DESCHAINTRE Valentin
LAURANS Barthelemy
TEYSSIER Loïc
MARCH Benjamin

LO43

Rapport de Projet : AngryUTBM

A2011

Sommaire

I] Présentation du sujet et cahier des charges

Surfant sur la vague Angry Birds, nous vous proposons de réaliser un jeu similaire **AngryUTBM**. L'objectif de ce jeu est simple et ludique : un ensemble d'oiseaux « en colère » est disponible pour tacher les vestes des passants. Le joueur a au départ entre 3 à 5 oiseaux, selon la difficulté du niveau. Chaque type d'oiseau possède des capacités de ponte et un temps de vol qui lui est propre. Pour chaque personne touchée, des points seront attribués au joueur. Dans le cas où toutes les personnes sont touchées, le niveau suivant peut être atteint.

Le cahier des charges nous a été défini ainsi :

Environnement : l'environnement du jeu est un terrain ayant des caractéristiques différentes en fonction de la difficulté. Il peut y avoir des cavernes, ponts, …

- **Personnes :** les personnages sont des entités générées aléatoirement de 1 à 10 suivant le niveau de difficulté. Chacune de ces entités ont des propriétés de déplacement différente (vitesse, direction,...). Elles se déplacent de manière aléatoire dans l'environnement en suivant le sol de l'environnement. Leur déplacement est géré par run thread propre.

- **Oiseaux en colères :** Les oiseaux sont de 3 types différents au moins :
    - Type moineaux : ces oiseaux peuvent voler sans réaliser de vol stationnaire, peu longtemps et ont une seule possibilité de ponte d'œuf de calibre moyen.
    - Type colibris : ces oiseaux peuvent voler longtemps, même en stationnaire mais n'ont qu'une seul possibilité de ponte de petit calibre.
    - Type pigeon : ces oiseaux peuvent voler longtemps, en stationnaire et réaliser trois pontes de gros calibre.

A partir de ce cahier des charges, il est demandé de réaliser un jeu simple et ludique. L'interface graphique, les commandes et les options ne sont pas détaillées afin vous laisser une latitude dans la réalisation.

Le joueur pourra sauvegarder sa partie et la recommencer au niveau ou à l'instant où il

s'est arrêté.

II] UML

1) Diagramme de cas d'utilisation

La première étape de la conception du jeu fut d'abord de réfléchir à ce que nous voulions que le jeu fasse, et comment le joueur interagirait avec celui-ci.

Ici, le scénario central se joue autour d'un niveau. Dans tous les cas, il y a certaines actions qui sont nécessaires au bon fonctionnement du niveau, comme le lancement d'un oiseau, la ponte d'un œuf ou l'arrêt /reprise du vol stationnaire.

Il existe également des fonctions indispensables au bon déroulement du jeu plus globalement. Les chargements de niveaux par exemple, la gestion de la perte ou de victoire également. Nous avons pensé judicieux de faire un système de pause pour un plus grand confort du joueur, afin de pouvoir aisément changer de niveau, quitter le jeu, faire une pause puis reprendre.... De plus, comme il a été demandé dans le cahier des charges, le joueur peut enregistrer sa progression. Tout ceci nous a conduits à l'élaboration du diagramme de cas d'utilisation disponible en Annexe 1.

2) Diagrammes de séquence

L'étape suivante fut la réalisation du diagramme de séquence qui nous a permis de nous intéresser sur comment devra réagir le jeu en fonction des différentes demandes de l'utilisateur, et quelles seront les étapes que cet utilisateur devra réaliser afin de réaliser l'action qu'il souhaite effectuer.

Le démarrage du jeu est centré sur un écran d'accueil proposant les boutons « Commencer » et « Quitter ».
Pour choisir son niveau, puis commencer une partie, le joueur devra cliquer sur « Commencer », puis cliquer sur le niveau de son choix.
Pour Sauvegarder, il devra mettre le jeu en pause, puis soit cliquer sur sauvegarder, soit quitter le jeu et répondre oui au message d'alerte.
Lorsque le joueur gagne ou perd, un écran correspondant est affiché, lui permettant de changer de niveau, recommencer, ou quitter le jeu.
Pour charger une partie, le joueur doit commencer une partie normalement, et si il sélectionne un niveau comportant une sauvegarde, l'application lui demandera s'il veut charger la partie ou non.
Les diagrammes de séquence sont disponibles en Annexe 2.

3) Diagramme de classes

Le prochain gros travail était de concevoir un diagramme de classes, pour fournir une structure squelettique au programme ainsi que définir les règles d'héritage et d'implémentation des différentes classes entre-elles. Le diagramme de classes est une partie indispensable de tout bon projet de conception logiciel, et nous avons pris le temps d'en établir un très complet avant d'entreprendre le codage en lui-même. La fonction d'exportation du logiciel de création du diagramme des classes a aussi été d'une grande aide quant à l'élaboration du squelette du projet. Le diagramme complet tel qu'il a été finalisé est disponible en annexe 3.

Les « ItemsDisplay »

Nous avons utilisé une structure basée autour d'un type central appelé « ItemDisplay », c'est à dire un item affichable quelconque. La plupart des objets du programme héritent de cette classe, ou sont implémentées par elle et ses descendants, et donc ceci a permis une grande facilité au niveau des acteurs du jeu, comme les personnages, les oiseaux et les obstacles.

Les threads

Autour de cette classe centrale nous avons beaucoup pensé et réfléchi aux divers acteurs qui seraient responsables de la gestion du programme, comme les Threads, que nous avons décidé de séparer en trois catégories. Le thread de calcul de position permet de calculer les nouvelles positions des objets afin de créer une animation. Le thread d'affichage permet d'afficher tous les objets d'un niveau, à chaque rafraichissement il affiche la nouvelle position des objets (calculé par le thread de calcul de position). Le thread de gestion de menus permet l'affichage des différents menus.

Fonction d'écoute des périphériques

Afin de pouvoir gérer les périphériques nous devions implémenter les classes « Listener », qui sont chargées de l'écoute du clavier et de la souris, pour récupérer les actions du joueur sur les items de la fenêtre de jeu.

Pour ce faire nous avons utilisé l'interface « KeyListener ».

Enregistrement et chargement

Pour gérer les enregistrements de progression, le chargement de niveaux et la sauvegarde des scores, nous avons opté pour l'utilisation d'une classe DataBase qui gère des fichiers XML dans les dossiers du projet. L'avantage est quelle est relativement simple à utiliser, pour des fonctions plutôt complexes, et nous avons pensé que le langage structuré du XML était pratique pour stocker les informations nécessaires.

Menus

Une grande partie du programme a été les menus. Ceux-ci sont de différentes sortes selon leurs fonctions, et ont nécessité chacune une classe spécifique, et sont gérés par une classe particulière appelée « MenuSelector ».

Après ces quelques réflexions, nous nous sommes lancés, et avons fini par élaborer le diagramme de classes disponible en annexe 3. Il a été remanié légèrement par la suite, au fur et à mesure que diverses mises à jour du projet sont venus modifier quelques façons de faire, et que des difficultés non-envisagées se sont présentées

III] Programmation JAVA

1) La Fenêtre

La classe fenêtre comporte la classe main de notre programme. C'est dans cette classe qu'est créée notre fenêtre d'affichage.
L'affichage est gérer via trois listes contenant des « backgrounds », des objets mouvants et des obstacles. Une fonction boucle en permanence sur ces trois listes et affiches ce qu'il y a à afficher dans la fenêtre. Si un oiseau détruit un piéton, le piéton est retiré de la liste, il n'est par conséquent plus affiché.

C'est aussi la classe fenêtre qui contient la fonction qui lance les divers threads du programme.

2) Les objets du niveau

Les objets faisant partie du niveau à proprement parler sont tous hérités d'une classe abstraite ItemDisplay. Celle-ci a été pensée autour de la caractéristique principale d'être « affichable » dans la fenêtre du jeu. Une seconde caractéristique est d'être apte à détecter des collisions. Pour répondre à ces demandes, nous avons affecté à la classe les paramètres suivants, :

- Les coordonnées en x et en y
- Largeur, hauteur
- Texture, c'est à dire l'image qui va la représenter à l'écran
- Un rectangle permettant de détecter les collisions.

Les collisions

Pour ce qui est des collisions, nous avons utilisé un procédé relativement simple : nous avons approximé les objets par les plus petits rectangles les encadrant, simplifiant ainsi les tests de collision. Le rectangle, qui apparaît sous la forme d'un paramètre _r, est calculé par une fonction d'une bibliothèque java existante. Afin de compléter ceci, et de rendre possible la création de décoration et du fond d'écran du jeu, nous avons ajouté un booléen, qui définit l'objet comme

étant solide ou non S'il n'est pas solide, les fonctions de gestion de collisions ne s'appliqueront pas à lui, ce qui permet sa position en tant qu'image de fond ou de décoration.

Les objets mouvants

La classe Movingitems est une classe fille des « ItemDisplay ». Celle-ci est la classe parente des objets mouvants du niveau, à savoir les personnages représentant les cibles et les oiseaux. Elle dispose des fonctions move en x et en y, ainsi que des fonctions de collisions, avec les autres objets stationnaires ou dynamiques.

En cas de collision, on appelle la fonction de destruction de l'objet. Celle-ci lance la phase de suppression avec l'animation qui correspond, puis supprime, le ou les objets « collisionnés » de la liste des objets à afficher afin de libérer la place à l'écran au prochain rafraichissement.

Les oiseaux

Les oiseaux sont de trois types : Colibri, Pigeon et Moineau.

Il existe une classe parente « Birds » qui rassemble les caractéristiques et « effets » communs aux trois oiseaux. Chaque type correspond à une classe et a des caractéristiques diverses comme le temps de vol, le nombre d'œuf ou encore la possibilité d'effectuer un vol stationnaire ou non. Les oiseaux ont la possibilité de « pondre » des œufs sur les piétons en contrebas ou de se crasher dessus pour les faire disparaître

Les piétons

Les piétons sont de trois types également : Fin, Gros Moyen.

Il existe une classe parente « Walkers » qui rassemble, tout comme pour « Birds » les « effets » et caractéristiques communes aux trois piétons. Chaque type a une classe et des caractéristiques différentes.

Ici ce sont : la vitesse de marche et la taille qui diffèrent selon le piéton. Les piétons font demi-tour aléatoirement à chaque déplacement les chances sont de 2 pour 1000

Les œufs

Les œufs ont la même structure que les oiseaux et les piétons et ont des tailles différentes selon la classe d'œuf.

Ils « détruisent » les piétons lorsqu'ils sont touchés.

Les animations des objets

Les piétons et les oiseaux ont un jeu d'image complet leur permettant, en alternant entre ces images, de simuler le battement des ailes ou une marche à pied. Ce jeu d'image change dans le cas des oiseaux qui se détruisent pour produire une disparition « fluide ».

Les images des objets

Afin de trouver les images de chaque objets et leur jeu de mouvements nous avons cherché et découpé des feuilles de « sprites » trouvées sur internet, c'est pourquoi certains piétons ou oiseaux peuvent rappeler d'autres jeux.

Les obstacles

Héritant également d' « ItemDisplay » nous avons la classe obstacle.
Cette classe désigne des objets non mobiles, mais sensible à la collision, c'est-à-dire qu'un oiseau qui rencontrera un obstacle se « crashera » et un piéton fera demi-tour.
Il est possible de donner différentes texture à un obstacle ainsi qu'une taille correspondante, c'est ce qui est fait lors de la création de niveaux.

3) Les menus

Les menus sont une partie intégrante du jeu. Ceux-ci servent à le lancer, gérer le passage de niveaux, mettre en pause... etc. Les commandes y sont envoyées à l'aide de boutons que nous avons personnalisés à partir de la classe JButton d'une bibliothèque existante.

Les menus sont de plusieurs sortes, pour les implémenter de façon satisfaisante, nous avons créé une classe parente « ZoneAff » qui définit une zone d'affichage. Le panneau de jeu hérite aussi de cette classe.

Le premier menu est celui qui s'affiche lorsqu'on lance le jeu. Celui-ci ne dispose que de deux boutons, permettant la sélection du niveau ou de quitter le jeu.

Le second menu permet la sélection de tous les niveaux débloqués, et informe le joueur de leurs « highscores ». Un clic sur le bouton du niveau charge celui-ci. La fonction de génération de ce menu comporte une partie de lecture de fichiers, afin de déterminer le nombre de « niveaux » à afficher. Cette structure rend très simple l'ajout ultérieur de niveaux au jeu, quels qu'ils soient.

Le troisième menu, le menu de pause, est lancé quand le joueur appuie sur Echap au cours d'une partie. Il informe le joueur de son score actuel, du highscore sur le niveau, en lui

permettant de recommencer le niveau, l'abandonner, le sauvegarder ou de reprendre le jeu.

Le quatrième menu s'affiche lorsque le joueur gagne le niveau. Il permet le passage au niveau suivant, ou de rejouer ce niveau si pour tenter un highscore plus élevé. Sinon, le joueur peut aussi choisir un autre niveau parmi ceux déjà débloqués. Le fait de réussir un niveau débloque le suivant dans la base de données des niveaux, même si le joueur ne choisit pas d'y jouer de suite, il peut aussi quitter le jeu.

Le cinquième menu est affiché lorsque le joueur échoue au niveau. Il dispose des mêmes fonctionnalités que le menu de victoire, sauf qu'il ne débloque pas le niveau suivant.

Enfin, nous avons fait un petit menu final, affichant les crédits du jeu lorsque le joueur termine tous les niveaux.

La gestion des menus se fait par un thread dédié. Celui-ci intercepte les actions de lancement et de quitte des menus.

4) Les niveaux

V] Conclusion

Ce projet nous as permis de faire un premier vrai travail de groupe. On a donc dû utilisé GIT qui nous a permis de travailler tous en même temps sur le projet. Nous avons aussi pu aiguiser notre connaissance en JAVA qui était pour la plupart d'entre nous inconnu jusqu'à lors.
Bien que nous ayons rencontré quelques problèmes tels que des boutons qui ne s'affichent pas ou les touches du clavier qui n'étaient pas capturé, nous avons réussi à mener ce projet là où nous le voulions dans le temps impartit.

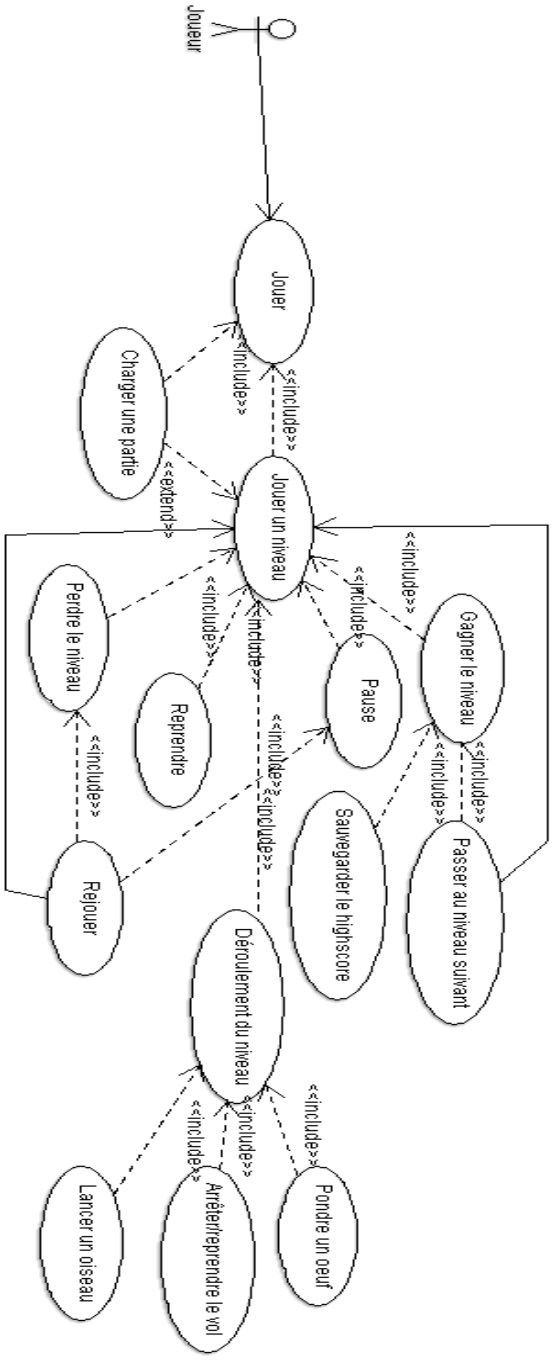
Nous avons cependant pensé à plusieurs amélioration tel qu'un meilleur calcul des scores, une définition des objets en tant que polygones et non en tant que rectangles afin d'augmenter la précision des collisions ou encore une meilleure équation de décollage permettant une courbe plus fluide dans le vol des oiseaux. Nous avions aussi pensé à intégrer une musique d'ambiance ainsi que des bruitages même si nous n'avons pas eu le temps. Nous avons écrit nos classes de manière à ce que chaque type d'objet qui se déplace puisse se déplacer et se détruire à sa propre façon, nous avons cependant groupé tout ceci afin de nous simplifier un peu la tâche.

Pour finir nous pouvons dire que ce projet fut très intéressant car plus qu'une simple application

de nos connaissances dans un langage donné, c’est une manière de travailler en groupe que nous avons appris.

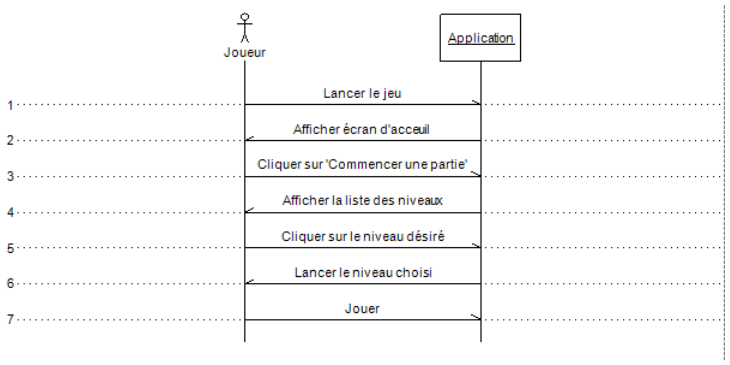
Annexes

Annexe 1 :

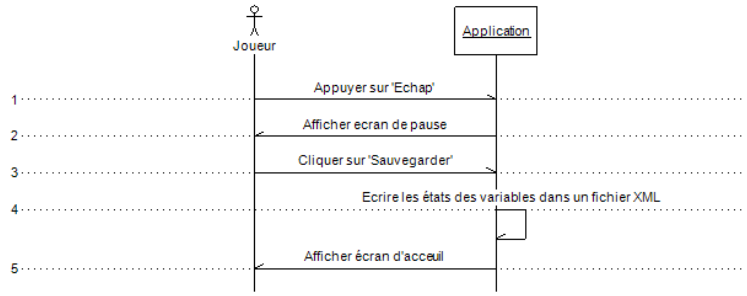


Annexe 2 :

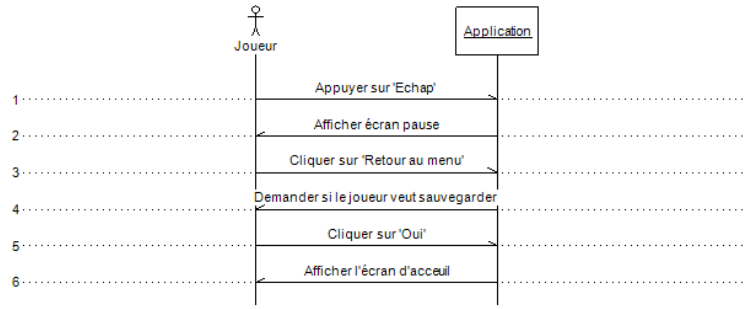
Commencer une partie :



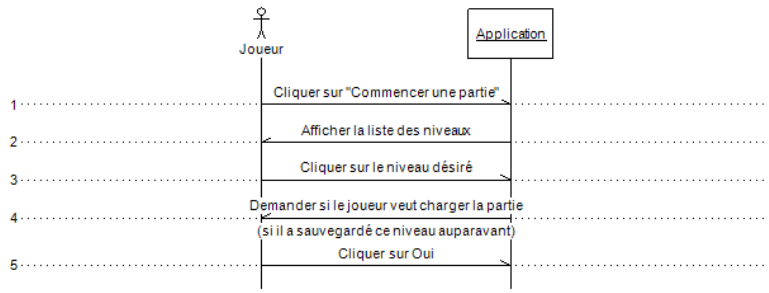
Sauvegarder la partie en cours (1/2) :



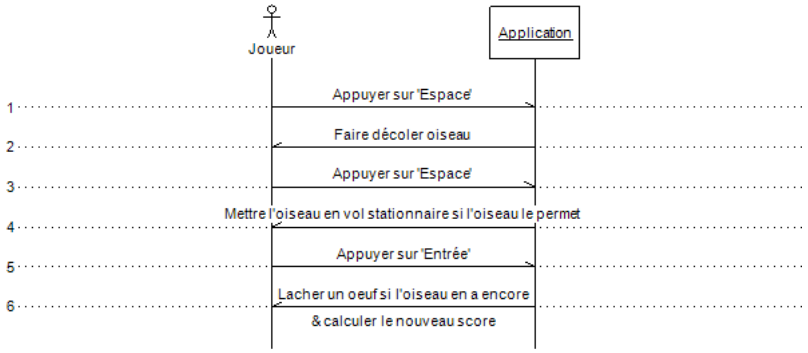
Sauvegarder la partie en cours (2/2) :



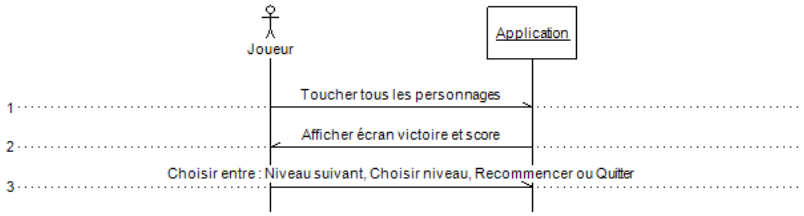
Charger une partie :



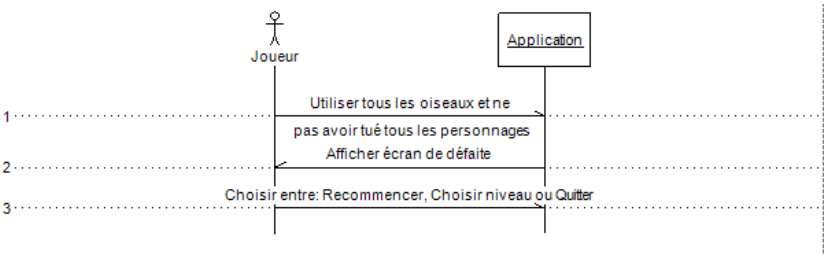
Jouer :



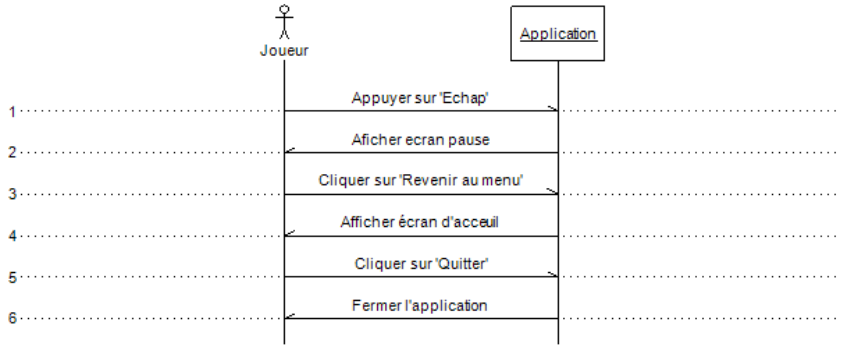
Gagner :



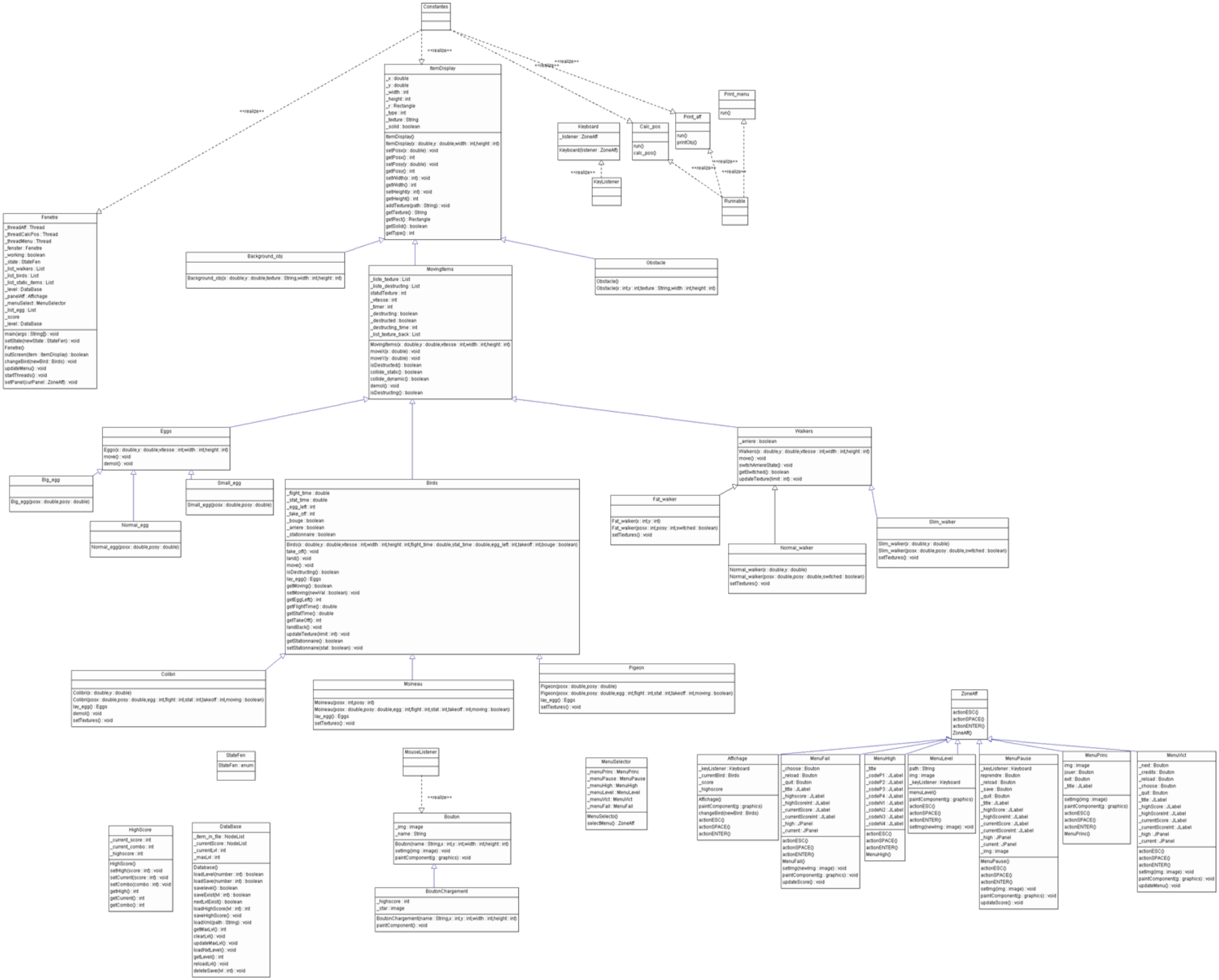
Perdre :



Quitter :



Annexe 3 :


Constantes
ItemDisplay
Fenetre
Background_obj
MovingItems
Obstacle
Keyboard
Calc_pos
Print_aff
Print_menu
KeyListener
Runnable
Eggs
Big_egg
Small_egg
Normal_egg
Birds
Walkers
Fat_walker
Normal_walker
Slim_walker
Colibri
Moineau
Pigeon
StateFen
MouseListener
Bouton
BoutonChargement
MenuSelector
HighScore
DataBase
ZoneAff
Affichage
MenuFail
MenuHigh
MenuLevel
MenuPause
MenuPrinc
MenuVict