Trajectory detection for game scoring

Reconnaissance de trajectoire pour le jeu vidéo

PFA - Manuel d'utilisation et de maintenance du code

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

Clients : Martial Bossard, Pierre-Marie Plans

Responsable Pédagogique : David Renault

8 avril 2019, Enseirb-Matmeca

Table des matières

Introduction

Le présent document a un double objectif. Tout d'abord, il s'agit de décrire comment utiliser le projet et son code. De plus, il est décrit ici la structure globale du code, et la façon dont il peut être maintenu / réutilisé.

Ce document n'est **pas** une documentation détaillé du code. Cette dernière est fournie avec le code source, dans le répertoire *Documentation*, elle se génère avec l'utilitaire Doxygen.

1 Ouverture du projet

1.1 Logiciels requis

Le présent projet a été développé à l'aide du moteur de jeu Unity dans sa version 2018.3.8f1. Par conséquent, il est nécessaire de disposer d'une version égale ou supérieur à celle-ci pour pouvoir ouvrir le projet sans problème. Pour les assets 3D, ceux-ci sont éditables avec le logiciel Blender.

Il est important de noter que le projet dans sa version exécutable ne nécessite aucune dépendance particulière.

1.2 Compilation

Pour créer un exécutable du projet, il est nécessaire de passer par l'interface de *Build* d'Unity. L'ensemble des paramètres de *build* ayant déjà été configurés au préalable, il n'est normalement pas nécessaire d'effectuer de manipulations supplémentaires.

Soyez néanmoins conscient du fait que ce projet utilise un certain nombre de fonctionnalités fournies par les *runtimes* dotNet4, assurez vous donc que celles-ci sont bien sélectionnées dans *Edit -> Project Settings* (cf. Figure 1).

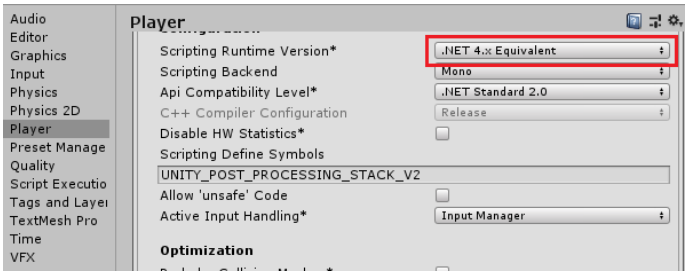

FIGURE 1 – Option de sélection des runtimes dotNet

A noter également que la phase de compilation n'est pas nécessaire pour essayer le projet : celui-ci peut être directement testé depuis l'éditeur d'Unity.

2 Description des fonctionnalités du jeu

2.1 Launcher Unity

Lors du lancement du jeu s'affiche tout d'abord le launcher Unity. Celui-ci permet de régler d'une part la qualité graphique du jeu, mais surtout de personnaliser les commandes selon la convenance du joueur, comme illustré en Figure 2. À noter que les commandes ne sont pas modifiables une fois le jeu lancé : il faudra quitter et relancer le jeu.

2.2 Écrans de jeu

Le jeu dispose de 3 écrans principaux : un menu principal, un menu d'options, et un niveau de jeu. Le menu principal est le premier écran s'affichant en jeu, et permet d'accéder au niveau, comme au menu d'options.

Le menu d'options permet de modifier divers coefficients physique de l'avion : puissance du moteur, intensité de la portance, coefficient de frottement... Il est également possible de choisir depuis ce menu l'algorithme de détection utilisé en jeu, et de changer la résolution du jeu.

Enfin, en cliquant sur "Jouer" depuis le menu principal, on accède au jeu en lui même.

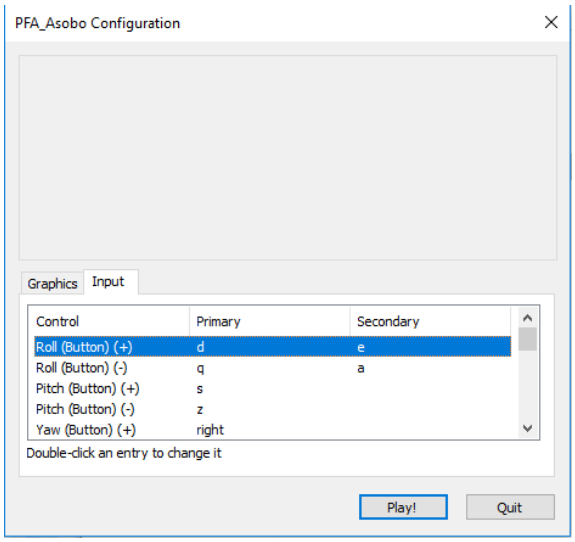

FIGURE 2 – Launcher Unity avec éditeur de touches

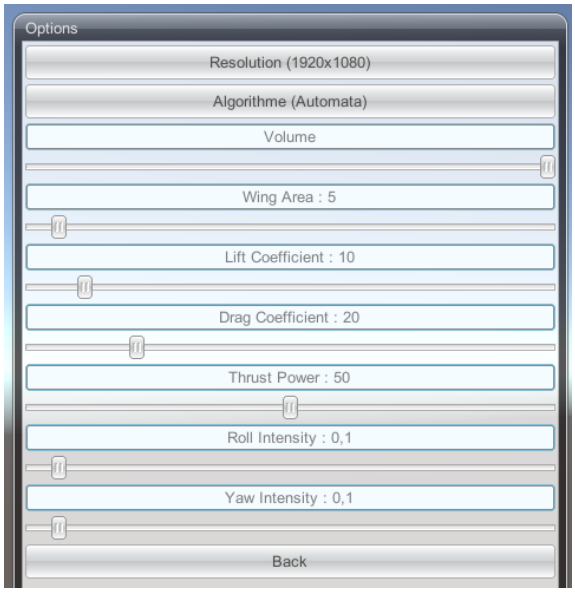

FIGURE 3 – Menu d'options en jeu

2.3 Contrôles de jeu

La liste des contrôles est aussi accessible depuis le menu principal :
— **ECHAP** fait apparaître le menu principal qui permet de relancer la scène de jeu et d'accéder aux options de jeu
— **R** relance la scène
— **P** permet d'alterner entre les algorithmes de reconnaissance ($P et automates)
— **touches directionnelles** et **Z,Q,S,D** contrôles de l'avion.
Certains contrôleurs de jeux, tels que les manettes Xbox, sont également supportés.

2.4 Séquences enregistrées

Il est possible de rejouer des séquences pré-enregistrées en cours de jeu : le joueur n'a ainsi plus besoin de toucher les commandes, la séquence choisie se répétant en boucle. Actuellement, trois figures sont disponibles en tant que séquences enregistrées : le *Loop*, le *Roll* et le *Cuban Eight*.

Il est également possible d'enregistrer ses séquences de jeu. Celles-ci sont placées dans des fichiers .csv dans le dossier parent du jeu. Ces fichiers décrivent l'entièreté des positions de l'avion au cours de l'enregistrement, ainsi que les *inputs* utilisateur. La procédure pour ajouter ses propres séquences rejouables sera décrite plus loin.

Les commandes associées à ces fonctionnalités sont :
— **Numpad −** Sélectionne la séquence à rejouer
— **Numpad Entrée** Lance la séquence sélectionnée
— **Numpad + (appui long)** Démarre/Interrompt l'enregistrement

Ajouter ses propres séquences La classe responsable des séquences enregistrées est `DummyPlayer`. Pour ajouter une nouvelle séquence enregistrer, il faut donc :
— Disposer d'un fichier .csv à 5 colonnes décrivant les inputs successifs à simuler (dans l'ordre : *Temps*, *Accélération*, *Roll*, *Pitch* et *Yaw*).
— Ajouter le nom de la figure dans `private string[] nameModes = "Aucun", "Loop", "Roll", "Cuban8" ;`
— Rajouter dans la méthode *Update* de la classe la lecture du fichier correspondant, par un appel à *replayFromFile*
La séquence sera dès lors sélectionnable depuis le jeu pour être rejouée.

3 Structure générale du projet

3.1 Contenu d'une scène de jeu

Une scène de jeu est constitué d'au moins 3 objets principaux : - un terrain de jeu - l'avion contrôlé par le joueur - la caméra suivant l'avion

Des *prefabs* ont été créés pour faciliter la création d'un niveau, et sont disponibles dans *Assets/Prefabs*. Ainsi, le prefab `StartLinePrefab` regroupe l'avion, la caméra, et différents scripts permettant notamment de redémarrer le niveau : il suffit donc de le glisser dans une scène et de rajouter un simple terrain pour obtenir une base de niveau fonctionnel.

Voici la liste des prefabs disponibles :
— `DummyPlayer` rejoue des séquences enregistrées
— `Explosion` prefab instancié par l'avion lors d'un impact avec le sol
— `PlaneCameraPrefab` caméra suivant sa cible avec un retard
— `PlanePrefab` avion contrôlé par le joueur
— `PlaneTracker` enregistre la trajectoire de l'avion dans un fichier csv
— `Rampe` simple rampe de décollage
— `StartLinePrefab` contient les bases d'un niveau
— `TrailSmokePrefab` particules traçant la trajectoire de l'avion
— `UI Dial` affichage d'informations sur l'avion
Il est important de noter que certains GameObjects doivent être liés entre eux. Ainsi, la caméra, le DummyPlayer et le PlanerTracker doivent être liés à l'avion, et l'avion doit posséder un lien vers des objets Text pour afficher les figures réalisées et l'algorithme utilisé.

3.2 Diagramme de classes

La Figure 4 présente les classes importantes du projet et leurs relations.

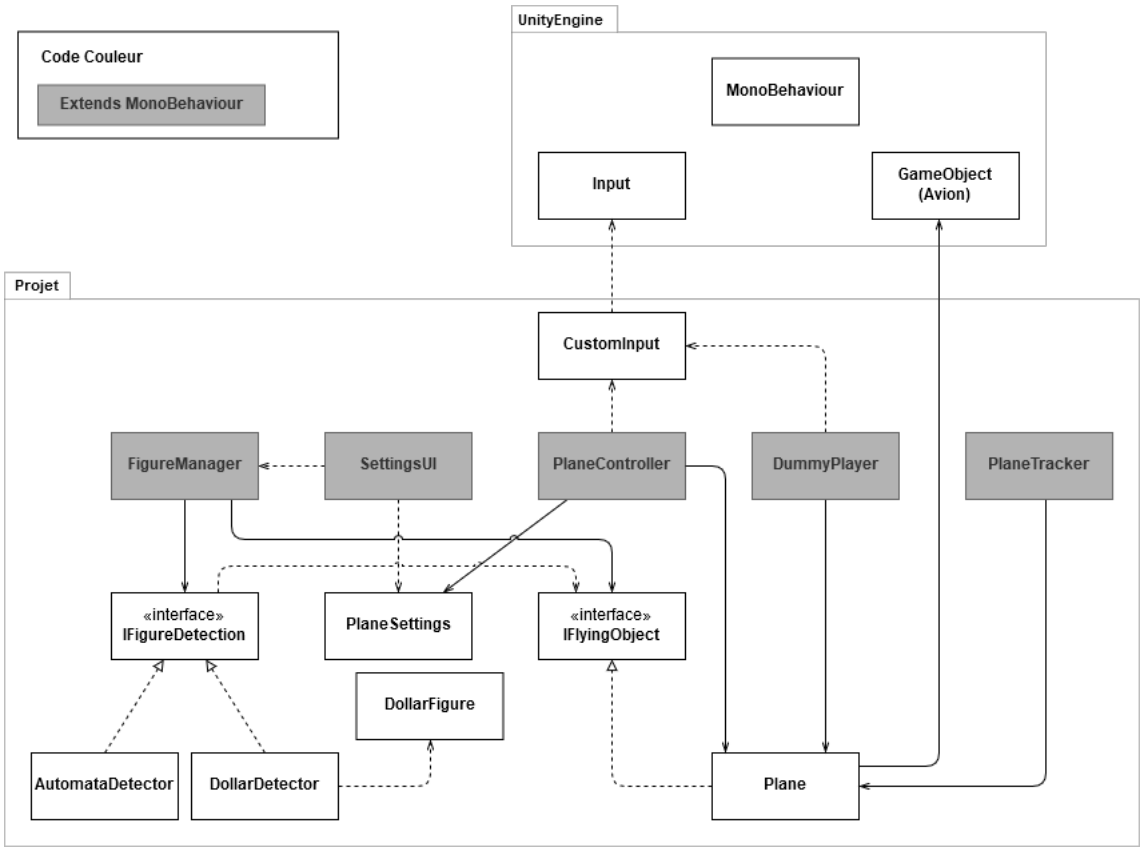

FIGURE 4 – Diagramme de classes du projet

Ce diagramme montre en particulier que les algorithmes de détection sont indépendants d'Unity : ils n'utilisent qu'un *IFlyingObject*, qui dans notre cas a été implémenté à l'aide du Gameobjet de l'avion dans le jeu.

4 Ajout d'une figure

Toutes les classes sont dans le dossier Assets/Scripts.

Le principe du projet est l'implémentation de détecteurs de figures, il est possible d'ajouter de nouvelles figures au jeu, il faut en revanche respecter certaines étapes pour qu'elles soient prises en compte par les détections :

1. ajouter un nouveau type de figure à la classe FigureManager
2. dans le cas d'un automate :
   (a) créer une nouvelle classe d'automate
   (b) ajouter l'automate au gestionnaire AutomataDetector
3. dans le cas de l'algorithme $P :
   (a) créer un fichier csv contenant la trajectoire de référence (trajectoire parfaite)
   (b) charger la trajectoire à l'initialisation du détecteur
   (c) ajouter l'analyse des résultats pour cette figure

Il n'est pas nécessaire d'ajouter la nouvelle figure aux deux algorithmes pour que le programme fonctionne.

4.1 Ajout de figure à FigureManager

Création du type de figure Il faut dans un premier temps déclarer le type de la figure s'il n'existe pas déjà, pour cela, on modifie l'énumération **figure_id** (fichier **TrajectoryStruct**) :

```
public enum figure_id {LOOP, ROLL, CUBANEIGHT, CUSTOMFIGURE};
```

Modification de FigureManager on doit modifier les attributs suivants de FigureManager :

```
private string[] _figureName = { "LOOP", "ROLL", "CUBANEIGHT", "CUSTOM" };
private int[] _figurePoint = { 20, 10, 50, 1 };
```

Le premier associe un nom à la figure pour l'afficher, le second paramètre correspond au score associé. Ici, le constructeur redéfinit les attributs, il faut donc les modifier dans le constructeur de la classe FigureManager.

Une fois la figure définie, on peut s'intéresser à sa détection, qui va différer selon l'algorithme utilisé.

4.2 Ajout de figure : automate

Comme dit précédemment, le création d'un automate détecteur de figure passe par deux étapes, la création d'une nouvelle classe d'automate puis l'ajout de cette nouvelle classe au gestionnaire des automates.

4.2.1 Création de nouvel Automate

Les automates doivent implémenter l'interface **IFigureAutomata**, il est possible de créer son propre automate indépendant ou bien utiliser les classes abstraites préexistantes :

- **SimpleAutomata** (recommandé) : dispose d'états simples qui représentent les orientations de l'avion, à utiliser pour construire des figures par morceaux, exemple : un looping suivi d'un quart de Aileron Roll.
- **AbstractComposedFigureAutomata** (non testé) : utile si on veut que des sous-figures s'exécutent pendant une figure principale, exemple : Barrel Roll composé d'un Aileron Roll dans un Looping. Nécessite des automates déjà construits.
- **AbstractSequentialFigureAutomata** (non testé) : utile pour effectuer des figures à la suite, exemple : deux Looping à la suite. Nécessite des automates déjà construits.

Utilisation de SimpleAutomata :

1. Créer une classe héritant de SimpleAutomata
2. Implémenter le constructeur, getFigureId(), getName() et calculateState() de la même manière que dans LoopingAutomata (par exemple).
3. Détails sur CalculateState :
   - Faire appel à init pour tout initialiser, puis faire un appel à isValid() pour vérifier que nous ne sommes pas dans un état final
   - Ajouter les différentes fonctions de vérifications si voulu(checkTime, checkForward, checkAltitude).
   - Créer l'automate : ajouter dans la tableau figure[] une séquence de quart de figures (Il faut modifier le nombre d'états dans le constructeur, paramètre n)
   - Faire appel à process() pour faire les changements éventuels d'états
   - Revérifier si l'on est dans un état final

Le fichier CustomAutomata est présent pour essayer directement une figure personnalisable. Pour créer un nouvel état à détecter (état Q1Chandelle par exemple), il faut le faire dans la classe abstraite SimpleAutomata.

4.2.2 Ajout à AutomataDetector

Une fois le nouvel automate créé, il faut l'ajouter au constructeur d'AutomataDetector, la ligne sera de la forme :

```
_myAutomatas.Add(new MyAutomata());
```

Si l'automate est bien construit et que sa figure est reconnue par FigureManager, le jeu devrait afficher le résultat quand on exécute la figure correctement.

4.3 Ajout d'une figure : $P

Il est possible d'ajouter une figure à la liste des reconnaissances de $P. Pour cela, il est nécessaire de disposer d'un fichier contenant la trajectoire de référence au format csv. Ce fichier peut être enregistré directement depuis le jeu (c.f. partie 2.4). Il faut ensuite modifier quelques fichiers :

- **DollarFigure.cs** : Il faut rajouter un tableau de chaînes de caractère donnant le nom des courbes de la nouvelle figure. Ce nom peut être commun à plusieurs figures et apparaître plusieurs fois dans le tableau. Ce nom sert à distinguer les courbes lors de la reconnaissance. Si deux courbes au même indice ont une allure similaire, il vaut mieux leur donner le même nom pour éviter les confusions de l'algorithme.
- **FigureLoaderP.cs** : Il faut rajouter, dans la fonction `LoadFigures()`, un appel à la fonction `Load()` avec, en paramètre, le fichier à charger et le tableau contenant le nom des courbes précédemment rajouté.
- **DollarDetector.cs** : Il faut écrire le code suivant dans la fonction `detection` en remplaçant <tableau de nom des figures> et <id figure> par les bonnes valeurs :

```
if (AnalyseResults(results, <tableau de nom des courbes>)) {
    figures.Add(new Figure(<id figure>, 1f));
    ClearLists();
}
```

5 Comparaison des algorithmes de détection

5.1 Qualité de détection

L'automate reconnaît les figures simples de manière quasi-systématique. Le Cuban eight est parfois reconnu sans que le joueur ne le veuille, cela est dû au fait qu'elle est composée de rotations simples de l'avion, et donc un joueur voulant faire deux looping en s'inclinant peut déclencher la reconnaissance du Cuban eight. $P reconnaît le Loop et le Roll mais cette détection se fait souvent trop tôt, au milieu de la figure. On peut donc faire une moitié de Roll qui est reconnue comme un Roll complet. Un autre problème est que le Cuban eight n'est pas reconnu, car cette figure nécessite un grand nombre de points (> 1000) qui ne rentre pas dans les tableaux de points que nous donnons en entrée à l'algorithme. Plus généralement, le problème majeur des automates est qu'il n'y a que deux issues possibles : si le joueur passe exactement par tous les états de la figure, il la réussit, sinon elle est ratée. Cela amène à des situations où, par exemple, le Loop n'est pas détecté car le joueur ne s'est pas suffisamment repositionné à la verticale. Les algorithmes $ n'ont pas ce problème : comme on cherche à identifier la figure dans sa globalité, une anomalie locale aura peu d'impact sur la détection finale.

5.2 Complexité des algorithmes

La complexité des algorithmes est un point crucial dans notre cas, car la détection est censé se faire en temps réel : un algorithme trop coûteux entraînera une chute du nombre d'images par seconde, rendant le jeu difficilement jouable. La détection par automate est ici très efficace : sachant que la vérification d'une transition se fait en temps constant, la détection par automate dans sa complexité est de complexité linéaire en le nombre de figures détectables. Les algorithmes $ sont, quant à eux, beaucoup moins performants. La complexité d'au pire $O(n^{3m})$, avec n le nombre de points d'échantillonnage et m le nombre de figures, de $P en fait ainsi un algorithme extrêmement lourd en terme d'utilisation processeur.

5.3 Facilité d'utilisation

Le gros avantage de la méthode automate est sa facilité d'évolution. Comme vu précédemment, grâce à l'interface designer, il est très simple de rajouter et créer un nouvel automate pour une nouvelle figure. Pour $P, la démarche est un peu plus complexe, mais reste aisée. Si l'on met de côté le problème de taille de fenêtre, ajouter une figure est très simple puisqu'il suffit de l'enregistrer en jeu, ce qui crée un fichier CSV que pourra utiliser l'algorithme en tant que figure de référence, et cela ne dépend pas de la complexité de la figure. Il faudra tout de même penser à ajouter, nommer chaque courbe composant la figure, pour que celles-ci puissent être identifiées.