جامعة عبد المالك السعدي
Université Abdelmalek Essaadi

Rapport Final de projet : JetPlane Game

Wafdi Ahmed,
Saoudi Anas,

sous la direction de monsieur
Pr. Jaber El bouhdidi.

المدرسة الوطنية للعلوم التطبيقية بتطوان
L'École Nationale des Sciences Appliquées de Tétouan

Année universitaire : 2016 - 2017

Master SEM

Table des matières

Table des figures

Introduction

1. Définition du projet

a. Énoncé

Le projet dont nous sommes en charge consiste à réaliser un jeu en java. Ce jeu sera simplifié par rapport aux règles du jeu normal, car il doit répondre aux contraintes du sujet. En effet, le but de ce projet est de mettre en exergue les principes fondamentaux de **la programmation objet**, c'est à dire l'**héritage** entre les classes, la gestion des **interfaces**, le **polymorphisme**, les **Threads**, etc.

Pour bien répondre au problème, nous nous inspirerons du cycle général de la conception objet que nous avons vu en cours. Cette méthode de conception nous permettra d'avoir une présentation claire et structurée de notre projet.

b. Plan

Nous présentons dans ce document la démarche que nous avons suivi pour tenter de mener à bien ce projet, en décrivant les deux grands axes de développement, c'est-à-dire la conception et l'analyse du projet avec diagrammes UML à l'appui, la réalisation technique (choix des outils, portions de code, modèle utilisé), notre avis sur les résultats obtenus et les objectifs initiaux, puis une conclusion personnelle sur ce projet.

c. Présentation générale

Le jeu commence par une interface simple dans laquelle le joueur est invité à créer son nom ou choisir son nom s'il est un ancien joueur. Après il choisit l'avion et le niveau souhaité. Ensuite la partie commence, le joueur déplace l'avion à droite à gauche an avant et en arrière pour éviter les ennemis et leurs missiles, le tir des missiles se fait par la touche 'espace'. Un tableau de score est placé en haut à droite qui affiche le score, niveau, et le nombre des vies restantes, ces derniers sont aussi représentés sous forme d'images en bas à gauche. Le niveau change en fonction du score atteint, si le score $\geq$ 1000 le niveau passe de 1 à 2 et les ennemis s'augmente avec une vitesse plus élevée, et ainsi de suite (niveau 3 score $\geq$ 2000 ...etc.) jusqu'au dernier niveau ou l'ennemi boss apparu. D'une manière aléatoire une sorte d'aide apparue sous forme PowerUps qui peut être soit tir des doubles missiles, triples missiles, protection contre les missiles et les ennemis, ajout de vies.

2. Cahier des charges

Notre programme doit offrir ces fonctionnalités :

Gestion du profil : si le joueur a déjà joué sur le jeu, il choisit son nom et l'application se charge de lui créer une nouvelle partie, sinon il choisit un nom et le joueur va être enregistré sur la base de données.
Traçabilité : l'ensemble des joueurs ainsi que leurs parties et le meilleur score atteint dans chaque partie.
GameBoard : après la sélection du nom du jouer, ce dernier doit choisir l'avion (attaque, agilité, nombre de vie) ainsi que le niveau (intermédiaire, haut niveau ...).
Avions Ennemis : ça dépend du niveau du joueur, on affiche des avions ennemis différents (vitesse, type d'ennemie).
Tir des missiles : l'avion doit tirer des missiles avers les ennemis.
Détection des collisions : si l'avion ennemi (respectivement le missile de l'avion ennemie) à toucher notre avion on doit diminuer le nombre de vie restant (après trois frappes).
InfoBoard : on doit afficher le nom du joueur, son score, niveau, ainsi que le nombre de vie restant.
PowerUps : un powerUps s'affiche à chaque fois d'une manière aléatoire qui peut être soit un ajout de vie, tir d'une double missile, tir d'une triple missile, protection contre les ennemis.
SFX : le tir des missiles, PowerUps, le passage d'un niveau à un autre est caractérisé par un effet sonore spécial.
Performances : nous avons essayé que le jeu soit le plus performant possible en niveau de consommation mémoire ainsi que la rapidité du jeu (Threads).

Analyse et conception

3. Analyse du travail réalisé

Un des objectifs principaux de ce projet a été de nous habituer au travail en groupe et aux notions de projet. Organiser son temps, établir les tâches ou encore nous obliger à respecter notre cahier de charge. Nous avons dans un premier temps travaillé ensemble sur les premières classes à établir (notre premier diagramme de classe en UML) afin d'établir un socle commun au sein de notre groupe en codant en dure.

4. Diagrammes UML

Après plusieurs premiers tests et après avoir imaginé de nombreux cas de figures, nous sommes arrivés à une configuration du jeu à travers deux schémas :

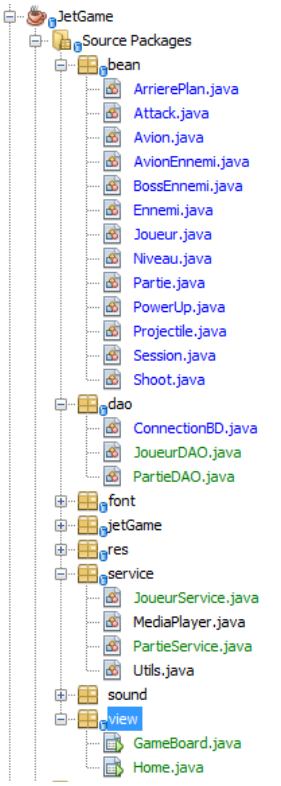
a. Classes (Structure du projet)

Voir **Annexe 1**.

Pour organiser le projet de sorte qu'on aura une structure bien défini on a implémenté le patron de conception **MVC** (**M**odel **V**iew **C**ontroller) afin de séparer l'affichage des informations, les actions de l'utilisateur et l'accès aux données.

- **bean :** les classes.
- **dao :** la connexion et l'interaction avec la base de données.
- **res :** les différents images (background, avion, missiles..).
- **service :** concernant le traitement de la musique et la liaison entre la couche view et dao.
- **view** : l'interaction avec l'utilisateur (création du joueur, choix d'avion et niveau ..).
- **sound :** les différents morceaux de musique utilisés.
- **font :** font externe pour l'affichage du score (Digital Font).

Figure 1 packages du projet



Réalisation

5. Outils et Langage

a. Bibliothèques Graphiques

AWT vs SWING ?

Nous allons, dans cette section, nous intéresser à la mise en œuvre d'interfaces graphiques en Java. En effet Java nous permet de créer toutes sortes d'interfaces, et d'y inclure tous types de composants : des boutons, des cases à cocher, des zones de saisie de texte, des images ...etc., avec lesquelles on peut faire pas mal de choses à savoir un jeu. Mais avant d'aller plus loin il nous faut être capable de faire un choix. En effet, deux API de mise en œuvre d'interfaces graphiques sont proposées par l'environnement Java **SE** (**S**tandard **E**dition) : l'**AWT** (**A**bstract **W**indow **T**oolkit) et **SWING**.

- Le principal avantage à utiliser l'AWT réside dans les performances en termes d'affichage.
- L'AWT ne contient que des composants graphiques qui existent sur tous les OS. Du coup, on peut affirmer que cette librairie est relativement pauvre.
- Dans AWT c'est que Java fait appel au système d'exploitation sous-jacent pour afficher les composants graphiques.
- Dans SWING ce n'est plus l'OS qui va pixéliser les différents éléments graphiques mais bien l'API Java Swing. Du coup, si un composant graphique n'est pas proposé par un système d'exploitation, il n'y a pas de soucis.
- Une application codée en SWING consommera plus de ressources (mémoire et CPU) qu'une application codée via l'AWT.

⇨ **AWT est adapté à nos besoin (performance du jeu élevé grâce à l'affichage rapide des composantes graphiques).**

b. Gestion de versions

La gestion de versions (en anglais version control ou revision control) consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers. Essentiellement utilisée dans le domaine de la création de logiciels, elle concerne surtout la gestion des codes source.

Dans notre cas on a utilisé git un logiciel de gestion de versions décentralisé qui permet :

- La gestion des branches. On peut travailler sur un projet en parallèle sans se marcher sur les pieds.
- Les algorithmes de fusion : quand un fichier a été modifié par plusieurs personnes en même temps, Git sait s'adapter et choisir un algorithme qui fusionne intelligemment les lignes du fichier qui ont été modifiées.
- Savoir qu'une fonction a été déplacée d'un fichier à un autre et par qui.
- Visualiser l'historique des validations.

⇨ **GIT permet de travailler en collaboration d'une manière efficace.**

c. Langage

Nous sommes invités à réaliser ce projet dans le cadre des études du module programmation orienté objet en java donc nous sommes obligés de réaliser le projet en java, qui offre malgré tout un certain nombre d'avantages :

- **Portable :** exécutable sur n'importe quel système, à condition d'avoir installé une JVM).
- **API :** Java offre un nombre d'API qui facilite la programmation et qui correspond à notre besoin.
- **Multi-plateforme** : Une application développer en java fonctionnera sous un système Unix, ou Windows ou n'importe quelle autre OS.

⇨ **Pour développer des jeux vidéo en tant que professionnel il faut opter pour le C++ comme langage de programmation.**

d. Plateforme de développement

NetBeans vs Eclipse ?

NetBeans est l'IDE supporté officiellement par Sun au contraire d'**Eclipse**. Donc quels sont les comparatifs entre les deux IDE :

- Eclipse est beaucoup plus utilisé (surtout dans les entreprises, ce qui est important).
- Eclipse propose plus de plugins (car il est plus utilisé).
- Eclipse a une meilleure gestion des raccourcie (pour gagner du temps).
- NetBeans est plus léger.
- NetBeans est plus rapide.
- NetBeans est plus agréable à l'utilisation.
- NetBeans a plus de plugins préinstallé (de fonction ou il faut un plugin sous Eclipse).

⇨ **Donc c'est une question de choix. Nous avons l'habitude de travailler avec NetBeans, donc pour l'IDE on a choisi NetBeans.**

6. Exemple du code

Nous n'avons pas jugé utile d'inclure l'intégralité de notre code dans ce rapport, mais nous décrirons seulement quelques extraits qui nous paraissent intéressants.

a. Création d'une partie

Une partie est créé en choisissant le nom du joueur ainsi que le niveau et l'avion souhaité. L'interface offre un **comboBox** dans laquelle on charge les listes des joueurs existe dans la base de données. Au cas où il a choisi **'Nouveau Joueur'** un **TextField** sera activé pour lui permettre de créer son nom. Sinon il choisit son nom (déjà existé sur la base) ou il aura la possibilité de jouer avec **DefaultPlayer.**

Code source :

- **Couche View :**

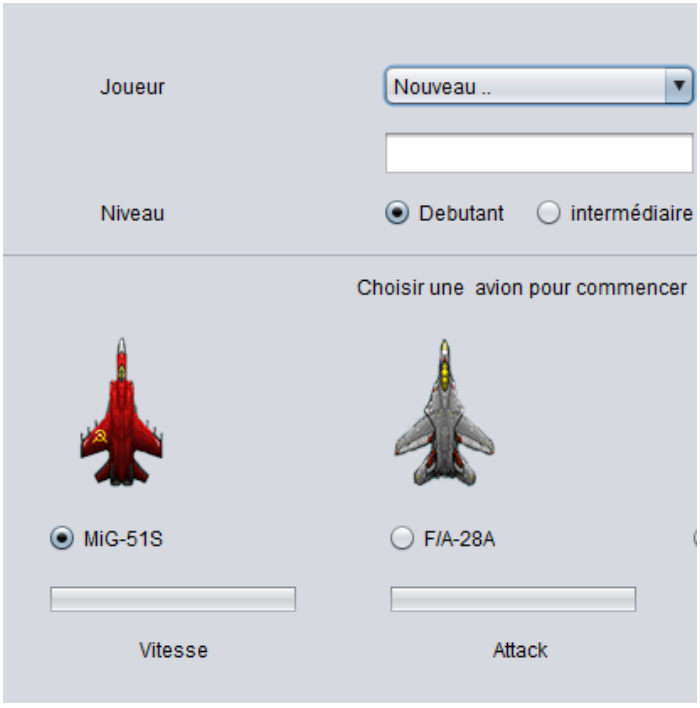


Figure 2 un nouveau joueur

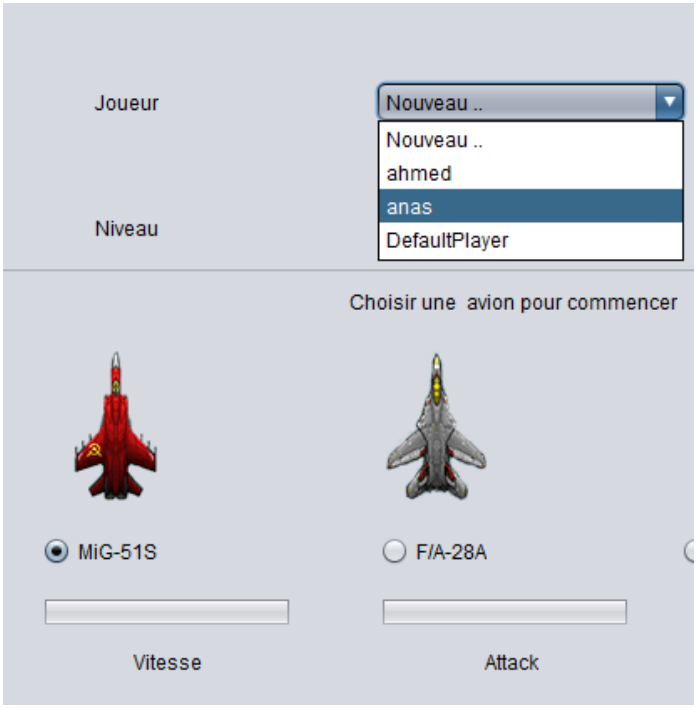


Figure 3 sélection d'un joueur

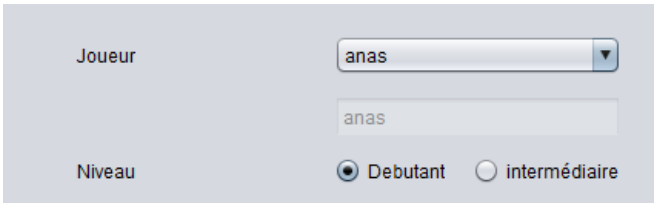


Figure 4 affectation du nom de joueur au TextField (désactivé)

Au niveau ComboBox il faut traiter l'activation et la désactivation du TextField en fonction de la sélection de '**Nouveau...**' ou **'Nom du Joueur'.**

```
private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (jComboBox1.getSelectedItem().equals("Nouveau ..")) {
        jTextField1.setText("");
        jTextField1.setEnabled(true);
    } else{
        jTextField1.setEnabled(false);
        jTextField1.setText(jComboBox1.getSelectedItem().toString());
    }
}
```

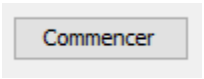
Figure 5 Traitement du TextField

La liste des joueurs déjà existé sur la base doit être chargé lors du chargement de l'interface. Donc dans le constructeur on fait appel à la méthode qui fait ce traitement.

```
public GameBoard() {

    initComponents();
    iniCombobox();

}
```

```
public void iniCombobox() {
    List<Joueur> joueurs = new ArrayList<>();
    try {
        joueurs = joueurService.findAll();
    } catch (SQLException ex) {
        Logger.getLogger(GameBoard.class.getName()).log(Level.SEVERE, null, ex);
    }
    if (joueurs != null) {
        for (Joueur joueur : joueurs) {
            jComboBox1.addItem(joueur.getNom());
        }
    } else {
        jComboBox1.addItem("--");
    }

}
```

Figure 6 Chargement de la liste des joueurs

Le vrai traitement commence en cliquant sur le bouton 


```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Joueur joueur = new Joueur();
        Partie p = getParams();
        joueur.setNom(jTextField1.getText());
        List<Joueur> joueurs = joueurService.findAll();
        if(joueurs!= null){
            for (Joueur joueur1 : joueurs) {
            if (joueur1.getNom().equals(joueur.getNom())) {
                    partieService.create(p);
                    break;
            }else if (!joueur1.getNom().equals(joueur.getNom())){
                partieService.create(p);
                joueurService.create(joueur);
            }
        }
        }
        Session.setAttributes(p, "partie");
```

Figure 7 Création du joueur et son partie

Dans un premier temps il faut charger la liste des joueurs existants dans la base de données, et pour respecter l'architecture MVC qui impose la séparation du traitement métier et l'affichage on crée un objet des classes **JoueurService** et **PartieService.** Ces derniers vont communiquer avec la couche DAO **(JoueurDAO, PartieDAO**) responsable de l'interaction avec la base de données.

Après importation de la liste des joueurs il faut créer la partie et le joueur en cas de 'Nouveau...' ou créer seulement la partie en cas de choix d'un joueur déjà existé. **(Figure 7).**

```
Session.setAttributes(p, "partie");
final JFrame frame = new JFrame("Java OpenStreetMap Editor");
StartingClass sc = new StartingClass();
sc.setStub(new AppletStub() {
    public void appletResize(int w, int h) {
        frame.setSize(w, h);
    }
    public AppletContext getAppletContext() {
        return null;
    }
    public URL getCodeBase() {
        try {
            return new File(".").toURI().toURL();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

Figure 8 Dispatching, Forwarding et Session

Le traitement qui suive consiste à mettre la partie courante dans la session pour qu'on puisse récupérer le nom du joueur dans la fenêtre du jeu (**Dispatching** et **Forwarding**). Après on invoque l'instance du class **StartingClass** qui contient le traitement du jeu (affichage de l'avion, les ennemies ...).

```
    public URL getDocumentBase() {
        return getCodeBase();
    }
    public String getParameter(String k) {
        return null;
    }
    public boolean isActive() {
        return true;
    }
});
setLocation(0, 0);
setSize(1360, 700);
sc.init();
sc.start();
add(sc);
remove(jPanel1);
```

➢ Couche Service :

Pour des raisons d'implémentation de l'architecture MVC la couche **view** ne communique jamais directement avec la couche **dao,** mais il faut passer par un pont c'est la couche **controller,** le controller est représenté dans notre projet par le package **service** qui sert à recevoir tous les événements de la vue et enclenche les actions à effectuer.

```
public class JoueurService {

    JoueurDAO joueurDAO = new JoueurDAO();

    public int create(Joueur joueur) throws SQLException {
        return joueurDAO.create(joueur);
    }

    public List<Joueur> findAll() throws SQLException{
        return joueurDAO.findAll();
    }

}
```

Figure 9 create() et findAll() du couche service

➢ Couche Dao :

```
public class JoueurDAO {

    public int create(Joueur joueur) throws SQLException {
        String requette = "insert into joueur values('" + joueur.getNom()+"')";
        System.out.println(requette);
        return ConnectionBD.exec(requette);
    }
```

```
public class ConnectionBD {
private String url = "jdbc:mysql://localhost:3306/bdjetgame";
private String user = "root";
private String passwd = "";
private Connection conn ;
private static ConnectionBD mycnx ;

public static ConnectionBD connectionFactory(){
if(mycnx == null )
mycnx =  new ConnectionBD();
return mycnx;
}

public static int exec(String req) throws SQLException {
Connection conn =  ConnectionBD.connectionFactory().getConn();
Statement state = (Statement) conn.createStatement();
return state.executeUpdate(req);// insert update delete
}
```

b. Déplacement de l'avion

Le déplacement se fait grâce aux touches flèche haut, bas, gauche, droite. Donc deux fonctions essentiellement s'intervient pour réaliser ce traitement.

```
@Override
public void keyPressed(KeyEvent e) {
    switch (e.getKeyCode()) { //if (e.getKeyCode()==KeyEvent.VK_UP)
        case KeyEvent.VK_UP:
            avion.up();
            avion.setDrawingimage(avion.getImageMoveUp());
            break;
        case KeyEvent.VK_DOWN:
            avion.setDrawingimage(avion.getImageMoveDown());
            avion.down();
            break;
        case KeyEvent.VK_LEFT:
            avion.setDrawingimage(avion.getImageMoveLeft());
            avion.moveLeft();
            break;
        case KeyEvent.VK_RIGHT:
            avion.setDrawingimage(avion.getImageMoveRight());
            avion.moveRight();
```

Figure 10 La méthode keyPressed()

```
@Override
public void keyReleased(KeyEvent e) {
    switch (e.getKeyCode()) { //if (e.getKeyCode()==KeyEvent.VK_UP)
        case KeyEvent.VK_UP:
            avion.stop();
            avion.setDrawingimage(avion.getImage());
            break;
        case KeyEvent.VK_DOWN:
            avion.stop();
            avion.setDrawingimage(avion.getImage());

            break;
        case KeyEvent.VK_LEFT:
            avion.stop();
            avion.setDrawingimage(avion.getImage());
```

Figure 11 La méthode keyReleased()

Les méthodes **up(), down()**, **moveLeft()**, **moveRight()**, **stop()** change les cordonnées de l'avion suivant les axes **Ox** ou **Oy** selon le déplacement.

```
public void up() {
    vitesseY = -globalSpeed;
}

public void down() {
    vitesseY = globalSpeed;
}
```

L'avion change de cordonnées en cas de déplacement, la méthode qui gérer ceci est la méthode **update()**. Avec un traitement de plus concernant les projectiles (**avion.getProjectiles().get(i).update()**) et le défilement de l'arrière-plan(**bg.update()**) et l'affichage des images avec des nouveau cordonnées (**repaint()**) on obtient un traitement représenté dans la méthode **whileAvionIsAlive()**.

```
public void whileAvionIsAlive() {
    niveauHaut();
    avion.update();
    if (Avion.projectiles.size() > 0) {
        for (int i = 0; i < Avion.projectiles.size(); i++) {
            avion.getProjectiles().get(i).update();
        }
    }
    bg1.update();
    bg2.update();
    updateShootEnnemie();
    repaint();
    attack.removeEnnemisOverLimitte();
}
```

Figure 12 Méthode whileAvionIsAlive()

Il faut toujours garder à l'esprit la notion de la performance de jeu, pour cela il faut penser à supprimer les ennemis de la liste quand ils dépassent l'écran (Stopper le Thread) ; c'est le rôle de la fonction **removeEnnemisOverLimitte()**.

c. Détection des collisions

L'idée c'est de dessiner un rectangle autour de chaque objet et vérifier après l'intersections des rectangles pour détecter une collision.

```
private Rectangle r = new Rectangle();
```

La taille du rectangle prend la taille de l'objet qu'on veut détecter sa collision mais les cordonnées dépend des cordonnées variés cad on les fixes dans la méthode **update().**

```
r.setBounds(centerX, centerY, 69, 60);
```

La méthode **checkCollision()** permet de gérer si une collision a été détecter ou pas.

```
public boolean checkCollision(Rectangle rect) {
    if (rect.intersects(r)) {
        return true;
    }
    return false;
}
```

Figure 13 Méthode checkCollision()

d. L'explosion des ennemis

Quand un projectile frappe un ennemi, ce dernier doit s'explosé et supprimer de la liste des avions ennemis.

```
public void destroy() {
    synchronized (moveAvionEnnemi) {
        this.setDetruit(true);
        MediaPlayer.playSound("/sound/Explosion.wav");
        Image im = toolkit.getImage("src/res/explode.gif");
        this.setImage(im);
        try {
            moveAvionEnnemi.wait(150);
        } catch (InterruptedException ex) {
            Logger.getLogger(AvionEnnemi.class.getName()).log(Level.SEVERE, null, ex);
        }
        Attack.avionEnnemis.remove(this);
        moveAvionEnnemi.stop();
        System.out.println("enemie Stooped");
    }
}
```

Figure 14 Méthode destroy()

e. Des ennemis vous suivent

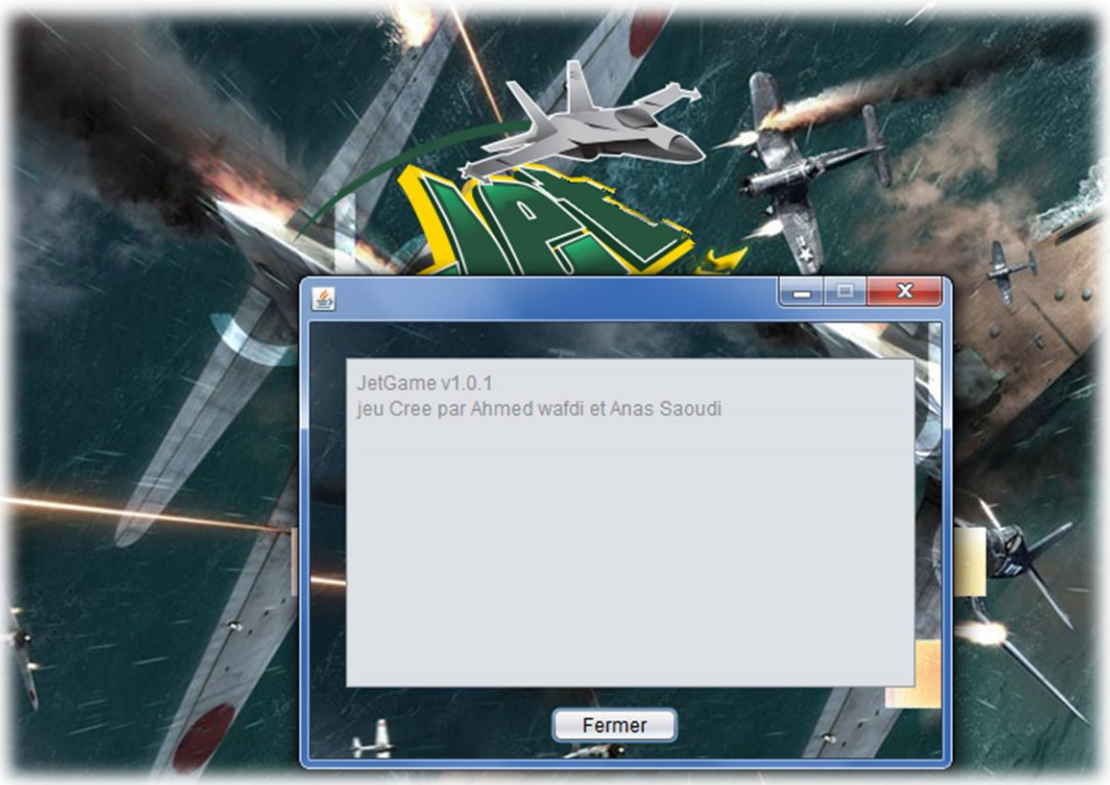
A partir du niveau 3 les ennemis suivent l'avion.

```
public void follow() {
    if (centerY < StartingClass.avion.getCenterY()) {
        if (centerX < StartingClass.avion.getCenterX()) {
            vitesseX = 3;
        } else if (centerX > StartingClass.avion.getCenterX()) {
            vitesseX = -3;
        } else if (centerX == StartingClass.avion.getCenterX()) {
            vitesseX = 0;
        }
    }
}
```

Figure 15 Méthode follow()

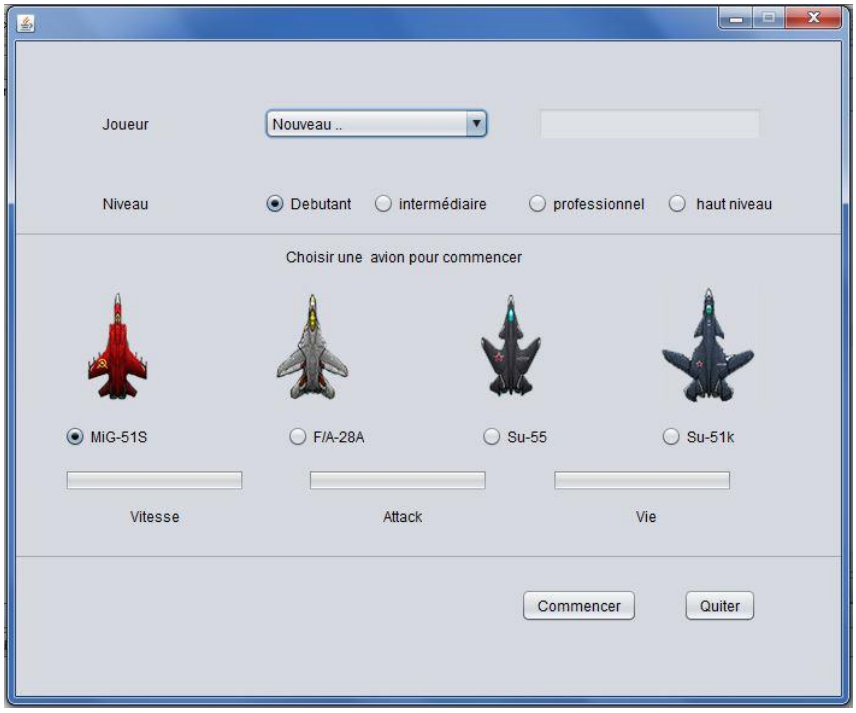
- **Le développement est géant (en termes de traitement) et on n'a pas pu intégrer tout le traitement fait dans le projet parce que chaque fonction dépend d'un grand nombre d'autres fonctions ce qui rend l'intégration de l'explication dans le rapport un peu délicat.**

7. Résultat Final



Le jeu commence par une interface simple de choix de l'avion et le nom du joueur ainsi que le niveau.

Figure 16 Interface GameBoard



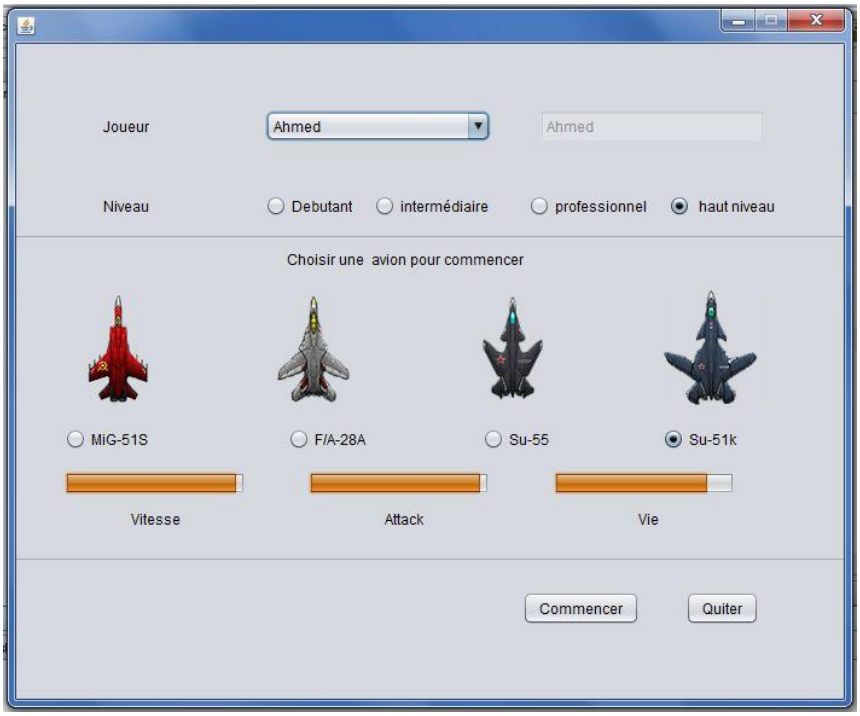


Figure 17 GameBoard avec des animations

L’interface de jeu se caractérise par un InfoBoard en haut à droite sur lequel on affiche le score le nom du joueur et le niveau.

En Bas à gauche le nombre de vie restant se représente par un nombre d‘images de l’avion.

Figure 18 Niveau 1

En cas d’explosion de l’ennemie une animation se produit.

Figure 19 Explosion de l'ennemie

A partir du niveau 3 les ennemis commencent à tirer des projectiles et leurs vitesses de déplacement s'augmente.

Figure 20 Tir des projectiles par l'ennemie

L'apparition de l'**ennemiBoss** se fait petit à petit avec un affichage d'un bar en bas à droite qui affiche **Health** de l'ennemiBoss.

Figure 21 Entrée du BossEnnemie

L’explosion de l’**ennemieBoss :**

Figure 22 Explosion de l'ennemieBoss

Les **PowerUps :**

Quand la partie est finie un **GameOver** s'affiche sur l'écran avec une petit fenêtre qui indique le score atteint et le nom du joueur.

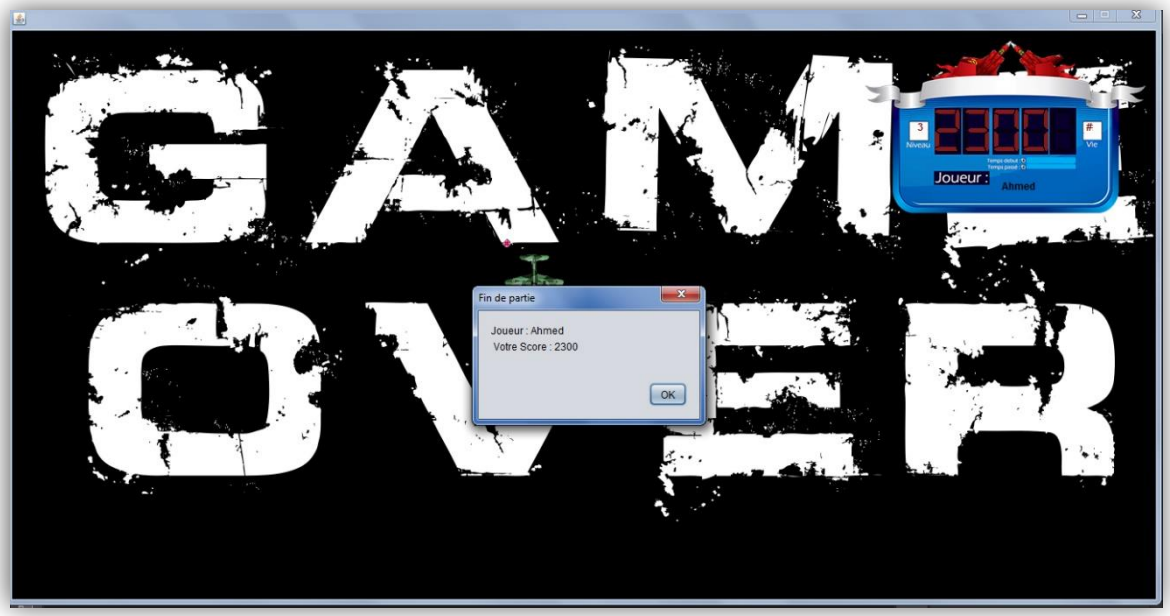


Figure 23 GameOver

Figure 24 Score Final

Conclusion

8. Apport du projet

a. Travail en groupe

Une des plus grosses difficultés tout au long du projet reste à nos yeux le travail en groupe. Parce que nous possédons des aptitudes déférentes, des intérêts différents pour la conception, la programmation ou encore la rédaction, il a fallu s'organiser dès le début pour déléguer les tâches à chacun et définir précisément la politique de développement (concepts et outils utilisés, style du code, etc.) pour que la mise en commun se fasse le plus simplement possible. Nous avons tous les deux encourager l'autre dans les moments difficiles (compilations échoués, erreurs mystérieuses), et partager nos réflexions sur le développement du projet.

a. Outils manipulés

Ce projet a aussi été l'occasion pour nous d''enrichir nos connaissances dans le langage Java, de nous familiariser avec la programmation graphique, le modèle vue-contrôleur, mais aussi de se construire un environnement de développement efficace et adapté à nos besoins. Nous avons utilisé pour ce projet :

- **Microssoft Office Word 2016** pour la rédaction de rapport.
- **Adobe Photoshop CC 2014** pour la manipulation des images.
- **Audacity** pour le traitement de la musique.
- **ArgoUML** pour les diagrammes UML.

Annexes

Annexe 1

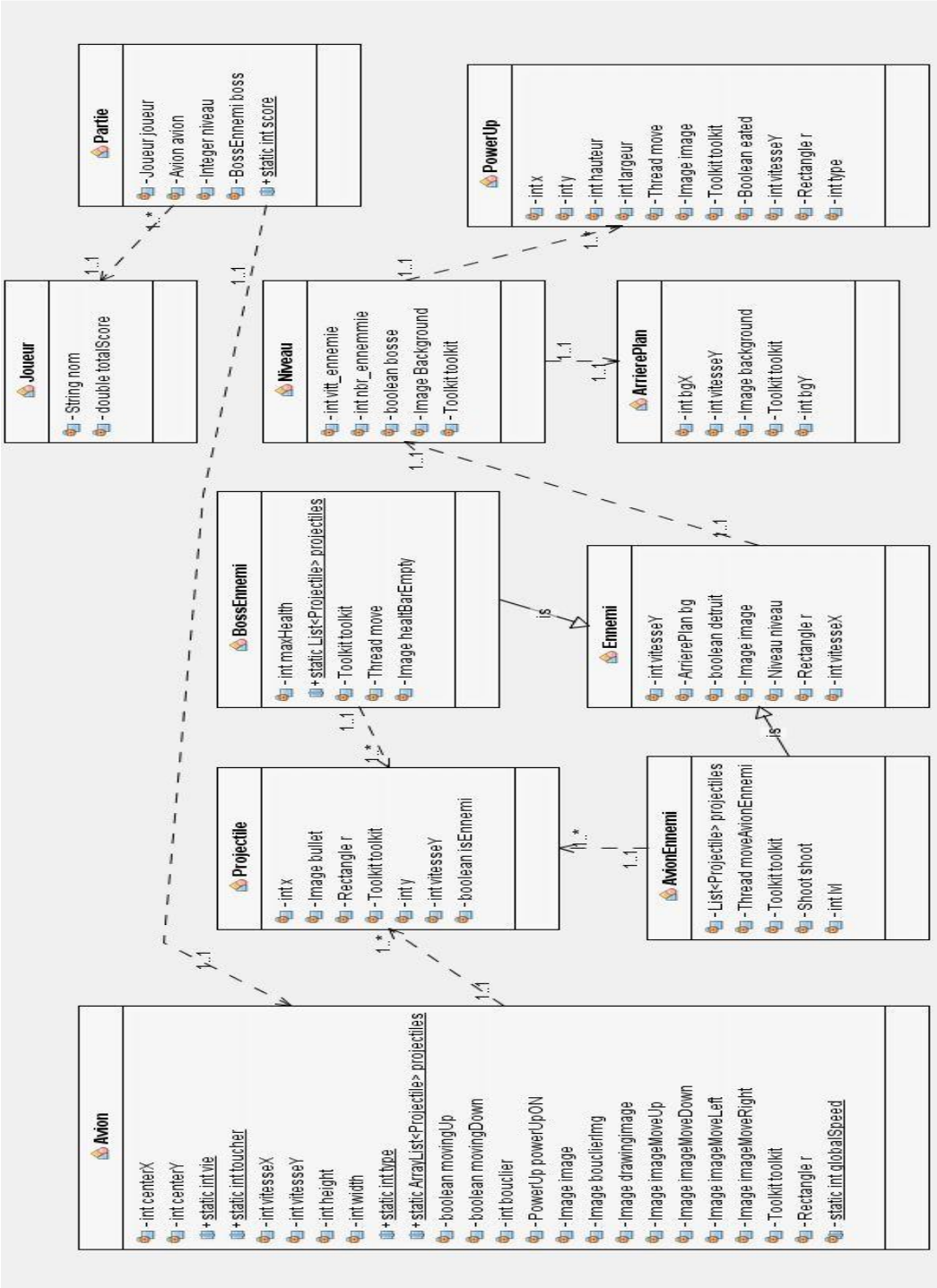


Figure 25 Diagrammes de classes

Sitographie

http://www.kilobolt.com/tutorials.html un tutorial pour développer des jeux vidéo en java.

http://www.deviantart.com la référence pour tous les images utilisées dans le projet.

https://github.com un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git.

http://stackoverflow.com la référence pour toute sorte d'erreurs.

https://www.google.com/ recherches, documentations**.**

Fin

30/04/2016

SAOUDI Anas

WAFDI Ahmed