TP3 – Flappy Butterfly

Programmation 2 – Nicolas Hurtubise

Contexte

Le second TP consiste à programmer un petit jeu inspiré de *Flappy Bird*.

Description du jeu

On incarne un papillon 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 perdre tous ses coeurs.

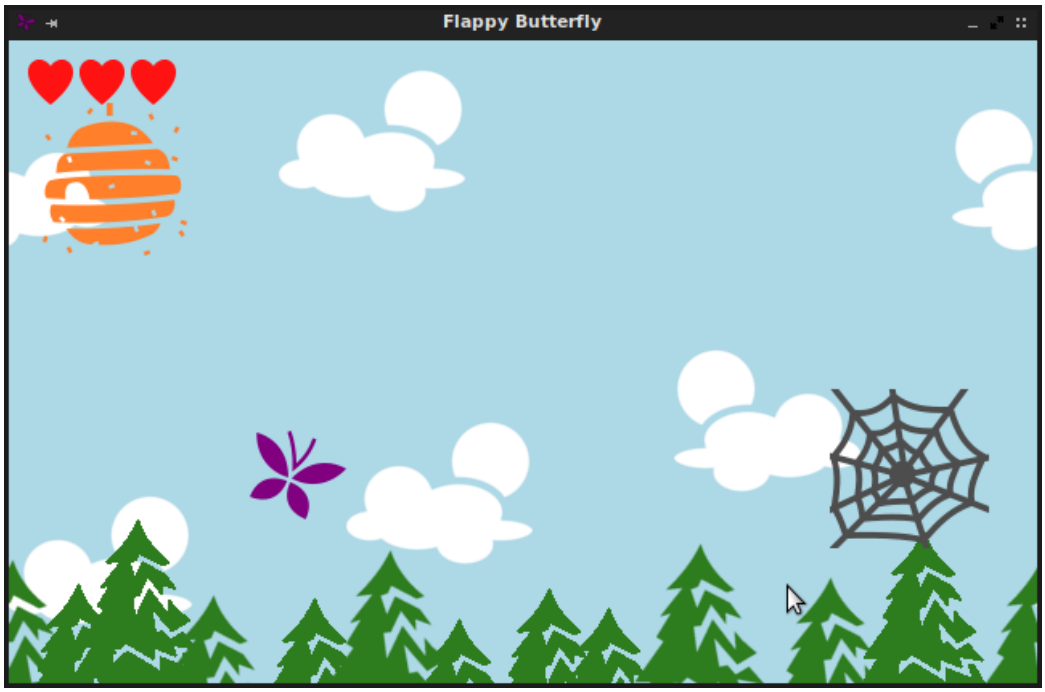


Figure 1: Déroulement du jeu

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

Obstacles

Il y a quelques types d'obstacles, certains bons pour le papillon, certains moins bons.

Un nouvel obstacle au hasard est ajouté automatiquement à chaque fois que le papillon a parcouru 480px (3/4 de l'écran). Lorsqu'un obstacle est ajouté, il doit avoir une position en `x` tout juste à l'extérieur de la fenêtre visible dans le jeu, pour donner l'impression qu'il a toujours été là à attendre.

Toile d'araignée (méchant)

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

Si on la touche, on perd un coeur.

Ruche d'abeilles (méchant)

La ruche se déplace en oscillant de haut en bas.

Si on la touche, on perd un coeur.

Filet à insectes (méchant)

Le filet est contrôlé par des forces quantiques subatomiques et peut se téléporter en une position Y aléatoire (dans les limites de l'écran) à toutes les 750ms.

Si on le touche, on perd un coeur.

Panneau d'accélération (bon)

Lorsque le papillon touche un panneau d'accélération, le panneau disparaît et le papillon accélère :

- La vitesse augmente de +50px/s
- L'accélération augmente de $+30\text{px}/s^2$

Bonus de vie (bon)

Lorsqu'on le touche, le bonus de vie redonne un coeur de plus au papillon, puis disparaît.

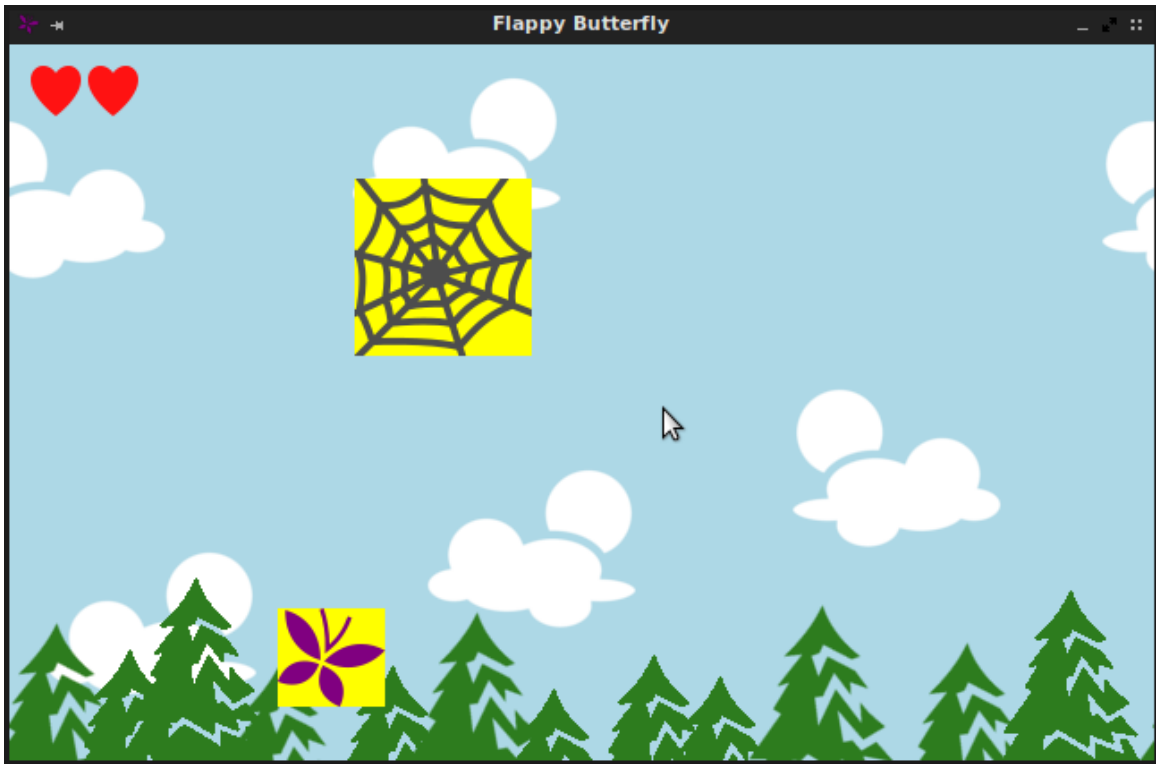
À noter : le papillon ne peut pas avoir plus que 3 coeurs à la fois.

Modélisation

Pour simplifier la détection de collisions, les obstacles et le papillon sont affichés avec des images mais sont en réalité modélisés par des rectangles qui ont la taille de l'image affichée.

	Largeur (pixels)	**Hauteur (pixels)**
Papillon	60	55
Toile d'araignée	99	99
Ruche	90	95
Filet	78	88
Panneau d'accélération	120	109
Bonus de vie	50	50

À des fins de débogage, on doit afficher un rectangle *jaune* (rouge=255, vert=255, bleu=0) derrière les objets lorsque la touche D est enfoncée.



Physique du papillon

Le papillon se déplace vers la droite à une vitesse constante initiale de `120` px/s.

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` px/s^2 vers le bas
- Un saut (barre d'espace) change la vitesse en `x` du papillon à `300` px/s vers le haut

La vitesse en `y` ne doit jamais dépasser `300` px/s 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).

Important : le papillon ne peut pas sortir des bords de l'écran : lorsque le papillon touche le haut ou le bas du niveau, il *rebondit* dans l'autre direction. Notez que contrairement au papillon, certains 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 papillon, 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 papillon est rendu dans le niveau.

Fin du jeu

Si le papillon n'a plus de coeurs, on affiche le message `Game Over` (c'est la même image que dans le TP1) pendant 3 secondes, puis on redémarre une nouvelle partie.

Save state/Load state (à faire en dernier, quand tout marche)

En bon joueur de jeux émulés, je me suis habitué à abuser des save state/load state :v)

Implantez ça dans votre jeu : quand on appuie sur la touche `S`, ça doit sauvegarder l'état du jeu dans un fichier nommé `flappybutterfly.sav`. Quand on appuie sur la touche `L`, ça doit charger l'état du jeu contenu dans ce fichier.

Notez donc que les Save-states persistent d'une exécution du programme à l'autre.

Gardez cette dernière fonctionnalité pour la fin, quand vous savez que tout fonctionne. Ça ne vaudra pas plus que 5% de la note finale et c'est difficile de faire des ajustements dans les classes une fois que cette fonctionnalité est codée. Préférez faire marcher les autres fonctionnalités d'abord.

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.

Code fourni

Aucun `:v)`

Référez-vous au TP1 et aux exemples vu en classe pour vous inspirer.

Points Bonus (jusqu'à +5%)

Autres obstacles (2%)

Ajoutez quelques types d'obstacles de votre choix. Soyez créatifs, rendez le jeu amusant, amusez-vous!

Le pourcent bonus sera donné sur l'originalité (autrement dit, si vous faites le strict minimum simplement pour avoir votre bonus, ça ne sera pas compté).

Code secret (3%)

Lorsque l'utilisateur tape `twado` dans la fenêtre, le papillon doit s'animer et changer de couleur graduellement avec le temps.

Basez-vous sur l'animation ici : http://165.227.34.49/~g316k/papillon_animation.gif

Notez : vous ne pouvez **pas** fournir 250 nouvelles images à charger dans le jeu. Vous **devez** calculer le changement de couleur :v)

Si vous choisissez de faire ce bonus, ajoutez un fichier `BONUS.txt` à votre remise dans lequel vous devez décrire ce que vous avez fait.

Barème

- **50%** ~ Exécution (fonctionnalités implantées tel que demandé)
- **30%** ~ Découpage en classes et utilisation judicieuse de l'héritage
- **20%** ~ Qualité du code
  - Code bien commenté (standard *JavaDoc*), bien indenté, bon découpage en fonctions, encapsulation, noms de variables bien choisis, performance du code raisonnable, CamelCase, pas trop compact, pas trop espacé, méthodes `const` au besoin lignes pas trop larges... bref, du code clair et lisible en suivant les principes que vous connaissez
  - **Pas de fuites de mémoire**. Chaque `new` doit avoir un `delete` correspondant.

Remise

Vous devez remettre sur Léa votre projet dans un **.zip**.

Assurez-vous de supprimer les fichiers de Visual Studio, si votre fichier `.zip` pèse ~250Mo, il y a un problème.

Note sur le plagiat

Le travail est à faire **en équipes de deux**. Ne partagez pas de code entre les équipes, ça constituerait un **plagiat**, et vous auriez *zéro*.