

Travail Pratique 2 - Flappy Ghost

IFT1025 - Programmation 2

Concepts appliqués – Programmation orientée objet, interfaces graphiques, animation, développement d'application complète

Contexte

Le second TP consiste à programmer un jeu en interface graphique avec la librairie *JavaFX*.

Description du jeu

Le joueur incarne un fantôme qui se promène dans un niveau rempli d'obstacles. Il n'y a pas d'objectif autre que d'avancer le plus longtemps possible dans le niveau sans toucher d'obstacle.

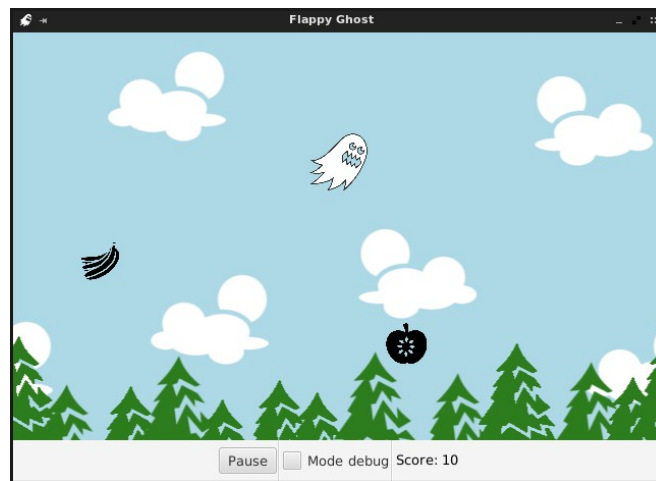


Figure 1: Déroulement du jeu

Le déplacement horizontal du joueur se fait automatiquement vers la droite. Le seul déplacement possible est de faire "sauter" le fantôme en appuyant sur la barre espace.

Pour rendre le jeu de plus en plus difficile à mesure que le score monte, pour chaque deux obstacles dépassés, la vitesse du fantôme accélère et la gravité augmente.

Obstacles

Les obstacles sont représentés par une des 27 images de l'archive `obstacles.zip`, qui sont des icônes de fruits et de légumes.

Un nouvel obstacle est ajouté dans le niveau à toutes les 3 secondes. Lorsqu'un obstacle est ajouté, il doit avoir une position en `x` tout juste à l'extérieur de la fenêtre du jeu.

Chaque fois que le joueur dépasse horizontalement un obstacle (autrement dit, lorsque son extrémité gauche dépasse l'extrémité droite de l'obstacle), son score augmente de 5 points.

Si le fantôme entre en collision avec un des obstacles, la partie recommence à zéro (score=0 et le fantôme recommence depuis le début du jeu).

Pour simplifier le modèle, les obstacles et le fantôme sont affichés avec des images mais sont en réalité modélisés par des cercles de certains rayons :

- Le fantôme a un rayon fixe de 30 pixels
- Les obstacles ont des rayons générés aléatoirement entre 10 et 45 pixels

Il y a trois types d'obstacles. Les différents types se distinguent par leur mouvement. Chaque obstacle ajouté est aléatoirement de l'un de ces trois types.

Obstacle simple

Les obstacles simples ont une position (`x`, `y`) définie lors de leur création et ne se déplacent pas.

Obstacle sinus

Les obstacles sinus ne se déplacent pas horizontalement mais suivent verticalement un mouvement sinusoïdal d'une amplitude de 50 pixels centré sur leur position `y` initiale :

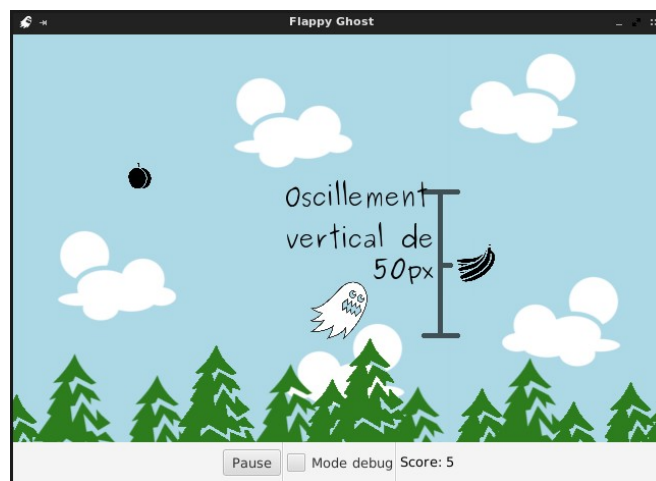


Figure 2: Mouvement sinusoïdal

Choisissez vous-mêmes une vitesse appropriée pour le mouvement vertical. Utilisez la fonction `Math.sin(.)`

Obstacle quantique

Les obstacles quantiques se téléportent d'une distance aléatoire comprise entre -30 et 30 pixels en x et en y, périodiquement à chaque 0.2 seconde. Notez qu'il est possible qu'un obstacle quantique se téléporte un peu vers l'arrière après que le fantôme l'ait dépassé, mais on ne peut pas "dépasser" une deuxième fois le même obstacle pour faire monter le score.

Physique du fantôme

Le fantôme se déplace vers la droite à une vitesse constante initiale de 120 pixels par seconde. À chaque deux obstacles dépassés, la vitesse horizontale du joueur augmente de 15 pixels par seconde.

La vitesse verticale du joueur est affectée par les sauts (avec la barre espace) et la gravité :

- La gravité (une accélération en y seulement) est initialement de 500 pixels par seconde² vers le bas
- À chaque deux obstacles rencontrés, la gravité doit augmenter de 15 pixels par seconde² vers le bas
- Un saut change instantanément la vitesse en y du fantôme à 300 pixels par seconde vers le haut

La vitesse en y ne doit jamais dépasser 300 pixels par sec. vers le haut ou vers le bas. Si jamais la vitesse dépasse les 300, on la force à rester à une magnitude de 300 (en considérant sa direction haut/bas).

Le fantôme ne peut pas sortir des bords de l'écran : lorsque le fantôme touche le haut ou le bas du niveau, il rebondit dans l'autre direction. Notez que contrairement au fantôme, les obstacles pourraient se retrouver hors de l'écran selon leurs déplacements.

Arrière-plan

La fenêtre de jeu est tout le temps centrée horizontalement sur le fantôme, l'arrière-plan et les obstacles défilent. L'image `bg.png` doit être affichée de façon cyclique et doit défiler selon la position à laquelle le fantôme est rendu dans le niveau.

Code & Design Orienté Objet

Ce sera à vous de choisir le découpage en classes optimal pour votre programme. Faites bon usage de l'orienté objet et de l'héritage lorsque nécessaire.

Vous **devez** utiliser une architecture du type *Modèle-Vue-Contrôleur* tel que vu en classe et vous serez évalués là-dessus.

Réfléchissez à ce qui constitue votre Modèle pour ce jeu, à comment définir une vue correctement et à comment faire usage du Contrôleur pour isoler complètement les classes de la Vue des classes du Modèle.

La classe principale du programme doit se nommer `FlappyGhost` et être dans le package par défaut (aka, pas de ligne `package ...` au début du fichier).

Design de l'interface

L'interface du jeu doit comporter les éléments illustrés dans la *Figure 2*.

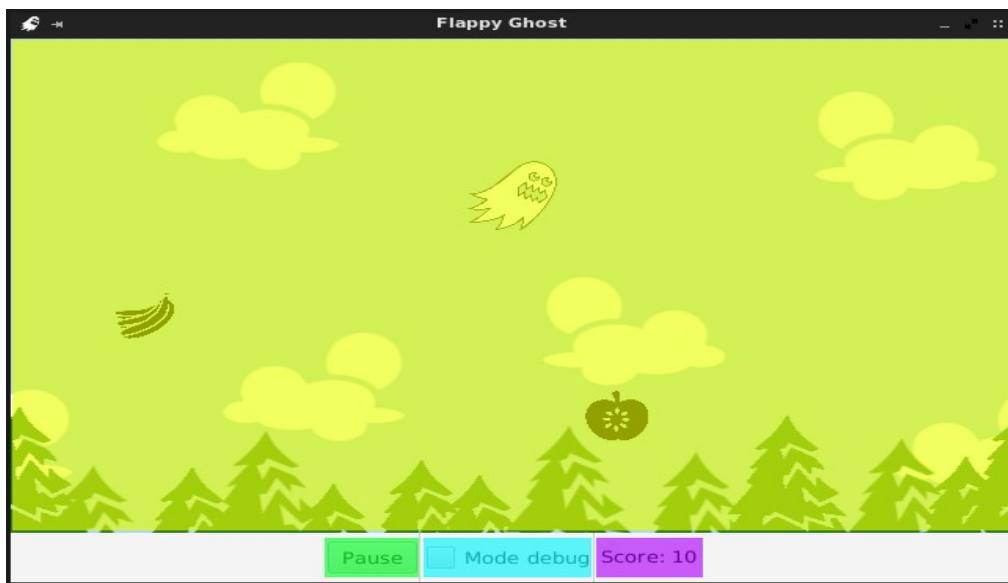


Figure 3: Interface graphique du jeu

Légende :

- **Jaune** : Le canvas dans lequel le jeu est affiché.
- **Vert** : Le bouton pause. Lorsqu'on clique dessus, le jeu doit se mettre en pause et le texte du bouton doit afficher "Resume". On peut cliquer sur ce bouton pour reprendre le jeu (le texte affiché redevient alors "Pause").
- **Bleu** : Une case à cocher/décocher pour activer/désactiver le mode debug (décrit plus loin).
- **Mauve** : Un texte qui permet d'afficher le score actuel.

Fenêtre de l'application

La fenêtre :

- Doit avoir une taille de 640 px de largeur et de 440 px de hauteur
- Le canvas (la fenêtre de jeu) dans l'application doit avoir une taille de 640 px par 400 px, les 40 px restants servent au menu de boutons
- Ne doit pas être resizable : la taille de la fenêtre est fixée au début et ne peut pas être modifiée par les utilisateurs
- Doit porter le titre "Flappy Ghost"
- Doit avoir l'image du fantôme en icône dans la barre de tâches

Finalement, le jeu doit se fermer lorsqu'on appuie sur la touche Escape Utilisez `Platform.exit()` pour mettre fin au programme.

Mode Debug

Pour faciliter les tests, vous devez inclure un mode debug à votre jeu, activable/désactivable via la case à cocher prévue à cet effet.

Le mode debug permet de tester les collisions et l'avancement dans le jeu : lorsqu'il est activé, il est impossible de perdre la partie. Les objets sont alors affichés non pas avec leurs images habituelles, mais avec une représentation qui correspond à leur modélisation pour les collisions :

- Le fantôme est affiché avec un cercle noir
- Les obstacles sont affichés avec des cercles rouges ou jaunes, selon s'ils sont actuellement en collision avec le fantôme ou non (jaune=pas de collision – rouge=collision)

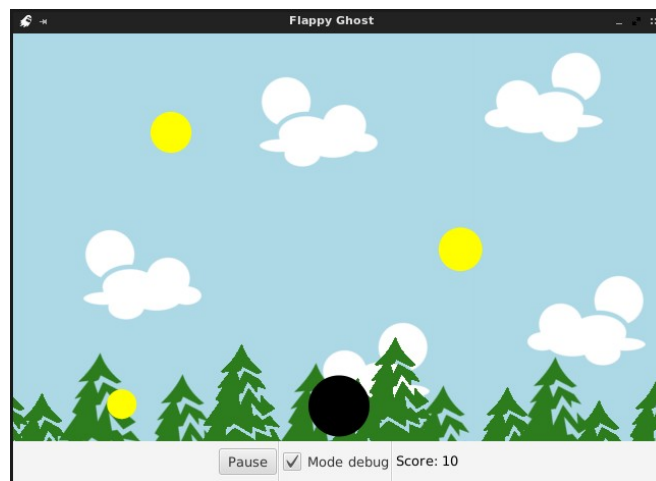


Figure 4: Mode debug

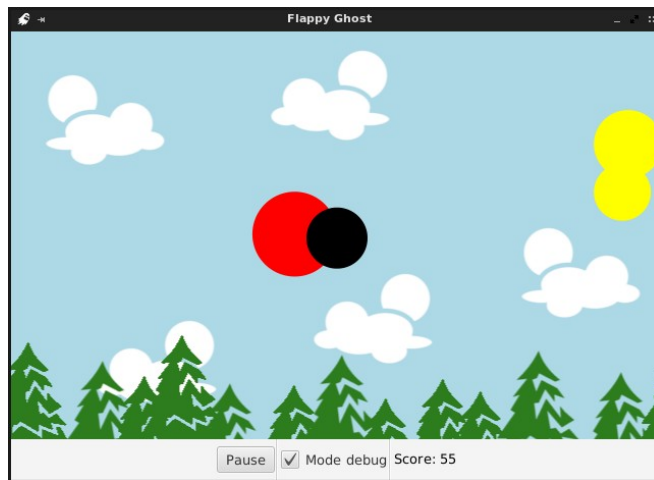


Figure 5: Le mode debug doit afficher les obstacles en collision avec le fantôme en rouge

Mise en garde focus de la souris

De base, si un bouton se trouve dans la fenêtre, ce bouton va prendre le focus. Appuyer sur la barre espace aura alors pour effet de déclencher l'action de ce bouton, et la scène elle-même ne capturera pas l'événement du clavier. Cela posera problème pour détecter les sauts.

Vous pouvez corriger ce problème en ajoutant le code suivant dans la fonction `start(:)`

```
/* Après l'exécution de la fonction, le
   focus va automatiquement au canvas */
Platform.runLater(() -> {
    canvas.requestFocus();
});

/* Lorsqu'on clique ailleurs sur la scène,
   le focus retourne sur le canvas */
scene.setOnMouseClicked((event) -> {
    canvas.requestFocus();
});
```

Barème

- **45%** ~ Exécution (tests du programme)
- **35%** ~ Découpage en classes et utilisation judicieuse de l'héritage
- **10%** ~ Découpage du programme en suivant l'architecture MVC
- **10%** ~ Qualité du code
 - Code bien commenté (*JavaDoc*), bien indenté, bon découpage en fonctions, encapsulation, noms de variables bien choisis, performance du code raisonnable, camelCase, pas trop compact, pas trop espacé. . . bref, du code clair et lisible en suivant les principes que vous connaissez
 - Limitez vos lignes de code à 120 caractères de large

Indications supplémentaires

- La date de remise est le 8 août 2022 à 23h59. Il y a une pénalité de **25%** pour chaque jour de retard.
- Vous **devez** faire le travail par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code, dans le fichier principal (`FlappyGhost.java`).
- Une seule personne par équipe doit remettre le travail sur StudiUM.
- Remettez tous les fichiers dans une archive **.zip**