Angry Birds – Projet S3

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

Sommaire :

Version 0.4 le 11/11/2015 :

La version 0.4 est la suite immédiate de la version 0.3, dernière version du le Jalon 1.
A cette version 0.3 on s'était arrété au fait de lancer 10 fois un oiseau sur une liste d'obstacles générés aléatoirement...

Pour se qui est de la version 0.4 :

- Restructuration complète des fonds
  - La classe fond a était supprimé (anciennement une énumération de lien vers des images)
  - Désormais le fond se fait grâce à des ImageCustomz, une ImageCustomz est un objet contenant un lien vers l'image, sa position en x et en y

- Création d'un langage propre au programme, le PFAG (Pack For AnGry birds), un fichier .pfag sera lu et interprété par le jeu pour de futur fonctionallités
  - Une classe PFAGReader et Constante permmettent de lire les PFAG
  - Une documentation du langage est fournie dans les ressources du jeu
  - Le PFAG permet actuellement de donner une liste d'ImageCustomz qui constitueront le fond du jeu, une liste d'obstacles et une dimension pour la fenêtre du jeu

PS : La technologie NIO a été utilisée pour la lecture de fichier (à la place de java.io -> java.nio)

ScreenShots :

- Création d'un menu encore en béta, le menu est animé et relativement jolie et sofistiqué, la technologie JavaFX a été utilisé pour le coup (comme prévue dans le Jalon 1)
    - Le menu permet actuellement de lancer le jeu et de quitter le menu, plus de fonctionnalités viendront avec le temps

ScreenShot du menu :

Version 0.5 le 22/11/2015 :

Dans la version 0.5,

- Du changement sur le menu, une musique est jouée dans le menu désormais, de plus les boutons tounent quand on les focus avec la souris.
- Dans le jeu, remake de classe Courbe qui prend désormais des paramètres qui seront utilisés pour créer deux fonctions du second degrés, la première servira à donner la courbe pour la position de l'entité en X et la deuxiéme en Y.
- Les entités ont aussi des courbes désormais (n'est plus externe aux entités, mais dans l'entité elle même) conséquence ? Toute les entités peuvent bouger désormais, un boolean "isMoving" permet de savoir si l'entité doit bouger ou pas.
- En vrac, les mises à jour mineures :
    - Rajout d'une icone pour la fenêtre du menu
    - Rajout d'un titre au dessus des boutons du menu "ANGRYBIRDS"
    - Rajout des options, non opérationnel encore
    - Rajout d'un nouveau skin de fond, le style "Retro 80" ou Laser...
    - Rajout de la classe LivTwo qui cause "l'obselètation" de la classe LivOne
    - Modification relatif au format MVC qui fait que la classe Entity n'a plus aucune relation avec une librairie java quelconque, alors qu'elle était en relation avec java.awt, le fait d'avoir changé le constructeur de HitBox et le rajout de la classe Skin, pour ne pas dire Graphics a changé cela.
    - Rajout d'une classe Skin qui définit clairement l'affichage "réel" d'une entité (sachant que Hitbox est son affichage "virtuel")
    - Changement des skins des oiseaux avec le rajout d'un oval autour du corps en noir, qui permet de mieux voir l'oiseau nottament si il est posé sur un fond qui a une même couleur que lui (comme le canard dans les nuages) pseudo technologie de Cel-shading pour avoir l'air trop moderne.

- A venir :
    - La classe "Luncher", qui permettra de lancer l'oiseau avec des paramètres définis par l'utilisateur (une étude est à faire), elle se lancera dans un premier temps uniquement grâce à l'angle d'inclinaison de l'oiseau au départ, puis part un vecteur force.
    - Des obstacles ronds

    - Des rajouts pour donner un style ultramoderne au jeu

Petit point à propos du format MVC :

Le jeu est constitué d'entités (Entity), une entité définie un objet du jeu par trois points, son Skin qui est son affichage réel, sa Hitbox qui est son affichage virtuel et sa Courbe.

Le but du Skin est, par le biais du Visualisateur de donner un affichage qui sera écrit dans la Graphics du Panel principale (AnimationJeu)
Quant à Hitbox, il a pour but et par le biais de son Comparator et de la classe Collision de savoir quand un Bird rencontre un Obstacle.

Au final tout cela se rencontre dans AnimationJeu qui avance par le biais de HeartCore qui gére plutot la partie calcul du Panel (modéle) et Visualisateur qui gére la vue.

Version 0.6 le 23/11/2015 :

Dans la version 0.6,

- L'oiseau peut désormais décoler de n'importe ou dans n'importe quel direction choisi par l'utilisateur
    - L'utilisateur peut orienter le pigeon dans la direction voulu, il lui suffit d'utiliser les fléches de son clavier pour orienter et d'appuyer sur espace pour lancer le pigeon. La vitesse n'est pas paramétrable pour l'instant.

- Amélioration du PFAG (rappel, langage créé par moi même et utilisé pour configurer une map) rajout des paramètres d'obstacles aX, bX, cX, aY, bY et cY permettant de paramètrer une courbe à un obstacle. PS : cX et cY sont en réalités inutile, si un utilisateur rentre "x=500" puis "cX=200" la position de l'obstacle sera celle du dernier paramètre lu, donc 200 en x ici. Par défaut la valeur de ces paramètres est 0. Second PS : Ces paramètres peuvent prendre des Doubles, sauf les cX/Y
- Une amélioation a été porté sur les angles des modules et un nettoyage a été fait pour éviter des imports inutiles ou plutot pour centraliser les import awt.Graphics sur Skin, ce qui fait que le Graphics n'intervient plus dans les packages "Entity & fils".

Version 0.7 le 11/12/2015 :

Dans la version 0.7,

- Le drag'n'drop a été implémenté, désormais le pigeon est lançable par les touches du clavier, comme dans la version 0.6 mais aussi par un drag and drop de la souris, le drag and drop peut se faire n'importe ou sur la fenètre.

Pour utiliser le drag'n'drop il suffit de cliquer et de rester cliquer sur un endroit de l'écran, puis une cible apparaitra, le centre est le bord minimal a atteindre pour avoir un changement, le cercle externe et le bord maximal de puissance, au delà l'oiseau aura atteint sa puissance maximal.

- Mais aussi les obstacles en mouvement ont été implémenté, désormais il suffira de spécifier dans un fichier PFAG, lors de l'instanciation d'un obstacle (rappel d'une commande pfag : obstacle:type=carre,bX=0.2,bY=0.2,mouvement=100,color=50,x=1,y=1,w=50,h=50) Cette exemple donne un obstacle, bleu foncer, qui se mouvois dans le temps de 100 mouvements en direction de sa courbe. Cette exemple est dans plaine.pfag.
-
- De plus des modifications mineurs ont été effectuer
    - L'UML est à jour, j'ai tenté de le rendre le plus joli possible, avec des couleurs pour chaque pattern, en séparant le contexte du menu qui est hors sujet au reste, mais en l'incluant tout de même, une classe reste manquante c'est Calcul, qui a été utile qu'aprés l'implémentation du drag'n'drop et donc aprés la création de l'UML.
    - Pour info, Calcul est une classe Singleton qui utilise ses propres constantes au même titre que Constante mais dans un but calculatoire et non de stockage.
    - Le .jar exécutable fonctionne, relativement bien, malgrés le fait d'utiliser des ressources externes et que les chemins soit TOUS correctes, l'affichage des images ne fonctionne pas...
    - Des tests sont à venir...
    - Un scénario de jeu pourrait être fait si le temps nous en donne l'occasion.
    - Des bugs restent à corriger pour la version 0.8, il n'y aura que ça pour la prochaine version qui sera peut être l'avant dernière du Jalon2, il est prévu d'entrer dans la version 1.0 au début du Jalon 3 si tout se passe bien.

A propos des patrons de conception et du MVC :

- Le format MVC n'a pas vraiment changé depuis la version 0.3 (pré Jalon2), le tout reste le même, à part certains points. L'architectrure en général a était modifier, le package "modele", qui contient les classes de calculs a été créé, il contient Calcul, Matrice, Courbe et PFAGReader et aussi ImageCustomz qui est lié entre le PFAGReader et le visu.

Pour le reste, Visualisateur s'occupe toujours de la partie visuel du jeu, j'ai tenté de créé une méthode prenant une liste d'énumération pour créer un visu capable d'afficher réellement se que l'on veut mais ça n'a pas fonctionné

Le HeartCore s'occupe de la partie calcul, entre chaque frame des choses bougent dans ce monde, et lorsque l'on doit voir que ça bouge, HeartCore nous le montre, HeartCore tente de montrer environs 60 images par seconde (fps) avec un sleep(16), c'est pas trés jolie certe... HeartCore n'a aucun lien avec le visu, il appel juste la méthode repaint du Panel central qui fait appel au visu lui.

La parte évenementiel est toujours géré par Collision et ses copains HitBox et Entity, l'Entity contient une HitBox à sa forme (plus ou moins) et lorsque un Bird rencontre un Obstacle, Collision le détecte. Collision est devenu un thread depuis le temps, se qui fait que le jeu tourne sur deux threads, mais il n'a pas de boucle, Collision run lors d'un repaint du Panel central, donc sur ordre de HeartCore indirectement entre deux Frame, si une collision est détecter par contre Collision stop HeartCore et fini le jeu 2000ms aprés l'impact. (Cela changement dans le Jalon 3 avec les rebons..; moteur physique etc).

Quant aux design patterns,

- Le jeu concentre un peu tout se qui doit être partagé entre deux classes, modifier par plusieurs classe ou normalisé à l'éxécution du jeu (nouveau Bird, nouveau PFAG etc) dans Constante. Constante ne sert qu'à stocker des variables static, il contient deux méthodes dont une hors contexte, l'autre iniz() sert à remettre le jeu à 0.
-
- La classe Calcul est aussi un singleton du même type que Constante mais plus orienté calcul, elle n'est pas dans l'UML final du Jalon 2 mais sera plus imposante dans le futur avec le moteur phyique.
-
- Le jeu contient (il n'y a que ça au final) des abstracts factory, l'entité représenté par Entity défini tout les éléments du jeu, à l'aide des classes Hitbox et Skin (pattern de façade ?) elle représente une entité du jeu virtuellement et réellement, ses fille Bird et Obstacle sont eux aussi des abstract factory.
-
- Le jeu contient aussi un decorator, Bird a une liste de moduleBird dans ses paramètres, on peut afficher un Bird sans module (ça sera moche, juste un rond) puis ont peut rajouter des modules, Bec, Oeil etc... (tous sont tiré de l'abstract ModuleBird) cela changera le skin de l'oiseau.
- J'ai tenté de faire en sorte que la création d'un Bird soit relativement simple, et cette stratégie est bien pratique parce que par exemple l'oeil est réutilisé dans le rouge gorge et aussi dans le canard, au final si le jeu prend de l'ampleur on pourra faire des osieau à partir d'autres oiseaux (mais il ne prendra pas de l'ampleur).
-
- Des prototypes ont été crée dans le menu afin de "pimper" certaines node comme les boutons et les séparateurs.
-

- Les façades, représentent une grosse moitié des pattern, si l'autre est l'abstract facotry, deux façades sont dans l'architecture, la première est celle de Entity, mélange entre Courbe (model), HitBox (controler) et Skin (vue).
- La deuxième façade est celle de animationJeu qui réuni HeartCore (model), Collision (controler) et Visualisateur (vue).
- Ces deux façades sont intiments liés, visu dessine les skins, collision gère les hitbox et heartcore fait avancer les courbes grâce au temps.

L'UML au complet, je vous conseille d'aller voir les screeshot plus détaillés dans le dossier UML>Livrable2, lUML a été fait avec BoUML, BoUML Viewer peut être gratuitement téléchargé pour voir l'UML encore plus en détail, si non il est en salle TP, 4e étage de l'IUT.

Legende :

Jaune : Rien de spécial.
Rose : Prototype.
Vert : Façade.
Rouge : Abstract factory.
Jaune : Singleton.
Bleu : Decorator.

La partie encadrée en haut représente le menu, le reste c'est le jeu

A propos du travail :

Comme au Jalon 1, le langage Java a été utilisé, sauf que cette fois une note importante est à prendre en compte, l'utilisation de fonctionnalités moderne ont été faites dans ce nouveau Jalon tel que nio2 et JavaFX, donc si vous n'avez pas de librairie Java 8 la compilation du projet vous sera impossible, JDK 1.8 minimum. Un peu de CSS viens se rajouter en parralèle au JavaFX, et le langage PFAG qui permet la création de map relativement simplement. Un luncher en bash ou C devait se faire à cause du problème du .jar qui compressais les fichiers, mais au final il n'a pas été fait.

La mise en commun du travail s'est fait du GitHub, depuis le Jalon 1 >*https://github.com/svevia/AngryBirdsS3K5*

La communication s'est beaucoups faite sur Facebook et Skype, plus de travail personnel chez soi qu'en salle TP, l'UML a était fait en salle TP vue que BoUML est un logiciel payant et disponible à l'IUT. Mais vue que le projet est en 1.8 et que les salles TP sont en 1.7 ! Nous avons plus travaillé sur nos machines qu'autre chose.

Le boulot était trés souvent centralisé sur une machine pour éviter les problèmes d'intégrations, par exemple un projet Drag'n'drop est créé pour étudier cette chose, une fois que la fonctionnalité fonctionne en externe, elle est envoyer chez l'intégrateur qui... l'intégre. Donc le plus souvent c'est Wissam qui intégra les projets.

Au niveau de la répartition du boulot :

L'UML a était fait par Wissam et Arthur.
Ce dossier a était fait par Wissam et Ali.
Le menu a était fait par Wissam et Aurélien.
Le déplacement des obstacles a était fait par Arthur et Aurélien
Le drag'n'drop a était fait par Wissam et Ali

Puis le reste des fonctionnalités a été souvent fait en binôme, Wissam et Ali d'un coté et Aurélien et Arthur de l'autre coté.

Des tentatives parfois non intégrés sont a prendre en compte malgrés leur abscense, Arthur a voulu utiliser les matrices pour bouger les entités, Ali a voulu créé un installateur du jeu, des modifications de structure ont été faites puis abandonné par l'ensemble du groupe.

Le travail s'est répartie sur tout les deux mois, bien qu'on soit assomé de DS en semaine, on a réussi à rendre quelque chose qui j'espère vous plaira.

Les tests :

Les tests ont était fait sur la plupart des classes gérant la calcul, donc le classe Calcul, Entity et sa façade arrière (Courbe et HitBox, il n'y a pas de calcul dans Skin) et les méthodes de move.
/!\ Pour plus d'information, tout les tests sont commenté afin de les comprendre.

Les tests de Calcul :

- testForce() : Le test consiste à créé un environnement ou l'utilisateur aurait cliqué et aurait drag sa souris sans encore drop, les tests testent à peu prés toute les situations possible.
- TestAngle() : Comme le test de force, l'angle à partir d'un environnement de drag sans drop, calcul l'angle actuel, le test teste quelques angles essentiel.

Les tests de HitBox :

- testCompareTo() : Le test compare deux hitboxs dans différentes situation, coller, éloigner, égales...
- testSetPosition() : Test si la méthode setPosition met bien à jour les deux position x et y avec des get x et y.
- TestToString() : Vérifie si le string returné correspond bien au valeur donné.

Sources :

La musique du menu : Calypso Medley - Trinidad&Tobago - Steel drums, lien youtube : https://youtu.be/q5yXCDw427w

L'ensemble des images du jeu ont été tiré sur google et modifié par Wissam avec les logiciels Paint et Photofiltre (parce que paint gère pas la transparance...), ainsi que le langage PFAG.

Rf fichier COPYRIGHT des Ressources pour plus d'information.