Trajectory detection for game scoring

PFA - Document de spécification des besoins

Louis Cesaro, Timothée Corsini, Jean Farines,
Arno Galvez, Adrien Lavancier, Anatole Martin, Félix Pierre

Client : Martial Bossard

Responsable Pédagogique : David Renault

11 décembre 2018, Enseirb-Matmeca

Table des matières

1 Introduction et motivation

1.1 Définition du projet

Ce dossier de spécification renferme la description complète du projet effectué dans le cadre du PFA à l'Enseirb-Matmeca, en collaboration avec Asobo Studio. Ce document est une vue d'ensemble du projet, et comprend la description des besoins, des algorithmes de reconnaissance de figures et de calcul de physique, la spécification de l'architecture de développement, les éventuelles améliorations à apporter et l'environnement de test.

Le projet décrit dans ce document est un prototype de jeu de course de style Micromachines™ avec obstacles capable de reconnaître des figures de voltige aérienne réalisées par le joueur (cf. partie 1.3). Les jeux Micromachines™ sont des jeux de course mettant en scène des voitures miniatures dans des circuits à échelle humaine (salon, table de billard, cuisine...). Le but serait donc d'avoir le même type de parcours avec des avions miniatures. Néanmoins, le cœur du projet sera l'algorithme de reconnaissance des figures, qui pourra être réutilisé par l'entreprise dans le cadre de futurs projets.

Figure 1 – Course de voitures Micromachines™ issue du jeu vidéo *Micro Machines World Series*

Figure 2 – Avion effectuant une figure lors d'un spectacle de voltige aérienne

1.2 Objectifs du projet

Le jeu final devra comporter au moins un parcours d'obstacle, où évoluera l'avion du joueur. Cet avion aura une physique similaire à un avion de voltige, avec les forces adaptées (portance, poids, accélération...), tout en restant dans un cadre ludique.

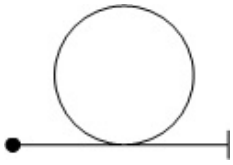
Lorsque le joueur effectue une figure, le logiciel la reconnaît et attribue des points selon la précision de la figure. Le logiciel de reconnaissance devra également prendre en compte les combinaisons de figures.

Le parcours consistera en un petit monde que le joueur devra traverser d'un bout à l'autre. Il comprendra des obstacles (statiques ou mobiles par la suite) qui inciteront le joueur à effectuer des figures pour les éviter. Le niveau pourra éventuellement être généré de manière procédurale.

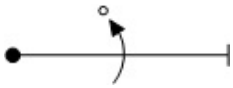
1.3 Figures

Une figure est une succession de manœuvres vérifiant un ensemble de conditions telles que la durée, l'orientation et l'altitude.

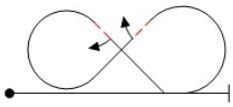
Le système de reconnaissance de figures gérera les figures reconnues officiellement par la FAI (Fédération Aéronautique Internationale) dans les documents de la CIVA (FAI Aerobatics Commission) [1]. Voici quelques exemples :



Loop : L'avion s'oriente vers le haut jusqu'à revenir dans sa position initiale. Il doit garder ses ailes horizontales lors de la manœuvre et terminer à la même altitude que celle à laquelle il commence.



Barrel Roll : L'avion effectue un *Loop* tout en effectuant une rotation sur lui-même, en s'orientant vers la gauche ou vers la droite. Sa trajectoire a la forme d'un tire-bouchon.



Cuban Eight : L'avion effectue 5/8 d'un *Loop* vers le haut puis fait la même chose vers le bas, jusqu'à revenir à sa position initiale. Les deux cercles doivent avoir le même rayon et la même altitude. L'avion doit aussi avoir la même altitude au début et à la fin de la manœuvre.

2 Besoins fonctionnels

2.1 Contenu visible

Ce jeu sera développé avec l'aide du moteur Unity. Puisque le projet est un jeu vidéo en 3 dimensions, il devra posséder un environnement visible. Dans cet environnement, les éléments suivants devront être présents et visibles par le joueur :

— **Avion :** Un modèle d'avion, créé ou pris d'un modèle libre de droits, qui sera contrôlé par le joueur.

— **Environnement :** Un environnement, constitué d'un sol et d'un ciel qui servira de décor pour le jeu et dans lequel le joueur pourra évoluer.

— **Parcours :** Un parcours constitué d'anneaux indépendants de l'environnement, dans lesquels le joueur devra passer pour gagner. Le jeu devra également comporter des parcours d'entraînement pour apprendre les figures. Ces parcours comporteront des anneaux pour guider le joueur lors de la réalisation de la figure.

2.2 Fonctionnalités

Le jeu devra également intégrer différents algorithmes permettant de réaliser les éléments suivants :

— **Vol :** Il doit être possible de faire voler un avion et de faire des figures. Pour cela, une physique simplifiée sera utilisée (cf. partie 2.3).

— **Détection de figures :** Le jeu doit pouvoir détecter les figures réalisées par le joueur (cf. partie 2.4)

— **Score :** Le jeu doit attribuer des points au joueur selon la figure réalisée et la qualité d'exécution de cette figure, selon les critères de notation de la CIVA.

2.3 Physique de vol

La physique sera proche du modèle défini sur la page wikipedia de la dynamique de vol [4] :

— **Forces :** Seules les forces relatives au poids, à la puissance du moteur (traction), à la portance et à la traînée seront considérées (cf. figure 3). La force du vent sur l'avion ne sera pas considéré.

— **Rotations :** Les trois rotations devront être considérées (lacet (*yaw*), tangage (*pitch*) et roulis (*roll*)) (cf. figure 4).

2.4 Détecteur de figures

2.4.1 Figures détectées

Dans un premier temps, le détecteur devra reconnaître les figures de base telles que les *Loops* et les *Barrels Roll* selon une méthode probabiliste (chaque figure aura un taux de correspondance associé, le taux le plus haut sera la figure reconnue). Des figures plus avancées telles que le *Cuban Eight* et des figures officielles de compétition présentées dans les documents de la CIVA [1] seront ajoutées au fur et à mesure à l'algorithme de reconnaissance. Les figures devront être reconnues si

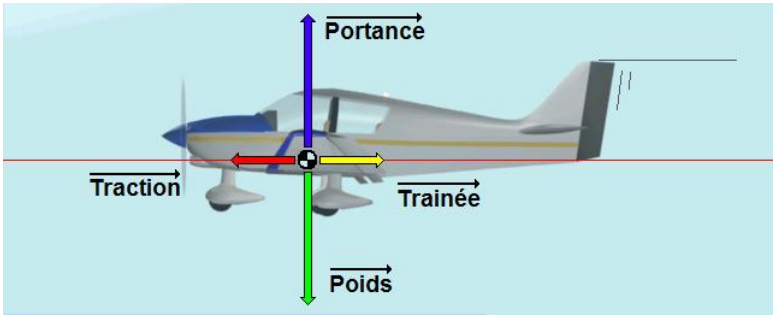


FIGURE 3 – Physique simplifiée de l'avion

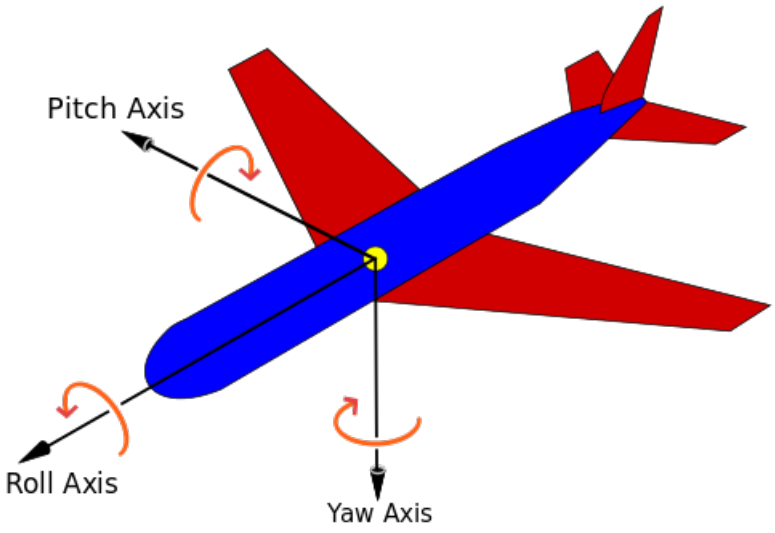


FIGURE 4 – Physique simplifiée de l'avion

elles possèdent un pourcentage de ressemblance suffisamment élevé par rapport au modèle parfait, et les faux positifs [1] devront être évités au maximum.

2.4.2 Algorithmes de détection

Le détecteur intégrera plusieurs méthodes de détection dont :

— une méthode "naïve", basée sur un automate dont les états suivent l'inclinaison de l'avion (cf. figures 5 et 6).

— une méthode plus avancée, basée sur du *pattern matching* [2] s'appuyant sur l'algorithme $1 [2]. Le *pattern matching* sera effectué sur l'évolution de la hauteur et des rotations de l'avion au fil du temps (cf. figure 7).

Ces méthodes devront pouvoir être comparées en termes de temps de calcul et de précision de la reconnaissance. Il faut de plus que ces méthodes puissent analyser les figures en continu afin de n'en manquer aucune, ainsi que prendre en compte la réalisation de plusieurs figures à la suite.

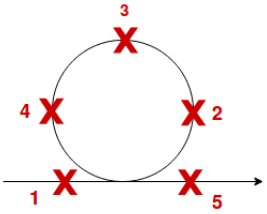


FIGURE 5 – Exemple de grammaire pour reconnaître un *loop*

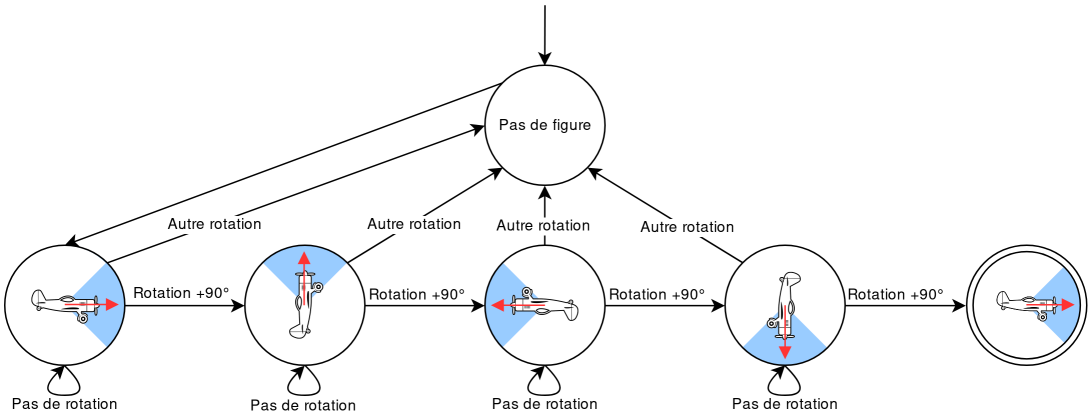


FIGURE 6 – Exemple d'automate capable de reconnaître un *loop*

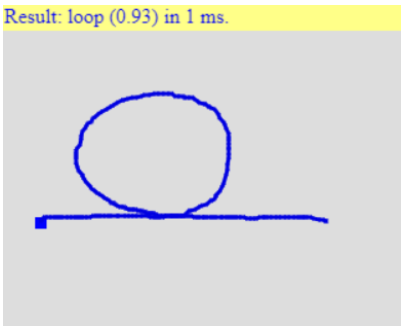


FIGURE 7 – Exemple de pattern matching de l'algorithme $1 appliqué à un *loop*

1. Trajectoires reconnues comme des figures valides alors qu'elles n'en sont pas
2. Reconnaissance de motifs

3 Besoins non fonctionnels

Le projet se présentant sous la forme d'un jeu vidéo, certains besoins doivent être remplis afin que l'utilisateur puisse tester les fonctionnalités dans le meilleur environnement possible. Le jeu se doit donc d'être :

— **Fluide** : Le *framerate* minimal sera de 30 images par secondes, afin que l'expérience de jeu soit fluide et non saccadée. En étant optimiste, cette barre pourra être mise par la suite à 60 IPS [3] minimum.

— **Ergonomique** : Piloter un avion est très complexe, notre but est de transcrire cela le plus intuitivement possible, permettant au joueur de comprendre rapidement comment évoluer dans l'environnement. Le joueur pourra choisir entre un clavier et une manette : il est beaucoup plus intuitif de commander un avion avec des joysticks, mais tout le monde n'en possède pas, une implantation des deux est donc nécessaire.

— **Réaliste** par rapport à la physique d'un avion : Afin de renforcer l'immersion, la physique de l'avion devra être ludique et réaliste : le joueur devra avoir le sentiment de piloter un vrai avion, sans pour autant devoir faire face aux contraintes auxquelles un vrai pilote ferait face. La physique devra être telle que le joueur puisse intuitivement exécuter des figures complexes, comme celles décrites plus haut, ainsi qu'interagir avec son environnement (passer sous un pont, dans un anneau....).

— **Rejouable** : l'ajout d'un score augmentant en fonction des figures effectuées par le joueur lui permettra d'avoir un retour direct sur ses performances. Une figure complexe et bien réalisée rapportera plus de points qu'une figure simple ou pauvrement exécutée. Un chronomètre lors des parcours permet au joueur d'avoir envie de le refaire mieux et plus rapidement.

— **Portable** : Il faut que le produit final soit compatible sur Windows 64-bits, et qu'il ne soit nécessaire d'installer aucun logiciel particulier tel qu'Unity ou Visual Studio pour pouvoir lancer le jeu.

4 Architecture et conception

Le projet sera réalisé avec l'aide du moteur de jeu Unity dans sa version 2018.2.15f1. Le langage utilisé pour le développement avec Unity est le C#, nous adopterons donc une architecture orientée objet dans l'ensemble de notre code.

L'utilisation du moteur se fait par l'utilisation des classes et méthodes du package `UnityEngine`[3]. En particulier, nous utiliserons principalement la classe `MonoBehaviour`, englobant les méthodes et classes appelées à chaque *frame*, `Input`, qui permet de récupérer les commandes de l'utilisateur, ainsi que `GameObjet`, permettant de contrôler les objets présents dans le jeu (affichés ou non).

Le code du projet va être amené à gérer différents éléments dans le jeu, nous prévoyons donc de découper nos classes selon le modèle suivant, illustré en Figure 8 et détaillé ci-après.

4.1 Gestion des objets dans le moteur de jeu

Le moteur de jeu Unity dispose par défaut d'un grand nombre de classes permettant d'interagir avec les objets présents dans le jeu. Il nous faut donc de notre coté des classes avec lesquels nous récupérons des informations sur les objets et les contrôlerons :

— `Plane` : Cette classe correspond à l'avion que contrôle le joueur. Elle fait le lien entre notre code et le modèle visible en jeu, et permet de regrouper les informations importantes (position, rotations, vitesse...).

— `PlaneController` : Classe gérant les calculs sur la physique de l'avion. Elle fait notamment le lien entre les *inputs* du joueur et leur influence sur les forces à appliquer.

3. Images Par Seconde

— `FigureManager` : Cette classe coordonne la reconnaissance périodique de figures. C'est elle qui communique à l'algorithme les positions de l'avion, traite le résultat, attribue les points, et communique les résultats au joueur.
— `SettingsUI` : Elle définit une interface utilisateur (*UI*) permettant la consultation et/ou la modification des paramètres de jeu.

4.2 Physique de l'avion et détection des figures

La modélisation physique de notre avion passe par le calcul de différentes forces. La reconnaissance de figures, quant à elle, se fait via plusieurs méthodes. Nous disposons donc des classes suivantes :

— `Force` : Ce sont les classes caractérisant chacune des forces s'appliquant sur notre avion. Elles permettent, à partir de l'état de l'avion à un instant donné, d'obtenir un vecteur force utilisable par le moteur de jeu.
— `FigureDetection` : Cette interface et ses implémentations sont les différentes méthodes de détection de figures que nous utiliserons dans notre projet.
— `Settings` : Cette classe rassemble la majorité des paramètres de jeu : propriétés de l'avion, figures à détecter, etc.

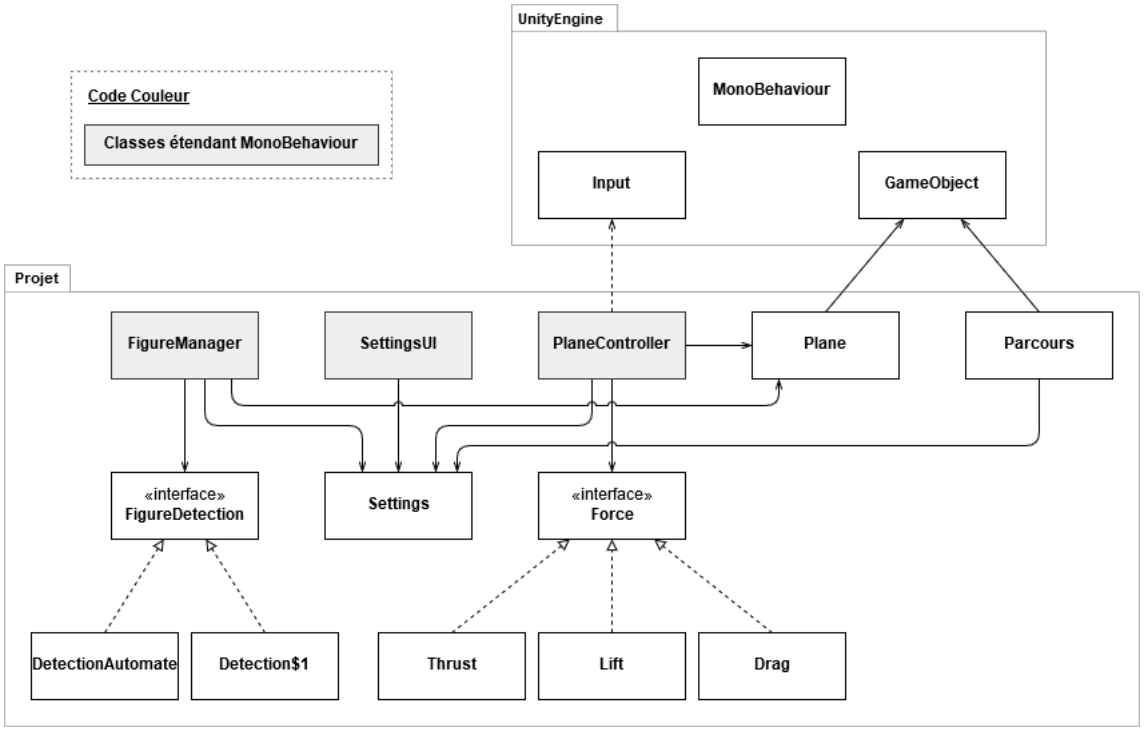


FIGURE 8 – Diagramme de classe prévisionnel du projet

5 Évolution du système

Par la suite, le projet pourra voir l'apparition de nouvelles fonctionnalités telles que :

— **Une génération procédurale** de parcours : le lieu du parcours et le parcours doivent pouvoir être générés de façon procédurale afin d'avoir une nouvelle expérience de jeu à chaque fois.

— **Une interface "d'entraînement"** avec mise en place d'anneaux requérant une rotation particulière : afin de pouvoir s'entraîner, il devra être possible de générer des parcours à suivre via des anneaux ou autres nécessitant un contact particulier afin de réussir à réaliser des figures plus ou moins complexes.

— **Différents modes de difficulté** : le joueur pourra être en mesure de modifier la difficulté du mode. Cela se traduira notamment par la baisse ou l'augmentation de la précision de la reconnaissance de figure.

— **La variété des modes de jeu** : divers modes de jeux pourront être implémentés et proposées au joueur, afin de pouvoir tester les différentes fonctionnalités : course contre la montre, freestyle...

— **L'ajout d'obstacles mobiles** : il sera possible de rendre les parcours plus difficiles en ajoutant, lors de la génération desdits parcours, des obstacles non plus statiques mais mobiles. Ce point, indiqué par le joueur avant de lancer un parcours, sera pris en compte lors de la création du parcours.

6 Plan de tests de validation

Pour vérifier le bon fonctionnement du projet, des tests à différentes échelles seront réalisés.

6.1 Tests sur les algorithmes de reconnaissance

Les tests sur les algorithmes de reconnaissance seront à faire de préférence sans Unity pour qu'ils soient indépendants du logiciel. On fera une série de tests pour chaque méthode de reconnaissance et pour chaque figure à reconnaître. Les figures seront enregistrées sous la forme de série de points, on commence par vérifier que les figures parfaites (en proportions) sont toujours reconnues. On prendra ensuite des figures jugées reconnaissables pour regarder leur validité. Enfin, on vérifiera que les séries de points qui ne représentent pas la forme attendue ne sont pas reconnues, ce qui permettra de vérifier qu'on distingue bien les figures différentes mais aussi qu'on ne reconnaît pas les déplacements qui ne représentent aucune figure. Un algorithme de test simple serait de la forme :

```
void testReconnaissance(methode, forme) {
    figure = formeParfaite(forme);
    assert(correspondance(methode(figure), forme));
    Pour figure dans listeFiguresValides {
        assert(correspondance(methode(figure), forme));
    }
    Pour figure dans listeFiguresInvalides {
        assert(!correspondance(methode(figure), forme));
    }
}
```

La liste de figures reconnues pourra être agrandie au fur et à mesure du projet. Des tests plus poussés pourraient évaluer des cas plus ambigus, par exemple la détection de deux figures avec un taux de reconnaissance similaire. Ces tests afficheront la vitesse de calcul des méthodes de reconnaissance.

6.2 Tests sur les déplacements de l'avion

Des tests sur le déplacement de l'avion permettront de vérifier que les forces s'appliquent correctement. On observe les déplacements de l'avion pour voir s'ils sont cohérents, notamment vis-à-vis de la vitesse de déplacement ou encore des virages. Ces tests seront réalisés sous Unity.

Une partie de ces tests pourrait fonctionner avec une série de commandes donnée en entrée, et vérifierait que le déplacement est cohérent.

Des tests de prise en main et de jouabilité lors du déplacement de l'avion pourront être réalisés au travers du "didacticiel", avec des éléments de décors comme repères tels que des cercles pour observer les déplacements de l'avion.

6.3 Tests d'intégration

Enfin, les tests d'intégration auront pour but de vérifier que le système de déplacement de l'avion fonctionne correctement avec le système de reconnaissance de figures. Nous vérifierons en temps

réel qu'une figure est correctement reconnue et de manière fluide, que l'avion bouge correctement selon les commandes, sans bugs d'affichage. Ce test sera fait sous la forme d'un didacticiel de prise en main avec pour objectif la réalisation de figures demandées, le déplacement dans les bonnes zones, avec un retour indiquant la réussite ou non des épreuves.

7 Planning prévisionnel

A la demande du client, un aperçu du projet sous la forme d'exécutable sera livré au client tous les troisièmes mardis de chaque mois, afin d'avoir une vue sur l'avancement du projet et de pouvoir corriger si besoin est.

Une découpe des tâches du projet comme indiqué sur la figure 9 sera à prévoir lors de la suite du projet.

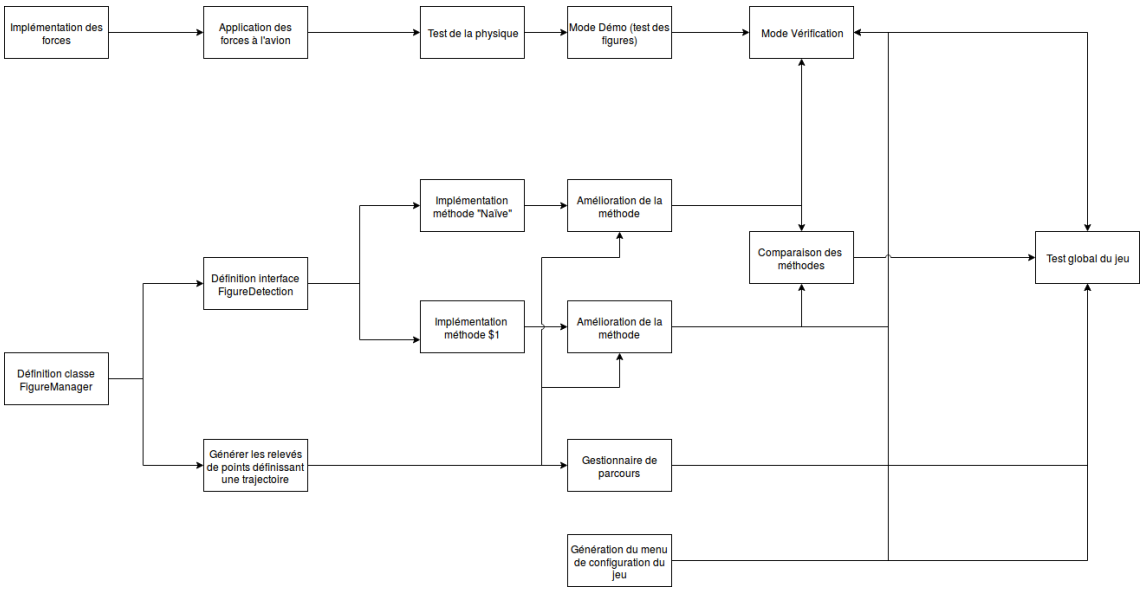


FIGURE 9 – Diagramme de découpe des tâches du projet

FIGURE 10 – Diagramme de Gantt du projet (durées relatives)

Références

[1] CIVA. *Free Known Programme 1 Power Intermediate / Yak-52, Advanced an Unlimited.* 2018. URL : `https://www.fai.org/sites/default/files/documents/civa_free_known_programme_guidance_-_power_aircraft_2018_v3a.pdf`.

[2] Yang Li Jacob O. Wobbrock Andrew D. Wilson. *Gestures without Libraries, Toolkits or Training :A $1 Recognizer for User Interface Prototypes*. 2007. URL : http://faculty.washington.edu/wobbrock/pubs/uist-07.01.pdf.

[3] Unity. *Unity Scripting Reference*. 21 Novembre 2018. URL : https://docs.unity3d.com/ScriptReference/.

[4] Wikipedia. *Flight dynamics (fixed-wing aircraft)*. 12 Juillet 2018. URL : https://en.wikipedia.org/wiki/Flight_dynamics_(fixed-wing_aircraft).