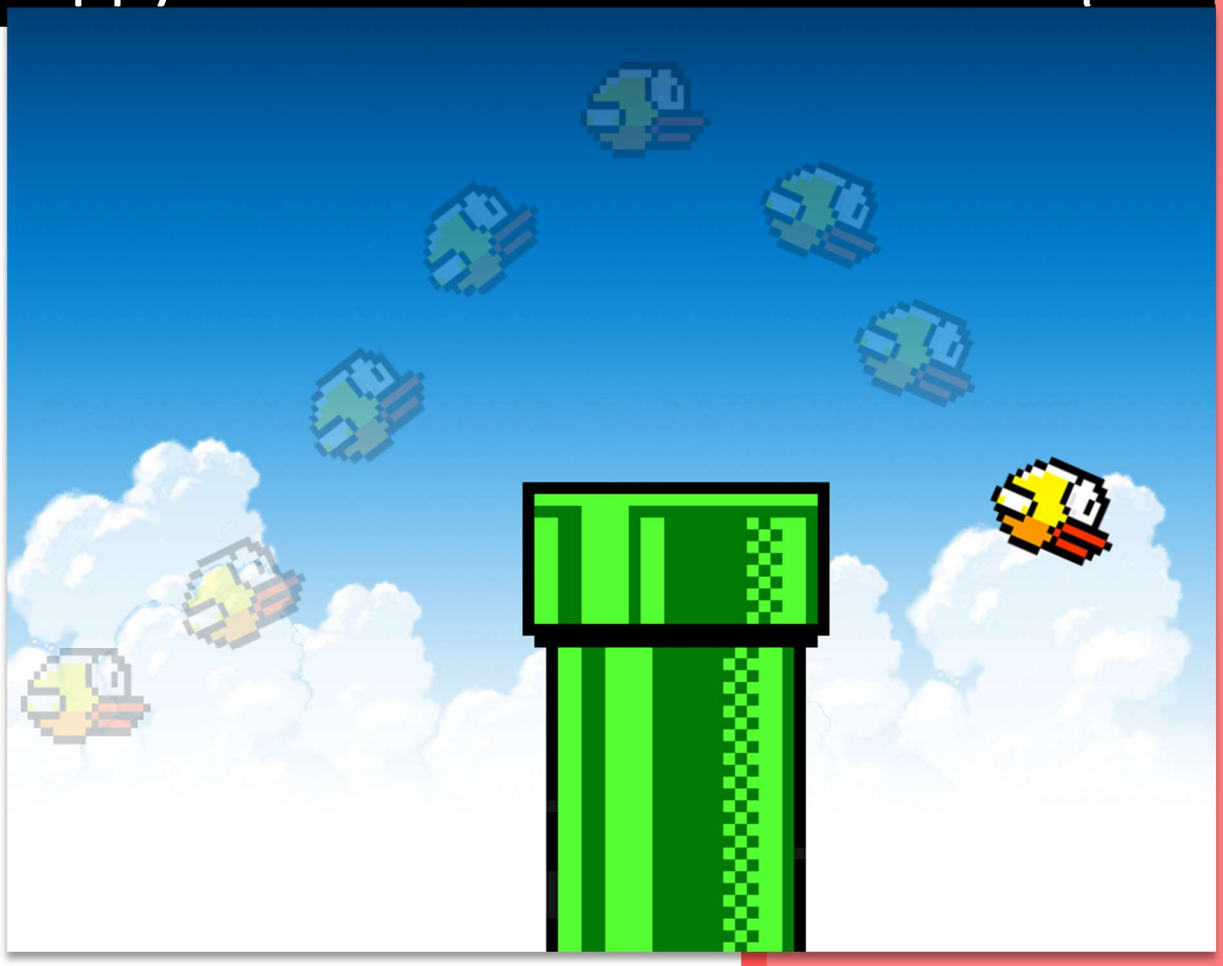
2020

FlappyBird – Documentation technique



Bovay Louis

DIVTEC

05/11/2020

Table des matières

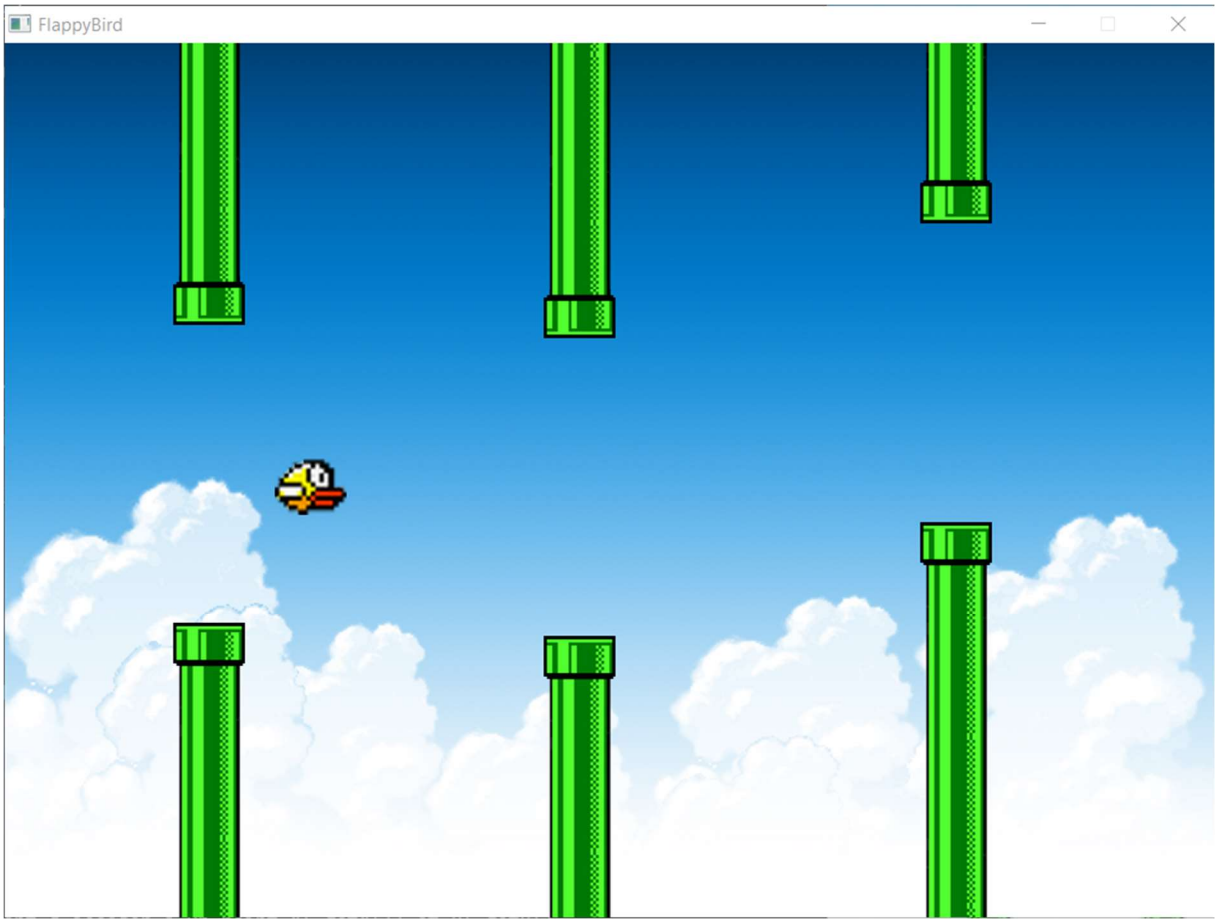
1. Description du projet

Le projet à 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.

S'il 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
- Tous les noms de fonction doivent décrire l'action faite dans la fonction et sont écrits en CamelCase.
- Tous les noms de classe sont écrits en CamelCase et commence par une majuscule

3. Explications supplémentaires

Cette partie est là pour expliquer ce qui ne pourrait pas être facile à comprendre ou pour facilité l'apprentissage de 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.

3.1 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 as 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 fonctions :

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

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

3.2 CSS dans le projet

```
Main.java
style.css
```

Le projet contient du CSS (le background est fait en CSS), et pour en utiliser il faut donner un ID à l'élément que l'ont souhait décorer, pour ensuite assigner du CSS à l'ID.

```
StackPane root = new StackPane();
//Attribution de l'ID
root.setId("pane");

//Récupération de la feuille de style css
scene.getStylesheets().add("css/style.css");
```

CSS :

```
#pane {
    -fx-padding: 10px;
    -fx-background-image: url("../Sprites/background.png");
}
```

3.3 Gestion des 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.

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

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

```java
public void formatCouples() {
    //tuyau du haut
    this.pipe1.setTranslateX(550);
    this.pipe1.setTranslateY(-350);
    //tuyau du bas
    this.pipe2.setTranslateX(550);
    this.pipe2.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`).

3.4 Gestion du temps et des frames

`Main.java`

3.4.1 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.

```java
//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);
```

3.4.2 Gestion

Deux variables sont à disposition au besoin

```
//Indice de temps, 1 = 60 frame
float t = 0;
//Indice de frame, 60 = 1 seconde
int frame = 0;
```

Elles s'actualisent à chaque update() et donc représentent chacune 1 seconde sous différentes formes, si par exemple nous souhaitons faire quelque chose toutes les 10 frames, ou toutes les 5 secondes.

```
//Incrémentation des indice de frame et de temps
//Quand t atteint 1, une seconde sera écoulée
//Quand frame atteint 60, une seconde sera écoulée
t += 0.016;
frame++;
```

4. 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.
- Précédents exercices réalisés en module