isHit(Area area1, Area area2)`).

Si un des 4 coins du premier objet se trouve entre deux points horizontaux et deux points verticaux de l'autre objet, cela signifie que le premier a touché le deuxième.

Toutes les classes héritières de Shape pourront accéder à cette fonction.

Figure 2 Fonctionnement de base de Area

```
Area.isHit(bird.getArea(), couple.topPipe.getArea()) ||
Area.isHit(bird.getArea(), couple.bottomPipe.getArea())
```

6.5 Gestion du temps et des frames

`Main.java`

6.5.1 Création

Lors de la création de la scène (`createContent()`), on lui assigne une timeline.

Chaque seconde, elle fait quelque chose, ici `update()`), on règle son cycle comme `Animation.INDEFINITE` pour la faire tourner à vie, à moins de l'arrêter manuellement, on la lance et on règle sa vitesse à 60, comme ça on arrive à 60 `update()` par seconde, donc 60 images par secondes.

```
//Timer basé sur 1 seconde
Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(1),
event -> {
    update();
}));
//nombre de fois qu'elle se répète
timeline.setCycleCount(Animation.INDEFINITE);
//lancer la timeline
timeline.play();
// vitesse 60 = 60 images par seconde
timeline.setRate(60);
```

6.5.2 Gestion

Une variable est à disposition

```
//Indice de temps, 60 = 1 secondes
double timeSpend = 0;
```

Elle s'actualise à chaque `update()` et donc représente 1/60 de seconde, donc si elle vaut 60, 1 seconde se sera écoulée.

```
//Incrément du temps (60 = 1 secondes)
timeSpend += 1;
```

6.6 Les sauts fluides

```
Bird.java
Main.java
```

Trois paramètres entre en compte concernant les sauts fluides :

- La gravité (`BIRD_GARVITY`)
- La vitesse de pointe (`BIRD_MOMENTUM`)
- La perte de vitesse (`BIRD_LOSSMOMENTUM`)

La vitesse de pointe doit toujours être de base supérieure à la gravité.

Au fil du temps, la perte de vitesse s'accumule jusqu'à ce que la gravité reprenne le dessus.

```
public void smoothFlap() {
    if (momentum > 0) {
        this.moveUp((int) momentum);
        momentum -= 0.5;
        birdSprite.setRotate(-20);
    } else {
        flying = false;
    }
}
```

`birdSprite.setRotate(-20)` sert à faire se pencher l'oiseau vers l'arrière pour donner l'impression qu'il tire vers le haut pour voler.

Il faut donc faire variés ces trois paramètre pour obtenir un saut fluides, pas trop arqué et qui ne donne pas l'impression d'être sur la lune.

Paramètres de base :

```
BIRD_GRAVITY = 8
BIRD_MOMENTUM = 19.5f
BIRD_LOSSMOMENTUM = -0.5f
```

6.7 Les formes et leur sprite

Shape.java

Pour afficher une image (un sprite) on crée une ImageView, dans laquelle on met une image.

```
new ImageView(new Image("Sprites/cloudbg1.png"));
```

Pour faire bouger le sprite avec la forme à qui il appartient, on fait en sorte qu'à chaque fois que ladite forme bouge, le centre du sprite suit le centre de la forme.

Chaque héritiers de la Shape actualise leur sprite à chaque mouvement, permettant d'avoir toujours leur forme, ainsi que leur sprite au même endroit.

```
/**
 * Rafraichit la position du sprite
 */
public void refreshSprite(){
    this.getSprite().setTranslateX(this.getTranslateX());
    this.getSprite().setTranslateY(this.getTranslateY());
}
```

Et lorsque qu'une Shape se déplace :

```
/**
 * Déplace la Shape vers le haut
 *
 * @param speed vitesse de déplacement
 */
void moveUp(int speed) {
    setTranslateY(getTranslateY() - speed);
    refreshSprite();
}
```

6.8 La detection des touches

`Main.java`

La détection des touches se fait de manière très simple :

```
scene.setOnKeyPressed(event -> {
    KeyCode keyCode = event.getCode();
    if (keyCode.equals(KeyCode.SPACE)) {
        //Faire quelque chose si SPACE est appuyé
    }
});
```

Une simple fonction qui écoute les touches du clavier et qui permet de détecter lorsqu'une touche est appuyée.

La fonction doit être placée dans le `start`, pour pouvoir accéder à la `scene`

Si l'on souhaite détecter lorsque la touche est relâchée, on remplace `SetOnKeyPressed` par `SetOnKeyRelease`.

Pour changer de touche, on modifie le `KeyCode`.

Pour n'effectuer qu'une seul fois une action, j'ai souvent mis en place des booléen qui dès que l'action est faite une fois, ne la laisse plus se reproduire tant que la touche n'a pas été relâchée.

6.9 Création d'un objet héritier de Shape

Voilà le constructeur de Shape :

```
Shape(int x, int y, int w, int h, Color color, StackPane
stackpane,String spritePath){}
```

On y fournit le sprite ainsi que la StackPane, ce qui permet de directement intégrer l'objet ainsi que le sprite dans la StackPane.

7. Tableau des scores et pseudo

```
ScoreBoard.java
PlayerScore.java
```

En fin de partie, le score et le nom du joueur sont écrit dans un fichier texte sous la forme suivant :

```
NomDuJoueur=Score;
```

Pour ce faire, il faut ouvrir le fichier texte des scores, écrire à la suite une chaine de caractères puis fermer le fichier.

```
/**
 * Écrit dans un fichier score
 * @param file fichier à créer si inexistant et dans lequel écrire
 * @param playerScore Score du joueur a écrire dans le fichier texte
 */
public static void writeInTxtFile(File file, PlayerScore
playerScore) {
    try {
        FileWriter writer = new FileWriter(file, true);
        PrintWriter printWriter = new PrintWriter(writer);
        printWriter.print(playerScore.getName() + "=" +
playerScore.getScore() + ";");
        printWriter.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
```

Afin de générer le tableau des scores, il faut ouvrir le fichier, récupérer l'entièreté du texte, et le décomposé pour ensuite l'enregistrer dans une liste de score de joueur

```
if (line != null) {
    //parcourir la ligne de texte
    for (int i = 0; i < line.length(); i++) {
        //Si le nom n'a pas été écrit
        if(!nameIsWrited){
            //Tant qu'un signe égale n'est pas atteint
            if(line.charAt(i) != '='){
                currentName.append(line.charAt(i));
            }
            //si le caractère est un signe égale
            else{
                nameIsWrited = true;
            }
        }
        //Le nom est écrit, donc écriture du score
        else{
            //tant qu'une virgule n'est pas atteinte
            if(line.charAt(i) != ';'){
                currentScore.append(line.charAt(i));
            }
            //le nom et le score ont été lu, donc enregistrement
            else{
                nameIsWrited = false;
                listOfScores.add(new
PlayerScore(currentName.toString(),
Integer.parseInt(currentScore.toString())));
                //On vide les variables
                currentScore.setLength(0);
                currentName.setLength(0);
            }
        }
    }
}
```

Une liste de score de joueur est un objet (PlayerScore) qui contient un paramètre Name pour le nom du joueur et le paramètre Score pour le score du joueur.

Après avoir récupérer la liste des score de joueur, on peu la trier, et l'afficher.

```
/**
 * Créer une chaîne de charactère formatée pour afficher un tableau
de 5 scores avec le nom des joueurs
 *
 * @param file fichier à ouvrir
 * @return une chaîne de caractère formatée
 */
public static String getscoreBoard(File file) {
    StringBuilder scoreBoard = new StringBuilder();
    ArrayList<PlayerScore> listScores;
    listScores = getScoreList(file);

    for (int i = 0; i < 5; i++) {
        //Retour de chariot
        if (i != 0) {
            scoreBoard.append("\n");
        }
        scoreBoard.append("   ").append(i + 1).append(".\t  ");
        //Insérer les scores
        if (i < listScores.size()) {

scoreBoard.append(listScores.get(i).getScore()).append("\t").append(
listScores.get(i).getName());
        }
    }
    return scoreBoard.toString();
}
```

8. Explication supplémentaires 3ème mode de jeu

« FLAPPY BIRD AGAINST THE SPACE VILLAINS PART II 4K » est le troisième mode de jeu disponible en appuyant sur G depuis l'écran d'accueil ou des scores, les règles changent totalement : l'oiseau n'est plus un oiseau mais un oiseau de l'espace qui peut tirer des projectiles contre des astéroïdes pour ensuite affronter un boss.

8.1 SpaceBird

`SpaceBird.java`

L'oiseau de l'espace (`SpaceBird`) hérite de la classe Bird.

Il peut périodiquement tirer des projectiles présent dans son chargeur (`spaceBird.magazine`) sur une courte distance afin de détruire des astéroïdes.

Le saut est changé pour donner une impression de flottement dû à la faible gravité.

8.2 Le tir

`SpaceBird.java`
`Projectile.java`

En appuyant sur une touche, un projectile est placé dans le chargeur de l'oiseau, à chaque fois qu'un projectile est présent dans le chargeur, il est automatiquement tiré (`SpaceBird.shoot()`) droit devant l'oiseau. Il continue d'avancer (`Projectile.travel()`) sur la droite le temps de sa durée de vie (`Projectile.lifeTime`), quand sa durée de vie tombe à zéro, il entame une phase de rapetassage (`Projectile.shrink()`) et rapetisse donc jusqu'à disparaître.

S'il le projectile rencontre un astéroïde, il est détruit.

8.3 Asteroide

`Asteroid.java`
`Main.java`

La particularité de l'astéroïde c'est qu'il n'a pas qu'un seul sprite : il peut en avoir 9.

Lors de sa création, son sprite de base est directement remplacé par un sprite tiré aléatoirement.

```
//Attribuer l'un des 9 sprites aléatoire
setSprite(pickImage(getRandomNumber(1, 10)));
```

L'astéroïde à plusieurs mode de déplacement : frontale et vertical (`hasVerticalsMovements`).

Dans les deux cas il se déplacera de droite à gauche avant de disparaître en sortant de l'écran. La différence est qu'il va plus vite en frontal, et plus lentement en vertical mais esquissera quelques courbes de haut en bas tout en se déplaçant.

Tous les astéroïdes présent en jeu se trouvent dans une liste.

Si un astéroïde est touché par un projectile, il est détruit et un point est comptabilisé.

8.4 Le boss

```
Main.java
BossManager.java
```

Le boss est un autre oiseau de l'espace, bien plus imposant et disposant de plus de point de vie.

Le boss est entièrement géré dans la classe BossManager, où ses différentes phases sont jouées :

- CREATION : le boss est créé et ajouté à la StackPane.
- ENTRANCE : le boss entre en scène jusqu'à atteindre la coordonnée voulue
- FIGHT : le boss se met à voler et tirer des projectiles vers notre oiseau, c'est aussi là que sa vie (`checkBossHealth()`) ainsi que celle de l'oiseau (`isSpaceBirdHit()`) sont vérifiées
- END : Si l'oiseau n'est toujours pas mort et que les point de vie du boss tombent à zéro, le boss entame une animation de mort et la partie se termine.

Si l'oiseau meurt, la partie prendra fin qu'importe la phases où le boss se trouvent.

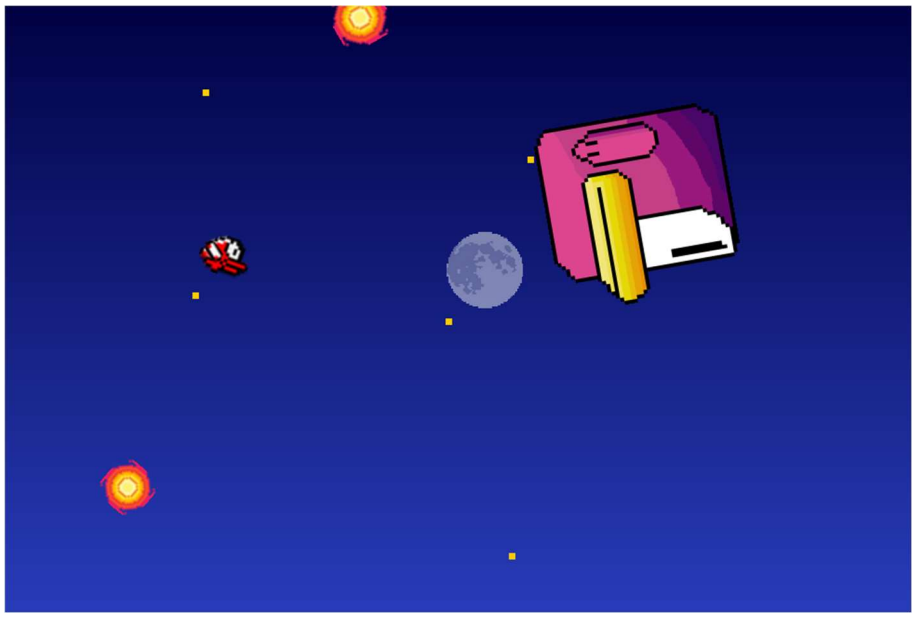


Figure 3 combat contre le boss

9. Création d'un exécutable

9.1 Paramétrer l'exécutable

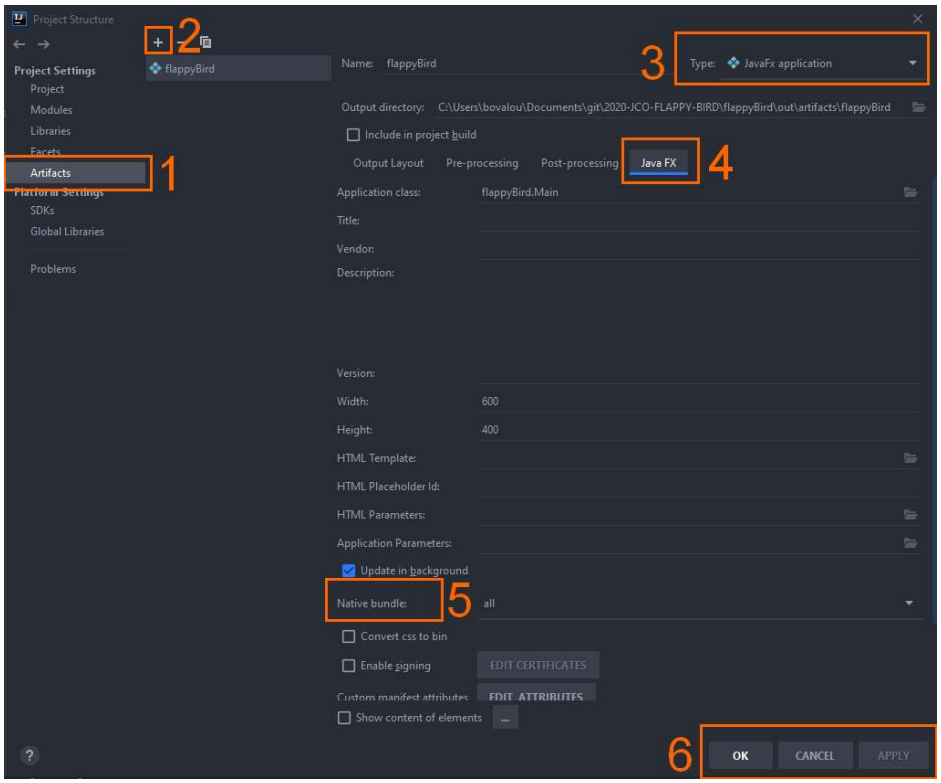
Créer un exécutable sert notamment à avoir une version portable du jeu mais aussi de les proposer plus facilement en release sur GitHub.

CTRL + ALT + MAJ + S ouvrira la structure du projet.

Dans l'onglet à gauche « Artifacts », appuyer sur le bouton « + » pour y sélectionner « Javafx Application from module flappybird».

Dans l'onglet « java FX » de l'artifacts , sélectionner la classe principale (ici flappyBird.Main.Java), puis choisir l'option « all » de « Native Bundle ».

Appliquer et valider.



9.2 Lancer la création d'un exécutable

Dans l'onglet Build > Build Artifacts, sélectionner "Build" pour créer un exécutable.

Par défaut, l'exécutable se trouve dans le dossier « out » du projet.

10. Sources

- Gluon : https://gluonhq.com/products/javafx/
- Classe Rectangle : https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Rectangle.html
- Détection des touches : https://stackoverflow.com/questions/37472273/detect-single-key-press-in-javafx
- Animation timer : https://stackoverflow.com/questions/50337303/how-do-i-change-the-speed-of-an-animationtimer-in-javafx
- Font javafx : https://www.tutorialspoint.com/javafx/javafx_text.htm#:~:text=You%20can%20change%20the%20font,scene.
- Rotation d'un imageView : https://stackoverflow.com/questions/34166627/javafx-rotate-imageview
- Inspiration pour sauts fluides : https://stackoverflow.com/questions/41314228/javafx-creating-smooth-animation-without-creating-a-new-thread
- Le jeu en ligne : http://flappybird.io/
- Précédents exercices réalisés en module