Angry Birds – Projet S3

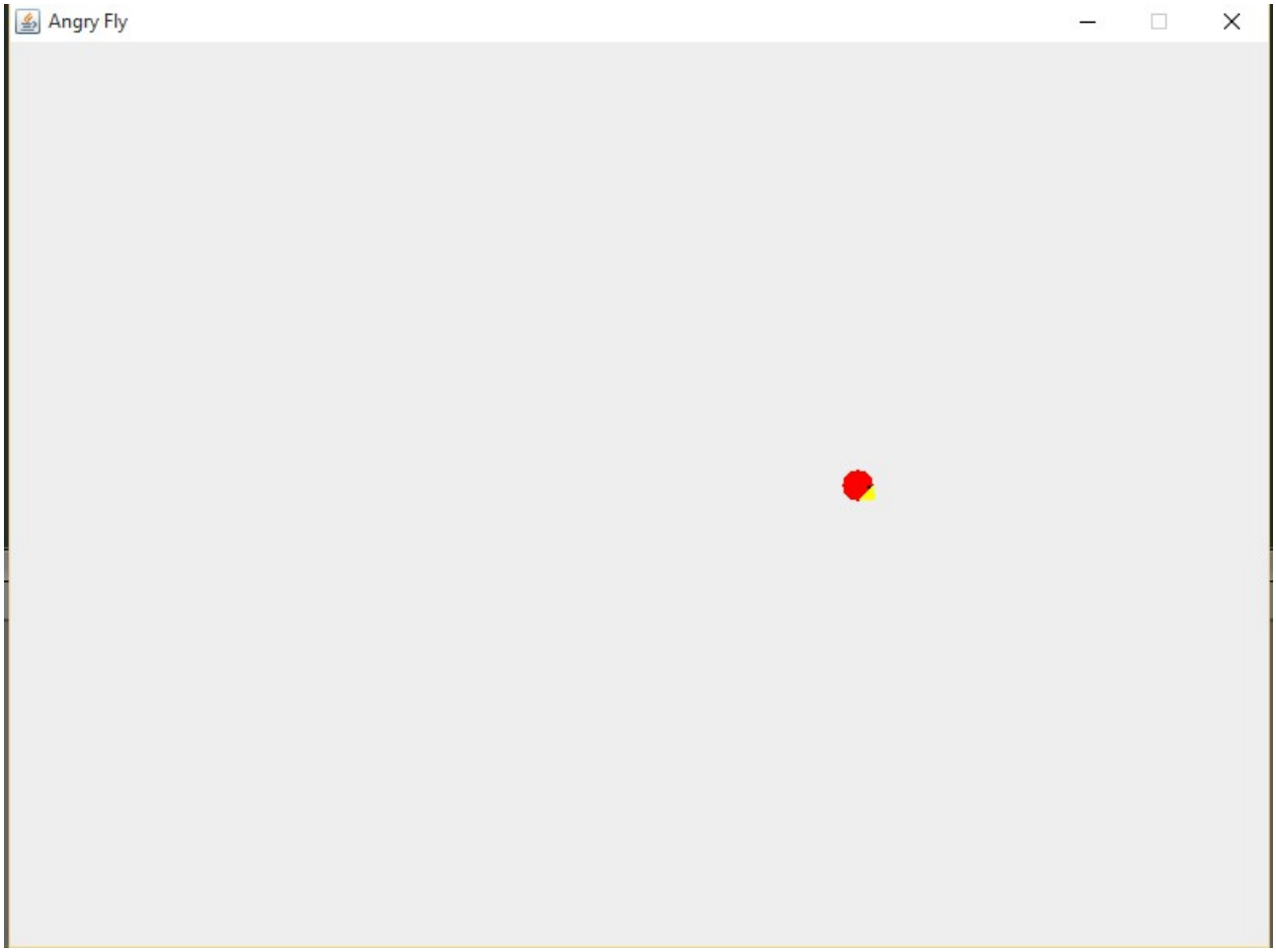
Groupe *K5** composé de Ali Douali, Aurélien Svevi, Wissam Lefèvre et Arthur Roland.

Sommaire :

PS : Ce document a été fait hors contexte du sujet, et avait pour but initial de lister les fonctionnalités rajoutés au jeu.

Version 0.1 :

Dans cette version du jeu, l'objectif a juste été de modéliser un oiseau et une courbe, et faire en sorte que l'oiseau suit cette courbe.



Ajout de -

- La classe AnimationOiseau qui est la fenêtre principale gérée par un JFrame, de taille 800 sur 600 par défaut.
- La fonction oiseau qui se dessine en un point x et y.
- La classe Courbe qui, en trois paramètres, nous donne une courbe.
- La sous-classe Avancement de AnimationOiseau, la classe implement un Runnable et est donc un thread qui est le centre de l'animation, le thread tourne jusqu'à ce que le x, position de l'oiseau, dépasse la fenêtre.

Version 0.11 29/09/2015 :

Cette version est une version améliorée de la 0.1 notament avec l'arrivée des nouveaux skins permettant de modifier l'appararence de son oiseau, de plus l'oiseau laisse une trainée désormais grâce aux tout nouveaux skins d'empreinte.

Ajout de :

- Des tout nouveaux skins d'oiseau :
  - Le Rouge Gorge, le skin de base est un petit oiseau rouge au bec jaune
  - Le Canard, le canard tout blanc et imposant !
  - La Soucoupe Volante, effrayera vos ennemis
    - Dans le prochain pack -> Le Shuriken et Le Papillon sont à venir
- Les toutes nouvelles empreintes, grâce à elles, votre oiseau laissera une jolie trainé derrière lui lors de sa course
  - De nombreux skins sont déjà paru tel que : La tracé de couleur, Arc En Ciel, Razer, Drapeau Gaypride et plusieurs pays ont leur drapeau déjà créé.
- L'oiseau maintenant inclinera son bec en fonction de sa direction
- Meilleur répartition des classe, désormais 2 packages co-existent avec le package principal "AngryBirds" et le package "Skin" qui contient la classe abstraite des skins et les skins eux même et l'énumération des empreintes "FootStep"

A venir :

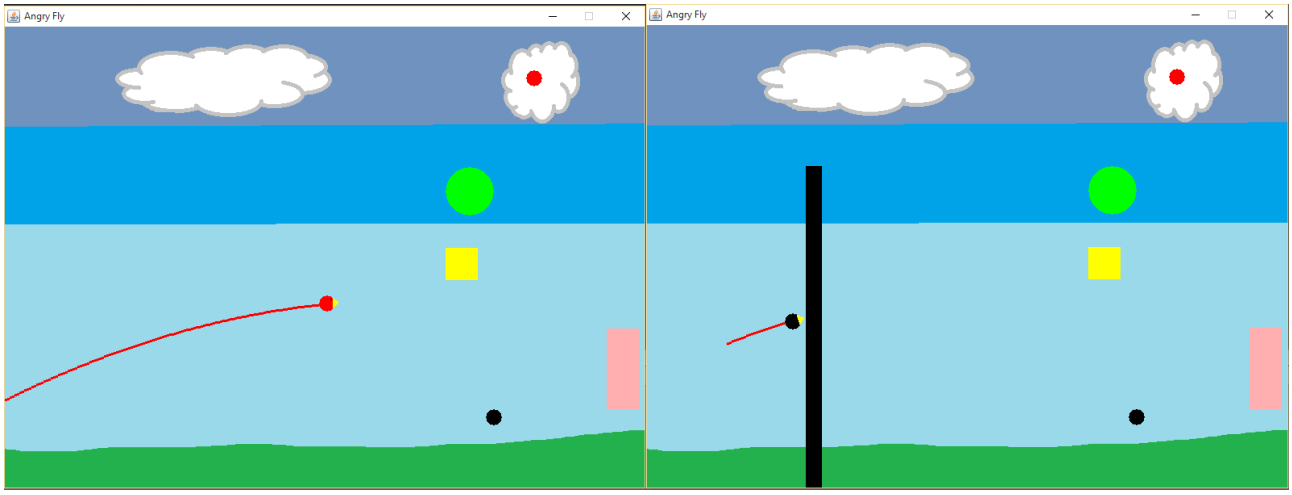
- La création des obstacles
- Un menu de démarrage "de test" pour paramètrer plus facilements les phases de test

de nos lancés
- Plus de skins
- Plus d'empreintes
- Un sol pour ne pas tomber dans le néant
- Des skins de sols !
- Un ciel et des nuages
- Des skins d'environnement !
- Et des skins d'obstacle !

Version 0.2 - 02/10/2015 – Avancement :

- ~~La création des obstacles~~
- Un menu de démarrage "de test" pour paramètrer plus facilements les phase de test de nos lancés -> En phase de developpement
- ~~Un sol pour ne pas tomber dans le néant~~
- ~~Un ciel et des nuages~~

Il a été fait dans cette version les obstacles, la hitbox et la collision. Cette dernière gère les collisions entre l'entité principale (le pigeon) et les obstacles, si un obstacle est percuté, une méthode appelée renvoie l'index de l'obstacle dans la liste des obstacles contenues dans la classe Constante, la hitbox est une nouvelle classe et un nouvelle attribue aux entités telle que l'oiseau et les obstacles, la hitbox est un carré invisible autour de chaque entité qui la suit et la situe virtuellement dans le décor. L'entité même ronde est virtuellement carré à cause de cela, mais cela changera avec la venue des chocs entre entités. Et enfin les obstacles sont la seconde partie modifiable du jeu, il s'interpose entre le pigeon et son (futur) objectif. Les obstacles sont actuellement fixes et carrés ou ronds, mais ça risque de changer.



A venir :

- Un lancement programmable in-game (avec les touches directionnelles ou à la souris)
- Un vecteur de force qui accentura la vitesse et la distance de vol du pigeon
- Des rebonds contre les obstacles, uniquement le pigeon pour commencer, puis un effet de réaction dû au choc du pigeon sur un obstacle pour aller plus loin
- Des obstacles triangulaires, et amovibles (qui comprend le fait d'être bougé et/ou éliminé)
- Un tout nouveau type de skin, le canon qui permettra de lancer le pigeon
- Un pseudo objectif
- Et le menu du jeu

Bilan pour la version 0.3 et pour le Jalon 1 du 06/11/15:

05/11/15 :

Un mois aprés le dernier Bilan (0.2), beaucoup de choses ont été modifié afin de simplifier le code et sa structure générale, améliorer les performances du jeu, permettre aux tests de fonctionner, d'améliorer le système d'add-on et les futures mises à jour du Jalon 2 et 3 (Ennemis, Canon, Vitesse, Obstacle en mouvement...)

A ce jour, voici le contenu :

Restructuration des entités :

- Restructuration des entitées en général, la classe Entity est mère de tout désormais.
  - La classe Bird étend la classe Entity et est mère de RougeGorge et Canard, les deux skins fonctionnels du jeu.
  - La classe Obstacle étends la classe Entity et est mère de Carré et Rond, les deux obstacles fonctionnels du jeu.
- De plus, chaque entité a désormais une Hitbox carré à sa taille (même les obstacles rond ont des hitbox carré), quand deux hitbox se rencontrent le jeu s'arrête, cette condition est gérée par la présence d'un Comparator<Hitbox> dans Hitbox et aux méthodes de la classe Collision (moteur de collision).
- De plus encore, les oiseaux tirés de la classe Bird ont un système de module, ModuleBird est la classe abstraite des modules et un module sert à composer l'oiseau, par exemple le module Oeil, ou le module BecCanard, l'oiseau sans module est juste un rond coloré.

Restructuration de l'affichage :

- Le jeu est désormais contenu dans un JPanel AnimationOiseau, ce qui permet le DoubleBuffer et améliore grandement l'affichage (plus de freeze ou de glitchs indésirés sur le rafraichissement) mais cela n'empéche pas une amélioration au niveau du repaint qui ne repaint que la l'oiseau à chaque réaffichage.
- Le Panel est lui contenu dans une JFrame GameFrame, qui prend en titre le numéro du lancé sur son nombre total de lancé, et un petit prototype de "module programmeur du jeu" a été fait, il ne permet que de mettre le jeu en pause pour l'instant, et il sera sûrement supprimé pour le final du jeu.
- Possibilité de voir les Hitbox en rajoutant une méthode du Visualisateur dans le paint de l'AnimationOiseau

Amélioration sur les calculs :

- Un prototype de classe Matrice a été lancé (suite au cours sur les Matrices) et dans le futur, les déplacements des entitées seront sûrement gérés par ces nouvelles méthodes
- Une amélioration a été faite sur la vitesse, l'oiseau peut désormais aller de 0 à infini kilomètre(s)/heure, tout est géré dans le HeartCore grâce à un système de pause, de calcul de déplacement fait et de rafraichissement/repaint calculé
- Dans Constante, les courbes et les obstacles sont générés aléatoirement grâce à des méthodes, mais il reste toutefois possible de les générer manuellement.

A venir pour le Jalon 2 :

- Un menu, fait en JavaFX (pour l'ergonomie et sa flexibilité).
- Un canon/fronde.
- Un meilleur système de fond qui permettra le changement de taille de la fenètre (fixe en jeu, modifiable dans les options du menu).
- Et plus de skin pour les fonds
- Les rebonds contre les obstacles, la fin du jeu sera quand l'oiseau ne bougera plus ou qu'il aille trop haut/loin dans le décors
- Les ennemis, et du coup un second type de fin, qui sera quand il n'y a plus d'ennemi
- La posibilité de lancer plusieurs oiseaux dans la même partie sans tout remettre à 0.
- Des obstacles qui bougent, qui se détruisent et qui détruisent l'oiseau sans lui donner une chance.
- Des pouvoirs d'oiseau (boost de vitesse pour le RougeGorge, boost de masse pour le Canard) la puissance d'un oiseau se caractérise par sa vitesse et sa masse.
- Et peut être plus...
- Plus de skin (bientôt en vente sur l'e-store).

A noter : La classe LivOne est la classe qui contient le main à utiliser pour la Jalon 1, il contient un thread qui lance le jeu à chaque appel, il peut être appelé "n" fois et lancera l'oiseau dans un environnement aléatoire (courbe générée, obstacles générés) L'environnement est aléatoire mais peut être paramétré si on le souhaite et juste en navigeant entre la méthode iniz() de Constante et du main de LivOne, on peut paramétrer le jeu. Un futur menu rendra cela utilisable sans les sources.

Bilan du travail fait par l'équipe au Jalon 1 :

Au cas par cas dans l'ordre alphabétique :

- *Ali Douali* : Travail d'algorithmie sur la gestion des collisions, la gestion des hitbox, comment représenter une entitée dans l'espace et comment détecter les collisions.

- *Wissam Lefèvre* : Travail d'algorithmie sur les entitées en général, comment une entitée peut suivre une courbe, la dessiner, la manipuler (rotation etc...), travail également effectuer sur la javadoc.

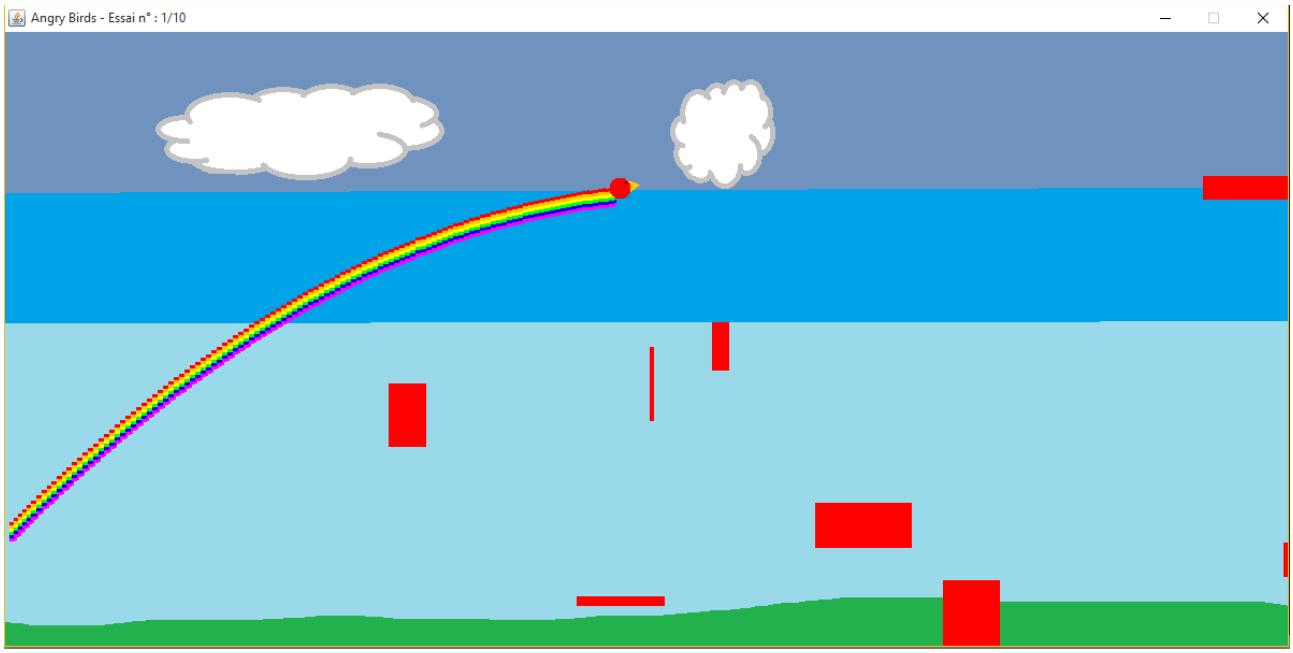
- *Arthur Roland* : Travail sur la sécurité du code et implémentation des tests, son travail consistait à tout surveiller, corriger les bugs, il a également travaillé sur comment implémenter les matrices du jeu.

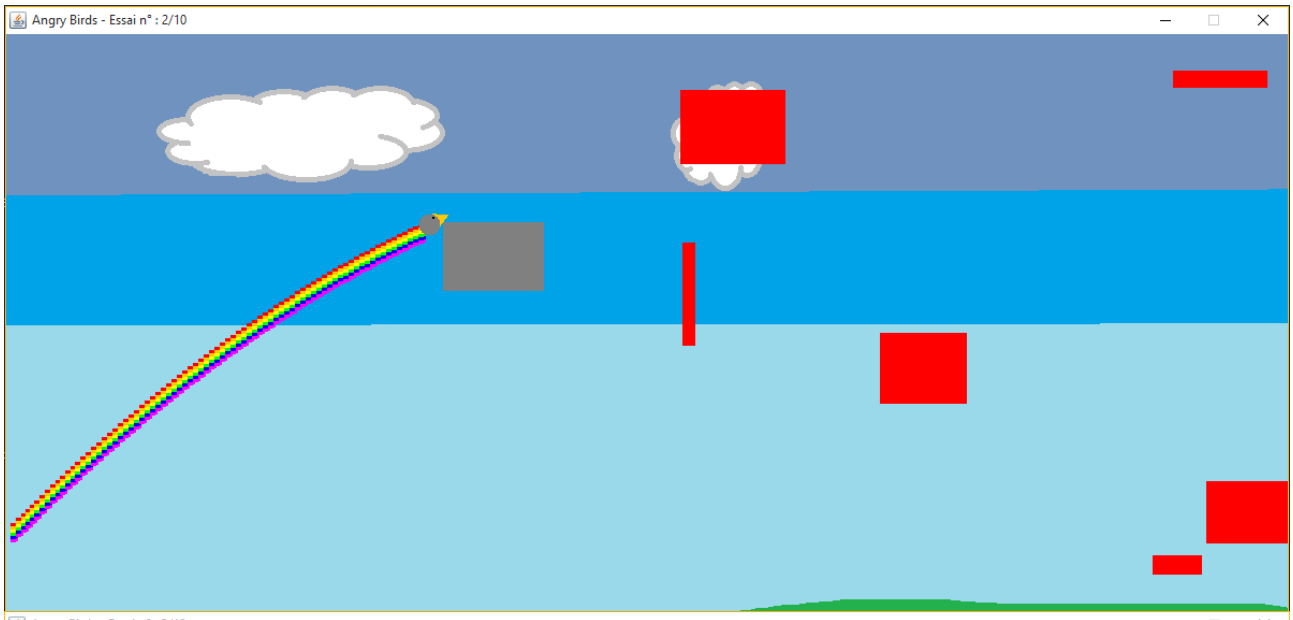
- *Aurélien Svevi* : Travail sur la courbe, comment générer la courbe, la tracer. Travail également effectuer sur l'UML du projet.

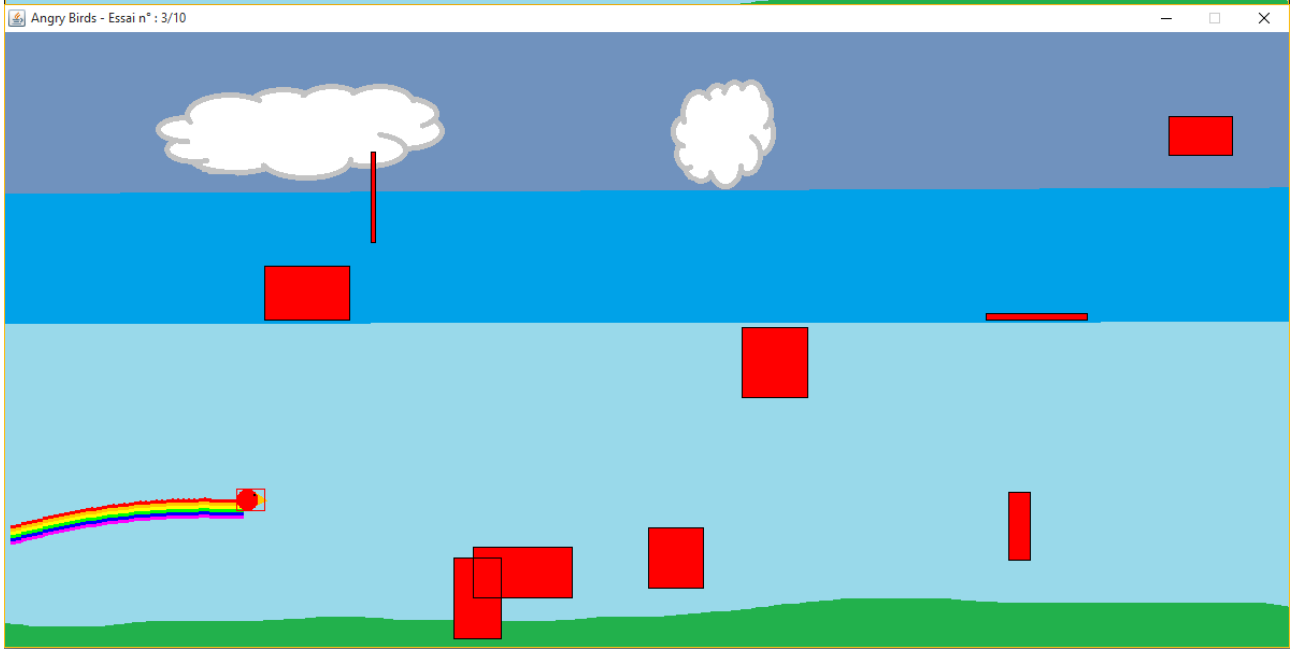
Travail d'équipe :

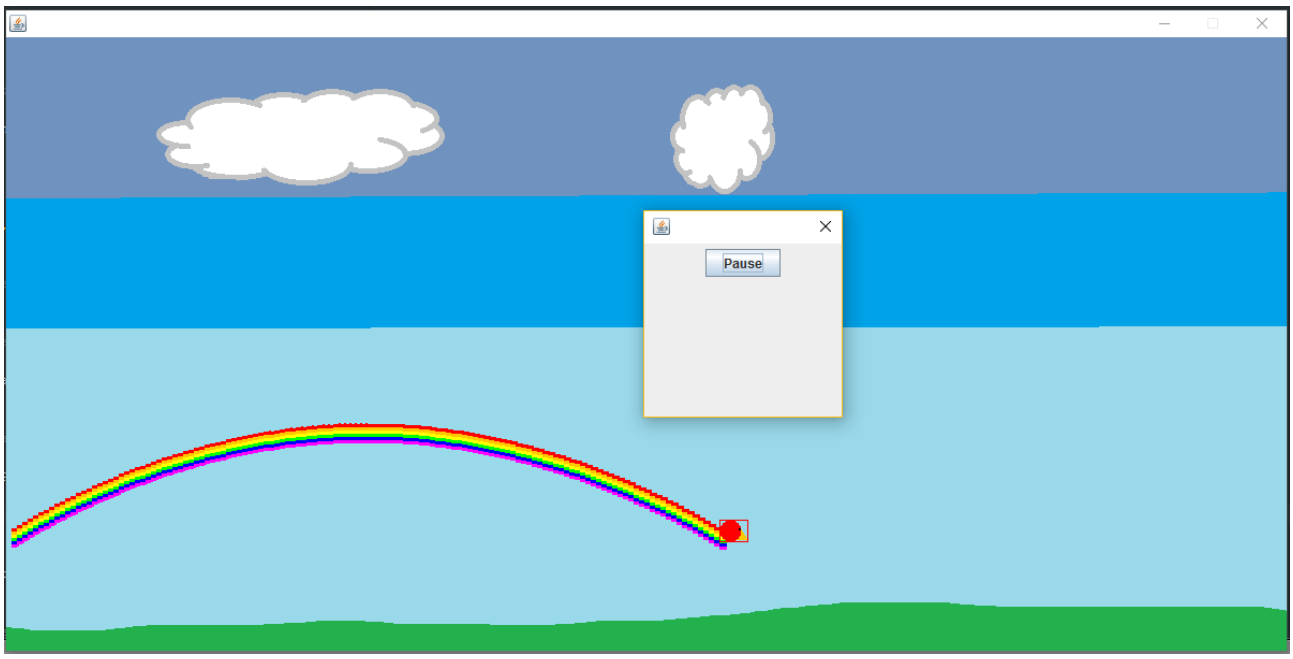
Le travail a été fait avec Github, technologie Java 8 (swing et awt essentiellement), l'équipe n'utilisait pas les mêmes IDE mais ça n'a pas posé de problème majeur. La plus part des push sur le Github sont fait par Wissam car l'équipe avait eu quelque soucis pour push (Au départ seul Aurélien pouvait push, puis ça a été au tour de Wissam). Au niveau de la communication, un groupe Facebook a été fait pour communiquer rapidement, et des conversations sur Skype pour coder ensemble ce qui a ainsi facilité la cohésion, le travail en groupe et une bonne organisation en ce qui concerne la répartition des tâches.

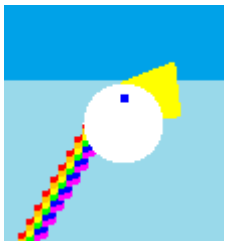
Screenshots du jeu à la veille du Jalon 1 :











```
entites.bird.Footstep

public static final Footstep ARCENCIEL

Au couleur de l'arc en ciel
```

```
ep.;
```

ARCENCIEL	Footstep
BRETZEL	Footstep
GAY	Footstep
NEERLANDE	Footstep
NOIR	Footstep
NONE	Footstep
RAZER	Footstep
ROSE	Footstep
ROUGE	Footstep
USA	Footstep

Informations relatives aux designs partterns

Le sujet nous spécifie d'utiliser au moins deux design pattern. Concrétement dans le projet :

- Type Fabrique abstraite (Abstract Factory) :
  Quelques classes sont issues d'une classe qui les définies. Le principe du programme est qu'il y a au centre de tout l'entitée (classe : Entity) qui définit tout ce qui est sur le jeu, oiseau, obstacles, ennemies... Mais ce n'est pas tout car avant qu'un oiseau soit un oiseau, il passe lui aussi par une classe abstraite, Bird, qui définie la base d'un oiseau, ses paramètres tirés en partie de Entity et ses méthodes de base. Il en est de même pour les obstacles, les modules.

- Type Builder :
  Le builder fait qu'une classe est constituée de plusieurs autres classes permettant de la simplfiée, cela est pour l'AnimationOiseau qui est en réalité une alliance entre elle même, Collision (qui gère les collisions entre les entititées et les bordures), HeartCore (qui gère le timer) et Visualisateur (qui gère l'affichage). Cela simplifie grandement la compréhension du projet et simplifie la forme et l'utilité d'une classe.

- Type Mediator :
  La classe Visualisateur, qui est la partie affichage du MVC, réunit tous les draw de toutes les entitées créées, c'est à dire un draw pour chacun : le fond, pour l'oiseau, pour les obstacles, les empreintes (footstep) et les ennemis, tous ces draw sont réunis ici et réutilisés indépendamment ou tous ensemble grâce aux méthodes du Visualisateur qui "simplifie" en autre l'utilisation des draw (il n'y a plus qu'un import à faire) et la gestion des entitées entre l'Animation, le Visualisateur et la classe Constante.

Liste des tests du Jalon 1 :

Tests de ConstanteTest.java :

- testIniz() :
  - Test si les listes réinitialisées dans la méthode sont bien vide à l'appel de la fonction

- testGenerateCourbe() :
  - Test si la courbe est bien comprise entre les valeurs voulues c'est à dire, entre 0,0007 et 0,0008 pour a et entre -2 et 0 pour b, le c est normalisé

- testGenerateListObstacle() :
  - Test d'abord si on initialise une liste avec 0 obstacle, qu'il n'y ai pas d'obstacle dans la liste retourné, puis test avec 5 obstacles, et test si les obscles contenues dans la liste sont bien des obstacles

- testGenereObstacle() :
  - Test la génération d'un obstacle, qu'il soit dans les bornes voulues, c'est à dire entre 200 et "x max de la fenetre" pour x, entre 0 et "y max de la fenetre" pour y, entre 0 et 100 pour sa taille h et w

Tests de CourbeTest.java :

- testGetYenX() :
  - Test si la veleur obtenue en x est bien celle attendu.