

# Flappy Bird

Encadrants: Vincent BREBION, Stefan DUFFNER

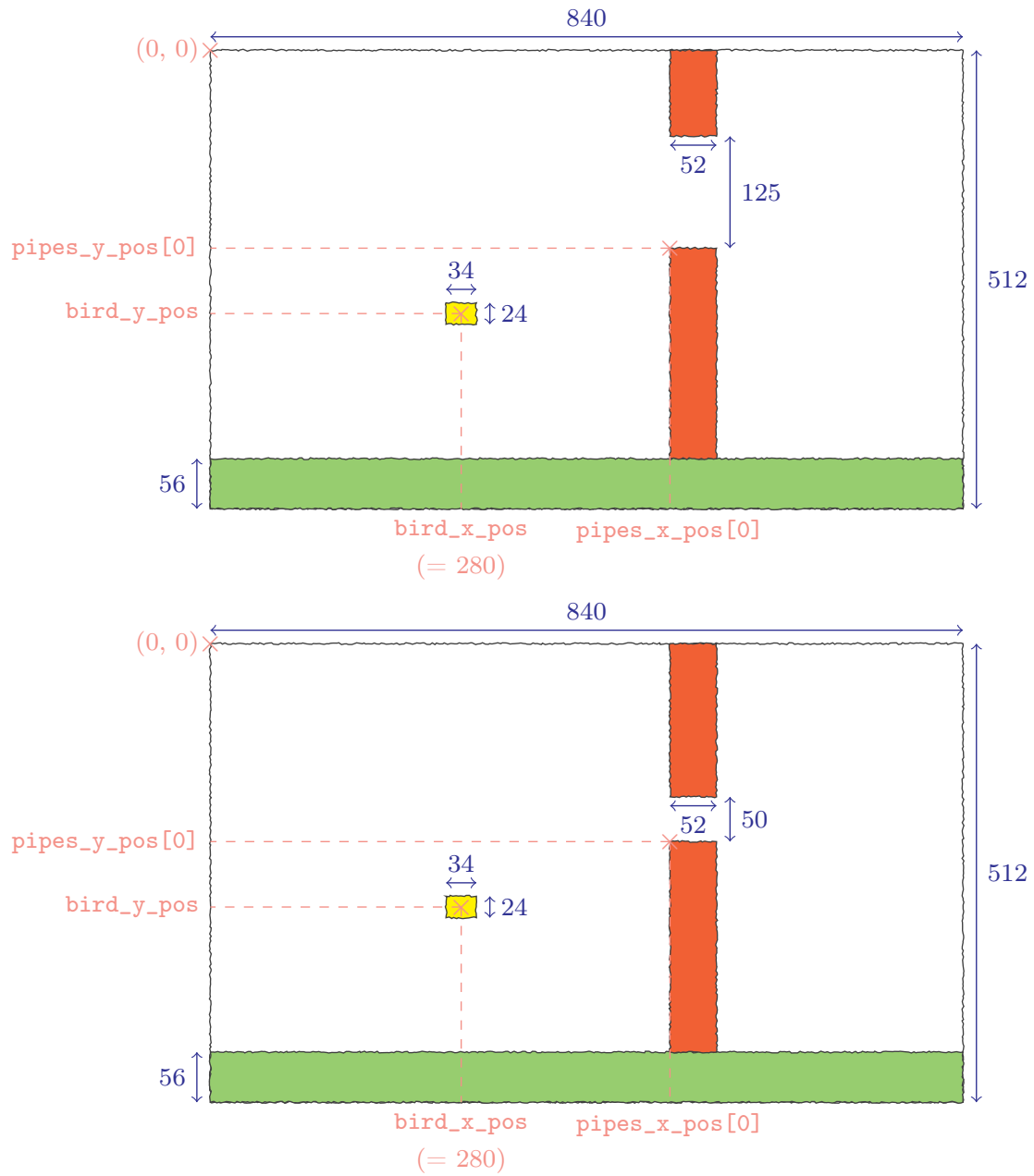


FIGURE 1 – Vue simplifiée de la fenêtre de jeu et de la taille / position des différents éléments qui la composent. La figure du haut est pour la version **facile**, celle du bas pour la version **difficile**. Toutes les valeurs présentées sont en pixels.

# 1 Objectif

Dans Flappy Bird, le joueur contrôle un oiseau, qui doit voler le plus longtemps possible en évitant de toucher le sol, le plafond, ou des obstacles ayant la forme de tuyaux.

Pour cela, le joueur dispose d'un seul et unique bouton. Tant qu'il n'appuie pas dessus, l'oiseau avance en permanence vers la droite, et perd constamment de l'altitude. En revanche, dès que le joueur appuie sur le bouton, l'oiseau se met à battre des ailes et regagne instantanément de l'altitude, lui permettant de slalomer entre les tuyaux.

Dans le cadre de ce projet, il vous est proposé de développer un algorithme permettant à l'ordinateur de jouer à Flappy Bird sans assistance du joueur, afin d'établir le meilleur score possible.

Deux versions vous sont proposées :

- une version **facile**, qui vous permet de prendre en main le jeu et le code associé, vous permettant rapidement d'obtenir des premiers résultats ;
- une version **difficile**, qui est le vrai challenge à relever, et sur lequel une plus profonde réflexion sera nécessaire.

## 2 Environnement de jeu

**Fenêtre** La fenêtre de jeu complète est présentée dans la Fig. 1. Elle a une taille fixe de  $840\text{px} \times 512\text{px}$ , avec le pixel de coordonnées  $(0, 0)$  situé dans le coin supérieur gauche.

**Obstacles — Sol et plafond** Le sol (en vert dans la Fig. 1) est un rectangle de taille fixe couvrant les 56 pixels du bas de l'image, tandis que le plafond correspond au haut de l'image.

**Obstacles — Tuyaux** Les tuyaux (en rouge dans la Fig. 1) constituent la seule partie aléatoire du jeu. Chaque tuyau est composé de deux parties, une partie inférieure se dressant depuis le sol, et une partie supérieure tombant depuis le plafond. Un tuyau a une largeur fixe de 52 pixels, et les deux parties du tuyau sont espacées verticalement de 125 pixels (version **facile**) ou de 50 pixels (version **difficile**). La position de chaque tuyau est représentée par le coin supérieur gauche de sa partie inférieure. La position horizontale  $x$  du tuyau est variable (le tuyau se déplaçant en permanence vers la gauche). La position verticale  $y$  est en revanche fixée lors de la création du tuyau, avec une valeur aléatoire entre  $y = 150\text{px}$  et  $y = 425\text{px}$  (version **facile**), ou entre  $y = 75\text{px}$  et  $y = 350\text{px}$  (version **difficile**). Deux tuyaux successifs sont toujours écartés d'une distance de 342 pixels ( $x_{\text{tuyau2}} - x_{\text{tuyau1}} = 342\text{px}$ ).

**Oiseau** Bien que l'oiseau ait une forme ovale dans le jeu, il s'agit en réalité d'un rectangle de taille fixe de  $34\text{px} \times 24\text{px}$  (en jaune dans la Fig. 1). Même si l'oiseau semble se déplacer vers la droite, sa position horizontale est en réalité fixée à  $x = 280\text{px}$  (l'impression de défilement est créée par le mouvement du sol et des tuyaux). La position verticale  $y$  de l'oiseau est en revanche variable ( $y = 256\text{px}$  au lancement du jeu, puis l'oiseau monte et descend pendant le jeu en fonction des actions du joueur).

## 3 Déplacement des objets

Un point central pour saisir le fonctionnement du jeu et pour être capable de faire en sorte que l'ordinateur joue seul à Flappy Bird est de comprendre l'échelle de temps du jeu, ainsi que comment les objets se déplacent.

**Échelle de temps** Pour simplifier l'échelle de temps, plutôt que de devoir compter des secondes / millisecondes, nous allons ici compter le temps en images. En effet, après avoir traité les touches pressées par l'utilisateur, après avoir fait mis à jour la position de l'oiseau, la position des tuyaux, le score, ..., le jeu génère une nouvelle image, qui est affichée à l'écran. Il est donc plus simple de calculer les positions et trajectoires en fonction du nombre d'images, et c'est l'échelle de temps que nous allons utiliser ici.

**Signes** Par convention, nous utiliserons ici les signes suivants :

- pour un déplacement horizontal (suivant l'axe  $x$ ), les valeurs seront positives pour un déplacement vers la droite, et négatives pour un déplacement vers la gauche ;
- pour un déplacement vertical (suivant l'axe  $y$ ), les valeurs seront positives pour un déplacement vers le bas, et négatives pour un déplacement vers le haut.

**Déplacement des tuyaux** Chaque tuyau se déplace vers la gauche. Leur vitesse est constante : elle est de  $-3\text{px/image}$  (version **facile**) ou  $-4\text{px/image}$  (version **difficile**). Cela veut dire que chaque fois qu’une image est générée, et donc que la position des tuyaux est mise à jour, chaque tuyau est décalé de 3 pixels (version **facile**) ou 4 pixels (version **difficile**) vers la gauche par rapport à sa position sur l’image précédente.

**Déplacement de l’oiseau** Le déplacement vertical de l’oiseau est un peu plus complexe. Considérons dans un premier temps le cas où l’utilisateur n’appuie pas sur la barre espace :

1. l’oiseau est au départ dans les airs, avec une vitesse initiale nulle ;
2. vu qu’il est dans les airs et qu’il ne bat pas des ailes, l’oiseau se met à chuter de plus en plus vite à cause de la gravité : à chaque nouvelle image, sa vitesse verticale est de plus en plus élevée, et sa position se rapproche donc de plus en plus du sol jusqu’à ce qu’il s’y écrase.

En revanche, si l’utilisateur appuie une fois sur la barre espace :

1. l’oiseau bat une fois des ailes et gagne instantanément un boost de vitesse négative, il se met donc à remonter dans les airs ;
2. sauf qu’à cause de la gravité, cette vitesse négative se rapproche de plus en plus de zéro : l’oiseau continue de monter, mais de moins en moins ;
3. au bout d’un certain temps, le boost de vitesse est totalement annulé par la gravité, et l’oiseau se remet à chuter de plus en plus vite jusqu’à s’écraser au sol.

Il y a donc 3 éléments à prendre en compte :

1. la **gravité**, qui est constante, et qui fait en sorte que la vitesse de l’oiseau en direction du sol augmente toujours de plus en plus, et donc que la position de l’oiseau se rapproche du sol ;
2. la **vitesse** de l’oiseau, qui est mise à jour à chaque image en fonction de la gravité, et en fonction de si l’oiseau a battu des ailes ;
3. la **position** verticale de l’oiseau, qui est mise à jour à chaque image en fonction de sa vitesse.

Dans le jeu, ici :

1. la **gravité** est constante, et est de  $0.2\text{px/img/img}$  (version **facile**) ou  $\frac{1}{6}\text{px/img/img}$  (version **difficile**), c’est-à-dire qu’à chaque nouvelle image,  $0.2\text{px/img}$  (version **facile**) ou  $\frac{1}{6}\text{px/img}$  (version **difficile**) sont ajoutés à la vitesse de l’oiseau ;
2. la **vitesse** de l’oiseau est  $0\text{px/img}$  au lancement du jeu, elle est mise instantanément à une valeur de  $-5\text{px/img}$  à chaque fois que l’oiseau bat des ailes (valeur négative = l’oiseau va vers le haut, comme indiqué précédemment), et la gravité fait augmenter cette valeur à chaque nouvelle image comme expliqué dans le point précédent ;
3. la **position** de l’oiseau est  $y = 256\text{px}$  au lancement du jeu (comme expliqué dans la Section 2), et à chaque nouvelle image, cette position est mise à jour en fonction de la vitesse : du fait, si par exemple la vitesse actuelle est de  $3\text{px/img}$ , et que la position est  $y = 200\text{px}$ , la position de l’oiseau pour cette image est augmentée de 3 pixels, et il se retrouve donc à une nouvelle position  $y = 203\text{px}$ .

## 4 Interaction avec le jeu

**Fichiers** Le code du jeu implémenté en Python vous est fourni en accompagnement de ce sujet, dans un fichier appelé `flappy_game.py`. Vous **ne devez pas** modifier ce fichier, et il ne vous est pas demandé de comprendre comment ce code fonctionne (tous les éléments importants étant décrits dans ce sujet). Vous disposez d’un deuxième fichier appelé `flappy_interactif.py`. Ce fichier vous présente comment le jeu est initialisé, et comment il est possible de faire tourner le jeu de manière interactive (c’est-à-dire que c’est au joueur d’appuyer sur la barre espace pour lancer le jeu et faire voler l’oiseau). Vous n’avez pas besoin de modifier ce fichier, mais vous pouvez l’exécuter pour jouer à Flappy Bird par vous-même. Vous disposez enfin d’un troisième et dernier fichier, appelé `flappy_programmable.py`, qui vous présente un exemple du jeu lancé de manière non-interactive, et où c’est l’ordinateur qui contrôle l’oiseau de manière très basique (il fait en sorte que l’oiseau batte des ailes chaque fois que sa position verticale est supérieure à  $y = 250\text{px}$ ). Ce fichier peut vous servir d’exemple pour implémenter votre propre code, n’hésitez pas à le modifier !

**Valeurs pouvant être lues** Voici l'ensemble des valeurs que vous pouvez récupérer et que vous allez sûrement devoir utiliser dans votre code (attention, vous n'avez **pas le droit** de modifier ces variables directement, vous n'avez que le droit de les lire pour faire vos calculs) :

- `game.bird_x_pos` : la position  $x$  de l'oiseau à l'écran (qui vaut toujours 280, comme indiqué dans la Section 2)
- `game.bird_y_pos` : la position  $y$  de l'oiseau à l'écran
- `game.pipes_x_pos` : une liste contenant la position  $x$  de l'ensemble des tuyaux (comme décrit dans la Section 2); attention, la taille de cette liste dépend du nombre de tuyaux à l'écran, elle est donc vide par exemple au lancement du jeu, et peut contenir jusqu'à 3 tuyaux
- `game.pipes_y_pos` : une liste contenant la position  $y$  de l'ensemble des tuyaux (comme décrit dans la Section 2); attention, la taille de cette liste dépend du nombre de tuyaux à l'écran, elle est donc vide par exemple au lancement du jeu, et peut contenir jusqu'à 3 tuyaux

Voici également un ensemble de valeurs qui sont mises à votre disposition et que vous pouvez lire si cela vous est nécessaire :

- `game.score` : le nombre de points actuel (remis à zéro après chaque collision de l'oiseau)
- `game.high_score` : le record de points effectué sur la session (cette valeur n'est pas sauvegardée lorsque vous quittez le jeu)
- `game.running` : valeur qui indique si le jeu est actuellement en train de s'exécuter (**False** au lancement du jeu ou après que l'oiseau soit rentré dans un obstacle, **True** sinon)
- `game.exit` : valeur qui indique si l'utilisateur a appuyé sur la croix pour fermer la fenêtre de jeu (**True** si c'est le cas, **False** sinon)
- `game.frame_count` : le nombre d'images qui ont été affichées à l'écran (incrémenté pour chaque nouvelle image, et remis à zéro au lancement du jeu et après chaque collision de l'oiseau)

**Fonctions pouvant être appelées** Une fonction centrale doit être appelée en permanence, pour calculer et mettre à jour la position de l'oiseau, des tuyaux, du score, et afficher la nouvelle image du jeu à l'écran :

- `game.update()` : il ne vous est pas nécessaire de comprendre comment cette fonction opère, mais vous devez toujours l'appeler en premier dans la boucle principale de votre programme avant de prendre la décision de faire battre des ailes ou non à l'oiseau (comme cela est fait ligne 25 dans les scripts d'exemple); cette fonction renvoie :
  - -3 si l'utilisateur a pressé la croix pour quitter le jeu;
  - -2 si le jeu est en attente (`game.running == False`), donc au lancement et chaque fois après que l'oiseau ait eu une collision;
  - -1 si le jeu est en train de tourner;
  - le score obtenu si lors de l'appel à la fonction l'oiseau a eu une collision avec un objet.

Sinon, comme pour un vrai joueur, une seule interaction est possible avec le jeu pour l'ordinateur, et il s'agit de la seule fonction dont vous aurez besoin de contrôler comment elle est appelée :

- `game.flap()` : fonction qui, lorsqu'elle est appelée, permet à l'oiseau de reprendre de l'altitude, suivant les indications données dans la Section 3; cette fonction ne prend aucune entrée et ne retourne rien, elle doit juste être appelée lorsque la stratégie que vous avez implémentée détermine que l'oiseau doit battre des ailes.

Si vous avez besoin de connaître la vitesse actuelle de l'oiseau, une fonction vous est fournie (ne lisez pas directement `game.bird_y_speed`!) :

- `game.get_bird_y_speed()` : fonction qui, lorsqu'elle est appelée, retourne la vitesse actuelle de l'oiseau en px/img.

Enfin, si cela peut vous aider pour déboguer votre code, une fonction vous est fournie pour tracer un cercle dans la fenêtre de jeu :

- `draw_circle(pos_x, pos_y, size)` : fonction qui, lorsqu'elle est appelée, trace un cercle à la position (`pos_x`, `pos_y`) de taille `size`; une fois tous les appels à cette fonction réalisés, un unique appel à `pygame.display.flip()` doit être fait pour afficher tous les cercles dans la fenêtre de jeu.

## 5 Proposition de découpage du projet

Même si ce n'est pas une obligation, nous vous conseillons de suivre le découpage suivant pour que ce projet se déroule dans les meilleures conditions au cours de ces deux semaines :

1. Lors de la première séance, prenez le temps de bien lire ce sujet ; n'hésitez pas à l'annoter et à tester le jeu dans sa version interactive pour bien en comprendre le fonctionnement.
2. Sur les séances suivantes, une fois le sujet compris, il va falloir que vous soyez capable de modéliser la trajectoire de l'oiseau, en fonction de si celui-ci bat des ailes ou non. Essayez donc de calculer à la main ou avec un tableur les positions de l'oiseau en fonction du temps (en fonction des éléments décrits dans la Section 3), et représentez-les sur un graphe simple pour visualiser les différentes trajectoires possibles.
3. En ayant en tête ces trajectoires, réfléchissez ensuite ensemble à quelles stratégies adopter pour contrôler l'oiseau de manière à ce qu'il survive le plus longtemps possible. N'hésitez pas à faire des schémas et à proposer plusieurs stratégies lors de cette phase : même si toutes ne seront pas forcément viables, il sera intéressant de les tester pour comparer leurs résultats.
4. Enfin, et seulement une fois tous ces éléments mis en place, vous pourrez passer à l'implémentation de ces stratégies en développant le code correspondant, et en comparant les résultats obtenus par chacune.

Les encadrants sont à votre disposition pour vous aider, n'hésitez pas à les solliciter si besoin !