INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

# Bataille aérienne


Par

DEBARGE Nicolas, LAJNEF Mohamed-Ali,

MEILLE Antoine, ROUSSELOT-PAILLEY Théo

Groupe 45

Villeurbanne, 4 mai 2022

## Table des matières :



# 1. Cahier des charges :

L'objectif du projet était de créer un jeu divertissant, au fonctionnement simple et à la prise en main facile afin d'avoir un rendu final de la meilleure qualité possible. C'est la raison pour laquelle nous avons choisi de travailler sur une simulation de bataille aérienne un contre un, en local. Pour cela, les deux joueurs utilisent une partie différente du même clavier afin de contrôler les avions. En début de partie, on peut choisir le skin de chaque avion ainsi que les pseudos des joueurs. En appuyant sur le menu "commandes", les joueurs ont accès à toutes leurs touches, qu'ils peuvent modifier. Il n'existe qu'un unique mode de jeu. Ce dernier est un mode où chaque avion a le même nombre de points de vie et où le gagnant est le survivant d'une bataille où l'on peut retirer des vies à l'adversaire en lui tirant des missiles dessus. Les deux joueurs ont aussi accès, en pressant une touche, à un boost qui augmente la vitesse de leur avion pendant 4 secondes et qui peut être réutilisé toutes les 10 secondes.

De plus, afin de rendre notre jeu plus réaliste, nous avons aussi pris en compte les effets de la physique sur le déplacement des avions et des missiles. Tout cet aspect de notre projet est détaillé dans la partie suivante. Par ailleurs, nous avons implémenté une "charge" des missiles, qui s'exprime par le fait que, plus la pression sur le bouton tir de missile est longue, plus la vitesse du missile sera forte. Par conséquent, la trajectoire du missile tendra à être rectiligne et elle ressemblera moins à une forme parabolique. Cette modélisation pourrait correspondre au fait que plus le missile prend de temps à être lancé (en raison de nécessités physiques comme la quantité de carburant dans le propulseur), meilleur sera le contrôle de la trajectoire du missile.

# 2. Description du problème physique et résolution :

Le défi le plus important de notre projet était de modéliser la physique des avions et des missiles. Tout d'abord, il faut savoir que notre programme fonctionne ainsi : plusieurs JLabels sont positionnés dans une fenêtre à deux dimensions et on peut faire évoluer leur position et/ou les rendre visibles en fonction des actions des joueurs. Ce fonctionnement est intéressant car nous pouvons gérer avec précision la position des éléments dans la fenêtre. Ainsi, si l'on réussit à calculer les trajectoires, nous pouvons facilement retranscrire une physique simplifiée des avions et des missiles en deux dimensions. Nous allons donc vous expliquer dans les prochains paragraphes notre solution et comment nous l'avons implémentée.

Tout se passe dans la méthode "déplacements" de la classe *Avion* et de la classe *Missile*. Dans cette méthode, nous avons décidé de simplement appliquer le principe fondamental de la dynamique. Pour cela, nous avons dû créer différents attributs dans notre classe :

```
// Vecteurs de l'avion
double[] vitesse = {0,0};
double[] acceleration = {0,0};
double[] position = {0, 0};
int[] forceDeplacement = {0,0};

long tempsPrecedent;
long deltaT;

final double cstePesenteur = 200;
final double csteFrottementX = 8;
final double csteFrottementY = 5;

int masse = 10; //masse en tonnes (peut varier en fonction de l'avion)
```

*Figure 1 : Capture d'écran de certains attributs de la classe Avion*

Les quatre premiers attributs sont des vecteurs. En effet, ils caractérisent le mouvement de l'avion et se mettent à jour tout au long de la partie. La manière dont nous nous en servons est décrite plus loin dans le document. Les deux attributs suivants sont là pour la cinématique de l'avion, ils vont servir de référence de temps pour faire les calculs, notamment pour les intégrations. Ensuite, nous avons trois constantes physiques qui n'ont pas été choisies au hasard. Elles ont été réglées après de nombreux tests et correspondent au mieux au rendu physique que nous attendions. Par exemple, il était intéressant pour nous de pouvoir régler différemment la constante de frottement sur l'avion suivant la direction x et celle suivant la direction y car nous souhaitions que les vitesses maximales soient différentes. Finalement, le dernier attribut est la masse de l'avion qui sert aussi pour les calculs.

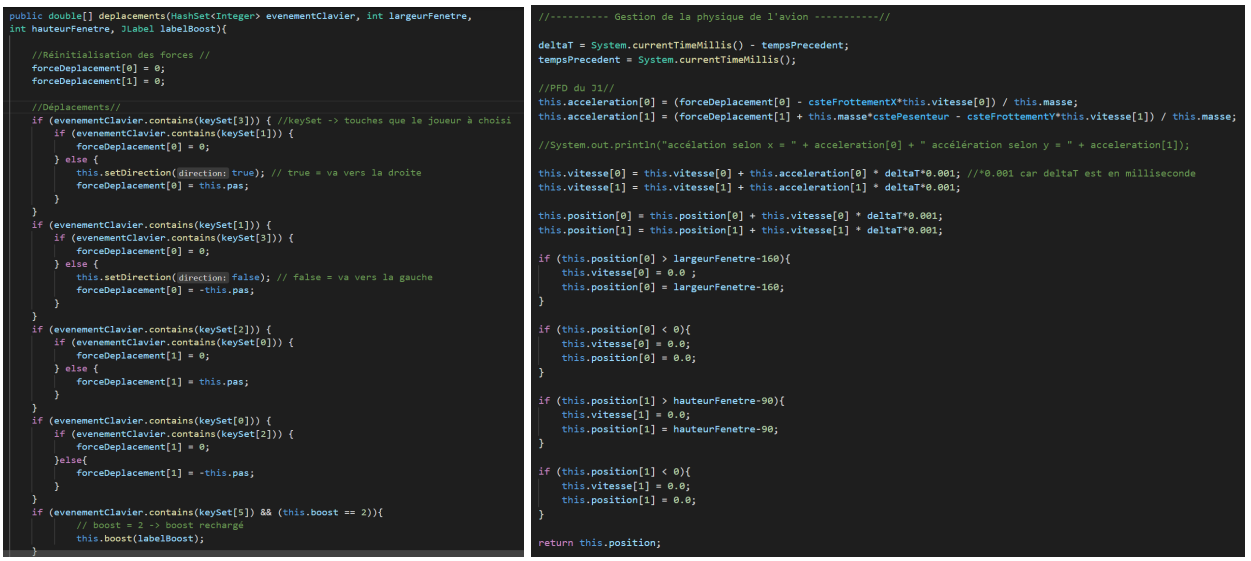
Nous pouvons donc maintenant passer à la description de l'algorithme. Tout d'abord, nous remarquons dans la première capture d'écran un peu plus loin, que la méthode "déplacements" prend en entrée une collection "evenementClavier" qui contient toutes les touches que les joueurs sont en train de presser. Ainsi, grâce à différentes boucles if, différentes actions sont exécutées en fonction des touches pressées. Notamment, lorsque des touches de déplacement sont appuyées, le vecteur "forceDeplacement" de l'avion est mis à jour dans la direction correspondante. Pour le missile, c'est un peu différent, nous y reviendrons dans la suite du document. Ainsi, en prenant en compte la gravité et les frottements, avec les forces de déplacement, nous avons un bilan des forces qui agissent sur notre avion ou notre missile. Nous pouvons ainsi, comme on le voit dans la seconde capture d'écran, appliquer le principe fondamental de la dynamique, ce qui nous permet de connaître l'accélération de l'avion ou du missile selon x et selon y. Une chose importante, il ne faut pas oublier de réinitialiser les forces de déplacements à chaque fois que la méthode est exécutée car un joueur peut avoir lâché une touche entre temps. Ensuite, une fois que nous avons l'accélération, nous pouvons l'intégrer pour obtenir la vitesse, puis réitérer l'opération pour finalement obtenir la position. Pour l'intégration, nous avons utilisé la méthode d'Euler afin de déterminer les primitives :

$$y_{i+1} = y_i + (x_{i+1} - x_i) f(x_i, y_i),\ i \in \{0, n-1\}$$

Avec : $\rightarrow x_{i+1} - x_i$ : l'intervalle de temps entre deux exécutions de la méthode, qui est calculé en faisant la différence entre l'attribut "tempsPrecedent" et le temps actuel donné par la fonction : System.currentTimeMillis().

→ $f(x_i, y_i)$ : ce que l'on cherche à intégrer.

Une fois que nous avons la position, nous n'avons plus qu'à vérifier que l'avion ne sorte pas de l'écran ou que le missile soit toujours dans l'écran pour ensuite la retourner afin qu'elle puisse être actualisée sur la fenêtre de jeu.



```
public double[] deplacements(HashSet<Integer> evenementClavier, int largeurFenetre,
int hauteurFenetre, JLabel labelBoost){

    //Réinitialisation des forces //
    forceDeplacement[0] = 0;
    forceDeplacement[1] = 0;

    //Déplacements//
    if (evenementClavier.contains(keySet[3])) { //keySet -> touches que le joueur à choisi
        if (evenementClavier.contains(keySet[1])) {
            forceDeplacement[0] = 0;
        } else {
            this.setDirection(direction: true); // true = va vers la droite
            forceDeplacement[0] = this.pas;
        }
    }
    if (evenementClavier.contains(keySet[1])) {
        if (evenementClavier.contains(keySet[3])) {
            forceDeplacement[0] = 0;
        } else {
            this.setDirection(direction: false); // false = va vers la gauche
            forceDeplacement[0] = -this.pas;
        }
    }
    if (evenementClavier.contains(keySet[2])) {
        if (evenementClavier.contains(keySet[0])) {
            forceDeplacement[1] = 0;
        } else {
            forceDeplacement[1] = this.pas;
        }
    }
    if (evenementClavier.contains(keySet[0])) {
        if (evenementClavier.contains(keySet[2])) {
            forceDeplacement[1] = 0;
        }else{
            forceDeplacement[1] = -this.pas;
        }
    }
    if (evenementClavier.contains(keySet[5]) && (this.boost == 2)){
            // boost = 2 -> boost rechargé
            this.boost(labelBoost);
    }
```

```
//---------- Gestion de la physique de l'avion ----------//

deltaT = System.currentTimeMillis() - tempsPrecedent;
tempsPrecedent = System.currentTimeMillis();

//PFD du JJ//
this.acceleration[0] = (forceDeplacement[0] - csteFrottementX*this.vitesse[0]) / this.masse;
this.acceleration[1] = (forceDeplacement[1] + this.masse*cstePesenteur - csteFrottementY*this.vitesse[1]) / this.masse;

//System.out.println("accélation selon x = " + acceleration[0] + " accélération selon y = " + acceleration[1]);

this.vitesse[0] = this.vitesse[0] + this.acceleration[0] * deltaT*0.001; //*0.001 car deltaT est en milliseconde
this.vitesse[1] = this.vitesse[1] + this.acceleration[1] * deltaT*0.001;

this.position[0] = this.position[0] + this.vitesse[0] * deltaT*0.001;
this.position[1] = this.position[1] + this.vitesse[1] * deltaT*0.001;

if (this.position[0] > largeurFenetre-160){
    this.vitesse[0] = 0.0 ;
    this.position[0] = largeurFenetre-160;
}

if (this.position[0] < 0){
    this.vitesse[0] = 0.0;
    this.position[0] = 0.0;
}

if (this.position[1] > hauteurFenetre-90){
    this.vitesse[1] = 0.0;
    this.position[1] = hauteurFenetre-90;
}

if (this.position[1] < 0){
    this.vitesse[1] = 0.0;
    this.position[1] = 0.0;
}

return this.position;
```

*Figure 2 : Capture d'écran de la méthode "deplacements" de la classe Avion*

Si vous lisez la méthode "deplacements" de la classe missile, vous remarquerez bien que la physique des missiles est gérée de la même manière, à la différence près qu'on n'introduit pas une force de déplacement lorsque le joueur appuie sur le bouton de tir, mais ici une vitesse initiale selon x et y. Cette vitesse initiale va être modulée proportionnellement au temps d'appui sur le bouton de tir, ce qui modifiera la trajectoire du missile qui sera donc plus ou moins parabolique.

# 3. Structuration du programme et des données :

Il existe trois fenêtres IHM dans notre projet :

- La classe FenetreCommandes qui permet de voir les commandes des joueurs et de les changer si besoin.

- La classe FenetreJeu où les deux joueurs vont pouvoir s'affronter.

- La classe FenetreMenu qui permet permet de sélectionner son type d'avion (aussi appelé skin) et son pseudo en plus d'appeler l'une des deux autres fenêtres par la méthode setVisible en fonction des actions des joueurs.

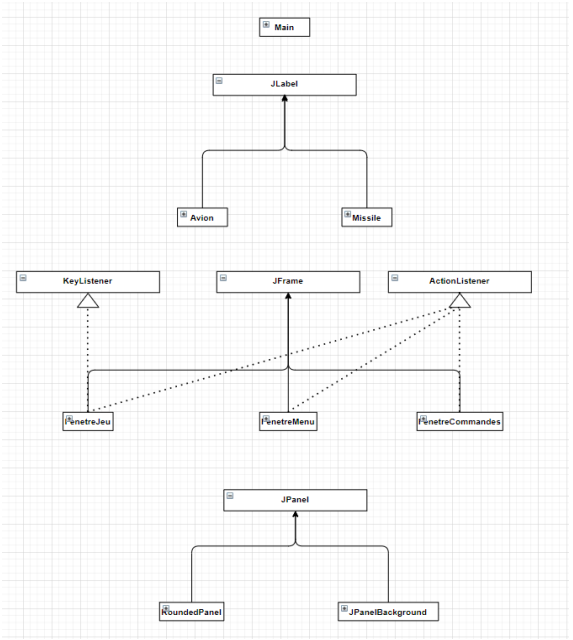


*Figure 3 : Arborescence du diagramme UML du projet*

Ces trois fenêtres contiennent tous les JButtons, les JTextField et les JLabels qui permettent le bon fonctionnement du programme. De plus, FenetreMenu comporte les images des différents avions et FenetreJeu, les objets avions et missiles. Il est à noter que l'on aurait pu créer une classe Joueur, afin d'alléger le code régissant les skins et les commandes caractéristiques du joueur.

Les classes avions et missiles sont des JLabels. On a fait ce choix afin de pouvoir déplacer ces objets à notre guise dans la « FenetreJeu » . Ces classes contiennent toutes comme attribut la masse, les constantes de frottement selon x et y, la constante de pesanteur, ainsi que les vecteurs accélération, vitesse et position. En effet ces deux classes fonctionnent de la même manière dans plusieurs aspects de leur fonctionnement (notamment la gestion de la physique), d'où ces similarités. Il aurait été alors intéressant de pouvoir créer une classe abstraite « objet physique » (avec « avion » et « missile » qui hériteraient de cette classe) qui reprendrait plusieurs de ces attributs afin de pouvoir factoriser encore plus de code. Cependant, par manque de temps, nous ne l'avons pas réalisé.

La classe JPanel RoundedPanel nous a permis de créer des JLabels avec des bords circulaires. Quant à la classe JPanel Background, celle-ci nous a permis de créer le background dans FenêtreJeu.

# 4. Suggestions d'améliorations et bugs connus :

Nous sommes parvenus à faire fonctionner le jeu de manière fluide, en prenant en compte la gravité et les forces de frottements s'appliquant aux avions et aux missiles et en donnant la possibilité aux joueurs de choisir un pseudo, un skin d'avion et leurs commandes. Avec plus de temps, nous aurions pu créer des avions avec différentes caractéristiques (missiles avec plus ou moins de dégâts, avions avec plus ou moins de vies), ou encore créer des modes de jeu, où les joueurs pourraient récupérer des bonus permettant par exemple d'améliorer temporairement les caractéristiques des avions ou encore de récupérer des vies. On aurait aussi pu créer un système de classement des joueurs suivant les pseudos avec le nombre de victoires et des statistiques. Enfin, nous avions au départ envisagé que chaque missile ne puisse être tiré qu'au bout d'un certain temps après le précédent (temps de rechargement), au lieu de notre système actuel où ils ne peuvent être tirés que si certaines conditions sont remplies (seulement si le missile tiré précédemment est sorti de la fenêtre ou a touché l'adversaire). Ceci n'a pas été réalisé par manque de temps, mais aurait pu être réalisé en assignant non pas un missile à chaque joueur mais une linkedList de missiles qui ajouterait des missiles à chaque fois que l'on tire (en assignant bien la condition du temps) et enlèverait le missile dès lors qu'il y a collision. Cette liste aurait une taille maximale, ce qui permettrait de limiter les missiles.

Parmi les bugs connus que nous n'avons pas eu le temps de résoudre, se trouve le problème consistant en ce qu'une même touche puisse être assignée à plusieurs actions dans le menu commandes tandis que la ligne de touches où se situent les chiffres ne peut

pas être assignée. Nous avons aussi remarqué le fait que lorsqu'un avion se déplace vers la gauche, celui-ci ne se stoppe pas lorsque l'on relâche les commandes, mais glisse à l'infini.

# 5. Carnet de route et échéancier :

Afin d'avoir la meilleure organisation possible de nos travaux, nous avons choisi d'utiliser le logiciel *Visual Studio Code* associé au logiciel *Git*, nous permettant de travailler, quand nous le souhaitons, sur différentes parties du code. Le code étant sauvegardé sur un *GitHub* que nous avons créé pour l'occasion. Nous avons également créé un groupe *Messenger* afin de pouvoir poser des questions aux autres membres du groupe si un problème se pose, ou bien faire des retours sur l'avancement de chacun. Comme le montre le graphique ci-dessous représentant le nombre de "commits" en fonction du temps, nous avons été assez réguliers dans notre travail :

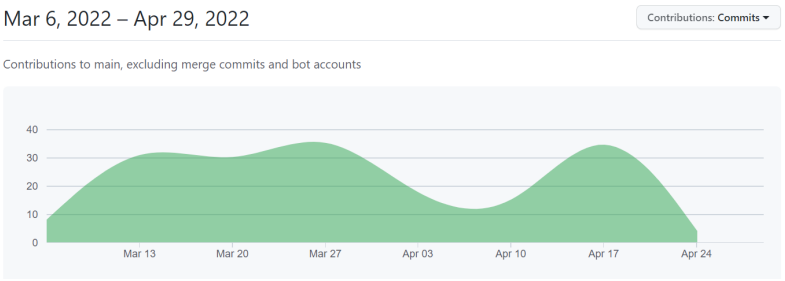



*Figure 4 : Capture d'écran de l'un des graphiques de l'onglet statistique du GitHub*

Nous avons bien sûr plus ou moins travaillé en fonction de la charge de travail, mais nous avons essayé d'avancer un minimum chaque semaine en attribuant un rôle à chaque membre. En effet, Antoine et Théo se sont tous deux occupés de l'IHM des deux fenêtres du projet, tandis que Mohamed et Nicolas se sont chargés du fonctionnement de la partie de jeu en elle-même.

Nous ne nous sommes pas véritablement fixés d'objectifs chaque semaine, car grâce au logiciel *Git*, chaque personne a pu avancer plus ou moins indépendamment et à son rythme. Nous n'avons donc pas de carnet de route, mais nous avons tout de même procédé dans un certain ordre :

- D'abord, nous avons créé séparément le menu de démarrage et la fenêtre de jeu, ce qui nous permettait donc de lancer individuellement les fenêtres et ainsi ne pas attendre que le menu soit fini pour créer le jeu.
- Ensuite, nous avons créé parallèlement une première version du jeu fonctionnelle sans physique et une première version simple du menu sans la possibilité de rentrer son pseudo, de choisir son skin d'avion ou de modifier les commandes.
- Nous avons par la suite rassemblé les deux fenêtres, nous permettant de lancer le jeu lorsque l'on appuie sur le bouton jouer du menu.
- Finalement, nous avons rajouté petit à petit la physique des avions et des missiles, un boost utilisable toutes les 15 secondes, la possibilité de changer les commandes et enfin de choisir son pseudo et son skin.

Nous avons décidé de nous arrêter là par manque de temps, mais, comme montré dans la partie précédente, il nous restait de nombreuses idées à implémenter dans notre jeu.

# 6. Pourcentage d'implication de chaque membre de l'équipe :

En conséquent, comme chaque personne a pu finir à son rythme toutes les tâches qui lui étaient attribuées en fonction de son niveau, et que tout le monde s'est impliqué à hauteur de ses capacités dans ce projet, nous considérons qu'il est juste que tout le monde ait le même pourcentage d'implication. Voici donc les pourcentages :

- DEBARGE Nicolas : 25%
- LAJNEF Mohamed-Ali : 25%
- MEILLE Antoine : 25%
- ROUSSELOT-PAILLEY Théo : 25%

# 7. Bibliographie :


- Classe RoundedPanel : https://www.codeproject.com/Articles/114959/Rounded-Border-JPanel-JPanel-graphics-improvements

- Fond d'écran du jeu : https://www.123rf.com/photo_86730108_pixel-art-mountains-grass-and-clouds-seamless-background-for-game-lanscape-or-application.html

- Image game over : http://pixelartmaker.com/art/0f529138e7cffe2

- Image boost : https://www.pixilart.com/art/frc-2018-boost-power-up-bf5b79e7343e8f2

- Image vies : https://fr.freepik.com/vecteurs-premium/barre-vie-du-jeu-pixel-vector-art-barre-coeur-sante-8-bits-controleur-jeu-jeu-symboles_9815969.htm

- Images décompte : http://pixelartmaker.com/art/112944c1e93844b

- Image missiles : https://fr.dreamstime.com/missile-d-art-pixel-vecteur-image102310237

- Images avions :

https://www.alamyimages.fr/pixel-art-de-l-avion-a-reaction-militaire-image425742569.html

https://stock.adobe.com/fr/images/vector-pixel-art-helicopter/173765743

https://www.istockphoto.com/fr/vectoriel/caricature-de-vecteur-pixel-art-gm854029630-140462303

https://www.shutterstock.com/fr/image-vector/vector-pixel-art-jet-isolated-721756837

https://www.shutterstock.com/fr/image-vector/vector-pixel-art-jet-isolated-721755052

https://de.123rf.com/photo_71502391_pixel-art-d%C3%BCsenflugzeug.html

- Nous nous sommes aussi aidés du site web https://stackoverflow.com pour résoudre certains de nos problèmes.

# 8. Annexe :

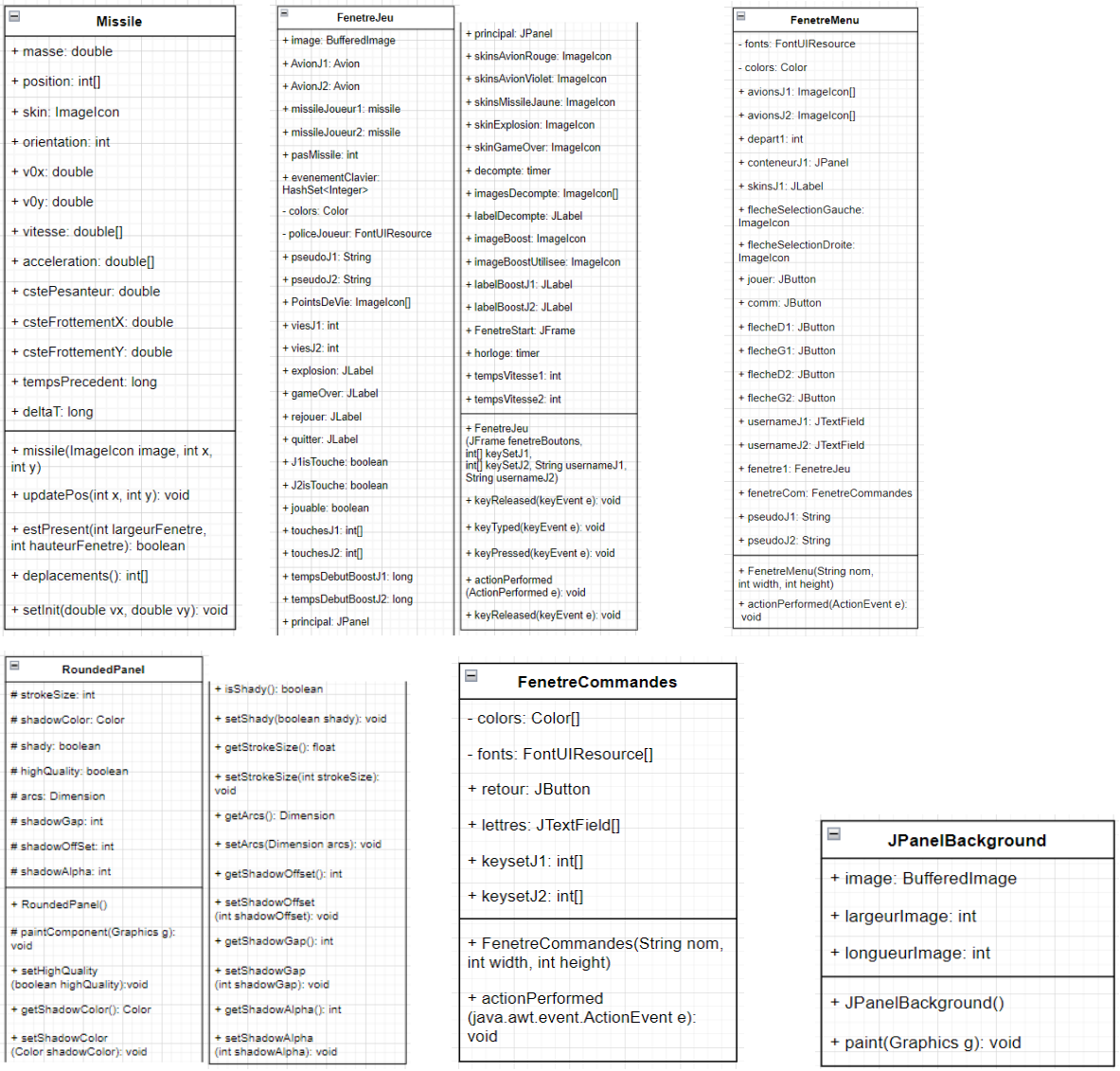



*Figure 5 : Capture d'écran du détail des classes du diagramme UML*